

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 и кибернетики  
Факультет информатики  
Кафедра технической кибернетики

**Отчет по лабораторной работе №2**

Дисциплина: «Инженерия данных»

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

Выполнили:

Мелешенко И.С.

Коршиков В.И.

Группа: 6233-010402D

Самара 2023

## Содержание

Часть 1. Подготовка к выполнению лабораторной работы.....	3
Часть 2. Построение пайплайна, который обучает любой классификатор из sklearn по заданному набору параметров .....	4
Шаг 1. Определение моделей и датасетов для работы.....	4
Шаг 2. Разработка DAG-а.....	5
Шаг 3. Разработка вспомогательных модулей. ....	5
Шаг 4. Обучение моделей. ....	7
Шаг 5. Запуск эксперимента .....	9
Часть 2. Построение пайплайна, который выбирает лучшую модель из обученных и производит её хостинг .....	11
Шаг 1. Разработка DAG-а.....	11
Шаг 2. Разработка кода валидации моделей.....	11
Шаг 3. Запуск DAG-а валидации моделей. ....	12
Шаг 4. Проверка отработки DAG-а.....	12
Заключение.....	14

## Часть 1. Подготовка к выполнению лабораторной работы.

В данной лабораторной работе нам необходимо реализовать обучение классификаторов из пакета `sklearn`. Для работы нам понадобятся `docker`-контейнеры с образами `Airflow` и `Mlfow`, а остальные отключить за ненадобностью.

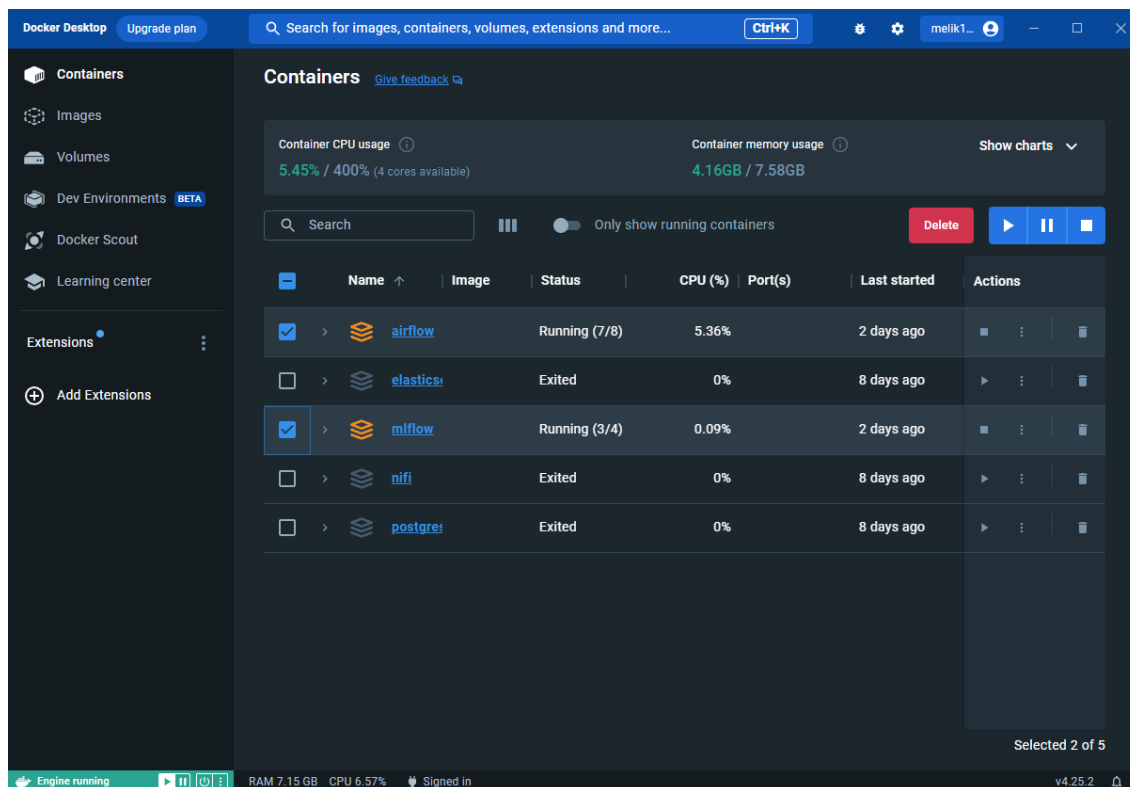


Рисунок 1 – Необходимые контейнеры для работы.

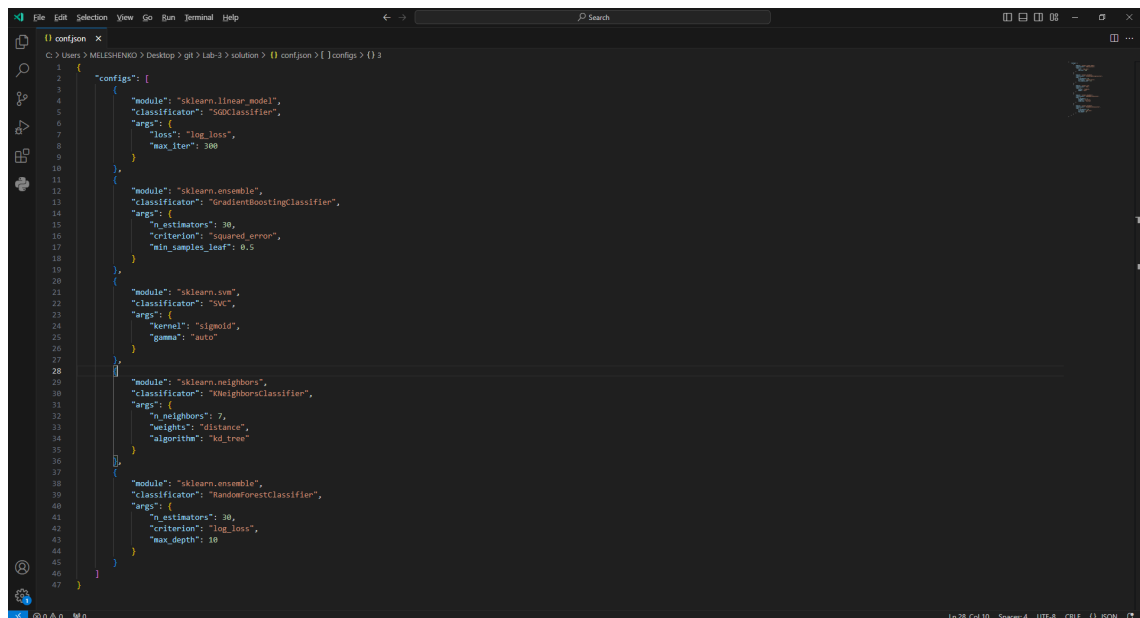
Часть 2. Построение пайплайна, который обучает любой классификатор из sklearn по заданному набору параметров

