

O'REILLY®

Разработка конвейеров машинного обучения

Автоматизация
жизненных циклов модели
с помощью TensorFlow



Ханнес Хапке
Кэтрин Нельсон

Ханнес Хапке
Кэтрин Нельсон

Разработка конвейеров машинного обучения

Автоматизация жизненных циклов
модели с помощью TensorFlow

Building Machine Learning Pipelines

Automating Model Life Cycles
with TensorFlow

Hannes Hapke
Catherine Nelson

Разработка конвейеров машинного обучения

Автоматизация жизненных циклов
модели с помощью TensorFlow

*Ханнес Ханке
Кэтрин Нельсон*



Москва, 2021

УДК 004.4
ББК 32.972
X12

Хапке Х., Нельсон К.

- X12** Разработка конвейеров машинного обучения. Автоматизация жизненных циклов модели с помощью TensorFlow / пер. с англ. Н. Б. Желновой. – М.: ДМК Пресс, 2021. – 346 с.: ил.

ISBN 978-5-97060-886-9

Машинное обучение становится важным элементом почти во всех отраслях. В этой книге представлено четкое и понятное руководство по автоматизации развертывания, управления и повторного использования моделей машинного обучения. Шаг за шагом описывается конкретный пример проекта, на котором можно отработать основные навыки в этой сфере. Благодаря множеству примеров кода и ясным, лаконичным объяснениям вы сможете создать свой собственный конвейер машинного обучения и запустите его в кратчайшие сроки.

Книга поможет ученым и инженерам, специализирующимся в области машинного обучения и искусственного интеллекта, выйти за рамки работы с единичной моделью и успешно реализовать свои проекты в области науки о данных. Также издание будет полезно менеджерам проектов в области науки о данных, разработчикам программного обеспечения и инженерам DevOps, которые хотят, чтобы их организация ускорила свои проекты, использующие технологии машинного обучения и искусственного интеллекта.

Читателю понадобится знание основных концепций машинного обучения и хотя бы одного из фреймворков, используемых в машинном обучении (например, PyTorch, TensorFlow, Keras).

УДК 004.4
ББК 32.972

DMK Press Authorized Russian translation of the English edition of Building Machine Learning Pipelines

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN (англ.) 978-1492053194
ISBN (рус.) 978-5-97060-886-9

© 2020 Hannes Hapke and Catherine Nelson
© Оформление, издание, перевод, ДМК Пресс, 2021

Оглавление

https://t.me/it_boooks

Предисловие от издательства	13
Предисловие	14
Введение	17
Для кого предназначена эта книга.....	18
Почему мы используем TensorFlow и TensorFlow Extended.....	19
Обзор глав	19
Условные обозначения, используемые в этой книге	21
Использование примеров кода.....	22
Онлайн-обучение O'Reilly.....	22
Как с нами связаться	23
Благодарности	23
Глава 1. Введение	26
Почему и где используются конвейеры машинного обучения	26
Когда следует подумать о конвейерах машинного обучения?.....	28
Обзор этапов конвейера машинного обучения	28
Этап загрузки данных и управление версиями данных.....	29
Проверка данных.....	29
Предварительная обработка данных	30
Обучение и настройка модели	31
Анализ модели.....	31
Управление версиями модели.....	32
Развертывание модели	32
Петли обратной связи	32
Приватность данных	33
Оркестровка конвейера.....	33
Для чего нужна оркестровка конвейера	33
Направленные ациклические графы	34
Наш демонстрационный проект машинного обучения	35
Структура проекта.....	36
Наша модель машинного обучения	36
Цель демонстрационного проекта	37
Резюме.....	37

Глава 2. Введение в TensorFlow Extended	38
Что такое TFX?	39
Установка TFX	40
Обзор компонентов TFX	41
Что такое метаданные ML Metadata?	42
Альтернативы TFX	45
Знакомство с Apache Beam	46
Установка	46
Базовый конвейер	47
Запуск элементарного конвейера	50
Резюме	50
Глава 3. Загрузка данных	51
Концепции загрузки данных	51
Загрузка локальных файлов данных	53
Загрузка удаленных файлов данных	57
Загрузка данных напрямую из баз данных	58
Подготовка данных	60
Разбиение наборов данных	60
Связующие наборы данных	62
Управление версиями наборов данных	63
Стратегии загрузки данных	64
Структурированные данные	64
Текстовые данные для задач обработки естественного языка	64
Графические данные для задач компьютерного зрения	64
Резюме	65
Глава 4. Проверка данных	66
Для чего нужна проверка данных?	67
TFDV	68
Установка	69
Генерация статистических показателей для набора данных	69
Генерация схемы на основе данных	71
Распознавание ошибок в данных	72
Сравнение наборов данных	73
Обновление схемы	75
Отклонения и дрейф данных	76
Наборы данных с систематической ошибкой выборки	77
Получение среза данных в TFDV	78
Обработка больших наборов данных с помощью Google Cloud Platform	80
Интеграция TFDV в конвейер машинного обучения	83
Резюме	84

Глава 5. Предварительная обработка данных	85
Для чего нужна предварительная обработка данных	86
Предварительная обработка данных в контексте всего набора данных	86
Масштабирование шагов предварительной обработки	86
Как избежать отклонения при обучении и работе модели	86
Развертывание шагов предварительной обработки и модели машинного обучения как единого артефакта	88
Проверка результатов предварительной обработки в конвейере	88
Предварительная обработка данных с помощью TFT	89
Установка	90
Стратегии предварительной обработки	90
Лучшие практики	92
Функции TFT	93
Автономная работа TFT	95
Интеграция TFT в конвейер машинного обучения	97
Резюме	101
Глава 6. Обучение модели	102
Определение модели для нашего демонстрационного проекта	103
Компонент TFX Trainer	106
Функция <code>run_fn()</code>	106
Запуск компонента <code>Trainer</code>	110
Другие соображения относительно компонента <code>Trainer</code>	112
Использование TensorBoard в интерактивном конвейере	113
Стратегии распределения	115
Настройка модели	118
Стратегии настройки гиперпараметров	118
Настройка гиперпараметров в конвейерах TFX	119
Резюме	119
Глава 7. Анализ и проверка модели	120
Как проанализировать модель	121
Метрики классификации	121
Метрики регрессии	124
Анализ модели TensorFlow	125
Анализ одной модели в TFMA	126
Анализ нескольких моделей в TFMA	129
Анализ достоверности модели	130
Формирование срезов для прогнозов модели в TFMA	132
Проверка пороговых значений решений с использованием метрик справедливости	134
Проведение более детального анализа с помощью инструмента анализа альтернатив (What-If Tool)	136

Объяснение модели	140
Генерация объяснений с помощью WIT	142
Другие методы объяснения	143
Анализ и проверка модели в TFX	145
ResolverNode	145
Компонент Evaluator	146
Проверка при помощи компонента Evaluator	147
Компонент TFX Pusher	148
Резюме	148

Глава 8. Развертывание модели с помощью

TensorFlow Serving	149
Простой сервер моделей	150
Недостатки развертывания моделей с помощью API на основе Python	151
Отсутствие разделения кода	151
Отсутствие контроля версий модели	152
Неэффективный вывод модели	152
TensorFlow Serving	152
Обзор архитектуры TensorFlow	153
Экспорт моделей для TensorFlow Serving	153
Сигнатуры моделей	155
Методы сигнатуры	155
Проверка экспортированных моделей	157
Проверка модели	158
Тестирование модели	159
Установка TensorFlow Serving	160
Установка Docker	160
Установка на Ubuntu	160
Сборка TensorFlow Serving из исходного кода	161
Настройка сервера TensorFlow	161
Конфигурация при работе с одной моделью	162
Конфигурация при работе с несколькими моделями	164
REST или gRPC	166
REST	166
gRPC	166
Выполнение прогнозов на сервере моделей	167
Получение прогнозов модели с использованием REST	167
Работа с TensorFlow Serving через gRPC	169
А/В-тестирование модели с использованием TensorFlow Serving	172
Запрос метаданных модели с сервера моделей	173
REST-запросы метаданных модели	173
Запросы gRPC для метаданных модели	174

Пакетные запросы на вывод прогнозов модели	175
Настройка использования пакетного режима в прогнозировании	177
Другие функции оптимизации TensorFlow Serving	179
Альтернативы TensorFlow Serving	180
BentoML	180
Seldon	180
GraphPipe	181
Simple TensorFlow Serving	181
MLflow	181
Ray Serve	181
Развертывание моделей с использованием услуг поставщиков облачных решений	182
Сценарии использования	182
Пример развертывания с помощью облачных платформ Google	182
Развертывание модели с помощью конвейеров TFX	188
Резюме	189

Глава 9. Расширенные концепции развертывания моделей с помощью TensorFlow Serving

Разделение зон ответственности в процессе развертывания	190
Обзор рабочего процесса	191
Оптимизация загрузки удаленной модели	193
Оптимизация модели для развертываний	194
Квантование	194
Сокращение	195
Дистилляция	196
Использование TensorRT совместно с TensorFlow Serving	196
TFLite	197
Шаги по оптимизации моделей машинного обучения с помощью TFLite	197
Развертывание моделей TFLite с помощью TensorFlow Serving	199
Мониторинг экземпляров TensorFlow Serving	200
Установка Prometheus	200
Конфигурация TensorFlow Serving	202
Простое масштабирование с помощью TensorFlow Serving и Kubernetes	204
Дополнительная литература о Kubernetes и Kubeflow	205
Резюме	206

Глава 10. Расширенные концепции TensorFlow Extended

Расширенные концепции конвейеров машинного обучения	207
Одновременное обучение нескольких моделей	208

Экспорт моделей TFLite	209
Ограничения TFLite.....	210
Обучение модели с «теплым» запуском	212
Участие человека в конвейере машинного обучения	212
Настройка компонента Slack	214
Как использовать компонент Slack	214
Пользовательские компоненты TFX	215
Сценарии использования пользовательских компонентов	216
Создание пользовательского компонента с нуля.....	216
Повторное использование существующих компонентов	225
Резюме.....	228
Глава 11. Конвейеры, часть 1: Apache Beam и Apache Airflow....	230
Какой инструмент оркестрации выбрать?.....	231
Apache Beam.....	231
Apache Airflow	231
Kubeflow Pipelines	231
Kubeflow Pipelines на платформе AI.....	232
Преобразование вашего интерактивного конвейера TFX в производственный конвейер.....	232
Преобразование элементарного интерактивного конвейера для Beam и Airflow	234
Введение в Apache Beam	235
Оркестрация конвейеров TFX с помощью Apache Beam	235
Введение в Apache Airflow.....	237
Установка и начальная настройка.....	237
Элементарный пример использования Airflow.....	239
Оркестрация конвейеров TFX с помощью Apache Airflow.....	242
Настройка конвейера	242
Запуск конвейера.....	244
Резюме.....	245
Глава 12. Конвейеры, часть 2:	
Kubeflow Pipelines	246
Введение в Kubeflow Pipelines.....	247
Установка и начальная настройка.....	249
Доступ к установленному экземпляру Kubeflow Pipelines.....	251
Оркестрация конвейеров TFX с помощью Kubeflow Pipelines.....	252
Настройка конвейера	254
Запуск конвейера.....	258
Полезные функции Kubeflow Pipelines.....	264

Конвейеры, работающие на Google Cloud AI Platform	269
Настройка конвейера	269
Настройка конвейера TFX.....	273
Запуск и работа конвейера	276
Резюме.....	277
Глава 13. Петли обратной связи	279
Явная и неявная обратная связь.....	280
Маховик данных	281
Петли обратной связи в реальном мире	282
Конструктивные шаблоны для сбора отзывов	284
Пользователи предпринимают определенные действия в результате прогноза	284
Пользователи оценивают качество прогноза.....	285
Пользователи исправляют прогноз.....	285
Краудсорсинг аннотаций	286
Экспертные аннотации	287
Обратная связь автоматически предоставляется системой	287
Как отслеживать петли обратной связи	287
Отслеживание явной обратной связи	288
Отслеживание неявной обратной связи	289
Резюме.....	289
Глава 14. Приватность данных, используемых для машинного обучения	290
Введение в приватность данных	290
Почему мы заботимся о приватности данных?	291
Самый простой способ повысить приватность данных	291
Какие данные должны быть приватными?	292
Дифференцированная приватность.....	293
Локальная и глобальная дифференцированная приватность.....	294
Эпсилон-дельта и бюджет приватности	295
Дифференцированная приватность в машинном обучении	296
Введение в TensorFlow Privacy	296
Обучение с оптимизатором, использующим подход дифференцированной приватности	297
Расчет параметра ϵ	298
Введение в федеративное обучение.....	299
Федеративное обучение в TensorFlow.....	301
Зашифрованное машинное обучение.....	302
Зашифрованное обучение модели	303
Преобразование обученной модели для обслуживания зашифрованных прогнозов	304

Другие методы обеспечения приватности данных.....	305
Резюме.....	305

Глава 15. Будущее конвейеров машинного обучения

и следующие шаги.....	307
Отслеживание экспериментов с моделью	307
Предложения в области управления релизами модели.....	308
Будущие возможности конвейеров.....	309
Использование TFX с другими фреймворками машинного обучения.....	310
Тестирование моделей машинного обучения	310
Системы непрерывной интеграции и развертывания для машинного обучения.....	311
Сообщество инженеров машинного обучения.....	311
Резюме.....	311

Приложение А. Введение в инфраструктуру машинного обучения.....	313
--	------------

Приложение В. Настройка кластера Kubernetes в Google Cloud	326
---	------------

Приложение С. Советы по работе с Kubeflow Pipelines	332
--	------------

Предметный указатель	340
-----------------------------------	------------

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Предисловие

Когда в 1913 году компания Генри Форда построила свой первый сборочный конвейер для производства своей легендарной Model T, время, необходимое для сборки каждой машины, сократилось с 12 до 3 часов. Затраты на производство резко снизились, что позволило Model T стать первым доступным автомобилем в истории. Это также сделало возможным массовое производство: вскоре Model T стала королевой автомобильных дорог.

Поскольку производственный процесс теперь представлял собой четкую последовательность четко определенных шагов (он же конвейер), стало возможно автоматизировать некоторые из этих шагов, сэкономя еще больше времени и денег. Сегодня производство автомобилей невозможно без автоматизации.

Но дело не только во времени и деньгах. При выполнении многих повторяющихся задач машина будет производить гораздо более стабильный результат, чем люди, в результате чего конечный продукт станет более предсказуемым, последовательным и надежным. Наконец, избавляя людей от тяжелого физического труда, автоматика позволяет значительно повысить безопасность. Многие рабочие от монотонной физической работы перешли к работе, требующей более высокого уровня квалификации (хотя, честно говоря, многие просто потеряли работу).

Если посмотреть на автоматизацию с другой стороны, установка автоматизированной сборочной линии может занять много времени и стать дорогостоящим проектом. Кроме того, сборочный конвейер – не идеальное решение, если вы хотите производить небольшие партии или продукцию по индивидуальному заказу. Форд сказал: «Цвет автомобиля может быть любым, при условии что он будет черным».

История автомобилестроения повторилась в индустрии программного обеспечения за последние пару десятилетий: каждая значительная часть программного обеспечения в настоящее время создается, тестируется и разворачивается с использованием таких инструментов автоматизации, как Jenkins или Travis. Однако метафоры Model T уже недостаточно. Программное обеспечение не просто разворачивается и работает как есть; его необходимо регулярно контролировать, поддерживать и обновлять. Программные конвейеры теперь больше похожи на динамические циклы, чем на статические производственные линии. Крайне важно иметь возможность быстро обновлять программное обеспечение (или сам конвейер), не нарушая его целостности. А программное обеспечение гораздо более вариативно, чем когда-либо была Model T: программное обеспечение можно раскрасить в любой цвет (попробуйте, например, подсчитать количество существующих вариантов для MS Office).

К сожалению, «классические» инструменты автоматизации не подходят для создания полноценного конвейера машинного обучения. Действительно, модель машинного обучения не является обычным программным обеспечением.

Во-первых, большая часть его поведения определяется данными, на которых он обучается. Следовательно, сами обучающие данные должны рассматриваться как код (и, соответственно, иметь версии). Это довольно сложная проблема, потому что новые данные появляются каждый день (часто в больших количествах), изменяются и дрейфуют с течением времени, часто включают персональные данные; также новые данные должны быть размечены, прежде чем вы сможете передать их в работу, которую выполняют алгоритмы машинного обучения.

Во-вторых, поведение модели нередко бывает довольно непрозрачным: она может пройти все тесты для одних данных, но полностью потерпеть неудачу для других. Таким образом, вы должны убедиться, что ваши тесты охватывают все области данных, на которых ваша модель будет использоваться в производстве. В частности, вы должны убедиться, что она не проявляет дискриминацию в отношении какой-либо группы ваших пользователей.

По этим (и другим) причинам специалисты по обработке данных и инженеры-программисты сначала начали создавать и обучать модели машинного обучения вручную, так сказать, «в своем гараже», и многие из них до сих пор это делают. Но за последние несколько лет были разработаны новые инструменты автоматизации, которые решают задачи конвейеров машинного обучения, такие как TensorFlow Extended (TFX) и Kubeflow. Все больше и больше организаций начинают использовать эти инструменты для создания конвейеров машинного обучения, которые автоматизируют большую часть (или все) этапов построения и обучения моделей машинного обучения. Преимущества этой автоматизации в основном те же, что и для автомобильной промышленности: экономия времени и денег; возможность создавать более качественные, надежные и безопасные модели и тратить больше времени на выполнение более полезных задач, чем на копирование данных или изучение кривых обучения. Однако построить конвейер машинного обучения непросто. Так с чего же начать?

Начать с этой книги!

В этой книге Ханнес и Кэтрин дают четкое и понятное руководство по автоматизации конвейеров машинного обучения. Как твердому стороннику практического подхода, особенно для такой технической темы, мне особенно понравилось то, как эта книга шаг за шагом проведет вас через конкретный пример проекта от начала до конца. Благодаря множеству примеров кода и ясным, лаконичным объяснениям вы сможете создать свой собственный конвейер машинного обучения и запустить его в кратчайшие сроки, а также все концептуальные инструменты, необходимые для адаптации этих конвейеров машинного обучения к вашим собственным вариантам использования. Я настоятельно рекомендую вам взять свой ноутбук и попробовать что-то во время чтения: так вы научитесь намного быстрее.

Я впервые встретился с Ханнесом и Кэтрин в октябре 2019 года на конференции TensorFlow World в Санта-Кларе, Калифорния, где я делал доклад о создании конвейеров машинного обучения с использованием TFX. Они работали над этой книгой по той же теме, и у нас был один редактор, так что, естественно, нам было о чем поговорить. Некоторые слушатели задавали технические вопросы о TensorFlow Serving (который является частью TFX), и у Ханнеса и Кэтрин были все ответы, которые я искал. Ханнес даже любезно принял мое при-

глашение выступить с докладом о расширенных функциях TensorFlow Serving в конце моего курса в очень короткие сроки. Его выступление было сокровищницей идей и полезных советов, которые вы найдете в этой книге, а также во многих, многих других.

Пришло время приступить к созданию профессиональных конвейеров машинного обучения!

– *Орелиен Жерон*,
бывший руководитель группы классификации видео YouTube,
автор книги «Практическое машинное обучение
с использованием Scikit-Learn, Keras и TensorFlow» (O'Reilly)
Окленд, Новая Зеландия, 18 июня 2020 г.

Введение

Все говорят о машинном обучении. Из академической дисциплины оно превратилось в одну из самых удивительных технологий. Машинное обучение используется везде – от обработки видеопотока, регистрируемого в автомобилях с автоматическим управлением, до персонализации назначений лекарств. Оно становится важным элементом в каждой отрасли. В то время как моделям архитектуры и концепциям уделялось большое внимание, машинное обучение еще не прошло стадию стандартизации процессов, появившихся в отрасли программного обеспечения в последнее десятилетие. В этой книге мы хотели бы показать вам, как создать стандартизированную систему машинного обучения, которая была бы автоматизированной и воспроизводимой.

За последние несколько лет разработки в области машинного обучения достигли впечатляющих результатов. Благодаря широкой доступности графических процессоров (Graphical Processing Units, GPU) и разработке новых концепций глубокого обучения, таких как Transformers (например, BERT) или Generative Adversarial Networks (например, DCGAN), количество проектов в области искусственного интеллекта, или ИИ (Artificial Intelligence, AI), резко возросло. Количество стартапов в сфере ИИ огромно. Корпорации применяют новейшие технологии машинного обучения для решения всех видов бизнес-задач. В этом стремлении к наиболее эффективному решению для задач машинного обучения мы наблюдали несколько вещей, которым уделялось меньше внимания. Мы увидели, что специалистам в области ИИ – ученым, специализирующимся в области машинного обучения и искусственного интеллекта, и инженерам по машинному обучению – не хватает хороших источников информации для создания концепций и инструментов для ускорения, повторного использования, управления и развертывания своих разработок. Необходима стандартизация конвейеров машинного обучения.

Конвейеры машинного обучения – это процессы для ускорения, повторного использования, управления и развертывания моделей машинного обучения. Примерно десять лет назад разработка программного обеспечения претерпела такие же изменения благодаря внедрению непрерывной интеграции (Continuous Integration, CI) и непрерывного развертывания (Continuous Deployment, CD). Когда-то это был длительный процесс тестирования и развертывания веб-приложения. В наши дни эти процессы были значительно упрощены с помощью нескольких инструментов и концепций. Ранее для развертывания веб-приложений требовалось сотрудничество между инженером DevOps и разработчиком программного обеспечения. Сегодня приложение можно надежно протестировать и развернуть за считанные минуты. Специалисты по обработке данных и инженеры машинного обучения могут заимствовать концепции рабочих процессов из программной инженерии.

Исходя из нашего личного опыта, большинство проектов в области науки о данных, нацеленных на внедрение моделей в производство, не могут позволить себе роскошь создавать большие команды, что затрудняет построение всего конвейера собственными силами с нуля. Это может означать, что проекты машинного обучения превращаются в одиночные попытки построения моделей, и их результативность ухудшается со временем; специалист тратит большую часть своего времени на исправление ошибок, когда меняются данные, лежащие в основе решения, или модель не находит широкого применения. Автоматизированный конвейер, позволяющий построить воспроизводимый сквозной рабочий процесс, уменьшает усилия, необходимые для развертывания модели. Конвейер машинного обучения должен включать процессы, которые:

- эффективно отслеживают изменение версий исходных данных и запускают новое обучение моделей;
- эффективно выполняют предварительную обработку данных для обучения и проверки модели;
- следят за версиями контрольных точек модели во время обучения;
- отслеживают ваши эксперименты по обучению моделей;
- анализируют и проверяют обученные и настроенные модели;
- выполняют развертывание проверенных моделей;
- масштабируют развернутую модель;
- собирают новые данные для обучения и регистрируют показатели точности модели, используя петли обратной связи.

В этом списке пропущен один важный момент: обучение и настройка модели. Мы предполагаем, что вы уже обладаете достаточными знаниями и опытом для выполнения этого шага. Если же вы только начинаете погружаться в тему машинного или глубокого обучения, следующие книги, опубликованные O'Reilly, станут отличной отправной точкой для знакомства с машинным обучением:

- *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms* 1st Edition by Nikhil Buduma, Nicholas Locascio (*Нихил Будума, Николас Локасио*. Основы глубокого обучения: разработка алгоритмов искусственного интеллекта следующего поколения. 1-е изд.);
- *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* 2nd Edition by Aurélien Géron (*Аурелиен Жерон*. Практическое машинное обучение с использованием Scikit-Learn, Keras и TensorFlow: концепции, инструменты и методы для построения интеллектуальных систем. 2-е изд.).

Для кого предназначена эта книга

Основная аудитория этой книги – ученые, специализирующиеся в области машинного обучения и искусственного интеллекта и инженеры по машинному обучению, которые хотят выйти за рамки обучения единичной модели машинного обучения и успешно реализовать свои проекты в области науки о данных. Вы должны быть знакомы с основными концепциями машинного обучения и хотя бы с одним из фреймворков, используемых в машинном

обучении (например, PyTorch, TensorFlow, Keras). Примеры в этой книге основаны на TensorFlow и Keras, но основные концепции могут быть применены к любой среде.

Также эта книга может быть полезна менеджерам проектов в области науки о данных, разработчикам программного обеспечения или инженерам DevOps, которые хотят, чтобы их организация ускорила свои проекты, использующие технологии машинного обучения и искусственного интеллекта. Если вы заинтересованы в лучшем понимании жизненного цикла разработки систем на основе автоматизированного машинного обучения и хотите понять, как это может принести пользу вашей организации, то в следующих главах будет представлен набор инструментов для достижения данной цели.

ПОЧЕМУ МЫ ИСПОЛЬЗУЕМ TENSORFLOW И TENSORFLOW EXTENDED

В этой книге во всех наших примерах конвейера будут использоваться инструменты из экосистемы TensorFlow, в частности TensorFlow Extended (TFX). Мы выбрали этот фреймворк по ряду причин:

- экосистема TensorFlow является наиболее доступной для машинного обучения на момент написания статьи. Она включает в себя несколько полезных проектов и библиотек поддержки, таких как TensorFlow Privacy и TensorFlow Probability;
- она популярна и широко используется как в малых, так и на крупных производственных предприятиях, и существует активное сообщество заинтересованных пользователей;
- поддерживаемые варианты использования простираются от академических исследований до машинного обучения в производственной среде. TFX тесно интегрирован с базовой платформой TensorFlow для использования в производственных процессах;
- и TensorFlow, и TFX являются инструментами с открытым исходным кодом, и нет никаких ограничений на их использование.

Однако все принципы, которые мы описываем в этой книге, применимы и к другим инструментам и фреймворкам.

ОБЗОР ГЛАВ

В следующих главах мы представим конкретные шаги для построения конвейеров машинного обучения и продемонстрируем, как они работают, на примере демонстрационного проекта.

Глава 1 «Введение». В этой главе представлен обзор конвейеров машинного обучения, обсуждается, когда их следует использовать, и описываются все этапы, входящие в состав конвейера. Также мы представляем здесь пример проекта, который будем использовать на протяжении всей книги.

Глава 2 «Введение в TensorFlow Extended» знакомит читателей с экосистемой TFX, объясняет, как задачи взаимодействуют друг с другом, и описывает внутреннюю работу компонентов TFX. Мы также рассмотрим ML MetadataStore,

покажем, как он используется в контексте TFX, и как Apache Beam запускает компоненты TFX без вмешательства пользователя.

Глава 3 «Загрузка данных» посвящена процессу систематической загрузки данных в наши конвейеры, а также в ней обсуждается концепция управления версиями данных.

Глава 4 «Проверка данных» объясняет, каким образом организовать эффективную проверку данных, поступающих в ваш конвейер, с помощью инструмента TensorFlow Data Validation. Система предупредит вас, если новые данные существенно изменятся по сравнению с предыдущими данными, что может повлиять на точность вашей модели.

Глава 5 «Предварительная обработка данных» посвящена подготовке данных (конструированию признаков) с использованием TensorFlow Transform для преобразования необработанных данных в признаки, подходящие для обучения модели.

Глава 6 «Обучение модели» объясняет, как обучать модели в рамках задачи машинного обучения. В этой главе мы также объясним концепцию настройки модели.

Глава 7 «Анализ и проверка модели» содержит полезные метрики для понимания, как работает ваша модель в производственной среде. В число этих метрик входят и те, которые могут позволить вам выявить ошибки в предсказаниях модели. В разделе «Анализ и проверка модели в TFX» объясняется, как управлять версиями вашей модели при улучшении одного из показателей в новой версии. Модель в конвейере может быть автоматически обновлена до новой версии.

Глава 8 «Развертывание моделей с помощью TensorFlow Serving» посвящена тому, как эффективно развернуть модель машинного обучения. Начиная с простой реализации Flask мы подчеркиваем ограничения в использовании таких пользовательских моделей в приложениях. Мы расскажем о TensorFlow Serving и о том, как настроить ваши исполнительные экземпляры. Мы также обсуждаем его пакетный режим работы и демонстрируем настройку клиентов для запроса результатов прогнозирования на основе модели.

Глава 9 «Расширенные варианты развертывания моделей с помощью TensorFlow Serving» показывает, как оптимизировать ваши варианты развертывания модели и как их контролировать. Мы обсуждаем стратегии оптимизации ваших моделей TensorFlow для повышения производительности. Мы также продемонстрируем базовый сценарий настройки развертывания с использованием Kubernetes.

Глава 10 «Расширенные концепции TensorFlow Extended» представляет концепцию специальных компонентов для ваших конвейеров машинного обучения, так что вы не ограничены стандартными компонентами в TFX. Если вы хотите добавить дополнительные этапы загрузки данных или преобразовать экспортированные модели в TensorFlow Lite (TFLite), мы проведем вас через все этапы создания таких компонентов.

Глава 11 «Конвейеры, часть 1: Apache Beam и Apache Airflow» собирает воедино все, что обсуждалось в предыдущих главах. Мы обсудим, как превратить ваши компоненты в конвейеры и как их настроить для выбранной платформы оркестровки. Мы также проведем вас от начала до конца, через все шаги конвейера, работающего на Apache Beam и Apache Airflow.

Глава 12 «Конвейеры, часть 2: Конвейеры Kubeflow Pipelines» является продолжением предыдущей главы. В ней рассматривается построение конвейера с использованием Kubeflow и платформы искусственного интеллекта Google.

Глава 13 «Петли обратной связи» посвящена тому, как превратить ваш модельный конвейер в цикл, позволяющий вносить улучшения на основе отзывов пользователей конечного продукта. Мы обсудим, какие типы данных необходимо собирать, чтобы улучшить модель в будущих версиях, и как передать эти данные обратно в конвейер.

Глава 14: «Приватность данных, используемых для машинного обучения» представляет концепцию быстро меняющейся области машинного обучения с сохранением приватности данных. В ней обсуждается три важных элемента этой концепции: дифференцированная приватность, федеративное обучение и зашифрованное машинное обучение.

Глава 15 «Будущее конвейеров машинного обучения и следующие шаги» содержит обзор технологий, которые повлияют на будущие конвейеры машинного обучения и на то, как мы будем смотреть на машинное обучение в ближайшие годы.

Приложение А «Введение в инфраструктуру машинного обучения» содержит краткое введение в Docker и Kubernetes.

Приложение В «Настройка кластера Kubernetes в Google Cloud» содержит дополнительные материалы по настройке Kubernetes в Google Cloud.

Приложение С «Советы по работе с конвейерами Kubeflow» содержит несколько полезных советов по работе с настройками конвейеров Kubeflow, включая обзор интерфейса командной строки TFX.

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ, ИСПОЛЬЗУЕМЫЕ В ЭТОЙ КНИГЕ

В этой книге используются следующие условные обозначения.

Курсив

Обозначает новые термины, URL-адреса, адреса электронной почты, имена файлов и расширения файлов.

Моноширинный шрифт

Используется для оформления листингов программ, а также в тексте для обозначения фрагментов программы, таких как имена переменных или функций, базы данных, типы данных, переменные среды, операторы и ключевые слова.

Жирный моноширинный шрифт

Показывает команды или другой текст, который пользователь должен набирать точь-в-точь так, как показано в книге.

Моноширинный шрифт, выделенный курсивом

Показывает текст, который следует заменить пользовательскими значениями или значениями, определяемыми контекстом.



Данный элемент означает совет или предложение.



Данный элемент обозначает общее примечание.



Так обозначаются предупреждения и предостережения.

ИСПОЛЬЗОВАНИЕ ПРИМЕРОВ КОДА

Дополнительные материалы (примеры кода и т. п.) доступны для загрузки по ссылке <https://oreil.ly/bmlp-git>.

Если у вас есть технический вопрос или проблема с использованием примеров кода, отправьте электронное письмо на адрес bookquestions@oreilly.com и buildingmlpipelines@gmail.com.

Эта книга предназначена для того, чтобы помочь вам в работе над проектами. Вы можете использовать приведенные в этой книге примеры кода в своих программах и документации. Вам не нужно связываться с нами для получения разрешения на использование этих примеров, если вы используете незначительную часть кода. Например, для написания программы, использующей несколько фрагментов кода из этой книги, не требуется разрешения. Однако для коммерческого использования или распространения примеров из книг O'Reilly потребуется разрешение. Для ответа на вопрос с использованием цитат из этой книги и примеров кода разрешение не требуется. Но для включения значительного количества примеров кода из этой книги в документацию к вашему программному продукту потребуется разрешение.

Мы ценим, но не требуем указания авторства. Обычно при цитировании указывается название, автор, издатель и ISBN, например: «Создание конвейеров машинного обучения», Ханнес Хапке и Кэтрин Нельсон (O'Reilly). © 2020 Ханнес Хапке и Кэтрин Нельсон, 978-1-492-05319-4».

Если вы считаете, что ваше использование примеров кода выходит за рамки условий добросовестного использования или разрешений, приведенных выше, обращайтесь к нам по адресу электронной почты permissions@oreilly.com.

ОНЛАЙН-ОБУЧЕНИЕ O'REILLY

На протяжении более 40 лет O'Reilly Media разрабатывает технологические и бизнес-тренинги, продвигая знания и развивая идеи, чтобы помочь компаниям добиться успеха.

Наша уникальная сеть экспертов и новаторов делится своими знаниями и опытом через книги, статьи и нашу платформу онлайн-обучения. Платформа онлайн-обучения O'Reilly предоставляет по запросу доступ к курсам обучения в режиме реального времени, схемам углубленного обучения, интерактивным средам программирования и обширной коллекции текстов и видео от O'Reilly и более 200 других издателей. Для получения дополнительной информации перейдите по ссылке <http://oreilly.com>.

КАК С НАМИ СВЯЗАТЬСЯ

Оба автора этой книги хотели бы поблагодарить вас за уделенное ей внимание. Если вы хотите связаться с ними, вы можете сделать это через их веб-сайт www.buildingmlpipelines.com или по электронной почте buildingmlpipelines@gmail.com. Авторы желают вам всяческих успехов в создании собственных конвейеров машинного обучения.

Комментарии и вопросы, касающиеся этой книги, просьба направлять издателю:

O'Reilly Media, Inc.

1005 Gravenstein Highway Северный Севастополь, CA 95472

800-998-9938 (в США или Канаде) 707-829-0515 (международный или местный)

707-829-0104 (факс)

Для этой книги разработана веб-страница, где мы публикуем исправления, примеры, а также другую дополнительную информацию. Для того чтобы ознакомиться с материалами, размещенными на этой странице, перейдите по ссылке: <https://oreil.ly/build-ml-pipelines>.

Для отправки комментариев или технических вопросов по содержанию данной книги используйте адрес электронной почты bookquestions@oreilly.com.

Новости и информацию о наших книгах и курсах можно найти на сайте <http://oreilly.com>.

Наша страница в Facebook: <http://facebook.com/oreilly>.

Мы в Twitter: <http://twitter.com/oreillymedia>.

Наш канал YouTube: <http://www.youtube.com/oreillymedia>.

БЛАГОДАРНОСТИ

В процессе написания этой книги нас поддерживали многие замечательные люди. Большое спасибо всем, кто помог воплотить в жизнь наши замыслы! Мы хотели бы выразить особую благодарность всем вам.

Со всей командой O'Reilly было здорово работать все время, пока готовилась эта книга. Спасибо нашим редакторам Мелиссе Поттер (Melissa Potter), Николь Таше (Nicole Taché) и Амелии Блевинс (Amelia Blevins) за постоянную поддержку и вдумчивые отзывы. Спасибо также Кэти Тозер (Katie Tozer) и Джонатану Хасселлу (Jonathan Hassell) за оказанную нам поддержку.

Спасибо Орелиену Жерону (Aurélien Geron), Роберту Кроу (Robert Crowe), Маргарет Мейнард-Рид (Margaret Maynard-Reid), Сергею Хоменко (Sergii Khomenko) и Викраму Тивари (Vikram Tiwari), которые внимательно просмотрели

рели всю книгу и дали множество полезных советов и ценных комментариев. Вы много сделали для того, чтобы ее окончательный вариант стал лучше. Спасибо за часы, потраченные вами на внимательное изучение материалов книги.

Спасибо Янну Дюпису (Yann Dupis), Джейсону Манкузо (Jason Mancuso) и Мортену Далю (Morten Dahl) за ваш тщательный и всесторонний обзор главы о конфиденциальности машинного обучения.

Мы получили фантастическую поддержку от многих замечательных людей в Google. Благодарим вас за помощь в поиске и исправлении ошибок, а также за то, что вы выпускаете инструменты машинного обучения в виде пакетов с открытым исходным кодом! Помимо сотрудников Google, мы хотим особо поблагодарить Эми Унру (Amy Unruh), Анушу Рамеш (Anusha Ramesh), Кристину Грир (Christina Greer), Клеменс Мевальд (Clemens Mewald), Дэвида Заца (David Zats), Эдда Уайлдера-Джеймса (Edd Wilder-James), Ирен Джаннумис (Irene Giannoumis), Ярека Вилькевича (Jarek Wilkiewicz), Джайи Чжао (Jiayi Zhao), Джири Симсу (Jiri Simsa), Константиноса Катсиаписана (Konstantinos Katsiapis), Лак Лакшана (Lak Lakshmanan), Майка Древеса (Mike Dreves), Пейджа Бейли (Paige Bailey), Педрама Пеймана (Pedram Pejman), Сару Робинсон (Sara Robinson), Сунсон Квон (Soonson Kwon), Тею Ламкин (Thea Lamkin), Триса Варкентина (Tris Warkentin), Варшу Наганатан (Varshaa Naganathan), Чжитао Ли (Zhitao Li) и Зохара Яхава (Zohar Yahav).

Мы выражаем огромную благодарность сообществу TensorFlow и Google Developer Expert и его замечательным участникам. Спасибо за поддержку в этом начинании.

Спасибо другим участникам, которые помогли на разных этапах: Барбаре Фусинска (Barbara Fusinska), Хамелю Хусайну (Hamel Husain), Михалу Ястшембскому (Michał Jastrzębski) и Яну Хенселю (Ian Hensel).

Спасибо сотрудникам Concur Labs (прошлым и настоящим) и другим сотрудникам SAP Concur за плодотворные обсуждения и полезные идеи. В частности, спасибо Джону Дитцу (John Dietz) и Ричарду Пакетту (Richard Puckett) за неоценимую поддержку, оказанную авторам книги.

Ханнес

Я хочу поблагодарить мою замечательную партнершу Уитни за ее огромную поддержку на протяжении всего времени работы над этой книгой. Спасибо за постоянную поддержку, за отклики и за терпение, когда я провожу долгие часы за написанием статей. Спасибо моей семье, особенно моим родителям, которые позволили мне следовать за моей мечтой по всему миру.

Эта книга не появилась бы, не будь у меня замечательных друзей. Спасибо, Коул Ховард (Cole Howard), за то, что вы прекрасный друг и учитель. Наше сотрудничество подтолкнуло меня к этой работе и размышлениям о конвейерах машинного обучения. Моим друзьям Тимо Метцгеру (Timo Metzger) и Аманде Райт (Amanda Wright): спасибо за то, что вы улучшили язык этой книги. И спасибо Еве и Килиану Рамбах (Eva Rambach, Kilian Rambach), а также Деб и Дэвиду Хаклеманам (Deb Hackleman, David Hackleman). Без вашей помощи я бы не добрался до Орегона.

Я хотел бы поблагодарить моих предыдущих работодателей, Cambia Health, Caravel и Talentpair, за то, что они позволили мне реализовать концепции этой публикации в производственных условиях, несмотря на то что эти идеи были новаторскими.

Эта книга не вышла бы без моего соавтора Кэтрин. Спасибо за дружбу, поддержку и бесконечное терпение. Я рад, что мы познакомились по чистой случайности в жизни. Я очень рад, что мы вместе написали эту книгу.

Кэтрин

Я написала много слов в этой книге, но нет слов, чтобы выразить, насколько я ценю поддержку, которую оказал мне мой муж Майк. Спасибо за теплую поддержку, за приготовленную еду, полезные обсуждения, сарказм и ценные отзывы. Спасибо моим родителям за то, что давным-давно поддерживали мой интерес к программированию, – чтобы вырасти, мне было нужно время, но вы всегда были правы!

Спасибо всем замечательным сообществам, частью которых мне посчастливилось быть. Я встретила множество замечательных людей через Seattle PyLadies, Women in Data Science и более широкое сообщество Python. Я очень ценю вашу поддержку.

И спасибо Ханнесу за то, что он пригласил меня в это путешествие! Без тебя ничего бы не случилось! Ваши знания, внимательность и настойчивость сделали весь этот проект успешным. И это было очень весело!

Глава 1

Введение

В первой главе этой книги мы познакомимся с конвейерами машинного обучения и рассмотрим все этапы их создания. Мы объясним, что должно произойти, чтобы превратить вашу модель машинного обучения из экспериментальной в надежную производственную систему. Мы также представим наш пример проекта, который будем использовать в оставшейся части книги, чтобы продемонстрировать описанные нами принципы.

Почему и где используются КОНВЕЙЕРЫ МАШИННОГО ОБУЧЕНИЯ

Ключевое преимущество конвейеров машинного обучения заключается в автоматизации этапов жизненного цикла модели. Когда становятся доступными новые обучающие данные, должен быть запущен рабочий процесс, который включает проверку данных, предварительную обработку, обучение модели, анализ и развертывание. Мы наблюдали, как слишком много рабочих групп, занятых анализом данных, вручную выполняют эти шаги. Это обходится очень дорого и часто является источником ошибок. Давайте подробно рассмотрим некоторые из преимуществ конвейеров машинного обучения.

Возможность сосредоточиться на новых моделях, а не на поддержке существующих моделей

Автоматизированные конвейеры машинного обучения освободят специалистов по обработке данных от необходимости поддерживать существующие модели. Мы видели, как множество специалистов по данным тратят человеко-дни на обновление ранее разработанных моделей. Они вручную запускают сценарии для предварительной обработки своих обучающих данных, они пишут одноразовые сценарии развертывания или вручную настраивают свои модели. Автоматизированные конвейеры позволяют специалистам по обработке данных разрабатывать новые модели – ведь это гораздо более интересно. В конечном итоге это будет способствовать повышению удовлетворенности работой и удержанию персонала на конкурентном рынке труда.

Предотвращение ошибок

Автоматизированные конвейеры машинного обучения могут предотвратить ошибки. Как мы увидим в следующих главах, вновь созданные модели

будут привязаны к набору версионированных данных, а шаги предварительной обработки будут привязаны к разработанной модели. Это означает, что при сборе новых данных будет сгенерирована новая модель. При изменении шагов предварительной обработки обучающие данные станут недействительными, и будет сгенерирована новая модель. В рабочих процессах ручного машинного обучения обычным источником ошибок является изменение этапа предварительной обработки после обучения модели. В этом случае мы бы развернули модель с инструкциями обработки, отличными от тех, при помощи которых мы обучали модель. Подобные ошибки может быть действительно сложно отладить, модель все еще выдает результат, но этот результат просто неверен. Такие ошибки можно предотвратить с помощью автоматизированных рабочих процессов.

Полезная документация

Средства отслеживания эксперимента и управления версиями модели генерируют протоколы, в которых хранится история изменений модели. Отслеживание эксперимента включает в себя отслеживание изменений гиперпараметров модели, используемых наборов данных и результирующих метрик модели (таких как, например, потери или точность). Инструменты управления версиями модели будут помогать отслеживать, какая модель была в конечном итоге выбрана и развернута. Такая документация особенно полезна, если команде специалистов в области машинного обучения и искусственного интеллекта необходимо повторно создать модель или отследить качество модели.

Стандартизация

Стандартизированные конвейеры машинного обучения улучшают работу команды специалистов в области машинного обучения и искусственного интеллекта. Благодаря стандартизированным настройкам инженеры, занимающиеся обработкой данных, могут быстро подключаться к работе или переходить из одной группы в другую и работать с одной и той же средой разработки. Это повышает их эффективность и сокращает время, затрачиваемое на вход в новый проект. Время, затраченное на настройку конвейеров машинного обучения, также может способствовать повышению коэффициента удержания персонала.

Бизнес-модель для конвейеров машинного обучения

Внедрение конвейеров машинного обучения позволит командам, занимающимся разработкой в области машинного обучения, достичь следующих результатов:

- ♦ уменьшение времени разработки новых моделей;
- ♦ упрощение процессов обновления существующих моделей;
- ♦ уменьшение затрат времени на воспроизведение моделей.

Все эти аспекты ведут к уменьшению стоимости проектов, реализуемых в области машинного обучения и искусственного интеллекта. Более того, конвейеры машинного обучения обладают следующими преимуществами:

- ♦ помогают обнаружить потенциальные смещения в наборах данных или в обученных моделях. Обнаружение таких смещений может предотвратить нанесение ущерба людям, которые используют результаты построения модели. Например, система автоматического просмотра резюме на базе машинного обучения, созданная Amazon, как оказалось, проявляла «предвзятость» в отношении женщин в результате смещения¹;
- ♦ протоколы, полученные в результате отслеживания эксперимента и управления версиями модели, пригодятся, если возникнут вопросы, касающиеся соответствия Генеральному регламенту о защите персональных данных (General Data Protection Regulation, GDPR);
- ♦ автоматизация обновлений модели высвободит время специалистов в области машинного обучения и искусственного интеллекта и повысит их удовлетворенность работой.

Когда следует подумать о конвейерах машинного обучения?

Конвейеры машинного обучения предоставляют множество преимуществ, но не каждый проект в области обработки данных, машинного обучения или искусственного интеллекта нуждается в автоматизации процессов при помощи конвейера. Иногда ученые, работающие с данными, просто хотят поэкспериментировать с новой моделью, испытать новую архитектуру модели или воспроизвести недавно опубликованный результат. Для подобных случаев конвейеры не принесут никакой пользы. Однако как только у модели появляются пользователи, например она становится частью пользовательского приложения, вам потребуются постоянно обновлять модель и выполнять ее тонкую настройку. В этих ситуациях мы возвращаемся к сценариям, которые обсуждали ранее, когда говорили о непрерывном обновлении моделей и уменьшении бремени этих задач, которое ложится на специалистов-исследователей в области машинного обучения и искусственного интеллекта.

Роль конвейеров возрастает по мере роста проекта машинного обучения. Если требования к набору данных или ресурсам велики, обсуждаемые нами подходы позволяют легко масштабировать инфраструктуру. Если важна повторяемость, это обеспечивается за счет автоматизации и ведения контрольного журнала конвейеров машинного обучения.

Обзор этапов конвейера машинного обучения

Работа конвейера машинного обучения начинается со сбора новых данных для обучения и заканчивается получением определенной обратной связи о том, как работает ваша недавно обученная модель. В качестве вариантов такой обратной связи может рассматриваться метрика достоверности модели или же отзывы пользователей вашего продукта. Конвейер включает в себя множество этапов, включая предварительную обработку данных, обучение и анализ мо-

¹ Статья Reuters от 9 октября 2018 г.

дели, а также ее развертывание. Вы можете себе представить, что выполнение этих шагов вручную чрезвычайно обременительно, и вероятность допустить ошибку при таком сценарии очень велика. В этой книге мы представим инструменты и решения для автоматизации жизненного цикла вашей модели.

Как вы можете видеть на рис. 1.1, жизненный цикл модели фактически является циклическим процессом. Данные могут собираться непрерывно, и поэтому модели машинного обучения могут обновляться. Большее количество данных обычно означает улучшение модели. И из-за этого постоянного притока данных именно автоматизация является ключевым фактором. В реальных приложениях вы часто хотите повторно обучать свои модели. Если это ручной процесс, когда необходимо вручную проверить новые данные обучения или проанализировать обновленные модели, у специалиста в области машинного обучения или работающего с данными инженера просто не останется времени для разработки новых моделей для новых бизнес-задач.

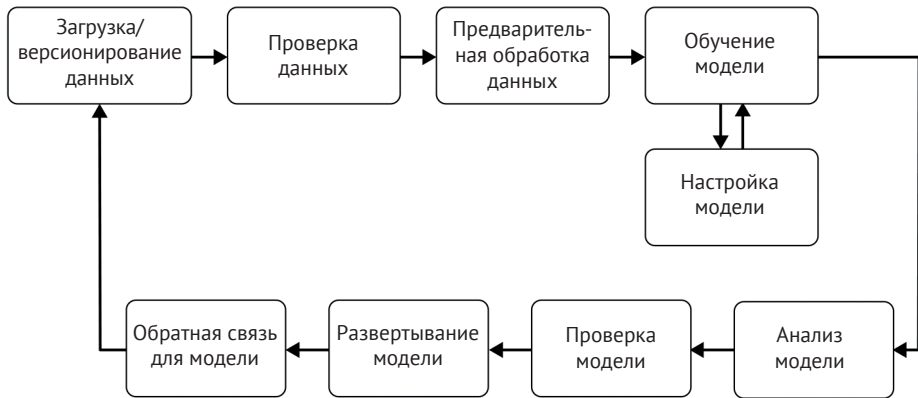


Рис. 1.1. Жизненный цикл модели

Жизненный цикл модели обычно включает в себя следующие этапы.

Этап загрузки данных и управление версиями данных

Загрузка данных – это начальный этап каждого жизненного цикла модели. На этом этапе работы конвейера мы обрабатываем данные, приводя их к формату, который могут воспринимать и обрабатывать следующие этапы. На этапе загрузки данных не выполняется никакое конструирование признаков (это происходит после этапа проверки данных). Это также хороший момент для создания новой версии поступающих данных, чтобы связать снимок данных с обученной моделью в конце рабочего процесса конвейера.

Проверка данных

Перед обучением новой версии модели нам нужно проверить новые данные. При проверке данных (о которой рассказывается в главе 4) мы фокусируемся на проверке статистических характеристик новых данных (таких как диапазон, количество категорий и распределение категорий), убеждаясь, что эти характеристики соответствуют ожиданиям, и предупреждаем специалиста, работа-

ющего с данными, в случае если обнаружены какие-либо аномалии. Например, если вы обучаете модель бинарной классификации, ваши данные, используемые для обучения, могут содержать 50 % выборок класса X и 50 % выборок класса Y. Инструменты проверки данных выдают предупреждения в случае, если пропорция в распределении выборок между этими классами изменяется, например когда вновь собранные данные распределяются между двумя классами в соотношении 70/30. Если модель обучается на таком несбалансированном обучающем наборе и разработчик не скорректировал функцию потерь модели или избыточно/недостаточно представленную в выборке категорию X или Y, прогнозы модели могут быть смещены в сторону преобладающей категории.

Стандартные инструменты проверки данных также позволят вам сравнивать различные наборы данных. Если у вас есть набор данных с преобладающей меткой и вы разбили исходные данные на обучающий и контрольный наборы, вы должны убедиться, что распределение меток между двумя этими наборами данных примерно одинаково. Инструменты проверки данных позволят вам сравнить наборы данных и выделить отклонения.

Если при проверке обнаруживается что-то необычное, жизненный цикл может быть остановлен на данном этапе, и разработчик может получить предупреждение о такой ситуации. Если обнаружено смещение выборки данных, ученый или инженер, специализирующийся на машинном обучении, может либо изменить критерии отбора отдельных выборок меток (например, выбрать только такое же количество выборок меток), либо изменить функцию потерь модели, запустить новый цикл конвейера для построения модели и перезапустить жизненный цикл модели.

Предварительная обработка данных

Вряд ли вы сможете использовать недавно собранные данные без обработки и напрямую обучать на них свою модель машинного обучения. Почти во всех случаях вам необходимо предварительно обработать данные, чтобы использовать их для обучения. Метки часто необходимо преобразовать в унитарные или многофакторные векторы¹. То же самое относится к входным данным модели. Если вы обучаете модель на текстовых данных, вам нужно преобразовать текстовые символы в индексы или текстовые маркеры в векторы слов. Поскольку предварительная обработка требуется только перед обучением модели, а не для каждого цикла обучения, имеет смысл выполнять предварительную обработку как отдельный этап жизненного цикла, перед обучением модели.

Для обработки данных существует множество различных инструментов. Список возможных вариантов решений поистине бесконечен – от простого сценария Python до сложных графовых моделей. Хотя большинство специалистов по работе с данными предпочитают использовать функции обработки данных, встроенные в любимые инструменты, также важно, чтобы изменения этапов предварительной обработки можно было связать с обработанными

¹ В задачах классификации с обучением, использующих несколько классов в качестве выходных данных, часто необходимо преобразовать категорию в вектор, например в унитарный вектор (0,1,0), или из списка категорий в вектор, например в многофакторный вектор (1,1,0).

данными, и наоборот. Это означает, что если кто-то изменяет этап обработки (например, вводит дополнительную метку в преобразование унитарного вектора), предыдущие обучающие данные должны стать недействительными, что, в свою очередь, приведет к перезапуску рабочего цикла конвейера машинного обучения. Об этом этапе конвейера рассказывается в главе 5.

Обучение и настройка модели

Этап обучения модели (см. главу 6) является основным этапом в работе конвейера машинного обучения. На этом этапе мы обучаем модель принимать входные данные и прогнозировать выходные данные с наименьшей возможной ошибкой. Для больших моделей, и в особенности для больших обучающих выборок, этот этап может быстро стать малоуправляемым. Поскольку память, как правило, является конечным ресурсом при выполнении вычислений, при обучении модели решающую роль играет эффективное распределение ресурсов.

В последнее время большое внимание уделяется настройке модели, поскольку она может привести к значительному улучшению качества и обеспечить конкурентное преимущество. В зависимости от того, как устроен ваш проект машинного обучения, вы можете настроить свою модель, прежде чем начнете думать о конвейерах машинного обучения, или же вы можете настроить ее как часть своего конвейера. Поскольку наши конвейеры масштабируемы, благодаря их базовой архитектуре мы можем запускать большое количество моделей параллельно или последовательно. Это позволяет выбрать оптимальные гиперпараметры модели для нашей результирующей промышленной модели.

Анализ модели

Как правило, мы будем использовать такие параметры, как точность или потери, для определения оптимального набора параметров модели. Но как только мы определились с окончательной версией модели, крайне полезно провести более глубокий анализ качества этой модели. Для этого может потребоваться вычислить другие показатели, такие как точность, полнота отклика и площадь под кривой (Area Under Curve, AUC), или же вычисление точности модели на наборе данных большего размера, нежели контрольный набор, используемый при обучении.

Еще одна причина для углубленного анализа модели заключается в проверке правильности прогнозов модели. Невозможно сказать, как модель будет работать для разных групп пользователей, если набор данных не будет разделен на части, и точность модели не будет рассчитываться для каждого такого поднабора. Мы также можем исследовать зависимость модели от признаков, используемых в обучении, и исследовать, как изменится предсказание модели, если мы изменим признаки в одном из обучающих наборов.

Подобно этапу настройки модели и окончательному выбору наиболее эффективной модели, на этом этапе рабочего процесса для модели требуется проверка специалиста по машинному обучению и искусственному интеллекту. Тем не менее мы покажем, как может быть автоматизирован весь процесс анализа, чтобы участие человека требовалось лишь на заключительном этапе. Автоматизация будет поддерживать анализ моделей в согласованном состоянии, пригодном для сравнения с другими процедурами и результатами анализа.

Управление версиями модели

Цель этапа управления версиями и проверки модели состоит в том, чтобы отслеживать, какая модель, какой набор гиперпараметров и какие наборы данных были выбраны в качестве следующей версии модели.

В основном на семантическом подходе управлении версиями в разработке ПО требуется увеличивать основной номер версии, когда вы вносите несовместимое изменение в свой API или добавляете новые функциональные возможности, внося существенные изменения. В противном случае вы бы увеличили дополнительный номер версии. Управление выпуском модели имеет еще одну степень свободы: набор данных. Существуют ситуации, когда вы можете достичь значительно отличающихся характеристик модели, не меняя определенный параметр модели или описание архитектуры, но предоставляя значительно больше данных для процесса обучения. Так что же, заметное повышение качества модели – повод для увеличения номера основной версии?

Хотя ответ на этот вопрос может быть разным для разных команд, работающих над проектами в области обработки данных и машинного обучения, важно документировать все входные данные для новой версии модели (гиперпараметры, наборы данных, архитектуру) и отслеживать их как часть этого этапа релиза.

Развертывание модели

После того как вы обучили, настроили и проанализировали свою модель, она готова к дальнейшему использованию. К сожалению, слишком много моделей развернуто как уникальные варианты реализации, и это делает процесс обновления моделей уязвимым местом.

Современные серверы моделей позволяют развертывать модели без разработки кода веб-приложений. Часто они предоставляют вам несколько интерфейсов API, таких как протоколы передачи представительного состояния (Representational State Transfer, REST) или удаленного вызова процедур (Remote Procedure Call, RPC), и позволяют одновременно размещать несколько версий одной и той же модели. Имея в своем распоряжении несколько версий одновременно, вы сможете выполнить A/B-тестирование на ваших моделях и получить ценную информацию об улучшениях вашей модели.

Серверы моделей также позволяют обновлять версию модели без повторного развертывания приложения, что сокращает время простоя вашего приложения и уменьшает необходимость в коммуникациях между разработчиками приложений и группами специалистов машинного обучения. Вопросы развертывания модели подробно обсуждаются в главах 8 и 9 этой книги.

Петли обратной связи

О последнем этапе жизненного цикла машинного обучения часто забывают, но он имеет решающее значение для успеха проектов в области науки о данных. Нам нужно замкнуть цикл и измерить эффективность и точность недавно развернутой модели.

На этом этапе мы можем собирать ценную информацию о качестве модели. В некоторых случаях мы можем собрать новые обучающие данные для увеличения наших наборов данных, чтобы обновить нашу модель. В петлях обратной связи может участвовать человек, либо этот этап может выполняться автоматически. Про петли обратной связи подробно рассказывается в главе 13.

За исключением двух шагов ручного просмотра, мы можем автоматизировать весь жизненный цикл. Специалисты по машинному обучению и искусственному интеллекту должны иметь возможность сосредоточиться на разработке новых моделей, а не на обновлении и поддержке существующих моделей.

Приватность данных

На момент написания этой книги вопросы приватности данных выходят за рамки стандартного жизненного цикла модели. Мы ожидаем, что это изменится в будущем, поскольку возрастает озабоченность пользователей в связи с использованием их данных и вводятся новые законы, ограничивающие использование персональных данных. Это приведет к интеграции методов сохранения приватности для машинного обучения в инструменты для построения конвейеров.

В главе 14 этой книги обсуждается несколько текущих вариантов повышения приватности в моделях машинного обучения:

- дифференцированная приватность, которая математически гарантирует, что предсказания модели не раскрывают данные пользователей;
- федеративное обучение, когда первичные данные не покидают пользовательское устройство;
- зашифрованное машинное обучение, когда либо весь процесс обучения может выполняться в зашифрованном пространстве, либо модель, обученная на первичных данных, может быть зашифрована.

ОРКЕСТРОВКА КОНВЕЙЕРА

Все компоненты конвейера машинного обучения, описанные в предыдущем разделе, должны быть выполнены или, как мы говорим, оркестрованы, чтобы порядок их выполнения был строго определен и не нарушался. Входные данные для каждого компонента должны вычисляться до того, как этот компонент будет выполнен. Эти шаги выполняются при помощи таких инструментов, как Apache Beam, Apache Airflow (эти инструменты обсуждаются в главе 11) или Kubeflow Pipelines для инфраструктуры Kubernetes (обсуждается в главе 12).

В то время как инструменты конвейера данных координируют этапы конвейера машинного обучения, хранилища артефактов конвейера, такие как TensorFlow ML MetadataStore, сохраняют выходные данные отдельных процессов. В главе 2 мы представим обзор MetadataStore TFX и заглянем за кулисы TFX и его компонентов конвейера.

Для чего нужна оркестровка конвейера

В 2015 году группа инженеров машинного обучения в Google пришла к выводу, что одна из причин, по которой проекты машинного обучения терпят неудачу, заключается в том, что большинство проектов используют пользова-

тельский код для интеграции между отдельными этапами конвейера машинного обучения¹. Однако этот нестандартный код нелегко перенести из одного проекта в другой. Исследователи обобщили свои выводы в статье «Скрытый технический долг в системах машинного обучения»². В этой статье авторы утверждают, что связующий код между этапами конвейера часто бывает уязвим и что, за редкими исключениями, пользовательские сценарии не масштабируются. Со временем были разработаны такие инструменты, как Apache Beam, Apache Airflow или Kubeflow Pipelines. Эти инструменты можно использовать для управления задачами конвейера машинного обучения; они обеспечивают стандартизированную оркестровку и абстракцию связующего кода.

Хотя на первый взгляд изучение нового инструмента (такого как Apache Beam или Airflow), новой интегрированной среды (например, Kubeflow) и настройка дополнительной инфраструктуры машинного обучения (например, Kubernetes) могут показаться слишком сложными, эти вложения времени окупятся очень быстро. Не применяя стандартизированные конвейеры машинного обучения, команды, использующие технологии машинного обучения и искусственного интеллекта, неизбежно столкнутся с такими препятствиями, как уникальные настройки проекта, произвольное расположение файлов журнала, уникальные шаги отладки и т. д. Список сложностей, с которыми потенциально можно столкнуться при такой организации проектов, может быть поистине бесконечным.

Направленные ациклические графы

Инструменты конвейеризации, такие как Apache Beam, Apache Airflow и Kubeflow, управляют потоком задач, используя представление зависимостей задач в виде графов.

Пример графа, приведенный на рис. 1.2, иллюстрирует наличие направленности шагов конвейера. Работа конвейера начинается с задачи A и заканчивается задачей E. Это означает, что путь выполнения четко определен зависимостями задач. Направленные графы помогают избежать ситуаций, когда некоторые задачи запускаются в то время, когда еще не все зависимые от них задачи определены. Поскольку мы знаем, что нам необходимо предварительно обработать наши обучающие данные перед обучением модели, представление конвейера в виде направленного графа не позволяет выполнить задачу обучения до завершения шага предварительной обработки данных.

Графы рабочего процесса конвейера также должны быть ациклическими, то есть в графе должны отсутствовать связи задач с ранее выполненными задачами. Если бы это условие не выполнялось, это означало бы существование бесконечных циклов работы конвейера и, следовательно, невозможности завершить рабочий процесс.

¹ Google запустил внутренний проект под названием Sibyl в 2007 году для управления внутренним промышленным конвейером машинного обучения. Однако в 2015 году эта тема привлекла более широкое внимание, когда коллективом авторов была опубликована статья «Скрытый технический долг в системах машинного обучения» (D. Sculley et al. «Hidden Technical Debt in Machine Learning Systems», <https://papers.nips.cc/paper/2015/file/86df7dcfd896fcdf2674f757a2463eba-Paper.pdf>).

² D. Sculley et al., «Hidden Technical Debt in Machine Learning Systems», Google, Inc. (2015).

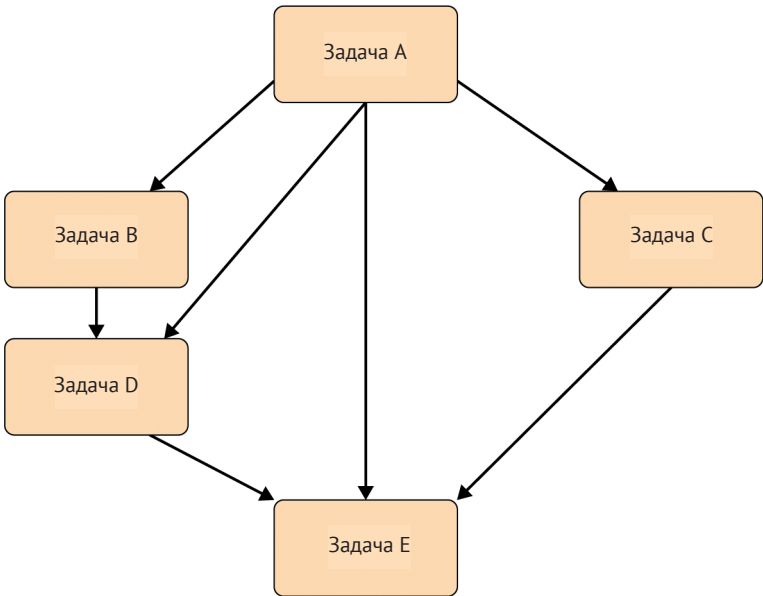


Рис. 1.2. Пример направленного ациклического графа

Из-за этих двух условий графы рабочего процесса конвейера являются направленными ациклическими графами (Directed Acyclic Graphs, DAGs). Далее вы сможете убедиться, что DAG – это основная концепция большинства инструментов рабочего процесса. Исполнение графов более подробно обсуждается в главах 11 и 12.

Наш демонстрационный проект машинного обучения

Чтобы пояснить содержимое глав этой книги на практических примерах, мы создали демонстрационный проект с использованием открытых данных и ПО с открытым исходным кодом. Используемый набор данных представляет собой жалобы потребителей на финансовые продукты в США и содержит комбинацию структурированных данных (категориальных/числовых данных) и неструктурированных данных (текста). Источником данных является Бюро по защите прав потребителей.

На рис. 1.3 показан фрагмент из этого набора данных.

	product	issue	consumer_complaint_narrative	company	state	company_response	timely_response	consumer_disputed
0	Mortgage	Loan servicing, payments, escrow account	My mortgage servicing provider (XXXX) transf...	SunTrust Banks, Inc.	TX	Closed with non-monetary relief	Yes	No
1	Debt collection	Cont'd attempts collect debt not owed	I HAVE NEVER RECEIVED ANY FORM OF NOTIFICATION...	ERC	CA	Closed with non-monetary relief	Yes	No
2	Debt collection	Disclosure verification of debt	i contacted walmart and the manager there said...	Synchrony Financial	MA	Closed with non-monetary relief	Yes	No
3	Credit reporting	Credit reporting company's investigation	I have filed multiple complaints XXXX on this ...	TransUnion Intermediate Holdings, Inc.	NY	Closed with explanation	Yes	Yes
4	Bank account or service	Account opening, closing, or management	Sofi has ignored my request to stop sending me...	Social Finance, Inc.	TX	Closed with explanation	Yes	No

Рис. 1.3. Образец данных

Задача машинного обучения заключается в том, чтобы, основываясь на данных о жалобе, предсказать, будет ли жалоба оспорена потребителем. В этом наборе данных оспаривается 30 % жалоб, поэтому набор данных не сбалансирован.

Структура проекта

Наш демонстрационный проект размещен в репозитории GitHub, и вы можете копировать его обычным способом, используя следующую команду:

```
$ git clone https://github.com/Building-ML-Pipelines/
  \ building-machine-learning-pipelines.git
```



Версии пакета Python

Для создания нашего демонстрационного проекта мы использовали Python 3.6–3.8. Мы использовали версии TensorFlow 2.2.0 и TFX 0.22.0. Мы сделаем все возможное, чтобы обновить репозиторий GitHub, загрузив в него последующие версии, однако мы не можем гарантировать, что проект будет работать с другими языками или версиями пакетов.

Структура нашего демонстрационного проекта:

- каталог *chapters*, который содержит блокноты для отдельных примеров из глав 3, 4, 7 и 14;
- каталог *components*, который содержит код для общих компонентов, таких как определение модели;
- завершенный интерактивный конвейер;
- пример эксперимента машинного обучения, который является отправной точкой для конвейера;
- законченные примеры конвейеров, оркестрованные с помощью Apache Beam, Apache Airflow и Kubeflow Pipelines;
- каталог *utility*, который содержит сценарий для загрузки данных.

В следующих главах мы проведем вас через шаги, необходимые для того, чтобы превратить пример эксперимента с машинным обучением (в нашем случае это блокнот Jupyter Notebook со структурой модели Keras) в законченный конвейер непрерывного машинного обучения.

Наша модель машинного обучения

Ядром нашего демонстрационного проекта глубокого обучения является модель, созданная функцией `get_model` в сценарии `components/module.py` нашего демонстрационного проекта. Модель предсказывает, оспорит ли потребитель жалобу, используя следующие признаки:

- финансовый продукт;
- подпродукт;
- ответ компании на жалобу;
- проблему, на которую пожаловался потребитель;
- штат США;
- почтовый индекс;
- текст жалобы (описание).

Для создания конвейера машинного обучения мы предполагаем, что проект архитектуры модели завершен, и мы не будем изменять модель. Мы обсудим архитектуру модели более подробно в главе 6; однако для этой книги архитектура модели – второстепенный вопрос. Здесь рассказывается о том, что вы можете делать с уже имеющейся у вас моделью.

Цель демонстрационного проекта

В ходе этой книги мы продемонстрируем необходимые структуры, компоненты и элементы инфраструктуры для непрерывного обучения нашей модели машинного обучения. Мы будем использовать стек, приведенный на схематической иллюстрации архитектуры на рис. 1.4.

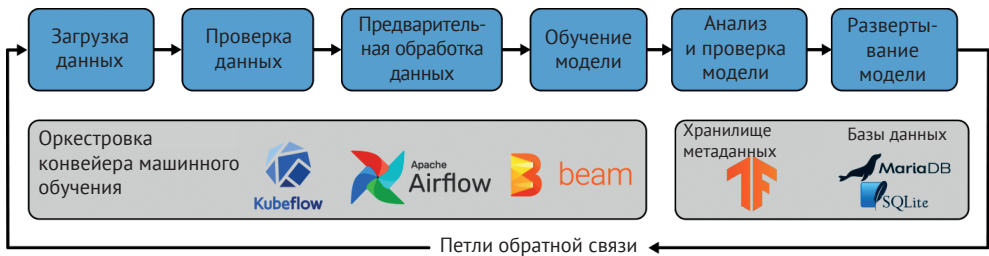


Рис. 1.4. Архитектура конвейера машинного обучения для нашего демонстрационного проекта

Мы попытались реализовать общую задачу машинного обучения, которую можно легко заменить вашей конкретной задачей машинного обучения. Структура и базовая настройка конвейера машинного обучения остаются прежними и могут быть перенесены на ваш вариант использования. Для каждого компонента потребуется выполнить некоторые настройки (например, указать, откуда брать данные), но, как мы увидим, объем этих настроек будет ограничен.

РЕЗЮМЕ

В этой главе мы представили концепцию конвейеров машинного обучения и показали отдельные этапы рабочего процесса конвейера. Мы также продемонстрировали преимущества автоматизации этого процесса. Кроме того, мы подготовили стартовую площадку для следующих глав, приведя краткое описание содержания каждой главы, и представили наш демонстрационный проект. В следующей главе мы приступим к построению нашего конвейера!

Глава 2

Введение в TensorFlow Extended

В предыдущей главе мы обсудили концепцию конвейеров машинного обучения и то, из каких компонентов состоит конвейер. В этой главе мы познакомимся с TensorFlow Extended (TFX). Библиотека TFX предоставляет все компоненты, которые нам понадобятся для наших конвейеров машинного обучения. Мы определяем задачи конвейера с помощью TFX, и затем их можно выполнять с помощью оркестровщика конвейера, такого как Airflow или Kubeflow Pipelines. На рис. 2.1 представлен обзор этапов конвейера и показано, как разные инструменты сочетаются друг с другом.

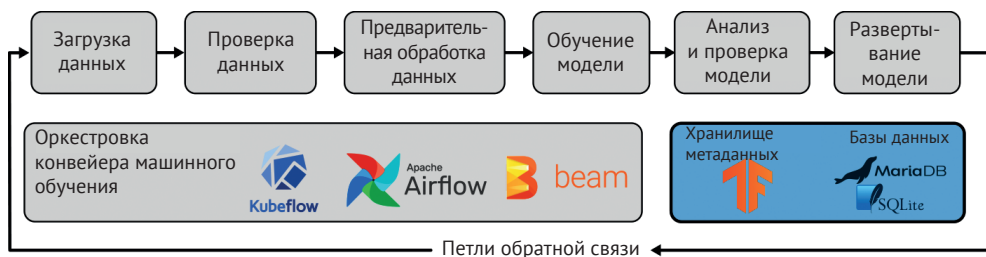


Рис. 2.1. TFX как часть конвейеров машинного обучения

В этой главе мы подробно рассмотрим установку TFX, объясняя основные концепции и терминологию, которые подготовят вас к прочтению следующих глав, в которых мы подробно рассмотрим отдельные компоненты, составляющие наши конвейеры. Здесь также представляем Apache Beam. Beam – это инструмент с открытым исходным кодом для определения и выполнения заданий по обработке данных. У него есть два способа применения в конвейерах TFX: во-первых, он работает «за кадром» многих компонентов TFX, выполняя такие этапы обработки, как проверка данных или предварительная обработка данных. Во-вторых, его можно использовать в качестве оркестровщика конвейера, как мы обсуждали в главе 1. Мы представляем Beam здесь, потому что он поможет вам понять, как устроены компоненты TFX. Это важно понимать, если вы хотите писать собственные компоненты, как будет показано в главе 10.

Что такое TFX?

Конвейеры машинного обучения могут быть очень сложными, и в такой ситуации им потребуется много ресурсов для управления зависимостями задач. В то же время конвейеры машинного обучения могут включать в себя множество задач, в том числе задачи проверки данных, предварительной обработки, обучения модели и любые задачи, выполняемые после обучения. Как мы обсуждали в главе 1, связи между задачами часто бывают нестабильными и приводят к сбою конвейеров. Эти связи названы связующим кодом в публикации «Скрытый технический долг в системах машинного обучения». Наличие нестабильных связей в конечном итоге означает, что производственные модели будут обновляться нечасто, а специалисты по обработке данных и инженеры по машинному обучению ненавидят обновление устаревших моделей. Для конвейеров также необходимо организовать хорошо управляемую распределенную обработку, поэтому TFX использует Apache Beam. Это особенно актуально для больших рабочих нагрузок.

Google столкнулся с подобной проблемой внутри компании и решил разработать платформу для упрощения определений конвейера и минимизации объема написанного шаблонного кода задачи. TFX представляет собой версию внутреннего конвейера машинного обучения Google с открытым исходным кодом.

На рис. 2.2 показана общая архитектура конвейера с TensorFlow Extended. Инструменты оркестровки конвейера являются основой для выполнения наших задач. Помимо инструментов оркестровки, нам необходимо хранилище данных для отслеживания промежуточных результатов работы конвейера. Отдельные компоненты подключаются к хранилищу данных для получения входных данных и возвращают результаты обратно в хранилище данных. Эти результаты могут затем использоваться в качестве входных данных для следующих задач. TFX представляет уровень, объединяющий все эти инструменты, и предоставляет отдельные компоненты для реализации основных задач конвейера.

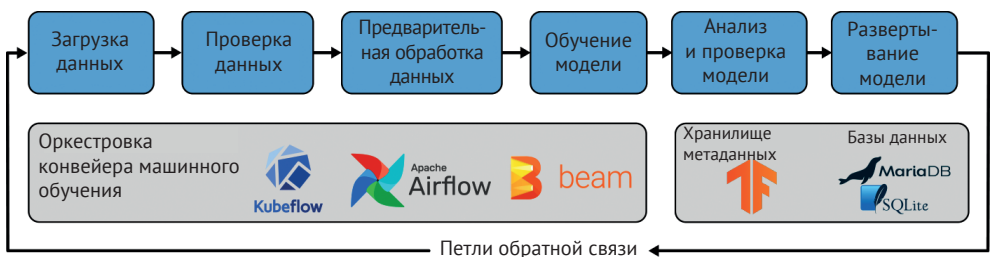


Рис. 2.2. Архитектура конвейера машинного обучения

Первоначально Google открыл исходный код некоторых функциональных возможностей конвейера в виде библиотек TensorFlow (например, TensorFlow Serving обсуждается в главе 8), опубликовав их как расширенные библиотеки TensorFlow. Весной 2019 года Google открыл *связующий код* конвейера, содержащий все необходимые компоненты, чтобы связать библиотеки и автоматически создать определения конвейера для инструментов оркестровки, таких как Apache Airflow, Apache Beam или Kubeflow Pipelines.

TFX предоставляет множество компонентов конвейера, которые охватывают большое количество вариантов использования. На момент написания этой книги доступны следующие компоненты:

- загрузка данных с помощью ExampleGen;
- проверка данных с помощью StatisticsGen, SchemaGen, а также ExampleValidator;
- предварительная обработка данных с помощью Transform;
- обучение модели с помощью Trainer;
- анализ и проверка модели с помощью Evaluator и ModelValidator соответственно;
- развертывание модели с помощью Pusher.

На рис. 2.3 показано, как компоненты конвейера и библиотеки работают друг с другом.

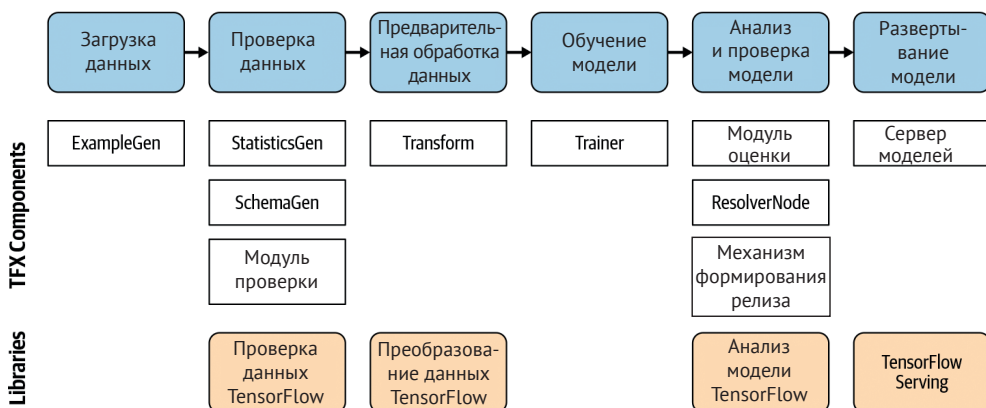


Рис. 2.3. Компоненты и библиотеки TFX

Мы обсудим компоненты и библиотеки более подробно в следующих главах. Если ваши потребности в области задач машинного обучения находятся за рамками стандартных функциональных возможностей, в главе 10 мы расскажем, как создавать пользовательские компоненты конвейера.



Стабильная версия TFX

На момент написания этой главы стабильная версия 1.X TFX еще не выпущена. API TFX, упомянутый в этой и следующих главах, в будущем может быть обновлен. Насколько нам известно, все примеры из этой книги будут работать с TFX версии 0.22.0.

УСТАНОВКА TFX

TensorFlow Extended можно легко установить, выполнив следующую команду инсталлятора Python:

```
$ pip install tfx
```

Пакет `tfx` поставляется со множеством дополнительных (зависимых) пакетов, которые будут установлены автоматически. При его установке устанавливаются не только отдельные пакеты TFX Python, как, например, TensorFlow Data Validation, но и зависимые пакеты, такие как Apache Beam.

После установки TensorFlow и TensorFlow Extended вы можете импортировать отдельные пакеты Python. Мы рекомендуем придерживаться этого подхода, если вы хотите использовать отдельные пакеты TFX (например, хотите проверить набор данных с помощью TensorFlow Data Validation, как будет обсуждаться в главе 4):

```
import tensorflow_data_validation as
tfdv import tensorflow_transform as tft
import tensorflow_transform.beam as tft_beam
...
```

или соответствующий компонент TFX (если используются компоненты в контексте конвейера):

```
from tfx.components import ExampleValidator
from tfx.components import Evaluator
from tfx.components import Transform
...
```

ОБЗОР КОМПОНЕНТОВ TFX

Компонент обрабатывает более сложный процесс, чем просто выполнение одной задачи. Все компоненты конвейера машинного обучения выполняют операцию чтения из канала, чтобы получить входные артефакты из хранилища метаданных. Затем данные загружаются из пути, указанного в хранилище метаданных, и обрабатываются. Выходные данные компонента – обработанные данные – потом передаются следующим компонентам конвейера. Общими внутренними компонентами всегда являются:

- получение определенных входных данных;
- выполнение действий;
- сохранение окончательного результата.

В терминах TFX три внутренние составные части компонента называются *driver*, *executor* и *publisher*. Driver обрабатывает запросы к хранилищу метаданных. Executor выполняет действия компонентов. И наконец, publisher управляет сохранением выходных метаданных в хранилище метаданных. Driver и publisher не перемещают какие-либо данные, а вместо этого считывают и записывают ссылки в хранилище метаданных. На рис. 2.4 показана структура компонента TFX.

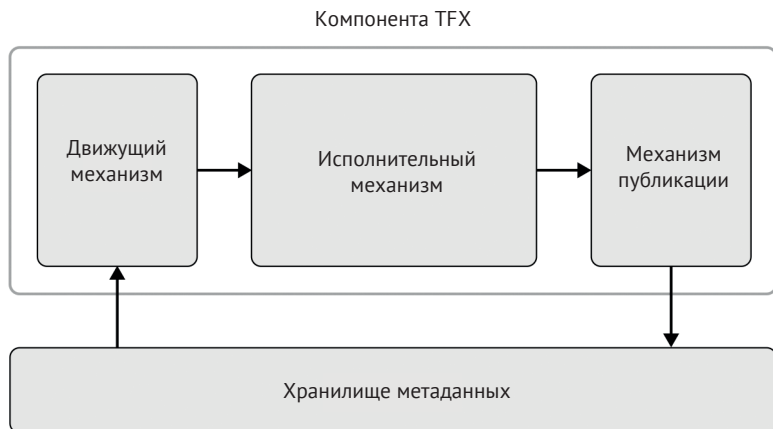


Рис. 2.4. Обзор компонентов

Входы и выходы компонентов называются артефактами (artifacts). Примерами артефактов являются необработанные входные данные, предварительно обработанные данные и обученные модели. Каждый артефакт связан с метаданными, хранящимися в хранилище метаданных. Метаданные артефакта состоят из типа артефакта, а также свойств артефакта. Эта структура артефакта гарантирует, что компоненты могут эффективно обмениваться данными. В настоящее время TFX предоставляет десять различных типов артефактов, которые мы более подробно рассмотрим в следующих главах.

Что такое метаданные ML Metadata?

Компоненты TFX «общаются» с помощью *метаданных*; вместо того чтобы передавать артефакты непосредственно между компонентами конвейера, компоненты потребляют и публикуют ссылки на артефакты конвейера. Артефактом может быть, например, необработанный набор данных, граф преобразования или экспортированная модель. Следовательно, метаданные – это основа наших конвейеров TFX. Одним из преимуществ передачи между компонентами метаданных, а не непосредственно артефактов является то, что информация может храниться централизованно.

На практике рабочий процесс выглядит следующим образом: когда мы выполняем компонент, он использует API метаданных ML Metadata (MLMD) для сохранения метаданных, соответствующих запуску на исполнение. Например, драйвер компонента получает ссылку на необработанный набор данных из хранилища метаданных. После выполнения компонента издатель компонента сохранит ссылки на выходные данные компонента в хранилище метаданных. MLMD последовательно сохраняет метаданные в MetadataStore, созданном на основе серверной части хранилища (бэкенда). В настоящее время MLMD поддерживает три типа бэкендов:

- базы данных в памяти (через SQLite);
- SQLite;
- MySQL.

Поскольку компоненты TFX постоянно отслеживаются, метаданные ML Metadata предоставляют множество полезных возможностей. Мы можем сравнить два артефакта из одного компонента (мы рассмотрим конкретный пример в главе 7, когда будем обсуждать проверку модели). В этом конкретном случае TFX сравнивает результаты анализа модели нашего текущего прогона с результатами предыдущего прогона, чтобы определить, имеет ли недавно обученная модель лучшую точность по сравнению с предыдущим результатом обучения или же нет). Метаданные также можно использовать для определения всех артефактов, основанных на другом, ранее созданном артефакте. Таким образом создается своего рода контрольный журнал для наших конвейеров машинного обучения.

На рис. 2.5 показано, что каждый компонент взаимодействует с MetadataStore, а MetadataStore хранит метаданные в предоставленной серверной части базы данных.

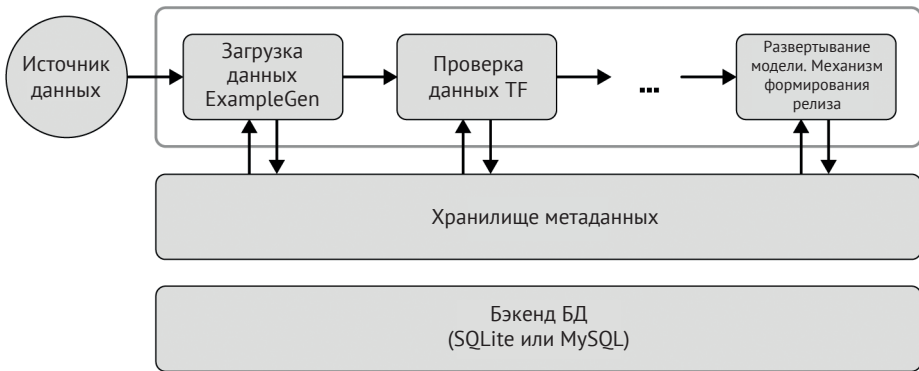


Рис. 2.5. Хранение метаданных в MLMD

На рис. 2.5 показано, что каждый компонент взаимодействует с MetadataStore, а MetadataStore хранит метаданные в базе данных, размещенной на бэкенде.

Проектирование и реализация конвейеров машинного обучения иногда могут стать причиной разочарования. Например, бывает сложно отладить компоненты в конвейере. Вот почему инструментарий TFX, построенный вокруг интерактивных конвейеров, столь популярен. В следующих главах мы будем поэтапно реализовывать конвейер машинного обучения и демонстрировать варианты реализации через интерактивный конвейер. Конвейер работает в Jupiter Notebook, и артефакты компонентов в этой интерактивной среде можно сразу же просмотреть. После проверки подтверждения полной функциональности вашего конвейера мы обсудим в главах 11 и 12, как можно преобразовать ваш интерактивный конвейер в готовый к работе конвейер, например, для работы в Apache Airflow.

Любой интерактивный конвейер программируется в контексте Jupiter Notebook или пользовательской сессии Google Colab. В отличие от инструментов оркестровки, которые будут обсуждаться в главах 11 и 12, «оркестровка» интерактивных конвейеров выполняется пользователем.

Вы можете запустить интерактивный конвейер, импортировав необходимые пакеты:


```
import tensorflow as tf
from tfx.orchestration.experimental.interactive.interactive_context import
    \ InteractiveContext
```

После того как нужные пакеты импортированы, вы можете создать объект `context`. Объект `context` обрабатывает выполнение компонента и отображает артефакты компонентов. На этом этапе `InteractiveContext` также настраивает простое хранилище метаданных машинного обучения в памяти:

```
context = InteractiveContext()
```

После настройки компонентов конвейера (например, `StatisticsGen`) вы можете выполнить каждый объект компонента, используя функцию `run` объекта `context`, как показано в следующем примере:

```
from tfx.components.statistics_gen.component import StatisticsGen

statistics_gen = StatisticsGen(
    examples=example_gen.outputs['examples'])
context.run(statistics_gen)
```

Сам компонент получает выходные данные предыдущего компонента (в нашем случае это компонент, выполняющий получение, внесение и обработку данных `ExampleGen`) в качестве аргумента операции создания экземпляра. После выполнения задач компонент автоматически записывает метаданные выходного артефакта в хранилище метаданных. Выводимые данные для некоторых компонентов могут при этом отображаться в вашем рабочем блокноте `Notebook`. Непосредственная доступность результатов и визуализаций удобна, например, для компонента `StatisticsGen` при проверке признаков набора данных.

```
context.show(statistics_gen.outputs['statistics'])
```

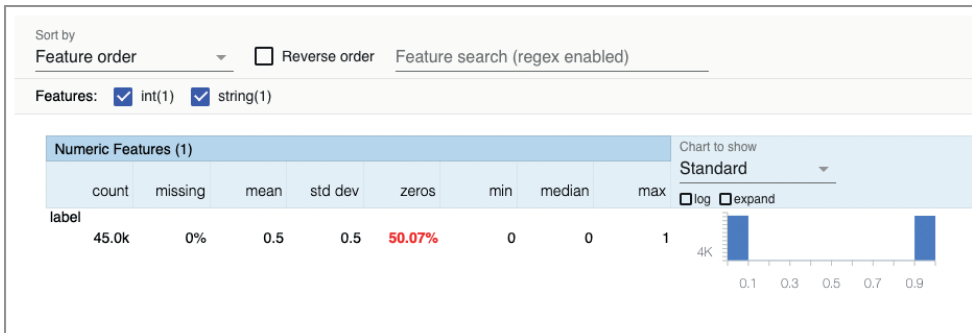
После запуска предыдущей функции `context` мы можем увидеть графическое представление статистики набора данных в вашем рабочем блокноте `Notebook`, как показано на рис. 2.6.

Иногда может быть полезно выполнять программную проверку выходных артефактов компонента. После того как объект компонента был запущен, мы можем получить доступ к свойствам артефакта, как показано в следующем примере. Набор свойств зависит от конкретного артефакта:

```
for artifact in statistics_gen.outputs['statistics'].get():
    print(artifact.split, artifact.uri)
```

Результат выполнения этого примера:

```
'/tmp/tfx-interactive-2020-05-15T04_50_16.251447/StatisticsGen/statistics/2'
```



Artifact at /tmp/tfx-interactive-2020-01-14T00_11_24.275697-n7laziqr/StatisticsGen/output/2/eval/

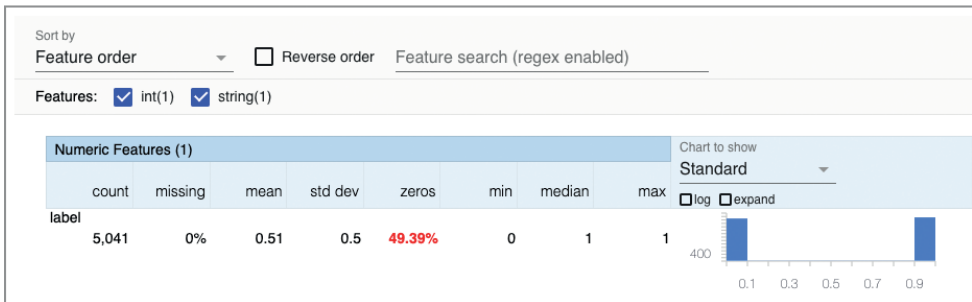


Рис. 2.6. Интерактивные конвейеры предоставляют возможности визуальной проверки наборов данных

В следующих главах мы проиллюстрируем запуск каждого компонента в интерактивном контексте. Затем в главах 11 и 12 покажем весь конвейер и варианты его экспортирования как в Airflow, так и в Kubeflow.

АЛЬТЕРНАТИВЫ TFX

Прежде чем мы углубимся в изучение компонент TFX в следующих главах, давайте рассмотрим альтернативы TFX. В последние несколько лет оркестровка конвейеров машинного обучения была серьезной инженерной задачей, и не удивительно, что многие крупные компании в Кремниевой долине разработали собственную инфраструктуру оркестровки. В табл. 3.1 перечислены некоторые интегрированные среды разработки.

Таблица 2.1. Использование интегрированных сред разработки для оркестровки конвейеров машинного обучения

Компания	Наименование интегрированной среды разработки	Ссылка
AirBnb	AeroSolve	https://github.com/airbnb/aerosolve
Stripe	Railyard	https://stripe.com/blog/railyard-training-models
Spotify	Luigi	https://github.com/spotify/luigi
Uber	Michelangelo	https://eng.uber.com/michelangelo/

Поскольку эти интегрированные среды разработки создавались крупными корпорациями, они разрабатывались с учетом специфики используемых технологий. Например, среда AeroSolve, разработанная в AirBnB, фокусируется на коде вывода на основе Java, а Luigi Spotify фокусируется на эффективной оркестровке. TensorFlow Extended ничем не отличается в этом отношении. На данный момент реализация архитектуры и структур данных предполагает, что вы используете TensorFlow (или Keras) в качестве среды машинного обучения. Некоторые компоненты TFX могут использоваться в сочетании с другими системами машинного обучения. Например, можно выполнить анализ данных с помощью TensorFlow Data Validation, а затем использовать эти данные в модели Scikit Learn. Однако структура TFX тесно связана с моделями TensorFlow или Keras. Поскольку TFX поддерживается сообществом TensorFlow, и все больше таких компаний, как Spotify, внедряют TFX, мы считаем, что это стабильная и зрелая среда, которая в конечном итоге получит более широкое применение у инженеров, специализирующихся на машинном обучении.

Знакомство с АРАЧЕ БЕАМ

Различные компоненты и библиотеки TFX (например, TensorFlow Transform) используют Apache Beam для эффективной обработки данных конвейера. Поскольку Apache Beam очень важен для экосистемы TFX, мы хотели бы вкратце познакомить вас с тем, как Apache Beam работает «за кулисами» компонентов TFX. Затем в главе 11 мы обсудим, как использовать Apache Beam для другой цели: в качестве инструмента оркестровщика конвейера.

Apache Beam предлагает вам открытый, независимый от поставщика способ описания этапов обработки данных, которые затем можно выполнить в различных средах. Поскольку Apache Beam универсален, его можно использовать для описания пакетных процессов, потоковых операций и конвейеров данных. Фактически TensorFlow Extended опирается на Apache Beam и использует его, так сказать, «за кадром», в различных компонентах (например, TensorFlow Transform или TensorFlow Data Validation). Мы обсудим конкретное использование Apache Beam в экосистеме TensorFlow Extended в соответствующих главах, например когда будем говорить об инструменте проверки данных TensorFlow Data Validation (в главе 4) и TensorFlow Transform (в главе 5).

Поскольку Apache Beam абстрагирует логику обработки данных от используемых им вспомогательных инструментов среды выполнения, он может выполняться в нескольких распределенных рабочих средах обработки данных. Это означает, что вы можете запустить один и тот же конвейер данных в Apache Spark или Google Cloud DataFlow, не выполняя никаких изменений в описании конвейера. Кроме того, Apache Beam также был разработан не только для описания процессов пакетной обработки, но и для поддержки непрерывного режима работы, реализованной таким образом, чтобы пользователям не приходилось выполнять никаких действий для этого.

Установка

Установка Apache Beam не вызывает сложностей. Вы можете установить последнюю версию, выполнив следующую команду:

```
$ pip install apache-beam
```

Если вы планируете использовать Apache Beam в контексте Google Cloud Platform, например если вы хотите обрабатывать данные из Google BigQuery или запускать конвейеры данных в Google Cloud Dataflow (как описано в разделе «Обработка больших наборов данных с помощью Google Cloud Platform» на стр. 80), вам следует использовать следующую команду для установки Apache Beam:

```
$ pip install 'apache-beam[gcp]'
```

Если вы планируете использовать Apache Beam в контексте Amazon Web Services (AWS) (например, если хотите загружать данные из корзины S3), вам следует установить Apache Beam следующим образом:

```
$ pip install 'apache-beam[boto]'
```

Если вы выполняете установку TensorFlow Extended с помощью системы управления пакетами Python `pip`, Apache Beam будет установлен автоматически.

Базовый конвейер

Абстракция Apache Beam основана на двух концепциях: коллекции (Collections) и преобразования (Transformations). С одной стороны, коллекции Beam описывают операции, когда данные считываются/записываются для/в определенный файл или поток. Все коллекции и преобразования выполняются в контексте конвейера (в Python это реализуется с помощью команды менеджера контекста `with`). Когда мы определяем наши коллекции или преобразования в нашем следующем примере, никакие данные на самом деле не загружаются и не преобразуются. Это происходит только тогда, когда конвейер работает в контексте среды выполнения, например `DirectRunner` (для Apache Beam), `Apache Spark`, `Apache Flink` или `Google Cloud Dataflow`.

Базовый пример коллекции

Работа конвейера данных обычно начинается и заканчивается чтением или записью данных, которые обрабатываются в Apache Beam посредством коллекций, часто называемых `PCollections`. Затем коллекции преобразуются, и конечный результат снова может быть представлен как коллекция и записан в файловую систему.

В следующем примере показано, как прочитать текстовый файл и вернуть все строки этого текстового файла.

```
import apache_beam as beam

with beam.Pipeline() as p: ❶
    lines = p | beam.io.ReadFromText(input_file) ❷
```

- ❶ Менеджер контекста для определения конвейера
- ❷ Считывание текста в `PCollection`

Подобно операции `ReadFromText`, Apache Beam предоставляет функции для записи коллекций в текстовый файл (например, в `WriteToText`). Операция записи обычно выполняется после завершения всех преобразований:

```
with beam.Pipeline() as p:
    ...
    output | beam.io.WriteToText(output_file) ❶
```

❶ Запись `output` в файл `output_file`

Базовый пример преобразования

В Apache Beam данные обрабатываются с помощью преобразований. Как мы увидим в этом примере и как будет показано в дальнейшем в главе 5, преобразования можно объединить в цепочку с помощью оператора `|`. Если вы объединяете несколько преобразований в цепочку, вы должны указать имя для операции, помеченное строковым идентификатором между оператором `|` и прямоугольными скобками.

В следующем примере мы последовательно применяем все преобразования к строкам, извлеченным из текстового файла:

```
counts = (
    lines
    | 'Split' >> beam.FlatMap(lambda x: re.findall(r'[A-Za-z\']+', x))
    | 'PairWithOne' >> beam.Map(lambda x: (x, 1))
    | 'GroupAndSum' >> beam.CombinePerKey(sum))
```

Давайте подробно рассмотрим этот код. В качестве примера возьмем фразы «*Hello, how do you do?*» и «*I am well, thank you*».

