Итоговый проект

**Эксперименты в машинном обучении**

[Цель](#_ekrvind7qip5)

[Задачи](#_sxp2w6609zjd)

[Ход работы](#_dxo0jow50r6b)

[Этап 1](#_xyhk1hupun1t)

[Цели этапа](#_h0kauovle0ad)

[Комментарии](#_gzctv7sb62k)

[Этап 2](#_xtfogae45yg4)

[Цели этапа](#_s9gevu58c0a4)

[Доступ к датасету в Min](#_u5b7gc6dxpib)IO

[Первая модель](#_6em0ggvezgbv)

[Этап 3](#_fl7k5hl06z19)

[Цели этапа](#_anhb5uduaja3)

[Первый коммит в репозиторий](#_xk75k6udd2hw)

[Манифест MLflow Project](#_2p8nrv8z73y5)

[Этап 4](#_9ivk7shvyxl0)

[Цели этапа](#_l0kf23mjs6eb)

[Фиксируем окружение](#_wl64qk8vc3ti)

[Запускаем эксперимент прямо из репозитория](#_hkh9vq1l0kfy)

[Этап 5](#_146ut6ttryc4)

[Дедлайн сдачи проекта](#_r5nb3blkofe1)

[Формат сдачи](#_vlpiyqevgx8h)

[Программное обеспечение](#_ezg5b3h6xcnz)

# Цель

Студенты, выполняющие проект, учатся автоматизировать процесс проведения экспериментов в ML: выбор методов препроцессинга данных и самой модели, подбор гиперпараметров и других факторов, которые влияют на результат.

# Задачи

1. Создать среду (MLflow Tracking, MLflow Project), которая позволит проводить серии воспроизводимых и контролируемых экспериментов, а также сохранять все их результаты и артефакты.
2. Сконфигурировать (Hyperopt) серию экспериментов по выбору модели, которая решает задачу классификации текстовых данных.
3. Провести серию экспериментов, выбрать лучшую модель, сериализовать модель в pickle-файл, который, в свою очередь, должен быть сохранён в MLflow как артефакт эксперимента.

# Ход работы

## Этап 1

### Цели этапа

1. Установить и запустить всё необходимое программное обеспечение.
2. Сохранить приложенный к проекту датасет в объектном хранилище MinIO.

Создайте папку mlflow-server, а в ней:

* Манифест Docker Compose, который описывает конфигурацию из трёх сервисов: [MLflow Server](https://mlflow.org/), [MySQL](https://www.mysql.com/) и [MinIO](https://min.io/):
  + [MLflow Server](https://mlflow.org/) — популярный инструмент для управления жизненным циклом моделей машинного обучения. MLflow позволяет журналировать и изучать результаты экспериментов, создавать универсальные окружения, которые делают результаты экспериментов воспроизводимыми, сохранять готовые модели машинного обучения и многое другое.
  + [MySQL](https://www.mysql.com/) — наиболее часто используемая СУБД (система управления базами данных). В данном проекте MLflow будет использовать MySQL для журналирования результатов ваших экспериментов.
  + [MinIO](https://min.io/) — высокопроизводительное объектное хранилище, совместимое (на уровне API) с Amazon S3[[1]](#footnote-0). Объектное хранилище — это особенная архитектура хранения данных, которая хорошо масштабируется и позволяет хранить огромное количество данных. В этом проекте MLflow будет сохранять артефакты ваших экспериментов (например, модели машинного обучения) в объектном хранилище MinIO.
* Папку mlflow, в которой:
  + Dockerfile — докер-файл для сборки образа MLflow Server.
  + requirements.txt — перечень необходимых Python-пакетов, которые будут установлены в образ MLflow Server.
* Файл .env, который содержит необходимые переменные окружения.

Такая конфигурация — упрощённая, но вполне работоспособная модель окружения, с которым вы можете столкнуться в своей профессиональной деятельности.

Ваша задача — сконфигурировать и запустить сервисы, описанные манифестом Docker Compose, и убедиться, что всё работает правильно. Выполните необходимую команду, и, если всё запустится:

* По адресу <http://localhost:5050/> будет доступен веб-интерфейс сервиса MLflow.
* По адресу <http://localhost:9001/> будет доступен веб-интерфейс MinIO. В качестве логина и пароля используйте значения переменных окружения (см. файл .env) AWS\_ACCESS\_KEY\_ID и AWS\_SECRET\_ACCESS\_KEY.

