1 Основы проектирования систем машинного обучения

В этой главе

- ✓ Что такое проектирование систем машинного обучения (ML system design), почему сложно дать определение этому процессу и где вы с ним можете впервые столкнуться
- ✓ Кто, по нашему мнению, получит наибольшую пользу от прочтения этой книги, какую информацию мы собираемся вам дать и как она будет структурирована
- ✓ Какие принципы проектирования ML-систем могут быть полезны и когда их лучше всего применять

Проектирование систем машинного обучения (ML system design) — это относительно новый термин, который часто вызывает замешательство. Бывает сложно сформулировать четкий круг обязанностей, стоящих за данным понятием, не говоря уже о попытках подобрать правильное название для соответствующей роли или должности. Эту работу могут выполнять как ML-инженеры, так и программисты-разработчики или даже специалисты data science в зависимости от обязанностей и функций роли, при этом уровень эффективности каждого из них будет разным.

Хотя все эти позиции существуют, мы считаем, что для того, чтобы стать опытным специалистом в проектировании ML-систем, необходимо объединить знания и навыки каждой из этих областей. И хотя некоторые моменты, рассматриваемые

в этой книге, специфичны для ML, другие же будут знакомы и читателям, разрабатывающим программные системы без ML (такая информация представлена в главах 2, 13 и 16). Это связано с тем, что, несмотря на свою новизну, проектирование ML-систем по-прежнему основывается на классических принципах разработки программного обеспечения.

Но прежде чем углубляться в детали, необходимо понять, что такое проектирование систем машинного обучения в целом. В этой вводной главе мы предложим наше определение этого понятия и подкрепим его примерами как из личного опыта, так и опыта наших коллег. Мы также рассмотрим навыки идеального специалиста в этой области и поделимся примерами из практики, демонстрирующими, почему последовательный и согласованный подход к проектированию ML-систем позволяет значительно сэкономить время в долгосрочной перспективе и при этом способствует достижению краткосрочных бизнес-результатов, что особенно важно для завоевания доверия коллег к этому новому методу работы с ранних этапов.

1.1. Что такое проектирование ML-систем?

Это понятие может быть вам знакомо, если вы когда-либо проходили собеседования в Deep Tech или Big Tech компаниях на должность ML-инженера или ML-менеджера. Понятие «Deep Tech» обычно относится к стартапам или R&D-отделам (Research and Development, отдел исследований и разработок) в крупных корпорациях, которые работают с самыми передовыми технологиями. Термин «Big Tech», в свою очередь, характеризует крупнейшие и наиболее влиятельные технологические компании в мире, известные своим высоким уровнем требований к сотрудникам и развитой инженерной культурой. Авторы этой книги имеют солидный опыт работы в сфере Deep Tech, поэтому, загоревшись идеей написать эту книгу, мы были уверены, что определение понятия, вынесенного в заголовок данного раздела, всем хорошо известно и не требует каких-либо дополнительных разъяснений.

Однако после обсуждения концепции книги с несколькими специалистами мы заметили, что интерпретация термина вызывает ряд разногласий. Возможно, это связано с тем, что в индустрии давно сложился четкий перечень должностей, позволяющий кандидатам достаточно точно понимать, какие обязанности и функции предполагает та или иная роль. Такие должности, как инженер-программист (software engineer), инженер-исследователь (research engineer), инженер по машинному обучению (ML engineer) и др., подразумевают выполнение определенного набора задач, описанных в учебниках и четко определенных в описании вакансий.

Отсюда возникает вопрос: существует ли отдельная должность, напрямую связанная с проектированием ML-систем? В настоящее время не существует позиции, полностью охватывающей весь объем работы, который мы будем

рассматривать в этой книге. Но если вы все же встретите специалиста, выполняющего все перечисленные задачи, его должность, скорее всего, будет обозначена как «специалист data science».

Пытаясь разобраться в причинах такой путаницы, мы пообщались с несколькими специалистами, работающими в сфере data science, и пришли к выводу, что эта роль подразумевает широкий и довольно размытый круг обязанностей. Поэтому если спросить десять специалистов data science, работающих в десяти разных компаниях, чем они занимаются, можно получить десять совершенно разных ответов:

- создание сводных таблиц в Excel;
- настройка распределенного кластера объемом в 10 Пбайт;
- разработка системы компьютерного зрения в режиме реального времени;
- развертывание множества чат-ботов;
- визуализация данных в Tableau/Metabase/Looker/PowerBI;
- написание SQL-скриптов;
- проведение A/B-тестирования;
- разработка рекомендательных систем;
- взаимодействие со стейкхолдерами;
- ответы на вопросы высшего руководства.