Преобразование `Split` использует `re.findall` для разделения каждой строки на список токенов; в результате мы получаем:

```
["Hello", "how", "do", "you", "do"]
["I", "am", "well", "thank", "you"]
```

`beam.FlatMap` преобразует результат в коллекцию `PCollection`:

```
"Hello" "how" "do" "you" "do" "I" "am" "well" "thank" "you"
```

Затем преобразование `PairWithOne` использует `beam.Map` для создания кортежа из каждого токена и счетчика (1 для каждого результата):

```
("Hello", 1) ("how", 1) ("do", 2) ("you", 2) ("I", 1) ("am", 1) ("well", 1)
("thank", 1)
```

Наконец, преобразование `GroupAndSum` суммирует все отдельные кортежи для каждого токена:

```
("Hello", 1) ("how", 1) ("do", 2) ("you", 2) ("I", 1) ("am", 1) ("well", 1)
("thank", 1)
```

Вы также можете использовать функции Python как часть преобразования. В следующем примере показано, как функцию `format_result` можно применить к ранее полученным результатам суммирования. Функция преобразует полученные кортежи в строку, которую затем можно записать в текстовый файл:

```
def format_result(word_count):
    """ Преобразование кортежей (token, count) в строку """

    (word, count) = word_count
    return "{}: {}".format(word, count)

output = counts | 'Format' >> beam.Map(format_result)
```

Apache Beam предоставляет множество predefined преобразований. Однако если подходящей вам операции нет, вы можете написать свои собственные преобразования, используя оператор `Map`. Просто имейте в виду, что операции должны выполняться распределенно, чтобы в полной мере использовать возможности среды исполнения.

Собираем все вместе

Теперь, после того как мы обсудили отдельные концепции конвейеров Apache Beam, давайте соединим все концепции в одном примере. Предыдущие фрагменты и следующие примеры являются модифицированной версией примера, взятого из документации *Apache Beam introduction*, размещенной по адресу <https://beam.apache.org/get-started/wordcount-example/>. Для удобства чтения пример приведен к минимальному коду Apache Beam.

```
import re

import apache_beam as beam
from apache_beam.io import ReadFromText
from apache_beam.io import WriteToText
from apache_beam.options.pipeline_options import PipelineOptions
from apache_beam.options.pipeline_options import SetupOptions

input_file = "gs://dataflow-samples/shakespeare/kinglear.txt"  output_file = "/tmp/output.txt" ❶

# Определяет объект параметров конвейера.
pipeline_options = PipelineOptions()

with beam.Pipeline(options=pipeline_options) as p: ❷
    # Читывает текстовый файл [шаблон] в PCollection.
    lines = p | ReadFromText(input_file) ❸

    # Подсчитывает число вхождений каждого слова.
    counts = ( ❹
        lines
        | 'Split' >> beam.FlatMap(lambda x: re.findall(r'[A-Za-z\']+', x))
        | 'PairWithOne' >> beam.Map(lambda x: (x, 1))
        | 'GroupAndSum' >> beam.CombinePerKey(sum))
```

```
# Преобразует счетчики в коллекцию строк (Pcollection).
def format_result(word_count):
    (word, count) = word_count
    return "{}: {}".format(word, count)

output = counts | 'Format' >> beam.Map(format_result)

# Записывает и выводит результат, используя преобразование "Write", имеющее некоторые
# побочные эффекты.
output | WriteToText(output_file)
```

- ❶ Текст хранится в сегменте Google Cloud Storage
- ❷ Настройка конвейера Apache Beam
- ❸ Создание коллекции данных путем чтения текстового файла
- ❹ Выполнение преобразований над коллекцией

Приведенный пример конвейера загружает произведение Шекспира «Король Лир» и выполняет подсчет токенов во всем тексте. Затем результаты записываются в текстовый файл, размещенный в `/tmp/output.txt`.

Запуск элементарного конвейера

В данном примере вы можете запустить конвейер с помощью `DirectRunner` в Apache Beam, выполнив следующую команду (предполагается, что предыдущий пример кода был сохранен в файле `basic_pipeline.py`). Если вы хотите запустить этот конвейер в разных средах Apache Beam, таких как Apache Spark или Apache Flink, вам нужно будет выполнить настройки конфигурации конвейера, используя объект `pipeline_options`:

```
python basic_pipeline.py
```

Результаты преобразований можно найти в указанном текстовом файле:

```
$ head /tmp/output.txt-00000-of-00001
KING: 243
LEAR: 236
DRAMATIS: 1
PERSONAE: 1
king: 65
...
```

РЕЗЮМЕ

В этой главе мы представили общий обзор TFX и поняли важную роль хранилища метаданных, а также общее внутреннее устройство компонента TFX. Мы также представили Apache Beam и показали, как выполнять простое преобразование данных с помощью Beam.

Материалы, которые мы обсуждали в этой главе, принесут вам практическую пользу, когда вы прочитаете главы 3–7, посвященные компонентам конвейера и оркестрации конвейера, описанным в главах 11 и 12. Чтобы сделать первый шаг, нужно поместить ваши данные в конвейер, и в главе 3 мы покажем, как это сделать.

Глава 3

Загрузка данных

После того как мы выполнили базовую настройку TFX и ML MetadataStore, в этой главе мы сосредоточимся на том, как загрузить ваши наборы данных в конвейер для использования в различных компонентах, как показано на рис. 3.1.

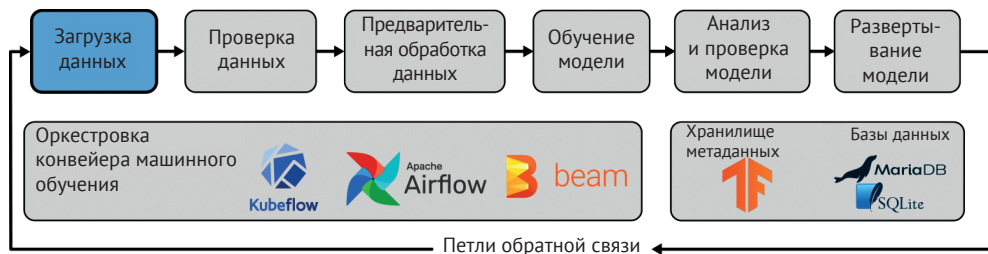


Рис. 3.1. Загрузка данных как часть конвейеров машинного обучения

TensorFlow Extended предоставляет нам компонент для получения данных из файлов или сервисов. В этой главе мы приведем основные понятия, объясним способы разделения наборов данных на обучающие и оценочные подмножества данных и продемонстрируем, как объединить несколько результатов экспорта данных в один комплексный набор данных.

Затем мы обсудим некоторые стратегии использования различных видов данных (структурированные, текстовые и графические), с которыми мы работали в предыдущих сценариях использования.

КОНЦЕПЦИИ ЗАГРУЗКИ ДАННЫХ

На этом этапе нашего конвейера мы читаем файлы данных или запрашиваем данные для нашего конвейера из внешней службы (например, Google Cloud BigQuery). Перед передачей полученного набора данных следующему компоненту мы разделяем доступные данные на набор обучающих данных и набор оценочных данных, а затем преобразуем эти наборы данных в TFRecords, где эти данные будут представлены в виде структур данных `tf.Example`.

TFRecord

TFRecord – это облегченный формат, оптимизированный для *поточной* передачи больших наборов данных. Хотя на практике большинство пользователей TensorFlow хранят сериализованные примеры буферов протокола в файлах TFRecord, формат файла TFRecord фактически поддерживает любые двоичные данные, как можно видеть в примере, приведенном ниже:

```
import tensorflow as tf

with tf.io.TFRecordWriter(«test.tfrecord») as w:
    w.write(b»First record»)
    w.write(b»Second record»)

for record in tf.data.TFRecordDataset(«test.tfrecord»):
    print(record)

tf.Tensor(b'First record', shape=(), dtype=string)
tf.Tensor(b'Second record', shape=(), dtype=string)
```

Если файлы TFRecord содержат записи `tf.Example`, каждая запись содержит одну или несколько функций, которые будут представлять столбцы в наших данных. Затем данные сохраняются в двоичных файлах, которые можно эффективно переваривать. Если вас интересует внутреннее устройство файлов TFRecord, мы рекомендуем ознакомиться с документацией TensorFlow (см. https://www.tensorflow.org/tutorials/load_data/tfrecord).

Хранение данных в виде TFRecords дает несколько преимуществ:

- 1) структура данных не зависит от системы, поскольку для сериализации данных она использует кросс-платформенную кросс-языковую библиотеку Protocol Buffers;
- 2) TFRecord оптимизирована для быстрой загрузки или записи больших объемов данных;
- 3) `tf.Example`, структура данных, представляющая строку данных в TFRecords, является структурой данных, используемой по умолчанию в экосистеме TensorFlow, и поэтому используется во всех компонентах TFX.

Процесс приема, разделения и преобразования наборов данных выполняется компонентом `ExampleGen`. Как мы видим в следующих примерах, наборы данных можно читать из локальных и удаленных папок, а также запрашивать у служб данных, таких как Google Cloud BigQuery.

Загрузка локальных файлов данных

Компонент ExampleGen может принимать несколько структур данных, в том числе CSV-файлов, предварительно подготовленных файлов TFRecords, а также сериализованных данных, получаемых из Apache Avro и Apache Parquet.

Преобразование данных в формате CSV в tf.Example

Наборы данных в случае использования структурированных или текстовых данных часто хранятся в файлах данных с разделителями-запятыми (Comma separated value, CSV). TFX предоставляет функциональные возможности для чтения и преобразования этих файлов в структуры данных *tf.Example*. Следующий пример кода демонстрирует загрузку каталога, содержащего данные в формате CSV, используемые в нашем демонстрационном проекте.

```
import os
from tfx.components import CsvExampleGen
from tfx.utils.dsl_utils import external_input

base_dir = os.getcwd()
data_dir = os.path.join(os.pardir, "data")
examples = external_input(os.path.join(base_dir, data_dir)) ❶
example_gen = CsvExampleGen(input=examples) ❷
context.run(example_gen) ❸
```

- ❶ Укажите путь к данным
- ❷ Создайте экземпляр компонента конвейера
- ❸ Запустите компонент в интерактивном режиме

Если вы запустите компонент как часть интерактивного конвейера, метаданные прогона будут отображены в Jupyter Notebook. Выходные данные, полученные в результате работы компонента, показаны на рис. 3.2. На рисунке выделены места хранения обучающих и оценочных наборов данных.

▼ ExecutionResult at 0x134603ef0

.execution_id 1

.component ▶ CsvExampleGen at 0x134603f60

.component.inputs

[input_base] ▶ Channel of type 'ExternalPath' (1 artifact) at 0x134603f28

.component.outputs

[examples] ▼ Channel of type 'ExamplesPath' (2 artifacts) at 0x134603e48

.type_name	ExamplesPath
.artifacts	<p>[0] ▶ Artifact of type 'ExamplesPath' (uri: /var/folders/_9/_7pgq8cd62s_m3rc63jjg5v00000gn/T/tfx-interactive-2020-01-26T15_11_50.585382-0uvnzhl1/CsvExampleGen/examples/1/train/) at 0x13461e0b8</p> <p>[1] ▶ Artifact of type 'ExamplesPath' (uri: /var/folders/_9/_7pgq8cd62s_m3rc63jjg5v00000gn/T/tfx-interactive-2020-01-26T15_11_50.585382-0uvnzhl1/CsvExampleGen/examples/1/eval/) at 0x13461e0f0</p>

Рис. 3.2. Выходные данные компонента ExampleGen



Структура каталогов

Ожидается, что путь `ExampleGen`, указанный для входных данных, содержит только файлы данных. Компонент пытается использовать все существующие файлы на уровне пути. Любые дополнительные файлы, например файлы метаданных, не обрабатываются компонентом, и их использование приведет к ошибке выполнения шага компонента. Компонент также не использует данные из подкаталогов указанного каталога, если такая настройка не указана в шаблоне входных данных.

Импорт существующих файлов `TFRecord`

Иногда наши данные не могут храниться в виде файлов в формате CSV, например когда мы хотим загрузить изображения для задач компьютерного зрения или большие текстовые массивы для задач обработки естественного языка. В таких случаях рекомендуется преобразовать эти наборы данных в структуры данных `TFRecord`, а затем загрузить сохраненные файлы `TFRecord`, используя компонент `ImportExampleGen`. Если вы хотите выполнить преобразование ваших данных в файлы `TFRecord` как часть конвейера, обратитесь к главе 10, в которой мы обсуждаем разработку пользовательских компонентов `TFX`, включая компонент загрузки данных. Файлы `TFRecord` можно загружать так, как показано в следующем примере:

```
import os
from tfx.components import ImportExampleGen
from tfx.utils.dsl_utils import external_input

base_dir = os.getcwd()
data_dir = os.path.join(os.pardir, "tfrecord_data")
examples = external_input(os.path.join(base_dir, data_dir))
example_gen = ImportExampleGen(input=examples)

context.run(example_gen)
```

Поскольку наборы данных уже сохранены как записи `tf.Example` в файлах `TFRecord`, они могут быть импортированы и не нуждаются ни в каком преобразовании. Компонент `ImportExampleGen` обрабатывает этот шаг импорта.

Преобразование сериализованных данных `Parquet` в `tf.Example`

В главе 2 мы обсудили внутреннюю архитектуру компонентов `TFX` и то, что поведение компонента определяется его объектом `executor`. Если мы хотим загрузить новые типы файлов в наш конвейер, мы можем переписать `executor_class`, вместо того чтобы разрабатывать совершенно новый компонент.

`TFX` включает классы `executor` для загрузки файлов различных типов, в том числе сериализованных данных `Parquet`. В следующем примере показано, как вы можете перезаписать `executor_class`, чтобы изменить поведение при загрузке. Вместо использования компонента `CsvExampleGen` или `ImportExampleGen` мы используем общий компонент загрузчика файлов `FileBasedExampleGen`, который позволяет переопределять `executor_class`.

```

from tfx.components import FileBasedExampleGen ❶
from tfx.components.example_gen.custom_executors import parquet_executor ❷
from tfx.utils.dsl_utils import external_input

examples = external_input(parquet_dir_path) example_gen = FileBasedExampleGen(
    input=examples,
    executor_class=parquet_executor.Executor) ❸

```

- ❶ Импорт компонента универсального загрузчика файлов
- ❷ Импорт executor, специфичного для Parquet
- ❸ Переопределение executor

Преобразование сериализованных данных Avro в tf.Example

Концепция переопределения `executor_class` может быть расширена почти до любого другого типа файла. TFX предоставляет дополнительные классы, как показано в следующем примере для загрузки сериализованных данных Avro.

```

from tfx.components import FileBasedExampleGen ❶
from tfx.components.example_gen.custom_executors import avro_executor ❷
from tfx.utils.dsl_utils import external_input

examples = external_input(avro_dir_path) example_gen = FileBasedExampleGen(
    input=examples,
    executor_class=avro_executor.Executor) ❸

```

- ❶ Импорт компонента универсального загрузчика файлов
- ❷ Импорт управляющего модуля, специфичного для Avro
- ❸ Переопределение executor

В случае если мы хотим загрузить другой тип файла, мы можем написать наш собственный executor, специфичный для нашего типа файла, и применить те же концепции перезаписи executor, приведенные ранее. В главе 10 мы расскажем вам о двух примерах написания собственных компонентов и executor для загрузки данных.

Преобразование пользовательских данных в TFRecords

Иногда проще преобразовать наши существующие наборы данных в TFRecords и затем загрузить их с помощью компонента `ImportExampleGen`, как было описано в разделе «Импорт существующих файлов TRecord». Этот подход полезен, если к нашим данным невозможно получить доступ через платформу данных, обеспечивающую эффективную передачу потоковых данных. Например, если мы обучаем модель компьютерного зрения и загружаем большое количество изображений в наш конвейер, мы должны в первую очередь преобразовать изображения в TFRecords (подробнее об этом будет рассказываться далее, в разделе «Графические данные для задач компьютерного зрения»).

В следующем примере мы преобразовываем структурированные данные в TFRecords. Представьте, что наши данные доступны не в формате CSV, а только в формате JSON или XML. Следующий пример можно использовать (с небольшими изменениями) для преобразования этих форматов данных с помощью компонента `ImportExampleGen` перед их использованием в конвейере.

Чтобы преобразовать данные в TFRecords, необходимо создать структуру `tf.Example` для каждой записи данных в вашем наборе данных. `tf.Example` – это простая, но очень гибкая структура данных, которая представляет собой пару ключ-значение.

```
{"string": value}
```

В случае TFRecords `tf.Example` ожидает объект `tf.Features`, который принимает словарь признаков, содержащий пары ключ-значение. Ключ всегда является строковым идентификатором, представляющим столбец признака, а значение является объектом `tf.train.Feature`.

Пример 3.1. Структура данных TFRecord

Record 1:

```
tf.Example
  tf.Features
    'column A': tf.train.Feature
    'column B': tf.train.Feature
    'column C': tf.train.Feature
```

`tf.train.Feature` допускает три типа данных:

- `tf.train.BytesList`;
- `tf.train.FlatList`;
- `tf.train.Int64List`.

Чтобы уменьшить избыточность кода, мы определим вспомогательные функции, которые помогут нам преобразовать записи данных в правильную структуру данных, используемую в `tf.Example`.

```
import tensorflow as tf
```

```
def _bytes_feature(value):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))
```

```
def _float_feature(value):
    return tf.train.Feature(float_list=tf.train.FloatList(value=[value]))
```

```
def _int64_feature(value):
    return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))
```

Имея вспомогательные функции, давайте посмотрим, как мы можем преобразовать наш демонстрационный набор данных в TFRecords. Сначала нам нужно прочитать наш исходный файл данных и преобразовать каждую запись данных в структуру данных `tf.Example`, а затем сохранить все записи в файле TFRecord. В следующем примере кода приведена сокращенная версия. Полный пример можно найти в репозитории GitHub этой книги (размещенном по ссылке <https://github.com/Building-ML-Pipelines/building-machine-learning-pipelines>), в каталоге *chapters/data_ingestion*.

```

import csv
import tensorflow as tf

original_data_file = os.path.join(
    os.pardir, os.pardir, "data",
    "consumer-complaints.csv")
tfrecord_filename = "consumer-complaints.tfrecord"
tf_record_writer = tf.io.TFRecordWriter(tfrecord_filename) ❶

with open(original_data_file) as csv_file:
    reader = csv.DictReader(csv_file, delimiter=",", quotechar='"')
    for row in reader:
        example = tf.train.Example(features=tf.train.Features(feature={ ❷
            "product": _bytes_feature(row["product"]),
            "sub_product": _bytes_feature(row["sub_product"]),
            "issue": _bytes_feature(row["issue"]),
            "sub_issue": _bytes_feature(row["sub_issue"]),
            "state": _bytes_feature(row["state"]),
            "zip_code": _int64_feature(int(float(row["zip_code"]))),
            "company": _bytes_feature(row["company"]),
            "company_response": _bytes_feature(row["company_response"]),
            "consumer_complaint_narrative": \
                _bytes_feature(row["consumer_complaint_narrative"]),
            "timely_response": _bytes_feature(row["timely_response"]),
            "consumer_disputed": _bytes_feature(row["consumer_disputed"]),
        }))
        tf_record_writer.write(example.SerializeToString()) ❸
tf_record_writer.close()

```

- ❶ Создание объекта `TFRecordWriter`, который сохраняет данные в каталоге, путь к которому указан в `tfrecords_filename`
- ❷ `tf.train.Example` для каждой записи данных
- ❸ Сериализация структуры данных

Сгенерированный файл `TFRecords 26k-consumer-complaints-modified.tfrecords` теперь можно импортировать с помощью компонента `ImportExampleGen`.

Загрузка удаленных файлов данных

Компонент `ExampleGen` может читать файлы из удаленных сегментов `Google Cloud Storage` или `AWS Simple Storage Service (S3)`¹. Пользователи `TFX` могут указать путь к удаленному сегменту функции `external_input`, как показано в следующем примере.

```

examples = external_input("gs://example_compliance_data/")
example_gen = CsvExampleGen(input=examples)

```

Для доступа к сегментам хранилища частного облака требуется настройка учетных данных поставщика облачных услуг. Настройка зависит от поставщика. `AWS` аутентифицирует пользователей с помощью пользовательского ключа

¹ Для чтения файлов из `AWS S3` требуется `Apache Beam 2.19` или выше, который поддерживается начиная с версии `TFX 0.22`.

ча доступа и секрета доступа. Чтобы получить доступ к частным сегментам AWS S3, вам потребуются пользовательский ключ доступа и секрет доступа¹. Google Cloud Platform (GCP), напротив, аутентифицирует пользователей через учетные записи служб. Чтобы получить доступ к частным сегментам хранилища GCP, вам необходимо создать файл учетной записи службы с разрешением на доступ к сегменту хранилища².

Загрузка данных напрямую из баз данных

TFX предоставляет два компонента для загрузки наборов данных непосредственно из баз данных. В следующих разделах мы представляем компонент `BigQueryExampleGen` для запроса данных из таблиц BigQuery и компонент `PrestoExampleGen` для запроса данных из баз данных Presto.

Google Cloud BigQuery

TFX предоставляет компонент для загрузки данных из таблиц BigQuery. Это очень эффективный способ использования структурированных данных, если мы запускаем наши конвейеры машинного обучения в экосистеме облачной платформы Google.

Учетные данные Google Cloud

Для запуска компонента `BigQueryExampleGen` необходимо, чтобы мы определили необходимые учетные данные Google Cloud в нашей локальной среде. Нам нужно создать учетную запись службы с нужными нам ролями (как минимум *BigQuery Data Viewer* и *BigQuery Job User*). Если вы запускаете исполнение компонента в интерактивном контексте с помощью Apache Beam или Apache Airflow, вам нужно указать путь к файлу учетных данных учетной записи службы через переменную среды `GOOGLE_APPLICATION_CREDENTIALS`, как показано в следующем фрагменте кода. Если вы запускаете исполнение компонента через Kubeflow Pipelines, вы можете предоставить информацию об учетной записи службы с помощью функций OrFunc, о которых рассказывается в разделе «Функции OrFunc» главы 12. Вы можете сделать это в Python следующим образом:

```
import os
os.environ[«GOOGLE_APPLICATION_CREDENTIALS»] =
    «/path/to/credential_file.json»
```

Дополнительные сведения см. в документации Google Cloud, размещенной по ссылке <https://cloud.google.com/docs/authentication/getting-started>.

¹ Для получения дополнительной информации об управлении ключами доступа AWS см. документацию, доступную по ссылке https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html#Using_CreateAccessKey.

² Для получения дополнительной информации о том, как создавать учетные записи служб и управлять ими, см. документацию, размещенную по ссылке <https://cloud.google.com/iam/docs/creating-managing-service-accounts>.

В следующем примере показан самый простой способ запроса наших таблиц BigQuery.

```
from tfx.components import BigQueryExampleGen

query = """
    SELECT * FROM `<project_id>.<database>.<table_name>`
    """
example_gen = BigQueryExampleGen(query=query)
```

Конечно, мы можем создавать более сложные запросы для выбора ваших данных, например запросы, выполняющие соединения нескольких таблиц.



Изменения компонента BigQueryExampleGen

В версиях TFX выше 0.22.0 компонент BigQueryExampleGen необходимо импортировать из `tfx.extensions.google_cloud_big_query`:

```
from tfx.extensions.google_cloud_big_query.example_gen \
    import component as big_query_example_gen_component
big_query_example_gen_component.BigQueryExampleGen(query=query)
```

Базы данных Presto

Если мы хотим получить данные из базы данных Presto, то можем использовать PrestoExampleGen. Его использование аналогично BigQueryExampleGen, где мы определили запрос к базе данных, а затем выполнили его. Для компонента PrestoExampleGen потребуется дополнительная настройка, чтобы указать параметры подключения для базы данных.

```
from proto import presto_config_pb2
from presto_component.component import PrestoExampleGen
query = """
    SELECT * FROM `<project_id>.<database>.<table_name>`
    """
presto_config = presto_config_pb2.PrestoConnConfig(
    host='localhost',
    port=8080)
example_gen = PrestoExampleGen(presto_config, query=query)
```



PrestoExampleGen требует отдельной установки

Начиная с версии TFX 0.22 PrestoExampleGen требует отдельного процесса установки. После установки компилятора protoc¹ вы можете установить компонент из исходного кода, выполнив следующие действия:

```
$ git clone git@github.com:tensorflow/tfx.git && cd tfx/
$ git checkout v0.22.0
$ cd tfx/examples/custom_components/presto_example_gen
$ pip install -e .
```

¹ Чтобы подробно узнать об установке protoc, обратитесь к документации API proto-lens в GitHub, перейдя по ссылке <http://google.github.io/proto-lens/installing-protoc.html>.

После установки вы сможете импортировать компонент `PrestoExampleGen` и его определения буферов протокола.

Подготовка данных

Каждый из представленных компонентов `ExampleGen` позволяет нам конфигурировать входные (`input_config`) и выходные (`output_config`) параметры для набора данных. Если мы хотим получать наборы данных инкрементально, то можем определить совокупность в качестве параметра конфигурации. В то же время мы можем настроить способ разделения данных. Часто нам бывает нужно создать обучающий набор вместе с оценочным и тестовым наборами. Мы можем указать подробности в конфигурации выхода.

Разбиение наборов данных

Далее в нашем конвейере нам нужно оценивать нашу модель машинного обучения во время обучающих прогонов и тестировать ее на этапе анализа модели. Поэтому полезно разбить набор данных на соответствующие подмножества.

Разбиение набора данных на несколько подмножеств

В следующем примере кода показано, как мы можем расширить наш процесс получения, внесения и обработки данных, добавив разбиение на три составляющие: обучающий, оценочный и тестовый наборы данных в соотношении 6:2:2. Настройки соотношения определяются через `hash_buckets`.

```
from tfx.components import CsvExampleGen
from tfx.proto import example_gen_pb2
from tfx.utils.dsl_utils import external_input

base_dir = os.getcwd()
data_dir = os.path.join(os.pardir, "data")
output = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=6),
        example_gen_pb2.SplitConfig.Split(name='eval', hash_buckets=2),
        example_gen_pb2.SplitConfig.Split(name='test', hash_buckets=2)
    ])
)

examples = external_input(os.path.join(base_dir, data_dir))
example_gen = CsvExampleGen(input=examples, output_config=output)

context.run(example_gen)
```

- ❶ Определите нужные разбиения
- ❷ Задайте коэффициенты соотношения
- ❸ Добавьте аргумент `output_config`

После запуска объекта `example_gen` мы можем проверить сгенерированные артефакты, выведя на печать список артефактов.

```
for artifact in example_gen.outputs['examples'].get():
    print(artifact)

Artifact(type_name: ExamplesPath,
         uri: /path/to/CsvExampleGen/examples/1/train/, split: train, id: 2)
Artifact(type_name: ExamplesPath,
         uri: /path/to/CsvExampleGen/examples/1/eval/, split: eval, id: 3)
Artifact(type_name: ExamplesPath,
         uri: /path/to/CsvExampleGen/examples/1/test/, split: test, id: 4)
```

В следующей главе мы обсудим, как выполняется исследование наборов данных, созданных для конвейера данных.



Разбиение по умолчанию

Если мы не задаем какую-либо конфигурацию выхода, компонент `ExampleGen` разбивает набор данных на два подмножества – обучающий и оценочный наборы данных с соотношением 2:1 по умолчанию.

Сохранение существующих разбиений

В некоторых случаях у нас уже имеются подмножества наборов данных, сгенерированные вне нашего рабочего процесса, и мы хотели бы сохранить эти разбиения, когда получаем эти наборы данных. Мы можем достичь этого, определив конфигурацию входа.

Для следующей конфигурации давайте предположим, что наш набор данных был разделен вне нашего рабочего процесса и сохранен в следующих подкаталогах:

```
└─ data
   ├── train
   │   └─ 20k-consumer-complaints-training.csv
   ├── eval
   │   └─ 4k-consumer-complaints-eval.csv
   └── test
       └─ 2k-consumer-complaints-test.csv
```

Мы можем сохранить существующее разбиение входных данных, определив следующую конфигурацию входа:

```
import os

from tfx.components import CsvExampleGen
from tfx.proto import example_gen_pb2
from tfx.utils.dsl_utils import external_input

base_dir = os.getcwd()
data_dir = os.path.join(os.pardir, "data")

input = example_gen_pb2.Input(splits=[
    example_gen_pb2.Input.Split(name='train', pattern='train/*'), ❶
    example_gen_pb2.Input.Split(name='eval', pattern='eval/*'),
    example_gen_pb2.Input.Split(name='test', pattern='test/*')
])
```

```
examples = external_input(os.path.join(base_dir, data_dir))
example_gen = CsvExampleGen(input=examples, input_config=input) ❷
```

- ❶ Указание существующих подкаталогов
- ❷ Добавление аргумента `input_config`

После определения конфигурации входа мы можем передать настройки компоненту `ExampleGen`, определив аргумент `input_config`.

Связующие наборы данных

Одним из важных вариантов использования конвейеров машинного обучения является то, что мы можем обновлять наши модели машинного обучения, когда становятся доступны новые данные. Для этого сценария компонент `ExampleGen` позволяет нам использовать *совокупности* (spans). Совокупность можно рассматривать как снимок данных. Например, каждый час, день или неделю процесс пакетного переноса данных из одного источника в другой (Extract, transform, load, ETL) может создавать такой моментальный снимок данных и тем самым создавать новую совокупность.

Совокупность может реплицировать существующие записи данных. Как показано ниже, *export-1* содержит данные из предыдущей совокупности *export-0*, а также вновь созданные записи, которые были добавлены после *export-0*:

```
├─ data
│ └─ export-0
│   └─ 20k-consumer-complaints.csv
└─ export-1
  └─ 24k-consumer-complaints.csv
└─ export-2
  └─ 26k-consumer-complaints.csv
```

Теперь мы можем указать шаблоны совокупностей. Конфигурация входа принимает поле для подстановки `{SPAN}`, которое представляет собой число (0, 1, 2,...), показанное в нашей структуре каталогов. При заданной входной конфигурации компонент `ExampleGen` теперь выбирает «последнюю» совокупность. В нашем примере это будут данные, доступные в каталоге *export-2*.

```
from tfx.components import CsvExampleGen
from tfx.proto import example_gen_pb2
from tfx.utils.dsl_utils import external_input

base_dir = os.getcwd()
data_dir = os.path.join(os.pardir, "data")

input = example_gen_pb2.Input(splits=[
    example_gen_pb2.Input.Split(pattern='export-{SPAN}/*')
])

examples = external_input(os.path.join(base_dir, data_dir))
example_gen = CsvExampleGen(input=examples, input_config=input)
context.run(example_gen)
```

Разумеется, в определениях входа также могут содержаться определения подкаталогов, если данные уже разделены.

```
input = example_gen_pb2.Input(splits=[
    example_gen_pb2.Input.Split(name='train',
                                pattern='export-{SPAN}/train/*'),
    example_gen_pb2.Input.Split(name='eval',
                                pattern='export-{SPAN}/eval/*')
])
```

Управление версиями наборов данных

В конвейерах машинного обучения необходимо отслеживать версии созданных моделей совместно с обучающими наборами данных, которые использовались для обучения этих моделей. Для этого пригодится механизм управления версиями наших наборов данных.

Управление версиями данных позволяет нам отслеживать полученные данные с более высокой степенью детализации. Это означает, что мы не просто сохраняем имя файла и путь к загруженным данным в хранилище метаданных *MetadataStore* (потому что в настоящее время он поддерживается компонентами TFX), но также отслеживаем больше метаданных о необработанном наборе данных, например хеш полученных данных. Такой способ реализации отслеживания версий позволит нам убедиться, что набор данных, используемый во время обучения, по-прежнему будет набором данных, используемым в более поздний момент времени. Подобная возможность имеет решающее значение для воспроизводимости непрерывного машинного обучения.

Однако эта функция в настоящее время не поддерживается компонентом TFX *ExampleGen*. Если вы хотите изменить версии своих наборов данных, то можете использовать сторонние инструменты управления версиями данных и определять версию данных до того, как наборы данных будут загружены в конвейер.

Если вы хотите изменить версии своих наборов данных, можете использовать один из следующих инструментов.

Data Version Control (DVC)

DVC – это система управления версиями с открытым исходным кодом для проектов машинного обучения. Он позволяет хранить версии хешей ваших наборов данных, а не размещать весь набор данных в качестве очередной версии. Таким образом, состояние набора данных отслеживается (например, через *git*), но целиком весь набор данных не хранится в репозитории.

Pachyderm

Pachyderm – это платформа машинного обучения с открытым исходным кодом, работающая поверх Kubernetes. Она развивалась на основе концепции управления версиями для данных («Git для данных»), но теперь расширилась до всей платформы данных, включая оркестрацию конвейера на основе версий данных.

СТРАТЕГИИ ЗАГРУЗКИ ДАННЫХ

До сих пор мы обсуждали различные способы ввода данных в наши конвейеры машинного обучения. Если вы начинаете совершенно новый проект, выбор правильной стратегии загрузки данных может дать поистине потрясающий эффект. В следующих разделах мы рассмотрим несколько стратегий для трех типов данных: структурированных, текстовых и графических.

Структурированные данные

Структурированные данные часто хранятся в базе данных или на диске в виде файлов, имеющих формат, совместимый с табличными данными. Если данные находятся в базе данных, мы можем либо экспортировать их в файлы в формате CSV, либо использовать напрямую с помощью компонента `PrestoExampleGen` или `BigQueryExampleGen` (если соответствующие службы доступны).

Данные, хранящиеся на диске в файлах, форматы которых поддерживают табличные данные, должны быть преобразованы в файлы в формате CSV, а затем переданы в конвейер с помощью компонента `CsvExampleGen`.

Если объем данных превышает несколько сотен мегабайт, мы должны позаботиться о преобразовании данных в `TFRecords` или сохранении их с помощью `Apache Parquet`.

Текстовые данные для задач обработки естественного языка

Массивы текста могут увеличиваться в объеме, достигая значительных размеров. Для эффективной загрузки таких наборов данных мы рекомендуем преобразовать наборы данных в `TFRecords` или `Apache Parquet`. Использование постоянных типов файлов данных обеспечивает эффективную и последовательную загрузку используемых документов. Вы также можете загружать текстовые массивы из базы данных; однако мы рекомендуем учитывать стоимость сетевого трафика и другие возможные ограничения.

Графические данные для задач компьютерного зрения

Мы рекомендуем преобразовывать наборы графических данных из файлов графических форматов в `TFRecords`, однако не декодировать изображения. Любое декодирование сильно сжатых изображений только увеличивает объем памяти, необходимый для хранения промежуточных записей `tf.Example`. Сжатые изображения могут быть сохранены в виде байтовых строк в записях `tf.Example`:

```
import tensorflow as tf

base_path = "/path/to/images" filenames = os.listdir(base_path)

def generate_label_from_path(image_path):
    ...
    return label
```

```
def _bytes_feature(value):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))

def _int64_feature(value):
    return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))
tfrecord_filename = 'data/image_dataset.tfrecord'

with tf.io.TFRecordWriter(tfrecord_filename) as writer:
    for img_path in filenames:
        image_path = os.path.join(base_path, img_path)
        try:
            raw_file = tf.io.read_file(image_path)
        except FileNotFoundError:
            print("File {} could not be found".format(image_path))
            continue
        example = tf.train.Example(features=tf.train.Features(feature={
            'image_raw': _bytes_feature(raw_file.numpy()),
            'label': _int64_feature(generate_label_from_path(image_path))
        }))
        writer.write(example.SerializeToString())
```

Приведенный выше пример кода считывает изображения из каталога, размещенного в указанном каталоге, путь к которому задается как `/path/to/images`, и преобразует их в байтовую строку `tf.Example`. В данный момент, на текущем этапе работы конвейера, мы не обрабатываем наши изображения. Несмотря на то что мы могли бы сэкономить значительный объем памяти, нам придется выполнить эти задачи на более поздних этапах работы конвейера. Отсутствие предварительной обработки на этом этапе помогает нам избежать ошибок и возможного отклонения обучения/обслуживания в дальнейшем.

Мы храним необработанное изображение вместе с метками в файле `tf.Examples`. В нашем примере мы получаем метку для каждого изображения из имени файла с помощью функции `generate_label_from_path`. Генерация меток зависит от набора данных; поэтому мы не включали ее в этот пример.

После преобразования изображений в файлы `TFRecord` мы можем эффективно использовать наборы данных при помощи компонента `ImportExampleGen` и применять те же стратегии, которые мы обсуждали в разделе «Импорт существующих файлов `TFRecord`» в этой главе.

РЕЗЮМЕ

В этой главе мы обсудили различные способы ввода данных в наш конвейер машинного обучения. Мы уделили особое внимание случаям использования наборов данных, хранящихся на диске, а также в базах данных. В процессе мы также обсудили, что принятые записи данных конвертируются в `tf.Example` (хранятся в файлах `TFRecord`) для использования нижестоящими компонентами.

В следующей главе мы рассмотрим, как мы можем использовать сгенерированные записи `tf.Example` на этапе проверки данных конвейера.

Глава 4

Проверка данных

В главе 3 мы обсудили, как можем загружать данные из различных источников в наш конвейер. В этой главе мы переходим к проверке данных перед их использованием, как показано на рис. 4.1.

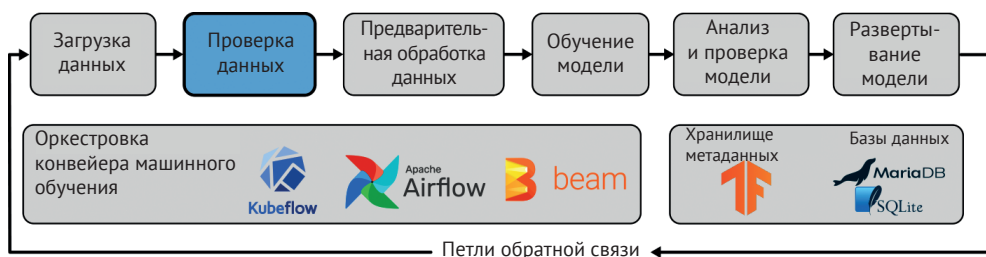


Рис. 4.1. Проверка данных как часть конвейера машинного обучения

Данные являются основой каждой модели машинного обучения, а полезность и качество модели зависят от данных, используемых для обучения, проверки и анализа модели. Как вы понимаете, без надежных данных мы не сможем построить надежные модели. В разговорной речи вы, возможно, слышали фразу «мусор на входе, мусор на выходе», означающий, что наши модели не будут работать, если базовые данные не проверены. Это точная цель нашего первого шага рабочего процесса в конвейере машинного обучения: проверка данных.

В этой главе мы сначала объясним идею проверки данных, а затем познакомим вас с пакетом Python из экосистемы TensorFlow Extended, который называется Tensor Flow Data Validation (TFDV). Мы покажем, как вы можете настроить пакет в проектах по науке о данных, проведем вас через распространенные варианты использования и выделим некоторые весьма полезные рабочие процессы.

На этапе проверки данных проверяется, соответствуют ли данные в конвейерах ожиданиям, появляющимся на этапе разработки функций. На этом этапе также выполняется сравнение нескольких наборов данных и определяется, изменяются ли ваши данные с течением времени, например ваши обучающие данные могут значительно отличаться от новых данных, используемых в вашей модели для формирования выводов и заключений.

В конце главы мы интегрируем наш первый шаг рабочего процесса в конвейер TFX.

Для чего нужна проверка данных?

В машинном обучении мы строим обучение на основе определенных закономерностей в наборах данных и обобщаем эти знания. Это означает, что наборы данных играют важнейшую роль в рабочих процессах машинного обучения, и качество данных становится основополагающим фактором успеха проектов машинного обучения.

Каждый шаг рабочего процесса в жизненном цикле машинного обучения определяет, может ли рабочий процесс перейти к следующему шагу или нужно прервать и перезапустить весь рабочий процесс, например предоставив дополнительные обучающие данные. Проверка данных – очень важный этап, поскольку в процессе проверки отслеживаются изменения в данных, загружаемых в конвейер машинного обучения, прежде чем процесс перейдет к трудоемким этапам предварительной обработки и обучения.

Если нашей целью является автоматизация обновлений модели машинного обучения, проверка данных имеет первостепенное значение. В частности, когда мы говорим о проверке, то имеем в виду три различных шага проверки данных:

- проверьте данные на присутствие аномалий данных;
- убедитесь, что схема данных не изменилась;
- убедитесь, что параметры статистики для новых наборов данных по-прежнему совпадают с соответствующими параметрами статистики для наших предыдущих обучающих наборов данных.

В процессе проверки данных выполняются все эти типы проверок и выявляются любые несоответствия или ошибки. Если обнаружено несоответствие либо ошибка, мы можем остановить рабочий процесс и скорректировать данные вручную, например обработав новый набор данных.

Также бывает полезно вернуться к этапу проверки данных из этапа обработки данных, следующего этапа в конвейере машинного обучения. В процессе проверки генерируются данные статистики на основе признаков данных и выделяются ситуации, когда какой-либо признак содержит большой процент пропущенных значений или когда признаки сильно коррелированы. Эта информация потребуется вам на этапе принятия решения о том, какие признаки нужно выбирать на этапе предварительной обработки и какой должна быть предварительная обработка данных.

Проверка данных позволяет сравнивать статистические показатели для разных наборов данных. Этот простой шаг может помочь вам в отладке и устранении проблем в модели. Например, проверка данных может сравнивать статистические показатели ваших обучающих и контрольных данных. Добавив всего лишь несколько строк кода, вы заметите огромную разницу. Вы можете обучить модель бинарной классификации с идеальным распределением меток 50/50 (50 % положительных и 50 % отрицательных меток), но в вашем контрольном наборе распределение будет отличаться от 50/50. Это различие в конечном итоге искажает ваши контрольные показатели.

В мире, где наборы данных постоянно растут, проверка данных имеет решающее значение, позволяя убедиться, что наши модели машинного обучения все еще соответствуют поставленной задаче. Поскольку мы можем сравнивать схемы, мы можем быстро определить, изменилась ли структура данных во вновь полученных наборах данных, например когда определенный признак устарел. Проверка данных также может обнаружить ситуацию, когда ваши данные начинают *дрейфовать*. Это означает, что вновь собранные данные имеют другую базовую статистику, нежели исходный набор данных, используемый для обучения вашей модели. Этот дрейф может означать, что вам нужно выбрать новые признаки или что необходимо обновить этапы предварительной обработки данных (например, если минимальное или максимальное значение столбца, содержащего численные значения, изменяется). Дрейф может происходить по ряду причин: тренд, лежащий в основе данных, сезонные изменения данных или наличие петли обратной связи. Эти вопросы будут обсуждаться в главе 13.

В следующих разделах мы рассмотрим различные варианты использования, о которых мы только что рассказали. Однако перед этим давайте рассмотрим предварительные шаги установки, необходимые, чтобы запустить TFDV.

TFDV

В состав экосистемы TensorFlow входит инструмент, который помогает в проверке данных; это TFDV. TFDV – часть проекта TFX. TFDV позволяет выполнять анализ, о котором мы уже говорили ранее (например, создание схем и проверка новых данных на соответствие существующей схеме). Он также предлагает инструменты визуализации, входящие в состав проекта Facets в Google PAIR. На рис. 4.2 показано, как выглядят эти инструменты.

TFDV принимает два входных формата для запуска проверки данных: TFRecord (внутренний формат TensorFlow) и файлы CSV. Как и другие компоненты TFX, он управляет задачами анализа с помощью Apache Beam.

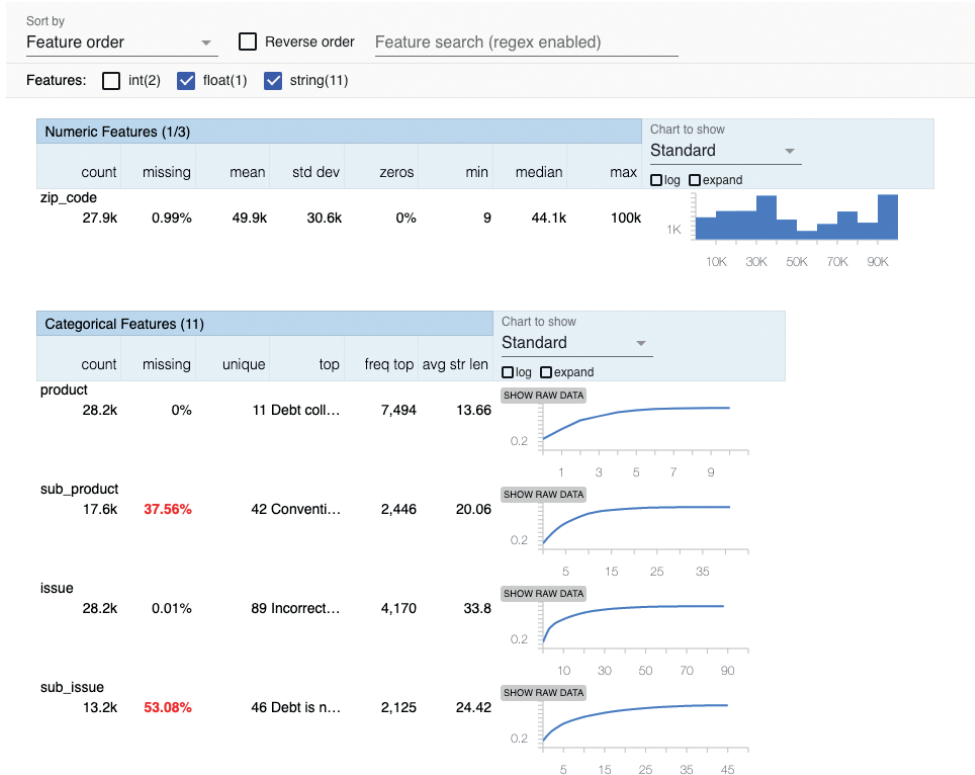


Рис. 4.2. Экранный снимок средства визуализации TFDV

Установка

Если мы установили пакет `tfx` так, как это было описано в главе 2, то пакет проверки данных TFDV уже был установлен как зависимый пакет. В случае если мы хотим использовать TFDV как отдельный пакет, мы можем установить его, выполнив следующую команду:

```
$ pip install tensorflow-data-validation
```

После установки `tfx` или `tensorflow-data-validation` мы можем интегрировать нашу проверку данных в рабочие процессы машинного обучения или визуально анализировать наши данные в Jupyter Notebook. В следующих разделах мы рассмотрим несколько примеров.

Генерация статистических показателей для набора данных

В качестве первого шага в нашем процессе проверки данных мы получим некоторую сводную статистику для наших данных. Например, мы можем загрузить данные CSV, содержащие жалобы потребителей, напрямую в TFDV и получить статистические показатели для каждого признака:

```
import tensorflow_data_validation as tfdv
stats = tfdv.generate_statistics_from_csv(

    data_location='/data/consumer_complaints.csv',
    delimiter=',')
```

Аналогичным образом мы можем получить статистические показатели для признаков из TFRecords:

```
stats = tfdv.generate_statistics_from_tfrecord(
    data_location='/data/consumer_complaints.tfrecord')
```

О том, как создавать файлы TFRecord, рассказывалось в главе 3.

Оба метода TFDV генерируют структуру данных, в которой хранятся такие метрики, как минимальные, максимальные и средние значения.

Структура данных выглядит следующим образом:

```
datasets {
  num_examples: 66799
  features {
    type: STRING
    string_stats {
      common_stats {
        num_non_missing: 66799
        min_num_values: 1
        max_num_values: 1
        avg_num_values: 1.0
        num_values_histogram {
          buckets { low_value: 1.0
                    high_value: 1.0
                    sample_count: 6679.9
          }
        }
      }
    }
  }
}
```

Для числовых значений TFDV вычисляет следующие параметры:

- общее число записей данных;
- процент пропущенных записей данных;
- среднее значение и стандартное отклонение для признака;
- минимальное и максимальное значения для признака;
- процент нулевых значений для признака.

Кроме того, TensorFlow Data Validation строит гистограмму значений для каждого признака.

Для категориальных признаков TFDV вычисляет следующие показатели статистики:

- общее число записей данных;
- процент пропущенных записей данных;
- количество уникальных записей;
- среднюю длину строки всех записей признака;
- для каждой категории TFDV определяет число образцов для каждой метки и ее ранг.

Сейчас вы увидите, как мы можем превратить эти показатели в нечто полезное.

Генерация схемы на основе данных

После генерации сводной статистики для данных следующим шагом будет создание схемы нашего набора данных. Схема данных – это форма описания представления ваших наборов данных. Схема определяет, какие признаки ожидаются в вашем наборе данных и к какому типу относится каждый признак. Кроме того, ваша схема должна определять границы ваших данных (например, выделение минимумов, максимумов, пороговых значений допустимых пропущенных записей для объекта).

Затем определение схемы вашего набора данных можно использовать для проверки будущего набора данных, чтобы определить, соответствуют ли они вашим предыдущим обучающим наборам. Схемы, сгенерированные TFDV, также можно использовать на следующем шаге рабочего процесса, когда вы будете предварительно обрабатывать свои наборы данных, чтобы преобразовать их в данные, которые можно использовать для обучения моделей машинного обучения.

Как показано в примере ниже, вы можете сгенерировать описание схемы из полученных данных статистики с помощью одного вызова функции:

```
schema = tfdv.infer_schema(stats)
```

tfdv.infer_schema генерирует протокол схемы, определенный TensorFlow1

```
feature {
  name: "product"
  type: BYTES
  domain: "product"
  presence {
    min_fraction: 1.0
    min_count: 1
  }
  shape {
    dim {
      size: 1
    }
  }
}
```

Вы можете отобразить схему с помощью одного вызова функции в любом блокноте Jupyter Notebook.

```
tfdv.display_schema(schema)
```

¹ Определения буферов протокола для протокола схемы можно найти в репозитории TensorFlow: https://github.com/tensorflow/metadata/blob/master/tensorflow_metadata/proto/v0/schema.proto.

Результат выполнения этой команды показан на рис. 4.3.

Feature name	Type	Presence	Valency	Domain
'product'	STRING	required		'product'
'sub_product'	STRING	optional	single	'sub_product'
'issue'	STRING	required		'issue'
'sub_issue'	STRING	optional	single	'sub_issue'
'consumer_complaint_narrative'	BYTES	required		-
'company'	BYTES	required		-
'state'	STRING	optional	single	'state'
'zip_code'	BYTES	optional	single	-
'company_response'	STRING	required		'company_response'
'timely_response'	STRING	required		'timely_response'
'consumer_disputed'	INT	required		-

Рис. 4.3. Экранный снимок визуализации схемы

В этой таблице Presence означает, должен ли признак присутствовать в 100 % образцов данных (required) или нет (optional). Valency означает количество значений, необходимых для каждого обучающего примера. Для категориальных признаков single будет означать, что каждый обучающий пример должен иметь ровно одну категорию для данного признака.

Сгенерированная здесь схема может отличаться от реальной, потому что она предполагает, что текущий набор данных в точности такой же, как и все будущие данные. Если некоторый признак будет присутствовать во всех обучающих примерах в этом наборе данных, он будет отмечен как обязательный, но на самом деле он может быть необязательным. О том, как обновить схему в соответствии с вашими собственными знаниями о наборе данных, будет рассказываться в разделе «Обновление схемы».

Теперь, когда схема определена, мы можем пойти дальше и сравнить наши обучающие или оценочные наборы данных или проверить наши наборы данных на наличие ошибок, которые могут повлиять на нашу модель.

РАСПОЗНАВАНИЕ ОШИБОК В ДАННЫХ

В предыдущих разделах мы обсуждали, как генерировать статистические показатели и схему для наших данных. Они описывают наши данные, но не обнаруживают потенциальных проблем с ними. В следующих разделах мы посмотрим, как TFDV может помочь нам выявлять проблемы в наших данных.

Сравнение наборов данных

Предположим, у нас имеется два набора данных: обучающий и контрольный наборы. Прежде чем приступить к обучению нашей модели, мы хотели бы определить, насколько репрезентативен контрольный набор данных по отношению к обучающему набору. Соответствуют ли контрольные данные нашей схеме обучающих данных? Отсутствуют ли в наборе данных какие-либо столбцы признаков или значительное количество значений признаков? Используя TFDV, мы можем быстро получить ответ.

Как показано в следующем примере, мы можем загрузить оба набора данных и затем визуализировать их вместе. Если мы запустим приведенный ниже код в Jupyter Notebook, то сможем легко сравнить статистические показатели для двух наборов данных.

```
train_stats = tfdv.generate_statistics_from_tfrecord(
    data_location=train_tfrecord_filename)
val_stats = tfdv.generate_statistics_from_tfrecord(
    data_location=val_tfrecord_filename)

tfdv.visualize_statistics(lhs_statistics=val_stats, rhs_statistics=train_stats,
    lhs_name='VAL_DATASET', rhs_name='TRAIN_DATASET')
```

Рисунок 4.4 иллюстрирует разницу между двумя наборами данных. Например, контрольный набор данных (содержащий 4998 записей) имеет более низкую частоту пропущенных значений `sub_issue`. Это может означать отклонение данного признака от обучающих данных. Что еще более важно, визуализация подчеркнула, что более половины всех записей не содержат данных `sub_issue`. Если `sub_issue` является важным признаком для обучения модели, мы должны исправить наши методы сбора данных, чтобы собрать новые данные с учетом выявленной проблемы.

Схема обучающих данных, которую мы генерировали ранее, теперь принимает очень удобный вид. TFDV позволяет нам проверять любую статистику данных по схеме и сообщает о любых аномалиях.

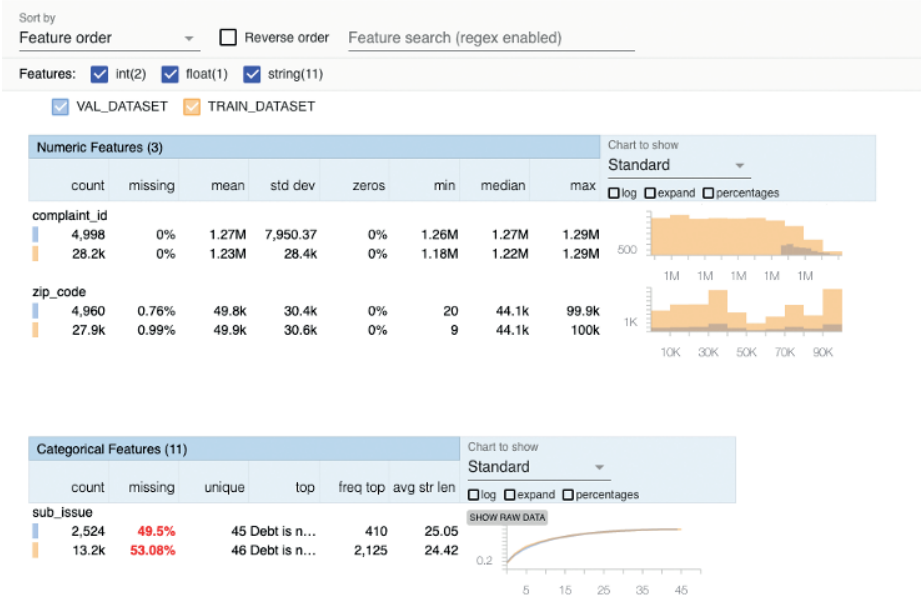


Рис. 4.4. Сравнение обучающего и контрольного наборов данных

Аномалии можно обнаружить с помощью следующего кода:

```
anomalies = tfdv.validate_statistics(statistics=val_stats, schema=schema)
```

Затем мы можем отобразить аномалии с помощью

```
tfdv.display_anomalies(anomalies)
```

В результате выполнения этой команды мы получим результат, показанный на рис. 4.5.

Feature name	Anomaly short description	Anomaly long description
"company"	Column dropped	The feature was present in fewer examples than expected.

Рис. 4.5. Визуализация аномалий в Jupyter Notebook

Следующий пример кода показывает основной протокол аномалий. Протокол содержит полезную информацию, которую мы можем использовать для автоматизации нашего процесса машинного обучения.

```
anomaly_info {
  key: "company"
  value {
    description: "The feature was present in fewer examples than expected."
    severity: ERROR
    short_description: "Column dropped"
    reason {
      type: FEATURE_TYPE_LOW_FRACTION_PRESENT
    }
  }
}
```

```

    short_description: "Column dropped"
    description: "The feature was present in fewer examples than expected."
  }
  path {
    step: "company"
  }
}
}
}

```

Обновление схемы

Протокол аномалий, приведенный выше, показывает нам, как обнаружить отклонения от схемы, автоматически генерируемые из нашего набора данных. Но другой вариант использования TFDV – это ручная настройка схемы в соответствии с нашими знаниями в предметной области, к которой относятся данные. Взяв признак `sub_issue`, как описано выше, мы принимаем решение: если мы решили, что нам нужно, чтобы этот признак присутствовал в > 90 % наших обучающих выборок, то можем обновить схему, чтобы выполнить это требование.

Во-первых, нам нужно загрузить схему из ее местоположения:

```
schema = tfdv.load_schema_text(schema_location)
```

Затем мы обновляем этот конкретный признак, требуя, чтобы он был обязательным в 90 % случаев:

```
sub_issue_feature = tfdv.get_feature(schema, 'sub_issue')
sub_issue_feature.presence.min_fraction = 0.9
```

Мы также можем обновить список штатов, чтобы удалить штат Аляска:

```
state_domain = tfdv.get_domain(schema, 'state')
state_domain.value.remove('AK')
```

Теперь, когда мы привели схему в порядок, мы записываем файл схемы в его сериализованное местоположение:

```
tfdv.write_schema_text(schema, schema_location)
```

Затем нам необходимо повторно проверить статистические показатели, чтобы просмотреть обновленные данные об аномалиях:

```
updated_anomalies = tfdv.validate_statistics(eval_stats, schema)
tfdv.display_anomalies(updated_anomalies)
```

Таким образом, мы можем корректировать аномалии до тех пор, пока не получим значения, подходящие для нашего набора данных¹.

¹ Вы можете настроить схему таким образом, чтобы в среде обучения и промышленной среде использовались различные признаки. Для получения более подробной информации обратитесь к документации, размещенной по ссылке https://www.tensorflow.org/tfx/data_validation/get_started#schema_environments.

Отклонения и дрейф данных

TFDV предоставляет встроенный «компаратор отклонений», который обнаруживает большие различия между статистическими показателями двух наборов данных. Это не статистическое определение расфазировки данных (набор данных, который асимметрично распределен вокруг своего среднего значения). В TFDV отклонение определяется как L-норма по бесконечности между `serve_statistics` двух наборов данных. Если разница между двумя наборами данных превышает ваш порог L1-нормы по бесконечности для данной функции, TensorFlow Data Validation выделяет ее как аномалию, используя способ обнаружения аномалии, описанный ранее в этой главе.



L-норма по бесконечности

L-норма по бесконечности – это выражение для определения разницы между двумя векторами (в нашем случае это действующая статистика). L-норма по бесконечности определяется как максимальное абсолютное значение записей вектора.

Например, L-норма по бесконечности вектора $[3, -10, -5]$ равна 10. Нормы часто используются для сравнения векторов. Если мы хотим сравнить векторы $[2, 4, -1]$ и $[9, 1, 8]$, то сначала вычисляем их разность, которая составляет $[-7, 3, -9]$, а затем вычисляем L-норму по бесконечности этого вектора, которая равна 9.

В случае TFDV два вектора представляют собой сводную статистику двух наборов данных. Возвращенная норма – это самая большая разница между этими двумя наборами статистических данных.

Приведенный ниже фрагмент кода показывает, как можно сравнить расхождение между наборами данных:

```
tfdv.get_feature(schema,
                  'company').skew_comparator.infinity_norm.threshold = 0.01
skew_anomalies = tfdv.validate_statistics(statistics=train_stats,
                                          schema=schema,
                                          serving_statistics=serving_stats)
```

На рис. 4.6 показан результат выполнения приведенного выше кода.

Feature name	Anomaly short description	Anomaly long description
"company"	High L-infinity distance between current and previous	The L-infinity distance between current and previous is 0.0170752 (up to six significant digits), above the threshold 0.01. The feature value with maximum difference is: Experian

Рис. 4.6. Визуализация расхождения данных между обучающим и основным наборами данных

TFDV также предоставляет компонент `drift_comparator` для сравнения статистики двух наборов данных одного типа, например двух обучающих наборов, собранных в два разных дня. Если обнаружен «дрейф», аналитик данных должен проверить либо архитектуру модели, либо определить, нужно ли снова выполнять конструирование признаков.

Как и в приведенном выше примере с отклонением данных, вы определяете свой `drift_comparator` для признаков, которые вы хотели бы просматривать и сравнивать. Затем вы можете вызвать `validate_statistics`, используя два варианта данных статистики для набора данных в качестве аргументов: один – для ваших основных данных (например, вчерашнего набора данных) и другой – вариант данных статистики для набора данных, с которым вы хотите выполнить сравнение (например, сегодняшний набор данных).

```
tfdv.get_feature(schema,
                  'company').drift_comparator.infinity_norm.threshold = 0.01
drift_anomalies = tfdv.validate_statistics(statistics=train_stats_today,
                                          schema=schema,
                                          previous_statistics=\
                                          train_stats_yesterday)
```

Результаты выполнения этого фрагмента кода приведены на рис. 4.7.

Feature name	Anomaly short description	Anomaly long description
"company"	High L-infinity distance between current and previous	The L-infinity distance between current and previous is 0.0170752 (up to six significant digits), above the threshold 0.01. The feature value with maximum difference is: Experian

Рис. 4.7. Визуализация дрейфа данных между двумя обучающими наборами

L-норма по бесконечности в `skew_comparator` и `drift_comparator` удобна для демонстрации больших различий между нашими наборами данных, особенно тех, которые могут показать нам, что что-то не так с нашим процессом сбора данных. Поскольку L-норма по бесконечности возвращает только одно число, схема может оказаться более полезна при обнаружении различий между наборами данных.

Наборы данных с систематической ошибкой выборки

Другая потенциальная проблема с нашим входным набором данных – это наличие систематической ошибки выборки, или асимметричность («предвзятость»). Мы определяем систематическую ошибку выборки (асимметричность) как данные, которые в некотором роде не соответствуют картине реального мира. Это ситуация, противоположная симметричности («справедливости»), которую мы определяем как предсказания, сделанные нашей моделью, оказывающие разнородное воздействие на различные группы людей.

Систематическая ошибка может быть привнесена в наши данные различными способами. Наш набор данных всегда является лишь некоторым подмножеством данных реального мира – мы не можем надеяться собрать полную информацию обо всем. То, как мы выбираем данные реального мира, всегда некоторым образом несет в себе определенную предвзятость. Один из типов асимметричности, который мы можем проверить, – это *систематическая ошибка выбора*: распределение нашего набора данных не совпадает с распределением данных в реальном мире.

Мы можем использовать TFDV для проверки смещения выбора с использованием визуализации данных статистики, которую мы описали выше. На-

пример, если наш набор данных содержит пол как категориальный признак, мы можем убедиться, что он не смещен в сторону признака «мужской». В нашем наборе данных, относящихся к потребителям, мы имеем категориальный признак State. В идеале распределение количества выборок по различным штатам США должно отражать относительную численность населения в каждом штате.

На рис. 4.8 мы можем видеть, что это не так (например, в Техасе, находящемся на 3-м месте, население больше, чем во Флориде, находящейся на 2-м месте). Если мы обнаружим этот тип систематической ошибки в наших данных и считаем, что эта систематическая ошибка может негативно повлиять на точность нашей модели, мы можем вернуться и собрать больше данных или сузить/расширить выборку наших данных для получения правильного распределения.

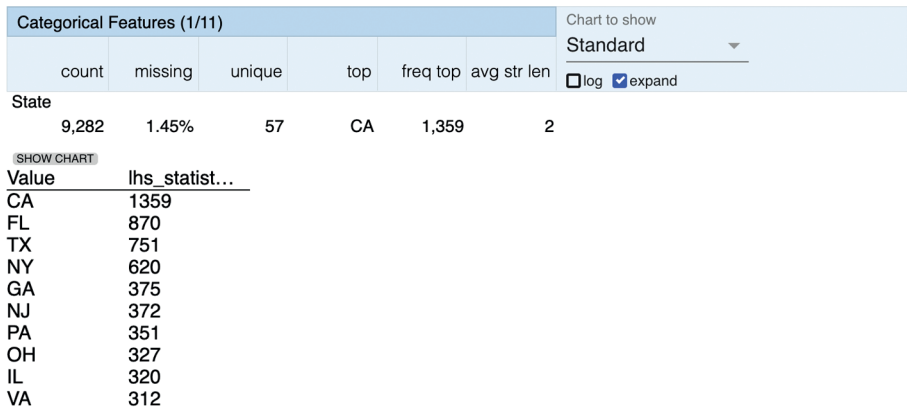


Рис. 4.8. Визуализация систематической ошибки для конкретного признака в нашем наборе данных

Вы также можете использовать протокол аномалий, о котором рассказывалось выше, для автоматических предупреждений о подобных проблемах. Используя знание предметной области, к которой относится ваш набор данных, вы можете установить ограничения для числовых значений, которые будут означать, что ваш набор данных настолько объективен, насколько это возможно, например если ваш набор данных содержит заработную плату людей в виде числового признака, вы можете установить, что среднее значение функции реалистично.

Для получения более подробной информации и знакомства с определением систематической ошибки просмотрите Google’s Machine Learning Crash Course (Ускоренный курс машинного обучения Google, доступный по ссылке <https://developers.google.com/machine-learning/crash-course/fairness/types-of-bias>), который содержит множество полезных материалов.

Получение среза данных в TFDV

Мы также можем использовать TFDV для получения среза наборов данных по выбранным нами признакам, чтобы показать, являются ли они наборами с систематической ошибкой выборки. Это похоже на вычисление достоверно-

сти модели на объектах, использующих срезы данных, которое мы описываем в главе 7. Например, способ привнесения систематической ошибки в наши данные, который бывает непросто обнаружить, – это отсутствие данных. Если данные пропущены не в случайном порядке, они могут чаще отсутствовать для определенной группы людей в наборе данных. Это может означать, что при обучении окончательной модели ее точность для этих групп ухудшается.

В этом примере мы рассмотрим данные из разных штатов США. Мы можем определить срез данных так, чтобы получать статистику только для Калифорнии, используя следующий код:

```
from tensorflow_data_validation.utils import slicing_util

slice_fn1 = slicing_util.get_feature_value_slicer(
    features={'state': [b'CA']}) ❶
slice_options = tfdv.StatsOptions(slice_functions=[slice_fn1])
slice_stats = tfdv.generate_statistics_from_csv(
    data_location='data/consumer_complaints.csv',
    stats_options=slice_options)
```

❶ Обратите внимание, что значение признака должно быть представлено в виде списка двоичных значений

Нам потребуется вспомогательный код, чтобы визуализировать данные статистики, полученные в срезе:

```
from tensorflow_metadata.proto.v0 import statistics_pb2

def display_slice_keys(stats):
    print(list(map(lambda x: x.name, slice_stats.datasets)))

def get_sliced_stats(stats, slice_key):
    for sliced_stats in stats.datasets:
        if sliced_stats.name == slice_key:
            result = statistics_pb2.DatasetFeatureStatisticsList()
            result.datasets.add().CopyFrom(sliced_stats)
            return result
    print('Invalid Slice key')

def compare_slices(stats, slice_key1, slice_key2):
    lhs_stats = get_sliced_stats(stats, slice_key1)
    rhs_stats = get_sliced_stats(stats, slice_key2)
    tfdv.visualize_statistics(lhs_stats, rhs_stats)
```

Мы можем визуализировать результаты, используя следующий код:

```
tfdv.visualize_statistics(get_sliced_stats(slice_stats, 'state_CA'))
```

Затем мы можем сравнить статистику по Калифорнии с общими результатами:

```
compare_slices(slice_stats, 'state_CA', 'All Examples')
```

Результаты выполнения этого кода показаны на рис. 4.9.

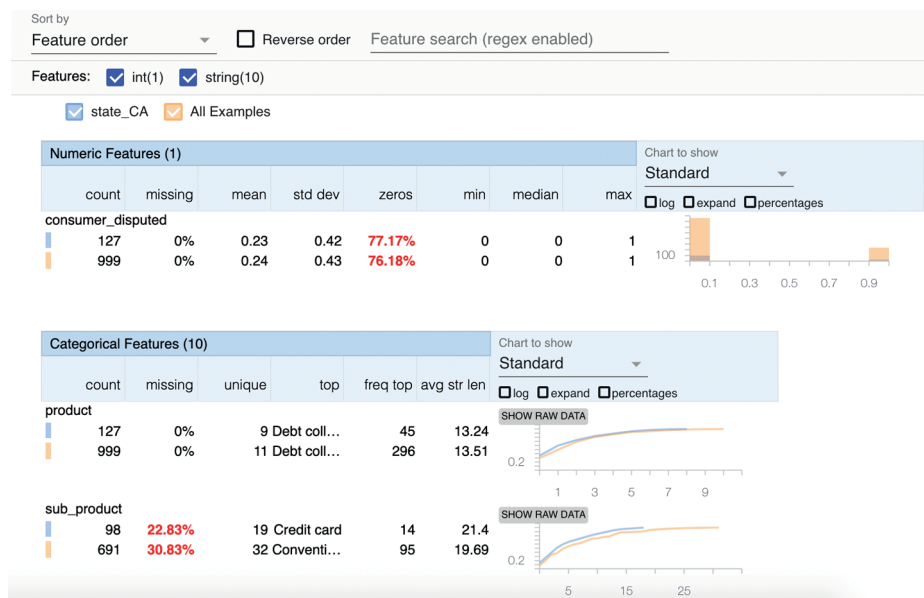


Рис. 4.9. Визуализация данных с разбивкой по значениям признаков

В этом разделе мы показали некоторые полезные функции TFDV, которые позволяют обнаруживать проблемы в ваших данных. Далее мы рассмотрим, как увеличить масштаб проверки данных с помощью программного продукта из Google Cloud.

ОБРАБОТКА БОЛЬШИХ НАБОРОВ ДАННЫХ С ПОМОЩЬЮ GOOGLE CLOUD PLATFORM

По мере того как мы собираем больше данных, проверка данных становится более трудоемким шагом в нашем рабочем процессе машинного обучения. Один из способов сократить время выполнения проверки – воспользоваться преимуществами доступных облачных решений. Используя облачного провайдера, мы не ограничиваемся вычислительной мощностью, ограниченной нашим ноутбуком или локальными вычислительными ресурсами.

В качестве примера применения облачного провайдера мы расскажем, как запустить проверку данных TensorFlow (TFDV) в продукте Google Cloud DataFlow. TensorFlow Data Validation работает на Apache Beam, что существенно упрощает переход на GCP DataFlow.

DataFlow от Google Cloud позволяет нам ускорить выполнение задач проверки данных, распараллеливая и распределяя их по выделенным для нашей задачи обработки данных узлам. Хотя DataFlow взимает плату за количество выделенных процессоров и гигабайт памяти, он может ускорить выполнение нашего этапа конвейера.

Мы продемонстрируем минимальную настройку для распределения наших задач проверки данных. Для получения дополнительной информации мы на-

стоятельно рекомендуем обратиться к расширенной документации Google Cloud Platform, размещенной по следующей ссылке: <https://cloud.google.com/dataflow/docs>. Мы полагаем, что у вас уже имеется учетная запись Google Cloud, настроены реквизиты для выставления счетов и настроена переменная среды `GOOGLE_APPLICATION_CREDENTIALS` в вашей терминальной оболочке. Если для того, чтобы начать работу, вам необходима помощь, вернитесь к содержанию главы 3 или к документации Google Cloud, размещенной по ссылке <https://cloud.google.com/docs/authentication/getting-started>.

Мы можем использовать метод, который мы обсуждали ранее, например `tfdv.generate_statistics_from_tfrecord`, но в данном случае методы потребуют дополнительных аргументов, `pipeline_options` и `output_path`. В то время как `output_path` указывает на сегмент Google Cloud, куда должны быть записаны результаты проверки данных, `pipeline_options` – это объект, который содержит всю необходимую информацию Google Cloud для запуска нашей проверки данных. В приведенном далее примере показано, как мы можем настроить такой объект конвейера.

```
from apache_beam.options.pipeline_options import (
    PipelineOptions, GoogleCloudOptions, StandardOptions)

options = PipelineOptions()
google_cloud_options = options.view_as(GoogleCloudOptions)
google_cloud_options.project = '<YOUR_GCP_PROJECT_ID>' ❶
google_cloud_options.job_name = '<YOUR_JOB_NAME>' ❷
google_cloud_options.staging_location = 'gs://<YOUR_GCP_BUCKET>/staging' ❸
google_cloud_options.temp_location = 'gs://<YOUR_GCP_BUCKET>/tmp'
options.view_as(StandardOptions).runner = 'DataflowRunner'
```

- ❶ Определите идентификатор вашего проекта
- ❷ Задайте имя для вашего задания (job)
- ❸ Укажите путь к хранилищу для промежуточных и временных файлов

Мы рекомендуем создать выделенный сегмент хранилища для ваших задач Dataflow. В корзине хранилища будут храниться все наборы данных и временные файлы.

После настройки параметров Google Cloud нам необходимо настроить параметры для объектов `worker` (исполнимых модулей) DataFlow. Все задачи выполняются в исполнимых модулях, где должны быть установлены необходимые пакеты для выполнения задач. В нашем случае нам нужно установить TFDV, указав его в качестве дополнительного пакета.

Для этого загрузите последний пакет проверки данных TensorFlow (двоичный файл `.whl`)¹ в вашу локальную систему. Выберите версию, которая может быть выполнена в системе Linux, например `tensorflow_data_validation-0.22.0-cp37-cp37m-manylinux2010_x86_64.whl`.

Чтобы задать параметры настройки объекта `worker`, укажите путь к загруженному пакету в списке `setup_options.extra_packages`, как показано в следующем примере:

¹ Вы можете загрузить пакеты TFDV, перейдя по ссылке <https://pypi.org/project/tensorflow-data-validation/#files>.

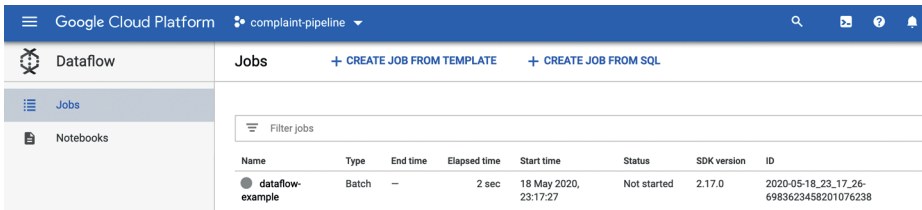
```
from apache_beam.options.pipeline_options import SetupOptions

setup_options = options.view_as(SetupOptions) setup_options.extra_packages = [
    '/path/to/tensorflow_data_validation'
    '-0.22.0-cp37-manylinux2010_x86_64.whl']
```

Определив все параметры, вы можете запустить задачу проверки данных со своего локального компьютера, и она будет выполняться в экземпляре Google Cloud DataFlow:

```
data_set_path = 'gs://<YOUR_GCP_BUCKET>/train_reviews.tfrecord'
output_path = 'gs://<YOUR_GCP_BUCKET>/'
tfdv.generate_statistics_from_tfrecord(data_set_path,
                                     output_path=output_path,
                                     pipeline_options=options)
```

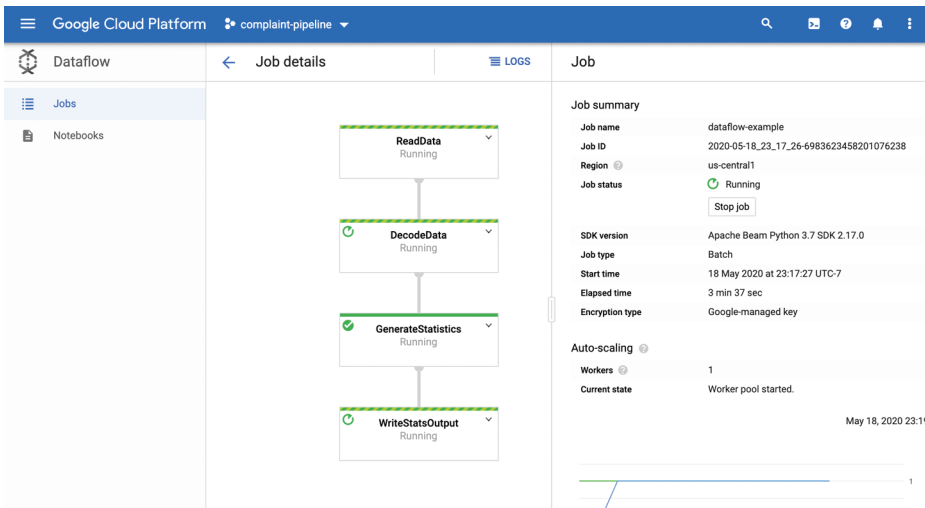
После того как вы начали проверку данных с помощью DataFlow, вы можете вернуться в консоль Google Cloud. Ваш недавно запущенный объект job должен отображаться в списке, аналогично тому, как это показано на рис. 4.10.



Name	Type	End time	Elapsed time	Start time	Status	SDK version	ID
dataflow-example	Batch	—	2 sec	18 May 2020, 23:17:27	Not started	2.17.0	2020-05-18_23_17-26-6983623458201076238

Рис. 4.10. Консоль Google Cloud DataFlow

Затем вы можете проверить информацию о выполняющемся задании (job), его статус и сведения о его автоматическом масштабировании, как показано на рис. 4.11.



ReadData Running

DecodeData Running

GenerateStatistics Running

WriteStatsOutput Running

Job summary

Job name: dataflow-example

Job ID: 2020-05-18_23_17-26-6983623458201076238

Region: us-central1

Job status: Running

SDK version: Apache Beam Python 3.7 SDK 2.17.0

Job type: Batch

Start time: 18 May 2020 at 23:17:27 UTC-7

Elapsed time: 3 min 37 sec

Encryption type: Google-managed key

Auto-scaling: Workers: 1, Current state: Worker pool started.

Рис. 4.11. Информация об объекте Job в Google Cloud DataFlow

С помощью нескольких шагов вы можете распараллелить и распределить ваши задачи проверки данных в облачной среде. В следующем разделе мы обсудим интеграцию задач проверки данных в наши автоматизированные конвейеры машинного обучения.

ИНТЕГРАЦИЯ TFDV В КОНВЕЙЕР МАШИННОГО ОБУЧЕНИЯ

Пока все методы, которые мы обсуждали, могли быть использованы в автономном варианте конфигурации конвейера. Это может быть полезно для изучения наборов данных за пределами нашего конвейера.

TFX включает компонент конвейера под названием StatisticsGen, который принимает выход из предыдущих компонентов ExampleGen в качестве входных данных, а затем выполняет генерацию статистики.

```
from tfx.components import StatisticsGen

statistics_gen = StatisticsGen(
    examples=example_gen.outputs['examples'])
context.run(statistics_gen)
```

Аналогично случаю, который мы обсуждали в главе 3, мы можем визуализировать вывод в интерактивном контексте, используя следующий код:

```
context.show(statistics_gen.outputs['statistics'])
```

Результат визуализации показан на рис. 4.11.

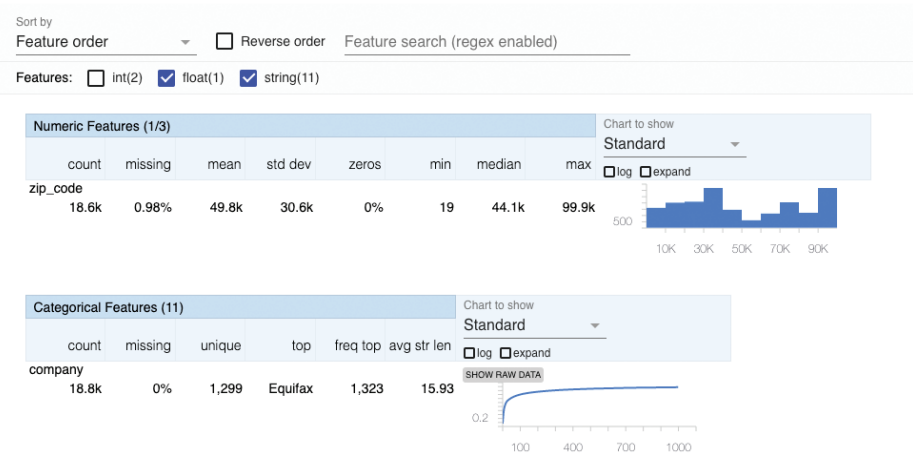


Рис. 4.11. Данные статистики, генерируемые компонентом StatisticsGen

Так же легко, как мы можем сгенерировать данные статистики, мы можем сгенерировать нашу схему данных:


```
from tfx.components import SchemaGen

schema_gen = SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    infer_feature_shape=True)
context.run(schema_gen)
```

Компонент `SchemaGen` генерирует схему только в том случае, если она еще не существует. Рекомендуется просмотреть схему при первом запуске этого компонента, а затем вручную изменить ее, если это необходимо, как мы обсуждали ранее (см. раздел «Обновление схемы»). Затем мы можем использовать эту схему до тех пор, пока не возникнет необходимость ее изменить, например если мы добавим новый признак.

Теперь, когда у нас есть данные статистики и схема, мы можем проверить наш новый набор данных:

```
from tfx.components import ExampleValidator

example_validator = ExampleValidator(
    statistics=statistics_gen.outputs['statistics'],
    schema=schema_gen.outputs['schema'])
context.run(example_validator)
```



`ExampleValidator` может автоматически обнаруживать отклонения от схемы с помощью компараторов отклонения и дрейфа данных, о которых рассказывалось ранее в этой главе. Однако это может не покрыть все потенциальные аномалии в ваших данных. Если вам нужно обнаружить некоторые другие специфические аномалии, вам нужно будет написать свой собственный компонент. Мы покажем, как это сделать, в главе 10.

Если компонент `ExampleValidator` обнаруживает несоответствие в статистических показателях между новым и предыдущим наборами данных для набора данных или схемы данных, компоненты будут иметь состояние *failed* в хранилище метаданных, и конвейер данных в конечном итоге остановится. В противном случае конвейер данных перейдет к следующему шагу – предварительной обработке данных.

РЕЗЮМЕ

В этой главе мы обсудили важность проверки данных и то, как вы можете эффективно выполнять и автоматизировать эту задачу. Мы обсудили, как генерировать статистические показатели и схемы данных и как сравнивать два разных набора данных на основе статистики и схем. Мы рассмотрели пример того, как вы можете выполнить проверку данных в Google Cloud с помощью `DataFlow`, и в конечном итоге мы интегрировали этот этап машинного обучения в наш автоматизированный конвейер. Это действительно важный шаг, так как он предотвращает попадание грязных данных на этапы предварительной обработки и обучения, требующие много времени.

Далее мы продолжим настраивать наш конвейер, начиная с предварительной обработки данных, о которой будет рассказываться в следующей главе.

Глава 5

Предварительная обработка данных

Данные, используемые для обучения наших моделей машинного обучения, часто предоставляются в форматах, с которыми модели машинного обучения не могут работать. Например, в нашем демонстрационном проекте признак, который мы хотим использовать для обучения нашей модели, представлен в виде тегов *Yes* и *No*. Для любой модели машинного обучения требуется числовое представление этих значений (например, 1 и 0). В этой главе мы объясним, как преобразовать признаки в согласованные числовые представления, чтобы обучать вашу модель с использованием числовых представлений признаков.

Один из основных аспектов, который мы обсуждаем в этой главе, – это упор на последовательную предварительную обработку. Как показано на рис. 5.1, предварительная обработка происходит после проверки данных, которую мы рассматривали в главе 4. TensorFlow Transform (TFT), компонент TFX для предварительной обработки данных, позволяет нам выстраивать наши шаги предварительной обработки в виде графов TensorFlow. В следующих разделах мы обсудим, почему и когда стоит использовать подобный подход и как экспортировать шаги предварительной обработки. В главе 6 мы будем использовать предварительно обработанные наборы данных и сохраненный граф преобразований для обучения и экспорта нашей модели машинного обучения.

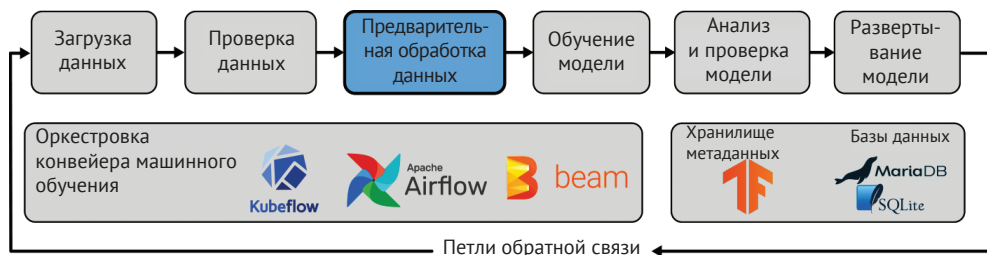


Рис. 5.1. Предварительная обработка данных как часть конвейеров машинного обучения

Специалисты по обработке данных могут посчитать этапы предварительной обработки, реализованные как операции TensorFlow, слишком большими

накладными расходами. В конце концов, для этого требуются другие возможности, чем те, которые вы могли бы использовать, автоматизировав шаг предварительной обработки с помощью Python `pandas` или `numpy`. Мы не агитируем за использование TFT на этапе экспериментов. Однако, как мы продемонстрируем в следующих разделах, преобразование шагов предварительной обработки в операции TensorFlow при переносе модели машинного обучения в производственную среду поможет избежать отклонений в поведении моделей при их обучении и производственной эксплуатации, как мы обсуждали в главе 4.

Для чего нужна предварительная обработка данных

По нашему опыту, из всех библиотек TFX самая крутая кривая обучения получается при использовании TFT, потому что в этом случае используется определение шагов предварительной обработки с помощью операций TensorFlow. Однако есть ряд веских причин, по которым предварительная обработка данных в конвейере машинного обучения должна быть стандартизирована и выполняться с использованием TFT. Вот несколько из них:

- эффективная предварительная обработка данных в контексте всего набора данных;
- эффективное масштабирование шагов предварительной обработки;
- предотвращение потенциального отклонения при обучении и работе модели.

Предварительная обработка данных в контексте всего набора данных

Когда мы хотим преобразовать наши данные в числовые представления, нам часто приходится делать это в контексте всего набора данных. Например, если мы хотим нормализовать числовой признак, мы должны сначала определить минимальное и максимальное значения признака в обучающем наборе. Когда определены границы, мы можем нормализовать наши данные в интервале значений от 0 до 1. Этот шаг нормализации обычно требует двух проходов по данным: один проход для определения границ и один для преобразования каждого значения признака. TFT предоставляет нам функции для управления операциями над данными без активного участия пользователя.

Масштабирование шагов предварительной обработки

TFT использует Apache Beam для выполнения инструкций предварительной обработки. Это позволяет нам при необходимости распределить предварительную обработку на выбранном нами бэкенде Apache Beam. Если у вас нет доступа к продукту Google Cloud Dataflow или кластеру Apache Spark или Apache Flink, Apache Beam по умолчанию вернется в режим Direct Runner.

Как избежать отклонения при обучении и работе модели

TFT создает и сохраняет граф шагов предварительной обработки TensorFlow. Во-первых, он создаст граф для обработки данных (например, для определения

минимальных и максимальных значений). Далее он сохранит граф и определенные границы. Затем этот граф можно использовать на этапе формирования выводов в контексте жизненного цикла модели. Этот процесс гарантирует, что модель на этапе формирования выводов видит те же шаги предварительной обработки, что и модель, используемая во время обучения.

Что такое отклонение при обучении и работе модели?

Мы говорим об отклонении при обучении и работе модели, когда шаги предварительной обработки, используемые во время обучения модели, не соответствуют шагам, используемым во время формирования выводов. Во многих случаях данные, применяемые для обучения моделей, обрабатываются в блокнотах Python Notebooks с помощью pandas или же с использованием задач (jobs) в Spark. Когда модель разворачивается в промышленной среде, этапы предварительной обработки реализуются в API до того, как данные попадут в модель для прогнозирования. Как вы видите на рис. 5.2, эти два процесса требуют координации, цель которой – гарантировать, что шаги всегда одинаковы.

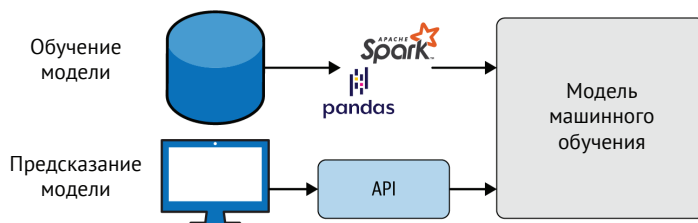


Рис. 5.2. Часто используемая конфигурация машинного обучения

С помощью TFT мы можем избежать отклонения на этапах предварительной обработки. Как показано на рис. 5.3, клиент, который хочет получить прогноз, теперь может отправить необработанные данные, а предварительная обработка происходит в графе модели, размещенной в промышленной среде.

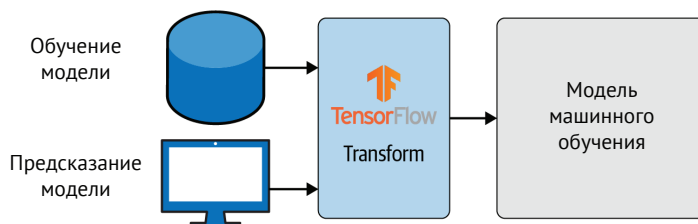


Рис. 5.3. Как избежать отклонения при обучении с помощью TFT

Такая установка сокращает объем усилий по координации и упрощает развертывание.

Развертывание шагов предварительной обработки и модели машинного обучения как единого артефакта

Чтобы избежать несоответствия между этапами предварительной обработки и обученной моделью, экспортированная модель нашего конвейера должна включать граф предварительной обработки и обученную модель. Затем мы можем развернуть модель, как любую другую модель TensorFlow, но во время формирования выводов данные будут предварительно обрабатываться на сервере модели как часть вывода модели. Это позволяет избежать необходимости выполнения предварительной обработки на стороне клиента и упрощает разработку клиентских приложений (например, веб-приложений или мобильных приложений), которые выполняют запросы к модели, чтобы получить результаты прогнозирования. В главах 11 и 12 мы обсудим, как весь сквозной конвейер создает такие «комбинированные» сохраненные модели.

Проверка результатов предварительной обработки в конвейере

Внедрение предварительной обработки данных с помощью TFT и ее интеграция в наш конвейер дают нам дополнительное преимущество. Мы можем генерировать данные статистики на основе предварительно обработанных данных и проверять, соответствуют ли они нашим требованиям к обучению модели. Примером этого варианта использования является преобразование текста в токены (tokens). Если текст содержит много новых слов, неизвестные токены будут преобразованы в так называемые UNK (от слова *unknown*), или *неизвестные* токены. Если определенное количество наших токенов просто неизвестно, модели машинного обучения часто бывает сложно эффективно обобщить данные, и, следовательно, это повлияет на точность модели. В наших конвейерах мы теперь можем проверять результаты этапа предварительной обработки, генерируя данные статистики (как рассказывалось в главе 4) после этапа предварительной обработки.



Различия между `tf.data` и `tf.transform`

Часто путают `tf.data` и `tf.transform`. `tf.data` – это API TensorFlow для создания эффективных конвейеров ввода при обучении модели с помощью TensorFlow. Главная задача библиотеки – оптимально использовать аппаратные ресурсы, такие как центральный процессор и оперативная память, для загрузки и предварительной обработки данных, которые происходят во время обучения. С другой стороны, `tf.transform` используется для выражения предварительной обработки, которая должна происходить как во время обучения, так и на этапе формирования выводов. Библиотека позволяет выполнять полнопроходный анализ входных данных (например, для вычислений словаря или статистики, используемой для нормализации данных), и этот анализ выполняется перед обучением.

ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ДАННЫХ С ПОМОЩЬЮ TFT

TFT – это библиотека для предварительной обработки данных в экосистеме TensorFlow. Как и TFDV, она является частью проекта TFX.

TFT обрабатывает данные, которые мы загрузили в наш конвейер, с помощью ранее созданной схемы набора данных, и выдает на выходе два артефакта:

- предварительно обработанные наборы данных для обучения и оценки в формате TFRecord. Созданные наборы данных можно использовать далее в работе нашего конвейера, в компоненте Trainer;
- экспортированный граф предварительной обработки (с ресурсами), который будет использоваться при экспорте нашей модели машинного обучения.

Ключевая функция TFT – это функция `preprocessing_fn`, как показано на рис. 5.4. Она определяет все преобразования, которые мы хотим применить к *необработанным* данным. Когда мы запускаем исполнение компонента Transform, функция `preprocessing_fn` получает необработанные данные, применяет преобразование и возвращает обработанные данные. Данные предоставляются как TensorFlow Tensors или SparseTensors (в зависимости от функции). Все преобразования, применяемые к тензорам, должны быть операциями TensorFlow. Это позволяет TFT эффективно распределить шаги предварительной обработки.

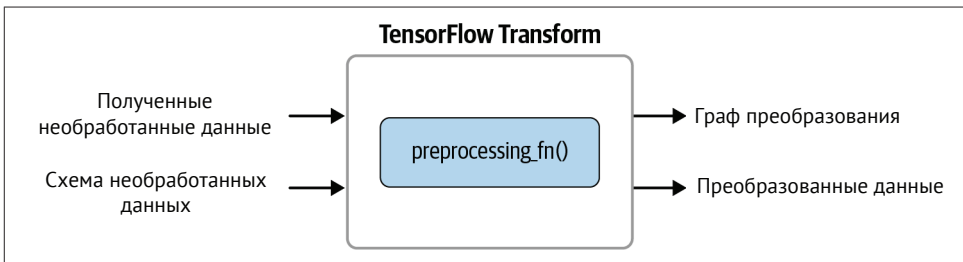


Рис. 5.4. Обзор TFT



Функции TFT

Функции TFT, которые выполняют сложные этапы обработки без активного участия пользователя, такие как `tft.compute_and_apply_vocabulary`, можно распознать по префиксу `tft`. Обычно TFT сопоставляется с аббревиатурой `tft` в пространстве имен Python. Обычные операции TensorFlow будут загружаться с общим префиксом `tf`, как, например, в `tf.reshape`.

TensorFlow Transform также имеет полезные функции (например, `tft.bucketize`, `tft.compute_and_apply_vocabulary` или `tft.scale_to_z_score`). Когда эти функции применяются к объекту набора данных, они выполняют проход по данным, а затем применяют полученные границы к данным. Например, `tft`.

`compute_and_apply_vocabulary` сгенерирует набор словарей совокупности данных, применит созданное отображение токена в индекс к признаку и вернет значение индекса. Функция может ограничить количество токенов словаря, взяв n первых наиболее релевантных токенов. В следующих разделах мы покажем некоторые из наиболее полезных операций с TFT.

Установка

При установке пакета `tfx`, описанной в главе 2, TFT был установлен как зависимый пакет. Если мы хотим использовать TFT как отдельный пакет, то можем установить пакет `PuPI`, выполнив команду

```
$ pip install tensorflow-transform
```

После установки `tfx` или `tensorflow-transform` мы можем интегрировать шаги предварительной обработки в наши конвейеры машинного обучения. Давайте рассмотрим пару примеров использования.

Стратегии предварительной обработки

Как мы обсуждали ранее, применяемые преобразования определяются в функции `preprocessing_fn()`. Затем эта функция будет использоваться нашим компонентом конвейера преобразования или нашим экземпляром TFT, развернутым при автономной установке. Вот пример функции предварительной обработки, которую мы подробно обсудим в следующих разделах:

```
def preprocessing_fn(inputs): x =
    inputs['x']
    x_normalized = tft.scale_to_0_1(x)
    return {
        'x_xf': x_normalized
    }
```

Функция получает пакет входных данных в виде словаря Python. Ключ – это имя функции и значения, представляющие необработанные данные до применения предварительной обработки. Сначала TFT выполняет этап анализа, как показано на рис. 5.5. В случае нашего небольшого демонстрационного примера он определит минимальное и максимальное значения нашей функции путем полного прохода по данным. Этот шаг может выполняться распределенно благодаря выполнению шагов предварительной обработки в Apache Beam.

Во втором проходе по данным определенные значения (в нашем случае минимальное и максимальное значения столбца функций) используются для нормирования нашей функции x в интервале значений между 0 и 1, как показано на рис. 5.6.

TFT также генерирует граф для формирования прогноза с сохраненными минимальным и максимальным значениями. Это обеспечивает согласованность выполнения этапов.

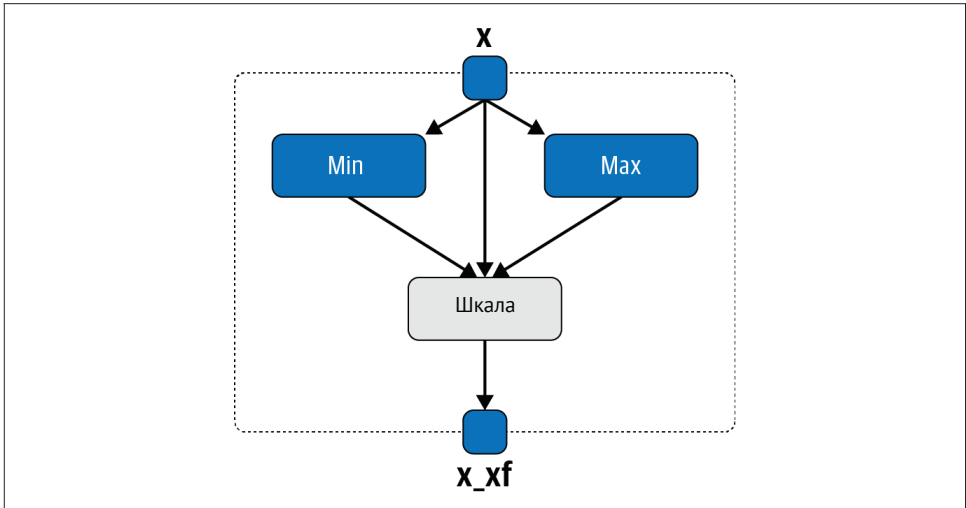


Рис. 5.5. Этап анализа при работе TFT

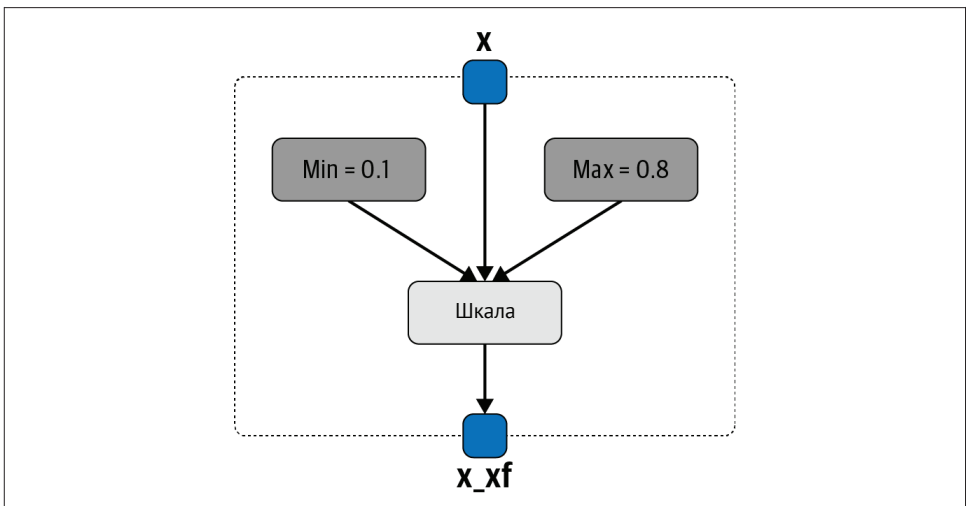


Рис. 5.6. Применение результатов этапа анализа



preprocessing_fn()

Обратите внимание, что TFT построит граф, используя функцию `preprocessing_fn()`, и будет работать в своем собственном сеансе. Ожидается, что функция вернет словарь с преобразованными признаками как значения словаря Python.

