Описание проекта

Целью проекта является разработка и реализация сервиса для предсказания погоды в нескольких городах при помощи ML и TS на основе исторических данных о погоде, полученных через API. На данный момент реализовано только предсказание температуры для Москвы.

Архитектура системы

Система состоит из двух основных компонентов: серверной части, реализованной с помощью FastAPI, и клиентского приложения, реализованного с помощью Streamlit.

Серверная часть (FastAPI)

Серверная часть отвечает за обработку запросов, загрузку предобученных моделей, обучение новых моделей и предсказание погоды. Она включает в себя следующие основные компоненты:

- Модели Pydantic для представления данных запросов и ответов
- Обработка запросов с использованием соответствующих HTTP-методов
- Логирование с механизмом ротации
- Загрузка предобученной модели при запуске сервера
- Обучение модели в отдельном процессе
- Предсказание погоды с использованием активной модели
- Создание новой модели с возможностью загрузки данных из CSV-файла

Клиентская часть (Streamlit)

Клиентская часть представляет собой веб-интерфейс, который позволяет пользователю взаимодействовать с сервером. Она включает в себя следующие основные компоненты:

- Загрузка данных из CSV-файла
- Анализ данных (EDA)
- Создание новой модели с возможностью выбора гиперпараметров
- Просмотр информации о загруженных моделях
- Инференс с использованием обученной модели
- Логирование с механизмом ротации

Реализация

Серверная часть (FastAPI)

Модели Pydantic

В проекте используются следующие Pydantic-модели для представления данных запросов и ответов:

- ModelDesc: Описание модели
- SetActiveModelRequest: Запрос на установку активной модели
- SetActiveModelResponse: Ответ на установку активной модели
- FitModelResponse: Ответ после запуска обучения модели
- PredictRequest: Запрос на предсказание
- PredictResponse: Ответ с предсказаниями
- LoadNewModelResponse: Ответ после загрузки новой модели
- CSVContent: Представление содержимого CSV-файла
- LoadNewModelRequest: Запрос на загрузку новой модели

Обработка запросов

Серверная часть обрабатывает следующие запросы:

- GET /models: Получение списка загруженных моделей
- POST /set_model: Установка активной модели
- POST /fit: Запуск обучения активной модели
- POST /predict: Получение предсказаний от активной модели
- POST /load_new_model: Загрузка новой модели
- POST /upload_csv: Загрузка CSV-файла на сервер

Логирование

В проекте реализовано полноценное логирование с механизмом ротации. Логи сохраняются в папку logs/.

Загрузка предобученной модели

При запуске сервера происходит загрузка предобученной модели из файла.

Обучение модели

Обучение модели происходит в отдельном процессе. Пользователь может отправить запрос на обучение модели, передав необходимые гиперпараметры. Если обучение занимает более 10 секунд, процесс будет прерван, и пользователю будет возвращен соответствующий ответ.

Предсказание

Пользователь может отправить запрос на предсказание погоды, передав начальное время для прогноза. Сервер вернет предсказания, сделанные активной моделью.

Загрузка новой модели

Пользователь может загрузить новую модель, передав необходимые параметры, такие как путь к CSV-файлу, имя модели и количество эпох для обучения. Если указан путь к CSV-файлу, он будет загружен в базу данных, и на его основе будет создана новая модель.

Streamlit приложение (далее — приложение)

Приложение имеет 3 страницы:

1. Стартовая страница (вкладка Start)
2. Страница с EDA-частью (вкладка EDA)
3. Страница с ML-частью (вкладка ML)

Для переключения между страницами пользователь должен кликнуть на соответствующую вкладку слева.

