Структура проекта

Проект состоит из двух основных компонентов:

- FastAPI сервиса для обработки данных, обучения и хранения моделей;
- Streamlit приложения для предоставления пользовательского интерфейса.

Ниже описана структура проекта и назначение каждого компонента.

FastAPI сервис

FastAPI сервис отвечает за предоставление API для обработки данных, обучения моделей машинного обучения и их хранения, а также для взаимодействия с этими моделями.

```
fastapi/
|── main.py              # Основной файл FastAPI приложения
|── data/                # Папка с данными для обучения моделей и стоп-словами
|   ├──  data.csv        # Файл с данными для обучения моделей
|   └── stopwords.txt    # Файл со стоп-словами
├── models/              # Папка с моделями сохраненными в формате pickle
|   ├── Model_LR.pkl     # Лучшая линнейная модель полученная на прошлом этапе
|   └── Model_SVC.pkl    # Лучшая модель метода опорных векторов
├── tests/               # Папка с тестами
|   └──  test.py         # Функции для тестирования сервиса
├── Dockerfile           # Dockerfile для контейнеризации FastAPI сервиса
├── extract-openapi.py   # файл для экспорта openapi.yaml
├── readme.md            # Описание проекта
├── requirements.txt     # Файл с зависимостями проекта
├── router.py            # Определение эндпоинтов для сервиса
├── models.py            # Определение pydantic моделей для запросов и ответов
|── settings.py          # Загрузка переменных окружения
|── utils.py             # Вспомогательные функции
├── .dockerignore        # Содержит список файлов и папок, которые необходимо игнорировать при сборке образа
|── openapi.yaml         # файл документации API-интерфейса
└── extract-openapi.py   # файл для экспорта openapi.yaml
```

Streamlit приложение

Streamlit приложение предоставляет пользовательский интерфейс для взаимодействия с FastAPI сервисом. Пользователи могут ознакомиться с задачей проекта, выбрать данные, ознакомиться с аналитикой данных, посмотреть предсказания лучших моделей, обучить свои модели, а также сравнить их.

```
fastapi/
|── main.py                  # Точка входа в приложение
|── pages/                   # Папка со страничками приложения
|   ├── __init__.py
|   ├── analytics/           # Папка со страничками аналитики
|   |   ├── __init__.py
|   |   ├── dinamic.py       # Страница отображающая динамику ставки ЦБ
|   |   ├── length_text.py   # Страница отображающая аналитику длины текстов пресс-релизов
|   |   ├── word_cloud.py    # Страница отображающая аналитику частотности слов в текстах пресс-релизов
|   |   ├── pie.py           # Страница отображающая распределение классов в данных
|   |   └── tsne.py          # Страница отображающая тескты пресс-релизов в двумерном пространстве
|   ├── machine_learning/    # Папка со страничками обучения моделей и предсказаний
|   |   ├── __init__.py
|   |   ├── fit.py           # Страница на которой происходит обучение модели
|   |   ├── metrics.py       # Страница для просмотра метрик и сравнения моделей
|   |   └── predict.py       # Страница для просмотра предсказаний моделей
├── tools/                   # Папка со страничками данных
|   ├── __init__.py
|   ├── api.py               # Класс для взаимодействия с FastAPI сервисом
|   ├── config.py            # Файл для загрузки переменных окружения и настроки логирования
|   ├── plots.py             # Файл с функциями для построения графиков
|   └── utils.py             # Файл с различными вспомогательными функциями
|── .env                     # Определение URL сервиса
├── Dockerfile               # Dockerfile для контейнеризации FastAPI сервиса
├── check_style.sh           # Скрипт для проверки стиля кода
├── readme.md                # Описание проекта
├── requirements.txt         # Файл с зависимостями проекта
└── pyproject.toml           # Файл с зависимостями проекта для использования poetry
```

Описание функционала

FastAPI сервис

`/` Статус сервера, количество наблюдений и обученных моделей

`/fit` Обучает модель, принимает тип модели, описание и гиперпараметры. Если модель обучается, то она добавляется в память, сохранение в файл произойдет при завершении сервиса.

Для обучения выбрал две модели, показавшие лучшие данные на прошлом чекпоинте, это LogisticRegression и SVC (но на их основе можно создавать много моделей в сервисе с разными гиперпараметрами)

Трансформер для них одинаковый - Tfidf

`/get_data` Выдает все наблюдения, хранящиеся в памяти сервиса

`/get_models` Выдает описания всех обученных моделей, готовых к инференсу

`/load_data` Принимает и запоминает наблюдения. Если такое наблюдение (по дате) уже существует - то обновляет его

`/predict` Параметры:

- model_id
- response Делает предсказание указанной моделью (если параметр response не задан, то текст релиза берется из последней записи данных в памяти сервиса) Возвращает класс и вероятности классов

`/sync_data` Загружает данные из S3 хранилища

`/remove_all` Удалает все модели и данные

`/calc_metrics/{model_id}/{window}` Проводит серию обучений и предсказаний выбранной модели на части данных

- model_id модель для экспериментов
- window ширина окна на котором учится модель на каждой итерации Возвращает:
- y_preds прогнозы на каждой итерации
- y_pred_probas вероятности на каждой итерации
- y_trues реальные значения на каждой итерации

Streamlit приложение

Streamlit приложение предоставляет собой многостраничный пользовательский интерфейс. Страницы разбиты на три блока:

Главная

- О проекте - Содержит название и описание проекта, также на этой странице можно загрузить свои данные, либо выбрать данные, которые мы получаем с помощью парсинга сайта ЦБ, это актуальные данные. Также на этой странице можно взглянуть на загруженные данные.