Как видно из этого списка, короткое и лаконичное название «специалист data science» объединяет в себе довольно разнородный набор задач. Фактически эта должность стала универсальным термином для всех видов деятельности, выходящих за рамки традиционных ролей инженера данных, ML-инженера и инженера-исследователя.

Однако в случае проектирования ML-систем ситуация прямо противоположная: существует нечто, пока не имеющее общепринятого названия, но при этом обладающее четко определенным набором функций и обязанностей. В данном случае наша задача заключается в том, чтобы систематизировать этот набор и представить в виде организованной структуры взаимосвязанных процессов.

В последующих главах мы предложим свой взгляд на проектирование ML-систем и даже представим несколько нестандартных идей и решений. Однако прежде чем углубляться в детали, рассмотрим само определение:

> Проектирование ML-систем — это сложный, многоэтапный процесс проектирования, внедрения и *сопровождения* систем на основе машинного обучения. Он включает в себя сочетание методов и навыков из различных областей, таких как машинное обучение, разработка программного обеспечения, проджект-менеджмент, продакт-менеджмент и управление.

Рисунок 1.1 иллюстрирует это определение.

Рис. 1.1. Навыки, необходимые для успешного проектирования ML-систем

Мы выделили слово *«сопровождение»* курсивом, так как считаем, что проектирование ML-систем не заканчивается на этапе релиза. Помимо обеспечения точности прогнозов и эффективного принятия решений, ваша система должна быть масштабируемой и достаточно гибкой, чтобы легко адаптироваться к изменяющимся бизнес-условиям или другим факторам — как внутренним, так и внешним. Таким образом, сразу после запуска вашей ML-системы в эксплуатацию ее сопровождение и тонкая настройка обеспечат ее эффективность в долгосрочной перспективе, что особенно важно в условиях ограниченного бюджета или небольших вычислительных мощностей.

Однако у тех, кто ознакомился с аннотацией книги или просмотрел ее содержание, возникли вопросы не только относительно самого термина *«проектирование систем машинного обучения»*. Мы также получили множество вопросов, касающихся различных аспектов книги, и ниже приведены наиболее интересные из них:

- «Специалист data science, ML-инженер и инженер-программист — это разные роли. Почему вы их объединяете?»
- «Мне непонятно, зачем книга о ML-системах затрагивает такие темы, как сбор данных и отчетность, ведь именно это отличает классическое машинное обучение от data science».
- «Я удивлен, почему в оглавлении нет упоминания MLOps — распространенного в индустрии термина, включающего в себя многие описанные вами компоненты (воспроизводимость, тестирование, пайплайны и т. д.)».

Для нас эти вопросы стали еще одним подтверждением существования путаницы между ML и data science, а также между ML-инженерами и специалистами data science. У нас есть своя точка зрения относительно этого вопроса, но сначала давайте попробуем прояснить нашу позицию.

Работая в Deep Tech компаниях, мы привыкли называть специалистов, занимающихся ML, *ML-инженерами*. Однако граница между ML-инженером

и инженером-программистом становится все более размытой, особенно с учетом того, что некоторые ведущие специалисты в данной области называют машинное обучение *«программным обеспечением 2.0»* (https://mng.bz/yoNe). В то же время должность *специалиста data science* чаще всего ассоциируется с продуктовыми аналитиками, работающими с метриками, инсайтами и подобными задачами. (Отметим, что мы опираемся на наш опыт работы в Deep Tech компаниях, но поскольку подобные компании нанимают тысячи высококвалифицированных специалистов, тем самым формируя стандарты для всей индустрии, мы рассматриваем их подход в качестве эталона.)

Когда кандидаты претендуют на вакансию ML-инженера, им, как правило, приходится проходить весь стандартный процесс отбора для инженеров-программистов, включающий дополнительные этапы. Ключевой из подобных этапов — проектирование ML-системы. Он нужен для того, чтобы оценить уровень навыков кандидата, его компетентность, а также способность анализировать сложные системы и декомпозировать их на взаимосвязанные блоки задач. Это непростое испытание, поскольку у соискателя есть всего 40–45 минут, чтобы представить свой проект системы, случайно выбранной экспертом, проводящим собеседование.

Элиезер Юдковский (Eliezer Yudkowsky), писатель и философ в области ИИ, писал: «Самая опасная привычка мышления, прививаемая еще в школе, состоит в том, что даже если вы чего-то не поняли, то все равно должны это повторить, как попугай» (https://mng.bz/r1Zy). Данное утверждение хорошо применимо к процессу технических собеседований в некоторых компаниях: интервьюер дает задачу и ожидает, что соискатель просто повторит заранее известный ответ. После того как интервьюируемый получает работу и затем сам становится интервьюером, эта вредная практика закрепляется и компания продолжает нанимать людей с идеально заученными фрагментарными знаниями из разных областей. Нет никакой гарантии, что эти люди понимают общую картину, и мы сами столкнулись с этим, когда проводили собеседования.

