

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 №3

Дисциплина: «Инженерия данных»

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

Выполнил:

Иванов И.А.

Группа: 6231-010402D

Самара 2024

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.

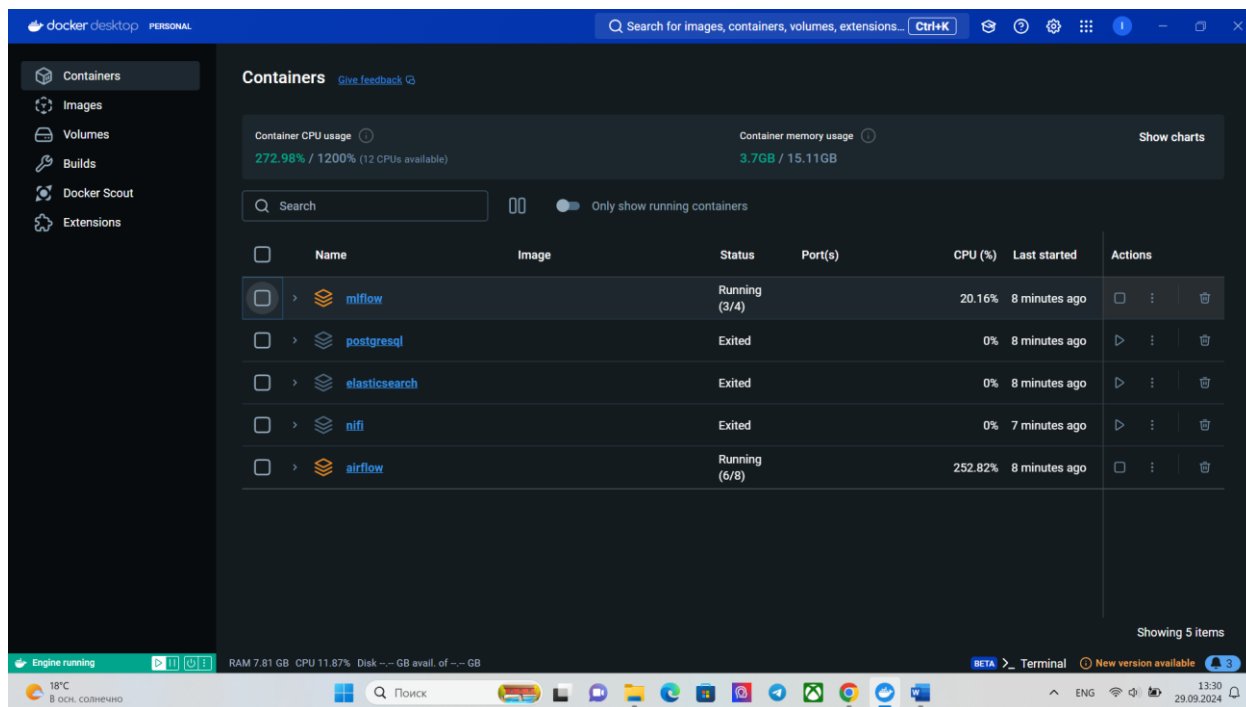


Рисунок 1 – Необходимые контейнеры для работы.

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:

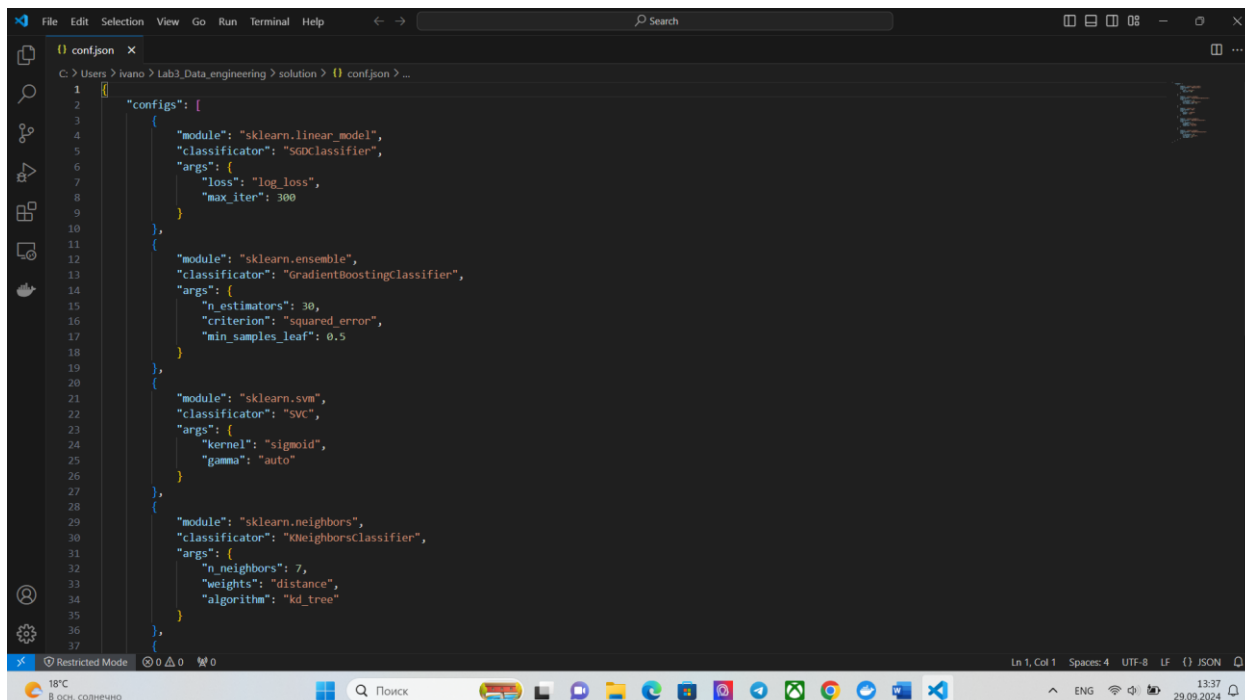


Рисунок 2 – Формирование конфигурационного файла с моделями

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.

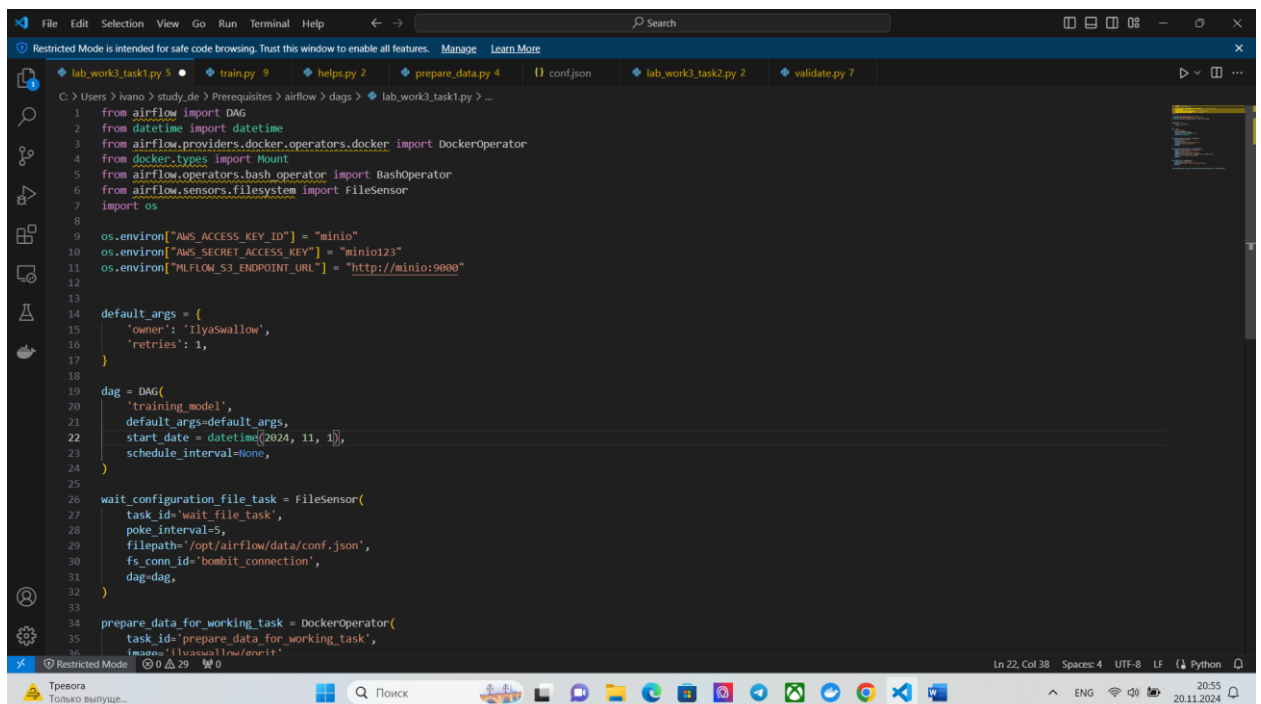
The image shows a screenshot of a code editor window with a dark theme. The editor displays Python code for an Airflow DAG. The code includes imports for Airflow DAG, datetime, DockerOperator, Mount, BashOperator, and FileSensor. It sets environment variables for AWS credentials and the MinIO endpoint. A default_args dictionary is defined with owner, retries, and start_date. The DAG is named 'training_model' and is configured with a start date of 2024-11-01. The first task, 'wait_configuration_file_task', is a FileSensor that monitors for a file named 'bombit_connection' in the directory '/opt/airflow/data/conf.json'. The second task, 'prepare_data_for_working_task', is a DockerOperator that runs a script named 'ilvacuallow/arpit'.

