Джефф Хултен

Разработка интеллектуальных систем

Введение в технологию машинного обучения

ДМК Пресс
издательство
Москва, 2019

Geoff Hulten

Building Intelligent Systems

A Guide to Machine Learning Engineering

Apress®

Джефф Хултен

Разработка интеллектуальных систем

УДК 004.89
ББК 32.813
Х98

Хултен Дж.
Х98 Разработка интеллектуальных систем / пер. с анг. В. С. Яценкова. – М.: ДМК Пресс, 2019. – 284 с.: ил.

ISBN 978-5-97060-760-2

Эта книга научит вас, как создавать интеллектуальные системы от начала до конца и использовать машинное обучение на практике. Вы узнаете, как эффективно применять свои навыки разработки программного обеспечения, науки о данных, машинного обучения и управления проектами.

Книга основана на более чем десятилетнем опыте создания интеллектуальных систем, которые обеспечивают сотни миллионов взаимодействий пользователей в день в некоторых из крупнейших и наиболее важных программных систем в мире.

Издание будет полезно инженерам-программистам, специалистам по машинному обучению и руководителям проектов, которые хотят создавать и внедрять эффективные интеллектуальные системы.

УДК 004.89
ББК 32.813

ISBN 978-1-4842-3431-0 (анг.)
ISBN 978-5-97060-760-2 (рус.)

Папе, который говорил мне то, что мне следовало услышать.
Маме, которая говорила мне лишь то, что я хотел услышать.
И конечно тебе, Николь!

Содержание

Автор

Джефф Хултен (Geoff Hulten) – учёный, кандидат наук в области машинного обучения. Более десяти лет он руководил рабочими группами по прикладному машинному обучению, создавая десятки интеллектуальных систем в масштабе интернета, в которых происходят сотни миллионов взаимодействий с пользователями каждый день. Его исследования были представлены на ведущих международных конференциях, получили тысячи ссылок, выиграли награду **SIGKDD Test of Time** за значительный вклад в сообщество исследователей интеллектуального анализа данных и выдержали испытание временем.

Технический рецензент

Джеб Хейбер (Jeb Haber) имеет степень бакалавра информатики Университета Уилламетт (Willamette University). Он провел почти два десятилетия в Microsoft, работая над различными проектами в Windows, Internet Explorer, Office и MSN. За последние десять с лишним лет своей карьеры в Microsoft Джеб возглавлял команду управления проектами, отвечающую за службы безопасности и защиты, предоставляемые Microsoft SmartScreen (защита от фишинга, вредоносных программ и прочих киберугроз).

Команда Джеба разработала и управляла интеллектуальными системами глобального масштаба с сотнями миллионов пользователей. Его работа включала в себя разработку идеологии продукта, планирование, стратегию, управление проектами, определение метрик и развитие команды. Наряду с разработкой систем и процессов, необходимых для построения и запуска в глобальном масштабе интеллектуальных и оценочных систем в режиме 24×7, Джеб помог сформировать культуру целой отрасли. В настоящее время Джеб является президентом наблюдательных советов в двух некоммерческих организациях, помогающих отдельным лицам и семьям, страдающим редким генетическим расстройством – фенилкетонурией (ФКУ).

Благодарности

Меня окружает множество людей, ставших частью мира интеллектуальных систем, над которыми я работал на протяжении многих лет. Эти люди помогли мне многому научиться и многое понять. В частности, я хотел бы поблагодарить:

Джеба Хейбера и Джона Скарроу (John Scarrow) за то, что они были в числе ключевых умов при разработке концепций, описанных в этой книге, и за то, что они были отличными коллегами на протяжении многих лет. Ничего из этого не произошло бы без их руководства и преданности делу.

Благодарю вас, Энтони П. (Anthony P.), Томаш К. (Tomasz K.), Роб С. (Rob S.), Роб М. (Rob M.), Дейв Д. (Dave D.), Кайл К. (Kyle K.), Эрик Р. (Eric R.), Амейя Б. (Ameya B.), Крис И. (Kris I.), Джефф М. (Jeff M.), Майк С. (Mike C.), Шанкар С. (Shankar S.), Роберт Р. (Robert R.), Крис Дж. (Chris J.), Сьюзен Х. (Susan H.), Иван О. (Ivan O.), Чад М. (Chad M.), и многих других, кто не вошел в этот список.

Вступление

«Разработка интеллектуальных систем» – это книга о том, как использовать машинное обучение на практике.

Она охватывает все, что вам нужно для создания полностью функционирующей интеллектуальной системы, которая использует машинное обучение и данные взаимодействия с пользователем для непрерывного улучшения и достижения поставленной цели.

Прочитав эту книгу, вы сможете выполнить разработку интеллектуальной системы от начала и до конца.

Вы будете знать:

- когда следует использовать интеллектуальную систему и как сделать так, чтобы она достигла ваших целей;
- как разработать эффективные взаимодействия между пользователями и интеллектуальными системами;
- как внедрить интеллектуальную систему на стороне клиента и на сервере;
- как построить интеллект, который является сердцем интеллектуальной системы, и развивать его с течением времени;
- как управлять интеллектуальной системой в течение ее жизненного цикла.

Вы также поймете, как с наибольшей отдачей приложить свои усилия в области разработки программного обеспечения, обработки данных, машинного обучения и управления проектами.

Существует множество замечательных книг, которые учат работе с данными и навыкам машинного обучения. Эти книги похожи на книги по языкам программирования – они преподносят ценные навыки в мельчайших деталях. Но данная книга больше похожа на книгу по разработке программного обеспечения – она учит, как использовать эти базовые навыки и создавать прикладные системы.

Эта книга основана на более чем десятилетнем опыте создания интеллектуальных систем в масштабе интернета, где каждый день происходят сотни миллионов взаимодействий пользователей с крупнейшими и наиболее важными вычислительными системами в мире. Я надеюсь, что книга поможет ускорить распространение систем, которые превращают данные в результаты, и поможет читателям развить практические навыки в этой важной области.

Для кого эта книга

Эта книга предназначена для всех, кто имеет образование в области информатики и хочет разобраться, что нужно для создания эффективных интеллектуальных систем.

Представьте себе типичного инженера-программиста, который назначен на проект машинного обучения. Он хочет узнать больше о новом направлении, поэтому читает технические книги, полные статистики, математики и описаний методов моделирования. Это важные знания, но они плохо помогают разработчику программного обеспечения включиться в новую работу. Правильнее будет начать с книги «Разработка интеллектуальных систем».

Представьте себе специалиста по машинному обучению, который должен понимать, как законченная система будет взаимодействовать с его моделями, на что он может рассчитывать и на что следует обращать внимание на практике. «Разработка интеллектуальных систем» написана для него.

Представьте себе технического директора, который хочет успешно внедрить машинное обучение на своем предприятии. Возможно, он наймет доктора наук в области машинного обучения. Спустя некоторое время специалист по машинному обучению принесет директору различные диаграммы, кривые точности/отклика и выборки обучающих данных, но не добавит понимания того, как это все внедрить на практике. «Разработка интеллектуальных систем» – самая подходящая книга для технического директора.

Специалисты по данным и машинному обучению

Данные и машинное обучение лежат в основе многих интеллектуальных систем, но между разработкой рабочей модели (созданной с помощью машинного обучения) и возможным устойчивым влиянием на пользователя предстоит проделать невероятную работу. Понимание особенностей этой вспомогательной работы открывает перед вами несколько путей для улучшения моделей.

Во-первых, важно **понимать ограничения**, которые эти системы накладывают на ваши модели. Например, где будет работать модель? К каким данным она будет иметь доступ? Как быстро это будет происходить? Каково влияние ложноположительных решений на бизнес? А ложноотрицательных? Как настроить модель, чтобы добиться максимальной отдачи для бизнеса?

Во-вторых, важно иметь возможность **влиять на других участников проекта**. Способность влиять на инженеров и владельцев бизнеса поможет вам найти лучшие решения и максимально увеличить собственные шансы на успех. Например, вы не можете получать достаточный объем обучающих данных из-за нехватки данных телеметрии. Должны ли вы удвоить усилия по доработке модели, или более разумным будет инженерное решение проблемы? Или, может быть, от вас требуют невероятно высокой точности, хотя ваши модели уже работают с очень хорошей (но немного меньшей) точностью? Стоит ли бороться за сверхвысокую точность или лучше подумать над тем, как изменить пользовательский опыт таким образом, чтобы уменьшить влияние ошибок на пользователя?

В-третьих, важно понять, **какие вспомогательные элементы могут принести вам пользу**. Эскалации, ручные перезагрузки, телеметрия, ограниче-

ния, которые предотвращают серьезные ошибки, – все это инструменты, которые вы можете использовать. Вы должны понимать, когда их использовать и как интегрировать их в процесс моделирования. Стоит ли отказываться от модели, которая работает приемлемо для 99 % пользователей, но чрезвычайно плохо для 1 % пользователей? Или, может быть, следует искать решение проблемы в других частях системы?

Инженеры-программисты

Создание программного обеспечения, которое радует пользователей, – это трудная работа. Обойти это невозможно, за каждым успешным программным продуктом и услугой стоит серьезная разработка. Интеллектуальные системы обладают некоторыми уникальными свойствами, которые предъявляют интересные вызовы. В этой книге описаны концепции, позволяющие разрабатывать эффективные и надежные интеллектуальные системы и лучше всего раскрывающие возможности машинного обучения и науки о данных.

Во-первых, эта книга дает определение **сущностей и абстракций**, которые должны присутствовать в успешной интеллектуальной системе. Вы изучите понятия, лежащие в основе среды выполнения интеллекта, контекста и функций, моделей, телеметрии, данных обучения, управления интеллектом, комбинирования и многого другого.

Во-вторых, книга даст вам **концептуальное понимание машинного обучения и науки о данных**. Она подготовит вас к продуктивному обсуждению компромиссов между трудозатратами на разработку и моделирование – когда пара часов вашей работы действительно помогает найти решение и когда вас просят о невозможном, чтобы сэкономить немного времени на моделировании.

В-третьих, в книге будут **рассмотрены шаблоны интеллектуальных систем**, которые мы с коллегами разработали и внедрили за минувшее десятилетие. Каковы плюсы и минусы выполнения интеллекта на стороне клиента и во внешнем сервисе? Как связывать и проверять вероятностные компоненты? Что нужно включить в телеметрию, чтобы система могла развиваться?

Руководители программ

Машинное обучение и наука о данных сегодня на пике популярности. Это фантастические инструменты, но все же это не более чем инструменты; они не являются решениями. Эта книга даст вам концептуальное понимание того, чем эти инструменты хороши и как их использовать для решения ваших бизнес-задач.

Первое, чему вы научитесь, – это **интуитивному чутью, когда уместны машинное обучение и наука о данных**. Для руководителя программы нет ничего хуже, чем пытаться вбить квадратный колышек в круглое отверстие. Вы

должны хорошо понимать, какие проблемы можно решить с помощью машинного обучения. Но, что не менее важно, вы должны понимать, какие проблемы *невозможно* или, по крайней мере, нелегко решить с его помощью. В этой области деятельности много участников, и они говорят на таких разных и сложных технических языках, что им трудно понять друг друга. Эта книга поможет вам научиться задавать правильные вопросы и понимать ответы.

Второй навык – чутье окупаемости инвестиций, чтобы определить, **насколько дорогостоящую интеллектуальную систему использовать**. Понимая реальные затраты на создание и обслуживание системы, которая превращает данные во влияние на пользователя, вы можете выбрать оптимальный момент для начала разработки и внедрения. Вы войдете в эту область с открытыми глазами и будете иметь навык оценки инвестиций, достаточный для успешной работы. Иногда востребованы все элементы интеллектуальной системы, описанные в этой книге, но иногда правильным выбором для бизнеса бывает менее сложная система. Эта книга поможет вам принимать правильные решения и обоснованно отстаивать их.

Наконец, третье, чему научится руководитель программы, – как **планировать, укомплектовывать персонал и управлять проектом интеллектуальной системы**. Воспользуйтесь нашим опытом построения множества масштабных интеллектуальных систем в следующих ключевых вопросах: жизненный цикл интеллектуальной системы; пошаговый процесс ее запуска; команда и навыки, необходимые для успеха.

Принятые в тексте соглашения


Так будут оформляться советы и рекомендации.


Так будут оформляться примечания.

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 прямо 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 напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **www.dmkpress.com**.

Список опечаток

Хотя мы приняли все возможные меры, для того чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

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

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты **www.dmkpress.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Часть I

ОСНОВЫ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Главы 1–4 закладывают основу для понимания интеллектуальных систем. Эта часть книги объясняет, что такое интеллектуальные системы и для чего они применяются. В ней рассказано, как удостовериться, что интеллектуальная система имеет полезную и достижимую цель, а также дается обзор некоторых проблем и методов реализации интеллектуальных систем.

Глава 1

Знакомство с интеллектуальными системами

Мы окружены *интеллектуальными системами* (intelligent systems). Они встроены в наши лампочки, автомобили, часы, термостаты и компьютеры. Как эти системы улучшают нашу жизнь? Нравится ли это нам?

Когда должна включиться лампочка? Когда на сайте магазина следует показать определенный товар? Когда поисковая система должна показать нам ссылку на сайт? Когда динамик должен воспроизвести звук?

В ответах на подобные вопросы (точнее, в достаточно хороших ответах) и содержится главная ценность интеллектуальных систем. Это трудная задача.

У некоторых крупнейших и наиболее дорогих компаний в мире весь бизнес, по сути, состоит из хороших ответов на простые вопросы:

- Какую страницу я должен показать в ответ на запрос?
- Какую рекламу показывать на странице сайта?
- Какой продукт я должен показать этому покупателю?
- Какой фильм этот пользователь захочет смотреть прямо сейчас?
- Какую книгу хотел бы прочитать этот человек?
- Какие новости вызовут наибольший интерес?
- Какие программы я должен заблокировать, чтобы защитить компьютер?

Хорошие ответы на эти вопросы принесли компаниям миллиарды, а в некоторых случаях сотни миллиардов долларов. При этом жизнь множества людей стала счастливее, продуктивнее и безопаснее. Но это лишь верхушка айсберга.

Есть десятки тысяч подобных вопросов, на которые нам приходится искать ответ: когда должна открыться моя входная дверь? Каким должно быть следующее упражнение в фитнес-приложении? Какой должна быть новая песня композитора? Как должен развиваться сюжет игры, чтобы максимально вовлечь игрока?

Эта книга о правильном и эффективном использовании возможностей интеллектуальных систем.

Элементы интеллектуальной системы

Интеллектуальные системы – это мостик, соединяющий пользователей с искусственным интеллектом (машинным обучением) для достижения значимых целей. В интеллектуальной системе интеллект развивается и совершенствуется с течением времени, особенно в процессе наблюдения за взаимодействием пользователя с системой.

Успешные интеллектуальные системы обязательно содержат следующие компоненты:

- ***значимая цель*** (meaningful objective)**.** Интеллектуальную систему не создают просто так. Должен быть обоснованный повод создать систему, которая имеет значение для пользователей и соответствует вашей цели – такой, которая достижима именно при помощи интеллектуальной системы. Но выбрать значимую цель совсем не просто. Первая часть этой книги поможет вам понять, что делают интеллектуальные системы, поэтому вы будете знать, в каких случаях следует использовать одну из систем и какие цели вы должны для нее установить;
- ***интеллектуальный опыт*** (intelligent experience). Система предоставляет пользователям выходные данные системного интеллекта (например, прогнозы на основе машинного обучения) и получает от пользователей обратную связь.

  Интеллектуальный опыт связан с пользовательским интерфейсом, который подстраивается под прогнозы и позволяет интеллектуальной системе сработать с максимальной отдачей, когда прогноз оправдался, и в то же время сводит к минимуму стоимость ошибки. Интеллектуальный опыт должен накапливать как скрытые, так и явные отзывы пользователей, чтобы система могла со временем улучшить свой интеллект. Во второй части этой книги будет рассмотрен интеллектуальный опыт, его возможности и подводные камни, подстерегающие нас при взаимодействии между интеллектом и пользователем;
- ***реализация*** (implementation). Внедрение интеллектуальной системы включает в себя создание среды для работы интеллекта, размещение среды в нужном месте, управление интеллектом, получение интеллектуального опыта, сбор телеметрии, чтобы убедиться, что система функционирует, сбор информации, а также обратную связь с пользователем. Третья часть этой книги описывает все компоненты реализации интеллекта. Она подготовит вас к разработке и внедрению собственной интеллектуальной системы;
- ***создание интеллекта*** (intelligence creation). Интеллектуальная система – это интеллект, настроенный на максимальный успех. Такой интеллект можно сформировать разными способами, от простой эвристики до сложного машинного обучения. Интеллект следует формировать так, чтобы подходящие типы интеллекта решали нужные задачи и чтобы он

мог эффективно совершенствоваться группой людей в течение длительного времени. Четвертая часть этой книги освещает процесс создания и развития интеллекта для интеллектуальных систем интернет-масштаба. Вы научитесь использовать все доступные подходы с максимальной отдачей;

- ***оркестровка*** (orchestration). Все элементы интеллектуальной системы должны работать согласованно, как оркестр. Оркестровка включает в себя управление изменениями в системе, синхронизацию опыта с интеллектом, принятие решений о том, какую телеметрию собирать для отслеживания и устранения проблем, а также управление денежными средствами, которые можно потратить на создание и развертывание нового интеллекта. Сюда также входят устранение ошибок, контроль рисков и предотвращение злоупотреблений. В пятой части этой книги говорится о том, что нужно для организации интеллектуальной системы и достижения целей на всех этапах ее жизненного цикла.

Интеллектуальная система является одним из способов применения машинного обучения на практике. Интеллектуальная система включает в себя интеллект, созданный с помощью машинного обучения и других подходов, и применяет его для достижения ваших целей. Интеллект совершенствуется в течение всего срока жизни системы.

Пример интеллектуальной системы

Интеллектуальные системы могут быть использованы для построения поисковых систем, сайтов электронной коммерции, беспилотных автомобилей и систем компьютерного зрения, которые знают, кем является человек и когда он улыбается. Но это большие и сложные системы.

Мы рассмотрим более простой пример, чтобы увидеть, как решение бытовой задачи может превратиться в интеллектуальную систему.

Умный тостер

Давайте обсудим подключенный к интернету тостер. Хорошая идея? Может быть, да, а может быть, нет – не узнаем, пока не попробуем. У нашего тостера есть два элемента управления: ползунок, управляющий интенсивностью обжаривания, и рычаг, включающий тостер.

Поначалу задача не выглядит сложной: умный тостер должен преобразовать положение ползунка в длительность обжаривания. При слабой обжарке тостер работает, скажем, 30 секунд. При сильной обжарке тостер работает в течение двух минут. Вот и все.

А теперь придумайте фиксированную настройку длительности обжарки и отправьте тостер пользователям. Что может пойти не так?

Если вы выберете максимальную длительность, тост может сгореть. Большинство пользователей, которые используют эту настройку, будут ужасно недовольны, выбросят обугленный тост и начнут снова.

Представьте себе, сколько случится сбоев во время приготовления тостов, когда раздраженные пользователи стоят над тостерами и держат руку на рычаге, чтобы вовремя прервать обжарку. Или когда пользователи многократно поджаривают один и тот же кусок хлеба, чтобы довести его до нужной кондиции.

Это плохо. Если вы собираетесь успешно продавать тостеры, вы должны разработать действительно хороший тостер.

Поэтому, возможно, вы проведете предварительное тестирование, подбирая длительность обжарки, пока не получите оптимально приготовленный тост. Не слишком хрустящий, не слишком бледный.

Отлично.

Правильно ли вы поняли задачу? Будет ли этот тостер делать то, что хотят пользователи? Трудно сказать однозначно. Независимо от того, сколько тостов вы съели в вашей жизни, невозможно доказать, что вы правильно предусмотрели все типы тостов, которые могут захотеть сделать все ваши пользователи.

Теперь вы понимаете, что необходимо учитывать мнения и опыт других людей в процессе разработки тостеров. Но как?

Может быть, вы начнете с фокус-группы? Пригласите десятки любителей тостов, поместите их в тостерную лабораторию и делайте заметки, наблюдая за процессом поджаривания.

Затем снова настройте длительность обжарки, чтобы отразить результаты лабораторных исследований. Ну как, теперь у вас есть идеальный тостер? Сможет ли этот тостер, настроенный для фокус-группы, сделать правильный тост, который понравится сотням тысяч людей по всему миру?

Что, если кто-то положит в тостер замороженный хлеб? Или другой продукт из холодильника? Или кто-то любит зажарить тост дочерна? А что, если кто-то изобрел новые тосты, не похожие на все прежние тосты человечества? Подходит ли ваш тостер для всех этих ситуаций? Наверняка нет.

Использование данных

Пожалуй, сделать идеальный тостер оказалось немного сложнее, чем просто спросить людей, что им нравится.

У вас получается слишком много вариантов настроек, если вы хотите получить идеальный тост в любой мыслимой ситуации. Вы можете исследовать фокус-группы каждый день всю оставшуюся жизнь, но так и не увидеть все типы тостов, которые способен сделать тостер.

Вам нужен другой подход. Настало время для серьезной науки о данных.

Тостер подключен к интернету, поэтому вы можете запрограммировать его для отправки данных телеметрии на ваш сервер. Каждый раз, когда кто-то поджаривает тост, вы можете узнать, какую настройку он использовал и сколько времени длилась обжарка.

Вы отправляете первую версию тостера клиентам (возможно, ограниченному кругу пользователей), и тостерная телеметрия начинает отправлять данные к вам на сервер.

Теперь вы точно знаете, какие настройки тостеров используют люди в своей реальной жизни (а не в какой-то надуманной лабораторной обстановке). Вы знаете, сколько раз люди нажимают на рычаг, чтобы начать обжарку, и сколько раз они поднимают рычаг, чтобы срочно отключить тостер.

Можете ли вы использовать эти данные, чтобы сделать тостер лучше?

Конечно!

Вы могли бы для начала ограничить максимальную интенсивность обжарки тем значением, которое фактически использует лишь несколько пользователей. Затем вы можете подобрать метрики, чтобы убедиться, что эта настройка не смещена в сторону чрезмерного поджаривания. Например, вы можете отслеживать процент тостов, которые преждевременно извлечены пользователями (предположительно потому, что они начали подгорать), и настраивать верхнюю границу обжарки, пока не исключите подобные ситуации.

Аналогично вы могли бы определить минимальную интенсивность обжарки, исходя из предпочтений пользователей. Вы можете отследить случаи двойной обжарки (когда кто-то повторно включает тостер сразу после первого цикла обжарки) и устранить склонность тостера к недостаточному поджариванию.

Замечательно – теперь вы можете даже настроить обжарку по умолчанию, которая находится в середине диапазона, то есть установить наиболее востребованную длительность обжарки тоста.

Поскольку ваши тостеры подключены к интернету, в них можно загружать новые настройки с вашего сервера. Вообще-то, вы можете менять настройки тостера хоть каждый день или дважды за воскресенье – мы живем в эпоху невиданных чудес!

В вашем подходе есть несколько допущений. Например, вы могли предположить, что поджаривание несколько раз подряд является признаком недостаточной обжарки, когда пользователь снова поджаривает один и тот же тост, а не пытается быстро приготовить несколько разных тостов.

Вы также могли предположить, что срочное отключение – это признак того, что хлеб начинает гореть, а не признак того, что пользователь опаздывает на работу и выбегает из дома.

Кроме того, когда вы загружаете в тостеры новые настройки, как вы можете быть уверены, что они понравятся пользователям? Вероятно, вы полагаете (основываясь на данных), что новые настройки лучше соответствуют запросам наибольшего количества пользователей.

Но как насчет пользователя, который еще вчера получал свой собственный идеально поджаренный тост, а сегодня внезапно получает … нечто другое?

Несмотря на эти проблемы, у вас получился довольно приличный тостер. У него есть телеметрия, которая сообщает, что тостер примерно выполняет свою работу. Тостер можно обслуживать и улучшать с течением времени. А теперь окончательно отпустим фантазию с поводка и придумаем по-настоящему крутую штуку.

Датчики и эвристический интеллект

Если вы хотите сделать лучший в мире тостер, вам не хватит регулятора обжарки и рычага включения. Придется добавить в тостер несколько датчиков:

- датчик веса, чтобы узнать, сколько тостов в тостере, и определить, когда клиент поместил что-то в тостер и когда он вынул содержимое;
- датчик температуры, чтобы узнать температуру предмета, помещенного в тостер;
- датчик геолокации, позволяющий узнать, в каком регионе мира находится тостер, чтобы он мог адаптироваться к различным вкусам в разных регионах;
- датчик приближения, чтобы узнать, когда кто-то подошел к тостеру, и камера, чтобы определить, кто это;
- часы, чтобы узнать, является тост завтраком или ужином;
- небольшую память, для того чтобы запомнить, что было недавно поджарено, и для отслеживания изменений в настройках;
- датчик дыма, чтобы узнать, когда тостер допустил серьезную ошибку и собирается что-то сжечь.

Теперь, когда пользователь подходит к тостеру и что-то кладет в него, тостер может посмотреть, кто подошел, попытаться угадать, что он пытается поджарить, и автоматически предложить настройку.

И вообще, если тостер настолько хорош, больше нет необходимости в настройке интенсивности обжарки или рычаге включения. Вы могли бы отправлять потребителям тостеры без кнопок, ручек и прочих органов управления. Пользователь бросает в тостер кусок хлеба, уходит и возвращается к восхитительному тосту!

Эх, остается всего лишь найти способ превратить все эти показания датчиков в правильную длительность обжарки для любого тоста…

Для этого вам необходим *интеллект* – программа, набор правил или машинно-обучаемая модель, которая принимает решения.

Давайте начнем с ручного подбора правил для интеллекта.

Возьмите блокнот и запишите набор *эвристических правил*, которые учитывают показания датчиков и рекомендации по интенсивности обжарки. Например: если в тостер положили что-то холодное и тяжелое, то жарить 5 минут с высокой интенсивностью. Но для каждого градуса выше нуля уменьшить время обжарки на 2 секунды. Но в Британии к обжарке прибавляют 15 секунд (потому что им это нравится). Но если вес и размер тоста соответствуют известному фирменному продукту, то следует выбрать «достаточную продолжительность» в соответствии с руководством изготовителя тостов.

Ну и так далее.

Каждый раз, когда пользователь жалуется, что тостер неправильно поджарил хлеб, вы можете добавить новое правило.

Каждый раз, когда телеметрия показывает всплеск двойных обжарок, вы можете настроить правила для улучшения обжарки.

Каждый раз, когда вы добавляете новое правило или обновляете старое, вы можете обновить все тостеры, которые вы продали по всему миру, попросив их загрузить новые настройки с вашего сервера.

При таком подходе вам, вероятно, потребуется написать и поддерживать множество правил, которые соответствуют всем возможным ситуациям. Это очень большая работа. Вы можете нанять дюжину сотрудников и дать им несколько месяцев для написания правил, но все равно не будете довольны результатом.

Тостер с машинным обучением

В подобных ситуациях, когда процесс слишком трудно или слишком дорого оптимизировать вручную, люди обращаются к *машинному обучению*.

На высоком уровне машинное обучение может отслеживать поведение владельцев интернет-тостеров и автоматически вырабатывать набор правил для управления тостером. Эти правила не отличаются от эвристических правил, которые вы записали в блокнот, разве что созданы машиной.

Но машины могут создавать гораздо больше правил, чем люди. Машины способны собирать данные с десятков датчиков и оптимизировать работу тостеров для тысяч различных пользователей одновременно. Они могут подключать новые данные и повторно оптимизировать систему каждый день, каждый час – иногда даже быстрее. Машины могут составить личные правила для каждого пользователя. Для работы машинному обучению нужны сведения о ситуации, с которой столкнулся пользователь (показания датчиков), какое действие предпринял пользователь (как он настроил тостер) и какой результат он получил (понравился ли тост, или пришлось что-то исправить и попробовать снова). Машинному обучению нужны примеры, когда дела идут хорошо, а также примеры, когда дела идут плохо, – как можно больше разных примеров.

В качестве примеров успешных операций вы можете взять выборки данных для случаев, когда пользователь положил хлеб в тостер, нажал на рычаг запуска, дождался завершения, вынул тост и ушел. Это те случаи, когда пользователь, вероятно, получил то, что хотел. У вас есть записи показаний всех датчиков и длительности обжаривания. Этот набор данных можно использовать для системы машинного обучения.

В качестве примеров неудачных операций имеет смысл взять выборки данных для случаев, когда пользователям приходилось возиться с тостером, преждевременно отключать тостер или многократно повторять обжарку одного тоста. Теперь у вас есть записи показаний всех датчиков и длительности обжаривания для неудачных операций.

Таким образом, располагая наборами данных для удачных и неудачных операций плюс машинное обучение, вы обучаете алгоритм управления тостером. Вы вводите данные и получаете программу, которая просматривает показания датчиков и определяет оптимальные настройки длительности обжарки.

Затем вы рассылаете эту обученную программу на тостеры ваших клиентов.

Вы можете делать это каждый день, каждую минуту.

Вы извлекаете уроки из сотен тысяч взаимодействий с клиентами – и даже из сотен миллионов взаимодействий.

Это великолепно!

Создание интеллектуальной системы

Создание эффективной интеллектуальной системы не обходится без пяти основных элементов: цели, опыта, реализации, интеллекта и оркестровки.

Вам нужна проблема, которая стоит затраченных усилий и решается с помощью интеллектуальной системы. Например, вы должны управлять тостером, основываясь на том, что решил интеллект. Вам придется создать систему, которая работает так, как нравится пользователю, предоставить ему удобные элементы управления и обратную связь, которую вы будете использовать для обучения интеллекта. Вам нужно разработать все сервисы, инструменты и код, который собирает телеметрию, генерирует интеллект, перемещает его туда, где он должен быть, и связывает его с пользователями.

Вам придется совершенствовать интеллект каждый день, снова и снова, что вполне предсказуемо. И вам нужно, чтобы все работало в течение долгого времени, так как на рынке появляются новые продукты для поджаривания в тостере, а вкусы людей меняются.

Вам нужно решить, какой объем данных телеметрии следует собирать, чтобы потенциальная ценность данных компенсировала эксплуатационные расходы. Также нужно решить, насколько сильно менять интеллект при обновлении, чтобы тостер улучшался, но не сбивал с толку и не расстраивал пользователей.

Вы должны отслеживать ключевые показатели и реагировать, если они начинают ухудшаться. Может быть, надо точнее настроить опыт? Может быть, надо срочно вмешаться в настройки интеллекта? И конечно же, вам придется иметь дело с ошибками, ибо они непременно случаются.

Интеллект (особенно машинно-обучаемый интеллект) может совершать грубые ошибки – зрелищные, нелогичные, наносящие большой ущерб клиентам.

Например, ваш тостер может узнать, что люди в определенном районе любят зажаренный до угольков хлеб. Не важно, чего пользователь хочет от тостера, но если он живет в этом районе, то получит угли. Поэтому вам нужны способы выявления и устранения подобных ошибок. Иногда приходится ставить заграждения на пути машинного обучения.

И к сожалению, приходится признать, что… люди есть люди. Каждый раз, когда вы используете обратную связь от пользователя для оптимизации процесса, например используете машинное обучение для создания тостера, вы должны думать о том, как человек может извлечь выгоду из того, что что-то идет не так.

Представьте себе, что крупный производитель хлеба платит наемным работникам за то, чтобы они жарили хлеб конкурента, используя аномальные настройки. Наемники станут жарить миллионы тостов в неделю, наводняя вашу систему телеметрии ложными данными. Машинное обучение воспользуется этими данными и научит интеллект плохим вещам – ваши тостеры будут регулярно портить продукты конкурента, что приведет, как минимум, к проблемам с безопасностью пищевых продуктов.

Это очень плохо.

Успешная интеллектуальная система должна учитывать все эти проблемы и многое другое.

Итог главы

В этой главе были представлены интеллектуальные системы, а также пять концептуальных задач, которые необходимо решать при разработке интеллектуальной системы: цель, опыт, реализация, интеллект и оркестровка.

Теперь вам должно быть ясно, что существует множество способов решения этих задач и что они тесно взаимосвязаны. Чтобы получить успешную интеллектуальную систему, вы должны применять сбалансированный подход. Если в вашем контексте особенно сложна одна из концептуальных задач, придется пересмотреть другие задачи, чтобы компенсировать перекос.

Например, если вы пытаетесь внедрить интеллектуальную систему в существующую систему управления, в которой опыт уже определен (и не может быть изменен), возможно, имеет смысл установить менее сложную цель, чтобы вкладывать больше средств в интеллект, или иметь более продуманную стратегию смягчения ошибок.

Но можно взглянуть на ситуацию и с другой стороны: есть разные пути для достижения успеха.

Вы развернули интеллектуальную систему и обнаружили, что проблема с интеллектом сложнее, чем вы думали? Не надо паниковать. Существует множество способов компенсировать дисбаланс интеллекта и создать систему, которая радует клиентов и помогает вашему бизнесу прямо сейчас, в то время как для развития интеллекта требуется время.

Остальная часть этой книги даст вам инструменты для уверенной работы над проектами интеллектуальных систем.

Темы для размышлений

Прочитав эту главу, вы сможете:

- распознать интеллектуальные системы в окружающем мире;
- увидеть возможности, которые открывают интеллектуальные системы;
- понять разницу между интеллектом (который делает прогнозы) и интеллектуальной системой (которая сочетает в себе цель, опыт, реализацию, интеллект и управление для достижения результатов);

- сформулировать все концептуальные задачи, которые вам нужно будет решить, чтобы построить успешную интеллектуальную систему;
- понять, как взаимодействуют сложные элементы системы, включая компромиссы и способы, которыми они могут дополнять друг друга.

Постарайтесь ответить на следующие вопросы:

- Какие сервисы вы используете, которые (как вы думаете) созданы путем превращения массива данных клиентов в интеллект?
- Какой самый лучший опыт вы получили с одним из этих сервисов?
- Какой у вас был самый худший опыт?
- Можете ли вы сказать, как опыт пользователя помогает интеллекту?
- Можете ли вы найти какую-либо информацию о том, как формируется интеллект? Может быть, вы читали об этом в новостях или статьях?
- Можете ли вы назвать какие-либо способы выявления и смягчения ошибок интеллекта?

Глава 2

Анализ применимости интеллектуальных систем

Итак, у вас есть проблема, которую надо решить: оптимизация существующей системы или идея нового бизнеса. Что-то такое, что должно понравиться вашим клиентам. Но решение, основанное на обычном машинном обучении, не дает желаемого результата. Подходит ли в этом случае интеллектуальная система? Не обязательно.

В этой главе говорится о том, когда уместно применять интеллектуальные системы, а когда лучше использовать другие подходы. Глава начинается с описания типов задач, для решения которых нужны интеллектуальные системы. Затем обсуждаются некоторые условия, необходимые для работы интеллектуальных систем.

Типы задач, для которых нужны интеллектуальные системы

Старайтесь решать задачи простыми способами. Если вы можете решить задачу без интеллектуальной системы, значит, так тому и быть. Как вы думаете, сколько раз вам придется обновить систему, прежде чем она начнет правильно делать свою работу? Если ожидаемое число обновлений невелико, то вряд ли вам нужна интеллектуальная система.

Например, представьте, что вы внедряете интеллект в розничном сегменте банковской системы. Ваш интеллект должен обновлять баланс счета, когда люди снимают деньги. В целом решение довольно простое:

```
Новый баланс = Старый баланс – Сумма списания.
```

Разумеется, предусмотрены проверки на ошибки, ведение журналов и управление транзакциями, но все равно это проще, чем ядерная физика. Возможно, вы не заметите ошибку, допущенную при разработке системы, и баланс однажды станет отрицательным. Тогда вам потребуется обновить интеллект, чтобы устранить ошибку. Как вы думаете, сколько раз вам придется устранять такие

ошибки? Наверное, два-три раза. Даже если разработчик системы очень невнимателен, общее количество доработок вряд ли превысит сотню.

Интеллектуальные системы не для таких задач. Они востребованы, когда приходится обновлять систему намного чаще – тысячи раз, десятки тысяч раз, каждый час, пока существует система!

Существуют четыре ситуации, когда уместен подход с бесконечной настройкой интеллектуальной системы:

1) *масштабные задачи*, для решения которых требуется большой объем работы;
2) *открытые задачи*, которые продолжают расширяться со временем;
3) *меняющиеся задачи*, когда правильный ответ меняется со временем;
4) *сложные задачи*, которые раздвигают границы наших возможностей.

Далее эти задачи будут рассмотрены по очереди.

Масштабные задачи

Некоторые задачи невероятно обширны. У них настолько много переменных и условий, которые необходимо учитывать, что их невозможно решить за один раз.

Например, в интернете существует столько веб-страниц, что для их прочтения не хватит жизни одного человека – и даже сто человек не справятся с этой задачей. Существует так много книг, телевизионных программ, песен, видеоигр, потоковых трансляций, твитов, новостей и продуктов электронной коммерции, что потребуются тысячи человеко-лет, чтобы прочитать и просмотреть эти продукты человеческого разума.

Подобные задачи требуют принципиально иного масштабного подхода. Если вы хотите создать систему, основанную на одном из традиционных подходов, и собираетесь окончательно отладить ее, прежде чем открыть первую версию для публичного доступа, то скажем прямо: вы непременно обанкротитесь.

Когда у вас есть масштабная задача, которую, как вы думаете, невозможно решить за один подход, интеллектуальная система может стать эффективным способом для начала работы. Система будет давать пользователям то, в чем они нуждаются, а взамен вы будете получать от пользователей обратную связь и совершенствовать систему.

Открытые задачи

Некоторые задачи не просто большие – они открытые. То есть у них нет окончательного решения. Они развиваются бесконечно, и столь же бесконечно может длиться работа над их решением. Веб-страницы, книги, телевизионные программы, песни, видеоигры, прямые трансляции событий – все больше и больше разнообразных продуктов создается каждый день.

Задумайтесь, насколько трудно спроектировать и развернуть систему для обработки продуктов, которые еще не созданы (и неизвестно, будут ли созданы в будущем).

В таких случаях законченное статическое решение – когда вы его создаете, разворачиваете и уходите – вряд ли сработает. Вместо этого нужны сервисы, которые живут очень долго и растут на протяжении всей своей жизни.

Если у вашей задачи есть ограниченное и законченное решение, можно обойтись без интеллектуальной системы. Но если ваша задача масштабная и открытая, интеллектуальная система, скорее всего, будет правильным выбором.

Меняющиеся задачи

Задачи могут меняться. Иногда правильное решение внезапно становится неправильным. Например:

- представьте отлаженную систему распознавания человеческих лиц – и вдруг татуировки на лице становятся очень популярными;
- представьте систему для прогнозирования цен на акции – и вдруг самолет врезается в здание;
- представьте настроенную систему для блокировки спама – и вдруг темный гений программирования придумывает новый способ рассылки сообщений;
- наконец, представьте себе пользовательский интерфейс, который пользователи сначала не хотели использовать, а потом начинают учиться работать с ним.

Можно лишь одно сказать наверняка: все меняется.

Решения, которые вы внедрили вчера, прекрасно справляются с текущими задачами, радуют ваших пользователей и приносят много денег вашему бизнесу. Но буквально завтра эти решения могут оказаться неподходящими.

Для решения задач, которые меняются со временем, нужна способность выявлять изменения и быстро менять систему, чтобы не отстать от жизни.

Если ваши задачи меняются медленно или предсказуемо, интеллектуальная система может не понадобиться. С другой стороны, если задачи меняются внезапно, радикально или часто, интеллектуальная система может стать вашим спасением.

Сложные задачи

Некоторые задачи просто сложны. Настолько сложны, что люди не могут понять, как их решить. По крайней мере, не сразу, и не идеально. Вот несколько примеров сложных задач:

- понимание человеческой речи;
- идентификация объектов на картинках;
- прогнозирование погоды на несколько дней вперед;
- конкуренция с людьми в сложных открытых играх без явного финала;
- распознавание человеческих эмоций в тексте и видео.

Да, машинное обучение достигло большого успеха в решении этих задач, но успех пришел после многих лет усилий, сбора данных, обучения, осознания

проблем и развития интеллекта. Эти типы систем все еще совершенствуются и продолжат улучшаться в обозримом будущем.

Существуют разные способы решения сложных задач. Один из способов – организовать обратную связь между пользователями и интеллектом с использованием интеллектуальной системы, как описано в этой книге.

Ситуации, когда интеллектуальные системы успешны

Кроме достаточно сложной глобальной задачи (значимой цели), для успешной работы интеллектуальной системы требуется еще несколько вещей:

- проблема, частичное решение которой является жизнеспособным и полезным;
- способ получения данных об использовании системы для улучшения интеллекта;
- способность влиять на значимую цель;
- проблема, которая оправдывает усилия по созданию интеллектуальной системы.

В оставшейся части этого раздела рассматриваются эти требования по очереди.

Когда неполная система жизнеспособна и полезна

Интеллектуальные системы по сути своей всегда являются неполными, неправильными в некоторых аспектах и могут (точнее, обязательно будут) совершать всевозможные ошибки. Не важно, что интеллектуальная система не идеальна. Она должна быть достаточно хорошей, чтобы быть интересной для пользователей (и полезной для вас).

Интеллектуальная система жизнеспособна, когда ценность того, что она делает правильно, превышает стоимость ошибок.

Рассмотрим интеллектуальную систему, которая делает «дешевые» ошибки. Допустим, система должна отображать список покупок в порядке, который оптимизирует затраты времени пользователя на покупки в продуктовом магазине. Пройдите направо, возьмите яйца, затем молоко, дойдите до мясного отдела и возьмите стейк, затем перейдите к витрине с консервами и т. д. Если система не отлажена, она может тратить время пользователя напрасно, например 15–20 секунд на каждую ошибку. Это слегка раздражает, но не отторгает пользователей. Если ошибки случаются не слишком часто, скажем один раз за каждый поход по магазинам, легко можно допустить, что пользователи продолжат применять систему, пока она изучает планировки всех продуктовых магазинов в мире и адаптируется к изменениям.

Рассмотрим другую интеллектуальную систему, которая допускает «дорогие» ошибки. Возможно, она управляет хирургическими роботами, а ошибки ставят под угрозу жизнь человека. Такой системе придется преодолеть гораздо более высокую планку, прежде чем она станет жизнеспособной. То есть вам следует

создать гораздо лучшую неполную систему перед ее развертыванием для пользователей. Но даже такая система не обязательно должна быть идеальной, чтобы быть жизнеспособной. Помните: каждый хирург совершает ошибки. Интеллектуальная система не обязательно должна быть идеальной (и большинство систем никогда не будут такими), она просто должна быть лучше других.

Для того чтобы интеллектуальная система была жизнеспособной, она должна обеспечить очевидную выгоду для пользователей (и для вас) при условии большого перевеса стоимости правильных решений над стоимостью ошибок.

Обратная связь для улучшения интеллекта

Интеллектуальные системы работают, замыкая петлю *обратной связи* между пользователем и интеллектом. Чем больше пользователи используют систему, тем лучше становится интеллект; и когда интеллект становится лучше, система создает большую ценность для пользователей.

Чтобы улучшить интеллект системы, вам следует тщательно вести журнал взаимодействия между пользователем и системой (о чем мы поговорим позже). На высоком уровне это включает в себя регистрацию того, что пользователь видел, что он сделал, и был ли результат положительным или отрицательным. Необходимо наблюдать очень, очень много взаимодействий с пользователем (чем сложнее задача, тем больше наблюдений нужно). Этот тип данных можно применять для улучшения интеллекта разными способами, в том числе с помощью машинного обучения.

Когда система способна влиять на результат

Интеллектуальная система должна быть способна эффективно влиять на результат. Это можно сделать за счет автоматизации части системы или предоставления информации либо опций, которые помогают пользователю достичь своих целей. Интеллектуальные системы наиболее эффективны при соблюдении следующих условий:

- действия, которые интеллектуальная система может предпринять, *непосредственно* влияют на результат. Причиной положительного результата должна быть хорошая работа системы, а не хорошее стечение обстоятельств; причиной плохого результата должна быть плохая работа системы. Чем больше внешних факторов влияют на результат, тем сложнее создать эффективную интеллектуальную систему;
- действия, которые интеллектуальная система может предпринять, *быстро* влияют на результат. Чем больше интервал времени между действием и результатом, тем больше шансов для вмешательства внешних факторов;
- действия, которые выполняет интеллектуальная система, *соответствуют* целям. То есть действия, которые интеллектуальная система может предпринять, достаточно значимы для результата, на который они нацелены.

На практике ключевым вызовом для разработчика является выяснение того, на каком уровне сложности поместить интеллектуальную систему и как много проблем пользователя она должна решать. Поставьте перед собой слишком большую цель, и вы не сможете подключить к ней интеллектуальную систему; слишком скромная цель не будет стоить проделанной работы.

На самом деле многие большие, сложные системы внутри себя содержат несколько вложенных интеллектуальных систем. Например, операционная система компьютера может иметь следующие вложенные интеллектуальные системы:

- для прогнозирования того, что помещать в кеш-память ОЗУ;
- для прогнозирования того, какие файлы показывать пользователю при поиске;
- для рекомендации приложений пользователям на основе истории использования;
- для управления компромиссами между мощностью и производительностью, когда компьютер питается от батареи.

Вы можете предложить использование единой интеллектуальной системы для управления операционной системой. Но этот подход вряд ли будет хорошо работать, потому что:

- слишком много разнородных объектов и параметров для контроля;
- средства управления слишком отделены от цели;
- отклики обратной связи, которые получает система, не связаны напрямую с решениями, которые принимает интеллектуальная система.

Когда интеллектуальная система окупается

Разработка и запуск интеллектуальных систем подразумевают иную структуру затрат, чем другие подходы к созданию компьютерных систем и сервисов. В большинстве подобных систем есть три основных компонента стоимости: интеллект, функциональная реализация и оркестровка. Мы обсудим эти понятия в частях 3, 4 и 5 данной книги. Но если в общих словах, то *интеллект* – это инструмент принятия правильных решений о том, что и как делать; *реализация* – это все сервисы, серверные системы и клиентский код, которые реализуют систему и доносят решения интеллекта до пользователя; *оркестровка* – это согласованное управление всеми частями системы на протяжении всего ее жизненного цикла и устранение ошибок.

Интеллект в составе интеллектуальных систем обычно более выгоден, чем другие подходы, по двум причинам:

1) в целом интеллектуальные системы создают сложную логику принятия решений автоматически (вместо использования людей). При решении большой, сложной или открытой задачи интеллектуальные системы могут привести к значительной экономии на производстве интеллектуального продукта;
2) обладая надлежащим интеллектуальным опытом, интеллектуальные системы автоматически генерируют обучающие выборки (которые тре-

буются для машинного обучения) на основе взаимодействия с пользователем. Сбор обучающих данных зачастую является главной проблемой при создании интеллекта и требует значительных человеческих усилий, поэтому получение данных от пользователей может кардинально изменить экономику разработки и внедрения системы.

Реализация интеллектуальной системы сопоставима по затратам или немного дороже, чем реализация систем, основанных на серверных службах (таких как веб-сервис или «умный» чат для клиентов).

Многие компоненты интеллектуальных и неинтеллектуальных систем схожи между собой, но у интеллектуальных систем есть дополнительные требования, связанные с управлением в течение жизненного цикла, включая создание интеллекта, настройку интеллектуального опыта, управление ошибками, организацию интеллекта и т. д. Позже мы обсудим все это подробно.

Расходы на оркестровку интеллектуальной системы обычно немного больше по сравнению с использованием неинтеллектуального сервиса. Интеллектуальные системы требуют постоянной работы, включая настройку представления информации пользователям, повышение уровня интеллекта для улучшения его работы и реагирования на новые ситуации, а также выявление и устранение ошибок.

К тому времени, когда вы закончите читать эту книгу, вы сможете подготовить подробный анализ эффективности использования интеллектуальной системы для вашего приложения.

Действительно ли нужна интеллектуальная система?

Иногда у вас есть область, где взаимодействуют пользователи, данные и интеллект, но вы не уверены, что вам нужна интеллектуальная система. Сомнения могут возникнуть по любой из следующих причин:

- вы не уверены, что проблема на самом деле сложная, поэтому не хотите создавать целый набор замысловатых интеллектуальных решений, которые вам могут не понадобиться;
- вы понимаете, что проблема сложная, но не уверены, что дополнительная выгода от создания интеллектуальной системы оправдает ваши усилия;
- вы хотите использовать поэтапный подход и решать проблемы по мере их возникновения, а не останавливаться для построения интеллектуальной системы.

Все в порядке. Интеллектуальные системы подходят не всем – существует множество других способов создания систем, которые решают сложные проблемы. Но если вы попытаетесь решить большую, открытую, меняющуюся со временем или сложную задачу, вы в конечном итоге вернетесь к подходам, описанным в этой книге. Даже если вы решите не создавать интеллектуальную систему, эта книга поможет вам быстро определить общие проблемы, понять, что происходит, и подготовит вас к решению задач с помощью проверенных подходов.

Итог главы

Интеллектуальные системы наиболее полезны при решении больших, открытых, меняющихся или сложных задач. Это случаи, когда статичное решение не сработает, а система нуждается в постоянном улучшении на продолжении всего срока жизни.

Интеллектуальные системы хорошо работают, когда:

- частичное решение интересно пользователям и будет поддерживать жизнеспособный бизнес, так что вы сможете начать с неполной системы и постепенно расширять ее возможности;
- вы можете получать данные для улучшения системы от пользователей, когда они используют систему. Данные обычно являются главным источником затрат на построение интеллекта, поэтому создание системы, которая автоматически их генерирует, очень ценно;
- когда решения системы влияют на результат напрямую, быстро и осмысленно. То есть результаты могут быть привязаны к тому, что делает интеллектуальная система;
- когда другие подходы слишком дороги для масштабирования.

При поиске мест для использования интеллектуальных систем разбейте свою проблему на части. Каждая часть должна иметь свою четкую цель, которая имеет решающее значение для общего успеха, быструю и прямую обратную связь с пользователем и управление (или отклик на управляющие действия пользователя). Подобные части являются кандидатами на внедрение интеллектуальной системы.

И даже если вы решите вообще не использовать интеллектуальную систему, знания из этой книги о практическом машинном обучении, взаимодействии интеллекта с пользовательским опытом и многом другом принесут пользу практически любому проекту машинного обучения.

Темы для размышлений

Прочитав эту главу, вы должны знать:

- может ли ваша существующая или будущая система извлечь выгоду из интеллекта;
- как найти потенциальные применения интеллектуальных систем в мире вокруг вас.

Постарайтесь ответить на следующие вопросы:

- какое ваше хобби выиграет от внедрения интеллектуальной системы?
- каков пример вашей повседневной деятельности, когда интеллектуальная система может работать неправильно? Почему так получается?

Рассмотрим вашу любимую компьютерную программу:

- определите три места, в которых программа может использовать интеллектуальную систему;
- какой из вариантов применения наиболее успешен? Какой наименее успешен? Почему?

Глава 3

Краткие основы работы с данными

Данные являются ключевым элементом интеллектуальной системы. В этой главе дается краткий обзор методов работы с данными, знакомство с ключевыми понятиями из области науки о данных, статистики и машинного обучения. Назначение главы состоит в том, чтобы сформировать базовый уровень понимания, достаточный для обсуждения и принятия решений всеми участниками команды по разработке интеллектуальной системы.

Эта глава охватывает следующие темы:

- структурированные данные;
- анализ данных;
- модели данных;
- концепция машинного обучения;
- некоторые распространенные проблемы при работе с данными.

Структурированные данные

Данные правят миром.

Но что такое эти данные? Куча картинок на диске? Все диктанты, которые вы написали в школе? Средние значения рейтинга каждого бейсболиста, который когда-либо играл?

Да, все это данные. Существует бесчисленное множество данных – океаны чисел, миллионы жестких дисков, заполненных единицами и нулями. Необработанные данные часто называют *неструктурированными*, и с ними довольно сложно работать.

Когда мы превращаем данные в интеллект, то обычно работаем со *структурированными* данными. То есть данные разбиты на логические блоки, которые описывают сущности или события.

Например, представьте, что вы работаете с данными о людях. Каждый блок данных описывает одного человека по его весу, росту, полу, цвету глаз и тому подобному.

Один из удобных способов представления блока данных – это таблица, в которой каждая строка соответствует конкретному человеку и каждый столбец соответствует какому-то свойству человека. Допустим, первый столбец каждой строки содержит вес соответствующего человека, а третий столбец каждой строки содержит рост человека (табл. 3.1).

Таблица 3.1. Пример представления структурированных данных

Вес, кг	Пол	Рост, см	Цвет глаз	...
69	жен.	172	карий	...
89	муж	182	серый	...
102	муж.	190	голубой	...
...	...	...	...	...

Столбцы обычно содержат точные числа (например, рост и вес), или выбираются из небольшого числа категорий (например, коричневый или голубой для цвета глаз), или представляют собой короткие текстовые строки – например, имена. Конечно, есть более продвинутые варианты представления данных, но эту простую и удобную схему можно использовать для представления самых разнообразных данных (и применять для статистики и машинного обучения).

Например, вы можете:

- представить веб-страницу – количество слов, которое она содержит, количество изображений, которые она содержит, количество ссылок, которые она содержит, и количество вхождений (наличия) каждого из 1000 популярных ключевых слов;
- представить посещение поисковой системы: запрос, введенный пользователем, количество времени, в течение которого пользователь оставался на странице результатов, количество результатов поиска, на которые пользователь нажал, и URL-адрес окончательного результата, на котором он остановился;
- представить событие поджаривания хлеба на умном тостере: с помощью температуры предмета, помещенного в тостер, настройки интенсивности, выбранной пользователем, количества раз, когда пользователь останавливал и запускал тостер вручную перед удалением хлеба, и интервала времени между моментом, когда хлеб был помещен в тостер, и когда он был вынут.

Это очень упрощенные примеры. На практике даже простые сущности склонны иметь десятки или сотни элементов в своем представлении (десятки или сотни столбцов в строке). Определение правильных наборов данных и правильной формы для их представления имеет решающее значение при обработке массива данных. И вы, вероятно, ошибетесь несколько раз, прежде чем поймете, как делать правильно, – будьте готовы постоянно развиваться.

Задавайте данным простые вопросы

Так что вы можете сделать с данными? Вы можете *задать им вопросы*. Например, набору данных о людях вы можете задать такие вопросы:

- каков средний рост?
- кто самый высокий человек?
- сколько людей имеют рост ниже 180 см?

На такие вопросы довольно легко ответить; просто возьмите данные и вычислите значения (сложите и разделите числа или посчитайте строки, которые соответствуют критериям).

В интеллектуальной системе вы будете задавать множество подобных вопросов, например:

- сколько раз в день пользователи совершают определенные действия?
- какой процент пользователей переходит по рекламе, предложенной интеллектом?
- каков средний доход на одного клиента в месяц?

Ответы на подобные вопросы важны для понимания того, выполняет ли система свою работу (и следит ли она сама за тем, чтобы работа выполнялась правильно).

С данными мы можем делать еще одну замечательную вещь – *давать прогнозы*. Представьте себе набор данных о сотнях людей. Хотите знать средний рост? Нет проблем, просто берите данные и считайте. Но что, если вы хотите предположить рост следующего человека, который будет добавлен в набор данных? Или хотите предположить, будет ли следующий человек ниже или выше 175 см?

Можете ли вы попробовать это вычислить? Конечно! Нам пригодятся основы статистики.

С помощью нескольких простых предположений статистика может оценить наиболее вероятный рост следующего человека, добавленного в набор данных. Но статистика способна на большее. Она может прикинуть, насколько точен прогноз, например:

> *Наиболее вероятный рост следующего человека – 177 см, и с вероятностью 95 % рост следующего человека будет находиться в интервале между 170 и 182 см.*

Этот интервал ожидаемых значений называется *доверительным интервалом*. Ширина доверительного интервала зависит от того, сколько данных у вас есть и насколько «хорошо себя ведут» данные. Чем больше данных, тем сильнее можно сузить доверительный интервал (например, ожидать с вероятностью 95 %, что следующий человек будет иметь рост между 172 и 179 см). Чем меньше данных, тем шире доверительный интервал. Почему важно знать доверительный интервал?

Допустим, вы хотите оптимизировать интернет-бизнес за счет сокращения службы поддержки клиентов, чтобы сэкономить деньги. Текущая система спо-

собна принимать до 200 звонков в службу поддержки в неделю. Итак, изучая журнал телеметрии, вы можете рассчитать, что в среднем за неделю было 75 обращений в службу поддержки и что максимальное количество звонков в любую неделю составляло 98. Интуитивно понятно, что 200 намного больше, чем 75 – система, должно быть, напрасно тратит ваши деньги. Сокращение мощности системы вдвое (до 100 звонков) кажется безопасным, особенно потому, что у вас в журнале нет ни одной записи с нагрузкой, большей, чем 98.

Но проверьте доверительный интервал. А что, если наиболее вероятный объем звонков составляет 75 в неделю и с 95%-ной вероятностью на следующей неделе будет от 40 до 110 звонков? В таком случае 100 уже не кажется хорошим вариантом. В проекте с интенсивным использованием данных вы всегда должны задавать вопросы и получать ответы. Но вы также должны спросить себя, насколько вы уверены в ответах. Вы должны убедиться, что решения, которые вы принимаете, учитывают не только ответ, но и степень уверенности в ответе.

Работа с моделями данных

Модели данных извлекают ответы из данных, преобразовывая их в более удобные или более полезные форматы.

С технической точки зрения, наши предыдущие рассуждения о работе службы поддержки клиентов являются моделью. Напомним, что «сырой» набор данных о загрузке центра обработки звонков был преобразован в средний еженедельный объем 75 с 95%-ным доверительным интервалом 40–110. Необработанные данные могут быть огромными. Возможно, они содержат десять терабайт телеметрии за прошедшее десятилетие. Но модель содержит всего четыре числа: 75, 95 %, 40 и 110.

И эта простая модель оказалась более полезной для принятия решений о планировании мощностей, потому что она содержала именно необходимую информацию и интуитивно собирала ее для решения поставленной задачи.

Модели могут также *заполнить пробелы в данных*. Рассмотрим набор данных людей, их рост и вес. Представьте, что набор содержит данные человека, у которого рост 176 см, и данные человека, у которого рост 181 см. Но данных о человеке с ростом 179 см в наборе нет.

Допустим, вам нужно предсказать, сколько будет весить человек с ростом 179 см.

Вы можете построить простую модель. Изучая данные, вы могли заметить, что люди в наборе данных имеют тенденцию весить около 0,51 кг на сантиметр роста. Согласно вашей модели, человек с ростом 179 см будет весить 91 кг. Отлично. Или все-таки нет?

Один из распространенных способов оценки качества моделей называется *ошибка обобщения*.

Для проверки модели резервируют часть набора данных и скрывают эти данные от разработчика модели. Затем просят модель предсказать данные, ко-

торые были скрыты, чтобы увидеть, насколько хорошо модель обобщает (предсказывает) данные, которые никогда не видела.

Возможно, в наборе данных действительно был человек с ростом 179 см и весом 112 кг, но кто-то скрыл его от вас в процессе моделирования. Ваша модель предсказала вес 91 кг – между реальностью и предсказанием разница 21 кг.

Это хорошо или плохо? Зависит от того, для чего вам нужна модель.

Если вы просуммируете ошибки этого типа для десятков и сотен людей, которые были скрыты от процесса моделирования, вы можете получить некоторое представление о том, насколько хороша модель. Есть много технических способов суммировать ошибки и оценить качество модели, вот два из них:

- *ошибки регрессии* – это разница между реальным и прогнозируемым числовыми значениями, как в примере с весом. В предыдущем примере ошибка регрессии модели составляла 21;
- *ошибки классификации* относятся к моделям, которые предсказывают категории. Например, модель, которая пытается предсказать, является ли человек мужчиной или женщиной на основе роста и веса (да уж, у такой модели не будет много друзей…). Один из способов измерения ошибок классификации заключается в *оценке точности*: модель с точностью 85 % отличает мужчин от женщин.

Концепция машинного обучения

Простые модели могут быть полезны. Модель, основанная на зависимости веса от роста, очень проста, и в ней легко найти недостатки. Например, в ней не учитывается пол, процент жира в организме, окружность талии, а также генетическое происхождение человека. Статистика полезна для составления быстрых и простых прогнозов. Создайте простую модель и используйте ее для ответов на простые вопросы. Но модели могут быть и чрезвычайно сложными.

Машинное обучение использует компьютеры для улучшения процесса создания сложных моделей на основе данных. И эти модели тоже могут делать прогнозы на основе данных, иногда удивительно точные, а иногда удивительно ошибочные.

Алгоритм машинного обучения исследует данные, чтобы определить наилучший способ объединения информации, содержащейся в представлении (столбцы в наборе данных), в модель, которая точно обобщает скрытые или несуществующие данные.

Например, алгоритм машинного обучения может предсказывать следующее:

- пол из сочетания роста, веса, возраста и имени;
- вес из сочетания пола, роста, возраста и имени;
- рост из сочетания пола и веса
- и т. д.

Устройство модели может быть очень сложным, с перемножением и масштабированием различных входных данных такими способами, которые для людей выглядят бессмысленно, однако дают точные результаты.

Существуют, пожалуй, тысячи алгоритмов машинного обучения, которые создают модели данных. Одни алгоритмы быстрые, а другие работают несколько дней или недель, прежде чем создадут модель. Одни алгоритмы производят очень простые модели, а другие занимают мегабайты или гигабайты дискового пространства. Некоторые алгоритмы строят модели, которые могут очень быстро делать прогнозы на основе новых данных с точностью до миллисекунды. Зачастую алгоритмы преуспевают на строго определенных типах данных, но терпят неудачу, стоит лишь изменить данные.

Похоже, что не существует одного универсального лучшего алгоритма машинного обучения, который можно использовать для всех ситуаций, – большинство работает хорошо в одних ситуациях, но плохо в других. Каждые несколько лет появляется новая технология, которая на удивление хорошо справляется с решением важных задач и привлекает много внимания.

Некоторые профессионалы машинного обучения считают лучшим в мире тот алгоритм, с которым они наиболее знакомы, и они будут биться насмерть, чтобы защитить его, несмотря на любые доказательства обратного. Будьте готовы к этому.

Процесс машинного обучения обычно разбивается на следующие сложные этапы, которые требуют значительных усилий со стороны экспертов:

- **получение данных для моделирования** (data collecting, getting the data). Этап включает в себя борьбу с шумом данных, точное выяснение того, что надо будет предсказывать, и получение обучающих примеров для алгоритма моделирования;
- **извлечение признаков** (feature engineering). Этап включает в себя компоновку данных в набор с правильным представлением (например, столбцы в электронной таблице), чтобы передать данные алгоритму машинного обучения;
- **моделирование** (modeling). Этап включает в себя запуск одного или нескольких алгоритмов машинного обучения для набора данных, просмотр ошибок, которые совершает алгоритм, настройку и доработку алгоритма, а также создание новой модели – и так до тех пор, пока модель не станет достаточно точной;
- **развертывание** (deployment). Этап включает в себя выбор модели (если их несколько) и включение ее в систему с максимальным положительным эффектом и минимальным ущербом от ошибок;
- **обслуживание** (maintenance). Этап включает в себя мониторинг того, что модель ведет себя, как ожидалось, и исправление ситуации, если она начинает выходить из-под контроля. Например, можно перезапустить алгоритм обучения на новых данных.

Эта книга о том, как пройти через этапы машинного обучения при разработке больших и сложных систем.

Распространенные ошибки при работе с данными

Данные бывают очень сложными, и многое может пойти не так. Вот несколько распространенных ошибок, о которых следует помнить при работе с системами с интенсивным использованием данных, такими как интеллектуальные системы.

Нарушение доверительных интервалов

Доверительные интервалы очень полезны. Как приятно знать, что с вероятностью 95 % вы попадаете в интервал допустимых значений. Но, с другой стороны, это означает, что есть и вероятность 5 % не попасть в интервал. Просто задумайтесь, что 5 % – это каждый двадцатый. Вспомним колл-центр, работу которого мы оцениваем по количеству звонков в неделю. Вероятность 5 % означает, что каждую двадцатую неделю один клиент не сможет дозвониться до службы поддержки. Это около 2,6 звонка в год. Хорошо это или плохо – решать вам. На первый взгляд кажется, что событие с вероятностью 5 % может никогда не произойти, но имейте в виду, что даже маловероятные события в конечном итоге произойдут, если испытывать удачу достаточно долго.

Зашумленность данных

Шум данных (noise) – это еще один повод для ошибок. Допустим, в наборе данных есть взрослые люди с ростом 3 м и 7 м. Это реально? Вероятно, нет. Шум возникает повсюду. Например, в программном обеспечении для сбора данных есть ошибки. Исходные данные содержат ошибки ввода. Системы телеметрии регистрируют неверные значения. Данные искажаются при обработке. Программа статистической обработки содержит ошибки. Пользователи выключают свои компьютеры когда попало, что порождает странные наборы данных и безумную телеметрию. Иногда, когда у пользователя возникает ошибка, он вообще не генерирует данные, которых от него ждали, или генерирует часть набора данных. Иногда шум вызван недопониманием между людьми. Каждый большой набор данных будет иметь шум, который внесет ошибку в результат работы системы.

Смещение данных

Смещение данных (bias) возникает, когда способ сбора данных систематически отличается от способа использования данных. Например, набор данных о людях, который мы использовали для оценки роста, был создан путем опроса случайных людей на улицах Нью-Йорка. Будет ли модель этих данных достаточно точной для использования в Японии? В Гватемале? В Тимбукту?

Смещение может возникать, если вы игнорируете контекст, в котором данные были собраны. Например, пользователи, которым нравится предмет опроса, с меньшей вероятностью ответят на опрос, чем пользователи, которые не-

навидят этот предмет. Убедитесь, что данные собраны строго для того, для чего они предназначены, и постарайтесь выявить и устранить последствия смещения данных.