Мы проводили собеседования и нанимали ML-инженеров для различных компаний. Среди них были как новички, так и опытные специалисты и сильные инженеры-программисты, решившие перейти в ML. И у всех тех, кто не прошел собеседование, наблюдалась одна общая черта — в процессе проектирования ML-системы они зацикливались на деталях, не видя общей картины.

Для нас такое поведение указывало на несоответствие ожиданий: будучи молодыми менеджерами по найму, мы были уверены, что человек, знающий все алгоритмы, инструменты и шаблоны, автоматически будет нам подходить. Однако со временем мы поняли, что иногда люди просто не могут объединить все свои знания в какую-то цельную картину.

Кроме того, разработка системы в реальных условиях кардинально отличается от ее обсуждения на собеседовании. Можно заучить ответы на десятки популярных вопросов по проектированию ML-систем («Как бы вы спроектировали систему

рекомендаций вакансий для сайта наподобие LinkedIn?») и все равно не справиться с аналогичной задачей в реальной работе.

Но давайте на время оставим тему собеседований. Специалистов по ML нанимают не просто так — компании нуждаются в них для создания, сопровождения, эксплуатации и улучшения систем, а не ради написания кода или закрытия тикетов в Jira. Бизнесу требуются надежные ML-системы для достижения определенных целей и решения конкретных задач.

Разработка ML-систем требует широкого набора навыков. Говоря кратко, человек, ответственный за это, должен быть способен ответить на три вопроса:

1. Что мы создаем?
2. Какова цель системы?
3. Как она должна быть построена?

На практике это требует сочетания навыков из нескольких областей: частично продакт-менеджмента — чтобы понимать основную цель и уметь донести ее до коллег и стейкхолдеров, частично ML-исследователя — для усиления системы и, конечно, создания прочного ПО-фундамента, чтобы продукт был удобным, сопровождаемым и надежным. Эксперт по проектированию ML-систем должен мыслить масштабно и, при необходимости, детально разбираться в отдельных аспектах.

Мало кто обладает всеми этими навыками на должном уровне. Однако сегодня создается множество ML-систем, а значит, их кто-то проектирует. Исходя из нашего опыта, проектирование ML-систем обычно поручается либо сильному специалисту в ML (потому что это ML), либо опытному инженеру-программисту (потому что это система). Они выполняют поставленные перед ними задачи, но зачастую сталкиваются с трудностями в тех областях, где не являются экспертами.

Подводя итог, можно сказать, что вся путаница вокруг проектирования ML-систем чаще всего возникает у кандидатов с недостаточным опытом, а также у менеджеров по найму или рекрутеров, которые ищут универсального специалиста. Однако если посмотреть на ситуацию с точки зрения руководителя или эксперта, картина становится гораздо шире. Они понимают, что специалистов в этой области нанимают для создания, сопровождения и улучшения ML-систем, и результат их работы с этими системами становится главным критерием их карьерного роста.

Мы считаем, что эксперт в области проектирования систем машинного обучения сочетает навыки специалиста по data science и инженера-программиста с опытом работы в академической сфере ML. Сотрудники, занимающиеся проектированием ML-систем, могут иметь различный профессиональный бэкграунд — разработка программного обеспечения, прикладное ML, академическое ML, исследование данных и т. д. Мы надеемся, что наш практический опыт

в сочетании с небольшими теоретическими вставками поможет таким специалистам устранить пробелы в знаниях, систематизировать их и почувствовать себя увереннее в тех сферах, где им не хватает опыта.

1.1.1. Почему проектирование ML-систем так важно

Хотя MLOps предоставляет набор инструментов для создания и сопровождения ML-систем, их проектирование можно рассматривать как фундаментальную архитектурную схему, на которую всегда можно опереться и которая обеспечивает необходимую масштабируемость и гибкость (глубокое понимание строительных блоков системы и их взаимосвязей помогает выявлять возможные проблемы и эффективно их решать). Однако самое главное — она задает фреймворк, связывающий всю систему воедино.

Некоторые проекты достаточно просты и не требуют такого скрупулезного подхода. Проведем аналогию со строительством: скорее всего, сарай можно построить и без предварительного плана. Но если речь идет о доме или небоскребе, обойтись без заранее подготовленного детализированного проекта уже невозможно. Проектирование ML-систем — это архитектурный подход к инженерии ML-систем, объединяющий опыт сотен специалистов, работавших в десятках компаний над множеством различных проектов.

1.1.2. Истоки проектирования ML-систем

Разработка сложных программных систем всегда была непростой задачей, и компании были вынуждены как-то структурировать этот процесс. В основе управления сложностью лежит принцип абстракции — сначала создаются низкоуровневые модули с инкапсулированной в них сложностью, затем они используются как «черные ящики» при построении более высокоуровневых компонентов, и т. д.

