МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

Отчет по лабораторной работе № 3
по курсу «Инженерия данных»

Тема: «Airflow и MLflow - логгирование экспериментов и версионирование моделей»

Выполнил Самигуллин Равиль
Группа 6233 – 010402D
Преподаватель Парингер Р.А

Самара

Содержание

Техническое задание

Airflow и MLflow - логгирование экспериментов и версионирование моделей

В рамках данной лабораторной работы предлагается построить два пайплайна:

1. Пайплайн, который обучает любой классификатор из sklearn по заданному набору параметров.
2. Пайплайн, который выбирает лучшую модель из обученных и производит её хостинг.

Для построения такого пайплайна воспользуемся следующими инструментами:

- Apache Airflow
- MLflow

Задание на лабораторную работу

Пайплайн для обучения классификаторов

Построенный пайплайн будет выполнять следующие действия поочередно:

1. Производить мониторинг целевой папки на предмет появления новых конфигурационных файлов классификаторов (в форматах .json или .yaml).
2. Обучать классификатор в соответствии с полученными параметрами.
3. Производить логгирование параметров модели в **MLflow**.
4. Производить логгирование процесса обучения **MLflow**.
5. Производить тестирование модели и сохранять его результаты в **MLflow**.
6. Сохранять обученный классификатор в model registry **MLflow**.

Пайплайн для хостинга лучшей модейли

Построенный пайплайн будет выполнять следующие действия поочередно:

1. В соответствии с таймером производит валидацию новых моделей из model registry.
2. Модель с лучшим показателем метрики переводится на stage: Production
3. (Опционально) произвести хостинг лучшей модели

Ход выполнения

Задание 1: Пайплайн для инференса данных.

1. Выбор данных и классификаторов для обучения:

Перед обучением и созданием выбрал несколько классификаторов из sklearn которые будут обрабатывать наши данные, для обработки данных так же нужно выбрать датасет на котором планирую обучать наши классификаторы, а так же я решил делать предобработку данных перед обучением, а так как я всё равно буду записывать аргументы и настройки классификаторов в json, то решил туда же добавлять всё то что хочу сделать с данными и то какие данные мы будем обучать, мне показалось это правильным так как, наш пайплайн сможет обрабатывать любые данные и обучать любые модели (из sklearn разумеется).

Я выбрал следующие модели:

- SGDClassifier
- RandomForestClassifier
- SVC
- KNeighborsClassifier
- DecisionTreeClassifier
- LogisticRegression
- Perceptron

В качестве датасета я выбрал titanic классический датасет который так же требует предобработки перед обучением, так как имеет множество не закодированных значений представленных в виде категорий, а так же значений которые не имеют смысла для классификации, как например номер билета или имя пассажира.

2. Создание json файла с настройкой конфигурации.

В json я записал данные по dataset расположение файлов csv, имена столбцов необходимые для кодировки, имена столбцов для удаления, после идут настройки классификаторов, модуль для импорта библиотеки, имя классификатора, и аргументы которые мы используем для обучения классификации.

3. Разработка DAG

dag_lr3_task_1.py > ...

```python
from airflow import DAG
from datetime import datetime
from airflow.providers.docker.operators.docker import DockerOperator
from docker.types import Mount
from airflow.operators.bash_operator import BashOperator
from airflow.sensors.filesystem import FileSensor
import os

os.environ["AWS_ACCESS_KEY_ID"] = "minio"
os.environ["AWS_SECRET_ACCESS_KEY"] = "minio123"
os.environ["MLFLOW_S3_ENDPOINT_URL"] = "http://minio:9000"

default_args = {
    'owner': 'darti',
    'start_date': datetime(2024, 1, 1),
    'retries': 1,
}

dag = DAG(
    'lr3_learning_models',
    default_args = default_args,
    description='dag for learn sclearn models and data',
    schedule_interval=None,
)

wait_for_new_file = FileSensor(
    task_id='wait_for_new_file',
    poke_interval=10,  # Interval to check for new files (in seconds)
    filepath= '/opt/airflow/data/lr3/data/*.json',
    fs_conn_id='fs_connection_default', # Target folder to monitor
    dag=dag,
)

train_model = BashOperator(
    task_id='train_model',
    bash_command='/opt/***/data/lr3/train.py {{ params.config_file }}',
    params={'config_file': '/opt/airflow/data/lr3/data/conf.json'},
    dag=dag,
)

wait_for_new_file >> train_model
```

В данном dag, мы сначала ждём файл wait for new file в папку мы ждём json.