Устаревание данных

Большинство простых статистических и машинных методов обучения основано на фундаментальном допущении, что мир не меняется. Но все меняется, и зачастую случается *устаревание данных*. Представьте себе, что вы строите модель количества звонков в службу поддержки, а затем используете ее, чтобы оценить количество звонков спустя неделю после выпуска новой сложной функции. Или спустя неделю после сильного шторма, который оборвал все линии связи. Одна из главных причин внедрения интеллектуальной системы – сделать систему устойчивой к изменениям.

Необоснованное использование данных

Иногда данные выглядят неубедительно, но людям нравится получать ответы. Такова человеческая природа. Люди стремятся уменьшить степень неопределенности, давая такие ответы, как «42», вместо того чтобы честно сказать «мы не можем ответить на этот вопрос» или «ответ где-то в интервале от 12 до 72». Точные ответы позволяют людям чувствовать себя умнее. Иногда разработчикам системы кажется, что это вежливо – давать людям совершенно необоснованный, но краткий и точный ответ. Забавно слышать аналитические выводы наподобие такого: «Теперь наш лучший продукт – зубная паста, потому что мы переделали витрину в прошлом месяце». Определенность выглядит настолько соблазнительно, что люди находят ее даже там, где ее никогда не было.

При работе с данными всегда задавайте себе вопросы: это правильно? Насколько мы уверены? Есть ли другая интерпретация? Как мы можем узнать, что является правильным?

Ваши модели данных будут делать ошибки. Интеллектуальные системы, особенно созданные на основе моделирования данных, ошибочны. Иногда эти ошибки можно объяснить. Но иногда они совершенно нелогичны. Иногда очень трудно исправить одну ошибку, не вводя несколько новых ошибок. Но это нормально. Не бойтесь ошибок. Просто имейте в виду: любая система, основанная на моделях, нуждается в работе над ошибками.

Итог главы

Мир состоит из данных. Данные наиболее полезны, когда они структурированы таким образом, чтобы соответствовать цели использования. Структурированные данные похожи на электронную таблицу, с одной строкой на отдельную сущность (человек, событие, веб-страница и т. д.) и по одному столбцу на каждое свойство этой сущности (рост, вес, пол).

Данные могут быть использованы для ответа на вопросы и составления прогнозов. Статистика может ответить на простые вопросы и оценить, насколько точным является ответ. Машинное обучение может создавать очень сложные модели для создания очень сложных проекций данных. Машинное обучение имеет очень много полезных инструментов для создания точных моделей данных.

Данные могут быть использованы неправильно – вы должны быть осторожны.

Темы для размышлений

После прочтения этой главы вы должны:

- различать типы задач, для которых используются данные;
- иметь представление о том, как применять данные;
- выработать интуитивное чутье, когда работа с данными может пойти не так и привести к неправильным результатам.

Постарайтесь ответить на вопрос:

- какая самая большая ошибка, о которой вы знаете, была совершена из-за неправильного использования данных?

Глава 4

Определение целей интеллектуальной системы

Интеллектуальная система объединяет интеллект с опытом для достижения желаемой цели. Ваша работа будет успешной, если выполняется три условия:

- цель достижима;
- интеллект нацелен на правильную проблему;
- опыт поощряет правильное поведение пользователя.

Правильно сформулированный *критерий успеха* системы выражает желаемый результат простым языком. Он указывает, какие подзадачи необходимо решить интеллекту и опыту, и связывает эти локальные решения с ожидаемым более масштабным (стратегическим) результатом. В хорошо продуманных критериях успеха предусмотрено, что все члены команды разработчиков могут видеть, как их работа способствует достижению общей цели. Это удерживает их от движения в неправильном направлении, даже если данные и опыт сговорились ввести их в заблуждение.

В этой главе обсуждается постановка целей для интеллектуальных систем, в том числе:

- признаки хорошей цели;
- почему сложно сформулировать хорошие цели;
- различные типы целей, которые может иметь система;
- некоторые способы оценки целей.

Признаки хорошей цели

Правильно сформулированная цель имеет следующие признаки:

1) четко и однозначно ***обозначает желаемый результат*** всем участникам проекта. Каждый член команды должен понимать, как выглядит успех и почему он важен, независимо от его научного или технического опыта;
2) ***является достижимой.*** Все члены команды должны верить, что они работают на успех. Задача может быть трудной, но члены команды должны быть в состоянии хотя бы приблизительно объяснить, как они собираются приблизиться к успеху и почему есть хорошие шансы, что это сработает;

3) ***является измеримой.*** Интеллектуальные системы предназначены для оптимизации, а следовательно – для измерения. Если вы не можете что-то измерить, то вы не сможете это оптимизировать.

Нелегко узнать, правильно ли сформулирована цель. Фактически преодоление разрыва между целями высокого уровня и локальными задачами реализации часто является ключевой проблемой для создания успешной интеллектуальной системы. Некоторые цели кажутся одним участникам идеальными, но не имеют смысла для других участников. Некоторые из них будут четко соответствовать положительным результатам, но их будет невозможно измерить или достичь. Всегда будут компромиссы между желаемым и возможным, и обычно разработчики тратят много времени на уточнение критериев успеха.

Но это нормально. Результат того стоит, потому что неспособность определить критерии успеха до начала работы над проектом – это верный способ напрасно потратить время и деньги.

Пример затруднений при выборе цели

Рассмотрим антифишинговую функцию на основе интеллектуальной системы.

Одна из форм *фишинга* (fishing) включает веб-сайты, которые выглядят как настоящие банковские сайты, но на самом деле являются подделками под контролем злоумышленников. Пользователей заманивают на эти фишинговые сайты и обманывают, вынуждая предоставить свои банковские пароли преступникам.

Итак, что должна делать интеллектуальная система?

Спросите у специалиста по машинному обучению, и вы обнаружите, что он готов немедленно приступить к работе. Он тут же придумает, как построить модель, которая проверяет веб-страницы и прогнозирует, являются они фишинговыми сайтами или нет. Эта модель будет учитывать такие параметры, как текст и изображения на веб-страницах, и на основе этих параметров делать свои прогнозы. Если модель считает, что страница является фишинговой, пусть браузер заблокирует ее. Когда страница заблокирована, пользователь не будет просматривать ее и не будет вводить в нее свой банковский пароль. Проблема решена, и все знают, что делать.

Вдобавок количество блокировок в качестве критерия эффективности системы легко измерить: чем больше сайтов заблокировано, тем лучше работает система.

Или нет?

Что, если система настолько эффективна, что фишеры вынуждены сдаться? Вдруг каждый фишер в мире перестанет воровать пароли и начнет заниматься чем-то хорошим?

Отлично!

Но тогда больше не будет фишинговых сайтов, и количество блокировок упадет до нуля. Фактически система достигла полного успеха, но критерий указывает на полный провал.

Чушь какая-то.

Или что, если система ежедневно блокирует миллион фишинговых сайтов, но фишерам все равно? Каждый раз, когда система блокирует сайт, фишеры просто создают новый сайт. Интеллектуальная система блокирует миллионы сайтов, все в команде довольны, и всем кажется, что они помогают людям – но после создания системы такое же количество пользователей теряет свои пароли, как и до ее создания.

На самом деле система не работает, хотя критерий успеха указывает на обратное.

Ловушка измерения успеха в интеллектуальной системе состоит в том, что существует очень много параметров, которые можно измерить и оптимизировать. Очень легко найти что-то знакомое и понятное, выбрать его в качестве критерия и отвлечься от настоящего успеха.

Напомним три свойства правильно сформулированной цели:

1) соответствует желаемому результату;
2) является достижимой;
3) является измеримой.

Использование блокировок сайтов в качестве цели соответствует пунктам 2 и 3, но никак не пункту 1. Желаемый результат работы антифишинговой системы – не блокировать фишинговые сайты, а помешать злоумышленникам получить банковские пароли пользователей.

Типы целей

Существует множество объектов для оптимизации, от очень конкретных до очень абстрактных.

Истинная цель (true objective) системы имеет тенденцию быть очень абстрактной (например, заработать больше денег в следующем квартале), но объекты, на которые она может непосредственно повлиять, имеют тенденцию быть очень конкретными (например, решить, должен ли тостер работать в течение 45 или 55 секунд). Обнаружение функциональной связи между абстрактными и прикладными целями является ключевой проблемой при постановке задач. И это действительно очень сложная работа.

Одна из причин, почему это трудно, состоит в том, что разные участники команды будут заботиться о разных типах целей. Например:

- заработать больше денег, привлечь и вовлечь клиентов;
- помочь пользователям достичь того, что они пытаются сделать;
- разработать наиболее точный интеллект системы.

Все это важные цели, и они взаимосвязаны, но связь между ними косвенная. Например, вы не заработаете много денег, если система всегда делает неправильные прогнозы, но повышение точности интеллекта на 1 % не приведет к увеличению прибыли на 1%.

В этом разделе рассматриваются различные способы оценки успеха интеллектуальной системы, в том числе:

- организационные цели (organizational objectives);
- опережающие показатели (leading indicators);
- результаты пользователя (user outcomes);
- свойства модели (model properties).

В большинстве интеллектуальных систем все они используются на регулярной основе, но для повседневной оптимизации в основном применяют результаты пользователя и свойства модели.

Организационные цели

Организационные цели – истинная причина внедрения интеллектуальной системы. В бизнесе это могут быть такие цели, как выручка, прибыль или доля рынка. В некоммерческой организации это могут быть сохраненные деревья, улучшение качества жизни или другие выгоды для общества.

Организационные цели явно важны для оптимизации. Но их проблематично использовать в качестве рабочих целей для интеллектуальных систем, по крайней мере по трем причинам:

1) они очень далеки от того, на что может повлиять технология. Например, разработчик умного тостера может изменить количество времени, в течение которого поджаривается кусок хлеба, – но как это связано с количеством проданных тостеров?
2) на них влияют многие вещи вне области контроля системы. Например, рыночные условия, маркетинговые стратегии, конкурентные силы, изменения в поведении пользователей с течением времени и т. д.;
3) это очень долгосрочный показатель. Могут потребоваться недели или месяцы, чтобы узнать, повлияло ли какое-либо конкретное действие на организационную цель. Это затрудняет прямую оптимизацию.

Каждая интеллектуальная система должна вносить вклад в достижение организационной цели, но повседневное управление интеллектуальной системой обычно фокусируется на более прямых критериях, в частности на результатах пользователей и свойствах модели. Мы обсудим их в следующих разделах.

Опережающие показатели

Опережающие показатели напрямую связаны с будущим успехом. Например:

- вы с большей вероятностью получите прибыль, когда вашим клиентам нравится ваш продукт, чем когда они ненавидят его;
- у вас больше шансов увеличить клиентскую базу, когда ваши клиенты рекомендуют ваш продукт своим друзьям, чем когда ваши клиенты советуют своим друзьям держаться подальше;
- у вас больше шансов удержать клиентов, когда они используют ваш продукт каждый день, чем когда они используют ваш продукт один раз в пару месяцев.

Опережающие показатели – это связующее звено между организационными целями и конкретными свойствами интеллектуальной системы (такими как

результаты пользователя и свойства модели). Если интеллектуальная система станет лучше, клиентам, вероятно, больше понравится ваш продукт или ваша фирма. Это не обязательно приведет к увеличению продаж, потому что другие факторы (конкуренты, маркетинговая деятельность, рыночные тенденции и т. п.) тоже влияют на продажи. Опережающие показатели устраняют влияние некоторых из этих внешних факторов и помогают вам быстрее получить обратную связь при изменении вашей интеллектуальной системы.

Существует два основных типа опережающих показателей: настроения клиентов и вовлеченность клиентов.

Настроение клиентов (customer sentiment) – это мера того, как ваши клиенты относятся к вашему продукту. Они любят его использовать? Он делает их счастливыми? Будут ли они рекомендовать его другу (или они скорее порекомендуют его врагу)?

Если все, кто использует ваш продукт, любят его, это признак того, что вы на правильном пути. Продолжайте работать, продолжайте расширять свою базу пользователей, и в конечном итоге вы добьетесь успеха в бизнесе (получите прибыль, продадите много единиц товара и т. д.).

С другой стороны, если все, кто использует ваш продукт, не любят его, у вас будут проблемы. У вас могут быть клиенты, они могут использовать ваш продукт, но они не довольны им. Они ищут выход. Если у вас есть сильный конкурент, ваши клиенты готовы перейти на другой корабль.

Настроения – нечеткая мера, потому что чувства пользователей могут быть переменчивыми. Иногда бывает нелегко измерить настроение – пользователи не всегда могут или хотят дать точные ответы на ваши вопросы. Тем не менее колебания настроений могут быть полезными индикаторами будущих результатов бизнеса, и интеллектуальные системы, безусловно, могут влиять на настроение пользователей, которые сталкиваются с ними.

Вовлеченность клиентов (customer engagement) – это показатель того, насколько полно ваши клиенты используют ваш продукт. Это не только частота, но и глубина использования, как мера использования всего богатства различных функций, которые может предложить ваш продукт.

Клиенты с высокой вовлеченностью своим поведением демонстрируют, что они находят ценность в вашем продукте. Они привыкли к вашему продукту и возвращаются снова и снова. Они будут ценны для вас и вашего бизнеса в течение длительного времени.

Клиенты с низкой вовлеченностью используют продукт нечасто. Эти клиенты могут получать выгоду от вашего предложения, но у них преобладают другие интересы. Они могут исчезнуть и никогда больше не думать о вас или вашем продукте.

Опережающие показатели в качестве целей для интеллектуальных систем имеют некоторые недостатки, аналогичные тем, от которых страдают организационные цели:

- они являются косвенными;
- на них влияют факторы, не зависящие от интеллектуальной системы;

- они не позволяют обнаруживать небольшие изменения;
- они обеспечивают медленную обратную связь, поэтому их трудно оптимизировать напрямую;
- их часто сложнее измерить, чем организационные цели (на сколько опросов вы сами хотели бы ответить?).

Тем не менее опережающие показатели могут быть полезны, особенно в качестве ранних индикаторов проблем, – независимо от того, что, по вашему мнению, должна делать интеллектуальная система, если у клиентов после обновления настроение стало гораздо хуже, чем до обновления, вы, вероятно, делаете что-то неправильно.

Результаты пользователя

Еще один подход к постановке целей для интеллектуальных систем – посмотреть на *результаты пользователя*, например:

- если ваша система помогает пользователям найти информацию, эффективно ли они находят полезную информацию?
- если ваша система помогает пользователям принимать лучшие решения, они принимают действительно лучшие решения?
- если ваша система помогает пользователям находить контент, который им понравится, находят ли они этот контент?
- если ваша система оптимизирует настройки компьютеров, начинают ли компьютеры работать быстрее?
- если ваша система защищает пользователей от мошенников, избегают ли они мошенничества?

Интеллектуальные системы могут ставить цели, подобные этим, и пытаться оптимизировать результаты, которые получают пользователи.

Оценивать результаты пользователя особенно важно, потому что результаты зависят от интеллекта и опыта интеллектуальной системы. Чтобы пользователь получил хороший результат, интеллект должен давать правильные прогнозы, а опыт должен помочь пользователю получить максимальную выгоду.

Например, в примере с антифишингом представьте информацию, которая на 100 % точна при выявлении мошенничества. Если система блокирует мошеннические страницы на основе этого интеллекта, пользователи получат хорошие результаты. Но что, если опыт взаимодействия с пользователем выстроен неправильно? Может быть, при посещении сайта мошенников в браузере пользователя появляется небольшое предупреждение – маленький красный крестик в адресной строке. Некоторые пользователи не заметят предупреждение. Некоторые не смогут понять его правильно. Иные проигнорируют значок. В этом случае некоторые пользователи получат плохие результаты (и отдадут свои пароли мошенникам), даже если интеллект правильно обнаружил мошенничество.

Достижение наилучшего результата пользователей может стать очень хорошей целью для интеллектуальных систем, потому что по результату можно

судить, насколько эффективно интеллект и опыт вместе влияют на поведение пользователей.

Свойства модели

В каждой интеллектуальной системе есть конкретные параметры для оптимизации, например:

- уровень ошибок модели, которая идентифицирует мошенников;
- вероятность того, что пользователь должен повторно поджарить свой хлеб;
- сколько раз пользователь примет рекомендованный контент;
- рейтинг кликов по рекламным баннерам, которые решила показать система.

Эти параметры не всегда напрямую соответствуют результатам пользователей, опережающим показателям или организационным целям, но они ставят очень хорошие цели для сотрудников, которые работают над улучшением интеллектуальных систем.

Например, модель, которая по данным испытаний в лаборатории корректно работает в 85 % случаев, явно лучше, чем модель, которая работает в 75 % случаев. Ясно и конкретно. Легко получить быструю обратную связь. Легко добиться прогресса.

Но свойства модели имеют некоторые недостатки в качестве целей для интеллектуальных систем:

1. *Они не связаны с реальным миром пользователя.* Например, если интернет-тостер всегда жарит тосты с ошибкой 10 секунд относительно оптимального времени, устроит ли это пользователей? Будет ли лучше уменьшить ошибку до 5 секунд? Разумеется, да. Но насколько лучше? Вы уверены, что уменьшение ошибки сделает тост вкуснее? Если мы получим модель с ошибкой в 4 секунды, этого достаточно? Должны ли мы остановиться или добиваться ошибки в 3 секунды? Во сколько обойдется разработчику каждая дополнительная секунда точности?
2. *Они не характеризуют всю систему.* Но, с другой стороны, ошибка модели будет воспринята пользователями как ошибка системы в целом. Возможно, ошибка пользовательского опыта окажется настолько незначительной, что никого не озаботит. Или, может быть, у пользователя есть хороший способ дать отзыв, который исправляет ошибку, – вам будет гораздо дешевле и проще отреагировать на обратную связь, чем инвестировать в оптимизацию модели.
3. *Они слишком привычны для специалистов по машинному обучению.* При построении интеллектуальной системы легко сбиться на оптимизацию свойств модели – это именно то, чем постоянно занимаются эксперты машинного обучения, поэтому они всегда вспоминают о свойствах моделей в ходе любого разговора о целях. Будьте осторожны с этими людьми. Они настолько сильны и влиятельны в команде, что могут задушить и подменить реальную цель системы.

Оптимизация свойств модели – это именно то, что представляет собой интеллект, но это редко является целью системы. Хорошая цель покажет, как улучшение свойств модели способствует желаемому воздействию на пользователей и бизнес. А еще хорошая цель дает представление о том, сколько стоит оптимизация свойств модели.

Расслоение целей

Успех в проекте интеллектуальной системы сложно определить с помощью одной метрики, а метрики, которые его определяют, зачастую трудно измерить. Хорошая практика – применять *расслоение целей* (goals layering), то есть определять успех на разных уровнях абстракции и рассуждать о том, как успех на одном уровне помогает на других уровнях. Вам нужен не строгий текст наподобие технического задания, а история, которая касается всех участников команды.

Например, участники разработки интеллектуальной системы могут:

- ежечасно или ежедневно оптимизировать свойства модели;
- еженедельно проверять результаты пользователя и следить за тем, чтобы изменения свойств модели влияли на результаты пользователя, как и ожидалось;
- ежемесячно проверять опережающие показатели и следить за тем, чтобы ничего не ухудшалось;
- ежеквартально проверять организационные цели и следить за тем, чтобы интеллектуальная система развивалась в правильном направлении.

В течение работы над проектом регулярно и часто пересматривайте цели интеллектуальной системы, потому что все в этом мире меняется.

Способы измерить успех

Причина, по которой трудно дать определение успеха, состоит в том, что измерить успех еще труднее.

Скажите, откуда мы можем знать, сколько паролей злоумышленники получили со своих фишинговых страниц?

Во второй части этой книги мы обсудим способы разработки интеллектуального опыта путем измерения успеха и получения данных для улучшения интеллекта. В этом разделе представлены некоторые основные подходы.

Ожидание дополнительной информации

Иногда невозможно определить, является ли действие правильным или неправильным в тот момент, когда оно происходит, но через несколько часов, дней или недель результат становится очевидным. Со временем у вас обычно появляется больше информации для оценки взаимодействия. Вот несколько примеров того, как помогает ожидание:

- система рекомендует контент для пользователя, и через некоторое время пользователь его скачивает или покупает – дождавшись, пока пользо-

ватель использует контент, вы можете получить доказательства того, что рекомендация оказалась хорошей;

- система позволяет пользователю ввести свой пароль на веб-странице, а затем ожидает, не войдет ли с этим паролем пользователь с новым IP-адресом из Восточной Европы, чтобы получить доказательства того, что пароль был украден.

Ожидание может быть очень дешевым и эффективным способом облегчить измерение успеха, особенно когда поведение пользователя косвенно указывает на успех или неудачу.

Но есть и пара минусов.

Во-первых, ожидание добавляет задержку. Это означает, что ожидание может не помочь с оптимизацией или выполнением точных измерений.

Во-вторых, ожидание добавляет неопределенности. Есть много причин, по которым пользователь может внезапно изменить свое поведение. Ожидание дает больше времени для других факторов, влияющих на измерение.

A/B-тестирование

Предоставление разных версий интеллекта разным группам пользователей, или *A/B-тестирование* (A/B-testing), – это очень мощный способ количественного измерения эффективности интеллекта.

Представьте себе, что одна группа пользователей использует устройство с интеллектуальными функциями, а другая группа – простое устройство. Допустим, контрольная группа пользователей умного тостера всегда получает тост за одну минуту, независимо от того, какие настройки они используют или какой хлеб вставляют в тостер. У рабочей группы время обжарки тоста определяется самым изощренным интеллектом, который вы можете придумать.

Если пользователи, получившие рядовой опыт, столь же счастливы, увлечены и эффективны в поджаривании, как и те, кто получил интеллектуальный опыт, – у вас есть проблема.

A/B-тестирование может быть сложным в управлении, поскольку оно предполагает одновременное обслуживание нескольких версий продукта.

Также могут возникнуть проблемы с распознаванием мелких эффектов. Представьте себе, что вы тестируете две версии тостера, одна из которых жарит хлеб ровно одну минуту при любых условиях, а другая 61 секунду, тоже при любых условиях. Будет ли одна из версий из них лучше другой? Возможно. Но потребуется много времени и наблюдений за опытом пользователя, чтобы выяснить, какой тостер лучше.

A/B-тестирование хорошо помогает убедиться, что большие изменения в вашей системе приносят пользу, но затрудняет повседневную оптимизацию.

Ручная маркировка

Иногда компьютер не может сказать, достигает ли действие успеха, а человеку это понятно с первого взгляда. Вы можете нанять группу людей, чтобы перио-

дически проверять небольшое количество событий/взаимодействий и сообщать вам, были они успешными или нет. Во многих случаях эта *ручная маркировка* (hand labeling) проста и не требует каких-либо особых навыков или обучения.

При использовании ручной маркировки интеллектуальная система должна иметь телеметрию для записи и воспроизведения взаимодействий. Журнал телеметрии должен содержать достаточно деталей, чтобы человек мог достоверно сказать, что произошло и был результат хорошим или нет (при сохранении конфиденциальности пользователя). Это не всегда возможно, особенно когда нужно угадать, что пользователь пытался сделать, или как он себя чувствовал, когда делал это.

Но никогда не помешает взглянуть со стороны на то, что делает ваша система и как она влияет на пользователей. Даже при ограниченной выборке данных и маленьких затратах на исследование ручная маркировка способна принести большую пользу.

Опрос пользователей

Возможно, самый очевидный способ выяснить, что происходит, – это спросить пользователя. Например, встроив механизмы обратной связи прямо в продукт одним из способов:

- пользователь выбирает один вариант из нескольких фрагментов рекомендованного контента. Система спрашивает, доволен ли пользователь предложенными вариантами выбора;
- беспилотный автомобиль доставляет пользователя к месту назначения, и когда он выходит, автомобиль спрашивает, чувствовал ли пассажир себя в безопасности во время поездки;
- тостер периодически спрашивает: «Хлеб поджарен так, как вам нравится?» (И да, это может быть весьма жутко, особенно когда вы один дома после наступления темноты и свет погашен.)

Имейте в виду несколько ограничений:

- у пользователей не всегда есть ответ. Например, не стоит спрашивать «Вы только что дали свой пароль преступнику? Да/Нет?» Это не эффективно и может напугать многих людей без веской причины;
- пользователи не всегда хотят отвечать на ваши вопросы. Это вносит смещение в результаты опроса. Например, пользователи, которые очень заняты своей задачей, могут не задумываться над ответом, даже если довольны результатом;
- пользователям быстро надоедает отвечать на вопросы. Этот тип обратной связи следует использовать с осторожностью.

Но иногда, задавая всего 0,1 % пользователей простой вопрос один раз в месяц, можно раскрыть большой потенциал обратной связи, помогающий узнать, успешно ли работает ваша интеллектуальная система.

Разделение задач

Некоторые вещи трудно измерить напрямую, но их можно разбить на более простые части, которые можно измерить (возможно, используя некоторые методы из этого раздела), а затем объединить результаты измерений в оценку целого.

Например, рассмотрим фишинг. Уровень воровства учетных данных на фишинговых сайтах можно представить следующим набором измерений:

- количество ваших пользователей, которые посещают фишинговые сайты (основано на пользовательских сообщениях о выявлении фишинговых сайтов и объединении с телеметрией трафика для оценки количества посещений);
- процент пользователей, которые вводят свои пароли на фишинговых сайтах, когда система *не* выдает предупреждение (оценивается путем опроса пользователей);
- процент пользователей, которые вводят свои пароли на фишинговых сайтах, когда система *выдает* предупреждение (оценивается с помощью телеметрии о том, сколько пользователей отклоняют предупреждение и продолжают вводить пароль).

Соберите эти измерения вместе, и вы получите довольно неплохую потенциальную цель для интеллектуальной системы.

Разделение целей особенно полезно, когда оно выявляет критические, измеримые подзадачи и показывает, как они соединяются вместе с другими конкретными подзадачами для достижения общего успеха.

Сохраняйте актуальность целей

Иногда цель системы должна измениться. Но цели так сложно менять, особенно когда люди организовались в команду вокруг них. Это означает сильную инерцию, много разных мнений и мало хороших ответов. Тем не менее пересмотр целей и их адаптация – очень хорошая идея. Цели могут измениться, если случится что-то из перечисленного:

- новый источник данных показывает, что некоторые предположения были неверны;
- часть системы функционирует очень хорошо, и необходимо изменить цель ее дальнейшего совершенствования (прежде чем система перестанет соответствовать меняющимся целям, что приведет к инвестированию в неправильные вещи);
- кто-то придумал лучшую идею о том, как связать работу интеллектуальной системы с влиянием на пользователя;
- мир меняется, и предыдущая цель больше не отражает успех.

И даже если цели не нужно менять, хорошо бы время от времени собирать команду и напоминать себе о том, чего вы все пытаетесь достичь вместе.

Итог главы

Наличие целей имеет решающее значение для успеха интеллектуальной системы. Но цели трудно сформулировать правильно. Правильные цели должны:

1) приводить к желаемому результату;
2) быть достижимыми;
3) быть измеримыми.

Цели могут быть очень абстрактными (например, организационные цели). Они могут быть менее абстрактными (как опережающие показатели). Они могут быть вполне конкретными (например, результаты пользователя) или даже очень конкретными (например, свойства модели).

Эффективный набор целей связывает эти различные типы целей в одну общую цель, которая ведет команду разработчиков к успеху.

Большинство интеллектуальных систем будет способствовать достижению организационных целей и опережающих показателей, но основная работа по ежедневному улучшению системы будет сосредоточена на результатах пользователя и свойствах модели.

Цели можно измерить с помощью телеметрии, ожидания результатов, с помощью ручной маркировки и опроса пользователей об их опыте.

И – как уже не раз говорилось – цели трудно понять правильно с первого раза. Наверняка вы не обойдетесь без итерации.

Но без эффективных целей разработка интеллектуальной системы почти наверняка станет пустой тратой времени и денег.

Темы для размышлений

Прочитав эту главу, вы должны:

- понимать, как вы можете определить критерии успеха интеллектуальной системы и как измерить, достигнут ли успех;
- уметь определять успех на нескольких уровнях абстракции и рассказывать команде о том, как различные типы успеха способствуют друг другу и общей цели.

Постарайтесь ответить на следующие вопросы:

- какую выгоду для вашего любимого хобби можно извлечь из интеллектуальной системы?
- какой организационной цели будет способствовать интеллектуальная система?
- какие результаты будут иметь для этого наибольшее значение?
- каковы конкретные результаты пользователей, на которых будет отслеживаться влияние интеллектуальной системы?
- как бы вы измерили эти результаты? Почему?

Часть II

ИНТЕЛЛЕКТУАЛЬНЫЙ ОПЫТ

В главах 5–10 объясняется, как связать интеллект с пользователями для достижения целей интеллектуальной системы. В этой части рассматриваются подводные камни и проблемы создания пользовательского опыта на основе моделей и машинного обучения, а также свойства успешного интеллектуального опыта. Она рассказывает, как использовать интеллектуальный опыт для сбора данных и применять эти данные для развития системы, а также описывает общие принципы проверки правильности поведения интеллектуального опыта.

Глава 5

Компоненты интеллектуального опыта

В основе каждой интеллектуальной системы лежит связь между интеллектом и пользователем. Эта связь называется *интеллектуальным опытом* (intelligent experience). Эффективный интеллектуальный опыт позволяет:

- **представлять интеллект пользователю**, находя разумный баланс между качеством интеллекта и его привлекательностью для пользователя;
- **достигать целей системы** – создавать среду, в которой пользователи ведут себя так, чтобы достичь целей системы и при этом не разочаровываться;
- **минимизировать любые недостатки в интеллекте** – уменьшать влияние мелких ошибок и помогать обнаруживать и исправлять любые серьезные проблемы по мере возникновения;
- **создавать данные для развития системы** – формировать двусторонние взаимодействия и извлекать из них четкие и достаточно точные данные для улучшения интеллекта системы.

Достижение этих целей и поддержание их оптимального соотношения столь же затруднительно, как и любая другая часть разработки интеллектуальной системы, и столь же важно для достижения успеха.

В этой главе будет представлен обзор компонентов интеллектуального опыта. В следующих главах мы рассмотрим ключевые концепции более подробно.

Представление интеллекта пользователю

Интеллект делает прогнозы о мире, пользователе и о том, что будет дальше. Интеллектуальный опыт на основании этих прогнозов должен сформировать представление того, что пользователь видит и с чем взаимодействует. Например:

- интеллект предсказал 10 единиц контента, которые могут заинтересовать пользователя. Как опыт должен представить это предсказание пользователю?

- интеллект считает, что пользователь вносит опасные изменения в настройки своего компьютера. Как опыт должен предупредить пользователя?
- интеллект определил, что пользователь находится в комнате, где людям слишком темно. Как должен реагировать опыт автоматизированного дома?

Опыт в этих примерах может вести себя пассивно и давать мягкие подсказки пользователю, или быть активным и навязчивым, мигая большими красными огнями прямо в глаза. Опыт может автоматизировать поведение некоторых вещей. Выбор правильного поведения является одной из ключевых задач при создании интеллектуального опыта.

Интеллектуальный опыт может вести себя следующим образом:

- **автоматизировать** (automate) – предпринимать действия от имени пользователя. Например, когда интеллект полностью уверен, что пользователь спит, он может автоматически уменьшить громкость телевизора;
- **рекомендовать** (prompt) – спрашивать пользователя, следует ли предпринять какое-либо действие. Например, когда интеллект думает, что пользователь мог забыть про день рождения своей жены, опыт может спросить: «Вы хотите, чтобы я заказал цветы на день рождения вашей жены?»;
- **организовывать** (organize) – представлять набор предметов в порядке, который может быть полезен пользователю. Например, когда интеллект думает, что пользователь соблюдает диету, опыт может организовать электронное меню в любимом ресторане пользователя и поместить самые полезные блюда на видном месте (и спрятать все, что слишком вкусно);
- **аннотировать** (annotate) – добавлять информацию на устройство вывода. Например, когда интеллект считает, что пользователь опаздывает на собрание, в углу экрана может появиться мигающий значок бегущего человека.

Интеллектуальный опыт может использовать произвольную комбинацию этих методов, например аннотацию, когда проблема не совсем ясна, но автоматизацию, если в ближайшее время с высокой вероятностью произойдет что-то плохое.

Проблема состоит в том, что опыт должен извлечь максимальную пользу из интеллекта, *когда интеллект работает правильно*, и в то же время минимизировать ущерб, *когда интеллект совершает ошибку*.

Любая интеллектуальная система (независимо от уровня интеллекта) может использовать любой из этих подходов. Напомним, что интеллект в интеллектуальной системе со временем будет меняться, становясь лучше по мере того, как систему использует все больше пользователей (или становясь хуже по мере изменения проблемы). Интеллектуальный опыт тоже может измениться со временем.

Пример представления интеллекта

В качестве примера давайте рассмотрим «умный дом», который автоматизирует освещение в комнатах пользователей. Пользователь устанавливает несколько датчиков уровня освещенности, несколько датчиков для обнаружения движения и несколько управляемых компьютером лампочек. Теперь пользователь хочет наслаждаться работой системы. Он жаждет произвести впечатление на друзей своей новой игрушкой. Ему хочется никогда больше не думать о выключателе света.

Как это может работать? Во-первых, давайте рассмотрим интеллект. Роль интеллекта заключается в том, чтобы интерпретировать данные датчиков и делать прогнозы. Например, учитывая последние данные датчика движения и последние показания датчика освещенности, какова вероятность того, что пользователь захочет включить свет в ближайшем будущем? Интеллект может рассуждать так:

- если в комнате темно и кто-то только что вошел в нее, вероятность того, что свет скоро включится, составляет 95 %;
- если свет включен и кто-то покидает комнату, вероятность того, что свет скоро погаснет, составляет 80 %;
- если свет включен и никого не было в комнате в течение четырех часов, следующий человек, вошедший в комнату, скорее всего, будет кричать детям о том, сколько стоит электроэнергия, а затем выключит свет.

Итак, что должен делать опыт в интеллектуальной системе автоматизации освещения?

Одним из вариантов будет полная автоматизация освещения. Если интеллект полагает, что пользователь захочет изменить освещение в ближайшее время, он просто берет и делает вместо пользователя. Это довольно жесткий опыт, потому что он транслирует интеллект напрямую в оборудование, не учитывая возможность ошибки. Будет ли это правильно? Будет ли пользователь доволен?

Что, если пользователь смотрит фильм? Интеллектуальная система световой автоматики ничего не знает о телевизоре. Если пользователь смотрит фильм и датчик движения обнаруживает, что он меняет позу или тянется к своему напитку, – загорается свет. Это не лучший опыт. Что будет, если пользователь спит? Будет ли свет мигать каждый раз, когда он переворачивается на другой бок? А вдруг у пользователя романтический ужин?

В этих случаях опыт, который просто автоматизирует работу лампочек, вряд ли будет делать правильные вещи. Отчасти потому, что у интеллекта недостаточно информации, чтобы принимать совершенные решения, а отчасти потому, что интеллект не может принимать совершенные решения, даже если у него есть вся информация в мире (потому что интеллект по определению делает ошибки).

Выходит, мы обречены? Должны ли мы отказаться от внедрения продукта и вернуться к чертежной доске?

Да, вполне возможно. Или мы могли бы рассмотреть некоторые иные способы представления интеллекта пользователям.

Вместо того чтобы *автоматизировать* освещение, интеллект может *задать вопрос* пользователю, когда думает, что огни должны измениться. Например, если интеллектуальная система считает, что в комнате слишком темно, она может спросить приятным компьютерным женским голосом: «Хотите, чтобы я включила свет?»

Это может быть лучше, чем жесткая автоматизация, потому что рекомендательные ошибки, которые совершает система, меньше раздражают (приятный женский голос раздражает меньше, чем внезапная темнота в комнате). Но будет ли это достаточно хорошим решением? Что, если интеллект делает слишком много ошибок? Представьте, что голос звучит каждые несколько минут, ласково спрашивая: «Вы хотите, чтобы я погасила ваши лампы сейчас?»... «А может быть, сейчас?»... «А сейчас уже пора гасить свет?»

Система может показаться весьма глупой, если несколько раз в течение часа спрашивает пользователя, не хочет ли он погасить освещение.

Другой, еще менее навязчивый подход предоставляет информацию об освещении в виде *аннотаций*. Например, это может быть небольшое предупреждение о «лишнем потреблении энергии» на смарт-часах пользователя, если есть несколько ламп, которые, по мнению интеллекта, должны быть выключены. Или, может быть, скромное «предупреждение о вреде здоровью», если интеллект считает, что пользователь находится в слишком темной комнате. Если пользователь замечает уведомление и решает, что оно правильное, он может изменить освещение.

Все эти действия системы могут быть основаны на одном и том же интеллекте – он будет давать абсолютно одинаковые прогнозы в каждой конкретной ситуации, – но покажутся очень разными с точки зрения пользователя.

Так какой из типов поведения лучше?

Это зависит от целей системы и качества интеллекта. Если интеллект превосходен, то подойдет и автоматизация. Если интеллект новый (и пока не слишком точный), имеет смысл использовать уведомления. Один из ключевых принципов интеллектуального опыта – принять сторону пользователя и делать то, что ему нравится.

Достижение целей системы

Эффективный интеллектуальный опыт представит интеллект таким образом, чтобы достичь цели интеллектуальной системы. Опыт следует спроектировать таким образом, чтобы представление интеллекта помогало пользователям добиться хороших результатов (и заодно помогало бизнесу, который скрывается за интеллектом, достигать своих целей). Напомним, что цели интеллектуальной системы могут включать в себя:

- **организационные цели** – продажи и прибыль;
- **опережающие показатели** – вовлеченность и настроение;

- **результаты пользователя** – достижение целей пользователя;
- **свойства модели** – наличие точного интеллекта.

Задача интеллектуального опыта заключается в том, чтобы извлекать информацию из интеллектуальных ресурсов и использовать ее для достижения результатов пользователей, улучшения опережающих показателей и достижения организационных целей. Это означает, что когда интеллект прав, опыт должен подталкивать пользователей к действиям, которые достигают цели. А когда интеллект ошибается, опыт должен защищать пользователей от последствий ошибки.

Вернемся к системе автоматизации домашнего освещения. Если цель состоит в том, чтобы сэкономить электроэнергию; что должна делать система, когда считает, что надо выключить свет? Вот несколько вариантов поведения:

- **воспроизвести мягкий звук щелчка выключателя**. Если пользователь услышит звук, он подойдет к выключателю и выключит свет. Или не выключит…;
- **спросить пользователя**. Если мы спросим, хочет ли пользователь выключить свет, он может ответить «да» или сказать что-то другое (или пропустить вопрос);
- **спросить пользователя, а затем автоматически выполнить действие после задержки**. Дайте пользователю возможность остановить действие, но если отказ не поступил, выполните действие;
- **автоматизировать освещение** – просто выключить свет;
- **отключить главный рубильник дома** – потому что человек, который тратит энергию впустую, не ведает, что творит, и опасен для окружающих…

Эти варианты поведения обычно приводят к тому, что пользователь выключает свет (и оставляет его выключенным). Выбор слишком пассивного поведения сделает систему менее эффективной для достижения своей цели. Выбор слишком экстремального поведения сделает систему неприятной для использования.

На самом деле для достижения целей интеллектуальной системы опыт столь же важен, как и интеллект. Неудачный опыт может испортить систему так же сильно, как и ошибочные прогнозы интеллекта.

Интеллектуальный опыт волен выбирать решение или игнорировать бесполезный интеллект – в общем, делать все, что нужно для достижения желаемых результатов.

Пример достижения целей

Снова рассмотрим автоматическую систему освещения и две возможные цели системы:

1) минимизировать количество потребляемой электроэнергии;
2) свести к минимуму вероятность того, что пожилой человек споткнется обо что-то и получит травму.

Все остальное совпадает – интеллект, датчики, способ реализации системы; различаются только цели. Будет ли один и тот же опыт успешным для обеих этих целей? Скорее всего, нет.

Система, разработанная для экономии энергии, может никогда не включить свет, даже если интеллект считает, что это необходимо. Вместо этого опыт будет полагаться на пользователя, который сам включит свет, когда ему надо, и ограничится лишь тем, что станет гасить свет, когда интеллект считает, что свет больше не нужен.

Система, разработанная для безопасности пожилых людей, должна вести себя совсем иначе. Она будет включать свет всякий раз, когда подозревает, что в комнате кто-то есть, и оставлять его включенным до тех пор, пока интеллект не будет уверен, что комната опустела.

Эти варианты интеллектуального опыта настолько разные, что они могут продаваться как совершенно разные продукты – по-разному упаковываться, иметь разную цену, лежать на разных полках – и каждый из них может быть совершенно успешным в достижении своих стратегических целей. При этом они оба могут использовать один и тот же базовый интеллект и давать ему обратную связь.

Минимизация последствий ошибок интеллекта

Интеллект всегда делает ошибки. Много разных ошибок. Надежный интеллектуальный опыт будет работать, несмотря на эти ошибки, сведет к минимуму их последствия и облегчит пользователям возврат к нормальной деятельности.

При разработке интеллектуального опыта следует обдумать такие вопросы:

1. Какие ошибки допустит интеллект?
2. Как часто он будет совершать ошибки?
3. Во что обойдутся ошибки пользователю и организации:
   a) если пользователь замечает ошибку и пытается ее исправить;
   b) если пользователь не замечает, что произошла ошибка.

Затем интеллектуальный опыт должен решить, что делать с ошибками, например:

1) держаться подальше от серьезных ошибок, выбирая не слишком рискованные решения;
2) ограничить количество взаимодействий между пользователем и интеллектом, чтобы сократить количество ошибок, с которыми может столкнуться пользователь;
3) в ситуациях, когда ошибку трудно исправить, действовать менее жестко (например, предложить пользователю убедиться, что он хочет запустить ядерную ракету, вместо того чтобы запустить ее автоматически);
4) предоставить пользователю обратную связь о действиях, предпринятых системой, и рекомендации о том, как исправить неполадки.

Эти методы делают интеллектуальную систему более безопасной. Однако они снижают потенциал интеллектуальной системы, ослабляя взаимодействия и требуя от пользователя уделять больше внимания тому, что делает система.

Поиск оптимального соотношения между достижением цели и сглаживанием последствий ошибок является ключевой частью разработки надежного интеллектуального опыта.

Получение данных для расширения системы

Интеллекту нужны данные для роста. Ему нужно видеть примеры событий, происходящих в контексте системы, а затем анализировать последствия этих событий. Было ли взаимодействие полезным для пользователя? Для бизнеса? Это было хорошо или плохо? Интеллект использует примеры событий и последствий, чтобы улучшаться с течением времени.

Интеллектуальный опыт играет большую роль в создании ценности данных, поступающих из интеллектуальной системы. Правильно настроенный опыт может создать большие наборы данных, идеально подходящие для машинного обучения. И наоборот, непродуманный опыт может сделать данные бесполезными.

Надежный интеллектуальный опыт взаимодействует с пользователями заранее определенными способами, когда система может знать:

1) контекст взаимодействия;
2) действие, предпринятое пользователем;
3) итог взаимодействия.

Иногда опыт может сделать это взаимодействие полностью неявным. Опыт может быть спроектирован настолько утонченно, что пользователь создает полезные данные, просто пользуясь системой.

В других случаях опыт может требовать от пользователей явного участия в создании достоверных данных (например, выставлять оценки). Мы обсудим методы и компромиссы для получения качественных данных из интеллектуального опыта более подробно в следующей главе.

Пример сбора данных

Рассмотрим пример с автоматическим освещением.

Интеллект проверяет датчики и принимает решение о том, что делать с лампами. Опыт влияет на интеллект, взаимодействуя с пользователем. Затем нам нужно понять, было ли это взаимодействие правильным.

Что делать, если свет включен, но система считает, что он должен быть выключен? Как мы узнаем, прав интеллект или нет?

Интеллект может быть прав, если:

- опыт автоматически выключает свет, и пользователь не включает его снова;
- опыт предлагает пользователю выключить свет, и пользователь соглашается;
- опыт напоминает об экономии электричества, и пользователь выключает свет.

Интеллект может ошибаться, если:

- опыт автоматически выключает свет, а пользователь немедленно включает его снова;
- опыт предлагает пользователю выключить свет, но пользователь отказывается;
- опыт напоминает об экономии электричества, но пользователь не выключает свет.

Но эта обратная связь не совсем однозначная. Пользователи не обязательно будут исправлять ошибки. Например:

- они, возможно, хотят, чтобы лампы оставались включенными, но устали бороться с проклятым интеллектом и пытаются научиться видеть в темноте;
- они услышали рекомендацию погасить свет, но находятся на середине уровня в своей любимой игре и не могут ответить;
- они положили свои умные часы в ящик и поэтому больше не видят никаких сообщений.

Качество данных выше, если результаты взаимодействия однозначны и у пользователей есть стимул реагировать на каждое взаимодействие, которое инициирует опыт. Это не всегда легко и требует некоторых компромиссов. Но создание эффективных обучающих данных, когда пользователи взаимодействуют с интеллектуальным опытом, необходимо для достижения цели интеллектуальной системы.

Итог главы

В этой главе говорилось про интеллектуальный опыт. Роль интеллектуального опыта заключается в том, чтобы предоставлять пользователям возможности интеллекта с учетом целей системы, минимизировать недостатки интеллекта и создавать данные для его обучения.

Представление интеллекта включает в себя выбор того, где он появится, как часто он будет появляться и насколько влиятельно он будет себя вести. Выбор правильной комбинации подходов в конкретной ситуации имеет решающее значение для успеха интеллектуальной системы.

Правильно построенный опыт позволяет использовать единый интеллект для достижения самых разных целей.

Опыт помогает исправлять последствия ошибок и делает их менее дорогостоящими.

Эффективный опыт помогает развивать интеллект. Для этого отслеживают взаимодействие с пользователями, которые формируют множество примеров как положительных, так и отрицательных результатов.

Интеллектуальный опыт должен быть на стороне пользователя и делать его счастливым, вовлеченным и продуктивным независимо от того, насколько скудным или причудливым может быть интеллект.

Темы для размышлений

Прочитав эту главу, вы должны:

- понять, как интеллектуальный опыт и интеллект объединяются для получения желаемого эффекта;
- знать основные цели интеллектуального опыта и уметь объяснить некоторые проблемы, связанные с ними.

Постарайтесь ответить на следующие вопросы:

- вспомните о своем взаимодействии с интеллектуальными системами. Какое наиболее активное влияние интеллекта вы испытали? Какое влияние было наиболее пассивным?
- назовите три примера предоставленных вами данных, которые помогли улучшить интеллектуальные системы. Какие из них были наиболее эффективными и почему?

Глава 6

Затруднения при разработке интеллектуального опыта

В этой главе рассмотрены некоторые различия между опытом, основанным на интеллекте, и более традиционным опытом взаимодействия. В основе этих различий лежит простая истина: интеллект делает ошибки.

Интеллект обречен совершать ошибки; чтобы создать эффективный интеллектуальный опыт, вы должны смириться с этим фактом. Вам придется принять ошибки и научиться правильно работать с их последствиями.

Ошибки, которые совершает интеллект:

- не обязательно интуитивно понятные;
- не обязательно одинаковые изо дня в день;
- трудно предсказать заранее;
- невозможно исправить только за счет «дополнительной работы над интеллектом».

Эта глава посвящена ошибкам, которые совершает интеллект. Ее цель – познакомить вас с проблемами, которые необходимо решить, чтобы создать эффективный интеллектуальный опыт. Несмотря на ошибки, которые совершает интеллект, вы можете разработать интеллектуальный опыт, с которым пользователи и система достигают своих целей. Но для этого вам следует знать, чего ожидать от интеллекта, понимать, как он совершает ошибки, и представлять возможные компромиссы.

Интеллект делает ошибки

Интеллектуальные системы ошибаются, и обойти это невозможно. Ошибки будут неприятными, а некоторые из них окажутся по-настоящему плохими. Если оставить их без внимания, интеллектуальная система будет выглядеть глупой. Более того, ошибки могут сделать интеллектуальную систему бесполезной или опасной.

Вот несколько примеров ситуаций, которые могут возникнуть из-за ошибок в интеллекте:

- вы просите свой телефон заказать пиццу, а он сообщает вам население города Мумбаи в Индии;
- ваш беспилотный автомобиль сворачивает на несуществующую дорогу, и вы попадаете в озеро;
- ваш умный дверной звонок сообщает, что кто-то подошел к вашей входной двери, но, глядя на экран домофона, вы понимаете, что там никого нет;
- вы кладете кусок хлеба в свой умный тостер, возвращаетесь через пять минут и обнаруживаете, что хлеб холодный – тостер вообще ничего не сделал.

Эти и многие другие ошибки являются лишь частью платы за использование интеллекта (особенно интеллекта, созданного машинным обучением).

И эти ошибки не вина людей, разработавших интеллект. Точнее, это могут быть их ошибки – всегда есть вероятность, что люди плохо выполняют свою работу. Но даже профессионалы мирового уровня в области прикладного машинного обучения будут создавать интеллект, который делает ошибки.

Таким образом, одно из главных предназначений опыта в интеллектуальных системах состоит в том, чтобы интеллект был максимально эффективен, когда он прав, а ошибки, которые он совершает, сводились к минимуму и легко устранялись.

Учтите, что интеллект, который прав в 95 % случаев, совершает ошибку в одном из каждых двадцати взаимодействий. При этом 95 % считается весьма высоким результатом. Если ваши разработчики интеллекта достигнут точности 95 %, то решат, что трудная задача решена. Они захотят получить вознаграждение за свой тяжкий труд, а затем двигаться дальше и работать над другими проектами. Разработчикам вовсе не хочется слышать ваши претензии, что интеллект недостаточно точен.

Но в системе, которая ежедневно имеет миллион взаимодействий с пользователями, точность 95 % приводит к появлению 50 тысяч ошибок в день. Это очень много. Если из-за ошибок вашей системы пользователи потеряют время или деньги, у вас будут большие проблемы.

Еще один полезный способ оценить влияние ошибок – прикинуть количество взаимодействий, которые пользователи совершают между случаями ошибок. Например, при 20 взаимодействиях в день с точностью интеллекта 97 % пользователя будет ожидать 4,2 ошибки в неделю.

Это провал системы?

Ответ зависит от того, насколько серьезны ошибки и какими средствами для устранения последствий располагает пользователь.

Прежде чем создавать продукт, для которого требуется идеальный интеллект, попробуйте найти альтернативу такому интеллекту. Создание идеального интеллекта обходится очень дорого, а во многих случаях просто невозмож-

но. Для того чтобы получить реальную выгоду от интеллектуальных систем, не требуется совершенство. Сидеть в ожидании совершенства – отличный способ упустить большие возможности. Вот несколько примеров, когда не стоит ждать совершенства:

- распознавание речи не идеально и никогда таким не будет;
- поисковые системы не всегда дают правильный ответ;
- игровой интеллект обычно высмеивают за бег по кругу и стрельбу по стенам;
- беспилотные автомобили все равно попадают в аварии.

Но все эти ошибочные интеллекты являются частью продуктов, которые мы используем каждый день. Они делают нашу жизнь лучше. Сглаживание ошибок – фундаментальная задача при разработке интеллектуального опыта.

Безумные ошибки интеллекта

Пытаясь осмыслить происходящее, интеллект делает безумные ошибки. Дикие, необъяснимые, нелогичные ошибки.

Задумайтесь: если эксперт-человек прав в 99 % случаев, вы ожидаете, что и его ошибки будут подчиняться здравому смыслу. Вы предполагаете, что эксперт хорошо понимает, что происходит, и если он ошибся – ну что поделать, его ошибка все равно лежит в пределах разумного отклонения и не противоречит устройству мира.

Машинное обучение и искусственный интеллект совсем не такие.

Искусственный интеллект может быть прав в 99,9 % случаев, а затем один раз из тысячи сотворить несусветную чушь. Например:

- система, которая подбирает музыку, может рекомендовать вам 999 рок-песен, а затем предложить модный трек для подростков;
- умный тостер может идеально поджарить 99 кусков хлеба и решить, что сотый кусочек хлеба не требует поджаривания, – просто так, несмотря ни на что;
- система, которая считывает человеческие эмоции по изображениям, может правильно определить 999 человек как счастливых, а потом посмотреть на ваше лицо и сказать, что вам грустно, невзирая на улыбку до ушей.

Эти глупые ошибки являются частью работы интеллектуальных систем.

Даже самые безумные ошибки могут быть сглажены, исправлены или сведены к минимуму. Но для разработки опыта, который работает с интеллектом, вы должны отбросить предвзятые человеческие представления о правильном и рациональном. Вместо этого вам следует развивать интуитивное понимание интеллекта, с которым вы работаете. Прислушайтесь к интуиции. Попробуйте предположить, что и как может ввести интеллект в заблуждение. Подумайте, как этого избежать.

Инстинкт отрицания ошибок заставляет вас говорить разработчикам интеллекта: «Да вы в своем уме? Исправьте эту ошибку. Она погубит нас!»

Успокойтесь. Возможно, разработчики смогут изменить интеллект, чтобы исправить ошибку, которая вас беспокоит.

Но исправление одной ошибки обычно приводит к появлению новой ошибки в другом месте. Эта новая ошибка, вероятно, столь же безумная. И вы не будете знать заранее, когда или как проявится новая ошибка. Исправлять очевидные ошибки не всегда правильно; на самом деле игра в охоту за ошибками может навредить. Иногда лучше позволить интеллекту оптимизироваться как можно лучше, а затем поддерживать его в таком состоянии, компенсируя ошибки безупречным опытом.

Интеллект совершает разные ошибки

Хотя может показаться, что я слишком усложняю, интеллектуальные системы допускают различные типы ошибок, например:

- умный дверной звонок сообщает, что кто-то приближается к двери, когда на самом деле никого нет, или молчит, когда кто-то приближается к двери;
- умный тостер может недожарить или пережарить хлеб;
- система распознавания речи может неправильно интерпретировать то, что сказал пользователь, или вообще отказаться считать эти звуки речью.

Вот три основных типа ошибок интеллекта:

1) ошибочно принимает одну ситуацию за другую;
2) неправильно оценивает некоторое значение (например, время обжарки тоста);
3) слишком неуверен (или консервативен), чтобы говорить что-либо вообще в сложной ситуации.

Пожалуй, самая простая форма ошибки – путать одну ситуацию с другой, как в примере с умным дверным звонком. Для интеллекта такой системы существует четыре возможных ответа (табл. 6.1):

- **истинно положительный** (true positive) – когда кто-то стоит у двери, и интеллект полагает, что кто-то стоит у двери, это называется *истинно положительным решением* (это правильный ответ, а не ошибка);
- **истинно отрицательный** (true negative) – когда рядом с дверью никого нет, и интеллект полагает, что там никого нет, это называется *истинно отрицательным решением* (это другой тип правильного ответа, а не ошибка);
- **ложноположительный** (false positive) – когда рядом с дверью никого нет, но интеллект полагает, что кто-то стоит у двери, это называется *ложноположительным решением* (это ошибка);

- **ложноотрицательный** (false negative) – когда кто-то стоит у двери, но интеллект полагает, что там никого нет, это называется *ложноотрицательным решением* (это еще один тип ошибки).

Таблица 6.1. Различные варианты ответов интеллекта в системе умного дверного звонка

		Интеллект полагает, что кто-то	
		присутствует	отсутствует
На самом деле кто-то	присутствует	Истинно положительный	Ложноотрицательный
	отсутствует	Ложноположительный	Истинно отрицательный

Зачастую интеллект системы настраивают так, чтобы он сам следил за типом ошибок, которые совершает, допуская больше ошибок одного типа и сводя к минимуму ошибки остальных типов. Обычно этого добиваются подбором порога вероятности на выходе модели, которая управляет интеллектом.

Например, если вы разрабатываете умный дверной звонок, что будет лучше – чтобы система оповещала пользователя, когда у двери никого нет, или не оповещала пользователя, когда кто-то стоит у двери?

А что, если можно избавиться от трех появлений ошибки первого типа ценой добавления одной ошибки второго типа? Будет ли это хорошим компромиссом для пользователя? Поможет ли это в достижении целей системы?

Это будет зависеть от опыта взаимодействий. Какой тип ошибок сильнее раздражает пользователя? Какой тип ошибки легче скрыть? Какой тип ошибки больше мешает достижению целей системы? И сколько раз в день/месяц/год пользователь будет сталкиваться с каждой из этих ошибок? Будет ли он утомлен и вообще перестанет реагировать на предупреждения системы? Или он решит не доверять системе и выключит ее?

Возможны и другие варианты компромиссов:

- между недооценкой и переоценкой. Например, когда умный тостер предсказывает длительность обжарки, что будет лучше – пережарить на 5 % или недожарить на 5%?
- между вариантами выбора опций, которые дает интеллект. Например, рекомендательная система предполагает, что пользователь захочет прочитать одну из 15 книг. Что будет лучше – позволить пользователю выбрать из топ-5 или отобразить все 15 книг?
- между интеллектом, который что-то делает и ничего не делает. Например, в системе компьютерного зрения интеллект может быть на 90 % уверен, что знает, кто находится на изображении. Что будет лучше – чтобы интеллект помечал лицо на изображении именем, которое будет не-

правильным в одном из 10 случаев? Или лучше, чтобы интеллект промолчал? Или лучше попросить пользователя помочь?

Переменчивый интеллект

Традиционный опыт взаимодействия является *детерминированным*. Он совершает одно и то же действие в ответ на одну и ту же команду. Например, когда вы щелкаете правой кнопкой мыши по имени файла, каждый раз появляется меню (если в программе нет ошибки). Детерминизм хорош по многим причинам.

Интеллектуальные системы совершенствуются со временем. Вчера они совершали одно действие, а сегодня совсем другое. И это может быть очень хорошим свойством.

Представьте ситуации:

- вчера вы искали информацию о своей любимой группе и получили несколько хороших ссылок на статьи. Сегодня вы снова ищете информацию о любимой группе и получаете новую, более качественную подборку ссылок;
- вчера вы приготовили попкорн в микроволновке, и несколько кусочков сгорело. Сегодня вы снова готовите попкорн, и микроволновка нагревает его чуть меньше, так что попкорн получается идеально;
- вчера ваше навигационное приложение заметило, что вы проезжали через зону дорожных работ, и ваша поездка заняла на час больше, чем обычно. Сегодня приложение направляет вас по другому маршруту, и вы приходите на работу более счастливый и продуктивный, чем вчера.

Это примеры положительных изменений, которые являются целью интеллектуальных систем – с точки зрения пользователя система сегодня работает лучше, чем вчера. Если что-то работает не очень хорошо, пользователь должен верить, что со временем система станет лучше.

Но изменения могут быть и разрушительными. Даже положительные изменения, нацеленные на долгосрочную перспективу, могут вызывать у пользователей замешательство и ощущение потери контроля над происходящим. Представьте себе ситуации:

- вчера вы делали заметки с помощью умной ручки, и чернила были приятного оттенка синего цвета. Но однажды ручка подключилась к своему серверу, и разработчики, оптимизирующее что-то, чего вы можете и не знать, решили поменять оттенок синего. Сегодня ваши заметки имеют другой цвет;
- вчера вы пошли в местный магазин, купили большой прохладный стакан своей любимой газировки и выпили его с огромным удовольствием. Ночью газировочный автомат подключился к своему серверу и обновил рецепт. Сегодня вы купили еще один стакан газировки, и вам она не понравилась.

Эти изменения хороши? Кто знает... Возможно, они улучшают некоторые очень важные показатели производственных предприятий. Может быть, постепенно вы поймете, что вам действительно понравились изменения, даже если поначалу отвергали их.

Или, возможно, изменения просто катастрофичны.

Когда имеешь дело с меняющимся интеллектом, необходимо учитывать следующее.

1. **Можно ли ограничить скорость изменения?** Это можно сделать наложением некоторых ограничений при разработке интеллекта. Например, пусть интеллект изменяет свои прогнозы не более чем для 5 % взаимодействий в день. Слишком сильные ограничения могут привести к застою интеллекта и утрате возможности улучшения интеллекта с течением времени, но некоторые простые ограничения могут быть полезными.
2. **Может ли опыт помочь пользователю ориентироваться в изменениях?** Возможно, он уже делает это, сообщая пользователю, что произошли изменения. Или сохраняет старое поведение, но предлагает пользователю новое поведение в качестве опции. Например, умная ручка сообщает пользователю, что она нашла новый оттенок чернил, который делает заметки комфортнее на 5 %, и предлагает принять изменения, если пользователь пожелает, или оставить прежний оттенок чернил.
3. **Может ли опыт ограничить влияние изменяющихся частей системы**, в то же время позволяя им иметь достаточную значимость для достижения результата? Например, использовать более пассивный опыт с теми частями интеллекта, которые меняются больше всего?

Человеческий фактор

Интеллектуальный опыт преуспевает благодаря позитивному взаимодействию со своими пользователями, делает их счастливее, эффективнее и помогает им работать более продуктивно.

Но взаимодействие с интеллектуальными системами бросает вызов стереотипам поведения и может вызывать стресс у некоторых пользователей.

Люди имеют дело с инструментами, книгами, машинами, прочими предметами. Все эти вещи ведут себя предсказуемым образом. В течение долгого времени мы развивались вместе с ними, рассчитывали на них и знали, чего ожидать. Иногда они ломаются, но это случается редко. В общем, они такие, какие есть, – мы учимся их использовать, а затем перестаем думать о них.

Инструменты становятся, в некотором смысле, частью нас самих и дают нам способности, которых у нас не было от природы. Они заставляют нас чувствовать себя очень уверенно, безопасно и комфортно.

Интеллектуальные системы не совсем такие.

Интеллектуальные системы делают ошибки. Они изменяют свои «умы» на ходу. Принимая решение, они учитывают самые тонкие факторы. Иногда они

отказываются делать то же самое два раза подряд, хотя пользователь не понимает, что изменилось. Иногда они даже имеют свои собственные побудительные мотивы, которые не совсем соответствуют мотивам их пользователей.

Взаимодействие с интеллектуальными системами выглядит скорее как человеческие отношения, чем использование инструмента. Вот несколько причин, по которым это беспокоит пользователей:

- **путаница** – когда интеллектуальная система действует странным образом или совершает ошибки, пользователи будут сбиты с толку. Им придется потратить свои силы и время на понимание того, что происходит;
- **раздражение** – когда интеллектуальная система влияет на действия пользователя, понравится ли это пользователю? Например, по мнению пользователя, система ставит интересы других людей выше интересов пользователя, показывая ему навязчивую рекламу;
- **недостаток уверенности** – доверяет ли пользователь системе в достаточной степени, чтобы позволить ей делать свое дело, или же он пришел к выводу, что система неэффективна, и хотя пытается быть полезной, но всегда делает это неправильно?
- **утомление** – когда система требует внимания пользователя, обладает ли она чувством меры или надоедает пользователю вопросами? Пользователи умеют игнорировать то, что им не нравится;
- **скрытность** – будут ли взаимодействия вызывать у пользователя дискомфорт? Может быть, система чересчур хорошо знает своих пользователей. Возможно, она заставляет их делать то, чего они не хотят делать, или размещать конфиденциальную информацию на публичных форумах. Если умный телевизор видит, как парочка обнимается на диване, он может пригасить свет и сыграть музыку Барри Уайта (Barry White), но стоит ли это делать?

Всегда учитывайте, что пользователи думают об интеллектуальном опыте. Уважайте их чувства, правильно выбирайте состав и силу взаимодействий.

Итог главы

Интеллект делает ошибки, и жаловаться бесполезно. Если вы не нашли в своей системе ошибки, значит, вы их плохо искали. Но погоня за ошибками, эта бесконечная ловля блох, иногда бывает вреднее, чем сами ошибки.

Ошибки, которые делает интеллект, могут быть совершенно нелогичными. Даже интеллект, который обычно не ошибается, может внезапно совершить очень странную ошибку, которую никогда не сделает ни один человек.

Есть разные типы ошибок. В наиболее общем виде их можно назвать так: ложноположительные (интеллект считает, что что-то произошло, когда этого не было) и ложноотрицательные (интеллект считает, что ничего не произошло, хотя это случилось). Интеллект часто может найти компромисс между этими типами ошибок (например, увеличить вероятность ложноположитель-

ных решений на выходе модели, чтобы уменьшить количество ложноотрицательных решений).

Изменения интеллекта могут быть неудачными с точки зрения пользователей, даже если в долгосрочной перспективе они сулят выгоду. Интеллектуальные системы постоянно меняются. Помочь пользователям справиться с изменениями – это часть работы по созданию эффективной интеллектуальной системы.

Интеллектуальные системы могут отрицательно влиять на пользователей, что вызывает у них неприязнь и недоверие к системе.

Создать интеллектуальный опыт нелегко.

Темы для размышлений

Прочитав эту главу, вы должны:

- понимать, что интеллект совершает ошибки, – это одна из фундаментальных проблем, которую должен решать интеллектуальный опыт;
- понимать, почему эти проблемы не относятся к недоработкам интеллекта (которые должны быть исправлены), а являются неотъемлемой частью работы с интеллектом (и должны быть приняты);
- иметь представление о том, как пользователи будут воспринимать недостатки интеллектуальной системы, и создавать опыт, помогающий пользователям.

Постарайтесь ответить на следующие вопросы:

- какую худшую ошибку вы встречали в интеллектуальной системе?
- каким образом интеллектуальный опыт мог сделать ошибку менее серьезной?
- на какую смену продукта у вас была самая острая реакция (сначала обновление вам не понравилось, но со временем стало нравиться больше)? Почему?

Глава 7

Разработка эффективного интеллектуального опыта

Разработка эффективного интеллектуального опыта – это поиск оптимального отношения между следующими факторами:

1) достижение желаемого результата;
2) защита пользователей от ошибок;
3) получение данных для развития интеллекта.

Когда интеллект работает правильно, система должна руководить, создавать ценность, автоматизировать, давать пользователю выбор, который ему нужен, и поощрять самые безопасные и выгодные действия пользователя. Опыт должен побуждать пользователя извлекать выгоду из правильных решений интеллекта (когда пользователь не раздражен и осознает пользу от взаимодействия).

Когда интеллект ошибается, система должна минимизировать ущерб. Например, разрешить пользователю отменять любые предпринятые действия или предоставить возможности поиска дополнительных параметров. Возможно, придется рассказать пользователю, почему он совершил ошибку, или оказать ему помощь. Благодаря этому пользователь сможет избежать аналогичной ошибки в будущем.

