report

Описание проекта

Данный проект представляет собой приложение для предсказания авторов по тексту. Мы собрали данные о 100 наиболее популярных авторов в мировой литературе, обучили классические ML-модели, которые позволяют предсказывать авторство по тексту и сделали MVP-приложение, используя FastAPI, streamlit и Docker.

Демонстрация работы приложения

Для демонстрации работы приложения мы сделали GIF-анимацию: https://github.com/Difraya/hse_nlp_project/blob/main/images/streamlit.gif

Если нужно более высокое качество и более детальная демонстрация, то можно посмотреть видео: https://github.com/Difraya/hse_nlp_project/blob/main/images/streamlit.m4v

Структура проекта

Весь проект, включая приложение и research имеет следующую структуру:

```
C:.
├───Baseline
├───data_mining
│   └───parsers
├───EDA
├───FastAPI
│   ├───logs
│   └───__pycache__
├───streamlit
└───temp
```

В папках `EDA` и `Baseline` можно найти все, что связано с research'ем, включая анализ данных и обучение ML-моделей, которые будут использоваться в приложении. В папках `FastAPI` и `streamlit` содержится все, что связанно с приложением: backend и fronted соответственно.

В FastAPI предоставляются следующие endpoint'ы:

- `GET / Read Root` - корневой endpoint, который позволяет предсказывать авторов

по тексту, используя # AJAX Interface.

- `GET /ModelsList` - возвращает список доступных моделей с описанием. На данный момент загружено 4 модели.
- `POST /setModel` - позволяет установить модель в качестве активной для инференса или обучения.
- `POST /PredictItem` - предсказывает автора короткого текст, который передается в JSON.
- `POST /PredictItemFile` - предсказывает автора короткого текст, который передается через файл формата `.txt`.
- `POST /PredictItemProba` - предсказывает вероятность авторов по короткому тексту, который передается через `.json`.
- `POST /PredictItemProbaFile` - предсказывает вероятность авторов по короткому тексту, который передается через файл формата `.txt`.
- `POST /PredictItems` - предсказывает автором не одного, а нескольких текстов.
- `POST /PredictItemsProba` - предсказывает вероятности авторов не одного, а нескольких

текстов.

- `POST /train_model` - позволяет обучать модели, используя новые данные, а также строить кривые обучения.
- `POST /partial_fit` - позволяет дообучать модели новыми данными.
- `POST /fine_tuning` - Выполняет fine-tuning активной модели, используя новые данные. Параметры: request_file: Файл, содержащий новые обучающие данные. Возвращает: Сообщение, подтверждающее успешное обновление всех моделей.

Все эти endpoin'ы обернуты в фронтед, сделанный на streamlit'e.

Краткое руководство по запуску приложения

MVP-версия приложения для предсказания авторов по тексту. Чтобы приложение работало нужно:

1. клонировать содержимое репозитория;
2. установить зависимости из `requirements.txt`;

3. запустить скрипт `download_files.py` , чтобы скачать необходимые данные;
4. пользоваться приложением.

Если нужно запустить приложение через докер, смотрите руководство ниже.

Docker: инструкция

Для создания docker-образа и запуска контейнеров с приложениями, в операционной системе Windows необходимо:

1. Запустить Docker engine.
2. Пройти в папку streamlit и заменить переменную в 20-й строке файла streamlit_nlp.py, на: API_URL = "http://fastapi:8000".
3. Перейти в корневой каталог склонированного репозитория запустить команду docker-compose up.
4. Дождаться запуска контейнеров и перейти по адресу 127.0.0.1:8501- для streamlit, или 127.0.0.1:8000 - для FastAPI.

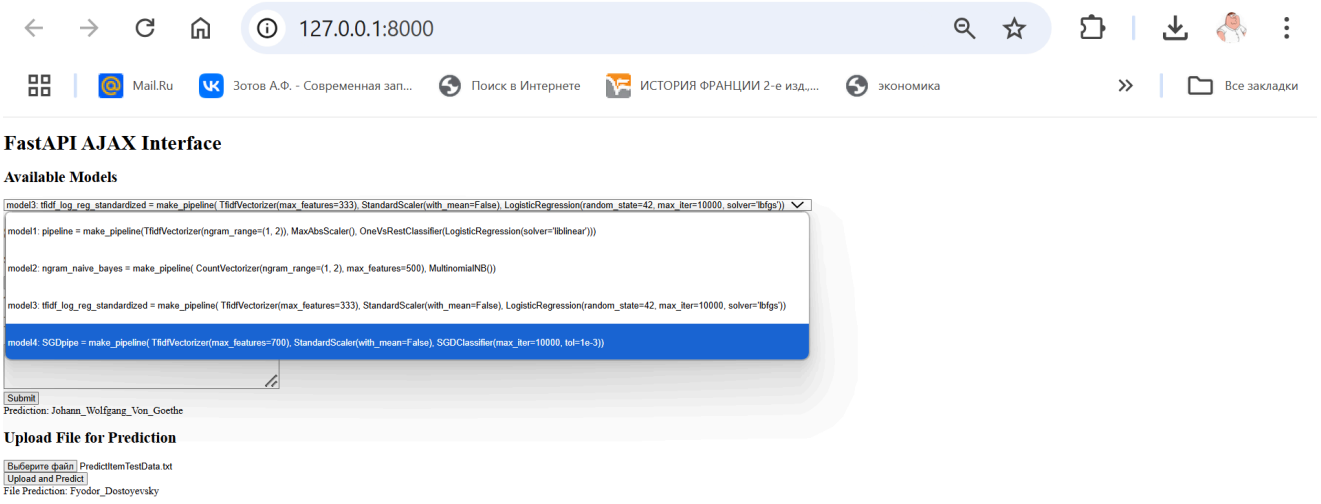
Далее можно пользоваться приложением.

Основные endpoint'ы FastAPI

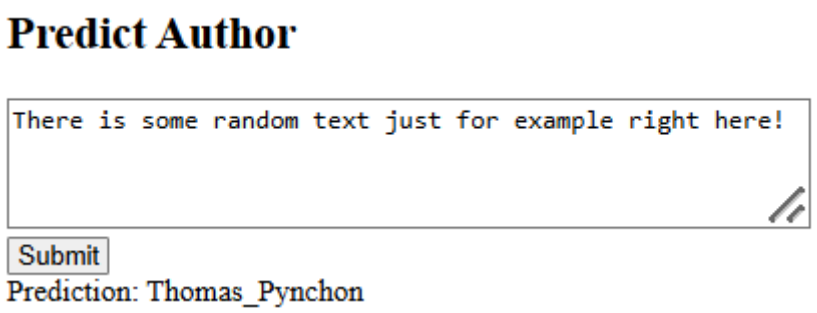
`/ (root)`

В корневом маршруте реализован базовый **FastAPI AJAX Interface** без перезагрузки страницы, который позволяет выбрать модель из обученных, быстро сделать предсказание автора по вручную написанному тексту или по файлу. Вот пример:

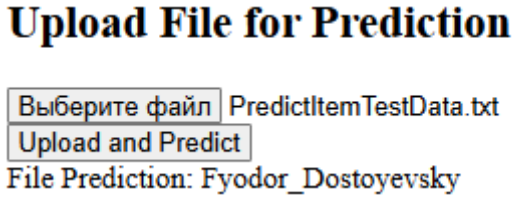
Мы мы можем выбрать модель из выпадающего списка:



Затем можно ввести какой-то текст и получить предсказание:



Если текст большой, то можно загрузить текст файлом и сделать предсказание:



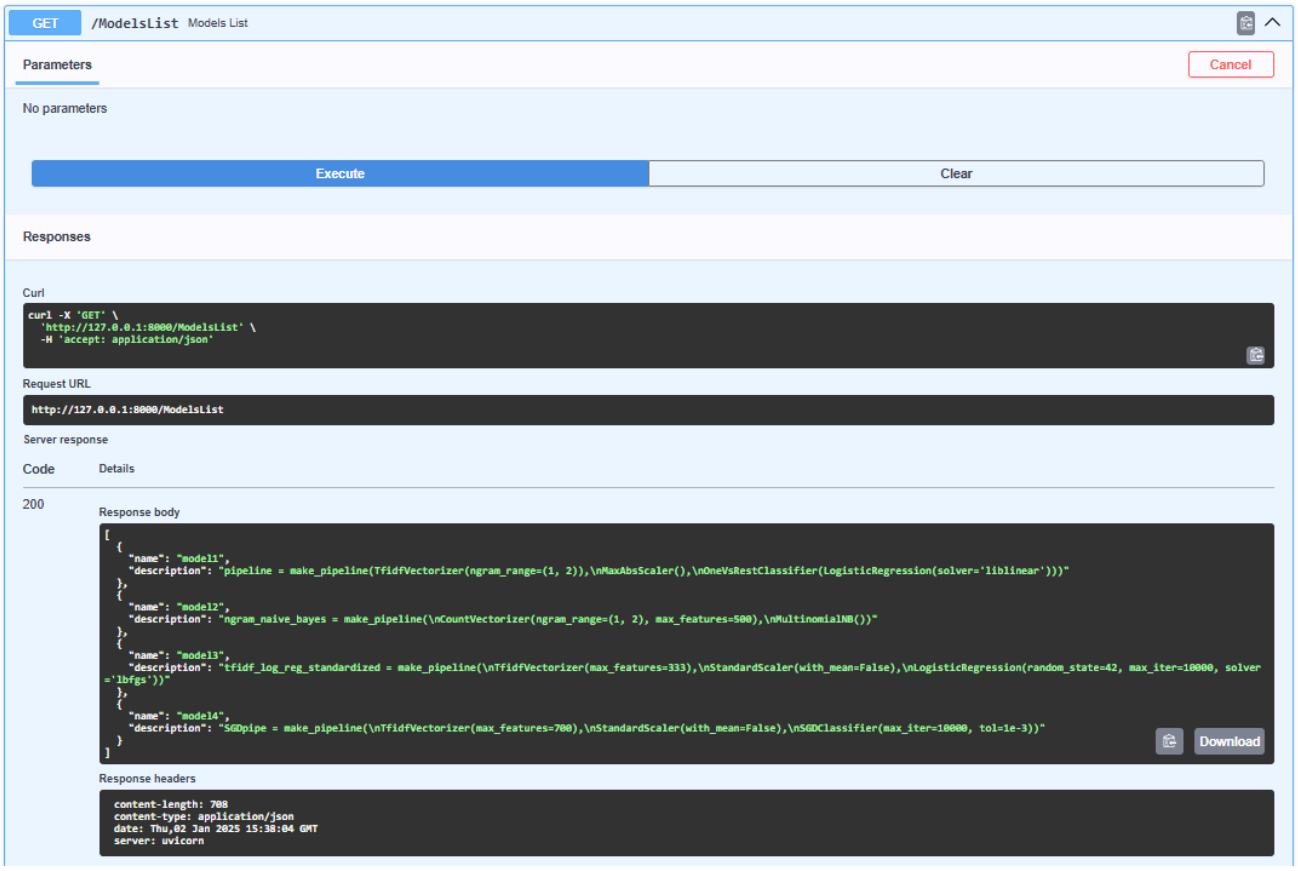
На вход принимается текст, далее нужно нажать на кнопку `Submit`. После этого получим предсказание автора. В данном случае - Достоевского.

`/ModelsList`

На сервер изначально загружено 4 уже обученных модели-пайплайна с автоматической обработкой текста, удалением стоп-слов, токенизацией и возможностью предсказания.

Чтобы посмотреть все изначально загруженные модели с уникальным ID и описанием, достаточно

отправить `GET`-запрос по маршруту `/ModelsList`:

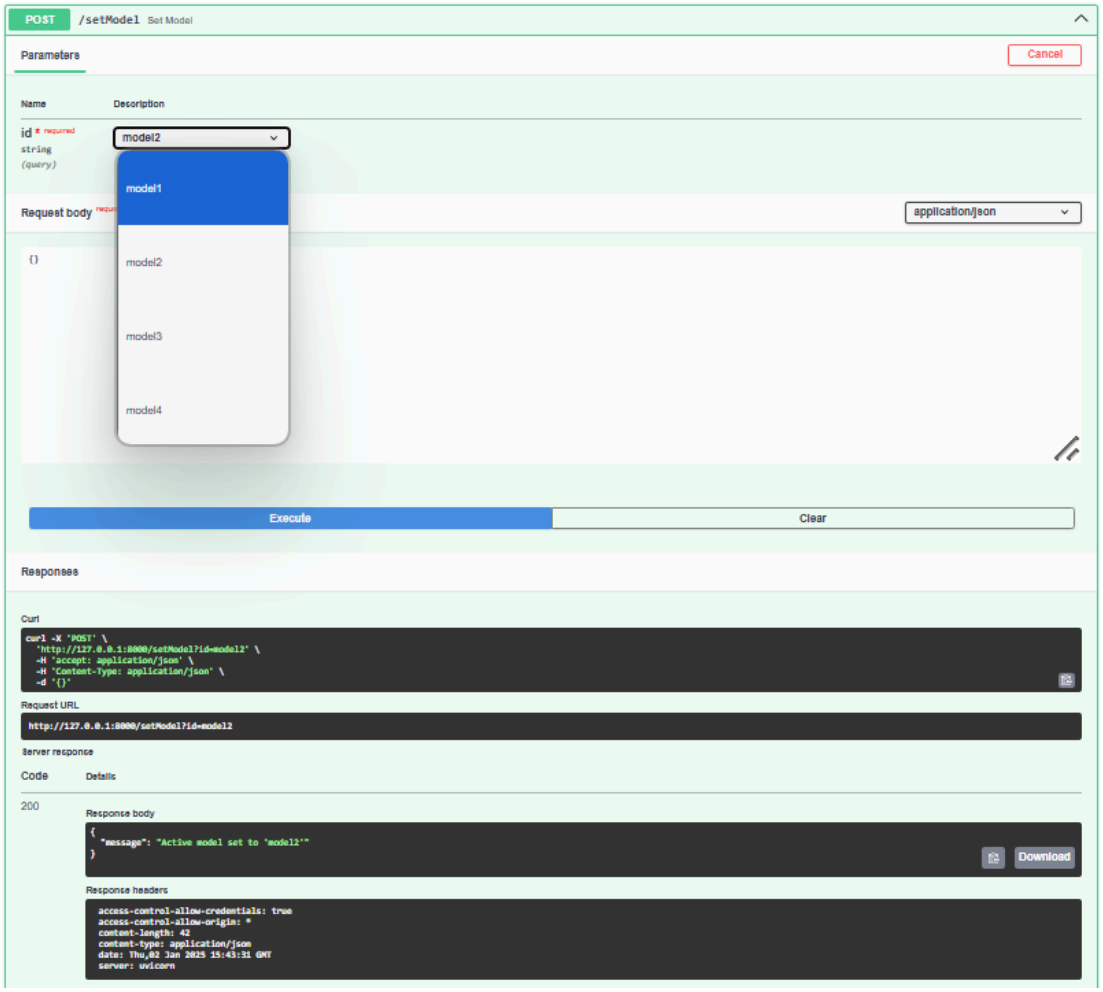


Можно видеть структуру пайплайнов, что позволяет пользователю выбрать подходящую модель и подходящие для нее гиперпараметры.

