Модуль 4. Управление данными.

Образовательный результат: в результате прохождения данного модуля слушатели научатся:

1. формулировать задачи для каждого этапа обработки данных в проекте машинного обучения, в том числе:
   1. сбор,
   2. препроцессинг (предварительная обработка данных),
   3. передача,
   4. сохранение,
   5. разведочный анализ данных,
   6. выбор признаков для обучения модели и задачи.
2. выбирать инструменты для решения этих задач,
3. применять инструмент dvc для решения практических задач для управления данными.

# В этом модуле:

Данные являются важной частью проекта машинного обучения обучения. С использованием наборов данных (датасетов) обучается модель машинного обучения, эксплуатация решения напрямую связана с данными, на которые применяется обученная модель. Наличие данных создает дополнительные требования к проекту, так как необходимо предусмотреть место для хранения данных, запланировать вычислительные ресурсы для обработки данных, организовать мониторинг качества данных. Задачи инженера данных (Data Engineer, DataOps) в проекте машинного обучения могут быть очень разнообразными, например: обеспечение доступности (демократизация) данных, контроль и обеспечение качества данных, управление версиями датасетов, организация конвейера обработки данных. В этом модуле вы узнаете о задачах инженера данных подробнее, изучите необходимый инструментарий и научитесь его применять для решения задач MLOps.

Темы, изучаемые в модуле:

1. Задачи инженерии данных
2. Этапы обработки данных
3. Управление данными на примере dvc

# Модуль 4. Юнит 1. Задачи инженерии данных.

*Введение:* В этом юните вы познакомитесь с задачами, которые решает инженер данных (Data Engineer, DataOps), наиболее актуальными проблемами при работе с данными, методами предупреждения или решения этих проблем, базовым инструментарием.

*Содержание юнита:*

Данных становится все больше. Все чаще появляются новые области знаний, о появлении которых люди даже не задумывались еще несколько лет назад. В любой предметной области, в любой отрасли генерируется огромное количество данных. Эмпирическим путем подмечено, что количество данных сейчас увеличивается по экспоненте, то есть скорость изменения количества данных линейно зависит от текущего объема данных, чем больше становится данных, тем быстрее скорость их увеличения.

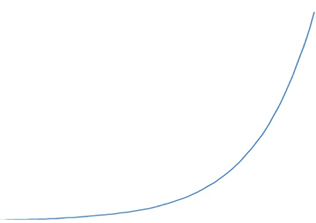


Рисунок “График экспоненты”.

Это очень серьезный тренд, и психологически, и технологически. Такой большой объем информации надо эффективно собирать, передавать, хранить. Однако самая большая проблема, связанная с данными, заключается не в их объеме, а в их неструктурированности. Данные появляются из различных не связанных между собой источников, в разных форматах, в разное время и по разному расписанию. *Поэтому перед использованием в практических задачах данные упорядочивают, преобразуют, приводят в форму, эффективную для хранения и использования. Все эти задачи решает инженер данных.*

Наличие данных отличает проекты машинного обучения от всех остальных проектов разработки программного обеспечения. Конечно, существуют информационные системы без машинного обучения, которые используют данные, ведь почти в каждом программном продукте есть своя база данных. **Однако особенность проектов машинного обучения состоит в том, что результат работы системы очень сильно зависит от качества и содержания данных на всех этапах жизненного цикла проекта: от обучения до эксплуатации.** Поэтому инженер данных является важным участником команды проекта машинного обучения. На практике часто получается, что задачи инженерии данных тесно пересекаются с задачами MLOps. Часто случается, что это один и тот же человек. Поэтому специалисту MLOps необходимо уметь разбираться в задачах, связанных с данными, и инструментах для их решения.

Основные задачи инженерии данных:

1. сбор и передача данных
   1. организация сбора данных (по расписанию, по триггеру, в пассивном режиме, настройка доступа к данным),
   2. создание каналов связи и настройка прикладного ПО для передачи данных, при необходимости создаются защищенные каналы связи с шифрованием,
   3. мониторинг и анализ работы источников данных, выявление проблем в сборе и передаче данных,
2. анализ данных
   1. обнаружение аномалий, пропусков, ошибок,
   2. статистические характеристики для числовых данных, сравнение с предыдущими характеристиками, выявление отклонений в поведении данных,
3. сохранение полученных данных
   1. проектирование базы данных
   2. организация хранилища, настройка оборудования и программного обеспечения, организация доступа
   3. контроль изменений в данных
   4. защита информации, шифрование, резервирование
4. предобработка данных
   1. приведение типов,
   2. устранение ошибок,
   3. обработка выбросов и аномалий,
   4. преобразование имеющихся признаков,
   5. создание новых признаков,
5. использование при обучении модели

**Согласно исследованию Forbes, 80% времени инженеры данных тратят на подготовку данных** (<https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says>).

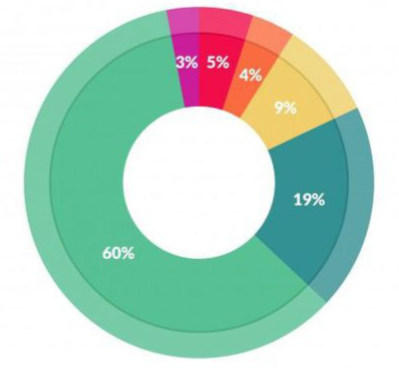


Рисунок “Распределение задач инженеров данных, согласно исследования Forbes”

Распределение объемов работ, согласно данным исследования, следующее:

* 3% создание тренировочных датасетов,
* 60% очистка и форматирование данных, организация структуры данных,
* 19% сбор датасетов,
* 9% поиск паттернов в данных,
* 4% улучшение алгоритмов,
* 5% остальные задачи.

Давайте разберем эти задачи подробнее.

1. **Сбор и передача данных.**

Источников для получения данных может быть очень много, у них различные форматы и регламенты получения данных.

|  |  |  |
| --- | --- | --- |
| Источник | Форматы | Объем наборов данных |
| Текстовый файл | txt, xlsx, csv/tsv | небольшой |
| IoT устройства (датчики, счетчики) | API, специализированные технологические протоколы (modbus) | небольшой |
| Информационные системы | API | средний |
| Базы данных | SQL, NoSQL, | большой |

На этом этапе важно контролировать исполнение так называемого «контракта данных», который описывает структуру, тип данных, регламент получения данных, расписание.

Собранные данные необходимо передавать дальше для обработки и хранения. Для систем получения и передачи данных важным является правильная оценка пропускной способности канала передачи данных. Для этого необходимо правильно определять точки концентрации данных для обработки и хранения, чтобы своевременно обнаруживать слабые места.

1. **Анализ данных.**

Наибольшая польза в использовании данных заключается в возможности проведения анализа для выявления полезных свойств данных, закономерностей, скрытой информации, имеющейся в данных. С этой целью дата инженеры и инженеры машинного обучения анализируют структур имеющихся данных, исследуют имеющиеся признаки и конструируют новые признаки для обучения моделей.

Важно понимать, что дата аналитики не являются классическими программистами. Обычный язык программирования, которым владеют аналитики данных, это Python, который предназначен не для разработки, а для экспериментов. Результат работы исследователя данных это Jupyter-ноутбук, который не является программным кодом в классическом смысле, а является законченным исследованием, включающим таблицы, графики, вычисления. Поэтому для исследователя данных надо очень точно формулировать - что является артефактом выполнения его части работы, чтобы не случился кризис контроля версий. Наиболее логичным артефактом на этапе анализа данных является

* набор данных (dataset), представляющий собой обработанные, структурированные, сохраненные в определенном формате данных, предназначенные для обучения модели машинного обучения,
* набор признаков, эффективный для обучения модели,
* статистическая информация о данных в целом и отдельных признаках.

1. **Сохранение данных**

Отдельно надо хранить данные для обучения, тестирования, валидации. Отдельно хранится разметка. Для проектов машинного обучения как правило требуются большие датасеты, которые занимают много места. Поэтому необходимо качественно планировать структуру датасета для хранения и необходимый объем памяти.

1. **Обработка данных.**

Сырые, необработанные данные всегда имеют скрытые проблемы: пропуски, разная кодировка, изменчивый характер, разные структуры таблиц, разные единицы измерения. Вручную такие ситуации обрабатывать и “чистить” тяжело, поэтому нужен предварительный анализ и автоматизация.

**Из всего набора данных полезно выделить наиболее эффективные для модели признаки, либо сконструировать новые, этот процесс называется конструирование признаков (feature engineering)**.