Описание содержания и функционала страниц

1. **Вкладка Start**
    1. Краткая навигация по приложению.
2. **Вкладка EDA**
    1. Содержит графики временных рядов для следующих погодных данных: температура, влажность, давление, скорость ветра за последние несколько дней (на основе доступных данных), а также их прогноз на следующие сутки и некоторую описательную статистику.
3. **Вкладка ML**
    1. **Раздел «Загрузка данных»**
        1. Пользователь загружает последние данные о погоде в формате csv-файла для обучения моделей.
            1. Если пользователь не загрузил данные, выводится информационная панель «Пожалуйста, загрузите не пустой CSV-файл.»
        2. После загрузки файла пользователю открывается окно данных со скроллингом для их предпросмотра.
        3. Далее пользователь нажимает кнопку «Сохранить загруженные данные».
            1. Файл с последними данными достаточно загрузить 1 раз.
            2. Если пользователь не нажал кнопку, приложение будет обучать новые модели на дефолтных данных (до 20.09.2024).
    2. **Раздел «Создание нового класса модели»**
        1. Пользователь вручную вводит id и гиперпараметры новой модели, которую он далее хочет обучить, и нажимает кнопку «Создать новый класс для модели».
        2. Если пользователь не ввел id или не нажал кнопку, приложение будет использовать модель по умолчанию в следующих пунктах.
        3. Если гиперпараметры не введены, то используются гиперпараметры по умолчанию.

        1. На данном этапе работы в бэкэнде фактически реализован только учет параметра epochs.
3. **Раздел «Обучение модели»**
    1. В разделе описаны параметры и кривая обучения предобученной LSTM-модели.
4. **Раздел «Загруженные модели»**
    1. При нажатии на кнопку «Показать все загруженные модели» приложение выводит список всех загруженных и обученных ML-моделей.
5. **Раздел «Установка активной модели»**
    1. Пользователь вручную вводит id модели, которая далее будет обучаться и по которой будет строиться прогноз, и нажимает кнопку «Установить активную модель».
    2. Если пользователь не ввел id или не нажал кнопку, приложение будет использовать модель по умолчанию в следующих пунктах.
6. **Раздел «Обучение активной модели»**
    1. Пользователь нажимает кнопку «Обучить активную модель».
    2. При успешном обучении модели приложение выводит сообщение «Модель {id} обучена»
7. **Раздел «Прогноз активной модели»**
    1. Пользователь вручную вводит дату и время, на которое строится прогноз, и горизонт прогнозирования.
        1. Если дата и время не введены, приложение выводит ошибку.
    2. Если горизонт не выбран, то используются горизонт по умолчанию.
    3. Пользователь нажимает кнопку «Показать прогноз температуры».
    4. При успешном получении прогноза, приложение выводит на экран таблицу с почасовым предсказанием и график температуры за последние несколько дней и прогноз.

Работа с базой postgresql

База postgresql была поднята в yandex-облаке. В этой базе мы храним данные с 1970 года для обучения моделей и EDA.

Для работы с базой был написан класс в файле postgres.py. Класс является оберткой над библиотекой `psycopg2` и предоставляет удобные для использования в нашем проекте функционалы:

- создания и удаления таблиц с нужной нам схемой,
- получения данных из таблиц,
- загрузки данных в таблицы батчами для быстрой обработки csv файлов,
- загрузки данных в таблицы по 1 записи для данных, получаемых в реальном времени.

Данные из базы используются при обучении модели и предсказании погоды на следующий период.

Сбор данных в режиме реального времени

В файле realtime_parser.py реализованы классы для получения данных из API сервиса с погодой и загрузки их в базу для постоянного пополнения датасета.

Этот процесс запускается из файла `main.py` в отдельном потоке, чтобы работать параллельно с fastapi.

Работа с github actions

Для удобства работы нами были настроены github actions: линтеры, сборка и пуш образов в dockerhub, их деплой на сервер.

Линтеры

Настроены по 3 линтера на `.py` файлы в frontend и backend: pylint, flake8, pycodestyle. Также настроен линтер hadolint на оба Dockerfile.

Все линтеры можно найти в linter.yml. Запускаются при каждом коммите.