Такой метод работал, но имел уязвимое место — кто-то должен был определить структуру всех этих компонентов (какие компоненты являются высокоуровневыми, какова их внутренняя организация, какие существуют уровни детализации до самого низкого уровня реализации). Наиболее ответственные решения принимали архитекторы ПО — опытные инженеры, работавшие с множеством различных систем.

Такой подход традиционно ассоциируется с методологией Waterfall («водопадный подход») и парадигмой Big Design Upfront («глобальное проектирование прежде всего»). Другими словами, он предполагает, что разработка ПО начинается постепенно и проходит тщательный анализ и документирование еще до написания первой строки кода. Данный метод зарекомендовал себя как весьма надежный, но в то же время инерционный и бюрократический. В условиях быстро развивающегося мира проект может потерять актуальность еще до своего завершения.

Противники такого медленного, но основательного подхода — сторонники гибкой методологии разработки программного обеспечения. Авторы *Agile-манифеста* (Manifesto for Agile Software Development) (https://agilemanifesto.org/) сформулировали четыре ключевые ценности:

- Люди и взаимодействие важнее процессов и инструментов.
- Рабочий продукт важнее подробной документации.
- Сотрудничество с заказчиком важнее согласования условий контракта.
- Готовность к изменениям важнее следования первоначальному плану.

Другими словами, сторонники данного метода вполне обоснованно утверждают, что многие программные системы не могут быть эффективны, если их создатели пытаются заранее спланировать и задокументировать абсолютно всё. Конечно, в некоторых случаях такая бюрократия оправданна, например, при разработке ПО для управления медицинскими приборами или самолетами. Однако большинство программистов работают над другими типами приложений — офисным ПО, развлекательными сервисами, веб-сайтами и мобильными приложениями. Именно поэтому роль архитектора ПО стала ассоциироваться с чем-то медленным и устаревшим — полной противоположностью быстрым хакерам, которые меняют мир, не дожидаясь утверждения технической спецификации всей иерархией архитекторов, менеджеров и других экспертов. Такой гибкий подход приобрел популярность после успеха компаний Кремниевой долины и тысяч других стартапов. Даже такие крупные компании, как Meta, стараются придерживаться подобной культуры — их корпоративный девиз гласит: «Действуй быстро, ломай все» (Move fast and break things).

Подведем итог этому небольшому историческому обзору: в какой-то момент индустрия столкнулась с двумя крайностями в разработке ПО — от строго регулируемого, управляемого архитекторами процесса до хаотичного, анархического стиля «к черту иерархию». И как это часто бывает, постепенно два подхода начали перемешиваться. И теперь более традиционные компании стремятся к максимальной гибкости, а анархичные стартапы со временем взрослеют, внедряя процессы и вводя отдельные роли.

Такая смесь привела к консенсусу, который на данный момент установился в технологических компаниях: вместо того чтобы передавать все решения отдельным специалистам, таким как, например, архитекторы ПО, ответственность за проектирование возлагается на рядовых инженеров-разработчиков. Они не только пишут код, но и проектируют системы, над которыми работают. Однако такая свобода не устранила изначальную потребность в четких архитектурных решениях — кто-то все равно должен принимать окончательные решения о структуре системы и нести ответственность за ее проектирование. Конечно, каждый инженер может в той или иной степени участвовать в этом процессе, но важно, чтобы кто-то видел всю картину целиком.

Навыки реализации низкоуровневых компонентов системы не равны навыкам проектирования правильной архитектуры. Именно поэтому в Deep Tech компаниях собеседования обычно состоят из двух частей: проверки навыков написания эффективного кода (этап, посвященный алгоритмам) и проектирования систем. Ожидается, что инженеры будут совмещать обе эти роли. Разделение между ними может быть разным: обычно джуниор-разработчики лишь следуют дизайн-документам, тогда как сеньоры становятся их авторами или активными участниками.

Так что общий консенсус таков: хороший инженер-разработчик способен делать все — от работы над низкоуровневой реализацией до принятия высокоуровневых архитектурных решений.

Все, что мы до сих пор говорили о проектировании систем, применимо к любому программному обеспечению — пока что мы не затрагивали ничего, связанного непосредственно с машинным обучением. Однако не каждый, кто умеет проектировать программные системы, способен успешно спроектировать ML-систему, поскольку это особый подкласс систем. При их проектировании специалисту необходимо учитывать множество аспектов, не имеющих значения в традиционной разработке ПО. В этой книге мы сосредоточимся именно на них, поэтому тем, кого интересуют более общие вопросы проектирования систем, лучше обратиться к другой литературе.