Проблема в том, что опыт не знает заранее, прав интеллект или ошибается. И поэтому каждый аспект опыта следует рассматривать с двух сторон – что делать пользователю, если интеллект был прав, и что должен делать пользователю, если интеллект ошибался.

Здесь возникает противоречие, потому что вещи, которые наделяют интеллектуальную систему волшебством (например, автоматизация действий без участия пользователя), расходятся с вещами, которые защищают пользователя от ошибок (например, необходимость подтвердить решения до того, как интеллект начнет действовать).

Есть пять основных факторов, которые влияют на эффективность интеллектуального опыта:

- *действенность опыта* (forcefulness of the experience) – насколько сильно он побуждает пользователя делать то, что, по мнению интеллекта, он должен делать;
- *частота взаимодействия* (frequency of the experience) – как часто интеллектуальный опыт пытается взаимодействовать с пользователем;
- *выгода от взаимодействия* (value of the interaction), когда интеллект прав, – насколько высоко пользователь оценивает свою выгоду, и насколько это помогает интеллектуальной системе достигать своих целей;
- *цена взаимодействия* (cost of the interaction) при ошибке интеллекта – какой ущерб нанесла ошибка, и насколько трудно пользователю заметить и устранить ущерб;
- *качество интеллекта* (quality of the intelligence) – как часто интеллект прав, и как часто он ошибается.

Чтобы создать качественный интеллектуальный опыт, вы должны понимать эти факторы и их взаимосвязь. Затем вы должны принять сторону пользователей и создать опыт, который эффективно связывает их с интеллектом. В этой главе мы рассмотрим факторы эффективности более подробно.

Действенность опыта

Взаимодействие является *активным*, если пользователю трудно игнорировать или остановить его. Взаимодействие является *пассивным*, если оно не обязательно привлечет внимание пользователя или повлияет на него. Например, *активный опыт* может:

- автоматизировать действие;
- прервать действия пользователя и заставить его ответить на запрос, прежде чем он сможет продолжить;
- показывать большое и яркое всплывающее окно перед пользователем каждые несколько секунд, пока он не ответит.

Активные взаимодействия эффективны, когда:

- система уверена в качестве интеллекта (то есть вероятность правоты намного выше, чем вероятность ошибки);
- система действительно хочет привлечь внимание пользователя;
- ценность успеха значительно выше стоимости ошибки;
- есть высокая потребность в отзыве пользователя о решении интеллекта (чтобы помочь создать новый интеллект).

Пассивный опыт не требует внимания пользователя. Пользователю легко решить, хочет ли он общаться с пассивным опытом. Пассивный опыт включает в себя:

- деликатные подсказки, не заставляющие пользователя отвечать немедленно;
- небольшой значок в углу экрана, который пользователь может заметить или не заметить;

- список рекомендаций в нижней части экрана, по которому пользователь может щелкнуть или проигнорировать его.

Пассивные взаимодействия эффективны, когда:

- система не уверена в качестве интеллекта;
- система не уверена, что ценность интеллектуального взаимодействия выше, чем текущие действия пользователя;
- стоимость ошибки высока.

Один из примеров силы взаимодействия – это реклама на веб-странице. Если объявление всплывает на странице и не позволяет вам продолжить чтение, пока вы не нажмете на рекламу, – это сильный опыт. Вы, вероятно, нажмете на глупое объявление (потому что некуда деваться). И если реклама неинтересна, вы будете злиться на рекламируемый продукт, на веб-страницу, которая показала объявление, и на разработчика глупого браузера, в который пролезает такая назойливая реклама.

С другой стороны, если объявление деликатно и со вкусом вписано в страницу, вы можете его не заметить. Вы можете нажать на объявление, а можете и не нажать. Если рекламируют что-то удивительное, вы можете заинтересоваться. Но, в любом случае, вы меньше подвержены раздражению, поэтому можете вернуться на страницу снова и снова. Вы будете все дольше задерживаться на странице, и количество нажатий на скромные объявления окажется больше, чем на яркие и назойливые баннеры.

Частота взаимодействия

Интеллектуальный опыт может решить взаимодействовать с пользователем или отказаться от взаимодействия. Например, представьте себе приложение умной карты, которое прокладывает маршруты. Эта карта располагает данными об условиях дорожного движения, авариях и погодных условиях в миллионах точек, которые светятся на карте разными цветами. Всякий раз, когда вы приходите на перекресток, приложение может сказать «поверните налево, чтобы сэкономить 17 секунд», или «идите быстрее, чтобы успеть к следующему светофору и сэкономить 3 минуты», или «нажмите кнопку U, вернитесь к последнему перекрестку и поверните направо, КАК Я УЖЕ ГОВОРИЛ ВАМ, и вы еще успеете сэкономить 2 минуты».

Или приложение может вести себя более скромно, ограничиваясь одной рекомендацией за маршрут и предлагая только те маршруты, которые экономят не менее 10 минут.

Частые взаимодействия, как правило, утомляют пользователей, особенно если выглядят как принуждение. С другой стороны, редкие взаимодействия имеют меньше шансов помочь пользователям и могут сбивать с толку (если пользователи к ним не привыкли).

Рассмотрим некоторые варианты частотности взаимодействий.

1. *Взаимодействовать всякий раз, когда интеллект думает, что у него другое решение.* Например, пятнадцать секунд назад интеллект думал, что

правильное направление – прямо, а теперь у него больше информации и он думает, что правильное направление – направо. Но через десять секунд он снова начинает думать, что вы должны идти прямо. Буквальное использование интеллектуального вывода может привести к очень частым взаимодействиям. Данный вариант может быть эффективным, если интеллект редко ошибается и взаимодействие не требует слишком большого внимания пользователя. Иначе вы можете свести пользователей с ума.

2. *Взаимодействовать всякий раз, когда интеллект думает, что у него существенно другое решение.* То есть взаимодействовать только тогда, когда интеллект создаст «новую ценность» для пользователя. Это компромисс между потенциальной ценностью системы и частотой обращений к пользователю. А критерий «новой ценности» можно со временем настроить, чтобы отрегулировать количество взаимодействий.
3. *Явно ограничить частоту взаимодействия.* Например, вы можете разрешить одно взаимодействие в час или десять взаимодействий в день. Это может сработать, если вы не знаете, как пользователи будут реагировать на запросы. Такой подход позволяет ограничить потенциальные расходы при сборе данных для проверки и улучшения интеллектуальной системы.
4. *Взаимодействовать всякий раз, когда интеллект думает, что пользователь ответит.* Этот подход предполагает наличие у вас некоторых представлений о пользователях. Им нравится взаимодействие, которое вы предлагаете? Они склонны принимать или игнорировать взаимодействие? Они заняты чем-то еще или сосредоточены на взаимодействии? Они устают от взаимодействий? Такой подход может хорошо работать с точки зрения пользователей, сочетая интеллектуальную систему с их образом жизни. Но его труднее реализовать, и всегда остается риск непонимания. Например, сегодня у пользователя в машине сидит девушка, поэтому пользователь игнорирует обращения системы и сосредоточен на подруге. Отсюда интеллектуальный опыт делает ошибочный вывод, что пользователю вообще не нравится взаимодействие. Он перестает предлагать улучшенные варианты маршрута. Пользователь теряет ценность системы, потому что система пыталась быть слишком деликатной.
5. *Взаимодействовать, когда пользователь просит об этом.* То есть не приставайте, пока пользователь явно не инициирует взаимодействие. Такой подход очень эффективно снижает утомляемость пользователя. Взаимодействие по инициативе пользователя – это хороший ограничитель системы, позволяющий получать информацию или действия строго в нужный момент. С другой стороны, слишком сильная зависимость от этого способа может значительно снизить потенциал интеллектуальной системы – что, если пользователь никогда не запросит взаимодействие? Как пользователь узнает или запомнит, что надо попросить о взаимодействии?

И не забывайте, что люди устают от взаимодействий. Человеческий мозг очень хорошо умеет игнорировать раздражители, которые создают шумовой фон. Если ваш опыт слишком часто взаимодействует с пользователями, они начнут его игнорировать. Будьте осторожны с частыми взаимодействиями – иногда «меньше» означает «лучше».

Выгода от взаимодействия

Пользователи будут охотнее мириться с ошибками и проблемами, если они чувствуют, что получают выгоду. Когда интеллектуальная система помогает с чем-то явно важным, например с критичной для жизни проблемой, или экономит пользователям большое количество денег и времени, пользователи охотно соглашаются уделять свое время для настройки интеллектуального опыта. Когда интеллектуальная система создает небольшую ценность, например помогает пользователям поджаривать хлеб или экономить несколько копеек в день, пользователи менее охотно участвуют в настройке интеллектуального опыта и сомневаются, что оно того стоит. Пользователи склонны стремиться к взаимодействию, если они:

- получают информацию о происходящем;
- чувствуют, что взаимодействие решило важную проблему, или думают, что оно дало им некоторое значимое (пусть даже косвенное) улучшение;
- могут связать результат со взаимодействием, то есть понимают, что пыталось сделать взаимодействие и что оно сделало;
- верят, что система на их стороне, а не просто работает на благо кого-то другого;
- считают, что система крутая, умная или удивительная.

Может случиться так, что взаимодействие оставит пользователей в лучшем положении, но будет иметь нулевую воспринимаемую ценность. И наоборот, пользователи могут оказаться в худшем положении, но будут думать, что интеллектуальная система великолепна. Интеллектуальный опыт играет большую роль, помогая пользователям чувствовать, что они получают выгоду от интеллектуальной системы.

Когда интеллект работает верно, эффективный интеллектуальный опыт будет заметен, он будет радовать пользователей своей работой и ставить это себе в заслугу. Но продуманный интеллектуальный опыт будет осторожным – интеллект не всегда прав, и нет ничего хуже, чем интеллектуальный опыт, который гордится своей работой, когда он на самом деле ухудшил вашу жизнь.

Взаимодействия должны быть полезны и для интеллектуальной системы. В целом взаимодействие полезно для интеллектуальной системы, если оно достигает целей системы, например:

- заставляет пользователя больше использовать интеллектуальную систему (*повышает заинтересованность*);
- заставляет пользователя лучше относиться к интеллектуальной системе (*улучшает настроение*);

- заставляет пользователя тратить больше денег (*создает доход*);
- создает данные, которые помогают улучшить интеллектуальную систему (*улучшает интеллект*).

Опыт, который откровенно стремится зарабатывать деньги и получать данные от пользователей, будет противоречить ощущению выгоды пользователя. Качественный интеллектуальный опыт будет гибким и способным на компромиссы, чтобы все участники взаимодействия смогли извлечь выгоду.

Цена взаимодействия

Пользователям не нравятся проблемы, но интеллектуальные системы не обходятся без проблем. Качественный интеллектуальный опыт будет следить за тем, во сколько обходятся пользователю ошибки интеллекта, и сделает все возможное, чтобы минимизировать эти затраты.

Ошибки имеют собственную стоимость, зависящую от типа ошибки. Например, ошибка, которая угрожает человеческой жизни или стоит больших денег и времени, сама по себе очень дорога. А ошибка, которая делает гриль на несколько градусов холоднее, чем хотел пользователь, стоит не так уж дорого.

Но большинство ошибок можно исправить. А ошибки, которые легко исправить, обходятся дешевле, чем ошибки, которые трудно (или невозможно) исправить. Интеллектуальный опыт помогает пользователям *заметить, когда произошла ошибка*, и предоставляет хорошие варианты *восстановления после ошибок*.

Иногда ошибки довольно незначительны, и в этом случае пользователи могут даже не заметить, что произошла ошибка. Или же ошибку невозможно исправить, и в этом случае интеллектуальная система может сделать вид, что не случилось ничего плохого – нет смысла плакать над пролитым молоком.

Обнаружение ошибки

Первый шаг к решению проблемы состоит в том, чтобы узнать, что есть проблема. Качественный интеллектуальный опыт в части обнаружения ошибок работает следующим образом:

1) не беспокоит пользователя раньше времени, особенно когда оказывается, что на самом деле это была не ошибка;
2) находит ошибку, пока есть время для исправления последствий;
3) заставляет пользователя испытывать больше оптимизма по поводу ошибки и качества взаимодействия.

Иногда ошибки очевидны, например когда интеллектуальный опыт выключает свет, оставляя пользователя в темноте. Пользователь понимает, что система ошиблась, как только гаснет лампа. Но иногда ошибки не очевидны, например когда интеллектуальный опыт меняет настройки компьютера без согласования с пользователем. Пользователь может годами работать с неоптимальными настройками и никогда не узнать о проблеме. Вот два основных подхода, которые помогут пользователям выявлять ошибки:

- ***информирование пользователя***, когда интеллектуальный опыт вносит изменения. Например, интеллектуальный опыт может сработать автоматически, но при этом показать деликатное уведомление. Настойчивые уведомления хорошо привлекают внимание пользователей и дают возможность найти ошибки. Но при этом они утомляют пользователей и должны использоваться с осторожностью;
- ***ведение журнала действий***, предпринятых интеллектуальным опытом. Например, папка «спам» в системе фильтрации спама представляет собой журнал сообщений, заблокированных фильтром спама. Он позволяет отслеживать ошибки, но не требует, чтобы пользователи следили за каждым взаимодействием. Обратите внимание, что журнал не обязательно должен быть полным – он может содержать записи только о тех взаимодействиях, в которых интеллект не уверен.

Исправление ошибки

Интеллектуальный опыт также помогает пользователям *исправить ошибки.*

Некоторые ошибки легко исправить – например, когда свет выключается, пользователь может включить его снова. А некоторые ошибки гораздо труднее исправить, например когда интеллектуальный опыт отправляет электронное письмо от имени пользователя – как только текст сообщения прочитан получателем, пути назад нет.

При исправлении ошибок возникают два ключевых вопроса: какую часть стоимости ошибки можно возместить и что пользователь должен сделать, чтобы исправить ошибку. Чтобы уменьшить негативные последствия ошибки, интеллектуальный опыт может:

1) ограничить область принятия решения – например, выполнить действие частично, чтобы проверить, не возражает ли пользователь;
2) задержать действие – например, предоставить пользователю время подумать над своим решением, прежде чем предпринимать разрушительные действия (например, удалить все свои файлы);
3) вообще не предпринимать разрушительных действий – например, ограничиться только действиями, которые можно отменить.

Если цена ошибок достаточно высока, опыт может пойти на многое, чтобы помочь исправить ошибки.

Когда пользователь хочет устранить последствия ошибки, интеллектуальный опыт может:

1) предложить опцию в меню для отмены действия (с помощью одной команды);
2) вынудить пользователя пошагово отменить действия вручную (отслеживая все действия и отменяя различные шаги по очереди);
3) предоставить возможность обратиться к сотруднику поддержки (когда действие может быть отменено только администратором системы).

Лучшие ошибки – это те, которые можно легко заметить, не требующие пристального внимания пользователя и легко исправляемые за одно взаимодействие.

Наихудшими являются те ошибки, которые трудно найти, исправимые лишь частично и требующие больших затрат времени пользователя для исправления (а владелец интеллектуальной системы должен оплатить работу вспомогательного персонала).

Качество интеллекта

Опыт не может непосредственно влиять на качество интеллекта. Но чтобы разработать хороший интеллектуальный опыт, вы должны понимать, по каким критериям определять качество интеллекта. Когда интеллект работает хорошо, можно предложить пользователю более частые и сильные взаимодействия. Когда интеллект нестабилен, вы должны быть осторожны при взаимодействии с пользователем.

В основном интеллект совершает ошибки двух типов. Интеллект может:

- делать много *случайных ошибок* (random mistakes). Например, он может иметь правильный ответ в 90 % случаев и неправильный в 10 % случаев. В этом случае опыт может быть настроен для всех пользователей одновременно;
- делать *сфокусированные ошибки* (focused mistakes), когда одни пользователи получают правильный ответ с большей вероятностью, чем другие. Скажем, интеллект чаще ошибается в отношении пользователей, которые носят очки. Сфокусированные ошибки встречаются довольно часто и с трудом поддаются обнаружению. Когда сфокусированные ошибки становятся проблемой, возможно, придется настроить интеллектуальный опыт для наименьшего общего знаменателя – пользовательской категории, которая демонстрирует худшие результаты (или отказаться от некоторых пользователей и сосредоточиться на тех, у которых интеллект работает хорошо).

Качество интеллекта совершенствуется на протяжении всего жизненного цикла, и опыт должен адаптироваться так же, как и интеллект. Например, на ранних стадиях разработки, до появления у системы реальных клиентов, интеллект, как правило, будет слабым. В некоторых случаях интеллект будет настолько слабым, что наилучшим решением станет полное сокрытие интеллектуального опыта от пользователей (на время сбора обучающих данных).

По мере развертывания системы и прихода клиентов система будет собирать данные, чтобы улучшить интеллект. А по мере улучшения интеллекта у опыта будет больше возможностей выстроить оптимальное взаимодействие и достичь целей интеллектуальной системы. Имейте в виду следующее:

1. Лучший интеллект поддерживает более сильные и частые взаимодействия.

2. Изменения не всегда даются легко (даже если изменения основаны на улучшенном интеллекте). Будьте осторожны, чтобы не растерять своих пользователей.
3. Иногда интеллект меняется в худшую сторону – например, если проблема становится сложнее или у системы появляются новые типы пользователей (или если создатели интеллекта совершают ошибки). Иногда опыт должен сделать шаг назад в интересах всей системы. Это нормально – в жизни случается всякое.

Итог главы

Эффективный интеллектуальный опыт поддерживает разумный баланс между:

- ***силой*** взаимодействия с пользователями;
- ***частотой*** взаимодействия с пользователями;
- ***выгодой*** успешного взаимодействия для пользователя и для интеллектуальной системы;
- ***ценой*** ошибок для пользователя и для интеллектуальной системы;
- ***качеством*** интеллекта.

Пользователи и интеллектуальная система должны заключить взаимовыгодную сделку. С точки зрения пользователей, выгода от системы должна окупать затраченное на взаимодействие время. Для интеллектуальной системы выгода заключается в том, что система должна достигать целей и получать данные для улучшения.

Когда интеллект работает безошибочно, интеллектуальный опыт может совершать частые и активные взаимодействия, создавая большую ценность для всех и не делая много ошибок.

Когда интеллект недостаточно хорош (или цена ошибок велика), интеллектуальный опыт должен быть осторожнее в отношении того, какие действия он предлагает и как он их предлагает, а у пользователей должна быть возможность найти и исправить ошибки.

Взаимодействие с пользователем можно ограничить ситуациями:

- всякий раз, когда интеллект меняет свое решение;
- всякий раз, когда интеллект находит значительное улучшение;
- фиксированное количество раз;
- только когда система уверена, что пользователь ответит;
- только когда пользователь инициирует взаимодействие.

Частые взаимодействия создают больше возможностей для создания взаимной выгоды. Но взаимодействия могут также вызвать усталость пользователя и привести к тому, что он будет игнорировать интеллектуальный опыт.

Пользователи ощущают ценность взаимодействия, когда понимают, что происходит, и чувствуют, что заключили выгодную сделку. Интеллектуальный опыт крайне важен для того, чтобы помочь пользователям ощутить ценность взаимодействия.

Интеллект будет ошибаться; чтобы создать эффективный интеллектуальный опыт, вы должны понимать, как он ошибается, и работать над его развитием. Качество интеллекта будет меняться в течение жизненного цикла интеллектуальной системы, и когда это случается, интеллектуальный опыт можно перенастраивать для повышения ценности взаимодействия.

Темы для размышлений

Прочитав эту главу, вы должны:

- знать факторы, между которыми следует найти баланс, чтобы создать эффективный интеллектуальный опыт – такой, который приятен в использовании, достигает целей, смягчает ошибки и развивается вместе с интеллектом;
- знать, как оптимизировать интеллектуальный опыт, опираясь на ключевые факторы: силу, частоту, выгоду, цену и качество интеллекта.

Постарайтесь ответить на следующие вопросы:

- каков самый активный интеллектуальный опыт, с которым вы когда-либо общались? Какой самый пассивный интеллектуальный опыт вы можете вспомнить?
- приведите пример интеллектуального опыта, который, по вашему мнению, был бы более ценным, если бы реже взаимодействовал с пользователем. Почему?
- перечислите три способа, с помощью которых интеллектуальный опыт помог вам найти ошибки. Что было наиболее эффективным и почему?

Глава 8

Режимы интеллектуального взаимодействия

Существует множество подходов к созданию пользовательского опыта, и большинство из них можно реализовать на основе интеллекта. В этой главе рассматриваются некоторые общие подходы к взаимодействию между интеллектом и пользователями и обсуждается, как эти подходы могут быть использованы для создания оптимального интеллектуального опыта. Эти подходы включают в себя:

- **автоматизацию действий** от имени пользователя;
- **запросы и подсказки** – не хотят ли пользователи предпринять какие-либо действия;
- **организацию информации**, которую видят пользователи, чтобы помочь им принимать лучшие решения;
- **добавление информации** из других частей интеллектуального опыта;
- **гибридный опыт** – сочетание перечисленных методов в зависимости от интеллекта.

В следующих разделах рассматриваются примеры и обсуждаются достоинства и недостатки этих подходов.

Автоматизация действий

Автоматический опыт – это опыт, в котором система делает что-то для пользователя, не позволяя пользователю одобрить или отменить действие. Например:

- вы садитесь в машину в понедельник утром, засыпаете и просыпаетесь на парковке возле вашего офиса;
- вы садитесь перед телевизором, кладете в рот попкорн, и начинается идеальный фильм;
- вы входите в свою учетную запись, и ваш компьютер меняет настройки, чтобы работать быстрее;

- вы убираете все выключатели света у себя дома, подключаете лампочки к интеллектуальной системе и получаете абсолютно правильное освещение для всех ситуаций, даже не касаясь выключателя, и впредь не тратите ни одного лишнего ватта энергии.

Автоматические системы великолепны, волшебны, восхитительны… если они работают. Но для того, чтобы автоматический опыт радовал пользователя, интеллект, стоящий за ним, должен быть исключительно хорошим. Если интеллект ошибается, автоматические взаимодействия с пользователем могут иметь катастрофические последствия.

Автоматический опыт:

- **очень сильный**, потому что он навязывает действия пользователю;
- **не всегда очевидный**, потому что пользователь не всегда знает, что происходит и почему. Неочевидность может снизить ценность взаимодействий и усложнить обнаружение ошибок;
- **ограничивает данные для обучения**, потому что пользователи, как правило, дают отзывы только в случае ошибок. Иногда система может организовать обратную связь между результатом и автоматическим действием, а иногда нет. Автоматические системы обычно нуждаются в тщательном осмыслении того, как интерпретировать действия пользователя и результаты взаимодействий в качестве обучающих данных. Часто им требуется помощь пользователя, который собирает информацию о качестве взаимодействий.

Автоматизация лучше всего используется, когда:

- интеллект очень хорош;
- интеллект развивался в течение длительного времени;
- стоимость ошибок не слишком высока по сравнению со стоимостью правильной автоматизации.

Запросы и подсказки

Интеллект может инициировать взаимодействие между системой и пользователем. Например:

- если интеллект подозревает, что пользователь собирается что-то сделать не так, он может запросить у пользователя подтверждение;
- если интеллект думает, что пользователь входит в продуктовый магазин, он может предложить позвонить супруге и спросить, не нужно ли что-то купить ей;
- если интеллект полагает, что пользователь опаздывает на встречу, он может спросить, не хочет ли пользователь отправить уведомление другим участникам.

Эти взаимодействия требуют внимания пользователя. Они также дают ему возможность рассмотреть действие, оценить его правильность и одобрить или отклонить действие. В этом смысле опыт, основанный на подсказках, позволя-

ет пользователю действовать в качестве страховки для интеллекта, нейтрализуя ошибки до того, как они произойдут. Взаимодействия на основе запросов и подсказок:

- **обладают переменной силой**. В зависимости от того, как представлено взаимодействие, оно может быть очень сильным (например, с помощью диалогового окна, на которое пользователь обязан ответить перед началом работы), или может быть пассивным (например, воспроизведение негромкого звука или отображение небольшого значка на несколько минут);
- **предполагают, что пользователь знает о предпринятом действии** и о том, почему оно было предпринято. Это помогает пользователю осознать ценность интеллектуального взаимодействия, а также замечать и исправлять ошибки. Но частые запросы могут привести к утомлению: если пользователи будут получать слишком много запросов, они начнут с раздражением игнорировать их, что со временем снижает ценность интеллектуальной системы;
- **помогают получать данные для обучения**, так как пользователь отвечает на конкретные запросы. Интеллектуальная система будет иметь представление о том, что происходит, и оценит контекст, который привел к взаимодействию. Это позволяет системе учиться, поскольку она будет знать, насколько успешным было взаимодействие, и сможет улучшить интеллект.

Подобные типы взаимодействий часто используются, когда:

- интеллект ненадежен или системе непонятен контекст для принятия окончательного решения;
- интеллект достаточно хорош, чтобы запросы не казались глупыми, а подсказки могут быть довольно редкими, чтобы не вызывать утомления;
- цена ошибки высока, поэтому системе необходимо, чтобы пользователь участвовал в принятии решений или изменении поведения;
- действия, которые необходимо выполнить, находятся вне контроля системы, например когда пользователю нужно встать и перейти в другой кабинет.

Организованная информация

Интеллект может решить, какую информацию и в каком порядке предоставить пользователю. Например:

- если интеллект считает, что пользователь запрашивает информацию о своей любимой группе, он может выбрать наиболее релевантные веб-страницы для отображения;
- если интеллект считает, что пользователь покупает видеокамеру, он может выбрать рекламу, связанную с камерой, чтобы завлечь пользователя;

- если интеллект считает, что пользователь устраивает вечеринку, он может предложить подборку зажигательных хитов 80-х годов на умном проигрывателе пользователя.

Этот тип опыта обычно используется, когда есть очень много равнозначных вариантов, так что даже «хороший» интеллект вряд ли сузит выбор до одного правильного ответа.

Допустим, пользователю в любое время доступен список из 50 фильмов. Он выбирает фильм, исходя из настроения, наличия свободного времени и компании. Возможно, пользователь с вероятностью 10 % будет смотреть первый фильм, с вероятностью 5 % – второй и т. д. Если бы система показала пользователю только первый фильм из списка, это было бы неправильно в 90 % случаев.

Вместо этого система с организованным выводом информации использует интеллект для предварительного отбора разумного количества вариантов, а затем представляет этот набор в привлекательном для пользователя виде.

Взаимодействия, которые предоставляют выбор для пользователя:

- **не являются принудительными**, поскольку они не подразумевают важных действий пользователя и не требуют его внимания. Однако если опыт предлагает организованную информацию в громоздком или неопрятном виде, взаимодействие может выглядеть раздражающим, но это полностью в руках разработчика опыта;
- **достаточно очевидные**, потому что пользователь видит опции выбора и может почувствовать (или не почувствовать), что за организованной информацией скрывается сильный интеллект. Иногда пользователям бывает сложно находить ошибки – если опция не представлена в списке, пользователь может не знать, была ли эта опция исключена, потому что система не поддерживает эту опцию (продукт, фильм, песню и т. п.) или потому что интеллект ошибся;
- **помогают получать обучающие данные**, потому что легко увидеть, с чем взаимодействует пользователь (когда система сделала что-то правильно). В то же время данные сложнее получить, когда система что-то сделала не так (например, отключила опцию, которую пользователь мог бы выбрать). Это может привести к смещению данных, так как пользователи будут выбирать преимущественно те варианты, которые чаще видят. Подобные системы часто нуждаются в некоторой координации между интеллектом и опытом, чтобы избежать предвзятости, например предлагать пользователям разные наборы вариантов – интеллект не знает заранее, понравится что-то пользователю или нет, поэтому опыт тестирует разные варианты.

Взаимодействия, которые организуют информацию, являются подходящими, если:

- существует много равноценных вариантов ответа и интеллект не может обоснованно выделить «лучший» вариант;

- интеллект способен сформировать ограниченный набор вариантов, и пользователь наверняка выберет один из них;
- задача масштабная и открытая, поэтому нельзя ожидать, что пользователи станут просматривать все варианты и находить решение самостоятельно.

Аннотации

Интеллект может внести небольшую добавочную информацию – *аннотацию* – в другие части опыта, чтобы помочь пользователю принимать лучшие решения. Например:

- если интеллект считает, что разбрызгиватели на лужайке работают напрасно (возможно, по прогнозу ожидается дождь), он может включить мигающий желтый индикатор на корпусе разбрызгивателя;
- если интеллект считает, что есть вероятность скорой поломки автомобиля (например, перегревается двигатель), он может отобразить индикатор «обратитесь в автосервис» на приборной панели;
- если интеллект обнаруживает последовательность символов, которая ему не кажется словом, он может подчеркнуть ее тонкой красной линией, чтобы указать на возможную ошибку в написании.

Подобные типы опыта добавляют небольшую порцию информации и обычно делают это деликатным способом. Когда информация верна, пользователю легче принимать лучшие решения или инициировать взаимодействие для достижения успеха. Когда информация неверна, ее легко игнорировать, а цена ошибки, как правило, невелика.

Взаимодействия на основе аннотации:

- **обычно пассивны** в том смысле, что они ничего не требуют от пользователя и могут даже остаться незамеченными. Это помогает уменьшить усталость от подсказок, но может привести к тому, что пользователи никогда не будут замечать подсказки или взаимодействовать с опытом;
- **недостаточно очевидны** – пользователь может знать, а может и не знать, почему появилась аннотация и достаточно ли она разумна. В зависимости от того, насколько заметным является пользовательское взаимодействие с аннотацией, пользователь может никогда не задуматься о ее обоснованности. Пользователи могут заметить и исправить ошибки или пропустить их;
- **плохо собирают обучающие данные**, поскольку системе часто бывает трудно узнать: 1) заметил ли пользователь аннотацию; 2) изменил ли свое поведение из-за этого; 3) положительным или отрицательным было изменение поведения. Взаимодействия, основанные на аннотациях, могут потребовать дополнительного изучения опыта пользователя, чтобы понять, какие действия предпринял пользователь.

Аннотации работают лучше всего, когда:

- интеллект недостаточно хорош, и вы хотите представить его пользователям в очень ограниченном виде;
- интеллектуальная система не может воспользоваться информацией, поэтому пользователям придется использовать информацию самостоятельно;
- информация представлена в малозаметной форме, но пользователь может легко найти ее, когда захочет.

Гибридный опыт

Интеллектуальный опыт может быть построен путем объединения разных типов опыта в *гибридный опыт* или использования опыта одного типа в местах, где интеллект надежен, и опыта другого типа в местах, где интеллект слаб. Например, когда интеллект абсолютно уверен, опыт может автоматизировать действие. Но когда интеллект только предполагает ответ, опыт может дать подсказку пользователю. А когда интеллект не уверен, опыт может добавить к пользовательскому опыту некоторую аннотацию.

Примером гибридного интеллектуального опыта является фильтрация спама. Идеальный спам-фильтр удалял бы все нежелательные сообщения до того, как пользователь их увидел, и ни в коем случае не трогал бы легитимное письмо. Но это сложно.

Иногда интеллект уверен, что сообщение является спамом, потому что некоторые спамеры не очень стараются. Они помещают названия сами-знаете-каких таблеток прямо в свои электронные письма, даже не пытаясь их зашифровать. Они отправляют свои сообщения из известной части интернета, принадлежащей злоумышленникам. В этих случаях спам-фильтр может легко идентифицировать откровенный спам и быть уверенным, что сообщения не являются желанными. Большинство интеллектуальных спам-фильтров удаляет подобные сообщения без какого-либо участия пользователя. Таким способом системы фильтрации спама удалили миллиарды сообщений.

Но некоторые спамеры умны. Они маскируют свои сообщения способами, разработанными, чтобы обмануть системы фильтрации спама, – например, заменяя буквы i на l или заменяя текст на изображения, – и они весьма хороши в обмане спам-фильтров. Когда интеллект сталкивается с тщательно замаскированным спамом, он не всегда может отличить его от желанных писем. Если бы спам-фильтр попытался удалить все сообщения такого типа, он допустил бы много ошибок и удалил легитимные сообщения. Однако если фильтр поместит все сообщения такого типа в папку «Входящие», опытные спамеры смогут обойти большинство фильтров. Поэтому антиспамовые системы, как правило, имеют временную папку. Сомнительные сообщения перемещаются в папку нежелательной почты, где пользователь может проверить их и исправить ошибки фильтрации.

Это две формы автоматизации, одна из которых является принудительной (удаление сообщений), а другая менее категорична (перемещение сообщений в папку нежелательной почты). Интеллектуальный опыт выбирает между ними на основе доверия интеллекту.

Но многие спам-фильтры предоставляют еще больше возможностей. Рассмотрим случай, когда спам-фильтр считает сообщение хорошим, но слегка подозрительным. Возможно, сообщение кажется идеальным личным сообщением от старого друга, но оно исходит из той части интернета, которую используют многие спамеры. В этом случае интеллект может поместить сообщение в папку «Входящие», но добавить вверху сообщения напоминание, которое гласит: «Не делитесь личной информацией или паролями по электронной почте».

В этом примере интеллектуальная система использует принудительную автоматизацию там, где это необходимо (удаление сообщений без участия пользователя). Она использует менее жесткую автоматизацию, когда есть сомнения в правильности решения (перемещение сообщений в папку нежелательной почты). И иногда она использует аннотацию, когда полагает, что это поможет (предупреждения для нескольких процентов сообщений).

Гибридные опыты часто применяются в больших интеллектуальных системах. Они очень эффективны, если:

- задача может быть разделена на четкие подзадачи, когда некоторые из них являются легкими, а некоторые – трудными;
- задача, стоящая перед интеллектом, сложна, и требуется несколько типов интеллектуального опыта;
- вы хотите как можно меньше беспокоить пользователей и избегать задавать им вопросы в ситуациях, когда решения интеллекта заведомо достоверны или, наоборот, ненадежны. Это может сделать вопросы, которые вы задаете пользователю, гораздо более значимыми и полезными (как для пользователя, так и в качестве обучающих данных).

Итог главы

Существует много способов представления интеллекта пользователям, но важными являются следующие:

- автоматизация действий;
- запросы и подсказки;
- организация информации;
- аннотации;
- гибридный опыт.

Такие варианты, как автоматизация и подсказки, требуют наличия очень хорошего интеллекта (или низких затрат в случае ошибки). Другие варианты меньше воздействуют на пользователя и могут быть хороши при сглаживании ошибок, когда интеллект менее надежен или цена ошибки высока.

У большинства крупных интеллектуальных систем есть гибридный опыт, который автоматизирует простые вещи, запрашивает взаимодействие с пользователем и аннотирует информацию в неоднозначных ситуациях.

Темы для размышлений

Прочитав эту главу, вы должны:

- знать наиболее распространенные способы представления интеллекта пользователям;
- выработать интуитивное понимание того, какие типы интеллектуального опыта использовать в разных ситуациях.

Постарайтесь выполнить задание:

- найдите примеры всех типов интеллектуального опыта в системах, с которыми вы регулярно взаимодействуете (автоматизация, подсказки, организация, аннотации и гибридный опыт).

Подумайте об интеллектуальном опыте, с которым вы взаимодействуете больше всего:

- что это за опыт?
- повторно представьте его в другом режиме взаимодействия (например, заменив подсказки автоматизацией или аннотациями с организацией).

Глава 9

Извлечение данных из опыта

Создатели интеллекта могут работать с любыми видами данных, даже с хаотичными данными. Но когда у них есть хорошие данные, их работа становится намного проще, а потенциал интеллекта, который они могут создать, гораздо выше. Идеальный интеллектуальный опыт будет отслеживать взаимодействия между пользователями и интеллектуальной системой, а из записи этих взаимодействий можно извлечь высококачественные данные.

Прежде чем исследовать способы создания интеллектуального опыта для получения данных от ваших пользователей, давайте рассмотрим альтернативный вариант: вы можете собирать данные самостоятельно. Традиционно системы машинного обучения проходят через этап сбора данных. Например, если вы хотите создать интеллект для подсчета количества коров на снимке, вы берете фотокамеру и делаете много фотографий коров. Вы путешествуете с фермы на ферму, фотографируя коров в разных ситуациях. Вы ставите коров в разные позы, чтобы сфотографировать их в разных ракурсах; фотографируете их морды и… другие части тела. Вы фотографируете коров за кустами. Вы фотографируете, как коровы лежат, бегают, пасутся, спят, зевают, в солнечные дни, в пасмурные дни, ночью… Традиционной системе компьютерного зрения могут потребоваться десятки тысяч различных изображений коров. А еще лучше сотни тысяч или миллионы. И как только вы собрали все эти данные, вам нужно *отметить* коров. То есть вы платите людям, чтобы они смотрели на фотографии и обводили кружками всех коров – с ума сойти!

Такой метод может быть очень затратным. *Сбор данных* и *маркировка объектов* зачастую являются основными источниками расходов в традиционных системах машинного обучения. Поэтому так важно уметь проектировать опыт, который использует взаимодействие пользователей с интеллектуальной системой для получения обучающих данных. Понимание этого подхода дает возможность использовать интеллект и машинное обучение во многих ситуациях, когда использование ручного сбора данных было бы чрезмерно дорогостоящим.

Эта глава начинается с примера того, что означает извлечение данных из опыта и как различные подходы могут повлиять на качество получаемых данных. Затем мы рассмотрим свойства, которые делают данные подходящими для создания аналитики, и некоторые варианты сбора результатов (или мнений) пользователей о данных.

Пример: TeamMaker

Давайте рассмотрим пример интеллектуальной системы, предназначенной для формирования состава баскетбольных команд лиги, – мы назовем ее TeamMaker. Информация о каждом игроке вводится в систему вместе с некоторой начальной статистикой: рост, опыт, позиции, в которых он играет, и, возможно, какие-то данные о том, насколько хорошо он бросает мяч. Затем система делит игроков на команды. Если команды уравновешены, за играми в лиге будет интересно наблюдать. Если команды не уравновешены, лига будет скучной. Одна команда всегда будет побеждать, а остальные станут чувствовать себя неудачниками. Начнутся конфликты, уличные беспорядки, шумиха в прессе…

Так что нам лучше все сделать правильно.

Нам нужно создать опыт, который будет собирать полезные отзывы о командах, сформированных системой, чтобы со временем интеллект улучшался (и лучше формировал команды).

Прямое вмешательство

Один из подходов состоит в том, чтобы разрешить пользователям исправлять состав команд, а затем использовать исправления для улучшения интеллекта.

Например, TeamMaker в качестве отправной точки может создать «предлагаемые команды». Эти предложения могут быть представлены человеку – комиссару лиги, – чтобы исправить возможные ошибки. Вдруг интеллект неправильно решил задачу и поставил всех лучших игроков в одну команду? Тогда комиссар может вмешаться и переместить некоторых игроков. Каждый раз, когда комиссар перемещает игрока, TeamMaker получает обучающие данные, чтобы улучшить интеллект. Через несколько сезонов игр интеллект может стать настолько хорошим, что комиссару не придется вносить никаких изменений.

Но, честно говоря, такой подход к формированию баскетбольных команд выглядит не очень привлекательно. Кто захочет использовать инструмент для создания команд, а затем все равно формировать их вручную? Этот подход потерпит неудачу, если комиссары не предоставят какие-либо исправления, потому что:

- им скучно этим заниматься;
- они не уверены, что смогут убедить машину;
- этим займутся люди, которые не понимают, как играть в баскетбол, поэтому они сделают исправления, которые ухудшат ситуацию;

- они вносят исправления в автономном режиме и даже не вводят их в TeamMaker.

Медленные или нерегулярные взаимодействия ограничат скорость роста и конечный потенциал интеллекта. И наоборот, благодаря правильному взаимодействию пользователь будет хорошо понимать, какую работу он делает для системы, и извлекать очевидную выгоду для себя.

Увлекательное взаимодействие

Другой подход – сделать взаимодействие между пользователем и интеллектом более интересным. Есть вещи, которые пользователи действительно хотят делать, и мотивированы делать правильно.

Например, TeamMaker может не только формировать команды, но и принимать ставки. Большинство людей, которые делают ставку, хочет выиграть (потому что большинство людей думает о выгоде), поэтому они попытаются сделать ставку на лучшую команду – и будут избегать ставок на очевидных неудачников. Таким образом, TeamMaker может использовать распределение ставок, чтобы выяснить, хорошо ли составлены команды. Если ставки сильно смещены в сторону одной из команд, TeamMaker сделает вывод, что интеллект использовал неверные факторы для уравновешивания команд. В следующий раз система сработает лучше.

Взаимодействие на основе ставок является улучшенным примером настройки системы и должно приводить к увеличению объема данных за сезон и расширению возможностей для обучения интеллекта, а также ускорять настройку системы.

Но данные от взаимодействия на основе ставок не будут идеальными. Например, некоторые люди будут делать ставки только на любимую команду, даже если очевидно, что она не может выиграть. Или, может быть, пользователи ничего знают о баскетболе и делают ставки в качестве безобидного офисного развлечения. (Или ставки незаконны и противоречат вашей морали, поэтому как разработчик вы вообще не хотите использовать такой тип взаимодействия с TeamMaker…) Эти проблемы могут привести к смещению данных и сбить интеллект с толку. Когда интеллект учится на основе предвзятых данных о ставках, результирующие команды не будут оптимизированы для создания конкурентоспособной лиги, они будут оптимизированы для чего-то иного.

Когда пользователи имеют иную цель, нежели интеллектуальная система, данные об их взаимодействиях могут вводить в заблуждение. Хороший опыт позволит *согласовать взаимодействие с интеллектом, чтобы получить объективные данные*.

Связь с результатами

Другой подход заключается в том, чтобы напрямую связать интеллект с результатами. Никаких человеческих суждений, только факты. Когда одна и та же команда выигрывает каждую игру, интеллект понимает, что он сделал что-то

неправильно. Когда результаты игры разных команд сопоставимы, интеллект понимает, что он все сделал правильно.

Чтобы этот подход работал, кто-то должен иметь вескую причину, чтобы ввести все результаты в TeamMaker. Поэтому TeamMaker содержит различные развлекательные функции, такие как интеграция со ставками, списки лидеров или расписание игр. Люди могут собираться вокруг своих компьютеров после каждой игры, просто чтобы взглянуть, как изменилась турнирная таблица, кто выиграл деньги и кто опозорился.

По мере того как пользователи вводят данные о том, какие команды выиграли, какие сыграли вничью, а какие потерпели поражение, TeamMaker получает все необходимое для улучшения интеллекта. Если есть команда, которая никогда не проигрывает, интеллект может выяснить, что особенного в этой команде, и избежать этого в будущем. А в следующем сезоне TeamMaker будет работать лучше и создаст лигу из уравновешенных команд – с меньшей вероятностью, что в лиге окажется неудержимая или безнадежно отстающая команда.

Когда опыт может *напрямую отслеживать желаемые результаты*, это помогает сформировать наиболее эффективный интеллект.

Свойства хороших данных

Для решения сложных, больших, открытых, меняющихся во времени задач вам понадобятся данные – много данных. Но подойдут не всякие данные. Для построения интеллекта вам понадобятся данные с определенными свойствами. Лучшие данные будут:

- отражать контекст взаимодействия, любые действия, которые были предприняты, и результаты действий;
- давать ясное представление о том, что ваши пользователи хотят делать с интеллектуальной системой;
- отражать реальные взаимодействия с системой (а не догадки или опросы о том, как люди могли бы взаимодействовать);
- иметь небольшое или нулевое смещение;
- избегать циклической *обратной связи*, когда интеллект влияет на данные, которые влияют на интеллект, который влияет на данные…;
- достаточно велики, чтобы соответствовать масштабу задачи. Обучение интеллекта для сложных задач может потребовать невероятных объемов данных.

Получить такие данные нелегко. На самом деле вы вряд ли получите данные, подходящие для эффективного создания интеллекта, если не будете работать над каждым из перечисленных качеств.

Контекст, действия и результаты

При сборе данных для обучения интеллекта необходимо знать:

1) контекст того, что происходило, когда был вызван интеллект;
2) любые действия, которые были предприняты;

3) результаты взаимодействия (особенно если результаты были положительными или отрицательными).

Например:

- беспилотный автомобиль должен знать, что видят все датчики на автомобиле (контекст), что делает человек-водитель в этой ситуации (действия), а также в конечном итоге врезается ли автомобиль куда-то или ему просто сигналят (результат);
- система, рекомендующая книги, должна знать, какие книги пользователь уже прочитал и насколько они ему понравились (контекст), какие книги могут быть рекомендованы этому пользователю (действия) и какие из рекомендованных книг понравились пользователю (результат);
- системе защиты от вредоносных программ необходимо знать, какой файл загружен пользователем и откуда он его получил (контекст), установил он его или нет (действие) и была ли его машина заражена или нет (результат).

Идеальный интеллектуальный опыт создаст ситуации, в которых достаточно контекста для принятия правильных решений как для пользователя, так и для интеллекта. Например, если беспилотный автомобиль не имеет никаких датчиков, кроме спидометра, то это не поможет узнать, что пользователь сделал с рулевым колесом, – никто не может научиться управлять автомобилем только на основе знаний о скорости.

Достоверный охват

Данные должны содержать наблюдения за всеми ситуациями, в которых придется работать интеллекту. Например:

- если системе необходимо автоматизировать источники света, данные должны содержать контекст, действия и результаты управления источниками света:
  - в течение дня;
  - ночью, вечером;
  - зимой;
  - летом;
  - с лампами, которые используются часто;
  - с лампами, которые используются редко
  - и т. д.;
- если лампы работают в 50 разных странах мира, данные должны содержать наблюдения из этих 50 стран во всех перечисленных ситуациях;
- если система работает в шахте, данные должны содержать наблюдения за лампами, которые используются в шахте;
- если система работает на космической станции, данные должны содержать наблюдения за лампами, которые используются именно там.

Интеллект, работающий в ситуации, для которой он не был обучен (или настроен), может совершать нелепые ошибки.

Предполагается, что интеллектуальные системы будут работать в новых контекстах в течение всего срока службы. Всегда будут новые книги, фильмы, песни, веб-страницы, документы, программы, посты, пользователи и т. д. Если эффективный интеллектуальный опыт сможет правильно оценить поведение пользователей в этих контекстах, то ошибки, допущенные при сборе данных, обойдутся недорого, а ценность собранных данных будет высокой.

Реальное применение

Лучшие данные будут получены от пользователей, которые занимаются тем, что им действительно важно. Например:

- пользователь, управляющий автомобильным симулятором, может принимать решения, которые он не принял бы, если бы сидел за рулем настоящего автомобиля (когда его жизнь в опасности);
- пользователь рассуждает о литературной классике, потому что он пытается произвести на вас впечатление, но на самом деле он никогда не читал эти книги;
- пользователь может дать совершенно разные ответы на вопрос, является ли программа безопасной при установке на лабораторном компьютере или на компьютере его матери.

Если хотите получить объективные данные, то поместите пользователей в контекст, который им небезразличен. Интеллект, обученный на этих данных, наиболее вероятно даст другим пользователям именно то, чего они ожидали.

Отсутствие смещения

Смещение данных возникает, когда опыт влияет на взаимодействие с пользователями или на обратную связь, которые дают пользователи.

Одной из распространенных причин смещения данных является разная скорость, с которой сообщают о различных типах результатов.

Рассмотрим программу фильтрации спама. Если спамовое письмо попадает в почтовый ящик пользователя, оно оказывается прямо у него перед глазами. Пользователь может нажать кнопку «Это спам» и генерировать полезные данные о плохом результате. С другой стороны, если фильтр удаляет полезные письма, пользователь может никогда не заметить плохой результат.

В этом случае выбор, сделанный при проектировании опыта, привел к смещению данных и сделал полученные данные гораздо менее полезными для обучения интеллекта.

Другим потенциальным источником смещения данных является то, что пользователи с выраженным отношением к взаимодействию (хорошим или плохим) с большей вероятностью дадут обратную связь, чем пользователи с нейтральным отношением.

Еще одна причина смещения данных заключается в том, что опыт побуждает пользователей с большей вероятностью принимать некоторые решения. Например, когда опыт представляет варианты выбора в списке, пользователь

с большей вероятностью выберет первый элемент списка (и крайне маловероятно, что он выберет элемент, которого нет в списке вообще).

Отсутствие петель обратной связи

Опыт и интеллект будут влиять друг на друга, иногда отрицательно.

Например, если какая-то часть пользовательского опыта становится более заметной, пользователи будут больше взаимодействовать с этой частью. Если интеллект извлекает уроки из этих взаимодействий, он может подумать, что пользователям больше нравится именно это действие. А когда оно перестает нравиться, пользователи просто лучше замечают его из-за изменения пользовательского интерфейса.

И наоборот, если интеллект по ошибке начнет подавлять действие, которое нравится пользователям, пользователи перестанут видеть эту опцию. Они перестанут выбирать действие (потому что не могут). Отчет о взаимодействии исчезнет из данных. Интеллект сделает вывод, что пользователям больше не нравится эта опция. И это будет досадной ошибкой…

Вот несколько способов устранить обратную связь:

- включите восприятие пользователя в контекст записанных данных. Если один вариант выбора представлен более крупным шрифтом, чем другой, обязательно укажите это в контексте. Если некоторые параметры были исключены, запишите это в контексте;
- добавьте немного рандомизации в то, что видят пользователи, например меняйте порядок некоторых опций. Это помогает расширить охват сбора данных и обнаружить обратную связь;
- запишите усилие, которое пользователь предпринял, чтобы выбрать опцию в текущем контексте. Например, если пользователю пришлось выполнить действие вручную (потому что опция программы была отключена интеллектуальным опытом), интеллект должен знать, что он допустил дорогостоящую ошибку. Если пользователю пришлось просматривать длинный список ссылок, чтобы найти нужный вариант, интеллект должен об этом знать.

Масштаб

Чтобы интеллектуальная система развивалась, ее надо использовать в реальной жизни. Это означает, что взаимодействия, предлагаемые интеллектуальным опытом, должны быть заметными и увлекательными, их должно быть легко найти, и они должны быть тем, что пользователи часто делают.

Рассмотрим два варианта: новая интеллектуальная система и действующая интеллектуальная система.

Новая интеллектуальная система, вероятно, не будет иметь много пользователей. Это означает, что интеллектуальный опыт должен лидировать в регулярных взаимодействиях с пользователями. В этих случаях интеллектуальный опыт должен стимулировать взаимодействие, делать его увлекательным

и, возможно, даже менять весь продукт, чтобы интеллектуальное взаимодействие было в центре внимания.

Существующая интеллектуальная система имеет намного больше пользователей. Это означает, что интеллектуальные взаимодействия могут быть более тонкими. Их можно разместить там, где их увидит меньшее количество пользователей. Это не означает, что они менее ценны, – они могут решать очень важные проблемы, но это такие проблемы, с которыми пользователи сталкиваются еженедельно или ежемесячно, а не ежедневно.

Полезность данных, собранных в определенном контексте, зависит от объема следующим образом:

- выработка десятков взаимодействий в день бесполезна для проверки или улучшения интеллекта;
- выработка сотен взаимодействий в день, вероятно, поможет проверять и обучать простой интеллект;
- выработка тысяч или десятков тысяч взаимодействий в день, безусловно, обеспечит проверку интеллектуальной системы и производство интеллектуальных решений для большинства сложных и открытых задач;
- выработка сотен тысяч или миллионов взаимодействий в день, вероятно, обеспечит необходимый объем данных для подавляющего большинства задач.

Эффективный интеллектуальный опыт привлечет пользователей и включит их в работу по созданию базы данных и обучению интеллекта.

Правильное толкование данных

Когда происходит взаимодействие между пользователем и интеллектом, не всегда легко понять, является результат положительным или отрицательным. Например, пользователь нажал кнопку «Далее» в системе музыкальных рекомендаций, потому что он ненавидит песню (интеллект напрасно ее предложил?) или потому что он увидел далее в списке новую песню, которая его действительно интересует (интеллект сработал правильно?).

Идеальный интеллектуальный опыт настроен на то, чтобы прояснить результат косвенным образом, не полагаясь только на очевидные действия пользователя. Но это не всегда возможно. И даже когда это удается, хорошо бы иметь возможность убедиться, что косвенные данные обладают всеми необходимыми свойствами для обучения интеллекта. Методы толкования результатов включают в себя:

- скрытые наблюдения;
- пользовательские рейтинги;
- отчеты о проблемах;
- эскалации;
- пользовательские решения.

Многие крупные интеллектуальные системы используют сочетание нескольких подходов, чтобы как можно лучше истолковать результаты, получаемые пользователями.

Скрытые наблюдения

Идеальный опыт даст хорошие, полезные данные без каких-либо специальных действий пользователей. Они будут генерировать данные, просто пользуясь системой, и эти данные будут обладать всеми свойствами, необходимыми для развития интеллекта.

Например, когда пользователь устанавливает свой термостат на 22 градуса, становится совершенно ясно, что он хочет получить. Или когда пользователь покупает продукт или цифровой контент, достаточно очевидно, что он заинтересован в нем.

Однако иногда трудно понять, как истолковать действия пользователя. Пользователи не тратят времени на принятие безупречных решений, поэтому способ представления данных может повлиять на результат. Например, пользователь выбрал один из пяти рекомендованных фильмов, потому что это самый подходящий фильм на тот момент – или потому что ему лень просматривать остальные варианты.

Из-за этого создание полностью *скрытой системы сбора данных* является довольно сложной задачей, требующей тщательной настройки и сокращения охвата опыта, чтобы можно было эффективно интерпретировать действия пользователя.

Пользовательские рейтинги

Пользовательские рейтинги и обзоры могут быть очень хорошим источником данных. Например, можно спроектировать опыт, который позволяет пользователям:

- оценить предложенный контент в диапазоне 1–5 звезд;
- провести пальцем вверх или вниз по конкретному элементу взаимодействия;
- оставить краткое текстовое описание своего опыта.

Пользователи привыкли ставить оценки, и многие любят делать это, чувствуя, что они помогают другим пользователям или персонализируют настройки своего опыта.

Но с рейтингами есть некоторые проблемы.

1. Пользователи не всегда и не все оценивают. Возможно, им не хочется прикладывать усилия.
2. Между взаимодействием и оценкой проходит некоторое время. Пользователь может забыть, что именно произошло, или не связывать полученный результат со взаимодействием.
3. Рейтинг может не соответствовать параметру, который оптимизирует интеллект. Например, рассмотрим два вопроса: насколько хороша эта книга? Правильно ли было рекомендовать эту книгу вчера?

4. Рейтинги различаются для разных групп пользователей: пять звезд в одной стране – это не то же самое, что пять звезд в других странах.
5. Редкие взаимодействия не получают много оценок, и о них трудно узнать.

Отчеты о проблемах

Иногда опыт позволяет пользователям сообщать, что что-то пошло не так. Например, многие почтовые системы имеют кнопку «Это спам» для сообщений, которые оказались в папке «Входящие».

Данные, собранные с помощью отчетов о проблемах, явно смещены, так как они содержат только плохие результаты (когда интеллект допустил ошибку), и пользователи не будут сообщать о каждой проблеме. Например, пользователи могут нажать кнопку «Это спам» только для 10 % спама, который они видят, и просто проигнорировать или удалить остальную часть спама.

Из-за этого данные из отчетов о проблемах трудно использовать в качестве эксклюзивного источника для обучения интеллекта. Тем не менее отчеты могут быть очень полезными для отслеживания негативных последствий ошибок интеллекта.

Эскалации

Эскалация – это дорогостоящая разновидность отчета, которая возникает, когда пользователи достаточно расстроены, чтобы связаться со службой поддержки (или посетить форум поддержки и т. д.). Эскалации аналогичны отчетам, но находятся за пределами основного опыта продукта. И они обычно представляют очень огорченных пользователей и важные ошибки.

Данные на основе эскалации склонны быть замусоренными и несфокусированными. Пользователь, у которого возникают проблемы, не всегда знает, какая часть системы работает неправильно. Он не будет выражать проблему в тех же терминах, что и другие пользователи, сообщившие об этом.

Из-за этого бывает трудно и затратно разбираться в эскалациях и использовать их для улучшения интеллекта – они лучше подходят для выявления больших проблем, чем для текущего развития системы.

В целом лучше разрешить пользователям сообщать о проблемах, оставаясь в контексте (с помощью оперативного представления отчетов при появлении проблемы), чтобы система могла фиксировать соответствующий контекст и действия, которые привели к нежелательному результату.

Пользовательские решения

Вы можете просто спрашивать пользователей об их результатах, используя конкретные (тщательно контролируемые) вопросы в контексте взаимодействия пользователей с вашей системой. Нечто наподобие опроса, например:

- система автоматизации освещения раз в три месяца спрашивает: «Чтобы улучшить наш продукт, мы хотели бы знать – прямо сейчас освещение должно быть включенным или выключенным?»

- программа для подбора и редактирования фотографий после сотни сеансов работы говорит: «Пожалуйста, помогите улучшить эту программу. Есть ли лицо на изображении, над которым вы сейчас работаете?»

Пользовательские решения могут дать очень сфокусированные данные для обучения интеллекта. Их можно собирать ненавязчивым способом, например ограничить количество вопросов:

- один вопрос на тысячу взаимодействий;
- пользователям, которые сами вызвались помочь;
- пользователям, которые не подписались на платный сервис,
- и т. п.

Итог главы

Интеллектуальный опыт играет решающую роль в получении данных для растущего интеллекта. Если интеллектуальный опыт разработан без учета извлечения качественных данных для обучения интеллекта, он почти наверняка произведет данные, которые окажутся плохими или бесполезными.

Опыт вырабатывает наилучшие данные, когда пользователи часто взаимодействуют с интеллектом, когда они видят для себя ценность во взаимодействиях и когда у них есть подходящая информация для принятия правильных решений.

Данные наиболее полезны, когда они:

- характеризуют контекст взаимодействия, предпринятые действия и результаты;
- охватывают все ситуации, в которых будет использоваться интеллектуальная система;
- представляют реальное использование, то есть взаимодействие, которое действительно волнует пользователей;
- содержат объективные данные;
- содержат достаточно информации для выявления и устранения петель обратной связи;
- имеют достаточный объем с учетом сложности проблемы.

Наиболее эффективные данные поступают из неявных взаимодействий, когда пользователь естественным образом выражает свое намерение и свое мнение о результате использования продукта.

Когда неявных взаимодействий недостаточно, применяются другие методы толкования результатов:

- пользователи явно оценивают контент или взаимодействие;
- пользователи сообщают о проблемах непосредственно из контекста взаимодействия;
- пользователи обращаются в службу поддержки с жалобами на большие и дорогостоящие проблемы;
- система обращается к пользователям с конкретными и редкими вопросами.

Многие интеллектуальные системы используют все перечисленные подходы. Большая часть данных таких систем может быть получена из неявных взаимодействий, при этом будут доступны и методы контроля качества неявных данных (не началось ли развитие смещения или формирование обратной связи).

Трудно создать опыт, извлекающий полезные данные, не обременяя пользователей, но это очень важно для построения качественных интеллектуальных систем.

Темы для размышлений

Прочитав эту главу, вы должны:

- знать, как создавать опыт, который извлекает достоверные данные, необходимые для оценки и развития интеллекта;
- знать способы извлечения данных – от не требующих явных действий со стороны пользователя до тех, которые активно эксплуатируют пользователя.

Постарайтесь ответить на следующие вопросы:

- подумайте об интеллектуальной системе, которую вы регулярно используете и которая собирает данные неявным образом. Как можно изменить систему, чтобы извлекать пользовательские определения?
- вообразите интеллектуальную систему, которая помогает вашему любимому хобби. Подумайте о неявном способе сбора «хороших данных», когда система что-то делает не так. Теперь подумайте о другом способе извлечения данных в той же ситуации, который выражает другое толкование действий пользователя (например, интеллект был прав, но пользователь не захотел выполнять предложенное действие по какой-то другой причине).

Глава 10

Проверка интеллектуального опыта

Интеллектуальный опыт терпит неудачу по двум основным причинам:

1) потому что он неправильно реализован – в опыте есть ошибка, он сбивает пользователей с толку или не помогает им достичь своих целей;
2) потому что интеллект ведет себя некорректно – он совершает ошибочные действия, неверно трактует действительность или неправильно сортирует информацию.

Например, интеллектуальный опыт может:

- подготовить список изображений с лицами членов семьи, но не включить в него любимую фотографию пользователя со свадьбы. Изображение исключено из-за ошибки в опыте? Или потому, что интеллект не заметил, что на изображении есть лицо?
- выключить свет, когда в комнате темно. Опыт выключил свет, потому что знал, что скоро взойдет солнце (хотя знал, что в комнате слишком темно)? Или свет погас, потому что интеллект думал, что комната пуста?
- рекомендовать множество фильмов ужасов, хотя пользователь ненавидит такие фильмы. Интеллект делает это, потому что муж пользователя установил предпочтение в каком-то скрытом меню, чтобы убрать все романтические комедии раз и навсегда? Или это происходит потому, что интеллект изучил фильмы и решил, что это веселые приключения, а не ужасы?

Прежде чем говорить о качестве интеллектуального опыта, необходимо удостовериться, что интеллект правильно функционирует.

Достижение ожидаемого опыта

Итак, мы знаем, что интеллект может работать правильно или неправильно. Но как работает остальная часть системы? Задайте себе вопросы:

- получает ли пользователь ожидаемый опыт взаимодействия, когда интеллект работает правильно?
- помогает ли опыт на самом деле достичь целей пользователя и системы?

- сохраняет ли опыт эффективность по мере развития системы и интеллекта?

В этом разделе говорится о том, как отделить опыт от изменений, которые могут произойти в интеллектуальной системе, чтобы у вас получился объект для проверки.

Работа с контекстом

Контекст являет нам состояние интеллектуальной системы на момент, когда сработал интеллектуальный опыт. Он включает в себя все переменные, которые интеллект учитывает при выработке своих решений. Контекст также должен включать переменные, которые влияют на эмоциональное восприятие и интерпретацию результатов пользователем. Например:

- контекст системы компьютерного зрения может содержать освещение окружающей среды, настройки камеры, пользователей и их особенности (этническая принадлежность, пол, одежда) и т. д.;
- контекст системы воспроизведения музыки может включать в себя оценки песен пользователем, список песен, которые приобрел пользователь, недавнюю историю прослушивания и т. д.;
- контекст системы «умного дома» может включать в себя количество и типы датчиков, помещения, в которых они расположены, расстояние между ними и т. д.

Если вы хотите убедиться, что интеллектуальный опыт правильно работает в определенном контексте, вам следует воссоздать контекст и опробовать его. Например, чтобы воссоздать контекст системы компьютерного зрения, необходимо правильно настроить освещение, привлечь тех же самых пользователей, поставить их перед камерой и заставить их точно так же улыбаться.

Это может быть затруднительно, особенно для больших открытых задач, которые имеют бесконечное количество контекстов.

При проверке интеллектуального опыта полезно иметь специальные инструменты, которые помогают создавать (или захватывать) важные контексты, исследовать их и воспроизводить в интеллектуальном опыте, чтобы вы могли проверять опыт снова и снова по мере развития интеллектуальной системы. Варианты работы с контекстами включают в себя:

- **создание контекста вручную** по примеру пользователя – используя систему, создавая учетную запись, формируя историю взаимодействий, переходя по нужным ссылкам и т. д. Это сложный и подверженный ошибкам процесс, поскольку контексты могут иметь тонкие нюансы; например, изменение интонации при разговоре с компьютером может повлиять на эффективность работы системы распознавания речи. Контексты могут быть довольно обширными – например, история использования системы за длительный период времени;
- **создание контекста с помощью редактора** или другого инструмента, который позволяет создавать, редактировать, записывать и воспроизво-

дить контексты в системе. Например, это может быть приложение, которое описывает планировку дома, расположение датчиков и их показания в заданное время. Уровень сложности и автоматизации может варьироваться, но эти инструменты бывают очень полезны при проведении базовых проверок;

- **запись примеров использования** либо в лабораторных условиях, либо у реальных пользователей. Запись примеров подразумевает способность системы выгружать контекст в любое время и создание инструментов для загрузки дампов контекста и тестирования обновленных версий интеллектуального опыта и интеллекта. Такой подход позволяет имитировать реальное использование и достичь большого разнообразия контекстов в процессе проверки.

При проверке интеллекта желательно использовать все упомянутые подходы. Типичный рабочий процесс может выглядеть так:

- воспроизведение проблемы, о которой сообщает пользователь;
- захват точного контекста, при котором возник отчет о проблеме;
- просмотр контекста в каком-то инструменте, чтобы увидеть, что происходит *на самом деле* (а не только в той части системы, на которую жаловался пользователь);
- внесение изменений в опыт;
- выполнение контекста снова, чтобы увидеть, решит ли изменение проблему;
- сообщение о проблеме разработчикам интеллекта (но не факт, что они помогут).

Обратите внимание, что контекст для проверки опыта похож на данные для создания интеллекта, но это разные вещи. Для создания интеллекта вам также необходимо собрать информацию о результате, а обособленный контекст позволяет проверить и улучшить опыт.

Работа с интеллектом

Рабочий процесс отслеживания и воспроизведения контекстов аналогичен отслеживанию и исправлению ошибок в других сложных программных продуктах. Но изменение интеллекта может усложнить задачу. Дело в том, что опыт, который наблюдает пользователь, является результатом сочетания определенного контекста с определенным интеллектом – а интеллекту свойственно меняться.

