

Министерство науки и высшего образования Российской Федерации

Национальный исследовательский университет ИТМО

Инфраструктура больших данных

Весна

2025

Лабораторная работа №1

## КЛАССИЧЕСКИЙ ЖИЗНЕННЫЙ ЦИКЛ РАЗРАБОТКИ МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ

Цель работы:

Получить навыки разработки CI/CD pipeline для ML моделей с достижением метрик моделей и качества.

Ход работы:

1. Создать репозитории модели на GitHub, регулярно проводить commit+push в ветку разработки, важна история коммитов;
2. Провести подготовку данных для набора данных, согласно варианту задания;
3. Разработать ML модель с ЛЮБЫМ классическим алгоритмом классификации, кластеризации, регрессии и т. д.;
4. Конвертировать модель из \*.ipynb в .py скрипты, реализовать API сервис с методом на вывод модели, фронтальная часть по желанию;
5. Покрыть код тестами, используя любой фреймворк/библиотеку;
6. Задействовать DVC;
7. Использовать Docker для создания docker image.
8. Наполнить дистрибутив конфигурационными файлами
  - 8.1. config.ini: гиперпараметры модели;
  - 8.2. Dockerfile и docker-compose.yml: конфигурация создания контейнера и образа модели;

- 8.3. requirements.txt: используемые зависимости (библиотеки) и их версии;
  - 8.4. dev\_sec\_ops.yml: подписи docker образа, хэш последних 5 коммитов в репозитории модели, степень покрытия тестами (необязательно);
  - 8.5. scenario.json: сценарии тестирования запущенного контейнера модели (необязательно).
- 9. Создать CI pipeline (Jenkins, Team City, Circle CI и др.) для сборки docker image и отправки его на DockerHub, сборка должна автоматически стартовать по pull request в основную ветку репозитория модели;
  - 10. Создать CD pipeline для запуска контейнера и проведения функционального тестирования по сценарию, запуск должен стартовать по требованию или расписанию или как вызов с последнего этапа CI pipeline;
  - 11. 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.

Обязательно обернуть модель в контейнер (этап CI) и запустить тесты внутри контейнера (этап CD).

## Отчет

1. Создаем и клонируем репозиторий на GitHub. Организуем структуру проекта без кода и данных.

- /data - содержит данные проекта (.csv таблица и .zip файлы).
- /experiments - веса и параметры конфигураций моделей.
- /notebooks - .ipynb файлы с экспериментами.
- /src - .py файлы с кодом предобработки данных, обучения и предсказания моделей, дополнительно содержит папку с unit тестами.
- /tests - .json файлы для функциональных тестов.
- /CI и /CD папки с кодом пайплайнов сборки на Jenkins.
- Dockerfile - файл с кодом для сборки изображения.
- docker-compose.yml - файл с кодом инфраструктуры контейнеров.
- requirements.txt - файл с Python зависимостями.

2. Датасет fashion-mnist состоит из двух csv таблиц для train и test подвыборок размером 133 и 22 Мб соответственно. Поскольку удаленного хранилища для хранения данных с помощью технологии DVC нет, то было решено хранить их в репозитории Git. Чтобы не возникало ошибок с лимитом по памяти во время передачи файлов, данные хранятся в формате .zip в папке /data. Код обработки данных располагается по адресу src/preprocess.py. Процесс обработки организован следующим образом: распаковка данных с помощью стандартной библиотеки zipfile, далее для удобства обучения моделей машинного обучения данные разделяются на метки и изображения. Параллельно в конфигурационный файл config.ini записываются пути к обработанным данным.

3. Для обучения моделей были использованы следующие алгоритмы из библиотеки sklearn RandomForestClassifier, DecisionTreeClassifier,

GaussianNB, KNeighborsClassifier, LogisticRegression. Предварительно данные были нормализованы при помощи StandardScaler.

Таблица 1. Метрики на test подвыборке.