После мы через bash operator выполняем скрипт train.py передавая ему в качестве аргумета config_file.

4. Подготовка данных

predobr.py > ...

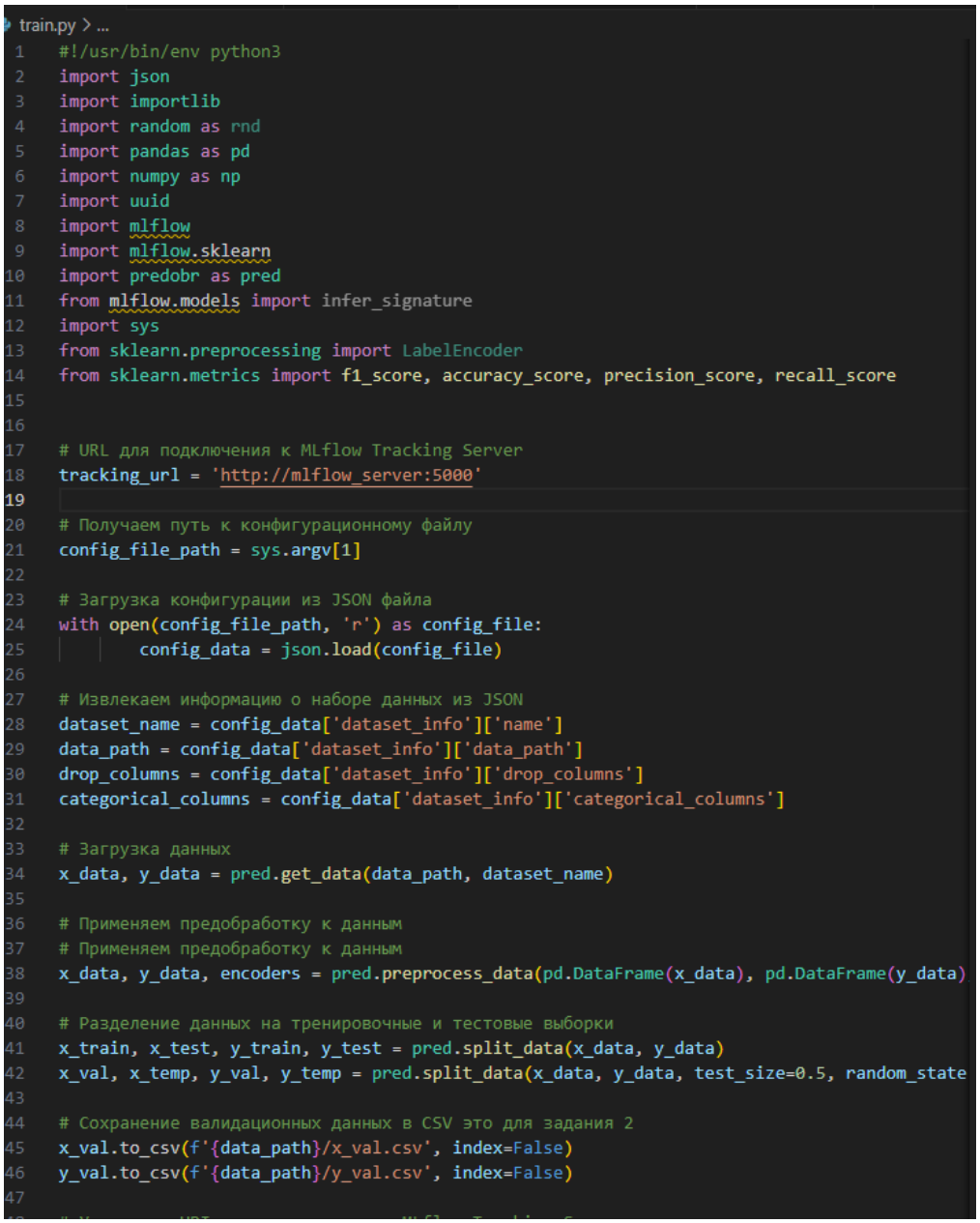
```python
#!/usr/bin/env python3
import mlflow
import mlflow.sklearn
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score
from sklearn.preprocessing import LabelEncoder

def get_data(data_path: str, labels: str):
    """Загрузка данных из CSV файлов."""
    df_features = pd.read_csv(f'{data_path}/{labels}_features.csv')
    df_target = pd.read_csv(f"{data_path}/targets_{labels}.csv")
    return df_features, df_target


def preprocess_data(df_features, df_labels, drop_columns, categorical_columns):
    # Удаляем ненужные столбцы
    df_features.drop(columns=drop_columns, inplace=True)

    # Обрабатываем категориальные столбцы, преобразовывая их в числовые
    for column in categorical_columns:
        df_features[column] = pd.Categorical(df_features[column]).codes

    # Заполняем пропуски в числовых столбцах медианой
    df_features.fillna(df_features.median(numeric_only=True), inplace=True)

    return df_features, df_labels


def split_data(x_data, y_data, test_size=0.2, random_state = 42):
    """Разделение данных на тренировочные и тестовые выборки."""
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=test_si
    return x_train, x_test, y_train, y_test


def log_metrics(config, y_test, y_pred):
    """Логирование метрик модели в MLflow."""
    mlflow.log_params(config)
    mlflow.log_metrics({
        "f1_score": f1_score(y_test, y_pred, average='weighted'),
        "accuracy": accuracy_score(y_test, y_pred),
        "precision": precision_score(y_test, y_pred, average='weighted'),
        "recall": recall_score(y_test, y_pred, average='weighted')
    })