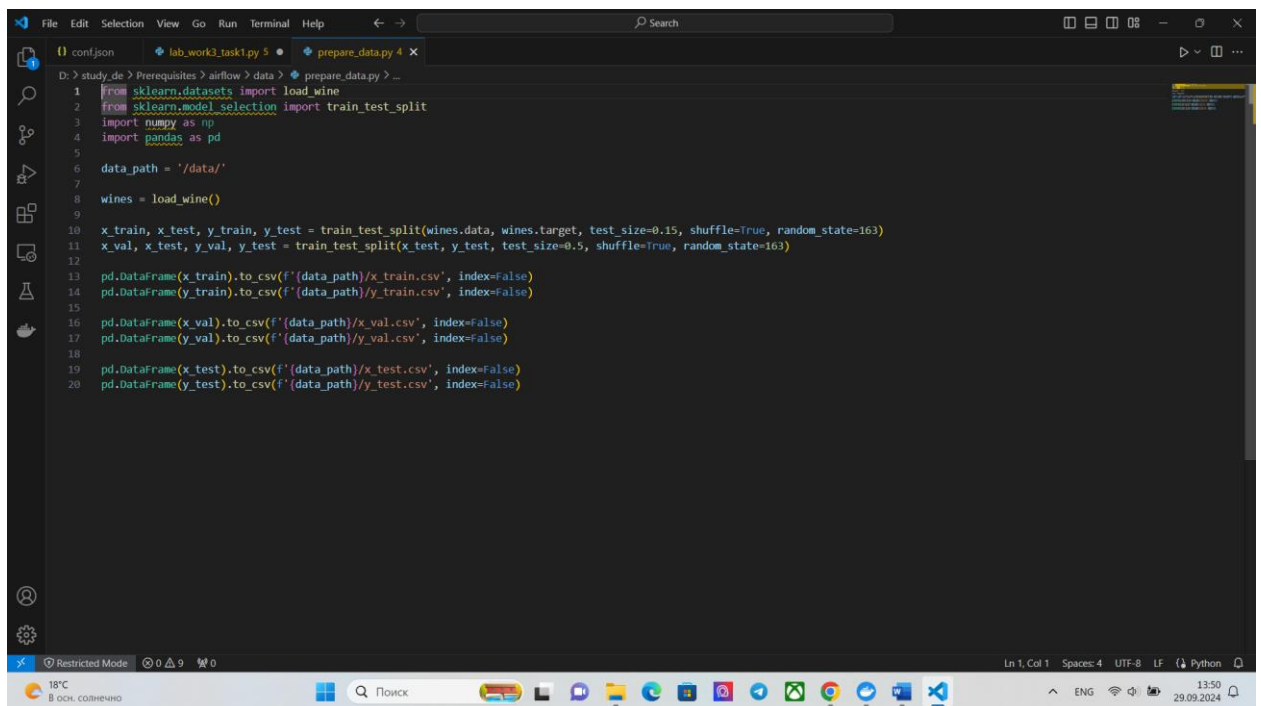
Рисунок 3 – Код DAG-а.

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-a, 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.

The image shows a screenshot of a code editor window with a dark theme. The editor has a menu bar at the top with options like File, Edit, Selection, View, Go, Run, Terminal, and Help. Below the menu bar is a search bar. The main area of the editor displays a Python script. The script starts with imports from sklearn.datasets, sklearn.model_selection, numpy, and pandas. It then defines a data_path variable and loads the 'wines' dataset. The dataset is split into training and testing sets using train_test_split. Finally, the data is saved to CSV files. The status bar at the bottom shows 'Ln 1, Col 1', 'Spaces: 4', 'UTF-8', 'LF', and 'Python'. The system tray at the very bottom shows the temperature as 18°C and the date as 29.09.2024.

```
1 from sklearn.datasets import load_wine
2 from sklearn.model_selection import train_test_split
3 import numpy as np
4 import pandas as pd
5
6 data_path = '/data/'
7
8 wines = load_wine()
9
10 x_train, x_test, y_train, y_test = train_test_split(wines.data, wines.target, test_size=0.15, shuffle=True, random_state=163)
11 x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=0.5, shuffle=True, random_state=163)
12
13 pd.DataFrame(x_train).to_csv(f'{data_path}/x_train.csv', index=False)
14 pd.DataFrame(y_train).to_csv(f'{data_path}/y_train.csv', index=False)
15
16 pd.DataFrame(x_val).to_csv(f'{data_path}/x_val.csv', index=False)
17 pd.DataFrame(y_val).to_csv(f'{data_path}/y_val.csv', index=False)
18
19 pd.DataFrame(x_test).to_csv(f'{data_path}/x_test.csv', index=False)
20 pd.DataFrame(y_test).to_csv(f'{data_path}/y_test.csv', index=False)
```