Model	Test accuracy
RandomForestClassifier	0.8861
DecisionTreeClassifier	0.8123
GaussianNB	0.5791
KNeighborsClassifier	0.863
LogisticRegression	0.8443

4. Чтобы предоставить доступ к модели извне, будет использована библиотека `fastapi` и `uvicorn`. Необходимо прописать эти зависимости в файле `requirements.txt`. Также добавим команду установки утилиты `curl` в `Dockerfile` веб сервиса, чтобы сделать тестовый запрос. Запуск веб сервера и код тестового запроса добавим в `docker-compose.yml` файл. Скрипт запустит веб-сервер на фоне и отправит заранее подготовленный json файл на реализованный endpoint. Код веб-сервиса находится по пути `src/app.py`.

5. Для тестирования использовался фреймворк `unittest`. Тестировались файлы предобработки данных и обучения моделей. Начнем с тестирования файла предобработки `/src/unit_tests/test_preprocess.py`:

**setUp** - инициализирует класс, который инкапсулирует предобработку данных.

**test\_split\_data** - вызывает метод **split\_data** класса **DataMaker** и проверяет, что возвращаемый результат равен **True**. Если возвращаемый

результат истинный, то файлы с разметками и изображениями для тестовой и тренировочной подвыборок были созданы и существуют.

**test\_split\_data\_labels** - проверяет метод **split\_data\_labels**, который разделяет данные на разметку и изображения. Сам метод возвращает соответственно пути к файлами, которые далее проверяются методом на существованием внутри кода теста.

Тестирование файла обучения осуществляется по пути **/src/unit\_tests/test\_training.py**:

**setUp** - инициализирует класс **MultiModel**, который предоставляет интерфейс для обучения моделей машинного обучения.

**test\_{model\_name}** - проверяет, что возвращаемое значение истинно. Если оно истинно, то веса модели после обучения были сохранены.

6. DVC был интегрирован в проект по следующему алгоритму:

1. прописываем **dvc** в **requirements.txt**
2. **pip install -r ./requirements.txt** - устанавливаем зависимости
3. **dvc init** - создаем файлы **.dvc/.gitignore** и **.dvc/config**
4. **git commit -m 'Initialize DVC'**
5. скачиваем **.zip** файл и кладем его в **/data**
6. **dvc add ./data** - начинаем отслеживать данные; появляется файл метаданных **data.dvc**, который является заглушкой оригинальных данных для системы **Git**.
7. **git add ./data.dvc ./.gitignore** - добавляем сгенерированные файлы
8. **git commit -m "Add raw data"**
9. Чтобы хранить данные будем использовать локальную директорию, которая находится на одном уровне с папкой проекта **../dvcstore**.

**dvc remote add -d myremote ../dvcstore**

10. **dvc push** - загружаем данные на «удаленный» репозиторий

11. **dvc pull** - выгружаем данные с «удаленного» репозитория

7, 8. Файл docker-compose содержит описание одного сервиса **web**. **web** сервис использует образ, созданный из Dockerfile в текущем каталоге. Затем он привязывает контейнер и хост-машину к открытому порту 8000. Также **web** сервис использует общедоступный образ, извлеченный из реестра Docker Hub.

Листинг 1 - Код файла docker-compose.yml

```
services:
  web:
    build: .
    command: bash -c "
      python src/preprocess.py
      && python src/train.py
      && python src/predict.py -m LOG_REG -t func
      && coverage run src/unit_tests/test_preprocess.py
      && coverage run -a src/unit_tests/test_training.py
      && coverage report -m"
    ports:
      - 8000:8000
    image: zarus03/ml-big-data-lab-1:latest
```

Листинг 2 - Код файла Dockerfile.

```
FROM python:3.8-slim
ENV PYTHONUNBUFFERED 1
WORKDIR /app
ADD . /app
RUN pip install -r requirements.txt
```

9. Скачивание Jenkins на локальные компьютер. Интерфейс взаимодействия со средой Jenkins располагается по адресу localhost:8080. Чтобы загружать изображения с Docker Hub во время работы сборки, необходимо залогиниться в самом сервисе. Для этого добавляем Credentials для Docker Hub в систему Jenkins.