1.2. Структура книги

Существует множество книг, посвященных проектированию программных систем, но вот литература по проектированию ML-систем встречается довольно редко. Мы решили внести свой вклад в эту область и восполнить разрыв между спросом и предложением. Наша цель — поделиться своими знаниями и опытом и собрать воедино все разрозненные сведения.

Эта книга представляет собой всестороннее практическое руководство по построению сложных и полноценно функционирующих ML-систем в различных сферах, независимо от размера компании, в которой вы работаете. В это руководство входят:

- общий обзор, включающий основные принципы структурирования, описание всех компонентов, из которых состоят такие системы, а также подводные камни, с которыми вы можете столкнуться;
- набор низкоуровневых чек-листов с перечнем инструментов, которые могут пригодиться на каждом этапе, и кратким пояснением их значимости.

Структура книги напоминает некий чек-лист или техническое руководство, дополненное историями из нашего собственного опыта. Ее можно сразу прочитать от начала до конца или же использовать в качестве справочника в процессе работы над конкретным аспектом ML-системы.

Каждая глава, в свою очередь, представляет собой высокоуровневый чек-лист, обязательный для любой ML-системы. При этом не все пункты обязательны для каждого проекта, но о каждом из них необходимо помнить и учитывать в процессе разработки.

Кроме того, в каждой главе рассматривается, почему и в каких ситуациях тот или иной аспект имеет критическое значение. Вы также найдете систематизированное описание доступных методов и инструментов, подходящих для решения поставленных задач (а не просто перечисление модных терминов). И хотя наше описание не претендует на исчерпывающую полноту, мы уверены, что читатели смогут сопоставить предложенные примеры со своим профессиональным опытом и сделать соответствующие выводы. При этом мы стараемся не превращать книгу в типичный учебник или курс по классическому машинному обучению или глубокому обучению.

Мы пришли в область ML из разных профессиональных сфер и поэтому отлично дополняем друг друга. В совокупности у нас более 20 лет опыта работы над проектами ML в самых разных ролях, компаниях и средах — от молодых стартапов до международных корпораций с многомиллиардными оборотами. Иногда мы работали как индивидуальные контрибьюторы, иногда же больше сосредоточивались на развитии команды и наставничестве молодых инженеров. Нам довелось стать свидетелями (и участниками) как успешных запусков, так и неудач, крупных поглощений и массовых сокращений. Конечно, мы также часто обсуждали успехи и провалы ML-проектов с друзьями и коллегами.

Однако, несмотря на различия между нашими карьерами, в одном мы абсолютно согласны — ML-проекты почти никогда не терпят неудачу из-за того, что их участники не умеют применять алгоритмы. Причины провалов могут быть совершенно другими: неправильно поставленная или вовсе ненужная задача, небрежная работа с данными, нерентабельное и неспособное к масштабированию решение — список можно продолжать бесконечно.

Существует одна распространенная закономерность, из-за которой нам придется повторять некоторые истории в разных частях книги: узкий специалист глубоко погружается в свою область — возможно, осваивает смежные сферы, но все равно не видит полной картины. В результате упускаются важные нюансы, что приводит к провалу проекта, срыву сроков и превышению бюджета.

В то время как большинство книг по машинному обучению предлагают «правильные» ответы, наша главная цель совершенно иная. Мы хотим научить вас задавать правильные вопросы — себе, своей команде, пользователям, стейкхолдерам — кому угодно. Каждый специалист в технологической отрасли накапливает огромное количество ценной информации, но не всегда способен связать ее воедино. Именно на этом этапе вовремя заданные вопросы помогают структурировать имеющиеся знания.

Мы разделили книгу на четыре основные части, чтобы ее структура соответствовала жизненному циклу любой системы: исследование, создание базовой версии, улучшение и сопровождение.

Первые две части посвящены ранним этапам проектирования систем машинного обучения. В первой части мы сосредоточимся на общем понимании проблемы, которую должна решить система, и определим шаги, которые необходимо выполнить до начала разработки. Данный этап редко связан с написанием кода и чаще фокусируется на создании небольших прототипов или доказательствах концепций. Вторая часть погружается в технические детали начального этапа работы. Этот этап требует значительного объема исследований, которые критически важны для понимания проблемы, определения возможных решений и согласования ожиданий между участниками проекта. Если сравнивать ML-систему с человеческим телом, то на этом этапе формируется ее скелет.

Третья часть посвящена промежуточным этапам жизненного цикла системы, когда работа инженеров зачастую кардинально меняется — становится меньше исследований и обсуждений и больше практической работы по реализации и улучшению системы. Здесь мы сосредоточимся на вопросах, касающихся того, как сделать систему надежной, точной и устойчивой. Продолжая метафору с человеческим телом, на данном этапе система наращивает мышцы.