Сборка и пуш докер образа в dockerhub

Для этого был заведен репозиторий в dockerhub. Пароль от токена для логина в dockerhub мы сохранили в secrets github репозитория и используем его в workflow следующим образом: `password: ${{secrets.DOCKER_PASSWORD}}`

Далее выполняются этапы сборки и пуша образов (сборка и пуш образов frontend и backend происходят отдельно друг от друга). Теги образов формируются отдельным шагом, у всех тегов есть префикс `frontend-/backend-`. А также пушатся теги `backend-latest/frontend-latest`.

Они описаны в `[build-and-publish.yml](../.github/workflows/build-and-publish.yml)`. Сборка и пуш образов запускаются при любом коммите и любом теге.

Деплой на удаленный сервер

Арендовали сервер в yandex-cloud. Создали отдельный новый ssh-ключ, создали пользователя на сервере, добавили ему публичную часть нового ключа. Приватную часть ssh-ключа положили в secret репозитория. Из github actions загружаем на сервер новую версию docker-compose.yml с помощью `scp`. Затем заходим на сервер под ssh из github actions, скачиваем новые образы и запускаем docker-compose. На сервере создали `.env` файл и записали туда пароль от базы и API-ключ для получения данных в реальном времени.

Деплой описан в последней job в файле `[build-and-publish.yml](../.github/workflows/build-and-publish.yml)`. Этот этап запускается только при пуше тегов в репозитории и зависит от успешной сборки образов.

Увидеть его работу можно тут: http://84.201.145.245:18501.

Модель прогноза погоды

Выбрана модель LSTM.

Причина выбора данной модели

Анализ температуры показал достаточно хаотичное поведение температуры во времени. День на день не приходится, однако ближайшие часы наивный прогноз показывал достаточно хорошо, если температура резко не менялась. Экспоненциальные средние не успевают поймать закономерности смены погоды, а увеличивание окна обработки данных вела к длительным ожиданиям значений

прогноза. Как показало исследование модель LSTM значительно легче в обслуживании и требует мало места в директории, в отличие от предобученных и записанных в отдельный файл моделей аримы.

Алгоритм работы модели

На вход подается ретроспектива температуры по часам. Данные обогащены математически выделенными 3 признаками: 1 - световой день переведенный в число, которые меняется с каждым временным интервалом. (10мин) 2,3 - это дополнительные признаки основанные на первом. Для обогащения модели известными признаками в будущем. Все дополнительные признаки не зависят от сторонних источников. Поэтому легко посчитать для любой точки земного шара. Для каждой локации необходимо делать собственный расчет, как и обучать саму модель.

Предобработка

- Интерполяция данных с часовых значений до 10минут. Это необходимо для более качественного прогноза краткосрочного горизонта прогнозирования.
- Стандартизация числовых данных по формуле: (признак - среднее арифметическое по всему признаку) / стандартное отклонение по данному признаку

Обучение

Ядро модели написано на TensorFlow. Для ускоренной возможности обучения модели локально, либо в коллаб, данные с помощью tf.data обрабатываются с выполнением перемешивания, пакетирования и кэширования.

Модель опирается на последние 5 дней. Это 720 значений = (5 дней) * (24 часа в сутках) * (6 интервалов по 10 минут). Для быстрого обучения используются 200 шагов. Время ожидания на CPU = 10 минут. В коллабе на TCP в 2 раза быстрее, ограничения имеются только на стороне коллаба. Добавляются данные в конец ретроспективы по известным новым признакам, а целевая переменная заменяется наивным прогнозом прошлого дня.

Прогноз

На основе данных LSTM прогнозирует на следующие 72 значения = (6 интервалов по 10 минут) * 12 часов с даты последнего известного значения температуры.

Вспомогательный источник

- Глубокое обучение на Python [Франсуа Шолле]
- https://coollib.cc/b/651746-fransua-sholle-glubokoe-obuchenie-na-python - скачивал тут
- https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/chapter10_dl-for-timeseries.ipynb код из книги