Рисунок 4 – Код, осуществляющий подготовку данных

В процессе подготовки данных был использован метод `save_data` из модуля `helps`. 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);

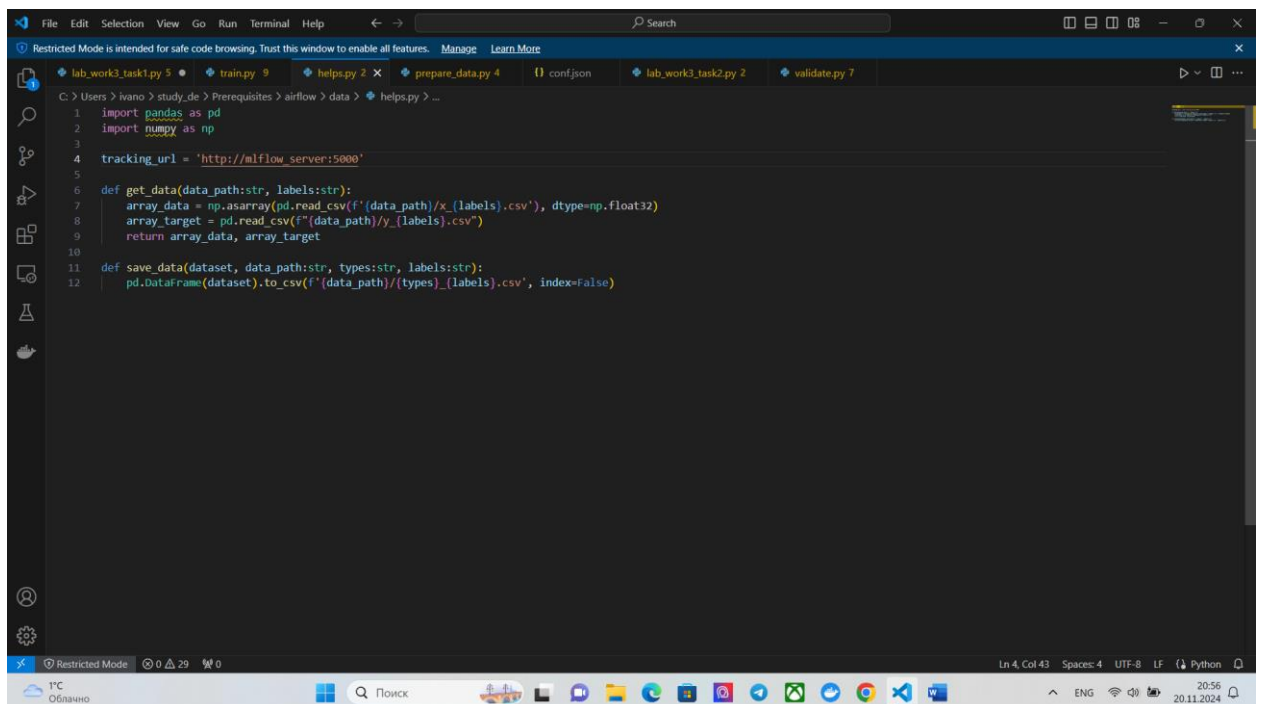
4) labels – метка выборки (обучающая, тестовая, валидационная).

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.



```
File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
lab_work3_task1.py 5 train.py 9 helps.py 2 x prepare_data.py 4 conf.json lab_work3_task2.py 2 validate.py 7
C:\Users\ivano> study.de > Prerequisites > airflow > data > helps.py > ...
1 import pandas as pd
2 import numpy as np
3
4 tracking_url = 'http://mlflow_server:5000'
5
6 def get_data(data_path:str, labels:str):
7     array_data = np.asarray(pd.read_csv(f'{data_path}/x_{labels}.csv'), dtype=np.float32)
8     array_target = pd.read_csv(f'{data_path}/y_{labels}.csv')
9     return array_data, array_target
10
11 def save_data(dataset, data_path:str, types:str, labels:str):
12     pd.DataFrame(dataset).to_csv(f'{data_path}/{types}_{labels}.csv', index=False)
```

Рисунок 5 – Код вспомогательного модуля `helps`

Шаг 4. Обучение моделей.

