**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/12 Интеллектуальный анализ больших данных в системах поддержки принятия решений.**

# О Т Ч Е Т

## по лабораторной работе № 2-4

**Название:** Ирисы Фишера

**Дисциплина:** Платформы промышленной аналитики

| | | | |
|---|---|---|---|
| Студент | ИУ6-43М | | Ф.А. Лучкин |
| | (Группа) | (Подпись, дата) | (И.О. Фамилия) |
| Студент | ИУ6-43М | | А.А. Павловский |
| | (Группа) | (Подпись, дата) | (И.О. Фамилия) |
| Преподаватель | | | М. А. Скворцова |
| | | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2025

**Цель лабораторной работы №2:** подготовить набор данных для построения модели машинного обучения.

**Цель лабораторной работы №3:** построение сложного паплайна, включающего в себя возможность проверки модели на не менее чем 3х различных методах машинного обучения, релевантных поставленной задаче.

**Цель лабораторной работы №4:** доработка модели машинного обучения, оценка ее качества и проверка решения задачи прогнозирования.

Из данных целей исходят следующие задания.

**Задания:**

1. Загрузить данные;

2. Выполнить визуальный анализ данных (pairplot);

3. Построить простую логистическую регрессию для бинарной классификации;

4. Продемонстрировать переобучение модели линейной регрессии на искусственных данных, описать, как можно бороться с переобучением;

5. Выполнить обогащение датасета, представить код генерации данных;

6. Выполнить построение AutoML пайплайнов (LightAutoML, FEDOT, TPOT), предоставить схемы и сравнение эффективности работы алгоритмов;

7. Выполнить оценку качества одной из моделей (улучшить модели, если возможно), выполнить визуализацию результатов, развернуть модель в MLFlow.

**Задание 1**

Загрузка данных и данные показаны на рисунке 1.

**Задание 2**

Для анализа данных были посчитаны различные статистки по фичам и таргетам, а также выполнено построение графиков pairplot для визуального анализ распределения данных. Вывод статистик показан на рисунке 2, а графики pairplot – на рисунке 3.

Загрузка данных

```
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
df = pd.read_csv(url, names=column_names)

df
```

✓ 2.2s

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| ... | ... | ... | ... | ... | ... |
| 145 | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

150 rows × 5 columns

Рисунок 1 – загрузка данных

```
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

Рисунок 2 – статистический анализ данных

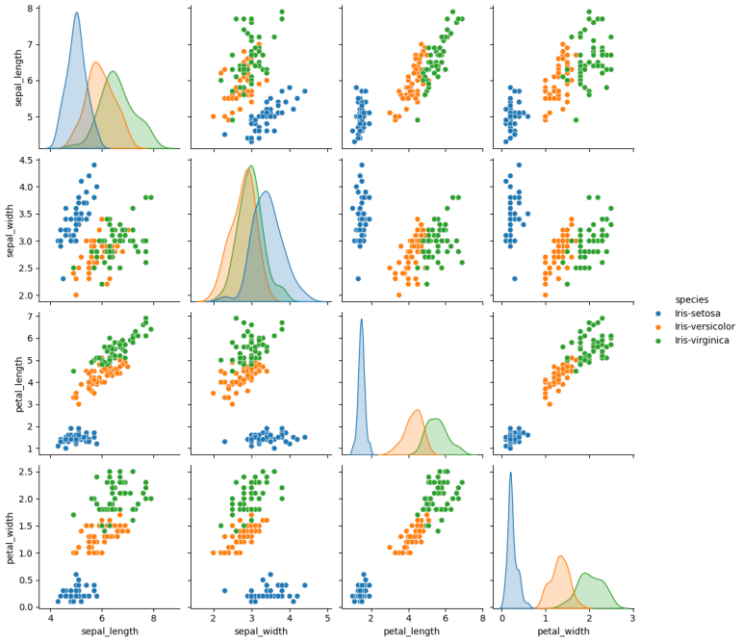


Рисунок 3 – pairplot графики

На основе полученных данных можно сделать вывод, что датасет содержит 3 класса, которые сбалансированы и довольно легко визуально-разделимы.

**Задание 3**

Для выполнения задачи бинарной классификации были взяты два наиболее сложно разделимых класса из трёх – virginica и versicolor.

Листинг программы построения и обучения модели бинарной

