Прошян Г.А. М4250.
Инфраструктура больших данных
Лабораторная работа 2. Вариант 9

Взаимодействие с источниками данных

В данном проекте мной проивзодится симуляция эксплуатации модели. Предполагается, что в базу данных поступают новые данные (частоты отраженного звука от неклассифицированных объектов) с подводной лодки. Сервис модели должен читать данные из БД и записывать туда предсказания для пока не классифициированных объектов.

Все взаимодействие сервиса модели с сервисом БД производится в скрипте inference.py.

В качестве базы данных была использована greenplum.

Структура БД

Хранится две таблицы:

1. `frequencies` с полем `id` и 60-ю полями вида `freq_{i}`, где i - порядок частоты.
2. `predictions` с полями:
   - `prediction_id`
   - `frequencies_id` - соответствует `frequencies.id`
   - `prediction` - Для каждого объекта хранит строку `'M'`, если объект является металлическим цилиндром (мина), `'R'`, если объект является камнем
   - `m_probability` - Вероятность, что объект является миной

Наполнение базы данных

Предполагается, что sql запросы на заполнение базы данных поступают из отдельного сервиса, который не представлен в проекте.

Заполнение БД осуществлено в файле `init.sql`. База данных наполняется всем датасетом `./data/sonar.all-data`. Файл `init.sql` генерируется скриптом ./src/create_init_sql.py.

Взаимодействие с БД

- С помощью библиотеки `greenplumpython` производим `left join exluding inner join` таблиц `frequencies` и `predictions`. Таким образом мы получаем входные данные модели только для необработаных моделью объектов и id этих объектов (`frequencies.id`).
- Предсказываем класс для каждого объекта
- Создаем pd.DataFrame с полями `prediction_id`, `prediction`, `m_probability`
- Производим insert в таблицу `predictions`

Аутентификация/авторизация

Credentials, используемые в БД:

- POSTGRES_DB
- POSTGRES_PASSWORD
- POSTGRES_USER
- POSTGRES_HOST_AUTH_METHOD

Эти данные хранятся в github secrets и передаются в docker-compose.yml как переменные окружения. В `inferece.py` они передаются как аргументы командной строки.

Результаты функционального тестирования и скрипты конфигурации CI/CD pipeline

Результаты unit тестов:

```
Name                                    Stmts   Miss  Cover   Missing
---------------------------------------------------------------------
src\dataset.py                             18      2    89%   20, 26
src\logger.py                              26      0   100%
src\model.py                                7      0   100%
src\prepare_data.py                       111     49    56%   50-61, 67-72, 84-110,
src\unit_tests\test_dataset.py             25      0   100%
src\unit_tests\test_model.py               23      0   100%
src\unit_tests\test_prepare_data.py        29      0   100%
```

```
----------------------------------------------------------------------
TOTAL                                   239      51     79%
```

Результат функционального теста test_0:

```
model: mlp
model params:
  input_size: '60'
  hidden_size: '40'
  output_size: '2'
  lr: '0.01'
  model_optimizer_loss_dict_path: .\experiments\mlp_adam_ce.pkl
accuracy: '1.0'
```

Результат функционального теста test_1:

```
model: mlp
model params:
  input_size: '60'
  hidden_size: '40'
  output_size: '2'
  lr: '0.01'
  model_optimizer_loss_dict_path: .\experiments\mlp_adam_ce.pkl
accuracy: '1.0'
```

