

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

по лабораторной работе № 2-4

Дисциплина: Платформы промышленной аналитики

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

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 (boxplot);
3. Проанализировать данные и провести их предобработку (очистку и генерацию признакового пространства);
4. Провести оценку взаимосвязи данных датасета;
5. Провести разведочный анализ данных с помощью методов PCA и TSNE, сделать вывод о структуре признаков данных;
6. Выполнить обогащение датасета;
7. Выполнить построение AutoML пайплайнов (LightAutoML, FEDOT, TPOT), предоставить схемы и сравнение эффективности работы алгоритмов;
8. 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 начала были просмотрены типы данных колонок и количество пропусков. Из полученной информации следовало, что по каждому наблюдению может быть несколько отчётов, в колонках report количество пропущенных значений растёт с увеличением номера отчёта, данные колонки имеют строковый тип. Также, нам не интересны колонки идентификаторов, поэтому их можно удалить.

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

```
url = 'https://raw.githubusercontent.com/AI-is-out-there/data2lab/refs/heads/main/модуль 2 - датасет - практика.csv'
response = requests.get(url)
response.encoding = 'utf-8' # Устанавливаем кодировку
data = StringIO(response.text)

df = pd.read_csv(data)
df
```

	subject_id	Count_subj	study_id	cart_id	Healthy_Status	eeg_time	eeg_date	report_0	report_1	report_2	...	filtering	rr_interval	p_onset	p_end	qrs_onset	qrs_end	t_end	p_axis
0	19557662	27	40000017	6848296	0	8:44 AM	27.06.2015	Sinus rhythm	Possible right atrial abnormality	NaN	...	60 Hz notch Baseline filter	659	40	128	170	258	518	81
1	18477137	93	40000029	6848296	0	9:54 AM	27.06.2015	Sinus rhythm	Possible right atrial abnormality	NaN	...	60 Hz notch Baseline filter	722	40	124	162	246	504	77
2	16598616	3	40000035	6376932	1	9:07 AM	28.06.2015	Sinus tachycardia	NaN	Normal ECG except for rate	...	60 Hz notch Baseline filter	600	40	130	162	244	474	79
3	16368287	7	40000079	6214760	1	5:14 PM	15.07.2015	Sinus rhythm	NaN	Normal ECG	...	60 Hz notch Baseline filter	659	40	146	180	254	538	79
4	18370366	2	40000084	6632385	0	1:52 PM	27.09.2015	Sinus rhythm	NaN	NaN	...	<not specified>	659	368	29999	504	590	868	84

Рисунок 1 – загрузка данных

	Healthy_Status	rr_interval	p_onset	p_end	qrs_onset
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	0.200700	880.614300	4781.92660	8930.689000	294.337500
std	0.400544	1350.168399	10879.47242	13602.948503	1266.163235
min	0.000000	0.000000	14.000000	0.000000	0.000000
25%	0.000000	682.000000	40.000000	144.000000	188.000000
50%	0.000000	810.000000	40.000000	158.000000	200.000000
75%	0.000000	952.000000	329.000000	29999.000000	228.000000
max	1.000000	29999.000000	29999.000000	29999.000000	29999.000000

	qrs_end	t_end	p_axis	qrs_axis	t_axis
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	403.255900	697.381600	5077.596000	144.047600	227.735800
std	1366.191721	1317.597859	11248.070976	1963.110038	2377.875452
min	139.000000	335.000000	-21846.000000	-178.000000	-180.000000
25%	278.000000	564.000000	37.000000	-15.000000	17.000000
50%	300.000000	610.000000	57.000000	14.000000	43.000000
75%	342.000000	668.000000	72.000000	46.000000	70.000000
max	29999.000000	29999.000000	32767.000000	29999.000000	32767.000000


```
0    7993
1    2007
Name: Healthy_Status, dtype: int64
```

Рисунок 2 – статистический анализ данных

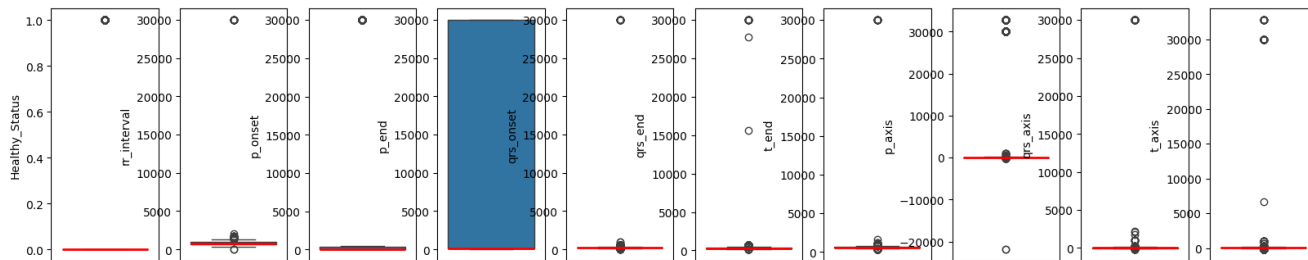


Рисунок 3 – boxplot графики

На основе полученных данных можно сделать вывод, что датасет содержит 2 класса, которые имеют сильный дисбаланс, а данные в конках фичей имеют явные выбросы.

Задание 3

Начнём фильтрацию данных.

Колонку `p_onset` отбрасываем, так как вывод `value_counts` говорит о бесполезности колонки в задаче прогнозирования. Вывод `value_counts` показан на рисунке 4.

```
filtered_df['p_onset'].value_counts()
✓ 0.0s
```

40	7043
24	2
14	1

Name: p_onset, dtype: int64

Рисунок 4 – value_counts колонки `p_onset`

На части данных чётко видны выбросы, избавимся от них. Также, на всякий случай отсеем строки, в которых время начала волны или комплекса больше, чем конец.

