

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:.\n|---Baseline\n|---data_mining\n|   |---parsers\n|---EDA\n|---FastAPI\n|   |---logs\n|   |---__pycache__\n|---streamlit\n|---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 связано с приложением: backend и frontend 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 текста, который передается в JSON.
- `POST /PredictItemFile` - предсказывает автора короткого текста, 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 `endpoints` обернуты в фронтенд, сделанный на `streamlit`.

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 клонированного 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.

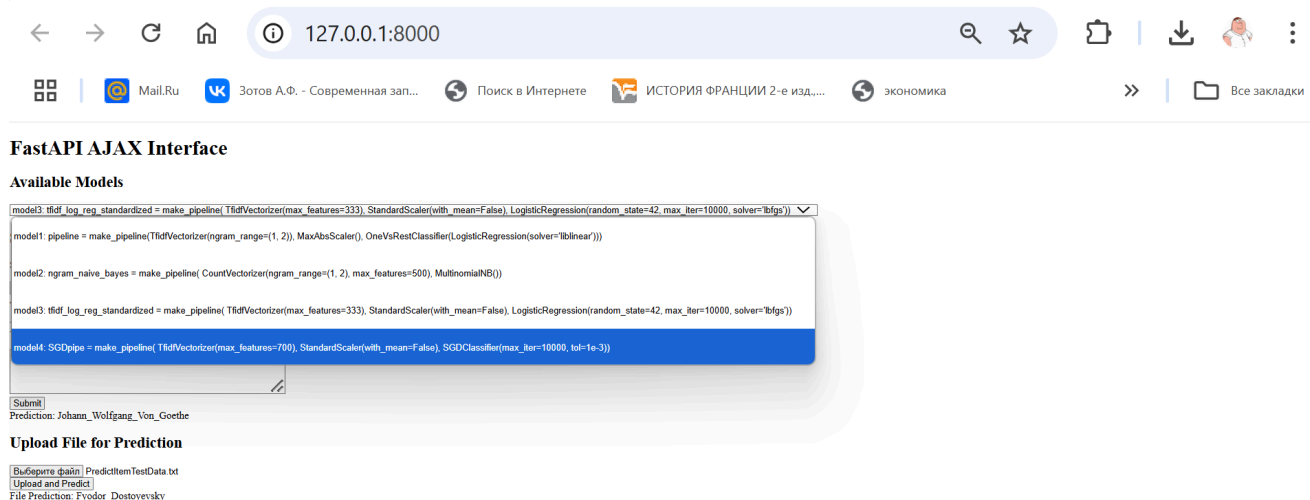
Далее можно пользоваться приложением.

Основные endpoint'ы FastAPI

/ (root)

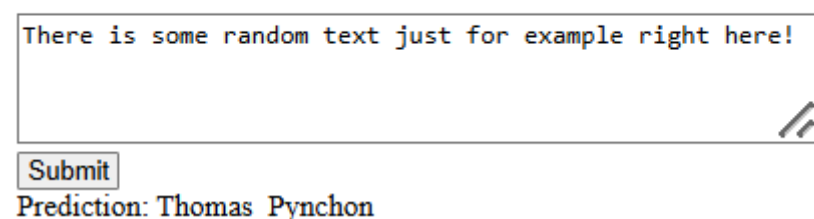
В корневом маршруте реализован базовый **FastAPI AJAX Interface** без перезагрузки страницы, который позволяет выбрать модель из обученных, быстро сделать предсказание автора по вручную написанному тексту или по файлу. Вот пример:

Мы можем выбрать модель из выпадающего списка:



Затем можно ввести какой-то текст и получить предсказание:

Predict Author



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

Upload File for Prediction

Выберите файл PredictItemTestData.txt
Upload and Predict
File Prediction: Fyodor_Dostoyevsky

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:

The screenshot shows a REST client interface with the following sections:

- Parameters:** A tab labeled "Parameters" with a "Cancel" button. Below it, it says "No parameters".
- Execute:** A blue button labeled "Execute" and a text input field labeled "Clear".
- Responses:** A section containing:
 - Curl:** A code block with the command:

```
curl -X 'GET' \
'http://127.0.0.1:8000/ModelsList' \
-H 'accept: application/json'
```
 - Request URL:** A text field containing `http://127.0.0.1:8000/ModelsList`.
 - Server response:** A section with a "Code" tab showing "200" and a "Details" tab. The "Response body" is displayed as a JSON array of four model objects, each with a "name" and a "description" containing a pipeline definition. A "Download" button is next to the JSON.
 - Response headers:** A code block showing headers:

```
content-length: 788
content-type: application/json
date: Thu, 02 Jan 2025 15:38:04 GMT
server: uvicorn
```

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

/SetModels

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

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

The screenshot shows a REST client interface for a POST request to `/setModel`. The 'Parameters' section has a dropdown for 'id' with 'model2' selected. The 'Request body' is an empty JSON object `{}`. The 'Responses' section shows a 200 status code and a JSON response body: `{ "message": "Active model set to 'model2'" }`. The response headers include `access-control-allow-credentials: true`, `access-control-allow-origin: *`, `content-length: 42`, `content-type: application/json`, `date: Thu, 02 Jan 2025 15:43:31 GMT`, and `server: uvicorn`.

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:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** /PredictItem
- Parameters:** No parameters
- Request body:**

```
{  
  "text": "ALEXEY Fyodorovitch Karamazov was the third son of Fyodor Pavlovitch Karamazov."  
}
```
- Execute:** A blue button to execute the request.
- Responses:** A section showing the response details.
 - Curl:**

```
curl -X 'POST' \\\n  'http://127.0.0.1:8000/PredictItem' \\\n  -H 'accept: application/json' \\\n  -H 'Content-Type: application/json' \\\n  -d '{  
    "text": "ALEXEY Fyodorovitch Karamazov was the third son of Fyodor Pavlovitch Karamazov."  
  }'
```
 - Request URL:** http://127.0.0.1:8000/PredictItem
 - Server response:**
 - Code:** 200
 - Response body:**

```
{  
  "id": "models",  
  "author": "Fyodor_Dostoyevsky"  
}
```
 - Response headers:**

```
access-control-allow-credentials: true  
access-control-allow-origin: *  
content-length: 45  
content-type: application/json  
date: Thu, 02 Jan 2025 16:07:13 GMT  
server: uicorn
```

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`:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** /PredictItemFile
- Parameters:** No parameters
- Request body:** multipart/form-data
- request:** PredictItemTestData.txt (file upload)
- Execute:** Button to send the request
- Response:** 200 OK
- Response body:** {"id": "model1", "author": "Fyodor_Dostoyevsky"}
- Response headers:** access-control-allow-credentials: true, access-control-allow-origin: *, content-length: 45, content-type: application/json, date: Thu, 02 Jan 2025 16:11:08 GMT, server: uvicorn

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:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** /PredictItemProba
- Parameters:** No parameters
- Request body:**

```
{  
  "text": "ALEXEY Fyodorovich Karamazov was the third son of Fyodor Pavlovitch Karamazov, a landowner well known in our district"  
}
```
- Response:** 200 OK
- Response body:**

```
{  
  "Fyodor_Dostoyevsky": 0.019734043277381948,  
  "Albert_Camus": 0.010001270645128825,  
  "Alexandre_Dumas": 0.010409776943636814,  
  "Boris_Pasternak": 0.01040807106658938,  
  "Aristophanes": 0.010407921543543121,  
  "Jean_Bacine": 0.01041012098067279,  
  "T_S_Eliot": 0.010268781415754684,  
  "Moliere": 0.01015433321817918,  
  "Haruki_Murakami": 0.01010630000153136,  
  "Huguido_MahFour": 0.010006797536181209,  
  "J_M_Conrad": 0.010093389554655787,  
  "William_Somerset_Maugham": 0.01008277133371805,  
  "Alphonse_Daudet": 0.0100718774800088,  
  "Alexander_Pushkin": 0.010060683546156793,  
  "O_Henry": 0.01005346084202551,  
  "Edgar_Allan_Poe": 0.010036323181210916,  
  "Pablo_Neruda": 0.010035705479151805,  
  "Mayne_Ride": 0.009986282919409828,  
  "Oscar_Wilde": 0.009980218826974885,  
  "William_Shakespeare": 0.00997938838458813
```