Шаг 1. Определение моделей и датасетов для работы.

Перед тем обучать модели необходимо выбрать их и сформировать конфигурационный файл, с которым будет работать наш DAG. Для выбора моделей воспользуемся официальной документацией по ссылке <https://scikit-learn.org/stable/modules/classes.html>. В процессе изучения данной ссылки мой выбор пал на:

- sklearn.linear\_model.SGDClassifier;
- sklearn.ensemble.GradientBoostingClassifier;
- sklearn.svm.SVC;
- sklearn.neighbors.KNeighborsClassifier;
- sklearn.ensemble.RandomForestClassifier;

С этими моделями мы будем работать. Для начала работы сформируем конфигурационный файл в формате json:



```
1 {
2   "configs": [
3     {
4       "module": "sklearn.linear_model",
5       "classifier": "SGDClassifier",
6       "args": {
7         "loss": "log_loss",
8         "max_iter": 300
9       }
10    },
11    {
12       "module": "sklearn.ensemble",
13       "classifier": "GradientBoostingClassifier",
14       "args": {
15         "n_estimators": 30,
16         "criterion": "squared_error",
17         "min_sample_leaf": 0.5
18       }
19    },
20    {
21       "module": "sklearn.svm",
22       "classifier": "SVC",
23       "args": {
24         "kernel": "sigmoid",
25         "gamma": "auto"
26       }
27    },
28    {
29       "module": "sklearn.neighbors",
30       "classifier": "KNeighborsClassifier",
31       "args": {
32         "n_neighbors": 7,
33         "weights": "distance",
34         "algorithm": "kd_tree"
35       }
36    },
37    {
38       "module": "sklearn.ensemble",
39       "classifier": "RandomForestClassifier",
40       "args": {
41         "n_estimators": 30,
42         "criterion": "log_loss",
43         "max_depth": 10
44       }
45    }
46  ]
47 }
```

Рисунок 2 – Формирование конфигурационного файла с моделями

В качестве источника данных для обучения и валидации моделей был выбран стандартный датасет вин из библиотеки sklearn: load\_wines. Для использования его в процессе работы DAG-а он будет разделен на тестовую, тренировочную и валидационную выборки.

## Шаг 2. Разработка DAG-а.

DAG реализующий пайплайн обучения классификаторов, состоит из 4 task-ов:

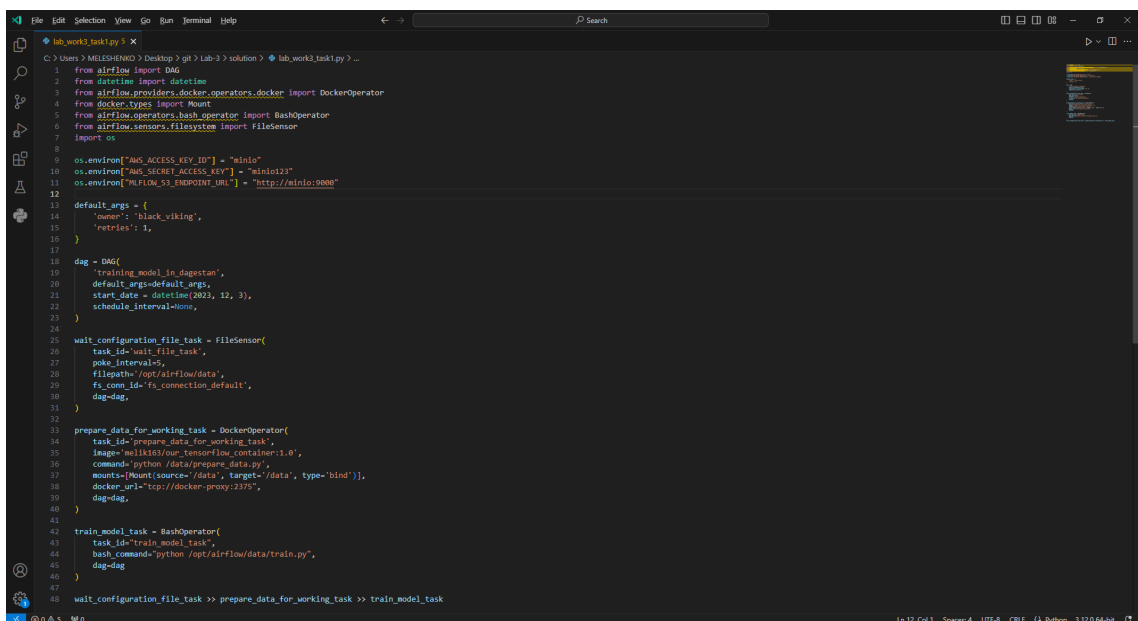
`wait_configuration_file_task` – осуществляет мониторинг папки, в которой должен появиться конфигурационный файл, с описанием моделей.