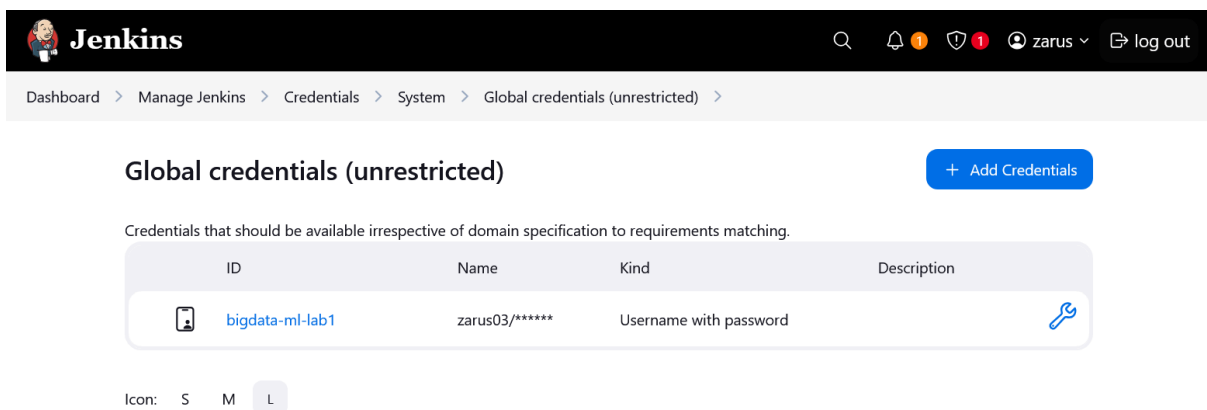


Рисунок 1 - добавляем глобальные Credentials от Docker Hub.

Настраиваем пайплайн CI через репозиторий на Github. В настройках конфигурации пайплайна указываем ссылку на репозиторий, ветку и путь к скрипту.

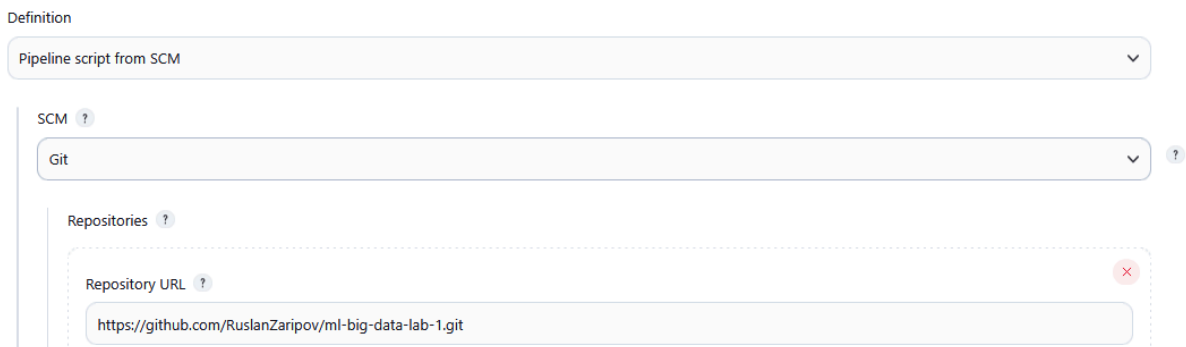


Рисунок 2 - Параметры пайплайна CI (ссылка на репозиторий).

