Проект "MLOps на примере задачи матчинга геолокаций (на основе соревнования Kaggle Foursquare - Location Matching)"

Автор проекта:
Клейн Илья
wisoffe@gmail.com

Репозиторий проекта: https://gitlab.com/mlops-23reg-team/mlops-23reg-project

Краткое описание ML проекта:

Цель проекта - освоить подходы MLOps на основе практической ML-задачи.

В ходе реализации не ставилась задача построения эффективной ML части, проект развивался как поэтапное построение ее обертки в виде MLOps.

Дополнительным артефактом, который будет загружен в репозиторий проекта, будет являться своего рода “дневник прохождения курса и реализации проекта”, в котором пошагово расписаны все этапы работы над ДЗ и финальные доработки.

Задачи и легенда проекта

Разработчики проекта являются сотрудниками компании, занимающейся поставкой данных по глобальным POI (point of interest). Необходимо создать внутренний сервис по нахождению дублей POI, со следующими задачами/ограничениями:

- сервис должен функционировать в виде api, пользовательский интерфейс не требуется;
- сервис должен принимать на вход .csv файл заданного формата, содержащий в себе информацию о собранных POI;
- сервис должен возвращать в качестве ответа .csv файл, заданного формата, с данными о найденных дубликатах POI;
- сервис будет функционировать исключительно внутри организации, в доверенной инфраструктуры, предпринимать меры безопасности, требуемые для возможности публикации сервиса в Интернет, не требуется.

Описание исходных данных

train.csv - представляет собой обучающий набор, состоящий из одиннадцати атрибутивных полей для более чем миллиона записей о POI, в том числе:

- ***id*** - уникальный идентификатор для каждой записи;
- ***point_of_interest*** - уникальные идентификаторы POI, сформированные предварительной разметкой (преимущественно экспертной); т.е. все строки, с одним и тем же значением *point_of_interest*, являются дублями; по сути, колонка является источником нашей целевой переменной, на основе которой мы сможем обучить модель; в реальных данных, с которыми в дальнейшем будет работать сервис, эта колонка отсутствует;
- ***latitude, longitude*** - географические координаты;
- **name, address, city, state, zip, country, url, phone, categories** - данные о названии, адресе, городе, регионе, индексе, стране, url-адресе, телефонном номере и категории соответственно.

sample_submission.csv - файл-пример требуемого выходного формата, количество строк, должно соответствовать количеству строк поданного на вход датасета, содержит в себе следующие колонки

- ***id*** - исходные id
- **matches** - список id найденных дублей, через пробел (список в том числе должен включать в себя сам исходный id, если дублей не найдено, то в ячейке будет значиться только исходный id)

Результаты первичного анализа и обработки данных

Первичный EDA-анализ показал, что в исходном датасете существуют колонки с множеством пропущенных значений, все составляющие ключа POI - категориальные, кроме географических координат долготы и широты. Больше всего пропущенных значений в колонках url, phone, и zip. В колонках имени, долготы и широты нет пропущенных значений, но могут быть неверные данные, кроме того, для одних и тех же POI, но полученных из разных источников мы можем наблюдать существенные различия по всем колонкам.

Описание бейзлайна

Сводим задачу к бинарной классификации, путем отбора кандидатов (строк) для последующего их сравнения на предмет идентичности. Сравнить все пары строк между собой не представляется возможным, т.к. мы для нашего случая получили бы парный датасет из более чем 10^12 строк.

В бейзлайне, отбор кандидатов производим, путем округления координат *latitude, longitude* до заданного знака после запятой (соответствующего например расстоянию 100м, 1 км, и т.п.), с последующим объединением строк в пары, через merge на основе одинаковых значений колонки = конкатенации округленных latitude, longitude.

Далее, для каждой строки полученного парного датасета, формируем следующие фичи:

- географическое расстояние в км. между исходными точками;
- расстояние Левенштейна между строками-названиями исходных точек.

Колонку целевой переменной формируем путем сравнения point_of_interest_1, point_of_interest_2, если они идентичны, целевая переменная выставляется в 1, иначе в 0.
Обучаем XGBClassifier на полученных данных.

На основе полученных предсказаний, формируется выходной .csv файл формата sample_submission.csv.

Метрики качества

В качестве основной метрики используется Jaccard score, который равен среднему значению из Jaccard index всех строк. Формула расчета Jaccard index приведена с статье на википедии https://en.wikipedia.org/wiki/Jaccard_index

В процессе обучения, Jaccard score рассчитывается на двух стадиях:

- на стадии формирования датасета пар кандидатов, это своего рода максимальный Jaccard score, который мы сможем получить, решая задачу бинарной классификации на текущем парном датасете;
- на финальной стадии, после получения предсказаний модели, это реальный, итоговый Jaccard score

Описание MLOPS подходов

Система контроля версий

Инструменты контроля codestyle

Шаблон проекта / шаблонизаторы / структура проекта

Workflow менеджеры

Инструменты трекинга экспериментов

Методы и инструменты тестирования

Описание CI пайплайна

Система контроля версий

В качестве системы контроля версий, выбран git. Причина выбора - наиболее распространенная система, отвечающий всем заявленным к проекту требованиям.