`prepare_data_for_working_task` – осуществляет подготовку данных, которые будут использоваться в процессе обучения и валидации моделей.

`train_model_task` – осуществляет процесс обучения моделей и их логирование.

О двух последних поговорим подробнее чуть ниже.

Код DAG-а первой части лабораторной работы приведен ниже, а также в репозитории с решением.



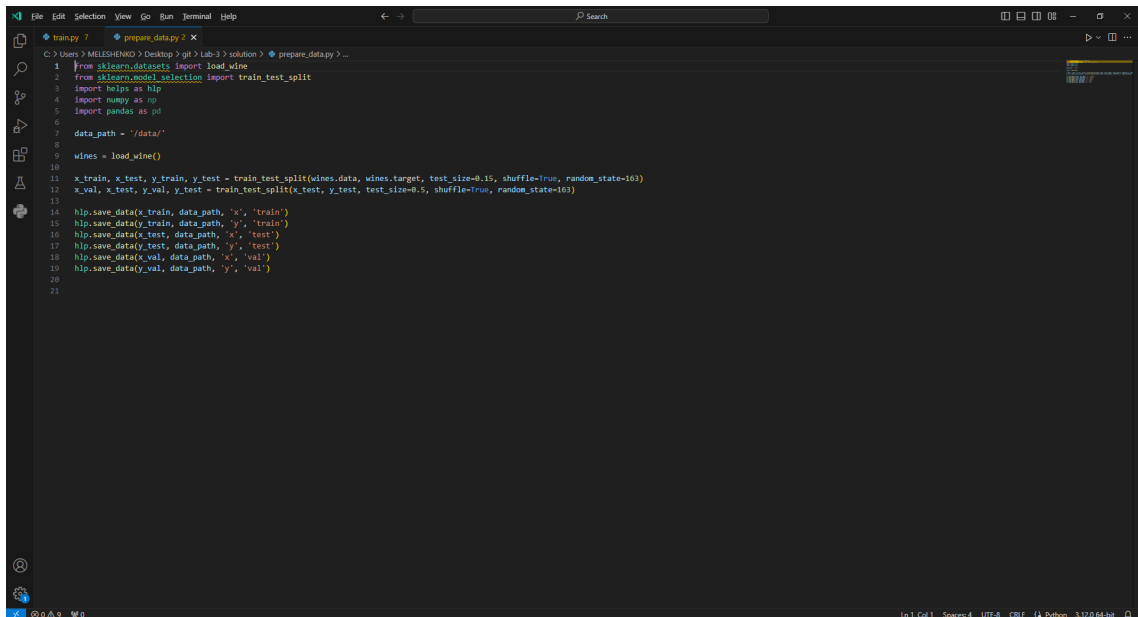
```
1 from airflow import DAG
2 from datetime import datetime
3 from airflow.providers.docker_operators.docker import DockerOperator
4 from docker.types import Mount
5 from airflow.operators.bash_operator import BashOperator
6 from airflow.sensors.filesystem import FileSensor
7 import os
8
9 os.environ["AWS_ACCESS_KEY_ID"] = "minio"
10 os.environ["AWS_SECRET_ACCESS_KEY"] = "minio123"
11 os.environ["MLFLOW_ENDPOINT_URL"] = "http://minio:9080"
12
13 default_args = {
14     'owner': 'black_viking',
15     'retries': 1,
16 }
17
18 dag = DAG(
19     'training_model_in_dagtest',
20     default_args=default_args,
21     start_date = datetime(2023, 12, 3),
22     schedule_interval=None,
23 )
24
25 wait_configuration_file_task = FileSensor(
26     task_id='wait_file_task',
27     poke_interval=5,
28     filepath='/opt/airflow/data',
29     fs_conn_id='fs_connection_default',
30     dag=dag,
31 )
32
33 prepare_data_for_working_task = DockerOperator(
34     task_id='prepare_data_for_working_task',
35     image='ml1k103/our_tensorflow_container:1.0',
36     command='python /data/prepare_data.py',
37     mounts=[Mount(source='/data', target='/data', type='bind')],
38     docker_url='tcp://docker-proxy:2375',
39     dag=dag,
40 )
41
42 train_model_task = BashOperator(
43     task_id='train_model_task',
44     bash_command='python /opt/airflow/data/train.py',
45     dag=dag,
46 )
47
48 wait_configuration_file_task >> prepare_data_for_working_task >> train_model_task
```

Рисунок 3 – Код DAG-а.

## Шаг 3. Разработка вспомогательных модулей.

Task-и `prepare_data_for_working_task` и `train_model_task` в своей работе используют подготовленные нами скрипты, рассмотрим их подробнее и начнем с task-а, который осуществляет подготовку данных `prepare_data_for_working_task`.

В данной скрипте мы определяем «корневую» папку в которой будем работать. Далее импортируем наш датасет. Это стандартный датасет вин из библиотеки `sklearn`. После того как подгрузили датасет, произведем его разделение на выборки: обучающую, тестовую и валидационную. Подготовленные данные сохраняем в файлы, для последующего использования. Код описанного скрипта приведен на рисунке ниже, а также в репозитории с решением.

