МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра теоретических основ компьютерной безопасности и криптографии

ОБНАРУЖЕНИЕ АНОМАЛИЙ В МУЛЬТИВАРИАТИВНЫХ ВРЕМЕННЫХ РЯДАХ МЕТОДАМИ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

ДИПЛОМНАЯ РАБОТА

студента 6 курса 631 группы
специальности 100501 — Компьютерная безопасность
факультета КНиИТ
Улитина Ивана Владимировича

Научный руководитель
доцент _____ Слеповичев И. И.

Заведующий кафедрой
д.ф.-м.н., доцент _____ Абросимов М. Б.

Саратов 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Временным рядом называют последовательность значений, упорядоченных по времени, в которое то или иное значение было получено/зафиксировано. В качестве примеров временного ряда можно выделить:

1. курсы валют/акций (стоимость валюты/акции в конкретный момент времени);
2. значение прогноза погоды, параметры работы двигателя (изменяющееся с течением времени значение некоторой физической величины);
3. электрокардиограмма (биологический параметр человека);
4. передаваемый объём сетевого трафика.

Временной ряд обладает типовыми характеристиками (иногда называемые компонентами), которые точно описывают характер временного ряда и в виде совокупности которых временной ряд можно представить:

1. тренд (долгосрочное увеличение или уменьшение значений ряда);
2. сезонность, сезонная вариация (краткосрочное регулярно повторяющееся колебание значений временного ряда вокруг тренда);
3. цикл, циклические колебания (характерные изменения ряда, связанные с повторяющимися глобальными причинами — цикл деловой активности или экономический цикл, состоящий из экономического подъема, спада, депрессии и оживления);
4. остаточная вариация, которая может быть двух видов:
   - аномальная вариация — неестественно большое отклонение временного ряда, которое оказывает воздействие на единичное наблюдение;
   - случайная вариация — малое отклонение, которое невозможно предвидеть.

Вследствие развития нейросетей и искусственного интеллекта, появилась возможность решать задачи, связанные с временными рядами, решение которых могло бы быть полезно для той или иной области жизнедеятельности человека. Например, в статье ”Human Activity Recognition Based on Time Series Analysis Using U-Net” показано, как архитектура U-Net может эффективно использоваться для распознавания активности человека на основе временных рядов. Применение такой модели позволяет точно идентифицировать последовательности движений, что особенно полезно в областях здравоохранения, мониторинга физической активности и систем безопасности. [1]

Особенно полезно это может быть в тех областях, где обычные статистические модели (обычная или интегрированная модель авторегрессии - скользящего среднего, векторная авторегрессия и т.д.) неприменимы или неэффективны.

Временные ряды могут содержать аномалии. **Аномалией** называется отклонение в стандартном поведении какого-то процесса, который описывается этим временным рядом. По сути её можно определить через аномальную вариацию, являющейся возможной компонентой этого временного ряда. В зависимости от предметной области описываемого процесса в выборке (датасете) могут быть аномалии разного вида. Можно выделить одни из самых распространенных видов аномалий:

1. точечные аномалии (наблюдается отклонение в поведении в отдельных точках);
2. групповые аномалии (группа точек, которая ведет себя аномально, но каждая точка которой отдельно аномальной не является);
3. контекстные аномалии, суть которых в связи аномалии с внешними данными, которые не присущи значениям ряда (например, отрицательная температура на улице летом).

Мультивариативный временной ряд представляет собой временной ряд, в котором одновременно записывается несколько переменных, например:

1. Измерения метеостанций: температура, влажность, атмосферное давление;
2. Параметры работы промышленного оборудования: ток, напряжение, вибрация;
3. Данные спутников: значения электрического или магнитного поля на разных частотах.

Обнаружение аномалий в таких рядах особенно важно в ряде практических задач: предсказание отказов оборудования на основании датчиков в сфере промышленности, обнаружение патологий в данных ЭКГ или ЭЭГ в сфере медицины, выявление подозрительных транзакций или внезапных рыночных колебаний в сфере анализа финансов, выявление редких или необычных событий наподобие всплесков радиации в сфере астрономии. [2], [3]

В данной работе исследуется возможность использования классических алгоритмов машинного обучения (случайный лес, метод k ближайших соседей, CatBoost) и нейросетевых моделей компьютерного зрения (ResNet34, Xception,

ViT) для классификации мультивариативных временных рядов при решении задачи обнаружения аномалий. В качестве набора данных используются данные, полученные со спутника GGS WIND. Это 256 временных рядов, которые представляют собой значения нормализованного напряжения электрического поля, каждый из которых соответствует определенной частоте.

В качестве предобработки данных были использованы два подхода:

1. Метод скользящего окна — для получения временных блоков фиксированной длительности;
2. Метод преобразования Фурье для построения спектрограмм.

Классические алгоритмы машинного обучения в качестве выборки использовали временные блоки, полученные первым методом, а модели компьютерного зрения - спектрограммы, полученные вторым методом. Сравнение результатов моделей проводилось по метрикам точности классификации.

1 Теоретическая часть

1.1 Временные ряды в задачах машинного обучения и их обработка

1.1.1 Временные ряды в задачах машинного обучения

Временной ряд — это упорядоченная последовательность данных, измеряемых через равные промежутки времени. Основной характеристикой временного ряда является наличие зависимости между наблюдениями, сделанными в различные моменты времени. Например, измерения температуры в течение суток, записи электрокардиограммы (ЭКГ) или значения продаж товаров по дням.

Мультивариативные временные ряды представляют собой более сложную форму временных рядов, в которых одновременно записывается несколько переменных. Эти ряды позволяют изучать взаимосвязи между различными параметрами системы, такие как изменения температуры, влажности и давления в метеорологии или параметры работы двигателя (температура, вибрация, ток).

Временные ряды являются важным источником информации в различных областях. [4] Их анализ помогает решать широкий круг задач:

1. Прогнозирование — предсказание будущих значений временного ряда, например, прогнозирование спроса на продукцию, движения фондового рынка или погоды;
2. Обнаружение аномалий — выявление отклонений от нормального поведения, таких как сбои оборудования, резкие изменения в сердечной активности или финансовые мошенничества;
3. Классификация — определение категории или состояния на основе временного ряда, например, диагностика заболеваний по данным ЭКГ;
4. Кластеризация — группировка временных рядов с похожими характеристиками, например, выявление схожих паттернов в потреблении электроэнергии.

Работа с временными рядами накладывает ряд специфических требований:

1. Наблюдения в ряде зависят от предшествующих значений; Например, в данных о погоде температура в текущий момент связана с температурой в предыдущие часы;
2. Временные ряды часто имеют большую длину, а мультивариативные ряды содержат множество переменных, что требует мощных вычислительных

ресурсов;

3. Данные могут быть зашумлены, содержать пропуски или выбросы, что делает их предварительную обработку важным этапом;
4. Временные ряды могут содержать как линейные, так и нелинейные зависимости, а также периодические и непериодические компоненты.

1.1.2 Обработка временных рядов

Обработка временных рядов представляет собой ключевой этап подготовки данных для последующего анализа и построения моделей машинного обучения. Временные ряды обладают специфическими особенностями, такими как наличие временной зависимости, высокая размерность, наличие шума и выбросов, которые требуют применения специализированных методов. Эффективная обработка временных рядов позволяет выделить скрытые закономерности, устранить помехи и подготовить данные для анализа.

Для успешного применения методов машинного обучения временные ряды часто преобразуются из исходного формата в более удобное для анализа представление. Среди наиболее распространенных методов преобразования временных рядов выделяются:

1. Быстрое преобразование Фурье (FFT).

   Этот метод используется для перехода от временной области к частотной. На его основе строятся спектры амплитуд и мощностей, которые дают представление о частотных компонентах сигнала. Например, частотный анализ позволяет выделить доминирующие частоты, связанные с определенными процессами, что полезно для диагностики технических систем или биологических сигналов.