Лучшие практики

За время работы с TFT мы извлекли немало уроков. Вот некоторые из них.

Важно правильно давать названия признакам

Правильно именовать выходные характеристики предварительной обработки – очень важный момент работы. Как вы увидите далее в примерах, иллюстрирующих использование TFT, для выходных признаков мы используем те же имена, что и для входных признаков, добавляя к ним `_xf`. Кроме того, имена входных узлов моделей TensorFlow должны совпадать с именами выходных признаков, получающихся в результате применения функции `preprocessing_fn`.

Продумайте использование типов данных

TFT ограничивает использование типов данных выходных признаков. Он экспортирует все предварительно обработанные признаки как значения `tf.string`, `tf.float32` или `tf.int64`. Об этом особенно важно подумать, если ваша модель не может использовать эти типы данных. Некоторые модели из TensorFlow Hub требуют, чтобы входные данные были представлены как значения `tf.int32` (например, модели BERT). Мы можем избежать данной ситуации, если приведем входные данные к правильным типам данных внутри наших моделей или если преобразуем типы данных с помощью входных признаков оценки.

Предварительная обработка выполняется в пакетном режиме

Когда вы разрабатываете функции предварительной обработки, вы можете думать об этом процессе как об обработке одной строки данных за один раз. Однако фактически TFT выполняет операции в пакетном режиме. Вот почему нам нужно будет преобразовать вывод функции `preprocessing_fn()` в `Tensor` или `SparseTensor`, когда мы будем использовать его в контексте нашего компонента `Transform`.

Помните: энергичные вычисления не используются

Признаки внутри `preprocessing_fn()` должны быть представлены операциями TensorFlow. Если вы хотите перевести в нижний регистр входную строку, вы не можете использовать функцию `lower()`. Вы должны использовать операцию TensorFlow `tf.strings.lower()`, чтобы выполнить ту же процедуру в графовом режиме. Энергичные вычисления (вычисления по принципу «вычислять, не откладывая, все, что возможно») не поддерживаются; все операции выражаются через операции TensorFlow, определяемые графом.

`tf.function` может использоваться в функциях `preprocessing_fn()`, но с ограничениями: вы можете использовать только `tf.function`, которая работает с тензорами (таким образом, `lower()` не будет работать, поскольку она не обладает подобной возможностью). Вы не можете вызвать анализатор TFT (или модуль сопоставления, которое работает с анализатором, например `tft.scale_to_z_score`).

Функции TFT

TFT предоставляет множество функций для облегчения эффективного проектирования функций. Список предоставляемых функций обширен и постоянно расширяется. Вот почему мы не претендуем на то, чтобы представить полный список поддерживаемых функций, но хотим выделить полезные операции, связанные с генерацией словаря, нормализацией и размещением признаков:

`tft.scale_to_z_score()`

Если вы хотите нормализовать функцию со средним значением 0 и стандартным отклонением 1, вы можете использовать эту полезную функцию TFT.

`tft.bucketize()`

Эта полезная функция позволяет распределить признак по группам. Он возвращает индекс контейнера или сегмента хранилища. Вы можете указать аргумент `num_buckets`, чтобы установить количество сегментов. Затем TFT выделит сегменты одинакового размера.

`tft.pca()`

Эта функция позволяет вам выполнять анализ главных компонентов (principal component analysis, PCA) для заданного признака. PCA – это распространенный метод уменьшения размерности путем линейного проецирования данных вниз в подпространство, которое лучше всего сохраняет дисперсию данных. Для того чтобы определить размерность вашего представления PCA, потребуется аргумент `output_dim`.

`tft.compute_and_apply_vocabulary()`

Это одна из самых удивительных функций TFT. Он вычисляет все уникальные значения столбца признаков, а затем сопоставляет наиболее частые значения с индексом. Это отображение индекса потом используется для преобразования объекта в числовое представление. Функция генерирует все параметры для вашего графа без необходимости вмешательства пользователя. Мы можем настроить параметр, определяющий понятие *наиболее частые*, двумя способами: либо путем определения *n* уникальных элементов с наивысшим рейтингом с помощью `top_k`, либо используя значение `frequency_threshold` для каждого элемента, рассматриваемого для включения в словарь.

`tft.apply_saved_model()`

Эта функция позволяет применять к признаку полные модели TensorFlow. Мы можем загрузить сохраненную модель с заданным тегом и `signature_name`, а затем передать входные данные `inputs` в модель. После этого будут возвращены результаты прогнозирования модели.

Текстовые данные для задач обработки естественного языка

Если вы работаете над задачами обработки естественного языка и намерены использовать TFT для предварительной обработки вашего текстового массива, чтобы преобразовать ваши документы в числовое представление, то в TFT имеется большое количество функций для этой цели. Помимо введенной функции `tft.compute_and_apply_vocabulary()`, вы также можете использовать следующие функции TFT.

`tft.ngrams()`

Эта функция генерирует n -граммы. В качестве входных данных она принимает набор данных `SparseTensor`, состоящий из строковых значений. Например, если вы хотите сгенерировать однограммы и биграммы для списка `['Tom', 'and', 'Jerry', 'are', 'friends']`, функция возвращает `[b'Tom', b'Tom and', b'and', b'and Jerry', b'Jerry', b'Jerry are', b'are', b'are friends', b'friends']`.

Помимо `SparseTensor`, функция принимает два дополнительных аргумента: `ngram_range` и `separator`. `ngram_range` устанавливает диапазон n для n -граммов. Если ваши n -граммы должны содержать одно- и биграммы, установите для `ngram_range` значение `(1, 2)`. Разделитель позволяет нам установить соединяющую строку или символ. В нашем примере мы установили разделитель `" "`.

`tft.bag_of_words()`

Эта функция использует `tft.ngrams` и генерирует вектор набора слов со строкой для каждой уникальной n -граммы. Исходный порядок n -граммов может не сохраниться, если, например, токены повторяются внутри ввода.

`tft.tfidf()`

Часто используемым понятием в задачах обработки естественного языка является TF-IDF, или «частота употребления слова в документе – обратная частота документа»¹. Эта функция генерирует два вывода: вектор с индексами токенов и вектор, представляющий их веса TF-IDF. Функция принимает на вход разреженный входной вектор, представляющий индексы токенов (результат применения функции `tft.compute_and_apply_vocabulary()`). Размерность этих векторов устанавливается входным аргументом `vocab_size`. Вес для каждого индекса токена рассчитывается путем умножения частоты токена в документе на обратную частоту документа. Для этих вычислений часто требуются значительные ресурсы. Следовательно, использование распределенных вычислений (для вычисления TFT) обеспечит большое преимущество.

Библиотека TensorFlow Text (см. <https://github.com/tensorflow/text>) предоставляет возможность использовать все ее функции. Библиотека предоставляет обширную поддержку TensorFlow для нормализации текста, токенизации текста, вычисления n -граммов и использования современных языковых моделей, таких как BERT.

Видеоинформация для задач компьютерного зрения

Если вы работаете с моделями компьютерного зрения, то с помощью TFT можете предварительно обработать наборы данных изображений. TensorFlow предоставляет возможность выполнять различные операции предварительной обработки изображений, используя API `tf.image` (см. документацию, доступную по ссылке https://www.tensorflow.org/api_docs/python/tf/image) и `tf.io` (см. документацию, доступную по ссылке https://www.tensorflow.org/api_docs/python/tf/io).

¹ TF-IDF представляет собой произведение двух сомножителей, первый из которых – отношение числа вхождений некоторого слова к общему числу слов документа, второй – обратная частота документа, или инверсия частоты, с которой некоторое слово встречается в коллекции документов. – Прим. перев.

`tf.io` предоставляет функции, предоставляющие возможность открытия изображений как части графа модели (например, `tf.io.decode_jpeg` и `tf.io.decode_png`). `tf.image` предоставляет функции обрезки или изменения размера изображений, преобразования цветовых схем, настройки изображений (например, контрастности, оттенка или яркости) или выполнения преобразований изображений, таких как переворачивание изображения, транспонирование и т. д.

В главе 3 мы обсуждали стратегии загрузки изображений в наши конвейеры. Теперь, используя TFT, мы можем читать закодированные изображения из файлов `TfRecord` и, например, изменять их размер до фиксированного размера или преобразовывать цветные изображения до изображений в оттенках серого. Вот пример реализации такой функции `preprocessing_fn`:

```
def process_image(raw_image):
    raw_image = tf.reshape(raw_image, [-1])
    img_rgb = tf.io.decode_jpeg(raw_image, channels=3) ❶
    img_gray = tf.image.rgb_to_grayscale(img_rgb) ❷
    img = tf.image.convert_image_dtype(img, tf.float32)
    resized_img = tf.image.resize_with_pad( ❸
        img, target_height=300,
        target_width=300
    )
    img_grayscale = tf.image.rgb_to_grayscale(resized_img) ❹
    return tf.reshape(img_grayscale, [-1, 300, 300, 1])
```

- ❶ Декодирование формата изображения JPEG
- ❷ Преобразование загруженного изображения RGB в оттенки серого
- ❸ Изменение размера изображения до 300×300 пикселей
- ❹ Преобразование изображения в оттенки серого

Одно замечание относительно операции `tf.reshape()` как части оператора `return`: TFT может обрабатывать входные данные в пакетном режиме. Поскольку размер пакета обрабатывается TFT (и Apache Beam), нам нужно изменить формат вывода нашей функции, чтобы обрабатывать любой размер пакета. Поэтому мы устанавливаем первую размерность нашего возвращаемого тензора равной `-1`. Остальные размерности представляют наши изображения. Мы изменяем их размер до 300×300 пикселей и преобразуем каналы RGB в каналы в оттенках серого, уменьшая размер изображений.

Автономная работа TFT

После того как мы определили нашу функцию `preprocessing_fn`, нам нужно сосредоточиться на том, как выполнить функцию `Transform`. У нас есть два варианта: мы можем выполнить преобразования предварительной обработки данных с помощью автономной установки или как часть нашего конвейера машинного обучения, как его компонента TFX. Оба варианта могут быть реализованы с использованием локальной настройки Apache Beam или службы Dataflow Google Cloud. В этом разделе мы обсудим автономные варианты работы TFT. Такой подход рекомендуется в ситуациях, когда вы хотите эффективно обрабатывать данные вне конвейеров. Если вас интересует, как интегрировать TFT в конвейер, смело переходите к разделу «Интеграция TFT в конвейер машинного обучения» этой главы.

Описание всех функциональных возможностей Apache Beam выходит за рамки данной книги. Эта тема заслуживает отдельной публикации. Однако мы хотим показать вам элементарный пример предварительной обработки с помощью Apache Beam.

В нашем примере мы хотели бы применить функцию предварительной обработки нормализации, которую представили ранее, к нашему очень маленькому набору необработанных данных, представленному в следующем исходном коде:

```
raw_data = [
    {'x': 1.20},
    {'x': 2.99},
    {'x': 100.00}
]
```

Во-первых, нам нужно определить схему данных. Мы можем сгенерировать схему из спецификации признаков, как показано в приведенном ниже исходном коде. Наш крошечный набор данных содержит только один признак с именем `x`. Мы определяем функцию с типом данных `tf.float32`:

```
import tensorflow as tf
from tensorflow_transform.tf_metadata import dataset_metadata
from tensorflow_transform.tf_metadata import schema_utils

raw_data_metadata = dataset_metadata.DatasetMetadata(
    schema_utils.schema_from_feature_spec({
        'x': tf.io.FixedLenFeature([], tf.float32),
    }))
```

После того как мы загрузили набор данных и сгенерировали схему данных, мы можем выполнить функцию предварительной обработки `preprocessing_fn`, которую определили ранее. TFT позволяет выполнить в Apache Beam функцию `AnalyzeAndTransformDataset`. Эта функция исполняет двухэтапный процесс, который мы обсуждали ранее: сначала анализирует набор данных, а затем преобразует его. Выполнение осуществляется с помощью диспетчера контекста Python `tft_beam.Context`, который позволяет нам установить, например, желаемый размер пакета. Однако мы рекомендуем использовать размер пакета по умолчанию, потому что в обычных случаях он более эффективен. В следующем примере показано использование функции `AnalyzeAndTransformDataset`:

```
import tempfile
import tensorflow_transform.beam.impl as tft_beam

with beam.Pipeline() as pipeline:
    with tft_beam.Context(temp_dir=tempfile.mkdtemp()):

        tftrecord_file = "/your/tf_records_file.tfrecord"
        raw_data = (
            pipeline | beam.io.ReadFromTFRecord(tftrecord_file))

        transformed_dataset, transform_fn = (
            (raw_data, raw_data_metadata) | tft_beam.AnalyzeAndTransformDataset(
                preprocessing_fn))
```

Синтаксис вызовов функций Apache Beam немного отличается от обычных вызовов Python. В предыдущем примере мы используем функцию `preprocessing_fn` с функцией Apache Beam `AnalyzeAndTransformDataset()` и передаем два аргумента с нашими данными `raw_data` и нашей определенной схемой метаданных `raw_data_metadata`. Затем `AnalyzeAndTransformDataset()` возвращает два артефакта: предварительно обработанный набор данных и функцию `transform_fn`, представляющую операции преобразования, примененные к нашему набору данных.

Если мы протестируем наш элементарный пример, выполним шаги предварительной обработки и распечатаем результаты, то увидим крошечный обработанный набор данных:

```
transformed_data, transformed_metadata = transformed_dataset
print(transformed_data)
[
    {'x_xf': 0.0},
    {'x_xf': 0.018117407},
    {'x_xf': 1.0}
]
```

В нашем элементарном примере мы полностью проигнорировали тот факт, что данные недоступны в виде словаря Python, который часто необходимо читать с диска. Apache Beam предоставляет возможности для эффективной обработки загрузки файлов (например, с использованием функций `beam.io.ReadFromText()` или `beam.io.ReadFromTFRecord()`) в контексте построения моделей TensorFlow.

Как видите, конфигурирование исполнения Apache Beam может быстро усложниться, и мы понимаем, что специалисты по обработке данных и инженеры по машинному обучению не занимаются написанием подобных инструкций «с нуля». Вот почему TFX так удобен. Он абстрагирует все скрытые инструкции и позволяет специалисту по обработке данных сосредоточиться на настройках для конкретных задач, таких как определение функции `preprocessing_fn()`. В следующем разделе мы более подробно рассмотрим настройку `Transform` для нашего демонстрационного проекта.

Интеграция TFT в конвейер машинного обучения

В последнем разделе этой главы мы обсудим, как применить возможности TFT в нашем демонстрационном проекте. В главе 4 мы исследовали набор данных и определили, какие признаки являются категориальными или числовыми, какие признаки должны быть разделены на сегменты, а какие признаки мы хотим преобразовать из строкового представления в векторное. Эта информация имеет решающее значение для целей конструирования признаков.

В следующем коде мы определяем наши признаки. Для упрощения обработки в дальнейшем мы группируем имена входных функций в словари, представляющих каждый тип выходных данных преобразования: признаки с однофакторным кодированием, контейнеризованные признаки и необработанные строковые представления:

```
import tensorflow as tf
import tensorflow_transform as tft

LABEL_KEY = "consumer_disputed"

# Наименование признака, размерность признака.
ONE_HOT_FEATURES = {
    "product": 11,
    "sub_product": 45,
    "company_response": 5,
    "state": 60,
    "issue": 90
}

# Наименование признака, число контейнеров.
BUCKET_FEATURES = {
    "zip_code": 10
}

# Наименование признака, значение не используется.
TEXT_FEATURES = {
    "consumer_complaint_narrative": None
}
```

Прежде чем мы сможем использовать эти входные словари признаков, давайте определим несколько вспомогательных функций для эффективного преобразования данных. Рекомендуются переименовывать признаки, добавляя суффикс к наименованию признака (например, `_xf`). Использование суффикса поможет различить, связаны ли ошибки с входными или выходными признаками, и предотвратит случайное использование непреобразованных признаков в нашей реальной модели:

```
def transformed_name(key):
    return key + '_xf'
```

Некоторые из наших признаков разрежены; однако TFT ожидает, что результаты преобразования будут плотными. Мы можем использовать следующую вспомогательную функцию для преобразования разреженных признаков в плотные и для заполнения отсутствующих значений значением по умолчанию:

```
def fill_in_missing(x):
    default_value = '' if x.dtype == tf.string or to_string else 0
    if type(x) == tf.SparseTensor:
        x = tf.sparse.to_dense(
            tf.SparseTensor(x.indices, x.values, [x.dense_shape[0], 1]), default_value)
    return tf.squeeze(x, axis=1)
```

В нашей модели мы представляем большинство входных признаков в виде закодированных однофакторных векторов. Следующая вспомогательная функция преобразует данный индекс в закодированный однофакторный вектор и возвращает полученный вектор:


```
def convert_num_to_one_hot(label_tensor, num_labels=2):
    one_hot_tensor = tf.one_hot(label_tensor, num_labels)
    return tf.reshape(one_hot_tensor, [-1, num_labels])
```

Прежде чем мы сможем обработать наши функции, нам понадобится еще одна вспомогательная функция для преобразования почтовых индексов, представленных в виде строк, в значения типа *float*. В нашем наборе данных почтовые индексы перечислены ниже:

```
zip_codes
97XXX
98XXX
```

Чтобы правильно контейнеризовать записи с отсутствующими почтовыми индексами, мы заменили заполнители нулями и разделили получившиеся числа типа *float* на 10 сегментов:

```
def convert_zip_code(zip_code):
    if zip_code == '':
        zip_code = "00000"
    zip_code = tf.strings.regex_replace(zip_code, r'X{0,5}', "0")
    zip_code = tf.strings.to_number(zip_code, out_type=tf.float32)
    return zip_code
```

Применив там, где нужно, все вспомогательные функции, мы теперь можем перебирать каждый столбец признаков и преобразовывать его в зависимости от типа. Например, чтобы наши признаки были преобразованы в однофакторные признаки, мы преобразуем наименования признаков в индекс с помощью `tft.compute_and_apply_vocabulary()`, а затем преобразуем индекс в быстрое векторное представление с помощью нашей вспомогательной функции `convert_num_to_one_hot()`. Поскольку мы используем `tft.compute_and_apply_vocabulary()`, TensorFlow Transform сначала выполнит цикл по всем категориям, а затем определит категорию для сопоставления индекса. Далее это сопоставление будет применяться на этапе оценки и обслуживания модели:

```
def preprocessing_fn(inputs): outputs = {}
for key in ONE_HOT_FEATURES.keys(): dim = ONE_HOT_FEATURES[key]
index = tft.compute_and_apply_vocabulary( fill_in_missing(inputs[key]), top_k=dim + 1)
outputs[transformed_name(key)] = convert_num_to_one_hot( index, num_labels=dim + 1)
...
return outputs
```

Наша обработка признаков в сегменте данных очень похожа на приведенный пример. Мы решили прибегнуть к контейнеризации почтовых индексов, потому что представление почтовых индексов в виде закодированных однофакторных векторов казалось слишком разреженным. В нашем случае каждый признак разделен на 10 контейнеров, и мы представляем индекс сегмента как однофакторный вектор:


```
for key, bucket_count in BUCKET_FEATURES.items():
    temp_feature = tft.bucketize(
        convert_zip_code(fill_in_missing(inputs[key])),
        bucket_count,
        always_return_num_quantiles=False)
    outputs[transformed_name(key)] = convert_num_to_one_hot( temp_feature,
        num_labels=bucket_count + 1)
```

Наши текстовые входные признаки, а также столбец с метками не требуют каких-либо преобразований; поэтому мы просто преобразуем их в плотные признаки в том случае, если имеем разреженный признак:

```
for key in TEXT_FEATURES.keys():
    outputs[transformed_name(key)] = \
        fill_in_missing(inputs[key])

outputs[transformed_name(LABEL_KEY)] = fill_in_missing(inputs[LABEL_KEY])
```



Почему мы не встроили текстовые признаки в векторы

Вы можете задаться вопросом, почему мы не встроили наши текстовые признаки в фиксированный вектор на этапе преобразования. Конечно, это возможно. Но мы решили вместо предварительной обработки загрузить модель TensorFlow Hub как часть модели. Основная причина этого решения заключалась в том, что мы могли сделать встраиваемые элементы обучаемыми и обновлять векторные представления на этапе обучения. Следовательно, они не могут быть жестко закодированы на этапе предварительной обработки и представлены в виде зафиксированного графа на этапе обучения.

Если мы используем компонент Transform из TFX в нашем конвейере, то ожидается, что код преобразования будет предоставлен в отдельном файле Python. Имя файла модуля может быть задано пользователем (например, в нашем случае `module.py`), но точка входа `preprocessing_fn()` должна находиться в файле модуля, и функция не может быть переименована:

```
transform = Transform(
    examples=example_gen.outputs['examples'],
    schema=schema_gen.outputs['schema'],
    module_file=os.path.abspath("module.py"))
context.run(transform)
```

Когда мы запускаем на исполнение компонент Transform, TFX применяет преобразования, определенные в нашем файле модуля `module.py`, к загруженным входным данным, которые были преобразованы в структуры данных TFRecord на этапе загрузки данных. Затем компонент выведет наши преобразованные данные, граф преобразования и необходимые метаданные.

Преобразованные данные и граф преобразования можно использовать на следующем шаге, в компоненте Trainer. Просмотрите раздел «Запуск компонента Trainer» главы 6, чтобы узнать, как использовать выходные данные нашего компонента Transform. В следующей главе также рассказывается, как

сгенерированный граф преобразования может быть объединен с обученной моделью для экспорта сохраненной модели. Более подробную информацию можно найти в примере 6.2.

РЕЗЮМЕ

В этой главе мы обсудили, как эффективно обрабатывать данные в наших конвейерах машинного обучения с помощью TFT. Мы показали, как писать функции `preprocessing_fn`, дали обзор некоторых доступных функций, предоставляемых TFT, и обсудили, как интегрировать шаги предварительной обработки в конвейер TFX. Теперь, когда предварительная обработка данных завершена, пришло время обучить нашу модель.

Глава 6

Обучение модели

Теперь, когда этап предварительной обработки данных завершен и данные преобразованы в необходимый для нашей модели формат, следующим шагом для нашего конвейера является обучение модели с использованием только что преобразованных данных.

Как мы обсуждали в главе 1, мы не будем рассматривать процесс выбора архитектуры вашей модели. Мы предполагаем, что у вас есть отдельный процесс проведения экспериментов, который проводился еще до того, как вы взяли в руки эту книгу, и что вы уже знаете тип модели, которую хотите обучать. Мы обсудим, как отслеживать процесс проведения экспериментов, в главе 15, как шаг, помогающий нам создать полный контрольный журнал для модели. Однако в данной книге мы не рассматриваем теоретические основы, необходимые для понимания процесса обучения модели. Если вы хотите узнать об этом больше, мы настоятельно рекомендуем публикацию O'Reilly *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd edition* («Практическое машинное обучение с помощью Scikit-Learn, Keras и TensorFlow», 2-е изд.).

В этой главе мы рассмотрим процесс обучения модели как часть конвейера машинного обучения, включая его автоматизацию в конвейере TFX. Мы также включаем некоторые подробности о стратегиях распределения, доступных в TensorFlow, и о том, как настраивать гиперпараметры в конвейере. Эта глава более сфокусирована на конвейерах TFX, чем большинство других, поскольку мы не рассматриваем обучение как отдельный процесс.

Как показано на рис. 6.1, на данный момент данные были загружены, проверены и предварительно обработаны. Это гарантирует, что все данные, необходимые для модели, присутствуют и что они воспроизводимо преобразованы в признаки, необходимые модели. Все эти шаги необходимы для бесперебойной работы конвейера. Мы хотим, чтобы обучение проходило гладко, потому что часто это наиболее трудоемкая часть всего процесса.

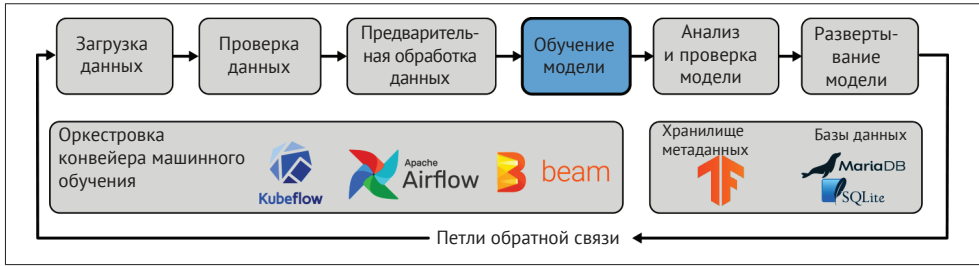


Рис. 6.1. Обучение модели как часть конвейеров машинного обучения

Одна очень важная особенность обучения модели в конвейере TFX заключается в том, что шаги предварительной обработки данных, которые мы обсуждали в главе 5, сохраняются вместе с весами обученной модели. Это окажется полезным после развертывания нашей модели в производственной среде, потому что означает, что этапы предварительной обработки всегда выдают те признаки, которые ожидает модель. Без этой возможности нам пришлось бы обновлять этапы предварительной обработки данных без обновления модели, и тогда модель перестала бы работать или прогнозы были бы основаны на неверных данных. Поскольку мы экспортируем шаги предварительной обработки и модель как один граф, то устраняем этот потенциальный источник ошибок.

В следующих двух разделах мы подробно рассмотрим шаги, необходимые для обучения модели `tf.keras` как части конвейера TFX¹.

ОПРЕДЕЛЕНИЕ МОДЕЛИ ДЛЯ НАШЕГО ДЕМОНСТРАЦИОННОГО ПРОЕКТА

Несмотря на то что архитектура модели уже определена, здесь потребуется дополнительный код. Нам нужно сделать возможным автоматизацию части конвейера обучения модели. В этом разделе мы кратко опишем модель, которую используем в данной главе.

Модель для нашего примера проекта является гипотетической реализацией, и мы, вероятно, могли бы оптимизировать архитектуру модели. Однако он демонстрирует некоторые общие компоненты многих моделей глубокого обучения:

- перенос обучения из предварительно обученной модели;
- плотные слои;
- слои конкатенации.

Как мы обсуждали в главе 1, набор данных для нашей модели представляет собой жалобы потребителей на финансовые продукты в США и содержит комбинацию структурированных данных (категориальных/числовых данных) и неструктурированных данных (текста). Источником данных является

¹ В нашем демонстрационном проекте мы используем модель Keras, но TFX также отлично работает с моделью Estimator. Примеры можно найти в документации TFX, доступной по ссылке <https://www.tensorflow.org/tfx/tutorials/tfx/components>.

Бюро по защите прав потребителей. Задача машинного обучения заключается в том, чтобы, основываясь на данных о жалобе, предсказать, будет ли жалоба оспорена потребителем. Признаки нашей модели включают финансовый продукт, ответ компании, штат США и описание жалоб потребителей. Наша модель основана на архитектуре широкой и глубокой модели (Wide and Deep model architecture, см. публикацию <https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html>) с добавлением модуля Universal Sentence Encoder (USE) (см. <https://arxiv.org/abs/1803.11175>), версия которого опубликована на TensorFlow Hub (см. <https://tfhub.dev/google/universal-sentence-encoder/4>), для кодирования признака, содержащего произвольный текст (описание жалобы потребителей).

Визуальное представление архитектуры нашей модели представлено на рис. 6.2, где текстовая функция (narrative_xf) идет по «глубокому» маршруту, а другие функции – по «широкому» маршруту.

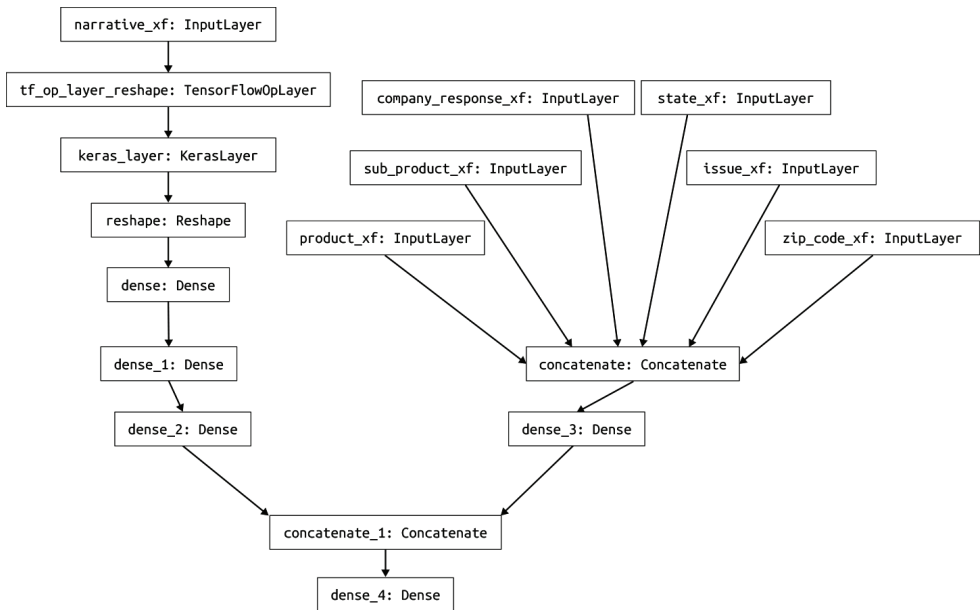


Рис. 6.2. Архитектура модели для нашего демонстрационного проекта

В примере 6.1 показано полное определение архитектуры модели. Поскольку мы хотим экспортировать модель совместно с шагами предварительной обработки, нам необходимо гарантировать, что входные имена модели соответствуют именам преобразованных признаков на выходе из `preprocessing_fn()`, о чем мы подробно говорили в главе 5. В нашем примере модели мы повторно используем функцию `transformed_name()`, показанную в главе 5, чтобы добавить суффикс `_xf` к нашим признакам.

Пример 6.1. Определение архитектуры модели

```
import tensorflow as tf import tensorflow_hub as hub

def transformed_name(key):
    return key + '_xf'
def get_model():

    # Однофакторные категориальные признаки
    input_features = []
    for key, dim in ONE_HOT_FEATURES.items(): ❶
        input_features.append(
            tf.keras.Input(shape=(dim + 1,),
                            name=transformed_name(key)))

    # Добавление конвейеризованных признаков
    for key, dim in BUCKET_FEATURES.items():
        input_features.append(
            tf.keras.Input(shape=(dim + 1,),
                            name=transformed_name(key)))

    # Добавление текстовых входных признаков
    input_texts = []
    for key in TEXT_FEATURES.keys(): input_texts.append(
        tf.keras.Input(shape=(1,),
                        name=transformed_name(key),
                        dtype=tf.string))

    inputs = input_features + input_texts

    # Встраивание текстовых признаков
    MODULE_URL = "https://tfhub.dev/google/universal-sentence-encoder/4"
    embed = hub.KerasLayer(MODULE_URL) ❷
    reshaped_narrative = tf.reshape(input_texts[0], [-1]) ❸
    embed_narrative = embed(reshaped_narrative)
    deep_ff = tf.keras.layers.Reshape((512, ), input_shape=(1, 512))(embed_narrative)

    deep = tf.keras.layers.Dense(256, activation='relu')(deep_ff)
    deep = tf.keras.layers.Dense(64, activation='relu')(deep)
    deep = tf.keras.layers.Dense(16, activation='relu')(deep)

    wide_ff = tf.keras.layers.concatenate(input_features)
    wide = tf.keras.layers.Dense(16, activation='relu')(wide_ff)
    both = tf.keras.layers.concatenate([deep, wide])

    output = tf.keras.layers.Dense(1, activation='sigmoid')(both)
    keras_model = tf.keras.models.Model(inputs, output) ❹

    keras_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                        loss='binary_crossentropy',
                        metrics=[
                            tf.keras.metrics.BinaryAccuracy(),
                            tf.keras.metrics.TruePositives()
                        ])

    return keras_model
```

- ❶ Проход по всем элементам и создание входных данных для каждого признака
- ❷ Загрузка модуля `tf.nn` модели кодировщика Universal Sentence Encoder
- ❸ Входные данные Keras двумерны, но кодировщик ожидает одномерные входные данные
- ❹ Сборка графа модели с функциональным API

Теперь, когда мы определили нашу модель, перейдем к описанию процесса ее интеграции в конвейер TFX.

Компонент TFX Trainer

Компонент TFX Trainer обрабатывает этап обучения в нашем конвейере. В этом разделе мы сначала опишем, как обучить модель Keras на демонстрационном проекте в одноразовом обучающем прогоне. В конце раздела мы добавим некоторые соображения для других обучающих ситуаций и для моделей Estimator.

Все шаги, которые мы опишем, могут показаться длинными и ненужными по сравнению с обычным обучающим кодом Keras. Однако ключевым моментом здесь является то, что компонент Trainer создаст модель, которая будет запущена в производство, где он преобразует новые данные и будет использовать модель для прогнозирования. Поскольку шаги преобразования (компонент Transform) включены в эту модель, шаги предварительной обработки данных всегда будут соответствовать ожиданиям модели. Это устраняет серьезный потенциальный источник ошибок при развертывании нашей модели.

В нашем демонстрационном проекте для компонента Trainer будут использоваться следующие входные данные:

- ранее сгенерированная схема данных, созданная на этапе проверки данных, описанном в главе 4;
- преобразованные данные и их граф предварительной обработки, о которых рассказывалось в главе 5;
- параметры обучения (например, количество шагов обучения);
- файл модуля, содержащий функцию `run_fn()`, которая определяет процесс обучения.

В следующем разделе мы обсудим настройку функции `run_fn`. Мы также расскажем, как обучить модель машинного обучения в нашем конвейере и экспортировать ее на следующий этап конвейера, о котором будет рассказываться в главе 7.

Функция `run_fn()`

Компонент Trainer будет искать функцию `run_fn()` в нашем файле модуля и использовать эту функцию как точку входа для выполнения процесса обучения. Файл модуля должен быть доступен для компонента Trainer. Если вы запускаете компонент в интерактивном контексте, вы можете просто определить абсолютный путь к файлу модуля и передать его компоненту. Если вы запускаете конвейеры в производственной среде, обратитесь к главе 11 или 12, чтобы узнать, как использовать файл модуля в этом случае.

Функция `run_fn()` – это общая точка входа в этапы обучения, а не специфичная функция `tf.keras`. Она выполняет следующие шаги:

- загрузку обучающих и контрольных данных (или генератор данных);
- определение архитектуры модели и компиляцию модели;
- обучение модели;
- экспорт модели для оценки на следующем этапе конвейера.

В примере 6.2 показано, как `run_fn` выполняет эти четыре шага для нашего демонстрационного проекта.

Пример 6.2. Функция `run_fn()` для нашего демонстрационного конвейера

`def run_fn(fn_args):`

```
tf_transform_output = tft.TFTransformOutput(fn_args.transform_output)
train_dataset = input_fn(fn_args.train_files, tf_transform_output) ❶
eval_dataset = input_fn(fn_args.eval_files, tf_transform_output)

model = get_model() ❷
model.fit(
    train_dataset,
    steps_per_epoch=fn_args.train_steps,
    validation_data=eval_dataset,
    validation_steps=fn_args.eval_steps) ❸

signatures = {
    'serving_default':
        _get_serve_tf_examples_fn(
            model,
            tf_transform_output).get_concrete_function(
                tf.TensorSpec(
                    shape=[None],
                    dtype=tf.string,
                    name='examples')
        )
} ❹
model.save(fn_args.serving_model_dir,
           save_format='tf', signatures=signatures)
```

- ❶ Вызов `input_fn`, чтобы получить генераторы данных
- ❷ Вызов функции `get_model`, чтобы получить скомпилированную модель `Keras`
- ❸ Обучение модели с использованием количества шагов обучения и оценок, пройденных компонентом `Trainer`
- ❹ Определение сигнатуры модели включающей функцию обслуживания, которую мы опишем далее

Эта функция довольно универсальна и может быть повторно использована с любой другой моделью `tf.keras`. Специфичные для проекта детали определены во вспомогательных функциях, таких как `get_model()` или `input_fn()`.

В следующих разделах мы хотим более подробно рассмотреть то, как мы загружаем данные, обучаем и экспортируем нашу модель машинного обучения внутри функции `run_fn()`.

Загрузка данных

Следующие строки в `run_fn` загружают наши обучающие и контрольные данные:

```
def run_fn(fn_args):
    tf_transform_output = tft.TFTransformOutput(fn_args.transform_output)
    train_dataset = input_fn(fn_args.train_files, tf_transform_output)
    eval_dataset = input_fn(fn_args.eval_files, tf_transform_output)
```

В первой строке функция `run_fn` получает набор аргументов, включая граф преобразования, примеры наборов данных и параметры обучения, используя объект `fn_args`.

Загрузка данных для обучения и проверки модели выполняется в пакетном режиме, и загрузка обрабатывается функцией `input_fn()`, как показано в примере 6.3.

Пример 6.3. Функция `input_fn` нашего демонстрационного конвейера

```
LABEL_KEY = 'labels'

def _gzip_reader_fn(filename):
    return tf.data.TFRecordDataset(filename,
                                     compression_type='GZIP')

def input_fn(file_pattern,
             tf_transform_output, batch_size=32):

    transformed_feature_spec = (
        tf_transform_output.transformed_feature_spec().copy())

    dataset = tf.data.experimental.make_batched_features_dataset(
        file_pattern=file_pattern,
        batch_size=batch_size, features=transformed_feature_spec,
        reader=_gzip_reader_fn,
        label_key=transformed_name(LABEL_KEY)) ❶

    return dataset
```

❶ Набор данных будет размещен в пакете правильного размера

Функция `input_fn` позволяет нам загружать сжатые, предварительно обработанные наборы данных, которые были сгенерированы на предыдущем шаге преобразования¹. Для этого нам нужно передать функцию `tf_transform_output`. В результате мы получаем схему данных для загрузки набора данных из структур данных `TFRecord`, созданных компонентом `Transform`. Используя предварительно обработанные наборы данных, мы можем избежать предварительной обработки данных во время обучения и ускорить процесс обучения.

¹ Компонент `Trainer` можно использовать без предыдущего компонента `Transform`, и мы можем загружать необработанные наборы данных. Однако в этом случае мы не сможем использовать отличную функцию `TFX`, которая экспортирует графы предварительной обработки и моделей в единый граф `SavedModel`.

Функция `input_fn` возвращает генератор (`batched_features_dataset`), который будет предоставлять данные модели по одному пакету за раз.

Компиляция и обучение модели

Теперь, когда мы определили шаги загрузки данных, следующим шагом будет определение архитектуры нашей модели и компиляция нашей модели. В нашем случае `run_fn` для этого потребует вызов `get_model()`, который мы описали, поэтому для этого нужна всего одна строка кода:

```
model = get_model()
```

Затем мы обучаем нашу скомпилированную модель `tf.Keras` с помощью метода `Keras fit()`:

```
model.fit(
    train_dataset,
    steps_per_epoch=fn_args.train_steps,
    validation_data=eval_dataset,
    validation_steps=fn_args.eval_steps)
```



Итерации обучения в сравнении с эпохами

Компонент TFX Trainer определяет процесс обучения по количеству итераций обучения, а не эпох. Итерация обучения – это обучение модели на одном пакете данных. Преимущество подхода, когда используются итерации, а не эпохи, заключается в том, что мы можем обучать или проверять модели с большими наборами данных и использовать только часть данных. В то же время, если вы хотите многократно перебирать обучающий набор данных во время обучения, вы можете увеличить размер итерации до кратного числа доступных выборок.

После завершения обучения модели следующим шагом будет экспорт обученной модели. Мы подробно обсудим экспорт моделей для развертывания в главе 8. В следующем разделе мы хотим показать, каким образом шаги предварительной обработки можно экспортировать вместе с моделью.

Экспорт модели

Наконец, мы экспортируем модель. Мы объединяем шаги предварительной обработки из предыдущего компонента конвейера с обученной моделью и сохраняем модель в формате *SavedModel* – внутреннем формате TensorFlow. Мы определяем *сигнатуру модели* на основе графа, созданного функцией, приведенной в примере 6.4. Мы рассмотрим сигнатуры моделей более подробно в разделе «Сигнатуры моделей» в главе 8.

В функции `run_fn` мы определяем подпись модели и сохраняем модель со следующим кодом:

```

signatures = {
    'serving_default':
        _get_serve_tf_examples_fn(
            model,
            tf_transform_output).get_concrete_function(
                tf.TensorSpec(
                    shape=[None],
                    dtype=tf.string,
                    name='examples')
            )
}
model.save(fn_args.serving_model_dir,
          save_format='tf', signatures=signatures)

```

Функция `run_fn` экспортирует `get_serve_tf_examples_fn` как часть сигнатуры модели. Когда модель была экспортирована и развернута, каждый запрос прогнозирования будет проходить через `serve_tf_examples_fn()`, как показано в примере 6.4. С каждым запросом мы анализируем сериализованные элементы `tf.Example` и применяем шаги предварительной обработки к необработанным данным запроса. Затем модель делает прогноз на основе предварительно обработанных данных.

Пример 6.4. Применение графа предварительной обработки к входным данным модели

```

def get_serve_tf_examples_fn(model, tf_transform_output):
    model.tft_layer = tf_transform_output.transform_features_layer() ❶
    @tf.function
    def serve_tf_examples_fn(serialized_tf_examples):
        feature_spec = tf_transform_output.raw_feature_spec()
        feature_spec.pop(LABEL_KEY)
        parsed_features = tf.io.parse_example(
            serialized_tf_examples, feature_spec) ❷

        transformed_features = model.tft_layer(parsed_features) ❸
        outputs = model(transformed_features) ❹
        return {'outputs': outputs}

    return serve_tf_examples_fn

```

- ❶ Загрузка графа предварительной обработки
- ❷ Разбор необработанных записей `tf.Example` из запроса.
- ❸ Применение преобразования предварительной обработки к необработанным данным
- ❹ Выполнение прогноза с предварительно обработанными данными

После того как мы определили нашу функцию `run_fn()`, давайте обсудим, как запустить компонент `Trainer`.

Запуск компонента `Trainer`

Как показано в примере 6.5, компонент `Trainer` принимает в качестве входных данных следующие элементы:

- файл модуля Python, здесь сохраненный как `module.py`, содержащий `run_fn()`, `input_fn()`, `get_serve_tf_examples_fn()` и другие связанные функции, которые мы обсуждали ранее;
- преобразованные примеры, созданные компонентом `Transform`;
- граф преобразования, созданный компонентом `Transform`;
- схему, созданную компонентом проверки данных;
- число этапов обучения и оценки.

Пример 6.5. Компонент Trainer

```
from tfx.components import Trainer
from tfx.components.base import executor_spec
from tfx.components.trainer.executor import GenericExecutor ❶
from tfx.proto import trainer_pb2

TRAINING_STEPS = 1000
EVALUATION_STEPS = 100

trainer = Trainer(
    module_file=os.path.abspath("module.py"),
    custom_executor_spec=executor_spec.ExecutorClassSpec(GenericExecutor), ❷
    transformed_examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    schema=schema_gen.outputs['schema'],
    train_args=trainer_pb2.TrainArgs(num_steps=TRAINING_STEPS),
    eval_args=trainer_pb2.EvalArgs(num_steps=EVALUATION_STEPS))
```

- ❶ Загрузка `GenericExecutor` с целью переопределить управляющую программу (`executor`) компонента `Trainer`
- ❷ Переопределение управляющей программы (`executor`) для загрузки функции `run_fn()`

В среде `notebook` (интерактивный контекст) мы можем запустить компонент `Trainer`, как и любой предыдущий компонент, с помощью следующей команды:

```
context.run(trainer)
```

После завершения обучения и экспорта модели компонент регистрирует путь к экспортированной модели в хранилище метаданных. Последующие компоненты могут выбрать модель для ее проверки.

Компонент `Trainer` является общим, и его функция не ограничивается запуском моделей `TensorFlow`. Однако компоненты конвейера, работающие на более поздних этапах, ожидают, что модель будет сохранена в формате `TensorFlow SavedModel` (об этом формате более подробно рассказывается в документации, опубликованной по ссылке https://www.tensorflow.org/api_docs/python/tf/saved_model/save). Граф `SavedModel` включает граф преобразования, поэтому этапы предварительной обработки данных являются частью модели.



Переопределение модуля исполнения (Executor) компонента Trainer

В нашем демонстрационном проекте мы переопределяем модуль исполнения (Executor) компонента Trainer, чтобы задействовать функцию `run_fn()` общей точки входа в обучение вместо функции по умолчанию `trainer_fn()`, которая поддерживает только модели `tf.Estimator`. В главе 12 мы познакомим вас с другим модулем исполнения Trainer, `ai_platform_trainer_executor.GenericExecutor`. Этот модуль исполнения позволяет вам обучать модели на платформе искусственного интеллекта Google Cloud, Google Cloud's AI Platform, а не внутри вашего конвейера. Это подходящий вариант, если для обучения вашей модели требуется специальное оборудование (например, графические процессоры или блоки тензорной обработки [TPU]), которые недоступны в вашей среде конвейера.

Другие соображения относительно компонента Trainer

В наших примерах до сих пор в этой главе мы рассматривали только один обучающий прогон модели Keras. Но мы также можем использовать компонент Trainer для точной настройки модели на основе предыдущего прогона или для обучения нескольких моделей одновременно (об этом будет рассказываться в разделе «Расширенные концепции конвейеров машинного обучения» главы 10). Мы также можем использовать этот компонент для оптимизации модели с помощью поиска гиперпараметров, и мы обсудим это подробнее в разделе «Настройка модели» данной главы.

В этом разделе мы также обсудим, как использовать компонент Trainer с моделью Estimator и как загрузить вашу SavedModel, экспортированную компонентом Trainer, вне конвейера TFX.

Использование компонента Trainer с моделью Estimator

До недавнего времени TFX поддерживал только модели `tf.Estimator`, а компонент Trainer был разработан исключительно для модулей Estimator. Реализация компонента Trainer по умолчанию использует функцию `trainer_fn()` как точку входа в процесс обучения, но эта точка входа использует специфичные для `tf.Estimator` способы работы с моделью. Компонент Trainer ожидает, что входные данные модуля оценки (Estimator) будут определены при помощи таких функций, как `train_input_fn()`, `eval_input_fn()` и `serve_receiver_fn()`¹.

Как мы обсуждали в разделе «Запуск компонента Trainer», основная функциональность компонента может быть заменена универсальным модулем запуска обучения `GenericExecutor`, который использует функцию `run_fn()` как точку входа в процесс обучения². Как следует из названия модуля, про-

¹ Модели `tf.Keras` можно преобразовать в модели `tf.Estimator` с помощью преобразования `tf.model_to_estimator()`. Однако в связи с недавними обновлениями TFX такой способ больше не рекомендуется применять.

² Более подробно об этапах разработки и повторном использовании модулей исполнения компонентов рассказывается в разделе «Повторное использование существующих компонентов» главы 10.

цесс обучения становится универсальным и не привязанным к моделям `tf.Estimator` или `tf.Keras`.

Использование SavedModel вне конвейера

Если мы хотим изучить экспортированную модель `SavedModel` вне конвейера TFX, мы можем загрузить модель как *определенную функцию*¹, которая представляет собой граф одной сигнатуры:

```
model_path = trainer.outputs.model.get()[0].uri
model = tf.saved_model.load(export_dir=model_path)
predict_fn = model.signatures["serving_default"]
```

Теперь, когда модель загружена как определенная функция, мы можем строить прогнозы. Экспортированная модель ожидает, что входные данные будут предоставлены в виде структуры данных `tf.Example`, как показано в следующем примере. Более подробную информацию о структуре данных `tf.Example` и о том, как можно преобразовать другие признаки (например, целые числа и числа с плавающей запятой), можно найти в примере 3.1. В приведенном ниже примере кода показано, как создать сериализованную структуру данных и выполнить прогнозирование модели путем вызова функции `prediction_fn()`:

```
example = tf.train.Example(features=tf.train.Features(feature={
    'feature_A': _bytes_feature(feature_A_value),
    ...
})) ❶

serialized_example = example.SerializeToString()
print(predict_fn(tf.constant([serialized_example])))
```

❶ Вспомогательная функция `_bytes_feature` определена в примере 3-1

Если вы хотите визуально наблюдать за прогрессом процесса обучения модели, то можете сделать это с помощью инструмента TensorBoard. В следующем разделе мы расскажем, как использовать TensorBoard в нашем конвейере.

ИСПОЛЬЗОВАНИЕ TENSORBOARD В ИНТЕРАКТИВНОМ КОНВЕЙЕРЕ

TensorBoard – еще один замечательный инструмент, который является частью экосистемы TensorFlow. В него включено множество полезных функций, которые мы можем использовать в наших конвейерах, например мониторинг показателей во время обучения, визуализация встраивания слов в задачах НЛП или просмотр активаций слоев в модели. Новая функция Profiler (более подробно об этой функции рассказывается в документации, размещенной по ссылке https://www.tensorflow.org/tensorboard/tensorboard_profiling_keras) позволяет нам выполнять профилирование модели, чтобы выявить ограничения производительности.

Пример стандартной визуализации TensorBoard показан на рис. 6.3.

¹ Дополнительные сведения об определенных функциях см. в документации TensorFlow (<https://www.tensorflow.org/guide/function>).

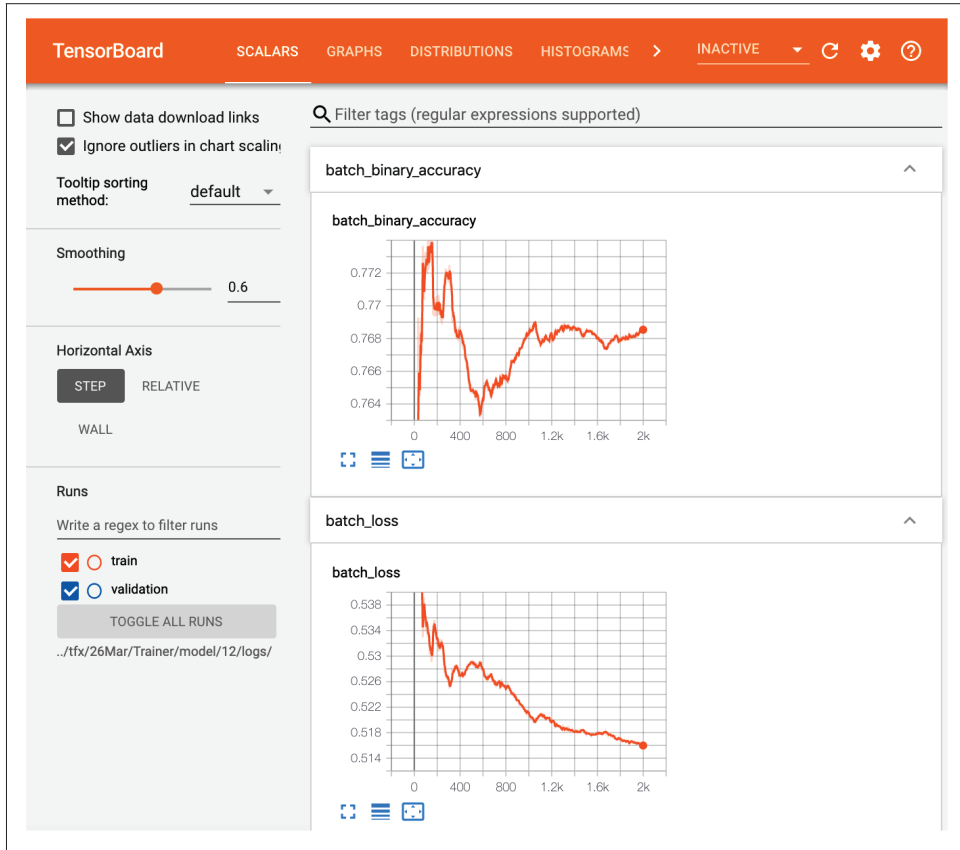


Рис. 6.3. Просмотр метрик в TensorBoard во время обучения

Чтобы иметь возможность использовать TensorBoard в нашем конвейере, нам нужно добавить обратные вызовы в функции `run_fn` и разместить файл журнала обучения в указанной нами папке:

```
log_dir = os.path.join(os.path.dirname(fn_args.serving_model_dir), 'logs')
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir, update_freq='batch')
```

Нам также нужно добавить обратный вызов для нашего обучения модели:

```
model.fit(
    train_dataset,
    steps_per_epoch=fn_args.train_steps,
    validation_data=eval_dataset,
    validation_steps=fn_args.eval_steps,
    callbacks=[tensorboard_callback])
```

Затем, чтобы просмотреть TensorBoard в интерактивной среде notebook, мы получаем расположение журналов обучения модели и передаем его в TensorBoard:

```
model_dir = trainer.outputs['output'].get()[0].uri
```

```
%load_ext tensorboard
%tensorboard --logdir {model_dir}
```

Мы также можем использовать TensorBoard вне интерактивной среды notebook, выполнив следующую команду:

```
tensorboard --logdir path/to/logs
```

Затем необходимо подключиться к <http://localhost:6006/>, чтобы просмотреть содержимое TensorBoard. Этот вариант предоставляет нам больше информации для просмотра.

Далее мы представим несколько полезных стратегий для обучения больших моделей на нескольких графических процессорах.

СТРАТЕГИИ РАСПРЕДЕЛЕНИЯ

TensorFlow предоставляет стратегии распределения для моделей машинного обучения, которые невозможно должным образом обучить на одном графическом процессоре. Возможно, вы захотите углубиться в тему выбора стратегий распределения, если хотите ускорить обучение, или в том случае, когда вы не можете разместить всю модель на одном графическом процессоре.

Описываемые здесь стратегии представляют собой абстракции для распределения параметров модели между несколькими графическими процессорами или даже несколькими серверами. В целом есть две группы стратегий: *синхронное* и *асинхронное обучение*. Для синхронных стратегий характерна синхронная работа всех рабочих процессов обучения (при этом каждый из таких процессов работает с разными срезами обучающих данных), а затем выполняется агрегация градиентов от всех рабочих процессов перед обновлением модели. Асинхронные стратегии обучают модели независимо, используя полный набор данных, в разных рабочих процессах. Каждый рабочий процесс обновляет градиенты модели асинхронно, не дожидаясь завершения работы других рабочих процессов. Как правило, синхронные стратегии координируются посредством операций *all-reduce*¹, а асинхронные стратегии – через архитектуру сервера параметров.

Существует несколько синхронных и асинхронных стратегий, и у каждой из них есть свои преимущества и недостатки. На момент написания этой главы Keras поддерживает следующие стратегии.

Стратегия *MirroredStrategy*

Эта стратегия актуальна для нескольких графических процессоров в одном экземпляре и использует синхронный шаблон обучения. Стратегия *зеркалирует* (*mirror*) модель и параметры для всех рабочих процессов, но каждый рабочий процесс получает свой пакет данных. *MirroredStrategy* – хорошая стратегия по умолчанию, если вы обучаете модель машинного обучения

¹ Операция *all-reduce* приводит информацию от всех рабочих процессов к единому виду; другими словами, она обеспечивает синхронизацию между всеми рабочими процессами обучения.

на одном узле с несколькими графическими процессорами и ваша модель машинного обучения умещается в памяти графического процессора.

Стратегия CentralStorageStrategy

В отличие от MirroredStrategy, переменные в этой стратегии не зеркалируются на всех графических процессорах. Вместо этого они сохраняются в памяти ЦП, а затем копируются в назначенный графический процессор для выполнения соответствующих операций. В случае одной операции с графическим процессором CentralStorageStrategy будет хранить переменные в графическом процессоре, а не в ЦПУ. CentralStorageStrategy – хорошая стратегия для распределения вашего процесса обучения, когда вы выполняете обучение на одном узле с несколькими графическими процессорами и ваша полная модель не помещается в памяти одного графического процессора, или когда скорость передачи данных между графическими процессорами слишком ограничена.

Стратегия MultiWorkerMirroredStrategy

Эта стратегия основывается на шаблоне MirroredStrategy, но копирует переменные через несколько рабочих процессов (например, экземпляры вычислений). Стратегия MultiWorkerMirroredStrategy – подходящий вариант, если одного узла недостаточно для обучения вашей модели.

Стратегия TPUStrategy

Эта стратегия позволяет использовать TPU в Google Cloud. Она следует схеме синхронного обучения и в основном работает аналогично MirroredStrategy, за исключением того, что использует TPU вместо графических процессоров. Для этого требуется собственная стратегия, поскольку MirroredStrategy использует специфичные для графического процессора функции all-reduce. TPU имеют огромный объем доступной оперативной памяти, а обмен данными между TPU сильно оптимизирован, поэтому в стратегии TPU используется зеркальный подход.

Стратегия ParameterServerStrategy

ParameterServerStrategy использует несколько узлов в качестве центрального хранилища переменных. Эта стратегия полезна для моделей, превышающих доступные ресурсы (например, объем оперативной памяти или производительности ввода-вывода) одного узла. ParameterServerStrategy – ваш единственный вариант, если вы не можете выполнять обучение на одном узле, а модель превышает ограничения памяти ОЗУ или производительности ввода-вывода узла.

Стратегия OneDeviceStrategy

Весь смысл OneDeviceStrategy состоит в том, чтобы протестировать всю конфигурацию модели перед тем, как приступить к настоящему распределенному обучению. Эта стратегия использует единственное устройство (например, один графический процессор) для обучения модели. Как только будет подтверждено, что конфигурация работает для обучения, стратегия может быть заменена.



Не все стратегии могут использоваться совместно с компонентом TFX Trainer

На момент написания этого раздела компонент TFX Trainer поддерживает только MirroredStrategy. Хотя в настоящее время с `tf.keras` можно использовать различные стратегии, согласно дорожной карте TFX они будут доступны для использования совместно с компонентом Trainer во второй половине 2020 года.

Поскольку MirroredStrategy поддерживается TFX Trainer, мы покажем здесь пример использования этого компонента для реализации данной стратегии. Мы можем легко применить MirroredStrategy, добавив несколько строк перед вызовом, создающим нашу модель, и последующим вызовом `model.compile()`:

```
mirrored_strategy = tf.distribute.MirroredStrategy() ❶
with mirrored_strategy.scope(): ❷
    model = get_model()
```

- ❶ Экземпляр стратегии распределения
- ❷ Создание и компиляция оболочки модели с помощью менеджера Python

В этом примере настройки мы создаем экземпляр MirroredStrategy. Чтобы применить стратегию распределения к нашей модели, мы завершаем создание и компиляцию модели с помощью менеджера Python (в нашем случае все это происходит внутри функции `get_model()`). Это создаст и скомпилирует нашу модель в выбранной нами области распределения. MirroredStrategy будет использовать все доступные графические процессоры экземпляра. Если вы хотите уменьшить количество используемых экземпляров графического процессора (например, в случае их совместного использования), то можете указать графические процессоры, которые будут использоваться с MirroredStrategy, изменив создание стратегии распределения:

```
mirrored_strategy = tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])
```

В этом примере мы выбираем два графических процессора, которые будут использоваться для наших тренировочных прогонов.



Требования к размеру пакета при использовании MirroredStrategy

MirroredStrategy требует, чтобы размер пакета был пропорционален количеству устройств. Например, если вы выполняете обучение с использованием пяти графических процессоров, размер пакета должен быть кратным количеству графических процессоров. Помните об этом при настройке функции `input_fn()`, как показано в примере 6.3.

Стратегии распределения, с которыми мы познакомились, полезны для больших задач машинного обучения, которые не умещаются в памяти одного графического процессора. Эти стратегии часто применяются при настройке модели, которую мы обсудим в следующем разделе.

Настройка модели

Настройка гиперпараметров – важная часть создания точной модели машинного обучения. В зависимости от варианта использования эта настройка может выполняться во время наших первоначальных экспериментов, или же она может использоваться в качестве составного элемента конвейера машинного обучения. Этот раздел не содержит подробных и полных начальных инструкций по настройке модели, мы приведем здесь лишь ее краткий обзор и опишем, как ее можно включить в конвейер.

Стратегии настройки гиперпараметров

В зависимости от типа модели, которую вы используете в вашем конвейере, выбор гиперпараметров будет разным. Если модель представляет собой глубокую нейронную сеть, настройка гиперпараметров особенно важна для достижения хорошей производительности нейронных сетей. Два наиболее важных набора гиперпараметров для настройки – это те параметры, которые контролируют оптимизацию и сетевую архитектуру.

Для оптимизации мы рекомендуем по умолчанию использовать оптимизаторы Adam (см. документацию по ссылке https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam) или NAdam (см. документацию по ссылке https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Nadam). Скорость обучения – очень важный параметр для экспериментов, и существует множество вариантов планировщиков скорости обучения (более подробно об этом см. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules). Мы рекомендуем использовать самый большой размер пакета, который помещается в память вашего графического процессора.

Для очень больших моделей мы рекомендуем следующие шаги:

- настройте начальную скорость обучения, начиная с 0,1;
- выберите количество шагов для тренировки (столько, сколько позволяет терпение);
- линейно уменьшите скорость обучения до 0 за указанное количество шагов.

Для небольших моделей мы рекомендуем использовать раннюю остановку (более подробно об этом см. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping), чтобы избежать переобучения. С помощью этого метода обучение модели останавливается, когда функция потерь проверки не улучшается после определенного пользователем количества эпох.

Для сетевой архитектуры два наиболее важных параметра, которые необходимо настроить, – это размер и количество слоев. Их увеличение повысит эффективность обучения, но может привести к переобучению и означает, что обучение модели займет больше времени. Вы также можете рассмотреть возможность добавления остаточных связей между слоями, особенно для глубоких архитектур.

Самыми популярными подходами к поиску гиперпараметров являются *поиск по сетке* (grid search) и *случайный поиск* (random search). При поиске по сетке полностью проверяется каждая комбинация параметров, тогда как при случайном поиске параметры выбираются из доступных вариантов и могут

включать не все возможные комбинации. Поиск по сетке может занять очень много времени, если количество возможных гиперпараметров велико. Попробовав диапазон значений параметров настроек, вы можете выполнить точную настройку, взяв самые эффективные гиперпараметры за основу и запустив новый поиск.

В экосистеме TensorFlow настройка гиперпараметров реализуется с помощью Keras Tuner (см. документацию, размещенную по ссылке <https://keras-team.github.io/keras-tuner/>), а также Katib (см. ссылку на репозиторий: <https://github.com/kubeflow/katib>), который обеспечивает настройку гиперпараметров в Kubeflow. В дополнение к поиску по сетке и случайному поиску оба этих пакета поддерживают байесовский поиск и алгоритм Hyperband (см. статью, размещенную по ссылке <https://arxiv.org/pdf/1603.06560.pdf>).

Настройка гиперпараметров в конвейерах TFX

В конвейере TFX настройка гиперпараметров принимает данные из компонента Transform и обучает различные модели для определения наилучших гиперпараметров. Затем гиперпараметры передаются компоненту Trainer, который обучает окончательную модель, используя их.

В этом случае функция, определяющая модель (в нашем примере – функция `get_model`), должна принять гиперпараметры в качестве входных данных и построить модель в соответствии с указанными гиперпараметрами. Так, например, в качестве входного аргумента необходимо определить количество слоев.



Компонент TFX Tuner

Компонент TFX Tuner был выпущен в то время, когда мы завершали работу над этой книгой. Вы можете просмотреть его исходный код в репозитории проекта на GitHub (см. <https://github.com/tensorflow/tfx/tree/master/tfx/components/tuner>).

РЕЗЮМЕ

В этой главе мы показали, как перенести обучение нашей модели из автономного сценария в интегрированную часть нашего конвейера. Это означает, что процесс можно автоматизировать и запускать, когда мы захотим, – как только новые данные поступят в конвейер или когда точность предыдущей модели окажется ниже заранее определенного уровня. Мы также рассказали, каким образом модель и шаги предварительной обработки данных сохраняются вместе, чтобы избежать ошибок из-за несоответствия между предварительной обработкой и обучением. Мы дополнительно рассмотрели стратегии распределения обучения модели и настройки гиперпараметров.

Теперь, когда у нас есть сохраненная модель, следующий шаг – во всех подробностях рассмотреть, что она может делать.

Глава 7

Анализ и проверка модели

На данном этапе конвейера машинного обучения мы проверили статистические параметры наших данных, преобразовали данные в соответствующие признаки и обучили нашу модель. Возможно, настало время приступить к промышленной эксплуатации нашей модели? По нашему мнению, перед тем как вы перейдете к развертыванию модели, необходимо сделать два дополнительных шага: выполнить углубленный анализ качества модели и проверить, что все выполненные нами этапы улучшают показатели работы нашей модели по сравнению с любой моделью, которая уже находится в производственной эксплуатации. На рис. 7.1 показано, как эти шаги реализуются в конвейере.

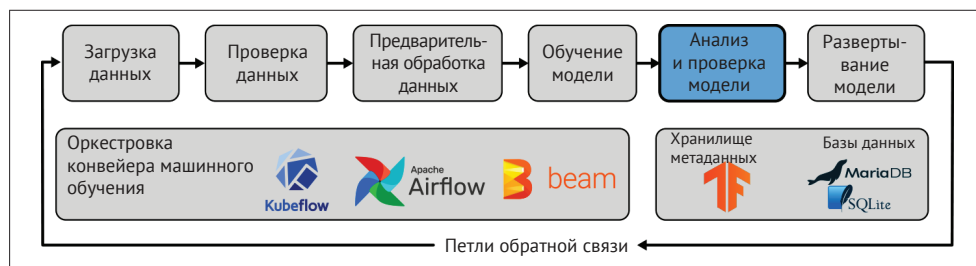


Рис. 7.1. Анализ и проверка модели как часть конвейеров машинного обучения

Пока мы обучаем модель, мы определяем ее качество и следим за ним на основании оценочного подмножества данных во время обучения, а также пробуем различные гиперпараметры, чтобы получить наилучшее качество. Но обычно во время обучения в качестве критерия оценки модели используется только один показатель, и часто этот показатель – точность.

Когда мы создаем конвейер машинного обучения, мы часто пытаемся ответить на сложный бизнес-вопрос или пытаемся смоделировать сложную реальную систему. Одной-единственной метрики часто недостаточно, чтобы определить, ответит ли наша модель на этот вопрос. Это особенно актуально в том случае, если наш набор данных несбалансирован или если некоторые выводы нашей модели более важны с точки зрения серьезности возможных последствий для пользователей, работающих с моделью, чем другие.

Кроме того, одна метрика, которая усредняет качество модели по всему набору оценок, может скрыть множество важных деталей. Если ваша модель рабо-

тает с данными о людях, получают ли все пользователи, работающие с моделью, однородные результаты? Ваша модель работает лучше для пользователей-женщин или для пользователей-мужчин? У пользователей из Японии результаты получаются хуже, чем у пользователей из США? Такие расхождения могут нанести ущерб коммерческой деятельности или причинить вред реальным людям. Если ваша модель распознает объекты и обнаруживает препятствия для автономного транспортного средства, работает ли она достаточно хорошо при любых условиях освещения? Используя одну метрику для всего обучающего набора, вы можете упустить важные пограничные случаи и ситуации, происходящие при превышении предельно допустимых параметров. Очень важно иметь возможность отслеживать показатели в разных срезах набора данных.

Также чрезвычайно важно отслеживать ваши показатели во времени – до развертывания, после развертывания и во время промышленной эксплуатации. Даже если ваша модель статична, данные, загружаемые в конвейер, со временем будут меняться, часто вызывая снижение качества прогнозов.

В этой главе мы продемонстрируем использование еще одного пакета из экосистемы TensorFlow: TensorFlow Model Analysis (TFMA), который обладает всеми этими возможностями. Мы покажем, как, используя его, можно получить детальные показатели качества своей модели, построить срез данных для получения показателей для разных групп и глубже исследовать объективность прогнозов модели с помощью показателей объективности (Fairness Indicators) и аналитического инструмента «Что, если» (What-If Tool). Затем мы расскажем, как можно выйти за рамки анализа и начать объяснять прогнозы, которые делает ваша модель.

Мы также опишем последний шаг перед развертыванием вашей новой модели: подтверждение того, что модель содержит улучшения по сравнению с любой предыдущей версией. Важно, чтобы любая новая модель, развернутая в промышленной среде, была лучше предыдущей модели, и, соответственно, другие сервисы, зависящие от этой модели, в свою очередь, также улучшались. Если новая модель не содержит никаких улучшений по сравнению с предыдущей, ее развертывание не стоит усилий.

КАК ПРОАНАЛИЗИРОВАТЬ МОДЕЛЬ

Наш процесс анализа модели начинается с выбора показателей. Как мы обсуждали ранее, их выбор чрезвычайно важен для успеха построения нашего конвейера машинного обучения. Хорошей практикой является выбор нескольких показателей, которые имеют значение для нашей бизнес-задачи, поскольку, если выбрать один показатель, можно упустить важные детали при анализе. В этом разделе мы рассмотрим некоторые из наиболее важных показателей для задач классификации и регрессии.

Метрики классификации

Чтобы вычислить значения множества метрик классификации, необходимо сначала подсчитать количество истинно положительных / ложноположительных примеров и истинно отрицательных / ложноотрицательных примеров в нашем оценочном наборе. В качестве примера возьмем любой класс из наших меток.

Истинно положительные результаты

Обучающие примеры, которые принадлежат к определенному классу и правильно помечены классификатором как принадлежащие к этому классу. Например, если истинное значение метки равно 1 и прогнозное значение метки также равно 1, то результат будет истинно положительным.

Ложноположительные результаты

Обучающие примеры, которые не принадлежат к определенному классу и неправильно помечены классификатором как принадлежащие к этому классу. Например, если истинное значение метки равно 0, а прогнозное значение метки равно 1, результат будет ложноположительным.

Истинно отрицательные результаты

Обучающие примеры, которые не принадлежат к определенному классу и правильно помечены классификатором как не принадлежащие к этому классу. Например, если истинное значение метки равно 0, а прогнозное значение метки равно 0, результат будет истинно отрицательным.

Ложноотрицательные результаты

Обучающие примеры, которые принадлежат к определенному классу и неправильно помечены классификатором как не принадлежащие к этому классу. Например, если истинное значение метки равно 1, а прогнозное значение метки равно 0, результат будет ложноотрицательным.

Все эти базовые показатели приведены в табл. 7.1.

Таблица 7-1. Матрица истинных и ложных результатов прогнозирования

	Результат прогнозирования: 1	Результат прогнозирования: 0
Истинное значение = 1	Число истинно положительных результатов	Число ложноотрицательных результатов
Истинное значение = 0	Число ложноположительных результатов	Число истинно отрицательных результатов

Если мы рассчитаем все эти метрики для модели из нашего демонстрационного проекта, то получим результаты, показанные на рис. 7.2.

	Predicted Yes		Predicted No		Total	
Actual Yes	1.1%	(11)	20.8%	(208)	21.9%	(219)
Actual No	0.7%	(7)	77.4%	(774)	78.1%	(781)
Total	1.8%	(18)	98.2%	(982)		

Рис. 7.2. Матрица истинных и ложных результатов прогнозирования для нашего демонстрационного проекта

Мы увидим, какую конкретную пользу приносят эти расчеты, когда будем говорить о достоверности модели далее в этой главе. Существует несколько других показателей для сравнения моделей, которые объединяют эти результаты расчетов в одно число.

Верность

Верность определяется как отношение «(число истинно положительных результатов + число истинно отрицательных результатов) / общее число примеров» и представляет собой долю примеров, которые были правильно классифицированы. Это подходящий показатель для набора данных, в котором положительный и отрицательный классы представлены в равных долях (набор данных сбалансирован в равных пропорциях), но он может ввести вас в заблуждение, если набор данных несбалансирован.

Точность

Точность определяется как отношение «число истинно положительных результатов / (число истинно отрицательных результатов + число ложноположительных результатов)» и представляет собой долю примеров, отнесенных в результате выполнения прогноза к положительному классу, которые были правильно классифицированы. Таким образом, если классификатор имеет высокую точность, большинство примеров, которые он в результате выполнения относит к положительному классу, действительно будут принадлежать к положительному классу.

Полнота

Полнота определяется как отношение «число истинно положительных результатов / (число истинно положительных результатов + ложноотрицательных результатов)» и представляет собой долю примеров, в которых истинное значение положительно и правильно идентифицировано классификатором. Таким образом, если классификатор имеет высокий уровень полноты, он правильно определит большинство примеров, действительно относящихся к положительному классу.

Другой способ получить некое обобщенное число, характеризующее качество модели, – это AUC (площадь под кривой). Под «кривой» здесь подразумевается ROC-кривая (receiver operating characteristic, рабочая характеристика приёмника), которая отображает соотношение частоты истинно положительных результатов (TPR, true positive rate) и частоты ложноположительных результатов (FPR, false positive rate).

Метрика TPR – это альтернативное название для *полноты*, которая определяется, как показано ниже:

$$\text{Частота истинно положительных результатов} = \frac{\text{число истинно положительных результатов}}{\text{число истинно положительных результатов} + \text{число ложноотрицательных результатов}}$$

Метрика FPR определяется следующим образом:

$$\text{Частота ложноположительных результатов} = \frac{\text{число ложноположительных результатов}}{\text{число ложноположительных результатов} + \text{число ложноотрицательных результатов}}$$

ROC-кривая строится путем вычисления TPR и FPR для всех порогов классификации. *Порог классификации* – это граница вероятности отнесения примеров к положительному или отрицательному классу (обычно значение этого параметра равно 0,5). На рис. 7.3 показаны ROC-кривая и AUC для нашего демонстрационного проекта. Для случайного предсказателя ROC-кривая будет прямой линией, выходящей из начала координат и входящей в точку с координатами [1,1]. По мере того как кривая ROC смещается все дальше от оси x в левый верхний угол графика, модель улучшается, а AUC увеличивается. AUC – еще одна полезная метрика, которую можно отобразить в TFMA.

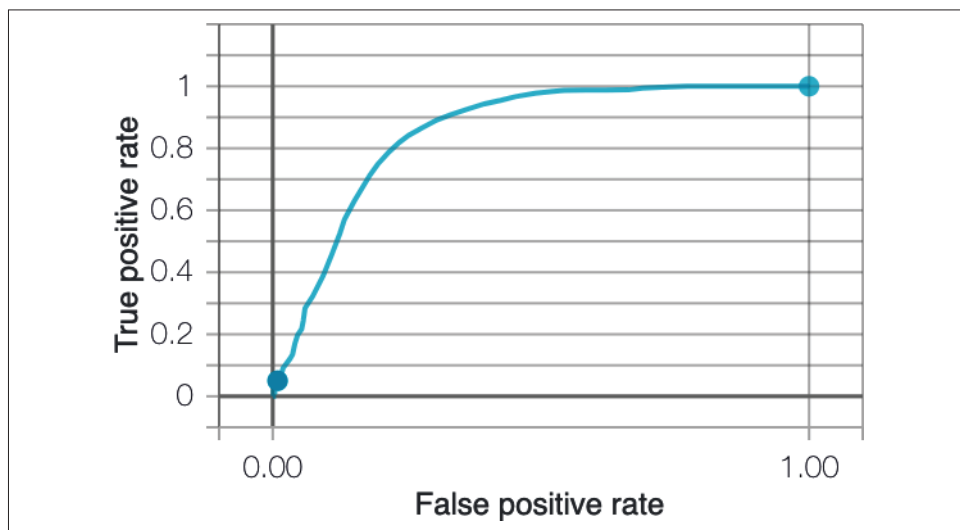


Рис. 7.3. ROC для нашего демонстрационного проекта

Метрики регрессии

В задаче регрессии модель предсказывает некоторое числовое значение для каждого обучающего примера, которое сравнивается с фактическим значением. Общие метрики регрессии, которые мы можем использовать в TFMA, включают:

Средняя абсолютная ошибка (mean absolute error, MAE)

MAE определяется следующим образом:

$$\text{MAE} = \frac{1}{n} \sum |y - \hat{y}|$$

где n – количество обучающих примеров, y – истинное значение, а \hat{y} – прогнозируемое значение. Для каждого обучающего примера вычисляется абсолютная разность между предсказанным значением и истинным значением. Другими словами, MAE – это средняя ошибка модели.

Средняя абсолютная ошибка в процентах (mean absolute percentage error, MAPE)

MAPE определяется следующим образом:

$$\text{MAPE} = \frac{1}{n} \sum \left| \frac{y - \hat{y}}{y} \right| \times 100 \text{ \%}.$$

Как следует из названия, эта метрика дает ошибку для всех примеров, выраженную в процентах. Это особенно полезно для случая, когда вам нужно определить, допускает ли модель систематические ошибки.

Среднеквадратичная ошибка (mean squared error, MSE)

MSE определяется следующим образом:

$$\text{MSE} = \frac{1}{n} \sum (y - \hat{y})^2.$$

Определение MSE похоже на MAE, за исключением того, что член $y - \hat{y}$ возведен в квадрат. Это делает среднеквадратичную ошибку, гораздо более чувствительную к выбросам значений, нежели средняя абсолютная ошибка.

После того как вы выбрали метрики, подходящие для вашей бизнес-задачи, следующим шагом будет их включение в конвейер машинного обучения. Вы можете сделать это с помощью пакета TFMA, о чем мы расскажем в следующем разделе.

АНАЛИЗ МОДЕЛИ TENSORFLOW

Пакет TFMA предоставляет нам простой способ получить более подробные метрики, чем те, которые используются во время обучения модели. Он позволяет нам визуализировать метрики в виде временных рядов по версиям модели и дает возможность просматривать метрики на различных срезах набора данных. Он также легко масштабируется до больших оценочных наборов благодаря Apache Beam.

В конвейере TFX TFMA рассчитывает метрики на основе сохраненной модели, которая экспортируется компонентом Trainer, и именно эта модель будет развернута. Таким образом, можно избежать путаницы между разными версиями модели. Во время обучения модели, в том случае, если вы используете TensorBoard, вы получите только приблизительные метрики, экстраполированные из измерений на мини-пакетах, но TFMA рассчитывает метрики по всему оценочному набору. Это особенно актуально для больших оценочных наборов.

Анализ одной модели в TFMA

В этом разделе главы мы покажем, как использовать TFMA как отдельный пакет. TFMA устанавливается следующим образом:

```
$ pip install tensorflow-model-analysis
```

В качестве входных данных требуется сохраненная модель и оценочный набор данных. В этом примере мы предположим, что модель Keras сохранена в формате SavedModel, а оценочный набор данных доступен в виде файла в формате TFRecord.

Для начала SavedModel необходимо преобразовать в EvalSharedModel:

```
import tensorflow_model_analysis as tfma

eval_shared_model = tfma.default_eval_shared_model(
    eval_saved_model_path=_MODEL_DIR,
    tags=[tf.saved_model.SERVING])
```

Далее нам потребуется файл EvalConfig. На этом этапе мы сообщаем TFMA, что представляет собой наша метка, предоставляем любые спецификации для разделения модели по одному из признаков и задаем все метрики, которые хотим рассчитывать и отображать с помощью TFMA:

```
eval_config=tfma.EvalConfig(
    model_specs=[tfma.ModelSpec(label_key='consumer_disputed')],
    slicing_specs=[tfma.SlicingSpec()],
    metrics_specs=[
        tfma.MetricsSpec(metrics=[
            tfma.MetricConfig(class_name='BinaryAccuracy'),
            tfma.MetricConfig(class_name='ExampleCount'),
            tfma.MetricConfig(class_name='FalsePositives'),
            tfma.MetricConfig(class_name='TruePositives'),
            tfma.MetricConfig(class_name='FalseNegatives'),
            tfma.MetricConfig(class_name='TrueNegatives')
        ])
    ])
)
```



Анализ моделей TFLite

Мы можем также анализировать модели TFLite в TFMA. В этом случае тип модели необходимо передать в ModelSpec:

```
eval_config = tfma.EvalConfig(
    model_specs=[tfma.ModelSpec(label_key='my_label',
                                model_type=tfma.TF_LITE)],
    ...
)
```

Более подробно TFLite будет обсуждаться в разделе «TFLite» главы 9.

Затем запустите анализ модели:

```
eval_result = tfma.run_model_analysis(
    eval_shared_model=eval_shared_model,
    eval_config=eval_config,
    data_location=_EVAL_DATA_FILE,
    output_path=_EVAL_RESULT_LOCATION,
    file_format='tfrecords')
```

Просмотрите полученные результаты в среде Jupyter Notebook:

```
tfma.view.render_slicing_metrics(eval_result)
```

Несмотря на то что мы хотим просмотреть общие метрики, мы все равно вызываем `render_slicing_metrics`. Срез в данном контексте – это общий срез, который представляет собой весь набор данных. Результат показан на рис. 7.4.

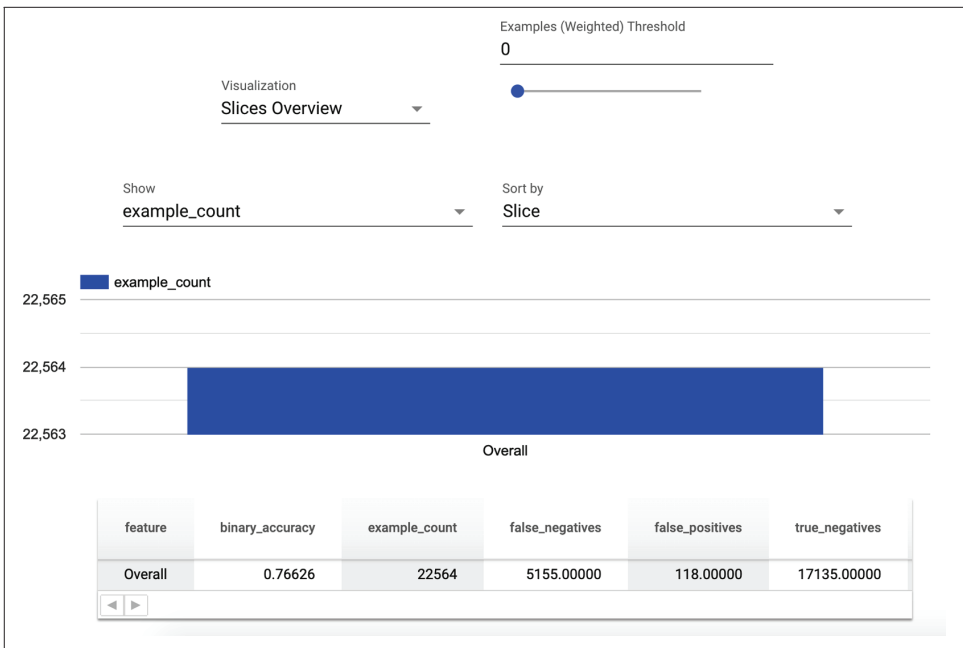


Рис. 7.4. Визуализация TFMA в блокноте Jupyter Notebook для общих показателей



Работа с TFMA в блокноте Jupyter Notebook

Как было показано ранее, TFMA работает в блокноте интерактивной среды Google Colab. Но для просмотра визуализаций в автономной среде Jupyter Notebook требуется несколько дополнительных шагов. Установите и включите расширение TFMA notebook, выполнив следующие команды:

```
$ jupyter nbextension enable --py widgetsnbextension
$ jupyter nbextension install --py \
  --symlink tensorflow_model_analysis
$ jupyter nbextension enable --py tensorflow_model_analysis
```

Добавьте флаг `--sys_prefix` к каждой из этих команд, если вы запускаете их в виртуальной среде Python. Вам также может потребоваться установить или обновить пакеты `widgetsnbextension`, `ipywidgets` и `jupyter_nbextensions_configurator`.

На тот момент, когда мы работали над этой книгой, визуализации TFMA были недоступны в Jupyter Lab, только в Jupyter Notebook.

Все метрики, которые мы описали в разделе «Как проанализировать модель» ранее в этой главе, можно отобразить в TFMA, перечислив их в аргументе `metrics_specs` файла `EvalConfig`:

```
metrics_specs=[
    tfma.MetricsSpec(metrics=[
        tfma.MetricConfig(class_name='BinaryAccuracy'),
        tfma.MetricConfig(class_name='AUC'),
        tfma.MetricConfig(class_name='ExampleCount'),
        tfma.MetricConfig(class_name='Precision'),
        tfma.MetricConfig(class_name='Recall')
    ])
]
```

Результаты, которые должны получиться в результате выполнения этих действий, показаны на рис. 7.5.

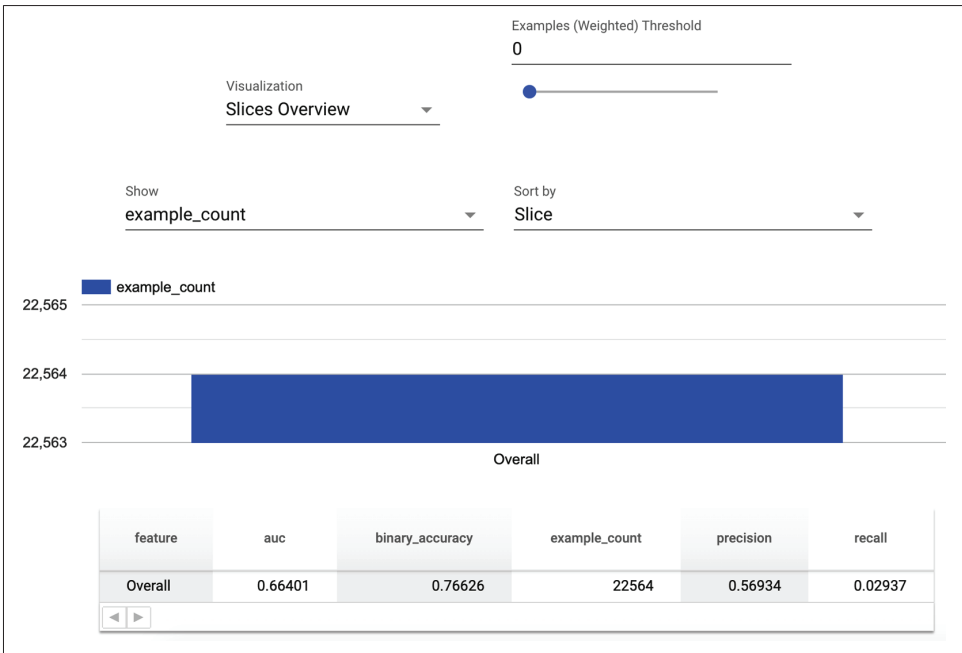


Рис. 7.5. Визуализация блокнота TFMA для других показателей

Анализ нескольких моделей в TFMA

Мы также можем использовать TFMA для сравнения наших показателей для нескольких моделей. Это может быть, например, одна и та же модель, обученная на разных наборах данных, или две модели с разными гиперпараметрами, обученные на одном и том же наборе данных.

Для начала нам необходимо сгенерировать `eval_result` для сравниваемых моделей, как в предыдущих примерах кода. Нам нужно убедиться, что мы правильно указали расположение `output_path` для сохранения моделей. Мы используем один и тот же файл `EvalConfig` для обеих моделей, чтобы у нас была возможность вычислять одни и те же метрики:

```
eval_shared_model_2 = tfma.default_eval_shared_model(
    eval_saved_model_path=_EVAL_MODEL_DIR, tags=[tf.saved_model.SERVING])

eval_result_2 = tfma.run_model_analysis(
    eval_shared_model=eval_shared_model_2,
    eval_config=eval_config,
    data_location=_EVAL_DATA_FILE,
    output_path=_EVAL_RESULT_LOCATION_2,
    file_format='tfrecords')
```

Затем мы должны загрузить их при помощи следующего фрагмента кода:

```
eval_results_from_disk = tfma.load_eval_results(
    [_EVAL_RESULT_LOCATION, _EVAL_RESULT_LOCATION_2],
    tfma.constants.MODEL_CENTRIC_MODE)
```

И мы можем визуализировать их с помощью следующего фрагмента кода:

```
tfma.view.render_time_series(eval_results_from_disk, slices[0])
```

Результат показан на рис. 7.6.



Рис. 7.6. Визуализация TFMA, сравнение двух моделей

Здесь важно отметить, что как для моделей классификации, так и для регрессионных моделей в TFMA можно просматривать множество метрик одновременно, а не ограничиваться одной или двумя из них во время обучения. Это помогает предотвратить неожиданное поведение модели после ее развертывания.

Мы также можем создавать срезы оценочных данных на основе признаков набора данных, например чтобы получить значение метрики верности по продуктам в нашем демонстрационном проекте. Мы расскажем, как это сделать, в следующем разделе.

Анализ достоверности модели

Во всех наборах данных, которые мы используем для обучения модели, некоторым образом присутствует смещение: реальный мир невероятно сложен, и невозможно взять образец данных, который адекватно отражает всю эту сложность. В главе 4, во время первых шагов на пути к построению целостного конвейера машинного обучения, мы познакомились с понятием смещения данных, а в этой главе мы посмотрим, верны ли прогнозы модели.



Справедливость и смещение данных

Термины «справедливость» и «предвзятость» часто используются как синонимы для обозначения того, различаются ли количественные характеристики качества прогнозов модели машинного обучения для разных групп людей. Здесь мы будем использовать термин «справедливость», чтобы избежать путаницы со смещением данных, или предвзятостью данных, которое мы обсуждали в главе 4.

Чтобы оценить, справедлива ли наша модель, нам нужно определить, когда и в каких случаях опыт одних групп людей отличается от опыта других групп людей, в результате чего возникают проблемы. Например, такой группой людей могут быть люди, которые не возвращают деньги, взятые займы. Если наша модель пытается предсказать, кому следует предоставить кредит, такая группа людей должна иметь результат предсказания, отличающийся от остальных групп. В данном случае мы хотим избежать проблемы, когда, например, все люди, которым ошибочно отказывают в кредите, принадлежат к определенной расе.

Ярким примером групп, получающих разные результаты при работе модели, является алгоритм COMPAS, прогнозирующий риск повторного совершения правонарушений преступниками. Согласно данным, опубликованным ProPublica (см. <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing>), частота ошибок алгоритма была примерно одинаковой для черных и белых обвиняемых. Однако вероятность ошибочного предсказания того, что чернокожие обвиняемые впоследствии совершат повторные правонарушения, была особенно велика и оказалась примерно вдвое больше, чем вероятность аналогичного неверного прогноза для белых подсудимых.

Мы должны попытаться распознать такие проблемы до того, как наши модели будут запущены в промышленную эксплуатацию. Для начала полезно дать численные определения того, что мы понимаем под справедливостью. Вот несколько примеров численных определений для решения задач классификации.

Демографический паритет

Модель выполняет оценку, выдавая одинаковую процентную ставку для всех групп. Так, например, модель не завышает процентную ставку по кредиту ни для мужчин (по сравнению с женщинами), ни для женщин (по сравнению с мужчинами).

Равные возможности

Частота ошибок для класса, предоставляющего возможность, одинакова для всех групп. В зависимости от того, как поставлена задача, это может быть положительный или отрицательный класс. Например, для представителей положительного класса – категории всех кредитоспособных людей – и мужчины, и женщины – смогут получить кредит с одинаковой процентной ставкой.

Одинаковая верность

Некоторые показатели, такие как верность, точность или AUC, одинаковы для всех групп. Например, система распознавания лиц должна выдавать настолько же верные результаты для темнокожих женщин, как и для белых мужчин.

Равная верность иногда может вводить в заблуждение, как в предыдущем примере с COMPAS. В этом примере верность прогноза была одинаковой для обеих групп, но ошибки при предсказании вероятности рецидива были намного выше для одной из групп. Важно определять и учитывать определенные тенденции для ошибок, которые могут иметь самые серьезные последствия для вашей модели.



Определения справедливости

Единого определения справедливости, подходящего для всех проектов машинного обучения, не существует. Вам нужно будет определить, что лучше всего подходит для вашей конкретной бизнес-задачи, принимая во внимание потенциальный вред и преимущества для пользователей, работающих с вашей моделью. За дополнительными рекомендациями обратитесь к литературе – книге Солон Барокаса и др. «Справедливость в машинном обучении» (Solon Barocas et al., *Fairness in Machine Learning*), статье Мартина Ваттенберга (Martin Wattenberg) и других авторов из Google, опубликованной по ссылке <http://research.google.com/bigpicture/attacking-discrimination-in-ml/>, а также к документации по метрикам справедливости Бена Хатчинсона (Ben Hutchinson) и других авторов (см. <https://oreil.ly/O237L>).

Группы, которые мы здесь упоминаем, могут быть разными типами клиентов, пользователями продуктов из разных стран или людьми разного пола и этнической принадлежности. В законодательстве США существует концепция защищенных групп, в которой люди защищены от дискриминации по признаку пола, расы, возраста, инвалидности, цвета кожи, вероисповедания, национального происхождения, религии и генетической информации. И эти группы пере-

секаются: вам может потребоваться проверить, что ваша модель не проявляет дискриминацию по отношению к нескольким комбинациям таких групп.



Группы – это упрощение

Группы людей в реальном мире никогда не бывают однозначно определенными. У каждого своя сложная история: например, кто-то мог сменить религию или пол. Кто-то может принадлежать к нескольким расам или нескольким национальностям. Ищите эти пограничные случаи и предоставьте людям возможность обратной связи, если ваша модель для них плохо работает.

Даже если вы не используете эти группы в качестве признаков в своей модели, это не означает, что ваша модель непредвзята. Многие другие признаки, такие как географическое положение, могут сильно коррелировать с одной из этих защищенных групп. Например, если вы используете почтовый индекс США в качестве признака, то такой признак сильно коррелирует с расой. Вы можете проверить наличие этих проблем, выделив отдельный набор данных для одной из защищенных групп, как мы опишем в следующем разделе, даже если это не признак, который вы использовали для обучения своей модели.

Справедливость – нелегкая тема, и она приводит нас ко многим этическим вопросам, которые могут быть сложны и неоднозначны. Однако есть несколько проектов, которые могут помочь нам проанализировать наши модели с точки зрения справедливости, и в следующих разделах этой главы мы расскажем, как их можно использовать. Такой анализ может дать вам преимущество этического и коммерческого плана, выдавая корректные результаты для всех пользователей безотносительно их принадлежности к определенной группе. Этот анализ даже может дать вам шанс исправить неявную несправедливость, заложенную в системе, для которой вы создаете модель, – так, например, анализ автоматизированного инструмента найма персонала в Amazon выявил дискриминацию в отношении кандидатов-женщин.

В следующих разделах мы опишем, как использовать три проекта для оценки справедливости в TensorFlow: TFMA, метрики справедливости и инструмент «Что, если».

Формирование срезов для прогнозов модели в TFMA

Первым шагом в оценке справедливости вашей модели машинного обучения является разделение прогнозов вашей модели по интересующим вас группам, например по признаку: полу, расе или стране. Эти срезы могут быть созданы с помощью TFMA или инструментов, вычисляющих метрики справедливости.

Чтобы создать срез данных в TFMA, столбец среза должен быть предоставлен как `SliceSpec`. В этом примере мы выберем признак `Product` для создания среза:

```
slices = [tfma.slicer.SingleSliceSpec(),
          tfma.slicer.SingleSliceSpec(columns=['Product'])]
```

`SingleSliceSpec` без указания аргументов возвращает весь набор данных. Затем запустите анализ модели с указанными срезами:

```
eval_result = tfma.run_model_analysis(  
    eval_shared_model=eval_shared_model,  
    eval_config=eval_config_viz,  
    data_location=_EVAL_DATA_FILE,  
    output_path=_EVAL_RESULT_LOCATION,  
    file_format='tfrecords',  
    slice_spec = slices)
```

Просмотрите результаты, как показано на рис. 7.7:

```
tfma.view.render_slicing_metrics(eval_result, slicing_spec=slices[1])
```

Если мы хотим исследовать демографический паритет (определение демографического паритета приведено ранее), то нам нужно проверить, принадлежит ли одинаковая доля людей в каждой группе к положительному классу. Мы можем проверить это, посмотрев на TPR и TNR для каждой группы.