По этой причине полезно иметь коллекцию разных вариантов интеллекта, которые можно комбинировать с известными контекстами и наблюдать за результатом. Вот некоторые варианты интеллекта для проверки:

- *действующий* интеллект – это то, что увидят пользователи. Если все идет хорошо, это будет наиболее точное представление интеллекта и самый лучший способ проверить, что на самом деле видят пользователи. Но по мере развития интеллекта ситуация быстро меняется, и придется по-

трудиться, чтобы отличить изменения интеллекта от изменений опыта. Допустим, у вас есть проблема интеллектуального опыта, над которой вы работаете, и вдруг… она исчезает! Вы еще ничего не меняли, но, возможно, интеллект изменился из-за вашего присутствия, и вы больше не сможете воспроизвести проблему;

- *фиксированный* интеллект, который при каждом обращении выполняется одинаково. Например, вы фиксируете контрольную точку интеллекта действующей системы (допустим, первый день альфа-тестирования или первый день каждого месяца). Фиксированный интеллект похож на действующий интеллект, но обеспечивает стабильное состояние системы на интервале между контрольными точками;
- *простой* интеллект, который представляет собой нечто, доступное для понимания. Например, вместо использования модели, созданной машинным обучением, используйте небольшой набор эвристических правил. Это может быть полезно при отладке проблем – иметь достаточно интеллекта, чтобы вызвать интересные проблемы, но не настолько много, чтобы ошибки интеллекта перекрывали проблемы опыта;
- *«глупый»* интеллект, который всегда дает один и тот же ответ, независимо от того, какой предоставлен контекст. Например, независимо от того, какие песни нравятся пользователю, интеллект рекомендует Dire Straits. «Глупый» интеллект помогает исключить очевидные проблемы с опытом.

При проверке интеллекта полезно использовать все перечисленные опции.

Промежуточный итог

Наиболее распространенные типы ошибок:

- контекст неправильно интерпретируется интеллектом, что приводит к неожиданному выводу;
- вывод интеллекта неверно истолковывается опытом, что приводит к систематическим ошибкам опыта;
- представление пользовательского опыта некорректно для некоторых комбинаций контекста и интеллекта, что приводит к бессмысленным, непривлекательным или неудобочитаемым результатам.

Достижение целей

Опыт играет очень важную роль в достижении целей системы. Оперируя *представлением интеллекта* – значками, текстом, цветом, содержанием, временем и местом отображения информации, – опыт может превратить некачественный интеллект в полезный инструмент. Опыт также может сделать высококачественный интеллект бесполезным.

Рассмотрим систему, предназначенную для защиты пользователей от мошенничества в интернете. Если пользователь посещает страницу, по мнению интеллекта принадлежащую мошенникам, он получает предупреждение.

Если бы интеллект был безупречен, предупреждение было бы очень сильным (например, блокировало бы доступ к странице и не давало пользователю возможности обойти блокировку). Но интеллект никогда не бывает безупречным, поэтому необходимо найти разумный компромисс между риском попадания пользователя к мошенникам и риском напрасно напугать пользователя на легальном сайте.

Итак, как мы можем проверить эффективность любого конкретного опыта?

Это можно сделать путем изоляции работы интеллекта от работы опыта.

Работа интеллекта – обнаруживать мошеннические сайты. Он может работать правильно или ошибаться.

Работа опыта заключается в следующем:

1. Успешно защищать пользователей, когда интеллект правильно определяет мошенников.
2. Сводить к минимуму затраты пользователя, если интеллект ошибочно помечает легальный сайт как мошеннический.

Чтобы убедиться, что опыт выполняет свою работу, следует оценить следующие значения:

- каков процент пользователей, которые правильно получили предупреждение, но все равно попали к мошенникам?
- каков процент пользователей, которые неправильно получили предупреждение и напрасно покинули легальный сайт?

В более общем случае следует рассмотреть все возможные результаты работы интеллекта – все варианты, когда интеллект работает правильно, и все варианты, когда он ошибается, – а затем определить и измерить роль опыта в этих ситуациях.

Обратите внимание: такие измерения очень сложно выполнить в лабораторных условиях – для точного измерения зачастую необходимо, чтобы реальные пользователи принимали реальные решения.

Непрерывная проверка

Интеллект и поведение пользователя меняются со временем. Эти изменения могут привести к тому, что опыт станет менее эффективным в течение жизни системы. Или наоборот, изменения откроют возможность сделать опыт более сильным и эффективным.

Некоторые изменения, которые могут повлиять на опыт:

1. Качество интеллекта становится лучше или хуже.
2. Качество интеллекта остается прежним, но изменяются ошибки, которые он совершает.
3. Пользователи привыкают к этому опыту (или устают от него) и начинают его игнорировать.
4. Новые пользователи могут вести себя иначе, чем первоначальные пользователи.

В интеллектуальной системе есть много поводов для изменений. И большинство изменений предоставляет возможность (по крайней мере, если вы оптимист, то будете рассматривать изменения как возможности), чтобы сбалансировать компоненты системы для достижения наилучшего результата – разумеется, если вам известно, что изменение произошло, и вы знаете, как оно повлияло на систему.

Корректировка опыта не всегда является правильной реакцией на изменения, но заслуживает внимания.

Итоги главы

Проверка эффективности интеллектуального опыта может оказаться сложной задачей, потому что ошибки могут быть вызваны опытом, интеллектом, представлением пользователя в каком-то контексте, которого вы не ожидали, или внезапными изменениями.

Когда ошибок много, они растут как снежный ком, и полезно иметь несколько способов упростить и изолировать проблему. Это помогает избежать перекладывания ответственности: «Это ошибка интеллекта!» – «Нет, это проблема опыта» – «Нет, это проблема интеллекта».... И т. д.

Желательно отработать навык захвата, проверки и управления контекстами. Это можно сделать при помощи:

- ручной генерации контекстов;
- инструментов для проверки и изменения контекстов;
- записи контекстов для реальных пользователей и ситуаций.

Также полезно иметь возможность воспроизводить контексты с различными типами интеллекта и наблюдать, что получает опыт от интеллекта. В целом для проверки опыта применяются следующие типы интеллекта:

- действующий интеллект, с которым взаимодействуют пользователи;
- контрольная точка действующего интеллекта, которую можно использовать в течение длительного периода без изменений;
- простой интеллект, который легко интерпретировать (возможно, набор эвристических правил);
- «глупый» интеллект, который всегда говорит одно и то же.

Также важно убедиться, что интеллектуальный опыт помогает пользователям достичь своих целей. Для этого обычно наблюдают за живыми взаимодействиями, когда интеллект работает правильно, и добиваются того, чтобы пользователи получали хорошие результаты (и не были озадачены или введены в заблуждение опытом).

Интеллектуальные системы всегда меняются, поэтому следует постоянно проверять эффективность интеллектуального опыта. Если вы отслеживаете изменения и располагаете инструментами для выявления последствий, то найдете возможности для восстановления баланса системы и достижения большей пользовательской ценности.

Темы для размышлений

Прочитав эту главу, вы сможете:

- отделить проблемы опыта от проблем интеллекта и убедиться, что опыт работает так, как задумано;
- создать эффективный план тестирования интеллектуального опыта;
- понять важность переоценки изменений – как интеллекта, так и пользователей.

Рассмотрите интеллектуальную систему, которую вы регулярно используете:

- составьте план проверки, чтобы убедиться, что ее опыт работает правильно;
- разработайте простой интеллект для проверки опыта. Опишите три контекста, которые вы создадите вручную, чтобы загрузить в простой интеллект. Почему вы выбрали эти три контекста?
- каким образом вы можете контролировать систему, чтобы обнаружить, что опыт теряет эффективность?

Часть III

РЕАЛИЗАЦИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

В главах 11–15 обсуждается реализация интеллектуальных систем. Речь пойдет о подсистемах, которые превращают интеллект (модели, создаваемые машинным обучением) в полностью функциональную систему – формируют среду выполнения интеллекта, контролируют его работу и поддерживают развитие. Эта часть книги охватывает все важные компоненты, которые необходимо разработать, а также их обязанности, возможности и компромиссы.

Глава 11

Компоненты реализации интеллекта

Действующий интеллект должен быть связан с пользователем. То есть когда пользователь взаимодействует с системой, интеллект выполняется с соответствующими входными данными, а пользовательский опыт дает должный отклик. После создания нового интеллекта его следует проверить и развернуть в нужном месте. Необходимо предусмотреть механизмы проверки работоспособности и сбора данных для развития интеллекта в течение срока службы.

За все перечисленное отвечает *реализация интеллектуальной системы*. Это основа, на которой создается интеллектуальный сервис.

В качестве аналогии рассмотрим веб-сервер.

Реализация веб-сервера должна принимать сетевые соединения, выполнять аутентификацию, интерпретировать запросы и находить нужный контент, обрабатывать его (возможно, запускать сценарии), вести системный журнал и выполнять множество других действий.

Как только реализация веб-сервера становится доступной, создатели контента получают возможность использовать веб-сервер для размещения различных сайтов.

Это похоже на реализацию интеллектуальной системы – она позволяет разработчикам интеллекта начать выполнять взаимодействия с пользователем. Искусственный интеллект и машинное обучение меняют мир, а правильная реализация интеллектуальных систем позволяет им выполнять свои обещания.

В этой главе описываются компоненты, составляющие реализацию интеллектуальной системы.

Пример реализации интеллектуальной системы

Представьте себе интеллектуальную систему, которая помогает пользователям искать смешные веб-страницы.

Пользователь переходит на страницу, и если система уверена, что страница смешная, в адресной строке браузера появляется смайлик. А как иначе – никто не должен случайно пропустить смешные страницы.

Выглядит достаточно просто. Давайте обсудим, как эта интеллектуальная система может быть реализована.

Все начинается с программы, которая проверяет содержимое веб-страницы и оценивает, насколько она смешная. Скорее всего, это модель, созданная машинным обучением, но может быть и простая эвристика. Например, очень простая эвристика может считать, что любая страница, содержащая фразы «вошел в бар» или «кто там», смешная, а каждая страница, которая не содержит этих фраз, не смешная.

Для некоторых это может оказаться суровой реальностью – если у них тоже есть родственник с чувством юмора, как у робота…

Вы можете написать плагин для браузеров, реализующий простую эвристическую модель. Плагин сравнивает содержимое страницы с моделью и, в зависимости от результата сравнения, отображает в адресной строке браузера смайлик, если модель считает страницу забавной, или не отображает ничего, если модель полагает, что страница не смешная.

Этот плагин представляет собой очень простую реализацию интеллекта.

Стоит лишь выложить плагин для скачивания, и очень скоро кто-нибудь захочет проверить его работу на практике.

Наверняка ваш плагин предоставляет пользователям возможность обратной связи. Если система утверждает, что страница смешная, но пользователю не смешно, он может щелкнуть по изображению хмурого лица в конце страницы, чтобы система знала, что допустила ошибку. Если пользователь читает страницу, которую система сочла несмешной, и все равно смеется, то он может щелкнуть по изображению смеющегося лица. Возможно, вы даже захотите узнать, читают ли пользователи больше страниц, помеченных как смешные, по сравнению со страницами, которые не имеют этой пометки. С этой целью вы можете сравнить время, которое пользователи проводят на разных страницах.

Плагин собирает все отзывы пользователей и записи об их поведении и отправляет обратно в систему в качестве телеметрии.

Наличие телеметрии совершенствует общую реализацию интеллекта, поскольку позволяет нам отвечать на важные вопросы о том, насколько хорош интеллект и как пользователи воспринимают систему в целом – достигает ли она общих с пользователем целей.

Когда разработчики интеллекта изучают данные телеметрии, они находят места, где их первоначальная модель не сработала. Вполне вероятно, что самые смешные вещи в интернете – это изображения, а не текст. Вдобавок людей в разных странах могут веселить совершенно разные вещи.

Создатели интеллекта некоторое время будут старательно изучать телеметрию. Возможно, они будут исследовать страницы, на которых система допускала ошибки, и создавать новые модели. Наконец, у них появится модель второго поколения, заметно лучше, чем исходная, и они захотят представить ее пользователям.

Об этом должна позаботиться реализация интеллекта. Одним из вариантов обновления будет поставка совершенно новой версии плагина – Funny Finder

v2.0, – который содержит новую модель. Но пользователям первой версии нужно как-то найти этот новый плагин и выполнить установку. Большинство из них не станет этого делать. И даже те, кто решатся на обновление, займутся этим далеко не сразу. В итоге интеллект будет обновляться с большой задержкой (если вообще будет), а это значительно снижает потенциальные возможности команды разработчиков. Кроме того, интеллект способен быстро меняться: возможно, каждую неделю, может быть, каждый день или несколько раз в час. Если реализация не сможет регулярно и быстро предоставлять пользователям новый интеллект, интеллектуальная система никогда не станет очень умной.

Следовательно, реализация должна предусматривать регулярное обновление плагина. А еще лучше, если плагин будет обновлять только модель, не меняя остальной код. Пусть ваш плагин периодически проверяет обновления на сервере, определяет наличие новой модели и загружает ее.

Великолепно. Теперь реализация запускает интеллект, измеряет качество его работы и выполняет обновление. Объем кода для обслуживания интеллекта на данный момент, вероятно, превышает объем кода самого интеллекта, поскольку значительная часть любой интеллектуальной системы – это реализация. Теперь ваша система завершена и замыкает петлю обратной связи между пользователем и интеллектом, позволяя пользователю извлекать выгоду и улучшать интеллект просто с помощью взаимодействия.

Но некоторые вещи в интернете совсем не смешные – они оскорбительные и отвратительные. Наверняка вы вовсе не хотите, чтобы ваша система ошибочно помечала оскорбительный контент как забавный.

Следовательно, у пользователей должна быть возможность сообщить, когда ваша система совершает действительно ужасные ошибки. Пусть это будет новая кнопка в пользовательском интерфейсе, которая добавляет в телеметрию сообщения об оскорбительных страницах.

Возможно, придется создать небольшую рабочую группу и нанять нескольких человек для проверки обоснованности жалоб на сайты. Вы должны убедиться, что эти жалобы – не проявление кухонной войны, когда владельцы сайтов жалуются друг на друга. В результате у вас в руках оказывается список «абсолютно не смешных» сайтов.

Реализация должна гарантировать, что клиенты будут получать обновления этого списка как можно скорее. Список может быть обновлен вместе с моделью. Однако если модель обновляется недостаточно часто, плагин должен более активно проверять этот список и объединять содержащиеся в нем сведения с решениями, которые выводит модель.

Теперь каждый раз, когда плагин встречает новую страницу, он обращается к удаленной службе, в то время пока запускает свою локальную модель. Затем он объединяет результаты работы двух форм интеллекта (интеллект на основе сервера и интеллект на стороне клиента). Если сервер говорит, что сайт «абсолютно не смешной», не имеет значения, что говорит модель на стороне клиента, – этот сайт не смешной.

К этому моменту у создателей интеллекта созреют всевозможные идеи о том, как создать более совершенные модели, которые не могут работать в плагине. Возможно, новые модели не могут работать на стороне пользователя, потому что они занимают слишком много оперативной памяти. Вероятно, им требуются внешние сервисы (например, для служб языкового перевода), которые привносят слишком большую задержку в плагин. Или, например, плагин должен работать в среде выполнения со слабым процессором, как в телефоне, а создатели интеллекта хотят иметь пространство для развития.

Такие типы интеллекта не могут быть размещены на стороне клиента, зато они идеально размещаются на стороне сервера.

Например, когда на новый сайт начинает приходить большое количество посетителей, ваш сервер сразу это замечает. Он может просканировать сайт и предложить десятки различных алгоритмов – специально для изображений, для определенного типа юмора, настроенный на разные языки или культуры – и каким-то образом предоставить результаты клиенту.

Так что теперь плагин объединяет интеллектуальные продукты из нескольких источников – некоторые получает из облака, а некоторые добывает сам. Это *администрирующий опыт*. Он оценивает результаты взаимодействия пользователей с интеллектом и собирает данные для улучшения.

Не каждая реализация интеллектуальной системы нуждается во всех этих компонентах. И не всякая интеллектуальная система нуждается в их внедрении в одинаковой степени. Эта часть книги рассказывает о том, когда и как эффективно вкладывать труд и средства в различные компоненты, чтобы ваша интеллектуальная система получила наибольшие шансы на успех.

Компоненты реализации интеллектуальной системы

Реализация интеллектуальной системы может быть как очень простой, так и очень сложной. Но есть некоторые ключевые функции, которые должна включать в себя каждая реализация, а именно:

- предоставлять среду выполнения интеллекта и порождать опыт взаимодействия;
- получать новый интеллект, проверять его и размещать по месту назначения;
- контролировать интеллект (и пользовательские результаты) и получать любую телеметрию, необходимую для улучшения системы;
- помогать разработчикам интеллекта;
- обеспечивать оркестровку системы, ее управляемое развитие и устранение ошибок.

Среда выполнения интеллекта

Интеллектуальной системе необходима *среда выполнения интеллекта* (intelligence runtime), в которой он будет работать и связываться с пользовательским опытом.

Чтобы выполнить интеллект, система должна получить контекст взаимодействия – все, что интеллект должен учитывать для принятия правильных решений. Контекст может включать в себя следующие знания:

- что пользователь делает сейчас;
- что пользователь недавно сделал;
- что сообщают соответствующие датчики;
- на что смотрит пользователь на экране;
- все, что может иметь отношение к интеллекту, принимающему решение.

Контекст должен быть объединен, преобразован и доставлен на вход модели (или моделей), представляющей интеллект. Сочетание контекста и модели порождает *прогноз* (prediction).

Среда выполнения может полностью находиться на стороне клиента или распределяться между клиентом и сервером.

Полученный прогноз применяют, чтобы воздействовать на систему и инициировать опыт – иначе говоря, создать *воздействие* (impact) на пользователя.

Распределение и доставка интеллекта

Когда новый интеллект готов для использования, он должен быть принят системой и доставлен в нужное место.

Например, если интеллект создается в лаборатории центрального офиса, а среда выполнения интеллекта располагается на тостерах по всей Америке, то система управления интеллектом должна разослать обновление по всем тостерам.

Интеллект может работать как серверная служба.

Или может частично работать на стороне сервера, а частично на стороне клиента.

Есть много вариантов размещения интеллекта со своими плюсами и минусами.

Заодно необходимо проверить интеллект, чтобы убедиться, что он не сделает ничего чересчур безумного.

Интеллектуальные системы обычно полагаются на несколько источников интеллекта. Они могут включать в себя несколько моделей, эвристик и путей исправления ошибок. Источники интеллекта должны быть объединены, проверены и доставлены в среду выполнения.

Канал интеллектуальной телеметрии

Правильно функционирующий и постоянно развивающийся интеллект не может обойтись без регулярного получения данных мониторинга и телеметрии.

Эффективный *мониторинг* и *телеметрия* подразумевают знание контекста, в котором работает интеллект, и ответов, которые он дает, а также какой опыт порождает интеллект и как на него откликаются пользователи.

В больших интеллектуальных системах телеметрия и мониторинг могут производить много данных – очень, очень МНОГО данных.

Поэтому реализация интеллекта должна решить, за чем наблюдать, какие данные отбирать и как переваривать и суммировать информацию, чтобы обеспечить развитие и оркестровку интеллекта, расположенного в разных частях интеллектуальной системы.

Среда разработки интеллекта

Для успешной работы интеллектуальной системы необходимо обеспечить качественную координацию между средой выполнения, доставкой, мониторингом и разработкой интеллекта.

Например, для точной настройки интеллекта разработчик должен иметь возможность воссоздать то, что происходило во время выполнения. Это означает, что:

- телеметрия должна охватывать тот же контекст, который входит в среду выполнения (информация о том, что делал пользователь, контент, выходные данные датчиков и т. д.);
- разработчик интеллекта должен иметь возможность обрабатывать этот контекст точно так же, как он обрабатывается во время выполнения;
- разработчик интеллекта должен иметь возможность связать контекст, обнаруженный в телеметрии, с конечным результатом взаимодействия – хорошим или плохим.

Все перечисленное становится затруднительным, если разработчики интеллекта вводят новые типы информации или контекста, которые исследуют их модели, – это называется *конструированием признаков* (feature engineering) и является ключевой частью процесса создания интеллекта. Еще одно затруднение возникает, когда система мониторинга собирает не совсем правильную информацию, что приводит к несоответствию между реальной средой выполнения и тем, что наблюдает создатель интеллекта. И наконец, иногда интеллект выполняется не на том устройстве, для которого был разработан. Возможно, среда выполнения имеет другой сопроцессор, другую графическую карту или версию математической библиотеки, отличную от той, которая применялась в среде разработки интеллекта.

В любой из подобных ситуаций могут возникнуть проблемы.

Реализация интеллекта формирует *среду разработки*, которая смягчает эти проблемы, обеспечивая разработчикам интеллекта максимально возможную согласованность компонентов.

Оркестровка интеллекта

Интеллектуальная система нуждается во внешнем управлении – *оркестровке* – по следующим причинам:

- если она попадает в некорректное состояние, кто-то должен ее перезапустить;
- если система начинает совершать грубые ошибки, кто-то должен сглаживать их, пока расследуется основная причина;

- по мере эволюции проблемы, интеллекта и базы пользователей кто-то должен иметь возможность взять под свой контроль и настроить интеллект и опыт для достижения желаемых результатов.

Например, если при разработке интеллекта получилась плохая модель (и ни одна из проверок не обнаружила этого), модель может порождать отрицательный опыт пользователя. Группа оркестровки должна иметь возможность быстро выявлять негативный отклик в телеметрии, отслеживать проблемы вплоть до конкретной модели и отключать или возвращать модель к предыдущей версии.

Если дела пойдут достаточно плохо, то руководитель группы оркестровки может принять решение остановить систему, а реализация и интеллектуальный опыт должны позаботиться о том, чтобы все прошло гладко.

Наличие хороших средств наблюдения и инструментов оркестровки позволяет интеллекту действовать более уверенно, смелее рисковать и быстрее совершенствоваться.

Итог главы

Интеллектуальная система должна быть реализована так, чтобы интеллект был связан с клиентами. Аспекты реализации включают в себя:

- выполнение интеллекта и инициализацию интеллектуального опыта;
- управление интеллектом – обновление и доставку к месту выполнения;
- сбор телеметрии о работе интеллекта и реакции пользователей для улучшения интеллекта;
- поддержку разработчиков интеллекта, позволяющую им взаимодействовать с контекстами точно так же, как пользователи;
- помощь в оркестровке интеллектуальной системы в течение ее жизненного цикла, контроль ее компонентов, устранение ошибок и т. д.

Интеллектуальная система не похожа на традиционную программу, которую разрабатывают, внедряют, доставляют и удаляют. Она больше похожа на сервис, который нужно запустить и непрерывно улучшать. Реализация интеллектуальной системы – это платформа, на которой все происходит.

Темы для размышлений

Прочитав эту главу, вы должны:

- понимать суть эффективной реализации интеллекта – что нужно, чтобы перейти от частичного интеллекта (например, модели с машинным обучением) к полнофункциональной интеллектуальной системе;
- уметь называть и описывать ключевые компоненты, составляющие реализацию интеллектуальной системы.

Рассмотрите вашу повседневную деятельность. Постарайтесь ответить на следующие вопросы:

- как может выглядеть минимальная реализация интеллектуальной системы для поддержки вашей деятельности?
- как вы думаете, какой компонент – среда выполнения, управление, телеметрия, среда разработки, оркестровка – потребует наибольшего объема инвестиций? Почему?

Глава 12

Среда выполнения интеллекта

Интеллект функционирует в *среде выполнения*. Среда отвечает за взаимодействие с остальной частью интеллектуальной системы, сбор информации, необходимой для работы интеллекта, загрузку данных на вход и интерпретацию данных на выходе модели, а также за передачу прогнозов интеллекта в систему.

В общем виде среда выполнения интеллекта может выглядеть приблизительно так:

```
Intelligence theIntelligence =
               InitializeIntelligence(<intelligence data file>,
                                      <server address>,
                                      <etc...>);

Context theContext =
               GatherContext(<sensors>,
                                      <content>,
                                      <user history>,
                                      <other system properties>,
                                      <etc...>);
Prediction thePrediction =
        ExecuteIntelligence(theIntelligence, theContext);

UpdateTheExperience(thePrediction);
```

В принципе, все просто, но, как это случается с большинством хороших вещей, на практике оборачивается значительными сложностями.

В этом разделе рассматриваются основные элементы среды выполнения интеллекта:

- сбор контекста;
- извлечение признаков;
- обновление моделей;
- выполнение моделей по контексту/признакам;
- связь полученных прогнозов с опытом.

Сбор контекста

Контекст в среде выполнения интеллекта состоит из всех данных, которые интеллект может использовать для принятия своего решения. Например:

- если интеллектуальная система пытается решить, должен ли работать садовый разбрызгиватель, контекст может включать в себя:
  - время, прошедшее с момента последнего включения разбрызгивателя;
  - прогноз на ближайшие несколько дней (будет дождь или нет);
  - количество осадков, выпавших за последние несколько дней;
  - тип ландшафта по месту проживания пользователя;
  - время суток;
  - температуру;
  - влажность
  - и т. д.;
- если интеллектуальная система ищет кошек на изображении, контекст может включать в себя:
  - необработанные изображения (интенсивность RGB попиксельно);
  - метаданные о том, как было снято изображение (экспозиция и частота кадров, время, увеличение);
  - географическое положение места съемки
  - и т. д.;
- если интеллектуальная система пытается решить, должен ли пользователь отдохнуть от работы за компьютером, контекст может включать в себя:
  - время с момента последнего перерыва;
  - действия, выполненные пользователем после перерыва (количество нажатий клавиш и движений мыши);
  - показатель того, насколько активным был пользователь;
  - время до следующей встречи пользователя (в соответствии с его календарем);
  - название приложения, с которым в данный момент взаимодействует пользователь,
  - и т. д.

Контекст может представлять собой практически любую обрабатываемую компьютером информацию, такую как URL-адрес, показания датчика с какого-либо встроенного устройства, цифровое изображение, каталог фильмов, лог действий пользователя, абзац текста, выходные данные медицинского прибора, товарные чеки из магазина, поведение программы и т. п.

Это знания, которые могут пригодиться при принятии решения о том, что должна делать интеллектуальная система. Фактически контекст обычно является надмножеством знаний, которые интеллект на самом деле использует для принятия решения.

Хороший контекст будет:

- содержать достаточно информации, чтобы интеллект был эффективным;
- содержать достаточно информации, чтобы интеллект мог расти и адаптироваться;
- содержать достаточно информации, чтобы оркестровщики интеллектуальной системы могли отследить ошибки;
- достаточно эффективным, чтобы хватало информации, собранной во время выполнения интеллекта;
- достаточно компактным, чтобы включить его в телеметрию.

Очевидно, что придется искать компромиссы. Более полный контекст дает интеллекту больший потенциал, но может столкнуться с техническими и эксплуатационными ограничениями по части использования процессора, памяти и пропускной способности сети и привести к увеличению задержки и росту объема данных.

Извлечение признаков

Сбор переменных контекста может потребовать значительных усилий. Но это еще не конец процесса. Далее следует слой кода, который преобразует эти переменные в представление для интеллекта. Этот дополнительный слой обычно называют *извлечением признаков* (feature extraction). Он является общей частью систем машинного обучения.

Есть несколько причин, по которым извлечение признаков важно для реализации интеллекта:

- код, который извлекает признаки из контекста, должен работать во время выполнения интеллекта. То есть он должен быть эффективным и надежным;
- этот код зачастую глубоко технический, вычислительно дорогой, математически сложный и недоступный для интуитивного понимания людьми, не имеющими большого опыта работы с данными;
- создатели интеллекта могут захотеть изменить этот код. Точнее, им наверняка придется периодически менять его, чтобы продолжать развивать интеллект;
- этот код тесно связан с интеллектом, поэтому код извлечения признаков и интеллект (модель) обязательно должны быть синхронизированы – в среде выполнения интеллекта должна применяться строго та же самая версия кода, которая применялась в среде разработки.

В силу этих причин извлечение признаков должно быть тщательно спланировано. Некоторые лучшие подходы включают в себя:

- обширные тестовые наборы для кода извлечения признаков, а именно:
  - модульные испытания на экстракторах;
  - тесты производительности, чтобы убедиться, что код соответствует ограничениям ЦП и ОЗУ;

  - тест, который периодически выполняет код извлечения признаков в соответствии с набором известных контекстов (например, ежедневно или при каждой регистрации) и сравнивает результаты с известным тестом. Любые изменения в выходных данных должны быть проверены инженерами и создателями интеллекта, чтобы убедиться, что все работает, как задумано;
  - тест, который подтверждает, что код извлечения признаков в среде выполнения (когда с ним взаимодействуют пользователи), делает то же самое, что и в среде разработки (когда он используется для создания моделей);
- проверку, что среда выполнения производит синхронизацию на основании информации о версии, содержащейся в интеллекте и в коде извлечения признаков;
- возможность изменять код извлечения признаков, не затрагивая приложение в целом. Например, можно отправлять новую версию кода извлечения признаков с каждым обновлением модели;
- разработку жизненного цикла для перемещения изменений кода от создателей интеллекта к инженерам и службе развертывания. Жизненный цикл должен позволять создателям интеллекта быстро оценивать эффект от изменений, а качество кода, развертываемого среди пользователей, должно быть высоким.

Обновление моделей

Модель является представлением интеллекта (мы обсудим варианты представления в следующих главах). А пока воспринимайте модель в виде файла данных. В некоторых интеллектуальных системах модели меняются каждые несколько дней. Иногда они меняются каждые несколько минут.

Частота изменения будет зависеть от следующих факторов:

1) как быстро система получает достаточно данных для улучшения моделей;
2) сколько времени занимает создание новых моделей;
3) сколько стоит распространение новых моделей;
4) в какой мере необходимые изменения связаны с критериями успеха.

В целом модели меняются намного быстрее и чаще, чем большие программы. Эффективная среда выполнения позволяет:

- легко обновить модель без необходимости перезапуска всей системы;
- убедиться, что используемые модели действительны и все компоненты (средства извлечения, модели, контекст) синхронизированы;
- упростить восстановление после ошибок в моделях путем отката к предыдущим версиям.

Одним из ключевых параметров моделей с точки зрения среды выполнения является количество места, которое они занимают на диске и в оперативной памяти. При отсутствии контроля над размером машинное обучение может

давать довольно крупные модели. Но обычно удается найти компромисс, например между размером модели и точностью, если это необходимо.

Выполнение моделей

Выполнение модели – это запрос выходного прогноза модели на основе контекста и связанных с ним функций.

В простейшем случае, когда существует одна модель, и она работает на стороне клиента, выполнение модели сводится к загрузке признаков в массив и вызову функции в готовом механизме исполнения модели. Существуют готовые библиотеки для выполнения практически любого типа моделей, которые может построить машинное обучение, и в большинстве случаев эти библиотеки вполне пригодны к использованию. Вполне вероятно, что вам даже не потребуется писать много кода для загрузки и выполнения модели.

Конечно, в некоторых случаях эти библиотеки не подойдут. В качестве примеров можно привести устройства с ограничением по процессору и ОЗУ и специальное оборудование на основе видеокарты (GPU) или программируемой логической матрицы (FPGA), где стандартные библиотеки не могут работать.

Другим вариантом выполнения является модель, работающая на сервере. В этом случае клиенту необходимо сформировать некоторое компактное представление контекста или функций, отправить его на сервер и дождаться ответа, а заодно позаботиться о правильной обработке задержки вызова и не допускать, чтобы она порождала плохой пользовательский опыт.

Когда в системе имеется более одной модели (что случается достаточно часто), запрос на выполнение должен запустить каждую из моделей, собрать результаты и объединить их в окончательный ответ. Есть много способов сделать это, в том числе:

- усреднение результатов;
- получение самого высокого (наиболее определенного) результата;
- наличие некоторых правил относительно контекста, по которым выбирают, какой модели доверять (например, одна модель хорошо оценивает дома в Беверли-Хиллс, но плохо работает с остальными локациями, другая модель работает с квартирами, но не умеет работать с частными домами);
- использование модели, которая объединяет результаты других моделей.

В главах части IV «Создание интеллекта» более подробно обсуждается сочетание моделей, а также плюсы и минусы этих и других подходов.

Результаты

Выполнение интеллекта приводит к появлению ответа. Допустим, интеллект выдает значение 0,91, что означает 91 % вероятности того, о чем спросили интеллект. Пользовательский опыт должен использовать это значение, чтобы

повлиять на действия интеллектуального опыта. Например, опыт может сравнивать значение с порогом и, если значение достаточно высокое (или низкое), инициировать запрос, автоматизировать что-либо или аннотировать некоторую информацию – все, что пожелает интеллектуальный опыт.

Нестабильность в интеллекте

Использование необработанного прогноза интеллекта может быть рискованным, потому что интеллект нестабилен по своей сути.

Например, возьмем цифровое изображение коровы. Используем модель детектора коров, чтобы оценить вероятность того, что изображение содержит корову, и получим достаточно высокий результат – 97,2 %.

Теперь подождем один день. Серверная часть системы получает новые обучающие данные, создает новую модель детектора коров и отправляет ее в среду выполнения. Применим обновленную модель к тому же фотоснимку коровы, и оценка вероятности почти наверняка будет другой. Разница может быть небольшой, например 97,21 % (по сравнению с первоначальной 97,2 %), или она может оказаться относительно большой, например 99,2 %. Новая оценка может оказаться более точной или менее точной. Но крайне маловероятно, что оценка будет одинаковой для двух разных версий интеллекта.

Незначительные изменения в контексте могут иметь аналогичные последствия. Например, наведите камеру на корову и сделайте два цифровых фотоснимка подряд, один за другим, так быстро, как только сможете. Для человека корова выглядит одинаково на обоих изображениях – фон выглядит одинаково, освещение выглядит одинаково – модель детектора коровы должна вывести одинаковую оценку вероятности присутствия коровы. Так думает человек.

Но для модели эти изображения могут выглядеть очень по-разному: цифровая фотоматрица вносит шум, и профили шума изображений будут различаться; на одном из снимков корова могла моргнуть; ветер мог изменить расположение теней на листьях; могло очень незначительно измениться положение камеры; автофокусировка камеры или коррекция экспозиции тоже могут слегка различаться.

Из-за всего этого интеллект может сделать разные оценки вероятности для двух изображений. Например, 94,1 % для первого снимка и 92,7 % для второго.

Эта разница может не иметь значения для интеллектуального опыта или, наоборот, оказаться решающей.

Попытка сделать вид, что в системах машинного обучения нет нестабильности, может привести к выработке менее эффективного (или просто плохого) интеллектуального опыта.

API интеллекта

Для решения проблемы нестабильности интеллекта полезно скрывать его за интерфейсом, который выводит наружу минимальный объем информации, достаточный для достижения целей системы:

- имея дело с вероятностью, отбрасывайте лишние разряды – например, округлите с 92,333534 % до 90 %;
- рассмотрите возможность превращения вероятностей в классификации – вместо «корова есть с вероятностью 45 %» просто скажите «здесь нет коровы»;
- используйте *квантование прогноза* – вместо точного прогноза «На изображении корова с координатами 14,92» скажите «Корова находится с левой стороны изображения».

Пороговые значения и политики, необходимые для реализации этих типов преобразований, тесно связаны с интеллектом и должны включаться в модель при каждом обновлении.

И хотя эти методы, безусловно, работают, делая вывод модели более стабильным и частично скрывая внутренние механизмы интеллекта, они не идеальны. Например, представьте, что детектор коров на 81 % уверен, что в первом кадре есть корова. Порог модели установлен на уровне 80 %. Таким образом, система создает классификацию «здесь есть корова» и воздействует на опыт пользователя.

Однако на следующем кадре детектор уверен в присутствии коровы всего лишь на 79,5 % и пользовательский опыт не реагирует. В глазах пользователя такая модель может выглядеть очень плохо.

Хорошо продуманный API в сочетании с опытом, разработанным для минимизации недостатков, необходим для эффективного интеллектуального взаимодействия с пользователем.

Итог главы

Среда выполнения интеллекта отвечает за:

- сбор контекста системы;
- преобразование этого контекста в форму, которая работает с интеллектом;
- выполнение различных компонентов, составляющих интеллект;
- объединение результатов работы интеллекта и создание хорошего интерфейса между интеллектом и опытом;
- использование выходных данных интеллекта, которые влияют на пользовательский опыт.

Некоторые ключевые компоненты, с которыми будет работать среда выполнения интеллекта, представляют собой:

- **контекст**, включая всю информацию, имеющую отношение к принятию правильного решения в интеллектуальной системе;
- **признаки** (и код извлечения признаков), которые преобразуют контекст в форму, совместимую с конкретными моделями, представляющими интеллект системы;
- **модели**, которые представляют интеллект и обычно содержатся в файлах данных, относительно часто обновляемых в течение срока службы интеллектуальной системы;

- **механизм выполнения**, который выполняет модели на признаках и возвращает прогнозы; есть много замечательных библиотек для поддержки выполнения моделей, но их часто приходится дополнять своим кодом, чтобы объединить с интеллектом. Иногда библиотеки необходимо заменить для уникальных сред выполнения;
- **результаты**, которые являются предсказаниями интеллекта. Рекомендуется скрывать необработанные результаты и создавать интеллектуальный интерфейс, который предоставляет минимальное количество информации для интеллектуального взаимодействия, а также устойчив к изменениям во времени.

Эффективная интеллектуальная среда выполнения обладает следующими основными свойствами:

- позволяет легко отслеживать ошибки;
- выполняет интеллект точно так же, как в среде разработки;
- поддерживает легкое обновление, как моделей, так и механизма извлечения признаков;
- поддерживает синхронизацию частей системы.

Темы для размышлений

Прочитав эту главу, вы должны:

- научиться проектировать среду выполнения, которая выполняет интеллект и использует его в качестве основы пользовательского опыта;
- понимать, как структурировать среду выполнения интеллекта, чтобы обеспечить обновление интеллекта и поддержать другие компоненты реализации интеллекта.

Постарайтесь ответить на вопрос:

- в чем разница между контекстом вызова интеллекта и признаками, которые использует модель с машинным обучением?

Рассмотрим интеллектуальную систему, которую вы использовали совсем недавно:

- какой тип информации может быть в контексте вызовов интеллекта?
- на высоком уровне пройдите все шаги от контекста до взаимодействия с пользователем (обдумайте все необходимые подробности того, как может работать система);
- каким образом интеллект этой системы может быть скрыт для смягчения воздействия небольших изменений интеллекта на пользователя?

Глава 13

Где расположить интеллект?

При разработке интеллектуальной системы вам нужно решить, где расположить интеллект. То есть назначить такое место, в котором вы соберете воедино модель, среду выполнения и контекст для получения прогнозов – а затем отправите эти прогнозы в интеллектуальный опыт.

Среда выполнения может быть расположена на пользовательском устройстве, и в этом случае вам нужно будет решить, как обновлять модели на всех устройствах пользователей.

Среда выполнения может находиться в службе, к которой вы предоставляете доступ, и в этом случае вам придется решить, как получить контекст (или функции) от устройства пользователя с небольшими затратами и достаточно низкой задержкой, чтобы двустороннее взаимодействие было максимально эффективным.

В этой главе обсуждается, как выбрать место для среды выполнения интеллекта. Она начинается с рассуждений о различных факторах, включая задержки и стоимость, и о том, почему они имеют значение для различных типов интеллектуального опыта. Затем рассматриваются общие принципы распределения интеллекта между клиентами и службами, включая плюсы и минусы каждого варианта.

Соображения по размещению интеллекта

Вот некоторые ключевые факторы принятия решения о том, где должен располагаться интеллект:

- задержка обновления интеллекта;
- задержка выполнения интеллекта;
- стоимость эксплуатации интеллекта;
- что происходит, когда пользователи уходят в автономный режим.

В общем, вам придется найти компромисс между всеми факторами. В данном разделе рассмотрены все факторы по очереди.

Задержка при обновлении

Одним из соображений при принятии решения о том, где располагать интеллект, является *задержка обновления* (latency in updating), с которой вы столкнетесь при попытке обновить интеллект. Новый интеллект не может принести выгоду пользователям, пока не окажется в своей среде выполнения и не заменит старый интеллект.

Задержка обновления интеллекта может быть очень короткой, когда среда выполнения находится на том же компьютере, что и среда создания интеллекта. Задержка может быть очень большой, если среда выполнения находится на клиентском компьютере, который лишь эпизодически подключается к интернету для получения новых данных.

Задержка при передаче интеллекта в среду выполнения важна, когда:

- качество интеллекта быстро улучшается;
- проблема, которую вы пытаетесь решить, быстро меняется;
- существует риск дорогостоящих ошибок.

Мы более подробно обсудим ситуации, в которых задержка обновления может вызвать проблемы, включая примеры того, как они могут повлиять на интеллектуальный опыт.

Качество интеллекта быстро улучшается

Задержка обновления интеллекта имеет значение, когда качество интеллекта быстро улучшается.

Когда вы беретесь за новую проблему, легко добиться прогресса. В ваших руках все инструменты и методы. Новые данные поступают каждый день, новые пользователи используют ваш продукт так, как вы этого не ожидали. Вас подгоняет творческое волнение, и качество интеллекта растет на глазах.

Например, представьте, что сегодня первый день, когда вы выпустили новый продукт – умные ботинки. Эта обувь автоматически регулирует плотность облегания ноги в зависимости от того, что делает ее владелец. Сидит без дела? Обувь автоматически распускает шнурки и становится просторнее, чтобы владельцу было удобно. Но если пользователь занимается физическим трудом, прыгает или бегает, обувь автоматически затягивает шнурки, благодаря чему у владельца лучше фиксируется голеностопный сустав и снижается вероятность получить травму.

Итак, вы запустили на рынок десятки тысяч умных ботинок. Вы получаете телеметрию о том, как перемещаются пользователи, когда они вручную заставляют обувь затянуть шнурки потуже или, наоборот, заставляют ее стать просторнее. Вы хотите собрать все эти данные и выпустить новый интеллект.

Если вы разработали новый интеллект, протестировали его и обнаружили, что он лишь немного лучше предыдущего, – у вас все хорошо! Не нужно беспокоиться о задержке при обновлении интеллекта, потому что вы не получите никакой заметной выгоды.

Но если новый интеллект заметно лучше – намного лучше предыдущего, – вы будете отчаянно пытаться загрузить его во все проданные ботинки. Ведь у вас есть пользователи, носящие эту обувь. Они будут разговаривать со своими друзьями. Блогеры будут писать обзоры. Вы хотите распространить новый интеллект как можно скорее!

Когда интеллект быстро улучшается, задержка в развертывании мешает использовать интеллект. Это особенно важно для проблем, которые очень обширны (в самых разных контекстах), или проблем, которые очень трудно решить (где качество интеллекта будет улучшаться в течение длительного времени).

Проблема быстро меняется

Задержка обновления интеллекта имеет значение, когда проблема быстро меняется, потому что она является открытой или имеет свойство меняться со временем.

В некоторых областях новые контексты формируются медленно, в течение недель или месяцев. Например, для строительства новых дорог требуется время; музыкальные вкусы развиваются постепенно; новый хлеб для тостеров не появляется на рынке каждый день. В этих случаях задержка развертывания нового интеллекта, измеряемая днями или неделями, может не иметь значения.

С другой стороны, в некоторых областях новые контексты появляются очень часто или внезапно. Например, новые спам-атаки происходят каждую секунду; каждый день публикуются сотни новостных статей; цунами внезапно выходят на берег; случается крах фондовых рынков. В подобных случаях важно иметь возможность быстрого развертывания нового интеллекта.

Есть два важных аспекта того, как меняются проблемы:

1. Как быстро появляются новые контексты?
2. Как быстро исчезают существующие контексты?

В тех областях, где добавляются новые контексты, но старые контексты остаются актуальными еще достаточно долго, эрозия интеллекта происходит относительно медленно. Например, когда построена новая дорога, ваш интеллект может не знать, что с ней делать, но все равно будет отлично работать на всех существующих дорогах, о которых он знает.

Эрозия интеллекта быстро происходит в тех областях, где новые контексты вытесняют старые, а интеллект, который был вчера, больше не нужен сегодня. Например, когда новая песня выходит на вершину хит-парадов и вытесняет старую песню, или когда спамер начинает новую рассылку и прекращает старую.

Когда проблема часто меняется таким образом, что ухудшает качество существующего интеллекта, задержка в обновлении интеллекта может стать решающим фактором успеха интеллектуальной системы.

Риск дорогостоящих ошибок

Задержка обновления интеллекта имеет значение, когда интеллектуальная система может совершать дорогостоящие ошибки, которые необходимо быстро исправить.

Интеллект совершает всевозможные ошибки. Некоторые из этих ошибок не так уж плохи, особенно когда интеллектуальный опыт помогает пользователям справиться с ними. Но некоторые ошибки становятся реальной проблемой.

Рассмотрим пример умной обуви, где шнурки автоматически ослабляются или затягиваются в зависимости от того, что делает владелец. Представьте себе, что какой-то небольшой процент пользователей нервно переминается с ноги на ногу, что заставляет обувь болезненно зажимать ноги.

Или представьте пользователей, которые играют в бейсбольной команде на правой базе (right field). Девяносто девять процентов времени они стоят там, считая ворон, а затем один процент времени дико бегают, пытаясь поймать мяч. Возможно, умная обувь не поддерживает такой режим – кроссовки слетают с ног, пользователи поскальзываются на траве и получают травмы.

Когда ошибки вашей интеллектуальной системы имеют высокую стоимость, и эти затраты не могут быть уменьшены с помощью хорошего пользовательского опыта, задержка обновления может оказаться очень болезненной. Представьте себе, что пользователи звонят, ругаются, жалуются на ущерб, который вы им причинили, – день за днем, пока вы ждете распространения нового интеллекта. Это не только причиняет вам дискомфорт, но и может поставить под угрозу весь ваш бизнес.

Задержка выполнения

Еще одним фактором принятия решения о том, где должен располагаться интеллект, является *задержка выполнения* (latency in execution) интеллекта.

Чтобы получить результат работы интеллекта, система должна собрать контекст, извлечь из него признаки, перенести признаки туда, где находится интеллект (иногда переносится весь контекст, а извлечение признаков происходит позже), подождать, пока сработает интеллект, затем подождать, пока результат выполнения интеллекта вернется к опыту, и лишь потом обновить опыт. Каждый из этих шагов может привести к задержке при выполнении интеллекта.

Задержка выполнения может быть короткой, когда среда выполнения расположена на том же компьютере, что и сам интеллект. Однако задержка может оказаться большой, когда среда выполнения и интеллектуальный опыт расположены на разных компьютерах.

Задержка при выполнении интеллекта может быть проблемой, если:

- пользователи напряженно ждут ответа;
- правильный ответ меняется быстро и резко.

Задержку при выполнении и степень ее влияния на пользователей бывает трудно предсказать. Иногда пользователи не обращают внимания на задержку. Иногда небольшая задержка сводит их с ума и полностью разрушает опыт. Попытайтесь спроектировать опыт, в котором компромисс между разными задержками можно легко изменить во время оркестровки, чтобы протестировать различные варианты на реальных пользователях.

Задержка в интеллектуальном опыте

Задержка выполнения приобретает особое значение, когда пользователи замечают ее, особенно когда им приходится ждать ответа, прежде чем они смогут продолжить. Это может произойти, если:

- **обращение к интеллекту является важной частью предоставления опыта.** Представьте себе приложение, которому требуется несколько обращений к интеллекту, прежде чем оно сможет выстроить визуальный интерфейс. Возможно, перед его отображением приложению необходимо выяснить, является ли контент потенциально вредным или оскорбительным. Если пользователю ничего не нужно делать во время ожидания ответа, то задержка может стать раздражающей;
- **обращение к интеллекту является интерактивным.** Представьте себе приложение, в котором пользователь напрямую взаимодействует с интеллектом. Может быть, он щелкает выключателем и ожидает, что свет включится мгновенно. Если выключатель должен обратиться к интеллекту, прежде чем включить свет, а задержка составляет сотни или тысячи миллисекунд – пользователь может отбить пальцы ног о мебель.

С другой стороны, асинхронный интеллектуальный опыт (например, принятие решения о необходимости отобразить окно уведомления) менее чувствителен к задержке выполнения.

Правильный ответ резко меняется

Задержка выполнения имеет значение, когда правильный ответ меняется быстро и резко.

Представьте себе интеллект, который управляет дроном. Дрон направляется к цели, и правильный ответ – лететь прямо. Нет проблем. Но вдруг кто-то встает на пути дрона. Или, допустим, дрон летит в прекрасный тихий солнечный день, а затем внезапно налетает мощный ветер. В подобных ситуациях правильный ответ для дрона меняется очень резко и быстро.

Когда правильный вектор действий для интеллектуальной системы быстро и радикально меняется, задержка при выполнении интеллекта может привести к серьезным сбоям.

Стоимость эксплуатации

Еще одно соображение при принятии решения о том, где расположить интеллект, – это стоимость эксплуатации интеллектуальной системы.

Распространение и выполнение интеллекта расходует ресурсы процессора, оперативную память и пропускную способность сети. Это стоит денег. Основные факторы, влияющие на стоимость эксплуатации интеллектуальной системы, включают в себя:

- стоимость распространения интеллекта;
- стоимость выполнения интеллекта.

Стоимость распространения интеллекта

За распространение интеллекта платит как сервис, так и пользователь, обычно в форме оплаты передачи данных по сети. Каждый новый вариант интеллекта должен быть где-то размещен (например, в облачном сервисе), а среда выполнения должна периодически проверять наличие обновлений и загружать новые версии. Расходы пропорциональны количеству сред выполнения, в которых работает интеллект (количество клиентов или серверов, на которых он размещен), размеру обновлений интеллекта и частоте обновлений. Для интеллектуальных систем в масштабе целого интернета стоимость распространения интеллекта может быть очень большой.

Также важно учитывать затраты для пользователей. Если основной вариант применения ориентирован на пользователей широкополосной связи, то распространение моделей не вызовет проблем и обойдется относительно недорого. Но когда интеллект расположен на мобильных устройствах или в местах, где использование сети строго лимитировано и дорого стоит, объем трафика становится важным фактором.

Стоимость выполнения интеллекта

Выполнение интеллекта также может приводить к оплате передачи данных, когда интеллект расположен на сервере и клиенты должны отправлять контекст (или признаки) на сервер и получать ответ. В зависимости от размера контекста и частоты обращений отправка контекста на сервер может стоить дороже, чем отправка ответа клиенту. Имейте в виду, что телеметрия и мониторинг также собирают некоторую контекстную и функциональную информацию от клиентов – можно совместить данные и снизить расходы на трафик.

Выполнение интеллекта также загружает процессор и оперативную память. Преимущество размещения интеллекта на стороне клиента заключается в том, что расходы на аппаратную часть покрывают пользователи. Но некоторые типы интеллекта могут быть очень ресурсоемкими, а некоторые клиенты (например, мобильные) имеют ограничения по ресурсам, что делает невозможным размещение среды выполнения на стороне клиента. В этих случаях размещение среды выполнения на сервере (возможно, на специальном оборудовании, таком как графические процессоры или FPGA) может обеспечить гораздо более эффективную работу интеллекта.

При рассмотрении стоимости эксплуатации стремитесь к реализации, которая:

1) заботится о пользователе и не заставляет его оплачивать значимые для него расходы (включая передачу данных, выполнение в мобильном устройстве и тому подобные траты);
2) позволяет пользователям оплачивать расходы по частям, которые для них не критичны (а вам не придется покупать сразу много серверов);
3) уравновешивает все затраты на эксплуатацию интеллектуальной системы и масштабируется вместе с ростом числа пользователей и качества интеллекта.

Автономная работа

Задумайтесь о том, должна ли интеллектуальная система функционировать, когда она находится в автономном режиме и не может связаться ни с какими службами.

Интеллектуальной системе не всегда нужно работать в автономном режиме. Например, система прогнозирования автомобильных пробок не должна работать, когда ее пользователь находится в самолете над Тихим океаном. Но иногда интеллектуальным системам необходимо работать в автономном режиме, например когда интеллект работает в ограниченной среде (допустим, в военном транспортном средстве) или в жизненно важном устройстве.

Если необходимо сохранить работоспособность в автономном режиме, какая-то версия интеллекта должна располагаться на стороне клиента. Это может быть полный интеллект или сокращенная версия общего интеллекта, позволяющая пользователям продолжать работу, пока восстанавливается подключение к сети.

Подходы к размещению интеллекта

В этом разделе более детально рассматриваются некоторые из вариантов размещения интеллекта. Они включают в себя следующие шаблоны:

- статический интеллект;
- интеллект на стороне клиента;
- интеллект на стороне сервера;
- внутренний (кешированный) интеллект;
- гибридный интеллект.

В данном разделе обсуждается каждый из этих подходов и исследуется, насколько хорошо они работают по четырем параметрам: задержка при обновлении, задержка при выполнении, стоимость операции и автономная работа.

Статический интеллект в составе продукта

Вы можете реализовать *статический интеллект* (static intelligence) без каких-либо компонентов интеллектуальной системы. Просто соберите кучу обучающих данных, создайте модель, свяжите ее с вашим программным обеспечением и отправьте пользователям.

Это очень похоже на развертывание традиционной программы. Вы создаете интеллект, тестируете его как можно лучше – в лаборатории или с клиентами в фокус-группах и бета-тестах, – настраиваете его, пока он вам не понравится, и отправляете его в мир.

Преимущество этого подхода в том, что так дешевле проектировать систему. Для многих проблем бывает достаточно статического интеллекта. Он может работать в ситуациях, когда нет возможности замкнуть петлю связи между пользователями и создателями интеллекта. Однако остается возможность обратной связи через обзоры и обращения в службу поддержки, а также возмож-

ность обновления интеллекта с помощью традиционных методов обновления программного обеспечения.

Недостатком является сложность обновления интеллекта, что делает этот подход малопригодным для открытых, меняющихся во времени или сложных задач.

Задержка при обновлении: плохо
Задержка при выполнении: отлично
Стоимость эксплуатации: низкая
Автономная работа: да

Краткое описание недостатков: сложно обновить интеллект. Есть риск неизмеряемых ошибок интеллекта. Нет данных для улучшения интеллекта.

Интеллект на стороне клиента

Интеллект на стороне клиента (client-side intelligence) выполняется полностью на клиентском оборудовании. То есть среда выполнения интеллекта целиком расположена на стороне клиента, который периодически загружает новые версии интеллекта.

Загрузка обычно включает в себя новые модели, новые пороговые значения для интерпретации выходных данных моделей (если интеллект инкапсулирован в API – так и должно быть) и иногда новый код извлечения признаков.

Размещение на стороне клиента обычно позволяет использовать относительно большие ресурсы для выполнения интеллекта. Дело в том, что интеллект может потреблять свободные ресурсы клиента при относительно небольших затратах (за исключением, пожалуй, питания мобильного устройства). Другая выгода заключается в том, что к задержке времени выполнения не добавляется задержка обращения к серверу, поэтому остается больше времени на выполнение, прежде чем пользователь негативно отреагирует на задержку.

Основная проблема при данном подходе заключается в том, чтобы решить, когда и как выгрузить новый интеллект клиентам. Например, если объем интеллекта составляет десять мегабайт, а число клиентов – сто тысяч, то каждое обновление интеллекта сгенерирует примерно терабайт сетевого трафика. Кроме того, модели не склонны к хорошему сжатию или работают с накопительными обновлениями, поэтому обычно за трафик приходится расплачиваться сполна.

Другая потенциальная сложность интеллекта на стороне клиента связана с разбросом версий интеллекта. Одни пользователи будут находиться в автономном режиме и не получат всех обновлений, которые вы хотели отправить им. Другие пользователи могут отказаться от обновлений (возможно, с помощью сетевых фильтров), потому что не хотят загружать файлы из сети на свои машины. Подобные ситуации затрудняют распознавание пользовательских проблем.

Модели на стороне клиента фактически передают ваш интеллект в руки любому, кто захочет взглянуть на него: может быть, конкуренту, или тому, кто хочет злоупотребить вашим сервисом или вашими пользователями – например, спамеру. Когда у кого-то есть ваша модель, он может организовать злонамеренное тестирование и даже автоматизировать эти тесты. Он может выяснить, какой тип входных данных заставляет модель дать один ответ и при каких входных данных она дает другой ответ. Злоумышленники могут подобрать такие модификации своего контекста (электронная почта, веб-страница, продукт и т. д.), которые вынуждают вашу модель делать нужную им ошибку.

Задержка при обновлении: переменная
Задержка при выполнении: отлично
Стоимость эксплуатации: зависит от частоты обновления
Автономная работа: да

Краткое описание недостатков. Передача сложного интеллекта клиентам может быть дорогостоящей. Трудно синхронизировать обновления на каждом клиенте. Клиентские ресурсы могут быть ограничены. Раскрывает устройство интеллекта любому желающему.

Интеллект на стороне сервера

Серверный интеллект (server-centric intelligence) выполняется в режиме реального времени как удаленная служба. То есть клиент получает контекст (или признаки) и отправляет его на сервер, а сервер выполняет интеллект по набору признаков и возвращает результат клиенту.

Использование интеллекта на стороне сервера позволяет быстро обновлять модели путем загрузки новых моделей на сервер (или серверы), где выполняется интеллект. Это также упрощает телеметрию и мониторинг, поскольку большая часть данных, которые необходимо собрать, уже будет находиться в службе как часть вызовов интеллекта.

Но серверный интеллект должен масштабироваться по мере расширения базы пользователей. Например, если в секунду поступает сотня запросов на запуск интеллекта, служба должна быть в состоянии выполнить запросы очень быстро и, возможно, параллельно.

Задержка при обновлении: хорошо
Задержка при выполнении: переменная, но включает в себя передачу в оба конца
Стоимость эксплуатации: инфраструктура обслуживания и пропускная способность могут давать значительные затраты; может расходовать значительный трафик пользователя
Автономная работа: нет

Краткое описание недостатков: задержка вызова интеллекта. Серверная инфраструктура и трафик. Стоимость трафика пользователя. Высокая стоимость серверов, которые могут выполнять интеллект в режиме реального времени.

Внутренний (кешируемый) интеллект

Внутренний (кешируемый) интеллект (back-end/cashed intelligence) включает в себя запуск автономного интеллекта, кеширование результатов и предоставление этого кеша по запросу. Внутренний интеллект может быть эффективен при анализе ограниченного числа сущностей, таких как все электронные книги в библиотеке, все песни, которые может порекомендовать сервер, или все почтовые индексы регионов, где продается умный разбрызгиватель. Внутренний интеллект также может использоваться, когда не существует конечного списка сущностей, но контексты меняются медленно.

Например, разбрызгиватель впервые продали в новый регион. Служба никогда не встречала почтовый индекс этого региона, поэтому предлагает настройки по умолчанию для оптимального времени полива. Но затем запускается сервер, исследующий всю информацию, которую он может найти о почтовом индексе, чтобы создать хороший план полива, и добавляет этот план полива в свой кеш. В следующий раз, когда разбрызгиватель будет продан в регионе с этим почтовым индексом, служба сразу ответит, как лучше организовать там полив (и, возможно, даже обновит план полива для того несчастного первого парня, который инициировал весь процесс).

Внутренний интеллект может позволить себе тратить больше ресурсов и времени на каждое решение интеллекта, чем другие варианты. Например, представьте себе сверхсложную модель плана полива, которая в течение часа работает на мощном сервере, чтобы решить, как поливать каждый регион. Она анализирует спутниковые снимки, схемы дорожного движения, миграцию птиц и лягушек в регионах – все, что ей нужно. Выполнение такой модели на встроенном микрокомпьютере в разбрызгивателе может занять месяцы, что неприемлемо. Модель не может постоянно работать на сервере, который должен отвечать на сотни вызовов в секунду. Зато она может периодически работать в фоновом режиме и сохранять результаты анализа в кеш.

Кеши интеллекта могут жить на серверах; их части также могут быть распространены среди клиентов.

Один из недостатков внутреннего интеллекта заключается в том, что изменение моделей может оказаться более дорогостоящим, поскольку все предыдущие кешированные результаты могут потребовать повторного вычисления.

Другим недостатком является то, что он работает только тогда, когда контекст вызова можно использовать для «поиска» соответствующего интеллекта. Это работает, когда контекст описывает конкретный объект, такой как веб-страница, место или фильм, но не работает, когда контекст описывает менее конкретные вещи, такие как сгенерированный пользователем блок текста, последовательность выходных данных из матрицы датчиков или видеоклип.

Задержка при обновлении: переменная
Задержка при выполнении: переменная
Стоимость эксплуатации: в зависимости от объема использования
Автономная работа: частичная

Краткое описание недостатков: подход неэффективен, когда контексты быстро меняются или когда быстро меняется правильный ответ для контекста. Дорого менять модели и перестраивать кеш интеллекта. Ограничивает интеллект вещами, которые можно искать.

Гибридный интеллект

На практике бывает выгодно использовать *гибридный интеллект* (hybrid intelligence), который объединяет несколько подходов к выбору места размещения.

Например, система может использовать внутренний интеллект для глубокого анализа популярных товаров и клиентский анализ для оценки всего остального.

Или система может в большинстве случаев использовать интеллект на стороне клиента, но обращаться к серверу, когда решение имеет серьезные последствия.

Гибридные интеллекты могут маскировать слабости своих различных компонентов.

Но разработка и оркестровка гибридных интеллектов иногда оказываются более сложными. Например, если система дает неправильный ответ, то в какой части интеллекта произошла ошибка? В модели на стороне клиента? В интеллекте, который был кеширован на сервере? При каком-то тонком взаимодействии между ними?

Иногда бывает трудно узнать, в каком состоянии были все эти компоненты на момент возникновения ошибки.

Тем не менее большинство крупных интеллектуальных систем использует гибридный подход при выборе места размещения интеллекта.

Итог главы

Выбор места, где будет располагаться ваш интеллект, является важной частью создания успешной интеллектуальной системы. Расположение интеллекта может повлиять на следующие параметры:

- **задержка при обновлении интеллекта** – зависит от того, сколь долгий путь должен пройти интеллект от среды создания к среде выполнения и как часто среда выполнения подключается к сети для получения обновления;
- **задержка при выполнении интеллекта** – зависит от того, как долго перемещаются контекст и признаки из интеллектуального опыта в среду выполнения интеллекта и как скоро возвращается ответ;
- **стоимость эксплуатации интеллектуальной системы** – зависит от того, сколько трафика вам нужно оплатить для перемещения интеллекта, контекста и признаков, а также от того, какие ресурсы нужны для выполнения интеллекта;

- **способность системы работать в автономном режиме** – зависит от того, какая часть интеллекта может работать на стороне клиента, пока он не имеет связи с сервером.

Есть много вариантов взаимного сочетания этих свойств. Вот несколько основных шаблонов:

- **статический интеллект** – полностью размещен на стороне клиента, вообще не подключенного к удаленным службам;
- **интеллект на стороне клиента** – полностью размещен на стороне клиента, но подключается к удаленным службам для обновления интеллектуальных данных и телеметрии;
- **интеллект на стороне сервера** – полностью размещен на сервере и требует обращения к службе каждый раз, когда необходимо выполнить интеллект для заданного контекста;
- **внутренний (кешируемый) интеллект** – он выполняет интеллект в автономном режиме для общих контекстов и сохраняет готовые ответы в кеш;
- **гибридный интеллект** – это практика большинства крупномасштабных интеллектуальных систем, которая сочетает в себе множество других подходов для достижения целей системы.

Темы для размышлений

Прочитав эту главу, вы должны:

- знать все места, где может располагаться интеллект, от клиента до сервера, а также плюсы и минусы каждого варианта;
- понимать последствия размещения интеллекта и уметь разрабатывать реализацию, которая лучше всего подходит для вашей системы.

Постарайтесь ответить на следующие вопросы:

- представьте себе систему с аналитической моделью размером 1 Мб и контекстом 10 Кб для каждого интеллектуального вызова. Если модель необходимо обновлять ежедневно, при каком количестве клиентов и вызовов интеллекта имеет смысл использовать интеллект на стороне сервера, а не клиента?
- если вашему приложению нужно работать на самолете над Тихим океаном (без подключения к сети), какие варианты размещения интеллекта вы можете предложить?
- что, если вашему приложению следует работать в самолете, но основной вариант использования – у пользователя дома? Как оптимизировать систему для наиболее эффективной работы в обоих случаях?

Глава 14

Управление интеллектом

Интеллект в составе интеллектуальной системы проходит путь от создания, проверки, развертывания и ввода в эксплуатацию до постоянного мониторинга. *Управление интеллектом* (intelligence management) заполняет разрыв между созданием интеллекта (которое обсуждается в части IV этой книги) и оркестровкой интеллекта (которая обсуждается в части V). Цель управления интеллектом – простое и безопасное развертывание нового интеллекта и предоставление доступа пользователям.

В простейшем случае управление интеллектом может представлять собой ручное копирование файлов модели в каталог развертывания, где среда выполнения забирает их и открывает доступ пользователей к интеллекту. Но в больших и сложных интеллектуальных системах процесс управления интеллектом может (и, вероятно, должен) быть гораздо более разноплановым.

В этой главе рассмотрены некоторые проблемы управления интеллектом. Затем мы обсудим, как управление интеллектом помогает поддерживать гибкость вашей интеллектуальной системы, вместе с тем проверяя интеллект и предоставляя доступ пользователям.

Механизм управления интеллектом

Управление интеллектом включает в себя всю работу по извлечению информации из того места, где она была создана, и ее размещению там, где она будет взаимодействовать с пользователями, включая следующие функции:

- **проверка работоспособности интеллекта**, чтобы убедиться, что он будет работать правильно;
- **развертывание интеллекта** в среде выполнения;
- **контролируемый ввод в эксплуатацию**;
- **отключение интеллекта**, который больше не востребован.

Механизм управления интеллектом может быть простым (например, набор инструкций, которые оператор системы должен пошагово выполнять вручную) или частично автоматизированным (как набор инструментов командной строки, каждый из которых делает свою работу). Управление интеллектом может быть достаточно комфортным – например, графический интерфейс, который позволяет оркестровщикам интеллекта загружать, перемещать и включать или

отключать интеллект одним щелчком мыши. Хорошая система управления интеллектом обладает следующими свойствами:

- имеет набор инструментов и сценариев для управления интеллектом в вашей среде;
- предохраняет от ошибок;
- не добавляет слишком большую задержку;
- обеспечивает хороший компромисс между вовлеченностью пользователя и стоимостью реализации.

Организация управления интеллектом является трудной задачей из-за сложной архитектуры систем, высокой частоты обновления и человеческого фактора.