   Быстрое преобразование Фурье (Fast Fourier Transform, FFT) — это алгоритм вычисления дискретного преобразования Фурье (DFT) и его обратного преобразования (IDFT). Оно позволяет анализировать временной сигнал, переводя его из временной области в частотную, чтобы определить амплитуды и фазы составляющих сигнал частот. [5]

   Для сигнала $x[n]$, содержащего N отсчетов, дискретное преобразование Фурье определяется следующим образом:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{kn}{N}},\ k = 0, 1, \ldots, N-1, \tag{1}$$

где $X[k]$ — N комплексных амплитуд синусоидальных сигналов, $x[n]$ — значение сигнала во временной области для n-го дискретного момента времени (индекса отсчета), N — количество значений сигнала, $e^{-j2\pi\frac{kn}{N}}$ — комплексный экспоненциальный множитель (база Фурье), k — индекс частоты.

Прямое вычисление DFT имеет сложность $O(N^2)$, так как для каждого k необходимо выполнить N умножений и сложений. Это делает применение DFT к длинным временным рядам вычислительно затратным. FFT — это оптимизация алгоритма DFT, которая уменьшает его сложность до $O(NlogN)$. Алгоритм FFT основан на принципе "разделяй и властвуй", разбивая исходную задачу на меньшие подзадачи.

2. Вейвлет-преобразование.

   Вейвлеты позволяют анализировать временной ряд на разных временных масштабах, что особенно важно для данных, содержащих локальные особенности или резкие изменения. Этот метод часто используется для выделения кратковременных событий, таких как скачки напряжения в энергетических системах или аномальные всплески в радиосигналах. [6]

3. Создание лаговых признаков.

   Для сохранения временной зависимости между наблюдениями в ряде, создаются дополнительные признаки, представляющие собой значения ряда на предыдущих шагах. Например, для предсказания температуры в момент времени t, могут быть использованы значения температуры в $t-1$, $t-2$ и так далее. Такой подход сохраняет информацию о динамике ряда, что важно для задач прогнозирования.

4. Шейплет-преобразование.

   Шейплет-преобразования (shapelet transform) — это метод анализа временных рядов, основанный на выявлении характерных подструктур (шейплетов), которые служат отличительными признаками для классификации или кластеризации данных. Шейплеты представляют собой короткие, локальные временные сегменты, которые обладают высоким различающим потенциалом и позволяют эффективно описывать временные ряды. Для шейплет-преобразования исходные временные ряды преобразуются в матрицу признаков, где каждая строка соответствует временному ряду, а каждый столбец — шейплету. Элементы этой матрицы отражают степень

схожести между каждым временным рядом и конкретным шейплетом. Таким образом, шейплет-преобразование позволяет перейти от анализа временных рядов к анализу традиционной матрицы признаков, что упрощает использование классических алгоритмов машинного обучения. [7]

Алгоритм получения шейплета для временных рядов начинается с выбора кандидатов. Рассматриваются все возможные подотрезки S длины m из временных рядов T в обучающем наборе, где $S \subseteq T$ и $|S| = m$. Для каждого такого кандидата вычисляется расстояние до всех подотрезков временного ряда T с использованием метрик, таких как евклидово расстояние. Минимальное значение расстояния по всем возможным подотрезкам временного ряда определяется как:

$$D(S, T) = \min_i \sqrt{\sum_{j=1}^{m} (s_j - t_{i+j-1})^2}. \quad (2)$$

Далее, каждому временному ряду T сопоставляется значение $D(S, T)$, которое представляет собой меру схожести ряда с данным кандидатом. Эти значения формируют карту признаков, используемую для последующей классификации. На следующем этапе для каждого кандидата S оценивается его значимость как признака. Это можно сделать, например, с использованием критерия информационной выгоды:

$$IG(S) = H(C) - \sum_k P_k H(C|D_k), \quad (3)$$

где $H(C)$ — энтропия классов, D_k — дискретизация расстояний $D(S, T)$, а P_k — вероятность попадания расстояний в интервал k.

После этого выбираются те шейплеты, которые демонстрируют максимальную информационную выгоду $IG(S)$ и лучше всего разделяют классы временных рядов. Чтобы учесть разнообразие паттернов в данных, данный процесс повторяется для разных длин шейплетов m, позволяя выявить локальные закономерности в временных рядах.

5. Сглаживание, нормализация и устранение выбросов.

   Сглаживание, нормализация и устранение выбросов — это этапы предобработки временных рядов, направленные на улучшение качества данных и повышение эффективности работы моделей машинного обучения. [8]

Сглаживание используется для уменьшения уровня шума в данных и выделения общих тенденций временного ряда. Этот процесс позволяет нивелировать влияние кратковременных флуктуаций, которые не несут существенной информации. Например, для сглаживания часто применяют скользящее среднее, где каждое значение заменяется средним арифметическим соседних точек на фиксированном интервале. Пусть временной ряд задан как $X = [x_1, x_2, \ldots, x_n]$. Тогда сглаженное значение для точки x_i определяется как:

$$\tilde{x}_i = \frac{1}{w} \sum_{j=i-\lfloor w/2 \rfloor}^{i+\lfloor w/2 \rfloor} x_j, \tag{4}$$

где w — длина окна сглаживания, т.е. это фиксированный интервал данных, используемый для вычисления среднего значения. Для каждой точки временного ряда x_i берутся w соседних значений (включая само x_i), и вычисляется их среднее арифметическое. Значение w определяет количество точек, влияющих на сглаженное значение $\tilde{x}_i$. Если w больше, сглаживание будет сильнее, поскольку будут усредняться данные из более широкого интервала. Это уменьшает шум, но может скрывать мелкие детали в данных. Если w меньше, сглаживание будет слабее, сохраняя более мелкие колебания.

Альтернативным методом является применение экспоненциального сглаживания, которое добавляет больший вес более недавним значениям и менее значительный вклад предыдущих.

Нормализация временных рядов заключается в приведении данных к единому масштабу, что особенно важно при наличии значений, различающихся на несколько порядков. Это помогает моделям эффективнее находить зависимости и предотвращает доминирование переменных с большими числовыми значениями. Одним из самых распространённых методов нормализации является линейное масштабирование на интервал $[0, 1]$, где каждое значение вычисляется как:

$$x_i^{\text{norm}} = \frac{x_i - \min(X)}{\max(X) - \min(X)}. \tag{5}$$

Если данные имеют распределение, близкое к нормальному, то часто ис-

пользуют стандартизацию, которая преобразует значения так, чтобы они имели среднее равное 0 и стандартное отклонение 1:

$$x_i^{\text{stand}} = \frac{x_i - \mu}{\sigma}, \tag{6}$$

где μ — среднее значение временного ряда, а σ — стандартное отклонение.

Устранение выбросов направлено на исключение экстремальных значений, которые значительно отклоняются от основной массы данных и могут негативно повлиять на результаты анализа или обучение модели. Выбросы определяются с использованием статистических методов, таких как межквартильный размах (IQR), при котором выбросами считаются значения за пределами интервала:

$$[\text{Q1} - 1.5 \times \text{IQR}, \text{Q3} + 1.5 \times \text{IQR}],$$

где Q1 и Q3 — первый и третий квартили соответственно, а $\text{IQR} = \text{Q3} - \text{Q1}$. Для временных рядов с известной физической интерпретацией можно использовать доменные знания для выявления выбросов, например, исключать значения, выходящие за пределы определённых диапазонов. После идентификации выбросов они могут быть заменены, например, медианным значением соседних точек, либо полностью удалены в зависимости от целей анализа.

Эти три этапа, применяемые в совокупности, обеспечивают очистку и унификацию данных, что повышает стабильность и точность методов анализа временных рядов.

1.1.3 Создание спектрограмм с помощью преобразования Фурье

Создание спектрограмм с помощью преобразования Фурье является важным этапом анализа временных рядов, особенно если сигнал содержит значимую информацию в частотной области. Преобразование Фурье позволяет представить сигнал, зависящий от времени, в виде набора частотных составляющих, выявляя амплитуды и интенсивности колебаний на разных частотах. Спектрограммы, построенные с использованием оконного преобразования Фурье, дают возможность визуализировать, как частотное содержание сигнала изменяется во времени. Это особенно полезно в задачах классификации временных ря-

дов, таких как обнаружение аномалий или выявление характерных паттернов. Использование спектрограмм помогает моделям машинного обучения и компьютерного зрения эффективно анализировать скрытые закономерности, недоступные при работе с сигналом в его исходной форме. [9], [10]

Основная идея алгоритма быстрого преобразования Фурье состоит из трёх этапов:

1. Сигнал $x[n]$ разбивается на две части — четные и нечетные индексы:

$$x_{\text{чет}}[n] = x[2n],\ x_{\text{нечет}}[n] = x[2n+1]. \tag{7}$$

2. Преобразования для четной и нечетной частей вычисляются отдельно:

$$X[k] = X_{\text{чет}}[k] + e^{-j2\pi\frac{k}{N}} X_{\text{нечет}}[k]. \tag{8}$$

3. Таким образом, вычисление преобразования сводится к рекурсивным вызовам для меньших подмассивов длиной $\frac{N}{2}$.