Сырые данные, как правило, являются незамеченными, поэтому если модель предполагает “обучение с учителем”, то организуется процесс разметки данных, ручной или автоматический.

1. **Использование данных при обучении и эксплуатации модели**

Проблема в датасете то, что он эволюционирует. При эксплуатации модели машинного обучения происходит ухудшение качества работы модели и это связано прежде всего с тем, что данные в промышленных системах изменяются со временем: меняется их структура, статистические характеристики. Информационные системы изменяются в процессе эксплуатации, поэтому при проектировании, разработке и эксплуатации моделей машинного обучения необходимо учитывать, что датасеты могут изменяться, следовательно, надо контролировать все эти изменения.

**Общая концепция, описывающая подходы к сбору, обработке и загрузке данных, называется ETL (Extract, Transfer, Load), подробно она описана в следующем юните.**

Причина важности контроля потока обработки данных состоит в том, что в данных могут быть ошибки, а цена таких ошибок может быть очень большой. В целом, для проектов машинного обучения выполняется правило, что лучше предотвратить повреждение данных, чем потом исправлять ошибки во всей системе. *Есть проекты, затраты на обеспечение надежности и качество в которых могут достигать больших объемов в совокупной стоимости проекта (более 50%), однако стоимость устранения неполадок, в которых несоизмеримо больше, поэтому на такие затраты идут.* *К таким проектам относят и работу с данными.* Поиск ошибок в проектах с данными в проектах ML может быть очень затратным по времени.

**Причины «поломок» в данных:**

1. Изменения в данных:
   1. изменяются правила срабатывания триггеров событий (ивент-трекеры, технологические уставки),
   2. не соблюдается «контракт данных» (меняется структура, типы данных)
   3. сбои в инфраструктуре (отключение датчика, поломки в сети передачи данных)
2. Неправильная обработка данных:
   1. изменение бизнес-логики обработки данных,
   2. изменение схемы данных.

**Если плохие данные уже поступили в модель, то она уже отравлена. Можно отлавливать проблему анализируя появление аномалий и проводить анализ корневых причин неисправности (Root Cause Analysis, RCA) с изучением ретроспективной корреляции, а можно заранее озаботиться тем, чтобы плохие данные не попали в систему.**

**Способы контроля качества данных в проекте машинного обучения:**

1. версионирование всех элементов решения, затрагивающих данные: код, схемы и процедуры обработки (от трекинга до визуализации). Версионирование становится полезным тогда, когда данных и моделей становится много и информация о той или иной версии хранится только в головах инженеров.
2. оценка влияния изменений на данные
   1. Assertions tests (тестирование правильности результатов на отдельных дтапах) позволяют устанавливать контрольные точки и правила
   2. Data Diff (отслеживание изменений) для определения того, в каких именно данных произошли изменения и какой процесс мог их поменять.
   3. Lineage (отслеживание взаимосвязей) позволяет проанализировать взаимосвязь в данных. Например, из каких таблиц берутся финальные данные, как появляются данные на предыдущем шаге и так далее.
3. Разработка регламента для каждого процесса, который делает изменения в данных, внедрение и строгий контроль выполнения требований этого регламента.

Применяя эти методы можно быстро изолировать проблему, откатить проблемное изменение назад, отслеживать историю изменений.

**Инструменты для решения этих задач:**

|  |  |  |
| --- | --- | --- |
| Инструмент | Ссылка | Описание |
| AirFlow | https://airflow.apache.org | Создание и контроль потоков автоматизации, например, обработки данных или обучения модели. Набор последовательно выполняемых операций представляется в виде направленного ациклического графа (DAG, Directed Acyclic Graph). |
| MLFlow | https://mlflow.org | Платформа для управления жизненным циклом моделей машинного обучения. Умеет версионировать данные. |
| Git-lfs | https://git-lfs.github.com | git large file storage, система git для управления большими файлами |
| dvc | dvc.org | data version control, система управления датасетами, моделями и конвейерами. |

Идеальная организация проекта машинного обучения позволяет любому участнику команды проекта запускать весь проект «по щелчку» с минимальными накладными расходами. Возможность использовать при этом правильный датасет является ключевой, так как повторяемость эксперимента невозможно обеспечить на разных наборах данных.

*Тест*

1. Отметьте причины, по которым в процессе эксплуатации модели машинного обучения происходит незначительное ухудшение качества работы модели (0.25)
   1. изменяется структура данных, например меняется набор параметров для наблюдения,
   2. выход из строя источников данных (датчики, счетчики),
   3. **кратковременный сбой в работе каналов передачи данных,**
   4. **шум в данных, обусловленный ошибкой датчика**
2. Какие изменения в данных приводят к необходимости изменения модели машинного обучения? (0.25)
   1. **изменение бизнес-логики процесса,**
   2. шум в данных,
   3. сбой в каналах передачи данных,
   4. **изменение структуры данных**
3. Какие задачи относятся к инженерии данных? (0.25)
   1. **сбор данных,**
   2. **анализ и преобразование данных,**
   3. подбор гиперпараметров модели машинного обучения,
   4. **организация хранения данных**
4. Какие инструменты используются для версионирования (0.25)
   1. **dvc,**
   2. **git,**
   3. jupyterhub,
   4. python.

# Итоги/выводы

Данные являются важнейшей частью проекта машинного обучения, поэтому роли инженера данных (data engineer, DataOps) и инженера машинного обучения (MLOps) очень тесно связаны. В этом юните вы узнали о задачах инженерии данных, к которым относятся:

* сбор,
* препроцессинг (предварительная обработка данных),
* передача,
* сохранение,
* разведочный анализ данных,
* выбор признаков для обучения модели и задачи.

Также вы кратко ознакомились с инструментами, которые использует в своей работе инженер данных. Эти темы более подробно будут рассмотрены в следующих юнитах.

# Модуль 4. Юнит 2. Архитектура для работы с данными. Принципы работы ETL систем.

*Введение:* Правильная промышленная архитектура для работы с данными состоит из множества различных компонентов, решающих множество задач. В этом юните вы познакомитесь с основными понятиями ETL для работы с данными, узнаете из каких элементов обычно состоят системы для работы с данными и как правильно организовывать их работу.

*Содержание юнита:*

Данных стало гораздо больше, чем раньше, и их количество стремительно увеличивается. Типичное хранилище данных (DWH, Data WareHouse) государственной или коммерческой организации, использующей в своей деятельности аналитику данных, состоит из десятков тысяч связанных таблиц, в которых информация распределяется по этим таблицам, образуя сложные взаимосвязи между отдельными частями данных. Эти данные используются для создания моделей машинного обучения или бизнес-отчетов, на базе которых принимаются стратегические решения. Ясно, что неверные или неполные данные на входе приведут к неверным решениям на выходе.

**Широко используемый термин ETL является сокращением от английских терминов Extract (достать, извлечь), Transform (преобразовать), Load (загрузить). Назначение ETL систем заключается в приведении к единой форме и обеспечении возможности использования данных из множества различных информационных систем.** Необходимость применения принципов ETL обусловлена большим разнообразием источников, форматов представления данных и способов их получения. Например, на крупном предприятии данные берутся из множества используемых информационных систем, каждая система имеет свой формат данных и регламент их получения. Поэтому **основные задачи ETL** это:

1. сбор данных от различных источников,
2. обработка данных для последующего хранения и использования,
3. сохранение данных,
4. журналирование операций, связанных с обработкой данных, чтобы в любой момент была возможность установить происхождение конкретных величин,
5. применение данных, поиск полезных взаимосвязей, применение для автоматизации, отчетности, машинного обучения.

Логика работы ETL очень простая, состоит из простых операций. На каждом из этапов существуют типовые сложности и проблемы, о которых необходимо знать при создании систем обработки данных



1. **Загрузка данных** состоит в получении данных из разных источников произвольного качества для дальнейшей обработки.

*Типичные проблемы:* на этом этапе возможны различные ошибки, связанные с получением и передачей информации, например, сбой в формировании ответа на SQL запрос или перебои в канале передачи данных могут привести к тому, что фактически будет получено меньше данных, чем должно быть.

Для организации процесса загрузки важно уметь оценить продолжительность загрузки данных и увязать ее с существующими бизнес и технологическими требованиями. Например, если данных окажется очень много и процесс загрузки займет неделю, то это может не соответствовать бизнес-модели, которую описал заказчик проекта в техническом задании. В таком случае возможно придется уточнять технические требования или искать дополнительные возможности оптимизации загрузки.