```
df_binary = df[df['species'].isin(['Iris-virginica', 'Iris-versicolor'])].copy()
df_binary['species'] = df_binary['species'].map({'Iris-virginica': 0, 'Iris-versicolor': 1})

X = df_binary.drop('species', axis=1)
y = df_binary['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

model = LogisticRegression(random_state=42, solver='liblinear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy, precision, recall = (
    accuracy_score(y_test, y_pred),
    precision_score(y_test, y_pred),
    recall_score(y_test, y_pred),
)
print(f'Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}')
```

Данные являются довольно простыми, а классы всё же легко разделимы, поэтому были получены высокие показатели метрик. Accuracy: 0.9, Precision: 1.0, Recall: 0.8235294117647058.

**Задание 4**

Для демонстрации переобучения был использован следующий код генерации искусственных данных.

Листинг программы генерации искусственных данных и переобученной модели

```
# Установка семени для воспроизводимости результатов
np.random.seed(5)

# Параметры моделирования
n_features = 1  # количество признаков (фичей)
n_samples = 6   # количество образцов в тренировочном наборе

# Генерация данных для тренировочного набора
w_true = np.random.randn(n_features)  # истинные веса модели
X_train = np.random.randn(n_samples, n_features)
y_train = X_train @ w_true + np.random.randn(n_samples)

# Генерация данных для валидационного набора
X_val = np.random.randn(n_samples, n_features)
y_val = X_val @ w_true + np.random.randn(n_samples)

# Генерация данных для тестового набора
X_test = np.random.randn(n_samples, n_features)
y_test = X_test @ w_true + np.random.randn(n_samples)

# Генерация полиномиального набора данных
x_begin = -1.05
x_end = 2.5
n_polynomial_terms = 5  # количество членов в полиноме
X_polynomial = np.hstack([np.ones((n_samples, 1)), X_train ** np.arange(1, n_polynomial_terms + 1)])
w_polynomial = np.linalg.inv(X_polynomial.T @ X_polynomial) @ X_polynomial.T @ y_train
y_polynomial = [
   np.array([x**i for i in range(n_polynomial_terms + 1)]) @ w_polynomial for x in np.linspace(x_begin, x_end)
]

# Визуализация данных
plt.plot(X_train, y_train, 'o', label='Тренировочный набор')
plt.plot(X_test, y_test, 'o', label='Тестовый набор')
plt.plot(np.linspace(x_begin, x_end), w_true * np.linspace(x_begin, x_end), '-', label='Истинная модель')
plt.plot(np.linspace(x_begin, x_end), y_polynomial, '-', label='Полиномиальная модель')

