|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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.**

**Отчет**

**по лабораторной работе № 1-4**

**Название ЛР 1:** исследование возможностей использования генеративных помощников и их применения в платформах промышленной аналитики

**Название ЛР 2:** сбор и предобработка данных для модели машинного обучения

**Название ЛР 3:** построение модели машинного обучения на базе платформ промышленной аналитики

**Название ЛР 4:** доработка модели и оценка ее качества

**Дисциплина:** платформы промышленной аналитики

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-43М |  |  | В.Н. Зыкин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | М.А.Скворцова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

**Лабораторная работа 1**

**Цель:** ознакомиться с возможностями работы чатов/генеративных помощников на базе LLM для применения в исследовательской деятельности студентов и решения профессиональных задач.

Для исследования были выбраны следующие генеративные помощники: СhatGPT-4o, DeepSeek, GigaChat и YandexGPT.

**Задание 1:** необходимо задать вопросы по вашей тематике магистерской диссертации и проанализировать ответы чат-ботов для актуальности и корректности применения данных ответов в ВКРМ (должно быть сформулировано и проверено не менее 5 гипотез по каждому генеративному помощнику).

Были выдвинуты следующие гипотезы:

*Гипотеза 1.* Для оптимизации нейронных сетей под мобильные устройства можно использовать следующие методы: квантование, прунинг, кластеризация, использование специальных фреймворков, использование облегчённых архитектур, сжатие, дистилляция знаний.

Вопрос: какие есть способы оптимизации нейронных сетей под мобильные устройства?

*Гипотеза 2.* Квантование нейронных сетей может быть постобучающим, с учетом обучения, динамическим и гибридным.

Вопрос: какие есть основные способы квантования нейронных сетей?

*Гипотеза 3*. Tensorflow предоставляет больше всего инструментов для оптимизации нейронных сетей под мобильные устройства.

Вопрос: Какой фреймворк предоставляет больше всего инструментов для оптимизации и инференса нейронных сетей на мобильных устройствах?

*Гипотеза 4.* Для обучения моделей семейства YOLO проще всего использовать библиотеку Ultralytics.

Вопрос: Какой фреймворк лучше использовать для обучения модели YOLOv8?

*Гипотеза 5.* Модели сеймества YOLO чаще всего используются для задач детекции.

Вопрос: для каких задач чаще всего используются модели семейства YOLO?

ChatGPT

Ответ ИИ на гипотезу 1.

Оптимизация нейронных сетей под мобильные устройства — важная задача, так как такие устройства ограничены по ресурсам (CPU, GPU, память, батарея).

Чтобы нейросеть работала на мобильном, её обычно упрощают. Самый популярный способ — квантизация: заменяют 32-битные числа на 8-битные. Это сильно ускоряет работу и уменьшает размер модели. Ещё модели «обрезают» — убирают неважные веса (прунинг). Иногда берут большую модель и обучают маленькую повторять её поведение — это называется дистилляция знаний.

Также есть специальные лёгкие архитектуры вроде MobileNet или EfficientNet-Lite — они изначально сделаны для телефонов. Чтобы всё это работало быстрее, используют фреймворки вроде TensorFlow Lite, Core ML или ONNX, которые умеют превращать модель в формат, удобный для смартфона. Иногда модель сжимают ещё больше с помощью матричных разложений или деления весов на общие группы. И, наконец, всё это хорошо работает только при наличии аппаратного ускорения — нейросети используют GPU, NPU или DSP, которые есть в современных телефонах.Начало формы

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 2.

Квантование нейронных сетей — это техника оптимизации, при которой значения весов и активаций сжимаются с помощью уменьшения разрядности (битности) представления чисел. Это позволяет сократить размер модели и ускорить её выполнение, особенно на устройствах с ограниченными ресурсами (мобильные устройства, edge-устройства и т.п.).

Вот основные способы квантования:

1. Пост-тренировочное квантование (Post-Training Quantization, PTQ)

Квантование применяется к уже обученной модели без дообучения.

Квантование весов – веса переводятся из 32-битных float в 8-битные int (INT8).

Квантование весов и активаций – и веса, и выходы слоёв преобразуются в INT8.

Преимущества:

Простота: не нужно заново обучать модель.

Быстрое применение.

Недостатки:

Возможна потеря точности (особенно у чувствительных моделей).

2. Квантование с дообучением (Quantization-Aware Training, QAT)

Модель обучается или дообучается с учётом будущего квантования.

Во время обучения симулируется эффект квантования.

Позволяет сохранить точность близкую к оригинальной модели.

Используется, когда PTQ даёт слишком большую деградацию качества.

3. Динамическое квантование (Dynamic Quantization)

Квантуются веса до INT8, но активации квантуются на лету во время инференса.

Применимо, например, к трансформерам (BERT, GPT) и RNN.

Хороший компромисс между скоростью и точностью.

Используется в PyTorch, например: torch.quantization.quantize\_dynamic.

4. Гибридное квантование (Hybrid Quantization)

Комбинируются разные типы квантования: часть слоёв квантована, часть остаётся в float32.

Пример: веса в INT8, активации – в float16 или float32.

Используется для балансировки между производительностью и точностью.

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 3.

На сегодняшний день наиболее полный и мощный фреймворк для оптимизации и инференса нейронных сетей на мобильных устройствах — это:

TensorFlow Lite (TFLite)

Почему именно он:

Поддержка широкого набора оптимизаций:

Квантование (Post-Training, QAT)

Pruning, Clustering

Delegate'ы (GPU, NNAPI, Hexagon, CoreML)

Поддержка Android и iOS

Совместим с моделями TensorFlow / Keras

Простой экспорт и конвертация

Есть TensorFlow Model Optimization Toolkit

Инструменты профилирования и анализа производительности

Лёгкий запуск на устройствах (C++, Java, Swift, Python)Начало формы

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 4.

Проще всего обучить модель семейства YOLO (например, YOLOv5, YOLOv8) с использованием готовых тулов, где уже всё "упаковано" — загрузка данных, аугментации, обучение, инференс и т.д.

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 5. Начало формы

Модели семейства YOLO чаще всего используются для задач обнаружения объектов (object detection).

Основные задачи:

* Обнаружение объектов на изображениях и видео (люди, машины, животные и т.д.)
* Системы видеонаблюдения и безопасности
* Аналитика в ритейле (подсчёт людей, товаров на полках)
* Робототехника и автономные системы
* Медицинская визуализация (обнаружение аномалий)
* Анализ изображений в агро- и индустрии (напр., распознавание дефектов)

Анализ: **ответ соответствует гипотезе**.

YandexGpt

Ответ ИИ на гипотезу 1.

Для оптимизации нейронных сетей под мобильные устройства можно использовать следующие методы:

1. Квантование весов и активаций:
   * Уменьшение точности вычислений (например, использование 16-битных чисел с плавающей точкой вместо 32-битных) позволяет сократить объём занимаемой памяти и ускорить вычисления.
2. Сжатие модели:
   * Применение алгоритмов сжатия, таких как Huffman coding или LZW, для уменьшения размера модели без значительной потери точности.
   * Использование методов низкорангового разложения матриц (например, Singular Value Decomposition, SVD) для сжатия весовых матриц.
3. Архитектура модели:
   * Выбор более простых архитектур, таких как MobileNet, SqueezeNet или ShuffleNet, которые специально разработаны для работы на мобильных устройствах.
   * Использование глубинных separable свёрток, которые значительно уменьшают количество параметров и вычислений по сравнению с обычными свёртками.
4. Оптимизация вычислений:
   * Применение библиотек и фреймворков, оптимизированных для мобильных устройств, таких как TensorFlow Lite, PyTorch Mobile или Core ML.
   * Использование аппаратных ускорителей, таких как GPU или DSP, для ускорения вычислений.

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 2.

Основные методы квантования нейронных сетей:

1. Квантование с фиксированной точкой (Fixed-point quantization) — преобразование весов и активаций из чисел с плавающей точкой в числа с фиксированной точкой для уменьшения размера модели и ускорения вычислений.
2. Квантование с обучением (Quantization-Aware Training, QAT) — учёт ограничений, связанных с квантованием, в процессе обучения модели для минимизации потерь точности.
3. Посттренировочное квантование (Post-Training Quantization, PTQ) — квантование модели после завершения обучения без дополнительного обучения.
4. Динамическое квантование (Dynamic Quantization) — адаптация уровня квантования в зависимости от текущих входных данных для оптимизации баланса между точностью и эффективностью.
5. Симметричное и асимметричное квантование:
   * Симметричное квантование использует одинаковый масштабный коэффициент для всех весов или активаций, упрощая вычисления.
   * Асимметричное квантование позволяет использовать разные масштабные коэффициенты для разных частей модели, что может улучшить точность.
6. Квантование с перекалибровкой (Quantization with Recalibration) — дополнительный этап перекалибровки модели после квантования для корректировки ошибок и улучшения точности.
7. Квантование с учётом ошибок (Error-Aware Quantization) — учёт ошибок, возникающих из-за уменьшения точности представления данных при квантовании, для минимизации их влияния на общую точность модели.

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 3.

Одним из наиболее популярных фреймворков, предоставляющих широкий набор инструментов для оптимизации и инференса нейронных сетей на мобильных устройствах, является TensorFlow Lite. Он разработан специально для развёртывания моделей машинного обучения на мобильных и встраиваемых устройствах и предлагает различные инструменты и техники для оптимизации моделей, включая квантование, обрезку и сжатие моделей.

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 4:

Для обучения модели семейства YOLO (You Only Look Once) можно следовать следующим шагам:

1. Выбор версии YOLO.
2. Подготовка данных.
3. Настройка конфигурации.
4. Использование готовых решений.
5. Запуск обучения.
6. Оценка и тестирование.
7. Доработка и оптимизация.
8. Использование предобученных моделей

Следуя этим шагам, вы сможете обучить модель YOLO для задачи обнаружения объектов. Важно помнить, что процесс обучения может потребовать значительных вычислительных ресурсов и времени, особенно при работе с большими наборами данных и сложными моделями.

Анализ: **ответ не соответствует гипотезе.**

Ответ ИИ на гипотезу 5.

Модели семейства YOLO чаще всего используются для задачи обнаружения объектов (Object Detection) — определения местоположения и классов различных объектов на изображениях или видео.

Анализ: **ответ соответствует гипотезе**.

GigaChat

Ответ ИИ на гипотезу 1.

Оптимизация нейронной сети для мобильных устройств — важная задача, особенно учитывая ограниченность ресурсов таких устройств, таких как память, вычислительная мощность и энергопотребление. Вот несколько ключевых способов сделать это:

1. Quantization

Quantization — это процесс уменьшения битности весовых коэффициентов модели (например, с 32 до 8 бит). Это значительно уменьшает размер модели и ускоряет вычисления, поскольку меньшие значения требуют меньше памяти и быстрее обрабатываются.

Существует несколько подходов:Post-training quantization: Применяется после тренировки модели, когда веса преобразуются в низкобитный формат.Quantization-aware training: Модель тренируется сразу с учетом будущей квантизации, что позволяет избежать ухудшения точности.

2. Pruning (Обрезка)

Pruning удаляет избыточные параметры модели, такие как неважные веса или целые слои. Существуют разные методы обрезки:Magnitude-based pruning: Удаляются веса с наименьшими значениями.Structured pruning: Обрезаются целые фильтры или каналы.

После обрезки часто применяется ретренировка для восстановления потерянной точности.

3. Knowledge Distillation (Дистилляция знаний)

Knowledge distillation используется для переноса знаний от большой модели (учителя) к маленькой модели (ученику). Ученик учится предсказывать выход учителя, а также исходные метки, что помогает сохранить точность даже при уменьшении размера модели.

4. Использование легких архитектур

Выбор архитектуры с минимальными требованиями к ресурсам — один из самых простых способов оптимизации. Например:MobileNet: Семейство моделей, разработанных специально для мобильных устройств. Они используют depthwise separable convolutions, что снижает количество операций.EfficientNet: Оптимизированная архитектура, основанная на MobileNet, которая сочетает глубину, ширину и разрешение входного изображения для достижения высокой эффективности.ShuffleNet: Использует pointwise group convolutions и shuffle operations для снижения вычислительной сложности.

5. Компрессия модели

Помимо квантизации и обрезки, существуют другие методы компрессии:Huffman coding: Сжатие весов модели с использованием кодирования Хаффмана.Sparse matrix representation: Представление весов в виде разреженных матриц, где сохраняются только ненулевые элементы.

6. Hardware acceleration

Использование аппаратных ускорителей, таких как GPU или специализированные ASIC/TPU чипы, может значительно ускорить выполнение модели на мобильном устройстве. Некоторые библиотеки, такие как TensorFlow Lite, поддерживают оптимизацию под конкретную архитектуру процессора.

7. Разделение вычислений между устройством и облаком

В некоторых случаях часть вычислений можно перенести на облачные серверы, оставляя на устройстве только обработку данных низкого уровня. Это снижает нагрузку на мобильное устройство, хотя требует постоянного подключения к интернету.

8. Минимизация ввода-вывода

Для ускорения работы модели важно минимизировать операции чтения-записи в память. Например, использование буферов и кеширование промежуточных результатов может существенно улучшить производительность.

9. Удаление ненужных слоев

Иногда можно удалить некоторые части модели, которые не критичны для конечного результата. Например, можно убрать избыточные полносвязные слои или уменьшить количество фильтров в свёрточных слоях.

Заключение

Выбор конкретного метода зависит от требований к производительности, размеру модели и допустимому уровню потери точности. Часто комбинирование нескольких методов даёт наилучший результат.

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 2.

Основные типы квантования нейронных сетей: Пост-тренировочное квантование — применяется после тренировки модели, веса и активации преобразуются в низкоразрядный формат (например, 8 бит). Быстрота внедрения, сохранение точности. Квантование с учётом обучения — интегрировано в процесс тренировки, учитывается влияние квантования на точность. Более точное, но требует большего времени на тренировку.

Типы квантования: Uniform (равномерное) Non-uniform (неравномерное) Fixed-point (с фиксированной точкой) Dynamic range (динамический диапазон)

Примеры библиотек: TensorFlow Lite, PyTorch, ONNX Runtime.

Анализ: **ответ не соответствует гипотезе.**

Ответ ИИ на гипотезу 3.

Краткий ответ: TensorFlow Lite. Этот фреймворк предлагает широкий набор инструментов для оптимизации и инференса нейронных сетей на мобильных устройствах, включая квантование, сжатие моделей и поддержку различных аппаратных ускорений.

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 4.

Для обучения модели семейства YOLOv8 рекомендуется использовать YOLOv8 (или YOLOv8s) от компании Roboflow. Это специализированный фреймворк, основанный на PyTorch, который разработан именно для работы с моделями YOLO последней версии. Он включает в себя все необходимые инструменты для подготовки данных, обучения и оценки моделей, а также имеет удобную документацию и сообщество поддержки.

Преимущества YOLOv8:Простая установка и настройка.Поддержка различных версий YOLO (включая v8).Гибкая система конфигурирования и управления обучением.Возможность интеграции с другими инструментами машинного обучения.

Таким образом, YOLOv8 является оптимальным выбором для обучения моделей данного семейства

Анализ: **ответ не соответствует гипотезе.**

Ответ ИИ на гипотезу 5.

Модели семейства YOLO (You Only Look Once) чаще всего используются для решения задач объектной детекции в режиме реального времени. Эти модели предназначены для обнаружения и классификации объектов на изображениях или видео, определяя их местоположение с помощью bounding box'ов (ограничивающих рамок).

Основные области применения YOLO включают:Автоматическое вождение: Детекция автомобилей, пешеходов, дорожных знаков и других объектов на дороге.Безопасность и видеонаблюдение: Определение людей, транспортных средств и подозрительных действий в системах видеонаблюдения.Робототехника: Ориентация роботов в пространстве и взаимодействие с окружающими объектами.Медицина: Анализ медицинских изображений, таких как рентгенограммы или МРТ-снимки, для выявления патологий.Розничная торговля: Подсчет товаров на полках, контроль за запасами и автоматизация кассовых процессов.Производство: Контроль качества продукции, обнаружение дефектов и автоматизация производственных линий.

Благодаря своей быстроте и точности, модели YOLO широко применяются там, где требуется оперативная обработка больших объемов визуальной информации.

Анализ: **ответ соответствует гипотезе**.

DeepSeek

Ответ ИИ на гипотезу 1.

Оптимизация нейронных сетей для мобильных устройств является важной задачей, так как мобильные устройства имеют ограниченные вычислительные ресурсы, память и энергопотребление. Вот основные способы оптимизации нейронных сетей для таких устройств:

### 1. Сжатие модели

* Квантование (Quantization)**:**
* Обрезка (Pruning)**:**
* Дистилляция (Knowledge Distillation)**:**

### 2. Использование легковесных архитектур

### 3. Оптимизация вычислений

* Использование специализированных библиотек
* Использование аппаратного ускорения
* Оптимизация операций

Анализ: **ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 2.

Квантование (quantization) — это процесс уменьшения точности представления чисел в нейронной сети, что позволяет сократить размер модели, ускорить её выполнение и снизить энергопотребление. Это особенно важно для мобильных устройств и embedded-систем, где ресурсы ограничены. Основные способы квантования нейронных сетей включают:

1. Посттренировочное квантование (Post-Training Quantization, PTQ)

Описание:

Модель квантуется после завершения обучения. Веса и активации преобразуются из 32-битных чисел с плавающей запятой (float32) в 8-битные целые числа (int8) или другие форматы.

Преимущества:

Простота реализации, не требует повторного обучения модели.

Недостатки:

Может привести к потере точности, особенно если модель не была оптимизирована для квантования.

Типы:

Только веса: Квантуются только веса, а вычисления выполняются в float32.

Веса и активации: Квантуются как веса, так и активации, что позволяет ускорить выполнение операций.

Примеры:

TensorFlow Lite, PyTorch, ONNX Runtime поддерживают посттренировочное квантование.

2. Квантование с учетом обучения (Quantization Aware Training, QAT)

Описание:

Модель обучается с учетом будущего квантования. В процессе обучения имитируется квантование, что позволяет модели адаптироваться к потере точности.

Преимущества:

Сохраняет высокую точность после квантования.

Недостатки:

Требует дополнительного времени на обучение.

Примеры:

TensorFlow (tf.quantization), PyTorch (torch.quantization).

3. Динамическое квантование (Dynamic Quantization)

Описание:

Веса квантуются заранее, а активации квантуются динамически во время выполнения. Это позволяет сохранить гибкость для разных входных данных.