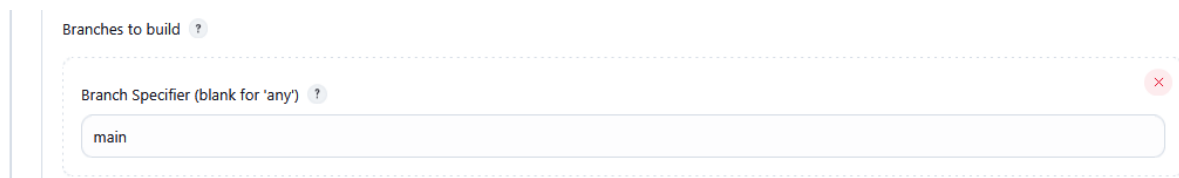
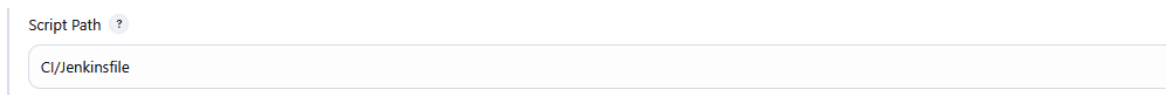


Рисунок 3 - Параметры пайплайна CI (ветка репозитория).

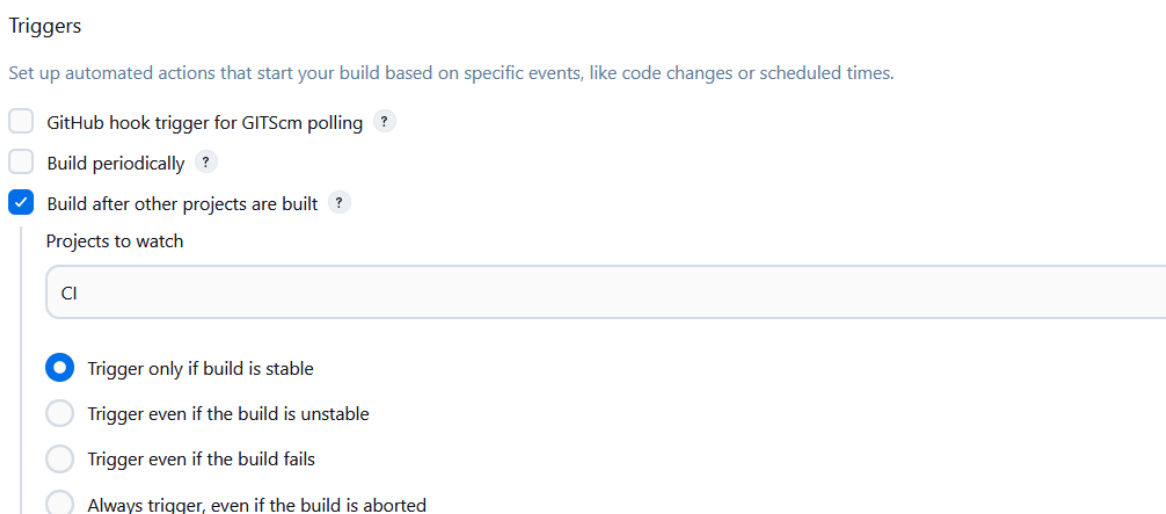


Script Path ?

CI/Jenkinsfile

Рисунок 4 - Параметры пайплайна CI (файл пайплайна на Jenkins).

10. CD пайплайн настраиваем аналогично предыдущему пункту. Дополнительно указываем триггер, который срабатывает по завершению сборки CI пайплайна.



Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

☐ GitHub hook trigger for GITScm polling ?

☐ Build periodically ?

☒ Build after other projects are built ?

Projects to watch

CI

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

☐ Always trigger, even if the build is aborted

Рисунок 5 - Триггер сборки пайплайна CD.

11. С результатами функционального тестирования можно ознакомиться в логах сборки CI пайплайна. В процессе сборки web сервиса запускается файл predict.py с типом модели LOG\_REG и типом тестов func. В случае успешного завершения функционального тестирования в логах будет присутствовать запись с префиксом INFO и соответствующим содержанием (см. Рисунок X). В случае ошибки в логах будет запись с перфиксом ERROR и причиной ошибки.

```
17:55:14 2025-02-25 14:55:14,194 - __main__ - INFO - /app/experiments/log_reg.sav passed func test /app/tests/test_0.json
17:55:14 2025-02-25 14:55:14,313 - __main__ - INFO - /app/experiments/log_reg.sav passed func test /app/tests/test_1.json
```

Рисунок 6 - скриншот логов успешного функционального тестирования.