# Добавление легенды и отображение графика
plt.legend(loc='best')
plt.show()
```

Результирующие графики показаны на рисунке 4.

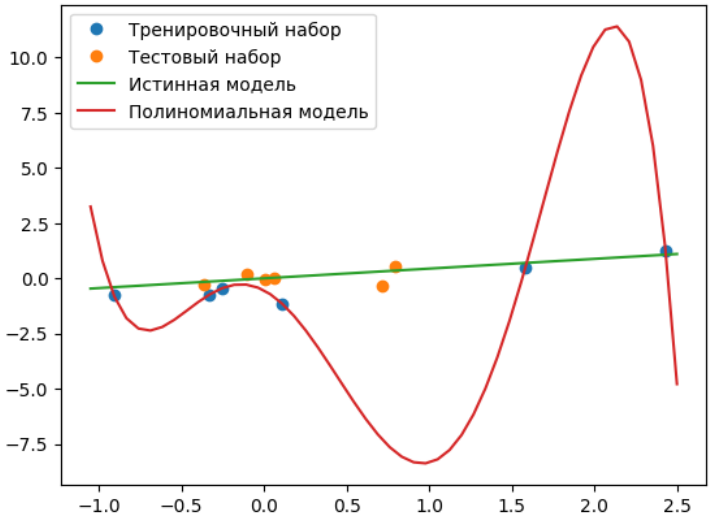


Рисунок 4 – демонстрация переобучения

Переобучение (overfitting) — это явление, когда модель обучается слишком хорошо на тренировочном наборе данных и плохо справляется с новыми данными. Это происходит из-за того, что модель становится слишком сложной и начинает учиться на шумах или случайных вариациях в данных.

Существет множество методов борьбы с переобучением, вот некотоые из них:

1. Упрощение модели: Сокращение количества параметров модели или использования более простой структуры. Например, вместо полиномиальной модели можно использовать линейную.

2. Регуляризация: Добавление небольшого значения к весам модели для предотвращения чрезмерного увеличения их величины. Это может быть сделано с помощью L1-регуляризации (Lasso) или L2-регуляризации (Ridge).

3. Понижение степени свободы: Уменьшение количества степеней свободы модели, что можно сделать с помощью методов снижения размерности, таких как PCA (Principal Component Analysis) или t-SNE.

4. Кросс-валидация: Разделение данных на тренировочный и тестовый наборы для оценки качества модели.

5. Early Stopping: Остановка обучения модели при достижении определенного показателя качества на тестовом наборе.

6. Дропаут: Удаление случайных нейронов или слоев во время обучения, чтобы предотвратить чрезмерное увеличение сложности модели.

7. Батч-нормализация: Нормализация входных данных для каждого батча (мини-блока) во время обучения, что помогает уменьшить влияние шума в данных.

8. Использование более простых функций активации: Использование функций активации, которые не имеют чрезмерного увеличения сложности, такие как ReLU или Sigmoid.

**Задание 5**

Для обогащения датасета был написан собственный метод. Нам уже был известен способ обогащения SMOTE, который используется для устранения дисбаланса классов. Ниже нами был реализован метод, со схожим механизмом генерации новых экземпляров через выпуклые комбинации пар точек, но расширяющий все классы, а не только меньший из них.

Листинг функции обогащения датасета

```
def generate_convex_combinations(group, num_combinations=1):
    new_samples = []
    n = len(group)
    if n < 2:
        return pd.DataFrame()  # Нельзя создать комбинации, если менее 2 образцов

    features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
    for _ in range(num_combinations):
        # Выбираем две случайные разные строки
        idx1, idx2 = np.random.choice(n, 2, replace=False)
        sample1 = group.iloc[idx1]
        sample2 = group.iloc[idx2]

        # Генерируем случайный коэффициент alpha между 0 и 1
        alpha = np.random.uniform(0, 1)

        # Выпуклая комбинация признаков
        new_features = alpha * sample1[features] + (1 - alpha) * sample2[features]

        # Создаём новую строку
        new_sample = new_features.to_dict()
        new_sample['species'] = sample1['species']
        new_samples.append(new_sample)

    return pd.DataFrame(new_samples)
```

Работа алгоритма:

1. Выбор двух случайных точек из одного класса

• Для каждого класса (например, Iris-setosa) берутся две случайные точки (A) и (B).

• Их признаки (длина/ширина чашелистиков и лепестков) рассматриваются как векторы в 4-мерном пространстве.

2. Вычисление выпуклой комбинации

• Генерируется случайный коэффициент (alpha in [0, 1]).

• Новая точка (C) вычисляется как: [C = alpha * A + (1 - alpha) * B]

• Это означает, что (C) лежит на отрезке между (A) и (B) в пространстве признаков.

3. Сохранение класса

• Новая точка (C) наследует метку класса от (A) и (B), так как они принадлежат одному классу.

Геометрическая интерпретация:

• В пространстве признаков (например, в 2D-проекции sepal_length и petal_length):

  - Исходные точки образуют облако (кластер).
  - Новые точки заполняют выпуклую оболочку этого кластера, не выходя за его границы.

• Если исходные точки лежат в некотором многообразии (например, на гиперплоскости), новые точки также останутся в нём.

В результате работы метода был увеличен исходный датасет:

```
Original size: 150
Augmented size: 225
Iris-setosa        75
Iris-versicolor    75
Iris-virginica     75
Name: species, dtype: int64
```

На рисунке 5 показан обновлённый pairplot.

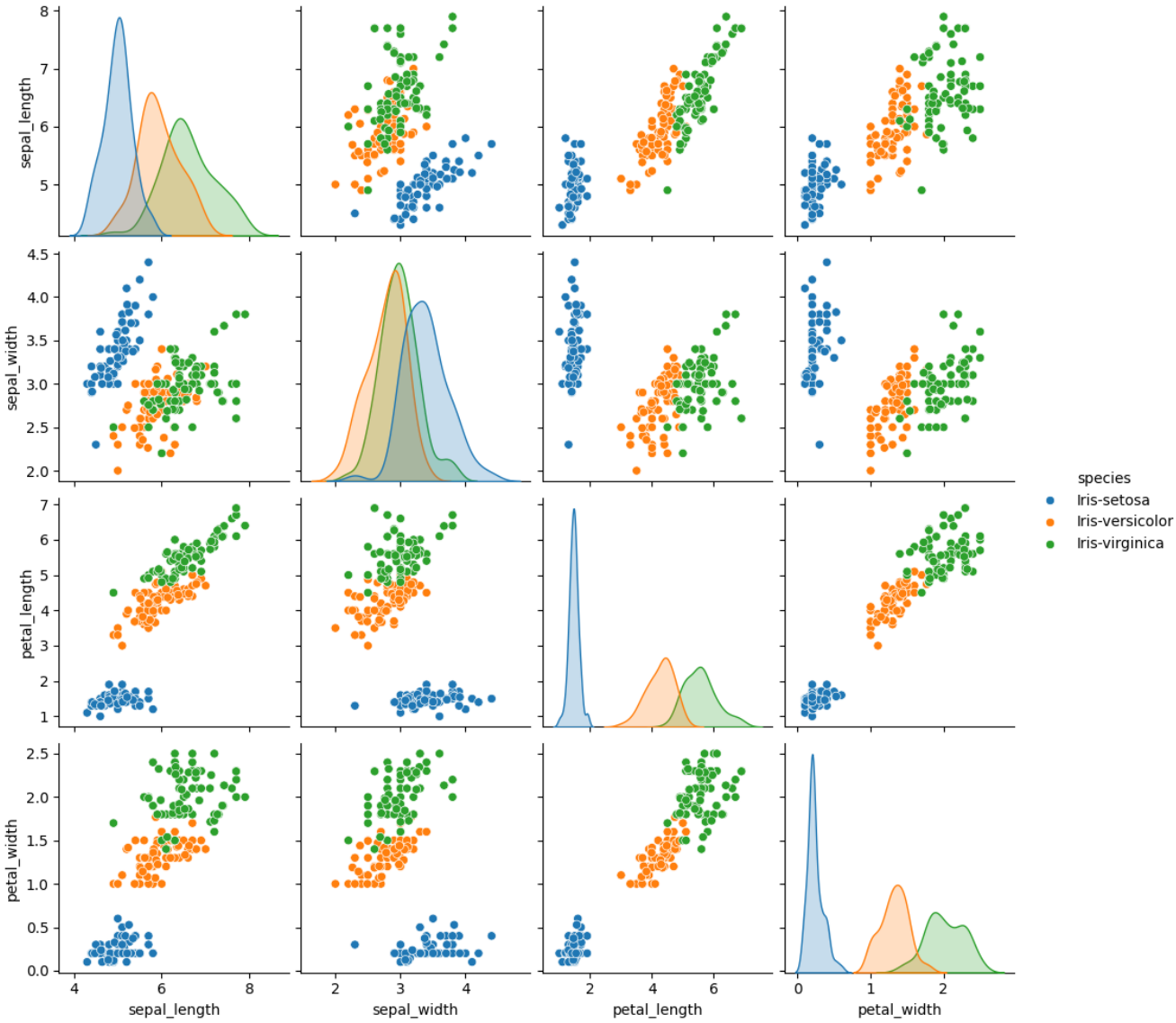


Рисунок 5 – pairplot обогащённого датасета

Как видно на графиках, распределения не поменялись, изменилось лишь количество экземпляров каждого из трёх классов.

**Задание 6**

В рамках задания были использованы LightAutoML, FEDOT и TPOT.

Код подготовки данных, обучения, использования и оценки качества модели при использовании с LightAutoML показан ниже.