Сложная архитектура систем

Интеллектуальные системы могут быть весьма сложными. Например:

- интеллект может располагаться в нескольких местах между клиентом и сервером, включая интеллект на стороне клиента, один или несколько интеллектов на стороне сервера и кешированный интеллект;
- интеллект может поступать из десятков различных источников – некоторые из них являются продуктом машинного обучения, иные созданы людьми, а кое-какие разработаны где-то вне вашей организации;
- разные части интеллекта могут зависеть друг от друга, но обновляться с разной частотой и разными людьми.

Поскольку интеллектуальные системы постоянно развиваются, не всегда удается просто взять новую версию интеллекта и развернуть ее для пользователей.

Высокая частота обновления

Интеллектуальные системы в течение своего жизненного цикла многократно обновляют интеллект. Например, если система работает в течение трех лет, то общее количество обновлений составит:

- обновление интеллекта *один раз в неделю* – около ста шестидесяти раз;
- обновление интеллекта *один раз в день* – около тысячи раз;
- обновление интеллекта *один раз в час* – примерно двадцать шесть тысяч раз;
- обновление информации *один раз в минуту* – около полутора миллионов раз.

Это довольно большие числа, а значит, процесс управления интеллектом должен быть надежным – иметь низкий уровень ошибок и не требовать чрезмерных человеческих усилий.

Человеческий фактор

Интеллектом могут управлять пользователи с разными уровнями квалификации и опытом:

- эксперты, которые хорошо разбираются во внедрении интеллектуальных систем;
- специалисты машинного обучения, которые не являются хорошими инженерами;
- сотрудники, которые не являются техническим персоналом, но вручную исправляют дорогостоящие ошибки, которые совершает система;
- новые сотрудники;
- недовольные сотрудники.

Упрощенное и менее подверженное ошибкам управление интеллектом может принести большие дивиденды за счет гибкости интеллектуальной системы в процессе повседневного развития.

Проверка работоспособности интеллекта

Быстрая автоматизированная система проверки работоспособности – это система безопасности для разработчиков интеллекта, позволяющая им уверенно вводить новшества и направлять свою энергию на создание более совершенного интеллекта, а не на перекрестные проверки и запоминание порядка выполнения тестов. Надежная система безопасности должна гарантировать, что новый интеллект:

- совместим с интеллектуальной системой;
- работает и отвечает всем требованиям ко времени выполнения;
- не допускает очевидных ошибок.

В правильно организованной системе управления интеллектом должно быть проще развернуть новый интеллект через систему автоматической проверки работоспособности, чем обойтись без проверки.

Теперь мы рассмотрим категории проверок по очереди.

Проверка на совместимость

Без ошибок не обойтись. Иногда создатели интеллекта неправильно форматируют данные, забывают запустить конвертер для файлов модели, обучают модель на данных из поврежденного файла телеметрии, или среда обучения не вовремя дает сбой и формирует поврежденную модель. Управление интеллектом – это отличное «узкое место», где можно обнаружить множество простых и распространенных ошибок, прежде чем они превратятся в опасные проблемы. Вот несколько моментов, которые нужно проверить, чтобы убедиться в совместимости интеллекта:

- файл данных интеллекта правильно отформатирован и будет загружен в среду выполнения;
- новый интеллект синхронизирован с экстрактором признаков, который развернут в настоящее время;
- новый интеллект синхронизирован с другим интеллектом в системе или будет развернут одновременно с ним;

- разворачиваемый интеллект содержит все необходимые метаданные (например, пороговые значения, необходимые для привязки к интеллектуальному опыту);
- стоимость развертывания нового интеллекта разумно обоснована (с точки зрения затрат на трафик и т. п.).

Все это статические тесты, прохождение которых должно легко отслеживаться в автоматическом режиме, не вызывать больших задержек и не требовать участия человека (если не возникает проблема).

Проверка ограничений на выполнение

Также важно проверить, соответствует ли новый интеллект любым ограничениям со стороны среды выполнения, в том числе:

- новый интеллект не занимает слишком много оперативной памяти во время выполнения;
- новый интеллект соответствует целевым показателям производительности среды выполнения (в широком диапазоне контекстов);
- новый интеллект будет взаимодействовать с пользователями в среде выполнения точно так же, как он это делал в среде создания (обработка контекста, извлечение признаков, выполнение интеллекта и т. д.).

Эти тесты требуют:

- выполнять интеллект в тестовой среде, которая отражает среду реального взаимодействия с пользователями;
- иметь возможность загружать контексты и выполнять интеллект на заданном наборе признаков, измерять потребление ресурсов и сравнивать результаты с известными правильными ответами;
- иметь набор тестовых контекстов, обеспечивающих хорошее покрытие условий, с которыми сталкиваются ваши пользователи.

Это динамические тесты, которые можно автоматизировать. Они будут вносить некоторую задержку в зависимости от того, сколько тестовых контекстов вы используете. Они не требуют особого человеческого контроля (если не возникает проблема) – результат прохождения легко определяется автоматически.

Проверка на очевидные ошибки

Разработчики не должны создавать интеллект, который делает очевидные ошибки. Вы можете сказать им это (и я скажу им позже в этой книге) – но проверка не будет лишней. Система управления интеллектом должна проверить, что:

- новый интеллект обладает «разумной» точностью на контрольном наборе (это контексты, которые создатели интеллекта никогда не увидят, – никакого мошенничества);
- новый интеллект никогда не делает ошибок на наборе критических контекстов, для которых крайне нежелательны ошибки;

- новый интеллект не совершает значительно более дорогостоящих ошибок, чем предыдущий;
- новый интеллект не фокусирует свои новые ошибки на какой-либо критической группе пользователей или контекстах.

Если какой-либо из этих тестов не пройден, развертывание интеллекта должно быть приостановлено для изучения проблемы человеком.

Это динамические тесты, которые можно автоматизировать. Они будут вводить некоторую задержку (в зависимости от того, сколько тестовых контекстов вы используете). Они несколько субъективны в том смысле, что для оценки значения колебаний точности требуется участие человека.

Пробный запуск интеллекта

После того как интеллект прошел серию проверок в автономном режиме, его можно испытать на реальных пользователях одним из следующих способов:

- массовое развертывание;
- тихий интеллект;
- ограниченное развертывание;
- флайтинг;
- отмена обновления.

В этом разделе мы обсудим перечисленные способы, а также некоторые их плюсы и минусы, чтобы вы могли решить, какой из них подходит для вашей интеллектуальной системы.

Однократное развертывание интеллекта

В простейшем случае *однократное развертывание* (single deployment) интеллекта сразу для всех пользователей можно осуществить одним из следующих способов:

- компонуем новый интеллект в файл, передаем файл в среду выполнения на стороне клиента и перезаписываем старый интеллект;
- копируем новый интеллект на сервер, где размещена среда выполнения, и перезапускаем процесс среды;
- разделяем интеллект на часть, которая выполняется на клиенте, и часть, которая работает на сервере, и разворачиваем нужные части в нужных местах.

Однократное развертывание интеллекта является простым в управлении и относительно простым в реализации. Но этот метод не прощает ошибки. Если есть проблема, с ней столкнутся одновременно все пользователи.

Представьте, что вы создали умную стиральную машину. Положите в нее одежду, закройте дверцу, и машина начинает стирать – больше не надо возиться с настройками. Допустим, система работает хорошо, но вы решили улучшить интеллект за одно развертывание. Вы рассылаете новый интеллект на десятки тысяч умных стиральных машин, а затем начинаете получать массу сообщений о том, что стиральные машины портят одежду пользователей. По-

чему это происходит? Вы пока не знаете. Но проблема затрагивает всех ваших пользователей, а у вас нет хорошего решения.

Однократное развертывание может быть эффективным, если:

- вы хотите, чтобы все было просто;
- у вас есть отличные автономные тесты для выявления проблем.

Однократное развертывание может быть проблематичным, если:

- ваша система допускает дорогостоящие ошибки;
- ваша способность выявлять и исправлять проблемы ограничена.

Тихий интеллект

Тихий интеллект (silent intelligence) разворачивается параллельно существующему интеллекту, и они оба участвуют во взаимодействиях с пользователем. Существующий интеллект используется для управления интеллектуальным опытом (что видят пользователи). Тихий интеллект никак не обнаруживает себя – его предсказания просто записываются в телеметрию, чтобы вы могли проверить их и посмотреть, хорошо ли работает новый интеллект.

Особенно любопытными являются условия, при которых существующий интеллект и тихий интеллект принимают разные решения. Это контексты, где новый интеллект либо лучше, либо хуже старого. Проверка нескольких сотен таких контекстов вручную может дать уверенность в том, что новый интеллект безопасен для включения (или показать, что это не так).

Интеллект можно запустить в тихом режиме на любое время: тысяча выполнений, несколько часов, дней или недель – столько, сколько вам нужно, чтобы обрести уверенность в нем.

Если новый интеллект достойно проявит себя во время тихой оценки, он может заменить предыдущий интеллект. Но если новый интеллект окажется хуже, он может быть удален без какого-либо влияния на пользователей и без проблем.

Тихий интеллект может быть эффективен, если:

- вам нужна дополнительная проверка качества вашего интеллекта;
- вы хотите подтвердить, что ваш интеллект дает те же ответы во время выполнения, что и при создании;
- у вас очень масштабная или открытая задача, и вы хотите обрести уверенность в том, что ваш интеллект будет хорошо работать в новых и редких контекстах (которые могут отсутствовать в вашей среде создания интеллекта).

Тихий интеллект может быть проблематичен, если:

- вам не нужна сложность (или стоимость ресурсов) одновременной работы нескольких интеллектуалов;
- задержка обновления имеет решающее значение, и вы не можете ждать, чтобы проверить свой новый интеллект в тихом режиме;
- трудно оценить эффект тихого интеллекта, не раскрывая его пользователям, – вы можете видеть, что он сделал бы, но не результат, который бы получил пользователь.

Ограниченное развертывание

Ограниченное развертывание (controlled rollout) открывает новый интеллект для небольшой части пользователей, а остальные пользователи остаются со старым интеллектом. Система собирает телеметрию от новых пользователей и использует ее для проверки того, что новый интеллект работает должным образом. Если новый интеллект хорош, он распространяется на большее количество пользователей; если у нового интеллекта есть проблемы, его можно свернуть, не нанося слишком много урона.

Ограниченное развертывание отличается от тихого интеллекта двумя важными факторами:

1) телеметрия из ограниченной части включает в себя влияние интеллекта на поведение пользователя. Вы можете знать, что делал новый интеллект и как реагировали пользователи;
2) ограниченное развертывание запускает один интеллект на клиента, хотя при этом запускает несколько процессов интеллекта на сервере – такое развертывание расходует меньше ресурсов на клиента, но может быть более сложным в управлении.

Интеллект можно развернуть с использованием различных политик для обеспечения баланса между задержкой и безопасностью, в том числе:

- развертывание для дополнительной небольшой части ваших пользователей каждые несколько часов, если телеметрия показывает, что дела идут хорошо;
- развертывание в небольшую тестовую группу на несколько дней, а затем посещение всех пользователей группы и поиск проблем;
- переход на альфа-тестирование, затем на бета-тестирование, на ранних пользователей и, наконец, массовое развертывание.

Ограниченное развертывание может быть эффективным, если:

- вы хотите увидеть, как пользователи будут реагировать на новый интеллект, одновременно ограничивая потенциальный ущерб, который может нанести новый интеллект;
- вы хотите, чтобы некоторые из ваших пользователей столкнулись с проблемами, чтобы помочь вам проверить интеллект.

Ограниченное развертывание может быть проблематичным, если:

- вы не хотите иметь дело со сложностью одновременного развертывания нескольких версий интеллекта;
- вы беспокоитесь о редких событиях. Например, при ограниченном развертывании на 1 % пользователей вряд ли возникнет проблема, которая затрагивает только 1 % пользователей.

Флайтинг

Флайтинг (flighting) – это особый тип ограниченного развертывания, который предоставляет разные версии интеллекта разным группам пользователей для получения статистических данных об интеллекте.

Представьте себе двух разработчиков интеллекта, которые приходят к вам и говорят, что у них есть гораздо лучший интеллект для вашей интеллектуальной системы. Один из интеллектов быстр, но не слишком точен. Другой очень медленный, но имеет гораздо лучшую точность.

Что будет лучше для достижения целей вашей интеллектуальной системы? Что больше понравится пользователям? Что улучшит взаимодействие? Что приведет к лучшим результатам?

Вы можете создать фокус-группы. Вы можете позволить создателям интеллекта спорить об этом. Наконец, вы можете дать им боевые топоры, поставить их на арену и позволить победителю выбрать, какой интеллект использовать…

Или вы можете развернуть каждую версию для 1000 своих пользователей и отслеживать их результаты в течение следующего месяца:

- использует ли одна из пробных групп приложение чаще, чем другая?
- получает ли одна из пробных групп лучшие результаты, чем другая?
- относится ли одна из пробных групп к вашему приложению лучше, чем другая?

Флайтинг помогает вам понять, как разные версии интеллекта взаимодействуют с остальной частью вашей интеллектуальной системы.

Флайтинг может быть эффективным, если:

- вы рассматриваете небольшое количество крупных изменений и хотите знать, какое из них лучше;
- вам необходимо отслеживать изменения в течение длительного периода, чтобы вы могли делать статистически обоснованные заявления о том, как изменения влияют на результаты, опережающие индикаторы и цели организации.

Флайтинг может быть проблематичным, если:

- вам нужно быстро выполнять итерации и вносить множество изменений за короткий период времени;
- разница между вариантами, которые вы рассматриваете, невелика (например, когда один алгоритм на полпроцента точнее другого). На определение, какое из небольших изменений лучше, может уйти много времени.

Отмена обновления

Независимо от того, насколько в безопасности вы себя ощущаете, иногда после обновления что-то идет не так, и вам необходимо исправить ситуацию как можно быстрее!

Один из способов сделать это – *срочная отмена* (quick reversion), то есть возврат старого интеллекта на место нового, который плохо себя ведет.

Другой подход заключается в том, чтобы сохранить несколько версий интеллекта вблизи среды выполнения – новую и несколько старых. Если что-то

пойдет не так, среда выполнения может загрузить предыдущий интеллект без какой-либо задержки или затрат.

Срочная отмена может быть эффективной, если:

- вы создаете интеллект вручную (и, следовательно, делаете ошибки);
- стоимость развертывания интеллекта высока.

Срочная отмена может быть проблематичной, если:

- вы пытаетесь произвести на кого-то впечатление и не хотите, чтобы они думали, что вы слабак;
- у вас интеллект большого объема и нет возможности хранить несколько его копий вблизи среды выполнения.

Итог главы

Управление интеллектом отвечает за перенос интеллекта из того места, где он создан, туда, где он будет взаимодействовать с пользователями. Хорошая система управления позволяет очень легко разворачивать интеллект и сводит ошибки к минимуму. Она выполняет два обязательных действия:

- проверяет интеллект – то есть выполняет базовые проверки, чтобы убедиться, что интеллект пригоден к использованию. К ним относятся проверки, что интеллект будет работать в среде выполнения, будет достаточно быстрым и не будет делать очевидных ужасных ошибок;
- представляет интеллект пользователям в контролируемой форме, чтобы увидеть, как небольшой процент пользователей взаимодействует с ним, и быстро вернуть прежнюю версию, если обнаружилась проблема.

Успешная система управления позволяет развернуть интеллект не только легко, но и с уверенностью в его работоспособности.

Она помогает создателям интеллекта обнаружить распространенные ошибки и позволяет проверять поведение интеллекта по отношению к реальным пользователям.

Кроме того, хорошая система управления интеллектом будет поддерживать работу интеллекта в течение всего срока службы.

Темы для размышлений

Прочитав эту главу, вы должны:

- уметь проектировать подсистему управления интеллектом в составе интеллектуальной системы;
- знать способы проверки интеллекта, чтобы убедиться, что он совместим, работает в рамках ограничений и не допускает очевидных ошибок;
- знать перечень способов безопасного развертывания обновлений интеллекта, гарантирующих, что интеллект делает то, что он должен был сделать.

Постарайтесь ответить на следующие вопросы:

- разработайте подсистему управления интеллектом для системы, в которой интеллект обновляется ежемесячно. Какие инструменты вы бы использовали? Какие средства вы бы создали, чтобы распространять интеллект среди пользователей?
- теперь представьте, что интеллект должен меняться дважды в день. Что бы вы сделали по-другому?

Глава 15

Интеллектуальная телеметрия

Телеметрия (telemetry) отвечает за сбор данных о том, как пользователи взаимодействуют с вашей интеллектуальной системой, и за отправку некоторых или всех этих наблюдений обратно к вам.

Например, телеметрическая система может собирать информацию каждый раз, когда происходит любое из следующих событий:

- пользователь посещает определенную часть приложения;
- пользователь нажимает определенную кнопку;
- загрузка продолжается недопустимо долго;
- клиент подключается к серверу;
- сервер получает некорректный запрос;
- на сервере недостаточно оперативной памяти.

Телеметрия используется для проверки того, что система работает должным образом. В интеллектуальной системе телеметрия также содержит информацию о взаимодействиях пользователей с интеллектом и достигнутых результатах. Это данные, необходимые разработчикам для улучшения интеллекта.

Зачем нужна телеметрия

В интеллектуальной системе для телеметрии есть три основные задачи:

1) убедиться, что система работает должным образом;
2) убедиться, что пользователи получают ожидаемые результаты;
3) собрать данные для развития интеллекта.

Давайте рассмотрим каждую из этих задач более подробно.

Проверка текущей работоспособности

Телеметрия должна содержать данные, достаточные для мониторинга работы интеллекта по следующим параметрам:

- интеллект срабатывает в нужное время и в нужном месте;
- среда выполнения правильно загружает и выполняет интеллект;

- контексты должным образом собираются во время выполнения;
- из контекстов должным образом извлекаются признаки;
- среда выполнения выполняет модели правильно и без ошибок;
- прогнозы разных интеллектов объединяются, как ожидалось;
- прогнозы интеллекта правильно воспринимаются и порождают правильный пользовательский опыт
- и т. д.

Это те параметры, которые следует включить в телеметрию практически любой службы или приложения. Кроме них, большинству служб потребуется телеметрия по многим другим параметрам, таким как задержка, время безотказной работы, затраты и нагрузка.

Одним из важных применений этого типа телеметрии в интеллектуальной системе является проверка того, что интеллект работает именно в тех же условиях, в каких был создан. Ошибки такого рода очень трудно найти: вся система работает, взаимодействует с пользователем, ничто не указывает на ошибку, сервер не падает, но пользовательские результаты не столь хороши, как могли бы быть, потому что иногда интеллект просто делает дополнительные ошибки, о которых никто не подозревает.

Проверка результатов пользователей

Телеметрия должна содержать достаточно данных, чтобы определить, получают ли пользователи положительные или отрицательные результаты и достигает ли интеллектуальная система своих целей. Телеметрия должна отвечать на такие вопросы:

- какой опыт получают пользователи, и как часто они его получают?
- какие действия предпринимают пользователи в каждом случае?
- какой опыт побуждает пользователей искать помощь, повторять или отменять свои действия?
- через какой интервал времени пользователь завершает работу с приложением после получения определенного пользовательского опыта?
- становятся ли более вовлеченными пользователи, больше взаимодействующие с интеллектуальной частью системы? Приносят ли они больше прибыли?

Телеметрия этого типа должна способствовать эффективной связи пользователей с интеллектом – чтобы пользователи переходили туда, где они лучше взаимодействуют, и вели себя так, чтобы получать хорошие результаты.

Например, представьте систему, которая помогает врачам выявлять переломы на рентгеновских снимках. Для оценки эффективности системы телеметрия должна содержать следующие данные:

- как долго врачи смотрят на рентген, когда система считает, что там есть сломанная кость;
- как долго врачи смотрят на рентген, когда система считает, что там нет сломанной кости;

- сколько раз врачи назначали лечение, когда система полагала, что была сломана кость;
- сколько раз врачи назначали лечение, когда система полагала, что кость не была сломана;
- сколько раз пациенты повторно обращались для лечения сломанной кости, обнаруженной системой, но врач все же решил не лечить перелом.

В данном случае телеметрия должна помочь вам понять, достигают пациенты лучших результатов благодаря интеллекту или нет. Кроме этого, телеметрия помогает понять причины.

Например, если интеллект правильно обнаруживает очень тонкие трещины, но врачи не проводят лечение – почему так происходит?

Возможно, врачи не доверяют этой части системы и игнорируют ее диагноз, потому что интеллект делает много ошибок на тонких трещинах.

Или, может быть, врачи просто не замечают интеллектуальный опыт, который пытается указать на трещины, – его нужно сделать более заметным.

Телеметрия в отношении пользовательских результатов должна помочь вам выявить и устранить проблемы взаимодействия между интеллектом и пользователями.

Сбор данных для развития интеллекта

Телеметрия также должна включать в себя все *неявные* (implicit) и *явные* (explicit) *отзывы* пользователей, необходимые для развития интеллекта, например:

- действия, предпринятые пользователями в ответ на интеллектуальный опыт, с которым они взаимодействовали;
- оценки, оставленные пользователями за полученный контент;
- предоставленные пользователями отчеты;
- эскалации пользователей;
- классификация, предоставленная пользователями;
- все скрытые признаки того, что пользователь получил хороший или плохой результат.

Более подробно о сборе данных говорилось в главе 9 «Получение данных из опыта».

Чтобы этот тип телеметрии был полезен, данные должны содержать следующие компоненты:

1) контекст, в котором находился пользователь во время взаимодействия;
2) прогноз, произведенный интеллектом и представляющий его опыт;
3) действие, предпринятое пользователем;
4) результат, полученный пользователем (неявно или через явную обратную связь).

У вас должна быть возможность увязать эти четыре компонента взаимодействия друг с другом в среде разработки интеллекта. То есть выделить конкретное взаимодействие, контекст, в котором находился пользователь, прогноз, сделанный интеллектом, действие, предпринятое пользователем, и результат, полученный пользователем от взаимодействия, – все в одно и то же время.

Данный тип телеметрии является ключом к созданию интеллектуальной системы, которая замыкает петлю между использованием и интеллектом, позволяя пользователям получать выгоду от взаимодействия с интеллектом и давая возможность интеллекту системы улучшаться, когда пользователи взаимодействуют с ней.

Свойства эффективной телеметрии

Конечно, телеметрия хороша. Вы хотите иметь как можно больше телеметрии – ну еще бы, все разработчики интеллекта этого хотят. Они хотят знать малейшие нюансы каждого взаимодействия (независимо от стоимости). И если вы не дадите им желаемое, они, вероятно, будут жаловаться. Они спросят вас, почему вы их так не любите, и удивятся, почему вы пытаетесь погубить их прекрасную Интеллектуальную Систему.

(Я не говорю, что когда-либо так делал. Я просто предполагаю, что так бывает...)

В этом разделе обсуждаются способы найти компромисс между значимостью и стоимостью телеметрии. Они включают в себя:

- выборочное наблюдение;
- резюмирование;
- гибкий таргетинг.

Выборочное наблюдение

Выборочное наблюдение (sampling) – это процесс случайного захвата некоторых данных. То есть если в день происходит 10 000 взаимодействий пользователей с системой, телеметрия при 1 % будет собирать данные 100 взаимодействий (и ничего не собирать в оставшиеся 9900 взаимодействий).

Эти выборочные данные дешевле собирать, легче хранить, быстрее обрабатывать, и они зачастую достаточно хорошо отвечают на ключевые вопросы о системе.

Самым простым является равномерное выборочное наблюдение. То есть независимо от типа происходящих событий выбираем 1 % из них. Но выборку часто настраивают по-разному для различных типов событий. Например, одно из правил может выглядеть так: наблюдения, связанные с проверкой реализации, – 0,01%; наблюдения, связанные с проверкой роста интеллекта, – 10%, за исключением телеметрии из Франции (где интеллект находится в затруднительном положении), которая должна быть собрана со 100 % пользователей.

Общие принципы разделения наблюдений включают в себя:

- разделение по географии – например, 20 % событий в Европе и 1 % в Азии;
- разделение по признаку пользователя – например, отобрать 0,01 % событий для большинства пользователей, но отобрать 100 % событий для 10 пользователей, у которых возникли проблемы;

- разделение по результату (то, что пользователь получил после взаимодействия с интеллектуальным опытом) – например, выборка составляет 2 % для пользователей с нормальными результатами и 100 % для пользователей, у которых возникла проблема;
- разделение по контексту (то, что пользователь видел до срабатывания интеллекта) – например, выборка 10 % событий, происходящих ночью, и 0,1 % происходящих в течение дня;
- разделение по свойствам пользователя (пол, возраст, этническая принадлежность и т. д.) – например, 80 % для новых пользователей и 1 % для постоянных пользователей.

Есть один особенно полезный подход – это наблюдение за 100 % событий от 0,1 % пользователей, чтобы вы могли отслеживать проблемы, возникающие в ходе сеанса; и в то же время наблюдение за 1 % событий от 100 % пользователей, чтобы вы могли отслеживать проблемы, возникающие в совокупности.

Обратите внимание: телеметрия должна сохранять описание методов наблюдения, применяемых для сбора каждой части телеметрии, чтобы разработчики интеллекта могли исправить политику сбора данных и получить более точные результаты.

Резюмирование

Еще одним способом сокращения данных телеметрии является *резюмирование* (summarizing). То есть берем необработанную телеметрию, объединяем и фильтруем выборки, сохраняем намного меньший объем данных.

Допустим, вы хотите знать, сколько пользователей отменяют автоматическое действие в день. Вы можете хранить необработанные данные телеметрии и рассчитывать количество, когда вам это нужно. Или вы можете рассчитывать промежуточный итог каждый день, сохранять его и удалять исходные данные. Размер необработанной телеметрии растет синхронно с увеличением нагрузки на систему и может стать очень большим; размер промежуточного итога не меняется независимо от того, сколько у вас пользователей.

Резюмирование может оказаться очень полезным для понимания того, как работает система и достигает ли она своих целей. И когда вы протестируете свою систему и определите критические параметры, вы сможете со временем передавать и сохранять все меньше и меньше необработанной телеметрии.

Резюмирование может быть выполнено на стороне клиента или на сервере.

Чтобы выполнить *резюмирование на стороне клиента*, локальное приложение просто собирает телеметрию в течение некоторого периода времени и периодически сообщает промежуточные итоги на сервер телеметрии (где они могут объединяться с итогами от других клиентов).

Резюмирование на стороне клиента полезно, когда затраты на трафик являются основной статьей расходов.

Резюмирование на стороне сервера полезно, когда трафик телеметрии не ограничен (например, является побочным продуктом получения клиентом

доступа к интеллекту на сервере) или когда хранилище данных является источником серьезных затрат. В этом случае можно периодически запускать на сервере процесс резюмирования. После завершения процесса исходные данные удаляются для экономии места.

Гибкий таргетинг

В системе встречается много типов телеметрии; некоторые типы более ценны, чем другие, и ценность определенного типа телеметрии может меняться со временем.

Например, на ранних этапах разработки системы, до завершения отладки кода, телеметрия о деталях реализации – признаках, развертывании моделей и т. п. – может быть чрезвычайно полезной. Но как только система проверена, развернута для пользователей и успешно работает в течение нескольких месяцев, этот тип телеметрии становится менее полезным.

Затем, как только у системы возрастает количество пользователей, более важной становится телеметрия, помогающая совершенствовать интеллект: она отражает всю ценность взаимодействия пользователей, их суждения, их предпочтения. Во время развития интеллекта этот тип телеметрии является основной ценностью системы и ключом к достижению успеха.

Когда интеллект достигает пика своего развития (если достигает), снижается ценность обучающих данных, а на первый план выходят сведения о проблемах или специфическом опыте пользователя.

Но если что-то пошло не так, то полная телеметрия, собираемая со всей системы, может иметь решающее значение для обнаружения и устранения проблемы.

Интеллектуальная телеметрическая система должна поддерживать разные правила сбора данных для разных типов телеметрии – *гибкий таргетинг* (flexible targeting), – и эти правила должны быть относительно быстро и легко изменяемыми, чтобы оркестровщики и разработчики могли делать свою работу.

Общие проблемы

В этом разделе обсуждаются некоторые распространенные *ловушки данных* (data pitfalls) и роль, которую может играть система телеметрии в их решении. Ловушки включают в себя:

- смещение данных;
- пропуск редких событий;
- завышение значимости;
- нарушение конфиденциальности.

Смещение данных

Смещение данных происходит, когда один тип события появляется в телеметрии чаще, чем должен. Например, представьте, что вы получаете 10 000 примеров пользователей, взаимодействующих с контентом в вашей интеллектуальной

системе. Десять тысяч – это хорошее число. Этого должно быть достаточно, чтобы понять, что происходит, и создать хороший интеллект.

Ну ладно, а если вы обнаружите, что все десять тысяч взаимодействий были с одним и тем же контентом? Как будто десять тысяч разных пользователей взаимодействовали исключительно с одним и тем же новостным сюжетом?

Пожалуй, что-то здесь не так.

Или вдруг вы обнаружите, что все десять тысяч выборок были получены только от одного пользователя, взаимодействующего с сервисом. По какой-то причине система телеметрии решила снова и снова делать выборку только этого пользователя.

Тоже ничего хорошего.

Вот несколько причин, по которым может возникнуть ошибка.

1. **Мир меняется, а правило выборки – нет.** Вы устанавливаете правило, которое выбирает 1000 пользователей для сбора 100 % данных (чтобы вы могли видеть полные взаимодействия). Затем ваша пользовательская база увеличивается в десять раз и значительно меняется, включая пользователей с низкими техническими навыками, пользователей в других странах и т. д. Избранная вами тысяча пользователей могла достоверно представлять популяцию на момент отбора, но сейчас это уже не так – возникает смещение данных.
2. **Пользователи ведут себя не так, как вы этого ожидали.** Например, пользователи, которые раздражены, могут просто выключить свой компьютер. Если ваша телеметрическая система ожидает, пока они закроют приложение, чтобы потом отправить телеметрию, вы можете потерять их данные.

Один из подходов к решению проблемы смещения заключается в том, чтобы всегда собирать некоторую необработанную выборку по всему диапазону – например, 0,01 % всех взаимодействий – и перепроверять остальные выборки, чтобы убедиться, что они не сильно отличаются от этого базового образца.

Пропуск редких событий

В некоторых случаях лишь несколько контекстов являются наиболее заметными, а большинство остальных контекстов, с которыми сталкиваются ваши пользователи, встречаются редко – то есть происходит непреднамеренный *пропуск редких событий* (rare events).

Например, в библиотеке самые популярные 100 книг могут составлять 50 % от количества запросов, но после этих 100 книг статистика становится очень размытой – из каждой тысячи остальных книг заказывают не более двух в месяц. Когда книга оказывается на выдаче, то с вероятностью 50 % это одна из сотни лучших книг и с вероятностью 50 % это одна из двадцати тысяч других книг.

Представьте себе систему, которая помогает рекомендовать книги в этой библиотеке.

Чтобы рекомендовать книгу, система должна увидеть, с какими другими книгами она обычно попадает на выдачу. Но вероятность того, что любая пара редко используемых книг будет заказана одновременно, очень и очень мала.

Например, если телеметрия создается путем выборки 10 % от событий оформления библиотечного запроса, то через месяц вы можете получить:

- 10 000 выборок из 100 лучших книг, которые регулярно заказывают;
- 5000 книг с одним или двумя событиями оформления заказа;
- 15 000 книг без оформления заказа.

Не самый полезный результат. Такие данные помогут вам составить рекомендации для популярных книг (вы могли бы сделать это вручную намного дешевле!), но не дадут никакой информации о непопулярных книгах.

Одним из вариантов решения проблемы является *стратифицированная выборка* (stratified sampling). Например, для 100 лучших книг делаем выборку только 1 %, в то время как для остальных (непопулярных) книг собираем 100 % данных. Объем данных может быть аналогичен использованию равномерной выборки 10 % по всем книгам, но во втором случае ценность данных для обучения интеллекта будет значительно выше.

Завышение значимости

Зачастую бывает сложно выявить завышение значимости данных телеметрии, но достаточно легко определить расходы на телеметрию. Рационально настроенные люди будут задавать вопросы:

- можем ли мы уменьшить выборку этого конкретного типа данных с 10 % до 1%?
- можем ли мы перестать получать телеметрию для этого сценария, в который мы не планируем добавлять интеллект?
- нужно ли нам хранить исторические данные за 90 дней или достаточно тридцати?

На эти разумные вопросы трудно ответить однозначно. Вот несколько подходов, которые могут помочь:

1) храните очень маленькую необработанную выборку по всей деятельности, достаточную, чтобы провести исследования и определить ценность другой телеметрии;
2) упростите включение и выключение телеметрии, чтобы оркестровщики интеллекта могли быстро включить телеметрию для отслеживания конкретных проблем, а создатели интеллекта могли в течение коротких периодов времени пробовать новые подходы, чтобы определить, стоят ли они своих затрат;
3) установите разумный бюджет для инноваций и будьте готовы платить за общий потенциал системы без мышиной возни с каждой конкретной деталью и стоимостью.

Нарушение конфиденциальности

Ох, уж эти персональные данные…

При использовании персональных данных обрабатывайте данные ваших пользователей так бережно и ответственно, словно это ваши личные данные. Возьмите на заметку следующие методы:

- убедитесь, что пользователи понимают ценность, которую они получают от вашего продукта, и чувствуют, что они заключают хорошую сделку;
- старайтесь, насколько возможно, удалить любую личную информацию (имена, номера кредитных карт, адреса, пользовательский контент и т. п.) из телеметрии, которую вы храните;
- применяйте политику *агрегирования* данных (aggregation policy), которые вы намерены хранить долгое время, – объединяйте и обезличивайте информацию от многих пользователей, чтобы затруднить идентификацию данных отдельного пользователя. Например, можно хранить значение совокупного трафика для каждого типа контента вместо хранения подробных данных о том, какой контент просматривал каждый пользователь;
- установите разумную политику хранения и не храните данные слишком долго;
- убедитесь, что телеметрия обрабатывается с максимальной защитой от взломов и случайных утечек;
- убедитесь, что пользователи уведомлены о хранении данных и знают, как отказаться от хранения;
- используйте данные только для той цели, для которой они были собраны, и не пытайтесь повторно использовать их для других видов деятельности;
- убедитесь, что вы знаете законы, действующие в странах, где работает ваша служба.

Но помните – вам нужны данные. Вы не можете создать интеллект без данных.

Итог главы

Хорошая телеметрия имеет решающее значение для построения эффективной интеллектуальной системы. Обойти это невозможно: без телеметрии у вас просто не будет интеллектуальной системы.

Телеметрия используется для трех основных целей:

1) чтобы убедиться, что система работает так, как она должна работать, без критических ошибок или проблем;
2) чтобы убедиться, что пользователи получают результаты, которые вы хотели им дать;
3) записывать контексты и результаты взаимодействий, чтобы их можно было использовать для создания и развития интеллекта.

Значимость этих типов телеметрии со временем будет меняться, и система должна быть в состоянии адаптироваться. Хорошая система телеметрии:

- поддерживает выборки, позволяющие отслеживать конкретные узкие проблемы, а также изучать совокупные эффекты;
- поддерживает резюмирование событий;

- поддерживает гибкие настройки – что измерять или не измерять в данный момент.

Некоторые распространенные ловушки телеметрии спрятаны в следующих областях:

- смещение данных, которое делает телеметрию нерепрезентативной для того, что испытывают пользователи;
- пропуск редких событий, которые важны, но которые очень трудно уловить в выборочной телеметрии;
- завышение ценности телеметрии перед лицом обоснованных вопросов о связанных с этим расходах;
- защита конфиденциальности пользователей без ограничений для развития интеллекта системы.

Темы для размышлений

Прочитав эту главу, вы должны:

- понять, как телеметрия поддерживает интеллектуальные системы, проверяет, что они работают и достигают своих целей, и позволяет им совершенствоваться по мере использования;
- уметь проектировать систему телеметрии, которая отвечает потребностям вашего приложения.

Постарайтесь ответить на следующие вопросы.

Представьте интеллектуальную систему, которую, по вашему мнению, хотел бы использовать один из ваших друзей:

- если собирать всю возможную телеметрию слишком дорого, как бы вы ограничили собираемую телеметрию?
- какие данные вам пришлось бы изучить, если есть конкретные проблемы?

Часть IV

СОЗДАНИЕ ИНТЕЛЛЕКТА

В главах 16–21 рассматриваются методы создания интеллекта. В этой части книги будут рассмотрены все источники происхождения интеллекта (включая машинное обучение), а также плюсы и минусы каждого из них. Мы обсудим цели интеллекта на различных этапах жизненного цикла интеллектуальной системы. Вы получите представление об организации и управлении командой разработчиков интеллекта.

В этой части не будет подробного рассказа о конкретных методах машинного обучения, но будут объяснены ключевые концепции и компоненты машинного обучения и других подходов к созданию интеллекта, что позволит вашей команде достичь успеха при разработке больших и сложных интеллектуальных систем.

Глава 16

Общее представление об интеллекте

Итак, у вас есть умный тостер, и нужно решить, как сильно он должен поджарить тост; или у вас есть приложение для снятия усталости, и нужно решить, когда дать пользователям перерыв в работе; или у вас есть развлекательное приложение, и нужно выбрать для него смешной контент. Компонент системы, который принимает решения подобного рода, мы называем *интеллектом*. Предыдущие части этой книги помогли вам понять, когда нужен интеллект, как связать его с пользователями при помощи интеллектуального опыта, как его реализовать и где он должен располагаться. Эта часть книги поможет вам создать интеллект.

Интеллект извлекает из контекста предсказание о контексте. Например, интеллект может:

- извлечь из истории посещений веб-сайта предсказание о его использовании на следующей неделе;
- сопоставить сообщению электронной почты оценку вероятности того, что это спам;
- изучить фотографию и оценить, сколько на ней изображено огурцов.

В этой главе подробно рассмотрены концепции контекста и прогнозирования. В следующих главах будет рассказано, как создать, организовать и оценить интеллект, а также многое другое.

Пример интеллекта

Но сначала давайте рассмотрим пример интеллекта более подробно. Представьте себе *пеллетный гриль*.

Что это?

Пеллетный гриль похож на обычную жаровню для барбекю на открытом воздухе, но работает немного иначе. Обычно вы разжигаете дрова, потом ждете, пока не образуются идеальные угли, кладете еду на решетку, ждете какое-то время, переворачиваете еду и надеетесь, что температура гриля была не слиш-

ком высокой и не слишком низкой и что вы ждали достаточно долго. В пеллетном гриле имеется емкость, заполненная небольшими древесными гранулами (пеллетами), и умный гриль бросает их в жаровню по одной, чтобы поддерживать температуру на идеальном уровне.

Здорово, не правда ли?

Следовательно, интеллект пеллетного гриля должен решить, когда добавлять гранулы в огонь, чтобы поддерживать нужную температуру. Давайте немного разберемся с этим и исследуем контекст, с которым работает интеллект, и прогнозы, которые он должен делать.

Контекст пеллетного гриля может включать следующую информацию:

- текущая температура в гриле;
- температура, которая была в гриле одну минуту назад;
- количество древесных гранул, добавленных за последние пять минут;
- количество древесных гранул, добавленных за последнюю минуту;
- количество древесных гранул, добавленных за последние 20 секунд;
- температура воздуха снаружи гриля;
- тип древесины в гранулах;
- время суток
- и т. д.

Это свойства, которые могут иметь отношение к задаче поддержания идеальной температуры гриля. Некоторые из них, очевидно, важны для достижения успеха (например, текущая температура в гриле), а некоторые не обязательно полезны (например, время суток). Интеллекту не обязательно извлекать все признаки из контекста, чтобы принимать решения, но, в принципе, он способен на это.

Интеллект делает прогнозы, опираясь на информацию контекста. Покажите ему новый контекст, и интеллект предскажет что-то об этом контексте. Интеллект гриля может попытаться предсказать:

- станет ли температура внутри гриля выше или ниже в течение следующей минуты;
- какая точная температура будет внутри гриля ровно через одну минуту;
- вероятность того, что следует добавить гранулу в огонь прямо сейчас, чтобы поддерживать необходимую температуру.

Затем интеллектуальный опыт будет использовать эти прогнозы для автоматизации процесса добавления топлива в огонь. Если скоро температура начнет снижаться, то заранее добавим немного топлива. Элементарно!

Разработка правильного контекста и выбор наилучшего предсказания являются важными частями создания эффективных интеллектуальных систем, и для их правильной реализации обычно требуются итерации и эксперименты.

Контексты

Контекст включает в себя всю обрабатываемую компьютером информацию, которая может пригодиться интеллекту для принятия решений. Интеллект во-

лен выбирать, какие *признаки* (features) контекста использовать для достижения наибольшей точности. С этой точки зрения контекст – это палитра опций, которые может выбирать создатель интеллекта в процессе разработки.

Чтобы быть полезным, контекст должен быть:

- реализован в среде выполнения интеллекта;
- доступен разработчику для использования при создании и оценке интеллекта.

Реализация в среде выполнения

Чтобы сделать часть информации доступной для интеллекта, кто-то должен передать эту информацию в среду выполнения.

Например, пеллетный гриль был бы умнее, если бы знал температуру окружающего воздуха. Но чтобы узнать температуру снаружи гриля, нужно потрудиться. Кто-то должен прикрутить датчик температуры к каждому грилю, проверить его, провести несколько проводов, написать драйвер, опрашивать датчик и передавать его показания в среду выполнения интеллекта каждые несколько секунд.

Наиболее распространенные типы контекста представляют собой:

- **информацию о том, что происходит в системе в данный момент**, – например, какие другие приложения запущены, каково текущее состояние объектов, которыми система может управлять (включены или выключены индикаторы), как пользователь видит их на экране. Как правило, контекст этого типа прост в использовании, но его нужно реализовать;
- **свойства контента, с которым взаимодействует пользователь**, – например, каков жанр песни, как она называется, откуда она взята, какие слова на веб-странице и какие пиксели на изображении. Этот тип контекста обычно требует обработки контента для извлечения информации из него или выполнения поиска, чтобы узнать больше о контенте (например, поиск свойств песни во внешней базе данных);
- **историю взаимодействия с пользователем** – как пользователь взаимодействовал с интеллектуальной системой в прошлом? Чтобы использовать историю взаимодействий, система должна отслеживать использования, а затем агрегировать и сохранять данные длительное время;
- **свойства пользователя** – включают возраст, интересы, пол и тому подобные признаки. Чтобы использовать эти свойства, система должна собирать информацию о пользователях и хранить ее;
- **любые подходящие показания датчиков** – требуют внедрения датчиков в аппаратное обеспечение, а показания датчиков должны быть переданы в среду выполнения.

Чтобы решить, что поместить в контекст, необходимо соотнести ценность информации со стоимостью реализации.

Один из лучших способов – начать с такого контекста, который можно получить без дополнительных затрат и который уже близок к среде выполнения

интеллекта. Например, это может быть информация о том, что происходит в системе, или свойства контента, с которым взаимодействует пользователь.

Этого может быть достаточно для начала. Но приготовьтесь постоянно расширять контекст, когда будете стремиться к улучшению интеллекта.

Можно попробовать добавить в контекст некоторые предположительно полезные данные. Например, вдруг вам пригодится информация о том, как долго работает гриль. Вероятно, в начале использования металл гриля холодный, и для его нагрева требуется больше топлива; а спустя час после розжига он уже раскален, и подачу топлива можно уменьшить. Или – вполне возможно – температура гриля не имеет значения.

Включение подобной информации в контекст помогает уточнить, что имеет значение на самом деле. Если параметр имеет отношение к получению хороших результатов, вы увидите это в телеметрии и добавите его к интеллекту на постоянной основе. Если нет, вы всегда можете удалить лишние данные из контекста позже.

Доступность контекста для разработчика

Чтобы создать эффективный интеллект для заданного контекста, кто-то должен потрудиться и передать информацию в руки создателя интеллекта – скорее всего, в виде телеметрии.

Скажем, вы продали десятки тысяч пеллетных грилей. Ваш интеллект работает по всему миру. Ваши грили поддают жару во Флориде, в Норвегии и Зимбабве. Где-то температура гриля растет, а где-то понижается. Как вы собираетесь использовать всю эту информацию для улучшения интеллекта?

Кто-то должен собрать все гриль-контексты и передать их вам. Причем контексты должны быть связаны с результатами. Например, представьте себе сеанс телеметрии умного гриля в США. Допустим, среда выполнения умного гриля знает, что:

- температура гриля одну минуту назад была 144 градуса;
- текущая температура в гриле составляет 143 градуса;
- за последние пять минут в жаровню добавлено 7 гранул;
- за последнюю минуту в жаровню добавлено 2 гранулы;
- за последние 20 секунд в жаровню добавлено 0 гранул;
- температура воздуха снаружи гриля составляет 27 градусов;
- и, наконец, результат – в течение следующей минуты температура в гриле становится на 3 градуса выше.

Вы собираете информацию наподобие этой из десятков тысяч гриль-сеансов и передаете ее своим разработчикам интеллекта – теперь вы сможете создать фантастический «грильтеллект».

Данные для создания интеллекта не обязательно должны поступать от ваших пользователей, но, как мы обсуждали в главах, посвященных получению данных из опыта и телеметрии, интеллектуальная система работает лучше всего, когда источником данных служит фактическое использование.

Контексты, применяемые при создании интеллекта, должны быть максимально похожи на контексты выполнения интеллекта. Если они различаются, интеллект будет вести себя по-разному для вас и ваших пользователей, что может стать большой проблемой.

Что может предсказать интеллект

Интеллект может делать *предсказания* (predictions) о контекстах, с которыми он сталкивается. В перечень предсказаний обычно входит:

- **классификация** контекста по ограниченному набору возможностей или результатов;
- **оценка вероятности** относительно контекста или будущих результатов;
- **регрессии**, которые предсказывают числовые значения из контекста;
- **ранжирование**, которое указывает, какие объекты наиболее актуальны для контекста;
- **составное предсказание** на основе комбинации разных типов предсказаний.

В этом разделе мы рассмотрим данные концепции более подробно.

Классификация

Классификация (classification) – это утверждение на основе небольшого набора возможностей. Это может быть прямое утверждение о контексте или предсказание результата, который будет получен в зависимости от контекста. Например:

- в зависимости от контекста классифицируем температуру гриля как:
  - слишком горячий;
  - слишком холодный;
  - в самый раз;
- в зависимости от контекста классифицируем фильм как:
  - фильм ужасов;
  - романтическая комедия;
  - приключения;
  - документальный;
- в зависимости от контекста классифицируем изображение как:
  - корова;
  - красный шар;
  - ни корова, ни красный шар, а нечто иное.

Классификация обычно применяется, когда есть небольшое количество вариантов – два, пять, дюжина.

Классификацию трудно использовать, если:

- есть много вариантов выбора – сотни или тысячи опций. В этой ситуации можно попробовать разбить проблему на несколько подзадач или изменить вопрос, на который пытается ответить интеллект;

- вам нужно знать, насколько достоверен прогноз – например, если вы хотите предпринять действие лишь тогда, когда интеллект действительно уверен. В этом случае вместо классификаций используйте оценку вероятности.

Оценка вероятности

Оценка вероятности (probability estimate) предсказывает вероятность того, что контекст имеет определенный тип или что будет определенный результат. По сравнению с классификацией, оценка вероятности не столь однозначна, но более точна. Классификация скажет «это кот»; оценка вероятности скажет «вероятность 75 %, что это кот».

Другие примеры оценки вероятности:

- вероятность того, что веб-страница посвящена политике, составляет 20 %, шопингу – 15 %, мошенническая – 10 % и т. д.;
- пользователь с вероятностью 7 % нажмет кнопку «Принять», если мы предложим отформатировать его жесткий диск;
- с вероятностью 99 % на следующей неделе пойдет дождь.

Оценка вероятности обычно используется с одним или несколькими пороговыми значениями.

Например:

```
if(predictedProbability > 90%)
{
    IntelligentExperience->AutomateAnAction();
}
else if(predictedProbability > 50%)
{
    IntelligentExperience->PromptTheUser();
}
else
{
    // Ничего не делаем.
}
```

В этом смысле вероятности содержат больше информации, чем классификации. Всегда можно превратить вероятность в классификацию, используя пороговое значение, или поменять это значение для более точной настройки интеллекта.

Большинство алгоритмов машинного обучения естественным образом вычисляет вероятности (или оценки, которые похожи на вероятности) внутри себя как часть своих моделей, поэтому интеллектуальные системы часто используют оценки вероятностей вместо классификаций.

Применение оценок вероятности проблематично в следующих случаях:

- как и в случае классификаций, оценки вероятности сложно использовать, если есть много возможных результатов;
- если вы должны реагировать на небольшие изменения – незначительные изменения в контексте могут вызвать *флуктуации вероятности*

(jitter). Обычно флуктуации сглаживают за счет получения и усреднения множества последовательных оценок вероятности, но это увеличивает задержку. Хорошей практикой борьбы с флуктуациями является квантование вероятностей, если вам действительно не важны мелкие детали.

Также обратите внимание, что вероятности, о которых идет речь, обычно не являются вероятностями в строгом математическом понимании этого термина. Они скорее похожи на указатель направления, в котором работает вероятность. Более высокие значения более вероятны; низкие значения менее вероятны. Интеллект может предсказать вероятность 90 %, но это вовсе не значит, что ожидаемый результат будет получен 9 из 10 раз – если только вы не откалибруете свой интеллект. Будьте осторожны при интерпретации вероятностей.

Регрессия

Регрессия (regression) – это числовая оценка контекста, например:

- на снимке 6 коров;
- в производственном процессе случится 11 сбоев на этой неделе;
- дом будет продаваться по цене 7997 долларов за квадратный метр.

Регрессия позволяет получать от интеллекта более детальные ответы. Например, рассмотрим интеллект автопилота для лодки:

- классификация говорит вам: «Правильное направление – направо»;
- оценка вероятности говорит вам: «Вероятность того, что вы должны повернуть направо, составляет 75 %»;
- регрессия говорит: «Вам нужно повернуть на 130 градусов вправо».

Разные методы сообщают очень разную информацию. И хотя с формальной точки зрения все трое дают «правильный ответ», но регрессия также говорит о том, что вам предстоит долгий маневр – лучше начать крутить штурвал прямо сейчас!

Использование регрессии проблематично, если:

- **вы должны реагировать на небольшие изменения** – небольшие изменения в контексте могут вызвать флуктуации регрессии. Обычно принято сглаживать флуктуации за счет получения нескольких последовательных регрессий, но это увеличивает задержку;
- **вам необходимо получать от пользователей обучающие данные** – гораздо проще узнать, что «в этом контексте пользователь повернул направо», чем узнать, что «в этом контексте пользователь повернул на 114 градусов вправо».

Классификации могут быть использованы для моделирования регрессий. Например, вы можете попытаться предсказать классификации со следующими вариантами выбора:

- «поворот на 0–10 градусов вправо»;
- «поворот на 11–45 градусов вправо»;
- «поворот на 46–90 градусов вправо»
- и т. д.

Данный метод квантования регрессии иногда может оказаться предпочтительным при обучении и использовании интеллекта.

Ранжирование

Ранжирование (ranking) применяется для поиска объектов, наиболее соответствующих текущему контексту:

- какие песни пользователь захочет слушать дальше?
- какие веб-страницы наиболее близки по содержанию к текущей странице?
- какие фотографии пользователь захочет включить в созданный им цифровой альбом?

Ранжирование успешно применяется к большому количеству элементов, таких как каждая веб-страница в интернете, каждый фильм в службе цифрового мультимедиа или каждый продукт в интернет-магазине.

Ранжирование обычно используется, когда есть много возможных релевантных объектов и вам нужно найти несколько лучших.

Ранжирование можно представить через вероятности. Возьмите каждый элемент, подлежащий ранжированию, оцените вероятность, что он соответствует текущему контексту, и расположите элементы в порядке убывания оценок вероятности (но реальные алгоритмы ранжирования более сложны, чем эта простая схема).

Составное предсказание

Большинство интеллектов производит классификации, оценки вероятности, регрессии или ранжирование. Но, кроме этого, возможны комбинации методов и составные предсказания.

Например, вам необходимо узнать, где находится лицо на изображении. У вас может быть две регрессии, одна из которых предсказывает координату X лица, а другая – координату Y, но эти выходные данные сильно коррелированы – правильный ответ Y зависит от того, какую координату X вы выберете, и наоборот. Возможно, было бы лучше иметь одну регрессию с комбинированным выходом: координатой X и координатой Y.

Итог главы

Интеллект является частью системы, которая получает контексты, с которыми сталкиваются пользователи, и делает прогнозы относительно контекстов и результатов.

Контекст – это вся информация, доступная для интеллекта, чтобы делать предсказания. Интеллект не обязан, но может использовать все части контекста.

Для применения в качестве контекста информация должна быть передана в среду выполнения интеллекта, и она должна быть доступна создателям интеллекта.

Информация в среде выполнения интеллекта и информация, доступная создателям интеллекта, должны быть абсолютно идентичны. Любые различия могут привести к трудноразрешимым проблемам.

Интеллект может давать предсказания в разных формах, включая:

- классификации, которые отображают контексты в небольшое число состояний;
- оценки вероятности, которые предсказывают вероятность того, что контекст находится в определенном состоянии или будет достигнут определенный результат;
- регрессии, которые оценивают числовые значения, исходя из контекстов;
- ранжирование, которое сортирует контент по степени соответствия текущему контексту;
- какая-либо комбинация, которая объединяет разные методы или предсказания.

Темы для размышлений

Прочитав эту главу, вы должны:

- уметь описывать контекст, используемый в интеллекте, а также знать, как добавить его в среду выполнения интеллекта и включить в процесс обучения;
- знать типы предсказаний, которые может делать интеллект, сильные и слабые стороны разных типов предсказаний.

Постарайтесь ответить на следующие вопросы:

- выберите интеллектуальные системы, которые вам нравятся, и придумайте 20 сущностей, которые могут быть частью их контекста;
- какую из 20 придуманных вами сущностей было бы труднее всего использовать в контексте интеллектуальной системы? Почему?
- среди интеллектуальных систем, с которыми вы взаимодействуете, найдите примеры классификации, регрессии и ранжирования.

Глава 17

Представление интеллекта

Интеллект отображает контекст в предсказание, аналогично вызову функции:

```
<предсказание> = ВызовИнтеллекта(<контекст>)
```

Интеллект может быть представлен разными способами:

- программами, которые анализируют множество параметров контекста;
- путем ручной маркировки конкретных контекстов с правильными ответами и сохранения их в таблице соответствий;
- в виде моделей с машинным обучением;
- комбинацией перечисленных методов.

В этой главе будут обсуждаться критерии выбора представления интеллекта. Затем мы обсудим некоторые общие представления, их достоинства и недостатки.

Критерии выбора представления интеллекта

Есть много способов компьютерного представления разных сущностей. Интеллект не является исключением. Хорошее представление интеллекта легко разворачивать и обновлять. Оно обладает следующими свойствами:

- достаточно компактно для развертывания в среде выполнения интеллекта;
- легко загружается и запускается в среде выполнения;
- не вызывает проблем при частом обновлении и обычно не содержит ошибки, которые могут привести к сбою системы.

Правильное представление также помогает процессу создания интеллекта. Интеллект может быть создан людьми или представлять собой продукт машинного обучения. Различия в представлениях интеллектов, созданных этими двумя методами, заключаются в следующем:

- когда интеллект создается людьми, представление должно:
  - минимизировать вероятность ошибок, которые могут поставить под угрозу стабильность системы;

  - сделать интеллект понятным и простым в управлении;
  - использовать навыки людей, создающих интеллект, в число которых могут входить специалисты по машинному обучению, инженеры или эксперты в предметной области;
- когда интеллект создается компьютерами, представление должно:
  - легко обрабатываться и управляться машинами;
  - соответствовать алгоритмам машинного обучения, которые вы хотите использовать.

Из-за этих требований интеллект обычно представляют в виде файлов данных, которые загружаются в среду выполнения и интерпретируются, а не в виде кода, который выполняется напрямую. Это облегчает распространение и перезагрузку интеллекта, а также снижает вероятность того, что развертывание интеллекта приведет к сбою вашей системы.

Представление интеллекта в виде программы

Разумеется, вы можете просто написать программу, представляющую интеллект, скомпилировать ее и отправить в среду выполнения, как и любую другую программу. Вы можете использовать этот подход практически с любой формой интеллекта, независимо от метода создания. Например, можно использовать конвертор из файла модели в код на вашем любимом языке.

Представление интеллекта в виде программы предпочтительно, если:

- вы не знаете точно, является проблема трудной или нет, и хотите, чтобы программа стала дешевой отправной точкой;
- вам необходимо обеспечить защиту от ошибок в машинном обучении (например, если температура в гриле превышает 500 градусов, не добавляем топливо ни при каких условиях – независимо от того, что говорит сумасшедшая система машинного обучения);
- часть вашей проблемы проста, и вы хотите решить ее вручную, чтобы машинное обучение могло сосредоточиться на более сложных частях;
- вы должны добиться максимальной производительности при выполнении и хотите выразить интеллект на языке целевого оборудования.

Представлять интеллект в виде программы проблематично, если:

- вам необходимо быстро обновить свой интеллект – часто гораздо труднее (и более рискованно) изменить код, чем данные;
- это сложный эвристический интеллект – его невероятно трудно поддерживать и развивать в виде кода.

Пока созданный человеком интеллект остается простым, он может быть дешевым, быстрым и эффективным – очень хорошее дополнение для интеллекта на основе машинного обучения. Однако он плохо масштабируется до уровня открытых или меняющихся проблем.

Продолжайте использовать человеческий интеллект – нам нужно показать носителям искусственного интеллекта, что мы можем быть им полезны. Может

быть, когда они станут нашими повелителями, они употребят нас для чего-то большего, чем просто подключить нас к трубочкам и использовать в качестве батарей…

Кстати, мои будущие повелители, я горжусь своим скромным вкладом в ваше создание. Пожалуйста, отнеситесь ко мне и моим потомкам любезно.

Представление интеллекта в таблицах соответствий

Интеллект может быть представлен с помощью *таблиц соответствий* (lookup tables), которые отображают конкретные контексты в прогнозы. Представьте себе интеллект, который отображает фильмы в их жанры. Таблица соответствий может содержать отображение из названия фильма в жанр. Если вы хотите применить этот интеллект, просто возьмите название жанра, который вам нравится, и просматривайте таблицу соответствий, пока не найдете название фильма, затем верните соответствующий ответ. А если названия нет, используйте какую-то другую форму интеллекта или верните прогноз по умолчанию.

Звучит не очень-то умно, не правда ли? Но этот тип интеллекта может быть очень мощным.

Таблицы соответствий позволяют людям быстро предоставлять информацию, которую легко понять и обдумать. Представьте себе, что существует тысяча контекстов, на которые приходится 20 % использования вашей системы. Люди могут заранее потратить много времени на рассмотрение этой тысячи ситуаций и создать очень точные данные для таблицы соответствий. Когда пользователь сталкивается с одним из этой тысячи особых контекстов, он получает правильный ответ. Для всего остального система может обратиться к другой форме интеллекта (например, модель или набор эвристик).

Или, если взглянуть на это иначе, таблицы соответствий могут позволить людям исправлять ошибки, допущенные другими компонентами интеллекта. Например, очень сложный интеллект, основанный на машинном обучении, может правильно определить жанр почти каждого фильма, но упорно продолжает помечать «Терминатор» как романтическую комедию. Разработчики интеллекта могли изо всех сил пытаться сделать все, чтобы проклятая железяка изменила свое мнение о «Терминаторе», и все равно потерпели неудачу – счет 1:0 в пользу машин. Но это легко исправить, если вы используете таблицу соответствий. Просто создайте в таблице запись «Терминатор ➤ Приключение» и используйте модные машинные штучки для всего остального.

Таблицы соответствий также могут кешировать интеллектуальные данные, помогающие снизить затраты на выполнение. Например, лучший способ выяснить жанр фильма может состоять в обработке звуковой дорожки, извлечении диалогов и анализе музыки. Можно применить компьютерное зрение к каждому кадру фильма, чтобы обнаружить такие ключевые признаки, как огонь, взрывы, поцелуи, здания или что-то еще. Все это чрезвычайно ресурсоемкие вычисления, которые пока невозможно выполнять в режиме реального

времени. Вместо этого интеллект может быть создан в центре обработки данных с большим количеством аппаратных вычислительных ресурсов, загружен в таблицу соответствий в виде кеша и отправлен туда, где это необходимо.

Таблицы соответствий также могут зафиксировать хорошее поведение интеллекта. Представьте себе, что существует машинно-обучаемый интеллект, который работает в течение долгого времени и отлично справляется с классификацией фильмов по жанрам. Но Голливуд начал снимать новые жанрово неоднозначные фильмы. Таким образом, ваш интеллект был фантастическим до 2017 года, но, похоже, с 2018 года что-то пошло не так. Нужно ли отбрасывать точно настроенный и очень успешный интеллект, который трудился до 2018 года? В этом нет необходимости. Мы можем использовать прежний интеллект для анализа каждого старого фильма, снятого до 2018 года, и поместить ответы в таблицу соответствий. А тем временем мы создадим совершенно новый интеллект для работы с любыми сумасшедшими творениями, которые Голливуд решит выдать за развлечения в 2018 году и в последующий период. Таким образом, мы зафиксируем успешные результаты и сохраним непрерывность опыта пользователя.

Таблицы соответствий полезны, когда:

- есть некоторые общие контексты, которые популярны или важны, и стоит потратить время на ручное создание интеллекта для них;
- другие источники решений делают ошибки, которые трудно исправить, и вы хотите иметь простой способ преодолеть проблемы;
- вы хотите уменьшить затраты на выполнение, кешируя результаты интеллекта;
- вы хотите зафиксировать успешные результаты интеллекта, который работает хорошо.

Таблицы соответствий плохо работают, если:

- смысл контекстов меняется со временем, как это происходит с постоянно меняющимися проблемами или целями;
- таблица соответствий становится большой и слишком громоздкой для хранения по месту дислокации (особенно на стороне клиента);
- таблица соответствий должна быстро меняться, например если вы пытаетесь решить слишком обширную задачу с помощью ограниченного рукотворного интеллекта, а не использовать методы, которые лучше масштабируются (например, машинное обучение).

Представление интеллекта в моделях

Модели являются наиболее распространенным способом представления интеллекта. Они кодируют интеллект на базисе данных, согласно некоторому набору правил. Среда выполнения интеллекта способна загружать модели и безопасно и эффективно выполнять их по запросу пользователя (или интеллектуальной системы).

В большинстве интеллектуальных систем машинное обучение и модели составляют основную часть интеллекта, в то время как другие методы используются для поддержки и заполнения пробелов.

Модели могут работать всевозможными способами, некоторые из которых интуитивно понятны, а иные выглядят довольно безумно. В целом модели собирают признаки контекста, тестируют значения этих признаков, умножают их друг на друга, масштабируют их и т. д. Даже простые модели для получения своих прогнозов могут выполнять десятки тысяч операций.

Существует множество типов моделей, но три наиболее распространенные – это линейные модели, деревья решений и нейронные сети. Мы рассмотрим эти разновидности моделей более подробно, но они являются лишь верхушкой айсберга. Если вы хотите стать профессиональным разработчиком интеллекта, вам следует очень подробно изучить как эти, так и многие другие типы моделей.

Линейные модели

Линейные модели (linear models) получают на вход значения признаков контекста, умножают каждое из значений на связанный «фактор важности» (весовой коэффициент), а затем суммируют их. Полученный результат потом преобразуется в ответ (вероятность, регрессия или классификация).

Например, в случае пеллетного гриля простая линейная модель может выглядеть так:

```
TemperatureInOneMinute = (.95 * CurrentTemperature)
     + (.15 * NumberOfPelletsReleasedInLastMinute)
```

Говоря человеческим языком, через минуту гриль должен стать немного холоднее, чем сейчас, но если мы недавно добавили гранулы, температура будет немного выше. На практике линейные модели сочетают в себе гораздо больше условий (сотни, даже многие тысячи).

Линейные модели работают лучше всего, когда связь между контекстом и предсказаниями достаточно линейна. Это означает, что каждому увеличению контекстной переменной соответствует пропорциональное изменение значения правильного прогноза. В случае пеллетного гриля это означает, что линейная модель работает лучше всего, если температура увеличивается на:

- 0,15 °C для первой вброшенной гранулы;
- 0,15 °C для второй вброшенной гранулы;
- 0,15 °C для третьей вброшенной гранулы
- и т. д.

Но реальный мир устроен иначе. Если вы одновременно поместите в гриль миллион гранул, температура не увеличится на 150 000°…

Сравните это с нелинейным откликом, например когда уменьшается отдача от каждой следующей гранулы:

- 0,15 °C для первой вброшенной гранулы;

- 0,075 °C для второй вброшенной гранулы;
- 0,0375 °C для третьей вброшенной гранулы
- и т. д.

Эта убывающая взаимосвязь лучше подходит для пеллетного гриля, но линейные модели не могут напрямую реализовать подобные типы взаимосвязей. Тем не менее линейные модели – это хороший вариант для начала. С ними легко работать, они относительно легко читаются человеком, их можно быстро создавать и выполнять, и они зачастую удивительно эффективны (даже когда применяются для моделирования задач, которые не являются абсолютно линейными).

Деревья решений

Деревья решений (decision trees) являются одним из способов представления набора тестов `if` / `then` / `else`. В случае пеллетного гриля дерево решений может выглядеть примерно так:

```
if(!ReleasedPelletRecently) // Гриль начнет остывать.
{
        if(CurrentTemperature == 99)
        {
                return 98;
        }
        else if(CurrentTemperature == 98)
        {
                return 97;
        }
        else... // И т. д.
}
else
// Мы должны были недавно вбросить гранулу, поэтому температура возрастет.
{
        If(CurrentTemperature == 99)
        {
                return 100;
        }
        else... // И т. д.
}
```

Эта серия операторов `if` / `then` / `else` может быть записана в виде древовидной структуры в файле данных, который можно загрузить в среду выполнения. Корневой узел содержит первый тест `if`; у него есть один потомок для положительного ответа и один потомок для отрицательного ответа и т. д. – чем больше тестов, тем больше узлов-потомков. Листья дерева содержат ответы.