В качестве хранилища удаленного репозитория, выбран GitLab. Причины выбора - необходимость получения практического опыта работы с GitLab (до курса, имел только незначительный опыт работы с GitHab).

Инструменты контроля codestyle

- форматеры - Black (максимальная длина строки 100 символов);

  кроме black были опробованы форматеры autopep8 и yapf (от Google), в итоге выбран black, как наиболее безкомпромисный, и действительно способный привести код разных разработчиков в единообразный формат;

- линтеры:
  - pylint - наиболее строгий и требовательный линтер (осуществляет как логические, так и стилистические проверки);
  - flake8 - очень распространенный и гибко настраиваемый линтер;
  - mypy - осуществляет логические проверки, касающиеся корректности работы с типами данных (основной источник информации это аннотации типов);
  - bandit - выполняет анализ кода на предмет выявления небезопасного кода.

Инструменты контроля codestyle

В качестве точек контроля выбраны:

- IDE (в моем случае VSCode)
- в пайплайне gitlab-ci.yml в задаче (job) test_lint стейджа tests, следующим образом

```
- poetry run flake8 src
- poetry run mypy --ignore-missing-imports src
- poetry run bandit src
- poetry run pylint src
```

Кроме того, был рассмотрен вариант добавления дополнительно точки проверки линтерами через git pre commit hooks, но посчитал этот вариант избыточным, т.к. у нас основная цель защитить основные ветки удаленного репозитория (в нашем случае main) от "плохого" кода, но иногда требуется сделать быстрый локальный коммит без отработки всех замечаний линтеров, а в случае pre commit hooks этого сделать не удастся.

Шаблон проекта / шаблонизаторы / структура проекта

В качестве шаблонизатора и шаблона был выбран Cookiecutter с шаблоном для Data Science от DrivenData (https://github.com/drivendata/cookiecutter-data-science)

В качестве менеджера зависимостей выбран poetry.

Исходный код проекта в соответствии с выбранным шаблоном располагается в подкаталогах директории src и оформлен в виде модуля. Для сборки модуля в пакет с помощью poetry, в файл pyproject.toml добавлены следующие настройки:

```
[tool.poetry]
name = "mlops23regproject"
version = "0.1.0"
description = "MLOps cource project by 23reg team."
authors = ["MLOps-23reg-team"]
license = "MIT"
packages = [
{ include = "src", from = "." }
]
```

Workflow менеджер и инструмент версионирования данных

- бэкэндом для версионирования данный является локально развернутый s3 сервис на базе minio, с выделенным бакетом dvc;
- учетные данные для доступа к s3 хранятся не в конфигурационном файле уровня проекта .dvc/config, а в локальном типе конфигурационного файла .dvc/config.local, который исключен из контроля git; для целей unit тестирования в ходе исполнения gitlab ci пайплайна, соответствующий файл генерируется "налету" в ходе исполнения самой задачи раннером, путем исполнения следующих команд:
  - poetry run dvc remote modify --local s3minio access_key_id $AWS_ACCESS_KEY_ID
  - poetry run dvc remote modify --local s3minio secret_access_key $AWS_SECRET_ACCESS_KEY
  - переменные $AWS_ACCESS_KEY_ID/$AWS_SECRET_ACCESS_KEY хранятся в разделе Variables гитлаба
- пайплайн DVC настроен таким образом, чтобы в s3 версионировались не все промежуточные файлы, а только те, для которых это действительно целесообразно

Инструменты трекинга экспериментов

Трекинг экспериментов и моделей осуществляется с помощью MLFlow. MLFlow развернут локально, в докер контейнере, по сценарию 4:

- в качестве СУБД (хранилище трекинга экспериментов) выступает локально развернутый контейнер с PostgreSQL;
- в качестве s3 хранилища сертификатов выступает локально развернутый minio (используется бакет mlflow).

Методы и инструменты тестирования

Unit-тестирование осуществляется следующими инструментами:

- pytest, с двумя скриптами в директории ./tests
  - test_evaluate.py - осуществляет имитацию вызова функции обернутой в декораторы click из CLI;
  - test_predict_model.py - осуществляет имитацию вызова функции обернутой в декораторы click из CLI, дополнительно, через библиотеку great_expectations, проверяет формат выходного файла, генерируемого по итогу запуска модуля (конкретно добавлены проверки на соответствие списку колонок, уникальность значений, отсутствие пропусков NaN);
- через запуск всего пайплайна командой dvc repro (тест будет пройден при отсутствии ошибок).