Преимущества:

Уменьшает размер модели и ускоряет выполнение.

Недостатки:

Менее эффективно, чем полное квантование весов и активаций.

Примеры:

PyTorch поддерживает динамическое квантование.

4. Статическое квантование (Static Quantization)

Описание:

Веса и активации квантуются заранее, на этапе подготовки модели. Для активаций используются заранее вычисленные диапазоны значений.

Преимущества:

Максимальное ускорение и уменьшение размера модели.

Недостатки:

Требует калибровки для определения диапазонов активаций.

Примеры:

TensorFlow Lite, PyTorch.

5. Гибридное квантование (Hybrid Quantization)

Описание:

Часть модели квантуется (например, сверточные слои), а другая часть остается в float32 (например, полносвязные слои).

Преимущества:

Позволяет сохранить точность в критически важных частях модели.

Недостатки:

Меньшая оптимизация по сравнению с полным квантованием.

Примеры:

TensorFlow Lite.

**Анализ: ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 3.

Если говорить о одном фреймворке, который предоставляет наибольшее количество инструментов для оптимизации и инференса нейронных сетей на Android, то это TensorFlow Lite. Он является наиболее полным и зрелым решением для мобильных устройств, особенно для Android, благодаря своей интеграции с экосистемой TensorFlow и поддержке множества аппаратных ускорителей.

**Анализ: ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 4.

Для обучения модели YOLOv8 лучше всего использовать фреймворк Ultralytics, так как YOLOv8 был разработан именно этой командой и максимально оптимизирован для работы с их инструментами. Ultralytics предоставляет простой и мощный API, который значительно упрощает процесс обучения, валидации и развертывания моделей YOLO.

**Анализ: ответ соответствует гипотезе.**

Ответ ИИ на гипотезу 5.

Модели семейства YOLO (You Only Look Once) чаще всего используются для задачи обнаружения объектов (Object Detection). Это одна из ключевых задач компьютерного зрения, которая заключается в локализации и классификации объектов на изображении или видео. YOLO особенно популярен благодаря своей скорости и точности, что делает его идеальным выбором для реального времени.

**Анализ: ответ соответствует гипотезе.**

**Задание 2:** сформировать аннотацию/введение для своей магистерской диссертации с помощью генеративных помощников. Определить какой из них справляется с этой задачей лучше.

Попытка ChatGPT

Выпускная квалификационная работа посвящена разработке программной системы визуального осмотра транспортного средства, предназначенной для упрощения, автоматизации и повышения достоверности процесса фиксации состояния автомобилей. Актуальность темы обусловлена растущими требованиями к цифровизации процессов в автомобильной, страховой и логистической сферах, где критически важным становится оперативный и объективный осмотр транспортных средств с возможностью последующего анализа и документирования.

Основная цель работы — создание инструмента, который позволяет пользователю с помощью мобильного устройства проводить осмотры автомобилей, фиксируя их текущее визуальное состояние. В процессе осмотра осуществляется фото- и видеосъёмка, на основании которой система извлекает ключевую информацию о транспортном средстве. Полученные материалы и данные автоматически сохраняются и используются для формирования отчётов, доступных как пользователям, так и специалистам, принимающим решения на их основе.

Разрабатываемая система ориентирована на практическое применение в таких сферах, как страхование (например, при оформлении страховых случаев или расчёте стоимости полиса), контроль состояния автопарков, приёмка и возврат автомобилей в сервисах аренды и каршеринга, а также в автосервисах и при предпродажной подготовке. Использование современных технологий обработки изображений и автоматического распознавания объектов позволяет сократить время осмотра, снизить влияние человеческого фактора и повысить объективность оценки.

Таким образом, разработанная программная система не только упрощает проведение осмотров, но и создаёт надёжную цифровую основу для хранения, анализа и документирования результатов, что делает её эффективным инструментом в современных бизнес-процессах, связанных с эксплуатацией автотранспорта.

Попытка YandexGpt

Дипломная работа посвящена разработке программной системы для визуального осмотра транспортных средств. Система представляет собой комплексное решение, включающее Android-приложение для проведения осмотров, серверную часть с базой данных и хранилищем MinIO для хранения фото- и видеоматериалов, а также сервис для формирования отчётов в формате PDF.

Основная цель работы — создание эффективного инструмента для визуального контроля состояния автомобилей, который позволит автоматизировать процесс сбора и анализа данных. В Android-приложении пользователь может проводить осмотры, делая снимки и видеозаписи. Во время съёмки задействуются нейронные сети, которые анализируют полученные данные и извлекают важную информацию об автомобиле.

Разработанная система предназначена для использования в автомобильной отрасли, включая сервисы технического обслуживания, страховые компании и другие организации, которым требуется быстрый и точный визуальный контроль транспортных средств. Она позволяет упростить процесс осмотра, сократить время на анализ данных и повысить точность результатов. Это способствует улучшению качества обслуживания клиентов и оптимизации рабочих процессов.

Попытка GigaChat

Выпускная квалификационная работа посвящена созданию программной системы для визуального осмотра транспортных средств. Система разработана для повышения эффективности и автоматизации процесса осмотра автомобилей, обеспечивая удобный интерфейс для пользователей. Основой системы является мобильное приложение на платформе Android, позволяющее проводить осмотр транспортных средств путём фотографирования и видеосъёмки. Во время съемки задействованы нейронные сети, обеспечивающие получение важных данных об автомобиле в реальном времени. Серверная часть системы включает базу данных и хранилище Minio для безопасного хранения фотографий и видеозаписей осмотров. Дополнительно предусмотрен сервис формирования отчётов в формате PDF, содержащих результаты проведённых осмотров. Система направлена на улучшение точности осмотров, сокращение временных затрат и повышение удобства работы пользователей.

**Попытка DeepSeek**

Дипломная работа посвящена разработке программной системы визуального осмотра транспортных средств, предназначенной для автоматизации и упрощения процесса диагностики автомобилей. Система представляет собой комплексное решение, состоящее из мобильного приложения на платформе Android, серверной части и облачного хранилища данных. Основная цель работы — создание инструмента, который позволяет пользователям проводить осмотры транспортных средств с использованием современных технологий компьютерного зрения и нейронных сетей, а также автоматически формировать отчёты о результатах осмотра.

Мобильное приложение является ключевым компонентом системы, предоставляя пользователю интуитивно понятный интерфейс для создания и проведения осмотров. В процессе съёмки транспортного средства задействуются нейронные сети, которые анализируют изображения и видео в реальном времени, извлекая данные о состоянии автомобиля, такие как наличие повреждений, дефектов или других особенностей. Это позволяет значительно ускорить процесс осмотра и повысить его точность за счёт автоматизированного анализа.

Серверная часть системы обеспечивает хранение и обработку данных. Для хранения информации о проведённых осмотрах используется база данных, а для хранения медиафайлов (фотографий и видео) — облачное хранилище MinIO. Это позволяет эффективно управлять большими объёмами данных и обеспечивает их доступность для дальнейшего анализа. Дополнительно реализован сервис формирования PDF-отчётов, который автоматически генерирует документы с результатами осмотра, что упрощает документирование и передачу информации заинтересованным сторонам.

Разработанная система имеет широкий спектр применения. Она может быть использована в автосервисах, страховых компаниях, а также при продаже подержанных автомобилей для объективной оценки их состояния. Внедрение такой системы позволяет минимизировать человеческий фактор, снизить время проведения осмотров и повысить качество диагностики.

В рамках дипломной работы были решены задачи проектирования архитектуры системы, разработки мобильного приложения, интеграции нейронных сетей для анализа изображений, настройки серверной части и облачного хранилища, а также создания сервиса формирования отчётов. Результатом работы стала полнофункциональная система, готовая к использованию в реальных условиях.

Данная работа вносит вклад в развитие технологий автоматизации процессов диагностики транспортных средств и демонстрирует возможности применения нейронных сетей и облачных технологий для решения практических задач в области автомобильной индустрии.

**Вывод к заданию 2:** лучше всех справились СhatGPT и DeepSeek, так как сформировали наиболее объёмный и содержательный текст.

**Задание 3:** описать достоинства и недостатки применения данного инструмента для профессиональной деятельности.

Использование генеративных помощников очень сильно упрощает поиск большого объёма информации. Зачастую они формируют достаточно структурированный и ёмкий ответ. Кроме того, они помогают в решение достаточно типовых проблем, что также сильно сокращает время. Однако если задачи касаются достаточно узких областей, то ИИ могут формировать странные и далёкие от реальности ответы, и приходится очень много времени тратить на написание понятного ИИ промта, что может быть дольше, чем самому решить проблему. В целом генеративные ИИ в несколько раз упрощают и ускоряют работу, однако к его ответы лучше дополнительно анализировать. Кроме того, некоторые модели страдают одной проблемой: склонность оформлять текст в виде огромного количества вложенных списков, что затрудняет использование их ответов в отчётах и работах. Для решения этой проблемы иногда приходится напрямую просить их оформить текст в как можно более сплошном виде.

**Задание 4:** приложить несколько запросов и ответов по использованию генеративных помощников по использованию в конструкторской/технологической части ВКРМ, при формировании списка литературы.

**Запрос**: я собираюсь использовать Kotlin при разработке приложения. Кратко одним абзацем опиши данный инструмент и почему он был выбран.

**ChatGPT**

Kotlin — это современный, статически типизированный язык программирования, разработанный JetBrains и полностью совместимый с Java, что делает его отличным выбором для разработки Android-приложений. Он предлагает более лаконичный, безопасный и выразительный синтаксис по сравнению с Java, снижает количество шаблонного кода и уменьшает вероятность ошибок, связанных с null-значениями благодаря встроенной системе null-безопасности. Kotlin активно поддерживается Google и стал официальным языком разработки под Android, что делает его актуальным, удобным и перспективным инструментом для мобильной разработки.

**YandexGpt**

Kotlin — это современный язык программирования, который работает на платформе Java и хорошо подходит для разработки приложений под Android. Он выбран за свою лаконичность, безопасность типов, поддержку функционального программирования и возможность полной интеграции с существующим Java-кодом, что упрощает процесс разработки и делает код более читаемым и поддерживаемым.

**GigaChat**

Kotlin — это современный статически типизированный язык программирования от JetBrains, который особенно популярен среди Android-разработчиков благодаря своей лаконичности, безопасности и совместимости с Java. Он позволяет писать чистый и понятный код, снижает вероятность ошибок типа NullPointerException благодаря системе типов, поддерживает многоплатформенную разработку и легко интегрируется с существующим Java-кодом. Выбор Kotlin обусловлен его производительностью, поддержкой от Google как основного языка для разработки под Android, а также активным сообществом разработчиков и богатой экосистемой библиотек.

**DeepSeek**

Kotlin — современный, статически типизированный язык программирования, работающий на JVM и полностью совместимый с Java, что делает его идеальным для Android-разработки и кроссплатформенных проектов. Его ключевые преимущества включают лаконичный синтаксис, сокращающий объем кода, встроенную защиту от ошибок (например, null-безопасность), поддержку функционального программирования и расширенные возможности для многопоточности. Kotlin был выбран благодаря официальной поддержке Google для Android, высокой производительности, активному сообществу, а также возможности интеграции с существующим Java-кодом, что ускоряет разработку и упрощает поддержку приложений.

**Запрос**: я использую следующие средства, инструменты, архитектуры разработки клиентской части: Model-View-ViewModel, Kotlin, Jetpack Compose, CameraX, Room, TensorFlow Lite, Gradle, Android Studio. Составь мне по ним список литературы по ГОСТ (для каждого по одному источнику).

**ChatGPT**

1. Госсман, Дж. Model-View-ViewModel (MVVM) [Электронный ресурс] // Википедия. URL: https://ru.wikipedia.org/wiki/Model-View-ViewModel (дата обращения: 02.04.2025).
2. Исакова, С., Жемеров, Д. Kotlin в действии. М.: Питер, 2018. 360 с.
3. Фацио, М. Kotlin and Android Development featuring Jetpack: Build Better, Safer Android Apps. Packt Publishing, 2020. 350 p.
4. Darwin, I. Android. Сборник рецептов: задачи и решения для разработчиков приложений. 2-е изд. СПб.: Питер, 2018. 608 с.
5. Глушенко, С. А., Долженко, А. И. Разработка мобильных приложений: учебное пособие. Ростов-на-Дону: РГЭУ (РИНХ), 2018. 221 с. URL: https://rsue.ru/sveden/files/09.03.03\_Razrabotka\_mobilynyx\_prilogheniy.pdf (дата обращения: 02.04.2025).
6. Kose, U., Deperlioglu, O. Deep Learning for Biomedical Applications. Springer, 2021. 189 p.
7. Loughran, S., Hatcher, E. Ant in Action. 2nd ed. Manning Publications, 2007. 600 p.
8. Черников, В. Разработка мобильных приложений на C# для iOS и Android. М.: ДМК Пресс, 2017. 350 с.

**YandexGpt**

1. Гулаков, С. Д. Android-разработка на Kotlin с нуля до PRO / С. Д. Гулаков. — Москва : Диалектика, 2020. — 464 с.
2. Бертт, Э. Программирование на Kotlin / Э. Бертт. — Москва : Вильямс, 2017. — 512 с.
3. Google. Compose for Android [Электронный ресурс] / Google. — Режим доступа: https://developer.android.com/jetpack/compose (дата обращения: [указать дату]).
4. Google. CameraX API [Электронный ресурс] / Google. — Режим доступа: https://developer.android.com/training/camerax (дата обращения: [указать дату]).
5. Fabián, G. Android Architecture Patterns with Kotlin: Implementing Clean Architecture, MVVM, and Repository Pattern / G. Fabián. — Birmingham : Packt Publishing, 2021. — 356 p.
6. Abhishek, N. TensorFlow 2.x YOLO for Object Detection: Build a real-time object detection system with TensorFlow Lite / N. Abhishek. — Birmingham : Packt Publishing, 2020. — 282 p.
7. Baeldung. Gradle Tutorial [Электронный ресурс] / Baeldung. — Режим доступа: https://www.baeldung.com/gradle (дата обращения: [указать дату]).
8. Google. Android Studio [Электронный ресурс] / Google. — Режим доступа: https://developer.android.com/studio (дата обращения: [указать дату]).

**GigaChat**

1. Model-View-ViewModel (MVVM)Блинов, В. Н. Разработка мобильных приложений на основе паттерна MVVM / В. Н. Блинов // Программирование и информационные технологии. — 2019. — № 4. — С. 20–25.
2. KotlinИванов, А. А. Основы программирования на языке Kotlin / А. А. Иванов. — М.: ДМК Пресс, 2018. — 272 с.
3. Jetpack ComposeAndroid Developers. Guide to Jetpack Compose [Электронный ресурс]. — Режим доступа: https://developer.android.com/jetpack/compose/documentation. — Дата обращения: 10.11.2023.
4. CameraXGoogle Developers. Introducing CameraX, a new Jetpack support library for Android camera development [Электронный ресурс]. — Режим доступа: https://developers.googleblog.com/2019/05/introducing-camerax-jetpack-support.html. — Дата обращения: 15.09.2023.
5. RoomPapinutto, F. Mastering Room: A deep dive into Android's powerful database library / F. Papinutto. — Packt Publishing, 2020. — 360 p.
6. TensorFlow LiteTensorFlow Team. An Introduction to TensorFlow Lite: On-device Machine Learning Framework [Электронный ресурс]. — Режим доступа: https://www.tensorflow.org/lite/guide/get\_started. — Дата обращения: 01.10.2023.
7. GradleHäyrynen, K. Gradle Beyond the Basics: Customizing Next-Generation Builds / K. Häyrynen. — O'Reilly Media, Inc., 2016. — 250 p.
8. Android StudioBurnette, E. Hello, Android: Introducing Google's Mobile Development Platform / E. Burnette. — Pragmatic Bookshelf, 2017. — 320 p.

**DeepSeek**

1. **Model-View-ViewModel (MVVM)**  
   Android Developers. ViewModel overview | App architecture [Электронный ресурс]. – URL: https://developer.android.com/topic/libraries/architecture/viewmodel (дата обращения: 03.04.2025).
2. **Kotlin**  
   JetBrains. Kotlin Language Documentation [Электронный ресурс]. – URL: https://kotlinlang.org/docs/home.html (дата обращения: 03.04.2025).
3. **Jetpack Compose**  
   Smith J. Jetpack Compose: A Complete Guide for Kotlin Android Developers. – 1st ed. – 2024. – 400 p. – ISBN-10: B0D6L39Y7V.
4. **CameraX**  
   Delasign. How to setup a camera in an Android app using Jetpack Compose [Электронный ресурс]. – 2023. – URL: https://www.delasign.com/blog/android-studio-jetpack-compose-kotlin-camerax-composable/ (дата обращения: 03.04.2025).
5. **Room**  
   Gures C. Basic Implementation of Room Database With Repository and ViewModel | Android Jetpack // Medium. – 2020. – URL: https://medium.com/swlh/basic-implementation-of-room-database-with-repository-and-viewmodel-android-jetpack-8945b364d322 (дата обращения: 03.04.2025).
6. **TensorFlow Lite**  
   TensorFlow Developers. TensorFlow Lite Documentation [Электронный ресурс]. – URL: https://www.tensorflow.org/lite (дата обращения: 03.04.2025).
7. **Gradle**  
   Gradle Inc. Gradle User Manual [Электронный ресурс]. – URL: https://docs.gradle.org/current/userguide/userguide.html (дата обращения: 03.04.2025).
8. **Android Studio**  
   Google Developers. Android Studio User Guide [Электронный ресурс]. – URL: https://developer.android.com/studio/intro (дата обращения: 03.04.2025)

**Вывод:** были изучены и исследованы на темах ВКРМ чатыChatGPT, GigaChat, YandexGpt и DeepSeek.

**Задача 1**

**Цель задания:** получить практику по использованию классического машинного обучения и настройки функционала работы с git

**Ключевой навык:** генерация случайной выборки, борьба с переобучением

**Датасет:** необходимо загрузить датасет, расположенный по адресу: https://archive.ics.uci.edu/ml/datasets/iris

**Задание:**

1. Постройте таблицу pairplot для визуального анализ данных,
2. Постройте LogisticRegression для бинарной классификации элементов датасета, на примере данных
3. Покажите переобучение модели линейной регрессии и предложите способ борьбы с ним
4. Обогатите датасет сгенерированными вами данными. Опишите какой способ вы применяли и приложите код генерации данных.
5. Поэкспериментируйте с возможными AutoML моделями для усовершенствования эталонного решения.