Заключительная часть посвящена интеграции и развитию системы. Для неопытного наблюдателя может показаться, что система уже полностью готова к работе, но это впечатление обманчиво. Существует множество (в основном инженерных) аспектов, которые необходимо учесть, чтобы система успешно запустилась в эксплуатацию. В мире ПО сбой системы редко становится катастрофой, как, например, происходит в строительстве, но этого все равно хочется избежать. На этом этапе вы узнаете, как сделать систему надежной, удобной в сопровождении и готовой к будущим изменениям. Если вы еще не устали от метафор, связанных с человеческим организмом, то можно сказать, что здесь система обретает разум, ведь необузданная сила без контроля может привести только к разрушению.

В целом первые главы содержат более общую информацию, которая, тем не менее, крайне важна для правильной постановки проблемы и формирования базовых принципов построения эффективной ML-системы. Однако по мере изучения книги материал будет становиться все более сложным и подробным, включая практические примеры и упражнения. Начиная со следующей главы, мы представим два вымышленных кейса, радикально отличающихся друг от друга, которые будут сопровождать нас на протяжении всей книги. Обе ситуации требуют разработки ML-системы для решения ряда задач, и обе будут развиваться по мере изучения материала.

В каждой части книги мы всегда отдаем предпочтение интуитивному пониманию, а не охвату как можно большего объема материала. Существует множество аспектов построения ML-систем, и каждый из них заслуживает отдельной книги.

Однако мы не планируем писать отдельное издание о сборе и подготовке данных, еще одно — о генерации признаков и еще одно — о метриках. Вместо этого мы описываем верхушку айсберга и анализируем общую структуру ландшафта, подкрепляя наши идеи и аргументы ссылками на значимые научные работы. Это позволяет читателям как ознакомиться с примерами более высокого уровня, так и дополнить предложенный каркас собственными знаниями. Кроме того, мы не стремимся объяснять технические детали, связанные с конкретными библиотеками или вычислительными движками. В некоторых главах мы упомянем ключевые примеры, но исключительно в иллюстративных целях для демонстрации более высокоуровневых абстракций.

Реальные системы всегда сложнее, чем примеры, которые можно встретить в блогах, на конференциях и, конечно же, на собеседованиях. Во всех этих сценариях обычно обсуждают высокоуровневые абстракции, но на практике именно детали играют ключевую роль. Вот почему мы считаем, что интуитивное понимание методов решения проблем так важно: успешный разработчик ML-систем должен не только знать готовый «рецепт» из учебника и воспроизводить его, но и уметь адаптироваться к специфике компании, что порой может полностью изменить правила игры.

Мы надеемся, что эта книга будет полезна:

- тем, кто готовится к собеседованию на позицию ML-инженера или ML-менеджера;
- разработчикам программного обеспечения, техническим менеджерам и специалистам в области ML, работающим с существующими системами и желающим их понять или улучшить;
- тем, кто планирует разработать собственную ML-систему или уже создал ее и хочет убедиться, что не упустил ничего критически важного.

В силу изложенной выше философии, эта книга не предназначена для начинающих. Мы ожидаем, что наши читатели уже знакомы с основами машинного обучения (например, способны понять учебник по ML для студентов бакалавриата) и обладают опытом прикладного программирования (например, сталкивались с реальными задачами разработки за пределами учебных курсов). В противном случае лучше сначала изучить базовый материал.

1.3. Где могут быть полезны принципы проектирования ML-систем

Как уже упоминалось ранее, применение данных принципов имеет решающее значение при построении системы, достаточно сложной, чтобы иметь множество возможных сценариев отказа. Проигнорировав эти принципы, можно создать систему «на глиняных ногах», то есть такую, которая на данный момент работает,

но при этом недостаточно устойчива, чтобы выдержать возможные изменения. Причины таких изменений могут быть сугубо техническими (что, если объем данных увеличится в 10 раз?), связанными с продуктом (как адаптироваться к новым пользовательским сценариям?), продиктованными бизнесом (что, если систему предстоит интегрировать в сторонний программный стек после поглощения компании?), юридическими (что, если государство введет новые нормы регулирования персональных данных?) или носить иной характер. Последние годы лишь подтвердили — предугадать каждый возможный риск невозможно.

Еще более важно уметь улучшать систему. Как будет подробно рассмотрено в последующих главах, создание систем с нуля — довольно редкое явление. Люди, далекие от индустрии, могут полагать, что разработчики ПО большую часть времени пишут код, но на практике, как вам хорошо известно, гораздо больше времени уходит на *чтение* кода. То же справедливо и для систем: обычно усилия концентрируются не на создании новых решений, а на улучшении и сопровождении уже существующих (что требует глубокого понимания внутреннего устройства системы).