```

Используя функции написанные в файле predobr.py

Мы удаляем ненужные столбцы, обрабатываем категориальные переменные (преобразует их в числовые коды) и заполняем пропуски в числовых столбцах медианой.

5. Обучение модели

Обучение модели а так же вызов всех вспомогательных функций реализован в файле train.py



```python
#!/usr/bin/env python3
import json
import importlib
import random as rnd
import pandas as pd
import numpy as np
import uuid
import mlflow
import mlflow.sklearn
import predobr as pred
from mlflow.models import infer_signature
import sys
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score


# URL для подключения к MLflow Tracking Server
tracking_url = 'http://mlflow_server:5000'

# Получаем путь к конфигурационному файлу
config_file_path = sys.argv[1]

# Загрузка конфигурации из JSON файла
with open(config_file_path, 'r') as config_file:
        config_data = json.load(config_file)

# Извлекаем информацию о наборе данных из JSON
dataset_name = config_data['dataset_info']['name']
data_path = config_data['dataset_info']['data_path']
drop_columns = config_data['dataset_info']['drop_columns']
categorical_columns = config_data['dataset_info']['categorical_columns']

# Загрузка данных
x_data, y_data = pred.get_data(data_path, dataset_name)

# Применяем предобработку к данным
# Применяем предобработку к данным
x_data, y_data, encoders = pred.preprocess_data(pd.DataFrame(x_data), pd.DataFrame(y_data)

# Разделение данных на тренировочные и тестовые выборки
x_train, x_test, y_train, y_test = pred.split_data(x_data, y_data)
x_val, x_temp, y_val, y_temp = pred.split_data(x_data, y_data, test_size=0.5, random_state

# Сохранение валидационных данных в CSV это для задания 2
x_val.to_csv(f'{data_path}/x_val.csv', index=False)
y_val.to_csv(f'{data_path}/y_val.csv', index=False)
```

train.py > ...

```python

# Установка URI для подключения к MLflow Tracking Server
mlflow.set_tracking_uri(tracking_url)

# Генерация уникального ID эксперимента
id_current_experiment = str(uuid.uuid4())
exp_id = mlflow.create_experiment(id_current_experiment)
mlflow.set_experiment(exp_id)

# Сохранение ID эксперимента в файл так же надо для 2 задания
with open(f'{data_path}/experiment_id.txt', 'w') as f:
    f.write(id_current_experiment)

# Проходим по каждой конфигурации в JSON
for i, config in enumerate(config_data['configs']):
    mlflow.start_run(run_name=config['classificator'], experiment_id=exp_id)

    # Загрузка модуля и классификатора
    module = importlib.import_module(config['module'])
    classificator_class = getattr(module, config['classificator'])
    model = classificator_class(**config['args'])

    # Проверка на наличие метода partial_fit
    if hasattr(model, 'partial_fit'):
        max_iter = config['args'].get('max_iter', 50)
        for iter in range(max_iter):
            model.partial_fit(x_train, y_train, classes=np.unique(y_train))
            y_pred = model.predict(x_test)
            mlflow.log_metrics({
                f"f1_score": f1_score(y_test, y_pred, average='weighted'),
                f"accuracy": accuracy_score(y_test, y_pred),
                f"precision": precision_score(y_test, y_pred, average='weighted'),
                f"recall": recall_score(y_test, y_pred, average='weighted')
            })
    else:
        model.fit(x_train, y_train)  # Стандартное обучение

    # Тестирование модели на тестовых данных
    y_pred_test = model.predict(x_test)
    # Логирование метрик на тестовых данных
    mlflow.log_metrics({
        "f1_score_test": f1_score(y_test, y_pred_test, average='weighted'),
        "accuracy_test": accuracy_score(y_test, y_pred_test),
        "precision_test": precision_score(y_test, y_pred_test, average='weighted'),
        "recall_test": recall_score(y_test, y_pred_test, average='weighted')
    })
```