```
# Преобразование данных
X = augmented_df.drop('species', axis=1)
mapper = {
    'Iris-virginica': 0,
    'Iris-versicolor': 1,
    'Iris-setosa': 2
}
y = augmented_df['species'].replace(mapper)

# Разделение данных
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Создание задачи
task = Task('multiclass')

# Настройка AutoML
automl = TabularAutoML(
    task=task,
    cpu_limit=-1,
    general_params={'use_algos': 'auto'},
    reader_params={'cv': 3, 'random_state': 42}
)

# Обучение модели
train_data = pd.concat([X_train, y_train], axis=1)
roles = {
    'target': 'species',
    'drop': []
}

oof_pred = automl.fit_predict(
    train_data=train_data,
    roles=roles,
    verbose=1
)

# Предсказание
test_pred = automl.predict(X_test)
y_pred = test_pred.data.argmax(axis=1)

# Оценка качества
accuracy, precision, recall = (
    accuracy_score(y_test, y_pred),
    precision_score(y_test, y_pred, average='weighted'),
    recall_score(y_test, y_pred, average='weighted'),
)
clear_output(wait=False)
print(f'Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}')
```

В результате была получена модель, показанная на рисунке 6.

```
Тип алгоритма: LinearLBFGS
Тип модели: TorchBasedLogisticRegression

=== Параметры TorchBasedLogisticRegression ===
categorical_idx: []
cs: [1e-05, 5e-05, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000]
data_size: 4
early_stopping: 2
embed_sizes: ()
fit: <bound method TorchBasedLinearEstimator.fit of <lightautoml.ml_algo.torch_based.linear_model.TorchBasedLogisticRegression object at 0x000002113FA846A0>>

Функция потерь: TorchLossWrapper
max_iter: 100
metric: <lightautoml.tasks.losses.base.MetricFunc object at 0x000002113D50CCA0>

Архитектура нейросети:
CatMulticlass(
  (linear): Linear(in_features=4, out_features=3, bias=False)
  (final_act): SoftmaxClip(
    (smax): Softmax(dim=1)
  )
)
output_size: 3
predict: <bound method TorchBasedLogisticRegression.predict of <lightautoml.ml_algo.torch_based.linear_model.TorchBasedLogisticRegression object at 0x000002113FA846A0>>
tol: 1e-06
```

Рисунок 6 – лучшая модель от LightAutoML

Данная модель представляет из себя искусственную нейронную сеть и идеально выполняет задачу мультиклассовой классификации. Accuracy: 1.0, Precision: 1.0, Recall: 1.0.

Полный пайплайн работы с LightAutoML показан на рисунке 7.


Рисунок 7 – пайплайн работы с LightAutoML

Как можно заметить, для корректной работы с LightAutoML необходимо преобразовывать таргеты в числа.

Код подготовки данных, обучения, использования и оценки качества модели при использовании с FEDOT показан ниже.

```
# Преобразование данных
X = augmented_df.drop('species', axis=1)
y = augmented_df['species']

# Разделение данных
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Инициализация Fedot с явным указанием доступных моделей
automl_model = Fedot(
    problem='classification',
    preset='fast_train',
    # timeout=10,  # 2 минуты на подбор
    available_operations=['rf', 'logit','mlp', 'xgboost']
    logging_level=logging.CRITICAL,
    with_tuning=True,
    n_jobs=-1,
    seed=42
)
```

```
try:
    # Обучение модели
    pipeline = automl_model.fit(features=X_train, target=y_train)

    # Предсказание
    y_pred = automl_model.predict(features=X_test)

    # Оценка качества
    accuracy, precision, recall = (
        accuracy_score(y_test, y_pred),
        precision_score(y_test, y_pred, average='weighted'),
        recall_score(y_test, y_pred, average='weighted'),
    )
    print(f'Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}')


except Exception as e:
    print(f'Произошла ошибка: {str(e)}')
```

В результате была получена модель, показанная на рисунке 8.