Разница между улучшением и сопровождением может показаться размытой. Поэтому для ясности под *улучшением* (improvement) мы будем подразумевать добавление нового функционала или существенные изменения существующего, тогда как *сопровождение* (maintenance) — это обеспечение работоспособности существующего функционала в условиях постоянно изменяющейся среды (новые пользователи, новые датасеты, развитие инфраструктуры и т. д.).

Некоторые принципы, представленные в этой книге, в первую очередь ориентированы на улучшение ML-систем. Они помогают выявить слабые места, точки роста системы, а иногда даже открывают возможности для новых способов ее применения.

Часть принципов относится скорее к сопровождению ML-систем. Печальная истина состоит в том, что зачастую сопровождением систем занимаются команды, которые не участвовали в их создании. Это палка о двух концах — команда, создающая систему, должна помнить о некоторых принципах, чтобы облегчить жизнь своим «последователям», а команда сопровождения должна понимать эти принципы, чтобы в разумные сроки разобраться в логике системы и найти подходящие решения, позволяющие поддерживать ее работоспособность в течение длительного времени.

Можно с уверенностью сказать, что практически 100 % проектов в области машинного обучения, у которых не было грамотно составленного дизайн-документа, потерпели неудачу. В то же время подавляющее большинство тщательно спланированных систем достигли успеха. Дизайн-документ вовсе не обязательно должен быть сложным и состоять из множества страниц, зачастую достаточно нескольких страниц со сжатой, но полезной информацией. В таком случае документ выполняет две ключевые функции: он не только помогает правильно расставить приоритеты в рамках проекта, но и позволяет понять, нужен ли этот

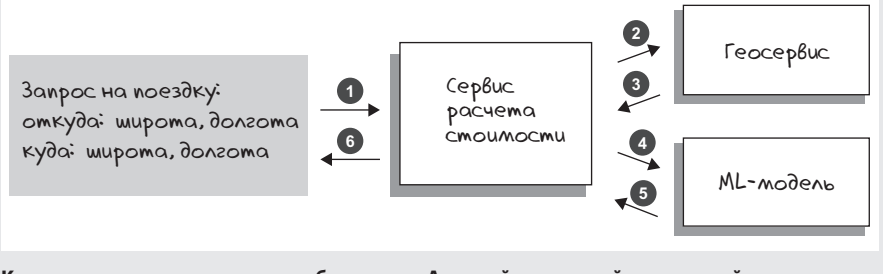
проект вообще, отвлекая ваше внимание от основной идеи (например, когда вы чрезмерно зациклены на самом проекте) и помогая взглянуть на ситуацию шире. Подробности приведены в главе 4.

Поработав в разных компаниях, мы с уверенностью можем утверждать: наличие структурированной документации, описывающей все аспекты функционирования вашей системы, в разы ускоряет любые процессы — от онбординга новых сотрудников до внедрения ключевых изменений. Вместо того чтобы искать единственного «хранителя знаний», в голове которого сосредоточена вся информация (причем без гарантий точности), вы просто обращаетесь к нужному документу в библиотеке.

Байка от Арсения

Давным-давно я работал в сервисе заказа такси. Одним из амбициозных проектов компании было создание системы для расчета стоимости поездки. Обычная модель ценообразования представляла собой классическую схему, используемую в «олдскульных» такси: стоимость = X * время + Y * расстояние. Компании было необходимо научиться оценивать стоимость поездки до ее фактического начала, чтобы информировать как водителя, так и пассажира.

Изначально задача казалась предельно ясной. Все, что нам нужно было сделать, — обучить простую модель, использующую геолокацию из картографического сервиса, и обернуть ее в микросервис. Проект казался настолько элементарным, что я даже не стал писать дизайн-документ.

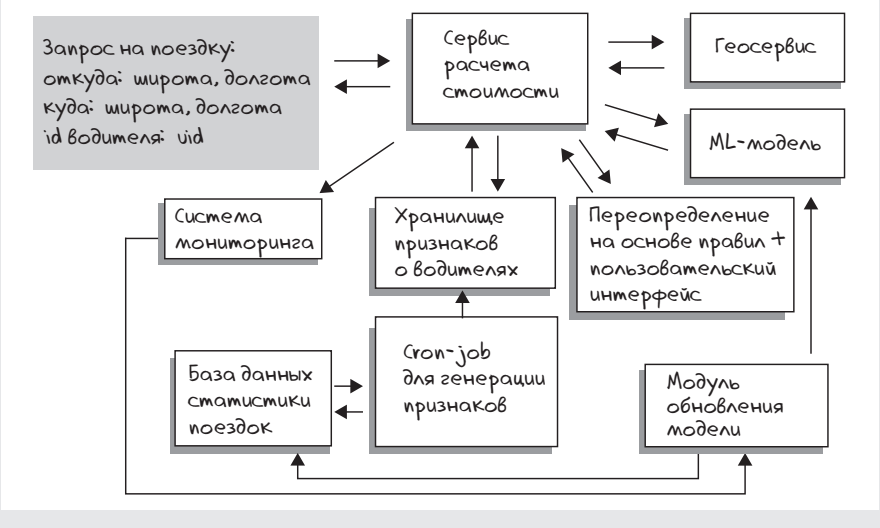


Как изначально представлял себе систему Арсений — простой пошаговый алгоритм

На практике же возникло множество подводных камней (большинство из них мы рассмотрим в следующих главах):