**Лабораторная работа № 1**

Была выполнена загрузка датасета с помощью «pandas», после чего был построен pairplot с целью визуального анализа данных, результат показан на рисунке 1.



Рисунок 1 – Pairplot

На рисунке можно заметить, что класс «iris-setosa» легко отличим от остальных классов по всем параметрам. Код загрузки датасета и построения pairplot:

from sklearn.linear\_model import LogisticRegression

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

dataset = pd.read\_csv('https://raw.githubusercontent.com/andriygav/MachineLearningSeminars/master/sem1/data/iris.csv',

                      header=None,

                      names=['длина чашелистика', 'ширина чашелистика',

                             'длина лепестка', 'ширина лепестка', 'класс'])

# Пример данных

dataset.sample(5, random\_state=0)

# Анализ данных

sns.pairplot(dataset, hue='класс', height=2)

plt.show()

Следующим шагом датасет был преобразован для задачи бинарной классификации, были оставлены классы «Iris-versicolor» и «Iris-virginica», так как их различие представляет наибольшую сложность. Код с удалением ненужных данных и подготовки датасета представлен ниже:

# Фильтрация данных для двух классов

binary\_dataset = dataset.drop(index=dataset.index[dataset['класс'] == 'Iris-setosa'])

binary\_dataset.sample(5, random\_state=0)

# Закодируем классы числами

binary\_dataset.loc[dataset['класс'] == 'Iris-versicolor', dataset.columns == 'класс'] = 0

binary\_dataset.loc[dataset['класс'] == 'Iris-virginica', dataset.columns == 'класс'] = 1

binary\_dataset.sample(5, random\_state=0)

Дополнительно были сгенерированы синтетические данные с помощью библиотеки sickit-learn. Функция make\_classification из библиотеки sklearn.datasets используется для генерации синтетического набора данных для задач классификации. Она создаёт новые данные на основе случайных комбинаций линейных зависимостей между признаками, при этом часть признаков является информативными (влияет на принадлежность к классу), а остальные — шумовыми или избыточными. Код представлен ниже:

from sklearn.datasets import make\_classification

import numpy as np

n\_samples = 50

n\_features = X\_train.shape[1]  # чтобы совпадало по размеру

# Генерация новых данных

X\_new, y\_new = make\_classification(n\_samples=n\_samples,

                                   n\_features=n\_features,

                                   n\_informative=n\_features - 1,

                                   n\_redundant=0,

                                   n\_classes=2,

                                   flip\_y=0.05,  # небольшой шум в метках

                                   random\_state=42)

# Проверим размерности

print("Сгенерировано:", X\_new.shape)

# Объединяем с существующим тренировочным набором

X\_train = np.vstack([X\_train, X\_new])

y\_train = np.hstack([y\_train, y\_new])

print("Итоговая выборка:", X\_train.shape)

**Лабораторная работа № 3**

1. **LogisticRegression**

С использованием библиотеки sickit-learn была построена модель для логистической регрессии и обучена на датасете. Так как по заданию необходимо продемонстрировать переобучение, специально была использована избыточно сложная модель с большой степенью полинома. Код для получения модели:

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, mean\_squared\_error

import numpy as np

from sklearn.preprocessing import PolynomialFeatures

# Разделение признаков и меток

X = binary\_dataset.drop('класс', axis=1).values

X = np.array(np.hstack([X, np.ones([len(X), 1])]), dtype=np.float64)

y = binary\_dataset['класс'].values.reshape(-1)

y = np.array(y, dtype=np.int64)

# Разделение на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Преобразуем признаки до 5-й степени

poly = PolynomialFeatures(degree=5, include\_bias=False)

X\_train\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

# Обучение модели логистической регрессии

model = LogisticRegression(random\_state=0, max\_iter=2000)

model.fit(X\_train\_poly, y\_train)

# Предсказания

y\_train\_pred = model.predict(X\_train\_poly)

y\_test\_pred = model.predict(X\_test\_poly)

# Accuracy

train\_acc = accuracy\_score(y\_train, y\_train\_pred)

test\_acc = accuracy\_score(y\_test, y\_test\_pred)

# MSE

train\_mse = mean\_squared\_error(y\_train, y\_train\_pred)

test\_mse = mean\_squared\_error(y\_test, y\_test\_pred)

# Вывод

print(f"Train Accuracy: {train\_acc:.4f}")

print(f"Test Accuracy: {test\_acc:.4f}")

print(f"Train MSE: {train\_mse:.4f}")

print(f"Test MSE: {test\_mse:.4f}")

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_test\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_test\_pred))

На рисунке 2 представлены метрики полученной модели:



Рисунок 2 – Метрики модели

Можно заметить, что на тренировочной выборке модель показывает себя немного лучше, чем на тестовой. Это может свидетельствовать о небольшом переобучении модели, когда она полностью подстроилась под обучающую выборку. Есть несколько средств борьбы с переобучением:

1. Увеличение датасета
2. Использование регуляризации
3. Упрощение модели
4. Кросс-валидация
5. и др.

В данной работе проблема переобучения решалась с помощью упрощения модели, а именно уменьшения степени полинома. Код обучения упрощённой модели представлен ниже:

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, mean\_squared\_error

import numpy as np

from sklearn.preprocessing import PolynomialFeatures

# Разделение признаков и меток

X = binary\_dataset.drop('класс', axis=1).values

X = np.array(np.hstack([X, np.ones([len(X), 1])]), dtype=np.float64)

y = binary\_dataset['класс'].values.reshape(-1)

y = np.array(y, dtype=np.int64)

# Разделение на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Преобразуем признаки до 2-й степени

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_train\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

# Обучение модели логистической регрессии

model = LogisticRegression(random\_state=0, max\_iter=2000)

model.fit(X\_train\_poly, y\_train)

# Предсказания

y\_train\_pred = model.predict(X\_train\_poly)

y\_test\_pred = model.predict(X\_test\_poly)

# Accuracy

train\_acc = accuracy\_score(y\_train, y\_train\_pred)

test\_acc = accuracy\_score(y\_test, y\_test\_pred)

# MSE

train\_mse = mean\_squared\_error(y\_train, y\_train\_pred)

test\_mse = mean\_squared\_error(y\_test, y\_test\_pred)

# Вывод

print(f"Train Accuracy: {train\_acc:.4f}")

print(f"Test Accuracy: {test\_acc:.4f}")

print(f"Train MSE: {train\_mse:.4f}")

print(f"Test MSE: {test\_mse:.4f}")

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_test\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_test\_pred))

На рисунке 3 показаны метрики упрощённой модели.

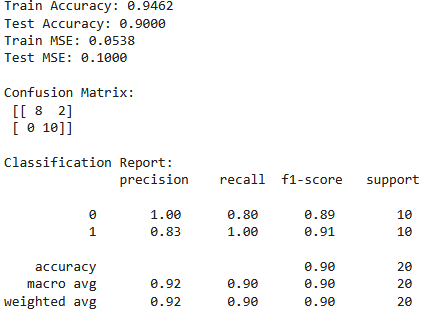


Рисунок 3 – Метрики упрощённой модели

Можно заметить, что теперь явного переобучения нет и результаты тестовой выборки практически совпадают с тренировочной.

1. **FEDOT**

Для получения модели бинарной классификации с помощью фреймворка FEDOT был реализован пайплайн, представленный на рисунке 4.



Рисунок 4 – Pipeline для обучения на FEDOT

Код для обучения модели представлен ниже:

from fedot.api.main import Fedot

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report

import numpy as np

# Обёртка FEDOT

automl = Fedot(problem='classification',

               timeout=5,

               preset='best\_quality')

# Обучение модели

automl.fit(features=X\_train, target=y\_train)

# Предсказание

y\_pred = automl.predict(features=X\_test)

# Предсказание (возвращает метки сразу)

y\_pred\_labels = automl.predict(features=X\_test)

# Оценка

print(f"\nAccuracy: {accuracy\_score(y\_test, y\_pred\_labels):.4f}")

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred\_labels))

Результаты тестирование модели представлены на рисунке 5.



Рисунок 5 – Модель FEDOT

1. **LAMA**

Для AutoML решения с помощью фреймворка LAMA был разработан пайплайн, аналогичный с FEDOT. Код обучения представлен ниже:

mport pandas as pd

import numpy as np

from lightautoml.tasks import Task

from lightautoml.automl.presets.tabular\_presets import TabularAutoML

from sklearn.metrics import accuracy\_score, classification\_report

import warnings

warnings.filterwarnings("ignore")

# Преобразуем массивы обратно в DataFrame и добавим целевой столбец

X\_train\_df = pd.DataFrame(X\_train)

X\_train\_df['target'] = y\_train

X\_test\_df = pd.DataFrame(X\_test)

X\_test\_df['target'] = y\_test

# LightAutoML требует строковые названия колонок

X\_train\_df.columns = [str(col) for col in X\_train\_df.columns]

X\_test\_df.columns = [str(col) for col in X\_test\_df.columns]

# Задача — бинарная классификация

task = Task('binary')

# Создаём и запускаем AutoML

automl = TabularAutoML(task=task,

                       timeout=180,            # Время на обучение в секундах

                       cpu\_limit=2)

# Обучение модели

oof\_pred = automl.fit\_predict(X\_train\_df, roles={'target': 'target'})

# Предсказание на тестовых данных

test\_pred = automl.predict(X\_test\_df)

# Получение финальных меток (классы из вероятностей)

y\_pred = (test\_pred.data[:, 0] > 0.5).astype(int)

y\_true = y\_test

print(f" LAMA Accuracy: {accuracy\_score(y\_true, y\_pred):.4f}")

print("\nClassification Report:\n", classification\_report(y\_true, y\_pred))

Результаты тестирования полученной модели показаны на рисунке 6.

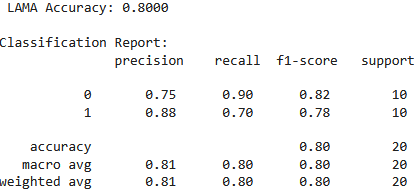


Рисунок 6 – Модель LAMA

В таблице 1 показано сравнение метрик для трёх моделей. Были задействованы следующие метрики: accuracy, f1-macro, recall-macro, precision-macro.

Таблица 1 – Сравнение моделей

|  |  |  |  |
| --- | --- | --- | --- |
| Метрика | LogisticRegression | FEDOT | LAMA |
| Accuracy | 0.90 | 0.8 | 0.8 |
| F1-macro | 0.90 | 0.8 | 0.8 |
| Recall-macro | 0.90 | 0.8 | 0.8 |
| Precision-macro | 0.92 | 0.81 | 0.81 |

Из таблицы можно сделать вывод, что построенная модель логистической регрессии лучше, чем модели FEDOT и LAMA, которые идентичны между собой.

**Лабораторная работа № 4**

Полученная модель была сохранена, а также протестирована:

y\_test\_pred = model.predict(X\_test\_poly)

# Accuracytest\_acc = accuracy\_score(y\_test, y\_test\_pred)

# MSE

test\_mse = mean\_squared\_error(y\_test, y\_test\_pred)

# Вывод

print(f"Train Accuracy: {train\_acc:.4f}")

print(f"Test Accuracy: {test\_acc:.4f}")

print(f"Train MSE: {train\_mse:.4f}")

print(f"Test MSE: {test\_mse:.4f}")

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_test\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_test\_pred))

На рисунке 7 показаны метрики модели.

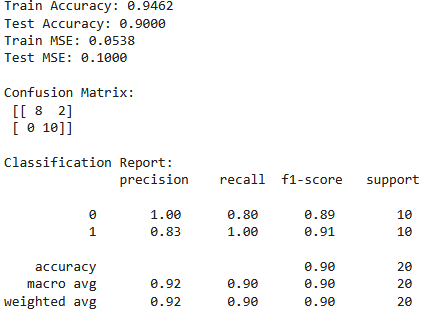


Рисунок 7 – Метрики модели

После тестирования модель была развёрнута в среде MLFlow. Результаты показаны на рисунках 8 и 9.

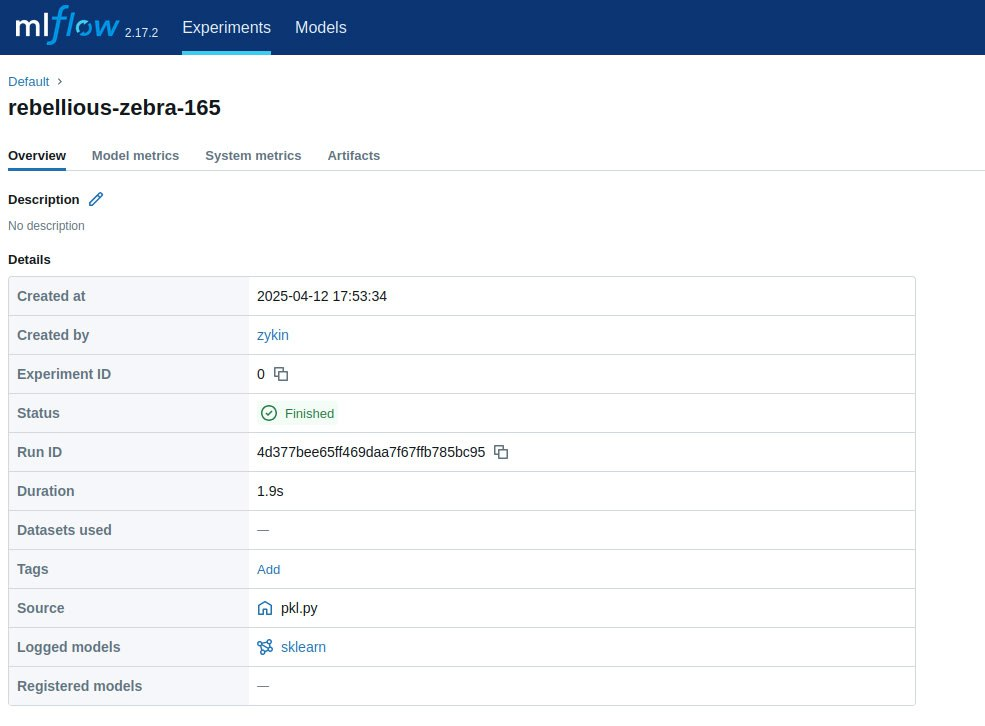


Рисунок 8 – Модель в MLFlow



Рисунок 9 – Модель в MLFlow 2

**Вывод**: в результате данной работы был сформирован датасет для классификации Iris. Была обучена модель логистической регрессии с переобучением и без него, были сгенерированы дополнительные данные с помощью make\_classification. Также были обучены модели с помощью FEDOT и LAMA. Все модели были протестированы, лучшими метрики обладала модель логистической регрессии. В результате модель была сохранена, протестирована и развёрнута в MLFlow.

**Задача 2**

**Цель задания:** обработка кардиологического дата-сета для решения задач классификации

**Ключевой навык:** подготовка данных, применение t-SNE и PCA

**Датасет:** необходимо загрузить датасет, расположенный по адресу (см. файлы с названием «модуль 2…): https://github.com/AI-is-out-there/data2lab.git

**Задание:**

1. Визуализируйте с помощью boxplot загруженные данные, опишите ваши мысли по данным.

2. Проанализируйте данные и проведите их предобработку (очистку, обогащение).

3. Сформируйте очищенный датасет, состоящий из числовых параметров ЭКГ и классификационного признака Healthy\_Status, заново построить boxplot и сравните результаты с первоначальным.

4. Проведите оценку взаимосвязи данных датасета с помощью функции heatmap.

5. На основе таргета (целевого признака) Healthy\_Status построить корреляционную матрицу с помощью pairplot.

6. Провести разведочный анализ данных с помощью метода PCA и с помощью TSNE.

7. Сделать вывод о структуре признаков данных.

8. Сделайте вывод о том какое из Auto ML решений с этой задачей справилось лучше.

**Лабораторная работа №2**

Сначала был загружен датасет и выполнена первичная обработка, чтобы объединить все колонки «report\_х» и перевести текст в числовые признаки, а также удалены ненужные колонки. Код представлен ниже:

# Предобрабатываем данные словестных репортов word2vec в числовой вектор

full\_df\_filtered = raw\_table\_data.copy()

# Слепляем все текстовые отчеты в один большой

reports = [f'report\_{x}' for x in range(18)]

full\_df\_filtered['report\_0'] = full\_df\_filtered[reports].astype(str).agg(' '.join, axis=1)

full\_df\_filtered['report\_0'] = full\_df\_filtered['report\_0'].str.replace(r'\bnan\b', '', regex=True).str.replace(r'\s+', ' ', regex=True).str.strip()

full\_df\_filtered.rename(columns={'report\_0': 'report'}, inplace=True)

reports\_to\_drop = [f'report\_{x}' for x in range(1, 18)]

full\_df\_filtered = full\_df\_filtered.drop(reports\_to\_drop, axis=1)

# Фиксим имена столбцов

full\_df\_filtered = full\_df\_filtered.rename(columns={'eeg\_time ': 'eeg\_time', 'eeg\_date ': 'eeg\_date'})

full\_df\_filtered = full\_df\_filtered.drop(columns = ['bandwidth', 'filtering'])

# Делаем колонку с таргетами крайней справа

full\_df\_filtered = full\_df\_filtered[[col for col in full\_df\_filtered.columns if col != 'Healthy\_Status'] + ['Healthy\_Status']]

# Разбиваем столбец с текстовым отчетом на токены

words = [text.split() for text in full\_df\_filtered['report']]

# Обучаем модель Word2Vec

w2v\_model = Word2Vec(words)

# Функция для получения среднего эмбеддинга строки

def get\_sentence\_embedding(sentence):

    words = sentence.split()

    word\_vectors = [w2v\_model.wv[word] for word in words if word in w2v\_model.wv]

    if word\_vectors:

        return np.mean(word\_vectors, axis=0)  # Усреднённый вектор

    else:

        return np.zeros(w2v\_model.vector\_size)  # Нулевой вектор, если слова нет в модели

# Применяем функцию ко всей колонке и переводим каждый вектор в одно число

full\_df\_filtered['report'] = full\_df\_filtered['report'].apply(lambda x: get\_sentence\_embedding(x).mean())

full\_df\_filtered.head()

# make new dataset from old columns

table\_data = full\_df\_filtered[['report','rr\_interval','p\_onset','p\_end', 'qrs\_onset','qrs\_end','t\_end','p\_axis','qrs\_axis','t\_axis','Healthy\_Status']].copy()

table\_data.head()

Для первичного анализа данных был построен boxplot, представленный на рисунке 10.

