Data Engineering

Page • 1 backlink • uni

Любая модель может обучиться на бесконечном количестве данных, но сложность как раз в том, что у нас на руках всегда конечное количество

Есть тенденция к росту числа данных и вычислительных мощностей – если у нас это все есть, получаем GPT-4

Предсказывают, что с текущей тенденцией модели будут обучаться на всем Интернете в 2028 году

Типичные вопросы

- как хранить датасеты
- как обрабатывать датасеты перед обучением
- как обучаться в реальном времени
- как анализировать датасеты
- что делать (?)
- сколько нужно железа – очень значимый вопрос, нужно доказывать

Прочие

- какой бюджет на вычислительную сложность модели
- как данные попадают в модель
- как готовить данные
- что можем позволить в runtime
- как отдавать данные в модели
- сколько закупать железа

Почему важен Data Engineering? Обычная проблема: по метрикам все хорошо, а пользователь жалуется. Почему? Модель переобучилась – случилась учетка обучающегося датасета в тестовый, поскольку перекладыванием данных занимается другая команда, ошибку тяжело будет выследить

Важная задача: моделирование поведения пользователей? Для того, чтобы оценивать качество наших решений. При сэмплировании сигналов важно учитывать возможные зависимости в выборках, а

также факторы вроде регионоспецифичности (хотим статитстику в среднем по популяции? на отдельном срезе?)

Литература:

- Fundamentals of Data Engineering – больше теории
- Designing Data-Intensive Applications – больше разработки

Основные сценарии:

- доставка данных от пользователя до пользователя
- доставка данных от пользователя до обучающего датасета

Стейкхолдеры

Моделирование данных

- какая схема таблицы? как разложить данные? – физический уровень
- какие сущности в данных? какие бизнес-правила? как связаны таблички? – логический уровень
- схемы продажи? приложение с точки зрения бизнеса – бизнесовый уровень

Дата-инженер общается с внешними пользователями (например, с толокерами) и с внутренними пользователями (например, с командой дата-саентистов)

Создают инфраструктуру и данные для дата-инженера: программные инженеры, дата саентисты, DevOps и SREs

Получают данные от дата-инженера: аналитики, ML-инженеры и (?)

System Stakeholder

Data Stakeholder – борьба с утечкой данных, проработка использования конфиденциальных данных и прочее

Абстракция хранения данных:

- уровень "сырых" компонент – HDD, SSD, RAM, networking, сериализация, сжатие, CPU
- системы хранения данных – HDFS, RDBMS, cache/memory-based, object storage, streaming storage
- (?)

Как выбрать storage?

- подходит ли оно под архитектуру с точки зрения скоростей read/write и storage capacity
- какой компонент будет ботлнеком в хранилище

- сможем ли масштабировать и как, сколько нужно будет средств (придет продакт и спросит "нам нужно столько-то RPS")

Виды хранилищ

- файловые хранилища: открытый доступ, простой интерфейс, просто применять в работе, но работает не быстро и размеры файлов не регулируются (document management, collaborative tools, data analytics)
- блоковые хранилища: храним больше блоки разнородных данных, быстрый доступ, тяжело привести в человекочитаемый вид, high-performance и low-latency (large databases, VMs, HPC, CDN) – самый низкоуровневый примитив, два других, как правило, строятся поверх него
- объектные хранилища: отсутствие иерархии, работаем с данными с помощью метаданных, доступ через API, быстро индексируется, хорошо масштабируется (Internet-of-Things data management, cloud storage systems)

Блоки данных напрямую маппятся на диск, важно понимать, какая скорость работы у нашего диска, какая нужна скорость доступа и надежность (до сих пор используют пленку, как раз из-за последнего)

На данный момент, NVMe SSD открывает большое количество архитектурных возможностей

Как работает SSD: куча блоков по несколько килобайт, процессор через шину прямо на физическом уровне считывает данные из нужного ему блока, доступ осуществляется через контроллер, при котором есть буфер, кэширующий популярные блоки (LRU)

Корректный подбор размера эмбеддинга, в частности, может опираться на размер блока

Важно уметь считать, эмбеды какого размера мы вообще можем использовать – разница в 2 раза линейно повлияет на время исполнения запроса и на пользовательский опыт

Storage tiers

- mission critical
- hot data
- warm data
- cold data

Под каждый тир подбираются свои хранилища

Blob storage (Redis – самый известный) – может быть на диске, может быть in-memory

File storage / object storage – S3

Какие нужны файлы? Текстовые или бинарные? Как правило, для HPC всегда используем бинарные

На данный момент принято хранить все данные в таблицах:

Файлы нужно изменять согласованно, разные данные могут быть связаны между собой

Важная проблема при работе с таблицами – это эволюция схемы: на примере схемы <class, split, feat_1, feat_2, ...> – в процессе жизненного цикла фичи передстают быть доступны, появляются новые, изменяется id

В зависимости от задач необходимо прорабатывать степень нормализации данных (степени нормализации – теория, но самое важное – степени уменьшают redundancy)

Уменьшение redundancy ускоряет JOIN, но это все равно очень ресурсоемкая и долгая операция, не нужно писать ее каждый раз (грамотный выбор базы данных помогает решить эту проблему)

JOIN оцениваются по количеству вычислений

Некоторые вариант:

- с помощью merge sort
- с помощью хэширования

Сейчас стараются реализовать работу с базами данных с помощью промптов – SQL-подобные языки, а современные БД позволяют делать UDF (увеличивает расширяемость)

Хранение данных:

- строчное (для разработчиков)
- столбцовое (для анализа)
- блочное колоночно-столбцовое (редко встречается)

Влияет на время доступа для разных запросов

Pandas – колоночная, numpy – строчное-столбцовая (матрица)

OLTP vs OLAP: транзакционная и аналитическая обработка

Первое – поменять фамилию и прочее (важна консистентность), второе – посчитать статистики

В OLTP важен ACID: atomicity, consistency, isolation (независимость транзакции), durability, делаем insert, update, delete и join

В OLAP (Clickhouse и Amazon) часто встречаются условные клозы, чаще всего делаем select-ы, а join-ы делать не любят (как минимум, в случае

онлайн-вычислений)

Проблемы таблиц: плохо моделируют социальные сети (есть базы данных для графов, но с ними "туго")

Базы для хранения временных рядов: монотиринги (Grafana)

Вообще данные бывают

- структруированные – четкая схема, быстрый поиск и доступ (data warehouse здесь же)
- неструктурированные – "свалили в кучу", быстро складываем

Что мы трансформируем? Важно понимать, какие у нас источники данных (есть готовые коннекторы для перекладываний)

- как поставляются данные
- как часто случаются ошибки
- будут ли дубликаты
- с какой нагрузкой готов справиться источник

И так далее

Ingestion: для каких целей мы потребляем данные? Куда отправляются данные? Какой объем и формат?

Transformation: Extract, Transform, Load – чаще всего делается в батчах

Ingestion frequencies: batch, micro-batch (?)

User Access: Single vs Multitenant (для последнего актуальны всякие SQL-инъекции)

Пример: доставка логов для обучения моделей

User → App → (доставка может стать узким местом) Log → Kafka → DWH → таблица → разметка → обучение → модель → деплой в App

Пример: структура хранения аудио для разметки в системах распознавания речи

Таблица <id, audio, user, lang>, <chunk, parent_id на 1, id>, <text, parent_id, id на 2, noised audio>, <id в 1, tag>

Аналитик джойнит последнее с первым, ему складывается куча аудио, которая занимает compute

Пример: online predictions