Исследовательский анализ

- Распределение решений - можно ознакомиться с анализом распределения классов целевой переменной в загруженных данных.
- Динамика ставки - можно ознакомиться с анализом динамики ставки ЦБ, курса доллара США, а также годовой инфляции.
- Анализ длины текстов - можно ознакомиться как с общим распределением количества символов в тесктах пресс-релизов ЦБ, так и с разделением в зависимости от таргета.
- Облака слов - можно ознакомиться с анализом частотности слов в тексте пресс-релизов, также есть разделение по таргету.
- t-SNE визуализация - за счет уменьшения размерности позволяет увидеть тексты пресс-релизов в двумерном пространстве.

Машинное обучение

- Обучение модели - можно обучить модель, выбрать тип модели, описать ее и выставить гиперпараметры.
- Предсказание - можно посмотреть предсказания моделей, выбрав модель и текст пресс-релиза.
- Метрики - можно ознакомиться с метриками и сравнить модели между собой.

Инструкция по использованию

Слева находится панель навигации с помощью которой мы можем переходить на разные страницы приложения. Если панель навигации отсутствует нажать `>` в левом верхнем углу.

На странице `О проекте` можно загрузить свои данные нажав кнопку `Browse files` и выбрав локольный файл для загрузки, если файл не загружать будут использоваться актуальные данные.

На страницах из блока `Исследовательский анализ` можно ознакомиться с аналитикой загруженных данных. Интерактивных действий на этих страницах не предусмотрено. На этих страницах и во всем проекте используются интерактивные графики `plotly` при наведении на график он будет отображать более подробную информацию о данных.

На странице `Обучение модели` мы можем обучить модель задав этой модели уникальный id и описание модели. В соответсвующем пункте мы можем выбрать тип модели логистическую регрессию или метод опорных векторов. В пункте `Выберите гиперпараметры модели` мы можем выбрать следующие гиперпараметры:

- `C` коэффициент регуляризации, меньшие значения указывают на более сильную регуляризацию.
- `tol` критерий останова, чем больше тем раньше остановится обучение.
- `penalty` тип регуляризации
- `solver` метод оптимизации
- `max_iter` максимальное количество итераций
- `kernel` - тип ядра для метода опорных векторов
- `degree` - степень ядра для метода опорных векторов, при выборе `poly`
- `gamma` - коэффициент ядра для метода опорных векторов, при выборе `rbf` , `poly` или `sigmoid`

Для более полного ознакомления с гиперпараметрами моделей рекомендуется прочитать Метод опорных векторов, Логистическая регрессия

После выбора гиперпараметров следует нажать кнопку `Обучить модель` , модель начнет обучать. Как только модель обучится появится оповещение, о том, что модель обучилась.

На странице `Предсказание` мы можем выбрать модель, мы можем выбрать модель по ее `id` и увидем описание модели, гиперпараметры и тип модели. Выбрав текст пресс-релиза и нажав кнопку `Предсказать` мы увидим предсказание модели.

На странице `Метрики` в соответсвующем пункте мы можем выбрать одну или две модели. Нажав на кнопку `Получить метрики` мы запустим цикл тестирования моделей. Как только цикл завершится мы увидим матрицу ошибок для каждой модели, таблицу основных метрик и график roc-кривой для каждого класса таргета.

Система сбора логов

Дополнительно была подготовлена система для сбора логов ELK. Система состоит из следующих компонентов:

- Elasticsearch - как хранилище логов
- Logstash - анализатор и обработчик логов
- Filebeat - модуль, наблюдающий за текстовыми файлами логов и отправляющий изменения в Logstash
- Kibana - веб-интерфейс системы

Система сбора логов реализована как многоконтейнерное приложение, запускаемое с помощью Docker Compose, и работающее параллельно с основным приложением. По мере наполнения логов основным приложением, компонент Filebeat отправляет логи в Logstash, который, в свою очередь, преобразует их и сохраняет в Elasticsearch.

Разворачивание приложения в Yandex Cloud

Для каждого компонента приложения (fastapi-сервис и streamlit-приложение) был подготовлен Dockerfile для построения образа. Таким образом, проект представляет собой многоконтейнерное приложение, запускаемое с помощью Docker Compose. Для иллюстрации приложение и система сбора логов были развернуты с использованием облачного провайдера Yandex Cloud.

1. В консоли управления создаем реестр контейнеров: Все сервисы - Контейнеры - Container Registry.
2. На блид-сервере аутентифицируемся в созданном реестре (oauth-токен получаем по инструкции):

```
echo <OAuth-токен>|docker login \
  --username oauth \
  --password-stdin \
 cr.yandex
```

3. Назначаем собранным образам с приложениями теги с указанием реестра:

```
docker tag cbr-api cr.yandex/<ID реестра>/cbr-api:latest
docker tag cbr-ui cr.yandex/<ID реестра>/cbr-ui:latest
```

4. Отправляем образы в реестр:

```
docker push cr.yandex/<ID реестра>/cbr-api:latest
docker push cr.yandex/<ID реестра>/cbr-ui:latest
```

5. В консоли управления в разделе Все сервисы - Compute Cloud - Виртуальный машины создаем новую виртуальную машину, на которой будет работать приложение.
6. Подключаемся к созданному инстансу, устанавливаем Docker обычным образом, в зависимости от используемой системы.
7. Аутентифицируемся в реестре образов, как в п. 1.2.
8. Задаем необходимые настройки в `.env` -файлах (см. примеры этих файлов), копируем `docker-compose.deploy.yml` (убрав `.deploy`) и `docker-compose.elk.yml` (опционально).
9. Поднимаем приложение `docker compose up -d` и `docker compose -f docker-compose.elk.yml up -d` (опционально).