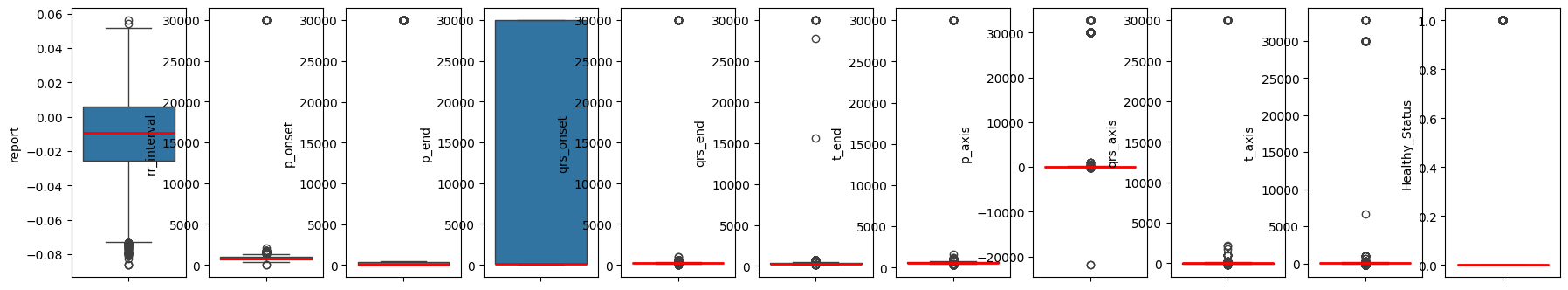


Рисунок 10 – BoxPlot перед обработкой

Из полученного графика видно, что в исходных данных очень много выбросов и аномальных значений, поэтому необходимо отфильтровать значения, избавившись от выбросов и аномалий. Код представлен ниже:

# Убираем выбросы

columns\_to\_filter = ['rr\_interval', 'p\_onset','p\_end', 'qrs\_onset', 'qrs\_end', 't\_end', 'p\_axis', 'qrs\_axis', 't\_axis']

table\_data = table\_data[(table\_data[columns\_to\_filter] < 2000).all(axis=1)]

table\_data = table\_data[(table\_data['p\_onset'] < table\_data['p\_end']) & (table\_data['qrs\_onset'] < table\_data['qrs\_end'])]

# удаляем аномалии из данных (убрать все значения более 5000)

for i, v in enumerate(table\_data):

  table\_data = table\_data.drop(table\_data[table\_data.iloc[:,i] > 10000].index)

# Удаляем столбец p\_onset - он неинформативный

table\_data = table\_data[['report','rr\_interval','p\_end', 'qrs\_onset','qrs\_end','t\_end','p\_axis','qrs\_axis','t\_axis','Healthy\_Status']].copy()

table\_data.head()

После фильтрации датасета boxplot был построен заново. Новый boxplot показан на рисунке 11.

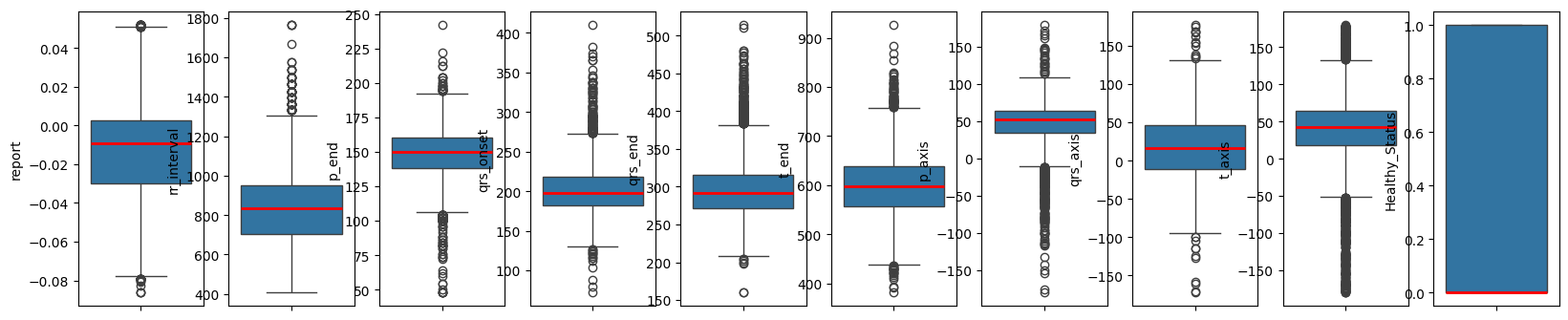


Рисунок 11 – BoxPlot после фильтрации

После фильтрации визуализация датасета стала намного лучше. Однако даже после неё в данных осталось много выбросов. Это может быть связано с тем, что симптомы пациентов могут слишком разниться, а значит и показания тоже. Поэтому такой разброс значений объясним предметной областью.

Следующим шагом была построена тепловая карта, чтобы оценить корреляцию данных. Она показана на рисунке 12.

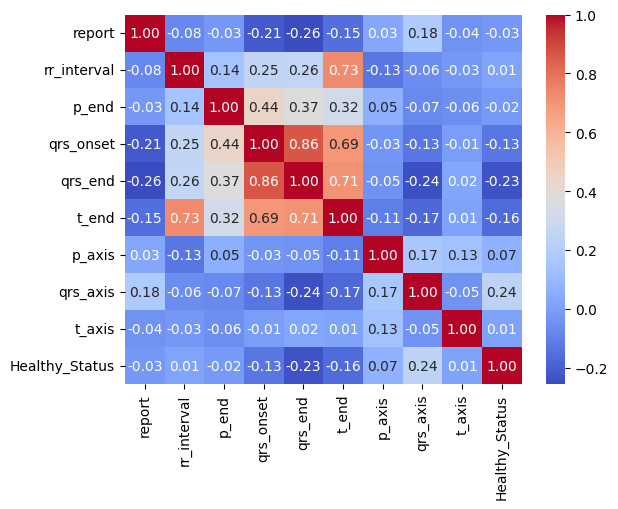


Рисунок 12 – Тепловая карта

Из таблицы тепловой карты можно заметить, что нет явно выраженной корреляции между целевым классом и отдельными параметрами. Это говорит о том, что зависимость более сложная, строящаяся на взаимосвязи совокупности параметров, а не по отдельности.

Затем по целевому классу был построен PairPlot. Результат показан на рисунке 13.

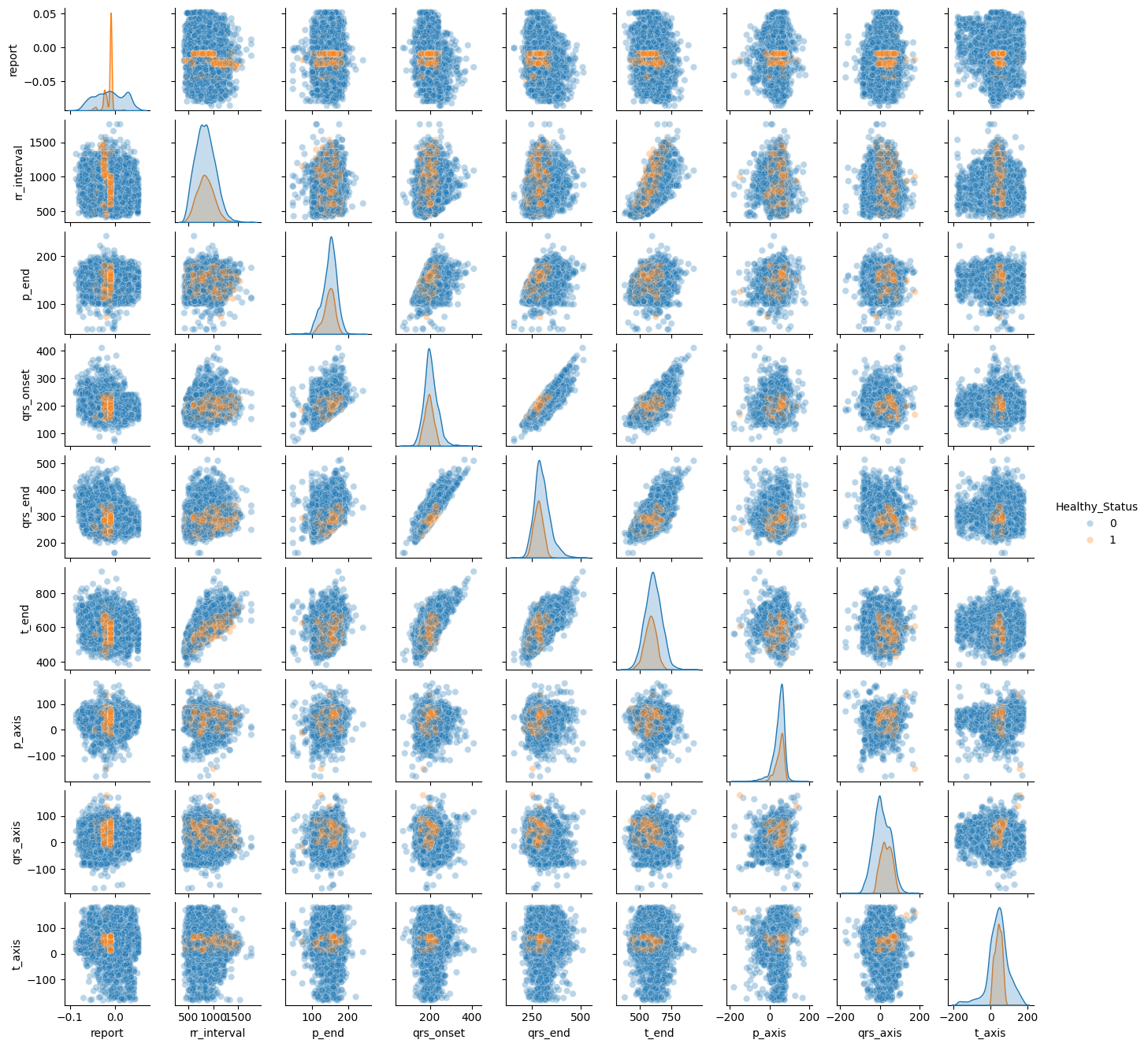


Рисунок 13 – PairPlot

Из рисунка можно проследить следующую тенденцию: значения для «условно-здоровых» более централизованы около среднего значения, в то время как для пациентов наоборот наблюдается очень большой разброс.

Следующим шагом был проведён PCA-анализ. Код представлен ниже:

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

table\_data\_pca = table\_data

table\_data\_pca = table\_data\_pca.drop('Healthy\_Status', axis = 1)

table\_data\_pca.columns

# Before applying PCA, each feature should be centered (zero mean) and with unit variance

scaled\_data = StandardScaler().fit(table\_data\_pca).transform(table\_data\_pca)

pca = PCA(n\_components = 2).fit(scaled\_data)

x\_pca = pca.transform(scaled\_data)

print(table\_data.shape, x\_pca.shape)

percent = pca.explained\_variance\_ratio\_

print(percent)

print(sum(percent))

plt.figure(figsize=(8,6))

plt.scatter(x\_pca[:,0], x\_pca[:,1], c=table\_data['Healthy\_Status'], cmap='plasma', alpha=0.4, edgecolors='black', s=65);

plt.xlabel('First Principal Component')

plt.ylabel('Second Principal Component')

# put feature values into dataframe

components = pd.DataFrame(pca.components\_.T, index=table\_data\_pca.columns, columns= ['PCA1','PCA2'])

# plot size

plt.figure(figsize=(10,8))

# main scatterplot

plt.scatter(x\_pca[:,0], x\_pca[:,1], c=table\_data['Healthy\_Status'],

            cmap='plasma', alpha=0.4, edgecolors='black', s=40);

plt.xlabel('First Principal Component')

plt.ylabel('Second Principal Component')

#plt.ylim(15,-15);

#plt.xlim(20,-20);

# individual feature values

ax2 = plt.twinx().twiny();

#ax2.set\_ylim(-0.5,0.5);

#ax2.set\_xlim(-0.5,0.5);

# reference lines

ax2.hlines(0,-0.5,0.5, linestyles='dotted', colors='grey')

ax2.vlines(0,-0.5,0.5, linestyles='dotted', colors='grey')

# offset for labels

offset = 0.95

# arrow & text

for a, i in enumerate(components.index):

  ax2.arrow(0, 0, components['PCA1'][a], -components['PCA2'][a], alpha=0.5, facecolor='white', head\_width=.01)

  ax2.annotate(i, (components['PCA1'][a]\*offset, -components['PCA2'][a]\*offset), color='orange')

На рисунке 14 показан результат PCA-анализа с двумя компонентами.

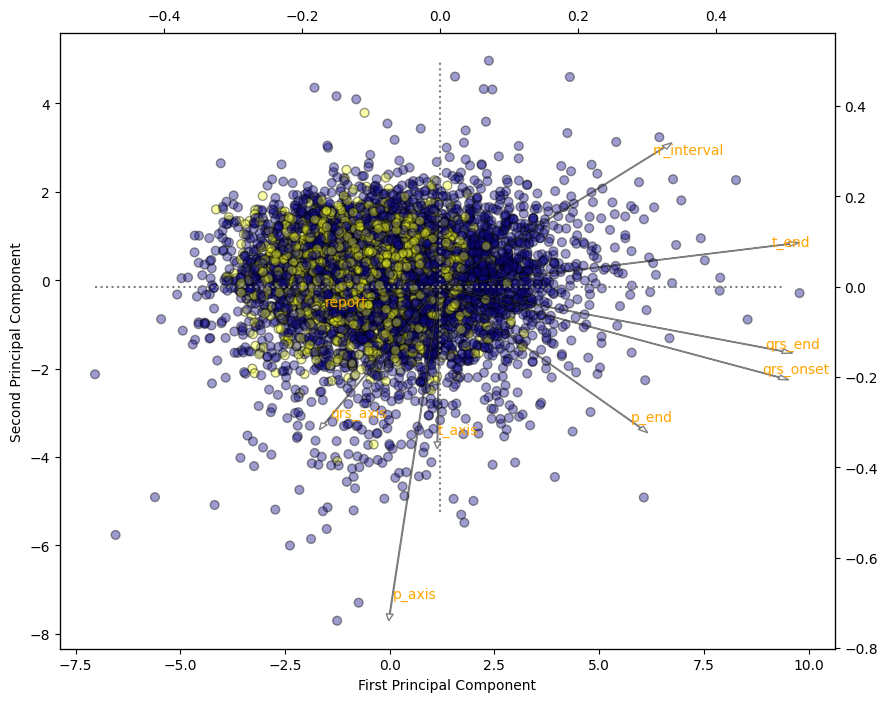


Рисунок 14 – Результат PCA-анализа

Из PCA-анализа видно, что в значениях присутствует большой разброс, но значения для класса 1 более «кучные». Кроме того, можно увидеть, что наибольшее внимание оказывает значение «p».

Затем был проведён t-SNE анализ, код представлен ниже:

from sklearn.manifold import TSNE

tsne = TSNE(random\_state = 0)

scaled\_data\_tsne = tsne.fit\_transform(scaled\_data)

plt.scatter(scaled\_data\_tsne[:,0], scaled\_data\_tsne[:,1], c=table\_data['Healthy\_Status'], cmap='plasma', alpha=0.4, edgecolors='black', s=65);

plt.xlabel('First Principal Component')

plt.ylabel('Second Principal Component')

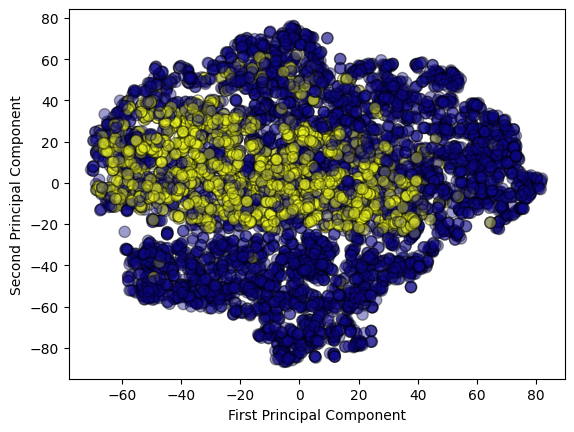


Рисунок 15 – Результат t-SNE анализа.

Результаты в целом соответствуют предыдущим методам, но в этом случае более явно выражена централизация значений для класса «1». В целом видно, что два класса различимы в пространственных координатах значений.

**Лабораторная работа №3**

1. **FEDOT**

Для обучения модели с помощью FEDOT был реализован pipeline представленный на рисунке 16.

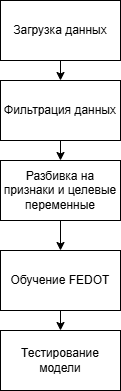


Рисунок 16 – Pipeline FEDOT

Код представлен ниже:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from fedot.api.main import Fedot

from fedot.core.repository.tasks import Task, TaskTypesEnum

from sklearn.metrics import accuracy\_score, classification\_report

# Разбиваем данные на признаки и целевой признак

X = table\_data.drop('Healthy\_Status', axis=1)

y = table\_data['Healthy\_Status']

# Выполняем разбиение на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Задаём тип задачи – классификация

# Создаём экземпляр Fedot для решения задачи классификации

model = Fedot(problem='classification', timeout=5, preset='best\_quality')

# Обучаем модель

model.fit(features=X\_train, target=y\_train)

# Получаем предсказания на тестовой выборке

predictions = model.predict(X\_test)

# Оценка

print(f"\nAccuracy: {accuracy\_score(y\_test, predictions):.4f}")

print("\nClassification Report:\n", classification\_report(y\_test, predictions))

Для тестирования моделей бинарной классификации были выбраны следующие метрики: accuracy, f1-macro, precision-macro и recall-macro.

Результаты тестирования модели показаны на рисунке 17.

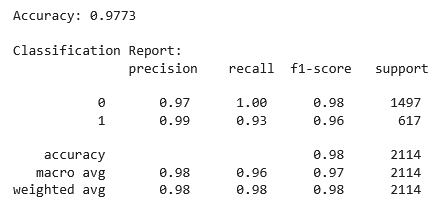
****

Рисунок 17 – Результаты FEDOT

1. **LAMA**

Для обучения на LAMA был реализован pipeline на рисунке 18.