1. **Проверка качества данных и очистка от ошибок.**

*Типичные проблемы:* ошибки могут быть самые разные, связанные как с человеческим фактором, так и со сбоями в хранении и передачи информации. Например, у данных могут оказаться разные форматы, когда часть информации представлена в числовом виде, а другая часть в текстовом. Возможны разные формы предоставления информации о датах, геолокации. Большую проблему представляют собой групповых ошибки в данных, когда каждое значение является легитимным, а группа значений содержит ошибку. Также плохо влияют на качество имеющиеся пропуски в данных, которые необходимо обрабатывать.

Создавать алгоритмы для проверки данных необходимо, чтобы исключить дальнейшее использование данных, способных повредить систему или нарушить процесс.

1. **Приведение данных в соответствие с целевой моделью.** На этом этапе обеспечивается единство структуры таблиц и форматов данных. Делается преобразование форматов и добавление необходимых таблиц. Имеющиеся данные могут быть преобразованы в другие данные, чтобы соответствовать целевой таблице.

*Типичные проблемы:* поскольку данные собираются из разрозненных источников, их использование в общей системе становится затруднительным, поскольку в разных системах разная структура таблиц, которые сложно соединить в общую структуру.

Перед использованием данных проводится анализ их структуры и проверка соответствия “контракту данных”, который определяет формат, структуру, регламент получения данных от разных источников. Отклонение от “контракта данных” должно быть соответствующим образом обработано.

1. **Процесс агрегации данных** позволяет реализовать промежуточную логику между слоем сбора данных OLTP (Online Transaction Process, обработка транзакций в реальном времени) и слоем применения методов аналитики OLAP (Online Analytical Processing, онлайн обработка данных).

*Типичные проблемы:* на этом этапе обрабатываются огромные массивы информации, небольшие ошибки в логике обработки данных на этом этапе ведут к серьезным накладным расходам, как временным, так и вычислительным.

1. **Выгрузка в систему, в которой будут использоваться данные.**
2. **Использование данных.** Иногда применяется термин *drill-down.* На этом этапе применяются разные методы формирования отчетов, аналитики и машинного обучения.

Рассмотрим подробнее последний этап, на котором используется результат работы всего конвейера ETL. **Углубление в данные (drill-down data)** широко используется в различных контекстах. **В** [**анализе данных**](https://wiki.loginom.ru/articles/data-analysis.html) **обычно имеется в виду сосредоточение, погружение в структуру данных, чтобы получить информацию, полезную для** [**поддержки принятия решений**](https://wiki.loginom.ru/articles/decision-support-system.html)**.** Проникновение в данные, как правило, начинается с верхних уровней, уровней наибольшего обобщения. Например, информация о количестве инцидентов компьютерной безопасности это всего одно число, которое само по себе не позволит проанализировать динамику компьютерных атак в течение года. Для получения информации о такой динамике необходимо получить более детализированные данные. Углубление (drill-down) в данные является одной из важнейших операций с так называемыми OLAP-кубами, она позволяет осуществлять навигацию между различными уровнями агрегирования данных. **OLAP-куб это многомерный массив данных, как правило, разреженный и долговременно хранимый, используемый в OLAP.** Индексам массива соответствуют измерения или оси куба, а значениям элементов массива меры куба.

**Давайте рассмотрим подробнее несколько важных элементов из ETL архитектуры:**

* **Озеро данных (data lake)** это большой репозиторий необработанных исходных данных, как неструктурированных, так и частично структурированных. Данные собираются из различных источников и хранятся в исходном виде, без каких-либо преобразований. Поэтому для дальнейшего использования этих данных требуется предварительная подготовка, например, очистка, форматирование, устранение выбросов. В озерах данных могут храниться как объекты структурированной информации (строки и таблицы из реляционных баз данных), так и полуструктурированной (например, xml или json файлы) и даже совсем неструктурированной (текстовые файлы, изображения и видео). Для данных в озере данных как правило фиксируются источник этих данных, владелец, регулярность получения, время хранения. Также могут присутствовать метаданные (то есть данные о данных). Несмотря на большие накладные расходы, связанные с хранением неструктурированной информации, озера данных дают выигрыш в том, что для их создания не применяются никакие аналитические алгоритмы, а современные средства хранения позволяют эффективно сохранять все большие объемы неструктурированной информации.
* **Хранилище данных (data warehouse)** представляет собой данные, агрегированные из разных источников в единый центральный репозиторий, который унифицирует их по качеству и формату. Далее эти данные могут быть использованы в таких сферах, как data mining, искусственный интеллект, машинное обучение и в бизнес-аналитике.
* **Витрина данных (data mart)** это хранилище данных, предназначенное для определенного круга пользователей в организации. Витрина данных может использоваться руководством компании для анализа производительности и контроля выполнения планов. Наборы данных в витрине данных часто используются в режиме реального времени для аналитики и получения практических результатов.

**Некоторые важные инструменты, о которых необходимо знать:**

* **Kafka** получает неотфильтрованные и необработанные сообщения и функционирует как принимающий узел в озере данных. Kafka обеспечивает надежный и высокопроизводительный сбор сообщений. Кластер Kafka обычно содержит несколько разделов для сырых данных, обработанных (для потоковой обработки) и недоставленных или искаженных данных.
* **Flink** принимает сообщение из узла c необработанными данных от Kafka, фильтрует данные и делает, при необходимости, предварительное обогащение. Затем данные передаются обратно в Kafka (в отдельный раздел для отфильтрованных и обогащенных данных). В случае сбоя, или при изменении бизнес-логики, эти сообщения можно будет вызвать повторно, т.к. что они сохраняются в Kafka. Это распространенное решение для в потоковых процессов. Между тем, Flink записывает все неправильно сформированные сообщения в другой раздел для дальнейшего анализа.
* Для управления потоками данных (data-flows) и ETL можно использовать **Apache Airflow**, который позволяет запускать конвейеры (pipeline) обработки данных с использованием Python и объектов типа Directed Acyclic Graph (DAG). Airflow позволяет гибко настраивать конвейеры и отслеживать выполнение задач через графический интерфейс. Также Airflow можно использовать для обработки внешних данных.
* **Spark** может использоваться для обогащения сырых отфильтрованных данных в соответствии с бизнес-задачами, после чего эти данные могут использоваться дата-аналитиками и бизнес-аналитиками. Для большой команды проекта Spark можно использовать для настройки многопользовательских интерфейсов для работы с данными, их сбором и анализом.
* **Hive** это система управления базами данных на основе системы Hadoop, позволяет реализовать стандартный интерфейс баз данных (например, SQL) для получения данных, хранящихся в Hadoop.
* **ElasticSearch** инструмент полнотекстового поиска в наборах данных, позволяет за счет индексации данных сделать такой поиск эффективнее, является важной частью ELK тека (ElasticSearch, Logstash, Kibana), в котором Logstash используется для приема и обработки log файлов, а Kibana для визуализации.
* HDFS файловая система **hadoop** для хранения больших объемов информации.