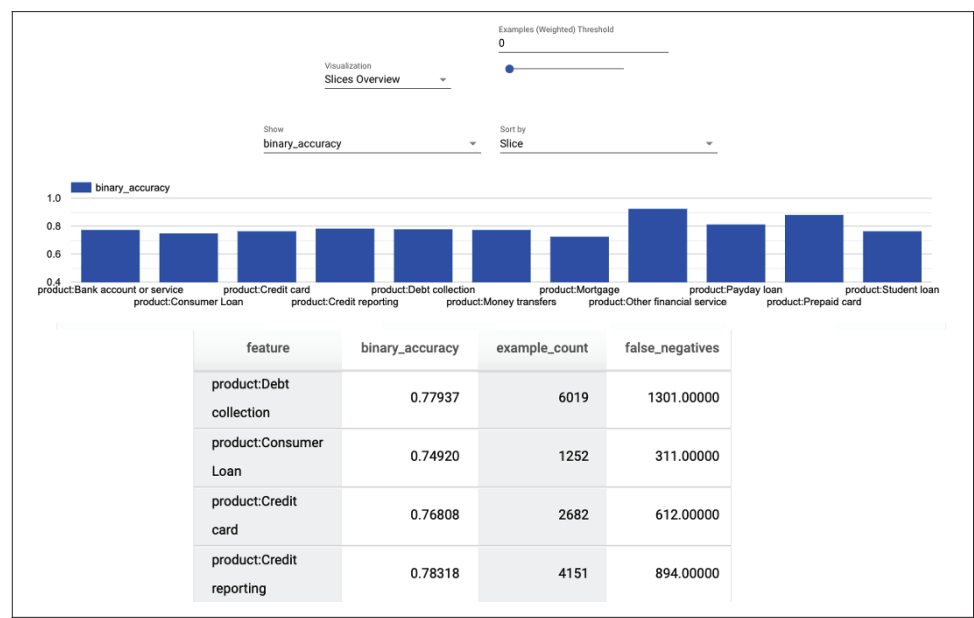


Рис. 7.7. Визуализация среза, формируемого в TFMA



Определите, какой класс получает преимущество

Чтобы определить, какой класс получает преимущество, мы должны исследовать выбор модели и его потенциальную выгоду. Здесь мы предполагаем, что модель делает выбор, который выгоден человеку, и предполагаем, что это положительный класс. Если положительный класс не приносит выгоды, а отрицательный класс приносит вред, то вместо этого мы должны рассматривать метрики, оперирующие с соотношениями истинно отрицательных результатов и ложноположительных результатов.

Для проверки равных возможностей мы можем проверить FPR для каждой группы. Для получения более подробной информации по этой теме обратитесь к документации из раздела полезных советов проекта «Метрики справедливости» (см. <https://oreil.ly/s8Do7>).

Проверка пороговых значений решений с использованием метрик справедливости

Метрики справедливости (Fairness Indicators) – чрезвычайно полезный инструмент для анализа модели. Некоторые из его возможностей аналогичны TFMA, но особенно полезной его функцией является возможность просмотра показателей, сгруппированных по признакам, с различными порогами принятия решения. Как мы обсуждали ранее, порог принятия решения – это оценка вероятности, по которой мы проводим границу между классами для модели классификации. Это позволяет нам проверить, справедлива ли наша модель по отношению к группам с разными порогами принятия решений.

Есть несколько способов получить доступ к инструменту определения метрик справедливости, и самый простой из них – использовать его как автономную библиотеку через TensorBoard. Мы также покажем, как загрузить его как часть конвейера TFX, в разделе «Компонент Evaluator». Для установки подключаемого модуля Tensor Board Fairness Indicators мы выполним команду

```
$ pip install tensorboard_plugin_fairness_indicators
```

Затем мы используем TFMA для оценки модели и просим ее вычислить метрики для набора пороговых значений решений, которые предоставим. Эти данные передаются в TFMA в качестве значения аргумента `metrics_spec` для `EvalConfig` вместе с любыми другими метриками, которые мы хотим вычислить:

```
eval_config=tfma.EvalConfig(
    model_specs=[tfma.ModelSpec(label_key='consumer_disputed')],
    slicing_specs=[tfma.SlicingSpec(),
        tfma.SlicingSpec(feature_keys=['product'])],
    metrics_specs=[
        tfma.MetricsSpec(metrics=[
            tfma.MetricConfig(class_name='FairnessIndicators',
                config='{"thresholds":[0.25, 0.5, 0.75]}')
        ])
    ])
)
```

Затем запустите шаг анализа модели, как и раньше, через `tfma.run_model_analysis`.

Далее запишите результат оценки TFMA в каталог журналов, чтобы TensorBoard мог его забрать:

```
from tensorboard_plugin_fairness_indicators import summary_v2

writer = tf.summary.create_file_writer('./fairness_indicator_logs')
with writer.as_default():
    summary_v2.FairnessIndicators('./eval_result_fairness', step=1)
writer.close()
```

И загрузите результат в TensorBoard в блокнот Jupyter Notebook:

```
%load_ext tensorboard
%tensorboard --logdir=./fairness_indicator_logs
```

Инструмент, с помощью которого мы получаем визуализацию индикаторов справедливости, выделяет отклонения от общего значения показателя, как показано на рис. 7.8.



Рис. 7.8. Визуализация срезов индикаторов справедливости

А для нашего демонстрационного проекта на рис. 7.9 показаны более сильные различия между группами, когда порог принятия решения снижен до 0,25.

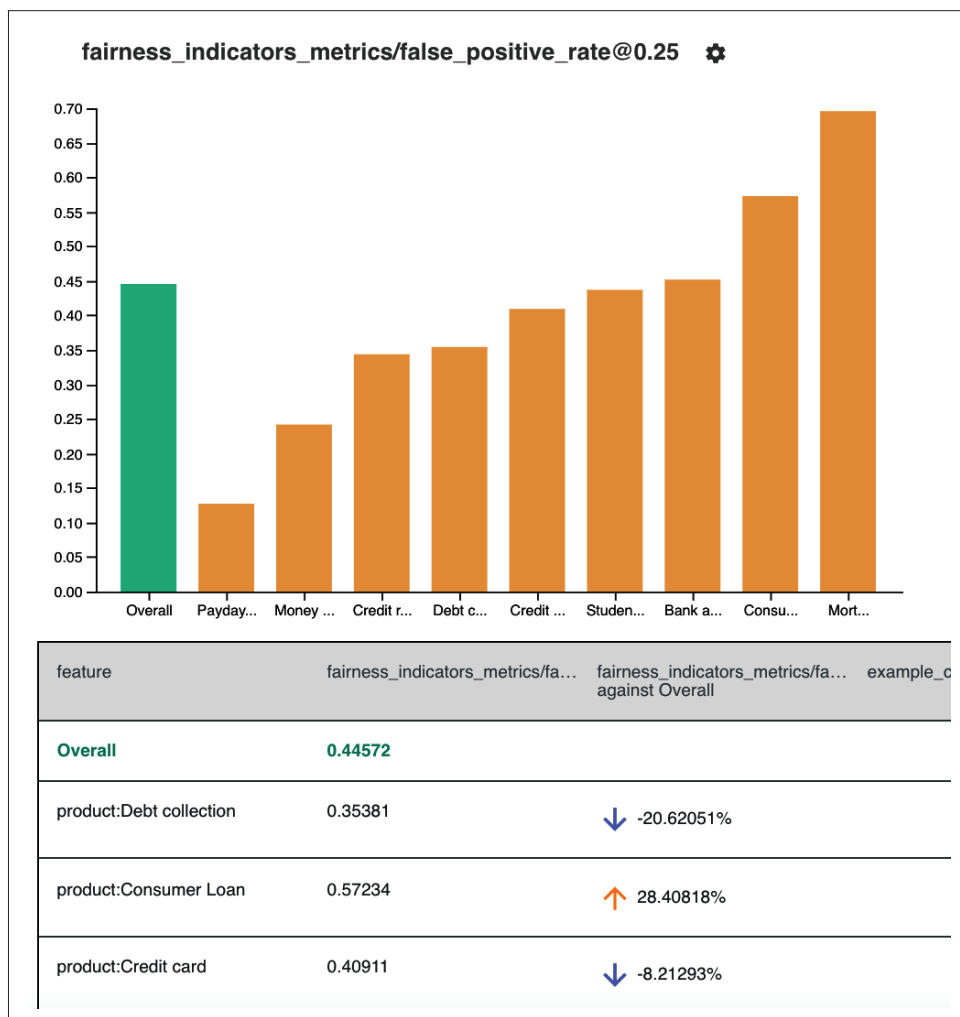


Рис. 7.9. Визуализация пороговых значений индикаторов справедливости

Помимо исследования значений метрик справедливости в общей модели, мы можем захотеть посмотреть на отдельные точки данных, чтобы увидеть, как наша модель влияет на отдельных пользователей. К счастью, в экосистеме TensorFlow есть еще один инструмент, который поможет нам в этом: инструмент анализа альтернатив (What-If Tool).

Проведение более детального анализа с помощью инструмента анализа альтернатив (What-If Tool)

Посмотрев на срезы нашего набора данных с помощью TFMA и визуальных инструментов отображения индикаторов справедливости, мы можем более подробно рассмотреть еще один проект Google: инструмент анализа альтернатив

What-If Tool (WIT). Этот инструмент позволяет нам создавать очень полезные визуализации, а также исследовать отдельные точки данных.

Есть несколько способов использовать WIT для вашей модели и данных. Его можно применять для анализа модели, которая уже была развернута с TensorFlow Serving через TenorBoard, или модели, которая работает на GCP. Его также можно использовать напрямую с моделью Estimator. Но для нашего демонстрационного проекта наиболее простой способ применить его – написать настраиваемую функцию прогнозирования, которая принимает на вход список обучающих примеров и возвращает прогнозы модели для этих примеров. Таким образом, мы можем загрузить визуализацию в автономный блокнот Jupyter Notebook.

Мы можем установить WIT, используя следующую команду:

```
$ pip install witwidget
```

Затем мы создаем TFRecordDataset для загрузки файла данных. Мы отбираем 1000 обучающих примеров и преобразуем их в список TFExamples. Визуализации, получаемые с помощью инструмента What-If Tool, хорошо работают с таким количеством обучающих примеров, но их становится труднее понять при больших размерах выборки:

```
eval_data = tf.data.TFRecordDataset(_EVAL_DATA_FILE)
subset = eval_data.take(1000)
eval_examples = [tf.train.Example.FromString(d.numpy()) for d in subset]
```

Затем мы загружаем модель и определяем функцию прогнозирования, которая принимает список TFExamples и возвращает прогнозы модели:

```
model = tf.saved_model.load(export_dir=_MODEL_DIR) predict_fn = model.signatures['serving_
default']

def predict(test_examples):
    test_examples = tf.constant([example.SerializeToString() for example in examples])
    preds = predict_fn(examples=test_examples)
    return preds['outputs'].numpy()
```

Потом мы настраиваем WIT, выполняя следующую команду:

```
from witwidget.notebook.visualization import WitConfigBuilder

config_builder = WitConfigBuilder(eval_examples).set_custom_predict_fn(predict)
```

И можем просмотреть результат в записной книжке, используя:

```
from witwidget.notebook.visualization import WitWidget

WitWidget(config_builder)
```

Результат визуализации показан на рис. 7.10.

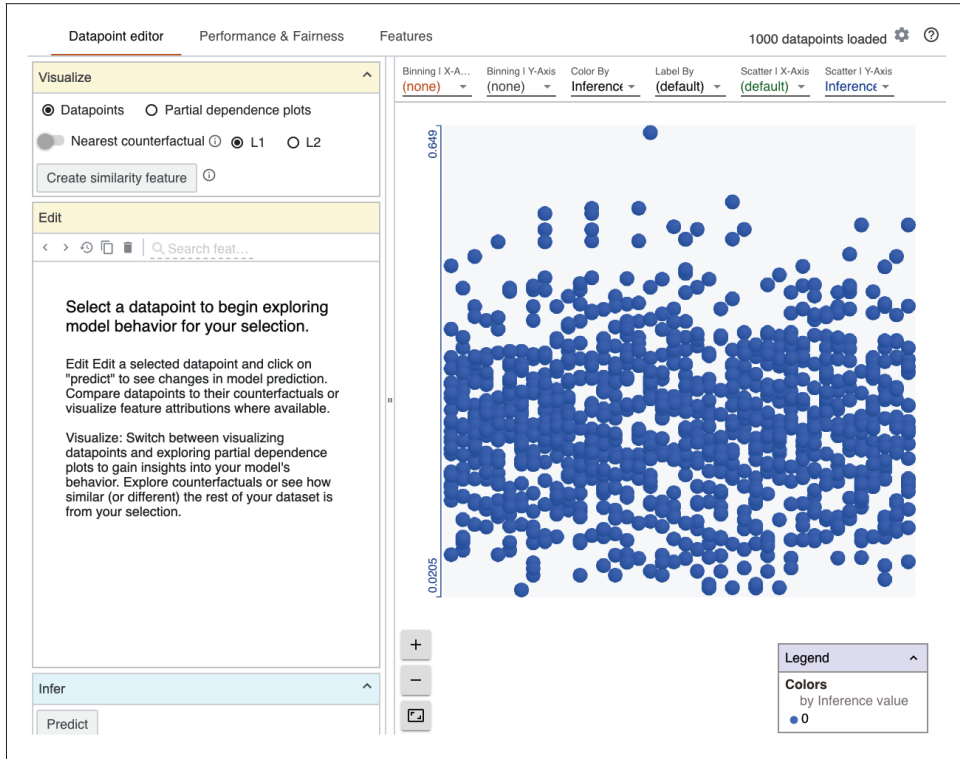


Рис. 7.10. Главная страница WIT



Использование WIT в блокноте Jupyter Notebook

Как и для TFMA, для запуска WIT в автономном блокноте требуется несколько дополнительных шагов. Установите и включите расширение WIT для блокнота с помощью следующих команд:

```
$ jupyter nbextension install --py --symlink \
--sys-prefix witwidget

$ jupyter nbextension enable witwidget --py --sys-prefix
```

Добавьте флаг `--sys_prefix` к каждой из этих команд, если вы запускаете их в виртуальной среде Python.

В WIT включено множество функций, и здесь мы опишем некоторые из наиболее полезных. Полная документация доступна на домашней странице проекта WIT (см. <https://pair-code.github.io/what-if-tool/index.html>).

WIT предоставляет возможности для альтернативных сценариев (контрфакты), которые для любого отдельного обучающего примера показывают его ближайшего соседа из другой классификации. Все признаки максимально похожи, но предсказание модели для альтернативного сценария попадает в другой класс. Это помогает нам увидеть, как каждый признак влияет на прогноз

модели для конкретного обучающего примера. Если мы видим, что изменение демографической характеристики (расы, пола и т. д.) меняет прогноз модели, относя его к другому классу, – это предупреждение о том, что модель может быть несправедливой по отношению к разным группам.

Мы можем продолжить исследовать данный признак, отредактировав выбранный пример в браузере. Затем мы можем снова вывести результат и посмотреть, как это повлияет на прогнозы, сделанные для конкретного примера. Этот подход можно использовать для проведения исследования, справедлива ли наша модель в отношении демографических признаков, а также в отношении любых других характеристик, и увидеть, что произойдет, если значения признаков будут изменены на противоположные.

Альтернативные сценарии также могут быть использованы для объяснения поведения модели. Но обратите внимание, что для каждой точки данных может существовать множество возможных контрфактов, которые близки к тому, чтобы быть ближайшим соседом, а также что между признаками, вероятно, будут существовать сложные взаимозависимости. Таким образом, не следует рассматривать альтернативные сценарии как полное объяснение поведения модели.

Еще одна особенно полезная функция WIT – графики частичной зависимости (partial dependence plots, PDP). Они показывают нам, как каждый из признаков влияет на прогнозы модели, например влияет ли увеличение значения числового признака на вероятность прогноза для класса. PDP показывает нам форму этой зависимости: линейная, монотонная или более сложная. PDP также могут быть созданы для категориальных признаков, как показано на рис. 7.11. Опять же, если прогнозы модели показывают зависимость от демографической характеристики, это может служить предупреждением о том, что прогнозы вашей модели недостоверны.



Рис. 7.11. WIT PDP

Более продвинутая функция, о которой мы не будем рассказывать подробно, – это оптимизация порогов принятия решения для стратегий справедливости. Результаты представлены на странице в WIT; можно автоматически устанавливать пороги принятия решений на основе выбранной стратегии, как показано на рис. 7.12.

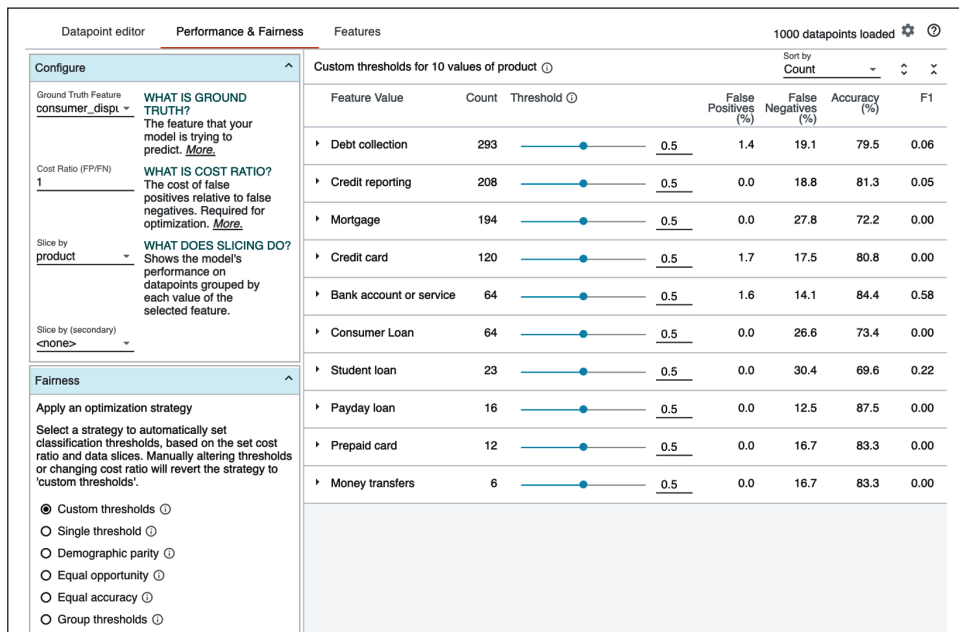


Рис. 7.12. Пороги принятия решений в WIT

Все инструменты, которые мы описываем в этом разделе, посвященном справедливости моделей, также можно использовать для исследования любой модели, даже если она не может нанести пользователям какой-либо вред. Их можно использовать, чтобы лучше понять поведение модели до ее развертывания, и они могут помочь избежать сюрпризов, когда модель попадет в реальный мир. Это область активных исследований, и новые инструменты для подобных исследований выпускаются часто. Одна из интересных разработок – это *ограниченная оптимизация* для метрики справедливости модели, при которой вместо простой оптимизации для одной метрики модели могут быть оптимизированы с учетом других ограничений, таких как равная точность. Для этого в TensorFlow существует экспериментальная библиотека (см. <https://ai.googleblog.com/2020/02/setting-fairness-goals-with-tensorflow.html>).

Объяснение модели

Обсуждение справедливости модели и использования WIT естественным образом приводит нас к вопросу о том, как мы можем не только описать характеристики нашей модели, но и объяснить, что происходит внутри нее. Мы кратко упомянули об этом в предыдущем разделе, посвященном справедливости, но здесь подробнее рассмотрим данный процесс.

В процессе объяснения модели мы пытаемся объяснить, почему предсказания, сделанные моделью, оказываются именно такими. Объяснение модели контрастирует с ее анализом, который описывает работу модели и ее качество при помощи различных показателей. Объяснимость машинного обучения – большая область, в которой сейчас ведется много активных исследований. Мы не можем автоматизировать объяснение модели как часть нашего конвейера, потому что, по определению, объяснения нужно показывать людям. Поэтому мы просто приведем здесь краткий обзор, а для получения более подробной информации рекомендуем электронную книгу «Интерпретируемое машинное обучение» Кристофа Мольнара (*Interpretable Machine Learning, Christoph Molnar*) и еще одну статью, посвященную этому вопросу (см. <https://storage.googleapis.com/cloud-ai-whitepapers/AI%20Explainability%20Whitepaper.pdf>).

Есть несколько возможных причин, по которым вам потребуется объяснить прогнозы вашей модели:

- помощь специалистам по анализу данных в отладке проблем с их моделью;
- повышение доверия к модели;
- аудит модели;
- объяснение предсказаний модели пользовательской аудитории.

Методы, которые мы обсудим далее, полезны во всех этих случаях.

Прогнозы, основанные на более простых моделях, гораздо легче объяснить, чем прогнозы на основе сложных моделей. Линейную регрессию, логистическую регрессию и деревья отдельных решений относительно легко интерпретировать. Мы можем просматривать веса для каждого признака и знаем точный вклад этого признака. Для таких моделей простой взгляд на всю модель целиком дает необходимое объяснение, потому что их архитектура делает их интерпретируемыми по замыслу, так что человек может понять их. Например, коэффициенты модели линейной регрессии дают объяснение, которое понятно и не требует дополнительной работы по разъяснению.

Сложнее объяснить модель случайного леса и другие ансамблевые модели, а труднее всего объяснить глубокие нейронные сети. Это связано с огромным количеством параметров и связей в нейронной сети, что приводит к чрезвычайно сложным взаимодействиям между признаками. Если прогнозы вашей модели имеют серьезные последствия и вам требуются объяснения, мы рекомендуем вам выбирать модели, которые легче объяснить. Более подробную информацию о том, как и когда использовать объяснения, можно найти в статье «Объясняемое машинное обучение на этапе развертывания» Уманга Бхатта и др. («Explainable Machine Learning in Deployment» by Umang Bhatt et al, см. <https://arxiv.org/pdf/1909.06342.pdf>).



Локальные и глобальные объяснения

Мы можем разделить методы объяснения машинного обучения на две большие группы: локальные и глобальные объяснения. Локальные объяснения стремятся объяснить, почему модель сделала конкретный прогноз для одной-единственной точки данных. Глобальные объяснения стремятся объяснить, как модель работает в целом, и исследования с целью объяснения проводятся с использованием большого набора точек данных. Мы познакомим вас с обеими этими техниками в следующем разделе.

А также разберем некоторые методы генерации объяснений на основе вашей модели.

Генерация объяснений с помощью WIT

В разделе «Проведение более детального анализа с помощью инструмента анализа альтернатив (What-If Tool)» мы показали, как можем использовать WIT, чтобы ответить на вопрос о такой характеристике нашей модели, как справедливость. Но инструмент WIT также может оказаться полезен для объяснения наших моделей – в частности, используя контрфакты и PDP, как мы уже упоминали ранее. Использование контрфактов может помочь нам сформировать локальные объяснения, но методы PDP могут давать локальные или глобальные объяснения. Ранее на рис. 7.11 мы приводили пример глобальных PDP, а теперь рассмотрим локальные PDP, как показано на рис. 7.13.

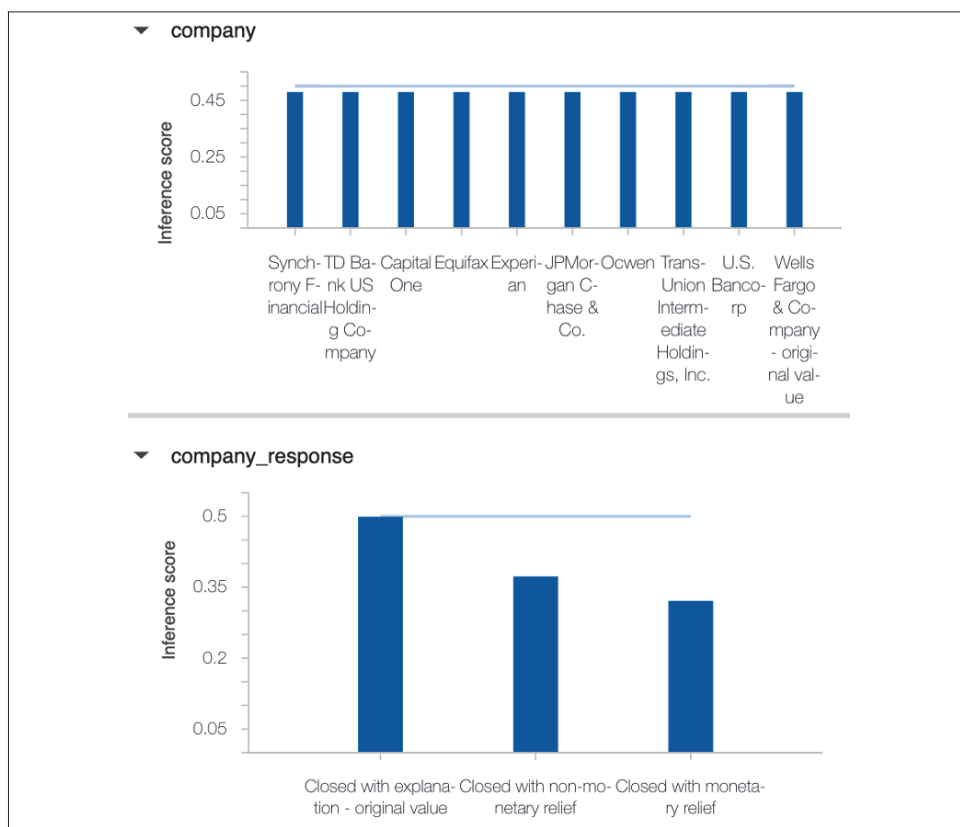


Рис. 7.13. Локальные PDP в WIT

PDP показывают изменение результатов прогноза (*оценку вывода*) для различных допустимых значений признака. Нет никаких изменений в оценке вывода по характеристикам компании, что показывает, что прогнозы для этой точки данных не зависят от значения этого признака. Но для признака `company_response` есть изменение в оценке вывода, которая показывает, что прогноз модели имеет некоторую зависимость от значения признака.



Допущения PDP

PDP содержит важное допущение: все признаки независимы друг от друга. Для большинства наборов данных, особенно тех, которые достаточно сложны, чтобы для точных прогнозов требовалась модель нейронной сети, это предположение не подходит. К получаемым с помощью этого инструмента графикам следует относиться с осторожностью: они могут дать вам представление о том, что делает ваша модель, но они не дают вам полноценного объяснения.

Если ваша модель развернута с использованием платформы искусственного интеллекта Google Cloud, вы можете увидеть *атрибуцию признаков* в WIT. Для одного примера данных атрибуция признаков дает положительные или отрицательные оценки для каждого признака, что указывает на эффект и величину вклада признака в прогноз модели. Их также можно агрегировать, чтобы дать общее объяснение важности признака в вашей модели. Атрибуция признаков основана на значениях *Шепли*, о которых будет рассказываться в следующем разделе. Значения Шепли не предполагают, что ваши признаки независимы, поэтому, в отличие от PDP, их рекомендуется использовать, если ваши признаки коррелированы друг с другом. На момент написания этой книги атрибуция признаков была доступна только для моделей, обученных с помощью TensorFlow 1.x.

Другие методы объяснения

Локальные интерпретируемые независимые от модели объяснения (*local interpretable model-agnostic explanations*, LIME, см. <https://oreil.ly/SrlWc>) – это еще один метод получения локальных объяснений. Он рассматривает модель как черный ящик и генерирует новые точки данных вокруг точки, для которой мы хотели бы получить объяснение. Затем LIME получает прогнозы модели для этих новых точек данных и обучает простую модель, используя эти точки. Веса для этой простой модели дают нам объяснения.

Библиотека SHAP, или Shapley Additive Explanations (аддитивные объяснения Шепли, см. <https://github.com/slundberg/shap>), предоставляет как глобальные, так и локальные объяснения с использованием значений Шепли. Для их вычисления потребуются значительные ресурсы, поэтому библиотека SHAP содержит реализации, которые ускоряют вычисления или используют приближения для усиленных деревьев и глубоких нейронных сетей. Эта библиотека предоставляет вам отличный способ показать важность признака для ваших моделей.

Значения Шепли

Значения Шепли используются как для локальных, так и для глобальных объяснений. Эта концепция представляет собой алгоритм, заимствованный из теории игр, который распределяет выигрыши и проигрыши между каждым игроком в совместной игре для достижения определенного результата игры. В контексте машинного обучения каждый признак рассматривается как «игрок», и значения Шепли можно получить следующим образом:

- 1) получите все возможные подмножества признаков, которые не содержат признак F;
- 2) определите влияние добавления F ко всем подмножествам на прогнозы модели;
- 3) объедините эти эффекты, чтобы получить важность признака F.

Все они относятся к некоторой *базе*. В нашем демонстрационном проекте мы могли бы сформулировать это как «сколько было предсказаний, основанных на том факте, что конечный статус ответа на описание проблемы company_response был Closed with explanation (закрыт с объяснением причины), а не Closed with monetary relief (закрыт с освобождением от платежа)». Значение Closed with monetary relief (закрыт с освобождением от платежа) является нашей базой.

Нам также хотелось бы упомянуть карточки модели (см. <https://modelcards.withgoogle.com/about>), основной инструмент для отчетности по моделям машинного обучения. Это формальный способ поделиться фактами о моделях машинного обучения и существующих в них ограничениях. Мы включаем описание этого инструмента в свою книгу, потому что, хотя они не объясняют, почему модель делает свои прогнозы, они чрезвычайно важны для формирования доверия к модели. Карточка модели должна содержать следующую информацию:

- оценка достоверности модели на общедоступных наборах данных, в том числе достоверность с разбивкой по демографическим характеристикам;
- ограничения модели, например раскрытие информации о том, приведет ли снижение качества изображения к менее точному результату для модели классификации изображений;
- любые компромиссные решения, использованные в модели, например объяснение того, требует ли изображение большего размера более длительного времени обработки.

Карточки модели чрезвычайно полезны для формирования представления о моделях в ситуациях, когда ставки весьма высоки; они побуждают специалистов по данным и инженеров по машинному обучению задокументировать варианты использования и ограничения создаваемых ими моделей.



Ограничения объяснений

Мы рекомендуем проявлять осторожность при объяснении модели. Эти техники могут вызвать у вас ощущение, что вы понимаете, что делает ваша модель, но на самом деле она может делать нечто очень сложное, что-то такое, что невозможно объяснить. В особенности это относится к моделям глубокого обучения.

Невозможно представить всю сложность глубокой нейронной сети, с ее миллионами весовых коэффициентов, в удобочитаемом для человека виде. В ситуациях, когда предсказания модели могут иметь серьезные последствия для реального мира, мы рекомендуем строить простейшие модели с признаками, которые легко объяснить.

Анализ и проверка модели в TFX

До этого момента мы рассматривали анализ модели с участием человека. Такие инструменты чрезвычайно полезны для мониторинга наших моделей и проверки, что они работают так, как мы хотим. Но для конвейера автоматизированного машинного обучения мы предъявляем несколько другие требования – мы хотим, чтобы конвейер работал бесперебойно и предупреждал нас о проблемах. В TFX есть несколько компонентов, которые автоматизируют эту часть конвейера: *Resolver*, *Evaluator* и *Pusher*. Вместе эти компоненты проверяют производительность модели на наборе оценочных данных и отправляют ее в то место, где она будет эксплуатироваться, в том случае если она улучшает предыдущую модель.

TFX использует концепцию *одобрения* (*blessing*) для описания процесса селекции для принятия решения о развертывании модели для ее эксплуатации. Если, сравнивая характеристики двух моделей на основе определенного нами порога, мы приходим к выводу, что новая модель лучше предыдущей, то новая модель одобряется, и конвейер может перейти к следующему шагу.

ResolverNode

Компонент *Resolver* необходим в случае, если мы хотим сравнить новую модель с предыдущей. *ResolverNodes* – это общие компоненты, которые делают запросы к хранилищу метаданных. В этом случае мы используем `latest_blessed_model_resolver`. Он проверяет наличие последней одобренной модели и возвращает ее в качестве базовой, чтобы ее можно было передать компоненту *Evaluator* вместе с новой моделью-кандидатом. *Resolver* не понадобится, если мы не хотим проверять нашу модель по порогу некоторой метрики, однако мы настоятельно рекомендуем выполнять этот шаг. Если вы не проверите новую модель, она будет автоматически отправлена в директорию, где размещаются рабочие модели, даже если ее рабочие характеристики окажутся хуже, чем у предыдущей модели. При первом запуске компонента *Evaluator*, когда у нас нет одобренной модели, *Evaluator* автоматически одобряет модель.

В интерактивном контексте мы можем запустить компонент *Resolver* следующим образом:

```
from tfx.components import ResolverNode
from tfx.dsl.experimental import latest_blessed_model_resolver
from tfx.types import Channel
from tfx.types.standard_artifacts import Model
from tfx.types.standard_artifacts import ModelBlessing

model_resolver = ResolverNode(
    instance_name='latest_blessed_model_resolver',
    resolver_class=latest_blessed_model_resolver.LatestBlessedModelResolver,
    model=Channel(type=Model),
    model_blessing=Channel(type=ModelBlessing)
)
context.run(model_resolver)
```

Компонент Evaluator

Компонент Evaluator использует библиотеку TFMA для оценки прогнозов модели на контрольном наборе данных. Он принимает входные данные из компонента ExampleGen, обученную модель из компонента Trainer и EvalConfig для TFMA (так же, как при использовании TFMA в качестве автономной библиотеки).

Сначала мы определяем EvalConfig:

```
import tensorflow_model_analysis as tfma

eval_config=tfma.EvalConfig(
    model_specs=[tfma.ModelSpec(label_key='consumer_disputed')],
    slicing_specs=[tfma.SlicingSpec(),
                   tfma.SlicingSpec(feature_keys=['product'])],
    metrics_specs=[
        tfma.MetricsSpec(metrics=[
            tfma.MetricConfig(class_name='BinaryAccuracy'),
            tfma.MetricConfig(class_name='ExampleCount'),
            tfma.MetricConfig(class_name='AUC')
        ])
    ]
)
```

Затем запускаем компонент Evaluator:

```
from tfx.components import Evaluator

evaluator = Evaluator(
    examples=example_gen.outputs['examples'],
    model=trainer.outputs['model'],
    baseline_model=model_resolver.outputs['model'],
    eval_config=eval_config
)
context.run(evaluator)
```

Мы также можем показать визуализацию TFMA с помощью:

```
eval_result = evaluator.outputs['evaluation'].get()[0].uri
tfma_result = tfma.load_eval_result(eval_result)
```

И мы можем загрузить индикаторы справедливости с помощью

```
tfma.addons.fairness.view.widget_view.render_fairness_indicator(tfma_result)
```

Проверка при помощи компонента Evaluator

Компонент Evaluator также выполняет проверку, в ходе которой он проверяет, имеет ли модель-кандидат, которую мы только что обучили, улучшенные параметры по сравнению с базовой моделью (например, моделью, которая в настоящее время находится в производственной среде). Он получает прогнозы обеих моделей на оценочном наборе данных и сравнивает метрики качества (например, точность модели) для обеих моделей. Если новая модель имеет лучшие значения метрик по сравнению с предыдущей моделью, новая модель получает признак «одобрение». В настоящее время можно рассчитать метрику только для всего оценочного набора данных, но не для его срезов.

Чтобы выполнить проверку, нам нужно установить порог в EvalConfig:

```
eval_config=tfma.EvalConfig(
    model_specs=[tfma.ModelSpec(label_key='consumer_disputed')],
    slicing_specs=[tfma.SlicingSpec(),
                  tfma.SlicingSpec(feature_keys=['product'])],
    metrics_specs=[
        tfma.MetricsSpec(
            metrics=[
                tfma.MetricConfig(class_name='BinaryAccuracy'),
                tfma.MetricConfig(class_name='ExampleCount'),
                tfma.MetricConfig(class_name='AUC')
            ],
            thresholds={
                'AUC':
                    tfma.config.MetricThreshold(
                        value_threshold=tfma.GenericValueThreshold(
                            lower_bound={'value': 0.65}),
                        change_threshold=tfma.GenericChangeThreshold(
                            direction=tfma.MetricDirection.HIGHER_IS_BETTER,
                            absolute={'value': 0.01}
                        )
                    )
            }
        )
    ]
)
```

В рассматриваемом примере мы объявляем, что AUC должна быть выше 0,65, и мы хотим, чтобы модель была проверена, если ее AUC как минимум на 0,01 выше, чем у базовой модели. Вместо AUC можно добавить любую другую метрику, но обратите внимание, что добавляемая метрика также должна быть включена в список метрик в MetricsSpec.

Мы можем проверить результаты с помощью следующей команды:

```
eval_result = evaluator.outputs['evaluation'].get()[0].uri
print(tfma.load_validation_result(eval_result))
```


Если проверка прошла успешно, она вернет следующий результат:

```
validation_ok: true
```

Компонент TFX Pusher

Компонент Pusher – небольшая, но важная часть нашего конвейера. Он принимает в качестве входных данных сохраненную модель, выходные данные компонента Evaluator и путь к файлу, в котором будут храниться наши модели для обслуживания. Затем он проверяет, одобрил ли компонент Evaluator модель (то есть модель обладает улучшенными характеристиками по сравнению с предыдущей версией, и она превышает те пороговые значения, которые мы установили). Если модель была одобрена, Pusher перемещает модель в рабочий каталог.

Компонент Pusher получает в качестве входных данных модель, полученную на выходе от компонента Evaluator, и путь к рабочему каталогу:

```
from tfx.components import Pusher
from tfx.proto import pusher_pb2
_serving_model_dir = "serving_model_dir"
pusher = Pusher(
    model=trainer.outputs['model'],
    model_blessing=evaluator.outputs['blessing'],
    push_destination=pusher_pb2.PushDestination(
        filesystem=pusher_pb2.PushDestination.Filesystem(
            base_directory=_serving_model_dir)))
context.run(pusher)
```

Как только новая модель будет помещена в рабочий каталог, она может быть выбрана службой TensorFlow Serving (подробнее об этом будет рассказываться в следующей главе).

РЕЗЮМЕ

В этой главе мы рассмотрели, как выполнять анализ рабочих характеристик модели; этот анализ гораздо более детализирован, чем анализ, проводимый во время обучения модели, и остановились на такой характеристике модели, как ее справедливость. Мы также обсудили процесс проверки улучшения рабочих характеристик модели по сравнению с ранее развернутой версией, познакомились с объяснением машинного обучения и дали краткий обзор некоторых технологий в этой области.

Однако мы должны посоветовать здесь проявлять осторожность: то, что вы подробно проанализировали характеристики своей модели с помощью индикаторов справедливости, не гарантирует, что ваша модель является справедливой или этически обоснованной. Важно продолжать наблюдать за работой вашей модели, после того как она будет запущена в производство, и предоставить пользователям способы обратной связи, если они считают, что прогнозы модели неверны. Это особенно важно, когда ставки высоки, а решения, принимаемые в рамках модели, могут оказать большое влияние на пользователей в реальном мире.

Теперь, когда мы выполнили анализ и проверку нашей модели, пора перейти к следующему важному этапу конвейера: развертыванию модели. Следующие две главы содержат необходимую информацию об этом важном этапе.

Глава 8

Развертывание модели с помощью TensorFlow Serving

Развертывание вашей модели машинного обучения – последний шаг перед тем, как пользователи смогут работать с вашей моделью и делать с ее помощью прогнозы. К сожалению, развертывание моделей машинного обучения попадает в «серую зону» в современных представлениях о разделении труда в цифровом мире. Это не просто задача DevOps, поскольку она требует некоторых знаний об архитектуре модели и ее требованиях к аппаратным средствам. В то же время задачи по развертыванию моделей машинного обучения немного выходят за пределы зоны комфорта инженеров по машинному обучению и специалистов по обработке данных. Они прекрасно знают свои модели, но, как правило, испытывают трудности с их развертыванием. В этой и следующей главах мы хотим преодолеть разрыв между этими двумя мирами и провести специалистов по данным и инженеров DevOps через этапы развертывания моделей машинного обучения. На рис. 8.1 показано место этапа развертывания в общей цепочке конвейера машинного обучения.

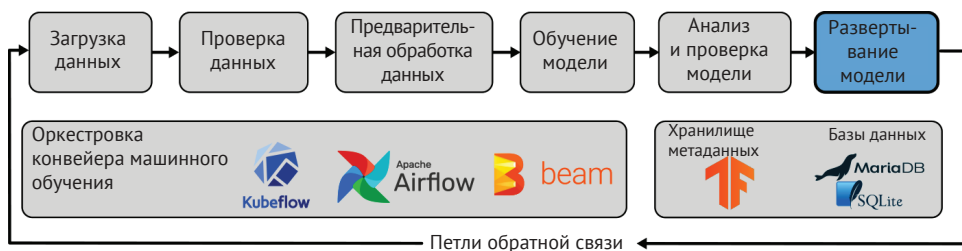


Рис. 8.1. Развертывание моделей как часть конвейера машинного обучения

Модели машинного обучения можно развернуть тремя основными способами: с помощью сервера моделей, в пользовательском браузере или на периферийном устройстве. Наиболее распространенным способом сегодня является развертывание модели машинного обучения на сервере моделей. Клиент, который хочет получить прогноз, отправляет исходные данные на сервер моделей и получает прогноз в ответ. Для этого необходимо, чтобы клиент мог подключиться к серверу моделей. В этой главе мы сосредоточимся на таком типе развертывания модели.

Существуют ситуации, когда пользователи не хотят отправлять исходные данные на сервер моделей, например когда эти данные являются конфиденциальными или когда возникают вопросы по поводу соблюдения конфиденциальности. В этой ситуации вы можете развернуть модель машинного обучения в браузере пользователя. Например, если вы хотите определить, содержит ли изображение конфиденциальную информацию, то можете классифицировать уровень конфиденциальности изображения до его загрузки на облачный сервер.

Однако существует и третий вариант развертывания модели: развертывание на периферийных устройствах. Существуют ситуации, когда вы не можете подключиться к серверу моделей, чтобы получать прогнозы, например когда поставщиками входных данных являются удаленные датчики или устройства IoT. Количество приложений, развертываемых на периферийных устройствах, увеличивается, и теперь такой способ является приемлемым вариантом развертывания моделей. В главе 10 мы расскажем, как можно преобразовать модели TensorFlow в модели TFLite, которые могут работать на периферийных устройствах.

В этой главе мы расскажем про модуль Tensorflow Serving – простой и последовательный способ развертывания моделей TensorFlow при помощи сервера моделей. Мы представим его настройку и обсудим эффективные варианты развертывания. Это не единственный способ развертывания моделей глубокого обучения; есть несколько альтернативных вариантов, которые мы обсудим в конце этой главы.

Давайте в начале главы рассмотрим примеры того, как *не* следует настраивать сервер моделей, прежде чем мы углубимся в тему TensorFlow Serving.

ПРОСТОЙ СЕРВЕР МОДЕЛЕЙ

Большинство вводных инструкций по развертыванию моделей машинного обучения сводятся примерно к одному и тому же сценарию:

- создание веб-приложения на Python (Flask или Django);
- создание конечной точки API в веб-приложении (как показано в примере 8.1);
- загрузка структуры модели и ее веса;
- вызов метода прогнозирования в загруженной модели;
- возвращение результатов прогноза в виде HTTP-запроса.

Пример 8.1. Пример настройки конечной точки Flask для вывода прогнозов модели

```
import json
from flask import Flask, request
from tensorflow.keras.models import load_model
from utils import preprocess ❶

model = load_model('model.h5')  app = Flask(__name__) ❷

@app.route('/classify', methods=['POST'])
def classify():
    complaint_data = request.form["complaint_data"]
    preprocessed_complaint_data = preprocess(complaint_data)
    prediction = model.predict([preprocessed_complaint_data]) ❸
    return json.dumps({"score": prediction}) ❹
```

- ❶ Предварительная обработка для преобразования структуры данных
- ❷ Загрузка вашей обученной модели
- ❸ Выполнение прогноза
- ❹ Возвращение прогноза в виде ответа HTTP

Этот вариант является быстрой и простой реализацией, идеально подходящей для демонстрационных проектов. Однако мы не рекомендуем использовать пример 8.1 для развертывания моделей машинного обучения на конечных точках, которые предполагается использовать в реальной рабочей среде.

В следующем разделе давайте обсудим, почему мы не рекомендуем развертывать модели глубокого обучения с использованием таких настроек. Причиной, почему мы не рекомендуем это делать, являются наши результаты сравнительного тестирования предлагаемого нами решения для развертывания.

НЕДОСТАТКИ РАЗВЕРТЫВАНИЯ МОДЕЛЕЙ С ПОМОЩЬЮ API НА ОСНОВЕ PYTHON

Хотя реализации, приведенной в примере 6.1, может быть вполне достаточно для демонстрационных целей, она имеет несколько существенных недостатков. Один из этих недостатков – невозможность корректного разделения API и кода науки о данных, несогласованная структура API и, как следствие, несогласованное управление версиями модели и неэффективные выводы модели. Мы более подробно рассмотрим эти проблемы в следующих разделах.

Отсутствие разделения кода

В нашем демонстрационном примере (пример 8.1) предполагается, что обученная модель развернута на базе исходного кода API и также размещается в кодовой базе. Это означает, что отсутствует разделение между кодом API и моделью машинного обучения. Это может создавать дополнительные сложности, когда специалисты по машинному обучению хотят обновить модель, и такое обновление требует координации с командой API. Для такой координации также требуется, чтобы команды API и Data Science работали синхронно, дабы избежать ненужных задержек при развертывании модели.

Такая смешанная кодовая база API и машинного обучения также создает неоднозначную ситуацию вокруг владения API.

Отсутствие разделения кода требует, чтобы модель загружалась на том же языке программирования, что и код API. Такое смешение кода серверной части и кода науки о данных может в конечном итоге помешать вашей команде API обновить серверную часть API. Однако он также обеспечивает хорошее разделение обязанностей: специалисты по обработке данных могут сосредоточиться на обучении моделей, а коллеги по DevOps могут сосредоточиться на развертывании обученных моделей.

Мы уделим особое внимание возможности эффективно отделить модели от кода API и упростить рабочие процессы развертывания с помощью TensorFlow Serving.

Отсутствие контроля версий модели

Пример 8.1 не предоставляет никаких возможностей для создания разных версий модели. Если вы хотите добавить новую версию, вам придется создать новую конечную точку (или добавить некоторую логику ветвления к существующей конечной точке). Это требует дополнительного внимания, чтобы сохранить все конечные точки структурно одинаковыми, и вам потребуется написать много шаблонного кода, реализуя функции, которые уже существуют в других системах.

Отсутствие контроля версий модели также накладывает дополнительные ограничения на совместную работу команды разработчиков API и специалистов по обработке данных и машинному обучению: они должны будут согласовать между собой, какая версия является версией по умолчанию и как вводить в эксплуатацию новые модели.

Неэффективный вывод модели

Как показано в примере 8.1, для любого запроса к вашей конечной точке выдачи результатов прогнозирования, указанной в настройке Flask, выполняется полный цикл обмена данными. Это означает, что каждый запрос обрабатывается индивидуально, и для каждого запроса отдельно выводятся его результаты. Основная причина, по которой мы утверждаем, что подобный вариант развертывания предназначен только для демонстрационных целей, заключается в том, что он крайне неэффективен. Во время обучения вашей модели вы, вероятно, используете метод пакетной обработки, который позволяет вам обрабатывать несколько выборок одновременно, а затем применять изменение градиента, вычисленного на основе пакета, к весам вашей сети. Вы можете применить ту же технику, когда хотите, чтобы модель делала прогнозы. Сервер моделей может собирать все запросы в течение ожидаемого периода времени или до тех пор, пока не будет сформирован полный пакет, и отправлять запрос модели для получения ее прогнозов. Этот метод особенно эффективен, когда логический вывод модели вычисляется на графических процессорах.

В разделе «Пакетные запросы на вывод прогнозов модели» этой главы мы расскажем, как можно легко настроить такое поведение пакетной обработки для вашего сервера моделей.

TENSORFLOW SERVING

Как вы видели в предыдущих главах этой книги, TensorFlow поставляется вместе с экосистемой расширений и инструментов. Одно из ранних расширений с открытым исходным кодом – TensorFlow Serving. Оно позволяет вам развернуть любой граф TensorFlow, и вы можете делать прогнозы из графа через стандартизированные конечные точки. Как мы вскоре покажем, TensorFlow Serving осуществляет управление моделями и версиями, позволяет обслуживать модели на основе политик и дает возможность загружать модели из различных источников. В то же время он ориентирован на высокую пропускную способность для уменьшения времени задержки при составлении и вы-

даче результатов прогнозов. Служба TensorFlow Serving используется в Google, а также многими другими корпорациями и стартапами¹.

ОБЗОР АРХИТЕКТУРЫ TENSORFLOW

TensorFlow Serving предоставляет вам возможности для загрузки моделей из заданного источника (например, сегментов AWS S3) и уведомляет модуль Model Loader, если источник изменился. Как показано на рис. 8.2, все, что находится «за кулисами» TensorFlow Serving, контролируется модулем Model Manager, который следит за тем, когда нужно обновлять модели и какая модель должна использоваться для прогнозов. Правила, определяющие вывод данных, устанавливаются политикой, которой управляет Model Manager. В зависимости от конфигурации вы можете, например, загружать по одной модели за раз и автоматически обновлять модель, когда модуль Model Source обнаружит более новую версию.

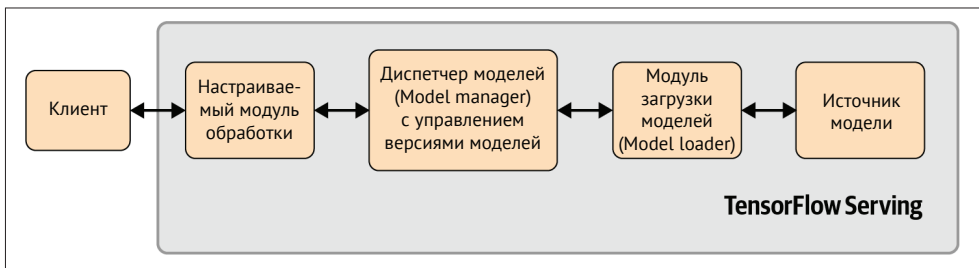


Рис. 8.2. Обзор архитектуры TensorFlow Serving

ЭКСПОРТ МОДЕЛЕЙ ДЛЯ TENSORFLOW SERVING

Прежде чем мы углубимся в тему конфигурации TensorFlow Serving, давайте обсудим, как вы можете экспортировать свои модели машинного обучения, чтобы они могли использоваться в TensorFlow Serving.

В зависимости от вашего типа модели TensorFlow этапы экспорта могут немного отличаться.

Экспортируемые модели имеют такую же файловую структуру, какую мы видим в примере 8.2.

Для моделей Keras вы можете использовать команду

```
saved_model_path = model.save(file_path="./saved_models", save_format="tf")
```

¹ Описание вариантов использования см. по ссылке <https://www.tensorflow.org/about/case-studies>.



Добавление метки времени к пути экспорта

При сохранении модели вручную для модели Keras рекомендуется добавить метку времени экспорта в путь экспорта. В отличие от метода сохранения `tf.Estimator`, `model.save()` не создает путь с меткой времени в автоматическом режиме. Вы можете легко создать путь к файлу, содержащий метку времени, с помощью следующего кода Python:

```
import time

ts = int(time.time())
file_path = "./saved_models/{}".format(ts)
saved_model_path = model.save(file_path=file_path,
                              save_format="tf")
```

Для моделей TensorFlow Estimator необходимо сначала объявить функцию `receiver`:

```
import tensorflow as tf

def serving_input_receiver_fn():
    # an example input feature
    input_feature = tf.compat.v1.placeholder(
        dtype=tf.string, shape=[None, 1], name="input")

    fn = tf.estimator.export.build_raw_serving_input_receiver_fn(
        features={"input_feature": input_feature})
    return fn
```

а затем экспортировать модель с помощью метода `export_saved_model` модуля Estimator:

```
estimator = tf.estimator.Estimator(model_fn, "model", params={})
estimator.export_saved_model(
    export_dir_base="saved_models/",
    serving_input_receiver_fn=serving_input_receiver_fn)
```

Результат при использовании обоих методов выглядит приблизительно так:

```
...
INFO:tensorflow:Signatures INCLUDED in export for Classify: None
INFO:tensorflow:Signatures INCLUDED in export for Regress: None
INFO:tensorflow:Signatures INCLUDED in export for Predict: ['serving_default']
INFO:tensorflow:Signatures INCLUDED in export for Train: None
INFO:tensorflow:Signatures INCLUDED in export for Eval: None
INFO:tensorflow:No assets to save.
INFO:tensorflow:No assets to write.
INFO:tensorflow:SavedModel written to: saved_models/1555875926/saved_model.pb
Model exported to: b'saved_models/1555875926'
```

В нашем примере мы использовали `saved_models/` в качестве выходного каталога, в котором будет размещена модель. Для каждой экспортируемой модели TensorFlow создает каталог с меткой времени экспорта в качестве имени каталога.

Пример 8.2. Каталог и файловая структура экспортируемых моделей

```
$ tree saved_models/ saved_models/
├── 1555875926
│   ├── assets
│   │   ├── saved_model.json
│   │   ├── saved_model.pb
│   │   └── variables
│   │       ├── checkpoint
│   │       ├── variables.data-00000-of-00001
│   │       └── variables.index
└── 3 directories, 5 files
```

Каталог содержит следующие файлы и подкаталоги:

saved_model.pb

Буферный файл двоичного протокола содержит экспортированную структуру графа модели в виде объекта `MetaGraphDef`.

variables (переменные)

Каталог содержит двоичные файлы со значениями экспортированных переменных и контрольными точками, соответствующими графу экспортированной модели.

assets (ресурсы)

Этот каталог создается, когда для загрузки экспортируемой модели требуются дополнительные файлы. Дополнительный файл может включать словари, о чем мы рассказывали в главе 5.

СИГНАТУРЫ МОДЕЛЕЙ

Сигнатуры модели идентифицируют входы и выходы графа модели, а также *метод* сигнатуры графа. Определение входных и выходных сигнатур позволяет нам сопоставить служебные входы с заданным узлом графа для вывода. Эти сопоставления полезны, если мы хотим обновлять модель без изменения запросов к серверу модели.

Кроме того, *метод* модели определяет ожидаемый образец ее входов и выходов. На данный момент существует три поддерживаемых типа сигнатуры: прогнозирование (`predict`), классификация (`classify`) и регрессия (`regress`). Мы более подробно рассмотрим эти методы в следующем разделе.

Методы сигнатуры

Самый гибкий метод сигнатуры – это *прогнозирование* (`predict`). Если мы не укажем другой метод сигнатуры, TensorFlow будет использовать метод `predict` по умолчанию. В примере 8.3 показан пример сигнатуры для метода `predict`. В этом примере мы сопоставляем ключ `inputs` с узлом графа с именем `sentence`. Прогноз модели – это вывод узла графа `y`, который мы сопоставляем с выходным ключом `scores`.

Метод `predict` позволяет вам определять дополнительные выходные узлы. Полезно добавить дополнительные выходные данные логического вывода, когда вы хотите зафиксировать выходные данные технического плана для визуализации или для отладки сетевого узла.

Пример 8.3. Пример сигнатуры predict модели

```
signature_def: {
  key : "prediction_signature"
  value: {
    inputs: {
      key : "inputs"
      value: {
        name: "sentence:0"
        dtype: DT_STRING
        tensor_shape: ...
      },
    },
    ...
  }
  outputs: {
    key : "scores"
    value: {
      name: "y:0"
      dtype: ...
      tensor_shape: ...
    }
  }
  method_name: "tensorflow/serving/predict"
}
```

Другой метод сигнатуры – классификация (*classify*). Метод принимает на вход один элемент с именем *inputs* и выдает два выходных тензора, *classes* и *scores*. Необходимо определить хотя бы один из выходных тензоров. В нашем примере (см. пример 8.4) модель классификации принимает на вход значения *sentence* и выводит предсказанные категории (*classes*), а также соответствующие оценки (*scores*).

Пример 8.4. Пример сигнатуры classify модели

```
signature_def: {
  key : "classification_signature"
  value: {
    inputs: {
      key : "inputs"
      value: {
        name: "sentence:0"
        dtype: DT_STRING
        tensor_shape: ...
      }
    },
    outputs: {
      key : "classes"
      value: {
        name: "y_classes:0"
        dtype: DT_UINT16
        tensor_shape: ...
      }
    },
    outputs: {
      key : "scores"
      value: {
```

```

        name: "y:0"
        dtype: DT_FLOAT
        tensor_shape: ...
    }
}
method_name: "tensorflow/serving/classify"
}

```

Третий доступный метод сигнатуры – регрессия (*regress*). Этот метод принимает на вход только один элемент с именем *inputs* и имеет только один выходной элемент с именем *outputs*. Данный метод сигнатуры разработан для регрессионных моделей. В примере 8.5 показан пример сигнатуры *regress*.

Пример 8.5. Пример сигнатуры *regress* модели

```

signature_def: {
  key : "regression_signature"
  value: {
    inputs: {
      key : "inputs"
      value: {
        name: "input_tensor_0"
        dtype: ...
        tensor_shape: ...
      }
    }
    outputs: {
      key : "outputs"
      value: {
        name: "y_outputs_0"
        dtype: DT_FLOAT
        tensor_shape: ...
      }
    }
  }
  method_name: "tensorflow/serving/regress"
}

```

В разделе «Структура URL-адреса» этой главы мы снова увидим методы сигнатуры, когда определим структуру URL-адресов для конечных точек нашей модели.

ПРОВЕРКА ЭКСПОТИРОВАННЫХ МОДЕЛЕЙ

После всего разговора об экспорте вашей модели и соответствующих сигнатурах модели давайте обсудим, как вы можете проверить экспортированные модели перед их развертыванием с помощью TensorFlow Serving.

Вы можете установить TensorFlow Serving Python API с помощью следующей команды:

```
$ pip install tensorflow-serving-api
```

После установки у вас есть доступ к полезному инструменту командной строки, который называется SavedModel Command Line Interface (CLI). Этот инструмент позволяет:

проверить подписи экспортированных моделей

Это очень полезно, в первую очередь когда вы не экспортируете модель самостоятельно и хотите узнать о входных и выходных данных графа модели:

протестировать экспортированные модели

Утилита SavedModel CLI позволяет вывести модель без ее развертывания с помощью TensorFlow Serving. Это чрезвычайно полезно, когда вы хотите протестировать входные данные вашей модели.

Мы рассмотрим оба варианта использования в следующих двух разделах.

Проверка модели

`saved_model_cli` помогает понять зависимости модели без проверки исходного кода графа.

Если вам неизвестны доступные наборы тегов¹, вы можете проверить модель с помощью:

```
$ saved_model_cli show --dir saved_models/  
The given SavedModel contains the following tag-sets:  
serve
```

Если ваша модель содержит разные графы для разных сред, например граф для вывода центрального процессора или графического процессора, вы увидите несколько тегов. Если ваша модель содержит несколько тегов, вам нужно указать тег, чтобы проверить соответствующие характеристики модели.

Если вам известен набор тегов `tag_set`, который вы хотите проверить, добавьте его в качестве аргумента, и `saved_model_cli` предоставит вам доступные сигнатуры модели. Наша модель, используемая в качестве примера, имеет только одну сигнатуру, которая называется `serving_default`.

```
$ saved_model_cli show --dir saved_models/ --tag_set serve  
The given SavedModel 'MetaGraphDef' contains 'SignatureDefs' with the following keys:  
SignatureDef key: «serving_default»
```

Используя информацию `tag_set` и `signature_def`, вы теперь можете проверить входные и выходные данные модели. Чтобы получить подробную информацию, добавьте `signature_def` к аргументам CLI.

Следующий пример сигнатуры взят из нашей модели, созданной нашим демонстрационным конвейером. В примере 6.4 мы определили нашу функцию сигнатуры, которая принимает сериализованные записи `tf.Example` в качестве входных данных и выдает прогноз посредством выводов `Tensor outputs`, как показано в следующем примере сигнатуры модели:

¹ Наборы тегов модели используются для идентификации `MetaGraphs` для загрузки. Модель может быть экспортирована с графом, указанным для обучения и обслуживания. Оба элемента `MetaGraph` могут быть представлены с помощью разных тегов модели.

```
$ saved_model_cli show --dir saved_models/ \
    --tag_set serve --signature_def serving_default
The given SavedModel SignatureDef contains the following input(s):
  inputs['examples'] tensor_info:
    dtype: DT_STRING
    shape: (-1)
    name: serving_default_examples:0
The given SavedModel SignatureDef contains the following output(s):
  outputs['outputs'] tensor_info:
    dtype: DT_FLOAT
    shape: (-1, 1)
    name: StatefulPartitionedCall_1:0
Method name is: tensorflow/serving/predict
```

Если вы хотите видеть все сигнатуры независимо от `tag_set` и `signature_def`, вы можете использовать аргумент `--all`:

```
$ saved_model_cli show --dir saved_models/ --all
...
```

После того как мы исследовали сигнатуру модели, теперь мы можем протестировать вывод модели перед развертыванием модели машинного обучения.

Тестирование модели

`saved_model_cli` также позволяет вам тестировать экспортируемую модель, используя выборочные наборы входных данных.

У вас есть три различных способа представить пример входных данных для получения тестового вывода модели.

`--inputs`

Аргумент указывает на файл NumPy, содержащий входные данные, отформатированные как `ndarray` в NumPy.

`--input_exprs`

Аргумент позволяет определить команду Python для указания входных данных. Вы можете использовать функциональность NumPy в своих командах.

`--input_examples`

Аргумент ожидает входные данные, имеющие формат структуры данных `tf.Example` (см. главу 4).

Для тестирования модели вы можете точно указать один из входных аргументов. Более того, `saved_model_cli` предоставляет три необязательных аргумента:

`--outdir`

`saved_model_cli` будет записывать любой вывод графа в стандартный вывод. Если вы предпочитаете записывать вывод в файл, то можете указать целевой каталог с помощью `--outdir`.

--overwrite

Если вы решили записать вывод в файл, вы с помощью `--overwrite` можете указать, что файлы могут быть перезаписаны.

--tf_debug

Если вы также хотите проверить модель, то можете выполнить обход графа модели с помощью отладчика TensorFlow Debugger (TFDBG).

```
$ saved_model_cli run --dir saved_models/ \  
    --tag_set serve \  
    --signature_def x1_x2_to_y \  
    --input_examples 'examples=[{"company": "HSBC", ...}]'
```

После знакомства с тем, как экспортировать модели и как их проверять, давайте перейдем к установке, настройке и эксплуатации TensorFlow Serving.

УСТАНОВКА TENSORFLOW SERVING

Существует два простых способа установить TensorFlow Serving на ваших экземплярах рабочих машин. Вы можете запустить TensorFlow Serving на Docker или, если ваши машины работают на ОС Ubuntu, можете установить пакет Ubuntu.

Установка Docker

Самый простой способ установить TensorFlow Serving – это загрузить предварительно созданный образ Docker. Как вы видели в главе 2, вы можете получить образ Docker, запустив команду

```
$ docker pull tensorflow/serving
```

Если вы запускаете контейнер Docker на вычислительном узле с графическими процессорами, вам необходимо загрузить последнюю сборку, поддерживающую графические процессоры.

```
$ docker pull tensorflow/serving:latest-gpu
```

Для образа Docker с поддержкой графических процессоров необходима аппаратная поддержка Docker для графических процессоров, предоставляемая NVIDIA.

Описание шагов установки можно найти на сайте компании (см. <https://github.com/NVIDIA/nvidia-docker#quick-start>).

Установка на Ubuntu

Если вы хотите запускать TensorFlow Serving без дополнительных затрат на запуск Docker, вы можете установить двоичные пакеты Linux, доступные для дистрибутивов Ubuntu.

Этапы установки аналогичны этапам установки других нестандартных пакетов Ubuntu. Во-первых, вам нужно добавить новый пакет в исходный список источников дистрибутива или добавить новый файл списка в каталог `sources.list.d`, выполнив следующую команду в терминале Linux:

```
$ echo "deb [arch=amd64] http://storage.googleapis.com/tensorflow-serving-apt \
stable tensorflow-model-server tensorflow-model-server-universal" \
| sudo tee /etc/apt/sources.list.d/tensorflow-serving.list
```

После обновления реестра пакетов вы можете установить TensorFlow Serving в вашей операционной системе Ubuntu.

```
$ apt-get update
$ apt-get install tensorflow-model-server
```



Два пакета Ubuntu для TensorFlow Serving

Google предоставляет два пакета Ubuntu для TensorFlow Serving. Пакет `tensorflow-model-server`, на который мы ранее ссылались, является пакетом по умолчанию, и он включает в себя специальные предварительно скомпилированные средства оптимизации ЦП (например, инструкции AVX).

На момент написания этой главы также был выпущен второй пакет с именем `tensorflow-model-server-universal`. Он не содержит предварительно скомпилированных оптимизаций и поэтому может быть запущен на старом оборудовании (например, ЦП без набора инструкций AVX).

Сборка TensorFlow Serving из исходного кода

Рекомендуется запускать TensorFlow Serving, используя при этом предварительно созданный образ Docker или пакеты Ubuntu. Однако в некоторых ситуациях вам может понадобиться скомпилировать пакет TensorFlow Serving, например когда вы хотите оптимизировать развертывание модели для вашего специфического аппаратного обеспечения. На данный момент вы можете собрать TensorFlow Serving только для операционных систем Linux, и для этого вам потребуется инструмент сборки `bazel`. Вы можете найти подробные инструкции в документации TensorFlow Serving (см. https://www.tensorflow.org/tfx/serving/setup#building_from_source).



Оптимизация экземпляров TensorFlow Serving

Если вы собираете TensorFlow Serving «с нуля», настоятельно рекомендуется скомпилировать версию Serving для конкретной версии TensorFlow, которая используется в ваших моделях и поддерживается аппаратными средствами ваших вычислительных узлов, осуществляющих развертывание модели.

НАСТРОЙКА СЕРВЕРА TENSORFLOW

По умолчанию TensorFlow Serving может работать в двух разных режимах. Вы можете указать модель, и TensorFlow Serving всегда будет выдавать самую последнюю модель. Вы также можете указать файл конфигурации со всеми моделями и версиями, которые будут загружены, и TensorFlow Serving загрузит все названные модели.

Конфигурация при работе с одной моделью

Если вы хотите запустить TensorFlow Serving, загрузив одну модель, и переключаться на более новые версии моделей, когда они будут доступны, предпочтительна конфигурация с одной моделью.

Если вы запускаете TensorFlow Serving в среде Docker, то можете запустить образ `tensorflow/serving` с помощью следующей команды:

```
$ docker run -p 8500:8500 \ ❶
    -p 8501:8501 \
    --mount type=bind,source=/tmp/models,target=/models/my_model \ ❷

    -e MODEL_NAME=my_model \ ❸
    -e MODEL_BASE_PATH=/models/my_model \
    -t tensorflow/serving ❹
```

- ❶ Укажите порты по умолчанию
- ❷ Смонтируйте каталог моделей
- ❸ Укажите вашу модель
- ❹ Укажите образ Docker

По умолчанию TensorFlow Serving настроен на создание конечной точки REST (Representational State Transfer) и gRPC (Google Remote Procedure Calls). Указав оба порта, 8500 и 8501, мы обеспечиваем поддержку REST и gRPC¹. Команда `Docker run` монтирует каталог на хост-системе (источнике) в файловую систему контейнера (цели). В главе 2 мы обсуждали, как передавать переменные среды в контейнер Docker. Чтобы запустить сервер в конфигурации одной модели, вам необходимо указать название модели `MODEL_NAME`.

Если вы хотите запустить образ Docker, предварительно созданный для использования графического процессора, вам нужно поменять имя образа Docker на имя последней сборки, поддерживающей использование графического процессора, с помощью команды

```
$ docker run ...
    -t tensorflow/serving:latest-gpu
```

Если вы решили запустить TensorFlow Serving без контейнера Docker, то можете запустить его с помощью следующей команды:

```
$ tensorflow_model_server --port=8500 \
    --rest_api_port=8501 \
    --model_name=my_model \
    --model_base_path=/models/my_model
```

В обоих сценариях вы должны получить результат, который выводится на терминал, аналогичный приведенному ниже:

¹ Более подробно о работе с REST и gRPC рассказывается в разделе «REST или gRPC» этой главы.

```

2019-04-26 03:51:20.304826: I
tensorflow_serving/model_servers/ server.cc:82]
  Building single TensorFlow model file config:
    model_name: my_model model_base_path: /models/my_model
2019-04-26 03:51:20: I tensorflow_serving/model_servers/server_core.cc:461]
  Adding/updating models.
2019-04-26 03:51:20: I
tensorflow_serving/model_servers/ server_core.cc:558]
  (Re-)adding model: my_model
...
2019-04-26 03:51:34.507436: I tensorflow_serving/core/loader_harness.cc:86]
  Successfully loaded servable version {name: my_model version: 1556250435}
2019-04-26 03:51:34.516601: I tensorflow_serving/model_servers/server.cc:313]
  Running gRPC ModelServer at 0.0.0.0:8500 ...
[warn] getaddrinfo: address family for nodename not supported
[evhttp_server.cc : 237] RAW: Entering the event loop ...
2019-04-26 03:51:34.520287: I tensorflow_serving/model_servers/server.cc:333]
  Exporting HTTP/REST API at:localhost:8501 ...

```

Из данных, полученных от сервера, вы можете видеть, что сервер успешно загрузил нашу модель `my_model` и создал две конечные точки: одну конечную точку REST и одну конечную точку gRPC.

TensorFlow Serving существенно упрощает развертывание моделей машинного обучения. Одним из больших преимуществ такого развертывания моделей является возможность «горячей замены». Если загружена новая модель, менеджер моделей сервера (модуль `model manager`) обнаружит новую версию, выгрузит существующую модель и загрузит более новую модель для работы с ней.

Допустим, вы обновили модель и экспортировали новую версию модели в смонтированный каталог на хост-компьютере (если вы работаете с установленным Docker), никаких изменений конфигурации не требуется. Model manager обнаружит более новую модель и перезагрузит конечные точки. Он уведомит вас о событиях выгрузки старой модели и загрузки новой модели. На вашем терминале должны отображаться сообщения, аналогичные приведенным ниже:

```

2019-04-30 00:21:56.486988: I tensorflow_serving/core/basic_manager.cc:739]
  Successfully reserved resources to load servable
    {name: my_model version: 1556583584}
2019-04-30 00:21:56.487043: I tensorflow_serving/core/loader_harness.cc:66]
  Approving load for servable version {name: my_model version: 1556583584}
2019-04-30 00:21:56.487071: I tensorflow_serving/core/loader_harness.cc:74]
  Loading servable version {name: my_model version: 1556583584}
...
2019-04-30 00:22:08.839375: I tensorflow_serving/core/loader_harness.cc:119]
  Unloading servable version {name: my_model version: 1556583236}
2019-04-30 00:22:10.292695: I ./tensorflow_serving/core/simple_loader.h:294]
  Calling MallocExtension_ReleaseToSystem() after servable unload with 1262338988
2019-04-30 00:22:10.292771: I tensorflow_serving/core/loader_harness.cc:127]
  Done unloading servable version {name: my_model version: 1556583236}

```

По умолчанию TensorFlow Serving загрузит модель, имеющую наибольший номер версии. Если вы используете методы экспорта, показанные ранее в этой главе, все модели будут экспортированы в каталоги с меткой времени `epoch`

в качестве имени каталога. Следовательно, более новые модели будут иметь больший номер версии, чем более старые модели.

Эта политика загрузки модели по умолчанию, используемая TensorFlow Serving, также допускает откаты модели. Если вы хотите откатить версию модели, вы можете удалить версию модели из базового пути. Затем сервер модели обнаружит, что версия была удалена, при следующем опросе файловой системы¹, выгрузит удаленную модель и загрузит самую последнюю существующую версию модели.

Конфигурация при работе с несколькими моделями

Вы можете сконфигурировать TensorFlow Serving для загрузки нескольких моделей одновременно. Для этого вам нужно создать файл конфигурации, в котором будут указаны используемые модели.

```
model_config_list {
  config {
    name: 'my_model'
    base_path: '/models/my_model/'
    model_platform: 'tensorflow'
  }
  config {
    name: 'another_model'
    base_path: '/models/another_model/'
    model_platform: 'tensorflow'
  }
}
```

Файл конфигурации содержит один или несколько словарей конфигурации, все они перечислены в списке `model_config_list`.

В конфигурации Docker вы можете смонтировать файл конфигурации и загрузить на сервер моделей файл конфигурации вместо единственной модели:

```
$ docker run -p 8500:8500 \
  -p 8501:8501 \
  --mount type=bind,source=/tmp/models,
  target=/models/my_model \ ❶
  --mount type=bind,source=/tmp/model_config,\ target=/models/model_config \
  -e MODEL_NAME=my_model \
  -t tensorflow/serving \
  --model_config_file=/models/model_config ❷
```

- ❶ Смонтируйте файл конфигурации
- ❷ Укажите файл конфигурации модели

Если вы используете TensorFlow Serving вне контейнера Docker, то можете указать серверу модели файл конфигурации с дополнительным аргументом `model_config_file`, и конфигурация будет загружена из этого файла:

¹ Загрузка и выгрузка моделей работает только в том случае, если для параметра `file_system_poll_wait_seconds` было задано значение, большее 0. Значение по умолчанию равно 2 секунды.

```
$ tensorflow_model_server --port=8500 \
    --rest_api_port=8501 \
    --model_config_file=/models/model_config
```

Конфигурация с выбранными версиями модели

Существуют ситуации, когда вы хотите загрузить не только последнюю версию модели, но или все, или конкретные версии модели. Например, вы можете захотеть провести A/B-тестирование модели, как будет обсуждаться в разделе «A/B-тестирование модели с использованием TensorFlow Serving», или использовать стабильную версию модели и версию, предназначенную для разработки. TensorFlow Serving по умолчанию всегда загружает последнюю версию модели. Если вы хотите загрузить все доступные версии модели, то можете расширить файл конфигурации модели с помощью команды

```
...
config {
  name: 'another_model'
  base_path: '/models/another_model/'
  model_version_policy: {all: {}}
}
...
```

Если вы хотите указать конкретные версии модели, вы также можете определить их:

```
config {
  name: 'another_model'
  base_path: '/models/another_model/'
  model_version_policy {
    specific {
      versions: 1556250435
      versions: 1556251435
    }
  }
}
...
```

Вы даже можете задать метки версий моделей. Метки могут быть очень полезны впоследствии, когда вы будете строить прогнозы на основании моделей. На момент написания этой главы метки версий были доступны только для конечных точек gRPC TensorFlow Serving:

```
...
model_version_policy {
  specific {
    versions: 1556250435
    versions: 1556251435
  }
}
```

```
version_labels {  
  key: 'stable'  
  value: 1556250435  
}  
version_labels {  
  key: 'testing'  
  value: 1556251435  
}  
...
```

Теперь, когда мы выбрали нужную версию модели, мы можем использовать эти конечные точки для выбранных версий для запуска нашего А/В-теста модели. Если вас интересует, как получить прогнозные данные этих различных версий модели, мы рекомендуем обратиться к разделу «А/В-тестирование модели с использованием TensorFlow Serving», где приведен пример простой реализации.

Начиная с TensorFlow Serving 2.3 функциональность `version_label` будет также доступна для конечных точек REST, а не только для gRPC TensorFlow Serving.

REST или gRPC

В разделе «Конфигурация при работе с одной моделью» мы обсуждали, что TensorFlow Serving поддерживает два разных типа API: REST и gRPC. Оба протокола имеют свои преимущества и недостатки, и мы хотели бы воспользоваться моментом, чтобы представить оба протокола, прежде чем мы начнем подробно разбирать, как вы можете организовать обмен данными с конечными точками этих протоколов.

REST

REST представляет собой протокол обмена данными, используемый современными веб-сервисами. Это не формальный протокол, а скорее формат обмена данными, определяющий, каким образом клиенты обмениваются сообщениями с веб-сервисами. Клиенты REST связываются с сервером, используя стандартные методы HTTP, такие как GET, POST, DELETE и т. д. Информационные сообщения запросов состоят из данных в формате XML или JSON.

gRPC

gRPC – это протокол удаленных вызовов процедур, разработанный Google. Хотя gRPC поддерживает различные форматы данных, стандартным форматом данных, используемым в gRPC, является буфер протокола (Protobuf), который мы применяли в этой книге. Характерными особенностями gRPC являются низкая задержка связи и информационные сообщения, которые несут меньшую нагрузку, если используются буферы протокола. gRPC был разработан, ориентируясь на API, и ошибки по большей части относятся к API. Недостатком является то, что передаваемые полезные данные представлены в первоначальном формате, что может затруднить быструю проверку.

Какой протокол использовать

На первый взгляд, очень удобно общаться с сервером моделей через REST. Данные легко получить от конечных точек, передаваемые полезные данные легко проверить, а конечные точки можно протестировать с помощью запросов `curl` или инструментов браузера. Библиотеки REST широко доступны для всех типов клиентов и часто уже бывают доступны в клиентской системе (т. е. в мобильном приложении).

С другой стороны, API gRPC изначально имеют более высокую входную нагрузку. Библиотеки gRPC часто необходимо устанавливать на стороне клиента. Однако они могут привести к значительному повышению производительности в зависимости от структур данных, выводимых моделью. Если к вашей модели выполняется много запросов, уменьшение объема передаваемых данных благодаря сериализации буфера протокола может оказаться очень полезным. TensorFlow Serving преобразует структуры данных JSON, переданные через REST, в структуры данных `tf.Example`, что может привести к снижению производительности. Поэтому вы можете получить более высокую производительность запросов gRPC, если требуется выполнить много различных типов преобразований (например, если вы отправляете большой массив со значениями с плавающей точкой).

Выполнение прогнозов на сервере моделей

До сих пор мы полностью сосредоточились на настройке сервера моделей. В этом разделе мы хотим продемонстрировать, как клиент, например веб-приложение, может взаимодействовать с сервером моделей. Все примеры кода, касающиеся запросов REST или gRPC, выполняются на стороне клиента.

Получение прогнозов модели с использованием REST

Чтобы выполнить запрос к серверу моделей через REST, вам понадобится библиотека Python, которая облегчит вам общение. Стандартная библиотека, используемая в настоящее время, — `requests`. После того как вы установите библиотеку с помощью команды

```
$ pip install requests
```

вы можете выполнить запросы. Пример ниже демонстрирует пример запроса POST.

```
import requests
```

```
url = "http://some-domain.abc" payload = {"key_1": "value_1"}
r = requests.post(url, json=payload) ❶
print(r.json()) ❷
# {'data': ...}
```

- ❶ Отправка запроса
- ❷ Просмотр ответа HTTP

Структура URL-адреса

URL для вашего http-запроса к серверу моделей содержит информацию о том, какую модель и какую версию вы бы хотели вывести.

```
http://{HOST}:{PORT}/v1/models/{MODEL_NAME}:{VERB}
```

HOST

Хост – это IP-адрес или доменное имя вашего сервера моделей. Если вы запускаете сервер моделей на той же машине, где вы запускаете код клиента, вы можете установить в качестве параметра HOST значение `localhost`.

PORT

Вам потребуется указать порт в URL вашего запроса. Стандартный порт для REST API – 8501. Если это значение конфликтует с другими сервисами в вашей экосистеме сервисов, вы можете изменить порт, изменив значения параметров вашего сервера во время его запуска.

MODEL_NAME

Имя модели должно совпадать с именем вашей модели, когда вы настраиваете конфигурацию модели или когда запускаете сервер моделей.

VERB

Тип модели указывается посредством команды в URL. Вы можете выбрать один из трех вариантов команды: `predict`, `classify` или `regress`. Параметр VERB соответствует методу сигнатуры, используемому для вашей конечной точки.

MODEL_VERSION

Если вы хотите строить прогнозы на основе конкретной версии модели, вам необходимо добавить в URL-адрес идентификатор нужной версии:

```
http://{HOST}:{PORT}/v1/models/{MODEL_NAME}[/versions/${MODEL_VERSION}]:{VERB}
```

Полезная нагрузка запросов

Посмотрев на структуру URL-адресов, давайте обсудим полезные нагрузки запросов. TensorFlow Serving принимает входные данные в виде структуры данных JSON, показанной в следующем примере:

```
{
  "signature_name": <string>,
  "instances": <value>
}
```

Параметр `signature_name` необязательный. Если он не указан, сервер моделей выведет граф модели, подписанный меткой `serving`, используемой по умолчанию.

В качестве входных данных может использоваться либо список объектов, либо список входных значений. Если вы хотите отправить несколько выборок данных, вы можете представить их в виде списка в параметре `instances`.

Если вы хотите, чтобы выводился только один пример данных, то можете использовать параметр `inputs` и перечислить все входные значения в виде списка.

Должен присутствовать только один из параметров, `instances` либо `inputs`, но не оба одновременно.

```
{
  "signature_name": <string>,
  "inputs": <value>
}
```

В примере 8.6 показан пример кода, позволяющий выполнить запрос для получения прогноза модели от нашей конечной точки TensorFlow Serving. Мы отправляем только один пример данных для получения ответа модели в нашем примере, но мы могли бы легко отправить список входных данных, представляющих несколько запросов.

Пример 8.6. Пример запроса прогноза модели с использованием клиента Python

```
import requests
```

```
def get_rest_request(text, model_name="my_model"):
    url = "http://localhost:8501/v1/models/{:}:predict".format(model_name) ❶
    payload = {"instances": [text]} ❷
    response = requests.post(url=url, json=payload)
    return response
```

```
rs_rest = get_rest_request(text="classify my text")
rs_rest.json()
```

- ❶ Замените `localhost` на IP-адрес, если сервер работает не на той же самой машине, что и клиент
- ❷ Добавьте больше примеров в список `instance`, если хотите вывести больше образцов данных

Работа с TensorFlow Serving через gRPC

Если вы хотите использовать модель с помощью gRPC, то шаги, которые вам необходимо выполнить, будут немного отличаться от запросов REST API.

Сначала вам необходимо создать канал gRPC (`channel`). Канал обеспечивает соединение с сервером gRPC, используя заданный адрес хоста и заданный порт. Если вам требуется безопасное соединение, вам нужно создать безопасный канал на данном этапе. Как только канал будет создан, вы создадите заглушку (`stub`). `stub` – это локальный объект, который дублирует доступные методы сервера.

```
import grpc
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc
import tensorflow as tf

def create_grpc_stub(host, port=8500):
    hostport = "{:}:{:}".format(host, port)
    channel = grpc.insecure_channel(hostport)
    stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
    return stub
```

После создания заглушки gRPC мы можем указать модель и сигнатуру для доступа к прогнозам из нужной нам модели и отправить наши данные для получения вывода модели:

```
def grpc_request(stub, data_sample, model_name='my_model', \
                 signature_name='classification'):
    request = predict_pb2.PredictRequest()
    request.model_spec.name = model_name
    request.model_spec.signature_name = signature_name

    request.inputs['inputs'].CopyFrom(tf.make_tensor_proto(data_sample,
                                                            shape=[1,1])) ❶

    result_future = stub.Predict.future(request, 10) ❷
    return result_future
```

❶ input – это имя входа нашей нейронной сети

❷ 10 – максимальное время до истечения времени ожидания функции

Теперь, когда мы можем использовать две эти функции, мы можем вывести наши наборы данных с помощью двух вызовов функций.

```
stub = create_grpc_stub(host, port='8500')
rs_grpc = grpc_request(stub, data)
```

Безопасные соединения

Библиотека gRPC также предоставляет функциональные возможности для безопасного подключения к конечным точкам gRPC. В следующем примере показано, как создать безопасный канал, использующий протокол gRPC, на стороне клиента:

```
import grpc

cert = open(client_cert_file, 'rb').read()
key = open(client_key_file, 'rb').read()
ca_cert = open(ca_cert_file, 'rb').read() if ca_cert_file else ''
credentials = grpc.ssl_channel_credentials(
    ca_cert, key, cert
)
channel = implementations.secure_channel(hostport, credentials)
```

Если используется SSL, на стороне сервера TensorFlow Serving может разрывать безопасные соединения. Чтобы TensorFlow Serving мог разрывать безопасные соединения, создайте файл конфигурации SSL, как показано в следующем примере¹:

¹ Файл конфигурации SSL использует буфер протокола конфигурации SSL, который можно найти в репозитории TensorFlow Serving API (см. https://github.com/tensorflow/serving/blob/master/tensorflow_serving/config/ssl_config.proto).

```
server_key: "-----BEGIN PRIVATE KEY \n
            <your_ssl_key>\n
            -----END PRIVATE KEY "
server_cert: "-----BEGIN CERTIFICATE \n
            <your_ssl_cert>\n
            -----END CERTIFICATE "
custom_ca: ""
client_verify: false
```

После того как вы создадите файл конфигурации, вы можете передать путь к этому файлу, используя аргумент Tensor Flow Serving `--ssl_config_file` при запуске TensorFlow Serving:

```
$ tensorflow_model_server --port=8500 \
    --rest_api_port=8501 \
    --model_name=my_model \
    --model_base_path=/models/my_model \
    --ssl_config_file="<path_to_config_file>"
```

Получение прогнозов из моделей классификации и регрессии

Если вы хотите получать прогнозы на основе моделей классификации и регрессионных моделей, вы можете сделать это с помощью API gRPC.

Если вы хотите получить прогнозы на основе модели классификации, вам нужно заменить приведенные ниже строки

```
from tensorflow_serving.apis import predict_pb2
...
request = predict_pb2.PredictRequest()
```

следующими строками:

```
from tensorflow_serving.apis import classification_pb2
...
request = classification_pb2.ClassificationRequest()
```

Если вы хотите получить прогнозы на основе регрессионной модели, вы можете использовать следующий вариант операции импорта:

```
from tensorflow_serving.apis import regression_pb2
...
regression_pb2.RegressionRequest()
```

Структура полезных данных

API gRPC использует буферы протокола в качестве структуры данных для запроса API. Благодаря использованию буферов протокола запросы API сжимаются и поэтому могут работать при меньшей пропускной способности, нежели протоколы, обменивающиеся данными в формате JSON. Кроме того, в зависимости от структуры входных данных модели, используя gRPC, вы можете получать прогнозы быстрее, чем в случае, когда используются конечные точки REST. Разница в производительности объясняется тем, что представленные

данные JSON должны быть преобразованы в структуру данных `tf.Example`. Такое преобразование может замедлить вывод данных сервера моделей, и в результате вы можете столкнуться с более низкой производительностью вывода, чем в случае API gRPC.

Ваши данные, отправленные на конечные точки gRPC, должны быть преобразованы в структуру данных буфера протокола. TensorFlow предоставляет вам удобную служебную функцию для выполнения преобразования под названием `tf.make_tensor_proto`. Эта функция допускает различные форматы данных, включая скаляры, списки, скаляры NumPy и массивы NumPy. Затем функция преобразует указанные структуры данных Python или NumPy в формат буфера протокола для вывода.

A/B-ТЕСТИРОВАНИЕ МОДЕЛИ С ИСПОЛЬЗОВАНИЕМ TENSORFLOW SERVING

A/B-тестирование – это отличная методология, позволяющая протестировать разные модели в реальных ситуациях. В нашем сценарии определенный процент клиентов будет получать прогнозы, выполненные версией модели A, а все остальные запросы будут обслуживаться версией модели B.

Ранее мы упоминали, что вы можете настроить TensorFlow Serving для загрузки нескольких версий модели, а затем указать версию модели в URL-адресе запроса REST или в спецификации gRPC.

TensorFlow Serving не поддерживает A/B-тестирование на стороне сервера, но, немного изменив URL нашего запроса, мы можем поддерживать случайное A/B-тестирование на стороне клиента¹.

```
from random import random ❶

def get_rest_url(model_name, host='localhost', port=8501,
                 verb='predict', version=None):
    url = "http://{}/{}/v1/models/{}/".format(host, port, model_name)
    if version:
        url += "versions/{}/".format(version)
    url += ":{}".format(verb)
    return url

...

# Submit 10% of all requests from this client to version 1. # 90% of the requests should go
to the default models. threshold = 0.1
version = 1 if random() < threshold else None ❷
url = get_rest_url(model_name='complaints_classification', version=version)
```

❶ Библиотека `random` поможет нам выбрать модель

❷ Если `version = None`, TensorFlow Serving будет использовать версию по умолчанию

¹ A/B-тестирование не будет полным без статистической проверки результатов, полученных от людей, взаимодействующих с этими двумя моделями. В приведенном здесь примере просто представлена реализация бэкенда A/B-тестирования.

Как видите, случайное изменение URL-адреса запроса для нашего вывода прогноза модели (в нашем примере REST API) может предоставить вам некоторые основные функциональные возможности А/В-тестирования. Если вы хотите расширить возможности, выполняя произвольную маршрутизацию вывода модели на стороне сервера, мы настоятельно рекомендуем для этой цели инструменты маршрутизации, такие как, например, *Istio* (см. <https://istio.io/>). Первоначально разработанный для веб-трафика, этот инструмент можно использовать для маршрутизации трафика на конкретные модели. Вы можете включать модели, выполнять А/В-тесты или создавать политики для данных, перенаправляемых на конкретные модели.

Когда вы выполняете А/В-тестирование, используя свои модели, часто бывает полезно запросить информацию о модели с сервера модели. В следующем разделе мы объясним, как можно запросить информацию о метаданных у TensorFlow Serving.

ЗАПРОС МЕТАДААННЫХ МОДЕЛИ С СЕРВЕРА МОДЕЛЕЙ

В начале книги мы представили жизненный цикл модели и описали то, каким образом мы хотим автоматизировать жизненный цикл машинного обучения. Важным компонентом непрерывного жизненного цикла является получение обратной связи относительно точности или общей производительности ваших версий модели. Мы подробно обсудим, как создавать эти петли обратной связи, в главе 13, а пока представьте, что ваша модель классифицирует некоторые данные, например эмоциональную тональность текста, и затем просит пользователя оценить прогноз. Информация о том, правильно ли модель спрогнозировала что-либо, ценна для улучшения будущих версий модели, но она может быть полезна, только если мы знаем, какая именно версия модели выполнила прогноз.

Метаданные, предоставляемые сервером модели, будут содержать информацию, которая будет отражена в комментариях к нашим циклам обратной связи.

REST-запросы метаданных модели

Запрашивать метаинформацию модели с помощью TensorFlow Serving чрезвычайно легко.

TensorFlow Serving предоставляет конечную точку для получения метаинформации модели.

```
http://{HOST}:{PORT}/v1/models/{MODEL_NAME}/versions/{MODEL_VERSION}/metadata
```

Подобно тому, как недавно было показано на примере запросов REST API, используемых для вывода данных прогнозов модели, у вас есть возможность указать версию модели в URL-адресе запроса; а в случае, если вы не укажете ее, сервер модели предоставит информацию о модели, используемой по умолчанию.

Как показано в примере 8.7, мы можем запросить метаданные модели с помощью одного запроса GET.

Пример 8.7. Пример запроса метаданных модели с использованием клиента Python

```
import requests
```

```
def metadata_rest_request(model_name, host="localhost",
                          port=8501, version=None):
    url = "http://{host}:{port}/v1/models/{model_name}".format(host, port, model_name)
    if version:
        url += "versions/{version}".format(version)
    url += "/metadata" ❶
    response = requests.get(url=url) ❷
    return response
```

❶ Добавьте /metadata, чтобы указать информацию о модели

❷ Выполните запрос GET

Сервер моделей вернет спецификации модели как словарь `model_spec` и определения модели – как словарь `metadata`.

```
{
  "model_spec": {
    "name": "complaints_classification",
    "signature_name": "",
    "version": "1556583584"
  },
  "metadata": {
    "signature_def": {
      "signature_def": {
        "classification": {
          "inputs": {
            "inputs": {
              "dtype": "DT_STRING",
              "tensor_shape": {
                ...
```

Запросы gRPC для метаданных модели

Запрашивать метаданные модели с помощью gRPC почти так же просто, как в случае использования REST API. В случае gRPC вы размещаете запрос `GetModelMetadataRequest`, добавляете имя модели в спецификации и отправляете запрос с помощью метода `GetModelMetadata` объекта `stub`.

```
from tensorflow_serving.apis import get_model_metadata_pb2

def get_model_version(model_name, stub):
    request = get_model_metadata_pb2.GetModelMetadataRequest()
    request.model_spec.name = model_name
    request.metadata_field.append("signature_def")
    response = stub.GetModelMetadata(request, 5)
    return response.model_spec

model_name = 'complaints_classification'
stub = create_grpc_stub('localhost')
get_model_version(model_name, stub)
```

```
name: "complaints_classification"
version {
  value: 1556583584
}
```

Ответ gRPC содержит объект `ModelSpec`, который включает номер версии загруженной модели.

Более интересным является случай получения информации о сигнатурах загруженных моделей. Используя почти те же самые функции запроса, мы можем получить метаданные модели. Единственное отличие состоит в том, что мы обращаемся не к атрибуту `model_spec` объекта `response`, а к `metadata`. Для повышения удобочитаемости информация должна быть сериализована, поэтому мы используем `SerializeToString` для преобразования информации буфера протокола.

```
from tensorflow_serving.apis import get_model_metadata_pb2

def get_model_meta(model_name, stub):
    request = get_model_metadata_pb2.GetModelMetadataRequest()
    request.model_spec.name = model_name
    request.metadata_field.append("signature_def")
    response = stub.GetModelMetadata(request, 5)
    return response.metadata['signature_def']

model_name = 'complaints_classification'
stub = create_grpc_stub('localhost')
meta = get_model_meta(model_name, stub)

print(meta.SerializeToString().decode("utf-8", 'ignore'))
# type.googleapis.com/tensorflow.serving.SignatureDefMap # serving_default
# complaints_classification_input
#   input_1:0
#   2@
# complaints_classification_output(
# dense_1/Sigmoid:0
#   tensorflow/serving/predict
```

Запросы gRPC сложнее запросов REST; однако в приложениях с высокими требованиями к производительности они могут обеспечить более высокую производительность прогнозирования. Еще один способ повысить производительность прогнозирования нашей модели – это размещение запросов на прогнозирование внутри пакетов обмена данными.

ПАКЕТНЫЕ ЗАПРОСЫ НА ВЫВОД ПРОГНОЗОВ МОДЕЛИ

Пакетные запросы на вывод прогнозов модели являются одной из самых мощных функций TensorFlow Serving. Во время обучения модели пакетная обработка запросов ускоряет процесс обучения, потому что мы можем распараллеливать вычисления наших обучающих выборок. Одновременно с этим мы также можем эффективно использовать вычислительное оборудование, если согласуем требования к памяти для наших пакетов с доступной памятью графического процессора.

Как показано на рис. 8.3, если вы запускаете TensorFlow Serving без включенной пакетной обработки, каждый клиентский запрос с одной или несколькими выборками данных создает отдельный вывод результатов модели. Например, если вы классифицируете изображение, ваш первый запрос будет определять модель вашего процессора или графического процессора, прежде чем будет классифицирован второй запрос, третий запрос и т. д. В этом случае мы неэффективно используем доступную память CPU или GPU.

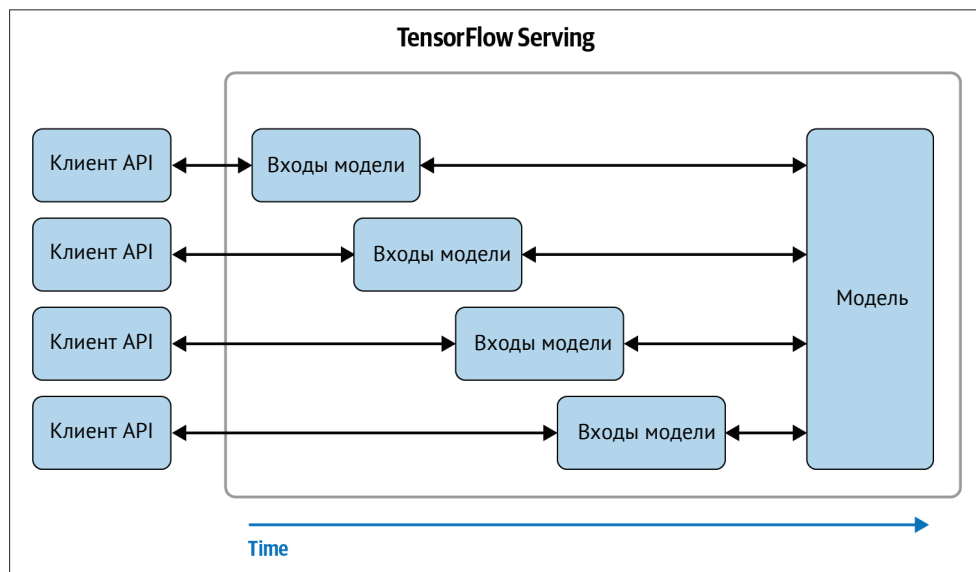


Рис. 8.3. Схема работы TensorFlow Serving без использования пакетного режима

Как показано на рис. 8.4, несколько клиентов могут запрашивать предсказания модели, а сервер модели объединяет различные клиентские запросы в единый «пакет» для выполнения вычислений. Обработка каждого отдельного запроса на шаге процесса, использующем пакетную обработку, может занять немного больше времени, чем обработка каждого отдельного запроса, из-за тайм-аута или ограничений пакета. Однако, как и на этапе обучения, мы можем обрабатывать пакет параллельно и возвращать результаты всем клиентам после завершения вычислений, связанных с обработкой пакета. Это позволит использовать оборудование более эффективно, чем в случае, если бы каждый отдельный клиентский запрос обрабатывался отдельно.