Листинг кода очистки данных от выбросов

```
outlier_columns = ['rr_interval', 'p_axis', 'p_end', 'p_onset', 'qrs_axis', 'qrs_end', 'qrs_onset',
't_axis', 't_end']
filtered_df = df.loc[
    (df[outlier_columns] < 2000).all(axis=1)
] 'p_onset' и 'qrs_onset'
filtered_df = filtered_df.query(
    '(p_onset < p_end) & (qrs_onset < qrs_end)'
)

filtered_df = filtered_df.rename(columns={'eeg_time ': 'eeg_time', 'eeg_date ': 'eeg_date'})
```

На очищенных данных заново построим boxplot. Обновлённый график показан на рисунке 5.

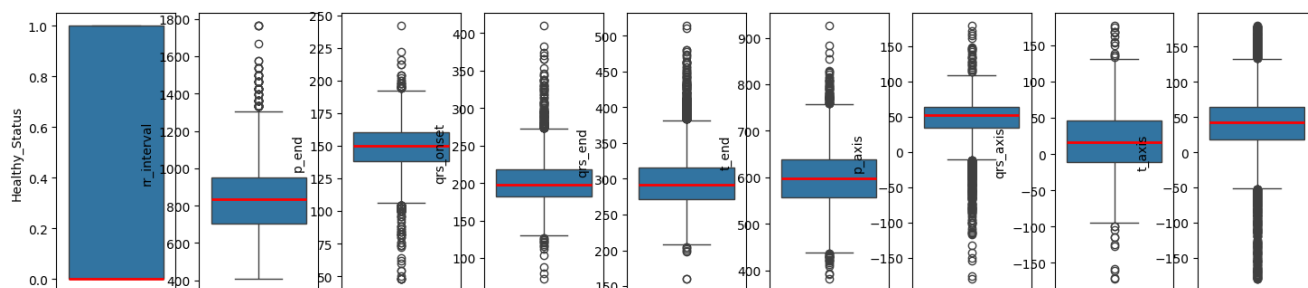


Рисунок 5 – boxplot графики на очищенных данных

После очистки можно заняться обработкой текстовой информации. Вариант, предложенный в эталонном не оптимален, так как при преобразовании текста в одно число теряется большая часть информации, в том числе и наиболее важная семантическая информация. Поэтому было принято решение построить полноценные эмбединги для текстов. Эмбединги будем строить, используя nlp-модель E5(EmbEddings from bidirectional Encoder rEpresentations). Выбор модели обусловлен рабочим опытом и ранее проведёнными нами исследованиями лучших SOTA решений в области NLP. Код генерации эмбедингов представлен ниже.

```
# Создаем список названий столбцов для отчетов
report_columns = [f'report_{x}' for x in range(18)]
# Объединяем значения в столбцах отчетов в одну строку
filtered_df['report'] = filtered_df[report_columns].astype(str).agg(' '.join, axis=1)
# Очищаем данные от NaN и лишних пробелов
filtered_df['report'] = (
    filtered_df['report']
    .str.replace(r'\bnan\b', '', regex=True)
    .str.replace(r'\s+', ' ', regex=True)
    .str.strip()
)
# Удаляем остальные столбцы с отчетами
reports_to_drop = [f'report_{x}' for x in range(18)]
filtered_df.drop(reports_to_drop, axis=1, inplace=True)

task = 'Find the most similar report for a given report'
tokenizer = AutoTokenizer.from_pretrained('intfloat/multilingual-e5-large-instruct')
model = AutoModel.from_pretrained('intfloat/multilingual-e5-large-instruct')

def get_detailed_instruct(task_description: str, query: str) -> str:
    return f'Инструкция: {task_description}\nЗапрос: {query}'

def text_normalization(text):
    """
    Нормализация текста - приведение к нижнему регистру,
```

```

удаление лишних символов.
"""
if not isinstance(text, str):
    return ""
text = text.lower()
text = re.sub(r'[\^\w\s]', ' ', text)
text = re.sub(' +', ' ', text)
text = text.strip()
return text

def average_pool(
    last_hidden_states: Tensor,
    attention_mask: Tensor
) -> Tensor:
    last_hidden = last_hidden_states.masked_fill(~attention_mask[..., None].bool(), 0.0)
    return last_hidden.sum(dim=1) / attention_mask.sum(dim=1)[..., None]

def emb_calc(text):
    model.eval()

    text = text_normalization(get_detailed_instruct(task, text))
    input_texts = [text]

    # Tokenize the input texts
    batch_dict = tokenizer(input_texts, max_length=512, padding=True, truncation=True,
return_tensors='pt')

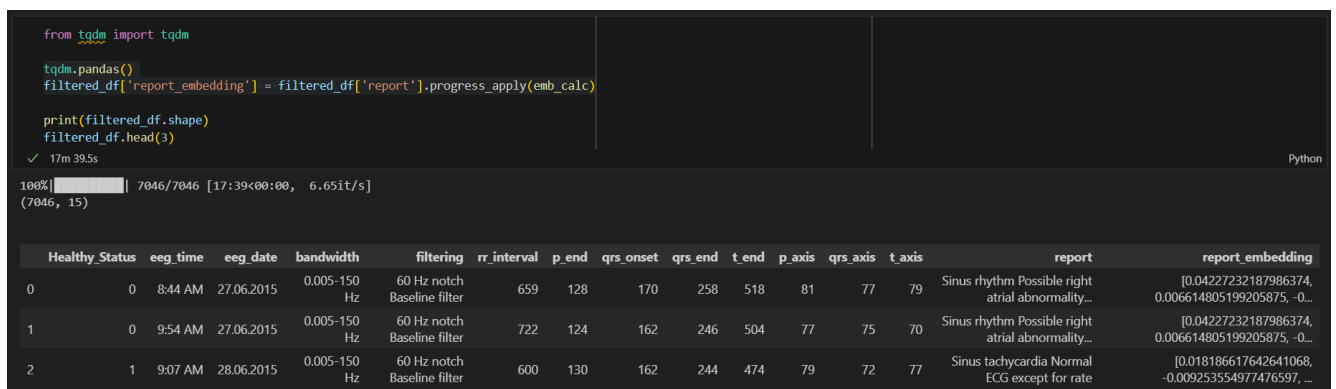
    outputs = model(**batch_dict)
    embeddings = average_pool(outputs.last_hidden_state, batch_dict['attention_mask'])
    # normalize embeddings
    embeddings = F_torch.normalize(embeddings, p=2, dim=1)

    return embeddings.flatten().tolist()

tqdm.pandas()
filtered_df['report_embedding'] = filtered_df['report'].progress_apply(emb_calc)