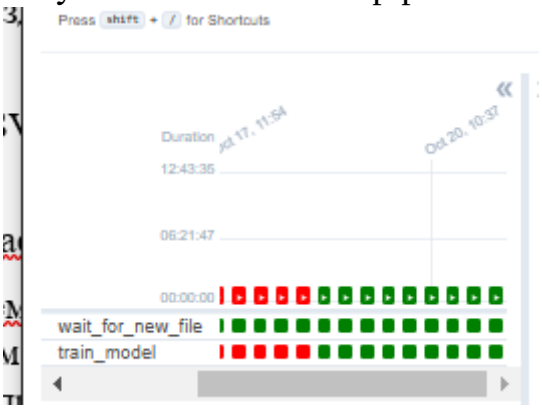
```python

    # Сохранение информации о модели в CSV
    model_info = mlflow.sklearn.log_model(sk_model=model, artifact_path=f'{config["module"

    # Регистрация модели в Model Registry
    model_uri = model_info.model_uri
    registered_model = mlflow.register_model(model_uri=model_uri, name=config['classificat


    df = pd.DataFrame({"name": config['classificator'], "uri": model_info.model_uri}, inde
    df.to_csv(f'{data_path}/models.csv', mode='a', header=False)  # Добавляем заголовки то

    mlflow.end_run()
```

В основном скрипте мы:

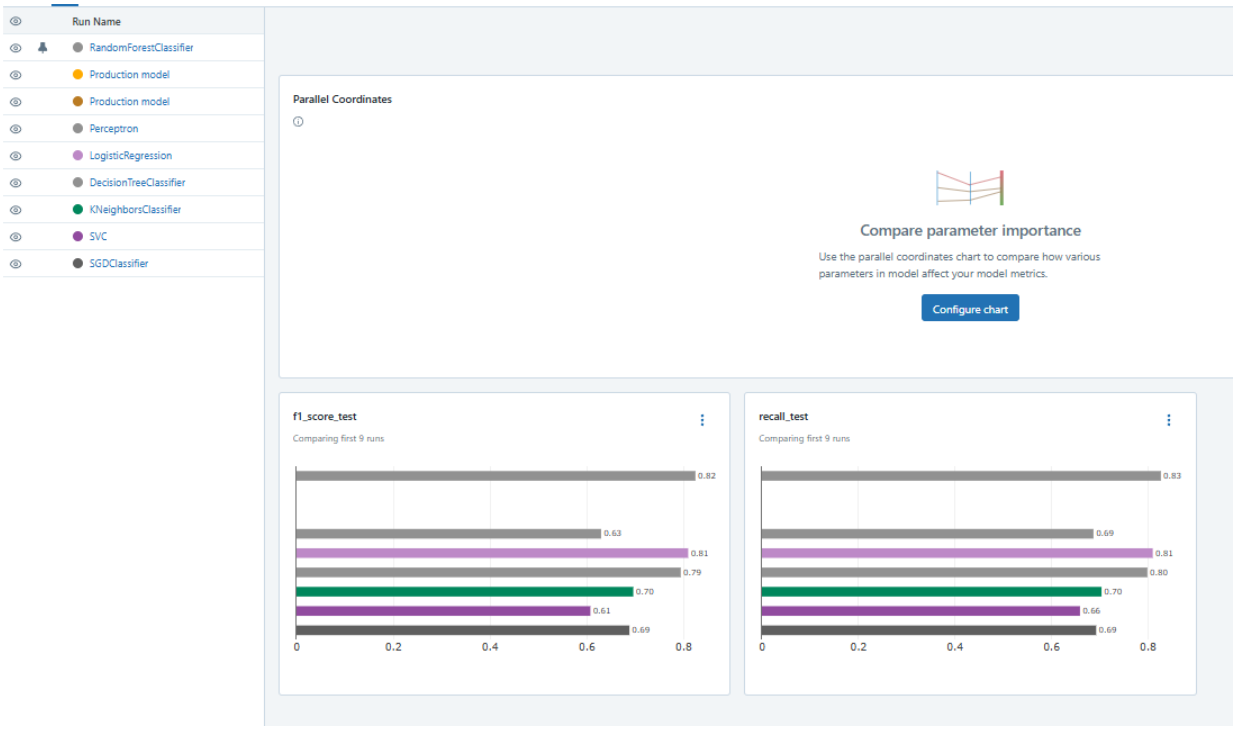
- Загружаем путь к конфигурационному файлу, читает его в формате JSON и извлекает необходимую информацию о наборе данных и настройках.
- Загружаем данные с помощью функции **get_data**.