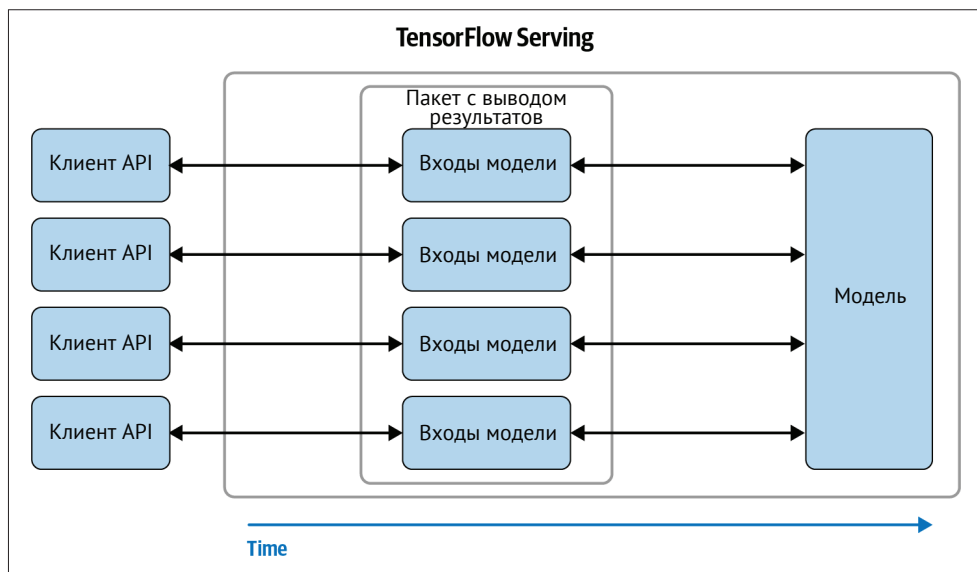


Рис. 8.4. Схема работы TensorFlow Serving с использованием пакетного режима

Настройка использования пакетного режима в прогнозировании

Пакетный режим в TensorFlow Serving необходимо включить, а затем настроить его для вашего варианта использования. Для настроек пакетного режима существует пять параметров конфигурации:

`max_batch_size`

Этот параметр контролирует размер пакета. Большие размеры пакетов увеличат задержку обработки запроса и могут привести к исчерпанию памяти графического процессора. Небольшой размер пакета сводит на нет преимущества оптимального использования вычислительных ресурсов.

`batch_timeout_micros`

Этот параметр устанавливает максимальное время ожидания для заполнения пакета. Удобен для ограничения задержки вывода.

`num_batch_threads`

Количество потоков определяет, сколько ядер процессора или графического процессора можно использовать параллельно.

`max_enqueued_batches`

Этот параметр устанавливает максимальное число пакетов, поставленных в очередь для получения прогнозов. Данный параметр конфигурации полезен, чтобы избежать необоснованно большого числа поставленных в очередь пакетов запросов. При достижении максимального числа пакетов, поставленных в очередь, сервер будет возвращать ошибку в ответ на запросы, а не помещать запросы в очередь.

`pad_variable_length_inputs`

Этот логический параметр определяет, будут ли входные тензоры переменной длины дополняться, чтобы длина массива была одинаковой для всех входных тензоров.

Как вы догадываетесь, определение значений параметров для оптимального использования пакетного режима требует определенных усилий и зависит от характерных особенностей вашего приложения. Если вы запускаете онлайн-приложение, требующее вывода данных, вы должны стремиться к ограничению задержки. Часто рекомендуется сначала установить нулевое значение `batch_timeout_micros` и настроить время ожидания, постепенно увеличивая его значение до 10 000 микросекунд. Напротив, для пакетных запросов будет использоваться более длительное время ожидания (от миллисекунды до секунды), чтобы установить нужный размер пакета, обеспечив таким образом оптимальную производительность. TensorFlow Serving будет выдавать прогнозы для пакета до тех пор, пока не будет достигнуто максимальное значение `max_batch_size` или не возникнет ситуация тайм-аута.

Если вы настраиваете TensorFlow Serving для прогнозов с выполнением вычислений на ЦП, установите для `num_batch_threads` значение, соответствующее количеству ядер ЦП. Если вы настраиваете TensorFlow Serving для прогнозов с выполнением вычислений на графическом процессоре, установите такое значение `max_batch_size`, чтобы получить оптимальное использование памяти графического процессора. При настройке конфигурации убедитесь, что вы установили для `max_enqueued_batches` достаточно большое значение, чтобы избежать ситуации, когда для некоторых запросов будут возвращены ответы, не содержащие корректного результата.

Вы можете задать значения параметров в текстовом файле, как показано в следующем примере. В нашем примере мы назвали файл конфигурации `batching_parameters.txt`, добавив в него соответствующие параметры:

```
max_batch_size { value: 32 }
batch_timeout_micros { value: 5000 }
pad_variable_length_inputs: true
```

Если вы хотите включить пакетный режим, вам нужно передать два дополнительных параметра в контейнер Docker, в котором работает TensorFlow Serving. Установите для `enable_batching` значение `true`, чтобы включить пакетный режим, и в качестве значения параметра `batching_parameters_file` укажите абсолютный путь к файлу конфигурации пакетного режима внутри контейнера. Помните, что вам необходимо смонтировать дополнительный каталог с файлом конфигурации, если этот файл размещен не в том же каталоге, что и версии модели.

Вот полный пример команды `docker run` для запуска контейнера Docker TensorFlow Serving с включенным пакетным режимом. Затем параметры будут переданы в экземпляр TensorFlow Serving.

```
docker run -p 8500:8500 \
  -p 8501:8501 \
  --mount type=bind,source=/path/to/models,target=/models/my_model \
  --mount type=bind,source=/path/to/batch_config,target=/server_config \
  -e MODEL_NAME=my_model -t tensorflow/serving \
  --enable_batching=true
  --batching_parameters_file=/server_config/batching_parameters.txt
```

Как объяснялось ранее, конфигурация пакетной обработки потребует дополнительной настройки, но прирост производительности должен компенсировать затраты времени на первоначальную настройку. Мы настоятельно рекомендуем включить эту функцию TensorFlow Serving. Эта настройка будет особенно полезна в случае, когда необходимо выводить большое количество выборок данных, используя автономные пакетные процессы.

ДРУГИЕ ФУНКЦИИ ОПТИМИЗАЦИИ TENSORFLOW SERVING

TensorFlow Serving поставляется со множеством дополнительных функций оптимизации. Флаги дополнительных функций:

`--file_system_poll_wait_seconds=1`

TensorFlow Serving будет спрашивать, доступна ли новая версия модели. Вы можете отключить функцию, установив ее значение, равное `-1` или, если вы хотите загрузить модель только один раз и никогда не обновлять ее, равное `0`. Параметр принимает целочисленные значения. Если вы загружаете модели из сегментов облачного хранилища, мы настоятельно рекомендуем увеличить время опроса, чтобы избежать ненужных финансовых расходов, связанных с оплатой услуг поставщика облачных решений, при частых операциях со списком в сегменте облачного хранилища.

`--tensorflow_session_parallelism=0`

TensorFlow Serving автоматически определяет, сколько потоков использовать для сессии TensorFlow. В случае если вы хотите установить число потоков вручную, вы можете переопределить его, установив для этого параметра любое положительное целое значение.

`--tensorflow_intra_op_parallelism=0`

Этот параметр устанавливает количество ядер, используемых для работы TensorFlow Serving. Количество доступных потоков определяет, сколько операций будет распараллелено. Если значение равно нулю, будут использованы все доступные ядра.

`--tensorflow_inter_op_parallelism=0`

Этот параметр устанавливает количество доступных потоков в пуле для выполнения операций TensorFlow. Это полезно для выполнения максимального числа независимых операций в графе TensorFlow. Если значение устанавливается равным нулю, все доступные ядра будут использоваться, и выделяется один поток на ядро.

Как и в наших предыдущих примерах, вы можете передать параметр конфигурации команде `docker run`, как показано в следующем примере:

```
docker run -p 8500:8500 \
  -p 8501:8501 \
  --mount type=bind,source=/path/to/models,target=/models/my_model \
  -e MODEL_NAME=my_model -t tensorflow/serving \
  --tensorflow_intra_op_parallelism=4 \
  --tensorflow_inter_op_parallelism=4 \
  --file_system_poll_wait_seconds=10 \
  --tensorflow_session_parallelism=2
```

Вышеперечисленные параметры конфигурации могут повысить производительность и избежать ненужных затрат, связанных с оплатой услуг поставщика облачных решений.

АЛЬТЕРНАТИВЫ TENSORFLOW SERVING

TensorFlow Serving – отличный способ развертывания моделей машинного обучения. Используя модели TensorFlow Estimators и Keras, вы можете выбирать различные концепции машинного обучения. Если вы хотите развернуть унаследованную модель или вы предпочитаете другую среду машинного обучения, отличную от TensorFlow или Keras, ниже мы приведем несколько вариантов для такой ситуации.

BentoML

BentoML – это платформонезависимая библиотека, которая развертывает модели машинного обучения. Она поддерживает модели, обученные с помощью PyTorch, scikit-learn, TensorFlow, Keras и XGBoost. Для моделей TensorFlow BentoML поддерживает формат SavedModel. BentoML поддерживает пакетные запросы.

Seldon

Британский стартап Seldon предоставляет множество инструментов с открытым исходным кодом для управления жизненным циклом моделей, и одним из основных продуктов является Seldon Core (см. <https://www.seldon.io/tech/products/core/>). Seldon Core предоставляет ряд инструментов для упаковки ваших моделей в образ Docker, который затем развертывается при помощи Seldon в кластере Kubernetes.

На момент написания этой главы Seldon поддерживал модели машинного обучения, написанные с использованием TensorFlow, scikit-learn, XGBoost и R.

Seldon поставляется со своей собственной экосистемой, которая позволяет встроить предварительную обработку в собственные образы Docker, развертывающиеся совместно с развертываемыми образами. Он также предоставляет службу маршрутизации, которая позволяет проводить A/B-тестирование или эксперименты с «многоруким бандитом» (рандомизацией выдачи).

Seldon тесно интегрирован со средой KubeFlow и, аналогично TensorFlow Serving, представляет собой способ развертывания моделей с KubeFlow в Kubernetes.

GraphPipe

GraphPipe (см. <https://oracle.github.io/graphpipe/#/>) – это еще один способ развертывания моделей TensorFlow, и не только TensorFlow. Oracle развивает этот проект с открытым исходным кодом. GraphPipe позволяет вам развертывать не только модели TensorFlow (включая Keras), но также модели Caffe2 и все модели машинного обучения, которые можно преобразовать в формат Open Neural Network Exchange (ONNX)¹. Используя формат ONNX, вы можете развернуть модели PyTorch с помощью GraphPipe.

Помимо предоставления сервера моделей для TensorFlow, PyTorch и т. д., GraphPipe также поддерживает реализацию клиентских приложений для различных языков программирования, таких как Python, Java и Go.

Simple TensorFlow Serving

Simple TensorFlow Serving (см. <https://stfs.readthedocs.io/en/latest/>) – разработка Дихао Чена (Dihao Chen) из 4Paradigm. Simple TensorFlow Serving поддерживает не только модели TensorFlow. Текущий список поддерживаемых интегрированных сред разработки моделей включает ONNX, Scikit-learn, XGBoost, PMML и H2O. Он поддерживает работу с несколькими моделями, прогнозы на графических процессорах и клиентский программный код на различных языках программирования.

Одним из важных аспектов простого обслуживания TensorFlow является то, что он поддерживает аутентификацию и зашифрованные соединения с сервером моделей. Аутентификация в настоящее время не предоставляется как функциональная возможность TensorFlow Serving, а поддержка SSL/TLS требует специальной сборки TensorFlow Serving.

MLflow

MLflow (см. <https://mlflow.org/>) поддерживает развертывание моделей машинного обучения, но это лишь одна из возможностей инструмента, созданного DataBricks. MLflow предназначен для управления экспериментами с моделями с помощью MLflow Tracking. В инструмент встроен сервер моделей, который предоставляет конечные точки REST API для моделей, управляемых через MLflow.

MLflow также предоставляет интерфейсы для прямого развертывания моделей из MLflow на платформе Microsoft AzureML и Amazon Web Service SageMaker.

Ray Serve

Ray Project (см. <https://twitter.com/raydistributed>) предоставляет функциональные возможности для развертывания моделей машинного обучения. Ray Serve не зависит от используемого фреймворка и поддерживает модели PyTorch, TensorFlow (включая Keras), Scikit-Learn или пользовательские прогнозы моделей. Библиотека предоставляет возможности пакетных запросов и позволяет маршрутизировать трафик между моделями и их версиями.

Ray Serve интегрирован в экосистему Ray Project и поддерживает установки распределенных вычислений.

¹ ONNX (см. <https://onnx.ai/>) – это способ описания моделей машинного обучения.

РАЗВЕРТЫВАНИЕ МОДЕЛЕЙ С ИСПОЛЬЗОВАНИЕМ УСЛУГ ПОСТАВЩИКОВ ОБЛАЧНЫХ РЕШЕНИЙ

Для всех серверных решений, предназначенных для создания моделей, которые мы обсуждали до этого момента, вы должны были самостоятельно выполнять установку и управлять ими. Однако все основные поставщики облачных услуг, такие как Google Cloud, Amazon Web Services (AWS) и Microsoft Azure, предлагают специализированные программные продукты для машинного обучения, включая размещение моделей машинного обучения.

В этой главе мы хотели бы познакомить вас с одним из примеров развертывания с использованием платформы искусственного интеллекта Google Cloud. Мы начнем с развертывания модели, а позже объясним, каким образом вы можете получать прогнозы на основе развернутой модели из клиентской части вашего приложения.

Сценарии использования

Управляемое развертывание моделей машинного обучения в облаке является хорошей альтернативой запуску экземпляров сервера моделей, если вы хотите беспрепятственно развернуть модель и не беспокоиться о ее масштабировании. Все поставщики облачных услуг предлагают варианты развертывания с возможностью масштабирования в зависимости от количества запросов на вывод.

Однако гибкость развертывания вашей модели обходится дорого. Управляемые сервисы обеспечивают легкое развертывание, но стоят дорого. Например, две версии модели, работающие полный рабочий день (требуются два вычислительных узла), стоят дороже, чем сопоставимый вычислительный узел, на котором запущен экземпляр TensorFlow Serving. Другим недостатком управляемого развертывания являются ограничения, связанные с использованием определенных программных продуктов и их компонентов. Некоторые поставщики облачных услуг требуют, чтобы развертывание выполнялось с использованием их собственных средств разработки программного обеспечения (Software Development Kits, SDK), у других есть ограничения на размер вычислительного узла и объем памяти, который может занимать ваша модель. Эти ограничения могут быть серьезным препятствием для развертывания масштабных моделей глубокого обучения, особенно если модели содержат очень много слоев (например, для языковых моделей).

Пример развертывания с помощью облачных платформ Google

В этом разделе мы рассмотрим пример развертывания на платформе искусственного интеллекта Google Cloud (Google Cloud's AI Platform, GCP). Вместо того чтобы писать файлы конфигурации и выполнять команды терминала, мы можем настроить конечные точки модели через веб-интерфейс.



Ограничения размера модели для платформы искусственного интеллекта GCP

Для конечных точек GCP существует ограничение размеров моделей до 500 МБ. Однако если вы развертываете свои конечные точки с помощью вычислительных машин типа *N1*, максимальный предел размеров модели увеличивается до 2 Гб. На момент написания этой книги такая возможность находилась в стадии бета-тестирования.

Развертывание модели

Развертывание состоит из трех этапов:

- обеспечение доступа к модели в Google Cloud;
- создание нового экземпляра модели с помощью платформы искусственного интеллекта Google Cloud's AI Platform;
- создание новой версии экземпляра модели.

Развертывание начинается с загрузки вашей экспортированной модели TensorFlow/Keras в хранилище. Как показано на рис. 8.5, вам необходимо загрузить всю экспортированную модель. После завершения загрузки модели скопируйте полный путь к месту хранения.

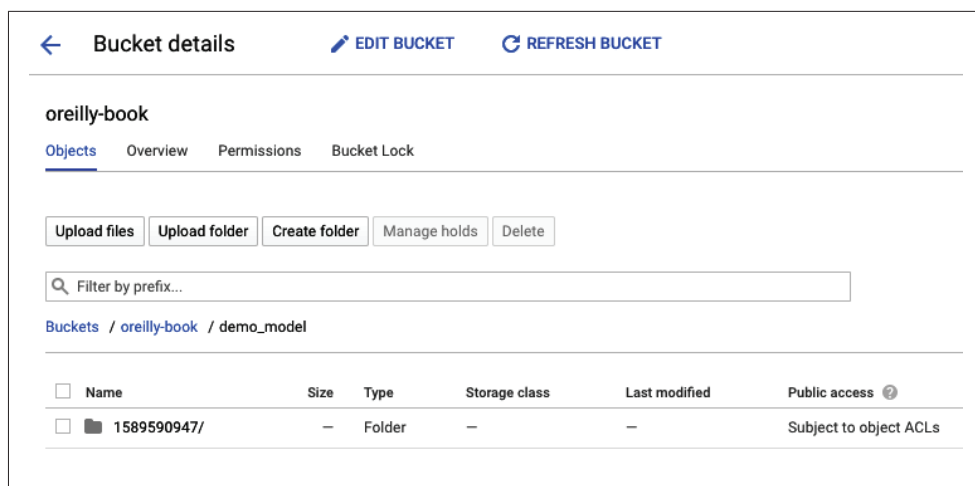


Рис. 8.5. Загрузка обученной модели в облачное хранилище

Загрузив модель машинного обучения, перейдите на платформу искусственного интеллекта Google Cloud (Google Cloud's AI Platform), чтобы настроить параметры конфигурации для развертывания модели машинного обучения. Если вы впервые используете облачную платформу искусственного интеллекта в своем проекте GCP, вам необходимо включить API. Процесс автоматического запуска Google Cloud может занять несколько минут.

Когда вы создаете новую модель, вам нужно присвоить модели уникальный идентификатор (см. рис. 8.6). Присвоив модели идентификатор, выбрав пред-

почитаемый регион развертывания¹ и введя описание проекта, нажмите кнопку **Create (Создать)**.

Create model

Model Name *
consumer_complaint_model

Name is permanent, is case-sensitive, must start with a letter, and must only contain letters, numbers and underscores. Model names must be unique within each project.
24 / 128

Region *
us-central1

Service availability may vary based on region and model framework. See available regions and services for [TensorFlow](#) and [XGBoost](#)

Description
Demo model for the book "Building ML Pipelines"

☒ Enable logging for this model ?

☐ Enable console logging for this model ?

i Logging settings are permanent for this model. To change your logging preference in the future, create a new model.

CREATE **CANCEL**

Рис. 8.6. Создание нового экземпляра модели

Как только новая модель зарегистрирована, вы можете создать новую версию модели внутри модели. Для этого раскройте на элементе интерфейса, представленном на рис. 6.7, меню overflow.

После регистрации новой модели информация о ней появится на информационной панели, как показано на рис. 8.7. Вы можете создать новую версию модели из информационной панели, щелкнув **Create version (Создать версию)** в дополнительном меню.

Filter by prefix...

<input type="checkbox"/>	Name	Default version	Description	Region	Labels	
<input type="checkbox"/>	consumer_complaint_model	-	Demo model for the book "Building ML Pipelines"	us-central1		<ul style="list-style-type: none"> View logs + Create version Delete

Рис. 8.7. Создание новой версии модели

¹ Чтобы минимизировать время задержки между отправкой данных и получением прогноза модели, выберите регион, ближайший к географическому региону, откуда поступают запросы к модели.

Когда вы создаете новую версию модели, вы настраиваете вычислительный узел, на котором работает ваша модель. Google Cloud предоставляет вам множество вариантов конфигурации (см. рис. 8.8). Важным элементом конфигурации является `version name`, так как вы будете ссылаться на `version name` позже при настройке клиента. Укажите в `Model URI` полный путь к месту хранения, который вы сохранили на предыдущем шаге.

Платформа Google Cloud AI Platform поддерживает различные среды машинного обучения, включая XGBoost и SciKit-Learn.

← Create version

To create a new version of your model, make necessary adjustments to your saved model file before exporting and store your exported model in Cloud Storage. [Learn more](#)

Name *
v1
Name cannot be changed, is case sensitive, must start with a letter, and may only contain letters, numbers, and underscores. 2 / 128

Description
First version of the demo model

Python version *
3.7
Select the Python version you used to train the model

Framework
TensorFlow

Framework version
2.1.0

ML runtime version *
2.1

Machine type *
Single core CPU

Model URI *
✓ gs://oreilly-book/demo_model/1589590947/ **BROWSE**
Cloud Storage path to the entire SavedModel directory. [Learn more](#)

Рис. 8.8. Определение параметров экземпляра

Google Cloud Platform также позволяет вам настраивать масштабирование экземпляра модели в случае, если ваша модель получает большое количество запросов на вывод. Вы можете настроить масштабирование, выбрав один из двух режимов масштабирования: *ручное масштабирование* или *автоматическое масштабирование*.

Ручное масштабирование дает вам возможность установить точное количество узлов, доступных для получения прогнозов вашей версии модели. Автоматическое масштабирование позволяет увеличивать и уменьшать количест-

во доступных узлов в зависимости от потребностей вашей конечной точки. Если ваши узлы не принимают никаких запросов, количество узлов может даже упасть до нуля. Обратите внимание, что если при автоматическом масштабировании количество узлов уменьшается до нуля, потребуется некоторое время для повторного создания версии вашей модели, когда следующий запрос будет получен на конечной точке вашей версии развернутой модели. Также, если вы запускаете узлы вывода прогнозов в режиме автоматического масштабирования, вам будут выставляться счета в 10-минутных интервалах. Это означает, что один запрос будет стоить вам как минимум 10 минут вычислительного времени.

После настройки всей версии модели Google Cloud запускает ваши экземпляры. Если все готово для работы модели, вы увидите зеленую галочку рядом с названием версии модели, как показано на рис. 8.9.

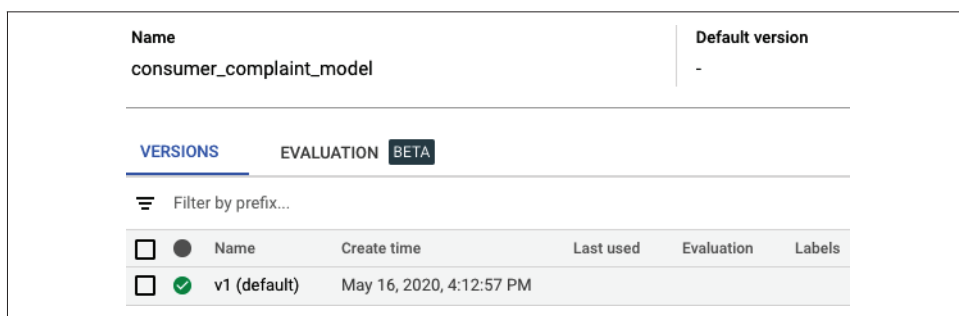


Рис. 8.9. Завершение развертывания новой версии (развернутая версия доступна)

Вы можете запускать несколько версий модели одновременно. В графической панели, отображающей версии модели, вы можете выбрать одну версию в качестве версии по умолчанию, и любой запрос на вывод прогноза без указания версии будет перенаправлен на «версию по умолчанию». Однако обратите внимание, что каждая версия модели будет размещена на отдельном узле, и затраты на GCP будут кумулятивно расти.

Вывод прогноза модели

Поскольку TensorFlow Serving проверен в Google в реальных рабочих условиях и интенсивно используется внутри компании, он также негласно применяется на облачной платформе Google. Вы заметите, что платформа искусственного интеллекта не просто использует тот же формат экспорта модели, который мы видели на наших экземплярах TensorFlow Serving, но передаваемые данные имеют ту же структуру, что и раньше.

Единственным существенным отличием является подключение API. Как вы увидите в этом разделе, вы будете подключаться к версии модели через GCP API, который обрабатывает запрос аутентификации.

Чтобы подключиться к Google Cloud API, вам нужно установить библиотеку `google-api-python-client` с помощью команды

```
$ pip install google-api-python-client
```

Все сервисы Google могут быть подключены через объект службы. Вспомогательная функция в следующем фрагменте кода показывает, как создать объект службы. Клиент Google API принимает `service name` и `service version` и возвращает объект, который предоставляет все функции API с помощью методов возвращаемого объекта:

```
import googleapiclient.discovery

def _connect_service():
    return googleapiclient.discovery.build(
        serviceName="ml", version="v1"
    )
```

Как и в наших предыдущих примерах, использующих REST и gRPC, мы размещаем наши выходные данные в фиксированном параметре `instances`, который содержит список входных словарей. Мы создали небольшую вспомогательную функцию для генерации пакетов полезных данных. Эта функция может выполнять любую предварительную обработку, если вам нужно изменить ваши входные данные перед выводом прогнозов.

```
def _generate_payload(sentence):
    return {"instances": [{"sentence": sentence}]}
```

После того как мы создали на стороне клиента объект службы и сгенерировали пакеты данных, мы можем запросить прогноз у модели машинного обучения, размещенной в Google Cloud.

Объект службы платформы искусственного интеллекта содержит метод `predict`, который принимает в качестве входных параметров `name` и `body`. Параметр `name` представляет собой строку пути, содержащую название проекта Google Cloud Platform, имя вашей модели и, если вы хотите получать прогнозы от определенной версии модели, название вашей версии. Если вы не укажете номер версии, для модели будет использоваться версия модели по умолчанию. Параметр `body` содержит структуру данных вывода прогноза, которую мы создали ранее.

```
project = "yourGCPProjectName"
model_name = "demo_model"
version_name = "v1"
request = service.projects().predict(
    name="projects/{}/models/{}/versions/{}".format(
        project, model_name, version_name),
    body=_generate_payload(sentence)
)
response = request.execute()
```

Ответ платформы Google Cloud AI Platform содержит прогнозные оценки для различных категорий, аналогичные ответу REST, получаемому от экземпляра TensorFlow Serving:


```
{'predictions': [
  {'label': [
    0.9000182151794434,
    0.02840868942439556,
    0.009750653058290482,
    0.06182243302464485
  ]}
]}
```

Продемонстрированный пример представляет собой быстрый способ развертывания модели машинного обучения без настройки всей инфраструктуры развертывания. Другие облачные провайдеры (например, AWS или Microsoft Azure) предлагают аналогичные услуги развертывания моделей машинного обучения. В зависимости от требований к развертыванию моделей машинного обучения облачные провайдеры могут быть хорошей альтернативой вариантам локального развертывания. Обратной стороной их использования являются потенциально более высокие затраты и отсутствие полной оптимизации конечных точек (благодаря возможности создания конечных точек gRPC или пакетных функций, как ранее обсуждалось в разделе «Пакетные запросы на вывод прогнозов модели» в этой главе).

РАЗВЕРТЫВАНИЕ МОДЕЛИ С ПОМОЩЬЮ КОНВЕЙЕРОВ TFX

Во введении к этой главе на рис. 8.1 мы представили шаги этапа развертывания как единых компонентов конвейера машинного обучения. После того как мы подробно обсудили все этапы развертывания моделей, и в особенности использования TensorFlow Serving, мы хотим составить полную картину, связав эти шаги с нашим конвейером машинного обучения в этом разделе.

На рис. 8.10 вы можете увидеть шаги, выполняемые при непрерывном развертывании модели. Мы предполагаем, что у нас работает служба TensorFlow, настроенная для загрузки моделей из заданного местоположения файла. Кроме того, мы предполагаем, что служба TensorFlow будет загружать модели из внешнего места размещения файлов (т. е. из сегмента облачного хранилища или смонтированного постоянного тома). Обе системы, конвейер TFX и экземпляр TensorFlow Serving, должны иметь доступ к одной и той же файловой системе.

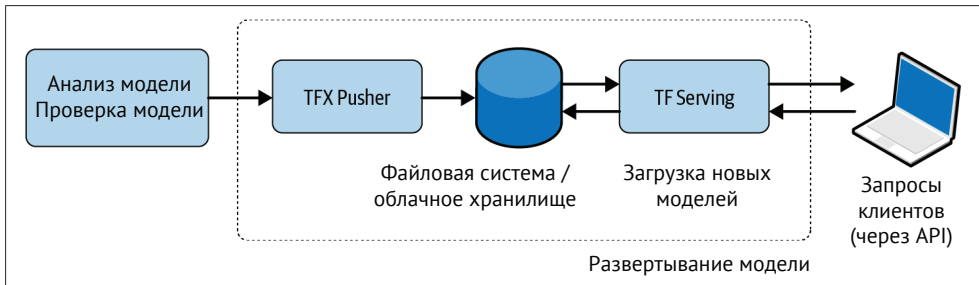


Рис. 8.10. Развертывание моделей из конвейеров TFX

В разделе «Компонент TFX Pusher» главы 7 мы обсуждали компонент Pusher. Этот компонент TFX позволяет нам отправлять проверенные модели в заданное место (например, в корзину облачного хранилища). TensorFlow Serving может получать новые версии моделей из облачного хранилища, выгружать более раннюю версию модели и загружать последнюю версию для данной конечной точки модели. Это политика по умолчанию для TensorFlow Serving по отношению к моделям.

Благодаря политике по умолчанию мы можем довольно легко создать простую настройку непрерывного развертывания с TFX и TensorFlow Serving.

РЕЗЮМЕ

В этой главе мы обсудили, как настроить TensorFlow Serving для развертывания моделей машинного обучения и почему сервер моделей является более масштабируемым вариантом, чем развертывание моделей машинного обучения через веб-приложение Flask. Мы пошагово прошли этапы установки и настройки, рассмотрели два основных варианта обмена данными, REST и gRPC, и кратко обсудили преимущества и недостатки обоих протоколов связи.

Кроме того, мы показали некоторые из больших преимуществ TensorFlow Serving, включая пакетную обработку запросов к модели и возможность получения метаданных о различных версиях модели. Мы также обсудили, как выполнить быструю настройку A/B-теста с помощью TensorFlow Serving.

Мы завершили эту главу кратким обзором управляемого развертывания модели в облачной службе на примере Google Cloud AI Platform. Управляемые облачные сервисы предоставляют вам возможность развертывать модели машинного обучения без развертывания и поддержки собственных экземпляров серверов.

В следующей главе мы обсудим, как улучшить наш опыт по развертыванию наших моделей, например загружая модели от облачных провайдеров или разворачивая TensorFlow Serving с использованием Kubernetes.

Глава 9

Расширенные концепции развертывания моделей с помощью TensorFlow Serving

В предыдущей главе мы обсудили вопросы развертывания моделей TensorFlow или Keras с использованием TensorFlow Serving. Познакомившись с основными приемами конфигурации TensorFlow Serving и развертывания модели, в этой главе мы представляем углубленные концепции и расширенные варианты развертывания модели машинного обучения. Сценарии использования затрагивают множество смежных тем: например, развертывание для A/B-тестирования модели, оптимизацию моделей для развертывания и масштабирования, а также мониторинг развертывания моделей. Если у вас не было возможности просмотреть предыдущую главу, мы рекомендуем это сделать, потому что она закладывает основы для данной главы.

РАЗДЕЛЕНИЕ ЗОН ОТВЕТСТВЕННОСТИ

В ПРОЦЕССЕ РАЗВЕРТЫВАНИЯ

Элементарные варианты развертывания, показанные в главе 8, работают хорошо, но у них есть одно ограничение: обученная и проверенная модель должна быть либо включена в образ контейнера развертывания на этапе сборки, либо смонтирована в контейнер во время выполнения контейнера, как мы обсуждали в предыдущей главе. Оба варианта требуют или знания процессов DevOps (например, обновления образов контейнеров Docker), или координации между командами специалистов по обработке и анализу данных и DevOps на этапе развертывания новой версии модели.

Как мы кратко упоминали в главе 8, TensorFlow Serving может загружать модели с удаленных устройств (например, из сегментов хранилища AWS S3 или GCP). Стандартная политика загрузчика TensorFlow Serving часто выполняет запросы к месту хранения модели, выгружает ранее загруженную модель и, если обнаружена новая модель, загружает ее. Используя такой сценарий загрузки, нам нужно развернуть обслуживающий контейнер нашей модели только

один раз, и он постоянно будет обновлять версии модели, как только они будут доступны в папке хранилища.

Обзор рабочего процесса

Прежде чем мы подробнее рассмотрим, как настроить TensorFlow Serving для загрузки моделей из удаленных хранилищ, давайте взглянем на предлагаемый нами рабочий процесс.

На рис. 9.1 показано разделение рабочих процессов. Контейнер для развертывания модели машинного обучения развертывается один раз. Инженеры и аналитики данных могут загружать новые версии моделей в сегменты хранилища либо через веб-интерфейс, либо с помощью операций копирования, выполняемых из командной строки. Любые изменения в версиях модели будут обнаружены экземплярами TensorFlow Serving. Новая сборка контейнера сервера моделей или повторное развертывание контейнера не требуется.

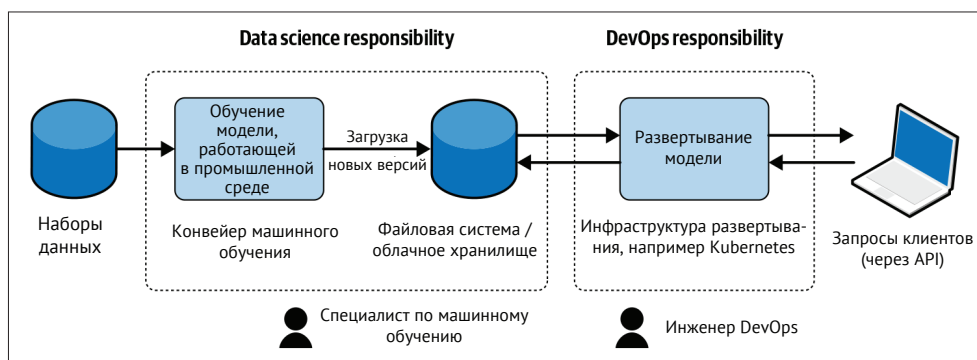


Рис. 9.1. Разделение зон ответственности в процессе развертывания между аналитиками данных и DevOps

Если ваши папки хранилища общедоступны, вы можете развертывать удаленные модели, просто изменив в коде путь к модели и указав путь к удаленному каталогу в качестве пути к модели:

```
docker run -p 8500:8500 \
  -p 8501:8501 \
  -e MODEL_BASE_PATH=s3://bucketname/model_path/ \ ❶
  -e MODEL_NAME=my_model \ ❷
  -t tensorflow/serving
```

- ❶ Путь к удаленному хранилищу
- ❷ Все остальные параметры конфигурации остаются прежними

Если ваши модели хранятся в сегментах частного облака, вам нужно определенным образом настроить TensorFlow Serving, предоставив учетные данные для доступа к месту хранения моделей. Конкретные параметры настройки зависят от поставщика. В этой главе мы рассмотрим два примера поставщиков: AWS и GCP.

Доступ к моделям из частного облачного хранилища AWS S3

AWS аутентифицирует пользователей с помощью пользовательского ключа доступа и «секрета» (объекта secret). Чтобы получить доступ к хранилищам частного облака AWS S3, вам необходимо создать ключ доступа пользователя и «секрет»¹.

Вы можете предоставить AWS ключ доступа и «секрет» через переменные среды для команды `docker run`. Это позволяет TensorFlow Serving получать учетные данные и доступ к хранилищам частного облака:

```
docker run -p 8500:8500 \
  -p 8501:8501 \
  -e MODEL_BASE_PATH=s3://bucketname/model_path/ \
  -e MODEL_NAME=my_model \
  -e AWS_ACCESS_KEY_ID=XXXXX \ ❶
  -e AWS_SECRET_ACCESS_KEY=XXXXX \
  -t tensorflow/serving
```

❶ Имя переменных среды – очень важная деталь

TensorFlow Serving полагается на стандартные переменные среды AWS и их значения по умолчанию. Вы можете изменить значения по умолчанию (например, если ваш сегмент не находится в регионе `us-east-1` или если вы хотите изменить конечную точку S3).

Доступны следующие варианты конфигурации:

- `AWS_REGION=us-east-1`;
- `S3_ENDPOINT=s3.us-east-1.amazonaws.com`;
- `S3_USE_HTTPS=1`;
- `S3_VERIFY_SSL=1`.

Параметры конфигурации могут быть добавлены как переменные среды или добавлены в `docker run`, как показано в следующем примере:

```
docker run -p 8500:8500 \
  -p 8501:8501 \
  -e MODEL_BASE_PATH=s3://bucketname/model_path/ \
  -e MODEL_NAME=my_model \
  -e AWS_ACCESS_KEY_ID=XXXXX \
  -e AWS_SECRET_ACCESS_KEY=XXXXX \
  -e AWS_REGION=us-west-1 \ ❶
  -t tensorflow/serving
```

❶ Дополнительные конфигурации могут быть добавлены через переменные среды

Благодаря этим нескольким дополнительным переменным среды, предоставленным TensorFlow Serving, вы теперь можете загружать модели из удаленных хранилищ AWS S3.

¹ Более подробную информацию об управлении ключами доступа AWS можно найти в документации, размещенной по ссылке https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html#Using_CreateAccessKey9.

Доступ к моделям из сегментов GCP

GCP аутентифицирует пользователей через *учетные записи служб*. Чтобы получить доступ к частным сегментам хранилища GCP, вам необходимо создать файл учетной записи службы¹.

В отличие от AWS, мы не можем просто предоставить учетные данные в переменной среды, поскольку аутентификация GCP ожидает файл JSON с учетными данными аккаунта сервиса. В случае GCP нам нужно смонтировать папку на хост-машине, содержащую учетные данные внутри контейнера Docker, а затем определить переменную среды, чтобы указать TensorFlow Serving на правильный файл учетных данных.

В следующем примере мы предполагаем, что вы сохранили недавно созданный файл учетных данных учетной записи службы в каталоге `/home/your_username/.credentials/` на вашем хост-компьютере. Мы загрузили учетные данные учетной записи службы из GCP и сохранили файл как `sa-credentials.json`. Вы можете дать файлу учетных данных любое имя, но вам нужно обновить переменную среды `GOOGLE_APPLICATION_CREDENTIALS`, указав полный путь внутри контейнера Docker:

```
docker run -p 8500:8500 \
  -p 8501:8501 \
  -e MODEL_BASE_PATH=gcp://bucketname/model_path/ \
  -e MODEL_NAME=my_model \
  -v /home/your_username/.credentials:/credentials/ ❶
  -e GOOGLE_APPLICATION_CREDENTIALS=/credentials/sa-credentials.json \ ❷
  -t tensorflow/serving
```

❶ Монтируется каталог хоста с учетными данными

❷ Указывается путь внутри контейнера

Выполнив эти шаги, вы настроили удаленный сегмент GCP в качестве хранилища, в котором размещены модели.

ОПТИМИЗАЦИЯ ЗАГРУЗКИ УДАЛЕННОЙ МОДЕЛИ

По умолчанию TensorFlow Serving опрашивает любую папку модели каждые 2 секунды, чтобы определить, были ли загружены обновленные версии модели, независимо от того, хранится модель в локальном или удаленном каталоге. Если ваша модель хранится в удаленном каталоге, операция опроса создает список доступных сегментов вашего облачного провайдера. Если вы постоянно обновляете версии своей модели, ваше хранилище может содержать большое количество файлов. Это приводит к появлению больших сообщений, содержащих список, и, следовательно, потребляет не слишком большой объем трафика, который, однако, со временем может стать источником дополнительных расходов. Ваш облачный провайдер, скорее всего, будет взимать плату за сетевой трафик, генерируемый этими операциями со списком. Чтобы избежать непри-

¹ Более подробную информацию о том, как создавать учетные записи служб и управлять ими, можно найти в документации, доступной по ссылке <https://cloud.google.com/iam/docs/creating-managing-service-accounts>.

ятных неожиданностей при оплате счетов облачного провайдера, мы рекомендуем снизить частоту опроса до одного раза в 120 секунд, что означает, что в час будет возможно получить до 30 потенциальных обновлений, но объем генерируемого при этом трафика будет в 60 раз меньше:

```
docker run -p 8500:8500 \
...
-t tensorflow/serving \
--file_system_poll_wait_seconds=120
```

Аргументы TensorFlow Serving необходимо добавить после спецификации образа команды `docker run`. Вы можете указать любое время ожидания опроса, превышающее 1 секунду. Если вы установите время ожидания равным нулю, TensorFlow Serving не будет пытаться обновить загруженную модель.

ОПТИМИЗАЦИЯ МОДЕЛИ ДЛЯ РАЗВЕРТЫВАНИЙ

С увеличением размера моделей машинного обучения оптимизация моделей начинает играть все более важное значение для эффективного развертывания. Квантование модели позволяет снизить сложность вычислений модели за счет уменьшения точности представления веса. Сокращение модели дает вам возможность неявно удалять ненужные веса, обнуляя их в вашей сети модели. А дистилляция модели заставит меньшую нейронную сеть использовать ответы более крупной нейронной сети.

Все три метода оптимизации нацелены на сокращение размера модели, при котором мы можем быстрее получить прогнозы модели. В следующих разделах мы более подробно рассмотрим три варианта оптимизации.

Квантование

Веса нейронной сети часто хранятся в виде 32-битных данных с плавающей запятой (или, как это называется в соответствии со стандартом IEEE 754, в виде данных двоичного формата с плавающей запятой с одинарной точностью). Числа с плавающей запятой хранятся следующим образом: 1 бит выделяется для хранения знака числа, 8 бит для порядка числа и 23 бита для указания значения числа с плавающей запятой с соответствующей точностью.

Однако веса сети могут быть представлены в формате чисел с плавающей запятой `bfloat16` – данный формат чисел с плавающей запятой занимает 16 бит в памяти компьютера – или в виде 8-битных целых чисел. Как мы видим на рис. 9.2, нам все еще требуется 1 бит для хранения знака числа. Показатель степени также по-прежнему представлен в виде 8 бит, когда мы сохраняем веса как `bfloat16` с плавающей запятой, потому что он используется TensorFlow. Однако размер дробного представления числа уменьшен с 23 до 7 бит. Иногда веса могут быть представлены как целые числа, и в таком случае используется только 8 бит.

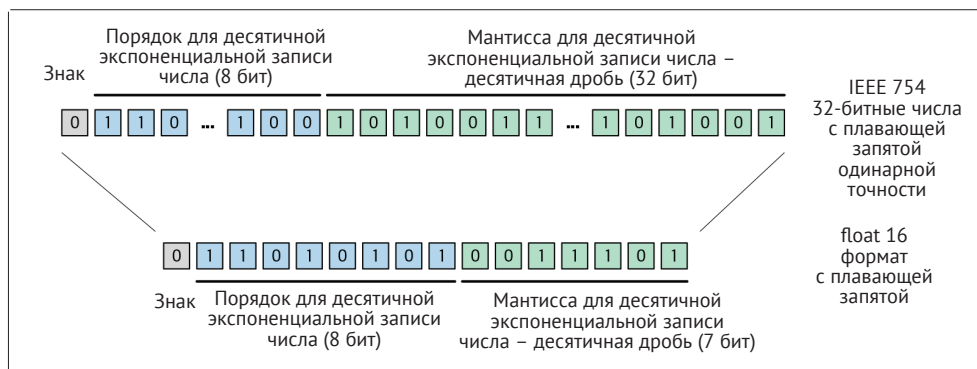


Рис. 9.2. Снижение точности чисел с плавающей запятой

Используя для представления веса сети 16-битные числа с плавающей запятой или целые числа, мы можем добиться следующих преимуществ:

- веса могут быть представлены меньшим количеством байтов, что потребует меньше памяти во время вывода модели;
- благодаря уменьшенному представлению весов прогнозы можно делать быстрее;
- квантование позволяет выполнять нейронные сети на 16-битных или даже 8-битных встроенных системах.

Текущие рабочие процессы для квантования модели применяются после обучения модели и часто называются *квантованием после обучения*. Поскольку квантованная модель может быть недостаточно качественной из-за отсутствия точности, мы настоятельно рекомендуем анализировать и проверять любую модель после квантования и перед развертыванием. В качестве примера квантования моделей мы рассматриваем библиотеку Nvidia TensorRT (см. раздел «Использование TensorRT совместно с TensorFlow Serving» этой главы) и библиотеку TFLite TensorFlow (см. раздел «TFLite» этой главы).

Сокращение

Альтернативой снижению точности весов сети является сокращение модели. Идея заключается в том, что обученная сеть может быть уменьшена до меньшей сети благодаря удалению всех ненужных весов. На практике это означает, что «ненужные» веса обнуляются. Обнулив ненужные веса, можно ускорить вывод получаемых от модели данных или получение прогноза. Кроме того, сокращенные модели можно сжимать до меньших размеров, поскольку малый вес приводит к более высокой степени сжатия.



Как сократить модели

Модели можно сокращать на этапе обучения с помощью таких инструментов, как пакет оптимизации моделей TensorFlow `tensorflow-model-optimisation`¹.

¹ Дополнительную информацию о методах оптимизации и подробные примеры сокращения вы можете найти на веб-сайте TensorFlow (см. <https://oreil.ly/n9rWc>).

Дистилляция

Вместо того чтобы сокращать количество сетевых подключений, мы можем также обучить меньшую, менее сложную нейронную сеть использовать предсказания, полученные в результате обучения гораздо более обширной сети. Такой подход называется дистилляцией. Вместо того чтобы непосредственно обучать меньшую модель машинного обучения на задаче, аналогичной большей модели, прогнозы, выдаваемые большей моделью (обучающей нейронной сетью), влияют на обновление весов меньшей модели (обучаемой нейронной сети), как показано на рис. 9.3. Используя прогнозы обучающей и обучаемой нейронных сетей, обучаемая сеть ученика может быть *вынуждена* «узнать» целевую функцию от обучающей нейронной сети. В конечном итоге мы можем выразить ту же целевую функцию модели, используя меньшее количество весов и архитектуру модели, которая не смогла бы получить требуемый результат, если бы обучающая сеть не обучила ее.

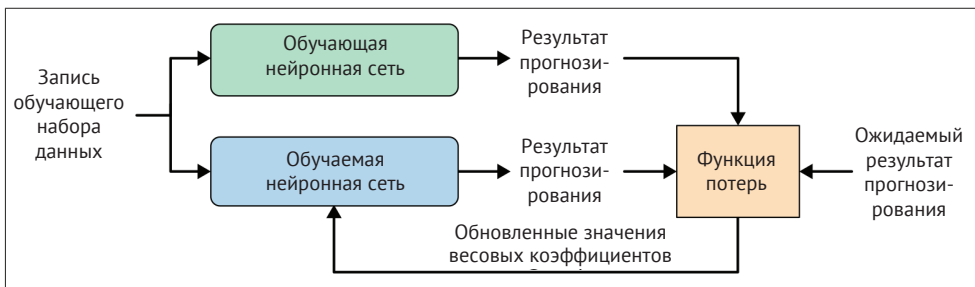


Рис. 9.3. Обучаемая сеть учится у обучающей сети

ИСПОЛЬЗОВАНИЕ TENSORRT СОВМЕСТНО С TENSORFLOW SERVING

Одним из способов выполнения квантования обученной модели TensorFlow перед ее развертыванием в производственной среде является преобразование модели с помощью библиотеки TensorRT от Nvidia.

Если вы запускаете модели глубокого обучения, требующие интенсивных вычислений на графическом процессоре Nvidia, то можете использовать этот дополнительный способ оптимизации своего сервера моделей. Nvidia предоставляет библиотеку TensorRT, которая оптимизирует вывод моделей глубокого обучения за счет снижения точности числовых представлений весов и смещений сети. TensorRT поддерживает представления int8 и float16. Уменьшение точности снизит задержку вывода результатов прогнозирования модели.

После обучения модели вам необходимо оптимизировать ее с помощью собственного оптимизатора TensorRT или с помощью `saved_model_cli` (см. документацию Nvidia для TensorRT, опубликованную по ссылке <https://oreil.ly/Ft8Y2>). Затем оптимизированную модель можно загрузить в TensorFlow Serving. На момент написания этой главы TensorRT поддерживал лишь некоторые процессоры в линейке продуктов Nvidia, включая Tesla V100 и P4.

Сначала преобразуем нашу модель глубокого обучения с помощью `saved_model_cli`:

```
$ saved_model_cli convert --dir saved_models/ \
                        --output_dir trt-savedmodel/ \
                        --tag_set serve tensorrt
```

После преобразования вы можете загрузить модель в TensorFlow Serving, сконфигурированный с настройками графического процессора, выполнив следующий фрагмент кода:

```
$ docker run --runtime=nvidia \
  -p 8500:8500 \
  -p 8501:8501 \
  --mount type=bind,source=/path/to/models,target=/models/my_model \
  -e MODEL_NAME=my_model \
  -t tensorflow/serving:latest-gpu
```

Если вы используете графические процессоры Nvidia для работы ваших моделей, то TenorRT поддерживает такие аппаратные средства. Переход на TensorRT может быть отличным способом еще больше снизить время ожидания логического вывода вашей модели.

TFLite

Если вы все же хотите оптимизировать модель машинного обучения, но не используете графические процессоры Nvidia, то можете использовать TFLite для оптимизации машинного обучения.

TFLite традиционно использовался для преобразования моделей машинного обучения в модели меньшего размера для развертывания на мобильных устройствах или IoT-устройствах. Однако эти модели также можно использовать с TensorFlow Serving. Поэтому вместо развертывания модели машинного обучения на пограничном устройстве вы можете развернуть модель машинного обучения с TensorFlow Serving, которая будет иметь низкую задержку вывода и требовать меньшего объема памяти.

Хотя оптимизация моделей с помощью TFLite выглядит очень привлекательно, есть несколько ограничений: на момент написания этого раздела поддержка TensorFlow Serving для моделей TFLite находится только на экспериментальной стадии. Более того, не все операции TensorFlow можно преобразовать в инструкции TFLite. Однако количество поддерживаемых операций неуклонно растет.

Шаги по оптимизации моделей машинного обучения с помощью TFLite

TFLite также можно использовать для оптимизации моделей TensorFlow и Keras. Эта библиотека предоставляет множество вариантов и инструментов оптимизации. Вы можете преобразовать свою модель с помощью инструментов командной строки или библиотеки Python.

Отправной точкой оптимизации всегда является обученная и экспортированная модель в формате SavedModel. В следующем примере мы будем использовать код на Python. Процесс конвертации состоит из четырех этапов:

- 1) загрузка экспортированной сохраненной модели;
- 2) определение целей оптимизации;
- 3) преобразование модели;
- 4) сохранение оптимизированной модели как модели TFLite.

```
import tensorflow as tf

saved_model_dir = "path_to_saved_model"
converter = tf.lite.TFLiteConverter.from_saved_model(
    saved_model_dir)

converter.optimizations = [
    tf.lite.Optimize.DEFAULT ❶
]
tflite_model = converter.convert()

with open("/tmp/model.tflite", "wb") as f:
    f.write(tflite_model)
```

❶ Выбор стратегии оптимизации

Оптимизация TFLite

TFLite предоставляет заранее определенные варианты оптимизации, преследующие различные цели. Выбирая вариант оптимизации, конвертер использует определенный способ оптимизации модели. Вот несколько вариантов оптимизации: DEFAULT, OPTIMIZE_FOR_LATENCY и OPTIMIZE_FOR_SIZE. В режиме DEFAULT ваша модель будет оптимизирована по задержке вывода результатов прогноза и по размеру, тогда как два других варианта выбирают одну из двух определенных целей. Вы можете установить параметры преобразования следующим образом:

```
...
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
converter.target_spec.supported_types = [tf.lite.constants.FLOAT16]
tflite_model = converter.convert()
...
```

Если ваша модель включает операцию TensorFlow, которая не поддерживается TFLite во время экспорта вашей модели, этап преобразования завершится ошибкой, и будет выдано сообщение об ошибке. Вы можете включить дополнительный набор выбранных операций TensorFlow, которые будут доступны для процесса преобразования. Однако это увеличит размер вашей модели TFLite приблизительно на 30 МБ. В следующем фрагменте кода показано, как включить дополнительные операции Tensor Flow перед выполнением преобразования:

```
...
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS,
                                       tf.lite.OpsSet.SELECT_TF_OPS]
tflite_model = converter.convert()
...
```

Если преобразование вашей модели по-прежнему завершается с ошибкой из-за неподдерживаемой операции TensorFlow, вы можете написать об этом в сообщество, развивающее TensorFlow. Сообщество активно увеличивает количество операций, поддерживаемых TFLite, и приветствует предложения по включению в TFLite расширенного набора операций. Форма обратной связи, позволяющая подать заявку на включение операции TFLite в список поддерживаемых TensorFlow операций, размещена по ссылке <https://oreil.ly/rPUqr>.

Развертывание моделей TFLite с помощью TensorFlow Serving

Последние версии TensorFlow Serving могут читать модели TFLite без необходимости каких-либо серьезных изменений конфигурации. Вам нужно только запустить TensorFlow Serving с включенным флагом `use_tflite_model`, и он загрузит оптимизированную модель, как показано в следующем примере:

```
docker run -p 8501:8501 \
  --mount type=bind,\
    source=/path/to/models,\
    target=/models/my_model \
  -e MODEL_BASE_PATH=/models \
  -e MODEL_NAME=my_model \
  -t tensorflow/serving:latest \
  --use_tflite_model=true ❶
```

❶ Включение флага, позволяющего загрузить модели TFLite

Оптимизированные модели TensorFlow Lite могут обеспечить развертывание моделей с малой задержкой вывода и малым объемом памяти.

Развертывание моделей машинного обучения на периферийных устройствах

После оптимизации модели TensorFlow или Keras и развертывания модели машинного обучения TFLite с помощью TensorFlow Serving вы также можете развернуть модель на различных мобильных и периферийных устройствах, таких как:

- мобильные телефоны Android и iOS;
- компьютеры на базе процессоров ARM64;
- микроконтроллеры и другие встроенные устройства (например, Raspberry Pi);
- конечные устройства (например, устройства IoT);
- тензорные процессоры (например, Coral).

Если вас интересуют варианты развертывания на мобильных или периферийных устройствах, мы рекомендуем ознакомиться с книгой «Практическое глубокое обучение для облачных хранилищ, мобильных устройств и периферийных устройств», авторства Анируда Коула и др., выпущенной издательством O'Reilly (*Practical Deep Learning for Cloud, Mobile, and Edge* by Anirudh Koul et al.). Если вы ищете материалы о периферийных устройствах с акцентом на TFMicro, мы рекомендуем книгу *TinyML* авторов Пита Уордена и Даниэля Ситунайке (*TinyML* by Pete Warden and Daniel Situnayake), также опубликованную O'Reilly.

Мониторинг экземпляров TensorFlow Serving

TensorFlow Serving позволяет отслеживать настройку логического вывода модели. Для этой задачи TensorFlow Serving предоставляет конечные точки API для сбора метрик, которые могут использоваться приложением Prometheus. Prometheus – это бесплатное приложение для регистрации событий и оповещения в реальном времени, лицензированное Apache License 2.0. Оно широко используется в сообществе Kubernetes, но его можно легко использовать и без Kubernetes.

Чтобы отслеживать метрики логического вывода модели, вам необходимо запустить TensorFlow Serving и Prometheus одновременно. Затем Prometheus можно настроить для непрерывного получения метрик из Tensor Flow Serving. Два приложения взаимодействуют через конечную точку REST, для чего требуется, чтобы конечные точки REST были включены в конфигурацию TensorFlow Serving, даже если в своем приложении вы используете только конечные точки gRPC.

Установка Prometheus

Перед настройкой TensorFlow Serving для вывода метрик в Prometheus нам необходимо выполнить установку и настройку нашего экземпляра Prometheus. Для простоты в этом примере мы запускаем два экземпляра Docker (TensorFlow Serving и Prometheus) вместе, как показано на рис. 9.4. При более сложной настройке приложения будут развертываться с помощью Kubernetes.

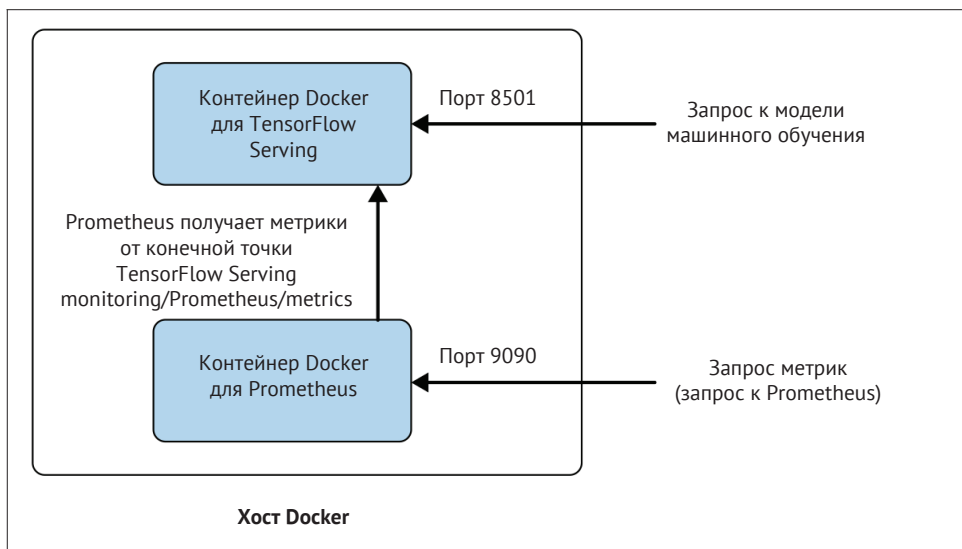


Рис. 9.4. Настройка Docker для Prometheus

Перед запуском Prometheus нам нужно создать файл конфигурации Prometheus. Для этого мы создадим файл конфигурации, расположенный в `/tmp/prometheus.yml`, и добавим в него следующие параметры конфигурации:

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
  external_labels:
    monitor: 'tf-serving-monitor'

scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s ❶
    metrics_path: /monitoring/prometheus/metrics ❷
    static_configs:
      - targets: ['host.docker.internal:8501'] ❸
```

- ❶ Интервал получения метрик
- ❷ Конечные точки метрик в TensorFlow Serving
- ❸ Значение этого параметра необходимо изменить, указав IP-адрес вашего приложения

В нашем примере конфигурации мы указали целевой хост `host.docker.internal`. Мы используем разрешение доменного имени Docker для доступа к контейнеру TensorFlow Serving через хост-компьютер. Docker автоматически преобразует доменное имя `host.docker.internal` в IP-адрес хоста.

После того как вы создали файл конфигурации Prometheus, вы можете запустить контейнер Docker, который запустит экземпляр Prometheus:

```
$ docker run -p 9090:9090 \ ❶
    -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml \ ❷
    prom/prometheus
```

❶ Включите порт 9090

❷ Смонтируйте файл конфигурации

В Prometheus имеется панель для отображения метрик, к которой мы позже будем обращаться через порт 9090.

Конфигурация TensorFlow Serving

Подобно нашей предыдущей конфигурации для пакетной обработки запросов на вывод прогнозов, нам нужно написать небольшой файл конфигурации для настройки параметров ведения журнала.

С помощью любого текстового редактора создайте текстовый файл, содержащий следующую конфигурацию (в нашем примере мы сохранили файл конфигурации в `/tmp/monitoring_config.txt`):

```
prometheus_config { enable: true,
path: "/monitoring/prometheus/metrics"
}
```

В файле конфигурации мы задаем путь в формате URL для данных метрик. Задаваемый путь должен соответствовать пути, который мы указали в ранее созданной конфигурации Prometheus (`/tmp/prometheus.yml`).

Чтобы включить функцию мониторинга, нам нужно только добавить путь для `monitoring_config_file`, и TensorFlow Serving предоставит конечной точке REST данные метрик для Prometheus:

```
$ docker run -p 8501:8501 \
    --mount type=bind,source=`pwd`,target=/models/my_model \
    --mount type=bind,source=/tmp,target=/model_config \
    tensorflow/serving \
    --monitoring_config_file=/model_config/monitoring_config.txt
```

Работа с Prometheus

Теперь, когда экземпляр Prometheus запущен, вы можете получить доступ к панели управления Prometheus для просмотра метрик TensorFlow Serving из пользовательского интерфейса Prometheus, как показано на рис. 9.5.

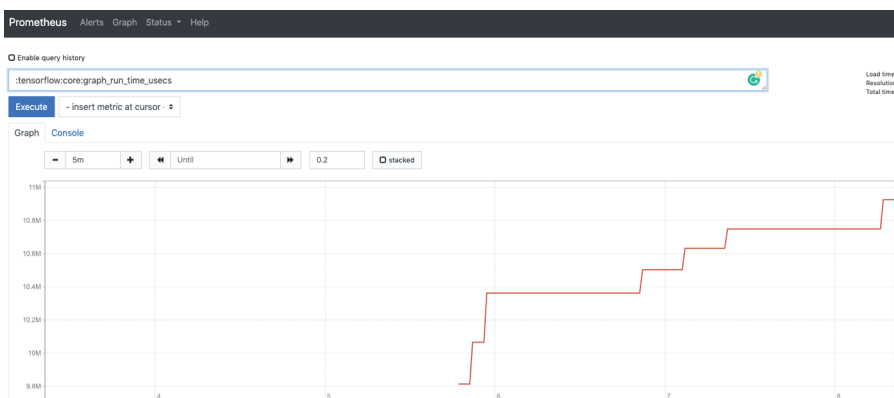


Рис. 9.5. Панель **Prometheus** для TensorFlow Serving

Prometheus предоставляет стандартизированный пользовательский интерфейс для общих метрик. Tensorflow Serving предоставляет множество вариантов метрик, включая количество запусков сессий, задержку загрузки или время выполнения определенного графа, как показано на рис. 9.6.

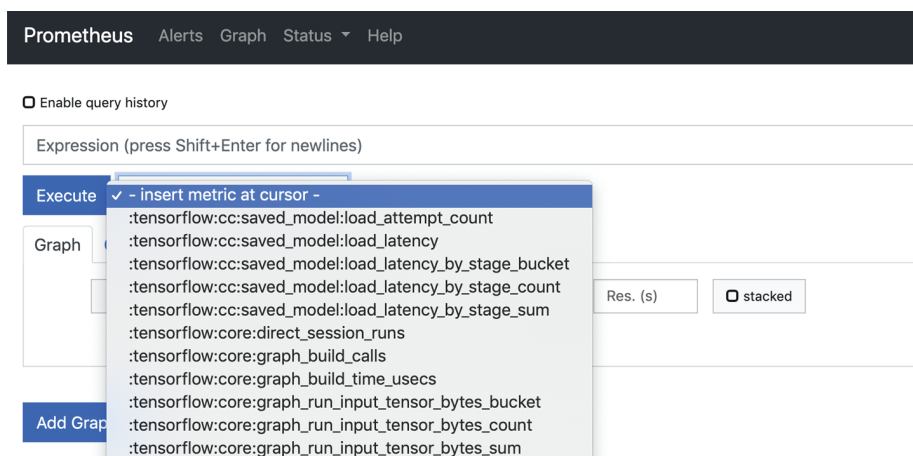


Рис. 9-6. Выбор метрик **Prometheus** для TensorFlow Serving

ПРОСТОЕ МАСШТАБИРОВАНИЕ С ПОМОЩЬЮ TENSORFLOW SERVING И KUBERNETES

До сих пор мы обсуждали развертывание одного экземпляра TensorFlow Serving, на котором размещена одна или несколько версий модели. Хотя это решение окажется достаточным для большинства развертываний, оно не будет достаточным для приложений с большим объемом запросов на прогнозирование. В этих ситуациях ваш единственный контейнер Docker с TensorFlow Serving необходимо реплицировать, чтобы ответить на избыточные запросы прогнозирования. Управление репликацией контейнера обычно осуществляется с помощью таких инструментов, как Docker Swarm или Kubernetes. Хотя подробное описание Kubernetes выходит за рамки данной публикации, мы хотели бы дать небольшой обзор того, как организовать развертывание вашего проекта машинного обучения при помощи Kubernetes.

В следующем примере мы используем GCP в качестве облачного провайдера для нашего развертывания¹:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: ml-pipelines
    name: ml-pipelines
spec:
  replicas: 1 ❶
  selector:
    matchLabels:
      app: ml-pipelines
  template:
    spec:
      containers:
        args:
          --rest_api_port=8501
          --model_name=my_model
          --model_base_path=gs://your_gcp_bucket/my_model ❷
        command:
          /usr/bin/tensorflow_model_server
      env:
        name: GOOGLE_APPLICATION_CREDENTIALS
        value: /secret/gcp-credentials/user-gcp-sa.json ❸
      image: tensorflow/serving ❹
      name: ml-pipelines
      ports:
        containerPort: 8501
      volumeMounts:
        mountPath: /secret/gcp-credentials ❺
        name: gcp-credentials
    volumes:
      name: gcp-credentials
      secret:
        secretName: gcp-credentials ❻
```

¹ Развертывание для AWS выполняется аналогично; вместо файла учетных данных вы должны предоставить параметры secret и key AWS как переменные среды.

- ❶ При необходимости можно увеличить число реплик
- ❷ Загрузка модели из удаленного каталога
- ❸ Здесь необходимо указать учетные данные облака для GCP
- ❹ Загрузка предварительно созданного образа TensorFlow Serving
- ❺ Монтирование файла учетных данных учетной записи службы (если кластер Kubernetes развернут через GCP)
- ❻ Загрузка файла учетных данных как тома

В этом примере мы теперь можем развертывать и масштабировать ваши модели TensorFlow или Keras без создания пользовательских образов Docker.

Вы можете создать файл учетных данных учетной записи службы в среде Kubernetes с помощью следующей команды:

```
$ kubectl create secret generic gcp-credentials \
  --from-file=/path/to/your/user-gcp-sa.json
```

Соответствующая настройка службы в Kubernetes для развертывания данной модели может выглядеть следующим образом:

```
apiVersion: v1
kind: Service
metadata:
  name: ml-pipelines
spec:
  ports:
    name: http
    nodePort: 30601
    port: 8501
  selector:
    app: ml-pipelines
  type: NodePort
```

С помощью нескольких строк кода конфигурации YAML теперь вы можете развернуть и, что наиболее важно, масштабировать свои развертывания машинного обучения. Для более сложных сценариев, таких как маршрутизация трафика к моделям машинного обучения, развернутым с помощью Istio, мы настоятельно рекомендуем углубиться в темы Kubernetes и Kubeflow.



Дополнительная литература о Kubernetes и Kubeflow

Kubernetes и Kubeflow – потрясающие инструменты DevOps, и мы не можем здесь дать развернутое представление о них. Эта тема требует отдельных публикаций. Если вы хотите ознакомиться с дополнительными материалами по этим темам, мы можем порекомендовать следующие публикации:

- Kubernetes: Up and Running, 2-е изд., авторы Брендан Бернс и др. (*Kubernetes: Up and Running*, 2nd edition by Brendan Burns et al. O'Reilly;

- Руководство по эксплуатации Kubeflow, авторы Джош Паттерсон и др. (*Kubeflow Operations Guide* by Josh Patterson et al. O'Reilly;
- Kubeflow для машинного обучения, авторы Холден Каррау и др. (готовится к выпуску в изд. O'Reilly) (*Kubeflow for Machine Learning* by Holden Karau et al).

РЕЗЮМЕ

В этой главе мы обсудили расширенные сценарии развертывания, такие как разделение жизненных циклов развертывания данных и моделей, с одной стороны, и DevOps – с другой, путем развертывания моделей в сегментах удаленного облачного хранилища, оптимизации моделей для уменьшения задержки прогнозирования и объема памяти модели, а также вопросы масштабирования при использовании сценариев развертывания.

В следующей главе мы хотим объединить все отдельные компоненты конвейера в один конвейер машинного обучения, чтобы обеспечить стабильные и воспроизводимые рабочие процессы машинного обучения.

Глава 10

Расширенные концепции TensorFlow Extended

В предыдущих двух главах, посвященных развертыванию моделей, мы завершили обзор отдельных компонентов конвейера. Прежде чем мы углубимся в тему оркестровки этих компонентов конвейера, мы хотим познакомить вас с расширенными концепциями TFX в данной главе.

С помощью компонентов конвейера, с которыми мы уже познакомились, мы можем создавать конвейеры машинного обучения для большинства типов задачи машинного обучения. Однако иногда нам нужно создать собственный компонент TFX или более сложные графы конвейера. Поэтому в этой главе мы сосредоточимся на том, как создавать собственные компоненты TFX. Мы рассмотрим эту тему на примере настраиваемого компонента загрузки, который загружает изображения, предназначенные для конвейеров машинного зрения. Кроме того, мы представим расширенные концепции структур конвейера: создание двух моделей одновременно (например, для развертываний с использованием TensorFlow Serving и TFLite), а также добавление действий человека, оценивающего выдаваемый моделью результат, в рабочий процесс конвейера.



Текущие разработки

На момент написания этой статьи некоторые из концепций, которые мы представляем здесь, все еще находятся в стадии разработки и, следовательно, могут быть изменены в будущем. Мы сделали все возможное, чтобы обновить примеры кода, которые затрагивают изменение функциональности TFX, во всей книге, и все приведенные нами примеры работают с версией TFX 0.22. Обновления API TFX можно найти в документации TFX (см. https://oreil.ly/POS_m).

РАСШИРЕННЫЕ КОНЦЕПЦИИ КОНВЕЙЕРОВ МАШИННОГО ОБУЧЕНИЯ

В этом разделе мы обсудим три дополнительные концепции, позволяющие усовершенствовать настройки вашего конвейера. До сих пор все концепции конвейеров, которые мы обсуждали, представляли собой линейные графы с одной

точкой входа и одной точкой выхода. В главе 1 мы познакомились с направленными ациклическими графами. До тех пор, пока наш граф конвейера представляет собой направленный граф и в нем отсутствуют «петли», мы можем проявить творческий подход к нашей настройке. В следующих разделах мы выделим несколько концепций повышения производительности конвейеров за счет:

- одновременного обучения нескольких моделей;
- экспорта моделей для мобильных вариантов развертывания;
- обучения модели с «теплым» запуском.

Одновременное обучение нескольких моделей

Как мы упоминали ранее, вы можете обучать несколько моделей одновременно. Обычный вариант использования для обучения нескольких моделей из одного и того же конвейера – это когда вы хотите обучить другой тип модели (например, упрощенную модель), но при этом хотите убедиться, что обученная модель использует те же самые преобразованные данные и тот же самый граф преобразования. На рис. 10.1 показано, как будет работать этот вариант конфигурации.

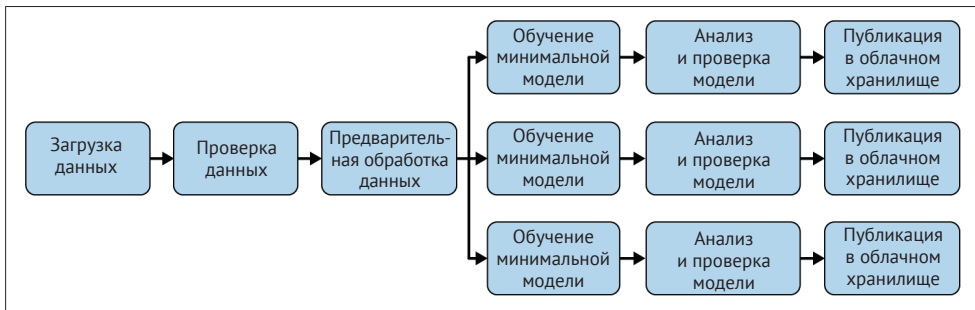


Рис. 10.1. Одновременное обучение нескольких моделей

Вы можете создать такой граф с помощью TFX, определив несколько компонентов `Trainer`, как показано в следующем примере кода:

```
def set_trainer(module_file, instance_name,
               train_steps=5000, eval_steps=100): ❶
    return Trainer(
        module_file=module_file,
        custom_executor_spec=executor_spec.ExecutorClassSpec(
            GenericExecutor),
        examples=transform.outputs['transformed_examples'],
        transform_graph=transform.outputs['transform_graph'],
        schema=schema_gen.outputs['schema'],

        train_args=trainer_pb2.TrainArgs(num_steps=train_steps),
        eval_args=trainer_pb2.EvalArgs(num_steps=eval_steps),
        instance_name=instance_name)

    prod_module_file = os.path.join(pipeline_dir, 'prod_module.py') ❷
    trial_module_file = os.path.join(pipeline_dir, 'trial_module.py')
    ...
```

```

trainer_prod_model = set_trainer(module_file, 'production_model') ❸
trainer_trial_model = set_trainer(trial_module_file, 'trial_model',
                                  train_steps=10000, eval_steps=500)
...

```

- ❶ Функция для эффективного создания экземпляра Trainer
- ❷ Загрузка файла модуля для каждого Trainer
- ❸ Создание экземпляра компонента Trainer для каждой ветви графа

На этом этапе мы в основном разветвляем граф на столько обучающих веток, сколько хотим запускать одновременно. Каждый из компонентов Trainer использует одни и те же входные данные от компонентов приема, схемы и преобразования. Ключевое различие между компонентами заключается в том, что каждый компонент может запускать различные настройки обучения, которые определены в отдельных файлах модуля обучения. Мы также добавили аргументы для шагов обучения и оценки в качестве параметра функции. Это позволяет нам обучать две модели с одной и той же настройкой обучения (то есть с одним и тем же файлом модуля), но мы можем сравнивать модели на основе разных прогонов обучения.

Каждый созданный экземпляр обучающего компонента должен использоваться своим собственным модулем оценки (Evaluator), как показано в следующем примере кода. После этого модели могут передаваться далее по цепочке конвейера с помощью компонентов Pusher:

```

evaluator_prod_model = Evaluator(
    examples=example_gen.outputs['examples'],
    model=trainer_prod_model.outputs['model'],
    eval_config=eval_config_prod_model,
    instance_name='production_model')

evaluator_trial_model = Evaluator(
    examples=example_gen.outputs['examples'],
    model=trainer_trial_model.outputs['model'],
    eval_config=eval_config_trial_model,
    instance_name='trial_model')

...

```

Как мы видим на примере материалов данного раздела, мы можем создать довольно сложные сценарии конвейера, используя TFX. В следующем разделе мы обсудим, как мы можем изменить настройку обучения для экспорта моделей для мобильных развертываний с TFLite.

Экспорт моделей TFLite

Мобильные платформы все чаще используются при развертывании моделей машинного обучения. Конвейеры машинного обучения могут облегчить процесс развертывания для мобильных приложений, в особенности в части согласованности моделей и данных при экспорте для мобильных развертываний. Для мобильного развертывания характерно очень малое количество изменений по сравнению с развертыванием на серверах моделей (например, на TensorFlow Serving, см. главу 8). Это помогает установить упорядоченный под-

ход к обновлению моделей, размещенных на мобильной платформе и на сервере, и способствует получению одинаковых результатов при работе пользователей с моделью при помощи различных устройств.



Ограничения TFLite

Из-за аппаратных ограничений, существующих для мобильных и периферийных устройств, TFLite не поддерживает все операции TensorFlow. Следовательно, не каждую модель можно преобразовать в модель, совместимую с TFLite. Дополнительная информация о поддерживаемых операциях TensorFlow опубликована на веб-сайте TFLite (см. <https://oreil.ly/LbDBK>).

В экосистеме TensorFlow TFLite – это решение для мобильных развертываний. TFLite – это версия TensorFlow, которую можно запускать на периферийных или мобильных устройствах. На рис. 10.2 показано, как конвейер может включать две обучающие ветви.

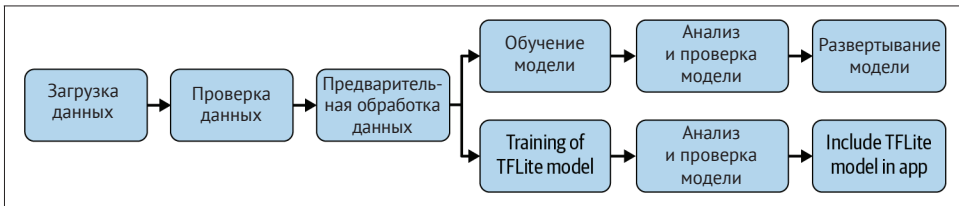


Рис. 10.2. Экспорт моделей для развертывания в мобильных приложениях

Мы можем использовать стратегию ветвления, которую обсуждали в предыдущем разделе, и изменить нашу функцию `run_fn` файла модуля таким образом, чтобы перевести сохраненные модели в формат, совместимый с TFLite.

В примере 10.1 показаны дополнительные функции, которые нам нужно добавить к нашей функции `run_fn`.

Пример 10.1. Пример TFX Rewriter

```
from tfx.components.trainer.executor import TrainerFnArgs from tfx.components.trainer.
rewriting import converters from tfx.components.trainer.rewriting import rewriter
from tfx.components.trainer.rewriting import rewriter_factory
```

```
def run_fn(fn_args: TrainerFnArgs):
    ...
    temp_saving_model_dir = os.path.join(fn_args.serving_model_dir, 'temp')
    model.save(temp_saving_model_dir,
               save_format='tf',
               signatures=signatures) ❶

    tfrw = rewriter_factory.create_rewriter(
        rewriter_factory.TFLITE_REWRITER,
        name='tflite_rewriter',
        enable_experimental_new_converter=True
    ) ❷
```

```
converters.rewrite_saved_model(temp_saving_model_dir, ❸
                               fn_args.serving_model_dir,
                               tfw,
                               rewriter.ModelType.TFLITE_MODEL)

tf.io.gfile.rmtree(temp_saving_model_dir) ❹
```

- ❶ Экспорт модели как сохраненной модели в требуемом формате
- ❷ Создание экземпляра TFLITE_REWRITER
- ❸ Преобразование модели в формат TFLite
- ❹ Удаление сохраненной модели после преобразования

Вместо того чтобы экспортировать сохраненную модель после обучения, мы преобразуем сохраненную модель в модель, совместимую с TFLite, и удаляем сохраненную модель после ее экспорта. Затем наш компонент `Trainer` экспортирует и регистрирует модель TFLite в хранилище метаданных. Последующие компоненты, такие как `Evaluator` или `Pusher`, могут затем использовать модель, совместимую с TFLite. В следующем примере показано, как мы можем оценить модель TFLite, определив, привели ли способы оптимизации модели (например, квантование) к снижению качества прогнозов модели:

```
eval_config = tfma.EvalConfig(
    model_specs=[tfma.ModelSpec(label_key='my_label', model_type=tfma.TF_LITE)],
    ...
)

evaluator = Evaluator(
    examples=example_gen.outputs['examples'],
    model=trainer_mobile_model.outputs['model'],
    eval_config=eval_config, instance_name='tflite_model')
```

С помощью представленной выше конфигурации конвейера мы теперь можем автоматически создавать модели для мобильного развертывания и размещать их в хранилищах артефактов, предназначенных для развертывания моделей в мобильных приложениях. Например, компонент `Pusher` может отправить созданную модель TFLite в облачное хранилище, где мобильный разрабочник может выбрать модель и развернуть ее с помощью Google ML Kit в мобильном приложении для iOS или Android.



Преобразование моделей в TensorFlow.js

Начиная с версии TFX 0.22 в TFX включена дополнительная возможность для библиотеки `rewriter_factory`. Ее назначение – преобразование существующих моделей TensorFlow в модели TensorFlow.js. Это преобразование позволяет развертывать модели в веб-браузерах и средах выполнения Node.js. Вы можете использовать эту новую возможность, заменив имя `rewriter_factory` на `rewriter_factory.TFJS_REWRITER` и установив для `rewriter.ModelType` значение `rewriter.ModelType.TFJS_MODEL`, как показано в примере 10.1.

Обучение модели с «теплым» запуском

В некоторых случаях мы можем не начинать обучение модели с нуля. «Теплый» запуск – это процесс начала обучения нашей модели с контрольной точки предыдущего цикла обучения, что особенно полезно, если модель большая и обучение занимает много времени. Это также может быть полезно в ситуациях, регламентируемых Общим регламентом ЕС по защите персональных данных (General Data Protection Regulation, GDPR), а также европейским законом о неприкосновенности данных, который гласит, что пользователь продукта может отозвать свое согласие на использование своих данных в любое время. Используя обучение с «теплым» запуском, мы можем удалить только данные, принадлежащие этому конкретному пользователю, и перенастроить модель, а не начинать обучение заново с нуля.

В конвейере TFX для обучения с «теплым» запуском используется компонент Resolver, о котором рассказывалось в главе 7. Resolver собирает подробные сведения о последней обученной модели и передает их компоненту Trainer:

```
latest_model_resolver = ResolverNode(
    instance_name='latest_model_resolver',
    resolver_class=latest_artifacts_resolver.LatestArtifactsResolver,
    latest_model=Channel(type=Model))
```

Затем последняя модель передается компоненту Trainer через аргумент `base_model`:

```
trainer = Trainer(
    module_file=trainer_file,
    transformed_examples=transform.outputs['transformed_examples'],
    custom_executor_spec=executor_spec.ExecutorClassSpec(GenericExecutor),
    schema=schema_gen.outputs['schema'],
    base_model=latest_model_resolver.outputs['latest_model'],
    transform_graph=transform.outputs['transform_graph'],
    train_args=trainer_pb2.TrainArgs(num_steps=TRAINING_STEPS),
    eval_args=trainer_pb2.EvalArgs(num_steps=EVALUATION_STEPS))
```

Потом конвейер продолжает работу в обычном режиме.

Далее мы хотим представить еще одну полезную функцию, которую можем добавить в наш конвейер.

УЧАСТИЕ ЧЕЛОВЕКА В КОНВЕЙЕРЕ МАШИННОГО ОБУЧЕНИЯ

В рамках расширенных концепций TFX мы хотим выделить экспериментальный компонент, который может улучшить настройку вашего конвейера. Все конвейеры, которые мы обсуждали до сих пор, работают автоматически от начала до конца и могут автоматически развернуть вашу модель машинного обучения. Некоторые пользователи TFX выразили обеспокоенность по поводу полностью автоматизированной работы, поскольку они хотели, чтобы человек проверил обученную модель после автоматического анализа модели. Это может быть выборочная проверка вашей обученной модели или инспекция, придающая уверенность в автоматической настройке конвейера.

В этом разделе мы обсудим функциональность компонента «человек в петле конвейера». В главе 7 мы обсуждали, что когда модель проходит этап валидации, она становится «одобренной». Компонент Pusher, находящийся далее в цикле конвейера, получает сигнал «одобрения», чтобы решить, следует переводить модель на следующий этап работы конвейера или нет. Но одобрить модель может и человек, как показано на рис. 10.3.

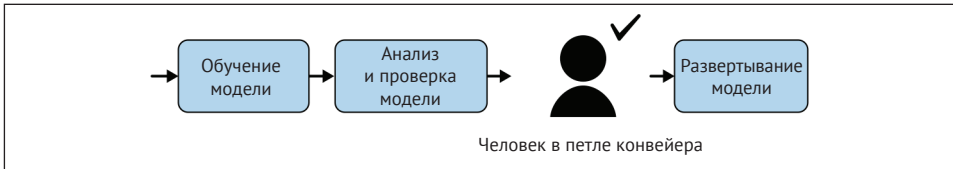


Рис. 10.3. Человек в петле конвейера

Команда Google TFX опубликовала компонент уведомления Slack в качестве примера использования этого настраиваемого компонента. Функциональность компонента, которую мы обсуждаем в этом разделе, может быть расширена и не ограничивается мессенджером Slack.

Эта функциональность довольно проста. Как только он запускается инструментом оркестрации, он отправляет в определенный канал Slack сообщение со ссылкой на последнюю экспортированную модель и запрашивает проверку у специалиста по данным (как показано на рис. 10.4). Специалист по данным теперь может исследовать модель вручную с помощью WIT, обращая особое внимание на пограничные случаи, которые не тестировались на этапе оценки модели.

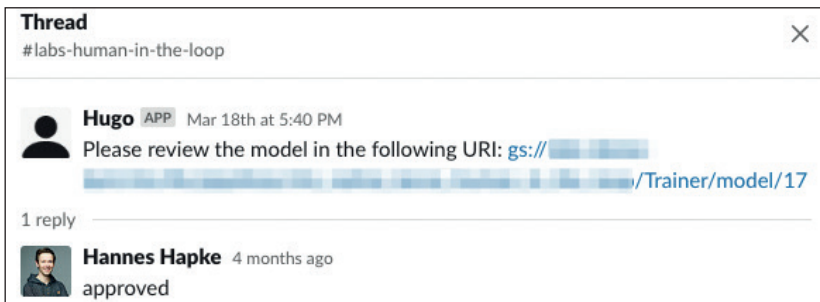


Рис. 10.4. Сообщение Slack с просьбой о проверке модели

После того как специалист по анализу данных завершит ручной анализ модели, он может написать в ветке Slack ответное сообщение с одобрением или отказом. Компонент TFX прослушивает ответы в Slack и сохраняет решение в хранилище метаданных. Затем решение может быть использовано нижестоящими компонентами. Эти решения регистрируются в журнале аудита модели. На рис. 10.5 показан пример записи из браузера Kubeflow Pipeline. В хранилище метаданных регистрируется одобрение модели специалистом по данным (то есть лицом, принимающим решение) и метка времени (идентификатор потока Slack 1584638332.0001 идентифицирует метку времени как время в формате времени Unix).

Type: ModelBlessing

URI

[gs://\[redacted\]_human_in_the_loop/SlackComponent/slack_blessing/39](gs://[redacted]_human_in_the_loop/SlackComponent/slack_blessing/39)**Properties****Custom Properties**

blessed 1	name slack_blessing	pipeline_name [redacted]_human_in_the_loop
slack_decision_maker WQ[redacted]	slack_decision_message approve	slack_decision_thread 1584638332.0001

Рис. 10.5. Контрольный журнал в Kubeflow Pipelines

Настройка компонента Slack

Чтобы компонент Slack мог взаимодействовать с вашей учетной записью Slack, ему требуется токен бота Slack. Вы можете запросить токен бота через Slack API. Получив токен, установите переменную среды в среде конвейера с помощью строки токена, как показано в следующей команде bash:

```
$ export SLACK_BOT_TOKEN={your_slack_bot_token}
```

Компонент Slack не является стандартным компонентом TFX, поэтому его необходимо устанавливать отдельно. Вы можете установить компонент, клонировав репозиторий TFX из GitHub, а затем отдельно установив данный компонент:

```
$ git clone https://github.com/tensorflow/tfx.git
```

```
$ cd tfx/tfx/examples/custom_components/slack
```

```
$ pip install -e .
```

После того как пакет компонентов будет установлен в вашей среде Python, этот компонент можно будет найти в пути Python и загрузить в ваши сценарии TFX. Пример такого сценария показан в следующем коде на языке Python. Также не забудьте установить компонент Slack в среде, в которой вы запускаете конвейеры TFX. Например, если вы запускаете свои конвейеры с помощью Kubeflow Pipelines, вам нужно будет создать собственный образ Docker для вашего компонента конвейера, который содержит исходный код компонента Slack (поскольку он не является стандартным компонентом TFX).

Как использовать компонент Slack

Установленный компонент Slack можно загрузить как любой другой компонент TFX:

```
from slack_component.component import SlackComponent

slack_validator = SlackComponent(
    model=trainer.outputs['model'],
    model_blessing=model_validator.outputs['blessing'],
```

```
slack_token=os.environ['SLACK_BOT_TOKEN'], ❶
slack_channel_id='my-channel-id', ❷
timeout_sec=3600,
)
```

- ❶ Загрузка токена Slack из вашей среды
- ❷ Выбор канала, на котором должно появиться сообщение

После запуска на выполнение компонент отправит сообщение и будет ждать ответа в течение часа (определяемого аргументом `timeout_sec`). В течение этого времени специалист по данным может оценить модель и написать ответное сообщение с одобрением или отказом. Следующий компонент конвейера (например, компонент `Pusher`) может использовать результат, выдаваемый компонентом Slack, как показано в следующем примере кода:

```
pusher = Pusher(
    model=trainer.outputs['model'],
    model_blessing=slack_validator.outputs['slack_blessing'], ❶
    push_destination=pusher_pb2.PushDestination(
        filesystem=pusher_pb2.PushDestination.Filesystem(
            base_directory=serving_model_dir)))
```

- ❶ Одобрение модели, передаваемое компонентом Slack

С помощью нескольких дополнительных шагов вы можете включить в свои конвейеры действия человека, выполняющего аудит моделей машинного обучения, который запускается самим конвейером. Это открывает гораздо больше возможностей для конвейерных приложений (например, для аудита статистики набора данных или анализа показателей смещения данных).



Стандарты Slack API

Реализация компонента Slack основана на протоколе обмена сообщениями в реальном времени (RTM). Этот протокол устарел и может быть заменен новым стандартом протокола, что повлияет на функциональность компонента.

Пользовательские компоненты TFX

В главе 2 мы обсудили архитектуру компонентов TFX и их структуру: как мы видели, каждый компонент состоит из трех элементов: `driver`, `executor` и `publisher`. В этом разделе мы хотим рассмотреть эту тему глубже и показать, как можно создавать свои собственные компоненты. Сначала мы обсудим, как написать компонент с нуля, а затем обсудим, как повторно использовать существующие компоненты и настроить их для ваших собственных сценариев использования. В общем случае всегда легче изменить функциональность существующего компонента, чем разработать компонент с нуля.

Чтобы продемонстрировать вам процесс реализации собственного компонента, как показано на рис. 10.6, мы разработаем специальный компонент для загрузки изображений JPEG и их меток в конвейер. Мы загрузим все изображения из определенного каталога и определим метку на основе имени файла.

В нашем примере мы хотим обучить модель машинного обучения для распознавания кошек и собак. Имена файлов наших изображений включают описание содержимого изображения (например, *dog-1.jpeg*), поэтому мы можем определить метку непосредственно на основании самого имени файла. Мы загрузим каждое изображение, преобразуем его в структуры данных `tf.Example` и сохраним все образцы как файлы `TFRecord` для использования нижестоящими компонентами.

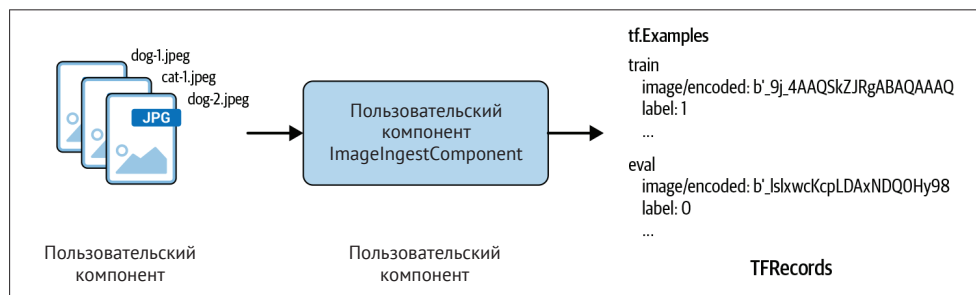


Рис. 10.6. Функциональность пользовательского компонента, разрабатываемого для демонстрационного проекта

Сценарии использования пользовательских компонентов

Несмотря на то что мы обсуждаем компонент загрузки данных в качестве примера для пользовательского компонента, при создании пользовательских компонентов у вас нет ограничений с точки зрения архитектуры. Пользовательский компонент можно включить в любом месте конвейера машинного обучения. Концепции, обсуждаемые в следующих разделах, обеспечивают максимальную гибкость для настройки конвейеров машинного обучения в соответствии с вашими потребностями. Вот несколько возможных сценариев применения пользовательских компонентов:

- получение данных из пользовательской базы данных;
- отправка электронной почты с полученными данными статистики команде специалистов по анализу данных;
- уведомление команды DevOps в случае наступления определенного события – например, события, когда новая модель была экспортирована;
- запуск процесса сборки после экспорта для контейнеров Docker;
- отслеживание дополнительной информации в журнале аудита машинного обучения.

Мы не будем описывать, как создать каждый из этих компонентов по отдельности, но если какая-либо из этих идей окажется для вас полезной, то, ознакомившись со следующими разделами, вы сможете реализовать ваши собственные компоненты.

Создание пользовательского компонента с нуля

Если мы хотим разработать собственный компонент с нуля, нам нужно будет реализовать несколько компонентов. Во-первых, мы должны определить входы и выходы нашего компонента как `ComponentSpec`. Затем мы можем указать

исполнителя нашего компонента, который определяет, как должны обрабатываться входные данные и как генерируются выходные данные. Если компоненту требуются входные данные, которые не зарегистрированы в хранилище метаданных, нам нужно будет написать драйвер для вашего компонента. Это тот случай, когда, например, мы хотим зарегистрировать путь к изображению в компоненте, а соответствующий артефакт не был ранее зарегистрирован в хранилище метаданных.

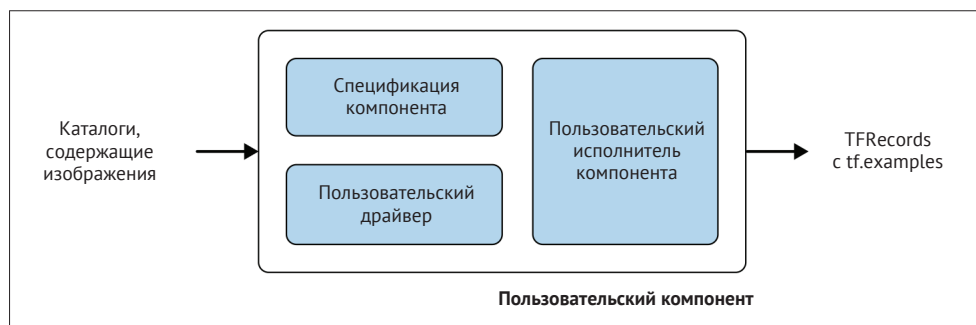


Рис. 10.7. Элементы нашего пользовательского компонента

Шаги на рис. 10.7 могут показаться сложными, но мы обсудим каждый из них по очереди в следующих разделах.



Попробуйте повторно использовать компоненты

Если вы думаете об изменении функциональности существующего компонента TFX, рассмотрите возможность повторного использования существующих компонентов TFX и смены исполнителя, вместо того чтобы начинать с нуля. Повторное использование компонентов TFX мы обсудим в разделе «Повторное использование существующих компонентов» далее в этой главе.

Спецификации компонентов

Спецификации компонентов, или `ComponentSpec`, определяют, как компоненты взаимодействуют друг с другом. Они описывают три важные детали каждого компонента: входные данные компонента, выходные данные компонента и потенциальные параметры компонента, которые используются во время выполнения компонента. Компоненты взаимодействуют через каналы, которые являются входами и выходами. Эти каналы являются типами, как мы увидим в следующем примере. Входные данные компонента определяют артефакты, которые компонент получит от ранее выполненных компонентов, или новые артефакты, такие как пути к файлам. Выходные данные компонента определяют, какие артефакты будут зарегистрированы в хранилище метаданных.

Параметры компонента определяют параметры, необходимые для выполнения, но недоступные в хранилище метаданных. Это может быть, например, `push_destination` для компонента `Pusher` или `train_args` для компонента `Trainer`. В следующем примере показано определение спецификаций компонента для нашего компонента приема изображений:

```

from tfx.types.component_spec import ChannelParameter
from tfx.types.component_spec import
ExecutionParameter from tfx.types import
standard_artifacts

class ImageIngestComponentSpec(types.ComponentSpec):
    «»»Спецификация компонента для пользовательского компонента загрузки изображения.»»»
    PARAMETERS = {
        'name': ExecutionParameter(type=Text),
    }
    INPUTS = {
        'input': ChannelParameter(type=standard_artifacts.ExternalArtifact), ❶
    }
    OUTPUTS = {
        'examples': ChannelParameter(type=standard_artifacts.Examples), ❷
    }

```

- ❶ Использование ExternalArtifact для указания новых путей ввода
- ❷ Экспорт Examples

В нашем примере реализации ImageIngestComponentSpec мы указываем путь для входных данных через входной аргумент input. Сгенерированные файлы TFRecord с преобразованными изображениями будут сохранены в каталоге, размещенном по указанному пути, переданному нижестоящим компонентам с помощью аргумента examples. Кроме того, мы определяем параметр с именем name для нашего компонента.

Каналы компонентов

В нашем примере ComponentSpec мы представили два типа каналов компонентов: ExternalArtifact и Examples. Это особый шаблон, используемый для компонентов загрузки данных, поскольку они обычно являются первым компонентом в конвейере, и не существует предшествующего компонента в конвейере, из которого мы могли бы получить уже обработанные образцы. Если вы разрабатываете компонент на более позднем этапе конвейера, то можете загрузить Examples. Следовательно, должен использоваться тип канала standard_artifacts.Examples. Но мы не ограничиваемся только двумя типами. TFX предоставляет множество типов. Ниже приведен краткий список доступных типов:

- ExampleStatistics;
- Model;
- ModelBlessing;
- Bytes;
- String;
- Integer;
- Float;
- HyperParameters.

Теперь, когда наш пример ComponentSpec настроен, давайте рассмотрим управляющую программу (исполнителя) компонента.

Исполнители компонентов

Исполнитель компонента определяет процессы внутри компонента, включая то, как входные данные используются для генерации выходных данных компонента. Несмотря на то что мы напишем этот элементарный компонент с нуля, мы можем полагаться на классы TFX для наследования шаблонов функций. Как часть объекта `Executor`, TFX будет искать функцию с именем `Do` для получения инструкций выполнения нашего компонента. Мы реализуем функциональность нашего компонента в этой функции:

```
from tfx.components.base import base_executor

class Executor(base_executor.BaseExecutor):
    «»«Объект Executor для компонента загрузки изображений.»»»

    def Do(self, input_dict: Dict[Text, List[types.Artifact]],
          output_dict: Dict[Text, List[types.Artifact]],
          exec_properties: Dict[Text, Any]) -> None:
        ...
```

Приведенный фрагмент кода показывает, что функция `Do` нашего объекта `Executor` принимает на вход три аргумента: `input_dict`, `output_dict` и `exec_properties`. Эти словари Python содержат ссылки на артефакты, которые мы передаем компоненту и из него, а также свойства выполнения.



Артефакты, содержащие ссылки

Данные, предоставленные посредством `input_dict` и `output_dict`, содержат информацию, хранящуюся в хранилище метаданных. Это ссылки на артефакты, а не на сами данные. Например, наш словарь `input_dict` будет содержать буфер протокола с информацией о местоположении файла, а не сами данные. Это позволяет нам эффективно обрабатывать данные с помощью специальных приложений, таких как Apache Beam.