Указанные тесты выполняются в ходе исполнения задачи (job) pytest, стеджа tests пайплайна gitlab-ci.yml

Описание CI пайплайна

Пайплайна gitlab-ci.yml состоит из 3-х стейджей, 2 из которых относятся к CI, и 1 относится к CD.

CI стейджи:

- tests - выполняется при коммите/пуше в любую ветку (кроме main), в рамках него параллельно запускаются следующие задачи:
  - test_lint - аудит кода линтерами;
  - pytest - юнит-тесты;
- build - выполняется только для коммитов в ветку main, в нашем случае, т.к. прямые коммиты запрещены, будет запускаться сразу после успешного отработанного merge-request в main; в рамках него запускается единственная задача:
  - docker_build - сборка и доставка в gitlab registry докер образа нашего сервиса dev_ml_service.

Описание получившегося сервиса/продукта

Сервис сервис реализован в виде докер контейнера dev_ml_service, предоставляющий API, функционал которого описан далее.

API

API доступен по адресу http://<ip or hostname>:8004/invocations

Принимает на вход POST запрос входными данными в виде одного параметра file представляющего из себя .csv файл исходного формата, пример входного файла:

```
id,name,latitude,longitude,address,city,state,zip,country,url,phone,categories
E_00001118ad0191,Jamu Petani Bagan Serai,5.012169,100.535805,,,,,MY,,,Cafés
E_000020eb6fed40,Johnny's Bar,40.43420889065938,-80.56416034698486,497 N 12th St,Weirton,WV,26062,US,,,Bars
E_00002f98667edf,QIWI,47.215134,39.686088,"Межевая улица, 60",Ростов-на-Дону,,,RU,https://qiwi.com,+78003011131,ATMs
E_001b6bad66eb98,"Gelora Sriwijaya, Jaka Baring Sport City",-3.01467472168758,104.79437444575598,,,,,ID,,,Stadiums
E_0283d9f61e569d,Stadion Gelora Sriwijaya,-3.021726757527373,104.78862762451172,Jalan Gubernur Hasan
Bastari,Palembang,South Sumatra,11480.0,ID,,,Soccer Stadiums
E_00002f98667edf_copy,QIWI,47.215134,39.686088,"Межевая улица, 60",Ростов-на-
Дону,,,RU,https://qiwi.com,+78003011131,ATMs
E_001b6bad66eb98_copy,"Gelora Sriwijaya, Jaka Baring Sport City",-3.01467472168758,104.79437444575598,,,,,ID,,,Stadiums
```

API

По итогу работы, возвращает файл требуемого по условию задачи формата, пример выходного файла:

```
id,matches
E_00001118ad0191,E_00001118ad0191
E_000020eb6fed40,E_000020eb6fed40
E_00002f98667edf,E_00002f98667edf E_00002f98667edf_copy
E_001b6bad66eb98,E_001b6bad66eb98 E_001b6bad66eb98_copy
E_0283d9f61e569d,E_0283d9f61e569d
E_00002f98667edf_copy,E_00002f98667edf_copy E_00002f98667edf
E_001b6bad66eb98_copy,E_001b6bad66eb98_copy E_001b6bad66eb98
```

Описание CD пайплайна

CD часть пайплайна реализована в виде следующего стейджа:

- deploy - выполняется загрузка готового докер образа нашего сервиса с gitlab registry и его деплой (запуск соответствующего контейнера); в рамках стейджа запускается единственная задача docker_deploy:
  - выполняется только удачного merge request в ветку main и только после успешного завершения CI стейджа build;
  - предварительно останавливаются и удаляются предыдущие версии контейнеров с заданным префиксом (_dev).

Итоговый технологический стек

- Язык разработки: Python 3.9.
- Менеджмент зависимостей: poetry.
- Шаблон проекта: cookiecutter data science.
- Система контроля версий кода – git/gitlab.
- Система версионирования данных – dvc + minio.
- workflow менеджер – dvc.
- Инструмент контроля codestyle.
  - Линтеры – pylint, flake8, mypy, bandit;
  - Форматтер - black.
- Трекинг экспериментов – mlflow по сценарию 4 (СУБД PostgreSQL, s3 minio).
- CLI: click.
- Модель – XGBClassifier.
- Тестирование: pytest, great expectations, dvc repro.
- Api – FastAPI+Uvicorn.
- Runtime – docker.

Проблемы и недостатки текущего workflow и получившихся результатов. Возможные улучшения.

- Для стабильного функционирования API сервиса необходимо дополнительно использовать какой-либо менеджер очередей, возможно в связке nginx;
- Необходимо добавить end-to-end тестирование работы сервиса, в том числе напрашивается dev и prod среда;
- Работу сервиса необходимо добавить в какую-либо систему мониторинга (которая по легенде уже обязательно должны быть в организации);
- При каждом удачно разрешенном merge-request в ветку main, докер контейнер нашего сервиса будет остановлен и перезапущен, даже если внесенные изменения не затрагивают функционал самого сервиса, необходимо проводить перезапуск только в случае необходимости.