Рекомендательная система для пользователей московских библиотек. Решение команды Pegasus.

Данный репозиторий хранит код и документацию решения задачи `Рекомендательная система для пользователей московских библиотек` команды `Pegasus`.

API решения доступен по адресу http://178.154.240.169:5000/get_recommendations/ (отвечает в формате `JSON`)

Формат запроса к API, на примере ID пользователя 1:
http://178.154.240.169:5000/get_recommendations/1

Веб-интерфейс решения доступен по ссылке http://178.154.240.169

Общий подход решения:

1. Раз в сутки запускается процесс расчёта рекомендаций, которые загружаются в базу данных.
2. В онлайн режиме база данных опрашивается через API или веб-интерфейс: по ID пользователя можно получить рекомендации этого пользователя, а также его историю (последние 20 полученных книг).

Мы выдаём рекомендации для всех пользователей из базы пользователей в файле `readers.csv`, а не только для тех 100 что были в файле `dataset_knigi.xlsx`.

Детальное описание:

Бекенд

Общее описание бекенда

Бекенд реализован на `Python`. Разработка и тестирование велись на `Python 3.8` в операционной системе `Ununtu 20.04`. Все необходимые зависимости для `Python` прописаны в файле

`requirements.txt` в корне проекта (устанавливаются через `pip`).

Бекенд реализован как приложение фреймворка `FastAPI`.

В качестве HTTP сервера выступает `Gunicorn`, запускающий приложение бекенда на асинхронных воркерах `uvicorn`.

В качестве базы данных используется RDBMS `MySQL`.

Запуск бекенда

Запуск бекенда через консоль на нашем сервере (сейчас он запущен в сессии `tmux`):

`cd /home/mos_lib_hack/mos_lib_hack`

`source .venv/bin/activate` (в данном окружении установлены пакеты из файла `requirements.txt`)

```
gunicorn backend_server:app --bind=0.0.0.0:5000 -w 4 -k
uvicorn.workers.UvicornWorker --timeout=3600
```

Использование API

Адрес API (на примере ID пользователя 1): http://178.154.240.169:5000/get_recommendations/1 (GET-запрос по URL).

Формат ответа - `JSON` по спецификации ТЗ хакатона.

Документация API в формате OPENAPI/Swagger: http://178.154.240.169:5000/docs . Здесь же можно запустить пробные запросы к API.

Модули бекенда

Код бекенд сервера находится в корне репозитория в файле `backend-server.py`.

API хендлеры (работающий сейчас, а также тестовый mock) - в `api/recommendations_api.py`.

Дополнительные сервисы и утилиты находятся в директории `services/`.

За загрузку данных в базу отвечает модуль `ml_pipelines.data_db_loader`.

Работа с базой данных

База данных (БД) для целей хакатона размещена на том же сервере, что и бекенд, и фронтенд.

Конфигурирование БД

Необходимая конфигурация БД для загрузки больших файлов с рекомендациями и историей пользователя прописывается в файле:

```
interactive_timeout = 600000
wait_timeout = 600000
```

```
mysqlx_wait_timeout = 600000
mysqlx_interactive_timeout = 600000
max_allowed_packet=964M
innodb_lock_wait_timeout = 6000
innodb_rollback_on_timeout=1
```

После изменения файла конфигурации нужно запустить команду `systemctl restart mysql.service`.

При клонировании репозитория на новый сервер возможны проблемы с авторизацией в БД через питоновский драйвер. В таком случае необходимо для бекенда создать нового юзера в БД и работать в коде через него (предпочтительный вариант), либо изменить плагин юзера root на mysql_native_password.

Работа с данными в БД

Данные в базу загружаются пайплайном загрузки, размещённом в модуле `ml_pipelines.data_db_loader`.

Исходные файлы для загрузки в БД рекомендаций и истории пользователя - `data/recommendations.csv` и `data/history.csv`.

Эти файлы появляются в своих директориях в результате работы модели машинного обучения.