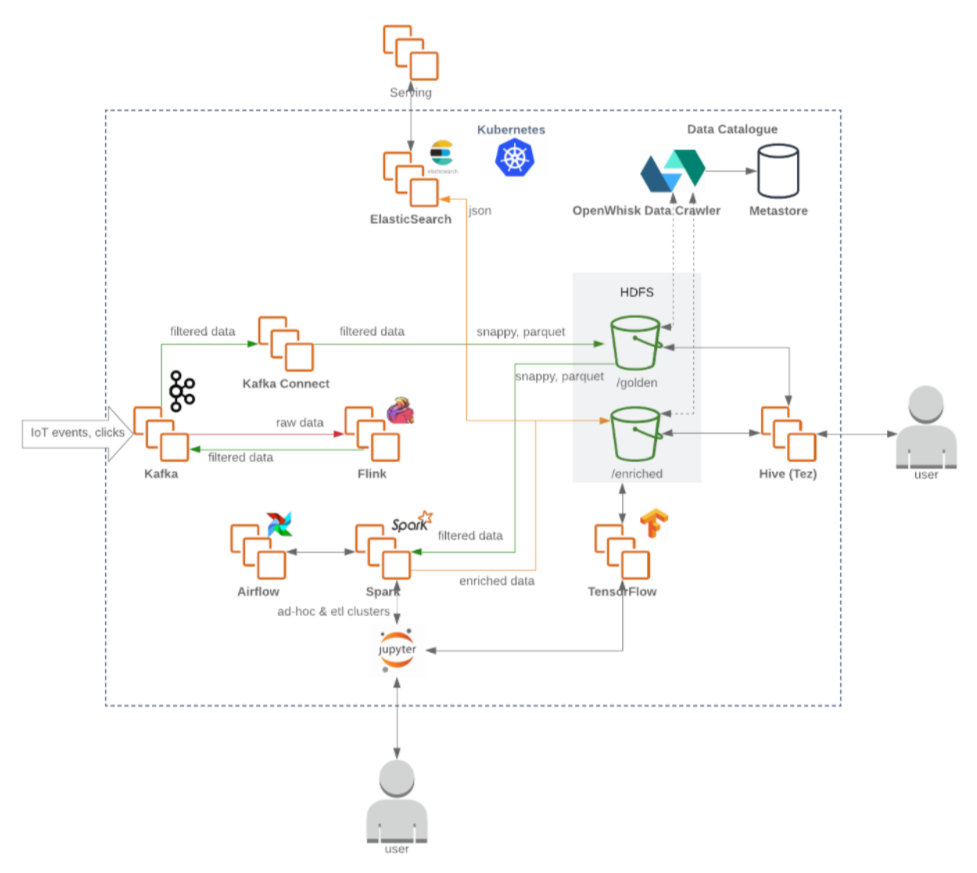


Рисунок “Вариант целевой архитектуры для организации ETL процессов, источник https://habr.com/ru/company/opsguru/blog/514322/”

*Тест*

1. Из каких операций состоит ETL? (0.25)
   1. **Extract, Transform, Load**
   2. Extract, Transfer, Load
   3. Enhance, Transit, List
   4. Enter, Test, Leave
2. Что такое витрина данных? (0.25)
   1. веб-интерфейс для просмотра данных
   2. специальный формат графика
   3. **специальная форма хранилища данных**
   4. элемент онлайн-площадки для торговли и обмена данными
3. Какие объекты являются видами корпоративных хранилищ данных? (0.25)
   1. **data lake**
   2. data river
   3. **data warehouse**
   4. data office
4. Что может быть причиной ошибок в данных? (0.25)
   1. **несовпадение форматов**
   2. **человеческий фактор**
   3. **сбои в передаче информации**
   4. **ошибки в преобразовании данных**

# Итоги/выводы

В этом юните вы изучили архитектуру ETL, ее основные задачи и этапы, использующиеся в организациях, применяющих аналитику данных. На практике в проектах машинного обучения данные могут поступать из различных источников. В этом смысле полезно знать архитектуру ETL как один из возможных источников для получения данных, а также понимать ее методы, чтобы по аналогии создавать похожие механизмы для работы с данными в своих проектах.

Для организации исследований в небольших проектах часто достаточно более простых и в основном open-source инструментов, некоторые из которых мы рассмотрим в следующих юнитах.

# Модуль 4. Юнит 3. Инструменты для управления данными. Примеры использования dvc.

*Введение:* Управление данными важным процессом в проектах машинного обучения. Наборы данных меняются в процессе работы над проектом. Разные участники команды используют разные версии датасетов для своих задач. Без системы контроля версий этим процессом невозможно управлять. **Одним из широко распространенных инструментов контроля версий датасетов, а также моделей машинного обучения, является dvc, работа с которым подробно рассмотрена в этом юните.**

*Содержание юнита:*

В ходе работы над проектом машинного обучения наборы данных постоянно меняются. Причины этих изменений могут быть как внутренние, обусловленные экспериментальным характером процесса построения модели, так и внешними, связанными с изменениями структуры и характеристик данных.

**Внутренние причины изменений в данных:**

* в результате обработки имеющихся данные для улучшения качества работы модели, проведена очистка данных,
* удаление или обработка выбросов и аномалий,
* шкалирование данных,
* приведение к наиболее эффективному формату для хранения,
* создание новых признаков, позволяющие лучше обучать модель,
* удаление неэффективных признаков.

**Внешние причины изменения данных:**

* обогащение данных за счет использования новых дополнительных источников информации,
* обновление данных, получение новых данных с объекта эксплуатации,
* изменения в структуре данных, отклонение от “контракта данных”.

Все это приводит к появлению новых наборов данных, которые необходимо сохранять, чтобы обеспечить повторяемость результата работы модели. *Например, если новый участник команды попробует использовать одну из обученных моделей, но при этом использует неправильный набор данных, то модель может аварийно завершить работу или, что еще хуже, выполнить работу с ухудшением качества, причину которого обнаружить будет очень трудно.*

Для решения этой проблемы в разработке программного обеспечения традиционно используют **системы контроля версий.** Система контроля версий записывает изменения в файле, происходившие в течение всего периода работы над проектом, запоминает дату изменения и автора, позволяет в любой момент вернуться обратно к нужной версии. Централизация системы контроля версий позволяет многим разработчикам иметь доступ к версиям, синхронизировать работу больших команд, управлять изменениями. Важно, чтобы эта система была распределенной, а не хранила всю информацию в одном месте, иначе выход центрального узла из строя может привести к потерям всей информации. Одна из самых популярных систем контроля версий это **git**. В распределенных git системах каждая копия репозитория является полной копией (бэкапом) всех данных проекта, опубликованных в репозитории.

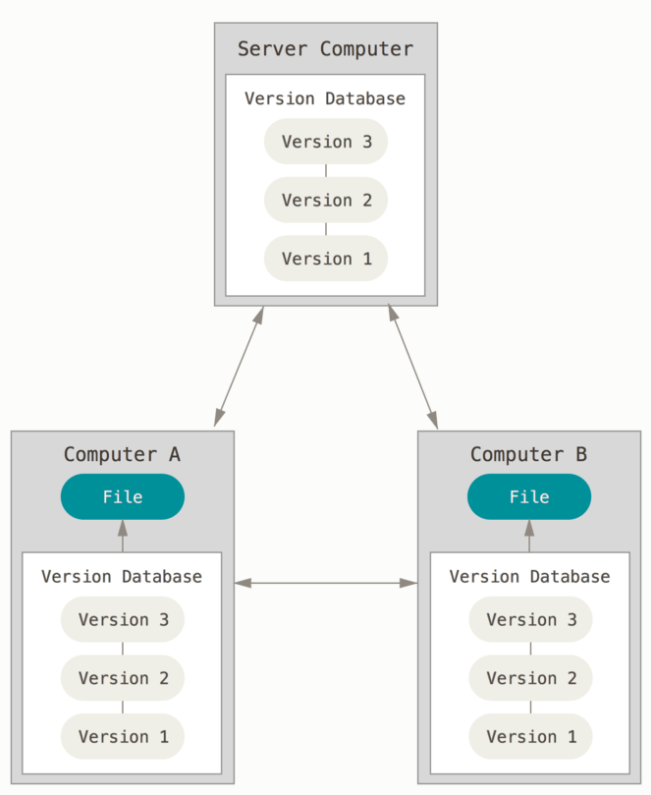


Рисунок “Схема работы распределенной системы версионирования”.

Такой подход позволяет делать разные ветки изменений в файлах проекта и управлять этой системой веток, обеспечивая “нелинейность” процесса разработки. Наиболее популярными инструментами для контроля изменений в процессе разработки программного обеспечения являются github (https://github.com) и gitlab (https://gitlab.com). Эти инструменты вы подробнее рассматривали в предыдущих курсах. Также легко можно найти информацию о применении этих инструментов в Интернет.

Для наборов данных в проектах машинного обучения контроль изменений так же важен, поэтому появились аналоги git для управления данными в проектах машинного обучения. Одним из наиболее популярных инструментов в настоящее время является **утилита dvc** (сокращение от data version control). Разработкой и поддержкой dvc занимается компания Iterative.ai (https://iterative.ai/). Проект разрабатывается в формате open-source, git репозиторий проекта находится здесь: <https://github.com/iterative/dvc>. Первая версия dvc появилась в 2017 году, однако, несмотря на относительно небольшой возраст, у решения dvc сформировалось большое сообщество пользователей и разработчиков.

**dvc предоставляет следующие возможности:**

* контроль изменений в наборах данных,
* версионирование моделей,
* контроль и версионирование экспериментов, благодаря которому можно быстро восстановить любой эксперимент,
* создание потоков операций для автоматизации (workflow), включая перевод обученной модели в эксплуатацию,
* хранение артефактов в локальном кэше или на локальном сервере с доступом по ssh,
* загрузка артефактов в удаленное хранилище, например,
  + Amazon S3,
  + Google Drive,
  + MS Azure,
  + Yandex DataSphere (https://cloud.yandex.ru/docs/datasphere/concepts/dvc-and-git),
* специализированный набор инструментов для воспроизводимости экспериментов.