- Применяем предварительную обработку и разделяет данные на тренировочные и тестовые выборки.
- Сохраняем валидационные наборы данных в CSV. (Это так же для второго задания)
- Устанавливаем URI для подключения к MLflow Tracking Server.
- Создаем новый эксперимент в MLflow и логируем его ID. (Сейчас, я думаю, что лучше сохранять не в txt, а в каком-нибудь csv, чтобы хранить не только текущий эксперимент, но и все предыдущие)
- Проходим по каждой конфигурации в JSON и создаем классификатор, загружая соответствующий модуль и класс.
- Обучаем модель, если в ней реализован метод **partial_fit** то обучаем по steps и сохраняем метрики каждого шага, я это сделал что бы можно было отследить процесс обучения у моделей у которых это возможно, или используем стандартное обучение.
- Логируем метрики на тестовых данных.
- Сохраняем информацию о модели в систему MLflow и регистрируем модель.
- Сохраняем информацию о модели в CSV файл. (Это нужно для второго шага)

6. Результат выполнения pipeline



Сначала сохранял метрики не правильно и каждая итерация была со своим графиком, после сохранил их с одним именем и получил хорошие графики.



Тут можно посмотреть точность моделей которые мы проверили на тестовых данных.

SGDClassifier

Run ID: 7ef538f854ea4840b7425ea3d0c97ffd

Status: FINISHED

Description Edit

Datasets

Parameters

Metrics (804)

Name	Value
accuracy_iter_0	0.732
accuracy_iter_1	0.654
accuracy_iter_10	0.732
accuracy_iter_100	0.754
accuracy_iter_101	0.76
accuracy_iter_102	0.771
accuracy_iter_103	0.642
accuracy_iter_104	0.76
accuracy_iter_105	0.76
accuracy_iter_106	0.821
accuracy_iter_107	0.788
accuracy_iter_108	0.821

db2759d0-b0b5-4651-82aa-e4cd3e4db576 › Perceptron ›

f1_score

Completed Runs 1/1

Points:

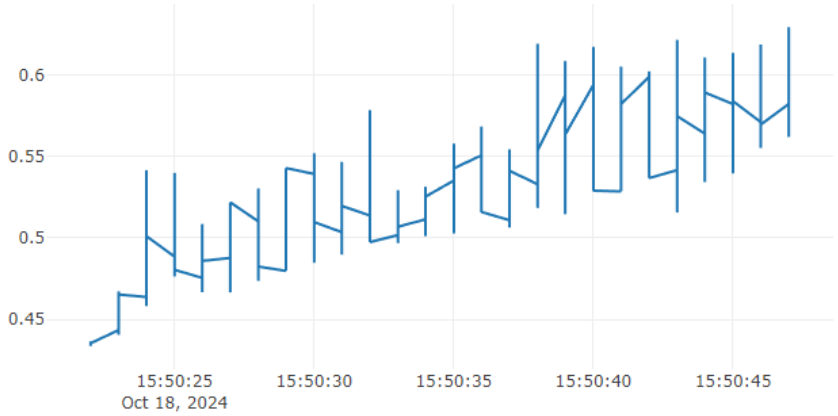
Line Smoothness 78

X-axis:
- Step
- Time (Wall)
- Time (Relative)

Y-axis: f1_score

Y-axis Log Scale:

Download CSV



Metric	Latest	Min	Max
f1_score	0.568 (step=0)	0.434 (step=0)	0.711 (step=0)

db2759d0-b0b5-4651-82aa-e4cd3e4db576 › Perceptron ›

f1_score

Completed Runs 1/1

Points:

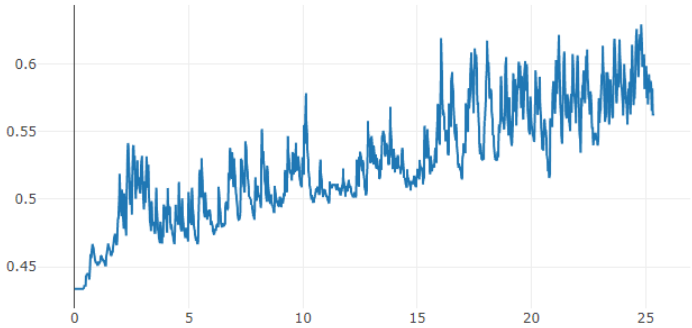
Line Smoothness 78

X-axis:
- Step
- Time (Wall)
- Time (Relative)