Рисунок 18 – Pipeline для LAMA

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from lightautoml.automl.presets.tabular\_presets import TabularAutoML

from lightautoml.tasks import Task

# Разбиваем данные на обучающую и тестовую выборки

train\_df, test\_df = train\_test\_split(table\_data, test\_size=0.3, random\_state=42)

# Создаём модель LightAutoML (TabularAutoML) для задачи бинарной классификации

# Параметр timeout задаёт максимальное время поиска модели (в секундах)

automl = TabularAutoML(task=Task('binary'), timeout=30)

# Обучаем модель. В качестве roles можно указать имя целевой переменной ('target')

oof\_pred = automl.fit\_predict(train\_df, roles={'target': 'Healthy\_Status'})

# Выполняем предсказание на тестовой выборке

pred\_test = automl.predict(test\_df)

# Получаем предсказанные вероятности. Преобразуем их в бинарные классы с порогом 0.5

y\_pred = (pred\_test.data[:, 0] > 0.5).astype(int)

# Выводим отчёт классификации

report = classification\_report(test\_df['Healthy\_Status'], y\_pred)

print(report)

Результаты тестирования модели показаны на рисунке 19.

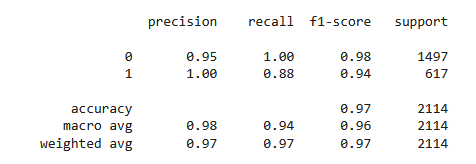


Рисунок 19 – Результаты LAMA

1. **H2O**

Для обучения с помощью H2O был реализован pipeline, представленный на рисунке 20.



Рисунок 20 – Pipeline H2O

import h2o

from h2o.automl import H2OAutoML

import pandas as pd

from sklearn.metrics import classification\_report

# Инициализируем кластер H2O

h2o.init()

# Конвертируем Pandas DataFrame в H2OFrame

hf = h2o.H2OFrame(table\_data)

# Для бинарной классификации целевой признак нужно привести к категориальному типу

hf['Healthy\_Status'] = hf['Healthy\_Status'].asfactor()

# Разбиваем данные на обучающую и тестовую выборки

train, test = hf.split\_frame(ratios=[0.7], seed=42)

# Определяем список признаков и целевую переменную

target = 'Healthy\_Status'

features = [col for col in hf.col\_names if col != target]

# Создаём и обучаем модель H2O AutoML

aml = H2OAutoML(max\_runtime\_secs=30, seed=42, project\_name="h2o\_automl\_example")

aml.train(x=features, y=target, training\_frame=train)

# Выводим таблицу лидеров

lb = aml.leaderboard

print(lb)

# Получаем предсказания на тестовой выборке

predictions = aml.leader.predict(test)

# Преобразуем предсказания и истинные значения в Pandas DataFrame для расчёта метрик

pred\_df = predictions.as\_data\_frame()

true\_df = test[target].as\_data\_frame()

# Извлекаем предсказанные классы и истинные метки

# H2O возвращает предсказания в столбце 'predict'

y\_pred = pred\_df['predict'].astype(int)

y\_true = true\_df[target].astype(int)

# Выводим отчёт классификации (precision, recall, f1-score и support)

report = classification\_report(y\_true, y\_pred)

print(report)

# По окончании работы можно остановить кластер H2O

h2o.shutdown(prompt=False)

Результаты тестирования модели показаны на рисунке 21.

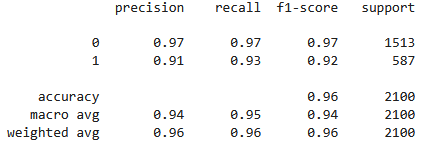


Рисунок 21 – Результаты H2O

В таблице 2 показаны результаты сравнения моделей.

Таблица 2 – Сравнение моделей

|  |  |  |  |
| --- | --- | --- | --- |
| Метрика | FEDOT | LAMA | H2O |
| Accuracy | 0.98 | 0.97 | 0.96 |
| F1-macro | 0.97 | 0.96 | 0.94 |
| Precision-macro | 0.98 | 0.98 | 0.94 |
| Recall-macro | 0.96 | 0.94 | 0.95 |

По таблице видно, что лучшие результаты показала модель, обученная с помощью FEDOT, потом идут LAMA и хуже всего показала H2O.

**Лабораторная работа №4**

После обучения модель была сохранена и протестирована:

predictions = model.predict(X\_test)

# Оценка

print(f"\nAccuracy: {accuracy\_score(y\_test, predictions):.4f}")

print("\nClassification Report:\n", classification\_report(y\_test, predictions))

Результаты тестирования показаны на рисунке 22.

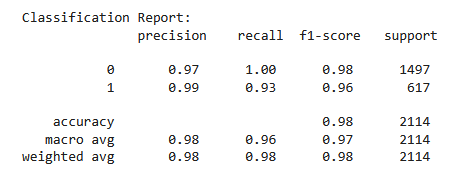


Рисунок 22 – Тестирование модели

После тестирования модель была развёрнута в MLFlow. Результаты развёртывания показаны на рисунках 23 и 24.

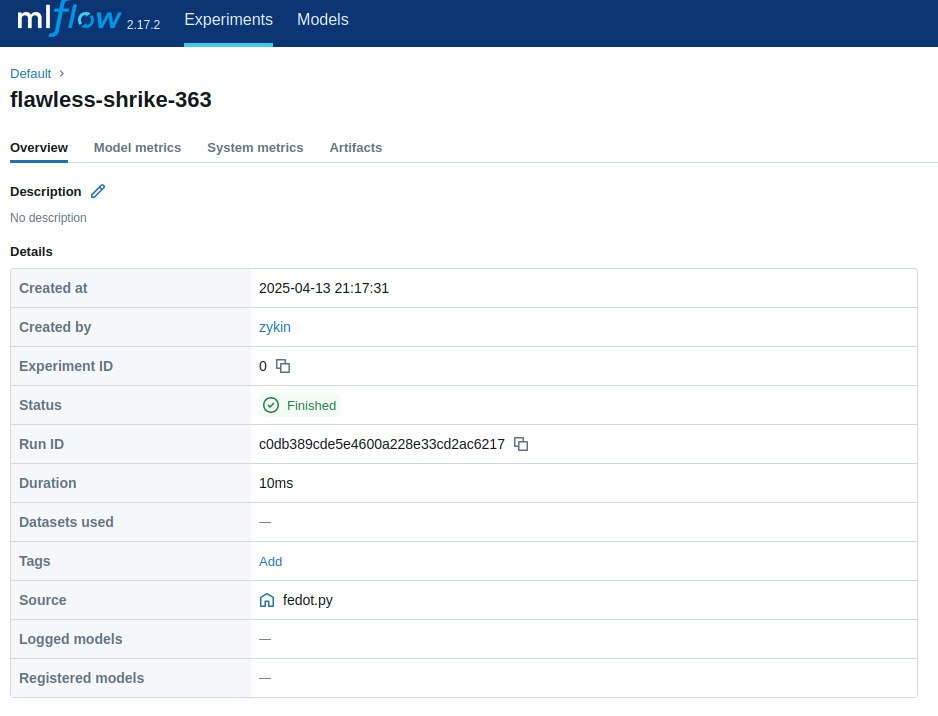


Рисунок 23 – Модель в MLFlow1

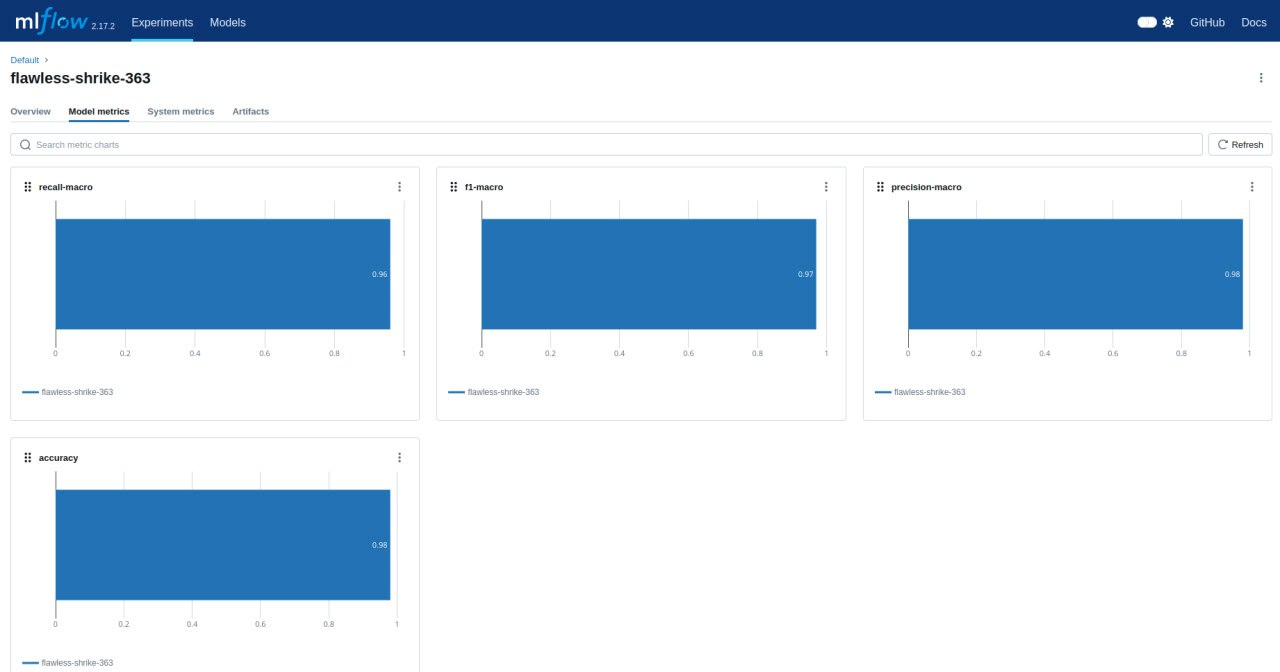


Рисунок 24 – Модель в MLFlow2

**Вывод:** в ходе данной работы был загружен датасет экг, после чего он был отфильтрован и проанализирован такими методами, как: boxplot, heatmap, pairplot, PCA, t-SNE. После подготовки были обучены следующие AutoML решения: FEDOT, LAMA и H2O. Лучше всех показала модель FEDOT. Затем модель была сохранена и развёрнута в MLFlow.

**Задача 5**

**Цель задания:** классифицировать временные ряды электроэнцефалографии для определения эпилепсии

**Ключевой навык:** сверточные нейронные сети для сложных временных рядов

**Датасет:** Необходимо перейти на страницу датасета, расположенного по адресу: https://physionet.org/content/chbmit/1.0.0/chb08/#files-panel .

**Задание:**

1. Сделайте скрипт для формирования обучающей выборки из базы данных ЭЭГ с указанными приступами
2. Сформируйте не менее 50 записей с приступами и не менее 50 записей ЭЭГ без приступов. Записи нужно «обрезать» по времени и построить для них вейвлет изображение (можно разбить на диапазоны ритмов мозга, можно не разбивать). В итоге должно быть 100 «картинок».
3. Полученные изображения сложите в отдельные папки (место хранения выбирайте на свое усмотрение). Папки нужны для того, чтобы разделить изображения для записей с приступом и для записей без приступа.
4. Спроектируйте сверточную нейронную сеть для обучения классификации ЭЭГ на те данные, которые имеют и не имеют приступ.
5. Сохраните модель обученной сети и проверьте ее работу.
6. Попробуйте проделать действия с использованием AutoML и сравните ратоту.

**Лабораторная работа № 2**

Для формирования обучающей выборки из базы данных CHB‑MIT был разработан скрипт, который считывал ЭЭГ-записи вместе с их метаданными. Сначала записи разделялись на временные сегменты фиксированной длительности – для записей с приступами выбирался интервал, охватывающий начало приступа, а для записей без приступов – случайное окно аналогичной длины. Затем каждый сегмент нормализовался (амплитуда приводилась к диапазону [0, 1]) и преобразовывался в вейвлет-образ, отражающий временно-частотные характеристики сигнала. Полученные изображения сохранялись в две отдельные папки в зависимости от наличия или отсутствия приступа. Такой подход позволил сформировать датасет, состоящий из не менее чем 50 изображений для каждого класса (приступ и отсутствующий приступ), что стало исходной базой для обучения модели.

Фрагмент основного цикла для формирования датасета представлен ниже:

for spisok in range(9,12,1):

  os.chdir('/content')

  #задаем путь до файла с описание ЭЭГ

  summary\_file\_path = '/physionet.org/files/chbmit/1.0.0/chb%d/chb%d-summary.txt' % (spisok+1, spisok+1)

  physionet\_summary\_file\_url = 'https:/'+summary\_file\_path

  #зaкачиваем в ноутбук файл описание ЭЭГ

  !wget -r -c -N --no-parent -np $physionet\_summary\_file\_url

  #распарсиваем записи из файла с описанием ЭЭГ на массив данных

  edf\_files\_list = parse\_summary\_file('/content'+summary\_file\_path)

  sampling\_rate = 256

  window\_count = 0.5

  flag = 0

  #основной цикл превращения EDF в изображение

  for edf\_file in edf\_files\_list:

    os.chdir('/content')

    edf\_directory = '/physionet.org/files/chbmit/1.0.0/chb%d/' % (spisok+1)

    edf\_file\_link = f"{'https:/'}{edf\_directory}{edf\_file['file\_name']}"

    #загружаем EDF в ноутбук

    !wget -r -c -N -np $edf\_file\_link

    #записывам сигналы и метки ЭЭГ в переменные

    edf\_file\_path = f"{'/content'}{edf\_directory}{edf\_file['file\_name']}"

    signals, labels = read\_edf(edf\_file\_path)

    # Если есть приступ, генерируем изображения для 3 временных интервалов

    if len(edf\_file['seizure\_start']) != 0:

      flag = 1

      seizure\_times = [(30, '30\_sec\_before'), (30, '60\_sec\_before'), (30, '90\_sec\_before')]

      number = 1

      for seiz\_time in range(len(edf\_file['seizure\_start'])):

        seizure\_st = edf\_file['seizure\_start'][seiz\_time]

        seizure\_en = edf\_file['seizure\_end'][seiz\_time]

        for time\_offset, time\_label in seizure\_times:

          time\_window\_start = seizure\_st - time\_offset

          time\_window\_end = seizure\_st  # Окно в 30 секунд

          seizure\_st = seizure\_st- 30

          time\_window\_start\_samples = time\_window\_start \* sampling\_rate

          time\_window\_end\_samples = time\_window\_end \* sampling\_rate

          signal\_s = np.delete(signals, [len(signals)-1], axis=0)

          time = np.arange(time\_window\_start, time\_window\_end, 1/sampling\_rate)

          # Усреднение сигналов

          avg\_signal = np.zeros(time\_window\_end\_samples - time\_window\_start\_samples)

          for i in range(len(signals)-1):

            avg\_signal += signal\_s[i, time\_window\_start\_samples:time\_window\_end\_samples]

          avg\_signal /= len(signals)

          # Применение фильтра Баттерворта

          filtered\_data = butter\_bandpass\_filter(avg\_signal, sampling\_rate, plot\_signal=False)

          filtered\_mean = mean\_delete(filtered\_data, window\_count, sampling\_rate,time,plot\_signal=False)

          # Определяем директорию и путь для сохранения изображения

          os.chdir('/content/drive/MyDrive/Example5/wavelet\_img')

          save\_dir = os.path.join(output\_dir, 'Seizure')

          os.makedirs(save\_dir, exist\_ok=True)

          # Сохранение изображения

          image\_path = f"{edf\_file['file\_name']}\_{time\_label}\_{number}\_wavelet.png"

          save\_wavelet\_transform(filtered\_mean, image\_path, save\_dir,

                                            time\_window\_start, time\_window\_end, sampling\_rate,time\_label)

          number+=1

    else:

      # Обработка для записей без приступа (Healthy)

      flag = 0

      #приступов нет, поэтому выбираем любое значение времени для выгрузки ЭЭГ

      #time\_window\_start = 2300

      time\_window\_start = int(signals.shape[1]/sampling\_rate/2)

      #time\_window\_end = 2330

      time\_label = 30 #ширина окна для выгрузки данных

      time\_window\_end = time\_window\_start + time\_label

      time\_window\_start\_samples = time\_window\_start \* sampling\_rate

      time\_window\_end\_samples = time\_window\_end \* sampling\_rate

      signals = np.delete(signals, [len(signals)-1], axis=0)

      time = np.arange(time\_window\_start, time\_window\_end, 1/sampling\_rate)

      # Усреднение сигналов

      avg\_signal = np.zeros(time\_window\_end\_samples - time\_window\_start\_samples)

      for i in range(len(signals)-1):

        # Проходим по всем каналам

        avg\_signal += signals[i, time\_window\_start\_samples:time\_window\_end\_samples]

        avg\_signal /= len(signals)  # Среднее значение по всем каналам

      # Фильр Баттерворта

      filtered\_data = butter\_bandpass\_filter(avg\_signal, sampling\_rate, plot\_signal=False)

      # Удаление среднего скользящим окном

      filtered\_mean = mean\_delete(filtered\_data, window\_count, sampling\_rate,time, plot\_signal=False)

      # Определяем директорию и путь для сохранения изображения

      save\_dir = os.path.join(output\_dir, 'Healthy')

      os.makedirs(save\_dir, exist\_ok=True)

      image\_path = f"{edf\_file['file\_name']}\_{time\_label}\_wavelet.png"

      # Сохранение изображения

      save\_wavelet\_transform(filtered\_mean, image\_path,

                                            save\_dir,time\_window\_start,

                                            time\_window\_end, sampling\_rate,time\_label)

Таким образом в ходе данной лабораторной работы было сформировано по 50 изображений каждого класса.

**Лабораторная работа № 3**

1. **Обучение CNN в Pytorch**

Была описана архитектура свёрточной нейронной сети в Pytorch для последующего обучения:

class SimpleCNN(nn.Module):

    def \_\_init\_\_(self, num\_classes=2):

        super(SimpleCNN, self).\_\_init\_\_()

        self.features = nn.Sequential(

            # Входное изображение имеет 3 канала (RGB)

            nn.Conv2d(3, 32, kernel\_size=3, stride=1, padding=1),

            nn.ReLU(),

            nn.MaxPool2d(kernel\_size=2, stride=2),

            nn.Conv2d(32, 64, kernel\_size=3, stride=1, padding=1),

            nn.ReLU(),

            nn.MaxPool2d(kernel\_size=2, stride=2),

            nn.Conv2d(64, 128, kernel\_size=3, stride=1, padding=1),

            nn.ReLU(),

            nn.MaxPool2d(kernel\_size=2, stride=2)

        )

        # При входном размере 128x128, после трёх раз максимального пула получаем размер 16x16

        self.classifier = nn.Sequential(

            nn.Linear(128 \* 16 \* 16, 256),

            nn.ReLU(),

            nn.Dropout(0.5),

            nn.Linear(256, num\_classes)

        )

    def forward(self, x):

        x = self.features(x)

        # Выравнивание тензора в вектор

        x = x.view(x.size(0), -1)

        x = self.classifier(x)

        return x

Следующем шагом модель была обучена. График обучения представлен на рисунке Х.