```

Результат работы данного кода представлен на рисунке 6.



```

from tqdm import tqdm

tqdm.pandas()
filtered_df['report_embedding'] = filtered_df['report'].progress_apply(emb_calc)

print(filtered_df.shape)
filtered_df.head(3)

```

✓ 17m 39.5s Python

100%|██████████| 7046/7046 [17:39<00:00, 6.65it/s]
(7046, 15)

	Healthy_Status	eeg_time	eeg_date	bandwidth	filtering	rr_interval	p_end	qrs_onset	qrs_end	t_end	p_axis	qrs_axis	t_axis	report	report_embedding
0	0	8:44 AM	27.06.2015	0.005-150 Hz	60 Hz notch Baseline filter	659	128	170	258	518	81	77	79	Sinus rhythm Possible right atrial abnormality...	[0.04227232187986374, 0.006614805199205875, -0...
1	0	9:54 AM	27.06.2015	0.005-150 Hz	60 Hz notch Baseline filter	722	124	162	246	504	77	75	70	Sinus rhythm Possible right atrial abnormality...	[0.04227232187986374, 0.006614805199205875, -0...
2	1	9:07 AM	28.06.2015	0.005-150 Hz	60 Hz notch Baseline filter	600	130	162	244	474	79	72	77	Sinus tachycardia Normal ECG except for rate	[0.018186617642641068, -0.009253554977476597, ...

Рисунок 6 – результат выполнения кода построения текстовых эмбедингов

В дальнейшем колонка эмбедингов разворачивается при помощи следующего кода:

```
df_expanded = filtered_df['report_embedding'].apply(pd.Series)
df_expanded.columns = [f'embedding_{i+1}' for i in df_expanded.columns]
df_expanded = pd.concat([filtered_df.drop('report_embedding', axis=1), df_expanded], axis=1)
```

Задание 4

Для оценки взаимосвязи данных были построены heatmap, scatter_matrix и pairplot графики, они приведены на рисунках 7-9.