### Комментарии

Рекомендуем разобраться в манифесте Docker Compose: в том, как именно он связывает между собой необходимые сервисы, откуда берутся значения переменных окружения, как собирается образ MLflow (из соответствующего Docker-файла) и как сервисы общаются друг с другом.

Кроме того, будет нелишним почитать материалы по теме, например [What is Object Storage](https://medium.com/codex/what-is-object-storage-object-storage-explained-in-simple-terms-19e9ffebb32c), зачем нужны хранилища данных и как они используются в индустрии.

Рекомендуемое время выполнения — 2-3 дня*.*

### Формат сдачи

Сделайте коммит в Git-репозиторий, который подготовлен специально для вас. Поставьте на коммит тег: stage1. Задание будет считаться выполненным, если:

1. Ваш репозиторий содержит все необходимые файлы и папки: манифест docker-compose, который верно конфигурирует все сервисы, файл .env с переменными окружения и папку mlflow с Dockerfile для mlflow и requirements.txt с необходимыми Python-пакетами.
2. Сервисы, которые сконфигурированы указанными файлами, запускаются.
3. На коммите в репозитории стоит правильный тег.

## Этап 2

### Цели этапа

1. Научиться считывать данные из объектного хранилища (по API S3) в Pandas-датафрейм.
2. Создать первую модель классификации в виде файла с кодом train.py.
3. Запротоколировать метрики модели в MLflow Server.
4. Сохранить в MinIO обученную модель в виде артефакта эксперимента MLflow.

#### Доступ к датасету в MinIO

Зайдите в веб-интерфейс MinIO, создайте новый бакет (bucket), назовите его datasets. Сохраните в бакете datasets датасет train.csv, который приложен к заданию проекта. Это данные, которые вы будете использовать, чтобы тренировать модель классификации. Теперь они загружены в объектное хранилище MinIO и доступны через API S3.

MLflow для доступа к артефактам, сохранённым на S3, использует Python-пакет Boto3. Поступите так же: создайте для проекта виртуальное окружение (предпочтительно с помощью venv) и установите в него пакет Boto3.

Теперь установите реквизиты доступа к объектному хранилищу (значение идентификатора ключа и сам секретный ключ для простоты хранятся в переменных окружения в файле .env). Установите переменные окружения (в вашей текущей оболочке) AWS\_ACCESS\_KEY\_ID и AWS\_SECRET\_ACCESS\_KEY в соответствующие значения, указанные в файле .env.

Далее, считайте train.csv из объектного хранилища в Pandas-датасет. Напишите соответствующий код.

Запустите код и убедитесь, что он работает. За подробностями обращайтесь [к официальной документации](https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html). Если всё сделано правильно, вы получите датасет, в котором всего две колонки:

1. **review** — отзыв на какой-либо фильм, размещённый на сайте Kinopoisk.
2. **sentiment** — тональность отзыва. Принимает одно из трёх значений: pos — отзыв положительный, neu — отзыв нейтральный, neg — отзыв отрицательный.

Колонка sentiment — целевая переменная. Как вы уже поняли, вам предстоит решить задачу классификации: определить по тексту отзыва, какую тональность он содержит.

#### Первая модель

Постройте первую модель, которая решает задачу классификации отзывов по тональности. Используете простейший векторизатор (например, CountVectorizer) и обучите логистическую регрессию. Протоколируйте в MLflow всё, что посчитаете нужным (параметры модели, метрики классификации). Кроме того, сохраните обученную модель в объектное хранилище MinIO, подключённое к MLflow (с помощью метода [mlflow.sklearn.log\_model](https://www.mlflow.org/docs/latest/python_api/mlflow.sklearn.html#mlflow.sklearn.log_model)).

Чтобы ваш код «увидел» сервер MLflow, который исполняется в Docker-контейнере, необходимо:

1. Если потребуется, установить в виртуальное окружение Python необходимые пакеты.
2. Установить в вашей текущей оболочке переменные окружения MLFLOW\_TRACKING\_URI и MLFLOW\_S3\_ENDPOINT\_URL в значения <http://localhost:5050> и <http://localhost:9000> соответственно. Так ваш код сможет «найти» MLflow Server, а MLflow Server — объектное хранилище.
3. Зайти в веб-интерфейс объектного хранилища MinIO (по адресу <http://localhost:9001/>) и создать bucket c именем mlflow.