Y-axis: f1_score

Y-axis Log Scale:

Download CSV



Metric	Latest	Min	Max
f1_score	0.568 (step=0)	0.434 (step=0)	0.711 (step=0)

И теперь можно посмотреть, как выглядит график обучения процептона или логистической регрессии

Задание 2: Пайплайн для хостинга лучшей модейли

Хостинг лучшей модели.

1. Разработка dag

Сначала производим валидацию моделей после, переводим лучшую модель в promote и пытаемся захостить её.

2. Создания скрипта validate

```python
#!/usr/bin/env python3
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import mlflow
import mlflow.sklearn

tracking_url = 'http://mlflow_server:5000'
data_path = '/opt/airflow/data/lr3'

# Чтение ID текущего эксперимента
with open(f'{data_path}/experiment_id.txt', 'r') as f:
    id_current_experiment = f.read()

mlflow.set_tracking_uri(tracking_url)
mlflow.set_experiment(id_current_experiment)

# Загрузка валидационных данных
x_val = pd.read_csv(f'{data_path}/x_val.csv')
y_val = pd.read_csv(f'{data_path}/y_val.csv')

list_models_for_validate = {}

with mlflow.start_run(run_name="Production model") as start_run:
    models_file = pd.read_csv(f"{data_path}/models.csv", header=0)
    for model_Info in models_file.iterrows():
        name = model_Info[1][1]
        uri = model_Info[1][2]
        print(f"Loading model: {name} from URI: {uri}")  # Логируем информацию о моделях
        list_models_for_validate[name + " " + uri] = mlflow.sklearn.load_model(uri)

    current_results = {}
    for name, model in list_models_for_validate.items():
        prediction = model.predict(x_val)
        current_results[name] = accuracy_score(y_val, prediction)

    # Находим лучшую модель
    best_model_in_list_validate_model = max(current_results, key=current_results.get)
    model_name, model_uri = best_model_in_list_validate_model.split(" ")

    # Сохранение лучшей модели в CSV
    best_model_data = pd.DataFrame({
        "name": [model_name],
        "uri": [model_uri],
        "accuracy": [current_results[best_model_in_list_validate_model]]
    })

    # Проверяем, существует ли уже файл с лучшими моделями
```

```python
# Проверяем, существует ли уже файл с лучшими моделями
try:
    best_models_csv = pd.read_csv(f"{data_path}/best_models.csv")
    # Добавляем новую строку с лучшей моделью
    best_model_data.to_csv(f"{data_path}/best_models.csv", mode='a', header=False, in
except FileNotFoundError:
    # Если файл не существует, создаем его с заголовками
    best_model_data.to_csv(f"{data_path}/best_models.csv", index=False)

print(f"Best model saved: {model_name} with URI: {model_uri}")
```

Этот скрипт делает следующее.

Инициализация и конфигурация:

- Устанавливает URL-адрес MLflow для отслеживания (tracking_url), путь к данным (data_path) и идентификатор текущего эксперимента, который читается из файла experiment_id.txt.
- Устанавливает текущий эксперимент для MLflow.

Загрузка данных:

- Загружает валидационные данные x_val и y_val из CSV-файлов, чтобы использовать их для проверки моделей.

Загрузка моделей для валидации:

- Загружает файл models.csv, содержащий список моделей с их именами и URI.
- Для каждой модели из файла загружает её из MLflow и добавляет в словарь list_models_for_validate, используя комбинацию имени и URI в качестве ключа для каждой модели.

Оценка моделей:

- Для каждой модели в list_models_for_validate делает предсказания на данных x_val и вычисляет точность предсказаний с помощью метрики accuracy_score.
- Сохраняет результаты точности каждой модели в словаре current_results.

Поиск лучшей модели:

- Определяет модель с наивысшей точностью на валидационных данных.
- Извлекает имя и URI этой модели.

Сохранение лучшей модели:

- Создаёт DataFrame best_model_data с информацией о лучшей модели (имя, URI и точность).
- Логирует сообщение о сохранённой лучшей модели.