`/SetModels`

Чтобы использовать какую-либо модель для inference'а или для переобучения/дообучения, нужно отправить `POST`-запрос по маршруту

`/setModel` , указав уникальный `id` модели:



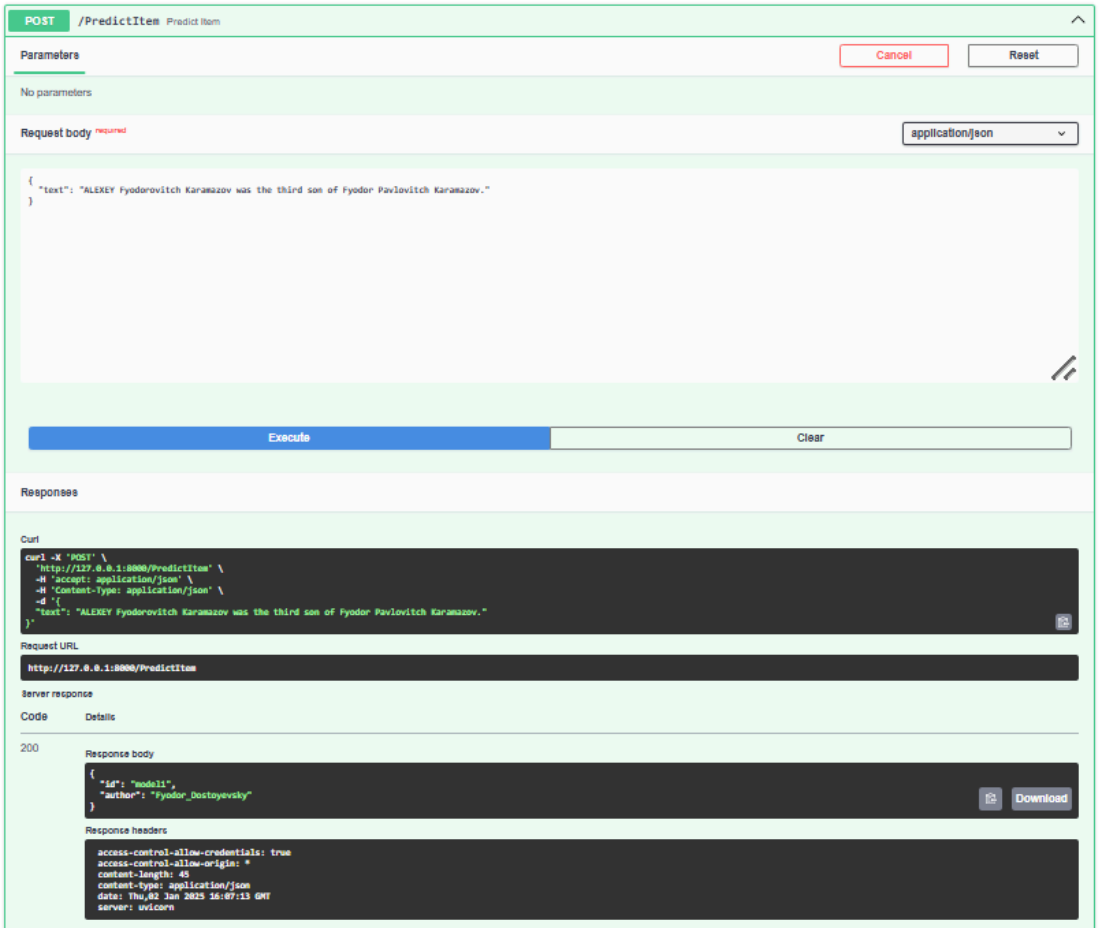
Если модель была сделана активной, то пользователю вернется соответствующие сообщение и код `200` .

`/PredictItem`

Чтобы предсказать автора по одному тексту (как правило, короткому тексту, т.к. для длинных текстов реализованы endpoint'ы для приема файлов), нужно отправить `POST` -запрос по маршруту `/PredictItem` , отправив текст в формате json:

```
{
        "text": "<какой-то текст>"
}
```

Например, можно отправить какой-то текст Достоевского и посмотреть на предсказания:

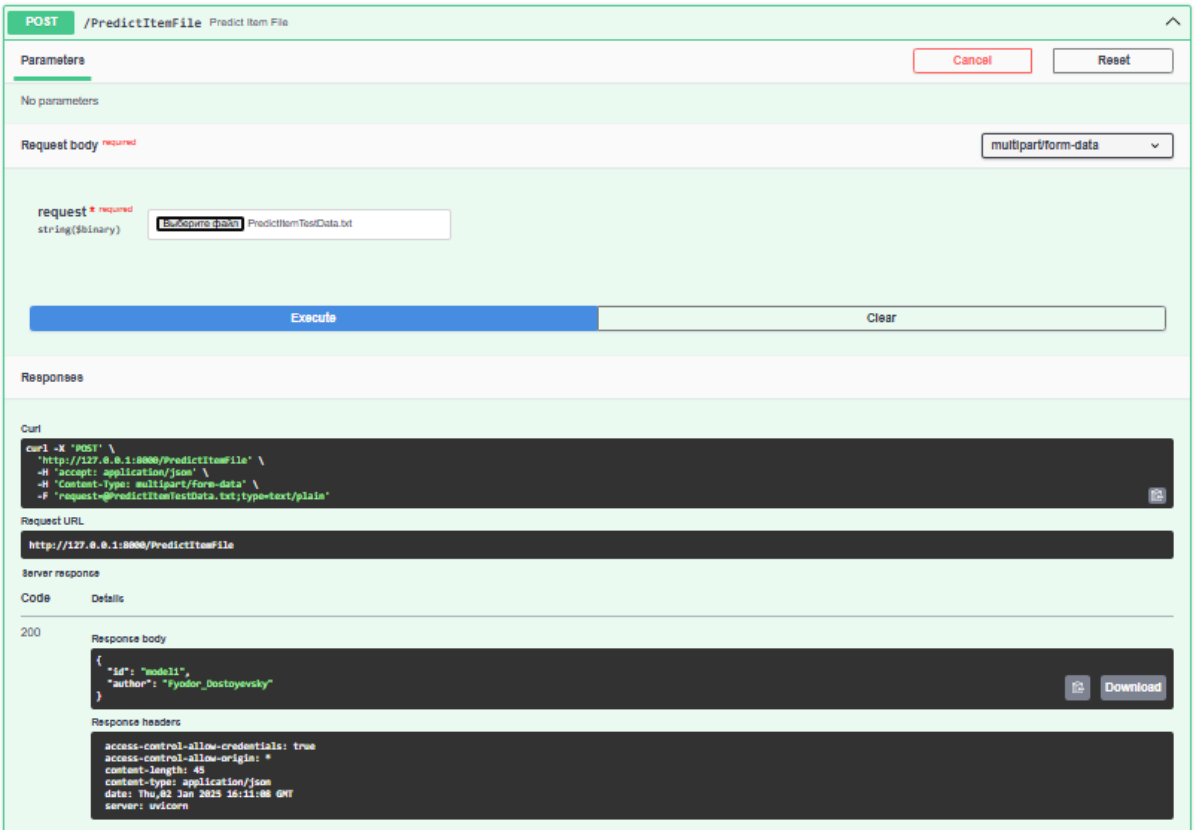


На выходе получаем response в формате JSON следующей структуры:

```
{
        "id": "<id модели>",
        "author": "<автор>"
}
```