Если вам удобно, можете писать код в Jupyter-ноутбуке, однако важно, чтобы в конце этапа у вас получился отдельный файл train.py, который содержит весь необходимый код:

1. Считывание данных из объектного хранилища.
2. Предобработка данных (если требуется).
3. Векторизация текста.
4. Обучение модели классификации.
5. Логирование метрик в MLflow.
6. Сохранение обученной модели как артефакта MLflow.

Если всё сделано правильно, в результате каждого запуска train.py:

* В веб-интерфейсе MLflow (<http://localhost:5050/>) будут появляться данные очередного прогона эксперимента (ID эксперимента по умолчанию — 0).
* В объектное хранилище MinIO будет записан MLflow-артефакт с вашей моделью.

Рекомендуемое время выполнения — одна неделя.

### Формат сдачи

Сделайте коммит с внесёнными изменениями в Git-репозиторий, который подготовлен специально для вас. Поставьте на коммит тег stage2. Задание будет считаться выполненным, если:

1. Репозиторий содержит файл train.py с кодом, который забирает файл из объектного хранилища, обучает модель, логгирует метрики в MlFlow Tracking Server и сохраняет обученную модель в S3 посредством log\_model.
2. Код запускается и работает правильно.
3. На коммите стоит соответствующий тег.

## Этап 3

### Цели этапа

1. Сохранить код в Git-репозитории.
2. Создать манифест ML Project.
3. Запустить код обучения модели (протоколирование метрик и сохранение артефактов) из Git-репозитория.

#### Первый коммит в новый репозиторий

Для вас создан индивидуальный Git-репозиторий. Да, ещё один. В этом репозитории мы продолжим работать с кодом модели, а в предыдущем оставим всё то, что касалось инфраструктуры для проведения экспериментов. Клонируйте свой репозиторий и сделайте коммит, содержащий ваш файл train.py. Зафиксируйте (push) ваш коммит на удалённом Git-сервере.

#### Манифест MLflow Project

Теперь ваша задача — создать манифест [MLflow Project](https://www.mlflow.org/docs/latest/projects.html). MLflow Project — это компонент MLflow, который регламентирует способ оформления data-science-кода. Каждый проект в MLflow — это, как правило, Git-репозиторий, содержащий ваш код, а также манифест ML Project (в виде YAML-файла, который должен находиться в репозитории проекта), который описывает основные атрибуты проекта: его имя, окружение и сценарии запуска. MLflow Project унифицирует запуск вашего кода в фиксированном окружении — так вашим коллегам (а также разным автоматизированным инструментам) будет понятно, как запускать ваш проект, чтобы всё работало так, как вы хотели.

Создайте в папке вашего проекта (там, где train.py) манифест **ML Project**, который просто запускает скрипт train.py.

Чтобы разобраться с форматом и назначением параметров файла, прочитайте [официальную документацию](https://www.mlflow.org/docs/latest/projects.html).

Убедитесь, что созданный манифест MLflow Project работает и эксперимент запускается. Выполните команду:

| mlflow run . --env-manager=local --experiment-name=kinopoisk |
| --- |

Если всё прошло правильно, MLflow запустит ваш проект, train.py обучит модель и запротоколирует все метрики в МLflow в новый эксперимент kinopoisk. Зайдите в веб-интерфейс MLflow и убедитесь, что это так.

Создайте новый коммит, содержащий все созданные файлы. Создайте новый Git-тег с именем stage3. Зафиксируйте все изменения на удалённом сервере.

Рекомендуемое время выполнения — два дня.

### Формат сдачи

Задание считается выполненным, если:

1. Репозиторий содержит код эксперимента и манифест ML Project.
2. Проект запускается из манифеста ML Project и работает правильно.
3. На коммите стоит соответствующий тег.

## Этап 4

### Цели этапа

1. Подготовить изолированное окружение для запуска экспериментов в виде образа Docker.
2. Запустить эксперимент MLflow в изолированном окружении из Git-коммита.

#### Фиксация окружения

На предыдущем этапе вы запустили эксперимент, оформленный как MLflow Project, в своём локальном окружении (на это указывает опция командной строки --env-manager=local). Другими словами, такой запуск практически ничем не отличается от обычного запуска Python train.py **в вашем** виртуальном окружении **на вашей** рабочей машине.

Это не очень удобно, потому что эксперимент должен быть воспроизводимым и повторяемым, кто бы и где бы его ни проводил. Для этого необходимо зафиксировать окружение, в котором он запускается.