Спектрограмма отображает сигнал как функцию времени и частоты, показывая интенсивность частотных компонентов в разных временных интервалах. Это достигается применением дискретного преобразования Фурье (DFT) к последовательным частям сигнала (окнам) с определенным перекрытием (overlap).

Оконное преобразование Фурье (Short-Time Fourier Transform, STFT) используется для анализа нестационарных сигналов, чьи частотные характеристики изменяются со временем. В отличие от классического преобразования Фурье, STFT позволяет локализовать частоты в определенные моменты времени за счет применения оконной функции. [11], [12]

Пусть задан сигнал $x(t)$, который необходимо анализировать во временной и частотной областях. Для этого сигнал разбивается на короткие интервалы (окна) $w(t)$ фиксированной длины, где сигнал считается стационарным. Затем для каждого окна применяется дискретное преобразование Фурье (DFT), чтобы вычислить спектр частот. STFT определяется следующим образом:

$$X(t, f) = \int_{-\infty}^{\infty} x(\tau) w(\tau - t) e^{-j2\pi f \tau}\, d\tau, \tag{9}$$

где $X(t, f)$ — комплексное значение спектра на момент времени t и частоте f, $w(t)$ — оконная функция, которая выделяет часть сигнала вокруг момента

t,$e^{-j2\pi f\tau}$ — гармоническая функция для преобразования Фурье.

На практике сигналы обычно дискретны, поэтому применяется дискретное оконное преобразование Фурье:

$$X_m[k] = \sum_{n=0}^{L-1} x[n + mR]w[n]e^{-j2\pi\frac{kn}{L}}, \tag{10}$$

где: $x[n]$ — дискретный сигнал, $w[n]$ — оконная функция длиной L, R — шаг окна (разница между началом текущего и следующего окна), m — индекс окна, k — индекс частоты.

Оконная функция $w(t)$ ограничивает анализируемый участок сигнала, подавляя значения вне текущего окна. Выбор оконной функции влияет на точность временного и частотного разрешения. В рамках данного исследования была использована косинус-оконная функция, которая также называется оконной функцией Тьюки.

Оконная функция Тьюки представляет собой гибрид прямоугольного и косинусного окна, что позволяет одновременно уменьшить утечку спектра и минимизировать амплитудные искажения на краях окна. [13] Функция Тьюки определяется формулой:

$$w(t) = \begin{cases} \frac{1}{2}\left(1 + \cos\left(\pi\left(\frac{2t}{\alpha T} - 1\right)\right)\right), & 0 \le t < \frac{\alpha T}{2}, \\ 1, & \frac{\alpha T}{2} \le t \le T - \frac{\alpha T}{2}, \\ \frac{1}{2}\left(1 + \cos\left(\pi\left(\frac{2t}{\alpha T} - 2/\alpha + 1\right)\right)\right), & T - \frac{\alpha T}{2} < t \le T, \end{cases} \tag{11}$$

где: α — параметр, определяющий ширину косинусного участка окна (от 0 до 1), T — длина окна.

Оконная функция Тьюки обладает следующими свойствами:

1. При $\alpha = 0$ окно становится прямоугольным;
2. При $\alpha = 1$ окно становится косинусным (окно Ханна);
3. Значение параметра α позволяет балансировать между временным и частотным разрешением.

Резюмируя, главная цель данной оконной функции — минимизация эффекта утечки спектра, возникающего из-за дискретизации сигнала. Она представляет собой гибрид прямоугольного и косинусного окна, что делает её уни-

версальным выбором для анализа сигналов.

Использование спектрограмм, основанных на преобразовании Фурье, является одним из ключевых инструментов для анализа временных рядов в частотно-временной области. Этот метод позволяет не только выявлять основные частотные составляющие сигнала, но и анализировать их изменения во времени, что критически важно в задачах классификации и детекции аномалий. Спектрограммы представляют собой удобный способ преобразования сложных временных данных в визуально и численно интерпретируемую форму, что открывает новые возможности для применения методов машинного обучения и глубокого обучения. Таким образом, спектрограммы служат мостом между сигналом и моделями анализа, существенно повышая эффективность обработки данных.

1.2 Обнаружение аномалий с помощью ИИ в мультивариативных временных рядах в сфере компьютерной безопасности

В современном мире информационных технологий обеспечение безопасности компьютерных систем становится все более сложной задачей. Сетевой трафик, являющийся по определению мультивариативным временным рядом, предоставляет богатый материал для анализа и обнаружения различных аномальных активностей, которые могут быть признаками нарушения безопасности. Обнаружение аномалий в сетевом трафике является одной из важнейших задач в области защиты информации. Аномалии могут указывать на различные угрозы, включая DDoS-атаки, проникновения и другие виды кибератак. Традиционные методы обнаружения, основанные на статических правилах и сигнатурах, часто оказываются недостаточно эффективными в условиях постоянно меняющихся угроз. В связи с этим методы ИИ, способные обучаться на основе исторических данных и адаптироваться к новым видам атак, становятся незаменимыми инструментами в обеспечении кибербезопасности.

В статье ”Multifaceted DDoS Attack Prediction by Multivariate Time Series and Ordinal Patterns” представлена архитектура для предсказания DDoS-атак, основанная на анализе мультивариативных временных рядов и порядковых шаблонов с использованием метода опорных векторов. [14] Модель преобразует данные временных рядов в последовательности порядковых символов, что обеспечивает устойчивость к шуму и сохранение временной структуры. Анализ корреляций между рядами и использование порядковых шаблонов для предска-

зания позволяет выявлять ранние признаки атак. Архитектура способна предсказывать атаки за 35 минут до их начала, эффективно комбинируя обработку временных и структурных характеристик данных. Такая интеграция обеспечивает высокую точность и адаптивность модели для защиты сетевых систем.

В статье "STFT-TCAN: A TCN-attention based multivariate time series anomaly detection architecture with time-frequency analysis for cyber-industrial systems" исследуется подход к обнаружению аномалий в киберсистемах с использованием гибридной архитектуры глубокого обучения. [15] Существующие алгоритмы для обнаружения аномалий преимущественно сосредоточены на моделировании временной области, что ограничивает их возможности по извлечению значимых признаков из частотной области. Это негативно сказывается на точности обнаружения. Чтобы решить эту проблему, авторы статьи предлагают новую модель — STFT-TCAN — для анализа временных рядов, которая интегрирует информацию как из временной, так и из частотной областей. Для извлечения частотных характеристик используется скользящее окно и коротковременное преобразование Фурье (STFT), что позволяет формировать матрицу частотных данных, объединяющую свойства обеих областей.

Таким образом, результаты исследования данной дипломной работы, которая посвящена обнаружению аномалий в мультивариативных временных рядах методами искусственного интеллекта, могут иметь приложения в области информационной безопасности, поскольку методы машинного и глубокого обучения находят широкое применение в задачах мониторинга и защиты сетевых систем. Аномалии в сетевом трафике, как указано в рассмотренных исследованиях, часто являются индикаторами кибератак, включая DDoS-атаки и нарушения работы оборудования. То есть технологии искусственного интеллекта играют ключевую роль в предотвращении атак, минимизации ущерба и обеспечении надежности критически важных систем.

1.3 Классические алгоритмы машинного обучения

1.3.1 Случайный лес

Случайный лес (Random Forest) — это ансамблевый метод машинного обучения, основанный на объединении множества деревьев решений (см. Рисунок 1). Его основная идея заключается в комбинировании предсказаний нескольких моделей для повышения точности и устойчивости результатов.