Чтобы показать вам шаг за шагом процесс реализации рабочего метода `Do` исполнителя компонента, мы будем повторно использовать реализацию, которую обсуждали в разделе «Графические данные для задач компьютерного зрения» в главе 3, предназначенную для преобразования изображений в структуры данных `TFRecord`. Эта глава содержит объяснение процесса преобразования и подробную информацию о структурах данных `TFRecord`.

Следующий код должен выглядеть для вас знакомо:

```
def convert_image_to_TFExample(image_filename, tf_writer, input_base_uri):
    image_path = os.path.join(input_base_uri, image_filename) ❶
    lowered_filename = image_path.lower() ❷
    if "dog" in lowered_filename:
        label = 0
    elif "cat" in lowered_filename:
        label = 1
    else:
        raise NotImplementedError("Found unknown image")
```



```

raw_file = tf.io.read_file(image_path) ❸
example = tf.train.Example(features=tf.train.Features(feature={ ❹
    'image_raw': _bytes_feature(raw_file.numpy()),
    'label': _int64_feature(label)
}))
writer.write(example.SerializeToString()) ❺

```

- ❶ Задание полного пути к изображению
- ❷ Определение метки для каждого изображения, за основу взят путь к файлу
- ❸ Чтение изображения с диска
- ❹ Создание структуры данных TensorFlow Example
- ❺ Запись tf.Example в файлы TFRecord

Завершив таким образом разработку универсальной функции, выполняющей чтение файла изображения и сохранение его в виде файлов, содержащих структуры данных TFRecord, мы можем сосредоточиться на пользовательском коде для конкретного компонента.

Мы хотим, чтобы наш простейший компонент выполнял загрузку наших изображений, преобразовывал их в tf.Examples и возвращал два набора изображений: обучающий набор данных и оценочный набор. Для простоты реализации мы жестко закодируем число оценочных примеров. В промышленной реализации компонента этот параметр должен динамически задаваться с помощью отдельного параметра в ComponentSpecs. Входными данными для нашего компонента будет путь к каталогу, содержащему все изображения. Выходными данными нашего компонента будет путь к каталогу, в котором мы будем хранить обучающие и оценочные наборы данных. Путь к каталогу будет содержать два подкаталога (train и eval), в которых размещены файлы TFRecord:

```

class ImageIngestExecutor(base_executor.BaseExecutor):

    def Do(self, input_dict: Dict[Text, List[types.Artifact]],
        output_dict: Dict[Text, List[types.Artifact]],
        exec_properties: Dict[Text, Any]) -> None:

        self._log_startup(input_dict, output_dict, exec_properties) ❶

        input_base_uri = artifact_utils.get_single_uri(input_dict['input']) ❷
        image_files = tf.io.gfile.listdir(input_base_uri) ❸
        random.shuffle(image_files)
        splits = get_splits(images)

        for split_name, images in splits:
            output_dir = artifact_utils.get_split_uri(
                output_dict['examples'], split_name) ❹

            tfrecord_filename = os.path.join(output_dir, 'images.tfrecord')
            options = tf.io.TFRecordOptions(compression_type=None)
            writer = tf.io.TFRecordWriter(tfrecord_filename, options=options) ❺
            for image in images:
                convert_image_to_TFExample(image, tf_writer, input_base_uri) ❻

```

- ❶ Запись аргументов
- ❷ Получение пути к каталогу из артефакта
- ❸ Чтение всех имен файлов
- ❹ Задание унифицированного идентификатора ресурса (Uniform Resource Identifier, URI) для подмножества данных
- ❺ Создание экземпляра записи TFRecord с параметрами
- ❻ Запись изображения в файл, содержащий структуры данных TFRecord

Наш базовый метод `Do` получает `input_dict`, `output_dict` и `exec_properties` в качестве аргументов метода. Первый аргумент содержит ссылки на артефакты из хранилища метаданных, хранящихся в виде словаря Python, второй аргумент получает ссылки, которые мы хотим получить из компонента, а последний аргумент метода содержит дополнительные параметры, такие как, в нашем случае, имя компонента. TFX предоставляет очень полезную функцию `artifact_utils`, которая позволяет нам обрабатывать нашу информацию об артефактах. Например, мы можем написать следующий код, чтобы получить путь, используемый для загрузки данных:

```
artifact_utils.get_single_uri(input_dict['input'])
```

Мы также можем установить имя пути вывода на основе имени подмножества:

```
artifact_utils.get_split_uri(output_dict['examples'], split_name
```

Последняя из упомянутых функций – хороший пример. Для простоты нашего примера мы проигнорировали параметры установки динамического разбиения наборов данных, о котором рассказывалось в главе 3. Фактически в нашем примере мы жестко кодируем имена и количество разбиений:

```
def get_splits(images: List, num_eval_samples=1000):
    «»» Разбиение списка имен файлов изображений на списки train/eval «»»
    train_images = images[num_test_samples:]
    eval_images = images[:num_test_samples]
    splits = [('train', train_images), ('eval', eval_images)]
    return splits
```

Подобная реализация нежелательна для компонента, развернутого в промышленной среде; однако его полноценная реализация выходит за рамки этой главы. В следующем разделе мы обсудим, как можно повторно использовать существующие функции компонентов и упростить реализацию. Наш компонент, который мы создали в этом разделе, будет иметь функциональность, аналогичную той, которую мы обсуждали в главе 3.

Драйверы компонентов

Если мы запустим компонент с модулем исполнения, который мы разрабатывали в этой главе, то столкнемся с ошибкой TFX, информирующей нас, что вход не зарегистрирован в хранилище метаданных и что нам нужно запустить предшествующий компонент перед запуском нашего пользовательского компонента. Но в нашем случае у нас отсутствует предшествующий компонент,

поскольку мы загружаем данные в наш конвейер. Загрузка данных – это начальный этап каждого конвейера. Так что же происходит?

Как мы обсуждали ранее, все компоненты в TFX взаимодействуют друг с другом посредством хранилища метаданных, и компоненты ожидают, что входные артефакты уже зарегистрированы в этом хранилище. В нашем случае мы хотим получить данные с диска, и мы читаем данные в первый раз за все время работы нашего конвейера; поэтому данные не передаются из другого компонента, и нам необходимо зарегистрировать источники данных в хранилище метаданных.



Пользовательские драйверы встречаются редко

Разрабатывать и использовать собственные драйверы требуется весьма редко. Если вы можете повторно использовать архитектуру ввода/вывода существующего компонента TFX или если входы уже зарегистрированы в хранилище метаданных, вам не нужно писать собственный драйвер, и вы можете просто пропустить этот шаг.

Подобно тому как мы делали это с нашим настраиваемым модулем исполнения, мы можем повторно использовать класс `BaseDriver`, предоставляемый TFX, для написания настраиваемого драйвера. Нам нужно изменить стандартное поведение компонента, и мы можем сделать это, переопределив метод `resolve_input_artifacts` в `BaseDriver`. Простой драйвер будет регистрировать наши входные данные, что очень просто. Нам нужно *разгруппировать* канал, чтобы получить `input_dict`. Просматривая все значения `input_dict`, мы можем получить доступ к каждому списку входных данных. Повторно просматривая каждый список, мы можем получить каждый вход, а затем зарегистрировать его в хранилище метаданных, передав его функции `publish_artifacts`. Функция `publish_artifacts` вызовет хранилище метаданных, опубликует артефакт и обновит состояние артефакта как готового к публикации:

```
class ImageIngestDriver(base_driver.BaseDriver):
    """Пользовательский драйвер ImageIngest."""

    def resolve_input_artifacts(
        self,
        input_channels: Dict[Text, types.Channel],
        exec_properties: Dict[Text, Any],
        driver_args: data_types.DriverArgs,
        pipeline_info: data_types.PipelineInfo) -> Dict[Text, List[types.Artifact]]:
        """Переопределяет BaseDriver.resolve_input_artifacts()."""
        del driver_args ❶
        del pipeline_info

        input_dict = channel_utils.unwrap_channel_dict(input_channels) ❷
        for input_list in input_dict.values():
            for single_input in input_list:
                self._metadata_handler.publish_artifacts([single_input]) ❸
                absl.logging.debug("Registered input: {}".format(single_input))
                absl.logging.debug("single_input.mlmd_artifact "
                                   "{}".format(single_input.mlmd_artifact)) ❹

        return input_dict
```

- ❶ Удаление неиспользуемых аргументов
- ❷ Развертывание канала для получения входного словаря
- ❸ Публикация артефакта
- ❹ Вывод информации об артефакте

Пока мы просматриваем каждый ввод, мы можем вывести дополнительную информацию:

```
print("Registered new input: {}".format(single_input))
print("Artifact URI: {}".format(single_input.uri))
print("MLMD Artifact Info: {}".format(single_input.mlmd_artifact))
```

Теперь, когда настраиваемый драйвер установлен и настроен, нам нужно собрать наш компонент.

Сборка пользовательского компонента

Определив `ImageIngestComponentSpec`, завершив разработку `ImageIngestExecutor` и настроив `ImageIngestDriver`, нам необходимо объединить все выполненные изменения вместе в нашем компоненте `ImageIngestComponent`. Затем мы сможем, например, загрузить компонент в конвейер, который обучает модели классификации изображений.

Чтобы определить фактический компонент, нам нужно определить классы спецификации, модуля исполнения и драйвера. Мы можем сделать это, задав `SPEC_CLASS`, `EXECUTOR_SPEC` и `DRIVER_CLASS`, как показано в приведенном ниже примере кода. В качестве последнего шага нам нужно создать экземпляр наших `ComponentSpecs` с аргументами компонента (например, примерами ввода и вывода и определенным именем) и передать его созданному экземпляру `ImageIngestComponent`.

В том маловероятном случае, когда мы не предоставляем выходной артефакт, мы можем установить для нашего выходного артефакта по умолчанию тип `tf.Example`, определить наши жестко закодированные имена разбиений и настроить его как канал:

```
from tfx.components.base import base_component
from tfx import types
from tfx.types import channel_utils

class ImageIngestComponent(base_component.BaseComponent):
    """Пользовательский компонент ImageIngestWorld."""
    SPEC_CLASS = ImageIngestComponentSpec
    EXECUTOR_SPEC = executor_spec.ExecutorClassSpec(ImageIngestExecutor)
    DRIVER_CLASS = ImageIngestDriver

    def __init__(self, input, output_data=None, name=None):
        if not output_data:
            examples_artifact = standard_artifacts.Examples()
            examples_artifact.split_names = \
                artifact_utils.encode_split_names(['train', 'eval'])
            output_data = channel_utils.as_channel([examples_artifact])
        spec = ImageIngestComponentSpec(input=input,
                                         examples=output_data,
                                         name=name)
        super(ImageIngestComponent, self).__init__(spec=spec)
```

Собрав наш ImageIngestComponent, мы связали вместе отдельные части нашего основного настраиваемого компонента. В следующем разделе мы рассмотрим, как можем запустить на исполнение наш базовый компонент.

Запуск элементарного пользовательского компонента

После того как мы реализовали элементарный компонент для загрузки изображений и преобразования их в файлы TFRecord, мы можем использовать этот компонент, как любой другой компонент в нашем конвейере. В следующем примере кода показано, как это сделать. Обратите внимание, что приведенный пример выглядит аналогично тому, как выполнялась настройка других компонентов загрузки, которые мы обсуждали в главе 3. Единственное отличие состоит в том, что нам нужно импортировать наш вновь созданный компонент, а затем запустить инициализированный компонент:

```
import os

from tfx.utils.dsl_utils import external_input
from tfx.orchestration.experimental.interactive.interactive_context import \
    InteractiveContext

from image_ingestion_component.component import \
    ImageIngestComponent context = InteractiveContext()

image_file_path = "/путь/к/файлам"
examples = external_input(dataimage_file_path_root)
example_gen = ImageIngestComponent(input=examples,
                                   name=u'ImageIngestComponent')
context.run(example_gen)
```

Выходные данные нашего компонента могут затем использоваться последующими компонентами, такими как StatisticsGen:

```
from tfx.components import StatisticsGen

statistics_gen = StatisticsGen(examples=example_gen.outputs['examples'])
context.run(statistics_gen)

context.show(statistics_gen.outputs['statistics'])
```



Простейшая реализация

Мы хотим особо отметить, что обсуждаемая реализация обеспечивает только базовую функциональность и не готова для промышленной эксплуатации. Подробнее об отсутствующих в нашем драйвере функциях мы расскажем в следующем разделе. Для реализации готового продукта см. обновленную реализацию компонента в следующих разделах.

Обзор реализации

В предыдущих разделах мы рассмотрели реализацию элементарного пользовательского компонента. В нем отсутствуют некоторые ключевые функциональные возможности, которые мы обсуждали в главе 3 (например, имена

динамического разбиения или соотношения разбиения), но мы ожидаем, что в нашем компоненте загрузки данных будут реализованы такие возможности. Элементарный пример реализации также использовал большое количество шаблонного кода (например, настройки драйвера компонента). Загрузка изображений должна осуществляться эффективно и предоставлять возможность масштабирования. Мы можем реализовать такую эффективную загрузку данных за счет использования Apache Beam, поверх которого будут работать компоненты TFX.

В следующем разделе мы обсудим, как можно упростить реализации и адаптировать шаблоны, которые мы обсуждали в главе 3, например получение данных из баз данных Presto. Повторно используя общие функции, такие как драйверы компонентов, мы можем ускорить и упростить внедрение и уменьшить количество ошибок в коде.

Повторное использование существующих компонентов

Вместо того чтобы писать компонент для TFX полностью с нуля, мы можем унаследовать существующий компонент и настроить его, изменив функциональность элемента executor. Как показано на рис. 10.8, это стандартный, предпочтительный подход, когда пользовательский компонент повторно использует архитектуру существующего компонента. В случае нашего компонента, используемого в качестве примера, архитектура эквивалентна компоненту загрузки файловой базы (например, `CsvExampleGen`). Такие компоненты получают путь к каталогу в качестве входных данных, загружают данные из указанного каталога, преобразуют данные в `tf.Examples` и возвращают структуры данных в файлах `TFRecord` в качестве выходных данных.

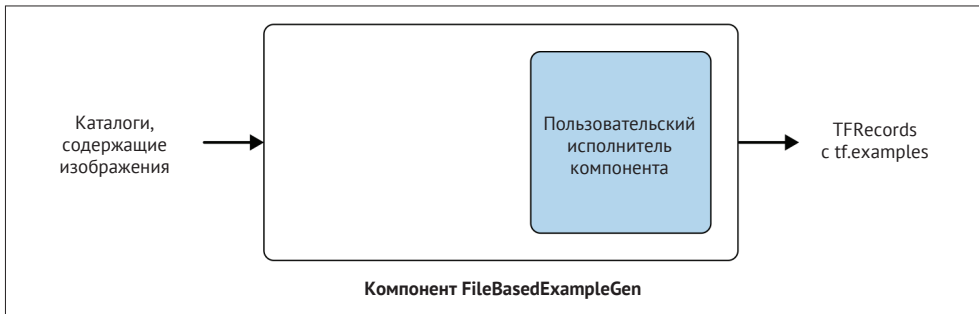


Рис. 10.8. Расширение существующих компонентов

Как мы обсуждали в главе 3, TFX для этой цели предоставляет `FileBasedExampleGen`. Поскольку мы собираемся повторно использовать существующий компонент, аналогично нашим примерам для Avro и Parquet, мы можем просто сосредоточиться на разработке нашего собственного элемента executor и сделать его более гибким, нежели пример, приведенный в предыдущем варианте реализации нашего элементарного компонента. Повторно используя существующую инфраструктуру кода, мы также можем использовать существующие реализации Apache Beam.

Повторно используя существующую архитектуру компонента для загрузки данных в наши конвейеры, мы можем повторно использовать и настройки для эффективной загрузки данных с помощью Apache Beam. TFX и Apache Beam предоставляют классы (например, `GetInputSourceToExamplePTransform`) и декораторы функций (например, `@beam.ptransform_fn`) для загрузки данных при помощи конвейеров Apache Beam. В нашем примере мы используем декоратор функции `@beam.ptransform_fn`, который позволяет нам определить преобразование Apache Beam (`PTransform`). Декоратор принимает на вход данные конвейера Apache Beam, выполняет заданное преобразование (например, в нашем случае загрузку изображений и их преобразование в `tf.Examples`) и возвращает обратно Apache Beam `PCollection` с результатами преобразования.

Преобразование выполняется функцией, очень похожей на функцию из нашей предыдущей реализации. Новая реализация преобразования имеет одно существенное отличие: нам не нужно создавать и использовать модуль записи `TfRecord`; вместо этого мы можем полностью сосредоточиться на загрузке изображений и преобразовании их в `tf.Examples`. Нам не нужно реализовывать какие-либо функции для записи `tf.Examples` в структуры данных `TfRecord`, потому что мы сделали это в нашей предыдущей реализации. Вместо этого мы возвращаем сгенерированные `tf.Examples` и позволяем нижележащему коду TFX/ Apache Beam обрабатывать запись файлов `TfRecord`. В следующем примере кода показана обновленная функция преобразования:

```
def convert_image_to_TFExample(image_path):  @@ 1 @@

    # Определите метку для каждого изображения на основе пути к файлу.
    lowered_filename = image_path.lower()
    print(lowered_filename)
    if "dog" in lowered_filename:
        label = 0
    elif "cat" in lowered_filename:
        label = 1
    else:
        raise NotImplementedError("Найдено неизвестное изображение")

    # Считывание изображения.
    raw_file = tf.io.read_file(image_path)

    # Создание структуры данных TensorFlow Example.
    example = tf.train.Example(features=tf.train.Features(feature={
        'image_raw': _bytes_feature(raw_file.numpy()),
        'label': _int64_feature(label)
    }))
    return example  2
```

❶ Требуется только путь к файлу

❷ Функция возвращает примеры, вместо того чтобы записывать их на диск

Имея обновленную функцию преобразования, мы можем сосредоточиться на реализации основных функций исполнителя компонентов – элемента `executor`. Поскольку мы изменяем существующую архитектуру компонентов, то можем использовать те же аргументы, которые мы обсуждали в главе 3, например раз-

делить шаблоны. Наша функция `image_to_example` в следующем примере кода принимает четыре входных аргумента: объект конвейера Apache Beam, `input_dict`, который содержит информацию об артефакте, словарь с параметрами выполнения и шаблоны разбиения для загрузки данных. В этой функции мы генерируем список доступных файлов в указанных каталогах и передаем список изображений конвейеру Apache Beam для преобразования каждого изображения, найденного во входных каталогах, из которых осуществляется загрузка, в `tf.Examples`:

```
@beam.ptransform_fn
def image_to_example(
    pipeline: beam.Pipeline,
    input_dict: Dict[Text, List[types.Artifact]],
    exec_properties: Dict[Text, Any],
    split_pattern: Text) -> beam.pvalue.PCollection:

    input_base_uri = artifact_utils.get_single_uri(input_dict['input'])
    image_pattern = os.path.join(input_base_uri, split_pattern)
    absl.logging.info(
        "Обработка входных данных изображений {} "
        "преобразование в tf.Example.".format(image_pattern))

    image_files = tf.io.gfile.glob(image_pattern) ❶
    if not image_files:
        raise RuntimeError(
            "Шаблон разбиения {} не нашел подходящего пути."
            "{}".format(image_pattern))

    p_collection = (
        pipeline
        | beam.Create(image_files) ❷
        | 'ConvertImagesToTFRecords' >> beam.Map(
            lambda image: convert_image_to_TFExample(image)) ❸
    )
    return p_collection
```

- ❶ Генерация списка файлов, присутствующих во входных путях
- ❷ Преобразование списка в коллекцию Beam PCollection
- ❸ Применение преобразования к каждому изображению

Последний шаг в нашем пользовательском исполнителе компонентов – переопределить преобразование `GetInputSourceToExamplePTransform`, заменив `BaseExampleGenExecutor` нашим `image_to_example`:

```
class ImageExampleGenExecutor(BaseExampleGenExecutor):

    @beam.ptransform_fn
    def image_to_example(...):
    ...

    def GetInputSourceToExamplePTransform(self) -> beam.PTransform:
        return image_to_example
```

Мы реализовали наш пользовательский компонент загрузки изображений!

Использование собственной реализации исполнителей компонентов

Поскольку мы повторно используем компонент загрузки изображений и заменяем исполнитель компонента обработки изображений собственной реализацией, то в данном случае мы можем следовать тем же шаблонам для загрузки данных Avro, которые мы обсуждали в главе 3, и использовать `custom_executor_spec`. Повторно используя компонент `FileBasedExampleGen` и переопределив исполнитель компонента, мы можем использовать все функциональные возможности компонентов загрузки, которые обсуждали в главе 3, такие как определение шаблонов разбиения при загрузке данных или разбиения обучающих/оценочных наборов. Приведенный ниже фрагмент кода дает полный пример использования нашего настраиваемого компонента:

```
from tfx.components import FileBasedExampleGen
from tfx.utils.dsl_utils import external_input
from image_ingestion_component.executor import
ImageExampleGenExecutor input_config = example_gen_pb2.Input(splits=[
    example_gen_pb2.Input.Split(name='images',
                                pattern='sub-directory/if/needed/*.jpg'),
])

output = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(
            name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(
            name='eval', hash_buckets=1)
    ])
)

example_gen = FileBasedExampleGen(
    input=external_input("/path/to/images/"),
    input_config=input_config,
    output_config=output,
    custom_executor_spec=executor_spec.ExecutorClassSpec(
        ImageExampleGenExecutor)
)
```

Как мы упоминали в этом разделе, изменение кода исполнителя компонента всегда будет более простой и быстрой реализацией, чем написание пользовательского компонента с нуля. Поэтому мы рекомендуем этот способ, если вы можете повторно использовать архитектуру существующих компонентов.

РЕЗЮМЕ

Эта глава познакомила вас с расширенными концепциями TFX, выходящими за рамки возможностей, которым были посвящены предыдущие главы. Мы подробно рассмотрели процесс создания собственных компонентов. Написание пользовательских компонентов дает нам возможность расширять существующие компоненты TFX и адаптировать их к требованиям нашего конвейера. Пользовательские компоненты позволяют нам интегрировать больше шагов в наши конвейеры машинного обучения. Добавляя дополнительные компо-

ненты в наш конвейер, мы можем гарантировать, что все модели, генерируемые конвейером, прошли одни и те же этапы. Реализация пользовательских компонентов может быть сложной, поэтому мы рассмотрели реализацию элементарного компонента с нуля и реализацию нового исполнителя компонента путем наследования существующих функций компонента.

Мы также обсудили расширенные настройки для обучения, такие как ветвящиеся графы конвейера для создания нескольких моделей на основе одного и того же исполнения конвейера машинного обучения. Эта функция может использоваться для создания моделей TFLite для развертывания в мобильных приложениях. Мы также обсудили «теплый» запуск процесса обучения для непрерывного обучения моделей машинного обучения. Обучение модели с использованием «теплого» запуска – отличный способ сократить шаги обучения для непрерывно обучаемых моделей.

Мы представили концепцию участия человека в контуре конвейера машинного обучения, а также обсудили, как можно реализовать экспериментальный компонент. Концепция «участие человека в конвейере машинного обучения» – это способ добавления экспертной оценки в качестве необходимого шага конвейера перед развертыванием моделей. Мы считаем, что сочетание полностью автоматизированных компонентов и нескольких операций, выполняемых специалистами по данным, станет дополнительным аргументом при внедрении конвейеров машинного обучения.

В следующих двух главах мы рассмотрим, как запустить наш конвейер TFХ в выбранной среде оркестрации.

Глава 11

Конвейеры, часть 1: Apache Beam и Apache Airflow

В предыдущих главах мы рассмотрели все необходимые компоненты для создания конвейера машинного обучения с использованием TFX. В этой главе мы соберем все компоненты вместе и покажем, как запустить полный конвейер с двумя оркестровщиками: Apache Beam и Apache Airflow. В главе 12 мы также покажем, как запустить конвейер с помощью Kubeflow Pipelines. Все эти инструменты используют схожие принципы, но мы покажем, в чем заключаются различия между ними, и предоставим примеры кода для каждого из них.

Как мы обсуждали в главе 1, инструмент оркестрации конвейера жизненно важен для абстрагирования связующего кода, который в противном случае нам пришлось бы написать для автоматизации конвейера машинного обучения. Как показано на рис. 11.1, компоненты оркестрации конвейера располагаются под компонентами, которые мы уже упоминали в предыдущих главах. Без этих инструментов оркестрации нам пришлось бы написать код, проверяющий, когда один компонент конвейера завершает работу, запускает следующий компонент, планирует запуски конвейера и т. д. К счастью, весь этот код уже существует в виде этих оркестровщиков!

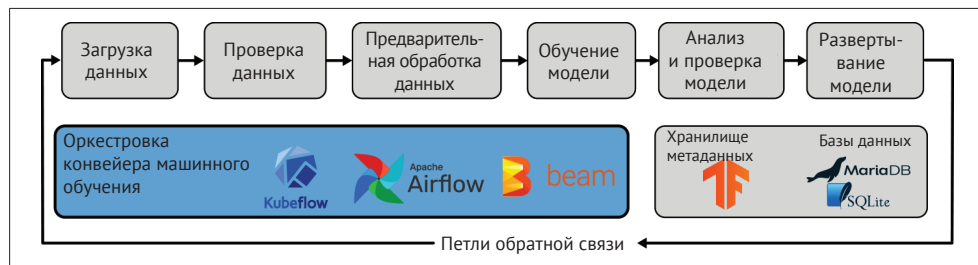


Рис. 11.1. Компоненты оркестрации конвейера

Мы начнем эту главу с обсуждения вариантов использования различных инструментов. Затем рассмотрим некоторый общий код, необходимый для перехода от интерактивного конвейера к конвейеру, который может быть создан с помощью этих инструментов. Apache Beam и Apache Airflow настроить про-

ще, чем конвейеры Kubeflow, поэтому мы обсудим их в этой главе, прежде чем перейти к более мощным конвейерам Kubeflow в главе 12.

КАКОЙ ИНСТРУМЕНТ ОРКЕСТРАЦИИ ВЫБРАТЬ?

В этой главе и в главе 12 мы обсудим три инструмента оркестрации, которые вы можете использовать для запуска ваших конвейеров: Apache Beam, Apache Airflow и Kubeflow Pipelines. Вам нужно выбрать только один из них для запуска каждого отдельного конвейера. Прежде чем мы углубимся в подробности того, как использовать все эти инструменты, мы опишем некоторые из преимуществ и недостатков каждого из них, что поможет вам решить, какой из этих инструментов лучше всего подходит для ваших нужд.

Apache Beam

Если вы используете TFX для своих задач конвейера, вы уже установили Apache Beam. Поэтому если вы рассматриваете минимальную установку, повторное использование Beam для оркестрации – логичный выбор. Его просто настроить, и он также позволяет использовать любую существующую инфраструктуру распределенной обработки данных, с которой вы, возможно, уже знакомы (например, Google Cloud Dataflow). Вы также можете использовать Beam в качестве промежуточного шага, чтобы убедиться, что ваш конвейер работает правильно, прежде чем переходить к Airflow или Kubeflow Pipelines.

Однако в Apache Beam отсутствуют различные инструменты для планирования обновлений вашей модели или мониторинга процесса выполнения задач конвейера. Вот где выгодно проявляются преимущества Apache Airflow и Kubeflow Pipelines.

Apache Airflow

Apache Airflow часто уже используется в различных компаниях, внедряющих машинное обучение, для задач загрузки данных. Расширение существующей настройки Apache Airflow для запуска вашего конвейера избавляет вас от необходимости изучать новый инструмент, такой как Kubeflow.

Если вы используете Apache Airflow в сочетании с готовой к работе базой данных, такой как PostgreSQL, то можете воспользоваться преимуществами создания и запуска частичных конвейеров. Это может сэкономить значительное количество времени, если трудоемкий конвейер выходит из строя и вы хотите избежать повторного запуска всех предыдущих этапов конвейера.

Kubeflow Pipelines

Если у вас уже есть опыт работы с Kubernetes и доступ к кластеру Kubernetes, имеет смысл рассмотреть такой инструмент, как Kubeflow Pipelines. Хотя установка Kubeflow более сложна, чем установка Airflow, она открывает множество новых возможностей, включая возможность просмотра визуализаций TFDV и TFMA, эволюцию модели и коллекции артефактов.

Kubernetes также является отличной инфраструктурной платформой для развертывания моделей машинного обучения. На текущий момент маршрутизация

логических выводов модели с помощью инструмента Kubernetes Istio является передовым достижением в области инфраструктуры машинного обучения.

Вы можете настроить кластер Kubernetes так, чтобы иметь возможность использовать сервисы различных поставщиков облачных услуг, и вам не придется ограничиваться лишь одним поставщиком. Kubeflow Pipelines также позволяет использовать самое современное оборудование для обучения, предоставляемое поставщиками облачных услуг. Вы можете запустить конвейер и эффективно масштабировать узлы кластера, наращивая или сокращая их мощности.

Kubeflow Pipelines на платформе AI

Также можно запустить Kubeflow Pipelines на платформе Google AI Platform, которая является частью GCP. Такое решение возьмет на себя функции обслуживания большей части инфраструктуры и упростит загрузку данных из облачных хранилищ Google Cloud Storage. Кроме того, интеграция Google Dataflow упрощает масштабирование ваших конвейеров. Однако подобный вариант имеет определенные ограничения – вы сможете использовать сервисы только одного облачного провайдера.

Если вы решите использовать Apache Beam или Airflow, из этой главы вы можете получить всю необходимую информацию. Если вы выберете Kubeflow (через Kubernetes или на платформе AI Google Cloud), вам нужно будет прочитать только следующий раздел этой главы. Материалы данной главы продемонстрируют вам, как преобразовать ваш интерактивный конвейер в сценарий, а затем вы сможете сразу перейти к главе 12.

ПРЕОБРАЗОВАНИЕ ВАШЕГО ИНТЕРАКТИВНОГО КОНВЕЙЕРА TFX В ПРОИЗВОДСТВЕННЫЙ КОНВЕЙЕР

До этого момента в наших примерах было показано, как запускать все различные компоненты конвейера TFX в такой среде, как, например, блокноты (notebooks) – иными словами, в *интерактивном контексте*. Чтобы запустить конвейер в блокноте, каждый компонент должен запускаться вручную после завершения работы предыдущего. Чтобы автоматизировать наши конвейеры, нам нужно будет написать сценарий Python, который будет запускать все эти компоненты без какого-либо участия с нашей стороны.

К счастью, у нас уже есть все части этого сценария. Мы объединяем вместе все компоненты конвейера, которые обсуждали до сих пор:

ExampleGen

получает новые данные из источника данных, который мы хотим использовать (см. главу 3);

StatisticsGen

вычисляет сводную статистику для новых данных (см. главу 4);

SchemaGen

определяет ожидаемые признаки модели, а также их типы и диапазоны (см. главу 4);

ExampleValidator

проверяет данные на соответствие схеме и помечает любые аномалии (см. главу 4);

Transform

предварительно обрабатывает данные в правильном числовом представлении, ожидаемом моделью (см. главу 5);

Trainer

обучает модель на новых данных (см. главу 6);

Resolver

проверяет наличие ранее одобренной модели и возвращает ее для сравнения (см. главу 7);

Evaluator

оценивает качество прогнозов модели на оценочном наборе данных и проверяет, улучшается ли качество предсказаний модели по сравнению с предыдущей версией (см. главу 7);

Pusher

отправляет модель в рабочий каталог, если она проходит этап проверки (см. главу 7).

Полный конвейер показан в примере 11.1.

Пример 11.1. Элементарный конвейер

```
import tensorflow_model_analysis as tfma
from tfx.components import (CsvExampleGen, Evaluator, ExampleValidator, Pusher,
                           ResolverNode, SchemaGen, StatisticsGen, Trainer,
                           Transform)
from tfx.components.base import executor_spec
from tfx.components.trainer.executor import GenericExecutor
from tfx.dsl.experimental import
latest_blessed_model_resolver from tfx.proto import
pusher_pb2, trainer_pb2
from tfx.types import Channel
from tfx.types.standard_artifacts import Model, ModelBlessing
from tfx.utils.dsl_utils import external_input

def init_components(data_dir, module_file, serving_model_dir,
                   training_steps=2000, eval_steps=200):

    examples = external_input(data_dir)
    example_gen = CsvExampleGen(...)
    statistics_gen = StatisticsGen(...)
    schema_gen = SchemaGen(...)
    example_validator = ExampleValidator(...)
    transform = Transform(...)
    trainer = Trainer(...)
    model_resolver = ResolverNode(...)
    eval_config=tfma.EvalConfig(...)
```

```

evaluator = Evaluator(...)
pusher = Pusher(...)

components = [
    example_gen,
    statistics_gen,
    schema_gen,
    example_validator,
    transform,
    trainer,
    model_resolver,
    evaluator,
    pusher
]
return components

```

В нашем демонстрационном проекте мы отделили создание экземпляра компонента от конфигурации конвейера, чтобы сосредоточиться на настройке конвейера для различных оркестровщиков.

Функция `init_components` создает экземпляры компонентов. В дополнение к количеству этапов обучения и оценки требуется три входных параметра:

`data_dir`

путь, по которому можно найти данные обучающего/оценочного набора;

`module_file`

модуль Python, необходимый для компонентов `Transform` и `Trainer`. Эти компоненты описаны в главах 5 и 6 соответственно;

`serve_model_dir`

путь к каталогу, в котором должна храниться экспортированная модель.

Помимо небольших изменений в настройке Google Cloud, которые мы обсудим в главе 12, настройка компонентов будет идентична для каждой платформы оркестровщика. Поэтому мы будем повторно использовать определение компонента в различных примерах настроек для Apache Beam, Apache Airflow и Kubeflow Pipelines. Если вы хотите использовать Kubeflow Pipelines, вы можете обнаружить, что для отладки вашего конвейера окажется полезен Beam. Но если вы хотите сразу перейти к Kubeflow Pipelines, переходите к следующей главе.

ПРЕОБРАЗОВАНИЕ ЭЛЕМЕНТАРНОГО ИНТЕРАКТИВНОГО КОНВЕЙЕРА ДЛЯ BEAM И AIRFLOW

Если вы хотите организовать конвейер с помощью Apache Beam или Airflow, то можете преобразовать блокнот (notebook) в конвейер, выполнив приведенные ниже действия. Для всех ячеек в блокноте, которые вы не хотите экспортировать, используйте волшебную команду `Jupyter %skip_for_export` в начале каждой ячейки.

Сначала задайте имя конвейера и инструмент оркестрации:

```
runner_type = 'beam' ❶
pipeline_name = 'consumer_complaints_beam'
```

❶ Как вариант — `airflow`.

Затем установите все соответствующие пути к файлам:

```
notebook_file = os.path.join(os.getcwd(), notebook_filename)

# Входы конвейера
data_dir = os.path.join(pipeline_dir, 'data')
module_file = os.path.join(pipeline_dir, 'components', 'module.py')
requirement_file = os.path.join(pipeline_dir, 'requirements.txt')

# Выходы конвейера
output_base = os.path.join(pipeline_dir, 'output', pipeline_name)
serving_model_dir = os.path.join(output_base, pipeline_name)
pipeline_root = os.path.join(output_base, 'pipeline_root')
metadata_path = os.path.join(pipeline_root, 'metadata.sqlite')
```

И далее перечислите компоненты, которые хотите включить в свой конвейер:

```
components = [
    example_gen, statistics_gen, schema_gen, example_validator,
    transform, trainer, evaluator, pusher
]
```

И выполните экспорт вашего конвейера:

```
pipeline_export_file = 'consumer_complaints_beam_export.py'
context.export_to_pipeline(notebook_file path=notebook_file,
                           export_file path=pipeline_export_file,
                           runner_type=runner_type)
```

Эта команда экспорта сгенерирует сценарий, который можно запустить с помощью Beam или Airflow, в зависимости от выбранного вами runner_type.

ВВЕДЕНИЕ В АРАСНЕ БЕАМ

Поскольку многие компоненты TFX работают поверх Apache Beam, мы кратко рассказали о нем в главе 2. Различные компоненты TFX (например, TFDV или Transform как компонент TensorFlow) используют Apache Beam для абстракции внутренней обработки данных. Но многие из тех же функций Beam также могут использоваться для запуска вашего конвейера. В следующем разделе мы покажем вам, как организовать наш демонстрационный проект на основе Beam.

ОРКЕСТРАЦИЯ КОНВЕЙЕРОВ TFX С ПОМОЩЬЮ АРАСНЕ БЕАМ

Apache Beam уже установлен как зависимый от TFX компонент, и это позволяет довольно легко начать использовать его в качестве инструмента оркестрации конвейера. Beam очень прост и не включает всех тех функций, которые есть

в Airflow или Kubeflow Pipelines, таких как визуализация графиков, выполнение по расписанию и т. д.

Beam также может быть хорошим способом отладки конвейера машинного обучения. Используя Beam во время отладки конвейера, а затем переходя к Airflow или Kubeflow Pipelines, вы можете исключить основные причины ошибок конвейера, возникающие из-за более сложных настроек Airflow или Kubeflow Pipelines.

В этом разделе мы рассмотрим, как настроить и запустить наш пример конвейера TFX с Beam. Мы представили функцию Beam Pipeline в главе 2. Это то, что мы будем использовать вместе с нашим сценарием из примера 11.1 для запуска конвейера. Мы определим Beam Pipeline, который принимает компоненты конвейера TFX в качестве аргумента, а также подключается к базе данных SQLite, содержащей хранилище метаданных ML:

```
import absl
from tfx.orchestration import metadata, pipeline

def init_beam_pipeline(components, pipeline_root, direct_num_workers):

    absl.logging.info("Pipeline root set to: {}".format(pipeline_root))
    beam_arg = [
        "--direct_num_workers={}".format(direct_num_workers), ❶
        "--requirements_file={}".format(requirement_file)
    ]

    P = pipeline.Pipeline( ❷
        pipeline_name=pipeline_name,
        pipeline_root=pipeline_root,
        components=components,
        enable_cache=False, ❸
        metadata_connection_config=\
            metadata.sqlite_metadata_connection_config(metadata_path),
        beam_pipeline_args=beam_arg)
    return P
```

- ❶ Beam позволяет указать количество рабочих процессов. Разумным значением по умолчанию для этого параметра является половина числа ЦП (если ЦП больше одного)
- ❷ Здесь вы определяете свой объект конвейера и его конфигурацию
- ❸ Мы можем установить для кеша значение True, если хотим избежать повторного запуска уже завершенных компонентов. Если мы установим этот флаг в значение False, все будет пересчитываться каждый раз, когда мы заново запускаем конвейер

Конфигурация конвейера Beam должна содержать имя конвейера, путь к корню каталога конвейера и список компонентов, которые будут выполняться как часть конвейера.

Затем мы инициализируем компоненты из примера 11.1, инициализируем конвейер, как и раньше, и запустим конвейер с помощью команды `BeamDagRunner().run(pipeline):`

```
from tfx.orchestration.beam.beam_dag_runner import BeamDagRunner

components = init_components(data_dir, module_file, serving_model_dir,
                             training_steps=100, eval_steps=100)
pipeline = init_beam_pipeline(components, pipeline_root, direct_num_workers)
BeamDagRunner().run(pipeline)
```

Это минимальная настройка, которую вы можете легко интегрировать с остальной инфраструктурой или задать рабочее расписание с помощью задачи cron. Вы также можете масштабировать этот конвейер, применяя Apache Flink (см. <https://flink.apache.org/>) или Spark. Пример использования Flink кратко описан в примере TFX, опубликованном по ссылке <https://oreil.ly/FYzLY>.

В следующем разделе мы перейдем к Apache Airflow. Apache Airflow содержит множество дополнительных функций, когда мы используем его для оркестрации наших конвейеров.

ВВЕДЕНИЕ В АРАШЕ АЙРФЛОУ

Airflow – это проект Apache для автоматизации рабочих процессов. Проект был запущен в 2016 году, и с тех пор он привлек значительное внимание крупных корпораций и всего сообщества специалистов по анализу данных. В декабре 2018 года проект вышел из «инкубатора» Apache и стал собственным проектом Apache.

Apache Airflow позволяет представлять задачи рабочего процесса в виде направленных ациклических графов (НАГ), представленных как код Python. Кроме того, Airflow позволяет планировать и контролировать рабочие процессы. Это делает его идеальным инструментом оркестрации для наших конвейеров TFX.

В данном разделе мы рассмотрим основные настройки Airflow. Затем мы покажем, как можем использовать его для запуска нашего демонстрационного проекта.

Установка и начальная настройка

Базовая настройка Apache Airflow проста. Если вы используете Mac или Linux, определите каталог, предназначенный для размещения данных Airflow, выполнив следующую команду:

```
$ export AIRFLOW_HOME=~/.airflow
```

Как только основная папка данных для Airflow определена, вы можете установить Airflow:

```
$ pip install apache-airflow
```

Airflow можно установить со множеством зависимостей. На момент написания этой книги список расширений включал поддержку PostgreSQL, Dask, Celery и Kubernetes.

Полный список расширений Airflow и способы их установки можно найти в документации Airflow (см. <https://oreil.ly/evVfy>).

Теперь, когда Airflow установлен, вам необходимо создать исходную базу данных, в которой будет храниться вся информация о статусе задачи. В Airflow имеется специальная команда для инициализации базы данных Airflow:

```
$ airflow initdb
```

Если вы используете стандартную конфигурацию Airflow и не изменяли никакие настройки, Airflow создаст экземпляр базы данных SQLite. Такой вариант установки подходит для выполнения демонстрационных проектов и небольших рабочих процессов. Если вы хотите масштабировать свой рабочий процесс с помощью Apache Airflow, мы настоятельно рекомендуем более глубоко погрузиться в документацию.

Минимальный вариант конфигурации Airflow включает планировщик Airflow, который координирует задачи и зависимости задач, а также веб-сервера, который предоставляет пользовательский интерфейс для запуска, остановки и мониторинга задач.

Запустите планировщик с помощью следующей команды:

```
$ airflow scheduler
```

В другом окне терминала запустите веб-сервер Airflow, выполнив следующую команду:

```
$ airflow webserver -p 8081
```

Аргумент команды `-p` определяет порт, через который ваш веб-браузер может получить доступ к интерфейсу Airflow. Когда все заработает, в браузере перейдите по ссылке <http://127.0.0.1:8081>. Вы должны увидеть интерфейс, показанный на рис. 11.2.

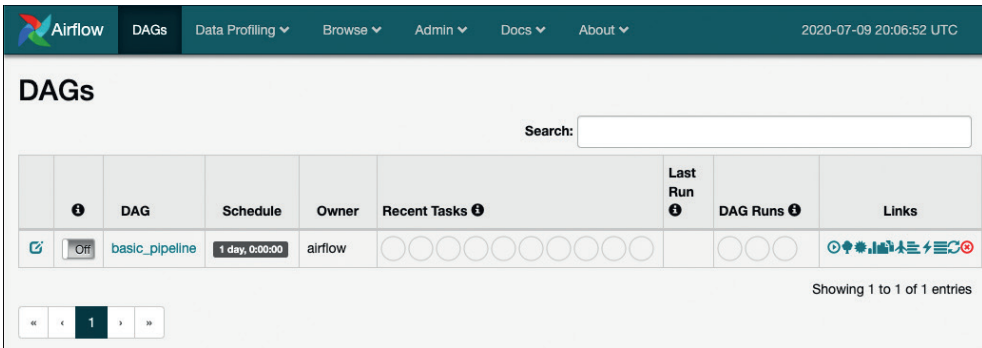


Рис. 11.2. Пользовательский интерфейс Apache Airflow



Конфигурация Airflow

Настройки Airflow по умолчанию можно переопределить, изменив соответствующие параметры в конфигурации Airflow. Если вы храните определения ваших графов не в каталоге `~/airflow/dags`, а в другом месте, вам может понадобиться изменить конфигурацию по умолчанию, указав новые местоположения графов конвейера в `~/airflow/airflow.cfg`.

Элементарный пример использования Airflow

Выполнив установку Airflow, давайте посмотрим, как настроить элементарный конвейер Airflow. В этом примере мы не будем использовать какие-либо компоненты TFX.

Конвейеры рабочего процесса определены как сценарии Python, и Airflow ожидает, что определения НАГ будут размещены в `~/airflow/dags`. Элементарный конвейер состоит из конфигураций Airflow для конкретного проекта, определений задач и определения зависимостей задач.

Конфигурация Airflow для конкретного проекта

Airflow дает вам возможность настраивать параметры для конкретного проекта, например определить, когда повторять неудачно завершившиеся рабочие процессы, или указать, кому присылать уведомления в случае сбоя рабочего процесса. Список вариантов конфигурации достаточно обширен. Мы рекомендуем вам обратиться к документации Airflow (см. <https://airflow.apache.org/>), где опубликован самый свежий обзор его возможностей.

Определение параметров конфигурации вашего конвейера Airflow начинается с импорта соответствующих модулей Python и указания определенных параметров конфигурации проекта:

```
from airflow import DAG
from datetime import datetime, timedelta

project_cfg = { ❶
    'owner': 'airflow',
    'email': ['your-email@example.com'],
    'email_on_failure': True,
    'start_date': datetime(2019, 8, 1),
    'retries': 1,
    'retry_delay': timedelta(hours=1),
}

dag = DAG( ❷
    'basic_pipeline',
    default_args=project_cfg,
    schedule_interval=timedelta(days=1))
```

- ❶ Местоположение для определения конфигурации проекта
- ❷ Airflow возьмет объект DAG

Airflow также предоставляет ряд параметров конфигурации для настройки объектов DAG.

Определения задач

После того как мы задали параметры конфигурации для настройки объекта DAG, мы можем создавать задачи рабочего процесса. Airflow предоставляет операторы задач, которые запускают выполнение задач в среде Bash или Python. Другие предопределенные операторы позволяют подключаться к сегментам облачного хранилища данных, таким как GCP Storage или AWS S3.

Ниже приведен простейший пример определения задач:

```
from airflow.operators.python_operator import PythonOperator

def example_task(_id, **kwargs):
    print("task {}".format(_id))
    return "completed task {}".format(_id)

task_1 = PythonOperator(
    task_id='task 1',
    provide_context=True,
    python_callable=example_task,
    op_kwargs={'_id': 1},
    dag=dag,
)

task_2 = PythonOperator(
    task_id='task 2',
    provide_context=True,
    python_callable=example_task,
    op_kwargs={'_id': 2},
    dag=dag,
)
```

В конвейере TFX вам не нужно определять эти задачи, потому что библиотека TFX выполняет за вас эту операцию. Но эти примеры помогут вам понять, что происходит «за кулисами».

Зависимости задач

В конвейерах машинного обучения задачи зависят друг от друга. Например, задачи по обучению модели требуют, чтобы проверка данных выполнялась до начала обучения. Airflow предоставляет вам множество возможностей для объявления этих зависимостей.

Предположим, что наша задача task_2 зависит от задачи task_1. Вы можете определить зависимость задач следующим образом:

```
task_1.set_downstream(task_2)
```

В Airflow также имеется возможность для обозначения зависимостей задач с помощью оператора битового сдвига:

```
task_1 >> task_2 >> task_X
```

В предыдущем примере мы определили цепочку задач. Каждая из задач будет выполняться в том случае, если предыдущая задача будет выполнена успешно. Если какая-либо из задач не завершится успешно, зависимые задачи не будут выполнены, и Airflow помечает их как пропущенные.

Опять же, в конвейере TFX все это берет на себя библиотека TFX.

Собираем все вместе

После того как мы познакомились со всеми этапами настройки по отдельности, давайте соберем все вместе. В каталоге DAG, размещенном в пути AIRFLOW_HOME (обычно это каталог) `~/airflow/dags`, создайте новый файл `basic_pipeline.py`:

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime, timedelta

project_cfg = {
    'owner': 'airflow',
    'email': ['your-email@example.com'],
    'email_on_failure': True,
    'start_date': datetime(2020, 5, 13),
    'retries': 1,
    'retry_delay': timedelta(hours=1),
}

dag = DAG('basic_pipeline',
          default_args=project_cfg,
          schedule_interval=timedelta(days=1))

def example_task(_id, **kwargs):
    print("Task {}".format(_id))
    return "completed task {}".format(_id)

task_1 = PythonOperator(
    task_id='task_1',
    provide_context=True,
    python_callable=example_task,
    op_kwargs={'_id': 1},
    dag=dag,
)

task_2 = PythonOperator(
    task_id='task_2',
    provide_context=True,
    python_callable=example_task,
    op_kwargs={'_id': 2},
    dag=dag,
)

task_1 >> task_2
```

Вы можете протестировать настройку конвейера, выполнив следующую команду в терминале:

```
python ~/airflow/dags/basic_pipeline.py
```

Наш оператор `print` будет выводить результат в файлы журнала Airflow, а не в терминал. Файл журнала Airflow находится в каталоге, путь к которому указан ниже:

```
~/airflow/logs/NAME OF YOUR PIPELINE/TASK NAME/EXECUTION TIME
```

Чтобы проверить результаты выполнения первой задачи из нашего основного конвейера, мы должны просмотреть файл журнала:

```
$ cat ../logs/basic_pipeline/task_1/2019-09-07T19\.:36\.:18.027474+00\.:00/1.log
```

```
...
[2019-09-07 19:36:25,165] {logging_mixin.py:95} INFO - Task 1  @@ 1 @@
[2019-09-07 19:36:25,166] {python_operator.py:114} INFO - Done. Returned value was:
completed task 1
[2019-09-07 19:36:26,112] {logging_mixin.py:95} INFO - [2019-09-07 19:36:26,112] ❶
[local_task_job.py:105] INFO - Task exited with return code 0
```

- ❶ Наш оператор `print`
- ❷ Наше ответное сообщение после успешного завершения

Чтобы проверить, распознал ли Airflow новый конвейер, вы можете выполнить следующую команду:

```
$ airflow list_dags
```

```
-----
DAGS
-----
basic_pipeline
```

Этот результат показывает, что конвейер был успешно распознан.

Теперь, когда у вас есть понимание основных принципов работы конвейера Airflow, давайте применим его на практике в нашем демонстрационном проекте.

ОРКЕСТРАЦИЯ КОНВЕЙЕРОВ TFX С ПОМОЩЬЮ APACHE AIRFLOW

В этом разделе мы продемонстрируем примеры оркестрации конвейеров TFX с помощью Airflow. Это позволяет нам использовать такие функции, как пользовательский интерфейс Airflow и его возможности планирования, которые очень полезны при развертывании в промышленной среде.

Настройка конвейера

Настройка конвейера TFX с помощью Airflow очень похожа на настройку `BeamDagRunner` для Beam, за исключением того, что нам нужно настроить дополнительные параметры для работы с Airflow.

Вместо того чтобы импортировать `BeamDagRunner`, мы будем использовать `AirflowDAGRunner`. Этот модуль запуска задач имеет дополнительный аргумент, принимающий определенную конфигурацию Apache Airflow (т. е. конфигурацию,

которую мы обсуждали в разделе «Конфигурация Airflow для конкретного проекта» этой главы). `AirflowDagRunner` берет на себя все определения задач и зависимостей, которые мы описали ранее, и нам остается сосредоточиться на настройке нашего конвейера.

Как мы упоминали ранее, файлы для конвейера Airflow должны находиться в каталоге `~/airflow/dags`. Мы также обсудили некоторые общие конфигурации для Airflow, такие как планирование. Для нашего конвейера мы определяем следующие параметры конфигурации:

```
airflow_config = {
    'schedule_interval': None,
    'start_date': datetime.datetime(2020, 4, 17),
    'pipeline_name': 'your_ml_pipeline',
}
```

Как и в примере с Beam, мы инициализируем компоненты и определяем количество рабочих процессов:

```
from tfx.orchestration import metadata, pipeline

def init_pipeline(components, pipeline_root:Text,
                 direct_num_workers:int) -> pipeline.Pipeline:

    beam_arg = [
        "--direct_num_workers={}".format(direct_num_workers),
    ]
    p = pipeline.Pipeline(pipeline_name=pipeline_name,
                          pipeline_root=pipeline_root,
                          components=components,
                          enable_cache=True,
                          metadata_connection_config=metadata,
                          sqlite_metadata_connection_config(metadata_path),
                          beam_pipeline_args=beam_arg)

    return p
```

Затем мы инициализируем конвейер и запускаем его:

```
from tfx.orchestration.airflow.airflow_dag_runner import AirflowDagRunner
from tfx.orchestration.airflow.airflow_dag_runner import AirflowPipelineConfig
from base_pipeline import init_components

components = init_components(data_dir, module_file, serving_model_dir,
                             training_steps=100, eval_steps=100)
pipeline = init_pipeline(components, pipeline_root, 0)
DAG = AirflowDagRunner(AirflowPipelineConfig(airflow_config)).run(pipeline)
```

Опять же, этот код очень похож на код для конвейера Apache Beam, но вместо `BeamDagRunner` мы используем `AirflowDagRunner` и `AirflowPipelineConfig`. Мы инициализируем компоненты, используя пример 11.1, а затем Airflow ищет переменную с именем DAG.

В репозитории GitHub, созданном для этой книги (репозиторий размещен по адресу <https://oreil.ly/bmlp-git>), мы предоставляем контейнер Docker, который позволяет легко запустить наш демонстрационный пример конвейера

с помощью Airflow. Он устанавливает веб-сервер Airflow и планировщик и перемещает файлы в нужные каталоги. Более подробная информация о Docker содержится в приложении А этой книги.

Запуск конвейера

Как мы обсуждали ранее, после того как мы запустили наш веб-сервер Airflow, мы можем открыть пользовательский интерфейс, используя указанный нами порт. Примерный вид пользовательского интерфейса показан на рис. 11.3. Чтобы запустить конвейер, нам нужно включить конвейер, а затем запустить его с помощью кнопки **Trigger DAG**, обозначенной значком **Play**.

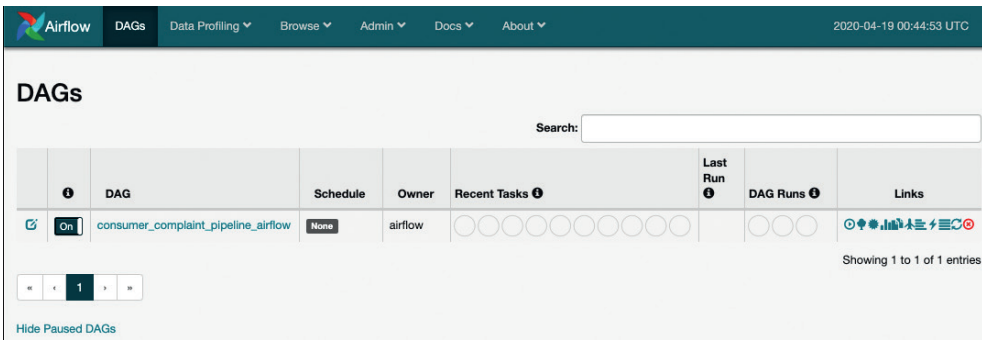


Рис. 11.3. Включение DAG в Airflow

Графическое представление графа конвейера в пользовательском интерфейсе веб-сервера (рис. 11.4) дает нам возможность просмотра зависимостей компонентов и хода выполнения конвейера.

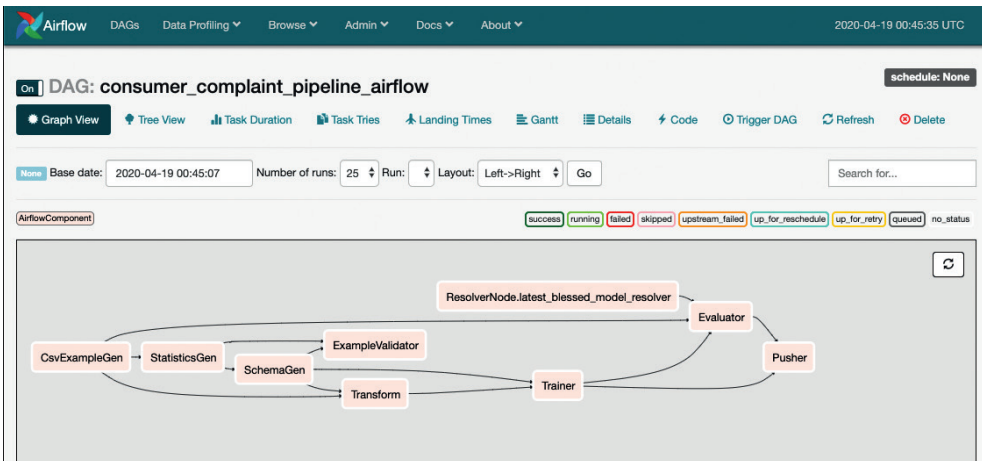


Рис. 11.4. Просмотр графа Airflow

Вам нужно будет обновить страницу браузера, чтобы увидеть обновления, отображающие прогресс работы конвейера. Компоненты, успешно завершив-

шие работу, будут отображаться с зеленой рамкой вокруг их названий, как показано на рис. 11.5. Вы можете просмотреть журналы каждого компонента, щелкнув его название.

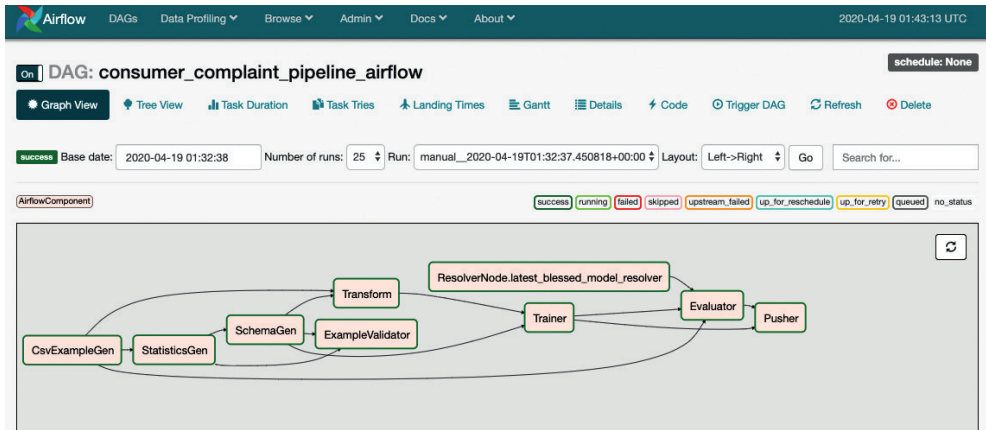


Рис. 11.5. Готовый конвейер в Airflow

Оркестровка конвейеров с помощью Airflow – хороший вариант, если вам нужны сравнительно несложная установка и пользовательский интерфейс или если ваша компания уже использует Airflow. Но если ваша компания уже использует кластеры Kubernetes, в следующей главе описываются конвейеры Kubeflow – гораздо более продвинутый инструмент оркестровки.

РЕЗЮМЕ

В этой главе мы обсудили различные варианты оркестровки конвейеров машинного обучения. Вам нужно выбрать инструмент, который лучше всего подходит для вашей настройки и вашего варианта использования. Мы продемонстрировали, как использовать Apache Beam для запуска конвейера, затем познакомили вас с Airflow и его основами и, наконец, показали, как запустить весь конвейер с помощью Airflow.

В следующей главе мы покажем, как запускать конвейеры с помощью Kubeflow Pipelines на Google AI Platform. Если вы не будете использовать этот вариант, то можете сразу перейти к главе 13, где мы покажем вам, как превратить ваш конвейер в цикл, используя петли обратной связи.

Глава 12

Конвейеры, часть 2: Kubeflow Pipelines

В главе 11 мы обсудили инструменты оркестрации конвейеров машинного обучения – Apache Beam и Apache Airflow. Эти два инструмента оркестровки имеют несколько больших преимуществ: Apache Beam прост в настройке, а Apache Airflow широко применяется для других задач ETL.

В данной главе мы хотим обсудить еще один инструмент оркестрации наших конвейеров – Kubeflow Pipelines. Kubeflow Pipelines позволяет нам выполнять задачи машинного обучения в кластерах Kubernetes, что обеспечивает высокую масштабируемость конвейерного решения. Как рассказывалось в главе 11 и как показано на рис. 12.1, наш инструмент оркестровки выполняет функции координации работы всех компонентов конвейера.

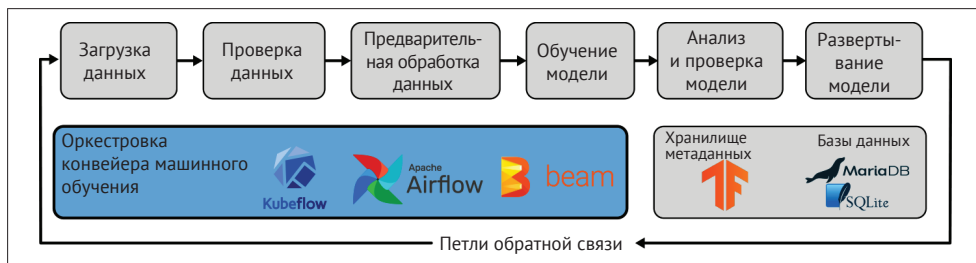


Рис. 12.1. Инструменты оркестрации конвейеров

Настройка Kubeflow Pipelines более сложна, чем установка Apache Airflow или Apache Beam. Но, как мы покажем далее в этой главе, данный инструмент содержит множество полезных функций, в числе которых Pipeline Lineage Browser – браузер, позволяющий просмотреть линии сценариев работы конвейера, TensorBoard Integration – инструмент, обеспечивающий интеграцию с TensorBoard, а также возможность просмотра визуализированных представлений TFDV и TFMA. Кроме того, он использует такие преимущества Kubernetes, как, например, автоматическое масштабирование модулей вычислений, постоянные тома, запросы и ограничения ресурсов.

Эта глава разделена на две части. В первой части мы обсудим, как настраивать и запускать конвейеры с помощью Kubeflow Pipelines.

Приведенный вариант установки не зависит от среды выполнения. Это может быть облачный провайдер, предлагающий управляемые кластеры Kubernetes, или локальная установка Kubernetes.



Введение в Kubernetes

Если вы незнакомы с концепциями и терминологией Kubernetes, мы предлагаем вам начать с приложений к этой книге. Приложение А содержит краткий обзор Kubernetes.

Во второй части главы мы обсудим, как запустить Kubeflow Pipelines, используя платформу Google Cloud AI. Такой вариант характерен для среды Google Cloud. Этот облачный провайдер берет на себя большую часть задач, связанных с инфраструктурой, и позволяет использовать Dataflow для простого масштабирования задач, предполагающих работу с данными (например, задач предварительной обработки данных). Мы рекомендуем этот путь, если вы хотите использовать Kubeflow Pipelines, но не хотите тратить время на управление инфраструктурой Kubernetes.

ВВЕДЕНИЕ В KUBEFLOW PIPELINES

Kubeflow Pipelines – это инструмент оркестрации на основе Kubernetes, ориентированный на машинное обучение. В то время как Apache Airflow был разработан для процессов ETL, в основе Kubeflow Pipelines лежит сквозное выполнение конвейеров машинного обучения.

Kubeflow Pipelines предоставляет удобный пользовательский интерфейс для мониторинга работы конвейеров машинного обучения. Этот интерфейс – центральное связующее звено для специалистов по обработке данных (мы расскажем об этом подробнее в разделе «Полезные функции Kubeflow Pipelines»), а также способ планирования запусков непрерывных построений модели. Кроме того, Kubeflow Pipelines предоставляет собственный пакет средств разработки программного обеспечения (Software Development Kit, SDK) для создания контейнеров Docker для построения и оркестровки контейнеров. Доменно-ориентированный язык (domain-specific language, DSL) Kubeflow Pipeline обеспечивает большую гибкость в настройке шагов конвейера, но также требует более высокой степени координации между компонентами. Мы считаем, что конвейеры TFX ведут к более высокому уровню стандартизации конвейеров и, следовательно, позволяют снизить число ошибок. Если вас интересуют более подробные сведения о пакете SDK Kubeflow Pipelines, мы можем порекомендовать литературу, приведенную в разделе «Kubeflow или Kubeflow Pipelines?» этой главы.

Когда мы настраиваем Kubeflow Pipelines, как будет показано в разделе «Установка и начальная настройка», Kubeflow Pipelines установит различные инструменты, включая пользовательский интерфейс, контроллер рабочего процесса, экземпляр базы данных MySQL и хранилище метаданных ML MetadataStore, о котором рассказывалось в разделе «Что такое метаданные ML Metadata?» главы 2.

Когда мы запустим наш конвейер TFX с Kubeflow Pipelines, вы заметите, что каждый компонент запускается как собственный модуль Kubernetes. Как пока-

зано на рис. 12.2, каждый компонент подключается к центральному хранилищу метаданных в кластере и может загружать артефакты либо из постоянного тома хранилища кластера Kubernetes, либо из облачного хранилища. Все выходные данные компонентов (например, статистика данных, собранная при выполнении TFDV или для экспортированных моделей) регистрируются в хранилище метаданных и сохраняются как артефакты на постоянном томе или в сегменте облачного хранилища.

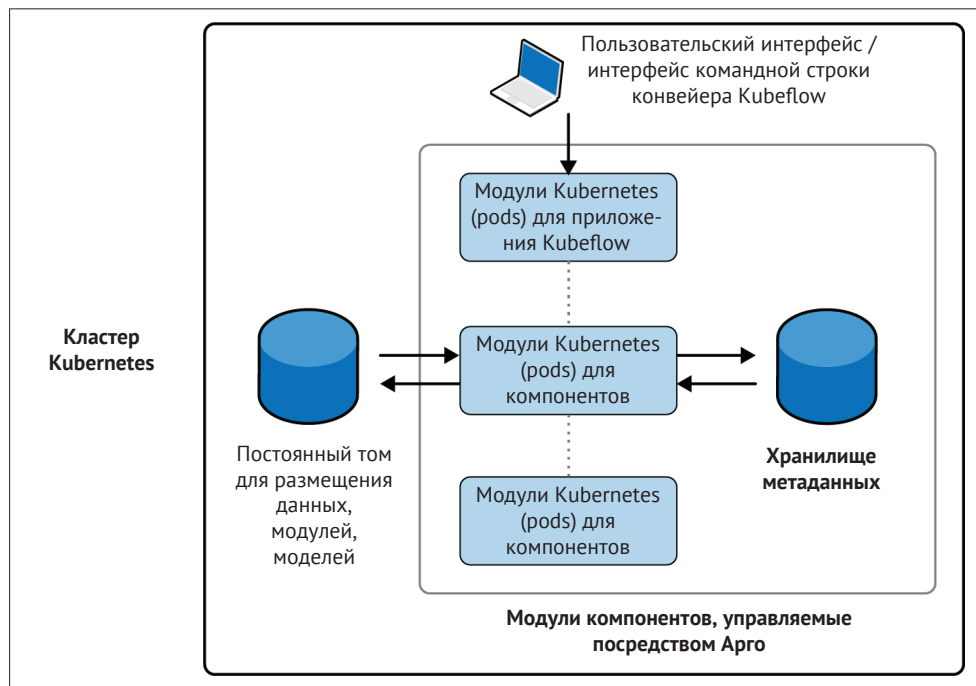


Рис. 12.2. Обзор Kubeflow Pipelines

Kubeflow или Kubeflow Pipelines?

Kubeflow Pipelines и Kubeflow часто путают. Kubeflow – это набор проектов с открытым исходным кодом, который включает в себя множество инструментов машинного обучения, включая TFJob для обучения машинных моделей, Katib для оптимизации гиперпараметров модели и KFServing для развертывания моделей машинного обучения. Kubeflow Pipelines – еще один из проектов пакета Kubeflow, предназначенный для решения задач развертывания и управления сквозными рабочими процессами машинного обучения. В этой главе мы сосредоточимся только на установке и эксплуатации Kubeflow Pipelines. Если вас интересует более глубокое знакомство с Kubeflow, мы рекомендуем документацию по проекту.

Кроме того, мы можем порекомендовать две книги по Kubeflow:

- «Руководство по эксплуатации Kubeflow» Джоша Паттерсона и др. (Kubeflow Operations Guide by Josh Patterson et al., O'Reilly);
- «Kubeflow для машинного обучения» Холдена Караяу и др. (Kubeflow for Machine Learning (forthcoming) by Holden Karau et al., O'Reilly) – эта книга готовится к выпуску.

Как будет показано в данной главе, Kubeflow Pipelines предоставляет хорошо масштабируемый способ запуска конвейеров машинного обучения. Kubeflow Pipelines незаметно запускает Argo для оркестровки зависимостей отдельных компонентов. Благодаря такой оркестрации при помощи Argo оркестровка нашего конвейера будет основываться на другом рабочем процессе, как мы уже упоминали в главе 11. Мы рассмотрим рабочий процесс оркестрации Kubeflow Pipelines в разделе «Оркестровка конвейеров TFX с помощью Kubeflow Pipelines» этой главы.

Что такое Argo?

Argo – это набор инструментов для управления рабочими процессами, развертыванием и задачами непрерывного развертывания приложений. Первоначально разработанный для управления задачами DevOps, он также является отличным менеджером рабочих процессов машинного обучения. Argo управляет всеми задачами как контейнерами в среде Kubernetes. Для дополнительной информации обратитесь к постоянно пополняемой документации Argo (см. <https://oreil.ly/K2R5H>).

Установка и начальная настройка

Kubeflow Pipelines работает внутри кластера Kubernetes. Здесь мы предполагаем, что у вас есть кластер Kubernetes, конфигурация которого включает как минимум 16 ГБ оперативной памяти и 8 ЦП в пуле вычислительных узлов, и что вы настроили `kubectl` для подключения к недавно созданному кластеру Kubernetes.

Создайте кластер Kubernetes

Элементарные примеры настройки кластера Kubernetes на локальном компьютере или у облачного провайдера, такого как Google Cloud, приводятся в приложениях А и В. Из-за требований к ресурсам, предъявляемых Kubeflow Pipelines, предпочтительным вариантом является использование Kubernetes с облачным провайдером. Вот несколько примеров управляемых сервисов Kubernetes, предоставляемых облачными провайдерами:

- 1) Amazon Elastic Kubernetes Service (Amazon EKS);
- 2) Google Kubernetes Engine (GKE);
- 3) служба Microsoft Azure Kubernetes (AKS);
- 4) служба IBM Kubernetes.

Для того чтобы глубже погрузиться в архитектуру Kubeflow – Kubernetes, мы настоятельно рекомендуем книгу «Работа с Kubernetes» Брендана Бернса и др. (Kubernetes: Up and Running by Brendan Burns et al., O'Reilly).

Для оркестровки нашего конвейера мы устанавливаем Kubeflow Pipelines как отдельное приложение и без дополнительных инструментов, которые являются частью проекта Kubeflow. С помощью следующих команд `bash` мы можем настроить нашу автономную установку Kubeflow Pipelines. Полная установка может занять около пяти минут.

```
$ export PIPELINE_VERSION=0.5.0
$ kubectl apply -k "github.com/kubeflow/pipelines/manifests/"\
  "kustomize/cluster-scoped-resources?ref=$PIPELINE_VERSION"
customresourcedefinition.apiextensions.k8s.io/
  applications.app.k8s.io created
...
clusterrolebinding.rbac.authorization.k8s.io/
  kubeflow-pipelines-cache-deployer-clusterrolebinding created

$ kubectl wait --for condition=established \
  --timeout=60s crd/applications.app.k8s.io/
customresourcedefinition.apiextensions.k8s.io/
  applications.app.k8s.io condition met

$ kubectl apply -k "github.com/kubeflow/pipelines/manifests/"\
  "kustomize/env/dev?ref=$PIPELINE_VERSION"
```

Вы можете проверить, как идет установка, выведя на экран информацию о созданных модулях (pods):

```
$ kubectl -n kubeflow get pods
```

NAME	READY	STATUS	AGE
cache-deployer-deployment-c6896d66b-62gc5	0/1	Pending	90s
cache-server-8869f945b-4k7qk	0/1	Pending	89s
controller-manager-5cbdfbc5bd-bnfx	0/1	Pending	89s
...			

Через несколько минут состояние всех модулей должно измениться, и в столбце STATUS должно появиться значение Running. Если в вашем конвейере возникают какие-либо ошибки (например, недостаточно вычислительных ресурсов), состояние модулей укажет на ошибку:

```
$ kubectl -n kubeflow get pods
```

NAME	READY	STATUS	AGE
cache-deployer-deployment-c6896d66b-62gc5	1/1	Running	4m6s
cache-server-8869f945b-4k7qk	1/1	Running	4m6s
controller-manager-5cbdfbc5bd-bnfx	1/1	Running	4m6s
...			

Информацию об отдельных модулях можно получить с помощью команды

```
kubectl -n kubeflow describe pod <pod name>
```



Управляемая установка Kubeflow Pipelines

Если вы хотите поэкспериментировать с Kubeflow Pipelines, Google Cloud предоставляет возможности управляемой установки на платформе Google Cloud. В разделе «Конвейеры на основе платформы искусственного интеллекта Google Cloud» этой главы мы подробно обсудим, как запускать конвейеры TFX на платформе искусственного интеллекта Google Cloud и как настраивать Kubeflow Pipelines, используя инструменты Google Cloud Marketplace.

Доступ к установленному экземпляру Kubeflow Pipelines

Если установка завершилась успешно, независимо от вашего облачного провайдера или службы Kubernetes вы можете получить доступ к пользовательскому интерфейсу Kubeflow Pipelines, настроив переадресацию порта с помощью Kubernetes:

```
$ kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80
```

При работающем перенаправлении порта вы можете получить доступ к конвейерам Kubeflow в своем браузере, перейдя по адресу <http://localhost:8080>. Для производственных сценариев использования необходимо создать балансировщик нагрузки для службы Kubernetes.

Пользователи Google Cloud могут получить доступ к Kubeflow Pipelines, получив доступ к общедоступному домену, созданному для вашей установки Kubeflow. Вы можете получить URL-адрес, выполнив команду

```
$ kubectl describe configmap inverse-proxy-config -n kubeflow \
| grep googleusercontent.com
<id>-dot-<region>.pipelines.googleusercontent.com
```

Затем вы можете получить доступ к предоставленному URL-адресу в любом браузере. Если все работает, вы увидите панель мониторинга Kubeflow Pipelines или целевую страницу, как показано на рис. 12.3.

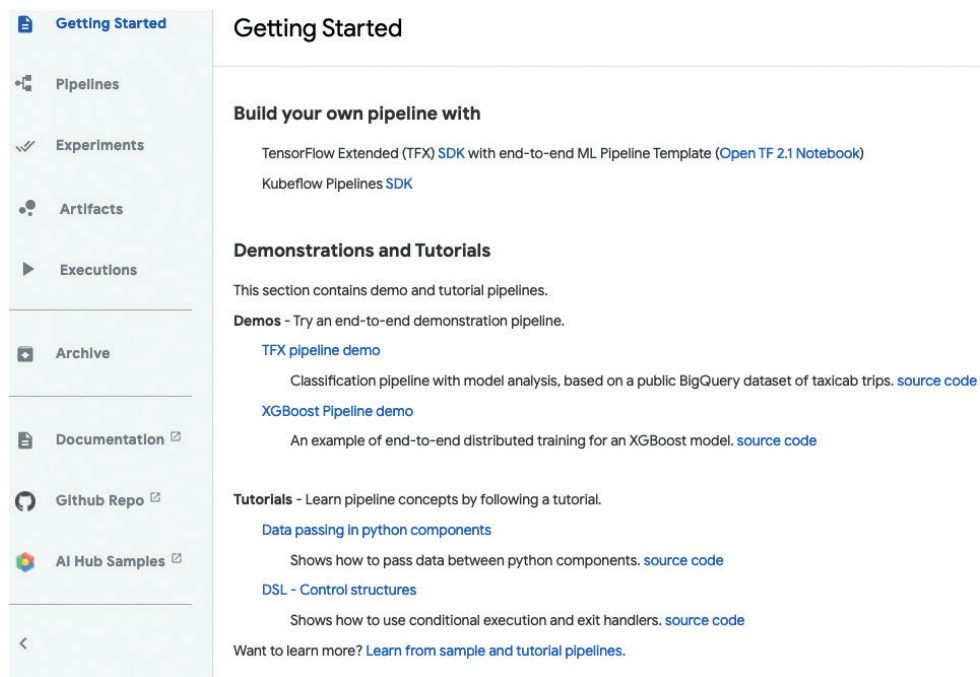


Рис. 12.3. Начало работы с Kubeflow Pipelines

Установив и запустив Kubeflow Pipelines, мы можем сосредоточиться на том, как запускать конвейеры. В следующем разделе мы обсудим оркестровку конвейера и рабочий процесс, начиная с TFX и заканчивая Kubeflow Pipelines.

ОРКЕСТРАЦИЯ КОНВЕЙЕРОВ TFX С ПОМОЩЬЮ KUBEFLOW PIPELINES

В предыдущих разделах мы обсуждали, как настроить приложение Kubeflow Pipelines в Kubernetes. В этом разделе мы опишем, как запускать ваши конвейеры в Kubeflow Pipelines, и рассмотрим случай их запуска только в ваших кластерах Kubernetes. Таким образом, конвейер может работать на кластерах независимо от поставщика облачных услуг. В разделе «Конвейеры на основе платформы Google Cloud AI Platform» мы покажем, как можем воспользоваться преимуществами управляемой облачной службы, такой как Dataflow от GCP, для расширения ваших конвейеров за пределы кластера Kubernetes.

Прежде чем мы погрузимся в подробности организации конвейеров машинного обучения с помощью Kubeflow Pipelines, мы хотим ненадолго вернуться назад. Весь рабочий процесс, начиная с кода TFX до запуска вашего конвейера, окажется немного сложнее, чем те примеры, которые мы обсуждали в главе 11, поэтому мы начнем с обзора полной картины. На рис. 12.4 показана общая архитектура.

Аналогично тому, как это было в случае с Airflow и Beam, нам все еще нужен скрипт Python, который определяет компоненты TFX в нашем конвейере. Мы будем повторно использовать сценарий примера 11.1 из главы 11. В отличие от работы исполнителей Apache Beam или Airflow TFX, исполнитель Kubeflow не запускает конвейер, а скорее генерирует файлы конфигурации для запуска на настроенном Kubeflow.

Как показано на рис. 12.4, TFX KubeflowRunner преобразует наши сценарии Python TFX со всеми спецификациями компонентов в инструкции Argo, которые затем можно выполнить с помощью Kubeflow Pipelines. Argo запускает каждый компонент TFX как собственный модуль Kubernetes и запускает TFX Executor для конкретного компонента в контейнере.

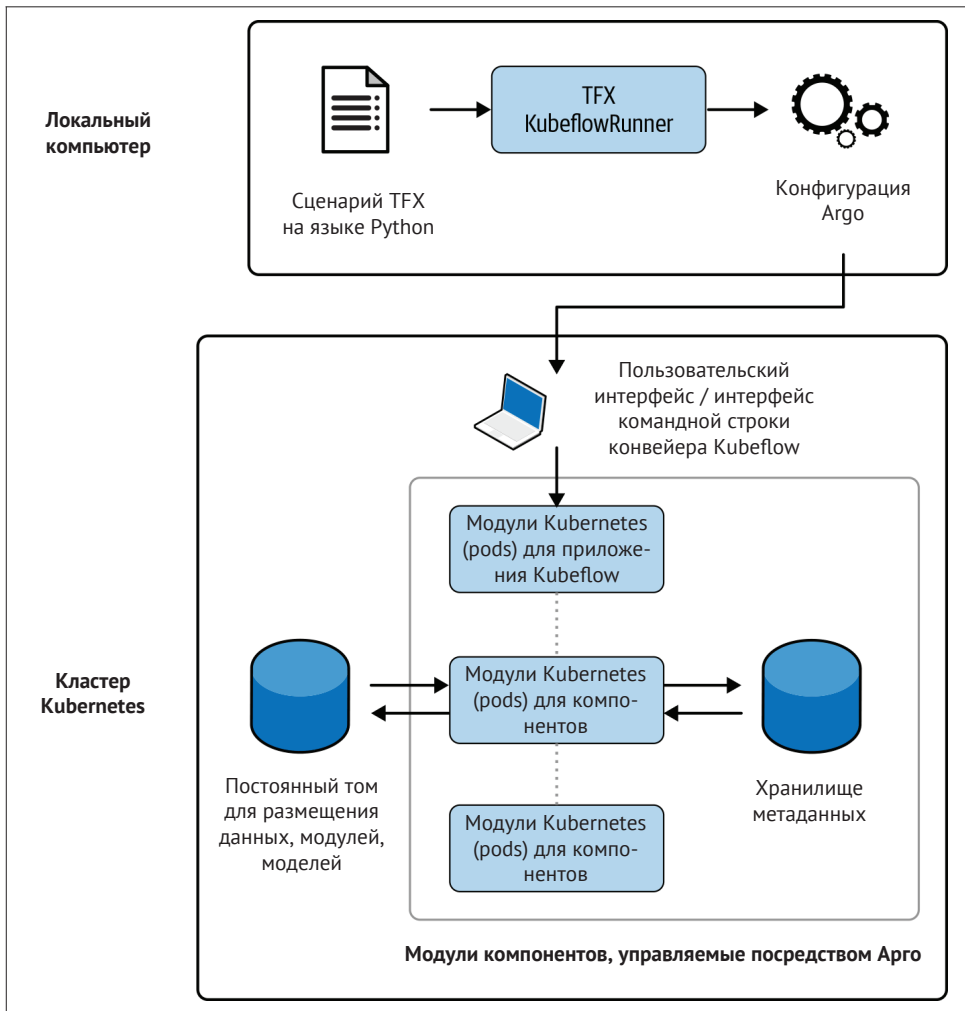


Рис. 12.4. Рабочий процесс от сценария TFX до Kubeflow Pipelines



Пользовательские образы контейнеров TFX

Образ TFX, используемый для всех контейнеров компонентов, должен включать все необходимые пакеты Python. Образ TFX по умолчанию содержит последнюю версию TensorFlow и базовые пакеты. Если вашему конвейеру требуются дополнительные пакеты, вам нужно будет создать собственный образ контейнера TFX и указать его в файле `KubeflowDagRunnerConfig`. Мы показываем, как это сделать, в приложении C.

Все компоненты должны иметь возможность чтения и записи в файловую систему вне самого контейнера исполнителя. Например, компонент загрузки данных должен иметь возможность считать данные из файловой системы, или окончательная модель должна быть перемещена модулем Pusher в определенное место. Было бы непрактично иметь возможность чтения и записи только внутри контейнера компонентов; поэтому мы рекомендуем хранить артефакты на жестких дисках, к которым могут получить доступ все компоненты (например, в сегментах облачного хранилища или постоянных томах в кластере Kubernetes). Если вас интересует, как настроить постоянный том, ознакомьтесь с разделом «Обмен данными через постоянные тома» в приложении C.

Настройка конвейера

Вы можете хранить свои обучающие данные, модуль Python и артефакты конвейера в сегменте облачного хранилища или на постоянном томе; это зависит от ваших предпочтений. Вашему конвейеру просто нужен доступ к файлам. Если вы выбираете вариант чтения и записи данных из/в сегменты облачного хранилища, убедитесь, что ваши компоненты TFX имеют все необходимые облачные учетные данные для работы в кластере Kubernetes.

Имея все файлы и настроенный пользователем образ TFX для наших контейнеров конвейеров машинного обучения (при необходимости), теперь мы можем «собрать» сценарий TFX Runner для генерации инструкций Argo YAML, необходимый для работы Kubeflow Pipelines¹.

Как уже обсуждалось в главе 11, мы можем повторно использовать функцию `init_components` для генерации наших компонентов. Это позволяет нам сосредоточиться на конфигурации, специфичной для Kubeflow.

Для начала давайте настроим путь к файлу для кода нашего модуля Python, необходимого для запуска компонентов `Transform` и `Trainer`. Кроме того, мы зададим путь к каталогам для наших необработанных обучающих данных, артефактов конвейера и для места, где должна храниться наша обученная модель. В следующем примере мы покажем вам, как смонтировать постоянный том с помощью TFX:

```
import os
pipeline_name = 'consumer_complaint_pipeline_kubeflow'
persistent_volume_claim = 'tfx-pvc'
persistent_volume = 'tfx-pv'
```

¹ Вы можете использовать сценарий, который генерирует инструкции Argo YAML, размещенный в репозитории GitHub книги (см. <https://oreil.ly/bmlp-gitkubeflowpy>).

```

persistent_volume_mount = '/tfx-data'

# Pipeline inputs
data_dir = os.path.join(persistent_volume_mount, 'data')
module_file = os.path.join(persistent_volume_mount, 'components', 'module.py')

# Pipeline outputs
output_base = os.path.join(persistent_volume_mount, 'output', pipeline_name)
serving_model_dir = os.path.join(output_base, pipeline_name)

```

Если вы решите воспользоваться услугами поставщиков облачных хранилищ, корневым каталогом структуры папок будет сегмент облачного хранилища, как показано в следующем примере:

```

import os
...
bucket = 'gs://tfx-demo-pipeline'

# Pipeline inputs
data_dir = os.path.join(bucket, 'data')
module_file = os.path.join(bucket, 'components', 'module.py')
...

```

Определив пути к файлам, теперь мы можем настроить наш `KubeflowDagRunnerConfig`. Для настройки TFX в нашей установке Kubeflow Pipelines важны три аргумента:

`kubeflow_metadata_config`

Kubeflow запускает базу данных MySQL внутри кластера Kubernetes. Вызов `get_default_kubeflow_metadata_config()` вернет информацию о базе данных, предоставленную кластером Kubernetes. Если вы хотите использовать управляемую базу данных (например, AWS RDS или Google Cloud Databases), то можете перезаписать данные подключения с помощью этого аргумента;

`tfx_image`

URI образа указывать необязательно. Если URI не определен, TFX установит образ, соответствующий версии TFX, запускающей runner. В нашем демонстрационном примере мы устанавливаем в качестве URI путь к образу в реестре контейнеров (например, `gcr.io/oreilly-book/ml-pipelines-tfx-custom:0.22.0`);

`pipeline_operator_funcs`

Этот аргумент обращается к списку, содержащему информацию о конфигурации, необходимой для запуска TFX внутри Kubeflow Pipelines (например, к имени службы и порту сервера gRPC). Поскольку эта информация может быть предоставлена Kubernetes ConfigMap¹, функция `get_default_pipeline_operator_funcs` прочитает ConfigMap и предоставит необходимую информацию аргументу `pipeline_operator_funcs`. В нашем примере проекта мы будем вручную монтировать постоянный том с данными нашего проекта; поэтому нам нужно добавить в список следующую информацию:

¹ Дополнительные сведения о Kubernetes ConfigMaps см. в разделе «Некоторые определения Kubernetes» приложения А.

```

from kfp import onprem
from tfx.orchestration.kubeflow import kubeflow_dag_runner

...
PROJECT_ID = 'oreilly-book'
IMAGE_NAME = 'ml-pipelines-tfx-custom' TFX_VERSION = '0.22.0'
metadata_config = \
    kubeflow_dag_runner.get_default_kubeflow_metadata_config() ❶
pipeline_operator_funcs = \
    kubeflow_dag_runner.get_default_pipeline_operator_funcs() ❷
pipeline_operator_funcs.append( ❸
    onprem.mount_pvc(persistent_volume_claim,
                    persistent_volume,
                    persistent_volume_mount))
runner_config = kubeflow_dag_runner.KubeflowDagRunnerConfig(
    kubeflow_metadata_config=metadata_config,
    tfx_image="gcr.io/{}/{}:{}".format(
        PROJECT_ID, IMAGE_NAME, TFX_VERSION), @@ 4 @@
    pipeline_operator_funcs=pipeline_operator_funcs
)

```

- ❶ Получите конфигурацию метаданных по умолчанию
- ❷ Получите функции OpFunc по умолчанию
- ❸ Смонтируйте тома, добавив их к функциям OpFunc
- ❹ При необходимости добавьте собственный образ TFX

Функции OpFunc

Функции OpFunc позволяют нам указывать специфичную для кластера информацию, которая важна для работы нашего конвейера. Эти функции позволяют нам взаимодействовать с объектами цифровой абонентской линии (Digital Subscriber Line DSL) в Kubeflow Pipelines. Функции OpFunc принимают DSL-объект Kubeflow Pipelines *dsl.ContainerOp* в качестве входных данных, применяют дополнительные функции и возвращают тот же объект.

Два распространенных случая добавления функций OpFunc к вашей `pipeline_operator_funcs` – это запрос минимума памяти или указание графических процессоров для работы контейнера. Но функции OpFunc также позволяют указывать учетные данные для облачных провайдеров или запрашивать TPU (для случая Google Cloud).

Давайте рассмотрим два наиболее распространенных варианта использования функций OpFunc: установка минимального лимита памяти для запуска контейнеров компонентов TFX и запрос графических процессоров для выполнения всех компонентов TFX. В следующем примере мы устанавливаем минимальные ресурсы памяти, необходимые для запуска каждого контейнера компонентов, равными 4 ГБ:

```
def request_min_4G_memory():
    def _set_memory_spec(container_op):
        container_op.set_memory_request('4G')
    return _set_memory_spec
...
pipeline_operator_funcs.append(request_min_4G_memory())
```

Функция получает объект `container_op`, устанавливает предел и возвращает саму функцию.

Мы можем запросить графический процессор для запуска наших контейнеров компонентов TFX таким же образом, как это показано в следующем примере. Если вам требуются графические процессоры для выполнения вашего контейнера, ваш конвейер будет работать только в том случае, если графические процессоры доступны и настроены в вашем кластере Kubernetes¹:

```
def request_gpu():
    def _set_gpu_limit(container_op):
        container_op.set_gpu_limit('1')
    return _set_gpu_limit
...
pipeline_op_funcs.append(request_gpu())
```

SDK Kubeflow Pipelines предоставляет общие функции `OpFunc` для каждого крупного облачного провайдера. В следующем примере показано, как добавить учетные данные AWS в контейнеры компонентов TFX:

```
from kfp import aws
...
pipeline_op_funcs.append(
    aws.use_aws_secret()
)
```

Функция `use_aws_secret()` предполагает, что `AWS_ACCESS_KEY_ID` и `AWS_SECRET_ACCESS_KEY` зарегистрированы как объекты `secret` Kubernetes в кодировке `base64`². Аналогичная функция для учетных данных Google Cloud называется `use_gcp_secrets()`.

Установив `runner_config`, мы можем инициализировать компоненты и запустить `KubeflowDagRunner`. Но вместо запуска конвейера компонент запуска (`runner`) выдаст конфигурацию Argo, которую мы загрузим в Kubeflow Pipelines в следующем разделе:

¹ Информация об установке последних версий драйверов для кластеров Kubernetes доступна на сайте Nvidia (см. <https://oreil.ly/HGj50>).

² Информация о секретах Kubernetes и о том, как их настроить, доступна в документации, размещенной по ссылке <https://oreil.ly/AxcHf>.

```

from tfx.orchestration.kubeflow import kubeflow_dag_runner
from pipelines.base_pipeline import init_components, init_pipeline ❶

components = init_components(data_dir, module_file, serving_model_dir,
                             training_steps=50000, eval_steps=15000)
p = init_pipeline(components, output_base, direct_num_workers=0)

output_filename = "{}.yaml".format(pipeline_name)
kubeflow_dag_runner.KubeflowDagRunner(config=runner_config,
                                     output_dir=output_dir, ❷
                                     output_filename=output_filename).run(p)

```

❶ Повторно используйте базовые модули для компонентов

❷ Необязательный аргумент

Аргументы `output_dir` и `output_filename` необязательны. Если они не указаны, конфигурация Argo будет представлена в виде сжатого файла `tar.gz`, размещенного в том же каталоге, из которого мы запустили следующий скрипт python. Для большей наглядности примера мы выбрали выходной формат YAML и указали конкретный выходной путь.

После выполнения следующей команды вы найдете файл конфигурации Argo configuration `consumer_complaint_pipeline_kubeflow.yaml` в каталоге `pipelines/kubeflow_pipelines/argo_pipeline_files/`:

```
$ python pipelines/kubeflow_pipelines/pipeline_kubeflow.py
```

Запуск конвейера

После того как мы выполнили необходимые настройки, мы можем получить доступ к панели инструментов Kubeflow Pipelines. Если вы хотите создать новый конвейер, нажмите кнопку **Upload pipeline** для загрузки конвейера, как показано на рис. 12.5. В качестве альтернативы вы можете выбрать существующий конвейер и загрузить новую версию.

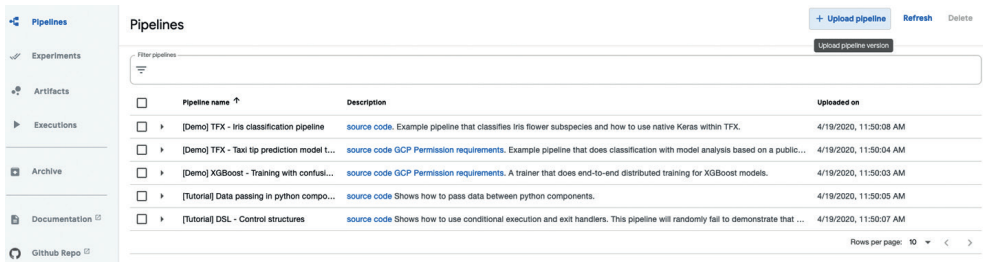


Рис. 12.5. Обзор загруженных конвейеров

Выберите конфигурацию Argo, как показано на рис. 12.6.

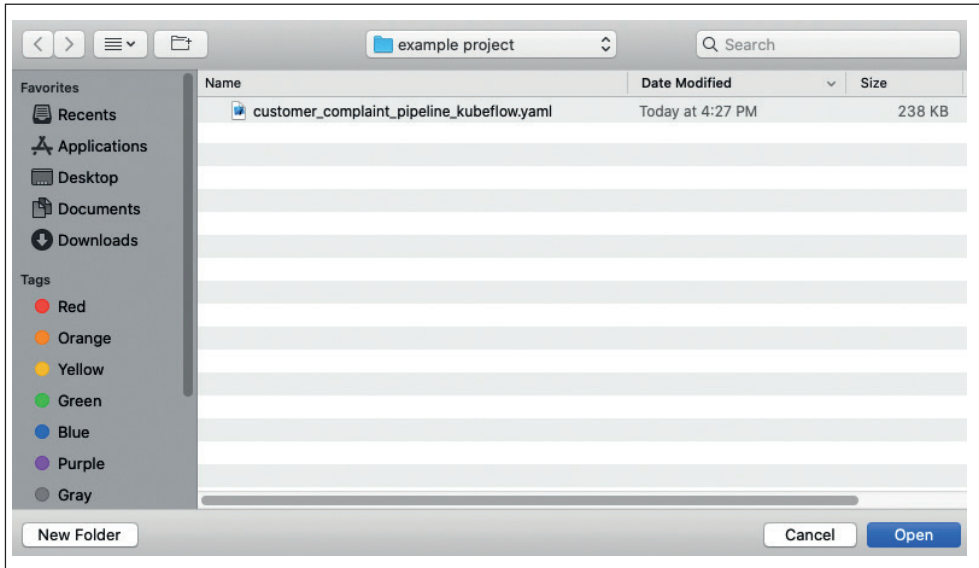


Рис. 12.6. Выбор сгенерированного файла конфигурации Argo

Kubeflow Pipelines визуализирует зависимости ваших компонентов. Если вы хотите запустить новый цикл конвейера, выберите **Create run**, как показано на рис. 12.7.

Теперь вы можете настроить работу конвейера. Конвейеры можно запускать однократно или повторно (например, с помощью задачи cron). Kubeflow Pipelines также позволяет группировать задачи запуска конвейера, объединяя их в *экспериментах*.

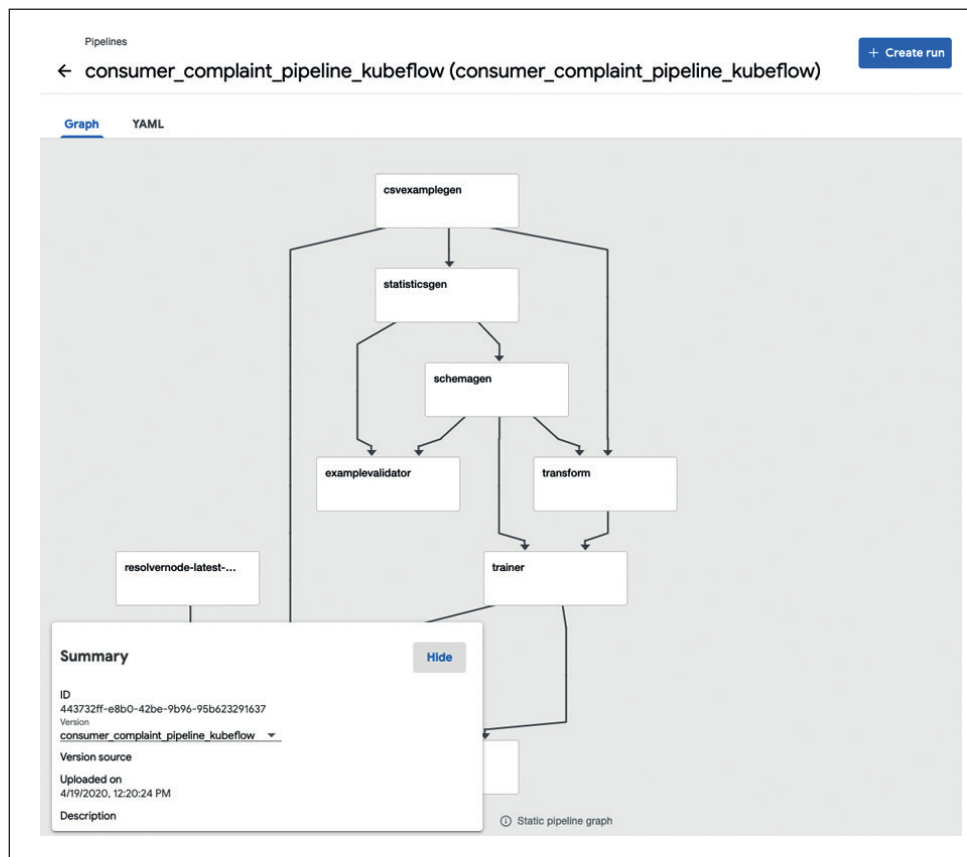


Рис. 12.7. Запуск конвейера

Как только вы нажмете **Start**, как показано на рис. 12.8, Kubeflow Pipelines с помощью Argo запустится и развернет модуль для каждого контейнера, основываясь на вашем прямом графе компонентов. Когда все условия для компонента будут выполнены, модуль компонента будет развернут и запустит исполнитель компонента.

Если вы хотите получить подробную информацию о выполнении текущего запуска, вы можете щелкнуть **Run name**, как показано на рис. 12.9.

Pipelines

Experiments

Artifacts

Executions

Archive

Documentation

Github Repo

AI Hub Samples

Experiments

← Start a run

Run details

Pipeline *

consumer_complaint_pipeline_kubeflow

Choose

Pipeline Version *

consumer_complaint_pipeline_kubeflow

Choose

Run name *

Run of consumer_complaint_pipeline_kubeflow (5aaf5)

Description (optional)

This run will be associated with the following experiment

Experiment *

Choose

Run Type

☒ One-off
 ☐ Recurring

Run parameters

Specify parameters required by the pipeline

pipeline-root

/tfx-data/output/consumer_complaint_pipeline_kubeflow

Start

Cancel

Рис. 12.8. Информация об определенном запуске конвейера

Experiments

+ Create run

+ Create experiment

All experiments

All runs

Filter runs

<input type="checkbox"/>	Run name	Status	Duration	Experiment	Pipeline Version
<input type="checkbox"/>	Run of consumer_complaint_pipeline_kubeflow (...)		-	Default	consumer_complaint_pipeline_...

Рис. 12.9. Запуск конвейера в процессе

Теперь вы можете проверять компоненты во время или после их запуска и работы. Например, вы можете просмотреть файлы журнала для определенного компонента, если компонент вышел из строя. На рис. 12.10 показан пример, в котором для компонента Transform отсутствует библиотека Python. Отсутствующие библиотеки можно добавить в собственный образ контейнера TFX, как описано в приложении С.

Experiments > Default

← ● Run of consumer_complaint_pipeline_kubeflow_version_at_2020-04-19T01:53:08.472Z (c2759)

Graph Run output Config

consumer-complaint-pipeline-kubeflow

This step is in Failed state with this message: failed with exit code 1

Artifacts Input/Output Volumes Manifest **Logs** Pod Events

Logs can also be viewed in Stackdriver Kubernetes Monitoring.