CI / CD представлен в файле ./.github/workflows/CI%20CD.yml:

```
name: CI CD

on:
  push:
    branches:
      - development
      - main
  pull_request:
    branches:
      - main

jobs:
  ci:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v3
```

```
        with:
          python-version: 3.11

      - name: Set up Docker
        uses: docker/setup-qemu-action@v2

      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_HUB_USERNAME }}
          password: ${{ secrets.DOCKER_HUB_PASSWORD }}

      - name: Build and push
        uses: docker/build-push-action@v4
        with:
          context: .
          push: true
          tags: ${{ secrets.DOCKER_HUB_USERNAME }}/mle-mines-vs-rocks

  cd:
    needs: ci
    runs-on: ubuntu-latest

    # env:
    #   POSTGRES_USER: ${{ secrets.POSTGRES_USER }}
    #   POSTGRES_PASSWORD: ${{ secrets.POSTGRES_PASSWORD }}
    #   POSTGRES_DBNAME: ${{ secrets.POSTGRES_DBNAME }}

    steps:

      - name: Checkout
        uses: actions/checkout@v3

      - name: Create .env file
        run: |
          touch .env
          echo POSTGRES_USER="${{ secrets.POSTGRES_USER }}" >> .env
          echo POSTGRES_PASSWORD="${{ secrets.POSTGRES_PASSWORD }}" >> .env
          echo POSTGRES_DBNAME="${{ secrets.POSTGRES_DBNAME }}" >> .env

      - name: Set up Docker
        uses: docker/setup-qemu-action@v2

      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_HUB_USERNAME }}
          password: ${{ secrets.DOCKER_HUB_PASSWORD }}

      - name: Add google service account key
        run: echo ${{ secrets.GOOGLE_SERVICE_ACCOUNT_JSON_KEY_BASE64_ENCODED }} |

      - name: Install DVC
```

```
        run: pip install dvc dvc-gdrive

      - name: Prepare DVC
        run: |
          dvc remote modify myremote gdrive_use_service_account true
          dvc remote modify myremote --local gdrive_service_account_json_file_pat

      - name: DVC PULL
        run: dvc pull

      - name: Pull images
        run: docker-compose pull

      - name: Run scripts and tests
        run: docker-compose up --abort-on-container-exit --build --force-recreate
```

6. Результаты функционального тестирования и скрипты конфигурации CI/CD pipeline приложить к отчёту.

Результаты работы:

1. Отчёт о проделанной работе;
2. Ссылка на репозиторий GitHub;
3. Ссылка на docker image в DockerHub;
4. Актуальный дистрибутив модели в zip архиве.

Классический жизненный цикл разработки моделей машинного обучения

Данный проект имеет цель на элементарной задаче потренироваться ставить воспроизводимые эксперименты с моделями машинного обучения. В проекте используется DVC для версионирования данных и моделей и хранения их в удаленном хранилище. Для CI/CD используется github actions. Эксперименты запускаются в docker контейнере, что позволяет легко переносить их на другие машины и воспроизводить эксперименты в одинаковых условиях.

Данные

В качестве датасета был использован Connectionist Bench (Sonar, Mines vs. Rocks). Датасет содержит 208 объектов, каждый из которых описывается 60 признаками. Признаки - амплитуды частот звукового сигнала, отраженного от объекта. Каждый объект принадлежит к одному из двух классов: `R` - камень, `M` - мина (металлический цилиндр).

Модель

В качестве модели был выбран многослойный перцптрон с одним скрытым слоем.

Результат обучения:

```
Epochs: 100%|██████| 80/80 [00:01<00:00, 48.04it/s, epoch 79  train: accuracy: 0.
```

Скрипты

В notebooks/classification_rocks_vs_mines.ipynb произведено обучение модели на представленном датасете. В данном блокноте код написан максимльно просто, его цель - "схематично" продемонстрировать пайплайн минимальными средствами.

Этот блокнот был переписан в виде множества скриптов, которые находятся в папке src

- train.py - обучение модели
- logger.py - определение класса Logger. Основной его метод - get_logger, который возвращает логгер с заданным именем. "Под капотом" вызывается logging.getLogger и производится настройка логгера.
- model.py - определение класса модели
- prepare_data.py - определение класса DataPreparer, основной метод которого - split_data, который разбивает данные на тренировочную и тестовую выборки и сохраняет путик ним в `config.ini`
- dataset.py - определение класса SonarDataset (наследник torch.utils.data.Dataset). Получает путь к X.csv и y.csv. При образении по индексу возвращает признаки и метки в виде torch.Tensor
- functional_test.py - функциональное тестирование. Для каждого теста из ./tests/ измеряет accuracy модели. Записывает в директории с названиями вида `./experiments/exp_{имя_теста_из_директория_tests}_{дата_и_время}` лог теста и yaml файл с параметрами модели использованной модели.