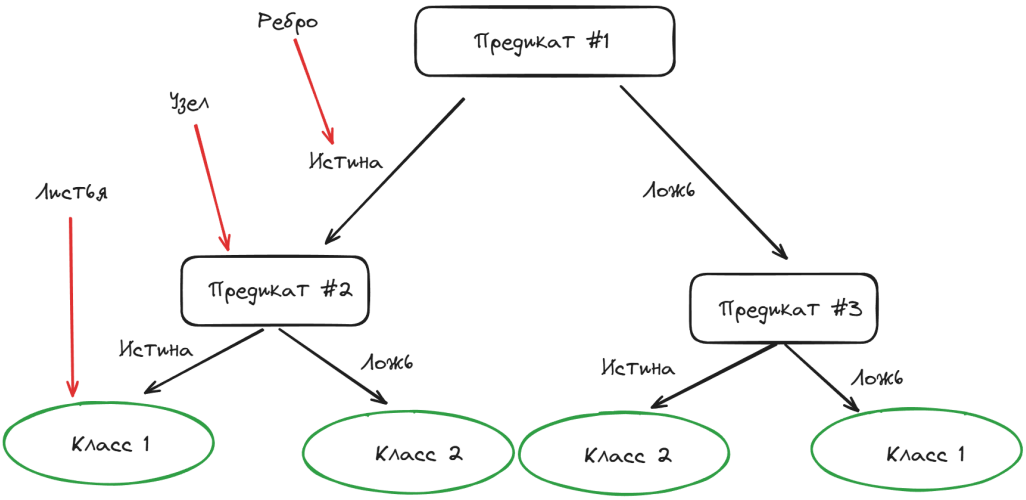

Рисунок 1 – Схема алгоритма дерева решений

Каждое дерево в случайном лесу представляет собой независимую модель, построенную на случайной подвыборке данных и случайном подмножестве признаков. Итоговое решение для классификации получается голосованием, а для регрессии — усреднением предсказаний деревьев. [16]

Алгоритм работает на основе ансамблевого подхода Bagging (или же Bootstrap Aggregating), где каждое дерево строится на уникальной подвыборке данных. Это помогает снизить переобучение, так как деревья становятся менее зависимыми друг от друга. Для классификации итоговое предсказание вычисляется как:

$$\hat{y} = \text{mode}(T_1(x), T_2(x), \ldots, T_N(x)), \tag{12}$$

где: $T_i(x)$ — предсказание i-го дерева для объекта x, mode — функция, возвращающая наиболее частое значение.

Для регрессии используется усреднение предсказаний всех деревьев:

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} T_i(x). \tag{13}$$

Случайный лес строится по следующему алгоритму:

1. Создание подвыборок данных: Каждое дерево строится на основе случайной подвыборки данных S_i, полученной методом бутстрепа из исходного набора S. Размер каждой подвыборки равен $|S|$, но элементы могут повторяться.

$$S_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}, \quad x_{ij} \in S.$$

2. Формирование случайного подмножества признаков: на каждом узле дерева выбирается случайное подмножество признаков размером k, где $k < d$ (d — общее число признаков). Лучший признак для разбиения узла выбирается только из этого подмножества.
3. Построение деревьев решений: каждое дерево строится до полной глубины (без обрезки), чтобы минимизировать ошибку на подвыборке.
4. Комбинирование деревьев: для классификации применяется голосование, для регрессии — усреднение.

Случайный лес обладает рядом преимуществ, которые делают его популярным инструментом в задачах машинного обучения. Одним из ключевых достоинств является устойчивость к переобучению, достигаемая благодаря использованию случайных подвыборок данных и признаков при построении деревьев. Этот подход позволяет алгоритму сохранять высокую точность предсказаний даже на сложных наборах данных. Кроме того, случайный лес предоставляет возможность оценки важности признаков. Для этого используется метрика уменьшения ошибки, определяемая как

$$\text{Importance}(j) = \frac{1}{N} \sum_{i=1}^{N} \left(\text{Err}_{OOB} - \text{Err}_{OOB}^{(j)} \right), \tag{14}$$

где Err_{OOB} обозначает ошибку на объектах вне подвыборки, а $\text{Err}_{OOB}^{(j)}$ — ошибку после случайной перестановки признака j. Это делает случайный лес не только эффективным инструментом для предсказания, но и мощным средством для анализа данных.

Однако алгоритм имеет и свои ограничения. Одной из проблем является сложность интерпретации результатов, особенно в случае работы с большим количеством деревьев. Кроме того, случайный лес может быть вычислительно затратным, особенно на больших наборах данных, что может ограничивать его применение в условиях ограниченных вычислительных ресурсов.

1.3.2 Boosting-алгоритм CatBoost

CatBoost (Categorical Boosting) — это инструмент для решения задач классификации и регрессии, основанный на градиентном бустинге над деревьями решений. Этот алгоритм был разработан компанией Яндекс с учётом оптимизации работы с категориальными признаками, которые часто встречаются в реальных данных.

Градиентный бустинг в основе CatBoost заключается в построении ансамбля деревьев решений, где каждое последующее дерево создаётся для исправления ошибок предыдущих моделей (см. Рисунок 2). Алгоритм минимизирует функцию потерь, последовательно добавляя деревья, которые корректируют остатки (разницу между фактическими значениями и предсказаниями текущей модели). [17]

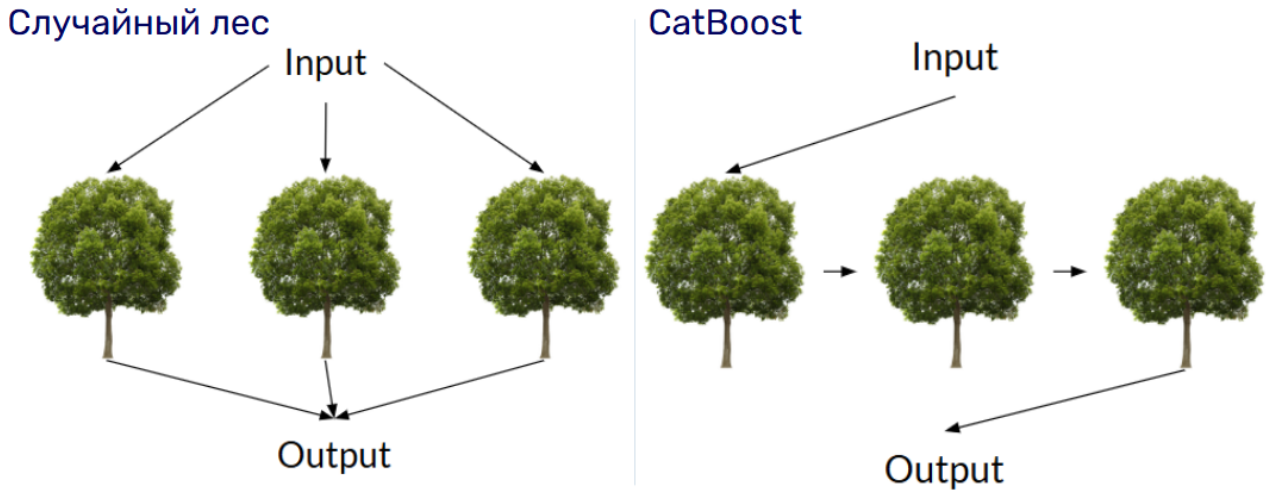

Рисунок 2 – Разница между случайным лесом и бустингом над деревьями решений: в первом случае деревья вычисляют значение параллельно, во втором - последовательно

Общая модель записывается как сумма отдельных деревьев:

$$F(x) = \sum_{m=1}^{M} \eta \cdot h_m(x), \tag{15}$$

где $h_m(x)$ — предсказание m-го дерева, η — коэффициент скорости обучения, а M — общее число деревьев.

Характерной особенностью CatBoost является эффективная работа с категориальными признаками. Вместо простой обработки категорий с помощью one-hot encoding или label encoding, CatBoost использует метод преобразования

на основе счётчиков (target encoding). Это позволяет учитывать распределение значений целевой переменной внутри категорий. Для предотвращения переобучения используется техника блочного разделения данных, что исключает утечку информации между обучением и тестированием.

CatBoost устраняет один из ключевых недостатков градиентного бустинга — смещение в предсказаниях, которое возникает из-за некорректной инициализации или порядка построения деревьев. Это достигается с помощью применения "упорядоченного бустинга"(ordered boosting), где каждый шаг обучения модели строится на подвыборке данных, недоступных для текущего шага.