3. Создание скрипта promote_model

```python
#!/usr/bin/env python3
import pandas as pd
import mlflow
from mlflow import MlflowClient
import subprocess
import requests
import time

tracking_url = 'http://mlflow_server:5000'
data_path = '/opt/airflow/data/lr3'

mlflow.set_tracking_uri(tracking_url)
client = MlflowClient()

models = client.search_registered_models()
for model in models:
    print(model.name)

# Чтение последней строки из файла CSV с лучшими моделями
best_model_df = pd.read_csv(f"{data_path}/best_models.csv")
best_model_info = best_model_df.iloc[-1]  # Берем последнюю строку

model_name = best_model_info['name']
model_uri = best_model_info['uri']

# Извлечение run_id из model_uri
run_id = model_uri.split('/')[1]

# Получение версии модели
print(f"Поиск модели с именем: {model_name} и run_id: {run_id}")
results = client.search_model_versions(f"name='{model_name}' and run_id='{run_id}'")
if not results:
    print(f"No model versions found for name: {model_name} and run_id: {run_id}")
else:
    version = results[0].version
    client.transition_model_version_stage(name=model_name, version=version, stage="Product
    print(f"Model {model_name} version {version} transitioned to Production.")
    model_version = f"models:/{model_name}/{version}"
    # Команда для запуска хостинга модели через mlflow serve
    host_command = f"mlflow models serve -m {model_version} --host 0.0.0.0 --port 5001"
    print(f"Запуск хостинга модели командой: {host_command}")



    # Запуск команды для хостинга модели
    return_code = subprocess.call(host_command, shell=True)
    print(f"Return code: {return_code}")
```

```python
time.sleep(5)

try:
    # Отправка POST-запроса для проверки
    response = requests.post(f'http://localhost/invocations',
                             json={
                                    "columns": ["Pclass", "Sex", "Age", "SibSp", "Parc
                                    "data": [
                                        [1, 0, 58.0, 0, 0, 26.55, 2],
                                        [2, 0, 23.0, 0, 0, 13.7917, 0],
                                        [3, 1, 23.0, 0, 0, 7.8542, 2]
                                    ]
                                    })
    print(f"Response from model: {response.json()}")
except requests.exceptions.RequestException as e:
    print(f"Error during request: {e}")
```

Инициализация и подключение:

- Устанавливается URL-адрес для MLflow (tracking_url), путь к данным (data_path), а также подключение к клиенту MlflowClient, что позволяет управлять моделями в MLflow.

Поиск зарегистрированных моделей:

- Выводит на экран имена всех зарегистрированных моделей, доступных в MLflow, используя search_registered_models().

Определение лучшей модели:

- Читает последний (последнюю строку) записанный результат из файла best_models.csv, где содержится информация о лучшей модели.
- Извлекает имя модели (model_name) и URI (model_uri), а также run_id, который используется для идентификации версии модели.

Поиск и перевод модели в "Production":

- Поиск версии модели с model_name и run_id.
- Если версия модели найдена, переводит её на "Production" с помощью transition_model_version_stage. Это обновляет стадию модели в MLflow.

Запуск сервера модели:

- Создаёт команду для запуска MLflow сервера на порту 5001 с использованием версии модели (model_version).
- Запускает команду через subprocess.call, что активирует сервер, который начнёт принимать запросы на предсказание.
- После запуска сервера делает паузу на 5 секунд, чтобы сервер успел инициализироваться.

Проверка работоспособности модели:

- Отправляет тестовый POST-запрос к запущенному серверу (по адресу /invocations), передавая несколько данных для проверки предсказания.
- Полученный ответ выводится на экран. Если возникает ошибка при выполнении запроса, выводит сообщение об ошибке.

Так должен был работать скрипт в теории, и да мне удалось переместить модель в promote состояние, но возникли проблемы с хостингом.

Сначала я подумал, что всё прошло отлично и команда для хостинга исполнилась так как пришла со статусом 0, и я подумал, что мне удалось замостить модель, но пост запрос не сработал и при проверке портов, можно было понять, что там пусто, я проверял порты в контейнере, для этого даже пришлось команду поставить в контейнер, но порт был пуст.

```
lient
    import boto3
ModuleNotFoundError: No module named 'boto3'
PS C:\Users\darti> docker exec -it ca7b68006b3d /bin/sh
#
# curl http://localhost:5003/invocations
/bin/sh: 2: curl: not found
# wget -qO- http://localhost:5003/invocations
/bin/sh: 3: wget: not found
# apt-get update && apt-get install -y curl
Get:1 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:2 http://deb.debian.org/debian-security bullseye-security InRelease [27.2 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [44.1 kB]
Get:4 http://deb.debian.org/debian bullseye/main amd64 Packages [8066 kB]
Get:5 http://deb.debian.org/debian-security bullseye-security/main amd64 Packages [304 kB]
Get:6 http://deb.debian.org/debian bullseye-updates/main amd64 Packages [18.8 kB]
Fetched 8575 kB in 3s (2564 kB/s)
```

Но всё же запускать эту команду в airflow наверно не совсем корректно.