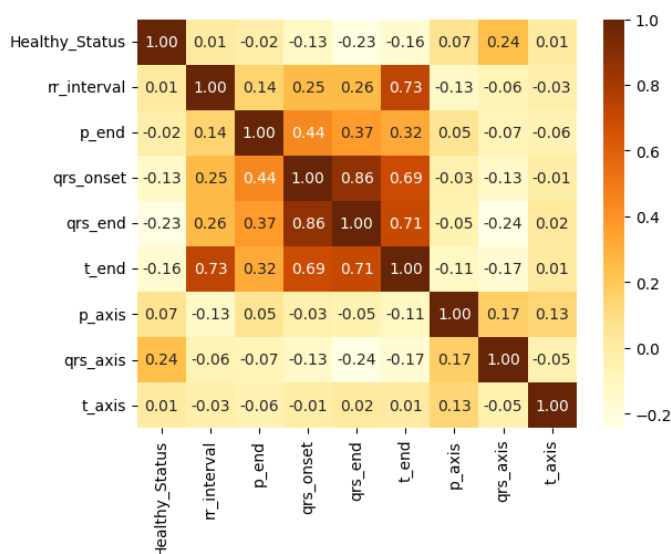


Рисунок 7 – heatmap таблица

Таблица анализа данных, коэффициент корреляции

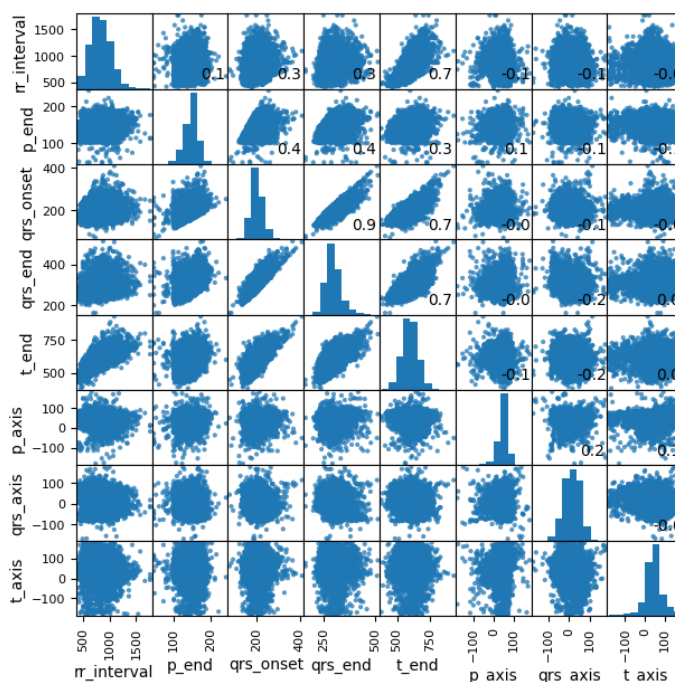


Рисунок 8 – scatter_matrix графики

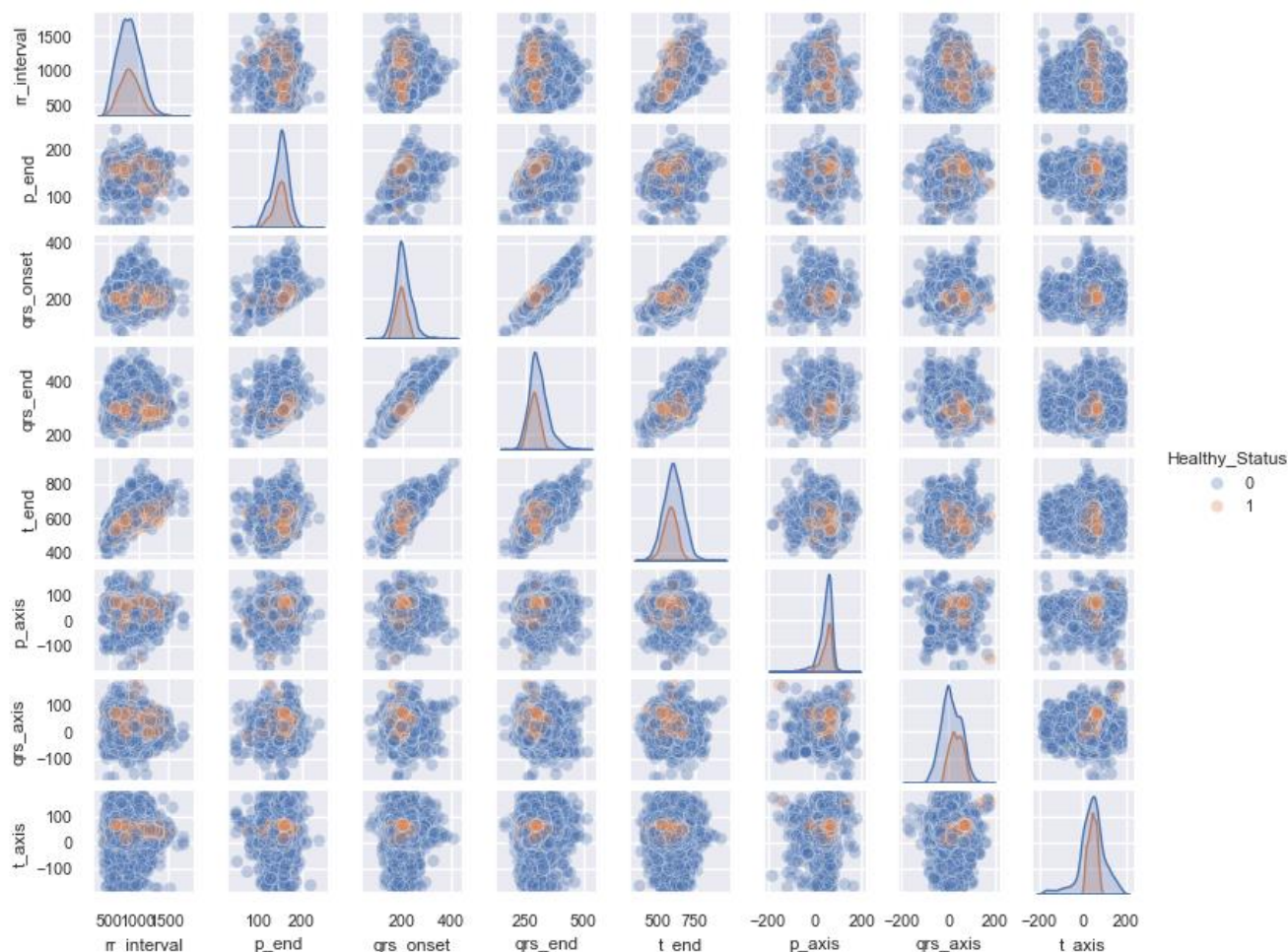


Рисунок 9 – pairplot графики

Из приведённых визуальных представлений данных можно сделать выводы, что имеется некоторая корреляция между фичами времени (в миллисекундах), однако явных корреляций, близких к 1, не обнаружено. Также, видно, что большая часть переменных имеет сильно перекрывающиеся распределения между классами, что говорит о том, что их будет сложно использовать по отдельности для хорошего разделения классов. Снова виден сильный дисбаланс классов.

Задание 5

Проведём разведочный анализ данных с помощью методов PCA и TSNE. Визуальные результаты работы кода показаны на рисунках 10-11.