```
Узел: mlp
Параметры: {}
Fitted_params: {
        activation: relu
        alpha: 0.0001
        batch_size: auto
        beta_1: 0.9
        beta_2: 0.999
        early_stopping: False
        epsilon: 1e-08
        hidden_layer_sizes: (100,)
        learning_rate: constant
        learning_rate_init: 0.001
        max_fun: 15000
        max_iter: 200
        momentum: 0.9
        n_iter_no_change: 10
        nesterovs_momentum: True
        power_t: 0.5
        random_state: None
        shuffle: True
        solver: adam
        tol: 0.0001
        validation_fraction: 0.1
        verbose: False
        warm_start: False
}
```

Рисунок 8 – лучшая модель от FEDOT

Данная модель представляет из себя искусственную нейронную сеть и идеально выполняет задачу мультиклассовой классификации. Accuracy: 1.0, Precision: 1.0, Recall: 1.0.

Полный пайплайн работы с FEDOT показан на рисунке 9.


Рисунок 9 – пайплайн работы с FEDOT

FEDOT отлично работает и без преобразования таргетов.

Код подготовки данных, обучения, использования и оценки качества модели при использовании с TPOT показан ниже.

```
# Преобразование данных
X = augmented_df.drop('species', axis=1)
y = augmented_df['species']

# Разделение данных
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

tpot = TPOTClassifier(generations=5, population_size=20, random_state=42, verbosity=2)
tpot.fit(X_train, y_train)

# Оценка производительности лучшей модели
y_pred = tpot.predict(X_test)
accuracy, precision, recall = (
    accuracy_score(y_test, y_pred),
    precision_score(y_test, y_pred, average='weighted'),
    recall_score(y_test, y_pred, average='weighted'),
)

print(f'Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}')
```

В результате была получена модель, показанная на рисунке 10.

```
Тип: MLPClassifier
Параметры:
        activation: relu
        alpha: 0.001
        batch_size: auto
        beta_1: 0.9
        beta_2: 0.999
        early_stopping: False
        epsilon: 1e-08
        hidden_layer_sizes: (100,)
        learning_rate: constant
        learning_rate_init: 0.01
        max_fun: 15000
        max_iter: 200
        momentum: 0.9
        n_iter_no_change: 10
        nesterovs_momentum: True
        power_t: 0.5
        random_state: 42
        shuffle: True
        solver: adam
        tol: 0.0001
        validation_fraction: 0.1
        verbose: False
        warm_start: False
```

Рисунок 10 – лучшая модель от TPOT

Данная модель представляет из себя искусственную нейронную сеть и идеально выполняет задачу мультиклассовой классификации. Accuracy: 1.0, Precision: 1.0, Recall: 1.0.

Полный пайплайн работы с TPOT показан на рисунке 11.


Рисунок 11 – пайплайн работы с TPOT

TPOT тоже работает без преобразования таргетов, а также, имеет очень простую настройку.

Таблица 1 – Результаты работы моделей

| параметр | LightAutoML | FEDOT | TPOT |
|---|---|---|---|
| F1-score | 1 | 1 | 1 |
| Precision | 1 | 1 | 1 |
| Recall | 1 | 1 | 1 |

Поставленная задача является очень простой, поэтому, как видно в таблице 1, все autoML решения достигли максимальных показателей метрик качества.

**Задание 6**

В рамках задания улучшение модели не выполнялось (был взят пайплайн TPOT), так как модель и так добились максимального качества. Результаты работы модели на тестовой выборке показаны на рисунке 12.

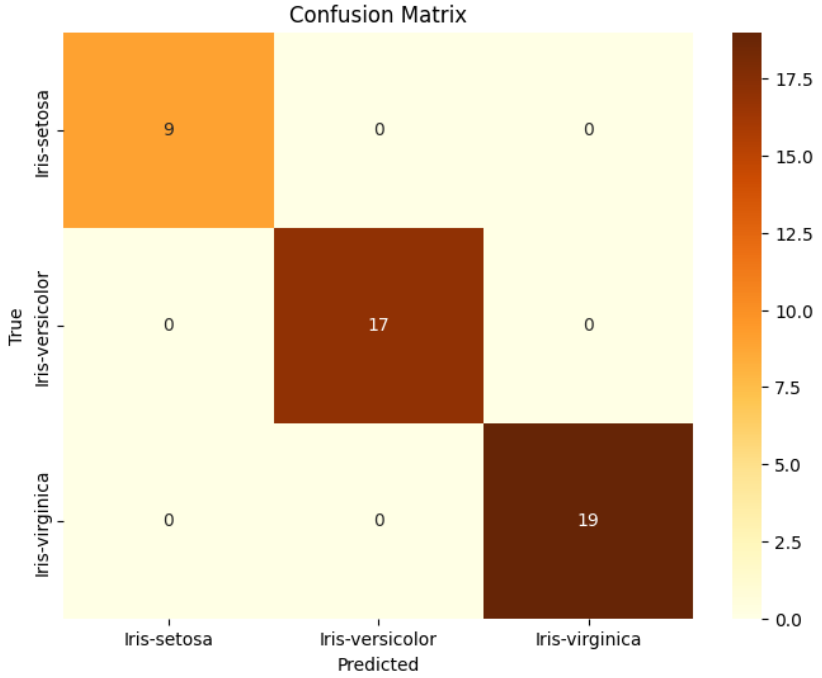


Рисунок 12 – визуализация работы модели на тестовой выборке

Как видно, модель действительно делает идеально точные прогнозы.

Запуск сервера MLFlow производится при помощи команды:

```
mlflow server --host 127.0.0.1 --port 8080
```

Интерфейс можно открыть в браузере по адресу: http://127.0.0.1:8080

Код работы с MLFlow на примере TPOT приведён ниже.

```
# Указываем URI для отслеживания
mlflow.set_tracking_uri("http://127.0.0.1:8080")

# Устанавливаем эксперимент
mlflow.set_experiment("Iris_Classification_TPOT")
```

```
with mlflow.start_run():
    # Параметры TPOT
    params = {
        'generations': 5,
        'population_size': 20,
        'random_state': 42,
        'verbosity': 2
    }

    # Логируем параметры
    mlflow.log_params(params)

    # 1. Обучение TPOT
    tpot = TPOTClassifier(**params)
    tpot.fit(X_train, y_train)

    # 2. Сохранение модели в .pkl
    with open('tpot_model.pkl', 'wb') as f:
        pickle.dump(tpot.fitted_pipeline_, f)

    # 3. Логируем .pkl файл в MLflow
    mlflow.log_artifact('tpot_model.pkl')

    # 4. Альтернативно: сохраняем как sklearn модель
    mlflow.sklearn.log_model(tpot.fitted_pipeline_, "sklearn_model")

    # Оценка и логирование метрик
    y_pred = tpot.predict(X_test)
    metrics = {
        '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')
    }
    mlflow.log_metrics(metrics)