A screenshot of a code editor window with a dark theme. The editor shows a Python script for data preparation. The script imports `load_wine` from `sklearn.datasets`, `train_test_split` from `sklearn.model_selection`, and `helps` from `helpers`. It also imports `numpy` as `np` and `pandas` as `pd`. The script defines a `data_path` variable pointing to `../data/`. It then loads the `wine` dataset and splits it into training, testing, and validation sets using `train_test_split`. Finally, it uses the `helps.save_data` function to save the data into files named `x_train`, `y_train`, `x_val`, `y_val`, `x_test`, and `y_test`.

```
1 from sklearn.datasets import load_wine
2 from sklearn.model_selection import train_test_split
3 import helps as hlp
4 import numpy as np
5 import pandas as pd
6
7 data_path = '../data/'
8
9 wines = load_wine()
10
11 x_train, x_test, y_train, y_test = train_test_split(wines.data, wines.target, test_size=0.15, shuffle=True, random_state=163)
12 x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=0.5, shuffle=True, random_state=103)
13
14 hlp.save_data(x_train, data_path, 'x', 'train')
15 hlp.save_data(y_train, data_path, 'y', 'train')
16 hlp.save_data(x_test, data_path, 'x', 'test')
17 hlp.save_data(y_test, data_path, 'y', 'test')
18 hlp.save_data(x_val, data_path, 'x', 'val')
19 hlp.save_data(y_val, data_path, 'y', 'val')
```

Рисунок 4 – Код, осуществляющий подготовку данных

В процессе подготовки данных был использован метод `save_data` из модуля `helpers`. Это небольшой вспомогательный скрипт, в который вынесены операции сохранения и получения датасетов из файлов. Рассмотрим их подробнее.

Метод `save_data()` необходим для сохранения датасетов в файл. На вход метод принимает 4 параметра:

- 1) `dataset` - сам датасет, который мы будем сохранять;
- 2) `data_path` – пусть куда будет сохранен файл;
- 3) `types` – тип датасета, который мы передаем (x или y);
- 4) `labels` – метка выборки (обучающая, тестовая, валидационная).

Метод `get_data()` необходим для получения датасетов из файлов. На вход метод принимает 2 параметра:

- 1) `data_path` – путь, откуда необходимо прочитать файл;
- 2) `labels` – метка выборки (обучающая, тестовая, валидационная).

На выходе возвращается два датасета: `array_data` и `array_target`

Реализация данного модуля представлена на рисунке ниже и в репозитории решения лабораторной работы.

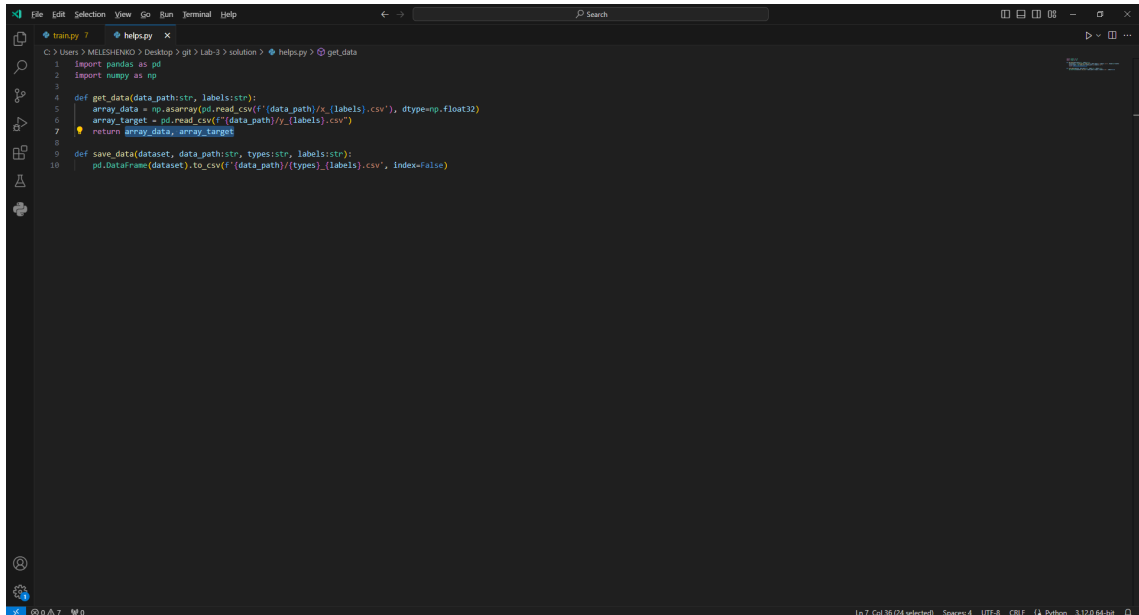


Рисунок 5 – Код вспомогательного модуля `helps`

#### Шаг 4. Обучение моделей.

Обучение моделей сосредоточено в `task-e - train_model_task`. Рассмотрим его подробнее.

В первую очередь получаем конфигурацию всех моделей, из полученного конфигурационного файла `conf.json`. Поскольку каждый запуск обучения моделей — это отдельный эксперимент, то для каждого необходим уникальный `experiment_id`. Для этого была создана функция генерации этого самого `experiment_id`: `generate_experiment_id()`.

На вход она принимает название файла, в который будет сохранен наш `experiment_id`.

В процессе работы, функция генерирует уникальный `experiment_id`, после чего пишет в указанный файл, а также возвращает в качестве выходного значения для продолжения работы. Код функции приведен ниже.

```
def generate_experiment_id(name_file:str):
    sources_string = '1234567890AaBbCcDdEeFfGgHhIiJjKkLlMm1234567890NnOoPpQqRrSsTtUuVvWwXxYyZz1234567890'
    list_str = []
    for i in range(19):
        list_str.append(sources_string[rnd.randint(0, len(sources_string)-1)])
    result_str = ''.join(list_str)
    f = open(f'{data_path}/{name_file}', 'w')
    f.write(result_str)
    f.close()
    return result_str
```

Рисунок 6 – Код генерации experiment\_id

Также перед началом обучения, мы создали функцию, которая будет осуществлять логирование метрик моделей в процессе обучения: logirovanie().

На вход данная функция принимает

- 1) cuerrnt\_configs – текущая конфигурация модели
- 2) y\_test\_dataset – датасет с истинными значениями.
- 3) current\_prediction – датасет с предсказанными значениями.