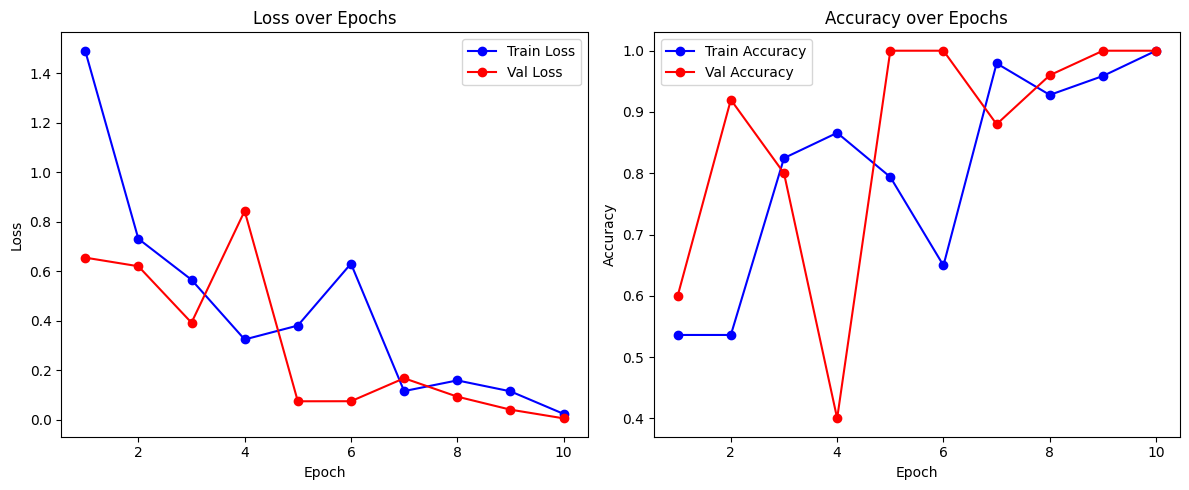


Рисунок Х – График обучения модели

Следующим шагом модель была сохранена и протестирована. На рисунках Х-Х представлены матрица ошибок, ROC-кривая, Precision-Recall кривая.

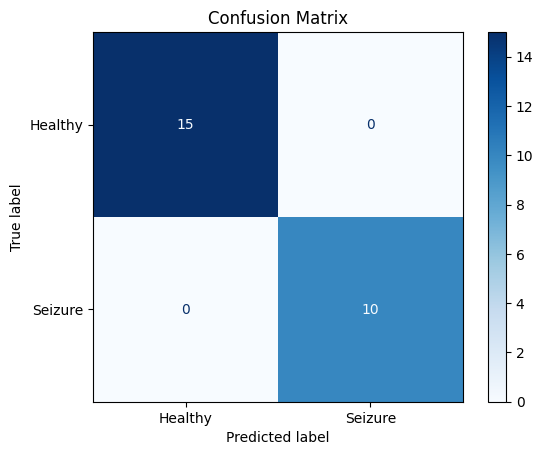


Рисунок Х – Матрица ошибок

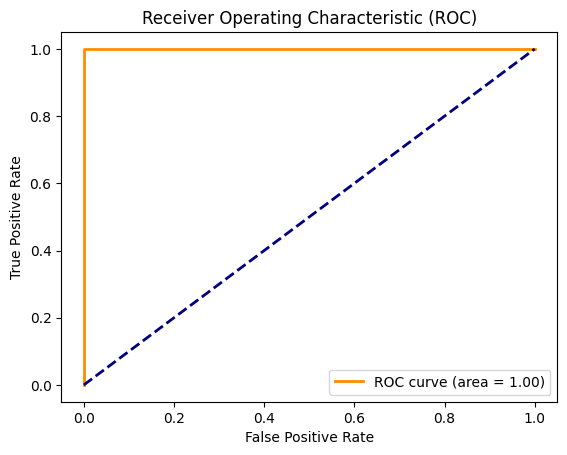


Рисунок Х – ROC-кривая

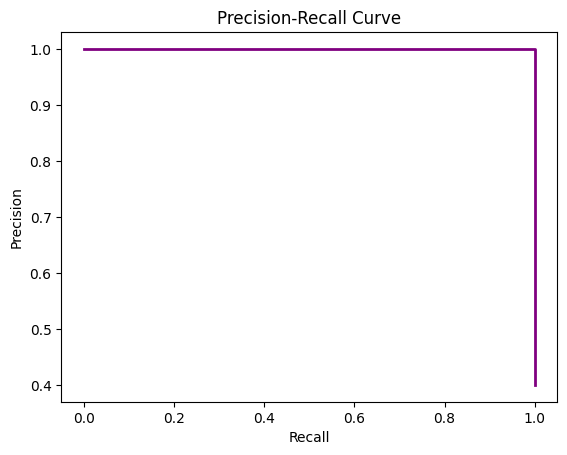


Рисунок Х – Precision-Recall Curve

На рисунке Х представлен отчёт классификации для обученной модели.



Рисунок Х – Отчёт классификации для CNN

1. **AutoML в FEDOT**

Для построения AutoML решения с помощью FEDOT был реализован пайплайн, представленный на рисунке X.



Рисунок Х – Pipline на основе FEDOT

Код программы для FEDOT:

import os

import numpy as np

import cv2

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import (classification\_report, confusion\_matrix, roc\_curve, auc,

                             precision\_recall\_curve)

import matplotlib.pyplot as plt

# Функция для получения метки по пути к изображению

def get\_id\_by\_path(path):

    class\_name = path.split(os.sep)[-2]

    return 0 if class\_name == 'Healthy' else 1

# Простое извлечение признаков: изменение размера и развёртка в вектор

def load\_and\_flatten\_image(path, size=(128, 128)):

    img = cv2.imread(path)

    if img is None:

        raise ValueError(f'Не удалось загрузить изображение: {path}')

    img = cv2.resize(img, size)

    # Нормализуем и разворачиваем в 1D-вектор

    img\_flat = img.flatten() / 255.0

    return img\_flat

X = []

y = []

for path in imgs:

    X.append(load\_and\_flatten\_image(path))

    y.append(get\_id\_by\_path(path))

X = np.array(X)

y = np.array(y)

# Разбиваем данные на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Импортируем FEDOT

from fedot.api.main import Fedot

# Инициализируем AutoML-модель для задачи классификации

automl\_model = Fedot(problem='classification', timeout=5,preset='best\_quality', n\_jobs=4)  # timeout в минутах (или задайте нужное время)

automl\_model.fit(features=X\_train, target=y\_train)

predictions = automl\_model.predict(X\_test)

print("Отчёт по классификации (FEDOT):")

print(classification\_report(y\_test, predictions, target\_names=['Healthy', 'Seizure']))

# Построим матрицу ошибок

conf\_matrix = confusion\_matrix(y\_test, predictions)

plt.figure()

plt.imshow(conf\_matrix, interpolation='nearest', cmap=plt.cm.Blues)

plt.title("Confusion Matrix (FEDOT)")

plt.colorbar()

tick\_marks = np.arange(2)

plt.xticks(tick\_marks, ['Healthy', 'Seizure'])

plt.yticks(tick\_marks, ['Healthy', 'Seizure'])

plt.xlabel("Predicted")

plt.ylabel("True")

plt.show()

# Получаем вероятности для положительного класса

try:

    probs = automl\_model.predict\_proba(X\_test)[:, 1]

except:

    # Если нет, можно сделать softmax от предсказанных "scores"

    probs = predictions

# ROC-кривая

fpr, tpr, \_ = roc\_curve(y\_test, probs)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve (FEDOT)")

plt.legend(loc="lower right")

plt.show()

# Precision-Recall кривая

precision, recall, \_ = precision\_recall\_curve(y\_test, probs)

plt.figure()

plt.plot(recall, precision, color='purple', lw=2)

plt.xlabel("Recall")

plt.ylabel("Precision")

plt.title("Precision-Recall Curve (FEDOT)")

plt.show()

Отчёт по классификации после получения AutoML решения с помощью FEDOT представлен на рисунке Х.

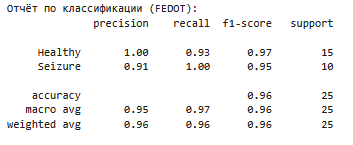


Рисунок Х – Результаты FEDOT

1. **AutoML в LAMA**

Для получения AutoML решения в LAMA был реализован pipeline, представленный на рисунке Х.



Рисунок Х – Pipline на основе LAMA

Код программы для LAMA:

import os

import cv2

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import (

    confusion\_matrix, roc\_curve, auc,

    precision\_recall\_curve, classification\_report

)

from lightautoml.automl.presets.tabular\_presets import TabularAutoML

from lightautoml.tasks import Task

# Функция для получения метки (имени класса) из пути

def get\_label\_from\_path(path):

    # Имя класса считается именем родительской папки

    return os.path.basename(os.path.dirname(path))

# Функция для загрузки и преобразования изображения в вектор признаков

def load\_and\_flatten\_image(path, size=(128, 128)):

    img = cv2.imread(path)

    if img is None:

        raise ValueError(f"Не удалось загрузить изображение: {path}")

    img = cv2.resize(img, size)

    # Приводим к типу float32, нормализуем пиксели (0-1) и возвращаем изображение (без разворота вектором)

    # Если требуется вектор, вызовите .flatten() позже

    return img.astype(np.float32) / 255.0

# Извлекаем признаки и метки для каждого изображения

features = []

labels = []

for path in imgs:

    # Загружаем изображение, изменяем размер, нормализуем и превращаем в одномерный вектор

    img\_vector = load\_and\_flatten\_image(path, size=(128, 128)).flatten()

    features.append(img\_vector)

    labels.append(get\_label\_from\_path(path))

features = np.array(features)

labels = np.array(labels)

# Приводим метки к числовому виду: 'Healthy' -> 0, 'Seizure' -> 1

class\_mapping = {'Healthy': 0, 'Seizure': 1}

y = np.array([class\_mapping[label] for label in labels])

# Создаем DataFrame с признаками; каждому признаку присваиваем имя feat\_0, feat\_1, ..., feat\_{n-1}

num\_features = features.shape[1]

feature\_columns = [f'feat\_{i}' for i in range(num\_features)]

df = pd.DataFrame(features, columns=feature\_columns)

df['target'] = y

# Разбиваем данные на обучающую и тестовую выборки

df\_train, df\_test = train\_test\_split(df, test\_size=0.2, random\_state=42)

# Определяем задачу: бинарная классификация

task = Task('binary')

# Создаем AutoML-модель LightAutoML с пресетом для табличных данных

automl = TabularAutoML(task=task, timeout=5)  # timeout задается в минутах

# Важно: явно задаем роли, чтобы указать, что столбец 'target' является целевой

roles = {'target': 'target'}

# Обучаем модель и делаем предсказания

oof\_pred = automl.fit\_predict(df\_train, roles=roles)

test\_pred = automl.predict(df\_test)

# Извлекаем вероятности для положительного класса (предполагается, что второй столбец содержит их)

probs = probs = test\_pred.data[:, 0]

# Получаем бинарные предсказания с порогом 0.5

predictions = (probs > 0.5).astype(int)

# Выводим отчет по классификации

print("Отчет по классификации (LightAutoML):")

print(classification\_report(df\_test['target'], predictions, target\_names=['Healthy', 'Seizure']))

# --- Построение графиков ---

# 1. Матрица ошибок

cm = confusion\_matrix(df\_test['target'], predictions)

plt.figure()

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.title("Confusion Matrix (LightAutoML)")

plt.colorbar()

tick\_marks = np.arange(2)

plt.xticks(tick\_marks, ['Healthy', 'Seizure'])

plt.yticks(tick\_marks, ['Healthy', 'Seizure'])

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# 2. ROC-кривая

fpr, tpr, \_ = roc\_curve(df\_test['target'], probs)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve (LightAutoML)")

plt.legend(loc="lower right")

plt.show()

# 3. Precision-Recall кривая

precision, recall, \_ = precision\_recall\_curve(df\_test['target'], probs)

plt.figure()

plt.plot(recall, precision, color='purple', lw=2)

plt.xlabel("Recall")

plt.ylabel("Precision")

plt.title("Precision-Recall Curve (LightAutoML)")

plt.show()

Отчёт по классификации после получения AutoML решения с помощью LAMA представлен на рисунке Х.

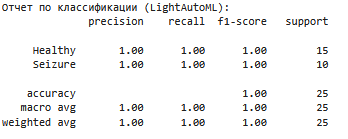


Рисунок Х - Рисунок Х – Результаты LAMA

В таблице Х представлены результаты работы всех полученных моделей.

Таблица Х – Результаты моделей

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **CNN** | **FEDOT** | **LAMA** |
| f1-score | 1 | 0.96 | 1 |
| macro avg precision | 1 | 0.95 | 1 |
| macro avg recall | 1 | 0.97 | 1 |
| Weighted avg precision | 1 | 0.96 | 1 |
| Weighted avg recall | 1 | 0.96 | 1 |

Из результатов можно сделать вывод, что лучше всех себя показали свёрточная нейронная сеть и AutoML решения на LAMA.

**Лабораторная работа № 4**

После обучения и тестирования модели, выполненных в лабораторной работе №3, модель была развёрнута в MLFlow:

mlflow models serve -m "runs:/<run\_id>/model" --no-conda -p 5000

Результаты развёртывания показаны на рисунке Х1 и Х2.

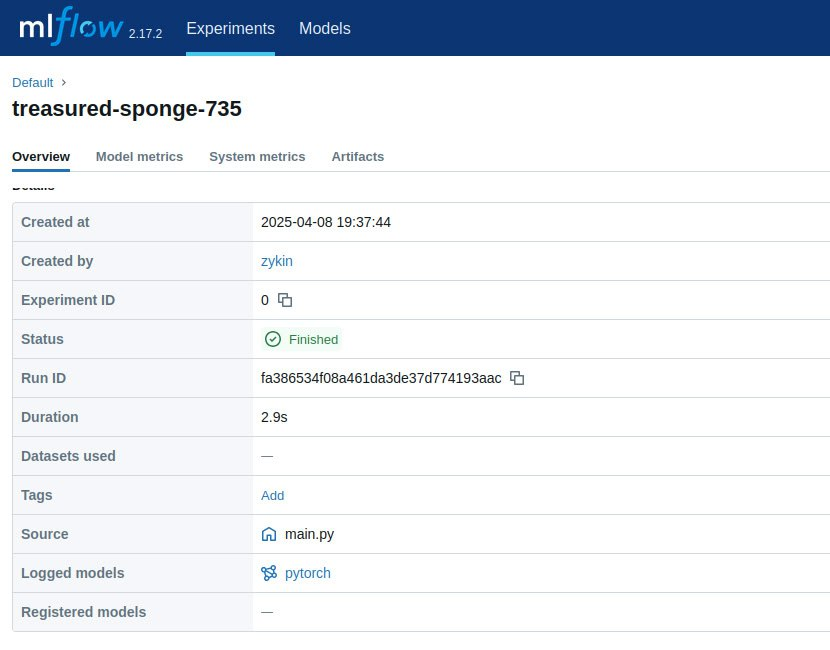


Рисунок X1 – Развёртывание модели в MLFlow

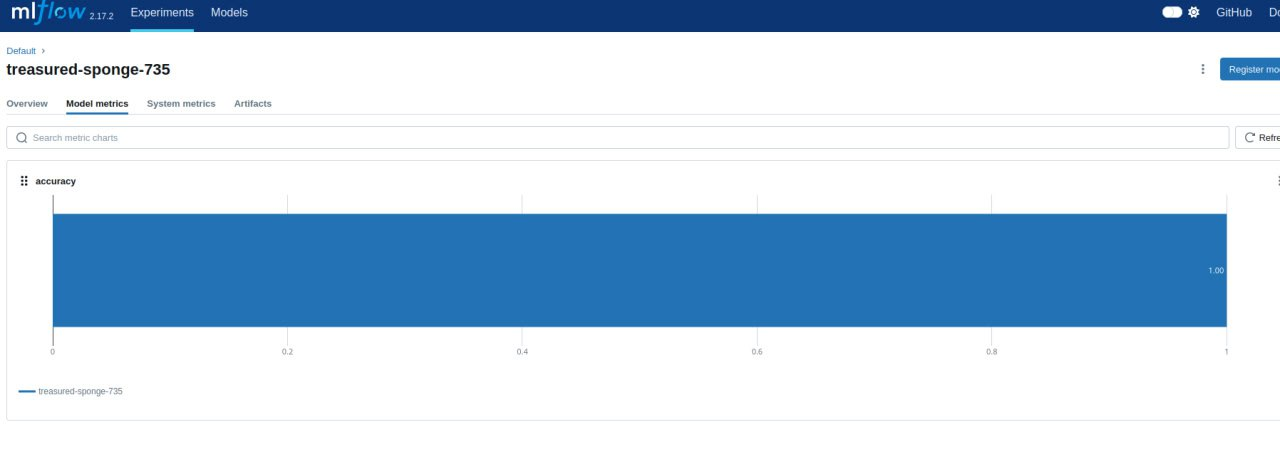


Рисунок X1 – Развёртывание модели MLFlow 2

**Вывод**: в результате данной работы был сформирован датасет для классификации ЭКГ-записей. Было обучено 3 модели: CNN с помощью PyTorch и два AutoML решения: FEDOT и LAMA. Точнее всех выступила обученная CNN. Затем модель была сохранена, протестирована и развёрнута в MLFlow.

**Задача 6**

**Цель задания:** исследование когнитивных характеристик мозга

**Ключевой навык:** алгоритмы внимания в нейронных сетях

**Датасет:** Необходимо перейти на страницу датасета, расположенного по адресу: https://www.physionet.org/content/mitdb/1.0.0/.

**Задание:**

1. Сформируйте тренировочный датасет.
2. Постройте нейронную сеть с эффектом памяти (LSTM) и примените механизм внимания,
3. Обучите ее классифицировать классы по заданной ЭКГ. Постройте матрицу ошибок (confusion matrix).
4. Проведите эксперименты по заданию на AutoML решении. Сравните результаты.