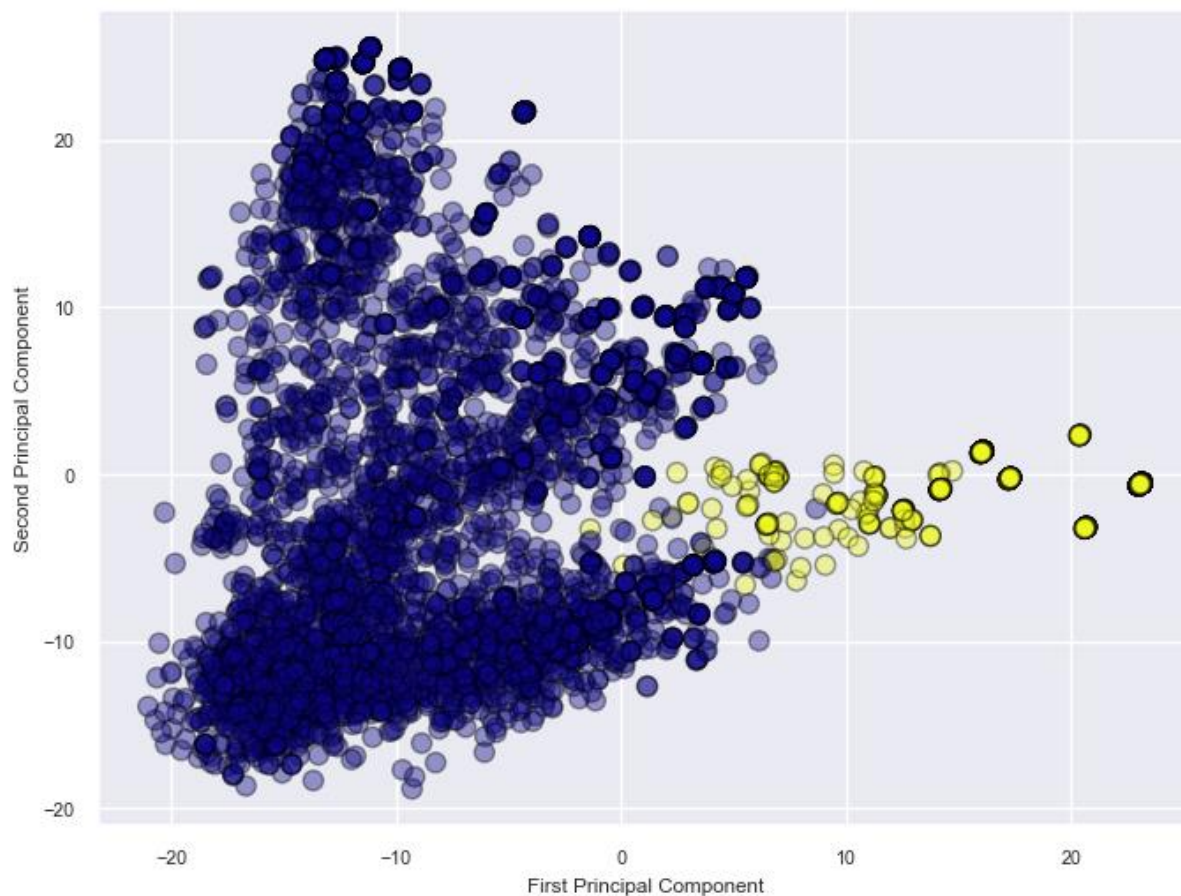


Рисунок 10 – результаты работы PCE

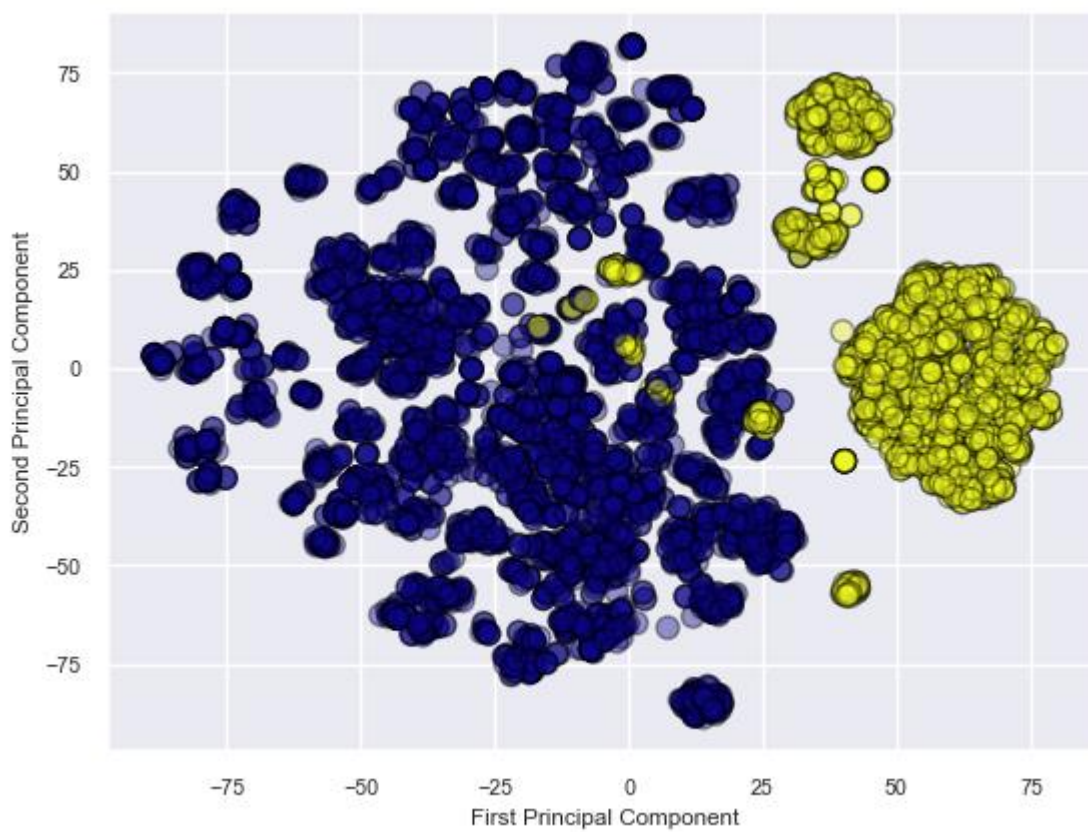


Рисунок 11 – результаты работы TSNE

Ранее, без текстовой информации, чёткого распределения кластеров не было видно, добавление фичей эмбединга дало хорошие результаты. Из рисунков выше, следует, что классы могут быть довольно легко разделены. И гораздо лучше, чем, в эталонном варианте!