Таким образом, кроме контроля версий датасетов dvc практически может быть полезен и для автоматизации и контроля проведения экспериментов, и для перевода модели машинного обучения в эксплуатацию.

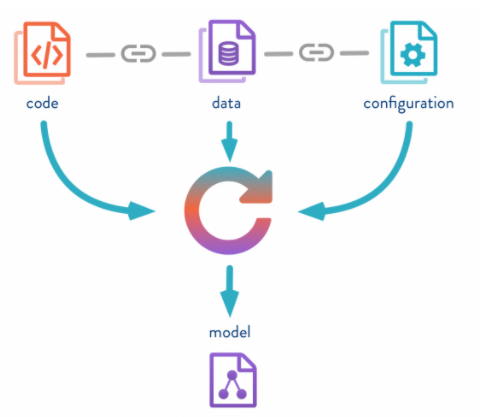


Рисунок “Использование dvc (с сайта dvc.org)”

*Утилита dvc работает только совместно с git.* Также многие стандартные команды git используются в dvc, например, add, commit, status, pull, push.

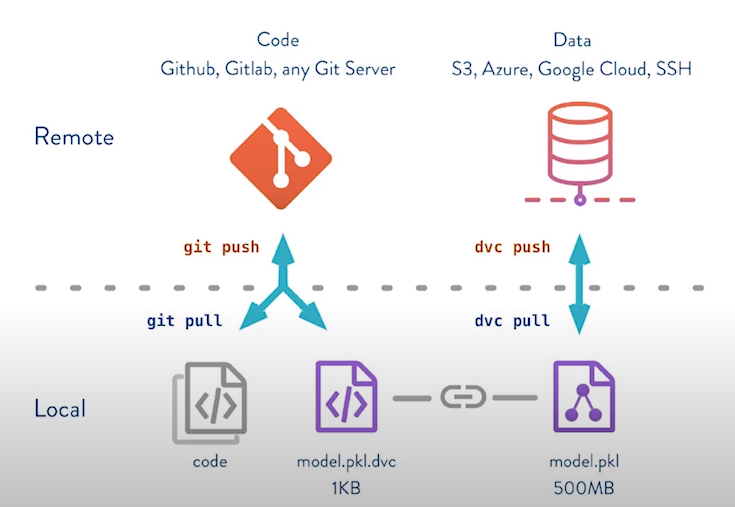
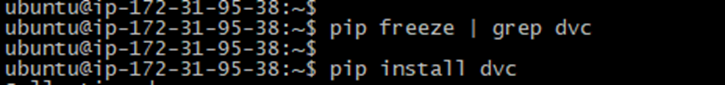
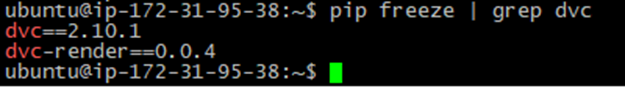


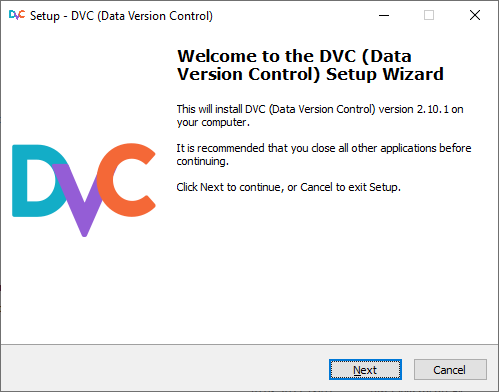
Рисунок “Совместная работа git и dvc (с сайта dvc.org)”

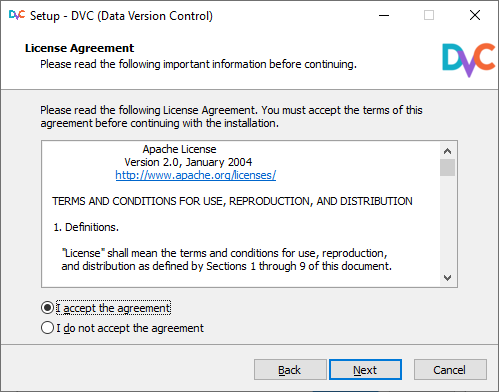
Установка dvc очень проста, поддерживается установка и использование для различных операционных систем. Например, в linux системе установка может быть выполнена с использованием установщика pip.

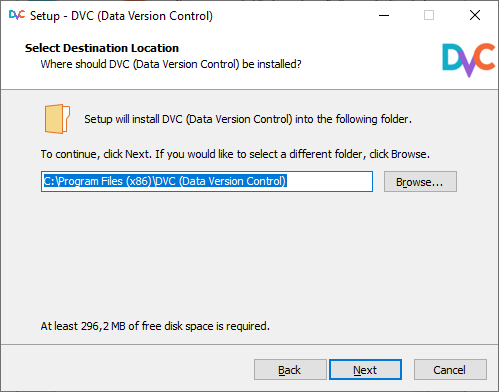


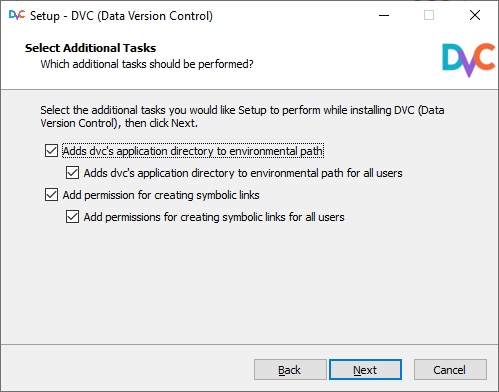


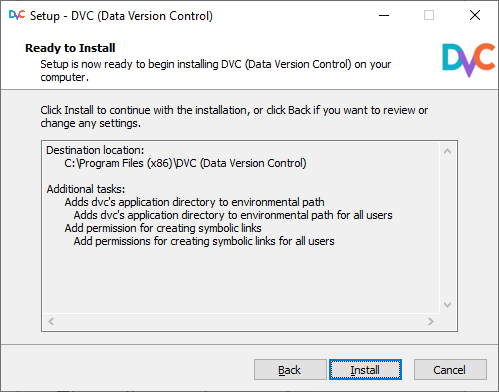
Для работы с dvc в операционной системе Windows необходимо скачать установочный exe файл с официального сайта dvc.org. После этого процедура установка стандартная для windows приложений.

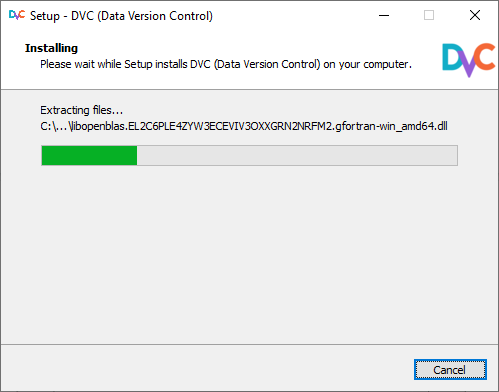


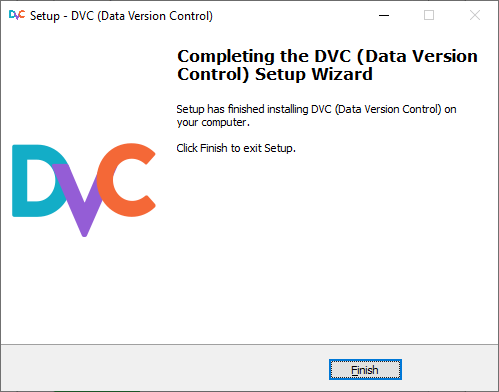








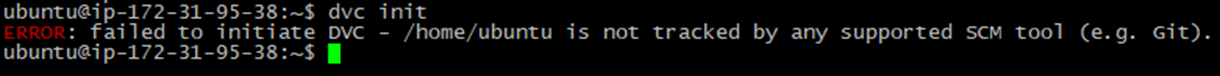




Теперь можно пользоваться dvc, например в PowerShell



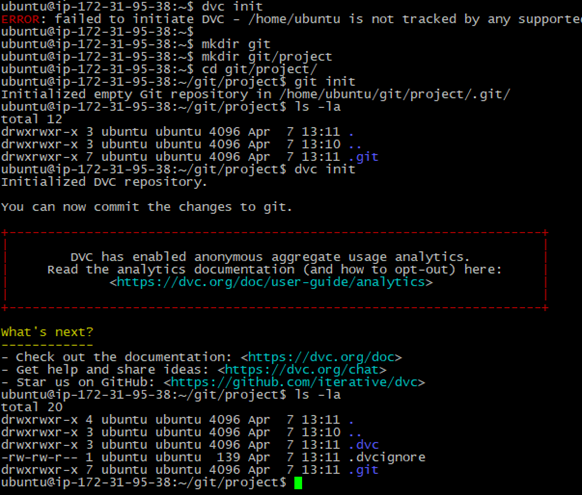
Далее мы рассматриваем примеры работы в linux операционных системах. Если после установки dvc сразу попробовать начать работу, то появится сообщение об ошибке, так как необходимо инициализировать git.