Надёжный способ создания окружения — использование Docker-контейнера, который фиксирует не только версию интерпретатора Python и версии используемых Python-пакетов (как делает виртуальное окружение conda или virtualenv), но и всё вокруг: версию операционной системы, системные библиотеки и прочее. MLflow поддерживает запуск экспериментов в изолированных Docker-контейнерах.

Вам необходимо создать Docker-файл, описывающий образ контейнера, в котором есть всё необходимое для запуска вашего эксперимента: python и нужные пакеты. Напишите такой Dockerfile. В файле requirements.txt запишите все зависимости (Python-пакеты с указанием версии), которые необходимы для запуска вашего эксперимента.

Соберите Docker-образ из созданного файла:

| docker build -f Dockerfile -t kinopoisk:1.0.0 . |
| --- |

Теперь измените манифест ML Project. Необходимо указать, что теперь ваш проект будет исполняться в окружении Docker-контейнера. Добавьте в манифест ML Project соответствующую секцию. Напомню, в ней вы должны указать:

1. Ключ **image**, который указывает на Docker-образ, который будет взят в качестве основы окружения вашего эксперимента.
2. Ключ **environment**, который указывает переменные окружения, которые будут установлены внутри запущенного контейнера. Обратите внимание: вы переопределили переменные MLFLOW\_TRACKING\_URI и MLFLOW\_S3\_ENDPOINT, которые на вашей локальной машине имеют значения <http://localhost:5050> и <http://localhost:9000>.

С помощью переменных MLFLOW\_TRACKING\_URI и MLFLOW\_S3\_ENDPOINT код, реализующий ваш эксперимент, находит (через сетевое соединение) MLflow Server и объектное хранилище MinIO. MLflow Server и MinIO, которые вы запустили на первом этапе этого задания, исполняются в собственных Docker-контейнерах. Эти контейнеры публикуют порты 5050 и 9000 (см. манифест Docker Compose), по которым программы, выполняющиеся на вашем компьютере, могут устанавливать с ними сетевые соединения (по адресу localhost). Однако теперь код вашего эксперимента изолирован в своём контейнере (словно в отдельном компьютере) и имеет свой адрес — localhost. Поэтому он не «увидит» ни MLflow Server, ни MinIO по адресам localhost:5050 и localhost:9000.

Если бы это происходило в продуктовом окружении, то, скорее всего, MLflow Server и MinIO исполнялись бы на отдельных серверах и были бы доступны в корпоративной сети, но вы строите модель настоящей системы, и все её компоненты заперты внутри своих контейнеров в вашем компьютере. Поэтому корпоративную сеть, в которой все сервисы «видят» друг друга, тоже придётся эмулировать. Docker предоставляет такую возможность с помощью [Docker networking](https://docs.docker.com/network/).

В манифесте Docker Compose, который описывает сервисы MLflow Server и MinIO, есть секция:

| networks:  mlflow\_net: |
| --- |

Эта секция определяет Docker-сеть. Все сервисы, описанные в манифесте, подключены к ней. Выполните команду:

| docker network ls |
| --- |

Docker покажет список всех своих сетей. Найдите среди них ту, в имени которой есть mlflow\_net (у неё будет префикс, совпадающий с названием директории, в которой находится манифест Docker Compose). Запомните её имя.

Теперь в директории с вашим проектом (там, где находится файл ML Project) запустите ваш эксперимент. Команда должна быть такой:

| mlflow run . --experiment-name=kinopoisk --docker-args network={network\_mlflow} --build-image |
| --- |

Вместо {network\_mlflow} укажите имя вашей сети — mlflow.

Если всё сделано правильно, MLflow Server создаст новый образ, унаследованный от kinopoisk:1.0.0, скопирует в него train.py и значения нужных переменных окружения (например, AWS\_ACCESS\_KEY\_ID и AWS\_SECRET\_ACCESS\_KEY), запустит на его основе контейнер и подключит его к сети (--docker-args network={network\_mlflow}), в которой находится сам MLflow Server и его объектное хранилище MinIO. Таким образом, все контейнеры будут «видеть» друг друга, а ваш train.py сможет скачать из MinIO данные (train.csv) и подключиться к MLflow Server, чтобы журналировать метрики эксперимента и сохранять его артефакты.