Также, сложно не заметить, что разделение при TSNE прошло лучше, чем при PCA, что говорит, что данные имеют нелинейные зависимости, учитывание которых позволит лучше разделять классы.

Дополнительный анализ, представленный на рисунке 12, говорит, что:

- Локальная структура (мелкие кластеры) доминирует при малых perplexity.
- Глобальная структура (крупные группы) проявляется при больших perplexity.
- Оптимальный результат достигается в среднем диапазоне (15–30), где кластеры хорошо различимы.

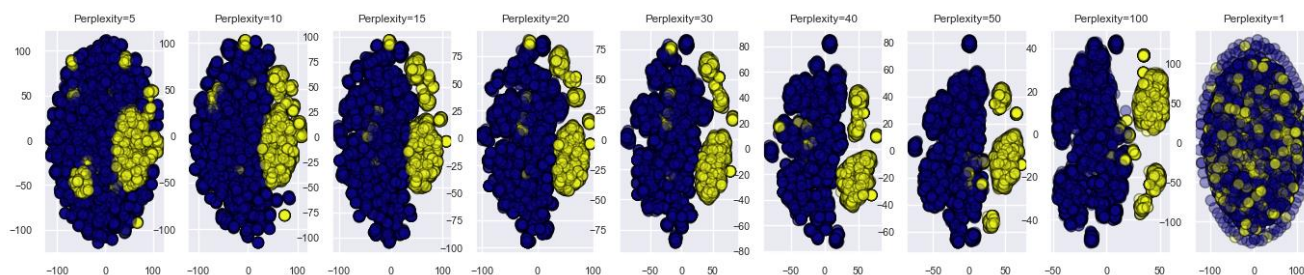


Рисунок 12 – результаты работы TSNE при разных perplexity

Из анализа выше ясно, что наибольший вклад дают эмбединги отчётов, поэтому можно безболезненно избавиться от всех других текстовых данных. Это также обеспечит более простую работу с autoML методами и методами обогащения датасета.

Задание 6

Для обогащения датасета был использован упомянутый ранее SMOTE метод, который используется для устранения дисбаланса классов. Ниже приведён код обогащения датасета

```
from imblearn.over_sampling import SMOTE

target_column = 'Healthy_Status'

smote = SMOTE(random_state=42)
X_expanded = df_expanded.drop(target_column, axis=1)
y_expanded = df_expanded[target_column]
```

```

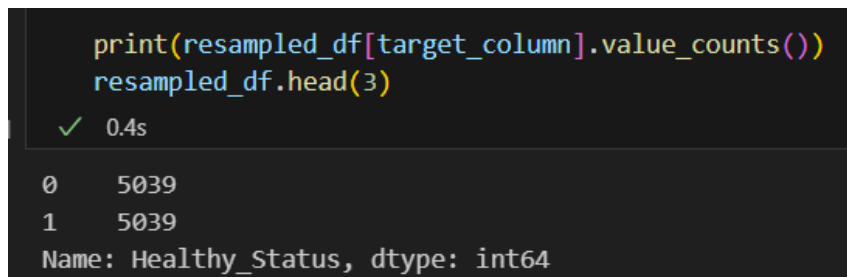
X_resampled, y_resampled = smote.fit_resample(X_expanded, y_expanded)

resampled_df = pd.DataFrame(X_resampled, columns=X_expanded.columns)
resampled_df[target_column] = y_resampled

print(resampled_df[target_column].value_counts())
resampled_df.head(3)

```

Результат обогащения показан на рисунке 13.



```

print(resampled_df[target_column].value_counts())
resampled_df.head(3)

```

✓ 0.4s

Healthy_Status	count
0	5039
1	5039

Name: Healthy_Status, dtype: int64

Рисунок 13 – результат обогащения датасета

Как видно, дисбаланс классов был успешно устранён путём увеличения малого класса за счёт выпуклых комбинаций экземпляров этого класса.

Задание 7

В рамках задания были использованы LightAutoML, FEDOT и TPOT.

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

```

target_column = 'Healthy_Status'

X = df_expanded.drop(target_column, axis=1)
y = df_expanded[target_column]

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

task = Task('multiclass')

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

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

oof_pred = automl.fit_predict(

```




Рисунок 15 – пайплайн работы с LightAutoML

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

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

    y_pred = automl_model.predict(features=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}')

except Exception as e:
    print(f'Произошла ошибка: {str(e)}')
    print('Попробуйте обновить Fedot или использовать другой пресет')
```

В результате была получена модель, показанная на рисунке 16.

```

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 16 – лучшая модель от FEDOT

Данная модель представляет из себя искусственную нейронную сеть, которая с высокой точностью выполняет задачу классификации. Accuracy: 0.9995, Precision: 0.9995, Recall: 0.9995.

Полный пайплайн работы с FEDOT показан на рисунке 17.



Рисунок 17 – 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.

```

tpot = TPOTClassifier(generations=4, population_size=10, random_state=42, verbosity=2,
n_jobs=-1, max_time_mins=30, cv=3)
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 18.