`/PredictItemFile`

Данный endpoint нужно использовать, когда текст для предсказания слишком большой. Чтобы совершить предсказание, нужно совершить `POST`-запрос по маршруту `/PredictItemFile`, отправив файл с текстом в формате `.txt`. Вот пример запроса с файлом `PredictItemTestData.txt`:



В результате получаем response в формате JSON такой структуры:

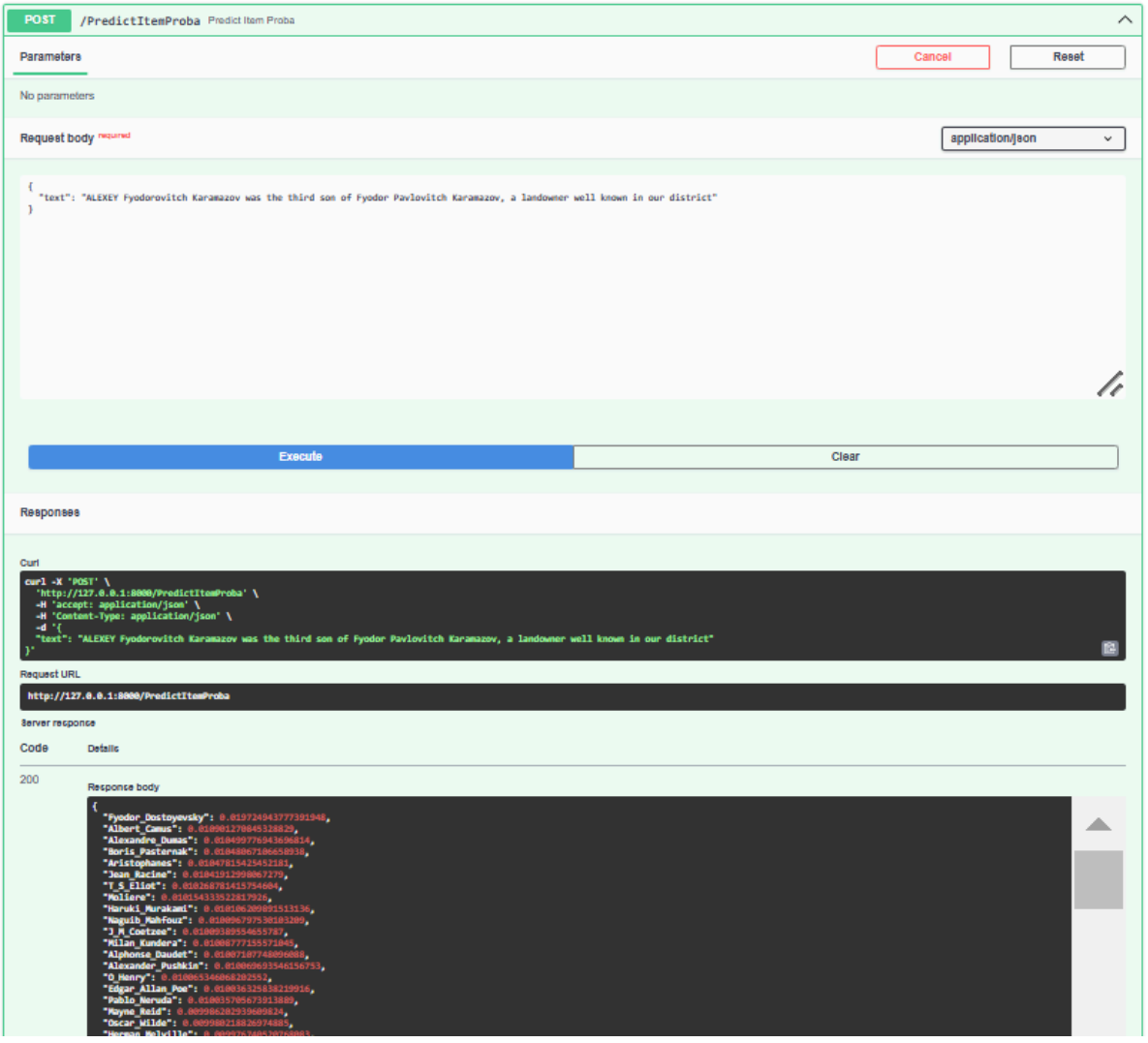
```
{
        "id": "<id модели>",
        "author": "<автор>"
}
```

`/PredictItemProba`

Чтобы предсказать вероятности возможных авторов по короткому тексту, нужно отправить `POST`-запрос по маршруту `/PredictItemProba` с JSON следующего формата:

```
{
        "text": "<какой-то текст>"
}
```

Вот пример отправки текста для оценки вероятностей:



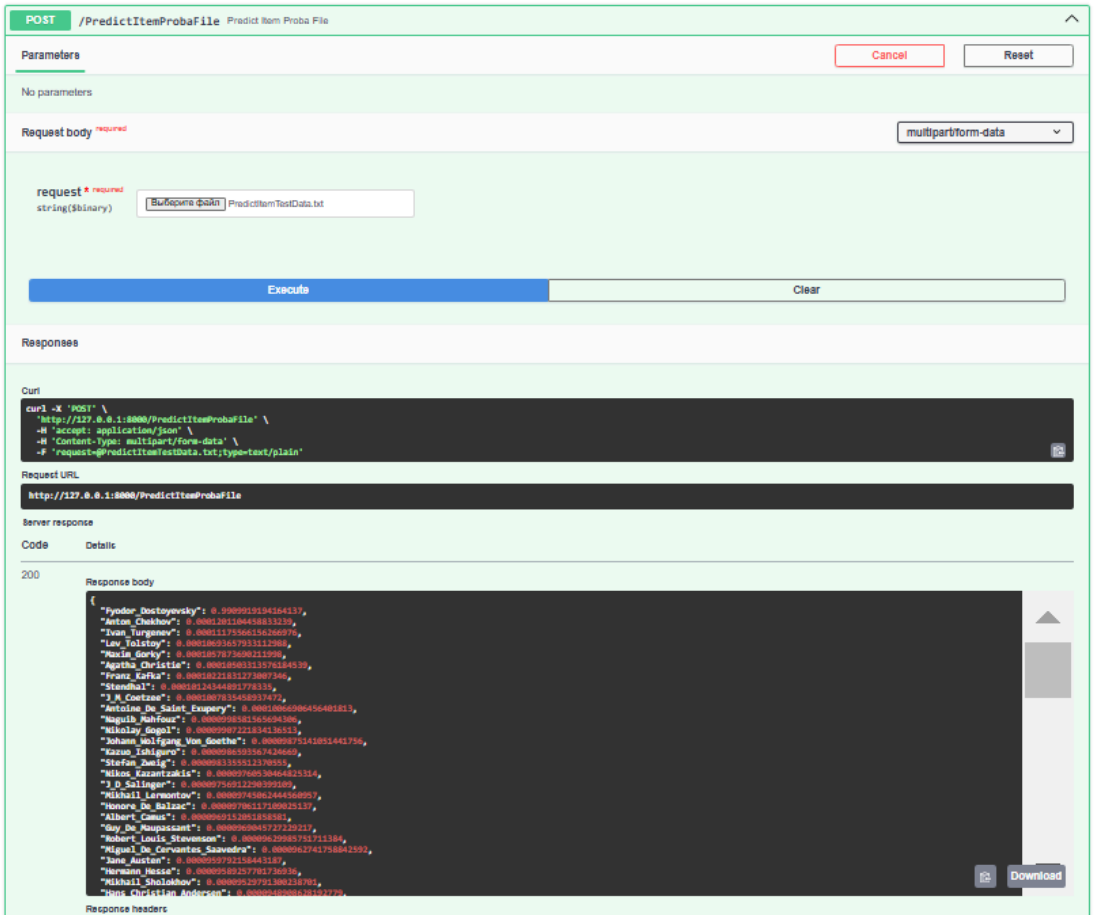
На выходе получается JSON-response с вероятностями для каждого автора из обучающей выборки, который имеет такую структуру:

```
{
        "<имя автора 1>": <float со значением вероятности автора1>,
        "<имя автора 2>": <float со значением вероятности автора2>,
        ...
}
```