```

1 2020-04-19 01:55:09.632860: W tensorflow/stream_executor/platform/default/dso_loader.cc:
2 2020-04-19 01:55:09.632830: W tensorflow/stream_executor/platform/default/dso_loader.cc:
3 2020-04-19 01:55:09.632975: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] Cann
4 INFO:absl:Using setup_file '/tfx-src/setup.py' to capture TFX dependencies
5 INFO:absl:Running driver for Transform
6 INFO:absl:MetadataStore with gRPC connection initialized
7 WARNING:absl:Could not find matching artifact class for type 'Examples' (proto: 'id: 8\n
8 WARNING:absl:Could not find matching artifact class for type 'Schema' (proto: 'id: 12\n
9 INFO:absl:Adding KFP pod name consumer-complaint-pipeline-kubeflow-2fsg-2030360007 to e
10 INFO:absl:Running executor for Transform
11 INFO:absl:Nonempty beam args setup file already includes dependency
12 WARNING:tensorflow:From /tfx-src/tfx/components/transform/executor.py:511: Schema (from
13 Instructions for updating:
14 Schema is a deprecated, use schema_utils.schema_from_feature_spec to create a 'Schema'
15 Traceback (most recent call last):
16   File "/tfx-src/tfx/orchestration/kubeflow/container_entrypoint.py", line 382, in <modu
17     main()
18   File "/tfx-src/tfx/orchestration/kubeflow/container_entrypoint.py", line 375, in main
19     execution_info = launcher.launch()
20   File "/tfx-src/tfx/orchestration/launcher/base_component_launcher.py", line 205, in la
21     execution_decision.exec_properties)
22   File "/tfx-src/tfx/orchestration/launcher/in_process_component_launcher.py", line 67,
23     executor.Do(input_dict, output_dict, exec_properties)
24   File "/tfx-src/tfx/components/transform/executor.py", line 391, in Do
25     self.Transform(label_inputs, label_outputs, status_file)
26   File "/tfx-src/tfx/components/transform/executor.py", line 834, in Transform
27     preprocessing_fn = self._GetPreprocessingFn(inputs, outputs)
28   File "/tfx-src/tfx/components/transform/executor.py", line 770, in _GetPreprocessingFn
29     'preprocessing_fn')
30   File "/tfx-src/tfx/utils/import_utils.py", line 67, in import_func_from_source
31     loader.exec_module(user_module)
32   File "<frozen importlib._bootstrap_external>", line 678, in exec_module
33   File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
34   File "/tfx-data/components/module.py", line 5, in <module>
35     import tensorflow_hub as hub
36 ModuleNotFoundError: No module named 'tensorflow_hub'
37

```

Рис. 12.10. Просмотр информации об отказе компонента

Успешный запуск конвейера показан на рис. 12.11. После завершения запуска вы можете найти проверенную и экспортированную модель машинного обучения в каталоге файловой системы, путь к которому задан в компоненте Pusher. В нашем примере мы поместили модель в каталог, путь к которому `/tfx-data/output/consumer_complaint_pipeline_kubeflow/`, размещенный на постоянном томе.

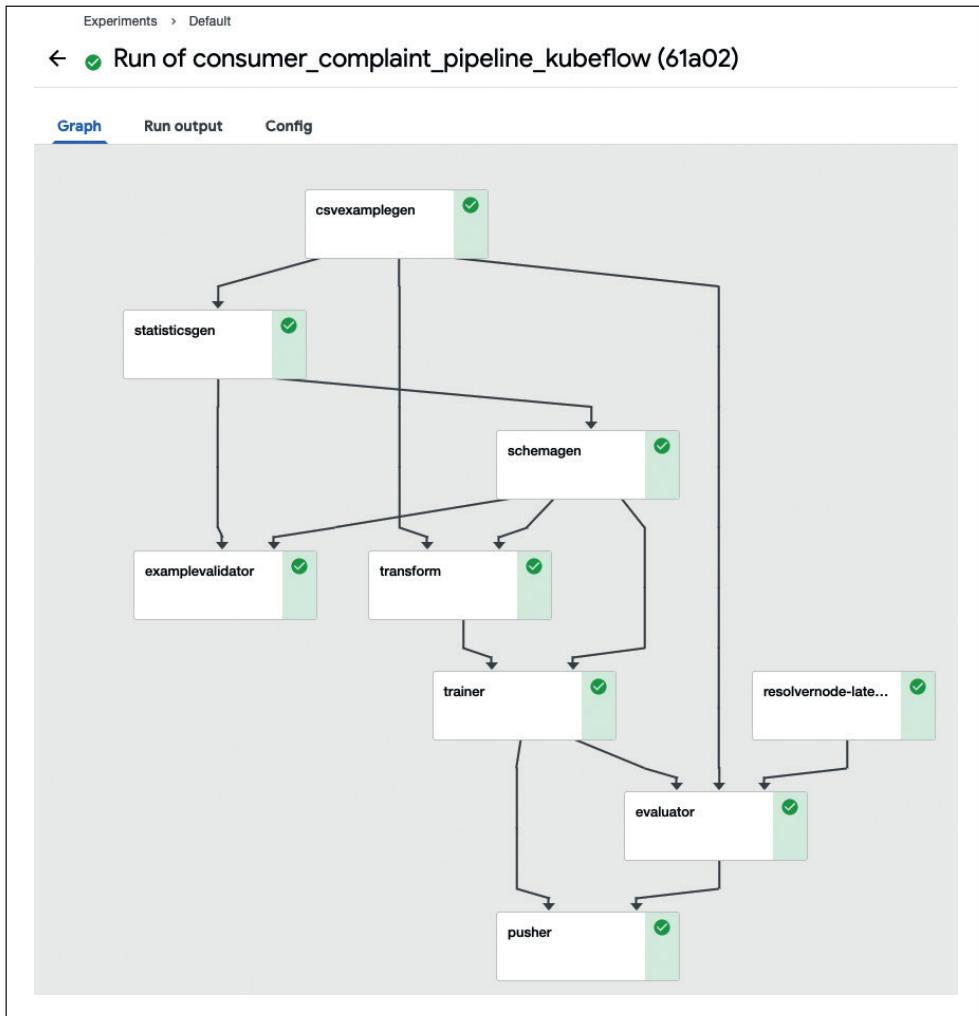


Рис. 12.11. Успешный запуск конвейера

Вы также можете проверить состояние вашего конвейера с помощью инструмента `kubectl`. Поскольку каждый компонент работает как отдельный модуль, все модули, для которых префикс имени совпадает с именем конвейера, должны иметь состояние *Completed*:

```
$ kubectl -n kubeflow get pods
```

NAME	READY	STATUS	AGE
cache-deployer-deployment-c6896d66b-gmkqf	1/1	Running	28m
cache-server-8869f945b-lb8tb	1/1	Running	28m
consumer-complaint-pipeline-kubeflow-nmvzb-1111865054	0/2	Completed	10m
consumer-complaint-pipeline-kubeflow-nmvzb-1148904497	0/2	Completed	3m38s
consumer-complaint-pipeline-kubeflow-nmvzb-1170114787	0/2	Completed	9m
consumer-complaint-pipeline-kubeflow-nmvzb-1528408999	0/2	Completed	5m43s
consumer-complaint-pipeline-kubeflow-nmvzb-2236032954	0/2	Completed	13m

consumer-complaint-pipeline-kubeflow-nmvzb-2253512504	0/2	Completed	13m
consumer-complaint-pipeline-kubeflow-nmvzb-2453066854	0/2	Completed	10m
consumer-complaint-pipeline-kubeflow-nmvzb-2732473209	0/2	Completed	11m
consumer-complaint-pipeline-kubeflow-nmvzb-997527881	0/2	Completed	10m
...			

Вы также можете просматривать журналы определенного компонента с помощью `kubectl`, выполнив следующую команду. Журналы для определенных компонентов можно получить через соответствующий модуль (`pod`):

```
$ kubectl logs -n kubeflow podname
```



Интерфейс командной строки TFX

В качестве альтернативы процессу создания конвейера на основе пользовательского интерфейса вы также можете создавать конвейеры и запускать их, используя интерфейс командной строки TFX (TFX CLI). Вы можете найти подробную информацию о том, как настроить TFX CLI и как развернуть конвейеры машинного обучения без пользовательского интерфейса, в разделе «Интерфейс командной строки TFX» приложения С.

Полезные функции Kubeflow Pipelines

В следующих разделах мы хотим выделить полезные функции Kubeflow Pipelines.

Перезапуск конвейеров после сбоя

Запуск и работа конвейера может занять некоторое время, иногда несколько часов. TFX хранит состояние каждого компонента в хранилище метаданных машинного обучения, и Kubeflow Pipelines может отслеживать успешно завершенные задачи компонентов конвейера. Таким образом, он предлагает функцию перезапуска конвейеров после сбоя, начиная с компонента, в котором произошел последний сбой. Это позволит избежать повторного запуска успешно завершивших работу компонентов и, следовательно, сэкономит время при повторном запуске конвейера.

Запуск конвейера по расписанию

Помимо запуска отдельных участков конвейера, Kubeflow Pipelines также позволяет нам запускать конвейер по расписанию. Как показано на рис. 12.12, мы можем планировать запуски аналогично тому, как мы делали это, определяя расписания в Apache Airflow.

Run Type

☐ One-off
 ☒ Recurring

Run trigger

Choose a method by which new runs will be triggered

Trigger type * Cron

Maximum concurrent runs * 10

☒ Has start date
 Start date 04/29/2020
Start time 03:46 PM

☐ Has end date

☒ Catchup ?

Run every Week

On: ☐ All S **M** T W T F S

☐ Allow editing cron expression. (format is specified [here](#))

cron expression 0 46 15 ? * 1

Note: Start and end dates/times are handled outside of cron.

Рис. 12.12. Планирование запусков конвейера с помощью Kubeflow Pipelines

Совместная работа и проверка запусков конвейера

Kubeflow Pipelines предоставляет специалистам по обработке данных интерфейсы для совместной работы и анализа работы конвейера. В главах 4 и 7 мы показывали визуальные инструменты, используемые, чтобы показать результаты проверки данных или модели. После завершения этих компонентов конвейера мы можем просмотреть результаты для компонентов.

На рис. 12.13 в качестве примера показаны результаты этапа проверки данных. Поскольку выходные данные компонента сохраняются на диск или в корзину облачного хранилища, мы также можем проверить работу конвейера позднее.

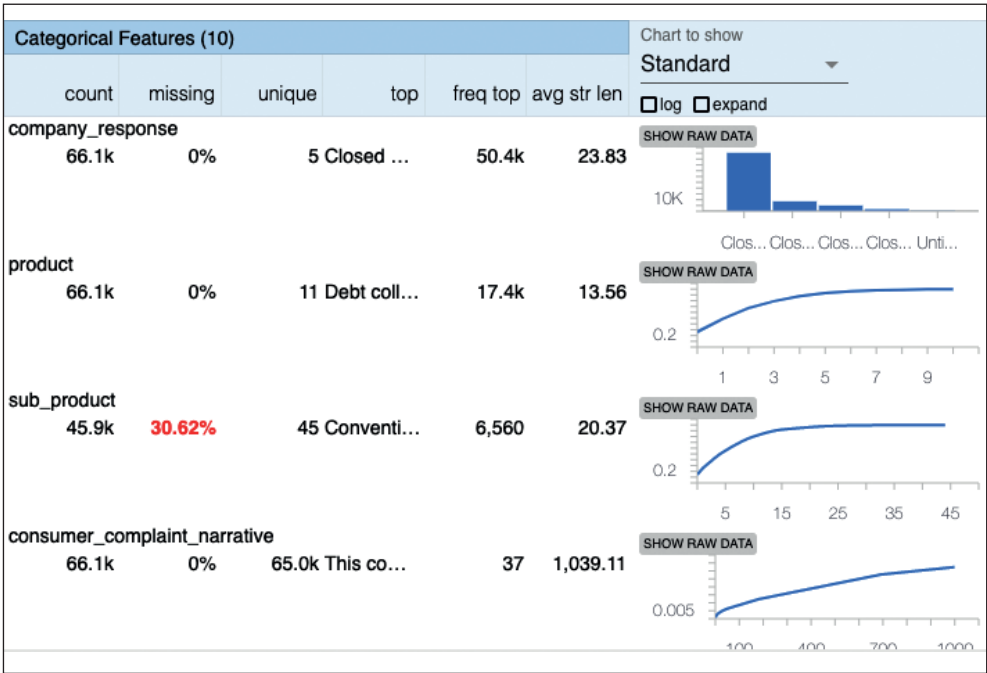


Рис. 12.13. В Kubeflow Pipelines доступна статистика TFDV

Поскольку результаты каждого прогона конвейера и компонента для этих прогонов сохраняются в хранилище метаданных машинного обучения, мы также можем сравнивать прогоны. Как показано на рис. 12.14, Kubeflow Pipelines предоставляет пользовательский интерфейс для сравнения прогонов конвейера.

Experiments

← Compare runs

Run overview

Filter runs

Run name

Status

Duration

Experiment

Run of consumer_complaint_pipeline_cloud_ai_to_c...

✓

1:21:18

Default

Run of consumer_complaint_pipeline_cloud_ai_t...

⚠

0:46:23

Default

Parameters

Run of consumer_complaint_pipeline_cloud_ai_to_cloud_bucket (d4cf4)

pipeline-root

gs://consumer_complaint_gcp_cloud_ai/tfx_pipeline/consumer_complaint_pipeline_cloud_ai_to_cloud_bucket

Metrics

Markdown

Tensorboard

Aggregated view

TF Version

TensorFlow 2.0.0

Start Combined Tensorboard

Run of consumer_complaint_pipeline_cloud_ai_to_bucket (61b88)

TF Version

TensorFlow 2.0.0

Start Tensorboard

Рис. 12.14. Сравнение прогонов конвейера с помощью Kubeflow Pipelines

Kubeflow Pipelines также прекрасно интегрируется с TensorBoard (элементом библиотеки TensorFlow). Как показано на рис. 12.15, мы можем просмотреть статистику прогонов обучения модели с помощью TensorBoard. После создания базового модуля Kubernetes мы можем просматривать статистику прогонов обучения модели с помощью TensorBoard.

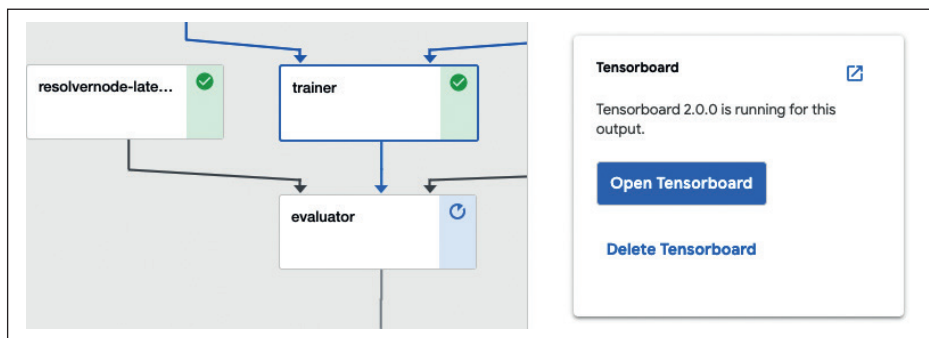


Рис. 12.15. Просмотр прогонов обучения с помощью TensorBoard

Аудит линии конвейера

Для более широкого внедрения машинного обучения критически важно пересмотреть создание модели. Если, например, специалисты по данным обнаруживают, что обученная модель выдает искаженный или несправедливый прогноз (как мы обсуждали в главе 7), важно проследить и воспроизвести данные или гиперпараметры, которые мы использовали. Как правило, для этого нужен контрольный журнал для каждой модели машинного обучения.

Kubeflow Pipelines предлагает решение для создания и просмотра такого контрольного журнала с помощью инструмента Kubeflow Lineage Explorer. Он включает пользовательский интерфейс, который может легко запрашивать данные хранилища метаданных машинного обучения.

Как видно в правом нижнем углу на рис. 12.16, модель машинного обучения была перенесена в определенный каталог. Lineage Explorer позволяет нам проследить все компоненты и артефакты, которые внесли свой вклад в экспортированную модель, вплоть до исходного, необработанного набора данных. Мы можем отследить, кто подписался на модель, если мы включаем действия человека в петлю конвейера (см. раздел «Участие человека в конвейере машинного обучения» главы 10), а также можем проверить результаты проверки данных и выяснить, начали ли начальные данные обучения смещаться.

Как видите, Kubeflow Pipelines – невероятно мощный инструмент для организации конвейеров машинного обучения. Если ваша инфраструктура основана на AWS или Azure или если вам нужен полный контроль над настройкой, мы рекомендуем использовать этот инструмент. Однако если вы уже используете GCP или хотите найти более простой способ использования конвейеров Kubeflow, мы рекомендуем ознакомиться с дальнейшими материалами в этой книге.

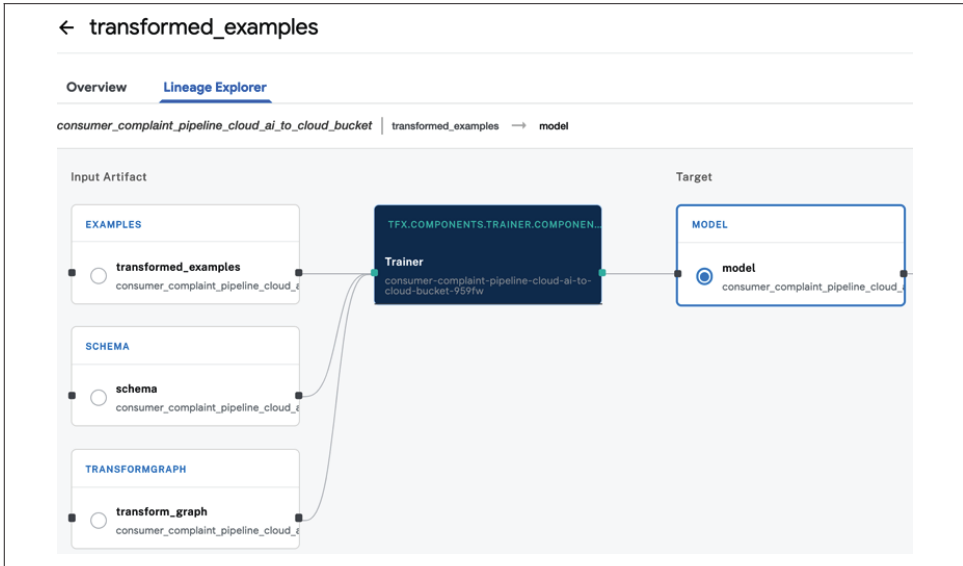


Рис. 12.16. Проверка линии конвейера с помощью Kubeflow Pipelines

КОНВЕЙЕРЫ, РАБОТАЮЩИЕ НА GOOGLE CLOUD AI PLATFORM

Если вы не хотите тратить время на администрирование собственной настройки Kubeflow Pipelines или хотите интегрироваться с платформой AI GCP или другими службами GCP, такими как Dataflow, обучение и развертывание модели на AI Platform и т. д., мы настоятельно рекомендуем ознакомиться с материалами данного раздела. В следующем разделе мы обсудим, как настроить конвейеры Kubeflow на платформе искусственного интеллекта Google Cloud AI Platform. Кроме того, мы расскажем, как можно обучать свои модели машинного обучения с помощью задач платформы Google Cloud AI Platform и масштабировать предварительную обработку с помощью Google Cloud Dataflow, который можно использовать в качестве средства запуска Apache Beam.

Настройка конвейера

Google AI Platform Pipelines (см. <https://oreil.ly/WAft5>) позволяет настраивать Kubeflow Pipelines через пользовательский интерфейс. На рис. 12.17 показана первая страница веб-интерфейса AI Platform Pipelines, с которой вы можете начать создавать свою настройку.



Бета-версия

Как можно заметить на рис. 12.17, на момент написания этой книги данный продукт Google Cloud все еще находится в стадии бета-тестирования. Представленные рабочие процессы со временем могут измениться.

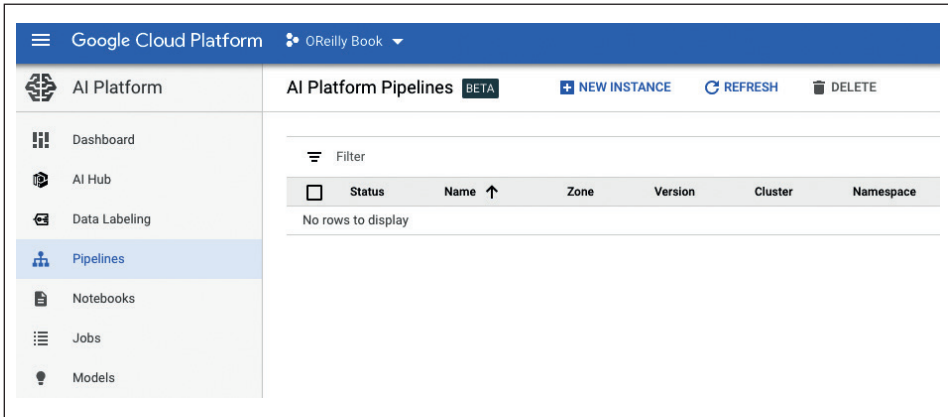


Рис. 12.17. Google Cloud AI Platform Pipelines

Когда вы нажимаете New Instance (в правом верхнем углу страницы), он отправляет вас в Google Marketplace, как показано на рис. 12.18.

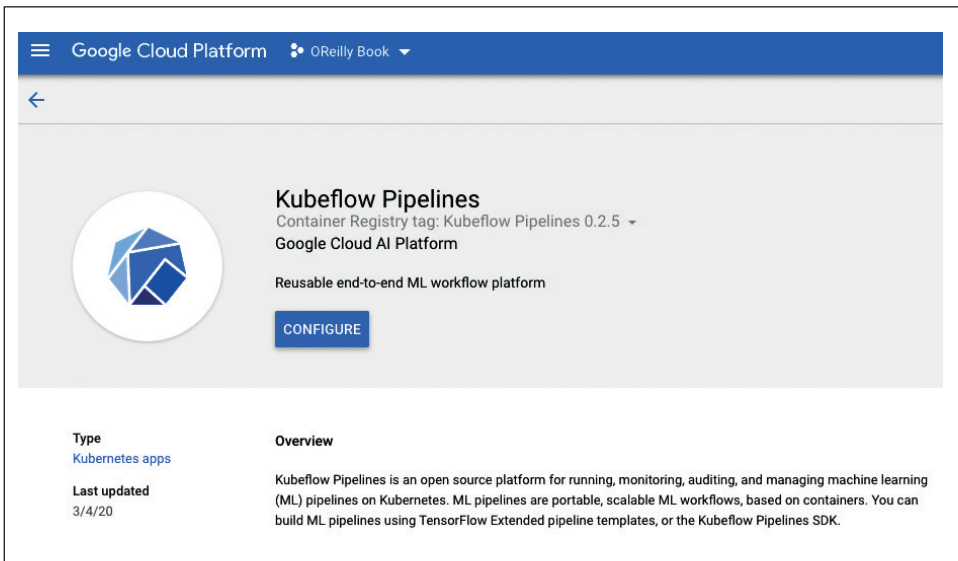


Рис. 12.18. Страница Google Cloud Marketplace для Kubeflow Pipelines

После того как вы нажмете **Configure**, вам будет предложено в верхней части меню выбрать существующий кластер Kubernetes или создать новый кластер, как показано на рис. 12.19.



Размеры узла

При создании нового кластера Kubernetes или выборе существующего кластера учитывайте доступную память узлов. Каждому экземпляру узла необходимо выделить достаточно памяти для хранения всей модели. Для нашего демонстрационного проекта мы выбрали тип экземпляра `n1-standard-4`. На момент написания этой книги мы не могли создать настраиваемый кластер при запуске Kubeflow Pipelines из Marketplace. Если для настройки конвейера требуются более масштабные экземпляры, мы рекомендуем сначала создать кластер и его узлы, а затем выбрать кластер из списка существующих кластеров при создании настройки Kubeflow Pipelines из GCP Marketplace.

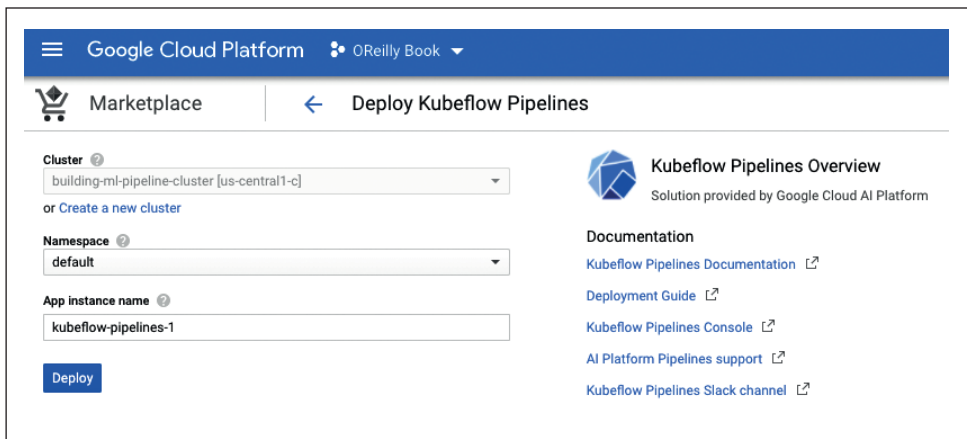


Рис. 12.19. Настройка кластера для Kubeflow Pipelines



Область доступа

Во время создания Kubeflow Pipelines в Marketplace или создания настраиваемого кластера выберите **Allow full access to all Cloud APIs** (Разрешить полный доступ ко всем облачным API), когда вас попросят указать область доступа узлов кластера. Для конвейеров Kubeflow требуется доступ к различным облачным API. Предоставление доступа ко всем облачным API упрощает процесс настройки.

После настройки кластера Kubernetes Google Cloud создаст экземпляр вашей настройки конвейеров Kubeflow, как показано на рис. 12.20.

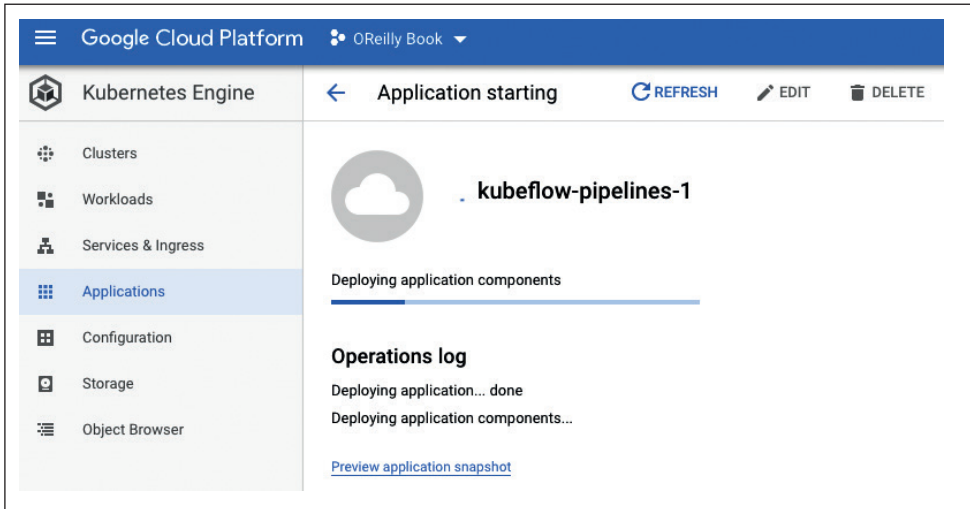


Рис. 12.20. Создание настройки Kubeflow Pipelines

Через несколько минут ваш настроенный вариант установки будет готов к использованию, и вы сможете найти установленный Kubeflow Pipelines как экземпляр, указанный в списке развернутых конфигураций Kubeflow Pipelines AI Platform. Если вы щелкнете **Open Pipelines Dashboard**, как показано на рис. 12.21, интерфейс перенаправит вас к недавно развернутому экземпляру Kubeflow Pipelines. Сам экземпляр Kubeflow Pipelines будет работать так, как мы обсуждали в предыдущем разделе, и пользовательский интерфейс будет выглядеть аналогично.

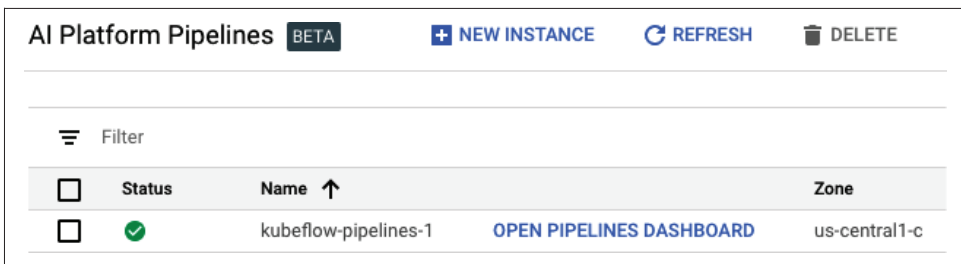


Рис. 12.21. Список развернутых экземпляров Kubeflow



Пошаговые сценарии установки, доступные на панели инструментов AI Platform Pipelines (AI Platform Pipelines Dashboard)

Если вы устанавливаете Kubeflow Pipelines вручную, шаг за шагом, как описано в разделе «Доступ к установленному экземпляру Kubeflow Pipelines» этой главы и в приложении В, ваш экземпляр Kubeflow Pipelines также будет указан в списке доступных экземпляров AI Platform Pipelines.

Настройка конвейера TFX

Конфигурация конвейеров TFX очень похожа на конфигурацию `KubeflowDagRunner`, которую мы обсудили ранее. Фактически если вы смонтируете постоянный том с требуемым модулем Python и обучающими данными, как описано в разделе «Настройка конвейера» этой главы, то можете запускать конвейеры TFX на AI Platform Pipelines.

В следующих разделах мы покажем вам несколько изменений в выполненной ранее настройке `Kubeflow Pipelines`, которые могут либо упростить ваш рабочий процесс (например, загрузка данных из сегментов хранилища `Google Storage`), либо помочь вам с масштабированием конвейеров за пределы кластера `Kubernetes` (например, путем обучения модели с использованием заданий `AI Platform Jobs`).

Использование сегментов `Cloud Storage` для обмена данными

В разделе «Настройка конвейера» этой главы мы упоминали, что можем загрузить данные и модули Python, необходимые для работы конвейера, из постоянного тома, смонтированного в кластере `Kubernetes`. Если вы запускаете конвейеры в экосистеме `Google Cloud`, вы также можете загружать данные из сегментов `Google Cloud Storage`. Это упростит рабочие процессы, позволяя загружать и просматривать файлы через веб-интерфейс `GCP` или используя `SDK gcloud`.

Пути к сегментам хранилища могут быть указаны аналогично тому, как указываются пути к файлам на диске, что продемонстрировано в следующем фрагменте кода:

```
input_bucket = 'gs://YOUR_INPUT_BUCKET'
output_bucket = 'gs://YOUR_OUTPUT_BUCKET'
data_dir = os.path.join(input_bucket, 'data')

tfx_root = os.path.join(output_bucket, 'tfx_pipeline')
pipeline_root = os.path.join(tfx_root, pipeline_name)
serving_model_dir = os.path.join(output_bucket, 'serving_model_dir')
module_file = os.path.join(input_bucket, 'components', 'module.py')
```

Часто бывает полезно разделить сегменты облачного хранилища, предназначенные для входных (например, модуля Python и обучающих данных) и выходных данных (например, обученных моделей), но вы также можете использовать одни и те же сегменты как для входных, так и для выходных данных.

Обучение моделей с использованием заданий `AI Platform Jobs`

Если вы хотите масштабировать обучение модели, используя графический процессор или `TPU`, то можете настроить конвейер для запуска этапа обучения модели машинного обучения на этом оборудовании:

```

project_id = 'YOUR_PROJECT_ID'
gcp_region = 'GCP_REGION>' ❶

ai_platform_training_args = {
    'project': project_id,
    'region': gcp_region,
    'masterConfig': {
        'imageUri': 'gcr.io/oreilly-book/ml-pipelines-tfx-custom:0.22.0' ❷
        'scaleTier': 'BASIC_GPU', ❸
    }
}

```

- ❶ Например, `us-central1`
- ❷ Укажите собственное изображение (при необходимости)
- ❸ Другие варианты значений этого параметра: `BASIC_TPU`, `STANDARD_1` и `PREMIUM_1`

Чтобы компонент `Trainer` наблюдал за конфигурацией AI Platform, вам необходимо настроить исполнитель (executor) и заменить им модуль `GenericExecutor`, который мы использовали до сих пор совместно с нашим компонентом `Trainer`. В следующем фрагменте кода показаны необходимые дополнительные аргументы:

```

from
tfx.extensions.google_cloud_ai_platform.trainer import executor \
as ai_platform_trainer_executor

trainer = Trainer(
...
    custom_executor_spec=executor_spec.ExecutorClassSpec(
        ai_platform_trainer_executor.GenericExecutor),
    custom_config = {
        ai_platform_trainer_executor.TRAINING_ARGS_KEY:
            ai_platform_training_args
    }
)

```

Вместо обучения моделей машинного обучения внутри кластера Kubernetes вы можете распределить обучение модели, используя AI Platform. В дополнение к возможностям распределенного обучения AI Platform предоставляет доступ к оборудованию для ускоренного обучения, такому как TPU.

Когда компонент `Trainer` запускается в конвейере, он запускает задачу обучения в AI Platform Jobs, как показано на рис. 12.22. Здесь вы можете проверить файлы журнала или статус выполнения задачи обучения.

Job ID	Type	HyperTune	HyperTune parameters
<input type="checkbox"/> tfx_20200423151030	Custom code training	No	
<input type="checkbox"/> tfx_20200422213552	Custom code training	No	
<input type="checkbox"/> tfx_20200422183838	Custom code training	No	
<input type="checkbox"/> tfx_20200422041444	Custom code training	No	
<input type="checkbox"/> tfx_20191205173323	Custom code training	No	
<input type="checkbox"/> tfx_20191204162117	Custom code training	No	
<input type="checkbox"/> tfx_20191203232017	Custom code training	No	
<input type="checkbox"/> tfx_20190630010158	Custom code training	No	
<input type="checkbox"/> tfx_20190630010058	Custom code training	No	

Рис. 12.22. Задачи по обучению моделей AI Platform

Развертывание моделей с использованием конечных точек AI Platform

Если вы запускаете свои конвейеры в экосистеме Google Cloud, вы также можете развертывать модели машинного обучения на конечных точках платформы AI. У этих конечных точек есть возможность масштабировать вашу модель в случае, если модель испытывает пики нагрузки при выводе результирующих данных.

Вместо настройки параметра `push_destination`, как обсуждалось в разделе «Компонент TFX Pusher» главы 7, мы можем переопределить исполнитель (executor) и предоставить данные Google Cloud для развертывания на AI Platform. В следующем фрагменте кода показаны необходимые данные для такой конфигурации:

```
ai_platform_serving_args = {
    'model_name': 'consumer_complaint',
    'project_id': project_id,
    'regions': [gcp_region],
}
```

Подобно настройке компонента `Trainer`, нам нужно изменить исполнитель (executor) компонента и предоставить `custom_config` необходимые данные для развертывания:

```
from tfx.extensions.google_cloud_ai_platform.pusher import executor \
    as ai_platform_pusher_executor

pusher = Pusher(
    ...
    custom_executor_spec=executor_spec.ExecutorClassSpec(
        ai_platform_pusher_executor.Executor),
    custom_config = {
        ai_platform_pusher_executor.SERVING_ARGS_KEY:
            ai_platform_serving_args
    }
)
```


Если вы определите конфигурацию компонента Pusher, вам не нужно будет настраивать и развертывать ваш экземпляр TensorFlow Serving на AI Platform.



Ограничения развертывания

На момент написания этой книги максимальный размер моделей для развертывания на AI Platform был ограничен 512 МБ. Наш демонстрационный проект превышает этот лимит, и, следовательно, в настоящий момент его невозможно развернуть, используя конечные точки AI Platform.

Масштабирование с использованием Google Dataflow

До сих пор все компоненты, которые зависят от Apache Beam, по умолчанию выполняли задачи обработки данных с помощью DirectRunner, и это означает, что задачи обработки будут выполняться в том же экземпляре, где Apache Beam инициировал выполнение задачи. В этой ситуации Apache Beam будет использовать как можно больше ядер ЦП, но не будет масштабироваться за пределы одного экземпляра.

Одна из существующих альтернатив – запустить Apache Beam с Google Cloud Dataflow. В этой ситуации TFX будет обрабатывать задания с помощью Apache Beam, а последний будет отправлять задачи в Dataflow. В зависимости от требований каждого задания Dataflow будет наращивать вычислительные экземпляры и распределять задания по экземплярам. Это довольно удобный способ масштабирования таких задач предварительной обработки данных, как генерация статистики или предварительная обработка данных.

Чтобы использовать возможности масштабирования Google Cloud Dataflow, нам нужно определить еще несколько дополнительных настроек конфигурации Beam, которые мы используем в нашей реализации конвейера:

```
tmp_file_location = os.path.join(output_bucket, "tmp")
beam_pipeline_args = [
    "--runner=DataflowRunner",
    "--experiments=shuffle_mode=auto", "--project={}".format(project_id),
    "--temp_location={}".format(tmp_file_location), "--region={}".format(gcp_region),
    "--disk_size_gb=50",
]
```

Помимо настройки, указывающей DataflowRunner в качестве типа runner, мы также установили для shuffle_mode значение auto. Это интересная особенность Dataflow. Вместо выполнения преобразований, таких как GroupByKey, в виртуальной машине Google Compute Engine операция будет обрабатываться в серверной части службы Dataflow. Это сокращает время выполнения и затраты на ЦП/память вычислительных экземпляров.

Запуск и работа конвейера

Запуск конвейера на платформе Google Cloud AI Platform аналогичен тому способу, который мы обсуждали в разделе «Оркестрация конвейеров TFX с помощью Kubeflow Pipelines» этой главы. Сценарий TFX сгенерирует конфи-

гурацию Argo. Затем конфигурацию можно загрузить в настройку Kubeflow Pipelines на платформе AI Platform.

Во время работы конвейера вы можете проверить задания по обучению модели, как описано в разделе «Обучение моделей с использованием заданий AI Platform Jobs» этой главы. Также вы можете наблюдать за исполнением заданий Dataflow, как показано на рис. 12.23.

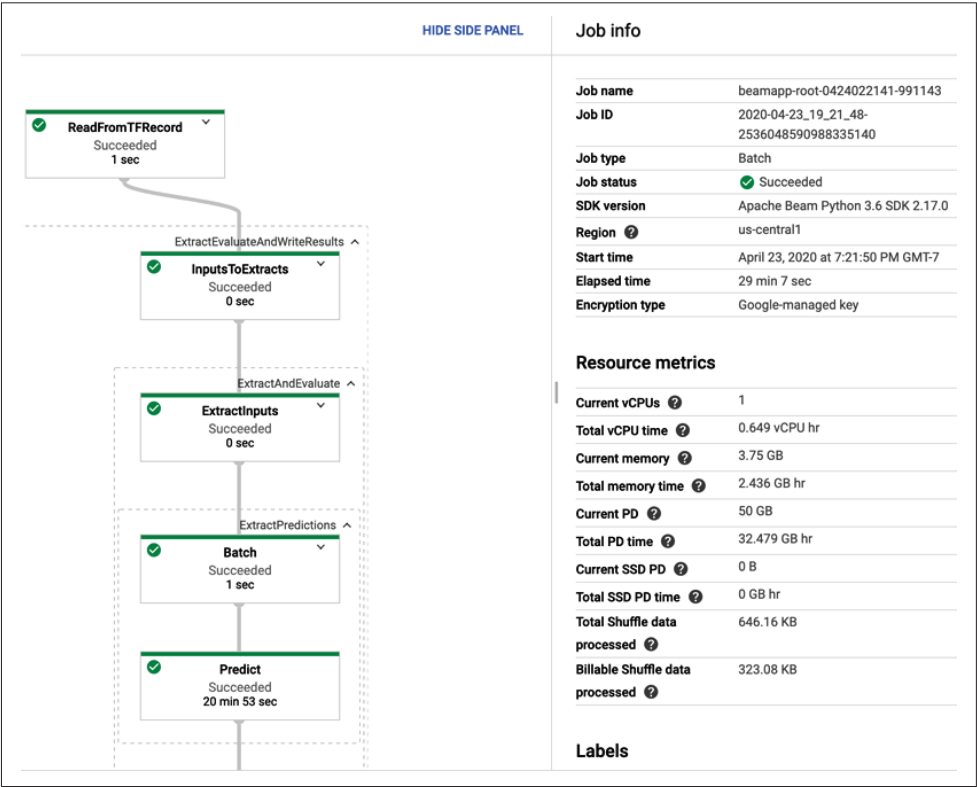


Рис. 12.23. Информация о задании Google Cloud Dataflow

Панель управления Dataflow (Dataflow Dashboard) предоставляет информацию о ходе выполнения вашего задания и о требованиях к масштабированию.

РЕЗЮМЕ

Запуск конвейеров с помощью Kubeflow Pipelines дает большие преимущества, компенсирующие, по нашему мнению, тот объем работы, который необходимо выполнить при дополнительной настройке. Мы имеем возможность наблюдать за работой линий конвейера, осуществлять бесшовную интеграцию с TensorBoard и выбирать различные варианты перезапуска конвейера. Все это – веские аргументы в пользу выбора Kubeflow Pipelines в качестве оркестровщика конвейера.

Как мы обсуждали ранее, текущий процесс запуска конвейеров TFX с Kubeflow Pipelines отличается от тех вариантов, которые используются для конвейеров, работающих на Apache Beam или Apache Airflow (см. главу 11). Однако в конфигурации компонентов TFX не произошло никаких изменений по сравнению с тем, что мы обсуждали в предыдущей главе.

В этой главе мы рассмотрели две настройки конвейера Kubeflow: первую настройку можно использовать практически с любой управляемой службой Kubernetes, такой как AWS Elastic Kubernetes Service или Microsoft Azure Kubernetes Service. Вторую настройку можно использовать с платформой искусственного интеллекта Google Cloud AI Platform.

В следующей главе мы обсудим, как можно превратить работу конвейера в цикл, используя петли обратной связи.

Глава 13

Петли обратной связи

Теперь, когда у нас есть непрерывный конвейер для развертывания модели машинного обучения в промышленной среде, мы хотим запускать его повторно. После развертывания модели не должны быть статичными. Собираются новые данные, распределение данных изменяется (как обсуждалось в главе 4), модели дрейфуют (что обсуждалось в главе 7), и, что наиболее важно, мы работаем над постоянным улучшением наших конвейеров.

Добавление обратной связи в конвейер машинного обучения превращает обычный рабочий процесс конвейера в жизненный цикл, как показано на рис. 13.1. Предсказания модели приводят к сбору новых данных, что улучшает модель.

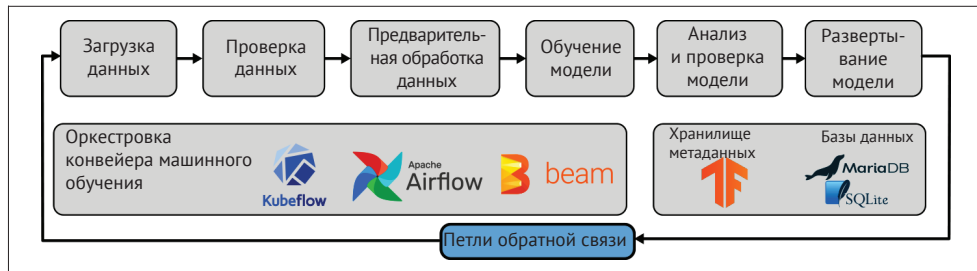


Рис. 13.1. Обратная связь модели машинного обучения как часть конвейера машинного обучения

Без свежих данных прогностическая значимость модели со временем снижается, так как входные данные модели меняются. Развертывание модели машинного обучения может фактически изменить поступающие обучающие данные, поскольку пользовательский опыт меняется: например, в системе видеорекомендаций лучшие рекомендации модели приводят к выбору пользователем других вариантов просмотра. Петли обратной связи особенно важны для персонализированных моделей, такие как рекомендательные системы или интеллектуальный ввод текста.

На данном этапе чрезвычайно важно правильно настроить оставшуюся часть конвейера. Подача новых данных должна приводить к остановке конвейера только в том случае, когда приток новых данных приводит к выходу ста-

истики данных за пределы, установленные при проверке данных, или когда статистика модели выходит за границы, установленные при анализе модели. В таких ситуациях мы можем впоследствии запустить переобучение модели, добавление новых признаков и т. д. Если произойдет одно из таких событий, новая модель должна получить новый номер версии.

На этом этапе крайне важно обеспечить надежную настройку остальной части конвейера. Загрузка новых данных должна вызывать сбой конвейера только в том случае, если статистические параметры данных выходят за ранее установленные ограничения или статистические параметры модели выходят за пределы, установленные вами при ее анализе. Такая ситуация в дальнейшем может вызвать определенные события, такие как переподготовка модели, конструирование новых признаков и т. д. Если срабатывает один из этих триггеров, новой модели должен быть присвоен новый номер версии.

Помимо сбора новых обучающих данных, петли обратной связи могут также дать вам информацию о реальном использовании вашей модели. Она может включать количество активных пользователей, время суток, когда они взаимодействуют с вашей моделью, и много других элементов данных. Этот тип данных чрезвычайно полезен для демонстрации ценности вашей модели заинтересованным сторонам.



Петли обратной связи могут нести негативные последствия

Петли обратной связи также могут иметь негативные последствия, и к ним следует подходить с осторожностью. Если вы вводите прогнозы модели в новые обучающие данные без участия человека, модель будет обучаться как на своих ошибочных, так и на своих правильных прогнозах. Петли обратной связи также могут усиливать любые смещения или диспропорции, присутствующие в исходных данных. Тщательный анализ модели может помочь вам обнаружить некоторые из подобных ситуаций.

ЯВНАЯ И НЕЯВНАЯ ОБРАТНАЯ СВЯЗЬ

Мы можем разделить нашу обратную связь на два основных типа: неявную и явную¹. Неявная обратная связь – это вариант, когда люди своими действиями при обычном использовании продукта дают обратную связь для модели, например совершая покупку, предложенную рекомендательной системой, или просматривая предлагаемый фильм. Для неявной обратной связи важнейшее значение имеет конфиденциальность пользователей, потому что это заманчиво простая идея – отслеживать каждое действие, которое пользователь совершает. Явная обратная связь – это когда пользователь вносит некоторый непосредственный вклад в прогноз, например выражает одобрение или неодобрение (ставит соответствующий «лайк»), или корректирует прогноз.

¹ За более подробной информацией обратитесь к руководству Google PAIR (см. https://oreil.ly/N_j4).

Маховик данных

В некоторых ситуациях у вас могут быть все данные, необходимые для создания нового продукта для машинного обучения. Но в других случаях вам может понадобиться собрать больше данных. Это происходит особенно часто, когда мы реализуем обучение с учителем. Обучение с учителем является более зрелым, чем обучение без учителя, и, как правило, дает более надежные результаты; поэтому большинство моделей, развертываемых в производственных системах, – это модели, в которых используется обучение с учителем. Часто возникает ситуация, когда у вас есть большие объемы немаркированных данных, но недостаточно маркированных данных (хотя развитие *переноса обучения* начинает устранять необходимость в огромных объемах маркированных данных для некоторых задач машинного обучения).

Если у вас много немаркированных данных и вам нужно собрать больше меток, особенно полезной окажется концепция *маховика данных*. Она позволяет вам расширять ваш обучающий набор данных, настраивая исходную модель с применением уже существующих данных из продукта, данных, размеченных вручную, или общедоступных данных. Собирая отзывы пользователей о первоначальной модели, вы можете маркировать данные, что улучшает прогнозы модели и, таким образом, привлекает к продукту больше пользователей, которые размечают больше данных и т. д., как показано на рис. 13.2.

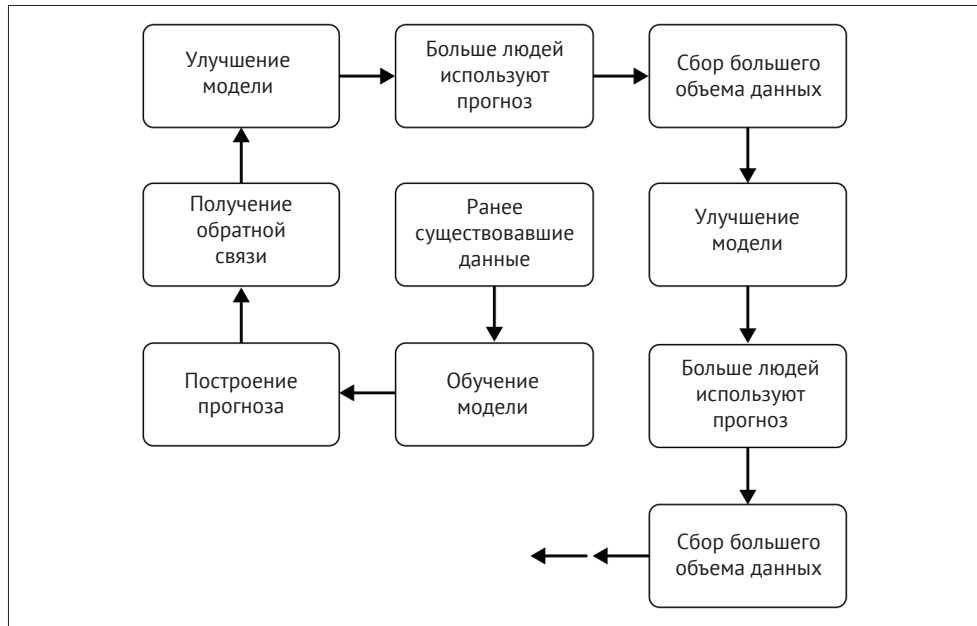


Рис. 13.2. Маховик данных

Петли обратной связи в реальном мире

Некоторые из наиболее известных примеров петель обратной связи в системах машинного обучения – это ситуации, когда клиенту выдаются предсказания модели. Такие сценарии особенно распространены в рекомендательных системах, где модель предсказывает верхние k наиболее подходящих вариантов. В этом случае часто бывает трудно собрать обучающие данные для рекомендательных систем до запуска продукта, поэтому эти системы часто в значительной степени зависят от обратной связи от их пользователей.

Система рекомендаций фильмов Netflix (см. <https://oreil.ly/uX9Oo>) является классическим примером петли обратной связи. Пользователю рекомендуют смотреть фильмы, после чего пользователь предоставляет обратную связь, оценивая прогнозы. По мере того как пользователи оценивают все больше фильмов, они получают рекомендации, которые более точно соответствуют их вкусам.

Первоначально, когда основной деятельностью Netflix была доставка DVD-дисков по почте, они использовали систему оценок 1–5 «звезд» при сборе оценок DVD; это также было сигналом о том, что клиент действительно смотрел DVD. Таким образом, Netflix смог собрать однозначные отзывы. Когда их бизнес переключился на потоковые фильмы онлайн, они также могли собрать неявную обратную связь о том, смотрел ли пользователь фильмы, которые ему рекомендовали, и смотрел ли пользователь весь фильм целиком. Затем Netflix переключился с рейтинговой системы 1–5 «звезд» на более простую систему «понравилось / не понравилось». Это позволило собрать больше отзывов, так как у пользователя уходило меньше времени на оценивание. Кроме того, более точные оценки могут быть не такими актуальными: как должна реагировать модель, если фильм имеет оценку 3 «звезды»? Такая оценка не говорит о том, что предсказание является правильным или неправильным, в то время как оценки «понравилось / не понравилось» дают модели четкий сигнал¹.

Другим примером петли обратной связи, и в данном случае отрицательным примером, является печально известный бот Microsoft TAY². Нашумевшей новостью в 2016 году стало отключение бота через 16 часов после запуска из-за его оскорбительных и иногда расистских твитов. К этому времени бот написал более 96 000 твитов. Он был автоматически переобучен на основании ответов на твиты, которые были намеренно провокационными. Петля обратной связи в данной ситуации образовывалась тогда, когда система брала ответы на свои первоначальные твиты и включала их в обучающие данные. Вероятно, это было сделано для того, чтобы бот казался более человечным, но в результате он собрал наихудшие ответы, и его реплики стали в высшей степени оскорбительными.

¹ Отзывы должно быть легко собирать, и они должны и давать действенные результаты.

² См. <https://oreil.ly/YM21r>.



Что может пойти не так?

Важно заранее подумать о том, что может пойти не так в петле обратной связи, а также подумать о наилучшем сценарии. Что является самым худшим из того, что могут сделать ваши пользователи? Как вы защищаетесь от плохих личностей, которые хотят разрушить вашу систему набегом организованной толпы или прибегнув к автоматизации?

Третий пример реальной петли обратной связи исходит от Stripe, компании онлайн-платежей¹. Stripe создала двоичный классификатор для прогнозирования мошенничества при транзакциях по кредитным картам. Их система блокировала транзакции, если модель предсказывала, что транзакции могут быть мошенническими. Они получили обучающий набор на основе данных прошлых транзакций и обучили на его основе модель, которая дала хорошие результаты на обучающих наборах. Однако было невозможно узнать точность предсказаний и воспроизвести ситуацию в промышленной среде, потому что если модель предсказывала, что транзакция была мошеннической, эта транзакция блокировалась. Узнать, была ли транзакция на самом деле мошеннической, было невозможно, потому что эта транзакция никогда не совершалась.

Другая большая проблема возникла, когда модель была переобучена с использованием новых данных: ее точность снизилась. В этом случае петля обратной связи привела к блокировке всех исходных типов мошеннических транзакций, и поэтому они были недоступны для новых обучающих данных. Новая модель обучалась на основе данных, где присутствовали мошеннические операции, которые не были обнаружены. Решение Stripe состояло в том, чтобы ослабить правила и позволить небольшому количеству подозрительных транзакций пройти, даже если модель предсказывала, что они будут мошенническими. Это позволило им оценить модель и предоставить новые релевантные обучающие данные.



Последствия применения петель обратной связи

Петли обратной связи часто будут иметь некоторые последствия, которые не были очевидны при их разработке. Важно продолжать мониторинг системы после ее развертывания. Мы предлагаем использовать для этого методы, о которых рассказывалось в главе 7.

В предыдущем примере из Stripe петля обратной связи привела к снижению точности модели. Однако повышение точности также может дать неожиданный отрицательный эффект. Система рекомендаций YouTube (см. <https://oreil.ly/QDCC2>) нацелена на увеличение количества времени, которое люди проводят за просмотром видео. Пользовательские отклики означают, что модель точно предсказывает, что они будут смотреть дальше. И она оказалась невероятно успешной: люди смотрят более 1 млрд часов видео на YouTube каждый день (см. <https://oreil.ly/KVF4M>). Тем не менее в последнее время возникают

¹ См. выступление Майкла Манапата «Контрфактическая оценка моделей машинного обучения» (Michael Manapat, «Counterfactual Evaluation of Machine Learning Models», презентация, PyData, Сиэтл, 2015 г.), доступное по ссылке <https://oreil.ly/rGCho>.

опасения, что эта система побуждает людей смотреть видео со все более экстремистским контентом (см. https://oreil.ly/_lubw). Когда такие системы становятся очень большими, чрезвычайно трудно предвидеть все последствия петли обратной связи. Поэтому не прекращайте их использовать, но убедитесь, что вы предпринимаете все необходимые меры для защиты ваших пользователей.

Как видно из приведенных примеров, петли обратной связи могут давать положительный эффект и помочь нам получить больше обучающих данных, которые мы можем использовать для улучшения модели и даже для построения бизнеса. Однако в то же самое время они могут привести к серьезным проблемам. Если вы тщательно выбрали метрики для своей модели, которые гарантируют, что ваша петля обратной связи будет приносить положительный эффект, то ваш следующий шаг – научиться собирать отзывы. Об этом мы расскажем в следующем разделе.

КОНСТРУКТИВНЫЕ ШАБЛОНЫ ДЛЯ СБОРА ОТЗЫВОВ

В этом разделе мы обсудим некоторые распространенные способы сбора отзывов. Ваш выбор метода будет зависеть от нескольких вещей:

- бизнес-проблема, которую вы пытаетесь решить;
- тип и дизайн приложения/продукта;
- тип модели машинного обучения: классификация, система рекомендаций и т. д.

Если вы планируете собирать отзывы от пользователей вашего продукта, очень важно информировать пользователей о том, что происходит, чтобы они могли дать согласие на обратную связь. Это также может помочь вам собрать больше отзывов: если пользователь принимает участие в улучшении системы, он с большей вероятностью предоставит отзыв.

Мы рассматриваем различные варианты сбора обратной связи в следующих разделах:

- «Пользователи предпринимают определенные действия в результате прогноза»;
- «Пользователи оценивают качество прогноза»;
- «Пользователи исправляют прогноз»;
- «Краудсорсинг аннотаций»;
- «Экспертные аннотации»;
- «Обратная связь автоматически предоставляется системой».

Хотя ваш выбор шаблона проектирования будет в определенной степени зависеть от проблемы, которую пытается решить ваш конвейер машинного обучения, сделанный вами выбор повлияет на то, как вы отслеживаете обратную связь, а также на то, как вы включаете ее в конвейер машинного обучения.

Пользователи предпринимают определенные действия в результате прогноза

В этом методе прогнозы нашей модели отображаются пользователю, который в результате совершает какое-то действие онлайн. Выполненное пользователем действие регистрируется, и полученная запись результата действия поль-

зователя предоставляет для нашей модели машинного обучения новые обучающие данные.

Примером этого может служить любая рекомендательная система для выбора продуктов, например та, которую использует Amazon, чтобы рекомендовать своим пользователям следующую покупку. Пользователю показывают набор продуктов, которые, по прогнозам модели, будут представлять для него интерес. Если пользователь просматривает один из этих продуктов или покупает его, рекомендация была хорошей. Тем не менее в данном сценарии отсутствует информация о том, относились ли другие продукты, которые пользователь не просматривал, к хорошим рекомендациям. Это неявная обратная связь: обратная связь не предоставляет именно те данные, которые нам нужны для обучения модели (такими данными будут являться данные ранжирования каждого отдельного прогноза). Вместо этого необходимо агрегировать обратную связь по многим различным пользователям, чтобы получить новые обучающие данные.

Пользователи оценивают качество прогноза

В этой методике предсказание модели отображается пользователю, и пользователь дает некоторый сигнал, чтобы показать, что ему нравится или не нравится предсказание. Это пример явной обратной связи, где пользователь должен предпринять некоторые дополнительные действия, предоставляющие новые данные. Обратная связь может быть оценкой с дискретной шкалой (число «звездочек») или простой двоичной оценкой «понравилось / не понравилось» («лайк»). Такой тип обратной связи хорошо подходит для рекомендательных систем и в особенности для персонализации. Необходимо позаботиться о том, чтобы обратная связь была действенной: оценка 3 «звездочки» из 5 не дает много информации о том, являются прогнозы модели полезными или точными.

Одним из ограничений данного метода выступает то, что обратная связь является косвенной – в случае рекомендательной системы пользователи однозначно высказываются о том, что такое плохие прогнозы, но не о том, каким должен быть правильный прогноз. Другое ограничение этой системы заключается в том, что существует множество способов интерпретации обратной связи. То, что понравилось пользователю, не обязательно означает что-то, чего он хочет видеть больше. Например, в системе рекомендаций фильмов ответ «нравится» от пользователя может означать, что ему нужно больше фильмов этого жанра, или того же режиссера, или с теми же актерами в главных ролях. Все эти нюансы теряются, когда возможна только бинарная обратная связь.

Пользователи исправляют прогноз

Этот метод является примером явной обратной связи и работает следующим образом:

- предсказания модели с более низкой точностью отображаются пользователю;
- пользователь принимает прогноз, если он верен, или редактирует его, если он неверен;
- прогнозы, проверенные пользователем, могут использоваться в качестве новых обучающих данных.

Этот способ лучше всего работает в тех случаях, когда пользователь сильно заинтересован в результате. Хорошим примером его использования может служить банковское приложение, в котором пользователь может размещать чеки в банке для последующего инкассирования. Модель, выполняющая распознавание изображений, автоматически заполняет сумму чека. Если сумма верна, пользователь подтверждает ее; если она неверна, пользователь вводит правильное значение. В этом случае в интересах пользователя ввести правильную сумму, чтобы деньги были зачислены на его счет. Приложение становится более точным с течением времени, так как пользователи генерируют все больше обучающих данных. Если вы можете использовать этот метод в петле обратной связи, то он может стать отличным способом быстрого сбора большого количества высококачественных новых данных.

Данный метод следует использовать с осторожностью и только в тех случаях, когда цели системы машинного обучения и пользователя строго совпадают. Если пользователь принимает неправильные ответы, потому что у него нет причин пытаться изменить их, в обучающих данных появляется множество ошибок, и модель со временем не становится более точной. И если пользователь получит некоторую выгоду, предоставив неверные результаты, это внесет смещение в новые обучающие данные.

Краудсорсинг аннотаций

Этот метод особенно полезен, если у вас имеется большой объем неразмеченных данных и невозможно получить метки от пользователей при обычном сценарии использования продукта. Многие задачи из областей НЛП и компьютерного зрения попадают в эту категорию: легко собрать большой массив изображений, но данные не размечены для конкретного варианта использования вашей модели машинного обучения. Например, если вы хотите обучить модель классификации изображений, которая классифицирует изображения с мобильных телефонов как «документы» или «не документы», вы можете попросить пользователей сделать много фотографий, но не предоставить вам ваши метки.

В этом случае обычно собирается большой пул неразмеченных данных, который затем передается на краудсорсинговую платформу, такую как AWS Mechanical Turk или Figure Eight. Людям, предоставляющим отзывы, платят некоторую (обычно небольшую) сумму за разметку данных. Такой метод наиболее подходит для задач, не требующих специальной подготовки.

При использовании этого метода необходимо контролировать качество разметки данных, и инструмент, с помощью которого выполняются аннотации, обычно настраивается таким образом, чтобы несколько человек обрабатывали один и тот же пример данных. Руководство Google PAIR (см. <https://oreil.ly/6FMFD>) дает отличные подробные рекомендации по организации процесса аннотирования; но главное, что нужно учитывать, – что способы поощрения людей, оставляющих аннотации, должны соответствовать результатам модели. Основным преимуществом данного метода является то, что можно предельно точно указать специфику создаваемых новых данных, поэтому они могут полностью соответствовать потребностям сложной модели.

Однако у этого подхода есть ряд недостатков, например он может оказаться не подходящим для работы с персональными или конфиденциальными дан-

ными. Будьте внимательны, чтобы убедиться, что существует разнообразная группа оценщиков, отражающая взгляды пользователей вашего продукта и общества в целом. Кроме того, этот подход также может оказаться дорогостоящим, и он может не масштабироваться для большого числа пользователей.

Экспертные аннотации

Экспертные аннотации создаются способом, аналогичным краудсорсингу, но в данном случае люди, предоставляющие аннотации, тщательно подбираются. Это можете быть вы, человек, строящий конвейер и использующий инструмент аннотации, такой как Prodigy (см. <https://prodi.gy/>), для текстовых данных. Или же это может быть эксперт предметной области, например если вы обучаете классификатор изображений на медицинских изображениях. Этот метод особенно подходит для следующих ситуаций:

- для аннотирования данных требуются специальные знания;
- данные являются закрытыми или в определенной степени конфиденциальными;
- требуется только небольшое количество меток (например, перенос обучения или метод машинного обучения с частичным привлечением учителя);
- ошибки в аннотациях имеют серьезные реальные последствия для людей.

Данный метод позволяет получить высококачественную обратную связь, но это дорогостоящий, ручной и плохо масштабируемый метод.

Обратная связь автоматически предоставляется системой

В некоторых конвейерах машинного обучения для сбора обратной связи не требуется участие человека. Модель делает прогноз, и происходит какое-то событие в будущем, которое говорит нам, был выданный прогноз модели правильным или нет. В этом случае новые обучающие данные собираются системой автоматически. Хотя этот метод не требует какой-либо отдельной инфраструктуры для сбора обратной связи, он все же требует осторожности: могут произойти неожиданные вещи, потому что наличие предсказаний может нарушить работу системы. Пример Stripe, приведенный выше, хорошо иллюстрирует это: модель оказывает влияние на свои собственные обучающие данные в будущем¹.

КАК ОТСЛЕЖИВАТЬ ПЕТЛИ ОБРАТНОЙ СВЯЗИ

После того как вы решили, какой тип обратной связи лучше всего подходит для вашей бизнес-задачи и вашего типа модели, вам необходимо включить ее в конвейер машинного обучения. Это тот самый момент, когда проверка модели, которую мы обсуждали в главе 7, становится абсолютно необходимой: новые данные будут распространяться через систему, и это не должно приводить к ухудшению качества системы по отслеживаемым показателям.

¹ Подробнее об этом можно прочитать в статье «Hidden Technical Debt in Machine Learning Systems», D. Sculley et al. («Скрытый технический долг в системах машинного обучения», автор Скалли и др.), см. <https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf>.

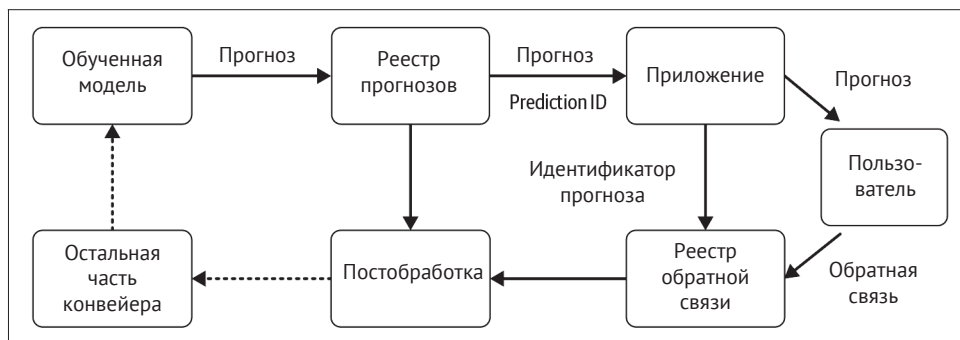


Рис. 13.3. Отслеживание обратной связи

Ключевым моментом в этом случае является то, что каждый прогноз должен получить идентификатор отслеживания, как показано на рис. 13.3. Это возможно реализовать с помощью своего рода «реестра прогнозов» – каждый прогноз сохраняется вместе с идентификатором отслеживания. Прогноз и присвоенный ему идентификатор передаются в приложение, а затем предсказание отображается пользователю. Если пользователь дает нам обратную связь, процесс продолжается.

Когда обратная связь получена, она сохраняется в «реестре обратной связи» вместе с идентификатором отслеживания предсказания. Этап обработки данных объединяет обратную связь с исходным прогнозом. Это позволяет отслеживать обратную связь с помощью этапов проверки данных и модели, чтобы вы всегда могли знать, какая обратная связь является основой новой версии модели.

Отслеживание явной обратной связи

Когда дело доходит до отслеживания, есть две формы явной обратной связи.

Бинарная обратная связь

В большинстве случаев только обратная связь, говорящая о правильности прогноза, может предоставить вам новые данные обучения вместе со связанным идентификатором отслеживания. Например, в системе многоклассовой классификации обратная связь от пользователя может только сообщить вам, верен предсказанный класс или нет. Если предсказанный класс помечен как неправильный, вы не знаете, какой из других классов будет правильным. Если предсказанный класс помечен как правильный, объединение данных с результатом прогноза формирует новый обучающий пример. Задача бинарной классификации – единственный случай, когда вы можете использовать обратную связь, что прогноз неверен. В данном случае эта обратная связь говорит нам, что пример относится к противоположному классу – единственному из возможных двух.

Переклассификация или исправление

Пользователь предоставляет модели правильный ответ. В этом случае сочетание «пара входных данных плюс новая классификация» является новым примером обучения и должно получить идентификатор отслеживания.

Отслеживание неявной обратной связи

Неявная обратная связь генерирует бинарную обратную связь. Если система рекомендаций предлагает продукт и пользователь просматривает этот продукт, сочетание «продукт и данные пользователя» формирует новый обучающий пример и получает идентификатор отслеживания. Однако если пользователь не просматривает продукт, это не означает, что рекомендация была плохой. В данной ситуации может потребоваться подождать, пока накопится множество элементов обратной связи, прежде чем переучивать модель.

РЕЗЮМЕ

Петли обратной связи превращают обычный рабочий процесс конвейера в жизненный цикл и помогают ему расти и совершенствоваться. Важно включать новые данные в нашу систему машинного обучения, чтобы модель не устаревала и ее точность не снижалась. Убедитесь, что вы выбрали метод обратной связи, который наиболее точно соответствует вашему типу модели и ее метрикам, по которым вы судите о ее успехе.

Для петель обратной связи требуется тщательный контроль. Как только вы начинаете собирать новые данные, очень легко нарушить одно из самых фундаментальных допущений многих алгоритмов машинного обучения: ваши обучающие и контрольные данные получены на основании одного и того же распределения. В идеале ваши обучающие и контрольные данные должны отражать реальный мир, который вы моделируете, но на практике это далеко не так. Поэтому когда вы собираете новые данные, важно создавать новые контрольные наборы данных, а также обучающие наборы данных.

При реализации петель обратной связи вам потребуется тесно сотрудничать с дизайнерами, разработчиками и экспертами по UX, которые занимаются вашим продуктом. Им нужно построить системы, которые будут собирать данные и улучшать вашу модель. Важно, чтобы вы работали с ними, чтобы превратить обратную связь в улучшения, которые увидят пользователи, и сформировать ожидания относительно того, когда обратная связь изменит продукт. Это побуждает пользователей вложить свои усилия в обратную связь.

Здесь следует обратить особое внимание на то, что петли обратной связи могут усилить любое негативное отклонение или несимметричность («несправедливость») в исходной модели. Никогда не забывайте, что в этом процессе может участвовать человек! Постарайтесь предложить пользователям метод обратной связи о том, что модель нанесла кому-то вред, чтобы им было легко отмечать ситуации, которые следует немедленно исправить. В этом случае потребуется гораздо больше подробностей, нежели оценка 1–5 «звезд».

Как только вы настроите петли обратной связи и сможете отслеживать прогнозы своей модели и ответы пользователей на эти прогнозы, у вас будут готовы все элементы, из которых состоит ваш конвейер.

Глава 14

Приватность данных, используемых для машинного обучения

В этой главе мы расскажем о некоторых аспектах приватности данных применительно к конвейерам машинного обучения. Сохранение конфиденциальности в процессе машинного обучения является очень активной областью исследований, и соответствующие возможности только начинают включаться в TensorFlow и другие фреймворки. Мы объясним некоторые принципы, лежащие в основе наиболее многообещающих методов на момент написания этой книги, и покажем несколько практических примеров того, как они могут встраиваться в конвейер машинного обучения.

В этой главе мы рассмотрим три основных метода машинного обучения с сохранением приватности данных: дифференцированная приватность, федеративное обучение и зашифрованное машинное обучение.

ВВЕДЕНИЕ В ПРИВАТНОСТЬ ДАННЫХ

Приватность данных – это вопросы доверия и ограничения доступа к данным, которые люди предпочитают хранить таким образом, чтобы они не являлись общедоступными. Существует много различных методов машинного обучения, сохраняющих приватность данных, и для того, чтобы выбрать подходящий вам способ, необходимо ответить на следующие вопросы:

- Для кого вы пытаетесь сделать данные недоступными?
- Какие части системы могут быть приватными, а какие могут быть открытыми для общего доступа?
- Кто входит в доверенные группы лиц, которые могут просматривать данные?

Ответы на эти вопросы помогут вам решить, какой из методов, описанных в этой главе, лучше всего подходит для вашего варианта использования.

Почему мы заботимся о приватности данных?

Приватность данных становится важной частью проекта машинного обучения. Существует множество правовых требований, касающихся конфиденциальности пользователей, такие как обеспечение соответствия требованиям Генерального регламента о защите персональных данных (General Data Protection Regulation, GDPR), которые вступили в силу в мае 2018 года, и Закон штата Калифорния о защите конфиденциальности потребителей (California Consumer Privacy Act), принятый в январе 2020 года. Существуют этические соображения относительно использования персональных данных для машинного обучения, и пользователи продуктов на базе машинного обучения начинают проявлять все более глубокий интерес к тому, что происходит с их данными. Поскольку машинное обучение традиционно являлось активным потребителем данных и поскольку многие прогнозы, сделанные моделями машинного обучения, основаны на персональных данных, получаемых от пользователей, оно находится на переднем крае дискуссий о приватности данных.

На момент написания этой книги конфиденциальность еще никому не доставалась бесплатно: обеспечение приватности данных сопряжено с затратами, связанными с точностью модели, временем вычислений или и тем, и другим видом затрат. С одной стороны, отсутствие сбора данных полностью обеспечит приватность, но не принесет никакой пользы; но с другой стороны, зная все подробности, пользователи могут позволить нам создавать очень точные модели машинного обучения, предоставляя данные. Машинное обучение, обеспечивающее конфиденциальность, только сейчас начинает развиваться, предлагая способы, где приватность данных может быть повышена без существенного ущерба для точности модели.

В некоторых ситуациях машинное обучение с сохранением конфиденциальности может помочь вам использовать данные, которые в противном случае были бы недоступны для обучения модели машинного обучения из-за несоблюдения конфиденциальности. Однако это не дает вам свободы действий с данными только потому, что вы используете один из методов, описанных в этой главе. Вам следует обсудить свои планы с другими заинтересованными сторонами, в число которых входят владельцы данных, эксперты по безопасности и юридический отдел вашей компании.

Самый простой способ повысить приватность данных

Часто стратегия по умолчанию для создания продукта, основанного на машинном обучении, состоит в том, чтобы собрать все возможные данные, а затем решить, что полезно для обучения модели. Несмотря на то что это делается с согласия пользователя, самый простой способ повысить приватность пользовательских данных – это собирать только те данные, которые необходимы для обучения конкретной модели. Для структурированных данных такие элементы данных, как имя, пол или раса, могут быть просто удалены. Текстовые или графические данные могут обрабатываться, чтобы удалить большую часть персональных данных, например удалить лица с изображений или имена из

текста. Однако в некоторых случаях это может снизить полезность данных или сделать невозможным обучение точной модели. И если данные о расе и поле не собираются, невозможно сказать, является ли модель «предвзятой» по отношению к определенной группе.

Контроль над тем, какие данные будут предоставлены, также может быть передан пользователю: согласие на сбор данных может быть разъяснено более подробно и реализовано более продуманно, чем простой вариант соглашения с условиями предоставления данных или отказ от него, например может предусматриваться вариант, когда пользователи продукта смогут точно указать, какие данные о них разрешено собирать. Это ставит сложные задачи перед проектированием продукта: должны ли пользователи, которые предоставляют меньше данных, получать менее точные прогнозы, нежели пользователи, которые предоставляют больше данных? Как мы отслеживаем согласие во всем нашем конвейере машинного обучения? Это вопросы, которые требуют дополнительного обсуждения в сообществе машинного обучения.

Какие данные должны быть приватными?

В конвейерах машинного обучения часто собираются данные о людях, но некоторые данные более остро нуждаются в сохранении приватности в процессе машинного обучения. Данные, идентифицирующие личность (Personally identifying information, PII), – это данные, которые позволяют непосредственно идентифицировать отдельного человека; к ним относятся, например, его имя, адрес(а) электронной почты, название улицы, указанной в адресе, идентификационный номер и т. д. Для всех этих данных необходимо обеспечивать приватность. Эти данные могут появиться в виде текста в свободном формате, таком как комментарии обратной связи или данные службы поддержки клиентов, а не только в случае, когда эти данные непосредственно запрашиваются у пользователя. Изображения людей в некоторых случаях также могут рассматриваться как PII. Использование и хранение таких данных часто регулируется правовыми и нормативными регламентами и актами, и если в вашей компании есть группа, специализирующаяся на вопросах приватности данных, лучше проконсультироваться с ней, прежде чем начинать проект с применением данных такого типа.

Также особого внимания требуют чувствительные данные. Чувствительные данные нередко определяются как данные, которые могут нанести кому-либо вред, если они будут опубликованы. К таким данным относятся данные о состоянии здоровья или данные о частных компаниях (например, финансовые данные). Следует позаботиться о том, чтобы такие данные не просочились в прогнозы модели машинного обучения.

Еще одна категория данных, для которых следует обеспечить приватность, – это квазиидентифицирующие данные. Такие данные позволяют однозначно идентифицировать кого-либо, если известно достаточное их количество, например по отслеживанию местоположения или данных транзакции по кредитной карте. Если для одного и того же человека известны несколько точек местоположения, то на основе этих данных формируется уникальный след. Такие данные могут впоследствии быть объединены с другими наборами данных для повторной идентификации этого человека. В декабре 2019 года New York Times опубликовала подробную статью (см. <https://oreil.ly/VPea0>) о повторной иден-

тификации с использованием данных мобильных телефонов. Это лишь один из нескольких голосов, ставящих под сомнение публикацию таких данных.

ДИФФЕРЕНЦИРОВАННАЯ ПРИВАТНОСТЬ

Если мы выявили необходимость обеспечения дополнительной приватности в нашем конвейере машинного обучения, существуют различные методы, которые могут помочь нам повысить приватность при сохранении максимально возможного количества данных. Первый из них, который мы обсудим, – это дифференцированная приватность¹. Дифференцированная приватность (Differential Privacy, DP) – это формализация идеи о том, что запрос или преобразование набора данных не должны показывать, находится ли определенная персона в этом наборе данных. Она выражает математическую меру потери приватности для персоны, данные о которой включены в набор данных, и сводит к минимуму эту потерю приватности за счет добавления шума.

Дифференцированная приватность описывает обещание, данное держателем данных или куратором субъекту данных, и это обещание выглядит так: «Вы не будете испытывать никакого воздействия, отрицательного или какого-либо иного, позволив использовать ваши данные в любом исследовании или анализе, независимо от того, какие другие исследования, наборы данных или источники информации будут доступны».

– Синтия Дворк²

Иными словами, преобразование набора данных не должно изменяться, если данные об одной персоне будут удалены из этого набора данных. В случае моделей машинного обучения прогнозы, которые делает модель, не должны меняться, если данные, относящиеся к одной персоне, будут удалены из обучающего набора. Дифференцированная приватность достигается добавлением определенной разновидности шума или случайности в алгоритм.

Чтобы привести более конкретный пример: одним из самых простых способов достижения дифференцированной приватности является концепция рандомизированного отклика. Это полезно в опросах, где мы задаем деликатный вопрос людям, которые должны на него ответить, например: «Вы когда-нибудь были осуждены за преступление?» Чтобы ответить на этот вопрос, человек, которого спрашивают, подбрасывает монету. Если выпадает «орел», человек отвечает правдиво. Если выпадает «решка», он снова подбрасывает монету

¹ См. публикацию Синтии Дворк «Дифференциальная приватность», в Энциклопедии криптографии и безопасности, изд. Хенк К. А. ван Тильборг и Сушил Яйодиа, Бостон: Springer, 2006 (Cynthia Dwork, «Differential Privacy», in Encyclopedia of Cryptography and Security, ed. Henk C. A. van Tilborg and Sushil Jajodia. Boston: Springer, 2006).

² См. публикацию Синтии Дворк и Аарона Рот «Алгоритмические основы дифференциальной конфиденциальности» // Основы и тенденции теоретической информатики. 9, № 3–4: 211–407, 2014, (Cynthia Dwork and Aaron Roth, «The Algorithmic Foundations of Differential Privacy», Foundations and Trends in Theoretical Computer Science 9, no. 3–4: 211–407, 2014, <https://www.cis.upenn.edu/~aaroht/Papers/privacybook.pdf>).

и отвечает «да», если выпадает «орел», и «нет», если выпадает «решка». Люди, которые участвуют в опросе, сохраняют свою конфиденциальность, потому что они могут сказать, что дали случайный, а не правдивый ответ. Поскольку мы знаем вероятности выпадения «орла» и «решки», то если мы зададим этот вопрос достаточно большому числу людей, то сможем вычислить процент осужденных за преступление с разумной точностью. Точность расчетов возрастает, когда в опросе участвует большее число людей.

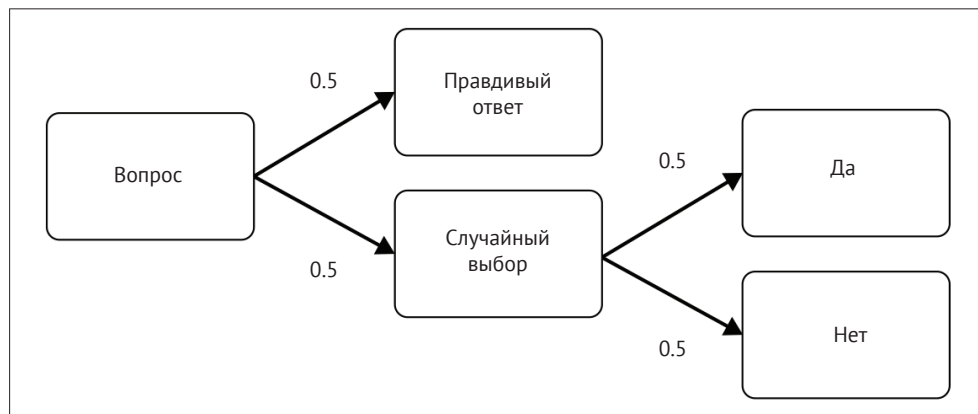


Рис. 14.1. Схема рандомизированного ответа

Эти рандомизированные преобразования являются ключевой концепцией дифференцированной приватности.



Один обучающий пример – один человек

В этой главе для простоты мы предполагаем, что каждый обучающий пример в наборе данных связан с одной персоной или собран на основе данных, относящихся к одной персоне.

Локальная и глобальная дифференцированная приватность

Дифференцированную приватность (differential privacy, DP) можно разделить на два основных метода: локальную и глобальную DP. В локальной DP шум или случайность добавляются на индивидуальном уровне, как в вышеприведенном примере с рандомизированным ответом, поэтому сохраняется приватность между человеком, предоставляющим данные, и исследователем, собирающим данные. В глобальной DP шум добавляется к результатам преобразования всего набора данных. Исследователю, собирающему данные, доверяют необработанные данные, но преобразование данных не раскрывает информацию о предоставившем их человеке.

Для глобальной DP требуется, чтобы мы добавляли меньше шума по сравнению с локальной DP, что приводит к повышению полезности или точности запроса для аналогичной гарантии конфиденциальности. Обратной стороной является то, что для глобальной DP исследователь, собирающий данные, дол-

жен быть доверенным, тогда как при реализации локального DP только отдельные пользователи могут видеть свои собственные необработанные данные.

Эпсилон-дельта и бюджет приватности

Вероятно, наиболее распространенным способом реализации дифференцированной приватности является дифференцированная приватность эпсилон-дельта (ϵ - δ DP). При сравнении преобразования в наборе данных, который включает в себя информацию об одном конкретном человеке, с преобразованием, которое не содержит информации об этом человеке, e^ϵ дает максимальную разницу между результатами этих преобразований. Таким образом, если значение $\epsilon = 0$, оба преобразования возвращают абсолютно одинаковый результат. Если значение ϵ невелико, вероятность того, что два этих преобразования дадут одинаковый результат, выше – чем ниже значение ϵ , тем выше приватность, потому что ϵ измеряет степень гарантии приватности. Если вы делаете запросы к набору данных более одного раза, вам нужно просуммировать значения ϵ каждого запроса, чтобы получить общий бюджет конфиденциальности.

δ – это вероятность того, что значение ϵ не сохраняется, иными словами, вероятность того, что данные человека будут раскрыты в результате рандомизированного преобразования. Обычно мы устанавливаем значение δ приблизительно равным численности генеральной совокупности: для набора данных, содержащего 2000 человек, мы устанавливаем значение $\delta = 1/1000$ ¹.

Возникает вопрос: какую величину ϵ следует выбрать? ϵ позволяет нам сравнивать степень приватности различных алгоритмов и подходов, но абсолютное значение, которое дает нам «достаточную» приватность, зависит от варианта использования².

Чтобы выбрать нужное значение ϵ , может оказаться полезно посмотреть на точность системы при уменьшении значения ϵ , а затем выбрать наиболее приватные параметры, сохранив приемлемую полезность данных для бизнес-задачи. В качестве альтернативы, если последствия утечки данных очень велики, вы можете сначала установить приемлемые значения ϵ и δ , а затем настроить другие гиперпараметры, чтобы получить наилучшую возможную точность модели. Одним из недостатков дифференцированной приватности ϵ - δ является то, что параметр ϵ нелегко интерпретировать. В настоящее время разрабатываются другие подходы, чтобы решить эту проблему, например внедрение объектов *secret* в обучающие данные модели и измерение вероятности того, что они будут раскрыты в прогнозах модели³.

¹ В статье *Algorithmic Foundations of Differential Privacy* («Алгоритмические основы дифференцированной приватности»), опубликованной по ссылке <https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf>, об этом рассказывается более подробно.

² Более подробную информацию можно найти в работе Джастина Хсу и др. «Дифференциальная конфиденциальность: экономический метод выбора Epsilon» (презентация в бумажном виде, симпозиум IEEE Computer Security Foundations 2014, Вена, Австрия, 17 февраля 2014 г.) («Differential Privacy: An Economic Method for Choosing Epsilon» (Paper presentation, 2014 IEEE Computer Security Foundations Symposium, Vienna, Austria, February 17, 2014)), опубликованной по ссылке <https://arxiv.org/pdf/1402.3329.pdf>.

³ См. статью Николаса Карлини и др. «Внедрение *secret*», июль 2019 г. (Carlini et al., 2018, <https://arxiv.org/pdf/1802.08232.pdf>).

Дифференцированная приватность в машинном обучении

Если мы хотим использовать дифференцированную приватность (differential privacy, DP) как часть нашего конвейера машинного обучения, в настоящее время есть несколько вариантов, где ее можно добавить, и мы ожидаем в будущем расширить область ее применения. Во-первых, DP может быть включена в систему федеративного обучения (см. раздел «Федеративное обучение в TensorFlow» этой главы), и в этом случае может применяться либо локальная, либо глобальная дифференцированная приватность. Во-вторых, библиотека TensorFlow Privacy является примером глобальной DP: предполагается, что необработанные данные доступны для обучения модели.

Третий вариант – подход «Приватное агрегирование учительских ансамблей» (Private Aggregation of Teacher Ensembles, PATE)¹. Это сценарий совместного использования данных: в случае если 10 человек разместили данные, а вы нет, они обучают модель локально, и каждый делает прогноз на ваших данных. Затем выполняется запрос с использованием дифференцированной приватности для генерации окончательного прогноза для каждого примера в вашем наборе данных таким образом, что вы не знаете, какая из 10 моделей сделала прогноз. Потом на этих предсказаниях обучается новая модель – эта модель включает информацию из 10 скрытых наборов данных таким образом, что невозможно получить информацию из этих скрытых наборов данных. Инфраструктура PATE показывает, как изменяется параметр ϵ в этом сценарии.

ВВЕДЕНИЕ В TENSORFLOW PRIVACY

TensorFlow Privacy (см. <http://www.cleverhans.io/privacy/2019/03/26/machine-learning-with-differential-privacy-in-tensorflow.html>) добавляет дифференцированную приватность в оптимизатор во время обучения модели. Тип дифференцированной приватности, используемый в TF-Privacy, является примером глобальной DP: шум добавляется после того, как данные собраны, так что конфиденциальные данные не отображаются в предсказаниях модели. Это позволяет нам обеспечивать строгие гарантии дифференцированной приватности, чтобы данные, относящиеся к отдельному человеку, не запоминались, при этом достигалась бы максимальная точность модели. Как показано на рис. 14.2, в этой ситуации необработанные данные доступны для надежного хранилища данных и обучающего механизма моделей, но окончательные прогнозы не являются надежными.

¹ См. публикацию Николаса Паперно и др. «Полуконтролируемая передача знаний для глубокого обучения на основе данных частного обучения», октябрь 2016 г. (Nicolas Papernot et al., «Semi-Supervised Knowledge Transfer for Deep Learning from Private Training Data», October 2016, <https://arxiv.org/abs/1610.05755>).

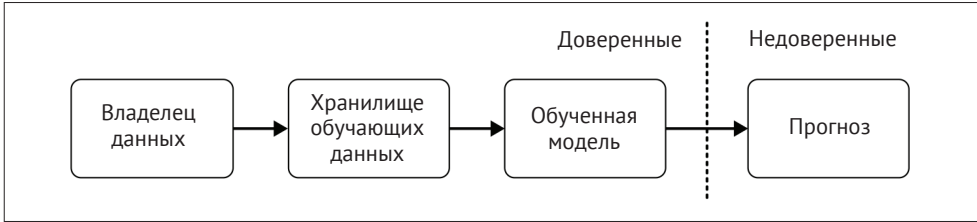


Рис. 14.2. Доверенные стороны для дифференцированной приватности

Обучение с оптимизатором, использующим подход дифференцированной приватности

Алгоритм оптимизатора модифицируется путем добавления случайных шумов к градиентам. Обновления градиента сравниваются с каждой отдельной точкой данных или без нее, гарантируя, что невозможно определить, была ли конкретная точка данных включена в обновление градиента. Кроме того, параметры градиента ограничиваются таким образом, чтобы они не становились слишком большими – это ограничивает вклад любого обучающего примера. В качестве приятного бонуса это также помогает предотвратить чрезмерное обучение.

TF-Privacy можно установить с помощью `pip`. На момент написания этой книги использовалась версия TensorFlow 1.x.

```
$ pip install tensorflow_privacy
```

Мы начнем с простого примера бинарной классификации `tf.keras`:

```
import tensorflow as tf

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Оптимизатор, использующий дифференцированную приватность, требует, чтобы мы установили два дополнительных гиперпараметра по сравнению с обычной моделью `tf.keras`: коэффициент шума и ограничение нормы L2. Лучше всего изменить их в соответствии с вашим набором данных и измерить их влияние на значение ϵ .

```
NOISE_MULTIPLIER = 2
NUM_MICROBATCHES = 32 ❶
LEARNING_RATE = 0.01
POPULATION_SIZE = 5760 ❷
L2_NORM_CLIP = 1.5
BATCH_SIZE = 32 ❸
EPOCHS = 70
```

- ❶ Размер пакета (BATCH_SIZE) должен быть делим без остатка на число микропакетов (NUM_MICROBATCHES)
- ❷ Количество примеров в обучающем наборе
- ❸ Размер генеральной совокупности (POPULATION_SIZE) должен быть делим без остатка на размер пакета (BATCH_SIZE)

Затем инициализируйте оптимизатор, использующий дифференцированную приватность:

```
from tensorflow_privacy.privacy.optimizers.dp_optimizer \
    import DPGradientDescentGaussianOptimizer

optimizer = DPGradientDescentGaussianOptimizer(
    l2_norm_clip=L2_NORM_CLIP,
    noise_multiplier=NOISE_MULTIPLIER,
    num_microbatches=NUM_MICROBATCHES,
    learning_rate=LEARNING_RATE)

loss = tf.keras.losses.BinaryCrossentropy(
    from_logits=True, reduction=tf.losses.Reduction.NONE) ❶
```

- ❶ Потеря должна быть рассчитана для каждого отдельного примера, а не для всего мини-пакета

Обучение приватной модели выполняется так же, как обычное обучение для модели `tf.keras`:

```
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

model.fit(X_train, y_train,
          epochs=EPOCHS,
          validation_data=(X_test, y_test),
          batch_size=BATCH_SIZE)
```

Расчет параметра ϵ

Теперь мы должны рассчитать значения параметров дифференцированной приватности для нашей модели и выбранного нами коэффициента шума и ограничения параметров градиента.

```
from tensorflow_privacy.privacy.analysis import compute_dp_sgd_privacy

compute_dp_sgd_privacy.compute_dp_sgd_privacy(n=POPULATION_SIZE,
                                              batch_size=BATCH_SIZE,
                                              noise_multiplier=NOISE_MULTIPLIER,
                                              epochs=EPOCHS,
                                              delta=1e-4) ❶
```

- ❶ Значение `delta` установлено равным $(1/\text{размер набора данных, округленный до ближайшего порядка})$.



TFP поддерживает только TensorFlow 1.X

Мы демонстрируем, как можно преобразовать пример проекта из предыдущих глав в модель DP в нашем репозитории GitHub (см. <https://oreil.ly/bmlp-git>). Оптимизатор, использующий подход дифференцированной приватности, добавлен в функцию `get_model` (об этой функции рассказывалось в главе 6). Однако эту модель нельзя использовать в нашем конвейере TFX, пока TFP не поддерживает TensorFlow 2.X.

Окончательный результат нашего расчета, значение ϵ , говорит нам о степени гарантии приватности для нашей конкретной модели. Затем мы можем исследовать, как изменение вышеуказанных гиперпараметров – ограничения нормы L2 и коэффициента шума – влияет как на значение ϵ , так и на точность нашей модели. Если значения этих двух гиперпараметров будут увеличены, а все остальные значения будут оставаться неизменными, ϵ будет уменьшаться (поэтому гарантия приватности становится выше). В какой-то момент точность модели начнет снижаться, и модель перестанет быть полезной. Эту обратную зависимость параметров можно исследовать, чтобы получить как можно более строгие гарантии приватности при сохранении точности используемой модели.

ВВЕДЕНИЕ В ФЕДЕРАТИВНОЕ ОБУЧЕНИЕ

Федеративное обучение (Federated learning, FL) – это протокол, в котором обучение модели машинного обучения распределяется по множеству различных устройств, а обученная модель объединяется на центральном сервере. Ключевым моментом является то, что необработанные данные никогда не покидают отдельные устройства и никогда не объединяются в одном месте. Это сильно отличается от традиционной архитектуры сбора набора данных в локализованном месте и последующего обучения модели.

Федеративное обучение чаще всего оказывается полезным в контексте использования с мобильных устройств с распределенными данными или в контексте использования из браузера пользователя. Другой возможный сценарий применения – это совместное использование конфиденциальных данных, которые распределяются между несколькими владельцами данных. Например, стартап, специализирующийся на разработке приложений с использованием технологий искусственного интеллекта, может планировать обучение модели для обнаружения рака кожи. Графические файлы с изображениями рака кожи принадлежат многим больницам, но они не могут быть централизованы в одном месте по соображениям конфиденциальности, а также вследствие юридических ограничений. FL позволяет стартапу обучать модель, при этом сами данные не покинут больницы.

В случае федеративного обучения каждый из клиентов получает архитектуру модели и некоторые инструкции для обучения. Затем модель обучается на каждом клиентском устройстве, и рассчитанные веса возвращаются на центральный сервер. Это немного повышает конфиденциальность пользователя, поскольку в случае перехвата данных перехватчику труднее узнать что-либо

о пользователе на основании данных о весах модели, чем получив необработанные данные; однако это не дает никакой гарантии конфиденциальности. Этап распределения обучения модели не предоставляет пользователю никакой повышенной конфиденциальности со стороны компании, собирающей данные, потому что компания часто может определить, какими были исходные данные, зная архитектуру модели и весовые коэффициенты.

Однако есть еще один очень важный шаг в федеративном обучении: безопасное агрегирование весов в центральную модель. Для этого существует ряд алгоритмов, но все они требуют, чтобы сторона, централизирующая данные, не пыталась проверять веса, прежде чем они будут объединены.