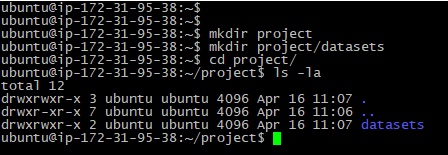
Для того, чтобы устранить эту проблему и начать пользоваться dvc необходимо сделать рабочую папку git репозиторием с использованием команды

**git init**

после этого инициализация dvc проходит успешно.



Для более подробного практического ознакомления с инструментом dvc для контроля версий наборов данных давайте на простом “игрушечном” примере проделаем стандартный набор операций с данными: создание, изменение, сохранение, загрузка нужной версии данных. Для этого создадим рабочую папку проекта в этой директории папку для датасетов.



Сделаем эту папку git репозиторием с помощью команды

**git init**

при этом появится служебная папка .git. После этого можно выполнить команду

**dvc init**

которая создаст необходимые папки и файлы для организации работы dvc.

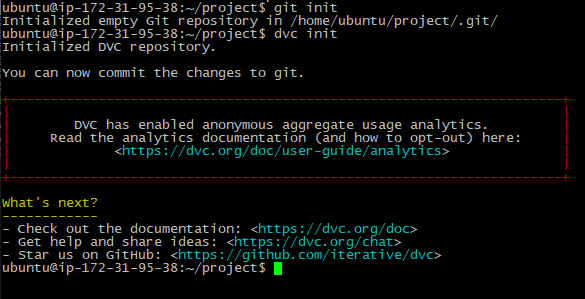
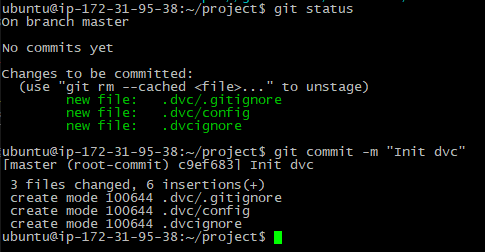


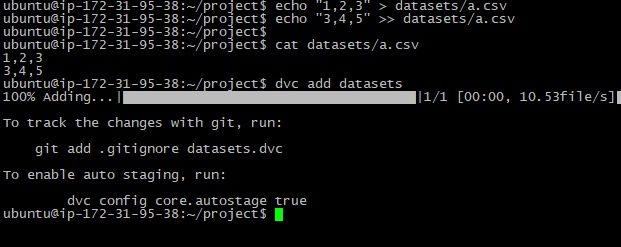
Рисунок “Результат выполнения git init и dvc init”.

Эту директорию со служебными файлами нужно сохранить в git (то есть сделать git commit).

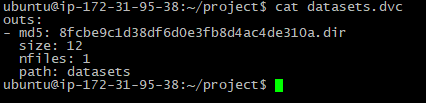


Теперь создадим датасет, сохраним его в папку datasets и начнем отслеживание (трекинг) папки datasets с датасетами с использованием команды

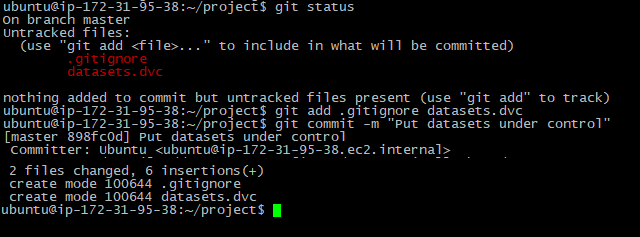
**dvc add datasets**



Вы видите подсказку системы, что для трекинга датасетов необходимо в git добавить файл .gitignore, который содержит информацию о том, чего не должно быть в git репозитории, а также файл datasets.dvc следующего содержания



Это служебный файл, который содержит хеш наблюдаемого объекта. А если заглянуть в файл .gitignore, который содержит перечень правил, запрещающих публикацию в git, то вы увидите там репозиторий /datasets, в котором хранятся наши данные. И это правильно, так как в git публиковать эти данные не требуется. Давайте добавим эти файлы в набор для commit и сделаем commit в git



Теперь мы опубликовали в git данные для того, чтобы работать с датасетами, но не сами датасеты. Для хранения датасетов нам надо определить хранилище. dvc поддерживает стандартные хранилища: Amazon S3, MS Azure, Google Drive, а также хранилище на сервере с доступом по ssh. Мы рассмотрим пример организации хранилища на Google Drive. Для этого понадобится аккаунт в Google, с использованием которого можно воспользоваться сервисом “Google Disc”. Создадим в этом сервисе папку для хранения версий наборов данных и установим права доступа, для возможности записи в папку необходимо установить роль “Редактор”. После этого можно получить ее идентификатор папки для использования в dvc.

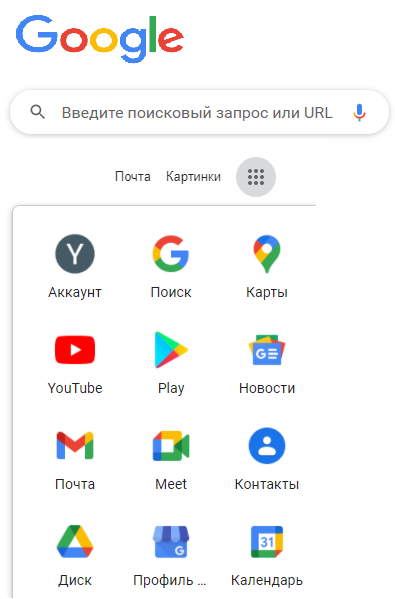


Рисунок “Запуск сервиса Google Disk в google.com”

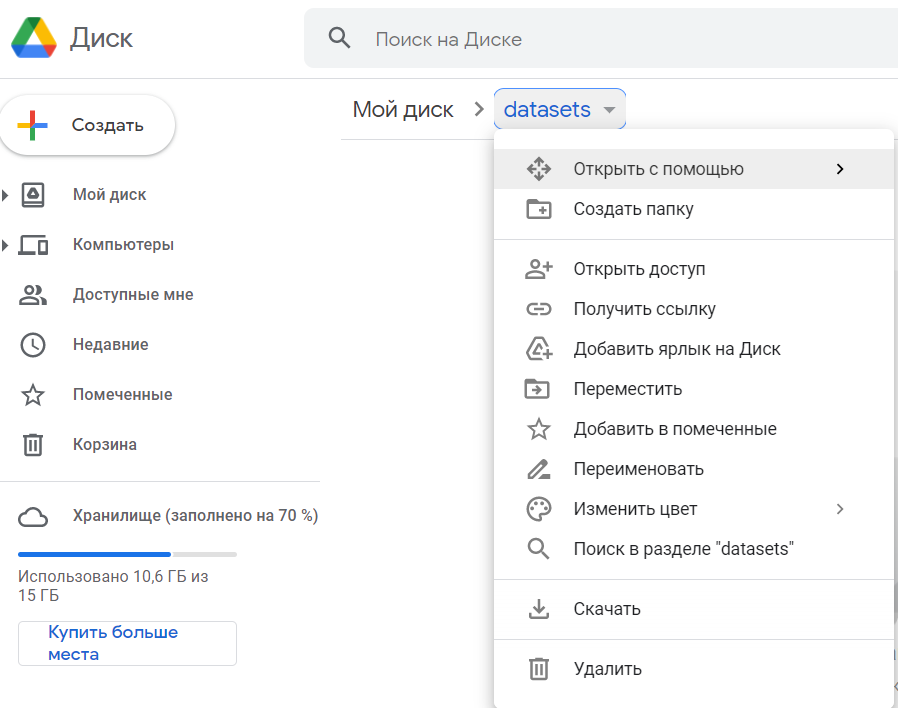


Рисунок “Создание папки в Google Disk”