CI пайплайн состоит из этапов клонирования репозитория, авторизации на DockerHub, сборки и запуска докер контейнера. Также прописаны этапы для вывода логов контейнера.

### Листинг 3. Код CI пайплайна.

```
pipeline {
  agent any

  environment {
    DOCKERHUB_CREDS = credentials('bigdata-ml-lab1')
    LC_ALL = 'en_US.UTF-8'
    LANG = 'en_US.UTF-8'
    LANGUAGE = 'en_US.UTF-8'
  }

  options {
    timestamps()
    skipDefaultCheckout(true)
  }
  stages {
    stage('Clone github repository') {
      steps {
        cleanWs()
        bat 'chcp 65001 && git clone -b main
https://github.com/RuslanZaripov/ml-big-data-lab-1.git'
      }
    }

    stage('Checkout repo dir') {
      steps {
        bat 'chcp 65001 && cd ml-big-data-lab-1 && dir'
      }
    }

    stage('Login') {
      steps {
        bat 'chcp 65001 && docker login -u %DOCKERHUB_CREDS_USR% -p
%DOCKERHUB_CREDS_PSW%'
      }
    }

    stage('Create and run docker container') {
      steps {
        script {
          try {
            bat 'chcp 65001 && cd ml-big-data-lab-1 && docker-compose
build'
          }
          finally {
            bat 'chcp 65001 && cd ml-big-data-lab-1 && docker-compose up
-d'
```

```

    }
  }
}

stage('Checkout container logs') {
  steps {
    dir('ml-big-data-lab-1') {
      bat '''
      docker-compose up -d
      for /f %%i in ('docker ps -qf
"name=^ml-big-data-lab-1-web-1"') do set containerId=%%i
      echo %containerId%
      IF "%containerId%" == "" (
        echo "No container running"
      )
      ELSE (
        docker logs --tail 1000 -f %containerId%
      )
      '''
    }
  }
}

stage('Checkout coverage report') {
  steps {
    dir('ml-big-data-lab-1') {
      bat '''
      docker-compose logs -t --tail 10
      '''
    }
  }
}

stage('Push') {
  steps {
    bat 'chcp 65001 && docker push
zarus03/ml-big-data-lab-1:latest'
  }
}

post {
  always {
    bat 'chcp 65001 && docker logout'
  }
}
}

```

Код CD пайплайна сначала авторизируется на DockerHub, потом достает сгенерированное на этапе CD изображение, запускает и останавливает контейнер.

#### Листинг 4. Код CD пайплайна.

```
pipeline {
  agent any

  environment {
    DOCKERHUB_CREDS = credentials('bigdata-ml-lab1')
    LC_ALL = 'en_US.UTF-8'
    LANG = 'en_US.UTF-8'
    LANGUAGE = 'en_US.UTF-8'
  }

  options {
    timestamps()
    skipDefaultCheckout(true)
  }
  stages {
    stage('Login') {
      steps {
        bat 'docker login -u %DOCKERHUB_CREDS_USR% -p
%DOCKERHUB_CREDS_PSW%'
      }
    }

    stage('Pull image') {
      steps {
        bat '''
        docker pull zarus03/ml-big-data-lab-1:latest
        '''
      }
    }

    stage('Run container') {
      steps {
        bat '''
        docker run --name ml-big-data-lab-1 -p 80:5556 -d
zarus03/ml-big-data-lab-1:latest
        '''
      }
    }
  }

  post {
    always {
      bat 'docker stop ml-big-data-lab-1 && docker logout'
    }
  }
}
```