Unit тесты

Unit тесты реализованы с помощью библиотеки `unittest`. Тесты находятся в директории ./src/unit_tests. Некоторые проверки:

- Детермнированность работы DataPreparer.split_data()
- Детерменированность forward() модели
- Корректность типов данных всех элементов тренировочного и тестового датасетов

Результаты тестирования

```
Name                                    Stmts   Miss  Cover   Missing
---------------------------------------------------------------------
src\dataset.py                             18      2    89%   20, 26
src\logger.py                              26      0   100%
src\model.py                                7      0   100%
src\prepare_data.py                       111     49    56%   50-61, 67-72, 84-110,
src\unit_tests\test_dataset.py             25      0   100%
src\unit_tests\test_model.py               23      0   100%
src\unit_tests\test_prepare_data.py        29      0   100%
---------------------------------------------------------------------
TOTAL                                     239     51    79%
```

Config.ini

В config.ini хранятся:

- Гиперпараметры модели. Записываются в скрипте train.py.
- Пути к разделенным данным. Записываются в результате работы скрипта prepare_data.py. Используются в скрипте train.py
- Параметры скрипта prepare_data.py. Каждый параметр равен последнему значению, которое переданному через командную строку. Если параметр не был передан, то он берется из config.ini. Если очистить config.ini и не передавать параметры через командную строку, то скрипт prepare_data.py закончится ошибкой.

DVC

DVC используется для версионирования исходных данных `data/sonar.all-data` и модели `experiments/mlp_adam_ce.pkl`.

В качестве remote хранилища была создана папка на google drive с id `1lh_wUfw88ceVCL04UtT0e0zQCoFpqLxY`. Всем в интернете был дан доступ на чтение google drive папки, благодаря чему любой авторизованный в google drive пользователь может производить команды `dvc get . data/sonar.all-data` и `dvc pull`. DVC автоматически генерирует ссылку на Google OAuth2 при первом выполнении одной из данных команд.

DVC команды, которые были использованы для добавления удаленного хранилища и сохранения на нем первых версий отслеживаемых файлов:

```
dvc init
dvc remote add -d myremote gdrive://1lh_wUfw88ceVCL04UtT0e0zQCoFpqLxY
dvc remote modify myremote gdrive_acknowledge_abuse true
dvc add data/sonar.all-data
dvc add experiments/mlp_adam_ce.pkl
dvc push

git add data/sonar.all-data.dvc data/.gitignore experiments/mlp_adam_ce.pkl exper
```

Для CI/CD был создан Google Cloud проект, в котором был создан service account. Service account'у были даны права на редактирование папки, являющейся remote хранилищем. Json ключ от service accaunt'а был закодирован base64 и записан в github secrets.

Более подробно с произведенной настройкой можно ознакомиться в официальной документации (разделы `Using a custom Google Cloud project (recommended)` и `Using service accounts`)

CI/CD

CI и CD являются job'ами одного github actions workflow'а. Workflow находится в .github/workflows/CI CD.yml. Workflow запускается при pull request'е в ветку main и при push'е в ветку development (push в ветку main запрещен). CD job запускается после успешного прохождения CI job.

На этапе CI производится сборка docker образа и его отправка в docker hub. В docker образ не включены файлы, отслеживаемые dvc.

На этапе CD на runner'е создается файл с ключом от google service аккаунта, производится dvc pull, который скачивает файл `data/sonar.all-data`, выполнются `docker-compose pull` и `docker-compose up`, в результате чего запускаются скрипты и тесты (соответствующая комманда прописана в `docker-compose.yml`). Файл `data/sonar.all-data` оказывается доступен внутри контейнера благодаря volume монтированию.

Docker

Docker образ на docker hub

Локальный запуск

Чтобы запустить проект локально необходимо:

1. Склонировать репозиторий
2. Установить dvc
3. Установить docker
4. Получить docker образ либо собрав его локально:

```
docker build -t proshian/mle-mines-vs-rocks:latest .
```

либо скачав его с docker hub:

```
docker pull proshian/mle-mines-vs-rocks:latest
```

5. Произвести dvc pull и пройти аутентификаицю в google drive, если она не была произведена ранее (OAuth2 ссылка будет сгенерирована автоматически).

```
dvc pull
```

6. Выполнить docker-compose up:

```
docker-compose up
```