#### Запускаем эксперимент прямо из репозитория

До сих пор код (train.py), на основе которого вы запускали эксперименты, находился в вашей директории. Несмотря на то что вы контролируете изменения в Git-репозитории, вы можете забыть сделать коммит или зафиксировать его на удалённом Git-сервере. Получается, что код train.py может измениться (преднамеренно или нет). Если вы этого не заметите, результаты эксперимента будут искажены.

MLflow Server позволяет запускать эксперименты на основе кода, который уже зафиксирован на удалённом Git-сервере. Вероятность изменения такого кода намного меньше, поэтому такие эксперименты более повторяемы и воспроизводимы.

Выполните команду:

| mlflow run {path\_to\_your\_git\_repo} --experiment-name=kinopoisk --docker-args network=exper\_mlflow\_net --build-image |
| --- |

Здесь {path\_to\_your\_git\_repo} — путь к вашему репозиторию на удалённом сервере.

Готово! Теперь ваш эксперимент запускается:

1. Из кода, зафиксированного в удалённом Git-репозитории.
2. В собственном изолированном Docker-контейнере с фиксированным окружением.

На этом настройка среды для повторяемых и воспроизводимых экспериментов заканчивается. Создайте новый коммит, содержащий все изменения. Создайте новый Git-тег с именем stage4. Зафиксируйте все изменения на удалённом сервере.

Рекомендуемое время выполнения — 2-3 дня.

### Формат сдачи

Задание считается выполненным, если:

1. Репозиторий содержит манифест Dockerfile, создающий окружение, в котором исполняется код эксперимента.
2. Из представленного Dockerfile собирается Docker-образ.
3. Эксперимент запускается из кода, зафиксированного в репозитории.
4. На коммите стоит соответствующий тег.

## Этап 5

Это самый творческий этап. Вам необходимо преобразовать ваш игрушечный train.py, который обучает простую модель, в код настоящего эксперимента по выбору лучшей модели, подбору её гиперпараметров и других факторов, влияющих на результат.

Используйте [Hyperopt](http://hyperopt.github.io/hyperopt/) — инструмент для автоматического подбора гиперпараметров в машинном обучении. Hyperopt работает с разными типами гиперпараметров: непрерывными, дискретными, категориальными и так далее. Это позволяет использовать Hyperopt как для оптимизации гиперпараметров одной модели (например, для модели уровня силы регуляризации логистической регрессии), так и для выбора самой модели (включая способ предобработки данных).

Опишите [пространство поиска](https://github.com/hyperopt/hyperopt/wiki/FMin#2-defining-a-search-space) (search space) для Hyperopt. Используйте разные способы предобработки данных (например, удаление стоп-слов), разные методы векторизации (например, CountVectorizer или TF-IDF), разные модели классификации — и для каждой задавайте диапазоны, в которых могут изменяться их гиперпараметры.

Создайте [целевую функцию](https://github.com/hyperopt/hyperopt/wiki/FMin#1-defining-a-function-to-minimize) (objective function), которую будет оптимизировать Hyperopt. Каждый вызов целевой функции — это набор параметров из вашего пространства поиска, которые выбрал для вас Hyperopt. Используйте эти параметры, чтобы обучить модель и вычислить метрики её качества. Другими словами, каждый вызов целевой функции — это эксперимент. Запротоколируйте все результаты эксперимента в MLflow и сохраните его главный артефакт — модель машинного обучения.

На этом этапе вам предстоит серьёзно переработать код train.py. Когда закончите, создайте новый коммит, содержащий все изменения. Создайте новый Git-тег с именем stage5. Зафиксируйте все изменения на удалённом сервере.

Рекомендуемое время выполнения — две недели.

### Формат сдачи

Задание считается выполненным, если:

1. Репозиторий код проведения экспериментов на основе библиотеки Hyperopt.
2. Эксперимент запускается из кода, зафиксированного в репозитории.
3. На коммите стоит соответствующий тег.

## Дедлайн сдачи проекта

24 января 2024 года.

## Программное обеспечение

* Docker.
* Docker Compose.
* Python 3.10.
* MLflow (Tracking server + Project).
* MinIO.
* MySQL.
* Hyperopt.

1. Amazon S3 — объектное хранилище, предоставляемое коммерческим публичным облаком Amazon Web Services (AWS). Amazon S3 настолько популярно, что его API де-факто стало стандартом API для облачных хранилищ. [↑](#footnote-ref-0)