```
Тип: RandomForestClassifier
Параметры:
  bootstrap: True
  ccp_alpha: 0.0
  class_weight: None
  criterion: gini
  max_depth: None
  max_features: 0.2
  max_leaf_nodes: None
  max_samples: None
  min_impurity_decrease: 0.0
  min_samples_leaf: 8
  min_samples_split: 4
  min_weight_fraction_leaf: 0.0
  n_estimators: 100
  n_jobs: None
  oob_score: False
  random_state: 42
  verbose: 0
  warm_start: False
```

Рисунок 18 – лучшая модель от ТРОТ

Данная модель представляет из себя ансамбль случайных деревьев, который с высокой точностью задачу классификации. Accuracy: 0.9986, Precision: 0.9986, Recall: 0.9986.

Полный пайплайн работы с ТРОТ показан на рисунке 19.



Рисунок 19 – 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
Accuracy	1	0.9995	1
Precision	1	0.9995	1
Recall	1	0.9995	1

Поставленная задача является очень простой, поэтому, как видно в таблице 1, все autoML решения достигли высоких показателей метрик качества. LightAutoML имеет максимальные показатели, поэтому именно эта модель и будет использоваться далее.

Задание 8

В рамках задания улучшение модели не выполнялось (был взят пайплайн LightAutoML), так как модель и так добились максимального качества. Результаты работы модели на тестовой выборке показаны на рисунке 20.

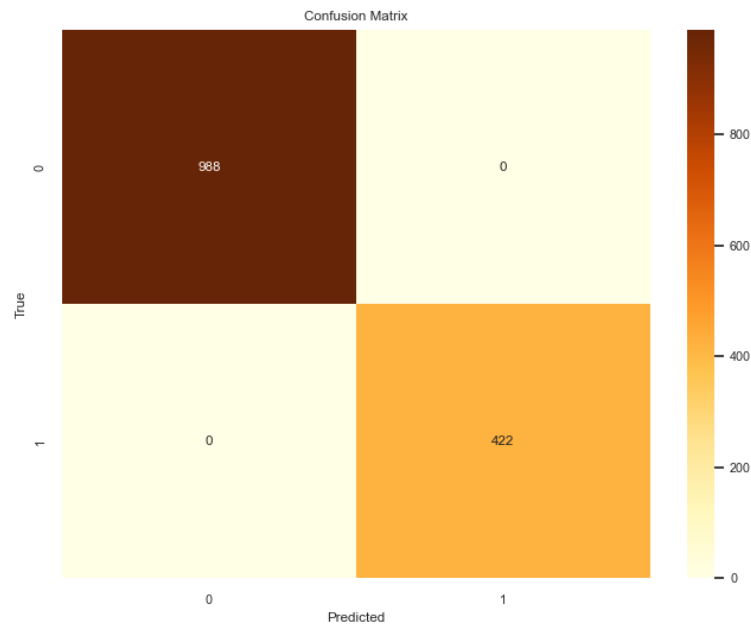


Рисунок 20 – визуализация работы модели на тестовой выборке

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.

```
mlflow.set_tracking_uri("http://127.0.0.1:8080")
mlflow.set_experiment("ECG_Classification") # Название эксперимента

params = {
    'task': 'multiclass',
    'cpu_limit': -1,
    'general_params': {'use_algos': 'auto'},
    'reader_params': {'cv': 3, 'random_state': 42}
}

roles = {
    'target': target_column,
    'drop': []
}

# Старт трекинга в MLflow
with mlflow.start_run():
    # Логирование параметров
    mlflow.log_params(params)
    mlflow.log_params(roles)
    # Инициализация и обучение модели
    automl = TabularAutoML(
        task=Task(params['task']),
        cpu_limit=params['cpu_limit'],
        general_params=params['general_params'],
        reader_params=params['reader_params']
```

```

)

train_data = pd.concat([X_train, y_train], axis=1)
oof_pred = automl.fit_predict(train_data, roles=roles, verbose=1)

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

# Расчет метрик
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({
    "accuracy": accuracy,
    "precision": precision,
    "recall": recall
})
clear_output(wait=False)
print(f'Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}')

# Сохранение модели в MLflow
mlflow.sklearn.log_model(automl, "lightautoml_model")
# Дополнительное сохранение в .pkl (опционально)
with open(f'{directory}/lightautoml_electrocardiogram_m2_model.pkl', 'wb') as f:
    pickle.dump(automl, f)
mlflow.log_artifact(f'{directory}/lightautoml_electrocardiogram_m2_model.pkl')

model_info = collect_model_info(loaded_automl)
mlflow.log_param("algorithm_type", model_info['algorithm_type'])
mlflow.log_param("model_type", model_info['model_type'])
mlflow.log_param("loss_function", model_info['loss_function'])