    # Экспорт лучшего пайплайна
    tpot.export('tpot_best_pipeline.py')
    mlflow.log_artifact('tpot_best_pipeline.py')

    print(f"Модель сохранена в tpot  model.pkl")
```

Результаты работы отображены на рисунках 13-я.

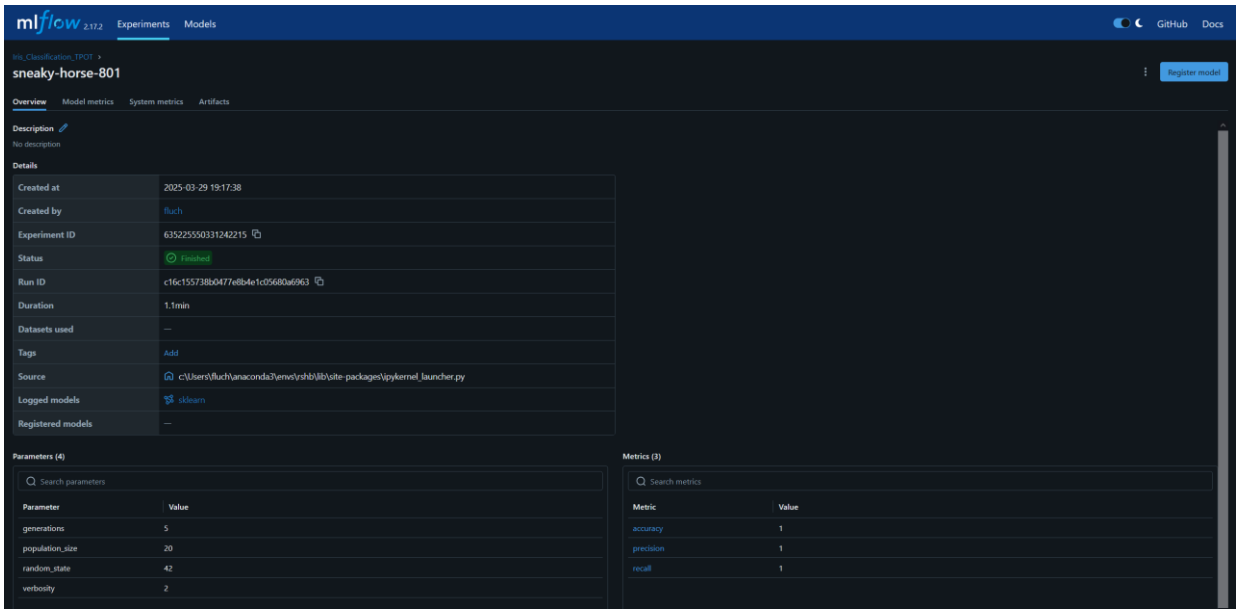


Рисунок 13 – страница рана модели


Рисунок 14 – страница метрик качества работы модели

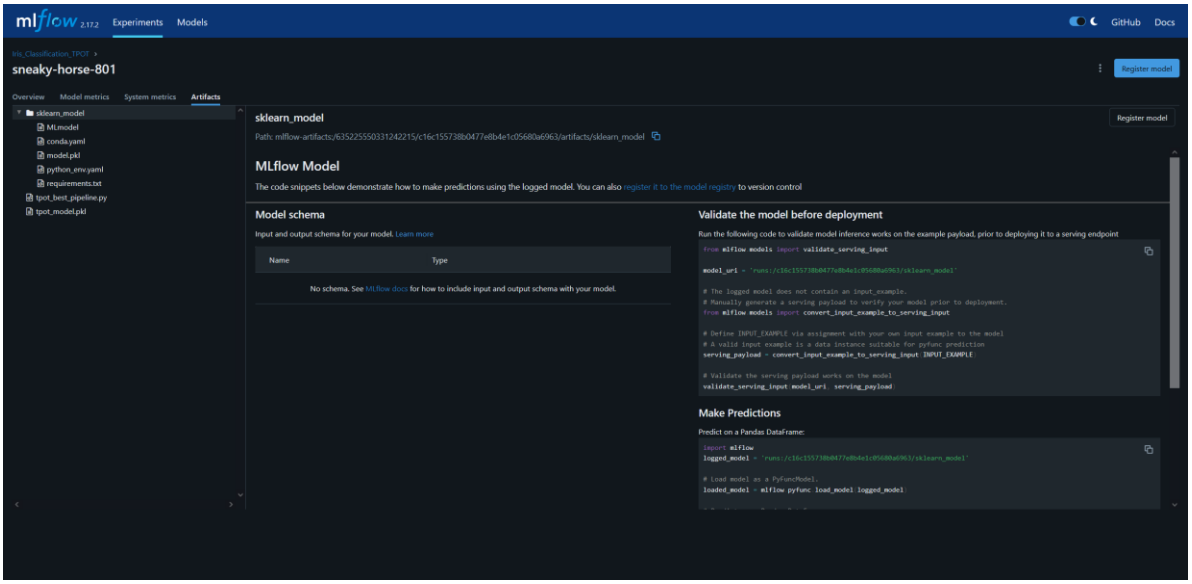


Рисунок 13 – страница артефактов, полученных при обучении модели

Как показано на рисунках выше, взаимодействие с MLFlow прошло успешно.

## Вывод

В процессе выполнения лабораторной работы был собран, проанализирован и обогащён датасет, обучена простая модель бинарной классификации, рассмотрена проблема переобучения моделей, а также выполнено построение моделей машинного обучения при помощи autoML решений. Результирующая подель и процесс её обучения и инференса были загружены на MLFlow.