Обучение моделей сосредоточено в task-е - `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

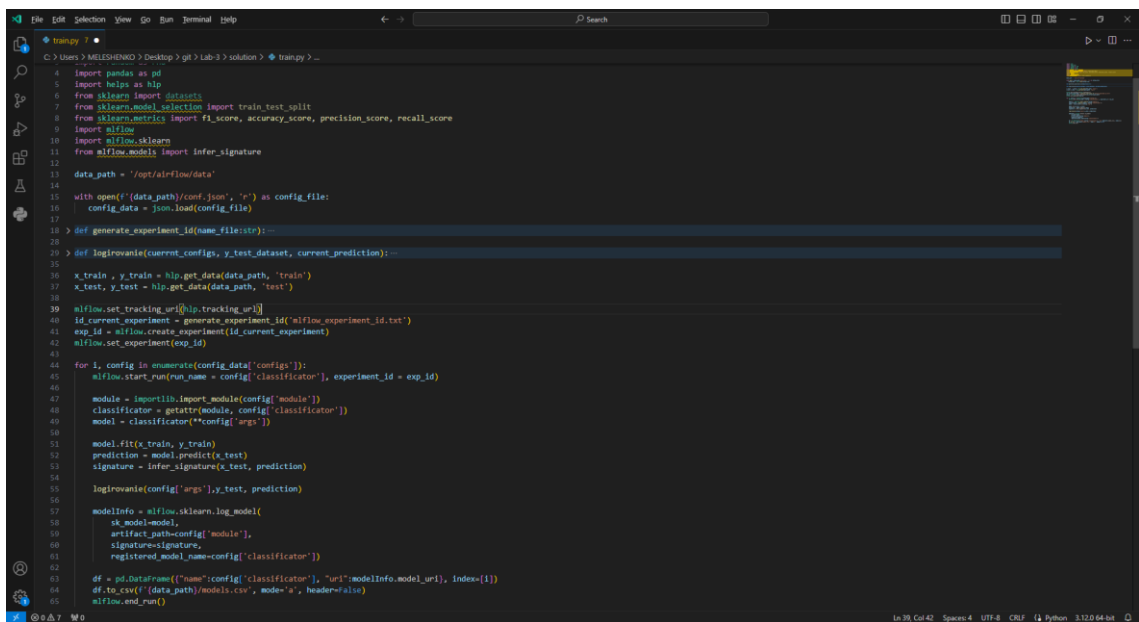
После всех приготовлений запускаем эксперимент по обучению моделей. При помощи функции `get_data()` из самописного модуля `helps` получаем данные для обучения моделей. Устанавливаем подключение к `mlflow`. Генерируем `experiment_id` при помощи функции

`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.