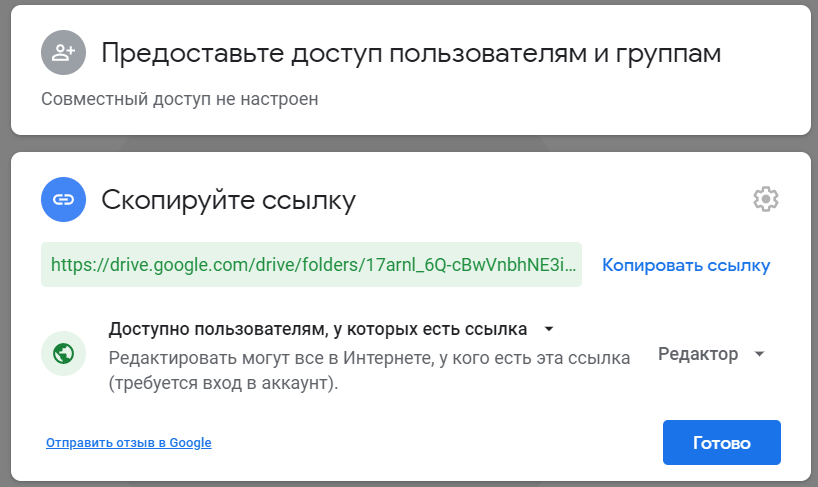


Рисунок “Редактирование прав доступа к папке в Google Disk”

Нам понадобится уникальный идентификатор этой папки для использования в dvc (у каждого пользователя и каждой папки он свой)

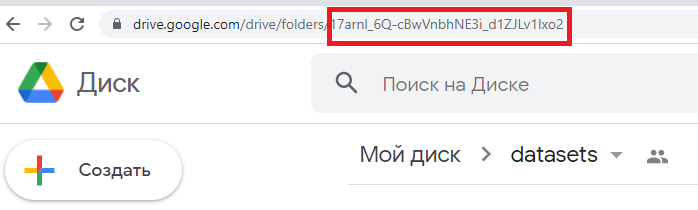
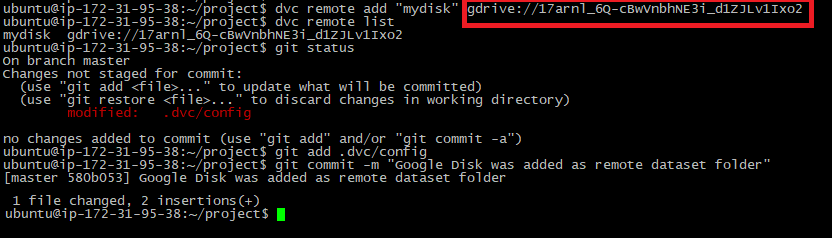


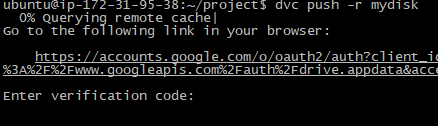
Рисунок “Уникальный идентификатор папки в Google Disk для хранения датасетов”

Это идентификатор используется при вызове команды dvc для того, чтобы добавить этот диск в перечень удаленных репозиториев для хранения датасетов

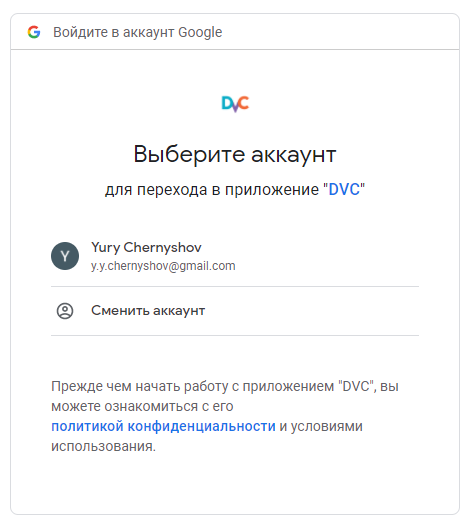
**dvc add remote “внутреннее имя репозитория” “уникальный идентификатор”**

****

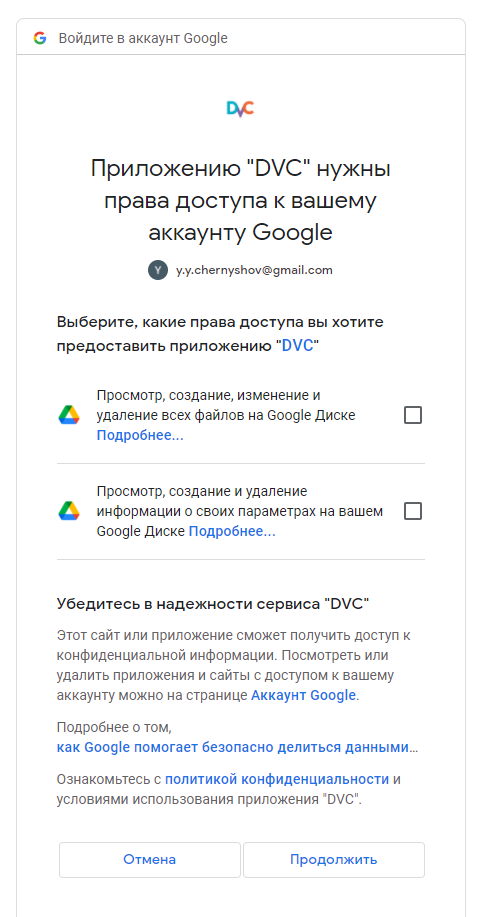
Необходимо учитывать, что для доступа к ресурсам Google требуется авторизация, про которую подробно можно почитать тут: <https://dvc.org/doc/user-guide/privacy>. Мы для упрощения выполним самый простой способ авторизации, когда проверка прав доступа выполняется вручную. Выполнив команду dvc push для публикации датасета в удаленном репозитории, который является аналогом команды git push, вы увидите следующее сообщение



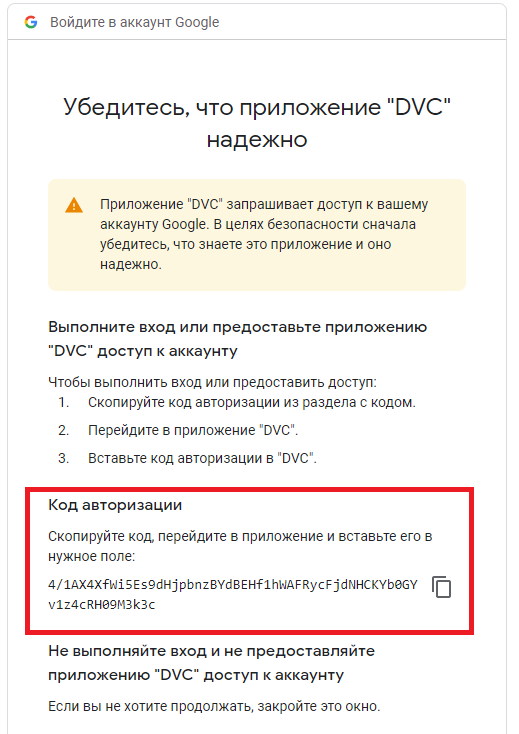
в котором предлагается пройти по указанной ссылке, получить код верификации и ввести его для продолжения авторизованной работы. После перехода по ссылке вы увидите следующее окно (каждый пользователь будет видеть свои персональные данные)



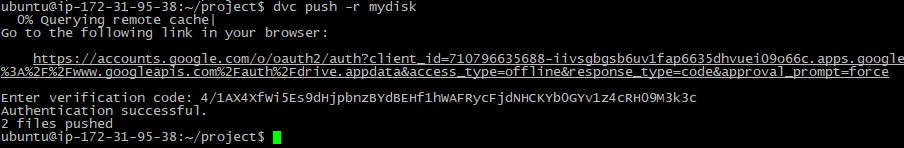
После входа в аккаунт необходимо разрешить приложению dvc взаимодействовать с Google Disk



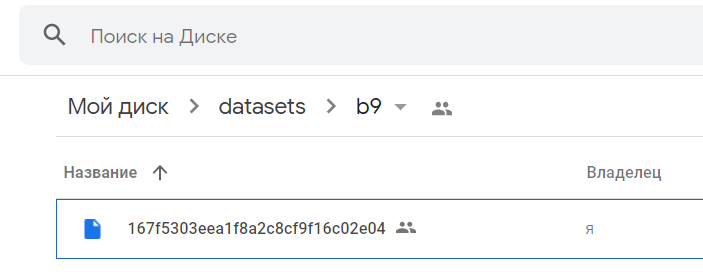
После этого вы увидите верификационный код, который необходимо скопировать и использовать для авторизации в утилите dvc.