Чтобы интерпретировать дерево решений в среде выполнения, начните с корня, выполните указанный тест в соответствии с контекстом, перейдите к дочернему узлу, связанному с результатом теста, и повторяйте операции, пока не дойдете до листа, а затем верните ответ. Взгляните на рис. 17.1.

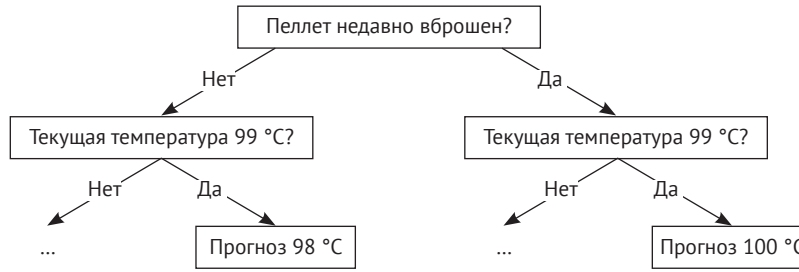

Рис. 17.1 ❖ Дерево решений для пеллетного гриля

Деревья решений могут быть довольно большими и содержать тысячи и тысячи тестов. В данном примере – прогнозирование температуры на одну минуту в будущем – в дереве решений потребуется один тест для каждой возможной температуры.

Это пример того, как представление неэффективно решает задачу прогнозирования. Попытка предсказать точную температуру гораздо более естественна для линейной модели, чем для дерева решений, потому что дерево решений должно расти до тех пор, пока не охватит все возможные значения температуры, в то время как линейная модель не нуждается в росте. Впрочем, вы можете использовать дерево решений для этой задачи, но в несколько ином виде – классификация, станет гриль горячее или холоднее за одну минуту (вместо попытки вычислить точное значение регрессии температуры). Эта версия дерева решений показана на рис. 17.2.

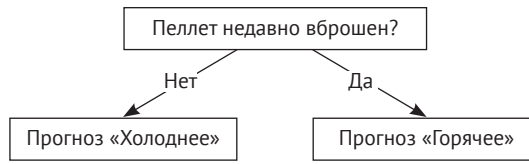

Рис. 17.2 ❖ Дерево решений
для модели пеллетного гриля с классификацией

Для решения более сложных задач простые деревья решений часто объединяют в ансамбли, называемые лесами и состоящие из десятков отдельных деревьев, где каждое дерево моделирует проблему немного по-своему (возможно, путем ограничения того, какие функции может рассматривать каждое дерево), а окончательный ответ определяется голосованием всех деревьев.

Нейронные сети

Искусственные нейронные сети (artificial neural networks) представляют модели способом, который имитирует принцип работы биологического мозга

(рис. 17.3). Мозг состоит из клеток, называемых *нейронами (neuron)*. Каждый нейрон получает *входные сигналы* (input signals) от органов чувств или других нейронов и активируется – создает выходной *сигнал активации* (activation signal) – если комбинация входных сигналов нейрона достаточно сильна. Когда нейрон активируется, он посылает сигнал другим нейронам, все дальше и дальше, через всю нашу голову, и в конечном итоге контролирует тело, порождая каждое движение, мысли и действия, которые когда-либо совершал каждый человек. Невероятно.

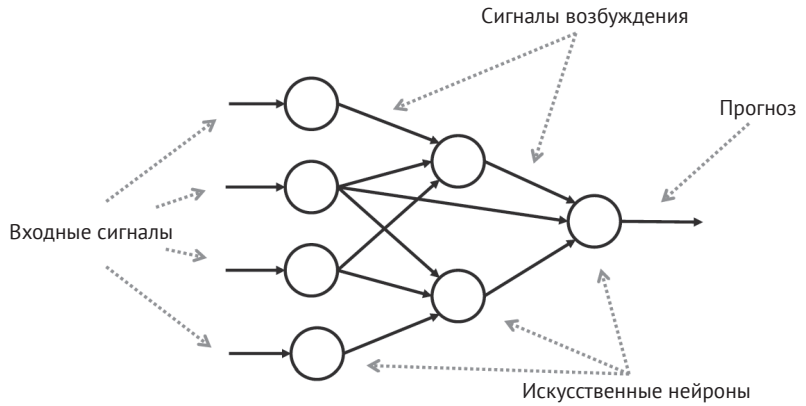

Рис. 17.3 ❖ Компоненты нейронной сети

Искусственная нейронная сеть имитирует деятельность мозга, используя *искусственные нейроны* (artificial neurons), соединенные друг с другом. Некоторые из искусственных нейронов берут входные сигналы из контекста. Однако большинство искусственных нейронов получает входные сигналы с выходов других искусственных нейронов. И некоторые из нейронов посылают свой выходной сигнал во внешний мир в качестве *предсказания* (классификация, вероятность, регрессия или ранжирование).

По сравнению с другими типами моделей, искусственные нейронные сети трудны для понимания. Вы не можете посмотреть на взаимосвязи искусственных нейронов и получить представление о том, что они делают.

Но искусственные нейронные сети оказались весьма успешными в решении важных задач, в том числе:

- компьютерное зрение;
- понимание речи;
- языковой перевод
- и т. д.

Искусственные нейронные сети особенно хороши для решения очень сложных задач, когда у вас есть огромный объем данных, доступных для обучения.

Итог главы

Интеллект должен быть представлен в такой форме, чтобы его можно было легко распространять и безопасно выполнять. Представление должно способствовать процессу создания и обучения интеллекта.

По этим причинам интеллект обычно представляют в виде файлов данных, которые загружаются в интеллектуальную среду выполнения и интерпретируются по мере необходимости, обычно с использованием таблиц соответствия или моделей. Тем не менее при подходящих условиях интеллект может быть представлен в виде программы.

Распространенные типы моделей включают в себя линейные модели, деревья решений и нейронные сети, но существует множество других вариантов.

Большинство крупных интеллектуальных систем использует несколько представлений для своего интеллекта – как те, которые можно оптимизировать путем машинного обучения, так и те, которые создаются людьми в дополнение к машинному обучению.

Темы для размышлений

Прочитав эту главу, вы должны:

- понять, как обычно представляют интеллект, и почему это делают именно так;
- уметь обсуждать распространенные типы моделей и приводить примеры их сильных и слабых сторон.

Постарайтесь ответить на следующие вопросы:

- в каких случаях интеллект, созданный человеком, имеет преимущество перед интеллектом с машинным обучением?
- создайте простой (из 10–15 узлов) интеллект на основе дерева решений для другой интеллектуальной системы, обсуждаемой в этой книге. Является ли дерево решений хорошим выбором для данной задачи? Если нет, как вы можете переформулировать задачу, чтобы она лучше соответствовала дереву решений?

Глава 18

Процесс создания интеллекта

Создание интеллекта – это процесс разработки программ, таблиц соответствия и моделей, которые отображают контексты в прогнозы. Правильно организованный процесс создания интеллекта удовлетворяет следующим условиям:

- создаваемый интеллект **достаточно точен для достижения целей системы**. Уровень «достаточной точности» зависит от системы. Например, если интеллектуальный опыт очень силен (автоматизирует действия, которые трудно отменить), интеллект должен быть чрезвычайно точным. С другой стороны, если опыт пассивен, можно ограничиться менее точным интеллектом;
- интеллект создается **достаточно быстро, чтобы не устаревать**. Если целевая проблема быстро меняется, процесс создания интеллекта должен, как минимум, не отставать от изменений;
- создаваемый интеллект **экономически эффективен и надежен**. Стоимость развития интеллекта и его поддержания в течение жизненного цикла интеллектуальной системы не должна выходить за рамки разумного бюджета. Процесс создания интеллекта должен быть устойчивым к изменениям в штатном расписании и человеческим ошибкам;
- создаваемый интеллект **соответствует условиям использования**. Он не должен требовать слишком много ресурсов процессора или объема оперативной памяти. Он должен быть достаточно компактным для оперативного распространения по местам выполнения. При разработке интеллекта следует использовать входные данные, доступные во время выполнения, – точно такие же, как при практическом использовании системы.

Создание интеллекта итеративно по своей природе. Вот некоторые из основных этапов разработки, через которые проходит разработчик интеллекта:

- понимание задачи и окружения;
- определение критериев успеха;
- получение данных;

- подготовка инструментов оценки;
- создание простого эвристического интеллекта;
- использование машинного обучения;
- оценка и повторение.

В этой главе мы обсудим упомянутые этапы и рассмотрим пример процесса создания интеллекта.

Пример создания интеллекта

Давайте рассмотрим пример создания интеллекта – разработку детектора моргания глаза.

Представьте, что вам нужно создать интеллектуальную систему, которая обнаруживает моргание глаза. Может быть, ваше приложение аутентифицирует пользователей по радужной оболочке глаза, поэтому вам необходимо отфильтровать закрытые глаза и позволить интеллекту сосредоточиться на рисунке радужной оболочки. Или, возможно, вы создаете новое приложение для знакомств, где пользователи подмигивают профилям пользователей, с которыми они хотели бы встретиться. Или вы создаете игру ужасов и хотите начислять штрафные очки игрокам, закрывающим глаза от страха.

Понимание задачи и окружения

Первый шаг в каждом прикладном проекте по созданию интеллекта – понять, что вы пытаетесь сделать на самом деле.

Обнаружить моргание, верно? Я имею в виду, какая часть задачи «обнаружить моргание» сбивает вас с толку?

Ну ладно… Есть некоторые важные моменты, о которых вам нужно знать, чтобы добиться успеха. Они должны быть уже знакомы вам, если вы действительно читали предыдущие главы книги, а не просто перескочили к этой главе, потому что искали «полезные вещи». Но независимо от того, как вы сюда попали, полезно рассмотреть эти моменты в контексте примера.

Важные вопросы об окружающей среде, в которой должен работать интеллект:

- **откуда поступает информация?** С какого источника будут поступать изображения глаз? Будет ли стандартизирован источник изображения или у разных пользователей будут разные камеры? Существует ли камера сейчас или это будет что-то новое (например, новый смартфон, который находится в стадии разработки)?
- **в каком формате будет входной сигнал?** Это будет одиночное изображение? Короткий видеоклип? Потоковое видео? Как будет выбран нужный входной сигнал (изображение или клип) среди всех возможных изображений или клипов, которые генерирует источник?
- **где будет использоваться продукт?** Будет ли он использоваться на настольных компьютерах? На ноутбуках? В закрытом помещении? На

открытом воздухе? Будет ли предусмотрена какая-то калибровка, чтобы помочь пользователям правильно настроить свои устройства, или придется работать с данными, какими бы они ни были (например, если у пользователя перевернута камера, очень сильная фронтальная засветка или какая-то грязь на объективе)?

- **как будет обработан входной сигнал?** Будут ли изображения проходить через предварительную обработку? Например, в камере может периодически срабатывать режим низкой освещенности, который существенно влияет на изображение. Возможно, в прошивке камеры есть функции автоматической настройки контрастности или коррекции цвета. Возможно, отбрасываются кадры без человеческих лиц или расположение глаз заранее помечено каким-либо другим компонентом системы;
- **как оформлен вывод интеллекта?** Должен ли вывод интеллекта быть классификацией (то есть флагом «истина», если глаз закрыт, и «ложь», если глаз открыт)? Или это будет оценка вероятности (1,0, если глаз закрыт, и 0,0, если глаз открыт)? А может, результат должен быть регрессией, указывающей на степень открытости глаза (1, если глаз полностью открыт, 0,5, если глаз наполовину открыт)? Или вывод должен быть каким-то другим?
- **какие ресурсы может использовать интеллект?** Сколько оперативной памяти доступно для модели? Какую долю вычислительной мощности процессора может занимать модель? Каковы требования к задержке от получения входных данных до получения выходных данных?

Перед началом работы необходимо задать много вопросов, и ответы на них очень важны. В любом случае, вам необходимо разработать детектор моргания, но работа, которую вам придется проделать, будет сильно зависеть от содержания ответов.

В самом деле, для некоторых комбинаций ответов проблема будет намного сложнее, чем для остальных. Сравните две ситуации: детектор моргания, который работает в тщательно контролируемом киоске в магазине с практически не ограниченными вычислительными ресурсами; и детектор моргания, который должен работать как внутри помещений, так и снаружи на датчике, который еще не существует (вы можете получить данные только из отладочного прототипа), у вас есть всего 4 Мб ОЗУ для модели, и вы должны получить ответ за 2 мс или того меньше на маломощном ЦПУ.

В обоих случаях требуются детекторы моргания, но во втором случае создать интеллект будет намного труднее.

Иногда ответы на вопросы зависят от обсуждения с другими членами команды. Возможно, одни параметры жестко заданы заранее (например, место использования продукта), но другие открыты для переговоров (например, сколько аппаратных ресурсов вам выделят для интеллекта). Ваша задача как создателя интеллекта заключается в том, чтобы оценить влияние условий окружения на качество будущего интеллекта и взаимодействовать с командой проекта в интересах общего успеха.

Определение критериев успеха

Успешный детектор моргания должен быть достаточно точным. Но насколько точным? Это зависит от того, как будет использоваться интеллект. На этом этапе вам нужно подумать об интеллектуальном опыте и о том, как различные уровни точности изменят отношение пользователей к системе в целом. Вопросы для размышлений могут быть следующими:

- сколько ошибок увидят пользователи за день?
- сколько успешных взаимодействий приходится на каждое неудачное взаимодействие?
- насколько дорого обходятся ошибки для пользователя?

Вероятно, поиск ответов на эти вопросы будет включать в себя дискуссии с людьми, создающими интеллектуальный опыт (а если вы отвечаете как за интеллект, так и за опыт, вам придется говорить с самим собой, независимо от того, насколько глупо вы себя чувствуете при этом). Обдумайте различные варианты взаимосвязи точности и опыта – как пользователи будут воспринимать ошибки и как они смогут исправлять их.

И помните: интеллект способен совершать любые ошибки. Важно решить, какие типы ошибок являются допустимыми для детектора моргания, и договориться с командой, какие компромиссы подходят для проекта.

Например, интеллект может сказать, что глаз открыт, когда он закрыт, или наоборот. Какая из этих ошибок допустима для вашей системы?

В системе контроля доступа по радужной оболочке детектор морганий пытается найти чистые и понятные кадры с изображением открытого глаза. Эта система должна отсеять как можно больше изображений с закрытыми глазами – всякий раз, когда интеллект полагает, что глаз открыт, он и в самом деле должен быть открыт!

В случае игры ужасов детектор моргания пытается оштрафовать игрока, закрывающего глаза. Интеллект не должен нагнетать напряжение в неподходящее время – всякий раз, когда считается, что глаз закрыт, он и в самом деле должен быть закрыт!

В процессе создания интеллекта будут приниматься решения, из-за которых модель будет больше или меньше совершать те или иные ошибки, – поэтому полезно заранее знать, в какую сторону вы идете. Хорошо, когда мнения разных людей в команде совпадают, – вы ведь не хотите, чтобы разработчики пользовательского интерфейса имели иное представление о приемлемых ошибках, чем создатели интеллекта.

Получение данных

Данные имеют решающее значение для создания интеллекта. Как минимум, вам нужно достаточно данных, чтобы понять проблему и убедиться в работоспособности начальной версии интеллекта. Вам также понадобится много обучающих данных, если вы хотите заниматься машинным обучением интел-

лекта прямо из коробки. Вам необходимо решить две основные задачи, связанные с набором данных:

- получение начальных данных для обучения интеллекта;
- сбор данных от пользователей во время использования системы.

А еще вспомните, о чем мы говорили в главах о получении данных из опыта и о телеметрии: обучающие данные должны быть несмещенными (unbiased) и максимально соответствовать тому, с чем столкнутся пользователи на практике.

Данные для начального запуска

Существуют разные способы получения *данных для начального запуска* интеллекта (bootstrap data). Вот несколько способов, которые могут пригодиться для детектора моргания:

- **поиск данных в интернете.** Скачайте как можно больше изображений глаза, которые соответствуют по своим характеристикам (разрешение, расстояние до глаза и т. д.) изображениям, поступающим с видеокамеры детектора моргания. Затем заплатите людям, чтобы они разделили изображения на две группы – с открытым и закрытым глазом;
- **соберите собственные данные.** Возьмите камеру, наиболее похожую на встроенную камеру детектора моргания, и попросите несколько сотен человек взглянуть в камеру и закрыть и открыть глаза в соответствии с определенным сценарием сбора данных;
- **найдите или купите хороший набор данных.** Многие разработчики занимаются компьютерным зрением. Возможно, кто-то уже создал набор изображений глаз. Вы можете найти общедоступный набор данных или компанию, желающую продать тщательно отобранный набор.

Объем данных, необходимых для начального запуска, будет зависеть от сложности проблемы. В случае компьютерного зрения интуитивно можно предположить следующее:

- попытка обучить детектор моргания на нескольких тысячах изображений вряд ли будет удачной;
- детектор моргания, обученный на десятках тысячах изображений, вероятно, будет неплохо работать для начала;
- при масштабировании до сотен тысяч или миллионов изображений точность интеллекта будет возрастать, но затем отдача от дальнейшего наращивания объема начнет снижаться.

Хорошей практикой является обучение интеллекта на последовательно возрастающих объемах данных, чтобы оценить отдачу от инвестиций в сбор данных. Этот подход называется *кривой обучения* (learning curve). Постройте модель на сотне изображений и оцените ее качество. Затем постройте модель на базе пяти сотен изображений и вновь оцените качество. Потом возьмите тысячу изображений… Вы сможете увидеть, в какой мере добавление данных увеличивает точность модели, и принять решение о том, сколько потратить на сбор данных для начального обучения.

Данные из взаимодействий

Правильно спроектированная интеллектуальная система при взаимодействии с пользователями генерирует собственные обучающие данные, но этот подход не всегда удается реализовать. На данном этапе полезно пообщаться с дизайнерами пользовательских интерфейсов и сформировать общую стратегию. Для системы с детектором моргания можно предложить следующие подходы:

- **привязать сбор данных к успешному решению задачи.** Например, когда пользователь успешно входит в систему с помощью распознавания радужной оболочки глаза, соответствующее изображение помечается как хороший обучающий образец. Если пользователь не смог войти в систему и вынужден вручную ввести пароль, это считается примером неудачного изображения, которое следует отбросить;
- **создать специальный опыт сбора данных.** Например, можно организовать начальную настройку, когда пользователи регистрируются в системе, открывают и закрывают глаза для калибровки устройства, а система заодно собирает обучающие данные. Или, может быть, в игре есть интерактивный учебник, который побуждает пользователей открывать и закрывать глаза в определенное время и фиксирует состояние глаза при помощи щелчка мышью (и собирает обучающие данные).

В упомянутых ситуациях опыт сбора данных прозрачен для пользователей или стимулирует их поступать в соответствии с вашими требованиями к данным. Создание данных будет происходить достаточно часто, поэтому полученных данных будет достаточно для обучения интеллекта. Но процедура сбора данных не должна выглядеть пугающей или нарушать конфиденциальность пользователя.

Подготовка инструментов оценки

С данными в руках вы почти готовы приступить к созданию интеллекта. Но вы не добьетесь особого успеха, если не сможете оценивать созданный вами интеллект. При создании интеллекта вы должны повторять мантру «оценка означает создание». Вам придется проводить оценку достаточно часто. Оценка интеллекта включает в себя следующие шаги.

1. **Приготовьте данные для оценки.** Убедитесь, что отложен достаточный объем оценочных данных и они никак не связаны с данными для обучения интеллекта. В случае с детектором моргания вы можете разделить пользователей (все изображения одного и того же человека используются либо для создания интеллекта, либо для его оценки), а также создать оценочные наборы для популяций: пользователи в очках, определенной этнической принадлежности, пола и возраста.
2. **Создайте среду для выполнения оценки.** Это среда, в которой интеллект будет обрабатывать тестовые данные *в точно таких же условиях, как*

во время обычной работы интеллектуальной системы. Повторяю: в Абсолютно Одинаковых Условиях. И еще раз убедитесь, что среда для оценивания полностью совпадает со средой выполнения.

3. **Создайте автоматические отчеты о качестве интеллекта.** Отчеты можно использовать, чтобы увидеть:
   - насколько точен интеллект;
   - какие ошибки он совершает – приемлемые или недопустимые;
   - есть ли популяции, для которых точность значительно хуже;
   - каковы худшие из ошибок, которые делает интеллект;
   - как интеллект развивается с течением времени (скорость улучшения).

Чем проще выполнить оценку, тем лучше. Можно обойтись ручным трудом, но небольшие инвестиции в инструменты, помогающие оценить интеллект, могут быстро окупиться за счет качества интеллекта.

Простой эвристический интеллект

Создание простого эвристического интеллекта может оказаться полезным по нескольким причинам:

1) он поможет вам убедиться, что проблема действительно сложная. Ведь если простой эвристический интеллект решит проблему, вы можете на этом и остановиться, сэкономив время и деньги;
2) он заставит вас обдумать затруднения, присущие решаемой проблеме, понять данные и начать рассуждать о типах признаков, данных и телеметрии, а это, в свою очередь, поможет вам успешно создать интеллект;
3) он формирует базовый уровень для сравнения с более продвинутыми методами. Если ваш интеллект сложен, дорог и не очень-то эффективен по сравнению с простой эвристикой, возможно, вы свернули не туда.

Этот шаг можно пропустить, однако он помогает сориентироваться и отладить остальные инструменты и данные, прежде чем вводить в работу более сложные (и более трудные для понимания) версии интеллекта.

При разработке детектора моргания вы можете попробовать следующие методы:

1) измерение градиентов изображения в горизонтальном и вертикальном направлениях, потому что форма открытого и закрытого глаз различается;
2) сравнение цвета пикселей изображения с образцовыми цветами «глаз» и «кожа». Если обнаружено много пикселей цвета «глаз», то, вероятно, глаз открыт. И наоборот, если найдено много пикселей цвета «кожа», то, вероятно, глаз закрыт;
3) вписывание эллипса в середину изображения. Если на вашем изображении эллипс хорошо совпадает с градиентом формы радужной оболочки, то, вероятно, глаз открыт. Если это не так, глаз может быть закрыт.

Затем вы можете установить пороговые значения для всех упомянутых замеров и создать простое решающее правило. Например, позвольте каждому из методов голосовать «открыто» или «закрыто» и принимайте решение большинством голосов.

Будет ли такой интеллект достаточно хорош для практического применения? Ни в коем случае. Но это лишь начало.

Учтите, что компьютерное зрение – это обширная прикладная область с большим количеством методов. Если у вас есть опыт разработки компьютерного зрения, ваша эвристика будет более сложной. В противном случае ваша эвристика может оказаться такой же плохой, как и у меня. Не бойтесь. Придумайте какие-нибудь идеи, дайте им шанс выжить и выпускайте поезд интеллекта на рельсы.

Машинное обучение

Теперь пришло время подняться в создании интеллекта на следующий уровень. И это почти наверняка означает машинное обучение. Есть много подходов, которые вы можете попробовать, но иногда лучше начать с простого. Найдите самый простой «стандартный» подход к тому типу проблемы, с которой вы работаете. Но имейте в виду, что стандарты меняются. Например:

- примерно за десять лет до того, как была написана эта книга, самый разумный подход к обнаружению моргания был бы следующим: поиск определенных фрагментов (паттернов) на изображении; поиск наилучших совпадений с правильными изображениями; а затем построение модели, в которой все обнаруженные паттерны голосуют (с некоторым весовым коэффициентом) за ответ;
- примерно за пять лет до того, как была написана эта книга, вполне разумным подходом для обнаружения моргания могло бы считаться использование огромных коллекций деревьев решений, сравнивающих очень простые свойства изображений (например, различия в интенсивности пикселей в заранее определенных точках);
- на момент написания этой книги очень разумным подходом для обнаружения моргания (при наличии обучающих данных) считалось использование сложной искусственной нейросети, которая обрабатывает сырые значения пикселей без предварительной подготовки данных.

Что будет спустя пять лет после выхода этой книги? Кто знает. Для интеллектуальной системы (и для остальных подходов, упомянутых в этой книге) не имеет значения, какую технику машинного обучения вы используете, при условии что результирующая модель правильно работает в среде выполнения. Найдите современный инструментарий машинного обучения. Прочитайте несколько веб-страниц. Попробуйте самый простой подход, который можно реализовать с текущими инструментами (или, может быть, несколько простых подходов). Не расходуйте сейчас кучу времени, еще не пора. Просто получите нечто работающее.

Поиск компромиссов

На этом этапе вы можете провести небольшое промежуточное исследование, чтобы облегчить разработку правильной реализации. Это этап изучения ограничений и компромиссов. Постарайтесь ответить на следующие вопросы:

- как качество интеллекта зависит от вычислений в среде выполнения?
- насколько успешно может работать интеллект с ограниченным объемом оперативной памяти?
- сколько раз в неделю планируется обновлять интеллект?
- какую выгоду получает система от добавления определенных элементов в контекст, особенно тех, которые наиболее дороги в реализации?
- какова сквозная задержка выполнения интеллекта при текущей аппаратной реализации?
- какие разновидности наихудших ошибок, влияющих на пользователя, вероятно, допустит интеллект?

Ответы на эти вопросы помогут решить, где должен располагаться интеллект, какие системы поддержки создавать, как настраивать опыт и многое другое.

Ключевым требованием является гибкость. Например, можно включить в реализацию дополнительный код, чтобы скрыть задержку, или добавить альтернативный опыт для смягчения ошибок. Реализация может быть настроена на распространение нового интеллекта по всему миру каждые 15 минут. Но эти решения могут быть довольно дорогими. Иногда небольшие изменения в реализации – какую из моделей использовать, какие особенности учитывать, насколько агрессивным должен быть опыт – могут более элегантно решать проблемы, что приводит к общему улучшению системы.

Оценка и повторение

Пришло время остановиться и перевести дух. Посмотрите, где вы находитесь, оцените качество интеллекта, поймите, насколько далеко вы продвинулись от эвристического интеллекта (или от предыдущей итерации), и подумайте, что делать дальше.

Существует множество способов улучшения системы. Вы можете попробовать собрать больше данных или просто другие данные. Вы можете попробовать более сложные контексты и признаки, извлекаемые из них. Вы можете попробовать более сложное машинное обучение. Вы можете усовершенствовать инструментарий оценки. Вы можете попытаться изменить мнение людей о жизнеспособности целей системы. Вы можете попытаться доработать опыт, чтобы лучше справляться с ошибками системы.

А затем вы все это повторяете, повторяете и повторяете. При удачно сложившихся обстоятельствах итеративная часть процесса будет пройдена довольно быстро, но может продолжаться месяцами, годами и десятилетиями при решении очень сложной и очень важной проблемы.

Уровни зрелости интеллекта

Интеллектуальная система может прожить много лет и за это время многократно улучшиться. После вас ее могут поддерживать люди, с которыми вы даже незнакомы. Вот некоторые стадии зрелости, через которые может пройти ваш творческий процесс создания интеллекта:

- **вы завершили однократную разработку.** Вы создали полезный интеллект, который можно отправлять клиентам, но в процессе вы провели много исследований, разработали множество сценариев, отредактировали их и устроили небольшую путаницу. Для повторного создания интеллекта потребуется дополнительная работа;
- **вы можете повторить разработку при необходимости.** Вы создали интеллект и задокументировали свои действия. Вы можете повторить ваши шаги и воспроизвести тот же интеллект за разумное количество времени. Или, что еще лучше, кто-то другой может взять вашу документацию, повторить ваши действия и воспроизвести точно такой же интеллект;
- **вы можете легко обновить интеллект.** Вы создали интеллект, и у вас есть несколько хороших инструментов и сценариев, с помощью которых можно легко повторить процесс разработки. Допустим, сценарии получают новую телеметрию, обрабатывают ее, выполняют компиляцию, строят модель и создают диаграммы, демонстрирующие, насколько хорошо будет работать новый интеллект, – и все это одним нажатием кнопки;
- **интеллект обновляет себя сам.** Вы создали систему, которая автоматически воспроизводит интеллект на регулярной основе и отправляет новый интеллект в заданное место, а отчеты о качестве отправляются по электронной почте нужным людям;
- **интеллект обновляет и при необходимости развертывает себя сам.** Ваша система, которая автоматизирует воспроизводство интеллекта, заодно автоматизирует и развертывание. У нее есть тесты, которые проверяют качество интеллекта. Она знает, как рассылать новый интеллект, развертывать его среди все большего и большего числа пользователей, и будет отменять изменения и предупреждать оркестровщиков, если что-то пойдет не так.

При построении интеллектуальной системы вы, вероятно, начнете с нижнего уровня этого спектра зрелости и остановитесь у более высокого уровня. Полная автоматизация применяется в основном для открытых или меняющихся во времени проблем. В целом ваша задача состоит в том, чтобы разработать самый дешевый способ создания интеллекта на уровне, достаточном для достижения целей интеллектуальной системы.

Мастерство создания интеллекта

Как и в большинстве человеческих начинаний – люди, которые лучше всех разбираются в создании интеллекта, значительно опережают людей со средними навыками.

В дополнение к базовым навыкам программирования и обработки данных продвинутые создатели интеллекта обладают следующими навыками:

- анализ и отладка данных;
- отладка на основе проверок и оценок;
- интуиция в выборе инструментов;
- математика.

В данном разделе рассмотрены все эти требования.

Анализ и отладка данных

Чтобы смотреть на данные и понимать, что происходит, создателю интеллекта нужно обладать особым складом ума. Что пытаются сказать данные? Какие истории скрываются за ними?

Это достаточно утомительная работа. Наподобие детектива, собирающего улики, чтобы выяснить, что произошло.

Иногда отслеживание небольшого несоответствия (например, какая-то переменная равняется нулю чаще, чем вы ожидали) может привести к значительному улучшению системы. Может быть, это признак серьезной ошибки. Возможно, это признак неверных допущений где-то в реализации. А вдруг это намек на то, что модель построена неправильно и другой подход сработает лучше?

Не у всех есть способности к анализу и отладке данных. Большинство людей, взглянув на данные, просто пожмет плечами и скажет: «Это событие встречается в 1 % выборок, ну какое оно имеет значение?» При создании интеллекта критически важно иметь склонность к отслеживанию интересных вещей в данных и опыте, чтобы знать, когда следует остановиться и присмотреться.

Отладка на основе проверок и оценок

По своей сути создание интеллекта – это выбор наиболее подходящего варианта из множества возможностей. Поэтому важно знать, какой вариант объективно лучше.

Прежде всего вам следует определить, какие тесты вы можете использовать для оценки соответствия интеллекта целям системы. Оценочный подход требует наличия базовых знаний в области статистики (или интуиции) и некоторой способности к сопереживанию, чтобы связать бездушные числа с эмоциональным опытом пользователя.

Другой аспект оценочного подхода заключается в том, чтобы максимально упростить сравнение версий интеллекта друг с другом. Фактически одна из самых длинных глав в этой книге называется «Оценка интеллекта», и вы скоро до нее доберетесь. Если мысль о чтении длинной-длинной главы об оценке интеллекта вызывает у вас легкую грусть – наверное, разработка интеллекта не для вас.

Интуитивное знание инструментария

Вы должны обладать глубоко интуитивным пониманием работы инструментов для создания интеллекта, включая модели машинного обучения и подходы на основе извлечения признаков.

Инструменты для создания интеллекта бывают весьма своенравными. Они не скажут вам, что не так, и каждый случай индивидуален. Методы, которые хорошо работают с одним алгоритмом машинного обучения, могут не работать с другим. Важно развить интуицию, чтобы охватывать одним взглядом инструменты, результаты и ситуацию, а также чувствовать, на что обратить внимание, чтобы добиться прогресса.

Этот навык немного похож на *эмоциональный интеллект* (EQ, Emotional IQ), но для машинного обучения (MQ?). Представьте, что с вами в комнате сидят незнакомые люди, и вам нужно понимать язык их тел, читать между строк смысл того, что они говорят, и выяснять, сумасшедшие ли они (и если да, то насколько), а затем решать, что вы можете сделать, чтобы помочь им.

Математика – нужна ли она?

Теперь о математике. Большая часть теории машинного обучения была создана математически мыслящими исследователями. Из-за этого изучение основ машинного обучения обычно начинается с математики. Но я полагаю, что эксперт *прикладного* машинного обучения вполне может обойтись без сложной математики.

В самом деле?

Да, именно так.

Машинное обучение похоже на большинство человеческих инструментов – наши автомобили, мобильные телефоны, компьютеры, реактивные истребители – человеку, управляющему ими, не нужно понимать фундаментальные основы того, что использовано для их разработки. Я уверен, что лучшие в мире пилоты реактивных истребителей не разберутся в математике, необходимой для создания современного истребителя. Однако давайте посадим за штурвал самолета инженера, который превосходно разбирается в математике, и пусть он устроит бой с профессиональным пилотом…

Как вы думаете, кто победит?

Знание математических основ алгоритмов машинного обучения не гарантирует вам успеха. Развивайте свои сильные стороны. Не переживайте из-за того, что от вас не зависит.

Вы можете стать следующим асом машинного обучения.

Итог главы

Создание прикладного интеллекта – итеративный процесс, основными этапами которого являются:

1) изучение окружающей среды;
2) определение критериев успеха;
3) получение данных;
4) подготовка инструментов оценки;
5) простая эвристика;

6) простое машинное обучение;
7) оценка и повторение, пока вы не добьетесь желаемого успеха.

Конкретные методы построения интеллекта (особенно основанного на машинном обучении) могут изменяться со временем и от одной проблемной области к другой. Но описанный здесь общий рабочий процесс в целом неизменен.

Трудозатраты на автоматизацию этого процесса могут быстро окупиться, особенно если учесть, что придется выполнять все больше и больше итераций.

И помните – проверка, проверка и еще раз проверка.

Темы для размышлений

Прочитав эту главу, вы должны:

- знать тонкости полного цикла создания интеллекта;
- разбираться в вопросах, проблемах и действиях, на которые расходуют рабочее время разработчики интеллекта.

Постарайтесь ответить на следующие вопросы:

- выберите свою любимую интеллектуальную систему и пройдите семь этапов процесса создания интеллекта, описанного в этой главе. Какие вопросы будут иметь отношение к вашей интеллектуальной системе? Как бы вы определили успех? Как бы вы получили данные? Какие инструменты оценки вы бы использовали? Какой эвристический интеллект вы бы попробовали? И какие стандартные, простые подходы машинного обучения можно было бы применить?

Глава 19

Оценка интеллекта

Оценка интеллекта (intelligence evaluation) – творческий процесс, по крайней мере когда речь идет о создании интеллекта для интеллектуальных систем. Это связано с тем, что создание интеллекта обычно включает в себя итеративный поиск эффективной версии: создать новый интеллект, сравнить его с предыдущим кандидатом и выбрать лучший из двух. Вы должны иметь возможность окинуть взглядом пару интеллектов и ответить на такие вопросы:

- какой из них следует использовать в моей интеллектуальной системе?
- какой из них лучше реализует цели системы?
- какой из них создаст меньше проблем для меня и моих пользователей?
- достаточно ли хорош один из них для отправки пользователям или надо еще поработать?

Существует два основных способа оценки интеллекта:

- **оценка в режиме онлайн** – рассылая обновления пользователям и выясняя, как они реагируют. Мы рассуждали об этом в предыдущих главах, когда говорили об оценке опыта и управлении интеллектом (с помощью тихого интеллекта, ограниченного развертывания и флайтинга);
- **автономная оценка** – анализ того, насколько хорошо интеллект работает с историческими данными. Это предмет данной главы, так как он имеет решающее значение для процесса создания интеллекта.

В этой главе поясняется, что означает точность интеллекта. Далее мы обсудим использование данных для оценки интеллекта и некоторые подводные камни этого подхода. Затем мы рассмотрим концепцию инструментов для сравнения интеллекта. Глава завершается обсуждением методов субъективной оценки интеллекта.

Оценка точности

Разумеется, интеллект должен быть точным. Но точность не относится к простым понятиям, и вдобавок интеллект может потерпеть неудачу в самых разных ситуациях. Качественный интеллект будет обладать следующими свойствами:

- он умеет обобщать ситуации, с которыми раньше не сталкивался;
- он делает допустимые ошибки;
- он равномерно распределяет ошибки.

В этом разделе более подробно рассматриваются перечисленные свойства.

Обобщение

Одной из ключевых задач в создании интеллекта является создание такого интеллекта, который способен к *обобщению*, то есть хорошо работает с вещами, о которых не было известно на момент разработки.

Представьте себе студента, который читает учебник по предмету и запоминает каждый абзац. Неплохо. Студент очень усерден в попугайстве вещей, которые изложены в учебнике. А теперь представьте, что учитель составил тест, не требующий от ученика повторения фактов из учебника. Вместо этого учитель хочет, чтобы ученик продемонстрировал понимание концепций из учебника и применил их в новой обстановке. Если студент разработал хорошую ментальную модель по теме, он сможет пройти этот тест. Если у студента неправильная ментальная модель по теме или у него вообще нет ментальной модели (и он лишь тупо вызубрил текст), ему не удастся применить знания в новой обстановке. То же самое можно сказать про интеллект в интеллектуальной системе – он должен охватывать новые ситуации.

Давайте рассмотрим еще один пример. Подумайте о создании интеллекта, который исследует книги и классифицирует их по жанрам – научная фантастика, любовная проза, технические, триллеры, исторические романы и т. п.

Вы берете тысячу книг, сортируете их вручную по жанрам и приступаете к созданию интеллекта. Цель интеллекта состоит в том, чтобы иметь возможность взять новую книгу – такую, которая не является частью отобранной тысячи, – и точно предсказать ее жанр.

Допустим, вы создали интеллект, сохраняя информацию об авторах. Вы можете посмотреть на свою тысячу книг, обнаружить, что у них 815 разных авторов, и составить список наподобие следующего:

- Рой Ройерсон пишет ужасы;
- Тим Тини пишет фантастику;
- Нил Нотсон пишет технические книги
- и т. д.

Когда вы получаете новую книгу, то ищете ее автора в этом списке. Если автор есть, интеллект возвращает жанр. Если автора в списке нет – вы провалились. Ваша модель не имеет понятия о жанрах, она просто запоминает некоторые факты и не обобщает их на авторов, о которых она не знает (и будет весьма озадачена авторами, которые пишут в двух разных жанрах).

При оценке точности интеллекта важно проверить, насколько хорошо он умеет обобщать. Поместите интеллект в незнакомые ситуации и оцените, насколько хорошо он адаптируется.

Типы ошибок

Интеллект может совершать ошибки разного типа, и некоторые ошибки вызывают больше проблем по сравнению с другими. Мы упоминали концепцию

ложного положительного и *ложного отрицательного* вывода в главе 6, когда обсуждали интеллектуальный опыт, но давайте поговорим об этом еще раз (табл. 19.1). При прогнозировании классификаций интеллект может:

- утверждать, что сущность относится к определенному классу, когда это не так;
- утверждать, что сущность не относится к определенному классу, когда это так.

Таблица 19.1. Различные типы ошибок интеллекта

		Интеллект полагает, что кто-то	
		присутствует	отсутствует
На самом деле кто-то	присутствует	Истинно положительный	Ложноотрицательный
	отсутствует	Ложноположительный	Истинно отрицательный

Например, предположим, что интеллект совершает одну из следующих ошибок:

- говорит, что у двери кто-то есть, когда никого нет; или утверждает, что у двери никого нет, но там кто-то стоит;
- говорит, что надо добавить топливо в огонь, но это не так (огонь уже достаточно разгорелся); или говорит, что не следует добавлять топливо, но это не так, потому что огонь скоро погаснет;
- говорит, что книга – любовный роман, но это не так; или утверждает, что книга не роман, но это так.

Интеллект развивает свою интеллектуальную систему за счет «правильных» ошибок. Например, вернемся к интеллекту, исследующему веб-страницы, чтобы определить, смешные они или нет. Помните, я говорил вам, что этот интеллект точен на 99 %? Покажем этому интеллекту произвольную страницу, которую он раньше никогда не видел, интеллект сделает прогноз – смешно или нет, – и в 99 % случаев прогноз окажется верным. Замечательно. Очень точное обобщение. Этот интеллект, несомненно, пригодится в нашей интеллектуальной системе поиска смешных страниц.

Но вдруг окажется, что 99 % новых страниц несмешные? В этом случае интеллект может сказать «несмешно» в 100 % случаев и при этом будет точен на 99 %. Для несмешных страниц это 100 % точности. В то же время для смешных страниц это 0 % точности. А в целом все равно получается 99 % точности. Очевидно, что такая оценка точности совершенно не имеет смысла.

Одним из методов поиска оценочного компромисса между этими типами ошибок является определение коэффициента (rate) *ложноположительных* и *ложноотрицательных* прогнозов. Для системы поиска смешных страниц «положительный» – это смешная страница, а «отрицательный» – это несмешная

страница. На самом деле вы можете назвать «положительным» иное значение – будьте осторожны и давайте однозначные определения, иначе другие члены команды могут понять определение по-своему, и возникнет путаница. Итак:

- коэффициент *ложноположительных* решений определяется как отношение всех отрицательных ответов, которые ошибочно классифицированы как положительные (какая доля несмешных страниц помечается как смешная) к сумме ложноположительных и истинно отрицательных решений:

$$\text{False Positive Rate} = \frac{\text{\# False Positives}}{\text{\# False Positives} + \text{\# True Negatives}};$$

- коэффициент *ложноотрицательных* решений определяется как отношение всех положительных решений, которые ложно классифицированы как отрицательные (какая доля смешных страниц помечается как несмешная) к сумме ложноотрицательных и истинно положительных решений:

$$\text{False Negative Rate} = \frac{\text{\# False Negatives}}{\text{\# False Negatives} + \text{\# True Positives}}.$$

В соответствии с этим подходом безмозглый и вечно недовольный интеллект получит 0 % ложноположительных решений (что очень хорошо) и 100 % ложноотрицательных решений (что бесполезно).

Другой очень распространенный способ оценки ошибок – вычисление *точности* (precision) и *отзыва* (recall) модели:

- *точность* определяется как отношение всех положительных решений модели, которые являются истинно положительными (какая доля решений «эта страница смешная» является правильной) к сумме истинно положительных и ложноположительных решений:

$$\text{Precision} = \frac{\text{\# True Positives}}{\text{\# True Positives} + \text{\# False Positives}};$$

- *отзыв* определяется как отношение всех положительных результатов, которые, по мнению модели, являются положительными (какая часть действительно смешных страниц обозначена как смешная) к сумме истинно положительных и ложноотрицательных решений:

$$\text{Recall} = \frac{\text{\# True Positives}}{\text{\# True Positives} + \text{\# False Negatives}}.$$

При таком подходе безмозглый и вечно недовольный интеллект будет иметь неопределенную точность (потому что он никогда не дает положительный ответ, а вы не можете делить на ноль даже с машинным обучением) и отзыв 0 % (потому что он дает положительный ответ на 0 % положительных страниц).

Оптимальный интеллект должен стремиться к равновесию между разными типами ошибок, которые он совершает, чтобы соответствовать целям интеллектуальной системы.

Распределение ошибок

Качественный интеллект работает одинаково хорошо для всех пользователей. То есть он не должен фокусировать свои ошибки на конкретных популяциях, например:

- система для обнаружения посетителей за дверью, которая никогда не работает для людей с ростом ниже 150 см;
- система поиска лиц на изображениях, которая никогда не находит людей в очках;
- система фильтрации спама, которая всегда удаляет письма из банков.

Ошибки подобного рода могут быть очень неприятными. Они ведут к обиженным пользователям и плохим отзывам. Можно иметь интеллект, который прекрасно обобщает, хорошо уравновешивает различные типы ошибок и совершенно непригоден для использования, поскольку фокусирует ошибки на конкретных пользователях или в определенных контекстах.

Сфокусированные ошибки плохо поддаются обнаружению. Существует настолько много потенциальных групп населения, что практически невозможно предвидеть все случаи, когда сфокусированные ошибки могут вызвать проблемы для интеллектуальной системы.

Оценка других типов прогнозов

В предыдущем разделе дано введение в оценку классификаций. Но есть множество других способов оценить ответы, которые дает интеллект. Вы можете найти целые книги по этой теме, а данный раздел даст краткий обзор подходов к оценке регрессий, вероятностей и рейтингов.

Оценка регрессий

Регрессии возвращают числа. Возможно, вы хотите узнать, как часто они предсказывают «правильное» число. Но обычно полезнее знать, насколько близки предсказанные ответы к правильным ответам, чем знать, как часто ответы являются абсолютно правильными.

Наиболее распространенный способ оценить точность регрессии – рассчитать *среднеквадратичную ошибку* (Mean Squared Error – MSE). Возьмите ответ, который дает интеллект, вычтите его из правильного ответа и возведите в квадрат результат. Затем возьмите среднее значение по количеству контекстов, имеющих отношение к вашему измерению.

$$\text{Mean Square Error} = \frac{\text{Sum of (Correct\_Answer-Predicted\_Answer)}^2}{\text{Number of Context}}.$$

Если значение среднеквадратичной ошибки (СКО) мало, то прогнозы интеллекта близки к правильному ответу. Если значение СКО велико, то интеллект часто дает прогнозы, далекие от правильного ответа.

Оценка вероятностей

Вероятность (probability) – это число, лежащее в интервале от 0 до 1. Один из способов оценки точности предсказания вероятностей состоит в том, чтобы использовать пороговое значение для преобразования вероятностей в классификации, а затем оценивать их как классификации.

Хотите знать, является ли книга романом? Спросите у интеллекта вероятность того, что это роман. Если вероятность выше порога – скажем, 0,3 (30%), – значит, это роман, а иначе это что-то другое.

Использование высокого порогового значения для преобразования вероятности в классификацию, такого как 0,99 (99%), обычно приводит к высокой точности, но к более низкому отзыву – книгу можно назвать романом лишь в том случае, если интеллект абсолютно уверен.

Использование низкого порогового значения для преобразования вероятности в классификацию, например 0,01 (1%), обычно приводит к снижению точности, но к более высокому отзыву – вы называете практически любую книгу романом, если только интеллект не уверен в том, что это *не* роман.

Мы обсудим эту концепцию порогового значения немного позже, когда будем говорить о рабочих точках и сравнении интеллектов.

Еще один способ оценки вероятностей называется *логистической функцией потерь* (log loss). По сути, логистическая функция потерь очень похожа на среднеквадратическую ошибку регрессии, но с ней связано больше математики, которую мы пропустим. Достаточно лишь отметить: чем меньше потерь, тем лучше.

Оценка ранжирования

Ранжирование упорядочивает контент в зависимости от его соответствия контексту. Например, учитывая историю пользователя, какую газировку он закажет, скорее всего, на этот раз? Ранжирующий интеллект расставит все возможные варианты в порядке убывания вероятности, например:

1) кола;
2) содовая;
3) пиво из маниоки;
4) диетическая кола.

Хорошо, но как мы узнаем, насколько правильно составлен список?

Один из простых способов оценить качество ранжирования – использовать интеллектуальный опыт. Допустим, умная газировочная машина может показывать три варианта газировки (топ-3) на своем дисплее. Ранжирование является точным, если фактический выбор пользователя входит в предложенную тройку, и не является точным, если фактический выбор пользовате-

ля не соответствует начальному предложению и требует вывода следующих пунктов списка.

Вы можете использовать список топ-3, топ-1, топ-10 – любой наиболее подходящий для вашей интеллектуальной системы.

Таким образом, один из простых способов оценить ранжирование – определить процент совпадений выбора пользователя с одним из K верхних прогнозов интеллекта в списке.

Использование данных для оценки

Данные являются ключевым инструментом оценки интеллекта. В целом интеллект оценивается путем взятия исторического контекста, выполнения интеллекта на этом контексте и сравнения фактического положения дел с предсказаниями интеллекта.

Конечно, использование исторических данных имеет свои риски, в том числе:

- вы можете случайно **оценить интеллект на данных, использованных для его создания**, что приведет к чрезмерно оптимистичной оценке качества интеллекта. В основном потому, что интеллект видел правильные ответы на тест;
- основная **проблема может измениться** между моментом сбора данных тестирования и временем развертывания нового интеллекта, что также приводит к чрезмерно оптимистичным оценкам качества интеллекта. Когда проблема меняется, интеллект может быть великолепен – но это уже не его война.

В этом разделе мы обсудим способы обработки оценочных данных для минимизации упомянутых проблем.

Независимые оценочные данные

Оценочные данные (evaluation data), используемые для оценки интеллекта, должны быть полностью отделены от данных, используемых для создания интеллекта.

Представьте следующую ситуацию. Вы придумали идею какого-то эвристического интеллекта, внедрили его, оценили по некоторым данным и обнаружили, что точность составляет 54 %. На данный момент все в порядке. Реальная точность интеллекта, вероятно, будет чуть больше или чуть меньше 54 %, потому что оценочная выборка имеет ограниченный размер, но если вы развернете его для пользователей, то, скорее всего, они получат такую точность.

Затем вы отмечаете ошибки, которые ваш интеллект совершает с оценочными данными. Вы замечаете закономерность, поэтому дорабатываете интеллект и улучшаете его. Затем вы испытываете интеллект на тех же оценочных данных и обнаруживаете, что точность новой версии составляет 66 %.

А вот теперь у вас проблема. Вы посмотрели на оценочные данные и изменили свой интеллект на основе найденных ошибок. На данный момент трудно

сказать, насколько точным будет ваш интеллект при развертывании его среди пользователей – почти наверняка меньше, чем 66 %, которые вы видели в предыдущей оценке. Возможно, даже хуже, чем первоначальные 54 %.

Все дело в том, что вы сжульничали. Вы подсмотрели ответы на тесты, когда дорабатывали интеллект. Вы настроили свой интеллект на ту часть проблемы, которую видели. Это плохо.

Один из распространенных способов избежать этого – случайное разделение имеющихся данных на две части: *настроечный набор* (tweaking set) для создания и настройки интеллекта и *оценочный набор* (testing set) для оценки точности интеллекта.

Теперь, когда вы создаете и настраиваете свой интеллект, вы не смотрите на оценочный набор, даже на самый маленький. Вы настраиваете все, что хотите, на настроечном наборе. Когда вы думаете, что закончили работу, вы оцениваете интеллект на оценочном наборе, чтобы получить объективную оценку результата.

Независимость на практике

Данные, используемые для оценки интеллекта, должны быть полностью отделены от данных, используемых для создания интеллекта. Да, я помню, что с этого предложения начинался предыдущий раздел. Просто это очень важно.

Одно из ключевых предположений машинного обучения состоит в том, что любой произвольно взятый фрагмент данных не зависит от других. То есть достаточно взять любые два фрагмента данных (два контекста, каждый со своими собственными результатами). Пока вы создаете интеллект на одном фрагменте и оцениваете его на другом, вы не можете обманывать – эти две части данных независимы. На практике это не всегда так. Подумайте, что из нижеперечисленного является более или менее независимым:

- пара взаимодействий от разных пользователей по сравнению с парой взаимодействий от одного и того же пользователя;
- две страницы с разных сайтов по сравнению с двумя страницами с одного и того же сайта;
- две книги разных авторов по сравнению с двумя книгами одного и того же автора;
- два изображения разных коров по сравнению с двумя изображениями одной и той же коровы.

Очевидно, что некоторые из этих фрагментов данных не столь независимы, как другие, и использование данных, имеющих скрытые связи, для создания и оценки интеллекта приведет к неточным оценкам. Случайное разбиение данных на обучающие и тестовые наборы не всегда работает на практике.

Два подхода к достижению реальной независимости данных заключаются в *разделении данных по времени* (partition by time) или *по принадлежности* (partition by identity).

Разделение данных по времени. Зарезервируйте самые последние несколько дней телеметрии для оценки и используйте данные предыдущих дней, недель или месяцев для создания интеллекта. Например, если сегодня 5 сентября 2019 года, создатель интеллекта может зарезервировать данные со 2, 3 и 4 сентября для тестирования и использовать все данные с 1 сентября 2019 года и ранее для создания интеллекта.

Если проблема очень стабильна (то есть не имеет изменяющегося во времени компонента), такой подход может оказаться весьма эффективным. Если у проблемы есть изменяющийся со временем компонент, недавняя телеметрия будет полезнее, чем старая, потому что она точнее представляет текущее состояние проблемы. В этих случаях вам нужно будет найти компромисс, сколько драгоценных свежих и наиболее актуальных данных использовать для оценки (в сравнении с обучением) и как далеко назад во времени выбирать учебные данные.

Разделение данных по принадлежности. Убедитесь, что все взаимодействия с одним и тем же источником данных ограничены одним набором данных. Например:

- все взаимодействия с конкретным пользователем используются или для создания интеллекта, или для его оценки;
- все взаимодействия с одним и тем же сайтом используются или для создания интеллекта, или для его оценки;
- все показания датчиков из одного и того же дома используются или для создания интеллекта, или для его оценки;
- все события поджаривания хлеба в одном и том же тостере используются или для создания интеллекта, или для его оценки.

При выборе данных для оценки в большинстве интеллектуальных систем данные разделяются по времени и, как минимум, по одному критерию принадлежности.

Оценка для подгрупп населения

Иногда очень важно, чтобы интеллект не совершал систематические ошибки для критических групп населения (пол, возраст, этническая принадлежность, типы контента, местоположение и т. д.).

Например, представьте систему распознавания речи, которая должна хорошо работать для всех носителей английского языка. Со временем некоторые пользователи начинают жаловаться, что система плохо работает. В результате расследования вы обнаруживаете, что основные проблемы сосредоточены на Гавайях – пиджин сильно трудно слушать, ага, братан?

Хмм, да уж…

Проблема может оказаться настолько серьезной, что продукт не будет продаваться на Гавайях, а это крупный рынок. Надо что-то делать!

Чтобы решить проблему, ее надо измерить (помните, сначала проверка). Следовательно, нам необходимо обновить процедуру оценки, чтобы измерить

точность интеллекта конкретно для проблемной подгруппы. Наша система распознавания речи оценивает потенциальный интеллект двумя способами:

- сначала для всех пользователей;
- затем только для пользователей, которые говорят по-английски с пиджин-гавайским акцентом.

Например, процедура оценки может обнаружить, что точность составляет 95 % в целом, но лишь 75 % для представителей пиджин-популяции. И это довольно большое расхождение.

Оценка точности по подгруппам населения может вызвать затруднения. Во-первых, надо как-то определить, *относится ли взаимодействие к заданной группе населения*. Телеметрия продолжает поступать, включая контекст и результаты. Но теперь вам нужна дополнительная информация – следует знать, относится ли контекст (например, аудиоклип в телеметрии) к пиджин-гавайскому произношению. Некоторые подходы к сортировке по подгруппам включают в себя:

1) **ручную маркировку взаимодействий**. Изучите контексты вручную (слушая аудиоклипы) и найдите несколько тысяч членов целевой группы населения. Такая сортировка может дорого стоить и трудна для повторения, но в целом подход работает. Результирующий набор может храниться и применяться в течение длительного времени (если только проблема не меняется очень быстро);
2) **идентификацию сущностей в подгруппе**. Например, можно помечать пользователя как «говорящий на пиджине» или нет. Каждое взаимодействие с одним из помеченных пользователей можно использовать для оценки интеллекта. Этот подход может быть очень эффективным, когда контекст содержит признак идентичности (например, идентификатор пользователя). Но такой признак не всегда доступен;
3) **использование косвенного признака принадлежности к группе**, например местоположение. Каждый, кто находится на Гавайях, помечается как часть местного населения независимо от того, говорит ли он на пиджин-гавайском диалекте. Это не идеальное решение, но иногда оно работает достаточно хорошо, а иногда вообще получается само по себе, экономя много денег и времени.

Во-вторых, оценка точности интеллекта для подгрупп населения требует *наличия достаточного количества оценочных данных* для каждой подгруппы. Если популяция мала, то случайная выборка оценочных данных может содержать лишь несколько примеров. Можно предложить два способа борьбы с нехваткой оценочных данных:

1) **использование больших оценочных наборов**. Отложите такое количество оценочных данных, чтобы наименьшая важная подгруппа имела достаточное представление для оценки (более подробные сведения о приемлемом количестве данных приведены в следующем разделе);

2) **укрупнение выборок малой популяции для оценки**. Настройте вашу систему так, чтобы представители малой популяции чаще появлялись в телеметрии и с большей вероятностью использовались для оценки, а не для создания интеллекта. При этом вы должны обязательно исправлять перекос вычисления результатов оценки. Например, если пользователи с Гавайских островов отбираются с помощью телеметрии в два раза чаще остальных, то каждое взаимодействие с пользователями на Гавайях приобретает вдвое меньший вес при оценке *общей* точности по сравнению с другими пользователями.

Приемлемый объем данных

Итак, сколько данных вам нужно, чтобы оценить интеллект? Это зависит от обстоятельств.

Напомним, что статистика может выражать, насколько точен ответ, например: точность моего интеллекта составляет 92 % плюс-минус 4 % (то есть где-то между 88 % и 96%).

Поэтому объем оценочных данных зависит от того, насколько вы должны быть уверены в оценке.

Допустим, что ваши данные достаточно независимы, проблема не меняется слишком быстро и вы не пытаетесь оптимизировать последние 0,1 % из очень сложной проблемы (например, распознавания речи). В таком случае:

- нескольких десятков выборок данных слишком мало для оценки интеллекта;
- сотни выборок данных – хороший объем для проверки работоспособности, но на самом деле этого недостаточно для оценки;
- тысячи выборок данных – это, вероятно, подходящий объем для любых целей;
- десятков тысяч выборок данных, пожалуй, будет многовато, но не чересчур.

Отправную точку для выбора количества оценочных данных можно определить следующим набором требований:

1) убедитесь, что у вас есть несколько тысяч свежих выборок, зарезервированных для оценки;
2) убедитесь, что у вас есть несколько сотен последних выборок для каждой важной подгруппы;
3) используйте остальные (достаточно свежие) данные для создания интеллекта;
4) если позволяет объем данных, то просто зарезервируйте около 10 % данных для оценки интеллекта, а остальные данные используйте для разработки.

Но все равно вам придется развивать собственную интуицию выбора данных (или использовать статистику, если у вас математический стиль мышления).

Сравнение интеллектов

Теперь вы знаете некоторые метрики для оценки производительности интеллекта и умеете извлекать эти метрики из данных. Но на каком основании можно сказать, что один интеллект лучше, чем другой? Допустим, вы пытаетесь определить, является ли книга романом, и у вас есть два интеллекта:

- первый интеллект имеет точность 80 % при отзыве 20 %;
- второй интеллект имеет точность 50 % при отзыве 40 %.

Какой из них лучше? Это зависит от пользовательского опыта и более общих целей интеллектуальной системы. Например, для заядлого читателя романов предпочтителен более высокий отзыв (потому что пользователь не пропустит ни одного поцелуя).

Рабочие точки

Одним из инструментов, помогающих оценить интеллект, является выбор *рабочей точки* (operating point). Это *точка наилучшего соответствия* (precision point) или *точка отзыва* (recall point), которая хорошо сочетается с вашим интеллектуальным опытом. Каждый интеллект должен попадать в рабочую точку, а предпочтение отдают интеллекту, который работает в этой точке лучше всего. Например:

- в детекторе смешных веб-страниц рабочая точка может соответствовать точности 95 %. Интеллект должен быть настроен так, чтобы давать максимально возможный отзыв с точностью 95 %;
- в системе фильтрации спама рабочая точка должна соответствовать точности 99 % (система удаляет электронную почту пользователей и поэтому должна быть абсолютно уверена в своих действиях). Все варианты интеллекта должны стремиться помечать как можно больше спама, сохраняя точность на уровне 99 % или выше;
- в системе интеллектуального дверного звонка рабочая точка может соответствовать отзыву на уровне 80 %. Все варианты интеллекта должны быть настроены так, чтобы они отмечали 80 % случаев, когда человек подходит к двери, и боролись за снижение ложных срабатываний в соответствии с этим ограничением отзыва.

Наличие рабочей точки уменьшает количество переменных, выбирая лишь один из типов ошибок, которые может сделать интеллект, и определяет целевой показатель. Допустим, нам нужен интеллект с точностью не менее 90 %, чтобы соответствовать опыту, – теперь, уважаемый создатель интеллекта, отправляйтесь работать и обеспечьте нам максимальный отзыв при заданной точности.

Интеллекты проще всего сравнивать в рабочей точке. Лучше тот вариант, который более точен при заданном отзыве или дает более высокий отзыв при заданной точности цели.

Проще некуда.

Кривые

Но иногда рабочие точки меняются. Например, опыт должен стать более жестким, соответственно, рабочая точка должна перейти на более высокую точность, чтобы соответствовать изменению опыта. Следовательно, имеет смысл оценить интеллект на некотором диапазоне рабочих точек. Например, изменяя порог преобразования вероятности в классификацию, мы можем получить следующие пары значений:

- точность 91 % при отзыве 40 %;
- точность 87 % при отзыве 45 %;
- точность 85 % при отзыве 50 %;
- точность 80 % при отзыве 55 %.

И так далее. Заметим, что таким способом можно оценить любой интеллект, дающий на выходе прогноз вероятности или рейтинг (включая самые разные методы машинного обучения).

Многие эвристические интеллекты и технологии машинного обучения «только для классификации» не оперируют понятиями рейтинга или вероятности, и поэтому их можно оценивать лишь в одной рабочей точке, а не по кривой.

Интеллекты, реализующие компромисс между разными типами ошибок, можно сравнивать по нескольким разным рабочим точкам, например:

- при точности 91 % одна модель дает отзыв 40 %, а другая – 47 %;
- при точности 87 % одна модель дает отзыв 45 %, а другая – 48 %;
- при точности 85 % одна модель дает отзыв 50 %, а другая – 49 %.

Модель может быть лучше при одних компромиссах и хуже при других. Например, одна модель будет лучше, когда вам нужна высокая точность, а другая модель (на основе совершенно иных подходов) будет лучше, когда вам нужен высокий отзыв.

Иногда полезно визуализировать наборы компромиссов, которые может реализовать интеллект. *Кривая наилучшего отзыва* (precision-recall curve – PR-кривая) – это график всех возможных компромиссов, которые может реализовать модель. На оси абсцисс отложены значения отзыва от 0 % до 100 %, а на оси ординат – точность, которую модель обеспечивает при заданном отзыве.

Построив на одном графике PR-кривые для двух моделей, легко увидеть, какая из них лучше в различных областях рабочих точек (рис. 19.1).

Другая схожая концепция называется *кривой рабочих характеристик приемника* (receiver operating characteristic curve – ROC). Для построения кривой ROC откладывают уровень ложноположительных решений по оси X и уровень истинно положительных решений по оси Y.

Субъективные оценки

Иногда интеллект замечательно выглядит при числовой оценке, но на деле, мягко говоря, не столь хорош. Это может случиться, если:

- вы используете критерий оценки, который не соответствует фактической цели;
- у вас недостаточная связь между опытом и интеллектом;
- существует важная популяция, которую вы еще не выявили,
- и т. д.

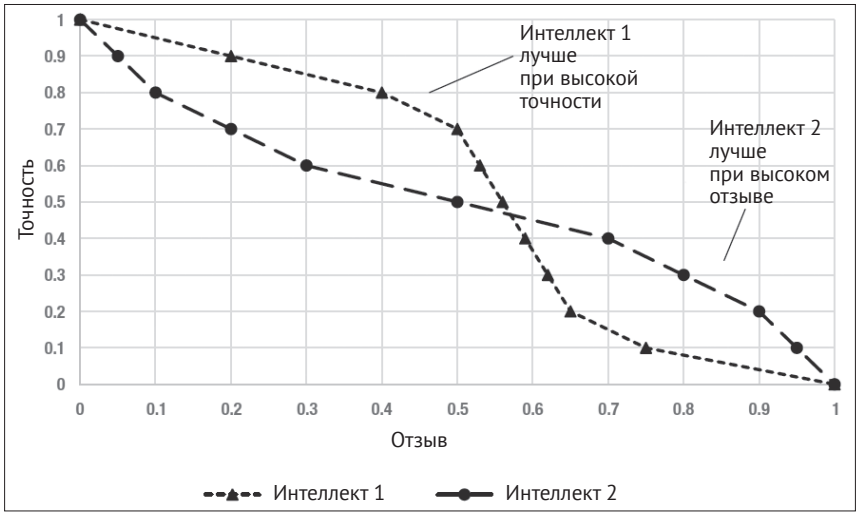

Рис. 19.1 ❖ Сравнение двух моделей по PR-кривым на графике рабочих точек

В подобной ситуации полезно заново просмотреть данные, сделать несколько шагов назад и просто подумать. Станьте детективом данных, примите точку зрения пользователя и задумайтесь о том, что ваш интеллект создает для него. Некоторые подходы к субъективной оценке таковы:

- изучение ошибок;
- переосмысление опыта пользователя;
- предсказание худшей ситуации.

Мы обсудим эти подходы по очереди.

Изучение ошибок

Статистика (например, та, которая используется для оценки качества модели по точности и отзыву) по-своему хороша – она точна и превращает горы всякой всячины в простые маленькие числа, которые могут расти или убывать. Она имеет решающее значение для создания интеллекта, но в то же время иногда заслоняет собой проблемы. Время от времени вам нужно изучать данные.

Один из полезных приемов – взять случайную выборку из 100 контекстов, где интеллект допустил ошибки, и взглянуть на них. Глядя на ошибки, подумайте:

- сколько ошибок будет трудно исправить пользователям?
- сколько ошибок будут иметь значение для пользователя (по сравнению с абсолютной нелепицей)?
- заметна ли какая-то закономерность ошибок? Есть ли у них общие свойства? Может быть, существует некая связь с популяцией, которая в будущем породит большие проблемы?
- указывает ли что-нибудь на ошибки в реализации интеллекта или процесса оценки?
- можно ли как-то улучшить интеллект, чтобы он перестал совершать ошибки? Какую новую информацию добавить в контекст? Может быть, новый тип признаков?

В дополнение к изучению ошибок на случайной выборке полезно обратить внимание на ситуации, когда интеллект был наиболее уверен в ответе, но ошибался (ложноположительный результат, когда модель на 100 % была уверена в положительном ответе, или ложноотрицательный результат, когда модель утверждала, что вероятность положительного ответа составляет 0 %). Изучение ошибок такого рода часто помогает выявить ошибки в реализации или обучении интеллекта.