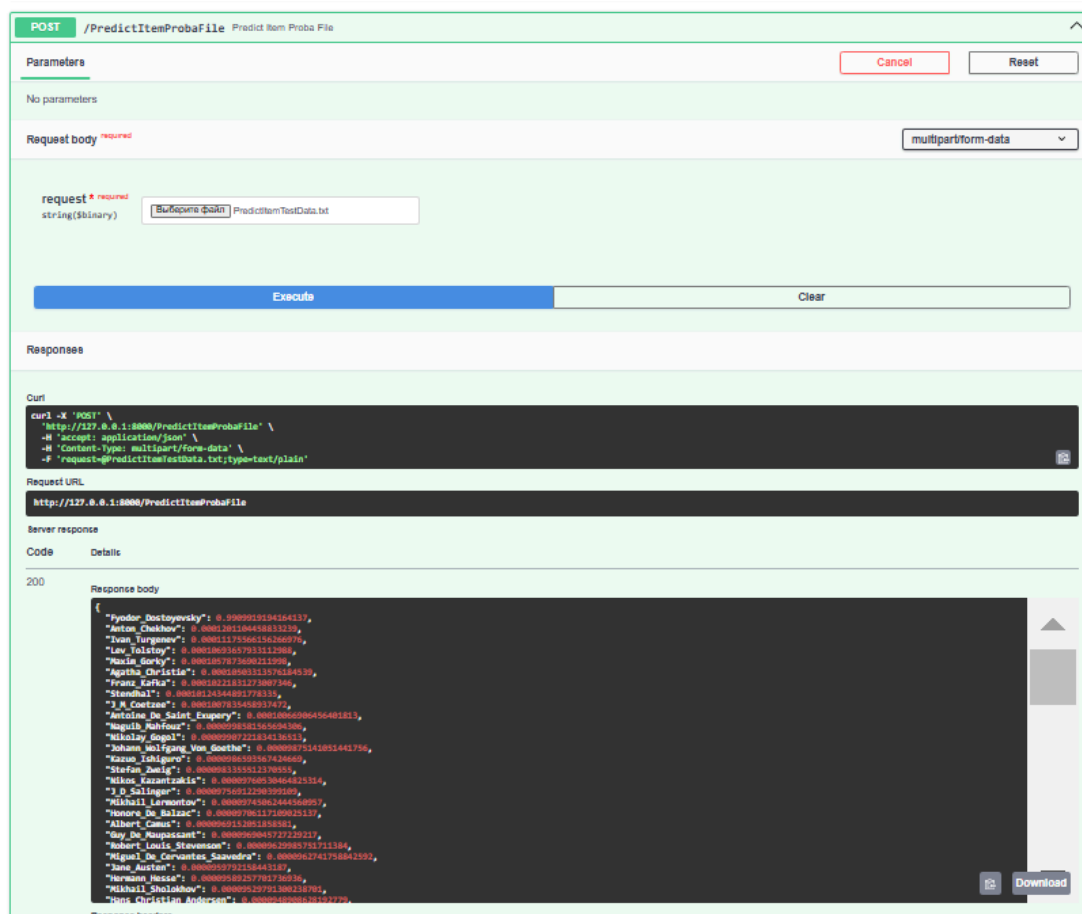
На выходе получается JSON-response с вероятностями для каждого автора из обучающей выборки, который имеет такую структуру:

```
{  
    "<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 соответствующим файлом.

Пример отправки отрывка текста 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'a нужно передать JSON следующей структуры:

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

Вот пример отправки подобного запроса:

The screenshot shows a REST client interface with the following sections:

- POST /PredictItems** (Method and URL)
- Parameters**: No parameters.
- Request body**: `application/json (selected). The body contains a JSON object:

```
{
  "texts": {
    "text1": "Some text here",
    "text2": "There is also a piece of random text",
    "text3": "Hello, my dear friend. How are you?"
  }
}
````
- Execute** button.
- Responses**:
 - Curl**:

```
curl -X 'POST' \
  'http://127.0.0.1:8080/PredictItems' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "texts": {
      "text1": "Some text here",
      "text2": "There is also a piece of random text",
      "text3": "Hello, my dear friend. How are you?"
    }
  }'
```
 - Request URL**: `http://127.0.0.1:8080/PredictItems`
 - Server response**:
 - Code**: 200
 - Response body**:

```
{
  "response": {
    "text1": "Mikhail Lermontov",
    "text2": "Haruki Murakami",
    "text3": "J.D. Salinger"
  }
}
```

В результате получается response в таком формате:

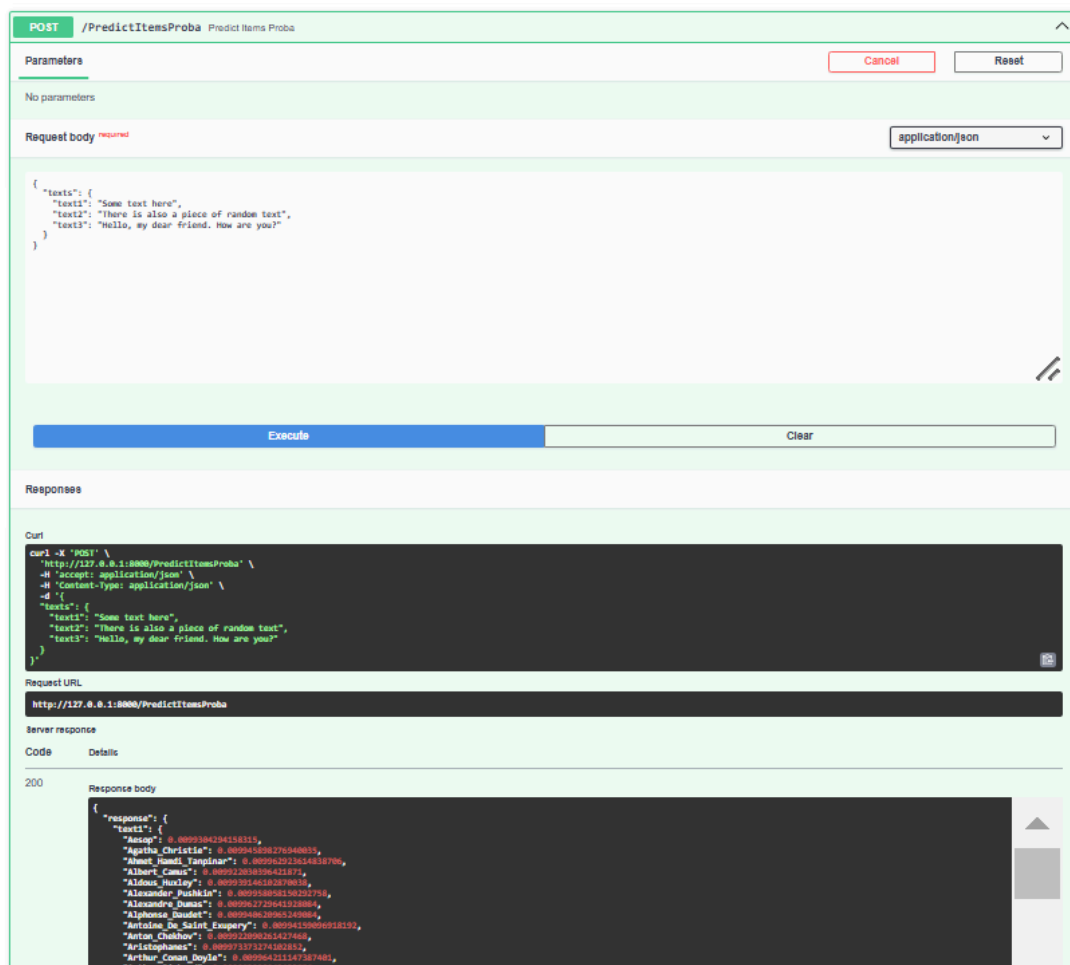
```
{
  "response": {
    "<id текста1>": "<автор текста1>",
    "<id текста2>": "<автор текста2>",
    "<id текста3>": "<автор текста3>",
    ...
  }
}
```

/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'a 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 переменная);
- **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:

POST /train_model Train Model

Обучает активную модель, используя предоставленные обучающие и тестовые датасеты и заданные гиперпараметры. Параметры: request: JSON строка, содержащая гиперпараметры для обучения. train_file: Обучающий датасет в формате parquet. test_file: Тестовый датасет в формате parquet. Возвращает: Словарь с ID модели, временем выполнения, метриками обучения и данными кривой обучения.

Parameters Cancel Reset

No parameters

Request body required multipart/form-data

request string Dict of hyperparameters as JSON string

☐ Send empty value

train_file required Training dataset in pq format
string(\$binary)

test_file required Testing dataset in pq format
string(\$binary)

Execute Clear

Responses

Curl