В процессе обучения производим логирование четырех метрик:

- F1
- Accuracy
- Precision
- Recall

Код приведен на рисунке ниже.

```
def logirovanie(cuerrnt_configs, y_test_dataset, current_prediction):
    mlflow.log_params(cuerrnt_configs)
    mlflow.log_metrics({'f1': f1_score(y_test_dataset, current_prediction, average='weighted')})
    mlflow.log_metrics({'acc': accuracy_score(y_test_dataset, current_prediction)})
    mlflow.log_metrics({'precision': precision_score(y_test_dataset, current_prediction, average='weighted')})
    mlflow.log_metrics({'recall': recall_score(y_test_dataset, current_prediction, average='weighted')})
```

Рисунок 7 – Код логирования метрик

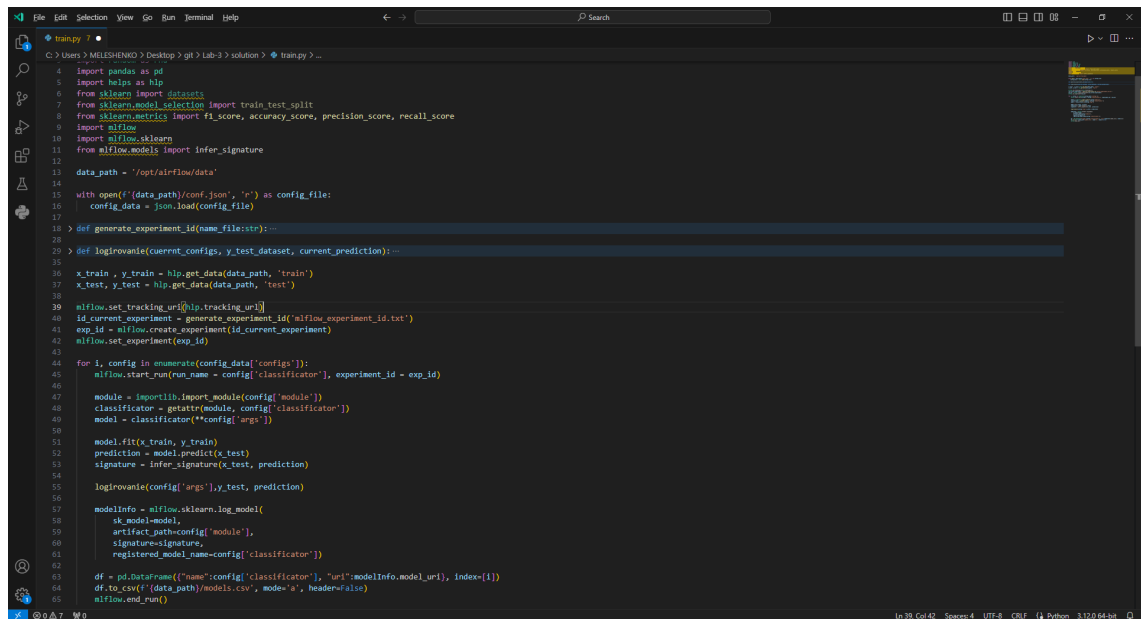
После всех приготовлений запускаем эксперимент по обучению моделей. При помощи функции get\_data() из самописного модуля helps получаем данные для обучения моделей. Устанавливаем подключение к mlflow. Генерируем experiment\_id при помощи функции generate\_experiment\_id() и подключаемся к эксперименту с только что созданным experiment\_id.

После чего в цикле для каждой модели проделываем следующие операции:



- 1) Получаем конфигурацию модели
- 2) Проводим процесс обучения
- 3) Логируем метрики
- 4) Логируем модель
- 5) Лог модели пишем в файл для дальнейшего использования во второй части лабораторной работы.

Код обучения моделей представлен на рисунке ниже, а также полная версия кода размещена в репозитории с решением лабораторной работы.



```
4 import pandas as pd
5 import hashlib as hlp
6 from sklearn import datasets
7 from sklearn.model_selection import train_test_split
8 from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score
9 import mlflow
10 from mlflow.sklearn import infer_signature
11
12 data_path = '/opt/airflow/data'
13
14 with open(f'{data_path}/conf.json', 'r') as config_file:
15     config_data = json.load(config_file)
16
17 > def generate_experiment_id(name_file:str):...
18
19 > def logitrovanie(cuerent_configs, y_test_dataset, current_prediction):...
20
21 x_train, y_train = hlp.get_data(data_path, 'train')
22 x_test, y_test = hlp.get_data(data_path, 'test')
23
24 mlflow.set_tracking_uri(hlp.tracking_uri)
25 if current_experiment = generate_experiment_id('mlflow_experiment_id.txt')
26 exp_id = mlflow.create_experiment(id_current_experiment)
27 mlflow.set_experiment(exp_id)
28
29 for i, config in enumerate(config_data['configs']):
30     mlflow.start_run(run_name = config['classifier'], experiment_id = exp_id)
31
32     module = importlib.import_module(config['module'])
33     classifier = getattr(module, config['classifier'])
34     model = classifier(**config['args'])
35
36     model.fit(x_train, y_train)
37     prediction = model.predict(x_test)
38     signature = infer_signature(x_test, prediction)
39     logitrovanie(config['args'], y_test, prediction)
40
41     modelInfo = mlflow.sklearn.log_model(
42         sk_model=model,
43         artifact_path=config['module'],
44         signature=signature,
45         registered_model_name=config['classifier'])
46
47 df = pd.DataFrame({'name':config['classifier'], 'url':modelInfo.model_url, index=1})
48 df.to_csv(f'{data_path}/models.csv', mode='a', header=False)
49 mlflow.end_run()
```

Рисунок 8 – Код обучения моделей

## Шаг 5. Запуск эксперимента

Теперь, когда весь код подготовлен, можно перейти в Airflow, и запустить наш DAG по обучению моделей.