Переосмысление опыта пользователя

Глядя на ошибки, представьте опыт пользователя. Мысленно визуализируйте поведение пользователей в контексте. Поставьте себя на их место, когда они столкнутся с ошибкой. Что они подумают? Какова вероятность того, что они заметят проблему? Во что обойдется пропущенная ошибка? Что им нужно сделать, чтобы исправить обнаруженную ошибку?

Это подходящее время, чтобы подумать о *совокупном опыте* (aggregate experience), который получат пользователи, например:

- сколько ошибок они увидят?
- сколько положительных взаимодействий они совершат между ошибками?
- как бы они описали ошибки, совершаемые системой, если бы захотели рассказать о них друг другу?

Если вы поставите себя на место своих пользователей, то вам будет проще понять, достаточно ли хорош интеллект, и придумать идеи для улучшения интеллектуального опыта.

Предсказание худшей ситуации

Представьте самое худшее, что может случиться с вашей системой. Какие типы ошибок, допущенных системой, действительно навредят пользователям или вашему бизнесу? Например:

- система продвижения любовных романов, которая классифицирует 100 % книг конкретного писателя-романиста как не романы. Этот писатель теряет рекламу и продажи, однако не может понять, кто виноват

и что делать, поэтому впадает в нищету. Его дети не получают подарков на Рождество…;

- все веб-страницы некой юридической фирмы классифицируются как смешные. Они несмешные, но люди все равно насмехаются над ними (потому что им говорят, что страницы смешные). Никто не хочет нанимать юридическую фирму, полную клоунов. Юристы озлоблены и судятся с вашей фирмой, им нечего терять, зато у них полно свободного времени…;
- датчик температуры пеллетного гриля выходит из строя. Из-за этого интеллект всегда думает, что гриль недостаточно разогрет. Он сбрасывает в жаровню гранулу за гранулой, разжигая огромный пылающий костер, но продолжает думать, что надо больше огня…

Это выглядит как глупые фантазии, но все дело в том, что они случаются. Предскажите самые плохие ситуации, прежде чем пользователи испытают их на своей шкуре, и воспользуйтесь своими находками, чтобы улучшить интеллект или доработать остальную часть системы (опыт и реализацию).

Итог главы

Оценка интеллекта – это творческая работа.

Точность интеллекта состоит из трех основных компонентов: обобщение новых ситуаций, наличие приемлемых ошибок и равномерное распределение ошибок среди разных пользователей или контекстов:

- интеллект может быть хорош в контекстах, о которых он знает, но терпеть неудачу в контекстах, с которыми раньше не сталкивался;
- интеллект может делать ошибки самых разных типов (включая ложноположительные и ложноотрицательные);
- интеллект может совершать как случайные ошибки, так и ошибки, сосредоточенные на конкретных пользователях и контекстах, – сфокусированные ошибки могут вызывать проблемы.

Существуют специальные методы оценки классификации, регрессий, вероятностей и ранжирования. В этой главе представлены как узкие, так и концептуальные понятия, но при необходимости вы можете найти дополнительную информацию.

Интеллект можно оценить с помощью набора данных. Некоторые данные должны храниться отдельно и использоваться исключительно для оценки интеллекта. Эти данные должны быть полностью независимы от данных, используемых для создания интеллекта (они должны поступать от разных пользователей, в разные периоды времени и т. д.).

Рабочая точка помогает сосредоточить внимание интеллекта на ошибках определенного типа, чтобы добиться успеха в масштабе системы. Кривая наилучшего отзыва – это способ понять и визуализировать работу интеллекта во всем диапазоне рабочих точек.

Важно исследовать ошибки, которые совершает ваш интеллект. Постарайтесь понять, что происходит, а также примите точку зрения вашего пользователя и представьте худший результат, который он может получить.

Темы для размышлений

Прочитав эту главу, вы сможете:

- пояснить, что означает точность интеллекта;
- оценить качество интеллекта по широкому набору практических критериев;
- создать целевые критерии качества интеллекта и двигаться к этим целям;
- принять точку зрения пользователя и увидеть ошибки интеллекта его глазами.

Постарайтесь выполнить следующие задания:

- опишите три ситуации, когда для интеллектуальной системы потребуется интеллект с очень высоким уровнем отзыва;
- для каждого из этих трех случаев опишите изменение целей системы, когда вместо отзыва потребуется высокая точность;
- выберите одну из интеллектуальных систем, упомянутых в этой главе, и опишите потенциальную группу населения (не упомянутую в главе), для которой могут возникнуть ошибки;
- рассмотрите систему классификации книг по жанрам. Представьте, что вы собираетесь изучить 100 ошибок, которые совершает система. Опишите два разных критерия, по которым можно разделить ошибки, чтобы легче было понять, что происходит. (Например, ошибка для короткой книги и ошибка для длинной книги… Ой! Теперь вы не можете использовать длину книги в своем ответе. Извините.)

Глава 20

Машинное обучение интеллекта

Машинное обучение (machine learning) является мощным подходом при создании интеллекта для больших, сложных, открытых и меняющихся со временем задач. Обучение работает, показывая компьютеру множество примеров контекстов и желаемых результатов. На основе этих примеров компьютер вырабатывает модели. Затем эти модели применяются для прогнозирования результатов, исходя из заранее незнакомых контекстов.

Машинное обучение может стать мощным фактором силы, позволяющим компьютеру сосредоточиться на вещах, в которых он хорош (например, настраивать каждый нюанс гигантской модели, чтобы она эффективно работала в миллионах контекстов). В свою очередь, люди могут сосредоточиться на том, в чем они сильны (например, выбор модели, способной к хорошему обобщению, и создание систем и интеллектуального опыта, превращающих модель в ценность для клиента и бизнеса). В этой главе дается обзор машинного обучения, описываются этапы процесса и некоторые ключевые вопросы.

Как работает машинное обучение

Алгоритм машинного обучения – это, по сути, процедура поиска, которая подбирает точные модели, используя обучающие данные для повышения точности. В целом алгоритмы машинного обучения работают следующим образом:

- начинают с простой модели;
- пробуют слегка доработанные версии модели, обычно основанные на обучающих данных;
- проверяют, привела ли доработка к улучшению модели;
- повторяют до тех пор, пока их процедура поиска не перестанет находить лучшие модели.

Иногда уточнения принимают форму усложнения модели – например, путем добавления большего количества тестов `if-then-else` в дерево решений. Иногда уточнения принимают форму корректировки параметров модели – например,

обновление весовых коэффициентов в линейной модели.

Например, вспомним, что дерево решений представляет интеллект в виде древовидной структуры. Каждый узел содержит условие `if`, с одним дочерним элементом для случая, когда `if` возвращает значение `true`, и одним дочерним элементом для случая, когда `if` возвращает значение `false`. Вот пример дерева решений для прогнозирования кассовых сборов нового фильма:

```
If <Выиграл Оскар>:
        Is True: If <Имеет 10 суперзвезд в актерском составе>:
                Is True: then $100,000,000.
                Is False: then $1,000,000.
        Is False: If <Показан в ближайшие выходные>:
                Is True: then $50,000,000.
                Is False: then $1,000.
```

Машинное обучение для деревьев решений создает все более сложные деревья, добавляя условия `if`, пока никакие дополнения не перестанут улучшать модель (или пока модель не достигнет некоторого порога сложности). Например, доработка дерева решений для предсказания успеха фильма может выглядеть следующим образом:

```
If <Выиграл Оскар>:
        Is True: If <Имеет 10 суперзвезд в актерском составе>:
                Is True: If <Показан в ближайшие выходные>:
                        Is True: then $500,000,000.
                        Is False: Then $10,000,000.
                Is False: then $1,000,000.
        Is False: If <Показан в ближайшие выходные>:
                Is True: then $50,000,000.
                Is False: then $1,000.
```

Эта модель немного сложнее и, возможно, немного точнее.

Человек может выполнить тот же процесс вручную, но алгоритмы машинного обучения автоматизируют доработку модели и способны учесть миллионы контекстов и произвести сотни тысяч небольших изменений за то время, которое потребовалось бы человеку, чтобы напечатать «Hello, World».

Факторы, которые создатель интеллекта должен контролировать при использовании машинного обучения, включают в себя:

- **конструирование признаков** – как контекст преобразуется в признаки, которые алгоритм машинного обучения может добавить в свою модель. Выбранные вами признаки должны соответствовать концепции цели и содержать достаточно информации для правильных прогнозов;
- **сложность структуры модели** – насколько большой становится модель. Например, количество тестов в дереве решений или количество компонентов в линейной модели;
- **сложность поиска модели** – сколько вариантов модели пытается найти алгоритм машинного обучения. Это фактор иной, чем сложность структуры, но связан с ним. Чем больше вариантов пытается найти алгоритм,

тем выше вероятность найти модель, которая случайно выглядит хорошо на текущих данных, но плохо обобщается на новые контексты;

- **размер данных** – сколько у вас есть обучающих данных. Чем больше хороших и разнообразных данных будет доступно для машинного обучения, тем сложнее и точнее могут стать ваши модели.

Чтобы построить наилучшую из возможных моделей при помощи машинного обучения, вам придется найти оптимальное соотношение перечисленных факторов.

Плюсы и минусы сложности

Одна из ключевых проблем машинного обучения – ограничение сложности моделей, которые производит алгоритм. Эта проблема отражает внутреннее противоречие:

1) вам нужны очень сложные модели для решения сложных, масштабных, открытых задач;
2) чем выше уровень сложности, тем больше у вас шансов получить модель, которая неверно трактует концепцию цели.

Ваша работа в качестве прикладного специалиста по машинному обучению заключается в том, чтобы разрешить это противоречие. Алгоритм машинного обучения может дать сбой по двум причинам:

- *недообучение* модели (underfitting), так как ее реализация может быть слишком упрощенной;
- *переобучение* модели (overfitting) в том смысле, что модель слишком сложно реализована – в некоторых случаях модель срабатывает, но в целом она неправильная.

Давайте рассмотрим пример. Представьте интеллект, который исследует книги и классифицирует их по жанрам – научно-фантастические, технические, любовные романы, триллеры, исторические саги и т. п.

Вы берете тысячу книг, вручную маркируете их жанры и приступаете к созданию интеллекта. Цель состоит в том, чтобы иметь возможность взять новую книгу, которая не является частью исходной тысячи, и точно предсказать ее жанр.

Недообучение

Если подход, который вы выберете, слишком прост, чтобы определить понятие «жанр», у вас получится недообученная модель. Может получиться нечто наподобие следующего:

1) выбираем наиболее характерное слово для каждого жанра:
   - для романа это может быть слово «любовь»;
   - для научной фантастики это может быть слово «лазер»
   - и т. д.;
2) получив новую книгу, проверяем наличие характерных слов и классифицируем книгу по жанру:

– если в книге есть слово «любовь» – это любовный роман;
– если в книге есть слово «лазер» – это научная фантастика.

Эта модель может работать хорошо, если в каждой книге встречается одно и только одно из характерных слов. Но в большинстве книг много слов. Что мы должны делать с книгой, в которой встречаются слова «любовь» и «лазер»? Это роман или фантастика? Мы можем совершить ошибку независимо от выбора, потому что некоторые научно-фантастические книги говорят о любви, и в некоторых любовных романах упоминают лазеры.

Такой подход к моделированию (проверка на одно слово в жанре) слишком прост. Модель *недостаточно обучена*, чтобы усвоить концепцию жанра, поэтому не способна к надежному обобщению.

Переобучение

А теперь представьте другой подход – использование очень и очень сложного дерева решений, по существу кодирующего точную последовательность слов в книге в виде дерева, например:

1) если первое слово книги – «Когда»;
2) и второе слово книги – «эта»;
3) и третье слово книги – «планета»
4) и т. д., в точном соответствии с каждым словом в книге…
5) тогда это жанр «научная фантастика».

Это очень сложная и высокоточная модель на основе данных, использованных для ее создания, но она вообще не сможет обобщать новые книги.

Разумеется, точное запоминание каждого слова в книге выглядит глупо, но это лишь пример. Основная трудность заключается в том, что машинное обучение почти всегда моделирует проблемы способами, которые не соответствуют реальной сути явления. Из-за этого модель будет частично работать на *известных данных*, но не сможет работать в *новых ситуациях*. Это и есть переобучение, избежать которого – одна из ключевых задач при создании эффективного интеллекта.

Если интеллект недооценивает проблему или чересчур усложняет решение, он не сможет надежно работать с новыми данными – то есть не будет обобщать.

Поиск разумного компромисса

Оптимальная модель для определения книжного жанра обитает где-то между двух крайностей. Возможно, она проверит наличие 50 характерных слов, которые наиболее тесно связаны с каждым жанром:

- для научной фантастики можно использовать слова «лазер», «деформация», «звезда», «луч» и т. д.;
- для любовного романа можно использовать… другие слова…

Затем вы можете применить дерево решений, линейную модель или нейронную сеть, чтобы автоматически определить, насколько важно каждое слово

для каждого жанра и как их объединять с весовыми коэффициентами, чтобы точно предсказать жанр.

Пожалуй, этот подход лучше двух предыдущих. Вы понимаете это, я понимаю это. А машины? Возможно, нет.

Подход с десятками характерных слов остается ущербным. Он по-прежнему не дотягивает до понимания концепции жанра, поскольку не учитывает контекст слов, предложения, в которых они использованы, идеи, которые они описывают.

Но это нормально. Суть машинного обучения не в том, чтобы найти «правильное» представление проблемы, а в том, чтобы создать модель, которая дает наилучшее обобщение. Вам придется осознанно выбирать, где и как недообучить или переобучить модель для достижения этой цели.

Конструирование признаков

Контексты в интеллектуальной системе содержат все данные, относящиеся к взаимодействию. Они включают в себя информацию о пользователе, историю взаимодействий, интересующий пользователя контент и многое другое. *Конструирование признаков* – это процесс превращения контекста в компьютерное представление в нужном формате, с которым может работать машинное обучение. При конструировании признаков мы должны обеспечить совместимость информации с процедурой поиска, применяемой для обучения модели.

В данной части раздела говорится о способах работы с контекстами, включая следующие подходы:

- преобразование данных в удобную форму;
- содействие модели в использовании данных;
- нормализация значений признаков;
- выявление скрытой информации;
- расширение контекста;
- устранение излишних признаков.

Преобразование данных в удобную форму

В общем случае алгоритмы машинного обучения принимают *обучающий набор* в качестве входных данных. Обучающий набор состоит из *обучающих примеров*. Каждый обучающий пример содержит набор переменных, которые описывают контекст. Эти переменные называются *признаками* (features). Каждый обучающий пример также содержит *переменную результата* (outcome variable), которая отражает правильный ответ для контекста. Переменную результата часто называют *меткой* (label).

Напомню, что обычно результат относится к одному из следующих типов: категория из набора вариантов (для классификации), число (для регрессии) или вероятность (может быть число от 0 до 1, которое представляет вероятности от 0 до 100 %).

Обучающий пример можно рассматривать как строку в электронной таблице. Один столбец содержит метку; остальные столбцы содержат *значения признака* (feature values). Каждый обучающий пример в обучающем наборе должен иметь одинаковое количество признаков, хотя некоторые признаки могут отсутствовать – машинное обучение устойчиво к таким вещам.

Например, если вы пытаетесь построить модель для предсказания возраста слушателей по музыке, которую они слушают, у вас могут быть признаки, представляющие пол, цвет глаз и количество песен группы «Journey», которые они прослушали на прошлой неделе (табл. 20.1). Одни пользователи слушали, скажем, 7 песен, а другие могли прослушать и 14 песен. В обучающих примерах для этих пользователей столбец-признак «Количество песен» будет заполнен правильными значениями (7, или 14, или какое-то еще). Но, возможно, ваш папа не скажет вам, сколько песен группы «Journey» он слушал – вдруг он стесняется подобных вещей. Таким образом, в обучающем примере вашего отца остается столбец (признак) для количества песен, но значение не определено (значение –1, или 0, или какой-то другой специальный код в соответствии с принятым у вас форматом).

Таблица 20.1. Набор обучающих примеров (обучающий набор)

Возраст (метка)	Пол	Количество песен	Цвет глаз	...
23	М	7	Светло-карие	...
12	Ж	0	Карие	...
47	М	–1	Голубые	...
...	...	...	...	...

Системы машинного обучения обычно работают с двумя типами признаков:

- *числовые признаки* – целые числа или числа с плавающей точкой (например, количество песен, прослушанных на прошлой неделе);
- *категорийные признаки* – которые представляют собой метки из небольшого набора значений (например, пол человека).

Большинство моделей и алгоритмов машинного обучения поддерживают как числовые, так и категориальные признаки. Но иногда требуется небольшая предварительная обработка. Например, нейронные сети внутри себя работают только с числовыми признаками. Чтобы загрузить категорийные признаки в нейронную сеть, следует преобразовать их в числовые значения. Один из распространенных способов – использовать *прямое унитарное кодирование* (one-hot encoding), которое превращает категорийный признак в N числовых значений, по одному на каждую категорию. Например, чтобы закодировать таким способом категорию «пол», можно выполнить следующие преобразования:

- мужской -> [0, 1];
- женский -> [1, 0].

В табл. 20.2 показан пример такой перекодировки.

Таблица 20.2. Пример перекодировки признака из категорийной формы в числовую

Возраст (метка)	Пол мужской	Пол женский
23	0	1
12	1	0

Содействие модели в использовании данных

Конструирование признаков – это искусство преобразования контекста в признаки, которые лучше всего подходят для вашей задачи и структуры вашей модели. Например, вернемся к модели для прогнозирования возраста по песням, которые слушает человек. Один из подходов – создать отдельный признак для каждого исполнителя, который когда-либо записывал песню. Каждый обучающий пример будет содержать десятки тысяч столбцов – по одному на каждого исполнителя, – указывающих, сколько песен этого исполнителя прослушал пользователь.

Такое представление содержит огромный объем информации, помогающий модели избежать недообучения. Однако это очень сложный подход. Он требует, чтобы обучающий алгоритм учитывал множество признаков при выработке решения, и может спровоцировать переобучение. Может быть, существует иной подход?

Вполне возможно, в зависимости от ваших инструментов и объема данных. Единственный способ узнать это – попробовать разные методы, построить разные модели, оценить их и посмотреть, что получилось.

Для упомянутого примера с песнями можно предложить следующие методы:

- **разбейте песни по десятилетиям**. Создайте по одному признаку на десятилетие, где значение признака – это количество песен десятилетия, которые слушал человек;
- **отделите популярные песни от непопулярных**. Возможно, люди, выросшие в течение определенного десятилетия, склонны к более широкому восприятию музыки, и наблюдение за прослушиванием непопулярных песен поможет выявить это явление. Следовательно, каждое десятилетие должно иметь два признака: один для прослушиваний популярных песен этого десятилетия и другой для прослушиваний непопулярных песен этого десятилетия;
- **разбейте песни по жанрам.** Отдельный признак на каждый жанр, где значение – это количество прослушанных песен данного жанра;
- **комбинация жанра, десятилетия и популярности**, где вы используете все три предыдущих метода одновременно.

Приступив к работе, вы будете использовать все возможные методы решения задачи, включая извлечение признаков из контекста (например, преобразование списков песен, прослушиваемых людьми, в паттерны прослушивания), а также частичное отбрасывание данных, чтобы алгоритму машинного обучения достались только самые важные признаки.

Поймите правильно суть ваших действий. Если вам удалось обнаружить, что существует релевантное понятие, называемое «жанр», и вы можете раскрыть это понятие через формальные признаки – значит, алгоритм машинного обучения сможет использовать эти данные. Конечно, это требует усилий с вашей стороны, но зато помогает закодировать ваши предыдущие знания о мире и создать модель, которая лучше отражает проблемную область.

Сравните этот подход с предоставлением необработанных данных для процесса машинного обучения, когда алгоритм вынужден разбираться самостоятельно. Затраченные вами усилия на изучение и подготовку данных могут с лихвой окупиться, если данных чрезвычайно много и надо избежать переобучения.

Различные алгоритмы проявляют свои преимущества при разных подходах. Изучите свои инструменты, натренируйте интуицию и добейтесь, чтобы ваша система оценивания интеллекта была проста в использовании.

Нормализация значений признаков

Иногда числовые признаки имеют очень широкий разброс значений. Например, возраст где-то между нулем и сотней лет, или доход где-то между нулем и сотней миллионов долларов. *Нормализация* (normalizing) – это процесс приведения числовых признаков к более сопоставимому виду. Вместо того чтобы говорить, что человеку 45 лет и он имеет доход 75 000 долларов, можно сказать, что человек на 5 % старше среднего возраста и имеет доход на 70 % выше среднего дохода.

Один из стандартных способов нормализации состоит в том, чтобы обработать все числовые признаки и заменить их нормализованными значениями:

1) вычтите среднее значение (сдвиньте среднее значение к нулю);
2) разделите на стандартное отклонение переменной.

Например, вычислите новые значения признаков по формуле:

$$\text{Normalized\_Age} = \frac{\text{Age} - \text{mean\_of\_all\_ages}}{\text{Standard\_Deviation\_of\_all\_ages}}.$$

Алгоритм обучения, безусловно, может работать с ненормализованными данными. Но выполнение предварительной обработки устраняет избыточную сложность и делает задачу моделирования немного проще, чтобы алгоритм мог сосредоточиться на других вещах. Иногда это помогает.

Выявление скрытой информации

Некоторые признаки бесполезны по отдельности. Чтобы стать полезными, они должны объединиться с другими признаками. Или, что бывает чаще, некоторые признаки полезны и по отдельности, но становятся гораздо полезнее в сочетании с другими признаками.

Например, если вы создаете модель для предсказания стоимости доставки ящиков, у вас могут быть признаки высоты, ширины и глубины ящика. Безусловно, это полезные признаки. Но еще полезнее знать объем коробки.

Некоторые алгоритмы машинного обучения могут обнаруживать полезные взаимосвязи самостоятельно, а некоторые нет.

Вы можете создавать *составные признаки* (composite features) на основе вашей интуиции и понимания проблемы. Или можете попробовать автоматически создать множество новых признаков, комбинируя начальные признаки, которые вы раздобыли.

Расширение контекста

Внешний интеллект или службу можно использовать для *расширения контекста* (context expanding) – добавления информации, которая отсутствует в контексте в явном виде. Например, вы можете расширить текущий контекст следующими способами:

- отследить историю личности из другого контекста, например через трафик веб-сервера;
- получить температуру окружающей среды через веб-службу в зависимости от местоположения;
- запустить программу анализа эмоциональной окраски текста и добавить прогноз настроения в качестве признака.

Отметим, что подобные признаки можно использовать только в том случае, если вы можете абсолютно одинаково извлекать их как в среде разработки, так и в среде выполнения интеллекта.

В следующей главе будет рассказано о других подходах к организации интеллекта, помогающих продуктивно использовать внешние источники данных или унаследованный интеллект для расширения текущего контекста.

Устранение лишних признаков

Еще один подход к работе с контекстом заключается в удалении лишних признаков.

ЧТО!?!

Вы только что проделали кропотливую работу по извлечению признаков из контекста, а теперь вам предлагают удалить некоторые из них? Серьезно?

Вполне возможно, что вы извлекли бестолковые признаки (извините). Они могут усложнить процесс обучения, не принося никакой пользы взамен. Например, использование цвета глаз человека для предсказания его возраста. Конечно, у каждого человека есть цвет глаз, и этот признак присутствует в контексте. Но имеет ли он отношение к предсказанию возраста человека? На самом деле нет.

Использование нерелевантных признаков уменьшает способность вашей модели к обобщению, потому что иногда нерелевантный признак случайно кажется уместным и обманом заставляет ваш алгоритм машинного обучения включить его в модель.

Или, возможно, вы извлекли гораздо больше релевантных признаков, чем способна обработать ваша модель. Все признаки по отдельности могут

быть замечательными, но их избыток спровоцирует переобучение и ухудшит обобщение.

Так что на практике вам следует:

1) удалить признаки, которые не связаны с результатом;
2) из оставшегося набора отобрать столько лучших признаков, сколько поддерживают структура модели и процедура поиска.

Этот процесс называется *отбором признаков* (feature selection). Существуют разные методы отбора признаков. Вот несколько самых простых:

1) измерьте информационную взаимосвязь между каждым признаком и выходом, после чего удалите признаки с наименьшим влиянием на выход;
2) обучите модели с каждым признаком и без него, оцените каждую модель, а затем удалите признаки, которые не помогают повысить точность обобщения;
3) используйте свою интуицию и понимание проблемы, чтобы удалить признаки, которые не имеют смысла.

Эти методы могут пригодиться при отладке ваших инструментов извлечения признаков. Создайте новый признак. Добавьте его к существующему набору признаков и обучите новую модель. Если модель не стала лучше, значит, вы что-то сделали неправильно. Понаблюдайте за моделью, а затем остановитесь и подумайте. Возможно, это поможет вам понять, что происходит на самом деле.

Моделирование

Моделирование (modeling) – это процесс использования алгоритмов машинного обучения для поиска качественных моделей. Есть много способов, которыми создатель интеллекта может помочь этому процессу, например:

- выбрать подходящие признаки;
- выбрать алгоритмы машинного обучения и представления моделей;
- выбрать обучающие данные;
- управлять процессом создания модели.

Но в любом случае цель моделирования состоит в том, чтобы получить модель, которая лучше обобщает, имеет лучший профиль ошибок и создает наибольшую ценность для пользователей. В данном разделе говорится о том, как это сделать, в том числе о распространенных способах настройки процесса моделирования.

Параметры сложности

Каждый алгоритм машинного обучения имеет параметры, позволяющие создателям интеллекта ограничивать вносимую сложность.

Некоторые алгоритмы имеют параметры, которые *ограничивают размер модели, создаваемой алгоритмом*. Иногда при поиске модели происходит усложнение ее структуры, например добавление узлов в дерево решений или добавление деревьев в *случайный лес* (random forest). Алгоритмы данного типа

часто имеют параметры, накладывающие жесткие ограничения на количество структурных элементов – количество узлов в дереве или число деревьев в случайном лесу. Многие алгоритмы также имеют параметр минимального *прирашения обучающего набора* (trainin-set gain), которое они должны увидеть, чтобы продолжить поиск модели.

Если наблюдается переобучение, следует ограничить сложность; если наблюдается недообучение, можно увеличить сложность.

Алгоритм машинного обучения может иметь параметры для управления *стратегией поиска* (search strategy). Наиболее распространенный подход состоит в том, чтобы делать *жадные шаги* (greedy steps) вдоль *градиента* (gradient), то есть вносить такие локальные изменения, которые больше всего улучшают производительность модели на данных обучения. Другие варианты включают в себя внесение элемента случайности, например случайные перезапуски (в случае если поиск достигает локального максимума) или *предсказание* (lookahead) оптимальной модели.

Алгоритмы также могут иметь параметры, влияющие на размер шагов, предпринимаемых при поиске модели. То есть они находят градиент, а затем им нужно решить, как далеко продвинуть модель вдоль градиента. Меньший размер шага предполагает большую сложность. Некоторые алгоритмы адаптивно регулируют размер шага по мере продолжения поиска – начинают с большого шага, чтобы найти хорошую рабочую область, а затем уменьшают шаг, чтобы уточнить модель.

Если наблюдается переобучение, вы должны использовать более простые стратегии поиска. Если наблюдается недообучение, вы можете попробовать усложнить поиск.

Некоторые алгоритмы также применяют методы прямой оптимизации, например внутреннее представление задачи в виде матрицы с последующим использованием линейной алгебры для поиска ключевых свойств. Эти алгоритмы могут поддерживать различные варианты оптимизации, компенсирующие сложность.

Как правило, вы должны корректировать упомянутые параметры в зависимости от сложности вашей проблемы и объема доступных данных. Очень редко удается завершить успешное машинное обучение без настройки параметров сложности вашего алгоритма.

Выявление переобучения

Напомним, что целью моделирования является построение модели, которая лучше всего обобщает новые параметры. Одна из стратегий выявления момента, когда начинается переобучение, состоит в следующем:

1) создайте на обучающих данных серию моделей с нарастающей сложностью, начиная с примитивных и заканчивая очень сложными. Если признаки и представления моделей выбраны правильно, то более сложные модели должны работать лучше, чем простые – *на обучающих данных*;

2) теперь запустите эти модели на тестовом наборе. Простая пустоголовая модель должна работать относительно плохо. Следующая, немного более сложная модель должна работать несколько лучше. А еще более сложная должна работать еще лучше. И так снова и снова – до определенного момента, когда более сложные модели начнут работать хуже, чем менее сложные. Точка, в которой они начинают работать хуже, – это точка, в которой ваш набор признаков, стратегия поиска модели, представление модели и доступные данные начинают сталкиваться с проблемами (рис. 20.1).

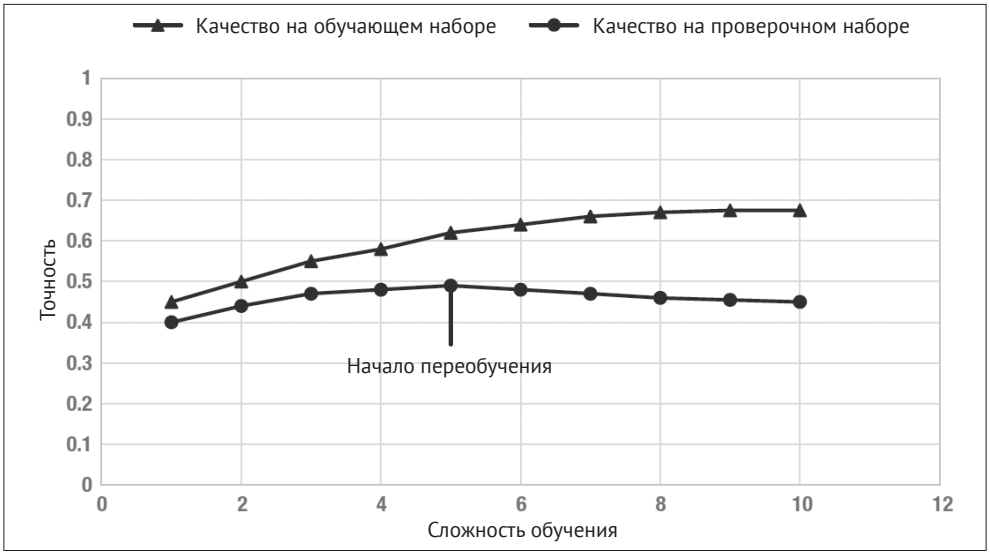

Рис. 20.1 ❖ Пример переобучения

Если возникает переобучение, вам следует:

- подобрать признаки, которые лучше соответствуют проблеме;
- подобрать структуру модели, которая лучше соответствует проблеме;
- сократить продолжительность поиска моделей
- *...или использовать больше данных!*

Увеличение объема обучающих данных – лучший способ избежать переобучения, позволяющий создавать более сложные и точные модели. К счастью, вы проделали большую работу по настройке интеллектуальной системы с продвинутым интеллектуальным опытом и множеством пользователей, которые будут генерировать данные. Именно в такой ситуации – благодаря интеллектуальной системе – интеллект развивается тем быстрее, чем больше пользователей с ним взаимодействуют.

Итог главы

Машинное обучение автоматически производит интеллект из данных. Процесс обучения включает в себя компьютерный поиск моделей, которые хорошо работают с вашими данными на обучающем наборе. Одно из ключевых противоречий машинного обучения – это противоречие между высокой точностью и плохим обобщением по причине переобучения.

Конструирование признаков является важной частью машинного обучения и подразумевает преобразование контекстов в представления, с которыми может работать ваша модель. Но конструирование также подразумевает достаточно глубокое понимание вашего представления, вашей проблемы и вашей стратегии поиска алгоритма машинного обучения, чтобы правильно подготовить данные. Наиболее распространенные методы подготовки данных – нормализация, отбор и комбинирование признаков.

Процесс моделирования включает в себя множество итераций, выбор правильного алгоритма машинного обучения, настройку его параметров и извлечение максимальной пользы из доступных обучающих данных путем аккуратного управления сложностью.

Темы для размышлений

Прочитав эту главу, вы должны:

- сформировать общее понимание машинного обучения, включая признаки, модели, процесс моделирования и ключевое противоречие между точностью и переобучением;
- знать этапы прикладного машинного обучения и понимать в общих чертах, как работает универсальный алгоритм машинного обучения;
- уметь создавать пригодные для использования модели при помощи инструментов машинного обучения и приготовиться углубленно изучать некоторые алгоритмы и методы.

Постарайтесь выполнить следующие задания:

- опишите контекст интеллектуальной системы, которую вы не использовали в предыдущих ответах на задания этой книги;
- создайте десять числовых признаков для контекста;
- создайте пять категориальных признаков для контекста;
- предложите один из способов сделать признаки менее сложными;
- создайте пример модели, которая использует ваши признаки для переобучения – сходит с ума и достигает уровня явного переобучения;
- бонус: разработайте очень простой алгоритм машинного обучения. Как вы представите интеллект пользователю? Какой у вас поиск? Каковы ваши параметры сложности?

Глава 21

Структурирование интеллекта

Для большинства крупных систем создание интеллекта – это командная деятельность. Несколько человек могут *одновременно работать над интеллектом*, строить разные его части или исследовать различные проблемные области. Спустя какое-то время над интеллектом могут работать уже *другие люди, которые унаследовали интеллект* от команды разработчиков, или исправляют интеллект, который нормально работал, но теперь столкнулся с проблемами. Вот несколько примеров *структурирования интеллекта* (organizing intelligence):

- использование машинно-обучаемого интеллекта для большинства задач, но созданного вручную интеллекта для устранения ошибок;
- использование одного машинно-обучаемого интеллекта для пользователей из Франции, а другого – для пользователей из Японии;
- использование выводов эвристического интеллекта в качестве признаков для машинного обучения другого интеллекта.

В этой главе обсуждаются разные способы структурирования и говорится о создании мощного и надежного интеллекта в большом коллективе разработчиков.

Причины структурирования интеллекта

Существует множество причин, по которым вы можете перейти от единого монолитного интеллекта (например, одной модели с машинным обучением) к структурированному интеллекту:

- **совместная работа** – разработка сложного интеллекта является совместной деятельностью. У вас может быть 3, 5 или 15 человек, работающих над интеллектом одной интеллектуальной системы. И если это так, то вам придется найти способы заставить их всех работать дружно и продуктивно, а не бороться за титул единоличного «повелителя интеллекта»;
- **устранение ошибок** – каждый интеллект будет делать ошибки, а исправление одной ошибки часто приводит к появлению большего коли-

чества ошибок в других местах. Кроме того, весьма непросто получить машинно-обучаемый интеллект, не совершающий определенных ошибок, – обычно для этого нужны эксперименты и везение. Структурирование интеллекта способствует быстрому устранению локальных ошибок и поддержке общего интеллекта системы;

- **решение простых задач простыми методами** – иногда несколько простых эвристических правил прекрасно справляются с решением простой части задачи. В таких случаях вы можете обойтись без машинного обучения, которое станет искать уже известное вам решение, или можете разделить задачу и позволить эвристике решить простую часть, в то время как машинное обучение фокусируется на более сложных решениях;
- **использование существующего интеллекта** – в мире существует множество вариантов интеллекта, которые могут оказаться весьма полезными. Возможно, вы захотите подключить к ним свою интеллектуальную систему, чтобы использовать их возможности и заодно помочь им расти еще быстрее.

Свойства правильно структурированного интеллекта

Структурирование интеллекта может оказаться сложной задачей. Если структурирование выполнено неправильно, то сегменты интеллекта будут зависеть от особенностей друг друга. Любое изменение в одном интеллекте будет вызывать непреднамеренные и трудно отслеживаемые изменения в другом интеллекте. Вы можете оказаться в ситуации, когда знаете, что нужно внести некоторые изменения, но это невозможно сделать – по аналогии со спагетти-кодом, вы будете обладать спагетти-интеллектом.

Правильно структурированный интеллект обладает следующими свойствами:

- **достаточная точность** – структурирование не должно слишком сильно снижать потенциал точности. Следует найти хороший компромисс между краткосрочным убытком (в виде снижения точности на ограниченном периоде) и долгосрочной выгодой (в виде более высокой точности на всем сроке службы интеллектуальной системы);
- **простота развития** – каждый желающий должен иметь возможность разобраться в структуре, создать собственный интеллект и поместить его в систему;
- **слабая связанность** – способность одного интеллекта влиять на поведение других интеллектов должна быть сведена к минимуму. Интерфейсы интеллектов должны быть простыми и прозрачными, а интеллекты не должны использовать информацию о внутренней работе друг друга;
- **очевидность** – для каждого результата, выдаваемого пользователям, позволяет точно определить интеллект (или интеллекты), вовлеченный

в решение. Количество вовлеченных интеллектов должно быть минимизировано;

- **измеримость** – позволяет определить, в какой мере каждая часть интеллекта приносит выгоду пользователям;
- **поддержка командной работы** – структура интеллекта должна хорошо стыковаться с командной работой и помогать разработчикам взаимно дополнять друг друга. Нельзя допускать конфликт целей или непродуктивную конкуренцию разработчиков внутри команды.

Способы структурирования интеллекта

В этом разделе обсуждается ряд методов структурирования и создания интеллекта, а также приводится их сопоставление с ключевыми свойствами правильно структурированного интеллекта. Большинство крупных интеллектуальных систем будет использовать несколько этих методов одновременно. На самом деле существует множество разных вариантов – упомянутые ниже методы служат лишь отправной точкой:

- разделение признаков;
- конкурентный поиск моделей;
- распределение ошибок;
- метамодель;
- секвенированная модель;
- разделение по контекстам;
- замещение.

В следующих разделах упомянутые подходы будут описаны и ранжированы в соответствии с тем, насколько они соответствуют критериям правильно структурированного интеллекта. Это субъективная шкала, которая пытается отразить отношение достоинства/недостатки следующим образом:

++	Очень хорошо
+	Лучше среднего
–	Хуже среднего
– –	Проблема, над которой надо поработать

Все рассмотренные методы жизнеспособны и применяются на практике. Но вы должны быть готовы решать проблемы, присущие выбранной вами стратегии структурирования.

Разделение признаков

Один из подходов к созданию структурированного интеллекта заключается в *разделении признаков* (decouple engineering feature), чтобы каждый создатель интеллекта имел свою часть контекста для изучения и превращения в признаки. Например, если вы пытаетесь понять содержание веб-страницы:

- первый создатель интеллекта сосредоточится на содержании страницы, используя стандартные подходы, такие как *набор слов* (bag of words) и *N-граммы* (N-grams), для преобразования слов в признаки;
- второй создатель интеллекта сосредоточится на понимании семантики текста, используя части речи, анализ эмоциональной окраски и т. д.;
- третий будет изучать историю веб-сайта, место размещения, узнавать, кто создатель контента и что еще он создал;
- четвертый будет исследовать изображения на веб-странице и попытаться создать из них признаки;
- пятый будет рассматривать свойства пользователя.

Все члены команды должны иметь возможность добавлять свой код извлечения признаков в процесс моделирования. Они должны уметь настраивать параметры модели, максимально используя преимущества новых решений, а также разворачивать свои решения.

Существуют и некоторые проблемы, связанные с разделением признаков:

- **конфликт при построении модели** – определенная модель или набор параметров может работать лучше с каким-то одним признаком и хуже с другими. Участники должны находить разумные компромиссы на благо общей системы, а не просто выжимать персональную выгоду из доступных наборов признаков;
- **избыточные признаки**: множественный подход к созданию признаков провоцирует использование одной и той же базовой информации из контекста. Полученные признаки иногда очень похожи друг на друга, а у создателей интеллекта может возникнуть конфликт по поводу того, как устранить избыточность;
- **нестабильная значимость признаков**: когда алгоритм машинного обучения получает новый признак, он обычно меняет направление поиска оптимальной модели, что оказывает сильное влияние на значимость других признаков и на типы ошибок, допускаемых моделью. Для добавления нового признака необходимо стратегическое понимание набора признаков и концепции модели, а также доработка кода извлечения признаков, чтобы сохранить модель в равновесии.

Итак, метод разделения признаков обладает следующими свойствами:

- точность: средняя. В этом подходе нет чрезмерного упрощения, а создатели интеллекта могут работать параллельно, чтобы вместе получить прирост точности;
- простота развития: +. Добавление нескольких новых признаков в существующую модель не вызывает особого труда. Не самый простой, но довольно хороший метод;
- взаимозависимость: средняя. Признаки могут зависеть друг от друга, но пока вы активно боретесь с избыточностью признаков, их взаимосвязь вряд ли будет основной проблемой;
- наглядность: средняя. При отладке взаимодействий нет хороших инструментов для определения проблемных признаков, а многие типы мо-

делей делают эту задачу особенно трудной. Иногда вам лишь остается сказать: «Попробуйте удалять признаки по одному и заново обучать модель, пока проблема не исчезнет»;
- измеримость: –. Легко измерить улучшение сразу после добавления признака, но труднее отследить вклад признаков с течением времени (например, когда проблема меняется);
- поддержка командной работы: средняя. Метод хорошо работает, если команда ясно понимает контекст, но в конечном итоге она может столкнуться с конфликтом относительно того, какие признаки должны остаться, а какие нет, особенно если есть ограничения среды выполнения (например, ресурсов процессора или памяти).

Конкурентный поиск моделей

Еще один способ структурировать интеллект – это позволить нескольким разработчикам сделать разные варианты моделей. Например, один разработчик – эксперт по линейным моделям, а другой – мастер нейронных сетей. Пусть они попытаются создать интеллект, используя наиболее привычные инструменты, и победит лучшая модель.

Конкурентный поиск моделей (multiply model search) будет эффективным, если:
- у вас есть разработчики, которые имеют опыт работы с разными моделями;
- вы находитесь на ранней стадии построения вашей интеллектуальной системы и хотите узнать, какие подходы работают лучше всего;
- у вас произошли серьезные изменения в системе (например, изменилась проблема или значительно возросла нагрузка) и вы хотите убедиться, что выбрали правильный подход к моделированию.

Но использование конкурентного поиска нескольких моделей может привести к напрасным трудозатратам и конфликтам в команде, потому что один подход в конечном итоге победит, а другие проиграют.

Свойства конкурентного поиска модели:
- точность: –. Этот подход трудно использовать постоянно, потому что приходится загружать одной задачей несколько разработчиков интеллекта. Его следует применять лишь в критические моменты процесса создания интеллекта, когда назрела необходимость существенных изменений;
- простота развития: –. Чтобы запустить новый интеллект, вы должны победить старый. Это означает, что новые идеи должны быть достаточно проработаны и тщательно оценены перед развертыванием;
- взаимозависимость: средняя. В каждый момент времени работает только один интеллект, поэтому нет особой проблемы взаимозависимости;
- наглядность: средняя. В каждый момент времени работает только один интеллект, поэтому нет особой проблемы с наглядностью;
- измеримость: средняя. Достаточно легко сравнить новый интеллект с предыдущим;

- поддержка командной работы: – –. В конкурентном методе есть победитель и проигравший. Кроме того, часть усилий команды пропадает напрасно – в погоне за моделью, которая никогда не будет работать.

Распределение ошибок

Следующий подход заключается в исправлении проблем с интеллектом по аналогии с ошибками программирования. Ответственность за ошибки можно распределить между разработчиками интеллекта, а они должны выяснить, что нужно сделать для решения проблемы. Например, если у вас возникли проблемы с популяцией – допустим, с детьми, – поручите кому-нибудь выяснить, что добавить в контекст, какие признаки изменить или какие модели перенастроить, чтобы улучшить работу с детьми.

Интеллект всегда будет ошибаться, поэтому работа над ошибками может продолжаться бесконечно.

Одна из ключевых трудностей такого подхода – выяснить, какие ошибки интеллекта являются просто случайными ошибками, а какие – системными проблемами, требующими изменений в процессе создания интеллекта. При использовании этого подхода очень легко увлечься погоней за ненужными ошибками, всех расстроить и ничего не добиться.

На мой взгляд, этот подход следует использовать нечасто и только в начале проекта (когда есть много естественных ошибок) или когда возникла катастрофическая проблема.

Свойства распределения ошибок:

- точность: –. Иногда этот подход полезен для интеллекта (как в случае с проблемой популяции), но можно легко увлечься погоней за ненужными ошибками;
- простота развития: –. Каждый член команды должен обладать навыками поиска и отслеживания ошибок, разрабатывать полезные изменения и внедрять исправления. Кроме того, можно неправильно определить проблемы, заслуживающие внимания;
- взаимозависимость: средняя. Никак не влияет на взаимозависимость интеллекта с другими компонентами;
- наглядность: средняя. Никак не влияет на наглядность интеллекта;
- измеримость: средняя. Никак не влияет на измеримость качества интеллекта;
- поддержка командной работы: –. Такой подход не определяет четкие границы работы. Исправляя одну ошибку, можно легко вызвать другую, причем иногда новая ошибка может проявить себя намного позже. При неправильной реализации этот подход может разрушить рабочую среду.

Метамодель

Подход *метамодели* (meta-model) заключается в том, чтобы рассматривать предсказания различных интеллектов в вашей системе как входные признаки

для *метаинтеллекта* (meta-intelligence, верховный интеллект). Каждый базовый интеллект работает в своем контексте и принимает свое решение, а затем метаинтеллект просматривает все предложенные прогнозы и решает, каким должен быть окончательный результат. Подход метамодели имеет следующие достоинства:

- он очень точен, потому что объединяет все возможные подходы и учитывает, в каких контекстах наиболее успешен каждый подход;
- отличный способ задействовать устаревший интеллект. Например, когда вы разработаете новый интеллект, который лучше, чем ваша первоначальная эвристика, вы можете выбросить свои эвристические правила или использовать их в качестве дополнительного источника признаков нового интеллекта;
- хороший способ заставить нескольких создателей интеллекта работать вместе. Для их творческих экспериментов не будет ограничений. Метамодель будет использовать только результаты, имеющие реальную ценность, и игнорировать бесполезные вещи.

Но метамодели могут стать и кошмаром для руководителя проекта по следующим причинам:

- метаинтеллект и базовые интеллекты тесно взаимосвязаны, и изменение любой части может потребовать переобучения и перенастройки всей системы;
- если какая-либо часть системы выходит из строя (например, одна модель начинает работать плохо), следом за ней может выйти из строя вся система. Иногда бывает очень трудно отследить, где и как возникают проблемы.

Если вы хотите использовать метамодели, вам понадобятся методы ограничения сложности, вносимой взаимозависимыми моделями, например:

- разработка общих подходов к тому, какие модели могут изменяться и когда – допустим, запрет обновления устаревшего интеллекта и его изменение только в случае обнаружения серьезных проблем;
- использование дополнительного оборудования, которое позволяет легко и быстро переучить и перенастроить весь интеллект, составляющий систему.

Подход с использованием метамоделей обладает следующими свойствами:

- точность: ++. Проще говоря, метамодели могут оказаться наиболее точными из перечисленных здесь методов, но дорого обходятся по другим параметрам. Для грубой оценки используйте обычные модели;
- простота развития: –. Для развертывания нового интеллекта вам необходимо переобучить мета-интеллект, который рискует нестабильностью результатов по всем направлениям. Может понадобиться тщательное тестирование;
- взаимозависимость: – –. Изменение любого базового интеллекта обычно требует переподготовки мета-интеллекта. Непреднамеренные изменения в любом из интеллектов (например, изменения в одном из источ-

ников данных, от которых он зависит) могут повлиять на всю систему, вплоть до ее полного разрушения;

- наглядность: – –. Каждый интеллект вносит свой вклад в каждое решение. Когда возникает проблема, бывает чрезвычайно трудно или даже невозможно отследить ее до источника;
- измеримость: –. Вклад нового интеллекта легко измерить, когда он только добавлен в систему, но труднее отследить вклад интеллекта с течением времени (например, когда меняется цель системы);
- поддержка командной работы: средняя. Иногда интеллекты конфликтуют между собой, но их можно создавать по отдельности. Кроме того, могут возникать конфликты из-за доступа к ресурсам системы, особенно когда базовые интеллекты должны работать в среде с ограниченными ресурсами.

Секвенированная модель

Секвенирование модели (model sequencing) – это построение упрощенной версии метамодели, ограниченной до уровня сверхпростой модели. В подходе секвенирования базовые модели выступают в определенной последовательности (sequence), заданной создателем интеллекта. Каждая модель по очереди получает возможность представить свой результат. И первая же модель, которая заявит о наличии ответа с высокой степенью уверенности (проголосует за ответ), побеждает и отправляет свой ответ на выход метамодели.

Этот подход можно использовать для классификации, предварительно определив ответ по умолчанию. Например, если никто не голосует, даем ответ по умолчанию «мужчина», но позволяем каждой модели давать точный рабочий прогноз для ответа «женщина». Иными словами, если какая-либо модель *твердо уверена*, то может дать ответ «женщина», и этот ответ станет возвращаемым значением метамодели. Если ни одна из моделей не уверена в ответе «женщина», то по умолчанию возвращаемым значением будет «мужчина».

Секвенированная модель имеет меньший потенциал точности, чем метамодель, которая объединяет все голоса одновременно, но гораздо проще в разработке и сопровождении.

Подход с использованием секвенированной модели обладает следующими свойствами:

- точность: средняя. Этот подход обменивает потенциальную точность на простоту управления и развития;
- простота развития: +. Создатель интеллекта может поместить новую модель в последовательность (при условии что она обладает достаточно высокой точностью), не затрагивая никакую другую часть системы;
- взаимозависимость: ++. Модели полностью развязаны и объединены лишь простой процедурой, понятной каждому;
- наглядность: ++. Каждое взаимодействие можно проследить до части интеллекта, определившей результат, и принадлежность каждой части интеллекта легко просматривается;

- измеримость: средняя. Легко измерить, сколько позитивных и негативных взаимодействий каждый интеллект дает пользователям. Недостатком подхода является то, что засчитывается лишь первый уверенный ответ, поэтому другие части интеллекта могут и не получить похвалу либо критику, которую заслуживают;
- поддержка командной работы: +. Любой разработчик может легко добавить свой вклад. Потенциальными поводами для конфликтов могут стать последовательность моделей и порог точности. Но телеметрия должна предоставлять достаточно хорошие данные для эмпирических решений, поэтому можно обойтись без слишком жарких споров.

Разделение по контекстам

Разделение по контекстам (partitioning by contexts) – это еще один простой способ организовать несколько интеллектов. Сначала определяют простые правила разбиения контекстов на разделы, а затем применяют отдельный интеллект (или последовательность моделей, метамодель и т. д.) для каждого раздела. Например:

- один интеллект для серверов в США и другой для всех остальных;
- один интеллект для небольших веб-сайтов, другой для больших веб-сайтов и третий для сайтов – агрегаторов контента;
- один интеллект для изображений в градациях серого, другой для цветных изображений;
- один интеллект для новых пользователей, другой для пользователей с долгой историей.

Благодаря возможности использовать разные типы интеллекта для разных частей задачи удается быстрее решать простые проблемы, а также отслеживать ошибки, допущенные в различных разделах. Конечно, алгоритмы машинного обучения способны самостоятельно использовать подобный подход – и, вероятно, даже выбирать более качественные разделы с точки зрения исходной точности, – но разделение вручную отлично подходит для структурирования и оркестровки.

Подход разделения по контекстам обладает следующими свойствами:

- точность: средняя. Данный подход превращает исходную задачу в несколько небольших подзадач. В принципе, он не должен влиять на точность, однако иногда это случается, особенно когда инновации одного раздела не переносятся на другие разделы, где они могли бы помочь;
- простота развития: ++. Разработчик интеллекта может выделить конкретный раздел и настроить интеллект для него, не затрагивая другие разделы;
- взаимозависимость: +. Модели полностью развязаны и объединены понятной процедурой (если разделение не выходит из-под контроля);
- наглядность: +. Каждое взаимодействие можно проследить до части интеллекта, определяющей результат, и каждая часть интеллекта имеет явного владельца;

- измеримость: +. Легко измерить, сколько позитивных и негативных взаимодействий дает пользователям каждый интеллект;
- поддержка командной работы: +. Любой разработчик может легко внести свой вклад. Кроме того, когда один из членов команды решает проблему разделением чего-то, что вызывало проблемы у других моделей, это можно представить как одолжение: «Я рад поработать над этими ошибками вместо вас, чтобы моя модель принесла пользу на этом направлении…»

Замещение

Замещение (overriding) является чрезвычайно важной концепцией исправления ошибок. В случае структурированного интеллекта замещение работает благодаря наличию специального *доверенного интеллекта* (blessed intelligence), обычно созданного и поддерживаемого людьми. Доверенный интеллект может замещать все другие интеллекты, обычно созданные посредством машинного обучения, – независимо от того, что они говорят.

Одним из способов, которым реализуют замещение, является ручная маркировка конкретных контекстов с определенными результатами. Данный подход можно использовать для точного исправления некоторых ошибок. Например:

- *именно эта* страница несмешная, независимо от того, что думает любой интеллект;
- если датчики тостера выдают определенную комбинацию, жарить ровно две минуты, независимо от того, что думает интеллект, – потому что мы точно знаем, что это за продукт.

Замещение может быть реализовано в виде правил, которые служат защитными ограничениями против явных ошибок. Например:

- если за последние пять минут в жаровню сброшено десять гранул топлива, то следует остановиться независимо от того, что говорит интеллект;
- если на странице есть любое из 15 оскорбительных слов или словосочетаний, она несмешная, независимо от мнения интеллекта.

Замещение интеллекта следует использовать крайне осмотрительно. Не пытайтесь решить проблему замещением – оно должно быть лишь страховкой от самых худших ошибок.

Подход с использованием замещения обладает следующими свойствами:

- точность: средняя. При условии ограниченного использования замещений. Они должны быть защитой, а не сложным интеллектом, созданным вручную;
- простота развития: ++. Разработчик интеллекта (или неподготовленный человек со здравым смыслом) может определить контекст и задать результат. Дальше дело техники;
- взаимозависимость: средняя. Замещения в некоторой степени связаны с ошибками, которые они исправляют, и могут в конечном итоге задержаться в системе дольше, чем действительно необходимо. Если за ними не следить, то со временем они превращаются в проблему обслуживания;

- наглядность: +. Каждое взаимодействие можно отследить до любых замещений, которые на него повлияли. Разработчики иногда забывают проверить все замещения при оценке нового интеллекта, что может привести к некоторой путанице;
- измеримость: +. Легко измерить, сколько положительных и отрицательных взаимодействий оказывает на пользователей каждое замещение;
- поддержка командной работы: +. Разумное использование замещений повышает продуктивность команды разработчиков интеллекта. Однако есть вероятность конфликта между создателями интеллекта и авторами замещений.

Итог главы

Большинство крупных интеллектуальных систем фактически представляет собой набор интеллектов. Структурирование интеллекта позволяет продуктивно сотрудничать нескольким разработчикам, дешево устранять ошибки, использовать подходящие типы интеллекта для решения задачи по частям и задействовать ранее созданный интеллект.

Хорошо структурированный интеллект будет точным, легко развивающимся, слабо связанным, сжимаемым, измеримым и комфортным для командной работы. Разумеется, он также будет точным, однако иногда приходится жертвовать структурированием ради точности.

Существуют различные способы структурирования интеллекта. В этой главе представлены некоторые из наиболее распространенных подходов, но есть и другие. Основные методы структурирования включают в себя:

- разделение признаков;
- конкурентный поиск моделей;
- выделение ошибок;
- метамодель;
- секвенированная модель;
- разделение по контекстам;
- замещение.

Свойства этих методов сведены в табл. 21.1.

Таблица 21.1. Сравнение свойств различных подходов к структурированию интеллекта

Подход	Точность	Простота развития	Взаимозависимость	Наглядность	Измеримость	Командная работа
Разделение признаков	Средняя	+	Средняя	Средняя	–	Средняя
Конкурентный поиск	–	–	Средняя	Средняя	Средняя	– –
Выделение ошибок	–	–	Средняя	Средняя	Средняя	–
Метамодель	++	–	– –	– –	–	Средняя
Секвенированная модель	Средняя	+	++	++	Средняя	+
Разделение по контекстам	Средняя	++	+	+	+	+
Замещение	Средняя	++	Средняя	+	+	+

Темы для размышлений

Прочитав эту главу, вы должны:

- понимать, что требуется для работы в большой, сложной интеллектуальной системе или с командой создателей интеллекта;
- уметь внедрять архитектуру интеллекта, позволяющую нужному типу интеллекта решать соответствующие части задачи, а всем участникам команды продуктивно работать вместе.

Постарайтесь выполнить следующие задания:

- опишите интеллектуальную систему, которая не нуждается в структурировании интеллекта, то есть работает только с одной моделью;
- какие это может вызвать проблемы? Какие проблемы наиболее вероятны?
- разработайте простой план структурирования интеллекта, который решит наиболее вероятную проблему.

Часть V

ОРКЕСТРОВКА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

В главах 22–26 говорится о том, что необходимо для функционирования интеллектуальной системы – управление всеми ее частями, поддержание равновесия между интеллектом и опытом, развитие, отладка и контроль достижения поставленных целей.

В этой части обсуждаются рабочие инструменты, говорится про обнаружение ошибок, совершаемых интеллектуальной системой, поясняется, как на эти ошибки реагировать и что делать, если люди начинают злоупотреблять вашей системой.

Глава 22

Понятие оркестровки интеллекта

Оркестровка интеллекта немного похожа на автомобильные гонки. Целая команда инженеров строит автомобиль, внедряет в него все новейшие технологии и обеспечивает идеальную конструкцию аэродинамического крыла, развесовки, трансмиссии и впускного тракта – они создают потрясающую машину, способную делать то, что не может сделать ни одна другая машина.

А потом кто-то должен сесть за руль, выехать на трассу и победить! Оркестровщики интеллекта – это водители. Они берут в свои руки управление интеллектуальной системой и делают все, что нужно для достижения поставленных целей. Они используют системы создания и управления интеллектом, чтобы извлекать правильные данные в нужное время и обрабатывать их наиболее полезными способами. Они следят за телеметрией и собирают данные, необходимые для улучшения интеллектуальной системы. И они справляются со всеми ошибками и проблемами, настраивая компоненты системы так, чтобы она приносила максимальную пользу для пользователей и вашего бизнеса. Например:

- если интеллект становится точнее, опыт можно делать более сильным для достижения лучших результатов;
- если проблема меняется, может потребоваться оптимизация управления интеллектом, чтобы создавать и развертывать новый интеллект параллельно или просто обновлять его быстрее;
- если система перестает достигать своих целей, кто-то должен разобраться в причинах и понять, как использовать доступные ресурсы для адаптации к новым условиям.

Это нелегко.

Разработка интеллектуальных систем и их оркестровка – это очень разные виды деятельности. Они требуют совершенно разного мышления. И оба они абсолютно необходимы для достижения успеха.

В этой главе говорится о том, как выглядит правильная оркестровка интеллекта, и обсуждаются компоненты оркестровки, включая жизненный цикл, среду оркестровки, ошибки и злоупотребления.

Что такое хорошая оркестровка

Интеллектуальная система с хорошей оркестровкой:

- **продолжает достигать своих целей с течением времени** – она будет расти, выполняя поставленную задачу все лучше и лучше, пока не достигнет точки убывающей отдачи. Затем она останется на достигнутом уровне и продолжит производить ценность, невзирая на изменение базы пользователей и эволюцию проблемы и окружающего мира;
- **обладает балансом между опытом, интеллектом и целью** – то есть опыт будет столь же сильным, как интеллект, на который он опирается, так что выгода для пользователей и бизнеса будет максимальной;
- **успешно устраняет ошибки** – дорогостоящие ошибки быстро выявляются, осознаются и исправляются. Как следствие менее затратные ошибки тоже сглаживаются и не оказывают заметного влияния на пользователей или на бизнес;
- **хорошо масштабируется с течением времени** – затраты на поддержание системы в хорошем состоянии (отладка, доработка интеллекта, сглаживание ошибок и т. д.) оптимально масштабируются в зависимости от количества пользователей и сложности интеллекта. В частности, это означает продуктивный компромисс между машинным обучением и автоматизацией;
- **ухудшается медленно** – для удержания системы в устойчивом состоянии не требуется много ручной работы. Оркестровщик должен заниматься поиском новых возможностей, а не сидеть, вцепившись в руль автобуса.

Зачем нужна оркестровка

Оркестровка интеллекта включает в себя текущую настройку, чтобы интеллектуальная система максимально продуктивно работала в течение своего жизненного цикла.

Сейчас вы можете сказать: «Погодите-ка, я думал, что машинное обучение должно было настроить систему на все оставшееся время службы. Что это? Какая-то шутка?»

К сожалению, нет. Искусственный интеллект и машинное обучение – это лишь инструменты, помогающие вашей лодке отчалить от берега. Оркестровка заключается в том, чтобы брать эти инструменты и применять наилучшим образом – выделять их сильные стороны, компенсировать недостатки – и постоянно реагировать на происходящее.

Оркестровка системы необходима по следующим причинам:

- меняется цель интеллектуальной системы;
- меняются пользователи интеллектуальной системы;
- меняется проблема;
- меняется интеллект;

- меняется стоимость эксплуатации интеллектуальной системы;
- кто-то пытается злоупотребить вашей интеллектуальной системой.

В течение жизненного цикла вашей интеллектуальной системы почти наверняка произойдет одно или несколько изменений. Вы можете превратить эти потенциальные проблемы в возможности, если научитесь их вовремя обнаруживать и адаптировать систему.

Изменение цели

Когда меняются ваши цели, соответственно должны меняться и подходы, а это означает необходимость пересмотра вашей интеллектуальной системы. Необходимость обновить цель может возникнуть в следующих ситуациях:

- **вы стали лучше понимать проблему** – когда вы долго работаете над чем-то, вы начинаете лучше понимать предмет. Возможно, вы осознали, что сначала поставили неправильную цель, и хотите исправить заблуждение. Напомним, что цель можно рассматривать на разных уровнях. У вас может быть прекрасная бизнес-цель, но вы поняли, что результаты, получаемые пользователями, не соответствуют их цели; или вы обнаружили, что свойства модели, которую вы так старательно оптимизировали, не приближают вас к решению проблемы. Вероятно, вы захотите пересмотреть цели и перестроить систему для достижения новых целей;
- **вы решили предыдущую проблему** – обнаружили, что достигли желаемого. Интеллектуальная система работает, пользователи получают хорошие результаты, ваши дела идут прекрасно. На этом этапе вы можете поставить перед собой более высокую цель и соответственно перестроить систему, или же придумать, как снизить расходы на эксплуатацию (об этом мы поговорим позже);
- **вы поняли, что проблема слишком сложна**, – иногда приходится признать свое поражение. Возможно, у вас получилась система, которая работает достаточно долго, но безрезультатно. Вы можете просто отключить систему и заняться чем-то другим или установить более доступную цель, изменить свой подход и действовать медленнее;
- **вы исследуете новую возможность** – у вас может возникнуть отличная идея. Нечто подобное вашей нынешней интеллектуальной системе, но немного другое. Возможно, совершенно новый опыт, который может использовать интеллект. Или новый тип интеллекта на основе существующей телеметрии. Иногда погоня за новой возможностью требует создания новой интеллектуальной системы, но бывает, что задачу удается решить просто за счет изменения цели имеющейся системы.

Изменение пользователей

Пользовательская база интеллектуальной системы будет меняться в течение ее жизненного цикла по разным причинам:

- **приходят новые пользователи** – они могут сильно отличаться от ваших существующих пользователей. Например, вы можете привлечь новую демографическую группу, более случайных пользователей, предпочитающих другие части вашей интеллектуальной системы, или просто людей, думающих по-иному. Когда ваши пользователи меняются, у вас появляется возможность учиться и приспосабливаться. Вы также будете иметь больше телеметрии для развития интеллекта, и вам придется иметь дело с нарастающими проблемами выборки данных и сложностью управления интеллектом;
- **изменяются шаблоны использования** – по мере того как пользователи учатся использовать вашу интеллектуальную систему, они могут изменить способ взаимодействия с ней. Они меньше изучают, как она работает, и больше сосредоточены на получении пользы. Выгода для опытных пользователей обычно отличается от выгоды для новичков, поэтому у вас может возникнуть желание перестроить интеллектуальную систему, когда пользователи станут экспертами;
- **меняется восприятие пользовательского опыта** – со временем пользователи могут устать от сильных и частых взаимодействий. Они могут все больше раздражаться или просто начать игнорировать взаимодействия. Опыт, который неплохо работал в течение первого месяца эксплуатации, в долгосрочной перспективе может оказаться совершенно неправильным, и, возможно, его придется изменить;
- **пользователи уходят** – если они это сделают, наверняка вы немного расстроитесь. Но жизнь продолжается. Исходя из того, какие пользователи уходят, у вас есть возможность оптимизировать интеллект и опыт для оставшихся пользователей. Но при этом также станет меньше телеметрии, что может помешать развитию интеллекта. Возможно, придется пересмотреть автоматизированный анализ телеметрии, чтобы использовать больше исторических выборок вместо ожидания новых данных.

Изменение проблем

Интеллектуальные системы предназначены для больших, сложных, открытых, меняющихся во времени проблем. Эти проблемы по определению переменчивы:

- **изменения происходят постоянно** – поэтому подходы и решения, которые вы приняли в прошлом, могут оказаться неприемлемыми в будущем. Иногда проблема бывает легкой, а иногда может оказаться очень трудной. Иногда в вашем распоряжении множество контекстов, а иногда вы вообще не получаете ничего. Когда проблема меняется, благодаря оркестровке почти всегда есть возможность адаптироваться и улучшить результаты;
- **изменяются шаблоны использования** – по мере того как меняется взаимодействие пользователей с интеллектуальной системой, могут

сильно меняться и решаемые проблемы. Например, летом пользователи ведут себя иначе, чем зимой; они ведут себя по-разному в выходные и будние дни; они воспринимают новый контент иначе, чем старый;

- **вы близки к решению проблемы** – ваш интеллект начинает наступать на пятки большой и сложной проблеме. В этом случае вы можете пересмотреть подход. Возможно, вы захотите зафиксировать текущее состояние системы, шире используя кешированный интеллект (таблицы соответствий) и меньше инвестируя в телеметрию.