Алгоритм также активно использует сжатие признаков и методы регуляризации, включая L_2-регуляризацию, что делает его менее склонным к переобучению по сравнению с другими библиотеками, такими как XGBoost или LightGBM.

CatBoost демонстрирует высокую точность предсказаний даже при работе с данными, содержащими большое количество категориальных признаков и пропущенных значений. Алгоритм требует минимальной предобработки данных, так как встроенные механизмы работы с категориями, пропусками и устойчивостью к шуму снижают необходимость в инженерии признаков. Более того, CatBoost обеспечивает эффективность вычислений благодаря использованию GPU и оптимизации производительности.

Процесс минимизации ошибки модели можно выразить через задачу оптимизации:

$$\min_F \sum_{i=1}^{n} L(y_i, F(x_i)), \tag{16}$$

где $L(y, F(x))$ — функция потерь, зависящая от фактического значения y и предсказания $F(x)$, а n — количество объектов. Градиентный бустинг аппроксимирует эту функцию, добавляя новые деревья $h_m(x)$, которые приближают антиградиент функции потерь:

$$g_i^{(m)} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}. \tag{17}$$

1.3.3 Метод k-ближайших соседей

Метод k-ближайших соседей (k-Nearest Neighbors, k-NN) — это один из простейших алгоритмов машинного обучения, который используется для ре-

шения задач классификации и регрессии. Его суть заключается в том, что для нового объекта x определяется k ближайших объектов из обучающей выборки, и решение принимается на основе их значений (см. Рисунок 3). [18]

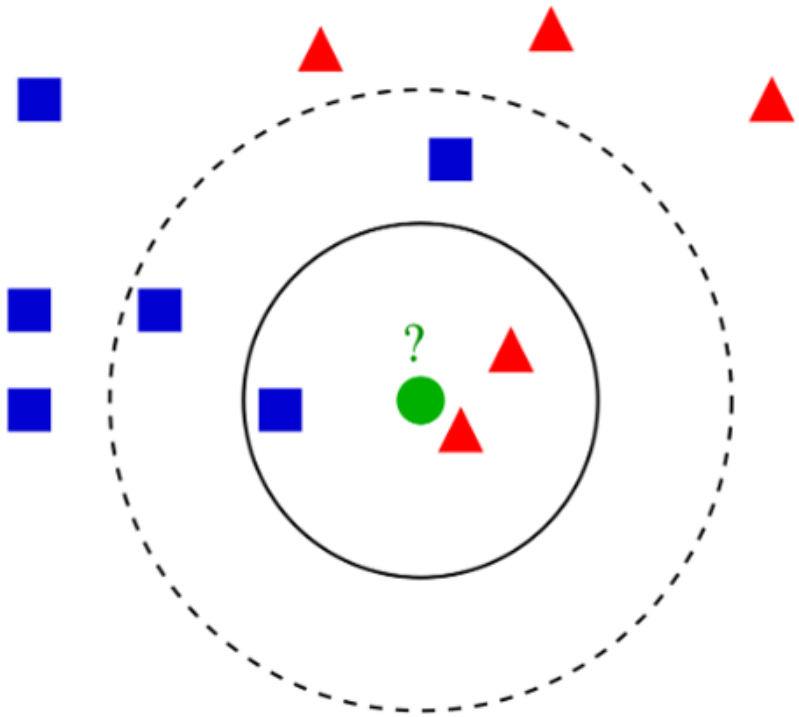

Рисунок 3 – Схема метода k-ближайших соседей

Для классификации метка класса нового объекта определяется по классу, наиболее часто встречающемуся среди k-ближайших соседей. Формально, метка $\hat{y}$ для объекта x задаётся следующим образом:

$$\hat{y} = \text{argmax}_c \sum_{i \in N_k(x)} \delta(y_i, c), \tag{18}$$

где $N_k(x)$ — множество k-ближайших объектов к x, $\delta(y_i, c)$ — индикаторная функция, равная 1, если $y_i = c$, и 0 в противном случае.

Для регрессии используется усреднение значений целевой переменной y соседей:

$$\hat{y} = \frac{1}{k} \sum_{i \in N_k(x)} y_i. \tag{19}$$

Определение расстояния между объектами осуществляется с помощью метрики рассчета расстояния. Выбор метрики расстояния между объектами

играет ключевую роль в работе метода k-ближайших соседей. Наиболее популярными являются:

1. Евклидово расстояние:

$$d(x, x') = \sqrt{\sum_{j=1}^{d} (x_j - x'_j)^2}, \tag{20}$$

где d — размерность пространства признаков.

2. Манхэттенское расстояние:

$$d(x, x') = \sum_{j=1}^{d} |x_j - x'_j|. \tag{21}$$

3. Косинусное расстояние:

$$d(x, x') = 1 - \frac{\sum_{j=1}^{d} x_j x'_j}{\sqrt{\sum_{j=1}^{d} x_j^2} \cdot \sqrt{\sum_{j=1}^{d} {x'_j}^2}}. \tag{22}$$

Евклидово расстояние является наиболее интуитивно понятной метрикой и работает хорошо, когда данные имеют линейную структуру и признаки измеряются в одинаковых единицах. Оно эффективно в задачах, где важно учитывать абсолютные различия между значениями признаков, например, в случаях с непрерывными числовыми признаками, которые могут быть близкими по масштабу. Однако Евклидово расстояние может плохо работать с разреженными данными или данными, где признаки имеют разную шкалу, так как большие различия в масштабе признаков могут сильно влиять на итоговое расстояние.

Манхэттэнское расстояние, в отличие от Евклидова, хорошо работает в ситуациях, когда данные состоят из признаков, которые могут быть независимыми и числовыми, но не обязательно измеряются в одинаковых единицах. Оно полезно, например, для данных, где важно учитывать только отличия в отдельных признаках, а не в их комбинациях. Манхэттэнское расстояние подходит для задач, в которых ключевыми являются линейные различия, но без учета возможных ”диагональных” расстояний между точками. Эта метрика часто используется в задачах с разреженными или категориальными данными.

Косинусное расстояние наиболее эффективно в контексте данных, где

важно не абсолютное расстояние между объектами, а их взаимная направленность или угол между векторами признаков. Это делает косинусное расстояние идеальным для работы с текстовыми данными (например, при анализе документов или векторизации текстов), где два документа могут быть схожи по смыслу, но отличаться по длине. Эта метрика предпочтительна для текстовых классификаций или рекомендаций, где важна степень схожести в контексте общей структуры данных, а не в абсолютных значениях признаков.

Для метода k ближайших соседей выбор метрики расстояния зависит от природы данных. Если данные являются числовыми и имеют нормализованные или схожие шкалы, то Евклидово расстояние обычно работает лучше, так как оно эффективно измеряет абсолютные различия. Для разреженных или категориальных данных, где важны отдельные различия в признаках, более подходящей может быть Манхэттэнская метрика. Косинусное расстояние предпочтительно, если данные представляют собой текстовые векторизованные представления, где важен не общий масштаб, а взаимная ориентация вектора признаков. Выбор метрики должен основываться на специфике задачи и типе данных, и, соответственно, каждая из метрик имеет свои преимущества в различных контекстах, влияя на точность и интерпретируемость результатов метода kNN.

1.4 Алгоритмы компьютерного зрения

1.4.1 ResNet34

ResNet34 (Residual Network с 34 слоями) — это одна из архитектур глубоких сверточных нейронных сетей, разработанная для эффективного обучения глубоких моделей. Основной особенностью ResNet является использование так называемых остаточных блоков (residual blocks), которые решают проблему деградации качества обучения с увеличением глубины сети. Этот подход делает ResNet34 особенно эффективным для задач классификации, в том числе для анализа временных рядов, преобразованных в изображения (например, спектрограммы). [19]

Главной инновацией ResNet34 является введение остаточных соединений, которые позволяют передавать информацию напрямую через слои, минуя основной поток вычислений. Это достигается за счёт добавления входного сигнала x к выходу некоторого нелинейного преобразования $F(x)$ (см. Рисунок 4).