- Геолокации оказалось недостаточно для точной оценки, а более сложные метрики требовали развитой инфраструктуры (так называемого *хранилища признаков*, хотя в то время такой термин и подход еще не были популярны). Этому будет посвящена глава 11.
- По мере усложнения модели ее прогнозы становились менее надежными (определенное количество результатов оказывались выбросами, то есть значениями, которые были слишком высокими или слишком низкими).

- Ошибки распределялись неравномерно, поэтому модель была предвзятой. Мы затронем эту тему в главе 9.
- Руководство время от времени хотело вручную изменять расчеты стоимости поездки, используя акции или эвристики. Данная тема обсуждается в главе 13.
- Было потрачено слишком много времени на построение модели, которая на самом деле не решала поставленную задачу. Мы возвращаемся к этому вопросу в главе 2.
- Проблема в целом была связана с неравномерным распределением ресурсов и, следовательно, требовала тщательного мониторинга. Подробнее мы рассмотрим эту тему в главе 14.
- Инфраструктура не была готова к предполагаемому сценарию, что привело к неприемлемым задержкам в часы пик. Поговорим об этом в главе 15.
- Некоторые другие команды не знали о том, что разрабатывается такая система, это привело к несовместимостям в API. Данная тема рассматривается в главе 16.



Как система выглядела после нескольких итераций

В итоге система так и не была развернута — до того, как все проблемы были устранены, ситуация на рынке существенно изменилась и потребность в исходной системе исчезла. Хотя изначальная идея была хорошей (некоторые конкуренты реализовали похожие решения), мои коллеги и я так и не смогли воплотить ее должным образом: были упущены ключевые аспекты — как технические, так и продуктовые, — и всплыли они лишь на поздних стадиях проекта, когда любые изменения были слишком затратными. При этом если бы эти аспекты были учтены на более ранних этапах, их решение оказалось бы элементарным. Если бы кто-то из нас — я, мой руководитель или члены команды — прочитал такую книгу, как эта, мы, возможно, избежали бы провала.

Тем не менее на несколько историй провала всегда находится своя история успеха. Следующий случай менее драматичен и может даже показаться скучным, но для баланса его все же стоит упомянуть. В свое время, когда Валерий работал в российском технологическом гиганте «Яндекс», компания приобрела стартап, занимавшийся предоставлением рекомендаций в реальном времени. При слиянии компаний требуется время для настройки взаимодействия между новыми и существующими подразделениями, онбординга новых сотрудников, синхронизации бизнес-процессов и т. д. Однако Валерий был поражен тем, насколько гладко и беспроблемно произошло интегрирование нового бизнеса в масштабную корпорацию. Причина заключалась в грамотно составленном дизайн-документе, благодаря которому такой переход стал возможным.

Подводя итог: мы искренне убеждены, что подготовка дизайн-документа — с постановкой правильных вопросов бизнесу и формулировкой корректных целей — является ключом к успеху при разработке ML-системы. А в ряде случаев это даже может стать основанием для отказа от проекта на ранней стадии, что само по себе является положительным результатом, учитывая, сколько времени, сил и денег можно сэкономить, если отказаться от неудачной идеи. Данному этапу проекта мы посвятим как минимум три главы ввиду его критической важности.

Итоги

- Несмотря на относительную новизну термина, проектирование ML-систем опирается на классические основы разработки программного обеспечения и включает в себя уже существующие знания из смежных дисциплин. В этой книге мы постараемся преобразовать эту базу знаний в набор рабочих алгоритмов.
- В то время как MLOps можно рассматривать как набор инструментов для создания и сопровождения ML-систем, проектирование ML-систем — это фреймворк, который объединяет все воедино.
- Для успешного проектирования ML-систем необходимо одинаково хорошо разбираться в таких областях, как машинное обучение, разработка ПО, проджект-менеджмент, продакт-менеджмент и управление людьми.
- Прежде чем приступить к проектированию ML-системы, вы должны четко понимать, что именно вы создаете, с какой целью и каким образом ваша система будет реализована.
- Основой успешно спроектированной ML-системы является согласованный подход, продуманный план работы и список предварительных действий, которые организуют вашу деятельность и сэкономят время в долгосрочной перспективе.