`/PredictItemProbaFile`

Этот endpoint используется для предсказания вероятностей авторов по тексту, оформленному в файл формата `.txt`. Нужно отправить `POST`-запрос с соответствующем файлом.

Пример отправки отрывка теста Достоевского в файле `PredictItemTestData.txt` :



В результате получаем ответ в формате JSON:

```
{
        "<имя автора 1>": <float со значением вероятности автора1>,
        "<имя автора 2>": <float со значением вероятности автора2>,
        ...
}
```

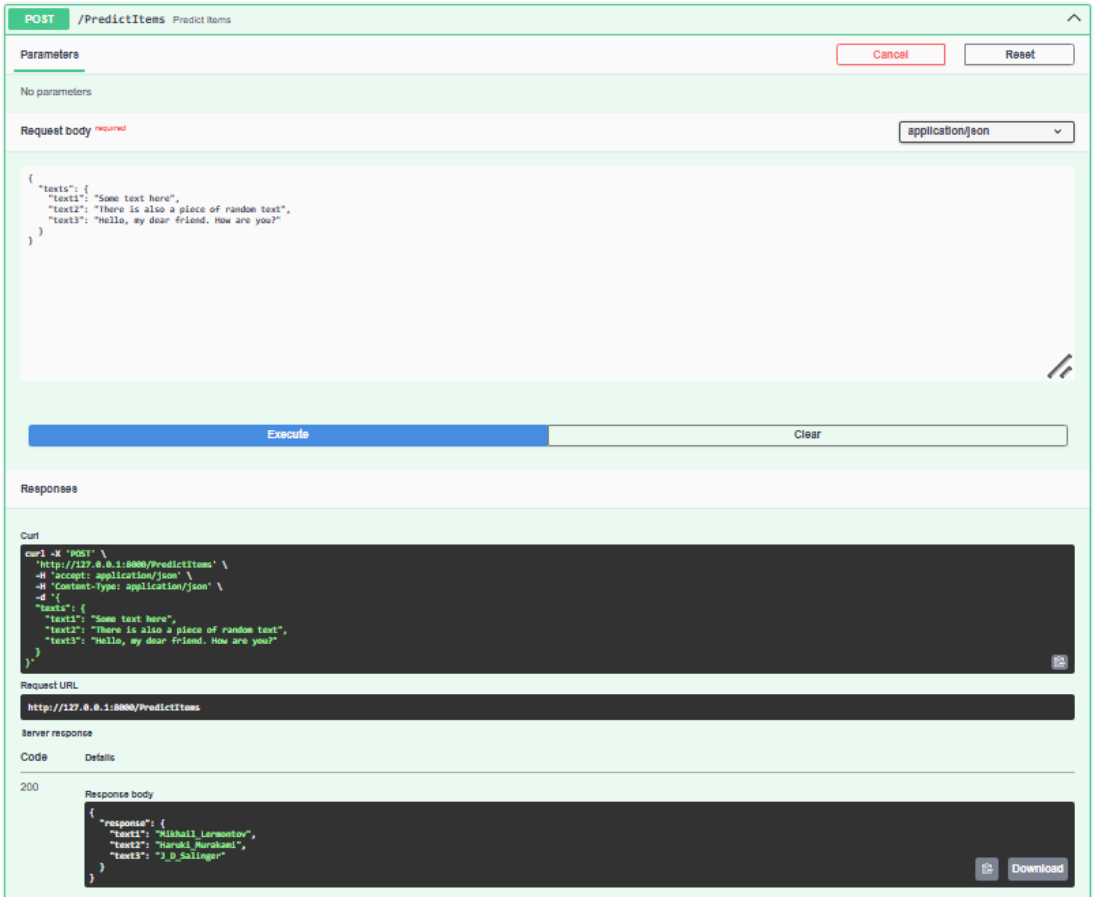
`/PredictItems`

Если нужно предсказать авторов сразу нескольких коротких текстов, используется `POST`-запрос по

маршруту `/PredictItems`. В качестве request'а нужно передать JSON следующей структуры:

```
{
  "texts": {
    "<id текста1>": "<текст1>",
    "<id текста2>": "<текст2>",
    "<id текста3>": "<текст3>"
  }
}
```

Вот пример отправки подобного запроса:



В результате получается response в таком формате:

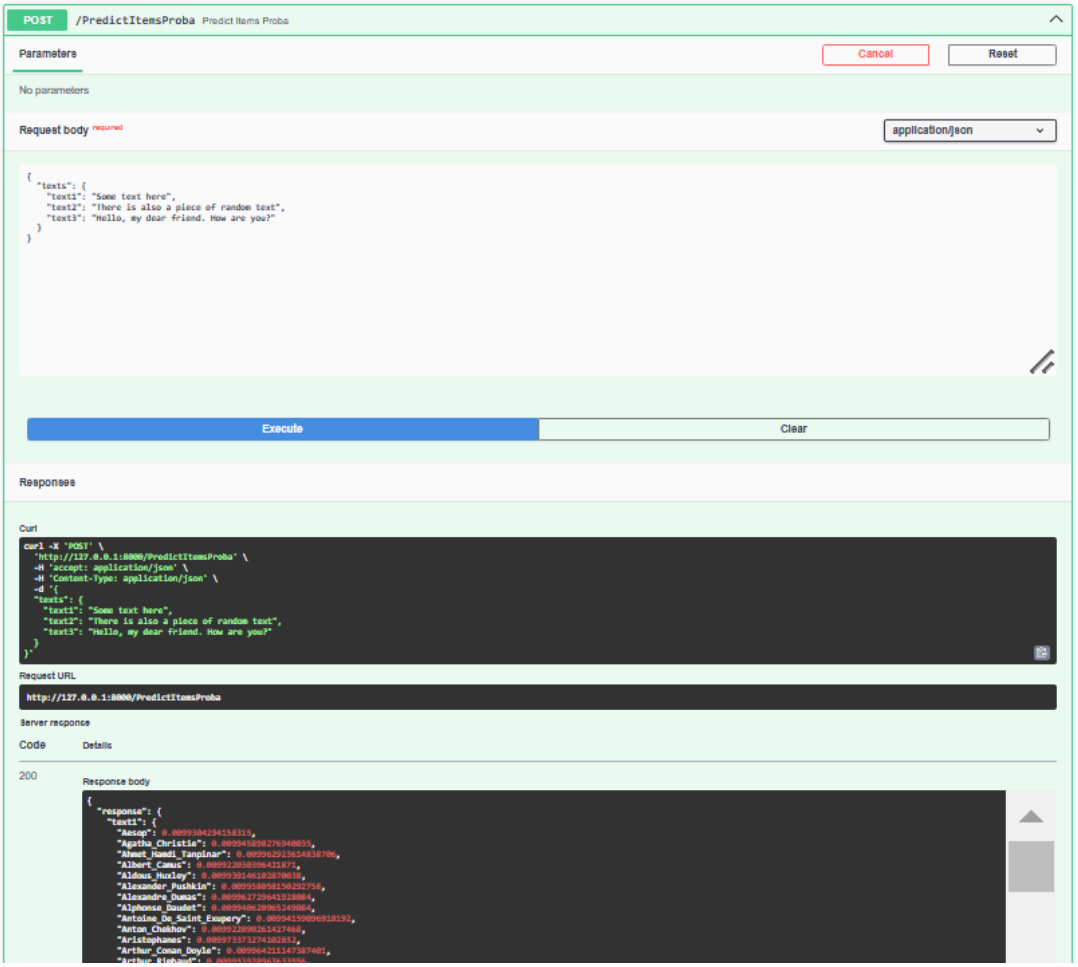
```
{
  "response": {
    "<id текста1>": "<автор текста1>",
    "<id текста2>": "<автор текста2>",
    "<id текста3>": "<автор текста3>",
    ...
  }
}
```