```
curl -X "POST" \
  "http://127.0.0.1:8080/train_model" \
  -H "accept: application/json" \
  -H "Content-Type: multipart/form-data" \
  -F "request={\"hyperparameters\":{}}" \
  -F "train_file=@train.pq" \
  -F "test_file=@test.pq"
```

Request URL

http://127.0.0.1:8080/train_model

Server response

Code Details

200

Response body

```
{
  "model_id": "model5",
  "execution_time": "31.58 seconds",
  "accuracy": "0.893",
  "precision": "0.8125",
  "recall": "0.879",
  "f1": "0.8333333333333334",
  "train_sizes": [
    3,
    11,
    19,
    27,
    36
  ]
}
```

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 переобучена, 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:

The screenshot shows a REST client interface for a POST request to `/partial_fit`. The request body is multipart-form-data. The parameters are:

- `id` (required, string): `model4`
- `request_file` (required, string(binary)): `Булевыми файн df_train.pq`

The response is a 200 status code with a JSON body:

```
{
  "message": "Model with id 'model4' successfully updated with new data."
}
```

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:

POST /PredictItem Predict Item

Эндпоинт для предсказания одного элемента текста. Параметры: request (PredictItemRequest): Запрос, содержащий текст для предсказания. Возвращает: PredictItemResponse: Ответ с идентификатором модели и предсказанным автором.

Parameters Cancel

No parameters

Request body required application/json

```
{
  "text": "string"
}
```

Execute Clear

Responses

Curl

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

Request URL

http://127.0.0.1:8000/PredictItem

Server response

Code	Details
200	<p>Response body</p> <pre>{ "model_id": "models", "author": "Agatha_Christie" }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-origin: * content-length: 46 content-type: application/json</pre>

/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:

POST /fine_tuning 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.

Parameters Cancel Reset

No parameters

Request body required multipart/form-data

request_file required string(binary) Выберите файл

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/fine_tuning' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'request_file=train.pg'
```

Request URL

http://127.0.0.1:8000/fine_tuning

Server response

Code Details

200

Response body

```
{
  "message": "All models successfully updated with new data."
}
```

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
content-length: 60
content-type: application/json
date: Fri, 03 Jun 2023 14:13:27 GMT
server: uvicorn
```

Responses

Code	Description	Links
------	-------------	-------

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.