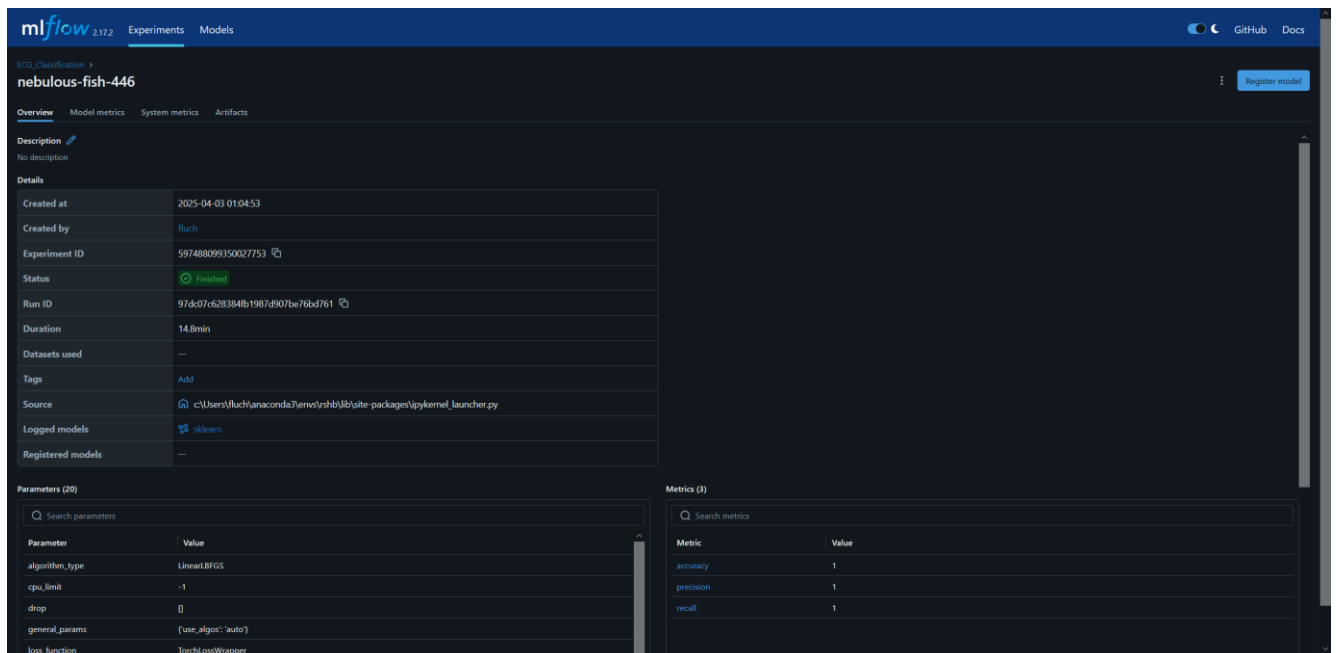
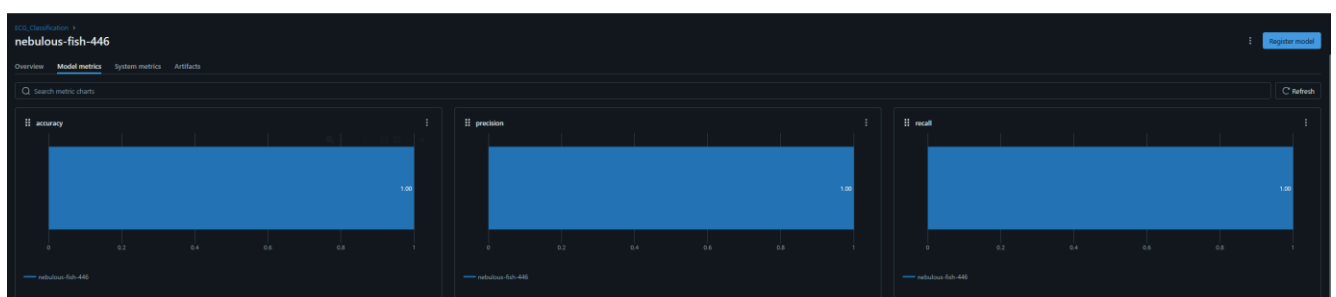
if model_info['architecture']:
    mlflow.log_text(model_info['architecture'], "model_architecture.txt")

for param, value in model_info['parameters'].items():
    mlflow.log_param(f"param_{param}", value)

if model_info['error']:
    mlflow.log_param("error", model_info['error'])
    mlflow.log_text(model_info['alternative_info'], "alternative_model_info.txt")

```

Результаты работы отображены на рисунках 21-24.

[illegible]

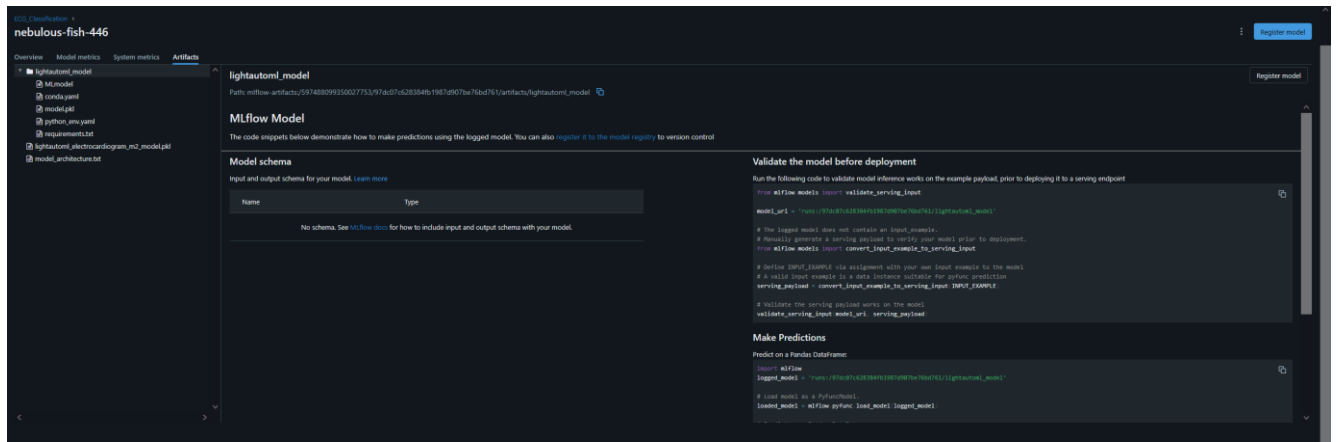


Рисунок 24 – страница артефактов, полученных при обучении модели

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

Вывод

В процессе выполнения лабораторной работы был собран, проанализирован (в том числе, при помощи методов PCA и TSNE), обогащён и сбалансирован датасет, а затем выполнено построение моделей машинного обучения при помощи autoML решений. Результирующая модель и процесс её обучения и инференса были загружены на MLFlow.