`/PredictItemsProba`

Чтобы предсказать вероятности авторов для нескольких текстов, нужно отправить `POST`-запрос с JSON'ом следующей структуры:

```
{
  "texts": {
    "<id текста1>": "<текст1>",
    "<id текста2>": "<текст2>",
    "<id текста3>": "<текст3>",
    ...
  }
}
```

Вот пример отправки запроса со случайными текстами:



В качестве response'а получается JSON следующей структуры:

```
{"response":
        {"<текст1>": {"<автор1>": <вероятность автора1>,
                                    "<автор2>": <вероятность автора2>,
                                    ...
                                    },
        },
        {"<текст2>": {"<автор1>": <вероятность
```

```
автора1>,
                                        "<автор2>":
<вероятность автора2>,
                                        ...
                                        },
        },
        ...
}
```

`/train_model`

Данный endpoint предназначен для переобучения (refit'a) модели из списка, а также возвращения значений кривых обучения. Он принимает следующие параметры, передаваемые `POST`-запросом:

- **`request`** (`str`): строка-json с гиперпараметрами для модели в формате, указанном ниже;

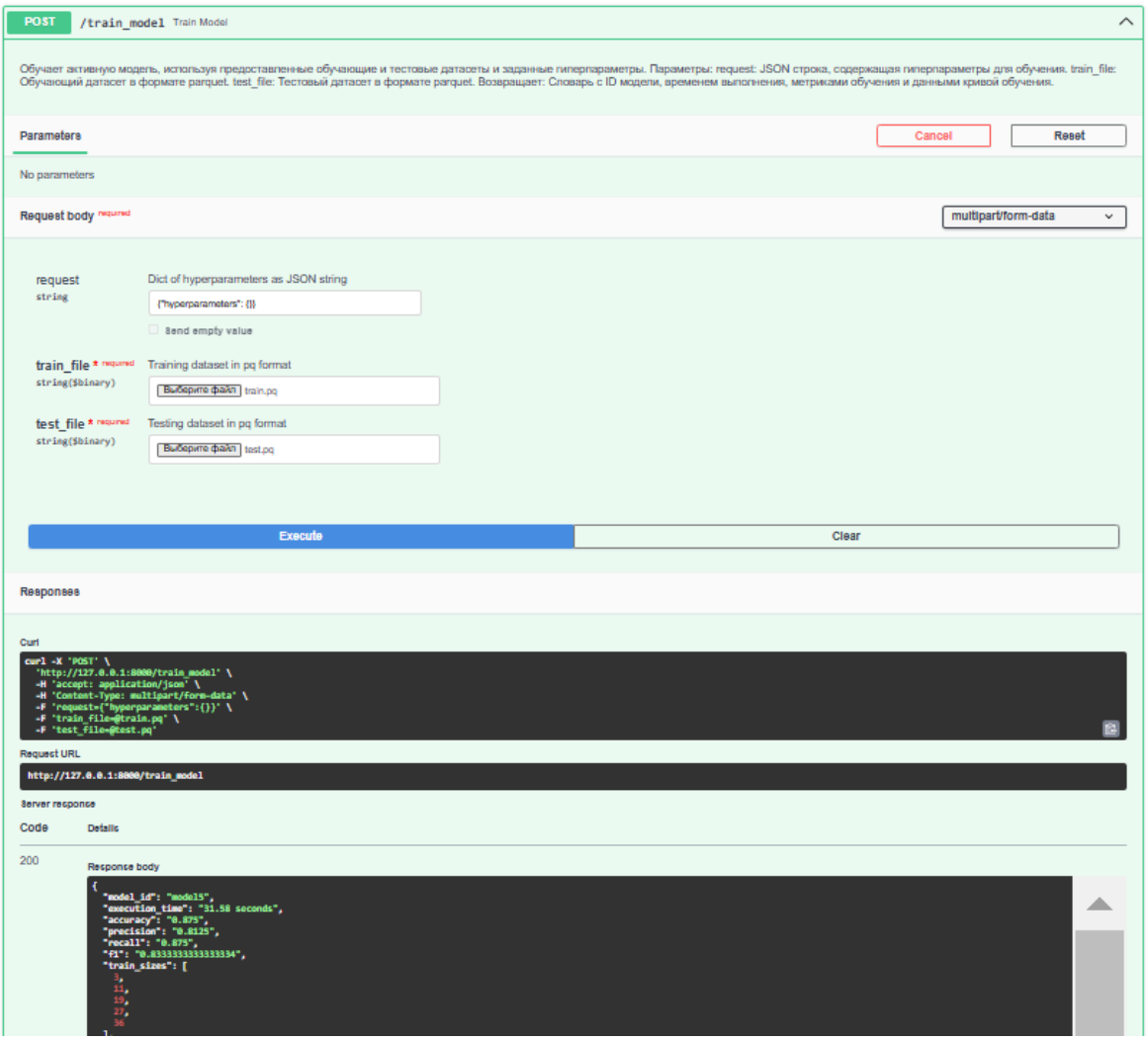
```
{"hyperparameters":
        {
        "<гиперпараметр1>": <значение
гиперпараметра 1>,
        "<гиперпараметр2>": <значение
```

```
гиперпараметра 2>,
        ...
        }
}
```

- **`train_file`** : тренировочный датасет формата `.parquet` , который должен иметь 2 столбца (`text` - текст автора и `author` - имя автора, целевая переметра);
- **`test_file`** : тестовый датасет формата `.parquet` , который также имеет 2 столбца (`text` - текст автора и `author` - имя автора, целевая переменная).

Важно: нужно передавать данные, в которых книг каждого автора больше 1, чтобы кривые обучения корректно отработали.