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

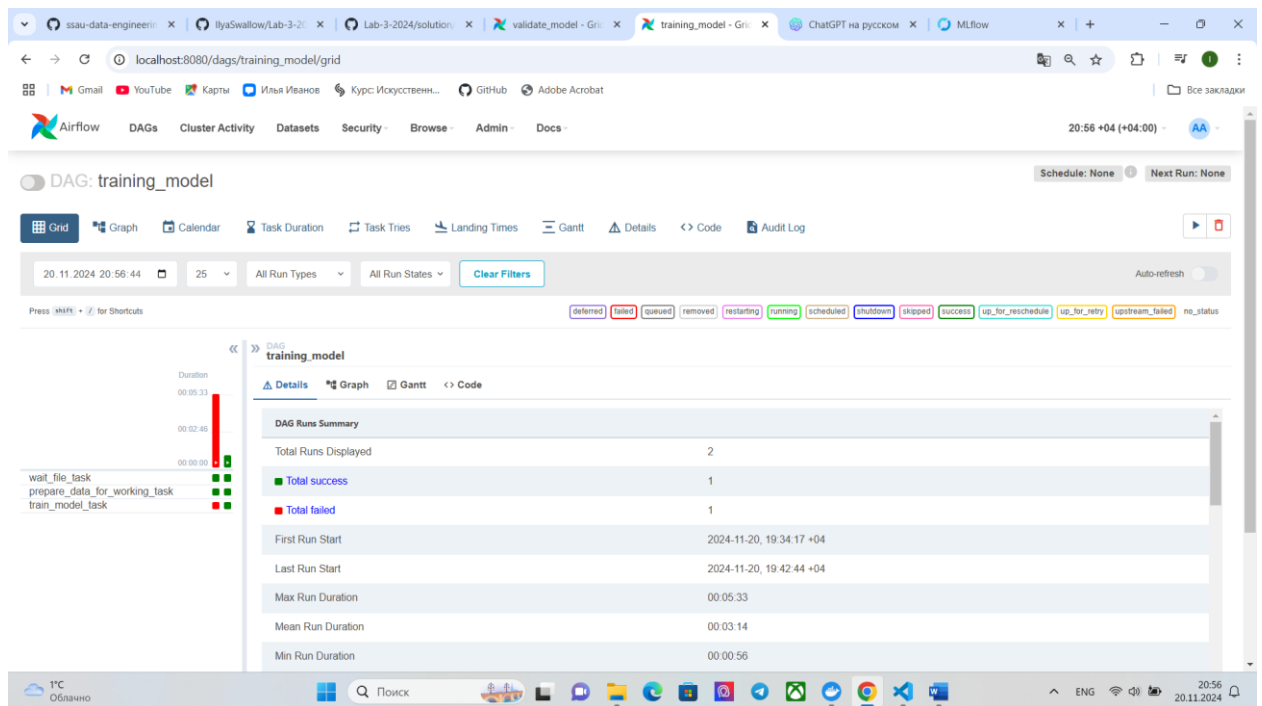


Рисунок 9 – Запуск DAG-а по обучению моделей.

После отработки DAG-а перейдем в Mlflow и посмотрим на результаты обучения, которые изображены на рисунке ниже.

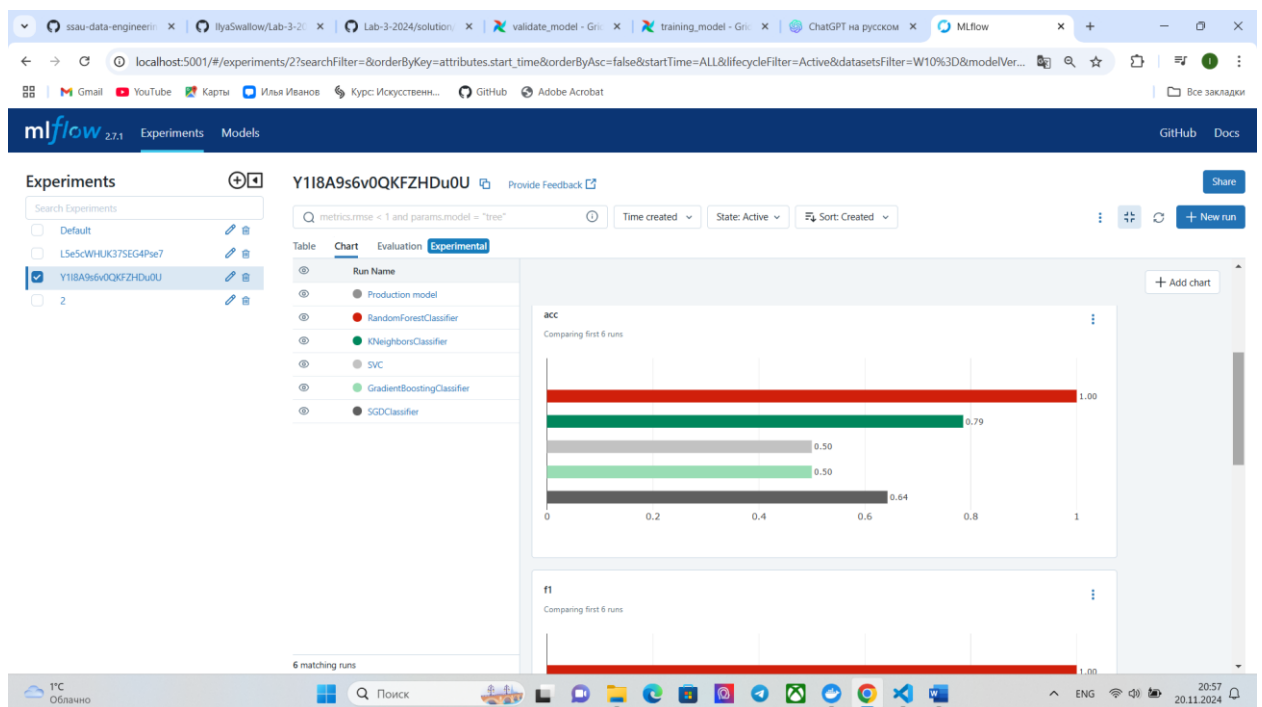


Рисунок 10 – Результат обучения моделей

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.