На рис. 14.3 показано, какие стороны имеют доступ к личным данным пользователей в условиях федеративного обучения. Компания, собирающая данные, может настроить засекречивающее усреднение так, чтобы не видеть веса моделей, возвращаемые пользователями, или же выполнить безопасную агрегацию может нейтральная третья сторона. В этом случае только сами пользователи будут видеть свои данные.

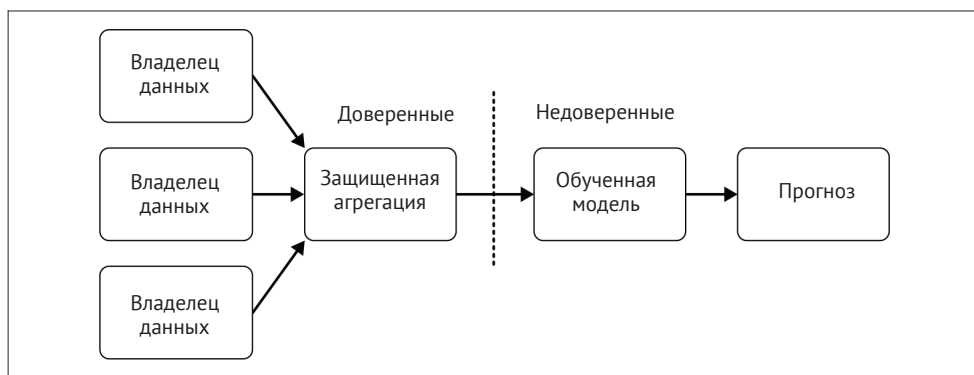


Рис. 14.3. Доверенные стороны в федеративном обучении

Дополнительным расширением FL, позволяющим сохранить конфиденциальность, является включение в эту методику дифференцированной приватности (differential privacy, DP). При этом DP ограничивает объем информации, которую каждый пользователь может передать в окончательную модель. Исследования показали, что полученные модели почти так же точны, как и модели без DP, если число пользователей достаточно велико¹. Однако пока такая возможность не была реализована ни для TensorFlow, ни для PyTorch.

В число примеров FL, выпущенных и работающих для широкой аудитории пользователей, входит клавиатура Google GBoard для мобильных телефонов Android (см. <https://oreil.ly/LXtSN>). Google может обучить модель, чтобы получить наиболее точное предсказание следующего слова, не узнавая ничего о личных сообщениях пользователей.

¹ См. публикацию Робина С. Гейера и др. «Дифференциально-частное федеративное обучение: перспектива на уровне клиента» (Robin C. Geyer et al., «Differentially Private Federated Learning: A Client Level Perspective», December 2017, <https://arxiv.org/abs/1712.07557>).

Федеративное обучение наиболее полезно в тех случаях, для которых характерны следующие особенности¹:

- необходимые для модели данные могут быть собраны только из распределенных источников;
- количество источников данных велико;
- данные имеют конфиденциальный характер;
- данные не требуют дополнительной маркировки – метки проставляются непосредственно пользователем и не покидают источник;
- в идеале данные берутся из почти идентичных распределений.

Федеративное обучение вводит много новых особенностей в устройство системы машинного обучения: например, не все устройства могут собирать новые данные между одним циклом обучения и следующим за ним циклом, не все устройства постоянно включены и т. д. Собранные данные не сбалансированы и практически уникальны для каждого устройства. Проще всего получить достаточно данных для каждого прогона обучения, когда пул возможных устройств велик. Для любого проекта, использующего федеративное обучение, должна быть разработана новая безопасная инфраструктура².

Необходимо соблюдать осторожность, чтобы избежать проблем с производительностью на устройствах, которые обучают модели FL. Обучение может быстро разрядить батарею на мобильном устройстве или вызвать интенсивное использование данных, что приведет к дополнительным затратам для пользователя. Несмотря на то что вычислительная мощность мобильных телефонов быстро растет, они по-прежнему способны обучать лишь небольшие модели, поэтому более сложные модели следует обучать на центральном сервере.

Федеративное обучение в TensorFlow

TensorFlow Federated (TFF) имитирует распределенный подход федеративного обучения и содержит версию реализации стохастического градиентного спуска (Stochastic Gradient Descent, SGD), который может вычислять обновления распределенных данных. Обычный SGD требует, чтобы обновления вычислялись для пакетов централизованного набора данных, и этот централизованный набор данных отсутствует в параметре федеративного обучения. На момент написания этой книги TFF в основном был ориентирован на исследования и эксперименты по новым федеративным алгоритмам.

¹ Более подробно об этом рассказывается в статье Х. Брендана МакМахана и др. «Изучение глубинных сетей на основе децентрализованных данных с точки зрения коммуникации» (H. Brendan McMahan et al., «Communication-Efficient Learning of Deep Networks from Decentralized Data», Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, PMLR 54 (2017): 1273–82, <https://arxiv.org/pdf/1602.05629.pdf>).

² Более подробно о проектировании системы для федеративного обучения рассказывается в докладе о проектировании системы для FL Кейта Бонавица и др. «На пути к федеративному обучению в масштабе: проектирование системы» (презентация, материалы 2-й конференции SysML, Пало-Альто, Калифорния, 2019 г.) (Keith Bonawitz et al., «Towards Federated Learning at Scale: System Design» (Presentation, Proceedings of the 2nd SysML Conference, Palo Alto, CA, 2019), <https://arxiv.org/pdf/1902.01046.pdf>).

PySyft (см. <https://oreil.ly/qlAWWh>) – это платформа Python с открытым исходным кодом для машинного обучения с сохранением конфиденциальности, разработанная организацией OpenMined. Она содержит реализацию федеративного обучения с использованием протокола безопасного многостороннего вычисления (об этом будет более подробно рассказываться в следующем разделе) для агрегирования данных. Первоначально она была разработана для поддержки моделей PyTorch, но недавно была выпущена версия для TensorFlow¹.

ЗАШИФРОВАННОЕ МАШИННОЕ ОБУЧЕНИЕ

Зашифрованное машинное обучение является еще одной областью машинного обучения с сохранением конфиденциальности пользователей, которой в настоящее время уделяется много внимания со стороны исследователей и практиков. Оно опирается на технологии и исследования криптографического сообщества и применяет эти методы для машинного обучения. Основными методами, которые были приняты до сих пор, являются гомоморфное шифрование (homomorphic encryption, HE) и безопасное многостороннее вычисление (secure multi-party computation, SMPC). Есть два способа использовать эти методы: шифрование модели, которая уже была обучена на незашифрованных текстовых данных, и шифрование всей системы (если данные должны оставаться зашифрованными во время обучения).

Гомоморфное шифрование (homomorphic encryption, HE) аналогично шифрованию с открытым ключом, но отличается тем, что данные не нужно дешифровать, прежде чем они будут использоваться в вычислениях. Вычисление (например, получение прогнозов из модели машинного обучения) может выполняться на зашифрованных данных. Пользователь может предоставить свои данные в зашифрованном виде, используя ключ шифрования, который хранится локально, затем получить зашифрованный прогноз и расшифровать его, чтобы получить прогноз модели для своих данных. Это гарантирует приватность пользователя, потому что его данные не передаются той стороне, которая обучала модель.

SMPC позволяет нескольким сторонам объединять данные и выполнять вычисления на их основе, а затем просматривать результаты вычислений на своих собственных данных, ничего не зная о данных других сторон. Это достигается путем «разделения секрета» (разделения secret между несколькими лицами), в основу которого положен следующий подход: любое отдельное значение делится на части, которые отправляются различным сторонам. Исходное значение не может быть восстановлено из какой-либо части, но вычисления могут выполняться для каждой части в отдельности. Результат вычислений не будет иметь смысла, пока все части не будут объединены.

Обе эти техники имеют определенную стоимость. На момент написания этой книги гомоморфное шифрование редко использовалось для обучения моделей машинного обучения: оно вызывало замедление на несколько порядков как в процессе обучения, так и в процессе выдачи прогнозов. Поэтому мы больше не будем обсуждать гомоморфное шифрование. SMPC также имеет определен-

¹ <https://medium.com/dropoutlabs/introducing-pysyft-tensorflow-cc361ac75137>.

ные накладные расходы с точки зрения времени сетевого обмена данными, когда общие ресурсы и результаты передаются между участвующими в обучении сторонами, но этот метод показывает значительно более высокую производительность, чем HE. Эти методы наряду с FL полезны в ситуациях, когда данные не могут быть собраны в одном месте. Однако они не мешают моделям запоминать конфиденциальные данные, и лучшим решением для устранения этого недостатка является DP.

Зашифрованное машинное обучение реализовано в TensorFlow в виде отдельного пакета TensorFlow Encrypted (TFE) (см. <https://tf-encrypted.io/>), начало разработке которого положила компания Cape Privacy (см. <https://capeprivacy.com/>). Оно также поддерживает безопасное усреднение, необходимое для федеративного обучения (см. <https://oreil.ly/VVPJx>).

Зашифрованное обучение модели

Первая ситуация, когда вы можете использовать зашифрованное машинное обучение, – это обучение моделей на уже зашифрованных данных. Это полезно, когда специалист в области искусственного интеллекта, занимающийся обучением модели, не должен иметь возможность прочитать необработанные данные или когда необработанными данными владеют две или более стороны. Они хотят обучать модель, используя данные, предоставляемые каждой из сторон, но не хотят делиться необработанными данными. Как показано на рис. 14.4, в этом сценарии доверяют только владельцу или владельцам данных.

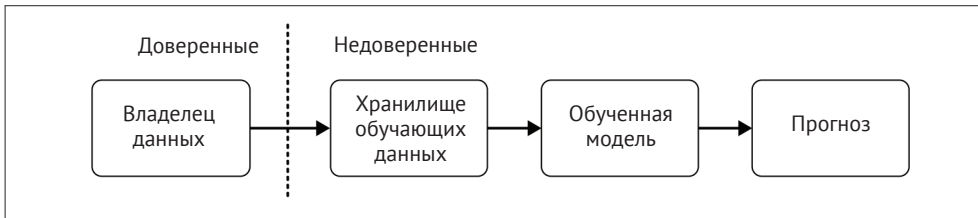


Рис. 14.4. Доверенные стороны в зашифрованном обучении модели

Пакет TFE может использоваться для зашифрованного обучения модели в данной ситуации. Как обычно, он устанавливается с помощью `pip`:

```
$ pip install tf_encrypted
```

Первым шагом в построении модели TF с шифрованием является определение класса, который выдает обучающие данные в виде пакетов. Этот класс реализуется локально владельцем или владельцами данных. Он преобразуется в зашифрованные данные с помощью декоратора:

```
@tfe.local_computation
```

Написание кода, реализующего обучение модели, в TF-Encrypted практически идентично обычному написанию кода для модели Keras – просто замените `tf` на `tfe`.

```
import tf_encrypted as tfe
```

```
model = tfe.keras.Sequential()
model.add(tfe.keras.layers.Dense(1, batch_input_shape=[batch_size, num_features]))
model.add(tfe.keras.layers.Activation('sigmoid'))
```

Единственное отличие состоит в том, что аргумент `batch_input_shape` должен быть передан в первый слой `Dense`.

Рабочие примеры, иллюстрирующие этот подход, включены в документацию TF Encrypted (см. <https://oreil.ly/ghGnu>). В настоящее время не все функциональные возможности стандартной библиотеки Keras включены в состав TF Encrypted, поэтому мы не можем показать пример для демонстрационного проекта в этом формате.

Преобразование обученной модели для обслуживания зашифрованных прогнозов

Второй сценарий, где TF Encrypted может быть полезен, – это когда вы хотите развернуть и обслуживать зашифрованные модели, которые были обучены на незашифрованных данных (см. <https://oreil.ly/HBUBj>). В этом случае, как показано на рис. 14.5, у вас имеется полный доступ к незашифрованным обучающим данным, но вы хотите, чтобы пользователи вашего приложения могли получать приватные прогнозы. Это обеспечивает приватность данных пользователей, которые загружают зашифрованные данные и получают зашифрованный прогноз.

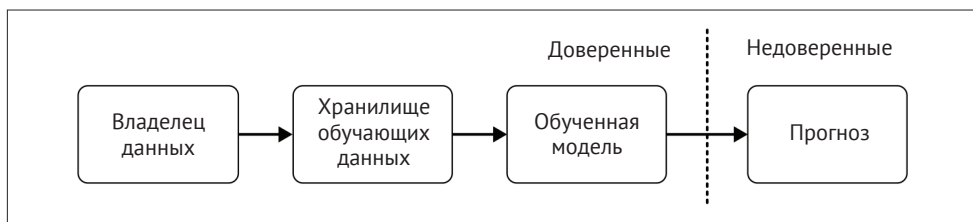


Рис. 14.5. Доверенные стороны при шифровании обученной модели

Такой метод может оказаться наиболее подходящим вариантом для современных конвейеров машинного обучения, поскольку модели можно обучать в обычном режиме, а затем преобразовывать в зашифрованную версию. Он также может использоваться для моделей, которые прошли обучение с использованием метода дифференцированной приватности. Основное отличие такого способа от незашифрованных моделей заключается в том, что требуется несколько серверов: на каждом размещается часть исходной модели. Если кто-то просматривает общий ресурс модели на одном сервере или одну часть данных, отправляемых на один сервер, он ничего не узнает о модели или о данных.

Модели Keras могут быть преобразованы в модели TF Encrypted с помощью следующей команды:

```
tfe_model = tfe.keras.models.clone_model(model)
```

В этом случае необходимо выполнить следующие шаги:

- загрузка и предварительная обработка данных локально на клиенте;
- шифрование данных на клиенте;
- отправка зашифрованных данных на серверы;
- выполнение прогноза на основе зашифрованных данных;
- отправка зашифрованного прогноза клиенту;
- расшифровка прогноза на клиенте и отображение результата пользователю.

TF Encrypted предоставляет серию блокнотов (Notebooks), показывающих, как обслуживать приватные прогнозы (см. <https://oreil.ly/r0cKP>).

ДРУГИЕ МЕТОДЫ ОБЕСПЕЧЕНИЯ ПРИВАТНОСТИ ДАННЫХ

Есть много других методов повышения конфиденциальности пользователей, чьи данные включены в модели машинного обучения. Простое удаление текстовых данных, содержащих имена, адреса, телефонные номера и т. д., может быть на удивление легко выполнено с помощью регулярных выражений и моделей распознавания именованных сущностей.



К-анонимность

К-анонимность (см. <https://oreil.ly/sxQet>), часто известная как «деперсонализация» или «обезличивание», не является хорошим способом повышения конфиденциальности в конвейерах машинного обучения. К-анонимность требует, чтобы каждый индивид в наборе данных был неотличим от $k - 1$ других в отношении их квазиидентификаторов (данных, которые могут косвенно идентифицировать индивидов, таких как пол, раса, почтовый индекс). Это достигается путем объединения или удаления данных до тех пор, пока набор данных не станет удовлетворять этому требованию. Такое удаление данных обычно приводит к значительному снижению точности моделей машинного обучения¹.

РЕЗЮМЕ

Когда вы работаете с личными или конфиденциальными данными, выберите методику обеспечения конфиденциальности данных, которая наилучшим образом подходит для вашего сценария использования, определив, кому можно доверять информацию, какова должна быть производительность вашей моде-

¹ Кроме того, было доказано, что персоны в «обезличенных» наборах данных могут быть повторно идентифицированы с использованием внешних данных – см. публикацию Люка Роше и др. «Оценка успеха повторной идентификации в неполных наборах данных с использованием генеративных моделей» (Luc Rocher et al., «Estimating the Success of Re-identifications in Incomplete Datasets Using Generative Models», Nature Communications 10, Article no. 3069 (2019), <https://www.nature.com/articles/s41467-019-10933-3>).

ли, получили ли вы согласие от пользователей на обработку их данных и каковы условия этого согласия.

Все техники, описанные в этой главе, относятся к новейшим, и их использование в промышленной среде еще не получило широкого распространения. Не думайте, что использование одной из платформ, описанных в данной главе, обеспечивает полную конфиденциальность для ваших пользователей. Здесь не обойтись без трудоемких дополнительных разработок, связанных с реализацией конфиденциальности пользователей в конвейере машинного обучения. Область машинного обучения с сохранением конфиденциальности быстро развивается, и сейчас ведутся новые исследования. Мы призываем вас следить за достижениями в этой области и поддерживаем проекты с открытым исходным кодом, нацеленные на обеспечение конфиденциальности данных, такие как PySyft (см. https://oreil.ly/rj0_c) и TFE (<https://oreil.ly/L5zik>).

Цели обеспечения конфиденциальности данных и машинного обучения часто хорошо согласованы в том смысле, что мы хотим изучать всю генеральную совокупность и делать прогнозы, которые одинаково подходят для всех людей, а не изучать индивидуальные случаи. Добавление конфиденциальности может остановить избыточное обучение модели и ее «подстройку» к данным, относящимся к одному человеку. Мы ожидаем, что в будущем конфиденциальность с самого начала будет являться одной из составных частей конвейеров машинного обучения, когда модели будут обучаться на персональных данных.

Глава 15

Будущее конвейеров машинного обучения и следующие шаги

В последних 14 главах мы описали текущее состояние конвейеров машинного обучения и дали наши рекомендации по их созданию. Конвейеры машинного обучения – это относительно новая концепция, и в этой области еще многое предстоит сделать. В этой главе мы обсудим несколько тем, которые, по нашему мнению, важны, но не используются широко в текущих практиках построения конвейеров, а также рассмотрим следующие шаги для создания конвейеров машинного обучения.

ОТСЛЕЖИВАНИЕ ЭКСПЕРИМЕНТОВ С МОДЕЛЬЮ

На протяжении всей книги мы предполагали, что вы уже проводили эксперименты, и архитектура модели в основном определена. Тем не менее мы хотели бы поделиться некоторыми мыслями о том, как отслеживать эксперименты и сделать проведение экспериментов плавным процессом. Ваш процесс проведения эксперимента может включать изучение потенциальных архитектур моделей, гиперпараметров и наборов признаков. Но независимо от того, что вы исследуете, ключевой момент, на котором мы хотели бы заострить внимание, заключается в том, что ваш экспериментальный процесс должен точно соответствовать вашему производственному процессу.

Независимо от того, оптимизируете ли вы свои модели вручную или настраиваете модели автоматически, получение и совместное использование результатов процесса оптимизации имеет важное значение. Члены команды могут быстро оценить процессы обновления модели. При этом автор моделей может получать автоматические записи проведенных экспериментов. Хорошо налаженное отслеживание экспериментов помогает командам по анализу данных стать более эффективными.

Отслеживание экспериментов также вносит свой вклад в контрольный журнал модели и может защитить вас от потенциальных споров. Если перед группой специалистов по анализу данных стоит вопрос, рассматривался ли по-

граничный случай при обучении модели, отслеживание экспериментов может помочь в отслеживании параметров модели и итераций.

Среди инструментов для отслеживания экспериментов мы можем назвать такие, как инструмент для анализа весов и смещений Weights and Biases (см. <https://www.wandb.com/>) и Sacred (см. <https://oreil.ly/6zK3V>). На рис. 15.1 показан пример использования инструмента Weights and Biases, при этом на графике отображаются потери для каждого прогона обучения модели в зависимости от эпохи обучения. Инструмент предлагает множество различных вариантов визуализации, и мы можем сохранить все гиперпараметры для каждого запуска модели.

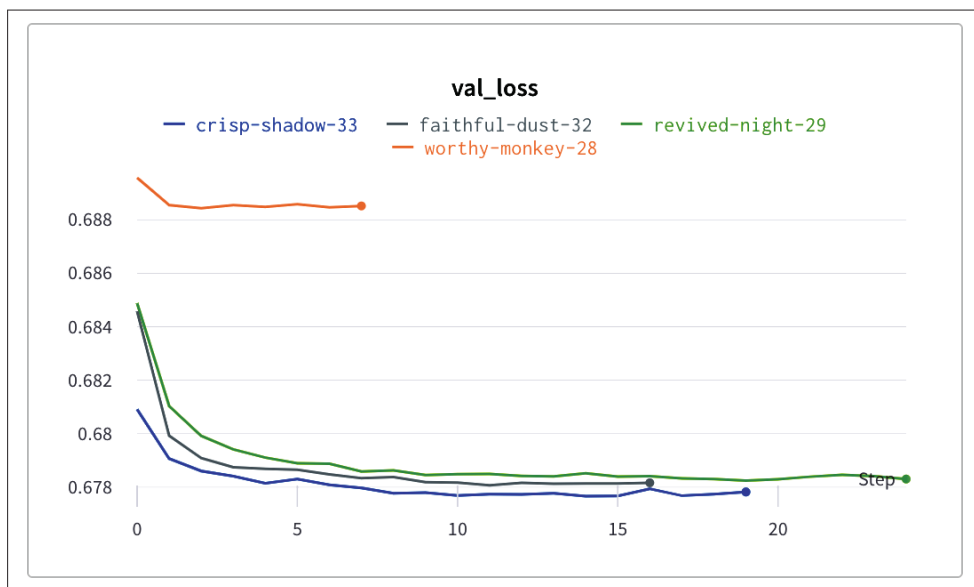


Рис. 15.1. Отслеживание экспериментов в Weights and Biases

В будущем мы ожидаем, что эксперимент и производственный процесс станут более тесно связанными, чтобы специалист по анализу данных мог плавно перейти от опробования новой архитектуры модели к добавлению ее в свой конвейер.

Предложения в области управления релизами модели

В программной инженерии существуют хорошо отработанные процедуры для управления версиями кода и управления выпусками. Большие изменения, которые могут привести к обратной несовместимости версий, находят отражение в изменении номера основной версии (например, с 0.x на 1.0). Небольшие дополнения к функциям отражаются в вариантах изменения в версии (например, номер версии изменяется с 1.0 до 1.1). Но как это применимо в мире машинного обучения? При переходе от одной модели машинного обучения к другой входной формат данных может оставаться неизменным, как и выходной формат прогнозов, поэтому критических изменений нет. Конвейер все

еще работает; никаких ошибок не выдается. Но качество новой модели может кардинально отличаться от предыдущей. Стандартизация конвейеров машинного обучения требует внедрения практики управления версиями моделей.

Мы предлагаем следующую стратегию управления версиями модели:

- при изменении входных данных изменяется дополнительный номер версии модели;
- если изменились гиперпараметры, должен измениться основной номер версии модели. Этот вариант также включает изменение количества слоев в сети или количества узлов в слое;
- если архитектура модели полностью изменяется (например, при переходе с рекуррентной нейронной сети [RNN] на архитектуру Transformer), то необходимо выпускать совершенно новую версию конвейера.

Этап проверки модели определяет, будет ли выпущен релиз – для выпуска нового релиза показатели качества новой модели должны быть лучше, чем у предыдущей модели. На момент написания этой книги на этапе проверки конвейер TFХ использует только одну метрику. Мы ожидаем, что в будущем этап проверки станет более сложным и будет включать другие факторы, такие как время, необходимое для получения результатов работы модели, или точность модели на разных срезах данных.

Будущие возможности конвейеров

В этой книге мы рассмотрели состояние конвейеров машинного обучения на момент ее написания. Но как будут выглядеть конвейеры машинного обучения в будущем? Вот некоторые из возможностей, которые мы хотели бы увидеть.

- Конфиденциальность данных и справедливость (непредвзятость) модели выходят на первый план: на момент написания предполагалось, что конвейер не содержит элементов обучения, для которых применимо понятие конфиденциальности данных и соответствующие ограничения. Анализ непредвзятости проводится, но на этапе ModelValidator можно использовать только общие показатели.
- Использование федеративного обучения, о котором рассказывалось в главе 14. Если предварительная обработка данных и обучение модели происходят на большом количестве отдельных устройств, конвейер машинного обучения должен будет сильно отличаться от того, который мы описали в этой книге.
- Возможность измерять углеродосодержащие выбросы применительно к нашим конвейерам. По мере того как модели становятся больше, их потребление энергии становится значительным. Хотя это часто более актуально в процессе экспериментов (особенно при поиске подходящих архитектур моделей), было бы очень полезно интегрировать измерение углеродосодержащих выбросов в конвейеры.
- Загрузка потоков данных: в этой книге мы рассмотрели только конвейеры, которые обучаются на пакетах данных. Но по мере того как конвейеры данных становятся более сложными, конвейеры машинного обучения должны иметь возможность загружать потоки данных.

Инструменты будущего смогут еще больше абстрагироваться от некоторых процессов, описанных в этой книге, и мы ожидаем, что будущие конвейеры будут еще более удобными, и степень их автоматизации станет еще выше.

Мы также предполагаем, что будущие конвейеры должны будут автоматизировать некоторые другие типы задач машинного обучения. Мы обсудили только обучение с учителем и почти исключительно проблемы классификации. Имеет смысл начать с задач контролируемой классификации, потому что эти задачи – одни из самых простых для понимания и встраивания в конвейеры. Задачи регрессии и другие разновидности контролируемого обучения, такие как подписи к изображениям или генерация текста, можно будет легко заменить в большинстве компонентов конвейера, который мы описываем в этой книге. Но задачи стимулированного обучения и обучения без учителя могут не так хорошо подходить для встраивания в конвейеры машинного обучения. Эти задачи по-прежнему редко используются в производственных системах, но мы ожидаем, что в будущем они получат большее распространение. Компоненты загрузки и проверки данных, а также конструирования признаков нашего конвейера не должны претерпеть существенных изменений для этих новых типов задач, но компоненты обучения, оценки и проверки потребуют значительных изменений. Петли обратной связи также будут выглядеть совсем иначе.

Использование TFX с другими фреймворками машинного обучения

Будущее конвейеров машинного обучения, вероятно, также будет включать открытость в отношении основных фреймворков машинного обучения, так что специалисту по данным не нужно будет делать выбор между средой TensorFlow, PyTorch, scikit-learn и другими средами при построении модели.

Приятно видеть, что TFX движется к независимости от TensorFlow. Как мы обсуждали в главе 4, некоторые компоненты TFX можно использовать с другими фреймворками машинного обучения. Другие компоненты претерпевают изменения, чтобы обеспечить интеграцию с иными фреймворками машинного обучения. Например, компонент `Trainer` теперь предоставляет модуль исполнения (`executor`), который позволяет обучать модели независимо от TensorFlow. Мы надеемся, что мы увидим больше общих компонентов, которые легко интегрируют такие фреймворки, как PyTorch или scikit-learn.

Тестирование моделей машинного обучения

Тестирование моделей машинного обучения – достаточно новая тема в инженерии машинного обучения. Здесь мы имеем в виду не проверку модели, которую мы обсуждали в главе 7, а скорее проверку результатов работы модели. Такие тесты могут быть модульными тестами для модели или полными сквозными тестами взаимодействия модели с приложением.

Помимо проверки непрерывной работы системы, другие тесты могут быть ориентированы на:

- время вывода результата работы модели;
- потребление памяти;
- расход заряда батареи на мобильных устройствах;
- компромисс между размером и точностью модели.

Мы с нетерпением ждем, когда передовой опыт разработки программного обеспечения объединится с практиками науки о данных и тестирование моделей станет частью единого жизненного цикла.

СИСТЕМЫ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ И РАЗВЕРТЫВАНИЯ ДЛЯ МАШИННОГО ОБУЧЕНИЯ

Поскольку оптимизация конвейеров машинного обучения – дело ближайших месяцев, мы увидим, что конвейеры машинного обучения будут двигаться ко все более интенсивному использованию практик непрерывной интеграции и развертывания. Как специалисты по обработке данных и инженеры по машинному обучению мы можем брать в качестве примера процессы разработки программного обеспечения. Например, мы видим возможности для улучшений в области интеграции управления версиями данных в конвейеры машинного обучения или использование передовых практик для облегчения отката развертывания моделей машинного обучения.

СООБЩЕСТВО ИНЖЕНЕРОВ МАШИННОГО ОБУЧЕНИЯ

Поскольку область машинного обучения находится в стадии становления, сообщество, посвященное этой теме, будет иметь огромное значение. Мы с нетерпением ждем возможности поделиться передовым опытом, настраиваемыми компонентами, рабочими процессами, сценариями использования и настройками конвейера с сообществом машинного обучения. Мы надеемся, что эта публикация станет небольшим вкладом в эту развивающуюся область. Подобно DevOps в разработке программного обеспечения, мы надеемся увидеть, что все больше специалистов по данным и инженеров по программному обеспечению будут интересоваться дисциплиной машинного обучения.

РЕЗЮМЕ

Эта книга содержит наши рекомендации о том, как превратить вашу модель машинного обучения в непрерывный конвейер. На рис. 15.2 показаны все шаги, которые, по нашему мнению, необходимы, и инструменты, которые мы считаем лучшими на момент написания этой книги. Мы рекомендуем вам следить за публикациями по данной теме, за новыми разработками и вносить свой вклад в различные проекты с открытым исходным кодом, связанные с конвейерами машинного обучения. Это область чрезвычайно активного развития, где часто выпускаются новые решения.

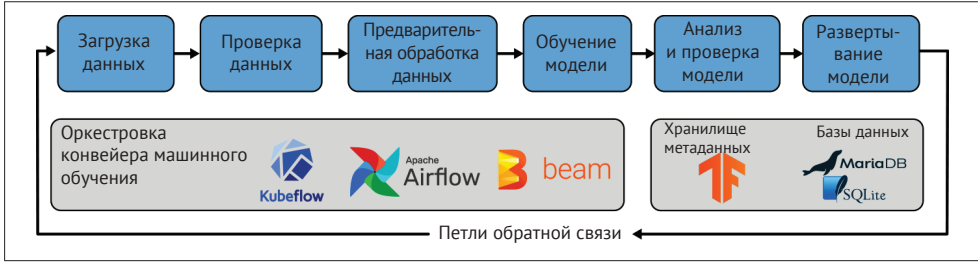


Рис. 15.2. Архитектура конвейера машинного обучения

Конвейер на рис. 15.2 имеет три чрезвычайно важные особенности: он *автоматизирован*, *масштабируется* и *воспроизводится*. Поскольку он автоматизирован, он освобождает специалистов по обработке данных от необходимости ручной поддержки моделей и дает им время для экспериментов с новыми моделями. Так как он масштабируется, он может расширяться для обработки больших объемов данных. А поскольку он воспроизводим, то как только вы настроите его в своей инфраструктуре для одного проекта, вам будет легко построить второй. Это необходимо для успешного построения конвейеров машинного обучения.

Приложение **A**

Введение в инфраструктуру машинного обучения

В этом приложении дается краткое введение в некоторые из наиболее полезных инструментов для построения инфраструктуры машинного обучения: контейнеры в форме Docker или Kubernetes. Хотя в этот момент вы уже можете передать свой конвейер команде разработчиков программного обеспечения, все же тому, кто создает конвейеры машинного обучения, полезно знать об этих инструментах.

Что такое контейнер?

Все операционные системы Linux основаны на файловой системе или структуре каталогов, которая включает все жесткие диски и разделы. Из корня этой файловой системы (обозначается как /) вы можете получить доступ почти ко всем аспектам системы Linux. Контейнеры создают новый корень меньшего размера и используют его как «меньший Linux» на более крупном хосте. В результате у вас в распоряжении имеется целый отдельный набор библиотек, посвященных конкретному контейнеру. Кроме того, контейнеры позволяют управлять такими ресурсами, как время ЦП или память для каждого контейнера.

Docker – это удобный API, который управляет контейнерами. Контейнеры можно создавать, объединять в другие контейнеры, сохранять и развертывать (в том числе повторно) с помощью Docker. Это также позволяет разработчикам создавать контейнеры локально, а затем публиковать их в центральном реестре, из которого другие могут извлечь и немедленно запустить контейнер.

Управление зависимостями – большая проблема в машинном обучении и науке о данных. Независимо от того, на каком языке вы пишете – на R или на Python, ваш код почти всегда зависит от сторонних модулей. Эти модули часто обновляются и могут вызвать критические изменения в вашем конвейере при конфликте версий. Используя контейнеры, вы можете предварительно упаковать свой код обработки данных вместе с нужными версиями модуля и избежать этих проблем.

ВВЕДЕНИЕ В DOCKER

Чтобы установить Docker на Mac или Windows, перейдите по ссылке <https://docs.docker.com/install> и загрузите последнюю стабильную версию Docker Desktop для своей операционной системы.

Для операционной системы Linux Docker предоставляет очень удобный сценарий для установки Docker с помощью всего пары команд:

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
```

Вы можете проверить, правильно ли работает ваш экземпляр Docker, используя команду

```
docker run hello-world
```

Начальные сведения об образах Docker

Образ Docker является основой контейнера и состоит из набора изменений в корневой файловой системе и параметров выполнения, необходимых для запуска контейнера. Образ сначала необходимо создать (выполнить сборку), прежде чем его можно будет запустить.

Полезная концепция образов Docker – это уровни хранения. Создание образа означает установку почти целой выделенной ОС Linux для вашего пакета. Чтобы избежать выполнения этой операции каждый раз, Docker использует многоуровневую файловую систему. Вот как это работает: если первый слой содержит файлы A и B, а второй слой добавляет файл C, в результирующих файловых системах отображаются A, B и C. Если мы хотим создать второй образ, использующий файлы A, B и D, нам нужно изменить только второй уровень, чтобы добавить файл D. Это означает, что у нас могут быть базовые образы со всеми основными пакетами, а затем мы можем выполнить необходимые изменения, специфичные для вашего образа, как показано на рис. А.1.

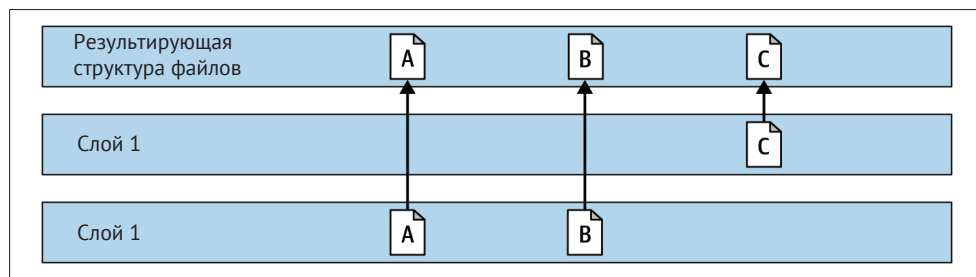


Рис. А.1. Пример многоуровневой файловой системы

Имена образов Docker называются тегами. Для имен тегов используется шаблон `реестр_docker/пространство_имен_docker/имя_образа:тег`. Например, `docker.io/tensorflow/tensorflow:nightly` будет указывать на образ `tensorflow` в DockerHub в пространстве имен `tensorflow`. Тег обычно используется для обозначения версий определенного образа. В нашем примере тег `nightly` зарезервирован для ночных сборок TensorFlow.

Образы Docker создаются на основе файла `Dockerfile`. Каждая строка в `Dockerfile` начинается с одного из нескольких операторов. Наиболее важные из них следующие.

FROM

Указывает основной контейнер Docker для сборки. Мы всегда будем использовать этот оператор. Для загрузки доступно множество основных контейнеров, например `ubuntu`.

RUN

Запускает `bash`. Это основная команда большинства образов Docker. Именно при помощи этой команды мы выполняем установку пакетов, создание каталогов и т. д. Поскольку каждая строка создает слой в образе, хорошая практика – включать установку пакетов и другие длинные задачи в качестве одной из первых строк в `Dockerfile`. Это означает, что во время перестроения Docker попытается использовать слои из кеша.

ARG

Конфигурирует аргументы. Этот оператор полезно использовать, если вы хотите иметь несколько вариантов одного и того же образа, например для среды разработки и производственной среды.

COPY

Копирует файлы из контекста. Путь к контексту – это аргумент, используемый в `docker build`. Контекст – это набор локальных файлов, которые Docker открывает во время сборки и использует их только в процессе. Этот оператор можно использовать для копирования исходного кода в контейнер.

ENV

Устанавливает переменную среды. Эта переменная будет частью образа и будет видна при сборке и запуске.

CMD

Это команда по умолчанию для контейнера. Хорошая практика для Docker – запускать одну команду для каждого контейнера. Затем Docker будет отслеживать эту команду, завершать работу при выходе и отправлять из нее `STDOUT` в журналы экземпляров Docker. Другой способ указать эту команду – использовать оператор `ENTRYPOINT`. Между ними есть несколько тонких различий, но здесь мы сосредоточимся на `CMD`.

USER

Пользователь по умолчанию в контексте контейнера. Этот пользователь отличается от пользователей хост-системы. Вы должны создать пользователя во время сборки, если хотите запускать команды вместе, как единый пакет.

WORKDIR

Каталог по умолчанию для образа. Это каталог, из которого будет запускаться команда по умолчанию.

EXPOSE

Указывает порты, которые будет использовать контейнер. Например, для службы HTTP необходимо указать `EXPOSE 80`.

Создание вашего первого образа Docker

Создадим наш первый образ!

Во-первых, нам нужно создать новый каталог для нашего небольшого проекта Docker:

```
$ mkdir hello-docker
$ cd hello-docker
```

В этом каталоге создайте файл *Dockerfile* со следующим содержимым:

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install cowsay
CMD /usr/games/cowsay "Hello Docker"
```

Для его сборки используйте команду `docker build . -t hello-docker`. Флаг `-t` указывает тег для этого образа. Вы увидите серию команд, которые выполняются в контейнере. Каждый слой в нашем образе (соответствующий каждой команде в *Dockerfile*) вызывается во временном контейнере, в котором запущены предыдущие уровни. Разница сохраняется, и мы получаем полный образ. Первый уровень (который мы не создаем) основан на Linux Ubuntu. Команда `FROM` в *Dockerfile* сообщает Docker, что нужно извлечь этот образ из реестра, в нашем случае DockerHub, и использовать его в качестве базового образа.

После завершения сборки команда `docker images` должна вывести приблизительно такой результат:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-docker	latest	af856e494ed4	2 minutes ago	155MB
ubuntu	latest	94e814e2efa8	5 weeks ago	88.9MB

В результате мы должны увидеть базовый образ Ubuntu и наш новый образ.

Даже если мы создали этот образ, это не значит, что он готов к использованию. Следующим шагом будет запуск образа. Команда запуска `docker run` – пожалуй, самая важная команда в Docker. Она создает новый контейнер из существующего образа (или, если образа нет в системе, она попытается извлечь его из реестра). Чтобы запустить наш образ, мы должны выполнить команду `docker run -it hello-docker`. Эта команда должна показать нам результат работы нашей команды `cowsay`.



Реестры Docker

Одна из самых сильных сторон Docker – простота публикации созданного образа. Хранилище образов Docker называется реестром. Реестр Docker по умолчанию, называемый DockerHub, поддерживается Docker, Inc. Учетная запись на DockerHub бесплатна и позволяет публиковать общедоступные образы.

Знакомство с интерфейсом командной строки Docker (Docker CLI)

Docker CLI – это основной способ взаимодействия с образами и контейнерами на вашем локальном компьютере. В этом разделе мы обсудим наиболее важные команды и параметры для этого. Начнем с `docker run`.

Есть много важных параметров, которые возможно передать в `docker run`. С их помощью мы можем переопределить большинство параметров, установленных в `Dockerfile`. Это важно, потому что многие образы Docker будут иметь стандартную команду по умолчанию, но часто мы хотим их запускать иначе, с другими параметрами. Давайте посмотрим на наш пример `cowsay`:

```
docker run -it hello-docker /usr/games/cowsay "Our own message"
```

Аргумент, который следует за тегом образа, переопределит команду по умолчанию, которую мы установили в `Dockerfile`. Это лучший способ указать наши собственные флаги командной строки для двоичных файлов, используемых по умолчанию. Другие полезные флаги для запуска Docker следующие.

`-it`

Обозначает «интерактивный» (i) и `tty` (t), что позволяет нам взаимодействовать с командой, выполняемой из нашей оболочки.

`-v`

Монтирует том Docker или каталог хоста в контейнер, например каталог, содержащий наборы данных.

`-e`

Передаёт конфигурации через переменные среды. Например, `docker run -e MYVARIABLE=value image` создаст в контейнере переменную среды `MYVARIABLE`.

`-d`

Позволяет запускать контейнер в автономном режиме, что идеально для длительных задач.

`-p`

Перенаправляет порт хоста в контейнер, чтобы внешние службы могли взаимодействовать с контейнером по сети. Например, `docker run -d -p 8080:8080 imagename` перенаправит `localhost:8080` на порт 8080 контейнера.



Docker Compose

Команда `docker run` может стать довольно сложной, когда вы начинаете монтировать каталоги, управлять ссылками на контейнеры и т. д. *Docker Compose* – проект, который поможет в этом. Он позволяет вам создать файл `dockercompose.yaml`, в котором вы можете указать все свои параметры Docker для любого количества контейнеров. Затем вы можете связать контейнеры вместе в сети или смонтировать одни и те же каталоги.

Другие полезные команды Docker перечислены ниже.

`docker ps`

Отображает все запущенные контейнеры. Чтобы также показать контейнеры, завершившие работу, добавьте флаг `-a`.

`docker images`

Список всех образов, имеющихся на компьютере.

`docker inspect container_id`

Позволяет подробно изучить конфигурацию контейнера.

`docker rm`

Удаляет контейнеры.

`docker rmi`

Удаляет образы.

`docker logs`

Отображает данные STDOUT и STDERR, созданные контейнером, что очень полезно при отладке.

`docker exec`

Позволяет вызывать команду в работающем контейнере. Например, `docker exec -it container_id bash` позволит вам войти в среду контейнера с помощью `bash` и изучить ее изнутри. Флаг `-it` работает аналогично тому, как он работает в `docker run`.

ВВЕДЕНИЕ В KUBERNETES

До сих пор мы только что говорили о контейнерах Docker, работающих на одной машине. Что произойдет, если вы захотите масштабировать ваш конвейер? Kubernetes – это проект с открытым исходным кодом, изначально разработанный Google, который управляет планированием и масштабированием вашей инфраструктуры. Он динамически масштабирует нагрузку на множество серверов и отслеживает вычислительные ресурсы. Kubernetes также позволяет достичь максимальной эффективности, размещая несколько контейнеров на одной машине (в зависимости от их размера и потребностей), и управляет обменом данными между контейнерами. Он может работать на любой облачной платформе – AWS, Azure или GCP.

Некоторые определения Kubernetes

Одна из самых сложных частей в самом начале работы с Kubernetes – это терминология. Вот несколько определений, которые могут вам помочь.

Кластер

Кластер – это набор машин, содержащий центральный узел, управляющий сервером Kubernetes API, и множество рабочих узлов.

Узел

Узел – это отдельная машина (физическая или виртуальная) в кластере.

Модуль (*Pod*)

Модуль – это группа контейнеров, которые работают вместе на одном узле. Часто модуль содержит только один контейнер.

Kubelet

Kubelet – это агент Kubernetes, который управляет связью с центральным узлом на каждом рабочем узле.

Сервис

Сервис – это группа модулей и политики для доступа к ним.

Том

Том – это пространство для хранения данных, совместно используемое всеми контейнерами в одном модуле.

Пространство имен

Пространство имен – это виртуальный кластер, который делит пространство в физическом кластере на разные среды. Например, мы можем разделить кластер на среду разработки и промышленную среду, или на среды для разных команд.

ConfigMap

ConfigMap предоставляет API для хранения неконфиденциальной информации о конфигурации (переменные среды, аргументы и т. д.) в Kubernetes. ConfigMaps полезны для изоляции конфигурации от образов контейнеров.

kubectl

kubectl – это интерфейс командной строки для Kubernetes.

Начало работы с Minikube и kubectl

Мы можем создать простой локальный кластер Kubernetes, используя инструмент под названием Minikube. Minikube позволяет легко настроить Kubernetes в любой операционной системе. Он создает виртуальную машину, устанавливает на нее Docker и Kubernetes и добавляет подключенного к ней локального пользователя.



Не используйте Minikube в промышленной среде

Minikube нельзя использовать в промышленной среде; скорее, он разработан как быстрая и легкая локальная среда. Самый простой способ получить доступ к продукту Kubernetes – приобрести Kubernetes как услугу у любого крупного провайдера облачных услуг.

Сначала установите kubectl, инструмент Kubernetes CLI. Для Mac kubectl можно установить с помощью команды brew:

```
brew install kubectl
```

Для Windows см. ресурсы, доступные по ссылке <https://oreil.ly/AhAwc>.

Для Linux:

```
curl -LO https://storage.googleapis.com/kubernetes-release/  
/release/v1.14.0/bin/linux/amd64/kubectl chmod +x ./kubectl  
sudo mv ./kubectl /usr/local/bin/kubectl
```

Чтобы установить Minikube, нам сначала нужно установить гипервизор, который создает и запускает виртуальные машины, такие как VirtualBox (для получения информации о VirtualBox см. <https://oreil.ly/LgFJ>).

На Mac Minikube можно установить с помощью команды brew:

```
brew install minikube
```

Для Windows см. ресурсы, доступные по ссылке <https://oreil.ly/awtxY>.

Для Linux выполните следующие действия:

```
curl -Lo minikube \ https://storage.googleapis.com/minikube/releases/latest/minikube-linux-  
amd64 chmod +x minikube  
sudo cp minikube /usr/local/bin && rm minikube
```

После завершения установки запустите простой кластер Kubernetes с помощью одной команды:

```
minikube start
```

Чтобы быстро проверить, подходит ли вам Minikube, мы можем попробовать вывести список узлов в кластере:

```
kubectl get nodes
```

Взаимодействие с Kubernetes CLI

Kubernetes API основан на ресурсах. Практически все в мире Kubernetes представлено как ресурс. kubectl построен с учетом этого, поэтому он будет следовать аналогичному шаблону для большинства взаимодействий с ресурсами.

Например, типичный вызов kubectl для вывода списка всех модулей (pods) выглядит следующим образом:

```
kubectl get pods
```

В результате выполнения этой команды должен быть выведен список всех работающих модулей (pods), но, поскольку мы еще не создали ни одного модуля, список будет пустым. Это не означает, что в нашем кластере в настоящее время не работают модули. Большинство ресурсов в Kubernetes можно разместить в определенном пространстве имен, и если вы не укажете конкретное пространство имен, они не будут выведены. Kubernetes запускает свои внутренние службы в пространстве имен, называемом kube-system. Чтобы вывести список всех модулей в любом пространстве имен, вы можете использовать параметр -n:

```
kubectl get pods -n kube-system
```

В результате выполнения этой команды должно быть возвращено несколько результирующих строк. Мы также можем использовать `--all-namespaces` для отображения всех модулей независимо от пространства имен.

Имя можно указывать для отображения только одного модуля:

```
kubectl get po mypod
```

Вы также можете применить фильтр по метке. Например, следующая команда должна вывести все модули, у которых есть компонент метки со значением `etcd` в `kube-system`:

```
kubectl get po -n kube-system -l component=etcd
```

Информацию, отображаемую `get`, также можно изменить. Например:

```
# Показать узлы и адреса модулей.
kubectl get po -n kube-system -o wide
# Показать определение pod mypod в yaml.
kubectl get po mypod -o yaml
```

Для создания нового ресурса `kubectl` предлагает две команды: `create` и `apply`. Разница между ними заключается в том, что `create` всегда будет пытаться создать новый ресурс (и потерпит неудачу, если он уже существует), тогда как `apply` либо создаст новый, либо обновит существующий ресурс.

Наиболее распространенный способ создания нового ресурса – использование файла YAML (или JSON) с определением ресурса, как мы увидим в следующем разделе. Следующие команды `kubectl` позволяют нам создавать обновляемые ресурсы Kubernetes (например, модули):

```
# Создать модуль, который определен в pod.yaml.
kubectl create -f pod.yaml
# В этой команде также можно указать HTTP ресурса.
kubectl create -f http://url-to-pod.yaml
# Apply позволяет вносить изменения в ресурсы.
kubectl apply -f pod.yaml
```

Чтобы удалить ресурс, используйте команду `kubectl delete`:

```
# Удалить модуль foo.
kubectl delete pod foo
# Удалить все ресурсы, определенные в pods.yaml.
kubectl delete -f pods.yaml
```

Вы можете использовать команду `kubectl edit` для быстрого обновления существующего ресурса. Откроется редактор, в котором вы можете редактировать загруженное определение ресурса:

```
kubectl edit pod foo
```

Определение ресурсов в Kubernetes

Ресурсы Kubernetes чаще всего определяются как YAML (хотя также можно использовать JSON). По сути, все ресурсы – это структуры данных с несколькими важными разделами.

apiVersion

Каждый ресурс является частью API, предоставляемого либо самим Kubernetes, либо третьими сторонами. `apiVersion` – номер версии API, который позволяет вам ориентироваться в том, насколько зрелую версию API вы используете.

kind

Тип ресурса (например, `pod`, `volume`, и т. д.).

metadata

Данные, необходимые для любого ресурса.

name

Ключ, по которому можно направлять запрос к каждому ресурсу; ключ должен быть уникальным.

labels

Каждый ресурс может иметь любое количество пар ключ-значение, называемых метками. Эти метки затем можно использовать в селекторах, для запроса ресурсов или просто как информацию.

annotations

Вторичные пары ключ-значение, которые носят чисто информационный характер и не могут использоваться в запросах или селекторах.

namespace

Метка, которая показывает, что ресурс принадлежит определенному пространству имен или команде.

spec

Конфигурация ресурса. Вся информация, необходимая для фактического выполнения, должна быть в спецификации. Каждая схема спецификации уникальна для определенного типа ресурса.

Вот пример файла `.yaml`, использующего эти определения:

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

В этом файле присутствуют `apiVersion` и `kind`, которые определяют, что это за ресурс. У нас есть метаданные, которые определяют имя и метку, и у нас есть спецификация, составляющая тело ресурса. Наш модуль состоит из одного контейнера, в котором выполняется команда `sh -c echo Hello Kubernetes! && sleep 3600` в образе `busybox`.

РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЙ В KUBERNETES

В этом разделе мы рассмотрим полное развертывание функционального Jupyter Notebook с помощью Minikube. Мы создадим постоянный том для наших блокнотов и создадим службу NodePort, которая позволит нам получить доступ к нашему блокноту.

Во-первых, нам нужно найти корректный образ Docker. *jupyter/tensorflow-notebook* – официальный образ, поддерживаемый сообществом Jupyter. Затем нам нужно будет выяснить, какой порт будет прослушивать наше приложение: в данном случае это 8888 (порт по умолчанию для Jupyter Notebooks).

Мы хотим, чтобы наш блокнот сохранялся между сеансами, поэтому нам нужно использовать постоянный том (`persistent volume claim, PVC`). Для этого мы создаем файл *pvc.yaml*:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: notebooks
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Теперь мы можем создать этот ресурс, выполнив команду

```
kubectl apply -f pvc.yaml
```

В результате ее выполнения должен быть создан том. Для подтверждения правильности выполнения команды мы можем перечислить все тома и PVC:

```
kubectl get pv
kubectl get pvc
kubectl describe pvc notebooks
```

Затем мы создаем наш файл развертывания *.yaml*. У нас будет один модуль (`pod`), который смонтирует наш том и откроет порт 8888:


```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: jupyter
  labels:
    app: jupyter
spec:
  selector:
    matchLabels:
      app: jupyter ❶
  template:
    metadata:
      labels:
        app: jupyter
    spec:
      containers:
        - image: jupyter/tensorflow-notebook ❷
          name: jupyter
          ports:
            - containerPort: 8888 name: jupyter
          volumeMounts:
            - name: notebooks
              mountPath: /home/jovyan
      volumes:
        - name: notebooks
          persistentVolumeClaim:
            claimName: notebooks

```

❶ Важно, чтобы этот селектор соответствовал ярлыкам в шаблоне

❷ Наш образ

Применяя этот ресурс (так же, как мы это делали с PVC), мы создадим модуль (pod) с экземпляром Jupyter:

```

# Посмотрим, готов ли наш развернутый экземпляр.
kubectl get deploy
# Список модулей, принадлежащих этому приложению.
kubectl get po -l app=jupyter

```

Когда наш модуль (pod) находится в состоянии Running, мы должны получить токен, с помощью которого мы сможем подключиться к нашему блокноту. Этот токен появится в журналах:

```

kubectl logs deploy/jupyter

```

Чтобы убедиться, что модуль работает, давайте обратимся к нашему блокноту, используя команду port-forward:

```

# Сначала нам нужно имя нашего модуля; он будет иметь рандомизированный суффикс.
kubectl get po -l app=jupyter
kubectl port-forward jupyter-84fd79f5f8-kb7dv 8888:8888

```

Таким образом, мы сможем получить доступ к блокноту по ссылке <http://localhost:8888>. Проблема в том, что никто другой не сможет это сделать,

поскольку трафик проксируется через наш локальный `kubectl`. Давайте создадим службу `NodePort`, чтобы получить доступ к блокноту:

```
apiVersion: v1
kind: Service
metadata:
  name: jupyter-service
  labels:
    app: jupyter
spec:
  ports:
    - port: 8888
      nodePort: 30888
  selector:
    app: jupyter
type: NodePort
```

Когда он будет создан, мы сможем получить доступ к нашему блокноту Jupyter Notebook. Но сначала нам нужно найти IP-адрес нашего модуля. Мы должны иметь доступ к Jupyter по следующему адресу и порту 30888:

```
minikube ip
# Эта команда покажет нам адрес нашего kubelet.
192.168.99.100:30888
```

Остальные пользователи теперь могут получить доступ к Jupyter Notebook, используя указанный IP-адрес и служебный порт (см. рис. А.2). Получив доступ к адресу в браузере, вы должны увидеть экземпляр Jupyter Notebook.

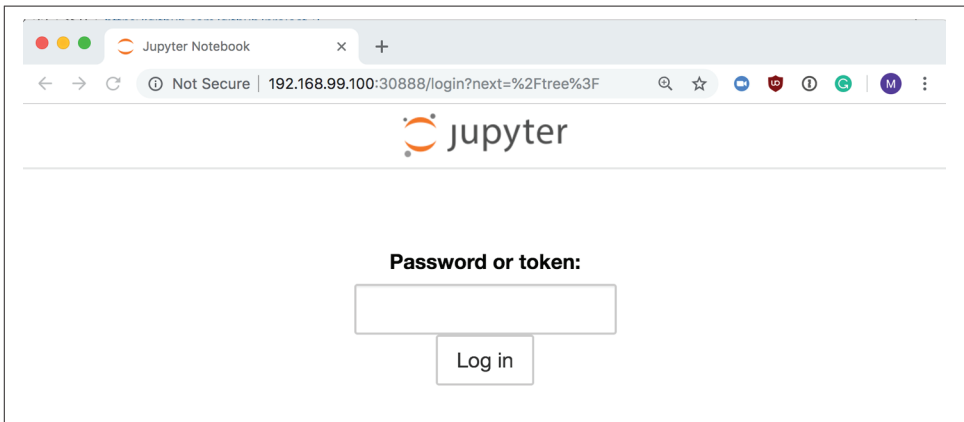


Рис. А.2. Блокнот Jupyter Notebook, работающий на Kubernetes

В этом приложении мы привели краткий обзор Kubernetes и его компонентов. Экосистема Kubernetes очень обширна, и краткое приложение не может дать целостного обзора. Для получения дополнительных сведений о базовой архитектуре Kubernetes Kubeflow мы настоятельно рекомендуем книгу «Работа с Kubernetes» Брендана Бернса и др., выпущенную в издательстве O'Reilly (Kubernetes: Up and Running by Brendan Burns et al., O'Reilly).

Приложение В

Настройка кластера Kubernetes в Google Cloud

В этом приложении представлен краткий обзор того, как создать кластер Kubernetes в Google Cloud, который запускает наш демонстрационный проект. Если вы новичок в Kubernetes, ознакомьтесь с приложением А и рекомендуемой литературой в конце главы 9. Хотя команды, которые мы рассмотрим, применимы только к Google Cloud, общий процесс установки аналогичен тому, который использовался для других управляемых сервисов Kubernetes, таких как AWS EKS или Microsoft Azure AKS.

ПРЕЖДЕ ЧЕМ ВЫ ПРИСТУПИТЕ К НАСТРОЙКЕ

Рекомендуя выполнить следующие шаги установки, мы предполагаем, что у вас есть учетная запись в Google Cloud. Если у вас нет учетной записи, вы можете ее создать. Кроме того, мы предполагаем, что вы установили Kubernetes `kubectl` (версия клиента 1.18.2 или выше) на свой локальный компьютер и что вы также можете запустить SDK Google Cloud `gcloud` (версия 289.0.0 или выше).



Следите за расходами на облачную инфраструктуру

Эксплуатация кластеров Kubernetes может повлечь значительные затраты на инфраструктуру. Поэтому мы настоятельно рекомендуем следить за расходами на инфраструктуру, задавая оповещения о выставлении счетов и бюджет. Подробности можно найти в документации Google Cloud (см. <https://oreil.ly/ubjAa>). Мы также рекомендуем отключать простаивающие вычислительные ресурсы из-за увеличения затрат, даже если они не используются и задачи конвейера не выполняются.

Инструкции по установке клиента `kubectl` для вашей операционной системы можно найти в документации Kubernetes (см. https://oreil.ly/syf_v). Документация Google Cloud (см. <https://oreil.ly/ZmhG5>) содержит пошаговые инструкции по установке клиента для вашей операционной системы.

KUBERNETES В GOOGLE CLOUD

В следующих пяти разделах мы проведем вас через пошаговый процесс создания кластера Kubernetes с нуля в Google Cloud.

Выбор проекта Google Cloud

Для кластера Kubernetes нам нужно создать новый проект Google Cloud или выбрать существующий проект на панели инструментов Google Cloud Project (см. <https://oreil.ly/LQS99>).

Обратите внимание на идентификатор проекта при выполнении следующих шагов. Мы развернем наш кластер в проекте с идентификатором `oreilly-book`, как показано на рис. В.1.

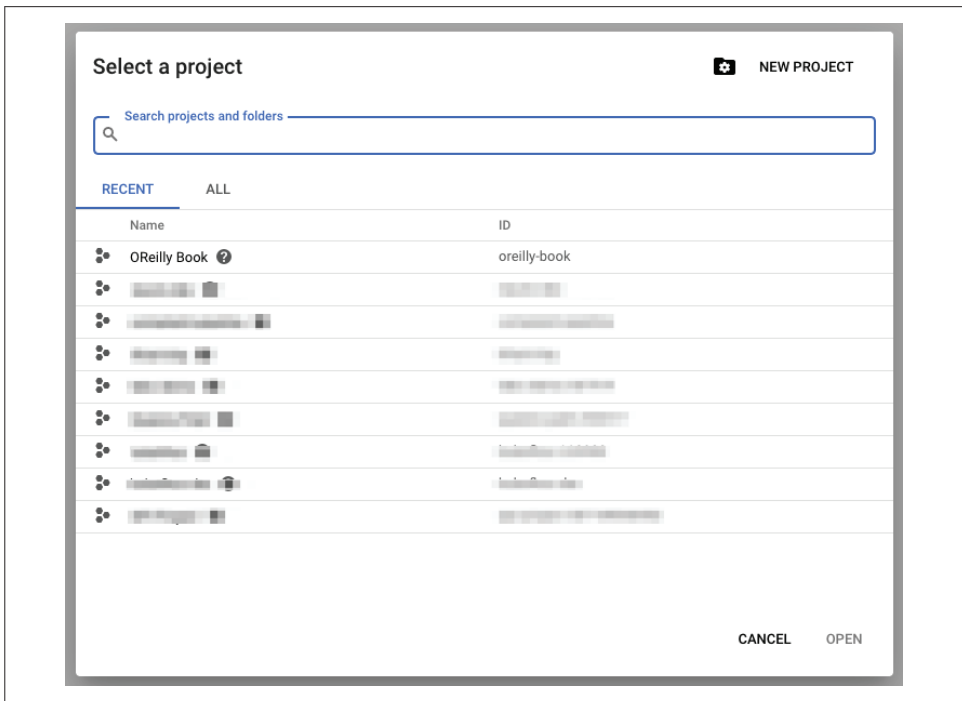


Рис. В.1. Панель управления Google Cloud Project

Настройка облачного проекта Google

Прежде чем создавать кластер Kubernetes, давайте настроим ваш проект Google Cloud. В терминале вашей операционной системы вы можете аутентифицировать своего клиента Google Cloud SDK с помощью следующей команды:

```
$ gcloud auth login
```

Затем обновите клиента SDK с помощью команды

```
$ gcloud components update
```

После успешной аутентификации и обновления клиента SDK давайте настроим несколько основных параметров. Сначала мы установим проект GCP в качестве проекта по умолчанию и выберем вычислительную зону в качестве зоны по умолчанию. В нашем примере мы выбрали `us-central-1`. Вы можете найти список всех доступных зон в документации Google Cloud (см. <https://oreil.ly/5beJg>). Выберите зону, ближайшую к вашему физическому местонахождению, или ту, где доступны необходимые сервисы Google Cloud (не все сервисы доступны во всех зонах).

Установив эти значения по умолчанию, нам не нужно указывать их позже в следующих командах. Мы также попросим включить API-интерфейсы контейнеров Google Cloud. Последний шаг необходимо выполнить только один раз для каждого проекта:

```
$ export PROJECT_ID=<your_gcp_project_id> ❶
$ export GCP_REGION=us-central1-c ❷
$ gcloud config set project $PROJECT_ID
$ gcloud config set compute/zone $GCP_REGION
$ gcloud services enable container.googleapis.com ❸
```

- ❶ Замените `your_gcp_project_id` на идентификатор проекта из предыдущего шага
- ❷ Выберите предпочитаемую зону или регион
- ❸ Включите API

Создание кластера Kubernetes

Теперь, когда наш проект Google Cloud готов к работе, мы можем создать кластер Kubernetes с несколькими вычислительными узлами как часть кластера. В нашем примере кластера под названием `kfp-oreilly-book` мы позволяем кластеру работать на количестве узлов от нуля до пяти в любой момент времени в нашем пуле с названием `kfp-pool`, и желаемое количество доступных узлов равно трем. Мы также назначаем сервисный аккаунт кластеру. Через учетную запись службы мы можем управлять разрешениями доступа для запросов от узлов кластера. Для получения дополнительной информации о сервисных аккаунтах в Google Cloud мы рекомендуем обратиться к онлайн-документации (см. <https://oreil.ly/7Ar4X>):

```
$ export CLUSTER_NAME=kfp-oreilly-book
$ export POOL_NAME=kfp-pool
$ export MAX_NODES=5
$ export NUM_NODES=3
$ export MIN_NODES=0
$ export SERVICE_ACCOUNT=service-account@oreilly-book.iam.gserviceaccount.com
```

Теперь, когда параметры кластера определены в переменной среды, мы можем выполнить следующую команду:

```
$ gcloud container clusters create $CLUSTER_NAME \
  --zone $GCP_REGION \
  --machine-type n1-standard-4 \
  --enable-autoscaling \
  --min-nodes=$MIN_NODES \
  --num-nodes=$NUM_NODES \
  --max-nodes=$MAX_NODES \
  --service-account=$SERVICE_ACCOUNT
```

Для нашего конвейера мы выбрали тип экземпляра `n1-standard-4`, который предоставляет 4 процессора и 15 ГБ памяти на один узел. Эти экземпляры предоставляют достаточно вычислительных ресурсов для обучения и оценки нашей модели машинного обучения и ее наборов данных. Вы можете найти полный список доступных типов экземпляров, выполнив следующую команду SDK:

```
$ gcloud compute machine-types list
```

Если вы хотите добавить графический процессор в кластер, вы можете указать тип графического процессора и количество графических процессоров, добавив аргумент ускорителя, как показано в следующем примере:

```
$ gcloud container clusters create $CLUSTER_NAME \
  ...
  --accelerator=type=nvidia-tesla-v100,count=1
```

Создание кластера Kubernetes может занять несколько минут, пока все ресурсы не будут полностью выделены вашему проекту и не станут доступными. Время зависит от запрашиваемых вами ресурсов и количества узлов. Для нашего кластера вам придется подождать около 5 минут, пока не будут доступны все ресурсы.

Доступ к кластеру Kubernetes с помощью kubectl

Когда ваш созданный кластер станет доступен, вы можете настроить `kubectl` для доступа к кластеру. SDK Google Cloud предоставляет команду для регистрации кластера в вашей локальной конфигурации `kubectl`:

```
$ gcloud container clusters get-credentials $CLUSTER_NAME --zone $GCP_REGION
```

После обновления конфигурации `kubectl` вы можете проверить, правильно ли выбран кластер, выполнив следующую команду:

```
$ kubectl config current-context
gke_oreilly-book-us-central1-c_kfp-oreilly-book
```

Использование кластера Kubernetes с kubectl

Поскольку ваш локальный `kubectl` может подключаться к вашему удаленному кластеру Kubernetes, все команды `kubectl`, такие как шаги Kubeflow Pipelines, упомянутые ниже и в главе 12, будут выполняться в удаленном кластере:

```
$ export PIPELINE_VERSION=0.5.0
$ kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/"\
    "cluster-scoped-resources?ref=$PIPELINE_VERSION"
$ kubectl wait --for condition=established \
    --timeout=60s crd/applications.app.k8s.io
$ kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/"\
    "env/dev?ref=$PIPELINE_VERSION"
```

НАСТРОЙКИ ПОСТОЯННОГО ТОМА ДЛЯ КОНВЕЙЕРОВ KUBEFLOW

В разделе «Обмен данными через постоянные тома» (см. приложение С) мы обсудим настройку постоянных томов в нашей настройке Kubeflow Pipelines. Полная конфигурация постоянного тома и заявка на выделение постоянного тома приведены в следующих фрагментах кода. Представленная настройка специфична для среды Google Cloud.

В примере В.1 показана конфигурация постоянного тома для нашего кластера Kubernetes:

Пример В.1. Конфигурация постоянного тома

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: tfx-pv
  namespace: kubeflow
  annotations:
    kubernetes.io/createdby: gce-pd-dynamic-provisioner
    pv.kubernetes.io/bound-by-controller: "yes"
    pv.kubernetes.io/provisioned-by: kubernetes.io/gce-pd
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 20Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: tfx-pvc
    namespace: kubeflow
  gcePersistentDisk:
    fsType: ext4
    pdName: tfx-pv-disk
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: failure-domain.beta.kubernetes.io/zone
              operator: In
              values:
                - us-central1-c
        - key: failure-domain.beta.kubernetes.io/region
              operator: In
              values:
                - us-central1
```

```
persistentVolumeReclaimPolicy: Delete
storageClassName: standard
volumeMode: Filesystem
status:
  phase: Bound
```

После создания постоянного тома мы можем запросить выделить часть или все доступное хранилище через запрос на постоянный том. Файл конфигурации можно увидеть в примере В.2.

Пример В.2. Конфигурация запроса на создание постоянного тома

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: tfx-pvc
  namespace: kubeflow
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
```

С помощью представленной конфигурации мы создали постоянный том и запрос на выделение постоянного тома в кластере Kubernetes. Теперь том можно смонтировать, как описано в разделе «Настройка конвейера» главы 12, или использовать, как описано в разделе «Обмен данными через постоянные тома» приложения С.

Приложение С

Советы по работе с Kubeflow Pipelines

Если вы управляете своими конвейерами TFX с помощью Kubeflow Pipelines, то можете настроить базовые образы контейнеров ваших компонентов TFX. Пользовательские образы TFX необходимы, если ваши компоненты полагаются на дополнительные зависимости Python вне пакетов TensorFlow и TFX. В случае нашего демонстрационного конвейера у нас есть дополнительная зависимость Python, библиотека TensorFlow Hub, для доступа к нашей языковой модели.

Во второй половине этого приложения мы хотим показать вам, как передавать данные на локальный компьютер и постоянный том и из него. Настройка постоянного тома будет полезна, если вы можете получить доступ к своим данным через поставщика облачного хранилища (например, с помощью локального кластера Kubernetes). Описанные далее шаги продемонстрируют процесс копирования данных в кластер и из него.

ПОЛЬЗОВАТЕЛЬСКИЕ ОБРАЗЫ TFX

В нашем демонстрационном проекте мы используем языковую модель, предоставленную TensorFlow Hub. Мы применяем библиотеку `tensorflow_hub` для эффективной загрузки языковой модели. Эта конкретная библиотека не является частью исходного образа TFX; поэтому нам нужно создать собственный образ TFX с необходимой библиотекой. Это также верно, если вы планируете использовать настраиваемые компоненты, подобные тем, которые мы обсуждали в главе 10.

К счастью, как мы обсуждали в приложении А, образы Docker можно создавать без особых проблем. Следующий файл *Dockerfile* демонстрирует пример настройки пользовательского образа:

```
FROM tensorflow/tfx:0.22.0
```

```
RUN python3.6 -m pip install "tensorflow-hub" ❶
```

```
RUN ... ❷
```

```
ENTRYPOINT ["python3.6", "/tfx-src/tfx/scripts/run_executor.py"] ❸
```

- ❶ Установите необходимые пакеты
- ❷ При необходимости установите дополнительные пакеты
- ❸ Не изменяйте точку входа в контейнер

Мы можем легко унаследовать стандартный образ TFX в качестве основы для нашего пользовательского образа. Чтобы избежать внезапных изменений в TFX API, мы настоятельно рекомендуем закрепить версию базового образа за конкретной сборкой (например, tensorflow/tfx:0.22.0), а не использовать общий последний тег. Образы TFX созданы на основе дистрибутива Ubuntu Linux и поставляются с предустановленным Python. В нашем случае мы можем просто установить дополнительный пакет Python для моделей Tensorflow Hub.

Очень важно использовать ту же точку входа, что и в базовом образе. Kubeflow Pipelines ожидает, что точка входа запустит исполнитель (executor) компонента.

После того как мы определили наш образ Docker, мы можем создать и отправить образ в реестр контейнера. Это может быть AWS Elastic, GCP или реестр контейнеров Azure. Важно убедиться, что работающий кластер Kubernetes может извлекать образы из реестра контейнеров и имеет необходимые права доступа для этого для частных контейнеров. В следующем примере кода мы демонстрируем эти шаги для реестра контейнеров GCP:

```
$ export TFX_VERSION=0.22.0
$ export PROJECT_ID=<your_gcp_project_id>
$ export IMAGE_NAME=ml-pipelines-tfx-custom

$ gcloud auth configure-docker
$ docker build pipelines/kubeflow_pipelines/tfx-docker-image/. \
  -t gcr.io/$PROJECT_ID/$IMAGE_NAME:$TFX_VERSION
$ docker push gcr.io/$PROJECT_ID/$IMAGE_NAME:$TFX_VERSION
```

После загрузки созданного образа вы можете увидеть образ, доступный в реестре контейнеров облачного провайдера, как показано на рис. С.1.



Образы для конкретных компонентов

На момент написания этой книги невозможно определить пользовательские образы для конкретных контейнеров компонентов. На данный момент требования ко всем компонентам должны быть включены в образ. Однако в настоящее время обсуждаются предложения, которые позволят в будущем использовать отдельные образы для конкретных компонентов.

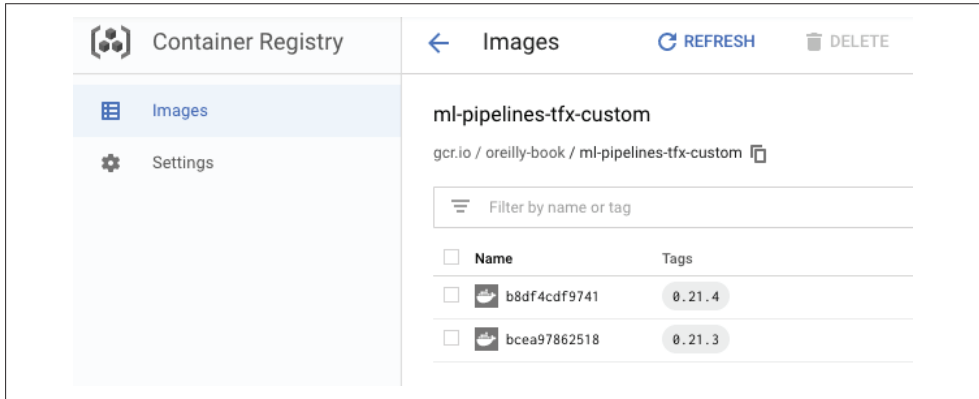


Рис. С.1. Реестр контейнеров Google Cloud

Теперь мы можем использовать этот образ контейнера для всех наших компонентов TFX в нашей настройке Kubeflow Pipelines.

ОБМЕН ДАННЫМИ ЧЕРЕЗ ПОСТОЯННЫЕ ТОМА

Как обсуждалось ранее, нам необходимо предоставить контейнеры для монтирования файловой системы для чтения и записи данных в места за пределами файловой системы контейнера. В мире Kubernetes мы можем монтировать файловые системы через постоянные тома (persistent volumes, PV) и запросы на постоянные тома (persistent volume claims, PVC). Проще говоря, мы можем предоставить диск, который будет доступен внутри кластера Kubernetes, а затем заявить права на эту файловую систему целиком или на часть ее пространства.

Вы можете настроить такие PV с помощью конфигураций Kubernetes, которые мы предоставляем в разделе «Настройки постоянного тома для конвейеров Kubeflow» приложения В. Если вы хотите использовать эту настройку, вам нужно будет создать диск у вашего облачного провайдера (например, AWS Elastic Block Storage или GCP Block Storage). В следующем примере мы создаем диск размером 20 ГБ с именем `tfx-pv-disk`:

```
$ export GCP_REGION=us-central1-c
$ gcloud compute disks create tfx-pv-disk --size=20Gi --zone=$GCP_REGION
```

Теперь мы можем подготовить диск для использования в качестве PV в нашем кластере Kubernetes. Следующая команда `kubectl` облегчит эту процедуру:

```
$ kubectl apply -f "https://github.com/Building-ML-Pipelines/"\
  "building-machine-learning-pipelines/blob/master/pipelines/"\
  "kubeflow_pipelines/kubeflow-config/storage.yaml"
$ kubectl apply -f "https://github.com/Building-ML-Pipelines/"\
  "building-machine-learning-pipelines/blob/master/pipelines/"\
  "kubeflow_pipelines/kubeflow-config/storage-claim.yaml"
```

После завершения подготовки мы можем проверить, сработало ли выполнение, выполнив команду `kubectl get pvc`, как показано в следующем примере:

```
$ kubectl -n kubeflow get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
tfx-pvc	Bound	tfx-pvc	20Gi	RWO	manual	2m

В Kubernetes `kubectl` имеется удобная команда `cp` для копирования данных с наших локальных машин на удаленный PV. Чтобы скопировать данные конвейера (например, модуль Python для шагов преобразования и обучения, а также обучающие данные), нам необходимо подключить том к модулю Kubernetes (pod). Для операций копирования мы создали простое приложение, которое в основном просто бездействует и позволяет нам получить доступ к PV. Вы можете создать модуль с помощью следующей команды `kubectl`:

```
$ kubectl apply -f "https://github.com/Building-ML-Pipelines/"\
  "building-machine-learning-pipelines/blob/master/pipelines/"\
  "kubeflow_pipelines/kubeflow-config/storage-access-pod.yaml"
```

Модуль `data-access` подключит PV, а затем мы сможем создать необходимые каталоги и скопировать нужные данные в том:

```
$ export DATA_POD=$(kubectl -n kubeflow get pods -o name | grep data-access)
$ kubectl -n kubeflow exec $DATA_POD -- mkdir /tfx-data/data
$ kubectl -n kubeflow exec $DATA_POD -- mkdir /tfx-data/components
$ kubectl -n kubeflow exec $DATA_POD -- mkdir /tfx-data/output
$ kubectl -n kubeflow cp \
  ../building-machine-learning-pipelines/components/module.py \
  ${DATA_POD#*/}:/tfx-data/components/module.py
$ kubectl -n kubeflow cp \
  ../building-machine-learning-pipelines/data/consumer_complaints.csv \
  ${DATA_POD#*/}:/tfx-data/data/consumer_complaints.csv
```

После того как все данные будут перенесены в PV, вы можете удалить модуль доступа к данным, выполнив следующую команду:

```
$ kubectl delete -f \
  pipelines/kubeflow_pipelines/kubeflow-config/storage-access-pod.yaml
```

Команда `cp` также работает для перемещения данных в другом направлении, и вы можете использовать ее, если хотите скопировать экспортированную модель из вашего кластера Kubernetes в другое место за пределами вашего кластера.

ИНТЕРФЕЙС КОМАНДНОЙ СТРОКИ TFX

TFX предоставляет интерфейс командной строки для управления вашими проектами TFX и их оркестрацией. Инструмент CLI предоставляет вам шаблоны TFX, предопределенную структуру каталогов и файлов. Проектами, использующими предоставленную структуру папок, можно будет управлять с помощью инструмента CLI, замещающего веб-интерфейс (для Kubeflow и Airflow).

Он также включает библиотеку Skaffold для автоматизации создания и публикации пользовательских образов TFX.



Интерфейс командной строки TFX находится в стадии активной разработки

На момент написания этого раздела интерфейс командной строки TFX находится в активной разработке. Команды могут измениться, а также могут быть добавлены дополнительные функции. Кроме того, в будущем могут появиться дополнительные шаблоны TFX.

TFX и его зависимости

Для работы TFX CLI требуется SDK Kubeflow Pipelines и Skaffold, инструмент Python для непрерывного создания и развертывания приложений Kubernetes.

Если вы не установили или не обновили TFX и Python SDK из Kubeflow Pipelines, выполните две команды установки `pip`:

```
$ pip install -U tfx
$ pip install -U kfp
```

Установка Skaffold зависит от вашей операционной системы:

Linux:

```
$ curl -Lo skaffold \
https://storage.googleapis.com/\
skaffold/releases/latest/skaffold-linux-amd64
$ sudo install skaffold /usr/local/bin/
```

MacOS:

```
$ brew install skaffold
```

Windows:

```
$ choco install -y skaffold
```

После установки Skaffold убедитесь, что путь к исполняемому файлу инструмента добавлен в `PATH` среды терминала, в которой вы запускаете инструмент TFX CLI. В следующем примере `bash` показано, как пользователи Linux могут добавить путь Skaffold к своей переменной `bash PATH`:

```
$ export PATH=$PATH:/usr/local/bin/
```

Прежде чем мы обсудим, как использовать инструмент TFX CLI, давайте кратко обсудим шаблоны TFX.

Шаблоны TFX

TFX предоставляет шаблоны проектов для организации проектов создания конвейеров машинного обучения. Шаблоны предоставляют предопределенную структуру папок и план для вашей функции, модель и определения предварительной обработки. Следующая команда копирования шаблона `tfx template copy` загрузит демонстрационный пример *taxi cab* проекта TFX:

```
$ export PIPELINE_NAME="customer_complaint"
$ export PROJECT_DIR=$PWD/$PIPELINE_NAME
$ tfx template copy --pipeline-name=$PIPELINE_NAME \
                  --destination-path=$PROJECT_DIR \
                  --model=taxi
```

Когда команда копирования завершит свое выполнение, вы сможете найти структуру папок, как показано в следующих выходных данных bash:

```
$ tree .
.
├── init .py
├── beam_dag_runner.py
├── data
│   └── data.csv
├── data_validation.ipynb
├── kubeflow_dag_runner.py
├── model_analysis.ipynb
├── models
│   ├── init .py
│   ├── features.py
│   ├── features_test.py
│   ├── keras
│   │   ├── init .py
│   │   ├── constants.py
│   │   ├── model.py
│   │   └── model_test.py
│   ├── preprocessing.py
│   └── preprocessing_test.py
├── pipeline
│   ├── init .py
│   ├── configs.py
│   └── pipeline.py
└── template_pipeline_test.tar.gz
```

Мы использовали шаблон `taxi cab`¹ и настроили наш демонстрационный проект в соответствии с этим шаблоном. Результаты можно найти в репозитории на GitHub, созданном для демонстрационных примеров этой книги (см. <https://oreil.ly/bmlp-git>). Если вы хотите использовать этот пример, скопируйте файл CSV `consumer_complaints.csv` в указанный каталог:

```
$pwd/$PIPELINE_NAME/data
```

Кроме того, дважды проверьте файл `pipelines/config.py`, который определяет сегмент GCS и другие параметры конфигурации конвейера. Обновите путь к сегменту GCS, указав сегмент, который вы создали, или используйте сегменты GCS, которые были созданы при создании установки Kubeflow Pipelines на платформе AI GCP. Вы можете узнать путь с помощью следующей команды:

```
$ gsutil -l
```

¹ На момент написания книги это был единственный доступный шаблон.

Публикация конвейера с помощью TFX CLI

Мы можем опубликовать конвейер TFX, который создали на основе шаблона TFX, в нашем приложении Kubeflow Pipelines. Чтобы получить доступ к настройке Kubeflow Pipelines, нам нужно указать наш проект GCP, путь к нашему образу контейнера TFX и URL-адрес конечной точки Kubeflow Pipelines. В разделе «Доступ к установленному экземпляру Kubeflow Pipelines» главы 12 мы обсудили, как получить URL-адрес конечной точки. Перед тем как опубликовать наш конвейер с помощью TFX CLI, давайте настроим необходимые переменные среды для нашего примера:

```
$ export PIPELINE_NAME="<pipeline name>"
$ export PROJECT_ID="<your gcp project id>"
$ export CUSTOM_TFX_IMAGE=gcr.io/$PROJECT_ID/tfx-pipeline
$ export ENDPOINT="<id>-dot-<region>.pipelines.googleusercontent.com"
```

Теперь, определив эти параметры, мы можем создать конвейер через TFX CLI с помощью следующей команды:

```
$ tfx pipeline create --pipeline-path=kubeflow_dag_runner.py \
  --endpoint=$ENDPOINT \
  --build-target-image=$CUSTOM_TFX_IMAGE
```

Команда `tfx pipeline create` выполняет множество задач. С помощью Scaffold она создает образ Docker, используемый по умолчанию, и публикует образ контейнера в Google Cloud Registry. Он также запускает Kubeflow Runner, как мы обсуждали в главе 12, и загружает конфигурацию Argo в конечную точку конвейера. После того как команда завершит выполнение, вы найдете два новых файла в структуре каталогов шаблона: *Dockerfile* и *build.yaml*.

Dockerfile содержит определение образа, аналогичное *Dockerfile*, которое мы обсуждали в разделе «Пользовательские образы TFX» приложения С. Файл *build.yaml* настраивает Scaffold и задает необходимые параметры для реестра образов Docker и политику тегов.

Теперь вы сможете увидеть зарегистрированный конвейер в пользовательском интерфейсе Kubeflow Pipelines. Вы можете запустить конвейер с помощью следующей команды:

```
$ tfx run create --pipeline-name=$PIPELINE_NAME \
  --endpoint=$ENDPOINT
```

```
Creating a run for pipeline: customer_complaint_tfx Detected Kubeflow.
Use --engine flag if you intend to use a different orchestrator. Run created for pipeline:
customer_complaint_tfx
```

```
+-----+-----+-----+-----+
| pipeline_name | run_id | status | created_at |
+-----+-----+-----+-----+
| customer_complaint_tfx | <run-id> | | 2020-05-31T21:30:03+00:00 |
+-----+-----+-----+-----+
```

Вы можете проверить состояние запуска конвейера с помощью такой команды:

```
$ tfx run status --pipeline-name=$PIPELINE_NAME \
--endpoint=$ENDPOINT \
--run_id <run_id>
```

Listing all runs of pipeline: customer_complaint_tfx

```
+-----+-----+-----+-----+
| pipeline_name | run_id | status | created_at |
+=====+=====+=====+=====+
| customer_complaint_tfx | <run-id> | Running | 2020-05-31T21:30:03+00:00 |
+-----+-----+-----+-----+
```



Останов и удаление запусков конвейера

Вы можете остановить запуск конвейера с помощью команды `tfx run terminate`. Запуск конвейера можно удалить с помощью команды `tfx run delete`.

TFX CLI – очень полезный инструмент в наборе инструментов TFX. Он поддерживает не только конвейеры Kubeflow, но и оркестраторы Apache Airflow и Apache Beam.

Предметный указатель

Г

граф модели машинного обучения 155

Д

данные

входные

распределение данных 279

обучающие 279

шифрование 302

ключ шифрования 302

З

зашифрованное машинное обучение

Keras 304

зашифрованное машинное

обучение 302

зашифрованное обучение

модели 304

комбинирование

с дифференцированной
приватностью 303

методы

безопасное многостороннее

вычисление 302

гомоморфное шифрование 302

преобразование модели, обученной

на незашифрованных данных 304

зашифрованное обучение модели 303

К

конвейер машинного обучения

будущие возможности 309

дальнейшие шаги 307

запуск

с помощью Apache Airflow 244

инструменты оркестрации 246

настройка

с помощью Apache Airflow 242

производственный 232

компоненты 232

участие человека в работе

конвейера 212

взаимодействие со Slack API 214

использование компонента

Slack 214

настройка компонента Slack 214

конвейер TFX

настройка 273

загрузка данных из Google Cloud

Storage 273

обучение моделей

на платформе Google Cloud AI

Platform 273

оркестрация

Kubeflow Pipelines 252

развертывание моделей

на платформе Google Cloud AI

Platform 275

конечная точка API 150, 151

Google Remote Procedure Calls

(gRPC). См. gRPC; gRPC

Representational State Transfer

(REST). См. REST

контейнер 313

машинное обучение

инфраструктура 313

платформы

Amazon Web Service SageMaker 181

Microsoft AzureML 181

системы непрерывной интеграции

и развертывания 311, 313

М

модель машинного обучения 149, 190

версии 165

вес 150

граф модели машинного обучения 155

дрейф 279

запросы метаданных 173

модель классификации 171

обучение приватной модели 298

обучение с «теплым» запуском 212

одновременное обучение нескольких

моделей 208

пакетные запросы 175
 персонализированная 279
 политика релизов 308
 получение прогнозов 167
 использование протокола
 REST 167
 прогностическая значимость 279
 развертывание 149, 150, 190
 «горячая замена» 163
 использование удаленных
 хранилищ 191, 193
 на периферийных устройствах 200
 недостатки способа развертывания
 с использованием Python 151, 152
 обзор 191
 оптимизация для удаленной
 модели. См. оптимизация
 моделей машинного обучения
 для развертывания;
 оптимизация моделей
 машинного обучения для
 развертывания
 расширенные концепции 190
 с использованием TensorFlow
 Serving. См. TensorFlow Serving
 (TFS);
 создание веб-приложения с
 использованием Python 150, 151
 развертывание на мобильных
 платформах 209
 развертывание с использованием
 «облачных» решений 182
 Google Clouds AI Platform 182, 183
 размещение у провайдеров
 «облачных» решений 182
 регрессионная модель 171

Н

направленный ациклический
 граф (НАГ) 237

О

обратная связь 288
 бинарная 288
 переклассификация 288
 обучение нейронных сетей
 обучаемая сеть 196
 обучающая сеть 196
 оптимизация моделей машинного
 обучения для развертывания 193, 194

дистилляция 196
 для загрузки удаленной модели 193
 инструменты 197
 квантование 194
 использование библиотеки
 TensorRT. См. TensorRT;
 сокращение 195
 TFLite. См. TFLite

П

петли обратной связи 279
 маховик данных 281
 отслеживание обратной связи 287, 288
 для случая неявной обратной
 связи 289
 для случая явной обратной
 связи 288
 примеры 282, 283, 284, 285
 шаблоны для сбора обратной
 связи 284, 285, 286, 287
 явная и неявная обратная связь 280
 платформа искусственного
 интеллекта 232
 Google AI Platform 232
 поставщики облачных услуг 182
 Amazon Web Services (AWS) 182
 Google Cloud 182
 Microsoft Azure 182
 приватность данных 290, 291
 бюджет приватности 295
 данные, идентифицирующие
 личность 292
 дифференцированная
 приватность 290, 293, 294
 глобальная 294, 296
 доверенные стороны 296
 локальная 294
 методы обеспечения 296
 объект secret 295
 оптимизация 297
 расчет параметров модели 299
 стохастический градиентный
 спуск 297
 эпсилон-дельта 295
 зашифрованное машинное
 обучение 290
 квазиидентифицирующие данные 292
 методы обеспечения
 приватности 305
 К-анонимность 305

способы повышения
 приватности 292
 федеративное обучение 290
 чувствительные данные 292

С

сервер моделей 150, 182
 сигнатура 155
 методы 155
 структура 150
 тестирование 159
 управляемое развертывание
 в «облаке» 182
 ограничения 182
 преимущества и недостатки 182
 эксперименты с моделью
 отслеживания 307
 экспорт 154
 граф модели машинного
 обучения 155
 для TFS 153
 контрольная точка 155
 проверка экспортированных
 моделей 157
 ресурсы 155
 A/B-тестирование 172
 простой пример реализации 151

Ф

федеративное обучение 299, 300
 алгоритмы 300, 301
 включение дифференцированной
 приватности 300
 доверенные стороны 300
 инфраструктура 301
 используемые сценарии 300
 особенности 301
 советы по применению 301

А

A/B-тестирование 190
 Apache Airflow 43, 230, 231, 234, 237, 246
 варианты рабочей
 конфигурации 239
 зависимости задач 240
 конфигурация 239
 конфигурация
 на примере конкретного
 проекта 239
 операторы задач 240

пример применения 239
 установка и настройка 237
 Apache Avro 53
 Apache Beam 46, 48, 68, 86, 230, 231,
 234, 235, 243, 246, 269
 Apache Parquet 53, 64
 API
 конечная точка. См. конечная API
 Argo 253, 258, 277
 AWS Simple Storage Service (S3) 57

D

Dataflow 269
 DataFlow 84
 DevOps 190
 DirectRunner 50
 Docker 160, 162, 190, 204, 313
 интерфейс командной строки 317
 контейнер 162
 конфигурация при работе
 с несколькими моделями
 машинного обучения 164
 конфигурация при работе с одной
 моделью машинного обучения 162
 образ 160, 314
 создание 316
 поддержка графических процессоров
 160
 установка 160

G

Google AI Platform Pipelines 269
 Google Cloud 84, 269, 271
 Google Cloud AI Platform 269, 276
 использование в конвейерах
 машинного обучения 269
 Google Cloud BigQuery 51, 59
 Google Cloud DataFlow 80
 Google Cloud Google CloudFfs AI
 Platform 112
 Google Cloud Marketplace 270
 Google Cloud Platform 47, 80
 Google Clouds AI Platform
 получение прогноза модели
 машинного обучения 187
 развертывание модели машинного
 обучения 183
 использование GCP API 186
 масштабирование 185
 работа с развернутой моделью 186

Google Cloud Storage 57, 273

gRPC 166

Н

H2O 181

Ж

Jupyter Notebook 43

Jupyter Notebook 53, 69, 71, 73, 74, 127

К

Keras 154, 190

Kubeflow 248, 250, 271

архитектура 250

Kubeflow Lineage Explorer 268

Kubeflow Pipelines 231, 246, 269, 332

запуск конвейера машинного обучения 258

использование на платформах искусственного интеллекта Google AI 232

использование постоянного тома 330, 334

настройка 246, 249, 269

настройка кластера 271

оркестрация конвейеров TFX 252

полезные функции 264

аудит линии конвейера 268

запуск конвейера

по расписанию 264

перезапуск конвейера 264

проверка запусков конвейера 265

пользовательские образы TFX 332

установка 249

использование кластера

Kubernetes 249

Kubernetes 204, 232, 249, 318

инструменты

kubectl 319

Minikube 319

кластер 319

определение ресурсов 322

основные определения 318

работа с Google Cloud 326

развертывание приложений 323

создание кластера

в Google Cloud 328

управляемые сервисы 249

Н

Nvidia 196

О

Open Neural Network

Exchange (ONNX) 181

Р

PMML 181

Prometheus 200

настройка 202

установка 200

PyTorch 181

Р

REST 162, 166

С

Scikit-learn 181

SciKit-Learn 185

Т

TensorBoard 113

TensorFlow 52, 190, 290

развертывание графа 152

TensorFlow Data

Validation 46, 71, 73, 76, 80

Tensor Flow Data Validation (TFDV) 66

TensorFlow Data Validation (TFDV) 68

TensorFlow Debugger 160

TensorFlow-Encrypted 303

TensorFlow Extended 46, 51

TensorFlow Extended (TFX) 38, 41, 207

зависимости 336

интерфейс командной строки 335

использование с другими фреймворками машинного обучения 310

пользовательские компоненты 215

драйверы 222

запуск 224

исполнители 219

каналы 218

расширение существующих компонентов 225

сборка 223

собственная реализация

исполнителей компонентов 228

создание с нуля 217

спецификации 217

сценарии использования 216

публикация конвейера машинного обучения 338

расширенные концепции 207

шаблоны 336

- TensorFlow Federated 301
 - TensorFlow Model
 - Analysis (TFMA) 121, 125
 - TensorFlow Privacy 296
 - оптимизатор, использующий дифференцированную приватность 297
 - стохастический градиентный спуск 297
 - расчет параметра ϵ 298
 - TensorFlow Serving 175
 - TensorFlow Serving (TFS) 150, 151, 152, 153, 190, 196, 197, 204, 209
 - альтернативы 180
 - BentoML 180
 - GraphPipe 181
 - Mlflow 181
 - Ray Serve 181
 - Seldon 180
 - Simple TensorFlow Serving 181
 - архитектура 153
 - использование удаленных хранилищ данных 191
 - AWS 192
 - GCP 193
 - масштабирование 204
 - мониторинг 200
 - метрики 200
 - Prometheus. См. Prometheus; См. Prometheus
 - настройка 161
 - конфигурации при работе с несколькими моделями машинного обучения 164
 - конфигурации при работе с одной моделью машинного обучения 163
 - конфигурация при работе с одной моделью машинного обучения 162
 - оптимизация 177, 179
 - пакетные запросы
 - настройка пакетного режима в прогнозировании 177
 - пакетные запросы на вывод прогнозов модели 175
 - поддержка gRPC 162
 - поддержка REST 162
 - протоколы обмена данными 166
 - сравнительные характеристики 167
 - REST. См. REST;
 - работа с выбранными версиями модели 165
 - сборка из исходного кода 161
 - установка 160
 - установка на Ubuntu 160
 - установка с использованием Docker. См. Docker;
 - TFS Python API 157
 - TensorFlow Transform 46
 - TensorFlow Transform (TFT) 85, 87, 89
 - TensorRT 196
 - TFLite 197, 198, 209, 210
 - инструменты оптимизации модели машинного обучения 197
 - ограничения 210
 - поддержка операций TensorFlow 198
 - развертывание моделей машинного обучения с помощью TensorFlow Serving 199
 - экспорт моделей 209
 - TFRecords 55, 56
 - TFX Evaluator 146, 147
 - TFX Pusher 148
 - TFX Resolver
 - ResolverNode 145
 - TFX Trainer 106
 - TFX Tuner 119
- U**
- Ubuntu 160, 161
 - Universal Sentence Encoder (USE) 104
- X**
- XGBoost 181, 185

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planet.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@aliants-kniga.ru**.

Ханнес Хапке
Кэтрин Нельсон

Разработка конвейеров машинного обучения

**Автоматизация жизненных циклов модели
с помощью TensorFlow**

Главный редактор	<i>Мовчан Д. А.</i> dmkpress@gmail.com
Зам. главного редактора	<i>Сенченкова Е. А.</i>
Перевод	<i>Желнова Н. Б.</i>
Корректор	<i>Синяева Г. И.</i>
Верстка	<i>Луценко С. В.</i>
Дизайн обложки	<i>Мовчан А. Г.</i>

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать цифровая.
Усл. печ. л. 28,11. Тираж 200 экз.

Веб-сайт издательства: www.dmkpress.com

В настоящее время компании тратят миллиарды долларов на проекты машинного обучения (МО), но эти средства могут быть потрачены впустую, если при этом не используется эффективное развертывание моделей МО. Перед вами практическое руководство, написанное Х. Хапке и К. Нельсон, с которым вы смело можете отправляться в путь по всем этапам автоматизации конвейера МО, построенного на основе экосистемы TensorFlow. Вы познакомитесь с методами и инструментами, которые существенно сократят время развертывания (с нескольких дней до нескольких минут), чтобы вы могли сосредоточиться на разработке новых моделей, а не на поддержке устаревших систем. Специалисты по анализу данных, инженеры по МО и инженеры DevOps узнают, как выйти за рамки простой разработки моделей и успешно реализовать свои проекты в области науки о данных, а менеджеры лучше поймут, как можно существенно сократить сроки, необходимые для реализации подобных проектов.

«Эта книга — выдающийся ресурс и исчерпывающий обзор систем машинного обучения в целом и TFX в частности. Она содержит самую точную информацию, сопровождаемую ясными краткими пояснениями и примерами».

*Роберт Кроу,
популяризатор
современных
практик разработки
TensorFlow, Google*

С помощью этой книги вы:

- поймете, как построить конвейер машинного обучения;
- создадите свой конвейер, используя компоненты TensorFlow Extended;
- сможете управлять конвейером машинного обучения с помощью Apache Beam, Apache Airflow и Kubeflow Pipelines;
- освоите работу с данными с помощью TensorFlow Data Validation и TensorFlow Transform;
- проведете анализ модели с помощью TensorFlow Model Analysis;
- определите такие параметры своей модели, как точность предсказаний и смещение;
- сможете выполнить развертывание модели с помощью TensorFlow Serving или TensorFlow Lite для мобильных устройств;
- познакомитесь с методами, обеспечивающими конфиденциальность данных в машинном обучении.

Ханнес Хапке и Кэтрин Нельсон — старшие специалисты по анализу данных в Concur Labs (одно из подразделений SAP Concur), где они исследуют инновационные способы использования машинного обучения для повышения качества обслуживания путешественников, совершающих деловые поездки. Область интересов Ханнес — программная инженерия в области машинного обучения (развертывание масштабируемых моделей) и задачи обработки естественного языка, а Кэтрин специализируется на анализе моделей и сохранении конфиденциальности данных в приложениях МО.

Интернет-магазин: www.dmkpress.com

Оптовая продажа: КТК «Галактика»
books@alians-kniga.ru



ISBN 978-5-97060-886-9