Вот пример запроса:



В качестве response'a возвращается JSON следующей структуры:

```
{
        "model_id": "<id модели>",
        "execution_time": "<время>",
        "accuracy": "<значение accuracy>",
        "precision": "<значение precision>",
        "recall": "<значение recall>",
        "f1": "<значение f1>",
        "train_sizes": [...],
```

```
        "train_scores_mean": [...],
        "test_scores_mean": [...]
}
```

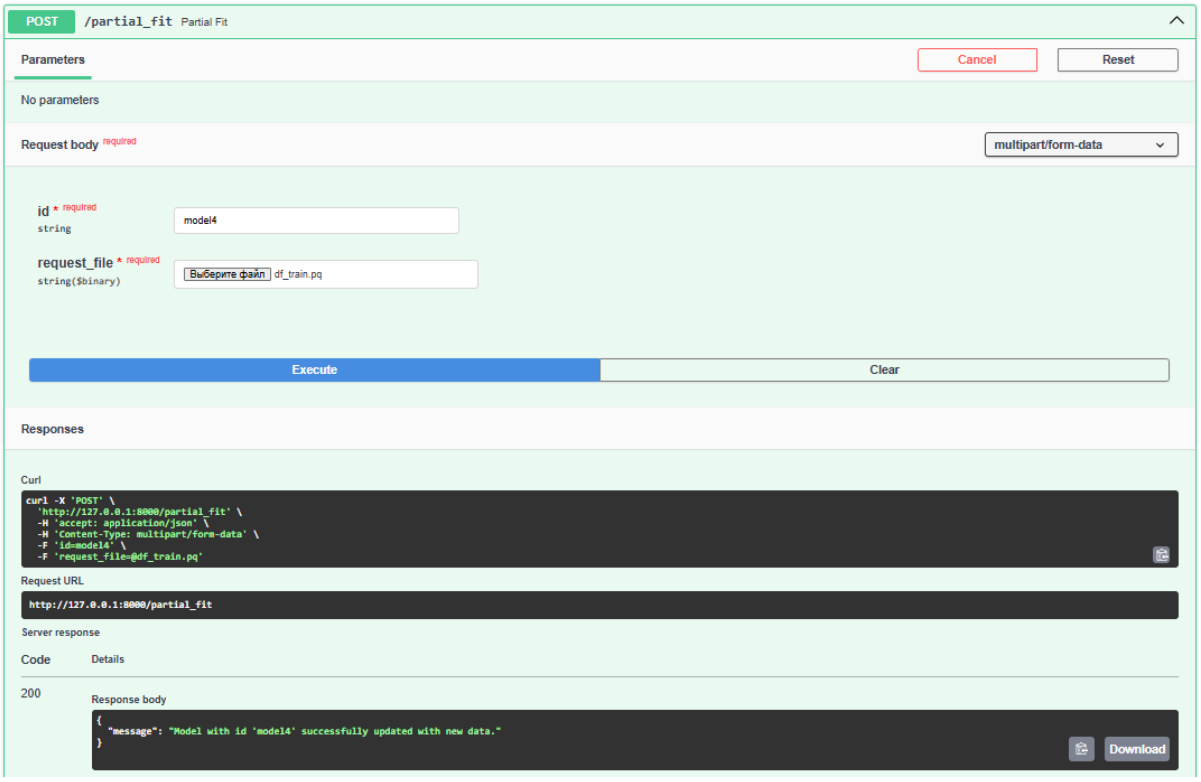
То есть модель переобучна, её результативность можно увидеть. Также можно построить кривые обучения.

`/partial_fit`

Данный endpoint предназначен для дообучения модели. На данный момент дообучение возможно только для `model4`. Чтобы дообучить модель, нужно передать в `POST`-запрос следующие параметры:

- `id` - строка с `id` модели, которая будет дообучаться;
- `request_file` : файл формата `.parquet` , имеющий 2 столбца (`text` - текст автора и `author` - имя автора, целевая переменная).

Вот пример для обучения:

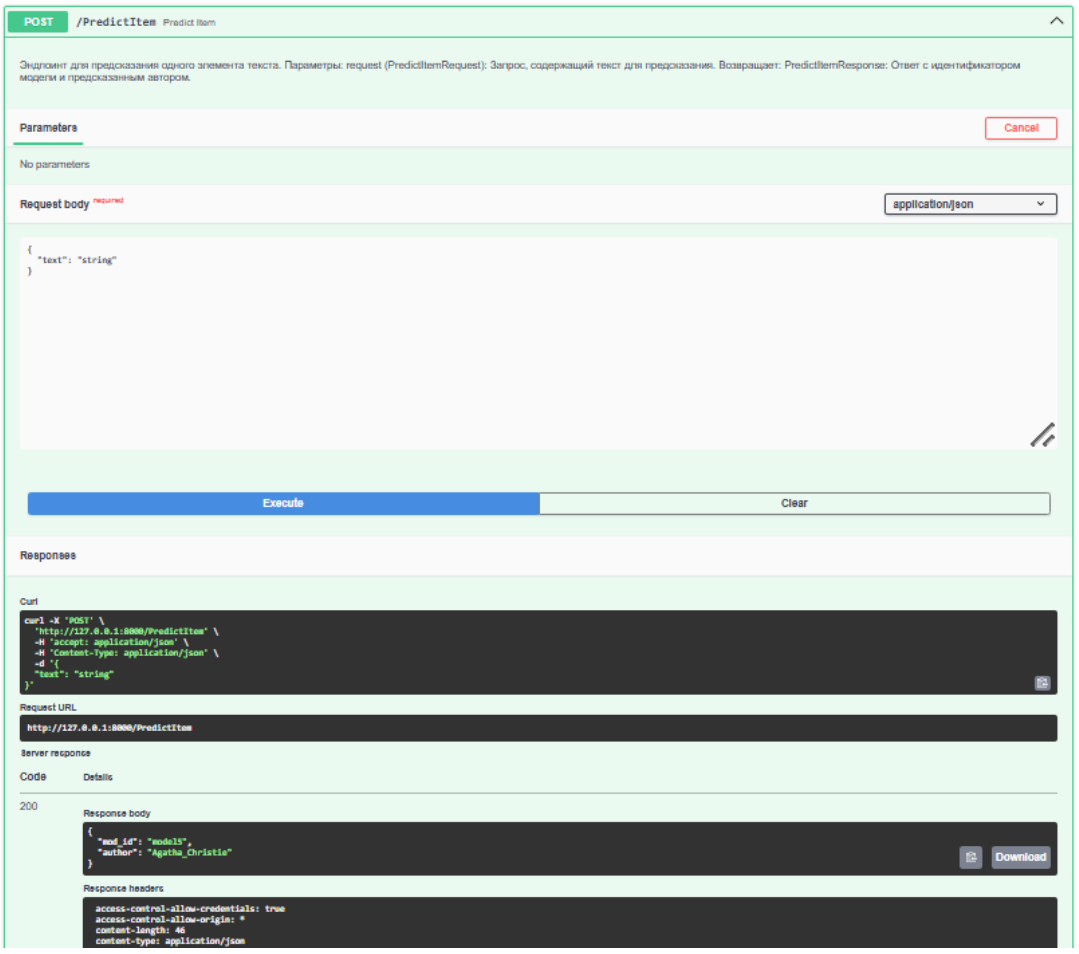


В качестве response приходит JSON такого формата:

```
{"message": "Model with id '<id модели>' successfully updated with new data."}
```

После обучения можно увидеть, что новая модель доступна теперь для инференса. Можно