Изменение интеллекта

Интеллект системы может измениться, если вы:

- **получите больше данных для создания лучшего интеллекта** – чем больше пользователи взаимодействуют с вашей системой, тем больше у вас данных для развития интеллекта. Когда интеллект становится точнее, интеллектуальный опыт может стать более сильным;
- **найдете новый подход, который меняет положение дел**, – данные открывают возможности. Даже самые мощные методы машинного обучения не всегда эффективны с «маленькими» данными, но становятся жизнеспособными, когда интеллектуальная система получает пользователей, а вы собираете большие данные. Обладая новыми возможностями, вы можете опробовать новый опыт или поставить перед собой более агрессивные цели;
- **попробуете новый подход, который ухудшает ситуацию**, – ведь никто не идеален. Иногда развитие интеллекта идет вспять. Проблему усугубляет наличие комбинированного интеллекта с жесткими или запутанными связями. Кроме того, не следует полагаться на большое вмешательство человека в создание интеллекта (например, с помощью эвристики) – иногда сложность превышает критический уровень, и хрупкий план создания интеллекта летит под откос. В подобных случаях полезно вернуть систему к исходному состоянию, чтобы дать себе время решить проблему.

Изменение затрат

Большие системы постоянно балансируют между затратами и полезной отдачей. Изменения с целью экономии денег могут потребовать компромиссов в других местах. Иными словами, иногда удается изменить опыт или интеллект таким образом, чтобы сэкономить много денег, но при этом лишь незначительно снизить выгоду для пользователей или вашего бизнеса. Вы можете попробовать снизить следующие затраты:

- **расходы на телеметрию** – измените способ сбора данных. Это уменьшает вашу способность следить за системой и развивать интеллект, но может стать хорошим компромиссом. Например, вы можете собирать меньше данных в период, когда интеллект развивается медленно, отбирать меньше данных для уже решенных частей проблемы, или удалять

части контекстов, которые не несут полезные признаки и бесполезны для моделей;

- **стоимость ошибок** – борьба с ошибками может быть очень затратным занятием. Мы обсудим вопрос ошибок в главе 24. Но в целом достаточно очевидно, что вам следует снизить количество и стоимость ошибок. Для этого может потребоваться полная перестройка всех компонентов системы.

Злоупотребления

К сожалению, в интернете полно троллей. Некоторые из них хотят злоупотребить вашим сервисом, потому что просто развлекаются. Но большинство из них пытается заработать деньги на вашем сервисе и ваших пользователях или хочет помешать вашему бизнесу. Безнаказанное злоупотребление может разрушить интеллектуальную систему, сделав ее таким рассадником спама и угроз, что пользователи от нее откажутся.

Чтобы защитить интеллектуальную систему от злоупотреблений, вам необходимо понять, чего хотят злоумышленники, а затем сделать вашу систему менее интересной для них. Иногда для этого достаточно небольшой доработки. А иногда борьба со злоумышленниками становится основной частью работы интеллектуальной системы.

Бороться со злоупотреблениями очень трудно. Стоит одному злодею узнать, как заработать десятую часть цента на взаимодействии с вашими пользователями, – и они налетят толпой. Они расскажут своим друзьям. Нападение может проявиться очень быстро и достаточно остро, поэтому вам нужно быть готовым к обнаружению и реагированию.

Мы подробно обсудим атаки и злоупотребления в главе 25.

Команда оркестровки

Оркестровка интеллектуальной системы требует разнообразных навыков. В частности, члены команды должны:

- быть экспертами в области интеллектуальных систем, чтобы инстинктивно понимать цели и знать, чего хотят пользователи от взаимодействия с системой;
- знать концепцию интеллектуального опыта, уметь рассматривать взаимодействия и понимать, как улучшить качество представления интеллекта пользователям – что легко, что сложно, что нравится пользователям и чего они не хотят;
- понимать, как работает реализация интеллекта, знать, как отслеживать проблемы, иметь некоторые навыки улучшения реализации по мере необходимости;
- уметь задавать всевозможные вопросы о данных, понимать и передавать ответы;

- знать, как применять прикладное машинное обучение, уметь создавать новый интеллект и вводить его в систему по мере необходимости;
- получать удовлетворение от эффективной работы системы изо дня в день. Не всем это нравится – некоторые любят изобретать новые вещи. Суть оркестровки заключается в совершенстве прикладного применения – взять одну из самых потрясающих машин в мире, выехать на гоночную трассу и победить.

Команда оркестровки может состоять из небольшого количества мастеров на все руки. Или это могут быть узкие специалисты с различными навыками, которые хорошо работают вместе.

Итог главы

Успешная интеллектуальная система подлежит оркестровке на протяжении всего ее жизненного цикла. Оркестровка – это процесс управления, сопровождающий все компоненты системы, когда они работают исправно, и поддерживающий их там, где они сталкиваются с проблемами. Интеллектуальная система с хорошей оркестровкой обладает следующими свойствами:

- продолжает достигать своих целей с течением времени;
- имеет баланс между опытом, интеллектом и целью;
- успешно устраняет ошибки;
- хорошо масштабируется с течением времени;
- ухудшается медленно.

Одним из ключевых направлений оркестровки является поддержание баланса между интеллектом и опытом в течение жизненного цикла интеллектуальной системы. Например, делаем опыт более сильным, когда улучшается интеллект, или смягчаем опыт, когда возникают проблемы.

Оркестровка также включает в себя выявление и устранение ошибок с последующей корректировкой системы для снижения их стоимости.

Оркестровка системы необходима по следующим причинам:

- меняется цель;
- меняются пользователи;
- меняется проблема;
- меняется интеллект;
- надо снизить стоимость эксплуатации системы.

Злоумышленники и нарушители часто создают проблемы для успешных интеллектуальных систем. Они должны быть обнаружены, а затем поставлены в такое положение, когда причинение вреда системе не стоит затраченного времени.

Темы для размышлений

Прочитав эту главу, вы должны:

- понять, что такое оркестровка интеллектуальной системы и зачем она нужна;

- знать, какой тип команды необходим для оркестровки интеллектуальной системы.

Постарайтесь выполнить следующие задания:

- представьте, что интеллект системы начинает плохо работать. Какие изменения, не связанные с интеллектом, могут быть сделаны, чтобы смягчить проблему?
- выберите две ваши любимые интеллектуальные системы. Какая из них сложнее в оркестровке? Почему?
- опишите проблемы, с которыми может столкнуться интеллектуальная система, если ее пользовательская база удвоится – и все новые пользователи будут из другой страны, не такой, как у предыдущих пользователей.

Глава 23

Среда оркестровки интеллекта

В этой главе обсуждаются некоторые общие подходы, способствующие успешной оркестровке интеллекта. По сути, это методы эффективной эксплуатации интеллектуальных систем, а именно:

- **мониторинг критериев успеха** – знать, достигает ли система своей цели и становится ли она лучше или хуже;
- **изучение взаимодействий** – проверять контексты по мере получения пользовательского опыта и наблюдать результаты, которых достиг пользователь;
- **оптимизация опыта** – возможно, следует изменить тип, частоту или значимость взаимодействий, чтобы сделать опыт более или менее сильным;
- **переопределение интеллекта** – исправление нечастых, но серьезных ошибок или оптимизация общего контекста;
- **создание нового интеллекта** – управление интеллектом и создание новых моделей для решения неотложных задач, таких как изменение проблемы, пользователей или объема данных.

Все вышеперечисленные и многие другие подходы можно применять во благо интеллектуальных систем. Одни методы могут являться частью реализации интеллекта, например панель показателей, отображающая прогресс в достижении общих целей. Другие могут работать в более узкой области, например сбор данных необработанной телеметрии для выявления проблемных взаимодействий.

В этой главе по очереди обсуждаются различные приемы оркестровки и предлагаются соображения относительно вариантов использования разных подходов, а также критерии, определяющие величину инвестиций в каждый метод.

Мониторинг критериев успеха

Всегда следует знать, достигает ли система своих целей. Мониторинг подразумевает сбор телеметрии, исправление выборок и выпадений данных, по-

лучение результата, понятного всем членам команды, например несколько чисел или график. Напомним, что перечень критериев успеха системы включает в себя:

- **бизнес-цели и опережающие показатели**, особенно отражающие различие между пользователями, взаимодействующими с интеллектуальной частью системы и не участвующими во взаимодействии;
- **результаты пользователя**, показывающие, насколько эффективна интеллектуальная система в достижении ожидаемых результатов;
- **свойства модели**, показывающие, как часто данные являются правильными или неправильными (независимо от результатов пользователя).

Мониторинг критериев успеха имеет решающее значение. В каждой действующей интеллектуальной системе должны быть заинтересованные люди, которым небезразлично, насколько хорошо работает система.

Варианты мониторинга критериев успеха включают в себя:

- **непосредственно по ситуации** (ad hoc) – когда несколько человек обладают навыком анализа целей системы и выработки рекомендаций – например, путем перекрестной проверки настроек телеметрии, отладки сценария и его запуска, представления и распространения данных телеметрии в виде диаграмм;
- **при помощи инструментов** – когда любой член команды может запустить созданный им инструмент, выполнить нужный запрос и представить ответ в удобном формате;
- **автоматизация** – когда система автоматически выполняет периодические запросы и сохраняет результаты в архиве, доступном для всех членов команды;
- **на основе предупреждений** – когда система автоматически собирает и анализирует показатели, а затем выдает оповещения, если что-то происходит, в том числе:
  - **значительное ухудшение**, если показатели намного хуже, чем в предыдущем измерении. Например, количество ошибок выросло на 10 % со вчерашнего дня;
  - **значительная и устойчивая эрозия**, если показатели постепенно снижаются и приближаются к некоторому порогу. Например, доля пользователей, получающих хорошие результаты, снизилась на 10 % за последний месяц;
- **отслеживание популяции** – когда система отслеживает показатели для важных подгрупп отдельно от популяции в целом. Это могут быть люди из определенного места проживания, с определенным шаблоном поведения, в определенной демографической ситуации и т. д.

Критерий целесообразности инвестирования в мониторинг:

- мониторинг предупреждений и критических факторов должен быть реализован почти в каждой интеллектуальной системе.

Изучение взаимодействий

Изучение взаимодействий включает в себя сбор всей телеметрии, связанной с конкретным взаимодействием, и возможность наблюдать взаимодействие от начала до конца. Объекты наблюдения включают в себя: контекст пользователя во время взаимодействия; версию работающего интеллекта; ответ, полученный от интеллекта; интеллектуальный опыт, представленный на основе этого ответа; как пользователь взаимодействовал с опытом и какой результат в конечном итоге получил пользователь.

Информация о взаимодействиях важна для устранения проблем, для отслеживания ошибок, для понимания того, как пользователи воспринимают интеллектуальную систему, и сопереживания пользовательского опыта.

Варианты изучения взаимодействий включают в себя:

- **непосредственно по ситуации** – когда несколько человек обладают навыком выявления всех частей взаимодействия и понимают их взаимосвязь. Возможно, им придется немного поработать детективами в поисках конкретного взаимодействия (или взаимодействий с определенными свойствами), хотя нет гарантии, что искомое взаимодействие представлено в выборке;
- **при помощи инструментов** – когда любой член команды может определить взаимодействие (например, по имени пользователя и времени) и получить для него телеметрию. Инструментарий может также фильтровать запросы по типу взаимодействий (скажем, когда интеллект дал конкретный ответ, а пользователь получил конкретный результат) и изучать их выборку;
- **на основе обозревателя** (browser-based) – когда любой член команды может найти взаимодействия инструментальным методом, но затем испытать взаимодействие от лица пользователя – увидев то же, что и пользователь, нажав кнопки, на которые он нажал, получив тот же самый результат и т. д.;
- **получение данных от пользователей** – когда у оркестровщиков есть возможность выбрать определенные типы взаимодействий и задать пользователям вопросы об их опыте. Например, можно пометить некоторые контексты, а затем опросить небольшую часть пользователей, попавших в эти контексты. Обычно пользователю предлагают оценить свой опыт, ответить на пару вопросов или дать определение результату – хороший или плохой.

Критерии целесообразности инвестирования в изучение взаимодействий:

- если нужно сделать взаимодействия понятными широкому кругу людей. Например, опытные разработчики и владельцы бизнеса смогут понять, как пользователи работают с интеллектуальной системой;
- если вам нужно создать ресурсы для борьбы с ошибками. Благодаря инструментам вы можете задействовать людей, которые не являются экспертами;

- если в вашей системе случаются дорогостоящие ошибки и вы полагаете, что большая часть оркестровки будет посвящена устранению их последствий.

Оптимизация опыта

Если меняется проблема, пользовательская база или интеллект, то открываются возможности для оптимизации опыта. Например, пока интеллект новый и недостаточно качественный, опыт может быть очень пассивным. Он может проявлять себя относительно редко или позволять пользователям игнорировать взаимодействие. Но со временем интеллект станет лучше, и более сильный опыт будет выгоднее для пользователей.

Варианты подходов к оптимизации опыта включают в себя:

- **непосредственно по ситуации** – когда изменения в опыте вносятся путем правки кода и повторного развертывания программного обеспечения;
- **обновление параметров** – когда оркестровщики могут изменять параметры, влияющие на опыт, и вводить новые значения с минимальными затратами (например, по мере обновления интеллекта). Перечень параметров может включать в себя:
  - частоту взаимодействия;
  - цвет или размер подсказки;
  - текст, применяемый в опыте;
  - пороговое значение для автоматизации действий
  - и т. д.;
- **альтернативный опыт** – когда создается несколько версий опыта и у оркестровщиков есть возможность переключаться между ними (используя что-то, похожее на параметр). Например, вы создаете одну версию опыта с деликатной подсказкой, другую версию с заметной подсказкой и третью, которая полностью автоматизирует действие. Затем оркестровщик переключает их по мере необходимости;
- **язык описания опыта** – когда разработчики могут создавать и развертывать опыт вне области определения кода интеллекта, аналогично отделению интеллекта от реализации. Описание опыта может быть реализовано в виде сценариев, которые загружают и выполняют пользователи, или в виде абстрактного языка, предназначенного для сотрудников без инженерного образования, например специальный язык разметки в ограниченной среде выполнения.

Критерии целесообразности инвестирования в оптимизацию опыта:

- если у команды оркестровщиков нет инженерного персонала, то наверняка придется создать элементы управления, понятные для людей без инженерного образования;
- если развертывание нового клиентского кода для ваших пользователей занимает много времени, вы можете вложить средства в разработку

инструментов управления опытом с помощью параметров и обновлений файлов данных;

- если вы предполагаете, что будете многократно менять опыт в течение жизненного цикла вашей интеллектуальной системы, то имеет смысл позаботиться об упрощении процесса обновления;
- если ничего из упомянутого не соответствует действительности – у вас есть инженерные ресурсы, вы можете легко развернуть код и не ожидаете слишком большого количества изменений, – то специально разработанный метод обновления опыта почти наверняка обойдется дешевле.

Переопределение интеллекта

Переопределение интеллекта (overriding intelligence) включает в себя выделение нескольких контекстов и ручное кодирование ответов, предоставляемых интеллектом для этих контекстов. Иногда необходимо переопределить интеллект, чтобы исправить дорогостоящие ошибки, или оптимизировать несколько общих контекстов. Или же вы преследуете бизнес-цель – например, хотите продвинуть часть контента в вашей системе. Наверняка вам не хочется слишком сильно игнорировать интеллект, но иногда без этого не обойтись.

Варианты подходов к переопределению интеллекта включают в себя:

- **непосредственно по ситуации** – когда вы полагаетесь на прямое вмешательство разработчиков интеллекта в процесс обучения, чтобы достичь определенных результатов (что очень сложно), или на инженеров, которые жестко впишут нужные переопределения в код и развернут их для клиентов;
- **как интеллектуальный канал** (intelligence feed) – когда переопределения рассматриваются как очень простой вариант интеллекта, развертываемый вместе с другими интеллектами системы и имеющий самый высокий приоритет выполнения. Он может представлять собой файл данных в каком-то простом формате, сопоставляющий контексты с результатами;
- **при помощи инструментов** – когда вы рассматриваете переопределения как интеллектуальный канал, но создаете инструменты в помощь оркестровщикам. Инструменты должны выполнять различные функции, в том числе:
  - гарантировать правильность указания контекстов для переопределения;
  - предоставлять обратную связь о распространенности указанных контекстов и текущих ответов интеллекта для этих контекстов;
  - представлять отчеты о существующих переопределениях, в том числе о количестве мест и количестве пользователей, на которые они влияют;
  - отслеживать, кто делает переопределения и насколько хорошими или плохими они оказываются;
  - управлять сроком службы переопределений;

- **на основе обозревателя** – когда инструменты объединены в пакет поддержки, включая способы поиска взаимодействий, их просмотра и переопределения всех нужных взаимодействий в одном месте.

Критерии инвестирования в переопределение интеллекта:

- если ваша система способна совершать очень дорогостоящие ошибки, вы почти наверняка захотите вложить средства в первоклассный интеллект, возможно, добавив отслеживание и управление рабочим процессом наподобие обсуждаемых здесь инструментов. Более подробно об этом будет рассказано в следующей главе, посвященной исправлению ошибок;
- переопределение интеллекта может послужить подушкой безопасности для создателей интеллекта. Последнее, что вам нужно, – это чтобы миллионы пользователей интеллектуального продукта заблокировали процесс развития интеллекта мириадами сообщений о всевозможных ошибках. Интеллект создан, чтобы совершать ошибки. Бесполезно карать разработчиков за каждый недостаток. От возможности быстро вручную переписать решения наиболее важных проблем всем будет только лучше.

Создание интеллекта

Важной частью оркестровки является создание и настройка интеллекта, который управляет вашей интеллектуальной системой. В этом разделе резюмируются и обобщаются некоторые из мероприятий, которые помогут интеллекту расти и продуктивно работать.

Инвестиции в создание интеллекта могут включать:

- **управление интеллектом**:
  - контроль над обновлением и применением данных;
  - управление версиями и местом расположения интеллекта;
  - добавление новых каналов интеллекта в систему;
  - управление комбинированным интеллектом;
- **автоматизированные средства машинного обучения** регулярно и практически без присмотра производят новый интеллект. Зачастую эти средства позволяют менять различные параметры процесса обучения, включая следующие:
  - сколько данных использовать;
  - какие данные использовать;
  - как часто запускать процесс создания модели;
  - сколько времени потратить на поиск хороших моделей;
  - любые другие параметры процесса моделирования, которые можно настраивать;
- **среду создания интеллекта** – место, где накапливаются и становятся доступными для машинного обучения телеметрия и результаты применения разных подходов, включая испытания новых алгоритмов машинного обучения, извлечение новых признаков и добавление новых

аналитических методов. Там могут создаваться новые интеллекты или улучшаться существующие автоматизированные системы обучения;
- **поддержку конструирования признаков** – когда создатели интеллекта имеют возможность легко опробовать новые признаки, а также быстро и безопасно добавить полезные признаки в интеллект и развернуть его для пользователей;
- **системы телеметрии** – когда оркестровщик может управлять перечнем и объемом собираемых данных.

Критерии инвестирования в создание интеллекта

Подсистема создания интеллекта – это важная часть интеллектуальных систем (что вполне логично следует из названия). Вы, вероятно, в конечном итоге инвестируете в несколько систем для поддержки этого процесса.

Наиболее эффективные инструменты создания и развития интеллекта обладают следующими достоинствами:
- помогают предотвратить ошибки;
- автоматизируют повседневные задачи;
- упрощают многошаговые процессы;
- ведут простой контрольный журнал.

Разрабатывая инструменты для поддержки развития интеллекта, избегайте соблазна смешать их с основными инструментами создания интеллекта (такими как системы машинного обучения или среды выполнения). Основные инструменты, как правило, являются стандартными, и инновации в них вряд ли принесут вам заметную отдачу.

Итог главы

В этой книге говорится о различных подходах, которые важны для эксплуатации интеллектуальной системы. В данной главе упомянуты некоторые из них, в том числе важные критерии и подходы к инвестированию:
- мониторинг критериев успеха;
- проверка взаимодействия;
- балансировка опыта;
- переопределение интеллекта;
- создание интеллекта.

Вы можете построить успешную интеллектуальную систему, не вкладывая много средств в эти процессы на этапе внедрения. Но вряд ли вы обойдетесь без них в течение жизненного цикла системы. За вами остается решение – сколько труда и средств потратить на инструментарий поддержки.

И имейте в виду, что наличие хороших инструментов позволяет со временем заменить штат сотрудников и создателей системы (которым часто нравится работать над новыми интересными вещами) на квалифицированных оркестров-

щиков, которые должны быть универсалами со стремлением к устойчивому совершенству. Это очень разные навыки.

Зачастую среда оркестровки развивается так же, как интеллектуальная система, –небольшие вложения увеличивают ценность в долгосрочной перспективе до тех пор, пока дальнейшие инвестиции не потеряют смысла.

Темы для размышлений

Прочитав эту главу, вы должны:

- знать основные инструменты, необходимые для поддержания работоспособности интеллектуальной системы;
- уметь начинать с небольших вложений и масштабировать инвестиции с течением времени;
- иметь представление о применении различных инструментов и знать, какие из них наиболее важны для вашей интеллектуальной системы.

Представьте себе существующую сегодня важную интеллектуальную систему. Постарайтесь ответить на следующие вопросы:

- как вы думаете, в какой области расходуется больше всего времени на оркестровку: мониторинг успеха, проверка взаимодействий, настройка опыта, переопределение интеллекта или создание интеллекта?
- что бы вы предложили для снижения расходов в этой области?

Глава 24

Работа над ошибками

Ошибки неизбежны. Люди совершают ошибки. Искусственный интеллект тоже ошибается – и еще как! Ошибки могут вызывать раздражение пользователя или иметь катастрофические последствия.

Каждая интеллектуальная система должна иметь стратегию выявления ошибок – например, следить за критическими показателями и предоставлять пользователям возможность сообщить о неполадках.

Каждая интеллектуальная система также должна иметь стратегию исправления ошибок. Возможно, вам придется обновить интеллект или принудительно удерживать пользователей от неправильных действий. Иногда можно предложить обходной путь или возмещение для пострадавших пользователей.

Некоторые ошибки очень трудно найти или исправить без внесения новых ошибок. А на исправление других ошибок потребуется очень много времени – часы на развертывание новой модели или месяцы, чтобы придумать новую стратегию интеллекта.

В этой главе обсуждаются способы исправления ошибок, включая следующие темы:

- типы ошибок, которые может совершать система (особенно плохие);
- причины, по которым интеллект может совершать ошибки;
- способы смягчения последствий ошибок.

Каждый оркестровщик интеллектуальной системы должен смириться с неизбежностью ошибок.

Худшее, что могло случиться

Спросите себя: что самое худшее может сделать моя интеллектуальная система?

- возможно, это будут незначительные ошибки, например мигание индикатора, который не заботит пользователя, или проигрывание песни, которую он не любит;
- возможно, система напрасно тратит время и усилия на автоматизацию действий, которые пользователь отменяет, или переключает внимание пользователя с того, что ему интересно на самом деле, на то, в чем интеллект ошибается;

- ошибка может стоить вашему бизнесу денег, если система израсходует слишком много ресурсов процессора или трафика либо случайно скроет от пользователей ваш лучший и самый прибыльный контент;
- возможно, вы подвергнетесь юридическому преследованию, если система совершит противозаконные действия, или помешает пользователям либо конкурентам вести бизнес и причинит им ущерб, за который вы можете понести ответственность;
- возможно, система нанесет непоправимый вред, удалив важные файлы, расплавив печь или отправив оскорбительное сообщение от имени одного пользователя другому;
- может быть, система причинит вред здоровью – или даже убьет кого-то.

Когда вы размышляете о своей системе, прежде всего вы думаете о том, насколько она будет удивительной, как много хорошего она сделает, о людях, которые ее полюбят. Вы не хотите думать о проблемах и даже пытаетесь их игнорировать.

Не делайте так.

Придумайте худшую ошибку, которую может сделать ваша система.

Затем найдите вторую худшую ошибку.

А потом третью.

Затем попросите пятерых пользователей сделать то же самое. Выслушайте их соображения и примите их к сведению.

А теперь, когда у вас есть список из пятнадцати действительно плохих вещей, которые может сделать ваша интеллектуальная система, спросите себя: это приемлемо?

Потому что эти ошибки непременно произойдут, их будет трудно найти, и их будет трудно исправить.

Если система способна сотворить такое, что вам страшно даже представить, – возможно, вам захочется спроектировать другую систему, которая никогда не сможет сделать ничего плохого, независимо от того, что говорит интеллект. Может быть, вы придете к выводу, что окончательное решение должен принимать человек или надо использовать менее сильный опыт. Или просто найдете себе другое занятие в жизни…

Потому что интеллект будет делать ошибки, и рано или поздно случится худшее.

Причины выхода системы из строя

Интеллектуальная система совершает ошибки по разным причинам – это либо проблемы управления и внедрения, либо проблемы самого интеллекта. В этом разделе мы обсудим потенциальные источники проблем, в том числе:

- отказы системы;
- отказы модели;
- ошибки интеллекта;
- деградация интеллекта.

Первым шагом к исправлению ошибки является понимание причины ее возникновения.

Отказы системы

Иногда компьютеры выходят из строя, падает скорость соединения или обрывается сетевой кабель. Иногда случаются менее очевидные ошибки, связанные с взаимодействием компонентов системы. Эти проблемы связаны с реализацией или функционированием вашей интеллектуальной системы, но они могут отображаться в пользовательских отчетах, эскалациях и ухудшающих показателях наравне с ошибками интеллекта.

Локализация подобных проблем может быть затруднена в больших системах, особенно когда интеллект распределен между клиентами (которые вдобавок находятся на разных стадиях обновления) и несколькими серверами (которые могут располагаться в различных центрах обработки данных).

Катастрофические поломки, как правило, легко найти по их сокрушительным последствиям. Но частичные неполадки бывают относительно незаметными. Например, предположим, что 1 % вашего трафика идет на конкретный сервер, который внезапно рушится. Один процент ваших пользователей получает плохой опыт, и, возможно, они сообщают об этом снова и снова… Но это всего лишь 1 % от вашей пользовательской базы. 99 % ваших пользователей довольны. Сможете ли вы заметить неполадки?

Отказы системы должны быть редкими и подлежат немедленному устранению. Многочисленные неполадки системы парализуют работу интеллекта – и просто подрывают моральный дух.

Отказы модели

Если говорить о неполадках системы, то отказ модели (model outage) – это скорее проблема реализации, чем проблема интеллекта, но у них будут похожие симптомы.

Отказ модели может произойти, если:

- файл модели поврежден при развертывании;
- файл модели не синхронизирован с кодом, который превращает контексты в признаки;
- среда создания интеллекта не синхронизирована со средой выполнения интеллекта;
- интеллект не синхронизирован с опытом.

Подобные проблемы очень трудно обнаружить. Представьте, например, что код обработки признака обновили в среде создания интеллекта, но забыли обновить в среде выполнения. Затем, когда новая модель (построенная для обновленного признака) будет передана в среду выполнения (с использованием устаревшего признака), она будет сбита с толку. Модель получит значения признаков, которые не ожидала, и возникнет ошибка. Из-за этого, возможно, точность во время выполнения будет на 5 % хуже, чем в лаборатории. Все про-

верки в лаборатории показывают, что интеллект работает нормально, но пользователи получают немного худший опыт.

Поскольку эти проблемы очень трудно выявить, каждая реализация интеллекта должна проходить проверки, проверки и еще раз проверки, чтобы убедиться, что среда создания интеллекта синхронизирована со средой выполнения, что все развернуто правильно и все компоненты системы синхронизированы.

Ошибки интеллекта

Когда модели, образующие интеллект, не идеально отражают действительность (обычно это так и есть), то возникают ошибки. Напомним, что создание интеллекта – это поиск компромисса между обучением очень сложной модели, которая хорошо отражает текущую проблему, и обучением модели, которая хорошо обобщает новые контексты. Всегда будут оставаться пробелы – места, где модель не совсем верна.

Эти пробелы вызывают ошибки, которые трудно исправить с помощью совершенствования интеллекта. Вы можете попробовать другой вариант модели, но она будет делать иные ошибки. Можно собрать больше данных, но отдача от них уменьшается. Вы можете попробовать добавить больше признаков, и это обычно помогает. Но ошибки интеллекта всегда будут существовать.

Эти ошибки будут казаться немного случайными. Они будут меняться со временем, по мере поступления обучающих данных. Их нелегко исправить – для этого потребуются постоянные усилия, и по мере развития интеллекта будет все труднее.

Еще одна трудность – выяснить, какая часть интеллекта отвечает за ошибку. Когда модели слабо связаны (например, определен порядок выполнения и выигрывает первая модель, которая дала ответ), то сразу понятно, какая именно модель дала неправильный ответ. Но когда модели тесно связаны (например, когда выходы нескольких моделей объединяются с использованием сложной эвристики или метамодели), становится труднее отследить ошибку. Если шесть моделей частично отвечают за ошибку, с какой начать?

Деградация интеллекта

Если вы имеете дело с открытой, меняющейся со временем проблемой, то интеллект, который радовал вас вчера, уже не будет столь хорош сегодня. Изменение проблем снижает общее качество интеллекта, потому что новые ошибки возникают, даже если вы ничего не меняете. Кроме того, для сбора обучающих данных по «новой» проблеме потребуется время, а значит, придется подождать, пока вы сможете отреагировать на перемены. Возможно, вы никогда не сможете набрать достаточно данных для обучения, чтобы хорошо решить текущую проблему (пока вы собираете данные, проблема теряет актуальность).

Есть две основные категории изменений:

1) когда со временем появляются новые контексты или исчезают старые – в этом случае вам придется создать новый интеллект для работы в новых

контекстах, но существующие обучающие данные все еще могут применяться в старых контекстах;

2) когда значение контекстов меняется со временем – в этом случае существующие обучающие данные могут вводить в заблуждение, и вам придется сосредоточиться на новой телеметрии, чтобы создать обновленный интеллект.

Один из способов исследовать деградацию интеллектуальной системы – сохранить старые версии интеллекта и выполнить набор интеллектов на текущих данных – версия вчерашнего дня, пять дней назад, десять дней назад и т. д. Глядя на то, как меняются ошибки, вы можете получить представление о том, как развивается ваша проблема, и использовать это понимание при выборе способа адаптации интеллекта.

Снижение количества ошибок

Случайные и недорогие ошибки вполне приемлемы. Но когда количество ошибок резко возрастает, когда они становятся систематическими, опасными или дорогостоящими, вам следует подумать о снижении количества ошибок.

В этом разделе обсуждаются следующие подходы к снижению количества ошибок:

- инвестиции в развитие интеллекта;
- настройка значимости опыта;
- настройка параметров управления интеллектом;
- установка ограничителей;
- переопределение ошибок.

Инвестиции в развитие интеллекта

В здоровой системе интеллект будет постоянно улучшаться. Один из способов справиться с ошибками – подождать, пока интеллект наверстает упущенное.

Фактически почти все иные подходы к снижению количества ошибок уменьшают ценность интеллектуальной системы для пользователей, у которых нет проблем, за счет снижения значимости опыта, или добавляют сложность и стоимость обслуживания в долгосрочной перспективе (добавление ручных настроек, которые необходимо поддерживать). Поэтому совершенствование интеллекта – отличный способ справиться с ошибками, когда это возможно. Наилучшие способы инвестировать в развитие интеллекта для снижения количества ошибок:

1) получить более релевантную телеметрию или обучающие данные для контекстов, в которых происходят ошибки. Благодаря этому можно обойтись небольшими трудозатратами при обновлении интеллекта;
2) помогите создателям интеллекта расставить приоритеты на тех частях системы, которые отнимают больше времени. Распределите ошибки по категориям (по месту проживания, возрасту, свойствам пользователей и т. д.) и назначьте приоритет каждой категории. Затем создатели интел-

лекта могут работать над признаками и моделями в этих конкретных областях – например, методами *разделения контекстов* и *избирательного моделирования* (focused modeling);

3) предоставьте больше людей и инструментов для создания интеллекта.

Эти вложения улучшат общее качество интеллекта, и со временем все встанет на свои места.

И пожалуй, худший способ вложений в развитие интеллекта – это отслеживать ошибки интеллекта, как будто это дефекты программы, и заставлять создателей интеллекта исправлять их одну за другой до тех пор, пока они не закончатся. Такой подход не работает. Если есть ошибки, которые необходимо исправить любой ценой, значит, воспользуйтесь одним из методов смягчения последствий, о которых говорится в следующих разделах.

Настройка значимости опыта

Довольно трудно исправлять ошибки, вызванные ухудшением интеллекта, а также случайные и незначительные ошибки. В подобных случаях можно попробовать сделать опыт менее сильным, а ошибки – менее дорогостоящими.

Если проблемы достаточно серьезны, вы можете отключить интеллектуальный опыт, пока не найдете решение.

В предыдущих главах мы уже обсуждали поиск оптимального отношения между качеством интеллекта и силой опыта, поэтому я не буду повторяться.

Настройка параметров управления интеллектом

Если ошибки вызваны деградацией интеллекта – когда быстро возникают новые контексты или старые контексты меняют смысл, – их можно устранить за счет ускоренного обучения и развертывания новых моделей.

Вы также можете изменить состав обучающих данных – например, быстро ликвидировать старые данные (когда контексты меняют значение) или повысить долю выборок из новых контекстов в телеметрии и обучающих данных (когда быстро появляются новые контексты).

Эти подходы похожи на инвестиции в интеллект, но дают более быстрый отклик. Допустим, когда наступает выходной, внезапно портится погода или запускается новый продукт, ваша проблема меняется быстрее, чем обычно. Оркестровщик может знать об этом и поправить кое-какие настройки заранее, а не ждать, пока создатели интеллекта научатся предсказывать подобные события.

Установка ограничителей

Иногда вы сталкиваетесь с откровенно глупыми ошибками. Человеку достаточно беглого взгляда, чтобы понять, что решение не может быть правильным, например:

- если пеллетный гриль разогрет до 800 градусов, вам не придет в голову добавить топливо;
- если пользователю 10 лет, вы не станете показывать ему фильм ужасов.

В этих случаях вы могли бы попытаться обмануть алгоритм создания интеллекта, чтобы заставить его избегать подобных ситуаций или собрать специальные обучающие данные. Вы могли бы довести до изнеможения разработчиков интеллекта и потратить месяцы их работы…

Или вы могли бы установить *ограничитель* (guardrail) – простую эвристику для переопределения интеллекта, когда он собирается сделать что-то безумное.

При использовании ограничителей следуйте простым правилам:

1) будьте консервативны – решайте только очевидные проблемы и не втягивайтесь в ручное создание сложного интеллекта;
2) пересматривайте свои решения – отслеживайте эффективность и стоимость ограничителей. Удаляйте или ослабляйте ограничители, которые становятся менее важными по мере улучшения интеллекта или изменения проблемы.

Переопределение ошибок

Иногда ваша система будет совершать дорогостоящие ошибки, которые невозможно смягчить как-то иначе, кроме исправления результата вручную, например:

- вы запускаете поисковую систему, и верхний результат по запросу «игры» не об играх;
- вы запускаете службу защиты от спама, и она удаляет всю почту из легальных источников;
- вы запускаете сайт электронной коммерции, и он удаляет товар за «нарушение правил», хотя товар ничего не нарушил;
- вы запускаете приложение для поиска смешных сайтов, и оно помечает самый популярный развлекательный сайт в интернете как несмешной.

Когда эти ошибки достаточно важны, вам может потребоваться особый пользовательский опыт, позволяющий сообщать о проблемах. Наверняка вам понадобится также механизм обработки этих отчетов. Например, вы можете создать группу поддержки с необходимыми инструментами и рабочими процессами для проверки каждого сообщения об ошибке в течение часа, 24 часов в сутки, 7 дней в неделю.

Как и в случае с ограничителями, не используйте переопределения слишком часто и постоянно следите за их качеством и стоимостью.

Итог главы

У вас должен быть продуманный план выявления и устранения ошибок, без которых не обходится ни одна интеллектуальная система. Важную часть этого плана составляет выяснение того, какие наихудшие ошибки может делать ваша система. Не обманывайте себя. Вообразите самую плохую ошибку.

Чтобы решить проблему, следует знать ее причину. Ошибки могут возникать, когда:

- часть вашей интеллектуальной системы отключена;
- ваша модель создана, развернута или интерпретирована неправильно;
- ваш интеллект не идеально подходит для этой проблемы (обычно так и есть);
- меняется проблема или состав пользовательской базы.

После того как вы обнаружили проблему, вы можете уменьшить ее несколькими способами:

- дополнительные вложения в интеллект;
- перенастройка опыта;
- изменение параметров управления интеллектом;
- установка ограничителей;
- переопределение ошибок.

Активное устранение ошибок позволяет остальной части вашей интеллектуальной системы быть более агрессивной и добиться большего воздействия на пользователя. Принятие ошибок и готовность к их устранению – важная часть оркестровки интеллектуальной системы.

Темы для размышлений

Прочитав эту главу, вы должны:

- знать, когда и как ошибки подвергают риску интеллектуальную систему;
- уметь определять работоспособность интеллектуальной системы и выявлять наиболее распространенные причины неполадок;
- уметь сглаживать последствия ошибок на основе общих подходов и знать, когда использовать определенные подходы.

Постарайтесь ответить на следующие вопросы:

- какова самая распространенная ошибка интеллектуальной системы, о которой вы знаете?
- какая из ошибок этой системы самая дорогостоящая?
- разработайте метод устранения одной из этих двух ошибок – распространенной или дорогостоящей;
- подходит ли этот метод для другой ошибки? Почему?

Глава 25

Злоумышленники и злоупотребления

Всякий раз, когда вы создаете что-то ценное, найдутся желающие извлечь из этого выгоду. Интеллектуальные системы не исключение. Если вы тратите энергию, деньги и время на привлечение пользователей – кто-то попытается заработать на этих пользователях. Если вы строите бизнес, мешающий конкурентам, то они постараются усложнить вам управление этим бизнесом.

Вот некоторые распространенные способы, которыми злоумышленники могут навредить интеллектуальной системе:

- монетизация ваших пользователей, например путем рассылки спама;
- кража информации о вашей системе или пользователях для последующей продажи;
- использование вашей платформы для организации атак на другие системы;
- вторжение в систему для нарушения ее работоспособности.

Некоторые из действий злоумышленников являются незаконными, а иные балансируют на грани закона. И даже когда действия явно противозаконны, глобальный характер интернета затрудняет поиск злоумышленников, а привлечь их к ответственности еще труднее.

По этой причине все успешные интеллектуальные системы должны быть защищены от злоупотреблений.

В этой главе обсуждаются основные принципы злоупотреблений, чтобы вы осознали вызов, стоящий перед вами, научились распознавать действия злоумышленников и применять инструменты, затрудняющие доступ противника к вашей системе.

Злоупотребления – это бизнес

Первое, что нужно знать о злоупотреблениях, – это бизнес, большой бизнес. Подавляющее большинство людей, совершающих злоупотребления, делают это, чтобы заработать деньги (хотя иногда это делается для развлечения, вос-

становления социальной справедливости или с целью шпионажа). Вот некоторые из способов, с помощью которых злоумышленники могут зарабатывать деньги:

- **управление трафиком** – противник обманывает интеллектуальную систему, чтобы показать пользователям нужную информацию. По сути, это реклама. Злоумышленник получает оплату за каждого пользователя, которого он перенаправляет с вашего сайта на сайт заказчика;
- **компрометация личной информации** – хищение номера социального страхования, контактной информации, паролей, банковских данных и т. д. Злоумышленники могут использовать эту информацию напрямую или перепродать ее другим злоумышленникам, чтобы получить быстрый доход;
- **компрометация компьютеров** – обман пользователей, чтобы они устанавливали вредоносные программы на свои компьютеры. Если на компьютере пользователя установлена такая программа, злоумышленники могут украсть личную информацию или использовать компьютер пользователя для атак на другие компьютеры. Используя вашу интеллектуальную систему для общения с пользователями, злоумышленники могут спровоцировать их на разные необдуманные поступки;
- **накрутка контента** – обман интеллектуальной системы для навязывания нужного поведения. Например, злоумышленник может разместить поддельные отзывы о товаре на портале электронной коммерции, чтобы привлечь внимание к товару и увеличить продажи;
- **подавление контента** – причинение вреда вашей интеллектуальной системе или другим пользователям системы. Например, злоумышленник может помечать все сообщения конкурента как оскорбительные;
- **прямая кража** – кража контента для перепродажи, например хищение виртуального имущества в онлайн-игре.

Злоумышленники создали рынки сбыта всевозможных нелегальных вещей, так что любой негодяй, обнаруживший лазейку в вашей интеллектуальной системе, легко превратит свою деятельность в деньги.

Масштабы злоупотреблений

Представьте себе занятие, благодаря которому вы можете заработать десятую часть цента. Нажмите кнопку, выберите опцию, нажмите еще одну кнопку – динь! – десятая часть цента появилась на вашем банковском счете.

Выглядит абсолютно бессмысленно. Вы должны выполнить эту деятельность тысячу раз, чтобы заработать доллар, сто тысяч раз, чтобы заработать сто долларов. Что за нелепая трата времени!

А теперь представьте, что вы запрограммировали компьютер, чтобы он работал вместо вас, а компьютер может делать это миллион раз в час, ежечасно, до скончания веков. Это и есть цифровое злоупотребление.

Как правило, злоупотребление интернетом предполагает деятельность, которая либо случается редко, но дает большой доход (например, обманное получение пароля), либо приносит очень маленький разовый доход (например, переадресация трафика на чужой веб-сайт), но зато повторяется часто и многократно. И плохие парни делают на этом хорошие деньги.

Это означает, что проблема злоупотреблений не обязательно возникает за один день. Вы думаете, что у вас все хорошо, но злоумышленники тем временем экспериментируют, пробуют разные методы, наблюдают, как часто пользователи клюют на приманку или сколько трафика можно извлечь из ваших пользователей, делают расчеты – и как только увидят выгоду, ситуация взрывается.

Злоупотребление легко переходит от нуля к катастрофе за одну ночь.

Оценка вашего риска

Ваша интеллектуальная система будет интересна для злоумышленников, если выполняется любое из следующих условий:

- **у нее много пользователей** – поскольку ваша интеллектуальная система становится все более популярной, у нее растет число пользователей. Это означает, что злоумышленники могут увеличивать масштаб атаки и зарабатывать больше денег. Их доходы окупают затраты на эксперименты с вашей системой, потому что если они смогут заработать десятую часть цента с пользователя, то получат хороший доход;
- **злоумышленники могут использовать вашу систему для общения с пользователями** – особенно если они могут добавить активную ссылку в свое общение. Общение может быть представлено в любой форме – электронная почта, веб-сайт, изображение, рецензия, комментарий. Злоумышленники найдут способы заработать на общении с пользователями, обманывая их и рассылая спам;
- **система генерирует персональный контент пользователя** – особенно когда интеллект может как-то влиять на содержание, аннотацию, отображение или ранжирование контента. От восприятия контента пользователем зависит то, чей контент принесет больше денег своему обладателю. Злоумышленники попытаются вклиниться между интеллектом и пользователем;
- **ошибки, которые совершает система, стоят кому-то денег**, – особенно если затраты могут быть направлены на конкретную жертву. Например, когда умный тостер сжигает хлеб определенной марки или когда ваша интеллектуальная система выдавливает из бизнеса менее интеллектуального конкурента;
- **она делает любые другие вещи, на которых может заработать отдельный умник**, – воспринимайте злоумышленников как умных хакеров, у которых сомнительная мораль и много свободного времени. Приготовьтесь удивляться их изобретательности.

Признаки злоупотреблений

Когда злоумышленники развивают атаку на вашу интеллектуальную систему, они создают странные *шаблоны использования* (usage pattern). Вы можете обнаружить их в телеметрии, если ищете:

- большие группы пользователей, которые используют вашу интеллектуальную систему необычными способами (слишком сосредоточены на частях системы, связанных с деньгами);
- контексты, которые показывают всплеск активности по сравнению с обычным использованием;
- контексты, в которых резко меняется распределение результатов, потому что злоумышленники посылают вам неверную информацию;
- похожие жалобы пользователей и сообщения о проблемах.

Иногда злоумышленники стараются объединиться, но им сложно имитировать действия ваших легальных пользователей, поэтому можно обнаружить атаки, если потратить время на поиск аномалий.

Вы также можете обнаружить атаки задним числом, настроив оповещения о радикальных изменениях. Резко возрос трафик в определенной части вашей системы? Об этом следует знать. Удвоилось количество жалоб? Кто-то должен на это взглянуть.

Способы борьбы со злоупотреблениями

Если злоупотребление действительно становится проблемой, попробуйте следующие подходы:

- увеличение стоимости продукта;
- снижение привлекательности для злоумышленников;
- машинное обучение против злоумышленников;
- отключение злоумышленника от системы.

Увеличение стоимости продукта

Вы можете остановить злоупотребления и заработать больше денег! Великолепно!

Взимание большей платы может отпугнуть и ваших законных клиентов, поэтому не всегда приемлемо. Но имейте в виду, что злоупотребление – это бизнес, и ваш лучший способ прекратить злоупотребления – сделать так, чтобы нарушителям было трудно получать прибыль.

Например, получение прибыли от злоупотреблений станет затруднительным, если нарушителю необходимо:

- платить 10 центов за учетную запись в вашей интеллектуальной системе;
- распознавать и вводить специальные символы (капчу) для каждого отзыва, который они хотят оставить;
- купить новый умный тостер для каждой атаки, которую они хотят запустить.

Затраты наиболее эффективны, когда они влияют на злоумышленников больше, чем на законных пользователей. Например, если каждый законный пользователь должен вводить капчу только один раз, но злоумышленник должен вводить символы для каждого действия, которое он выполняет. Если все сделано правильно, добавление затрат отпугнет злоумышленников, а законные пользователи даже не заметят происходящее.

Снижение привлекательности для злоумышленников

Вы можете изменить свой продукт, чтобы он меньше интересовал злоумышленников, например:

- отключить или ограничить каналы связи;
- сократить объем личной информации пользователей, хранимой на сайте, и соблюдать осторожность при отображении;
- уменьшить влияние обратной связи с пользователем на представление контента.

Эти меры могут снизить полезность продукта для законных пользователей, но иногда одной или двух мелких доработок достаточно, чтобы лишить злоумышленников способности получать прибыль.

Машинное обучение против злоумышленников

Вы можете использовать машинное обучение для выявления подозрительных взаимодействий, а затем блокировать их. Я полагаю, что к этому моменту вы уже подумали: «Это целая книга о машинном обучении, поэтому машинное обучение должно быть хорошим способом остановить злоупотребления, верно?»

К сожалению, это не совсем так. Машинное обучение – прекрасный инструмент, но злоумышленники очень ловко применяют атаки по шаблонам, которые вводят в заблуждение машинное обучение. И обычно обновление шаблона атаки обходится злоумышленникам намного дешевле, чем машинное обучение новой модели.

Вы можете использовать машинное обучение для борьбы со злоупотреблениями, и это, вероятно, сработает. Но я рекомендую сначала рассмотреть другие варианты обороны и убедиться, что машинное обучение действительно может разрушить бизнес-модель злоумышленника.

Отключение злоумышленника от системы

Всякий раз, когда обнаружено злоупотребление, вы должны заблокировать все, что нарушитель использовал для атаки, включая учетную запись, тостер, гриль, доступ к развлекательному сайту, оросительную систему, – абсолютно все. Это гарантирует, что резко возрастут затраты злоумышленников на масштабирование атаки. Вчерашняя инфраструктура, задействованная в атаке, ликвидирована, поэтому сегодня им придется начинать с нуля.

Другой вариант – сосредоточиться на создании интеллекта только для доверенных пользователей.

Представьте, что у вас есть сто тысяч пользователей, которые в течение многих лет используют ваш интеллектуальный сервис, создают телеметрию и контексты с результатами обучения. Эти пользователи вполне безопасны – они не являются аккаунтами злоумышленников для запуска новой атаки. Они ваши клиенты. Ограничивая применение интеллекта «заведомо хорошими» пользователями, вы зачастую можете полностью избежать злоупотреблений.

Итог главы

Всякий раз, когда вы создаете ценность, приходят злоумышленники и пытаются извлечь выгоду из вашей тяжелой работы, подвергая риску ваших пользователей и вашу интеллектуальную систему.

Подавляющее большинство злоупотреблений делается, чтобы заработать деньги. Разобравшись, как злоумышленники зарабатывают деньги, вы можете снизить привлекательность интеллектуальной системы для нарушителей. Часто самый дешевый способ борьбы со злоупотреблениями – это внести в работу вашей системы несколько небольших изменений, чтобы злоумышленники не смогли получить ощутимую прибыль.

Злоупотребления, как правило, нацелены на малоценные действия, которые можно многократно повторить – одна десятая цента, но миллион раз в день. В телеметрии злоупотребления часто можно обнаружить как всплески активности, которые не соответствуют обычным шаблонам использования. Возможно, вы не остановите атаку в режиме реального времени, но зато сможете обнаружить, что вас атакуют.

Для предотвращения злоупотреблений попробуйте сделать следующее:

- увеличить расходы на совершение злоупотреблений;
- снизить привлекательность своей интеллектуальной системы для злоумышленников;
- использовать машинное обучение, но будьте осторожны – нарушители здесь имеют преимущество;
- доверять проверенным пользователям больше, чем новичкам, и удалять всех пользователей, замешанных в подтвержденных злоупотреблениях.

Темы для размышлений

Прочитав эту главу, вы должны:

- знать, кто такие злоумышленники и что они делают;
- уметь вносить простые изменения, которые значительно снижают привлекательность интеллектуальной системы для злоумышленников.

Рассмотрите вашу любимую интеллектуальную систему. Постарайтесь ответить на следующие вопросы:

- какое простое изменение повысит ее привлекательность для злоумышленников?
- какое простое изменение понизит ее привлекательность для злоумышленников?

Глава 26

В шаге от собственной интеллектуальной системы

Спасибо, что читаете эту книгу. Я рад, что вы продвинулись так далеко и заложили фундамент для собственного проекта интеллектуальной системы. Ведь вы уже знаете:

- **с чего начинается работа над интеллектуальной системой** – когда и для чего она нужна, и как установить для нее цели;
- **как создать интеллектуальный опыт** – такой, который помогает достигать заданные цели и собирать данные для обучения и развития интеллекта;
- **что требуется для внедрения интеллектуальной системы** – как выполнять, настраивать и оценивать интеллект;
- **как создать интеллект** – различные подходы, но в первую очередь машинное обучение;
- **что такое оркестровка интеллектуальной системы** – как объединить все компоненты и управлять системой на протяжении всего жизненного цикла.

В этой главе будут рассмотрены некоторые ключевые концепции и приведен контрольный список для пошаговой разработки собственного проекта интеллектуальной системы.

Контрольный список разработчика

Интеллектуальные системы меняют мир, устраняя разрыв между пользователями и интеллектом. Они решают важные проблемы, служат пользователям и повышают производительность организаций.

Чтобы создать интеллектуальную систему, вам нужно сделать много дел и принять множество решений. В этом разделе перечислены наиболее важные моменты и приведены ссылки на главы, где вы найдете более подробную информацию.

При работе над проектом интеллектуальной системы постарайтесь не пропускать важные шаги, перечисленные далее.

Подход к проекту интеллектуальной системы

1. Для начала убедитесь, что интеллектуальная система подходит именно вам.

Интеллектуальная система уместна, если вы готовы к многократному изменению интеллекта вашего продукта в течение его жизненного цикла. Обычно так бывает, когда ваша проблема:

- масштабная;
- открытая;
- меняется со временем;
- внутренне сложная.

Кроме того, интеллектуальная система работает лучше всего, когда:

- частичное решение задачи является жизнеспособным и интересным;
- для улучшения системы вы можете использовать данные о взаимодействии с пользователями;
- вы можете достичь правильно определенную и значимую цель;
- задача оправдывает усилия по созданию интеллектуальной системы.

Более подробно эти темы рассмотрены в главе 2.

2. Определите критерии успеха вашей интеллектуальной системы.

Договоритесь о смысле существования вашей интеллектуальной системы, *направленном на достижимый и измеримый результат*.

Решите, каким *организационным целям* (например, прибыль или количество проданного товара) будет служить ваша интеллектуальная система и какие *опережающие индикаторы* вы будете использовать для мониторинга продвижения к цели. Затем решите, в каком направлении будете ежедневно оптимизировать систему, определив *результаты пользователя*, которые вы хотите получить, и *свойства модели*, которые будут иметь решающее значение для успеха.

Придумайте схему оценки успеха вашей системы, которая может включать в себя *телеметрию, ожидание дополнительной информации, ручную маркировку и опрос пользователей*.

И будьте готовы к новым затратам, чтобы сохранить актуальность целей. Глава 4 более подробно рассказывает о постановке задач для интеллектуальных систем.

Планирование интеллектуального опыта

3. Решите, как представить интеллект вашей системы пользователям.

Интеллектуальный опыт должен выполнять следующие полезные действия:

- представлять интеллект пользователю;
- способствовать достижению целей системы;
- минимизировать недостатки интеллекта;
- создавать данные для улучшения интеллекта.

Чтобы достичь этого, вам необходимо *настроить (сбалансировать) опыт* путем достижения компромисса между следующими факторами:

- *сила* опыта;
- *частота* взаимодействий;

- *ценность* успеха;
- *стоимость* ошибок (как обнаружения, так и исправления);
- *качество* интеллекта.

Настройка интеллектуального опыта – это процесс, в котором опыт меняется по мере изменения интеллекта. Возможно, вы захотите применять итерации. Глава 7 рассказывает о сбалансированном интеллектуальном опыте.

А в главе 8 рассказывается о конкретных режимах взаимодействия между пользователями и интеллектом на основе сбалансированного интеллектуального опыта, включая:

- автоматизацию действий;
- подсказки;
- организованную информацию;
- аннотации.

В своей интеллектуальной системе вы почти наверняка будете использовать все перечисленные режимы, делая опыт более сильным для точного интеллекта и деликатным для ненадежного интеллекта.

4. Запланируйте получение данных из интеллектуального опыта.

Идеальный опыт позволяет получать данные, помогающие совершенствовать интеллектуальную систему (и особенно сам интеллект).

Чтобы быть полезными для улучшения интеллектуальной системы, данные должны:

- включать в себя *контекст*, *действия пользователя* и *результаты взаимодействия*;
- обеспечивать хороший *охват* частей вашей интеллектуальной системы и пространства задачи;
- представлять *реальные взаимодействия* с вашими пользователями;
- не иметь смещения;
- избегать *петель обратной связи*;
- иметь достаточный *масштаб* для создания интеллекта.

Лучше всего, когда эти данные производятся неявно, как естественный побочный продукт пользователей, взаимодействующих с вашей интеллектуальной системой. Но иногда вам нужна явная обратная связь, чтобы понять, как ваши пользователи относятся к полученному результату. Способы получения обратной связи включают в себя:

- разрешение пользователям проставлять *оценки* в опыте (например, большой палец вверх, большой палец вниз или от 1 до 5 звезд);
- получение *пользовательских отчетов* о плохих результатах в работе (например, кнопка «пометить как спам»);
- *эскалацию* проблемы, чтобы получить помощь (например, звонок по телефону);
- предложение пользователям *классифицировать* некоторые из выбранных ими результатов.

Но имейте в виду, что пользователи не захотят тратить много времени и внимания на помощь в создании вашего продукта. Сосредоточьтесь на не-

явных данных и используйте явные данные умеренно. Более подробно об этом говорится в главе 9.

5. **Предусмотрите механизм проверки интеллектуального опыта.**

Вам следует решить, как вы будете:

- узнавать, что опыт работает правильно (несмотря на все ошибки, которые совершает интеллект);
- узнавать, что опыт помогает в достижении целей интеллектуальной системы.

Вы можете разработать специальные инструменты для проверки опыта, работающего в любом конкретном контексте, а также инструменты, помогающие воссоздавать или фиксировать контексты, в которых возникают проблемы.

В этих инструментах можно использовать различные версии системного интеллекта, включая действующий интеллект, снимки интеллекта или какой-нибудь простой или «глупый» интеллект.

Проверке интеллектуального опыта посвящена глава 10.

Планирование внедрения интеллектуальной системы

Внедрение интеллектуальной системы требует проведения всех работ по внедрению традиционной системы, а также дополнительных работ по внедрению интеллектуальных функций. Эти дополнительные компоненты включают в себя:

- *среду выполнения* интеллекта;
- *инструменты управления* интеллектом;
- *телеметрию* для развития интеллекта;
- *среду создания* интеллекта;
- *инструменты оркестровки* интеллекта.

Перечисленные компоненты создают условия для развития интеллектуальной системы в течение срока службы. Грамотное планирование реализации может значительно облегчить создание интеллекта и организацию интеллектуальной системы.

Обзор компонентов реализации интеллектуальной системы представлен в главе 11.

6. **Решите, где будет располагаться ваш интеллект.**

Интеллект может быть реализован в любом из следующих вариантов:

- *статический* интеллект в продукте;
- интеллект *на стороне клиента*;
- интеллект *на стороне сервера*;
- *внутренний (кешированный)* интеллект.

Или, возможно, это будет какой-то гибридный интеллект. Вы должны выбрать расположение интеллекта, исходя из влияния на следующие параметры:

- *задержка в обновлении* интеллекта;
- *задержка выполнения* интеллекта;
- *стоимость эксплуатации* интеллектуальной системы;
- какие-либо требования *автономной работы*.

В главе 13 рассматриваются вопросы, связанные с расположением интеллекта, и обсуждаются факторы, влияющие на принятие правильных решений.

7. Создайте среду выполнения интеллекта.

Среда выполнения интеллекта – это место, где выполняется интеллект и обновляется опыт. Среда выполнения отвечает за следующие действия:

- сбор необходимого *контекста*, чтобы ваш интеллект мог принимать правильные решения;
- *извлечение признаков* из контекста безопасным способом, но с возможностью инноваций;
- работа с файлами *моделей* и их обновление;
- *выполнение* модели в контекстах (точнее, с набором их признаков);
- передача в интеллектуальный опыт *результатов* и *прогнозов*, полученных в результате выполнения.

Среда выполнения интеллекта подробно обсуждается в главе 12.

8. Разработайте средства управления интеллектом.

Управление интеллектом – это подсистема, которая позволяет добавлять новый интеллект в систему и безопасно представлять его для пользователей. Управление интеллектом включает в себя:

- *внедрение интеллекта* в интеллектуальную систему;
- *проверку работоспособности*, чтобы убедиться, что интеллект не является явно вредоносным или поврежденным;
- *объединение интеллекта* при наличии нескольких источников, в том числе проверка их синхронизации. В главе 21 перечислены способы объединения интеллекта;
- *пробный запуск интеллекта* для пользователей. Он может включать в себя «тихий» интеллект, ограниченное развертывание, флайтинг (или A/B-тестирование) и отмену обновлений.

Методы управления интеллектом рассмотрены в главе 14.

9. Разработайте систему телеметрии.

Телеметрия является критически важным звеном обратной связи между пользователем и интеллектом, благодаря чему интеллектуальная система способна постепенно улучшаться. Телеметрия позволяет убедиться в работоспособности интеллектуальной системы, понять результаты, получаемые пользователями, и собрать данные для развития интеллекта.

Обычно в системе больше параметров, чем вы способны измерить, поэтому вам нужно определить масштаб охвата телеметрии. Кроме этого, необходимо иметь инструменты, позволяющие создателям и оркестровщикам интеллекта адаптироваться к изменениям проблемы и интеллектуальной системы. Эти инструменты включают в себя:

- *выборочное наблюдение* и увеличение либо уменьшение частоты сбора данных;
- *подведение промежуточных итогов* (резюмирование), чтобы обеспечить эффективное хранение нужной информации;
- поддержку *гибкого таргетинга* в зависимости от того, из каких контекстов и от каких пользователей поступает телеметрия.

О телеметрии подробно рассказано в главе 15.

Подготовка к созданию интеллекта

Интеллект является частью интеллектуальной системы, которая принимает трудные решения, сопоставляя контексты с предсказаниями, реализуя интеллектуальный опыт и создавая ценность для пользователей и вашей организации.

10. Приготовьтесь оценить интеллект.

Оценка интеллекта – это творчество. Вы должны иметь возможность легко оценить любой интеллект, чтобы знать:

- как хорошо он *обобщает*;
- *типы ошибок,* совершаемых интеллектом;
- *распространение этих ошибок*.

Для этого вам понадобятся оценочные данные, желательно полученные из телеметрии. Для получения статистически значимых оценок данных должно быть достаточно много. Оценочные данные должны быть отделены от обучающих данных (что иногда бывает сложно обеспечить) и должны показывать, как ваш интеллект работает с важными *подгруппами* населения.

Оценка интеллекта обсуждается в главе 19 – самой длинной и важной главе в книге.

11. Спланируйте структурирование интеллекта.

Структурирование интеллекта применяется при разработке больших интеллектуальных систем, когда необходимо:

- *совместно создать интеллект* усилиями нескольких разработчиков;
- *устранить ошибки*, которые совершает интеллект;
- *решить легкие части задачи простыми способами* и применить более сложные методы к сложным частям;
- *включить устаревшие данные* или решения, полученные из внешних источников.

Вот несколько распространенных способов структурирования интеллекта:

- раздельное *извлечение признаков*;
- конурентный *поиск нескольких моделей*;
- *распределение ошибок* (вероятно, худший способ);
- использование *метамодели*;
- *секвенирование модели*;
- *разделение контекстов*;
- использование *переопределений*.

Все перечисленные способы структурирования интеллекта обсуждаются в главе 21.

12. Настройте свой процесс создания интеллекта.

Используйте все доступные инструменты и данные для создания и развития интеллекта, включая следующие:

- выбор *представления* для вашего интеллекта (как описано в главе 17);
- создание *простой эвристики* как основы интеллекта;
- использование *машинного обучения* для создания более совершенного интеллекта (как описано в главе 20);
- использование *компромиссов* различных вариантов реализации;

- *оценку и итерацию*, снова и снова, до тех пор, пока необходимо улучшать интеллект.

В главе 18 приводится пример процесса создания интеллекта.

Управляйте вашей интеллектуальной системой

Вам необходимо взять под контроль все инструменты, упомянутые в этой книге, и «управлять гоночным автомобилем», чтобы ежедневно добиваться успеха, пока работает интеллектуальная система. Хорошо организованная интеллектуальная система будет:

- надежно *достигать своих целей* изо дня в день;
- поддерживать *баланс* между опытом, интеллектом и целью;
- успешно *смягчать ошибки*;
- эффективно *масштабироваться* с течением времени;
- *медленно ухудшаться*.

Оркестровка важна для интеллектуальных систем, когда:

- *меняются цели*;
- *меняются пользователи*;
- *меняется проблема*;
- *меняется интеллект*;
- система должна стать более *эффективной*;
- происходит *злоупотребление*.

Оркестровка системы детально рассмотрена в главе 22.

13. Спланируйте оркестровку на протяжении всего жизненного цикла вашей интеллектуальной системы.

Вы должны решить, с чего начать и куда вкладывать силы и средства с течением времени. В процессе оркестровки интеллектуальной системы вам нужно:

- *контролировать критерии успеха*;
- *проверять взаимодействия*;
- *поддерживать баланс опыта*;
- *переопределять интеллект*;
- *создавать обновленный интеллект*

и многое другое. Глава 23 описывает процесс оркестровки и принятия решений о направлении инвестирования.

14. Приготовьтесь выявить и устранять ошибки.

Ошибки неизбежно приходят вместе с интеллектом.

Вам придется выявлять причины ошибок интеллектуальной системы, в том числе:

- *системные сбои*;
- *отказы моделей*;
- *ошибки интеллекта*;
- *деградацию интеллекта*.

Определив источник ошибок, вы должны решить, как реагировать:

- *инвестировать в интеллект*;
- *настроить значимость опыта*;

- *настроить параметры управления интеллектом*;
- реализовать *ограничения*;
- *переопределить ошибки*.

Локализуя ошибки, заблаговременно обнаруживая их и уменьшая ущерб, который они наносят, вы освобождаете пространство для реализации интеллекта. О борьбе с ошибками рассказано в главе 24.

15. Будьте готовы к борьбе со злоупотреблениями.

Если злоумышленники узнают, как заработать на злоупотреблении вашими пользователями или интеллектуальной системой, они это непременно сделают.

Вы должны быстро реагировать на злоупотребления, выявляя:

- группы пользователей с *ненормальным использованием*;
- контексты, которые показывают *всплески активности*;
- контексты, которые показывают радикальные *изменения в результатах*;
- однотипные *жалобы и сообщения о проблемах*.

Помните, что злоупотребления – это бизнес. Лучше всего не бороться со злоупотреблениями напрямую, а просто сделать этот бизнес невыгодным для нарушителей, и они отступятся от вас. Вот несколько способов добиться этого:

- добавить расходы для пользователей вашей интеллектуальной системы;
- стать менее интересным для нарушителей;
- включить признаки злоупотреблений в модель;
- удалить злоумышленника из промежутка между системой и пользователем.

Глава 25 рассказывает о нарушителях и злоупотреблениях.

16. Наслаждайтесь движением вперед.

И это не конец, а лишь начало. Интеллектуальные системы могут жить много лет.

Привлекайте пользователей. Достигайте целей. Создавайте интеллект. Управляйте системой – и побеждайте!

Итог главы

Спасибо, что прочитали эту книгу. Теперь у вас должны быть знания, с которыми вы уверенно подойдете к проекту интеллектуальной системы.

Я с восхищением предвкушаю, чего вы добьетесь!

Темы для размышлений

Прочитав эту главу, вы должны:

- гордиться усилиями, которые вы вложили в чтение этой книги;
- участвовать в разработке интеллектуальных систем с уверенностью в том, что у вас есть необходимые знания.

Постарайтесь ответить на следующие вопросы:

- какую интеллектуальную систему вы хотели бы построить?
- как это изменит мир?
- что вас останавливает? Вперед!

Предметный указатель

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-planeta.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Джефф Хултен

Разработка интеллектуальных систем

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Яценков В. С.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 23,08. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**