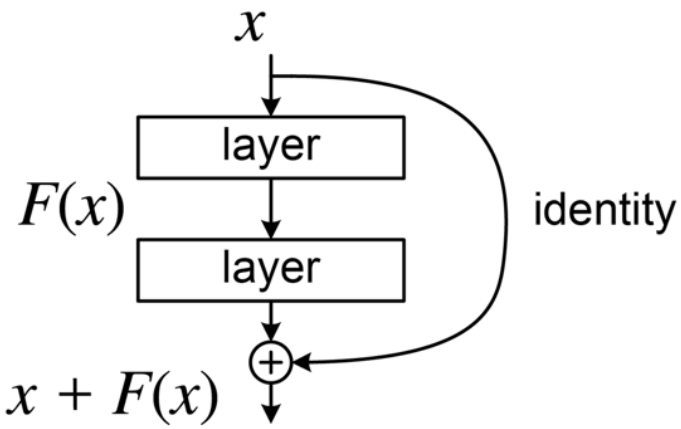

Рисунок 4 – Особенность ResNet сети — Residual block

Математически это можно представить как:

$$y = F(x) + x, \tag{23}$$

где $F(x)$ — это результат применения свёрток, нелинейных функций активации и нормализации. Остаточные соединения упрощают обучение, так как градиенты эффективно проходят через сеть даже при её значительной глубине, уменьшая риск исчезающих градиентов.

ResNet34 состоит из 34 слоев, включая свёрточные, нормализационные и полносвязные слои. В архитектуре предусмотрено использование остаточных блоков на разных уровнях, что позволяет модели обучать как низкоуровневые, так и высокоуровневые признаки.

В задачах классификации временных рядов ResNet34 применяется не напрямую к исходным данным, а к их визуальным представлениям, таким как спектрограммы. Это связано с тем, что спектрограммы содержат временную и частотную информацию, что делает их подходящим входом для сверточных нейронных сетей. Использование ResNet34 в таких задачах позволяет:

1. Извлекать пространственные и частотные паттерны, которые важны для классификации временных рядов.
2. Эффективно использовать остаточные соединения для обучения глубоких признаков.
3. Уменьшить риск переобучения даже при небольшом количестве данных благодаря мощной регуляризации и архитектурным особенностям.

Одним из ключевых преимуществ ResNet34 является её способность обу-

чаться на больших глубинах без ухудшения производительности. Это делает её подходящей для сложных задач с большим количеством данных. Кроме того, ResNet34 демонстрирует высокую обобщающую способность, что особенно важно для задач классификации временных рядов, где данные могут быть зашумлёнными или иметь сложные зависимости.

Однако использование ResNet34 требует значительных вычислительных ресурсов, что может быть ограничением для применения на устройствах с низкой вычислительной мощностью. Кроме того, для достижения оптимальной производительности требуется тщательная настройка гиперпараметров, включая выбор архитектурных модификаций и методов регуляризации.

Благодаря своей архитектуре с остаточными соединениями и способности эффективно обучаться на глубоких сетях, ResNet34 становится одним из наиболее популярных выборов для задач классификации, где требуется извлечение сложных пространственных и временных паттернов. В контексте дипломной работы ResNet34 был использован для классификации спектрограмм, что позволило извлечь ключевые признаки, связанные с присутствием или отсутствием аврорального километрового радиоизлучения.

1.4.2 Xception

Xception (Extreme Inception) — это архитектура глубоких сверточных нейронных сетей, предложенная Франсуа Шолле, которая является развитием идей, заложенных в архитектуре Inception. Основной особенностью Xception является использование глубинных (separable) сверточных операций вместо стандартных сверточных слоев. Это делает модель более эффективной и вычислительно оптимизированной, сохраняя высокую способность к извлечению признаков. [20]

Xception основывается на предположении, что пространственные и канальные зависимости в данных можно разъединить и обрабатывать независимо. В классических сверточных слоях фильтры одновременно анализируют пространственные и канальные связи. В Xception эти два процесса разделены (см. Рисунок 5):

1. Глубинная свёртка (depthwise convolution): каждая свёртка применяется к отдельному каналу данных, что позволяет обрабатывать пространственную информацию независимо.
2. Точечная свёртка (pointwise convolution): применяется 1×1-свёртка, которая агрегирует информацию из всех каналов.

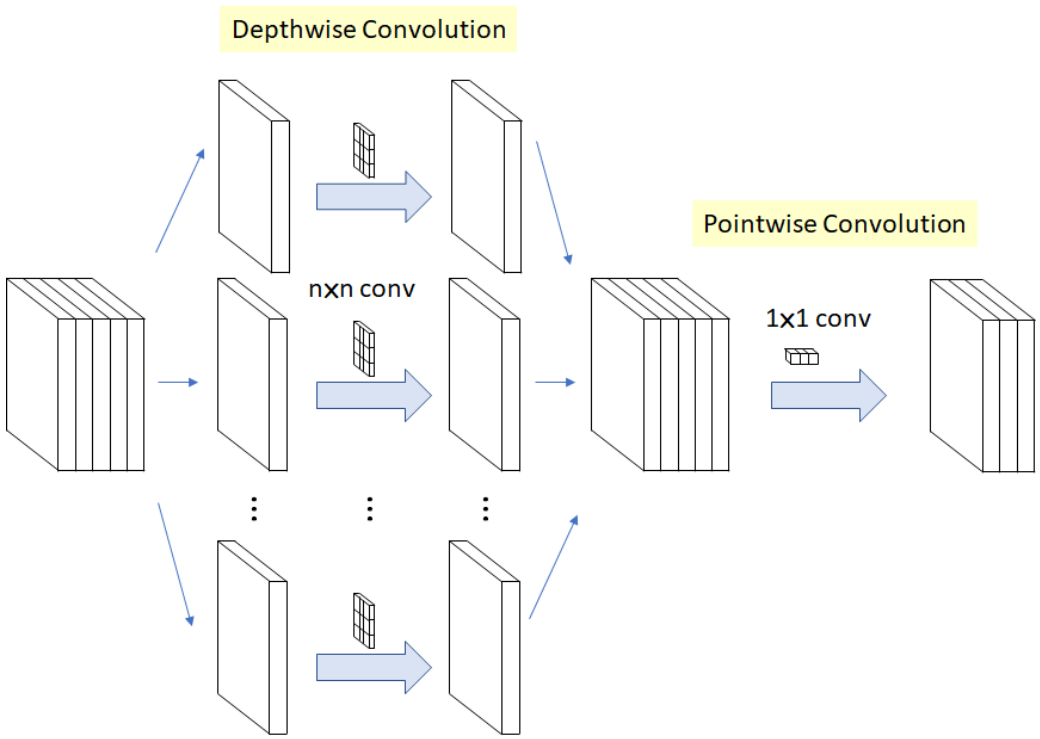

Рисунок 5 – Визуализация глубинной и точечной свертки

Такой подход можно выразить математически. Пусть входной тензор имеет размерность $H \times W \times C_{\text{in}}$, где H и W — пространственные размеры, а C_{in} — количество входных каналов. Глубинная свёртка вычисляется как:

$$\text{DepthwiseConv}(x)_{i,j,k} = \sum_{m,n} x_{i+m,j+n,k} \cdot w_{m,n,k}, \tag{24}$$

где $w_{m,n,k}$ — веса фильтра для каждого канала. После этого точечная свёртка объединяет результаты:

$$\text{PointwiseConv}(y)_{i,j,c} = \sum_{k} y_{i,j,k} \cdot w_{k,c}. \tag{25}$$

Этот процесс уменьшает вычислительную сложность с $O(H \cdot W \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot K^2)$ до $O(H \cdot W \cdot (C_{\text{in}} \cdot K^2 + C_{\text{in}} \cdot C_{\text{out}}))$, где K — размер ядра свёртки.

Для задач классификации временных рядов Xception применяется к их визуальным представлениям, таким как спектрограммы. Это позволяет использовать пространственно-канальную декомпозицию для анализа временной и частотной информации. Xception оказывается особенно эффективным в таких задачах благодаря следующим свойствам:

1. Разделение пространственных и канальных свёрток позволяет лучше адаптироваться к сложной структуре спектрограмм.
2. Небольшое количество параметров модели делает её более подходящей для обучения на ограниченных ресурсах.
3. Глубокие слои с эффективным извлечением признаков обеспечивают высокую точность даже на сложных наборах данных.