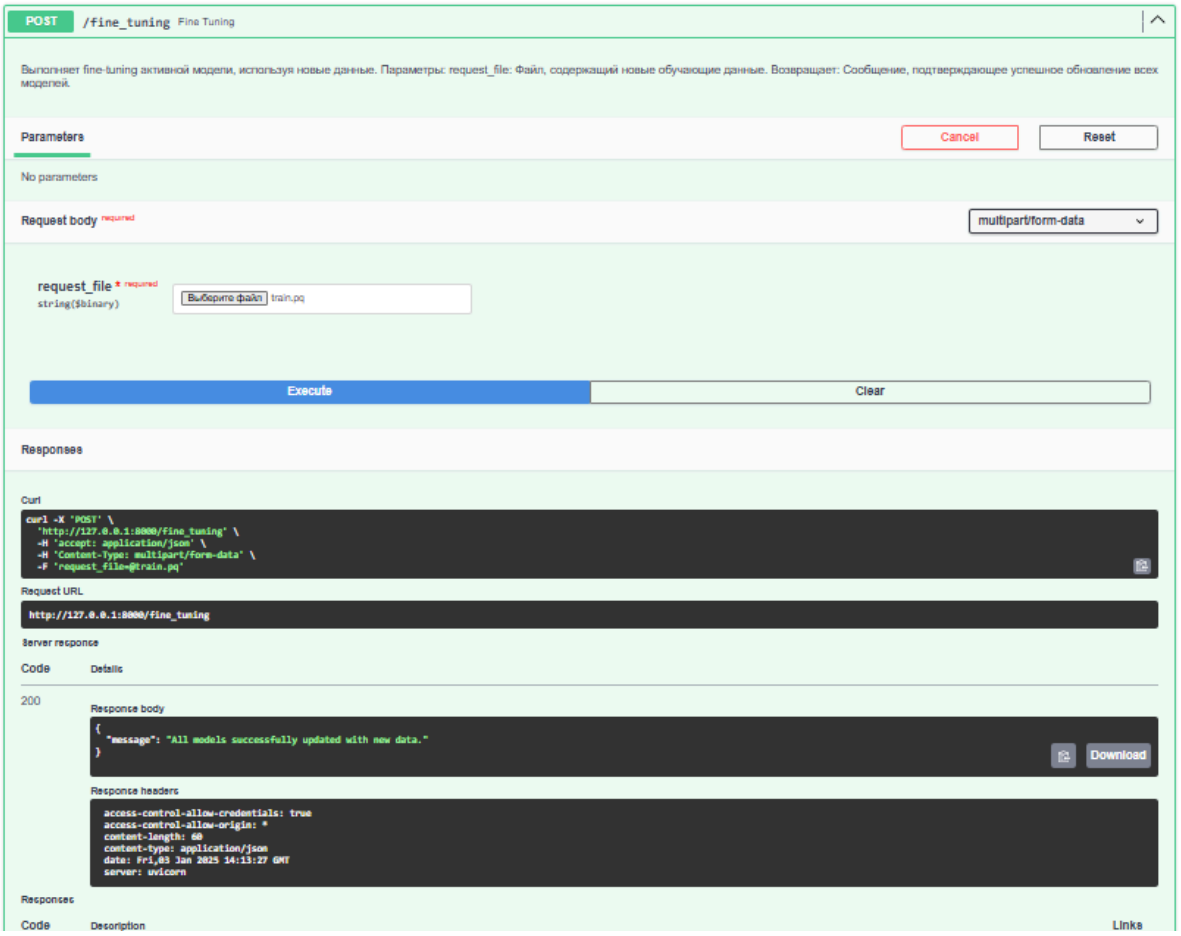
выбрать её и сделать предсказание:



`/fine_tuning`

Чтобы дообучить все модели новыми данными, нужно использовать этот endpoint. Нужно просто прикрепить файл с новыми данными в формате `.parquet`. В нем должны быть колонки `text` и `author`.

Вот пример запроса:



Теперь все модели переобучены. То есть пользователь может легко обучить все модели на своих данных.

Streamlit

Краткое описание

Наше Streamlit-приложение предоставляет удобный интерфейс для анализа текстов и предсказания их авторства. Оно включает в себя базовые инструменты для обработки текста, визуализации, а также функции предсказания и

обучения/дообучения моделей машинного обучения. Приложение ориентировано на работу с текстовыми данными и поддерживает выбор готовых моделей, а также загрузку пользовательских данных для дообучения.

Системные требования

Главные системные требования:

- Python 3.12 или выше.
- Streamlit 1.20 или выше.
- NLTK и дополнительные пакеты (punkt, stopwords, averaged_perceptron_tagger).
- Установленный сервер API с поддержкой предсказаний.

Установка зависимостей:

```
pip install -r requirements.txt
```

Запуск приложения:

```
streamlit run streamlit_nlp.py
```

Примеры команд для API
Предсказание авторства текста:

```
curl -X POST
"http://127.0.0.1:8000/PredictItem"
         -H "Content-Type: application/json"
         -d '{"text": "Пример текста"}'
```

Загрузка файла и предсказание:

```
curl -X POST
"http://127.0.0.1:8000/PredictItemFile"
         -F "request=@example.txt"
```

Техническая реализаций (Языки и библиотеки)

Приложение написано на Python 3.12+ с использованием следующих библиотек:

- **`Streamlit`** — создание веб-интерфейса.
- **`NLTK`** — обработка текста (токенизация, POS-теги, n-граммы).
- **`Pandas`** — работа с табличными данными.
- **`Plotly`** — построение интерактивных графиков.

- `Requests` — взаимодействие с REST API.
- `JSON` — обработка гиперпараметров.

Архитектура

Приложение разделено на три основные вкладки, каждая из которых реализует свою логику:

I. Пользовательская часть
Функционал:

- Анализ текста, включая:
- Подсчёт количества слов, уникальных слов и знаков препинания.
- Построение гистограмм по частям речи и n-граммам (1–3).
- Преобразование текста с удалением стоп-слов для подготовки данных.
- Интеграция с API для предсказания авторства текста и вероятностных оценок.
- Поддержка загрузки файлов в формате .txt для пакетной обработки.

Ключевые функции:

- `get_top_words` — получение наиболее

частотных слов.
- `get_ngrams` — генерация n-грамм.
- `most_common_ngrams` — подсчёт самых популярных n-грамм.
- `del_stopwords` — удаление стоп-слов.
- `get_pos` — анализ частей речи.
- `punctuation` — подсчёт знаков препинания.

API-интеграция:

- POST-запросы для передачи текста или файла в REST API и получения предсказаний.

II. Информация про модели и данные

Функционал:

- Запрос списка доступных моделей с описанием.
- Установка активной модели для работы с текстами.
  API-интеграция:
- GET-запрос для получения списка моделей.
- POST-запрос для выбора активной модели.

III. Обучи свою модель

Функционал:

- Загрузка пользовательских данных в формате .parquet.
- Настройка гиперпараметров через текстовое поле (JSON).
- Вызов API для обучения новой модели.
- Поддержка дообучения (partial_fit) на новых данных.
  API-интеграция:
- POST-запросы с файлами данных и гиперпараметрами для обучения.
- Обработка ошибок в API и вывод логов для отладки.

Приложение работает с REST API по адресу http://127.0.0.1:8000. Основные эндпоинты можно посмотреть выше.

Форматы данных:

- Печатный текст
- JSON для текстов и параметров.
- Multipart/form-data для файлов.