**Лабораторная работа № 2**

Для формирования датасета были использованы таблицы в формате «.сsv» с сайта Kaggle. Из-за большого количества данных они были урезаны, последний столбец был переименован в «сlass».

Сформированный тренировочная выборка была несбалансирована, поэтому был использован RandomUnderSampler, чтобы решить эту проблему. Кроме того, были удалены данные, содержащие пустые значения. В итоге получился датасет, пригодный для обучения модели. Код формирования датасета представлен ниже:

#подключаем гугл-диск для загрузки датасета

from google.colab import drive

drive.mount('/content/drive')

file\_path\_train = '/content/drive/MyDrive/Example6/mitbih\_train.csv'

file\_path\_test = '/content/drive/MyDrive/Example6/mitbih\_test.csv'

#датасет большой, поэтому сформируем случайным образом тренировочный датасет

import random

n = 80000 #number of records in file

s = 50000 #desired sample size

skip = sorted(random.sample(range(n),n-s))

df\_mitbih\_train = pd.read\_csv(file\_path\_train, header=None, skiprows=skip)

df\_mitbih\_test = pd.read\_csv(file\_path\_test, header=None, skiprows=skip)

df\_mitbih = pd.concat([df\_mitbih\_train, df\_mitbih\_test], axis=0)

# добавим текстовое описание к размеченным классам

id\_to\_label = {

    0: "Normal",     #0: "Нормальный"

    1: "Artial Premature", # "Преждевременное сокращение сердца (экстрасистолия)"

    2: "Premature ventricular contraction", # "Преждевременное сокращение желудочков"

    3: "Fusion of ventricular and normal",  # "Cлияние желудочкового и нормального сердечного ритма"

    4: "Fusion of paced and normal" #"Cлияние стимулированного и нормального ритма"

}

df\_mitbih['label'] = df\_mitbih.iloc[:, -1].map(id\_to\_label)

df\_mitbih\_test['label'] = df\_mitbih\_test.iloc[:, -1].map(id\_to\_label)

df\_mitbih\_train['label'] = df\_mitbih\_train.iloc[:, -1].map(id\_to\_label)

#создаем процедуру перемешивания данных и их балансировки

from sklearn.utils import shuffle

df\_mitbih\_test = shuffle(df\_mitbih\_test)

df\_mitbih\_train = shuffle(df\_mitbih\_train)

df\_mitbih\_test = df\_mitbih\_test.reset\_index(drop='True')

df\_mitbih\_train = df\_mitbih\_train.reset\_index(drop='True')

from imblearn.under\_sampling import RandomUnderSampler

# Создание экземпляра RandomUnderSampler

rus\_method = RandomUnderSampler()

df = df\_mitbih\_train

# Применение уменьшения выборки к данным

df.columns = df.columns.astype(str)

X\_resampled, y\_resampled = rus\_method.fit\_resample(df, df['class'])

new\_df =pd.concat([X\_resampled, y\_resampled])

#не забываем удалить NaN

new\_df = new\_df.dropna()

new\_df['class'].value\_counts()

#получаем взвешенную выборку

**Лабораторная работа № 3**

В рамках поставленной задачи необходимо классифицировать классы по заданной ЭКГ. Так как классов несколько, это относится к задаче многоклассовой классификации. Для данной задачи были выбраны следующие основные метрики оценки модели: Матрица ошибок, Precision, Recall, f1-score (взвешенный), и общая точность.

1. **LSTM с механизмом внимания**

Для обучения LSTM с механизмом внимания был использован фреймворк PyTorch. Были разработаны датасет и архитекутра модели, включая в себя LSTM и механизм внимания. Для улучшения качества обучения в качестве оптимизатора был использован AdamW с регуляризацией, кроме того, использовалась косинусная функция уменьшения размера шага. Код обучения:

import torch

import torch.nn as nn

import torch.optim as optim

import torch.nn.functional as F

from torch.optim.lr\_scheduler import CosineAnnealingLR

from torch.utils.data import Dataset, DataLoader

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report, roc\_curve, auc

# Функция для преобразования DataFrame в датасет

class ECGDataset(Dataset):

    def \_\_init\_\_(self, features\_df, labels\_series):

        self.X = features\_df.values.astype(np.float32)  # (N, 187)

        self.y = labels\_series.values.astype(np.int64)    # (N,)

    def \_\_len\_\_(self):

        return len(self.X)

    def \_\_getitem\_\_(self, idx):

        sample = self.X[idx].reshape(-1, 1)  # преобразуем (187,) в (187, 1)

        label = self.y[idx]

        return sample, label

# Подготовка данных

X\_train = pd.DataFrame(df\_mitbih\_train.iloc[:, :187].values,

                       columns=[str(i) for i in range(187)]).reset\_index(drop=True)

X\_val = pd.DataFrame(df\_mitbih\_test.iloc[:, :187].values,

                     columns=[str(i) for i in range(187)]).reset\_index(drop=True)

# Извлекаем целевую переменную

y\_train = pd.Series(df\_mitbih\_train.iloc[:, 187].values).astype(int).reset\_index(drop=True)

y\_val = pd.Series(df\_mitbih\_test.iloc[:, 187].values).astype(int).reset\_index(drop=True)

# Создаем датасеты и DataLoader'ы

train\_dataset = ECGDataset(X\_train, y\_train)

val\_dataset = ECGDataset(X\_val, y\_val)

batch\_size = 32

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

val\_loader = DataLoader(val\_dataset, batch\_size=batch\_size, shuffle=False)

# Модель LSTM с механизмом внимания

class Attention(nn.Module):

    def \_\_init\_\_(self, hidden\_dim):

        super(Attention, self).\_\_init\_\_()

        self.attention = nn.Linear(hidden\_dim, 1)  # Преобразует скрытое состояние в скалярную оценку

    def forward(self, lstm\_outputs):

        # lstm\_outputs: (batch, seq\_len, hidden\_dim)

        scores = self.attention(lstm\_outputs)                # (batch, seq\_len, 1)

        attn\_weights = torch.softmax(scores, dim=1)           # (batch, seq\_len, 1)

        context = torch.sum(attn\_weights \* lstm\_outputs, dim=1) # (batch, hidden\_dim)

        return context, attn\_weights

class LSTMAttentionClassifier(nn.Module):

    def \_\_init\_\_(self, input\_dim=1, hidden\_dim=64, num\_layers=2, num\_classes=5, dropout=0.5):

        super(LSTMAttentionClassifier, self).\_\_init\_\_()

        self.lstm = nn.LSTM(input\_dim, hidden\_dim, num\_layers,

                            batch\_first=True, dropout=dropout, bidirectional=False)

        self.attention = Attention(hidden\_dim)

        self.fc = nn.Linear(hidden\_dim, num\_classes)

    def forward(self, x):

        # x: (batch, seq\_len, input\_dim) -> (batch, 187, 1)

        lstm\_out, \_ = self.lstm(x)                         # (batch, 187, hidden\_dim)

        context, attn\_weights = self.attention(lstm\_out)     # (batch, hidden\_dim)

        logits = self.fc(context)                          # (batch, num\_classes)

        return logits, attn\_weights

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model = LSTMAttentionClassifier(input\_dim=1, hidden\_dim=64, num\_layers=3, num\_classes=5, dropout=0.4)

model.to(device)

# Обучение

criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(model.parameters(), lr=0.001, weight\_decay=1e-4)

num\_epochs = 500

scheduler = CosineAnnealingLR(optimizer, T\_max=num\_epochs, eta\_min=1e-6)

train\_losses = []

train\_accuracies = []

val\_losses = []

val\_accuracies = []

for epoch in range(num\_epochs):

    # Обучение

    model.train()

    running\_loss = 0.0

    running\_correct = 0

    total\_train = 0

    for inputs, labels in train\_loader:

        inputs = inputs.to(device)   # (batch, 187, 1)

        labels = labels.to(device)

        optimizer.zero\_grad()

        outputs, \_ = model(inputs)     # outputs: (batch, num\_classes)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item() \* inputs.size(0)

        \_, predicted = torch.max(outputs, 1)

        total\_train += labels.size(0)

        running\_correct += (predicted == labels).sum().item()

    epoch\_train\_loss = running\_loss / total\_train

    epoch\_train\_acc = running\_correct / total\_train

    train\_losses.append(epoch\_train\_loss)

    train\_accuracies.append(epoch\_train\_acc)

    # Валидация

    model.eval()

    val\_loss = 0.0

    correct\_val = 0

    total\_val = 0

    with torch.no\_grad():

        for inputs, labels in val\_loader:

            inputs = inputs.to(device)

            labels = labels.to(device)

            outputs, \_ = model(inputs)

            loss = criterion(outputs, labels)

            val\_loss += loss.item() \* inputs.size(0)

            \_, predicted = torch.max(outputs, 1)

            total\_val += labels.size(0)

            correct\_val += (predicted == labels).sum().item()

    epoch\_val\_loss = val\_loss / total\_val

    epoch\_val\_acc = correct\_val / total\_val

    val\_losses.append(epoch\_val\_loss)

    val\_accuracies.append(epoch\_val\_acc)

    scheduler.step()

    print(f"Epoch {epoch+1}/{num\_epochs}: "

          f"Train Loss = {epoch\_train\_loss:.4f}, Train Acc = {epoch\_train\_acc:.4f} | "

          f"Val Loss = {epoch\_val\_loss:.4f}, Val Acc = {epoch\_val\_acc:.4f}")

# Графики обучения

plt.figure(figsize=(14,5))

plt.subplot(1,2,1)

plt.plot(range(1, num\_epochs+1), train\_losses, label='Train Loss', marker='o')

plt.plot(range(1, num\_epochs+1), val\_losses, label='Val Loss', marker='o')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.title('Loss vs Epochs')

plt.legend()

plt.subplot(1,2,2)

plt.plot(range(1, num\_epochs+1), train\_accuracies, label='Train Acc', marker='o')

plt.plot(range(1, num\_epochs+1), val\_accuracies, label='Val Acc', marker='o')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.title('Accuracy vs Epochs')

plt.legend()

plt.show()

# Оценка на тестовом наборе, построение confusion matrix, ROC-кривых и classification report

model.eval()

all\_preds = []

all\_true = []

all\_probs = []

with torch.no\_grad():

    for inputs, labels in val\_loader:

        inputs = inputs.to(device)

        labels = labels.to(device)

        outputs, \_ = model(inputs)

        probs = F.softmax(outputs, dim=1)

        \_, predicted = torch.max(outputs, 1)

        all\_preds.extend(predicted.cpu().numpy())

        all\_true.extend(labels.cpu().numpy())

        all\_probs.extend(probs.cpu().numpy())

cm = confusion\_matrix(all\_true, all\_preds)

acc = accuracy\_score(all\_true, all\_preds)

report = classification\_report(all\_true, all\_preds, digits=4)

print("Confusion Matrix:")

print(cm)

print(f"Test Accuracy: {acc:.4f}")

print("Classification Report:")

print(report)

plt.figure(figsize=(8,6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel("Predicted")

plt.ylabel("True")

plt.title("Confusion Matrix")

plt.show()

# ROC-кривые для мультклассовой классификации:

# Для ROC нужно бинараизовать метки.

from sklearn.preprocessing import label\_binarize

n\_classes = 5

all\_true\_np = np.array(all\_true)

all\_probs\_np = np.array(all\_probs)  # shape: (n\_samples, 5)

all\_true\_bin = label\_binarize(all\_true\_np, classes=range(n\_classes))

plt.figure(figsize=(10,8))

for i in range(n\_classes):

    fpr, tpr, \_ = roc\_curve(all\_true\_bin[:, i], all\_probs\_np[:, i])

    roc\_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, lw=2, label=f'Class {i} (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curves for ECG Classification')

plt.legend(loc="lower right")

plt.show()

На рисунках Х1-ХN представлены графики обучения, матрица ошибок, ROC-кривая и отчёт по классификации с основными метриками

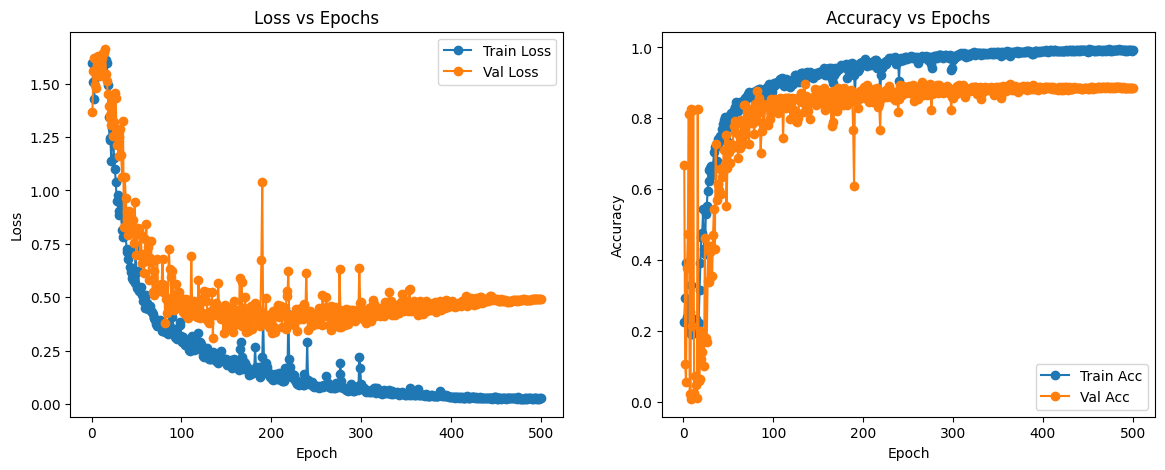


Рисунок Х – Графики обучения

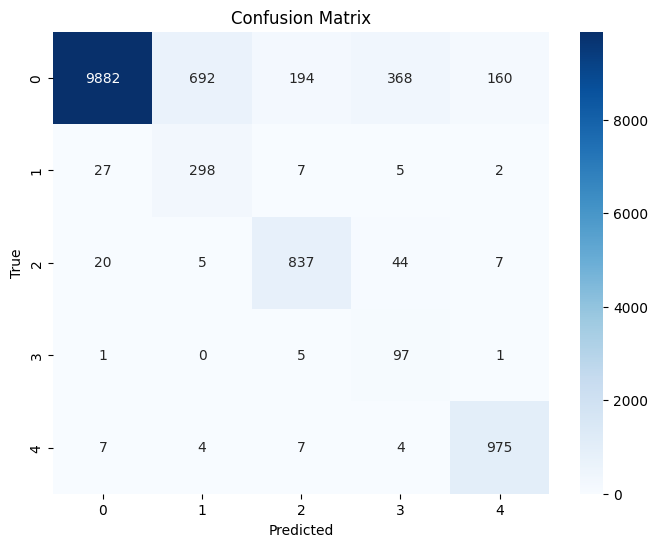


Рисунок Х – Матрица ошибок

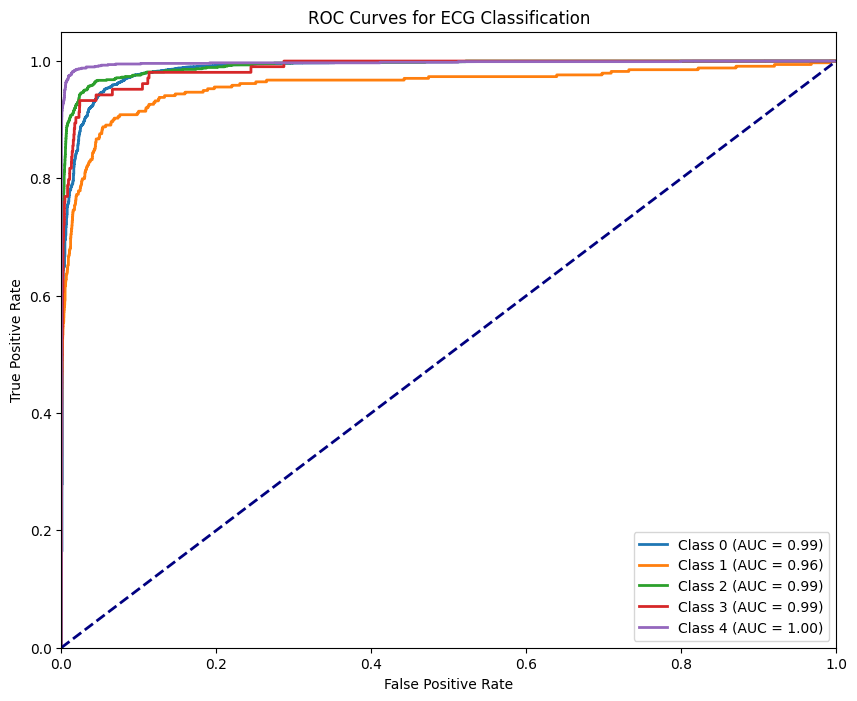


Рисунок Х – ROC-кривая

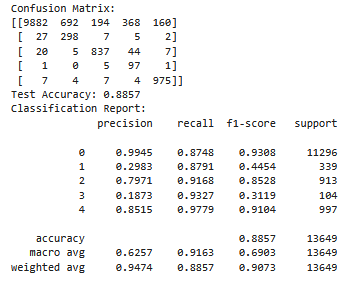


Рисунок Х – Отчёт классификации

1. **FEDOT**

Для обучения модели с помощью AutoML решения FEDOT был реализован следующий пайплайн, представленный на рисунке Х.



Код обучения на FEDOT:

from fedot.api.main import Fedot

from sklearn.metrics import (confusion\_matrix, accuracy\_score, classification\_report,

                             precision\_score, recall\_score, f1\_score, cohen\_kappa\_score, matthews\_corrcoef)

import numpy as np

import numpy as np

# Извлекаем признаки и метки из DataFrame

X\_train = df\_mitbih\_train.iloc[:, :187].values

y\_train = df\_mitbih\_train['label'].values

X\_test = df\_mitbih\_test.iloc[:, :187].values

y\_test = df\_mitbih\_test['label'].values

# Задаём тип задачи: мультклассовая классификация

automl\_fedot = Fedot(problem='classification', timeout=15, cv\_folds=5, n\_jobs=4)

automl\_fedot.fit(features=X\_train, target=y\_train)

preds\_fedot = automl\_fedot.predict(features=X\_test)

preds\_fedot = np.array(preds\_fedot.data)

# Оценим результат

cm\_fedot = confusion\_matrix(y\_test, preds\_fedot)

acc\_fedot = accuracy\_score(y\_test, preds\_fedot)

precision\_fedot = precision\_score(y\_test, preds\_fedot, average='macro')

recall\_fedot = recall\_score(y\_test, preds\_fedot, average='macro')

f1\_fedot = f1\_score(y\_test, preds\_fedot, average='macro')

report\_fedot = classification\_report(y\_test, preds\_fedot, digits=4)

print("Confusion Matrix:\n", cm\_fedot)

print("Accuracy: {:.4f}".format(acc\_fedot))

print("Precision (macro): {:.4f}".format(precision\_fedot))

print("Recall (macro): {:.4f}".format(recall\_fedot))

print("F1 Score (macro): {:.4f}".format(f1\_fedot))

print("Classification Report:\n", report\_fedot)

Результаты обучения представлены на рисунке Х.