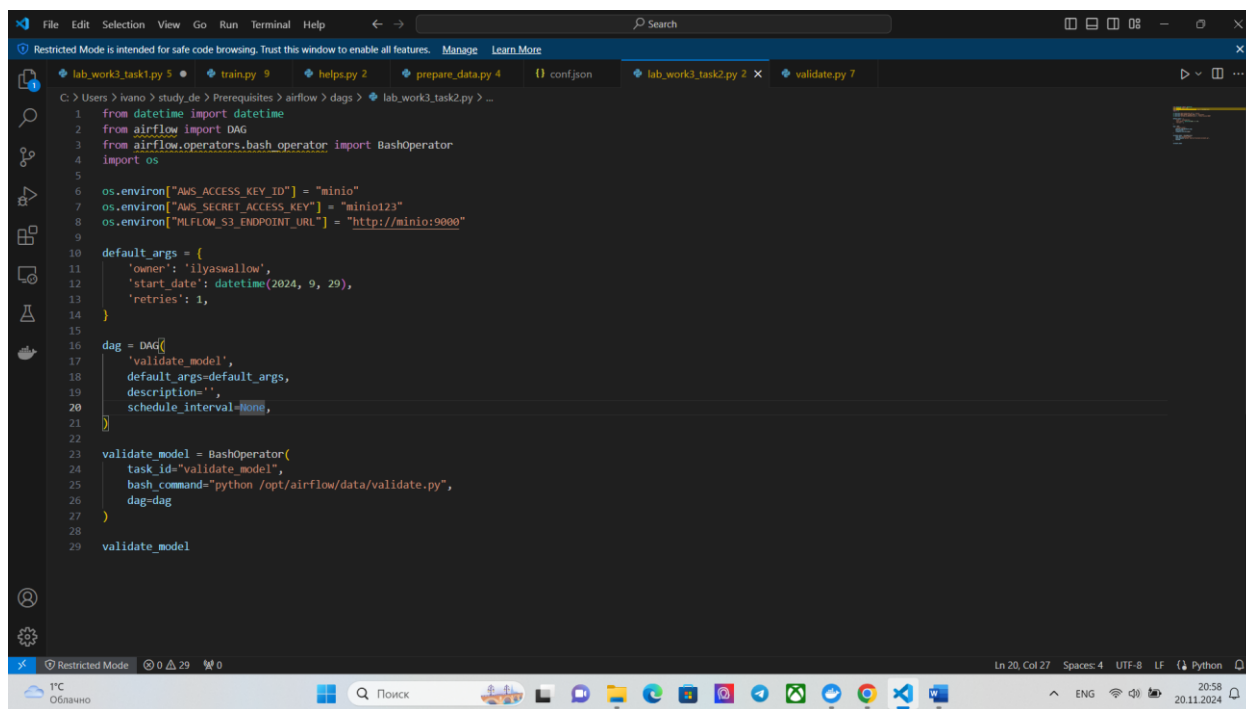


Рисунок 11 – DAG валидации моделей

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.

```
validate.py | X | lab_work_task1.py | lab_work_task2.py |
1 import pandas as pd
2 import numpy as np
3 from sklearn.metrics import accuracy_score
4 import heapq as hp
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/mlflow/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 = hp.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         uri = model_info[1][2]