Файлы загружаются в базу `mos_lib_hack`, в таблицы `recommendations` и `history`. Именно к этим таблицам обращается бекенд при запросе через API или веб-интерфейс.

Фронтенд

Приложение создано для показа рекомендаций и истории пользователей библиотек.

Интерфейс

Интерфейс содержит одну страницу, на которой можно вбить id пользователя библиотеки и получить список рекомендуемых книг и историю.

Логика работы

1. Приложение получает id пользователя библиотеки либо из url-а при открытии страницы `host/users/<user_id>`, либо из формы на странице.
2. Делается запрос к бэкенду для получения рекомендаций и истории по id.
3. Полученные данные отображаются на странице.

Технологии

- Для запуска фронтенда используется Docker. Docker позволяет быстро развертывать и масштабировать приложения в любой среде и сохранять уверенность в том, что код будет работать.

- Приложение написано на React. React - JavaScript-библиотека для создания пользовательских интерфейсов. React предоставляет высокую скорость, простоту и масштабируемость.
- В качестве веб-сервера используется Nginx. Nginx позволяет обрабатывать сотни тысяч единовременных подключений на одном физическом сервере.

Запуск приложения

Для запуска приложения нужно выполнить следующие две команды:

1. `docker build -t client-app .`
2. `docker run -p 80:80 client-app`

Структура кода

Основные файлы

- `Dockerfile` - докерфайл для запуска приложения;
- `src/App.js` и `src/index.js` - код с реализацией логики React приложения;
- `nginx/nginx.conf` и `nginx/site.conf` - файлы с конфигурацией nginx.

Рекомендательный движок

ALS (основной)

- Модель из класса алгоритмов коллаборативной фильтрации.
- Использует неявные взаимодействия (взятие книги из библиотеки) в качестве целевой перемнной.
- Основная идея – выучить векторы пользователей и книг так, чтобы вектор юзера отражал его интересы в пространстве книг, а вектор книги отражали портрет юзеров, которые их берут. Похожие книги будут иметь близкие вектора в смысле косинусного расстояния. Такое же будет наблюдаться и для векторов похожих пользователей.
- После нахождения латентных представлений (векторов или эмбеддингов) юзеров и айтемов, каждому пользователю можно найти ближайшие *topN* книг по скалярному произведению вектора пользователя и векторов книг.
- Чтобы обучить модель и подготовить рекомендации, нужно запустить команду `python __init__.py` в директории `/ml_pipelines/als/`.
- В файл *recommendations.csv* сохранятся рекомендации для каждого из пользователей *dataset_knigi.csv* и для всех из файлов *circulaton_.csv*.
- В файл *recommendations.csv* также сохранятся рекомендации для "холодного" пользователя (по которому нет истории взятий книг). Он имеет id = 0.
- В файл *history.csv* сохранятся история (ограниченная до последних 20 взятых уникальных книг) для каждого из пользователей *dataset_knigi.csv* и для всех из файлов *circulaton_.csv*.
- По умолчанию, все необходимые данные должны лежать в папке **/data** относительно корня модуля.

Item2vec (дополнительный вариант)

- Модель, базированная на методе **skip-gram**, учится для слова (в данном случае книги) предсказывать его контекст. Основной класс – **Word2VecRecommender**.

- Основная идея – выучить векторы книг аналогично тому, как **word2vec** выучивает векторы слов имея множество примеров каждого слова в разных контекстах (под контекстом подразумеваются слова, стоящие рядом в предложении). В данном случае мы выучиваем векторы книг в зависимости от того, с какими книгами они берутся вместе.
- Далее считаем, что среднее векторов книг по истории пользователя – вектор пользователя (=средняя книга среди взятых), к которому можно найти ближайшие *topN* книг по косинусной близости.
- Чтобы обучить модель и подготовить рекомендации, нужно запустить команду `python item2vec.py` в директории `/ml_pipelines/item2vec/`.
- В файл *recommendations_item2vec.csv* сохранятся рекомендации для каждого из пользователей *dataset_knigi.csv*.
- По умолчанию, все необходимые данные должны лежать в папке **/data** относительно корня модуля.