После я решил попробовать ввести команду напрямую и зашел в контейнер mlflow, но и тут ничего не получилось, контейнер mlflow, просто не воспринимал команду, которую я использую говорил, что mlflow вообще нет тут.

В конечном итоге я пытался перестроить контейнер и добавить новую службу для хостинга модели.

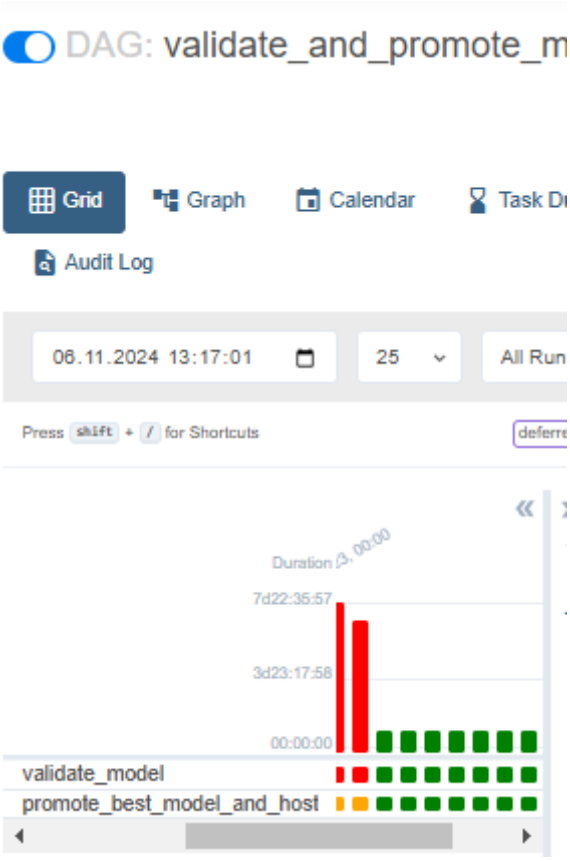
```yaml
model_server:
  restart: unless-stopped
  image: python:3.8  # Укажите базовый образ с Python и MLflow
  container_name: model_server
  depends_on:
    - web
  ports:
    - "5002:5000"  # Порт для доступа к модели
  environment:
    - MLFLOW_S3_ENDPOINT_URL=http://minio:9000
    - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
    - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
  command: >
    mlflow models serve -m runs:/6c5de147109f491fb88538d1ddbcfe06/sklearn.ensemble/RandomForestClassifier
    --host 0.0.0.0 --port 5000
```

В общем я не уверен в том сделал ли я всё правильно.

Это тоже не совсем сработало, я так и не смог нормально захостить модель.

Решил посмотреть примеры в сети, и почему-то все люди ставили либо себе на пк либо на удалённый сервер, в итоге мне не удалось это реализовать для контейнера.

4. Выполнение dag



Так что мне пришлось довольствоваться переводом модели в promote

Но тем не мение лучшая модель оказалась в promote и основные захадания были выполнены.

Registered Models

Filter registered models...

Name	Latest version	Staging	Production	Created by	Last modified	Tags
DecisionTreeClassifier	Version 3	—	—		2024-10-20 23:...	—
KNeighborsClassifier	Version 3	—	—		2024-10-20 23:...	—
LogisticRegression	Version 2	—	—		2024-10-20 23:...	—
Perceptron	Version 1	—	—		2024-10-20 23:...	—
RandomForestClassifier	Version 3	—	Version 3		2024-10-20 23:...	—
SGDClassifier	Version 3	—	—		2024-10-20 23:...	—
SVC	Version 3	—	—		2024-10-20 23:...	—

ЗАКЛЮЧЕНИЕ

В результате выполнения ЛР, мы создали пайплайн который принимает значения для предобработки данных и обучения моделей с заданными аргументами логирует все данные в mlflow. После мы создали пайплайн, который реализует валидацию моделей и выбирает лушую модель и переводит её в promote, к сожелению хостинг не совсем получился, я понял что его можно реализовать через команду, но выполнить её в контейнере не получилось, возможно на настоящих серверах, с отдельной машиной с mlflow это бы получилось.