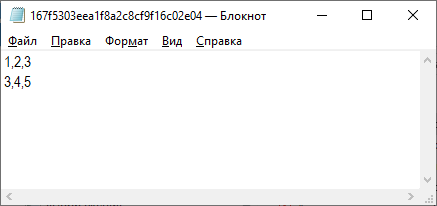
Из этого окна необходимо скопировать код авторизации и подставить его в консоли



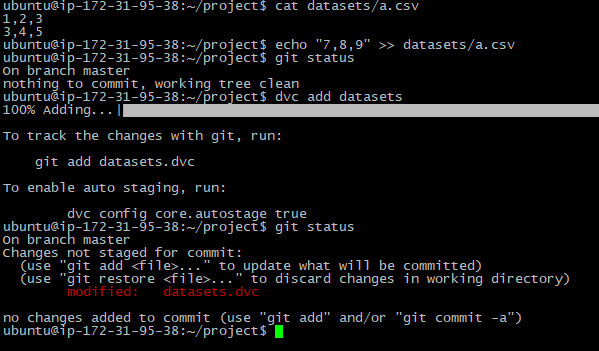
На этом процедура авторизации завершена, теперь вы можете пользоваться удаленным репозиторием для хранения датасетов, расположенным на Google Disk. В частности, мы видим сообщение “2 filed pushed”. Файлы датасетов хранятся в удаленном репозитории.



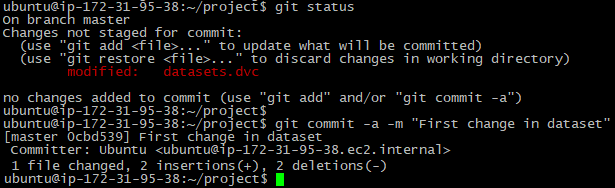
Если этот файл открыть в текстовом редакторе, то мы увидим содержание нашего датасета.



Теперь попробуем внести изменения в данные, сохранить их и выполнить переключение между различными версиями в данных. Для этого в наш датасет ./datasets/a.csv добавим еще одну строчку.



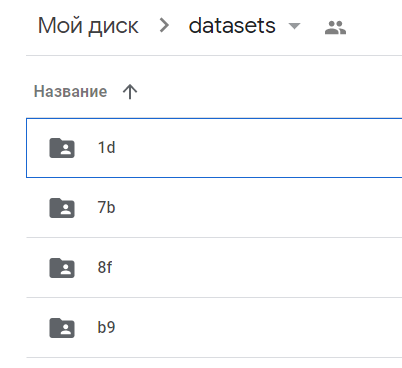
После изменения файла a.csv команда git status не видит изменений в файлах, так как папка датасетов ./datasets указана в .gitignore как запрещенная для коммита в git, что правильно, так как мы не публикуем датасеты в git. Для фиксации изменений в датасете необходимо выполнить команду dvc add, после чего будут сделаны изменения в файле datasets.dvc, которые уже необходимо опубликовать в git.



Выполнив команду dvc push мы можем опубликовать новую версию датасета в репозиторий

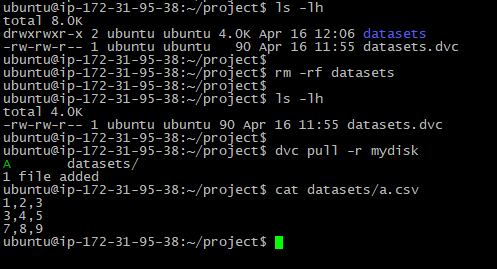


после чего в папке datasets на Google Disk появится еще папки.

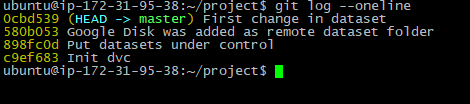


Папок появляется каждый раз по две, так как мы сохраняем информацию об изменениях в файле a.out и папке datasets.

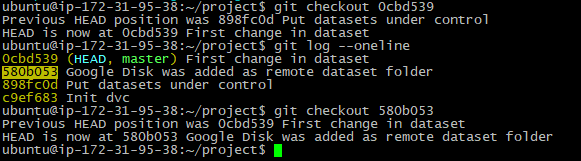
Теперь наш датасет и изменение в нем сохранены в удаленном репозитории. Давайте удалим датасет и загрузим его последнюю версию из удаленного репозитория.



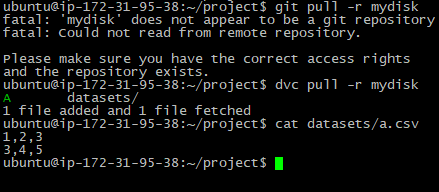
Как видите, мы нормально восстановили последнюю версию датасета из репозитория. Давайте теперь попробуем выбрать старую версию датасета. Все наши публикации изменений (коммиты) можно увидеть с помощью команды git log



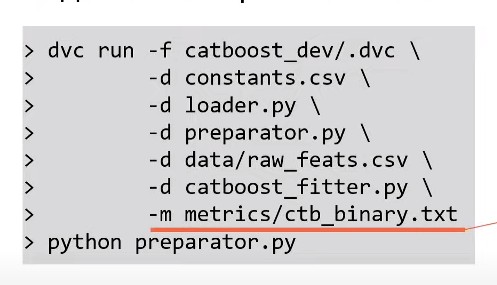
Вы можете перевести указатель HEAD на один шаг назад с помощью HEAD^1, либо в явном виде указать хэш коммита, на который хотите перейти



Теперь мы можем загрузить предыдущую версию датасета



Необходимо отметить, что также в dvc можно создавать пайплайны работы с моделями машинного обучения с использованием команды **dvc run**, при этом описание пайплайна делается обычными ключами, в том числе результат работы модели сохранять в определенный выбранный файл:



В следующих модулях мы также рассмотрим другие инструменты для создания пайплайнов.

*Тест*

1. без какой утилиты не будет работать dvc (0.25)
   1. java
   2. **git**
   3. ntp
   4. netstat
2. какие объекты можно версионировать в dvc? (0.25)
   1. **датасеты**
   2. **модели машинного обучения**
   3. **конвейеры машинного обучения**
   4. **метрики**
3. какая команда создает удаленный репозиторий? (0.25)
   1. **dvc remote add**
   2. dvc remote create
   3. dvc remote connect
   4. dvc remote setup
4. Что может быть использовано в качестве хранилища для dvc (0.25)
   1. **Amazon S3**
   2. **Google Drive**
   3. **сервер с доступом по ssh**
   4. **локальный жесткий диск**

# Итоги/выводы

В юните вы узнали как использовать утилиту dvc для организации работы с наборами данных (датасетами). Вы научились:

* создавать необходимую структуру папок
* версионировать и публиковать изменения в датасетах
* менять рабочие версии датасетов
* публиковать метаданные в git
* быстро разворачивать необходимую структуру с использованием клонирования git репозитория и вытягивания (pull) датасетов.

# Практическое задание на модуль.

В практическом задании данного модуля вам необходимо продемонстрировать навыки практического использования утилиты dvc для работы с данными. В результате выполнения этих заданий вы выполните все основные операции с dvc и закрепите полученные теоретические знания практическими действиями.

Этапы задания:

1. Установите git и dvc.
2. Создайте папку проекта.
3. Настройте папку проекта для работы с git и dvc.
4. Настройте git репозиторий.
5. Настройте удаленное хранилище файлов, например на Google Disk.
6. Создайте датасет о пассажирах “Титаника”, например, catboost.titanic().
7. Создайте датасет, в котором содержится информация о классе (“Pclass”), поле (“Sex”) и возрасте (“Age”) пассажира. Сделайте коммит в git и push в dvc.
8. Создайте новую версию датасета, в котором пропущенные (nan) значения в поле “Age” будут заполнены средним значением. Сделайте коммит в git и push в dvc.
9. Создайте новый признак с использованием one-hot-encoding для строкового признака “Пол” (“Sex”). Сделайте коммит в git и push в dvc.
10. Выполните переключение между всеми созданными версиями датасета.

При правильном выполнении задания и вас появится git репозиторий с опубликованной метаинформацией и папка на Google Disk, в которой хранятся различные версии датасетов.

В постановке задачи используется датасет из конкурса “Titanic Disaster”, однако вы можете использовать свои наборы данных, в этом случае в п.п.6-9 необходимо использовать информацию и признаки из вашего датасета.

Рекомендация: вы можете пользоваться описанием соответствующих процедур, которые были в юните 3 данного модуля.

В качестве результата необходимо прислать ссылку на репозиторий git.

# 