Ключевым преимуществом Xception является её способность обрабатывать данные с высокой вычислительной эффективностью, сохраняя точность предсказаний. Модель хорошо справляется с извлечением сложных пространственно-частотных признаков из спектрограмм, что делает её подходящей для задач анализа временных рядов. Однако Xception требует большого объема данных для обучения, так как её глубокая архитектура может склоняться к переобучению на малых выборках.

Xception демонстрирует свою эффективность в задачах классификации временных рядов, где требуется анализ сложных структур данных. В контексте дипломной работы архитектура Xception использовалась для обработки спектрограмм, что позволило извлечь детализированные признаки, отражающие как временные, так и частотные аспекты сигнала. Модель доказала свою эффективность благодаря гибкости и вычислительной оптимизации, обеспечив высокую точность в задаче детекции аврорального километрового радиоизлучения.

1.4.3 ViT

Vision Transformer (ViT) представляет собой архитектуру глубокого обучения, впервые предложенную для обработки изображений, основанную на механизме самовнимания (self-attention). ViT отличается от классических сверточных сетей (CNN) тем, что не использует свёртки для извлечения признаков, а применяет трансформеры, изначально разработанные для задач обработки естественного языка. [21] Это делает ViT мощным инструментом для анализа структурированных данных, таких как спектрограммы временных рядов (см. Рисунок 6).

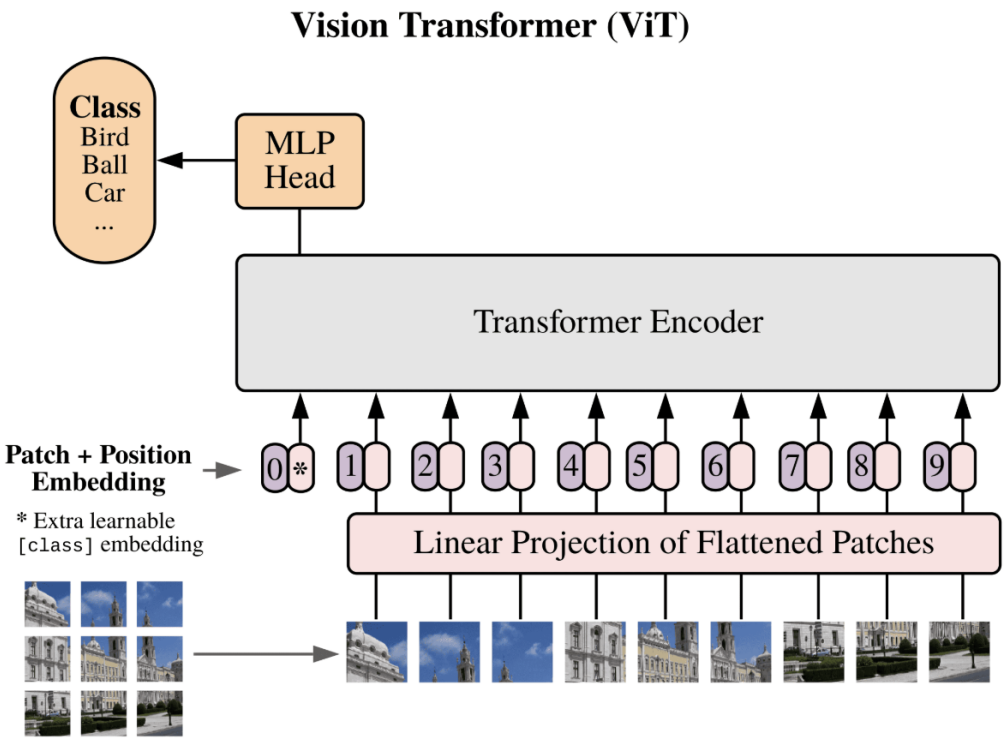

Рисунок 6 – Архитектура Vision Transformer [22]

В основе механизма самовнимания, используемого в Vision Transformer (ViT), лежит определение матриц запросов (Q), ключей (K) и значений (V). Эти матрицы определяют, как различные части данных, такие как патчи изображения или временные интервалы в спектрограммах, взаимодействуют друг с другом. Для каждого входного патча x_p создаются три линейных преобразования, которые формируют его представление как запрос (q_p), ключ (k_p) и значение (v_p). Математически это выражается через линейные операции:

$$q_p = W_q \cdot x_p, \quad k_p = W_k \cdot x_p, \quad v_p = W_v \cdot x_p, \tag{26}$$

где W_q, W_k, W_v — обучаемые матрицы весов, а x_p — исходное представление патча.

Итоговое представление патча формируется как взвешенная сумма значений v_j, где веса определяются матрицей внимания. Такой подход позволяет модели выделять наиболее значимые связи между различными частями данных, что делает ViT особенно эффективным в анализе структурированных данных, таких как изображения или спектрограммы временных рядов.

ViT работает с изображениями или другими матрицами данных следующим образом:

1. Изображение в ViT сначала разбивается на несколько маленьких равных прямоугольных блоков (патчей), обычно размером 16×16 пикселей или 32×32 пикселей, в зависимости от конфигурации. Каждый такой патч x_p преобразуется в одномерный вектор с помощью операции "плоского" преобразования (flattening), где все значения пикселей патча, например, в случае RGB изображения, будут записаны в длинный вектор. Размер этого вектора будет зависеть от размеров патча и числа цветовых каналов (например, для RGB-изображения с патчами 16×16 это будет вектор размером $16 \times 16 \times 3 = 768$).

   Таким образом, x_p — это одномерный вектор, представляющий собой информацию о конкретном патче изображения или фрагменте данных (в случае анализа временных рядов, например, это может быть вектор, описывающий фрагмент временного ряда).
2. Каждый патч преобразуется с помощью линейного слоя, формируя векторное представление z_p, где p — индекс патча:

$$z_p = W_p \cdot \text{vec}(x_p) + b_p, \tag{27}$$

   где W_p — матрица весов, vec — операция векторизации, а x_p представляет собой векторное представление патча изображения.
3. К каждому вектору добавляется позиционная информация, чтобы модель могла учитывать относительное расположение патчей:

$$z_p^{(0)} = z_p + \text{pos}_p. \tag{28}$$

   Поскольку в стандартных трансформерах отсутствует явная информация о порядке входных элементов (например, о пространственном расположении пикселей в изображении), позиционное кодирование добавляется к представлениям патчей, чтобы предоставить информацию о пространственной или временной позиции каждого патча в исходном изображении. Позиционное кодирование pos_p для каждого патча p — это вектор фиксированного размера, который добавляется к соответствующему вектору патча. Это позволяет модели учитывать порядок и пространственные отношения между патчами. В ViT позиционные кодировки обычно создаются с использованием синусоидальных функций или обучаемых векторов.

4. Представления патчей обрабатываются через трансформер, который вычисляет взвешенные связи между всеми патчами, определяя их взаимное влияние:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{29}$$

где Q, K, V — матрицы запросов, ключей и значений, d_k — размерность представлений, а softmax — функция, которая преобразует вектор чисел в вероятностное распределение, где сумма всех элементов равна 1. Для заданного вектора $z = [z_1, z_2, \ldots, z_n]$, функция softmax вычисляется следующим образом:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)}, \tag{30}$$

где: z_i — i-й элемент входного вектора, $\exp(z_i)$ — экспонента от z_i, т.е. e^{z_i}, где e — основание натурального логарифма, $\sum_{j=1}^{n} \exp(z_j)$ — нормирующий коэффициент, равный сумме всех экспонент элементов вектора.

5. После нескольких слоев самовнимания и нормализации итоговое представление патчей передается в полносвязный слой, чтобы получить предсказание.

ViT был адаптирован для анализа временных рядов путём преобразования их в спектрограммы, которые затем рассматриваются как изображения. Основное преимущество ViT в этой задаче — его способность моделировать глобальные зависимости между разными частями данных, что особенно важно для временных рядов с комплексными взаимосвязями.

При обработке спектрограмм временных рядов Vision Transformer (ViT) демонстрирует ряд ключевых свойств. Во-первых, механизм самовнимания позволяет учитывать взаимосвязи между удалёнными регионами спектрограммы, что обеспечивает глубокое понимание глобального контекста и особенно важно для временных рядов с долгосрочной зависимостью. Во-вторых, ViT отличается гибкостью: отказ от использования свёрток предоставляет модели больше свободы в обработке данных с нетривиальными структурными особенностями, позволяя эффективно адаптироваться к различным типам спектрограмм. Наконец, ViT демонстрирует высокую производительность на больших наборах данных благодаря своей архитектуре, однако может уступать классическим

сверточным нейронным сетям (CNN) на небольших выборках из-за высокой сложности и большого числа параметров, что требует значительных объемов данных для обучения.