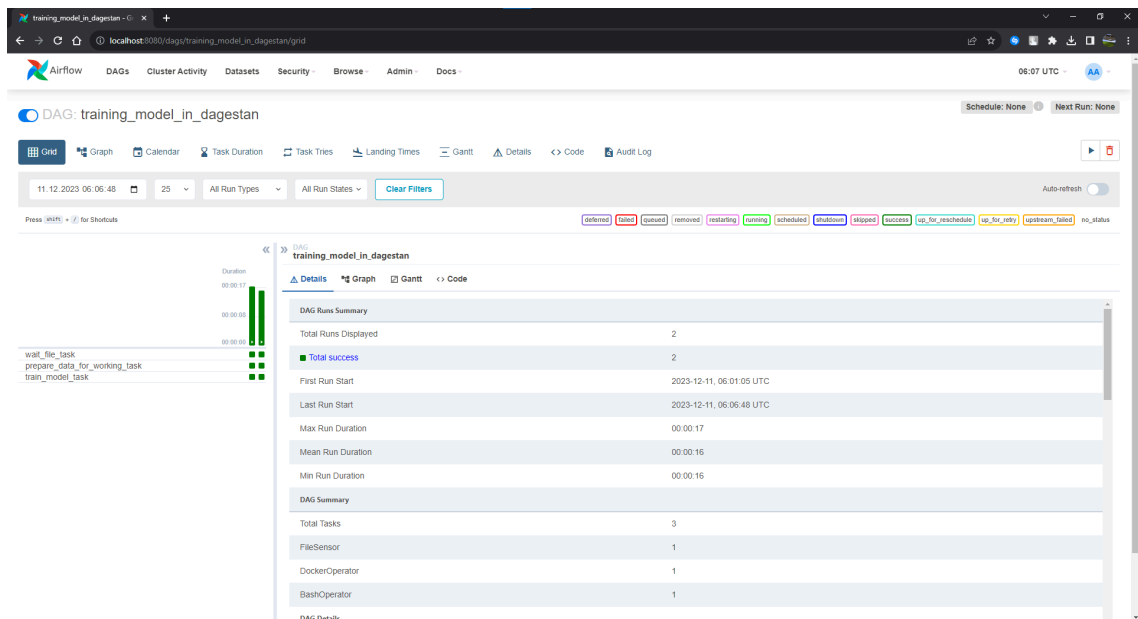


Рисунок 9 – Запуск DAG-а по обучению моделей.

После отработки DAG-а перейдем в Mlflow по адресу <http://localhost:5001>, и посмотрим на результаты обучения, которые изображены на рисунке ниже.