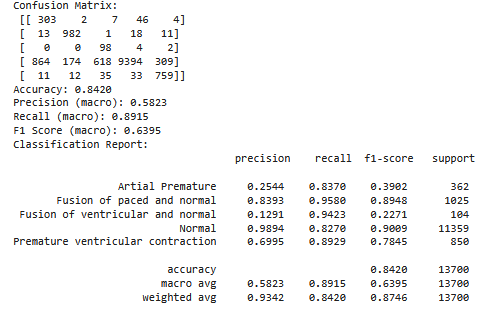


Рисунок Х – Отчёт по классификации

1. **LAMA**

Для обучения модели в LAMA был реализован пайплайн, аналогичный для FEDOT. Код обучения представлен ниже:

import pandas as pd

import numpy as np

from lightautoml.automl.presets.tabular\_presets import TabularAutoML

from lightautoml.tasks import Task

from sklearn.metrics import (

    confusion\_matrix, accuracy\_score, classification\_report,

    precision\_score, recall\_score, f1\_score, cohen\_kappa\_score, matthews\_corrcoef

)

# Создаем DataFrame для признаков с именами столбцов-строк.

X\_train = pd.DataFrame(df\_mitbih\_train.iloc[:, :187].values,

                       columns=[str(i) for i in range(187)]).reset\_index(drop=True)

X\_test  = pd.DataFrame(df\_mitbih\_test.iloc[:, :187].values,

                       columns=[str(i) for i in range(187)]).reset\_index(drop=True)

# Извлекаем целевую переменную

y\_train = pd.Series(df\_mitbih\_train.iloc[:, 187].values).astype(int).reset\_index(drop=True)

y\_test  = pd.Series(df\_mitbih\_test.iloc[:, 187].values).astype(int).reset\_index(drop=True)

# Задаем имя для Series target – это важно для ролей.

y\_train.name = 'target'

y\_test.name  = 'target'

# Объединяем признаки и target в один DataFrame для обучения.

df\_train = X\_train.copy()

df\_train['target'] = y\_train

df\_test = X\_test.copy()

# Определяем задачу – мультклассовая классификация

task = Task('multiclass')

# Создаем объект LightAutoML

automl\_lama = TabularAutoML(task=task, timeout=15, cpu\_limit=4)

# Обучаем модель

oof\_preds = automl\_lama.fit\_predict(df\_train, roles={'target': 'target'}).data

# Выполняем предсказание для тестовых данных.

test\_preds\_obj = automl\_lama.predict(df\_test)

# Преобразуем вероятностные предсказания в финальные классы.

test\_preds = np.argmax(test\_preds\_obj.data, axis=1)

# Метрики

cm = confusion\_matrix(y\_test, test\_preds)

acc = accuracy\_score(y\_test, test\_preds)

precision\_macro = precision\_score(y\_test, test\_preds, average='macro')

recall\_macro = recall\_score(y\_test, test\_preds, average='macro')

f1\_macro = f1\_score(y\_test, test\_preds, average='macro')

report = classification\_report(y\_test, test\_preds, digits=4)

print("Confusion Matrix:\n", cm)

print("Accuracy: {:.4f}".format(acc))

print("Precision (macro): {:.4f}".format(precision\_macro))

print("Recall (macro): {:.4f}".format(recall\_macro))

print("F1 Score (macro): {:.4f}".format(f1\_macro))

print("Classification Report:\n", report)

Результаты обучения представлены на рисунке Х.

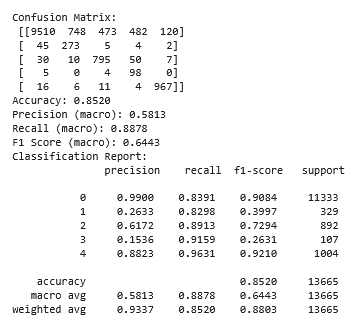


Рисунок Х – Результаты обучения в LAMA

В таблице Х представлено сравнение моделей из разных решений.

Таблица Х – Сравнение результатов

|  |  |  |  |
| --- | --- | --- | --- |
| Метрика | LSTM + внимание | FEDOT | LAMA |
| F1(macro) | 0.6903 | 0.6395 | 0.6443 |
| Precision (macro) | 0.6257 | 0.5823 | 0.5813 |
| Recall (macro) | 0.9163 | 0.8915 | 0.8878 |
| Accuracy | 0.8857 | 0.8420 | 0.8520 |
| Матрица ошибок | [[9882 692 194 368 160]  [ 27 298 7 5 2]  [ 20 5 837 44 7]  [ 1 0 5 97 1]  [ 7 4 7 4 975]] | [[ 303 2 7 46 4]  [ 13 982 1 18 11]  [ 0 0 98 4 2]  [ 864 174 618 9394 309]  [ 11 12 35 33 759]] | [[9510 748 473 482 120]  [ 45 273 5 4 2]  [ 30 10 795 50 7]  [ 5 0 4 98 0]  [ 16 6 11 4 967]] |

Из таблицы видно, что лучше всего себя показала модель LSTM с механизмом внимания. FEDOT и LAMA показывают сопоставимые результаты. Кроме того, по матрице ошибок можно заметить, что модель на FEDOT преимущественно предсказывает 3 класс («Преждевременное сокращение желудочков»), в то время как LSTM и LAMA преимущественно 0 класс («Нормальный»). В условиях, когда намного предпочтительнее ошибочно указать на имеющиеся проблемы, чем ошибочно указать на «нормальное» состояние, вариант с FEDOT может быть предпочтительнее, но всё зависит от целей, для которых используется модель.

**Лабораторная работа №4**

После обучения модель была сохранена и протестирована:

# Оценка на тестовом наборе, построение confusion matrix, ROC-кривых и classification report

model.eval()

all\_preds = []

all\_true = []

all\_probs = []

with torch.no\_grad():

    for inputs, labels in val\_loader:

        inputs = inputs.to(device)

        labels = labels.to(device)

        outputs, \_ = model(inputs)

        probs = F.softmax(outputs, dim=1)

        \_, predicted = torch.max(outputs, 1)

        all\_preds.extend(predicted.cpu().numpy())

        all\_true.extend(labels.cpu().numpy())

        all\_probs.extend(probs.cpu().numpy())

cm = confusion\_matrix(all\_true, all\_preds)

acc = accuracy\_score(all\_true, all\_preds)

report = classification\_report(all\_true, all\_preds, digits=4)

print("Confusion Matrix:")

print(cm)

print(f"Test Accuracy: {acc:.4f}")

print("Classification Report:")

print(report)

plt.figure(figsize=(8,6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel("Predicted")

plt.ylabel("True")

plt.title("Confusion Matrix")

plt.show()

# ROC-кривые для мультклассовой классификации:

# Для ROC нужно бинараизовать метки.

from sklearn.preprocessing import label\_binarize

n\_classes = 5

all\_true\_np = np.array(all\_true)

all\_probs\_np = np.array(all\_probs)  # shape: (n\_samples, 5)

all\_true\_bin = label\_binarize(all\_true\_np, classes=range(n\_classes))

plt.figure(figsize=(10,8))

for i in range(n\_classes):

    fpr, tpr, \_ = roc\_curve(all\_true\_bin[:, i], all\_probs\_np[:, i])

    roc\_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, lw=2, label=f'Class {i} (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curves for ECG Classification')

plt.legend(loc="lower right")

plt.show()

На рисунках Х1-Х2 представлены полученная матрица ошибок и ROC-кривая.

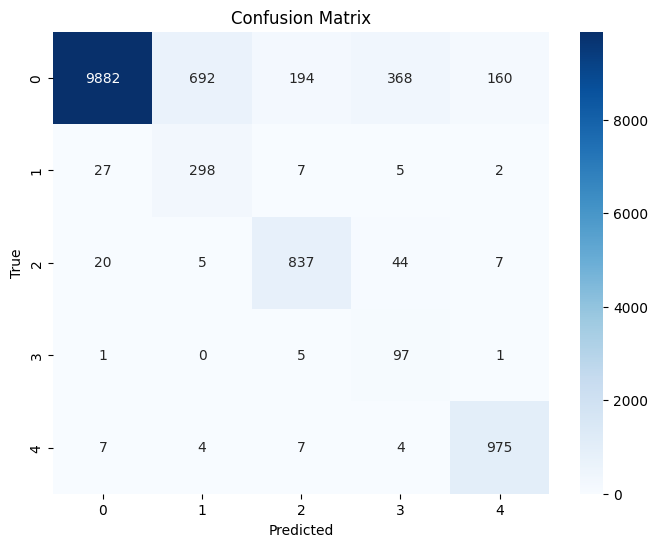


Рисунок Х – Матрица ошибок

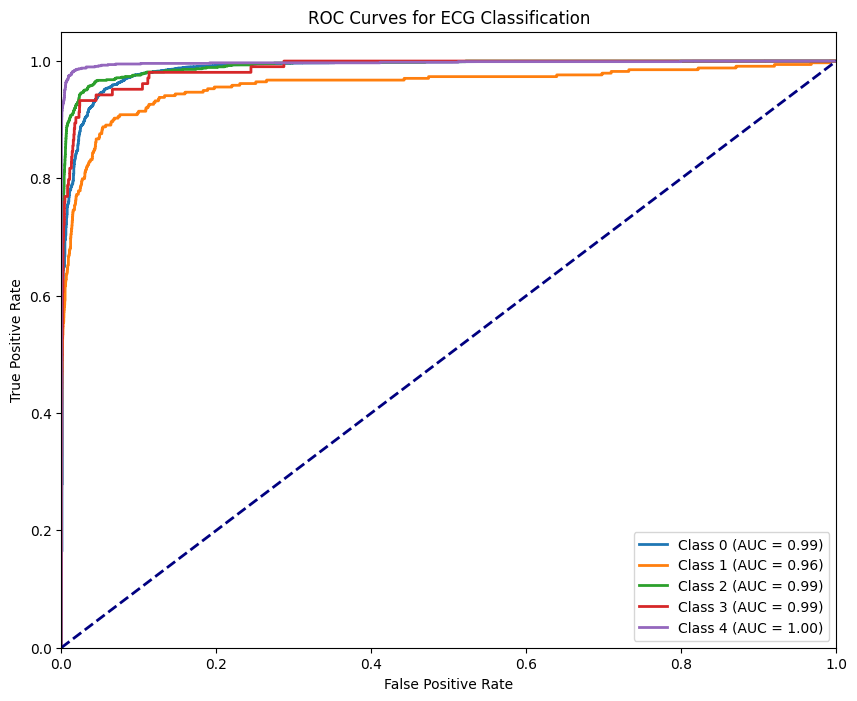


Рисунок Х – ROC-кривая

После тестирования модель была развёрнута в MLFlow, что показано на рисунках Х1 и Х2.

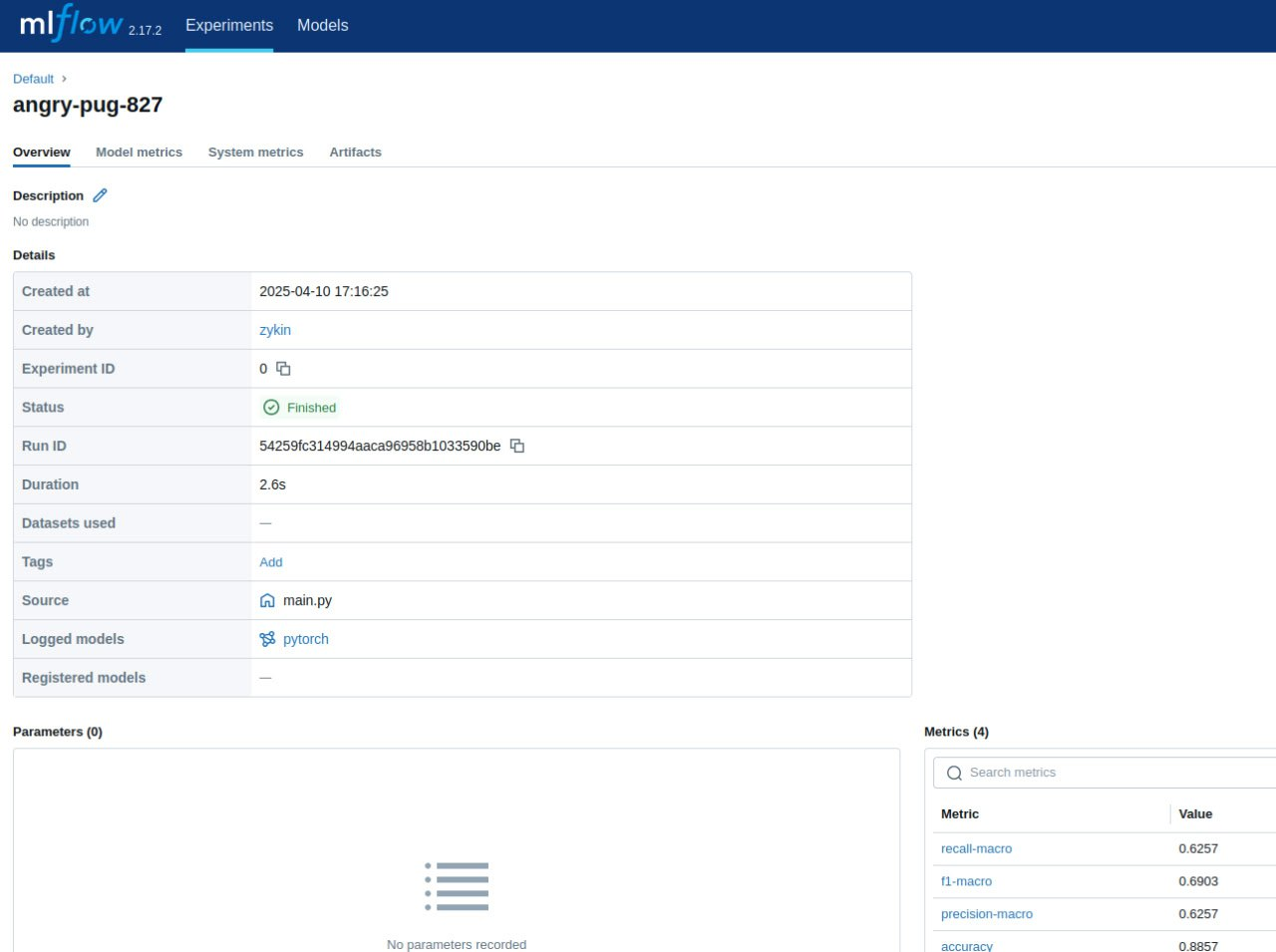


Рисунок Х - Результат развёртывания в MLFlow

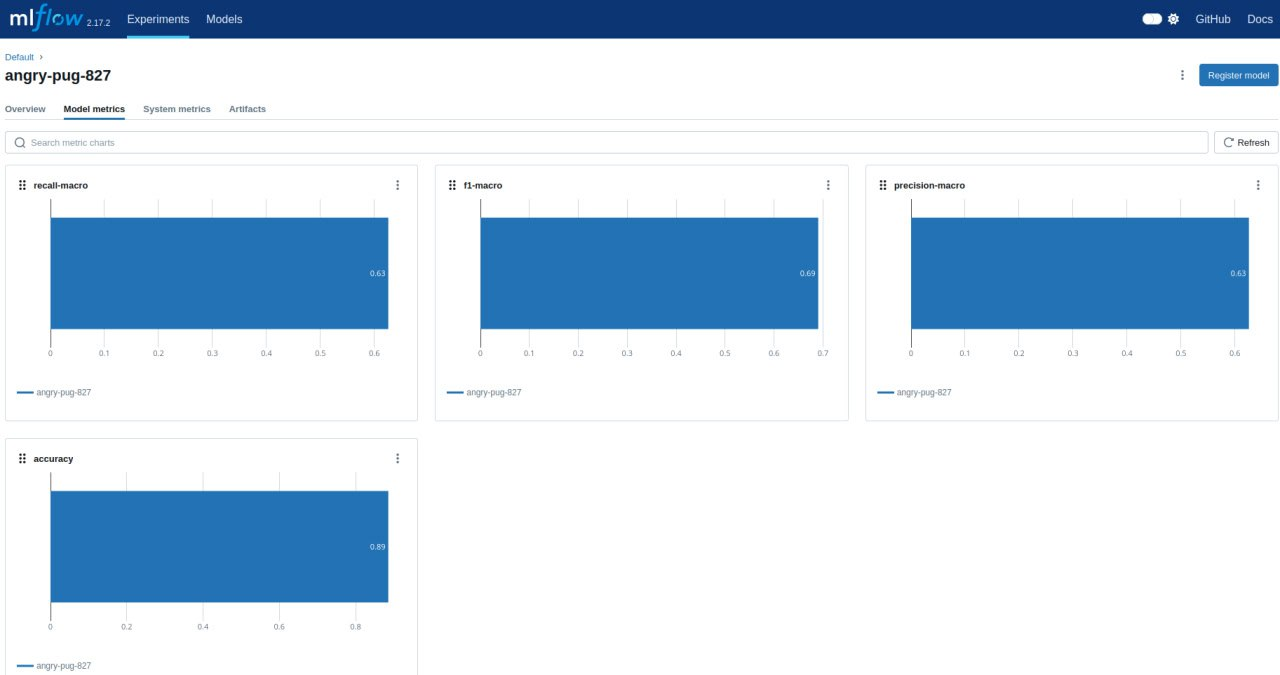


Рисунок Х - Результат развёртывания в MLFlow 2

**Вывод**: в результате данной работы был сформирован датасет для классификации ЭКГ-записей. Было обучено 3 модели: LSTM с механизмом внимания с помощью PyTorch и два AutoML решения: FEDOT и LAMA. Точнее всех выступила обученная LSTM. Затем модель была сохранена, протестирована и развёрнута в MLFlow.