27         list_models_for_validate.append(name + " " + uri)
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.

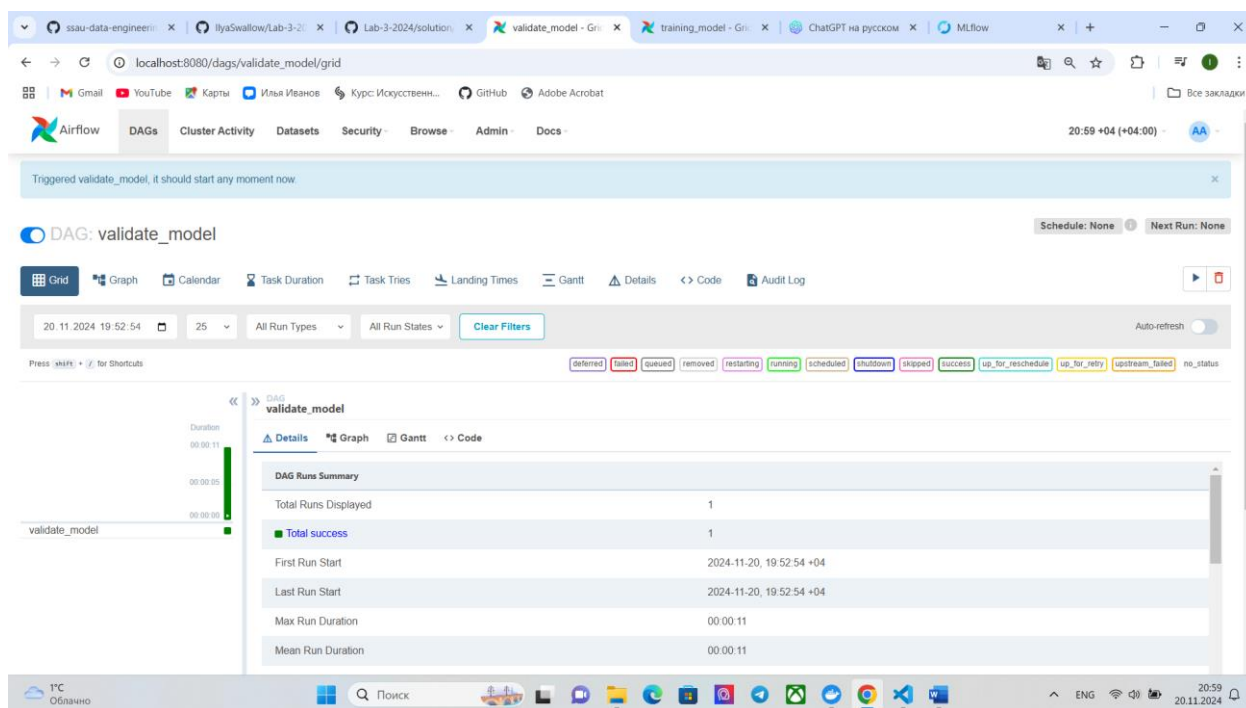


Рисунок 13 – Запуск DAG-а валидации моделей

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.

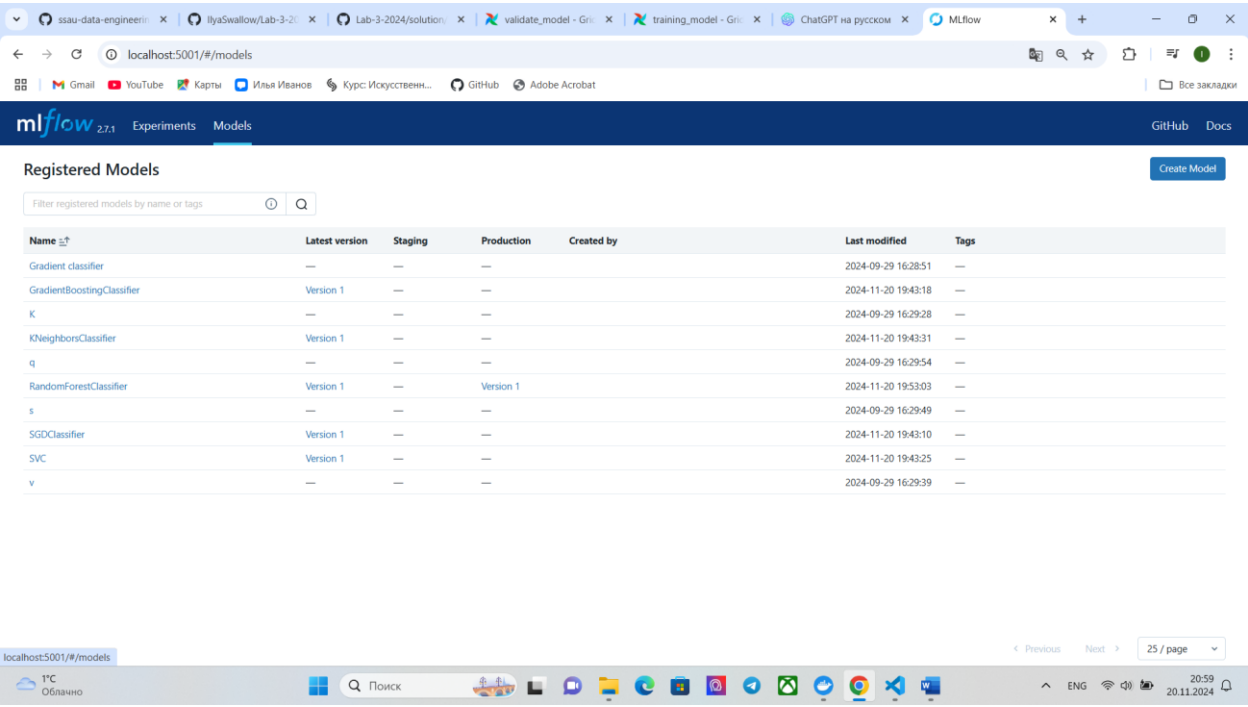


Рисунок 13 – Результат выполнения, DAG-а валидации моделей

Заключение

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