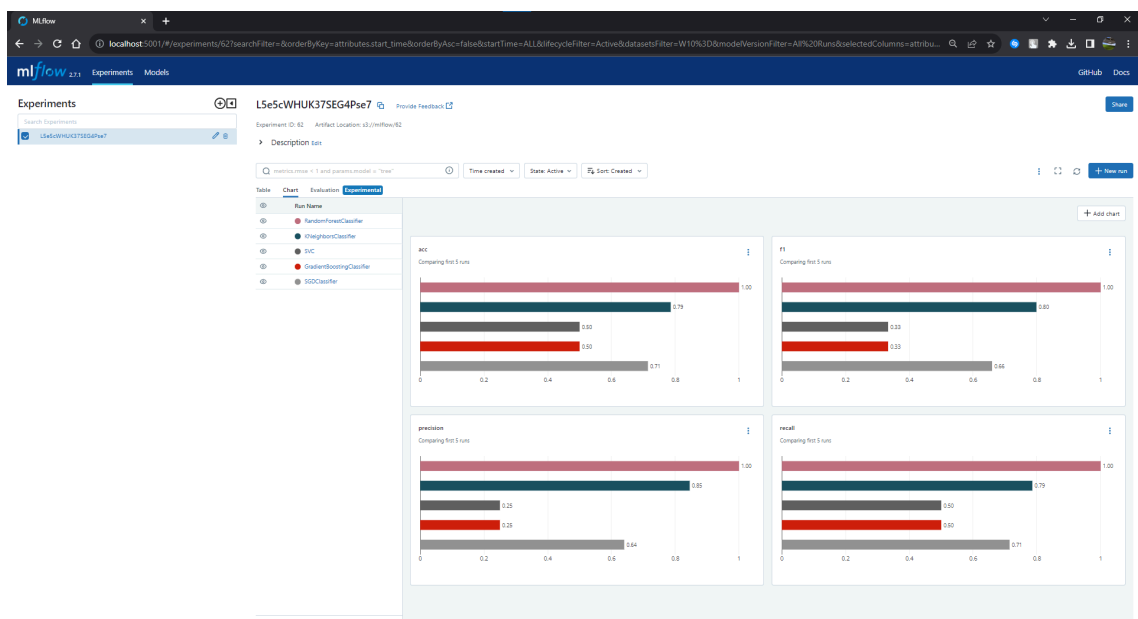


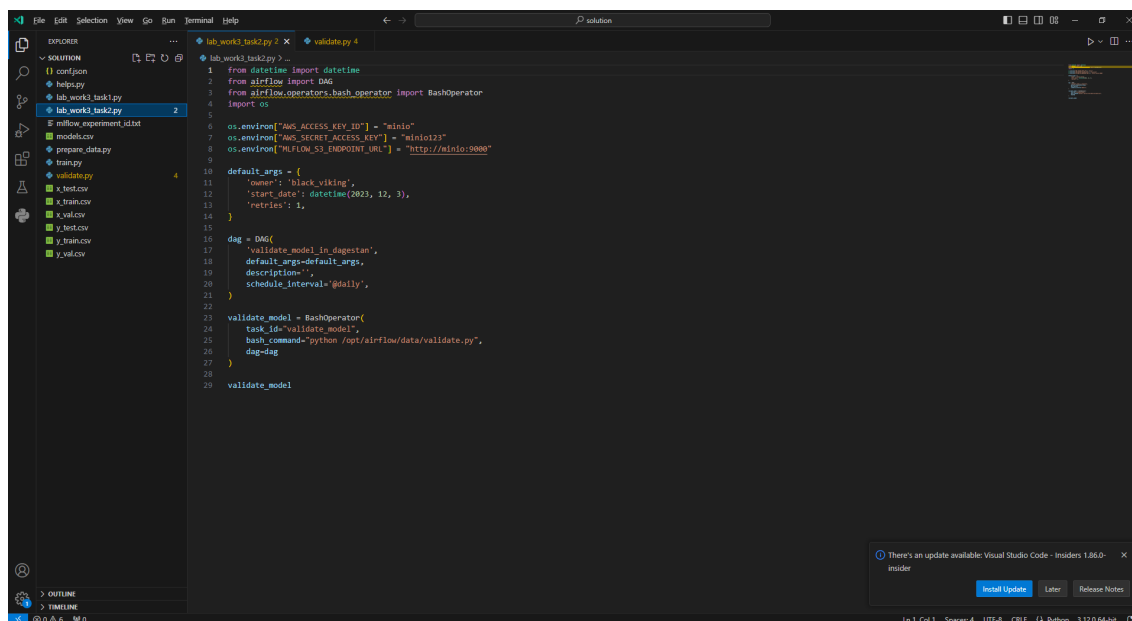
Рисунок 10 – Результат обучения моделей

В результате мы получили набор обученных моделей, которые будут провалидированы на следующем этапе лабораторной работы.

Часть 2. Построение пайплайна, который выбирает лучшую модель из обученных и производит её хостинг

### Шаг 1. Разработка DAG-а.

DAG валидации моделей состоит из одного task-а, который осуществляет процесс валидации моделей. Код DAG-а представлен на рисунке ниже, а также в репозитории с решением лабораторной работы.



```
1 from datetime import datetime
2 from airflow import DAG
3 from airflow.operators.bash_operator import BashOperator
4 import os
5
6 os.environ["AWS_ACCESS_KEY_ID"] = "minio"
7 os.environ["AWS_SECRET_ACCESS_KEY"] = "minio123"
8 os.environ["MLFLOW_S3_ENDPOINT_URL"] = "http://minio:9000"
9
10 default_args = {
11     'owner': 'black_viking',
12     'start_date': datetime(2023, 12, 3),
13     'retries': 1,
14 }
15
16 dag = DAG(
17     'validate_model_in_dagstan',
18     default_args=default_args,
19     description='',
20     schedule_interval="@daily",
21 )
22
23 validate_model = BashOperator(
24     task_id='validate_model',
25     bash_command='python /opt/airflow/data/validate.py',
26     dag=dag
27 )
28
29 validate_model
```

Рисунок 11 – DAG валидации моделей

### Шаг 2. Разработка кода валидации моделей.

После того как подготовили DAG, перейдем к разработке кода, осуществляющего процесс валидации модели. В начале получаем `experiment_id`, который был сохранен в файл, на этапе обучения моделей, а после подключаемся к уже существующему эксперименту.

После подключения подгружаем данные для валидации моделей, которые были сохранены на этапе подготовки данных. Используя список сохраненных моделей на этапе обучения, определяем лучшую, на основании показателя “Accuracy” и выводим в ее в состояние “Production”. В итоге среди всех моделей хотя бы одна модель должна быть в состоянии “Production”.

Код валидации моделей представлен на рисунке ниже, а также в репозитории решения лабораторной работы.

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.metrics import accuracy_score
4 import h5py as h5p
5
6 import mlflow
7 import mlflow.sklearn
8 from mlflow import MlflowClient
9
10 tracking_url = 'http://mlflow-server:5000'
11 data_path = '/opt/airflow/data'
12
13 with open(f'{data_path}/mlflow_experiment_id.txt', 'r') as f:
14     id_current_experiment = f.read()
15
16 mlflow.set_tracking_uri(tracking_url)
17 mlflow.set_experiment(id_current_experiment)
18
19 x_val, y_val = h5p.get_data(data_path, 'val')
20 list_models_for_validate = []
21
22 with mlflow.start_run(run_name="Production model") as start_run:
23     models_file = pd.read_csv(f'{data_path}/models.csv', header=None)
24     for model_info in models_file.iterrows():
25         name = model_info[1][1]
26         url = model_info[1][2]
27         list_models_for_validate[name + " " + url] = mlflow.sklearn.load_model(url)
28
29     current_results = {}
30     for name, model in list_models_for_validate.items():
31         prediction = model.predict(x_val)
32         current_results[name] = accuracy_score(y_val, prediction)
33
34     best_model_in_list_validate_model = max(current_results, key=current_results.get)
35
36 client = MlflowClient()
37 version = client.search_model_versions(f"name='{best_model_in_list_validate_model.split(' ')[0]}' and run_id='{best_model_in_list_validate_model.split(' ')[1].split('/')[1]}'")[0].version
38 client.transition_model_version_stage(name=best_model_in_list_validate_model.split(' ')[0], version=version, stage="Production")
39
```

Рисунок 12 – Код валидации моделей

### Шаг 3. Запуск DAG-а валидации моделей.

После окончания всех приготовлений перейдем в Airflow, для запуска DAG-а, который осуществит валидацию моделей. На рисунке ниже мы можем наблюдать успешную отработку кода.

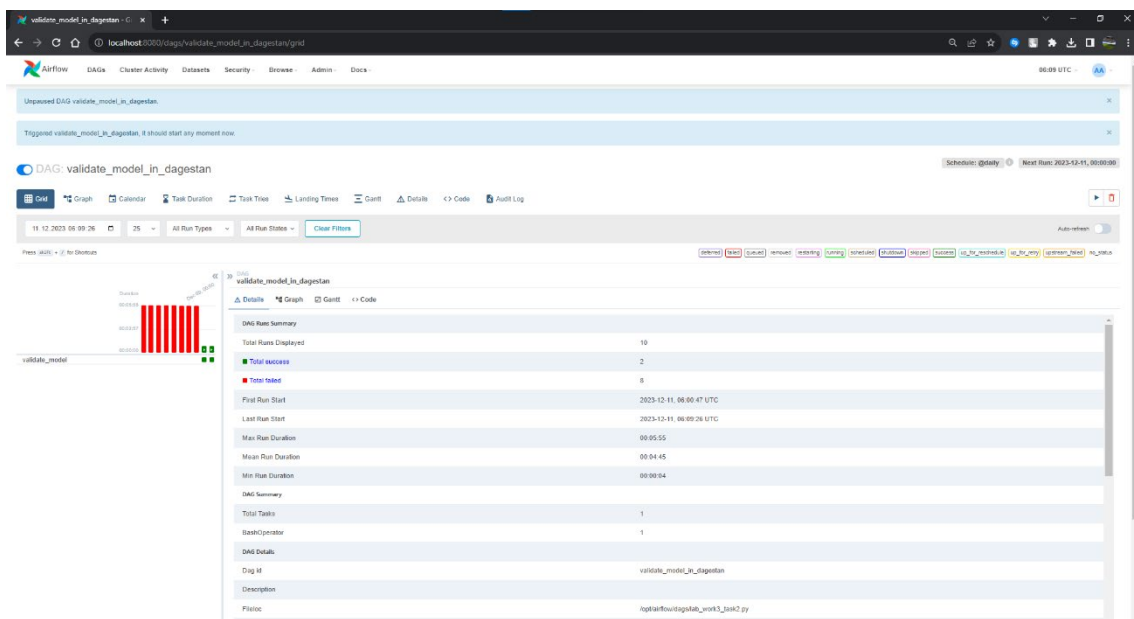
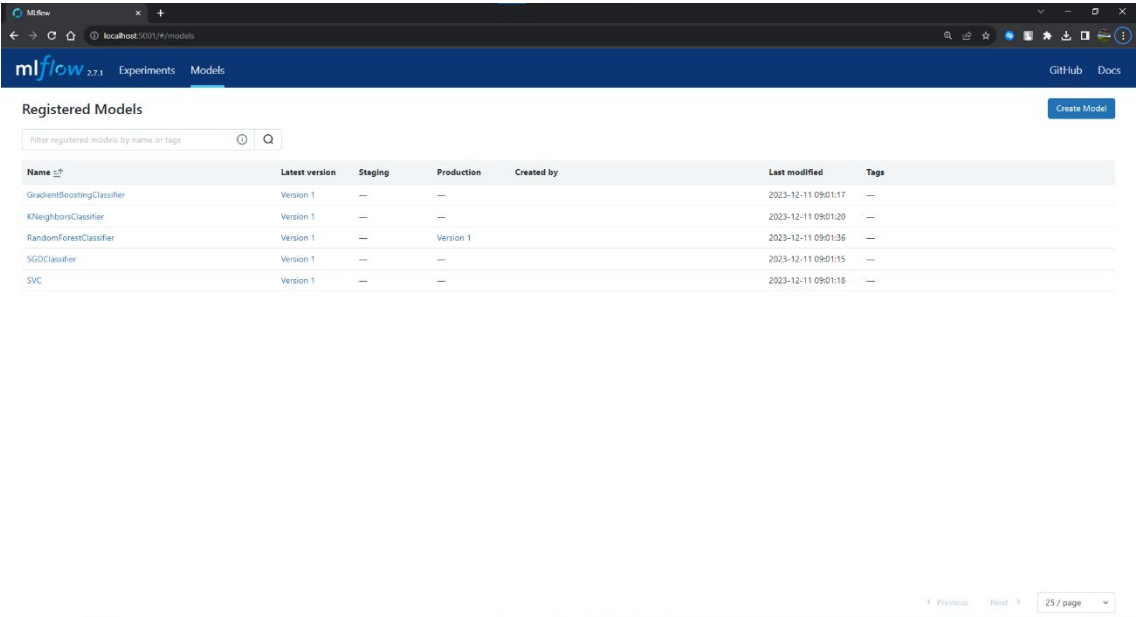


Рисунок 13 – Запуск DAG-а валидации моделей

### Шаг 4. Проверка отработки DAG-а.

Для того чтобы убедиться, в корректности отработки нашего DAG-а, перейдем в Mlflow, и проверим какие статусы имеют зарегистрированные

модели. На рисунке ниже наблюдаем, что модель RandomForestClassifier перешла в состояние “Production”, что говорит об успешной отработке DAG-а.



The screenshot shows the mlflow web interface at localhost:5001/#/models. The 'Registered Models' section displays a table with the following data:

Name	Latest version	Staging	Production	Created by	Last modified	Tags
GradientBoostingClassifier	Version 1	—	—		2023-12-11 09:01:17	—
KNeighborsClassifier	Version 1	—	—		2023-12-11 09:01:20	—
RandomForestClassifier	Version 1	—	Version 1		2023-12-11 09:01:36	—
SGDClassifier	Version 1	—	—		2023-12-11 09:01:15	—
SVC	Version 1	—	—		2023-12-11 09:01:18	—

Рисунок 13 – Результат выполнения, DAG-а валидации моделей

## Заключение

В результате выполнения лабораторной работы получены навыки работы с логированием моделей и выводом моделей в “Production”.