ViT обладает выдающейся способностью извлекать детализированные признаки и анализировать взаимосвязи между частями данных. Однако одной из сложностей является необходимость большого объема данных для обучения, так как трансформеры имеют тенденцию к переобучению на малых выборках. Кроме того, разделение данных на патчи может приводить к потере мелкомасштабной информации, особенно если размер патча выбран неудачно.

ViT представляет собой инновационный подход к обработке временных рядов, предлагая эффективное использование спектрограмм для анализа. В рамках дипломной работы использование ViT позволило исследовать глобальные зависимости в данных, что является важным для классификации сложных временных сигналов, таких как авроральное километровое радиоизлучение. Модель показала высокую точность и способность к генерализации, что подчёркивает её потенциал в задачах анализа временных рядов.

1.5 Метрики оценки качества обучения

В задачах классификации, включая задачи обнаружения аномалий в мультивариативных временных рядах, важно правильно оценивать качество работы модели. Одним из ключевых инструментов для этого являются метрики, которые позволяют понять, насколько точно и правильно модель классифицирует данные. В этой главе рассмотрены две важные метрики — Confusion Matrix и F1-score которые активно используются в практике машинного обучения для оценки эффективности классификаторов. [23], [24]

Матрица ошибок или Confusion Matrix представляет собой таблицу, которая отображает количество правильных и неправильных предсказаний модели в различных категориях. Она показывает, как алгоритм классификации ошибается в отношении каждой из классов. Матрица состоит из следующих элементов:

	$y = 1$	$y = 0$
$\hat{y} = 1$	True Positive (TP)	False Positive (FP)
$\hat{y} = 0$	False Negative (FN)	True Negative (TN)

Рисунок 7 – Матрица ошибок

Где:

1. TP (True Positive) — количество истинных положительных предсказаний (модель правильно предсказала положительный класс).
2. TN (True Negative) — количество истинных отрицательных предсказаний (модель правильно предсказала отрицательный класс).
3. FP (False Positive) — количество ложных положительных предсказаний (модель ошибочно предсказала положительный класс, когда на самом деле класс был отрицательным).
4. FN (False Negative) — количество ложных отрицательных предсказаний (модель ошибочно предсказала отрицательный класс, когда на самом деле класс был положительным).

С помощью матрицы ошибок можно вычислить несколько ключевых метрик, таких как точность, полнота, F1-score, которые помогут оценить работу модели. Например:

Точность (Accuracy) — доля правильных предсказаний, вычисляется как:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{31}$$

Полнота (Recall) — доля правильных положительных предсказаний среди всех реально положительных случаев:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{32}$$

Точность (Precision) — доля правильных положительных предсказаний среди всех предсказанных положительных случаев:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{33}$$

F1-score является гармоническим средним между точностью (Precision) и полнотой (Recall). Эта метрика особенно полезна, когда важно сбалансировать

точность и полноту, и когда набор данных может быть несимметричным, то есть один класс значительно преобладает над другим.

F1-score вычисляется по следующей формуле:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{34}$$

F1-score принимает значения от 0 до 1, где значение 1 означает идеальное качество модели (все предсказания правильные), а значение 0 указывает на полное отсутствие предсказаний. F1-score особенно полезен в тех случаях, когда важно учитывать как ложные отрицательные, так и ложные положительные предсказания, а не просто количество правильных предсказаний.

В задачах обнаружения аномалий матрица ошибок и F1-score позволяют осуществить оценку эффективности модели. Аномалии, как правило, составляют меньшинство в наборе данных, что делает задачу несимметричной. В таких случаях модель может показывать высокую точность за счет предсказания большинства примеров как нормальных, но при этом пропускать аномалии. В таких ситуациях F1-score является более информативной метрикой, так как она учитывает и ложные отрицательные, и ложные положительные предсказания, помогая обеспечить более сбалансированную оценку работы модели.

Для задач с большими дисбалансами в данных (когда один класс значительно преобладает над другим) F1-score может быть особенно полезным, так как простое использование точности может не давать полной картины эффективности модели.

2 Практическая часть

В открытом доступе существует крайне ограниченное количество реальных данных логирования параметров атак на компьютерные системы. Это объясняется как соображениями конфиденциальности, так и сложностью сбора и предоставления таких данных для публичного анализа. Использование синтетических данных для моделирования сетевых атак, хотя и является распространённой практикой, имеет свои ограничения. Синтетические данные часто не отражают всех важных характеристик и параметров реальных атак, что снижает их ценность для тестирования и анализа алгоритмов машинного обучения в условиях, приближенных к реальным.

В связи с этим было принято решение исследовать методы машинного обучения на примерах реальных задач, которые, хотя и не относятся напрямую к области компьютерной безопасности, обладают схожими характеристиками по структуре данных и сложности анализа. Такой подход позволяет избежать недостатков, связанных с использованием синтетических данных, и сосредоточиться на разработке методик, применимых к широкому спектру задач.

Полученная в ходе данного исследования методика обработки данных и применения алгоритмов машинного обучения может быть адаптирована и использована для анализа данных, относящихся к компьютерной безопасности. Это открывает возможности для разработки более точных и надёжных решений, применяемых в задачах обнаружения аномалий и предотвращения кибератак.

Цель данного практического исследования состоит в поиске, изучении и рассмотрении возможных способов качественного обнаружения аномалий в мультивариативном временном ряду на примере задачи детекции аврорального километрового радиоизлучения. Данная задача имеет важное значение для исследований в области космической физики, поскольку авроральное радиоизлучение связано с электромагнитными процессами в магнитосфере Земли, влияющими на космическую погоду и работу спутниковых систем. Актуальность задачи обусловлена растущей необходимостью мониторинга и прогнозирования таких явлений для минимизации рисков, связанных с влиянием космической погоды на телекоммуникации и навигацию.

2.1 Описание исходного набора данных

Данные были получены с официального сайта ”NASA’s Space Physics Data Facility (SPDF)”. NASA’s Space Physics Data Facility (SPDF) представляет собой онлайн-ресурс, предоставляющий доступ к обширному набору данных, связанных с физикой космоса и исследованиями солнечно-земных взаимодействий. Этот ресурс является хранилищем научных данных, собранных с помощью спутников NASA, международных космических миссий и наземных наблюдений. На сайте SPDF можно найти временные ряды, спектры, данные о магнитных полях, электрических полях, частицах и электромагнитных волнах в различных космических средах, таких как межпланетное пространство, магнитосфера Земли и солнечный ветер. Доступные данные включают как исторические записи, так и текущие результаты миссий, таких как WIND, THEMIS и MAVEN. Ресурс активно используется исследователями для изучения космической погоды, анализа авроральных процессов и других явлений, что делает его важным инструментом для работы в области астрофизики и космических технологий. [25]

В данной работе в качестве мультивариативного временного ряда использовались данные спутника GGS WIND, которые представляют собой значения напряженности электрического поля для частот от 20 кГц до 13825 кГц, измеряемые раз в минуту на протяжении 18720 минут. Измерения были зафиксированы в период времени с 1 июля 2020 года в 00:00:30 по 13 июля 2020 года в 23:59:30. Каждой частоте в диапазоне с шагом 4 кГц соответствовал отдельный временной ряд, фиксирующий напряженность электрического поля в конкретный момент времени.

Данные на сайте NASA’s Space Physics Data Facility (SPDF) обычно хранятся в формате Common Data Format (CDF). Этот формат оптимизирован для хранения научных данных, таких как временные ряды, спектры или многомерные массивы, и поддерживает метаданные, описывающие структуру и содержание файла. Файлы CDF содержат переменные, которые могут быть скалярными (одномерными) или многомерными массивами. Каждая переменная сопровождается атрибутами, описывающими единицы измерения, источник данных и другие характеристики.

Данные для исследования содержались в 13 файлах формата cdf. Структура данных в каждом файле CDF выглядит следующим образом:

```
E_VOLTAGE_RAD1: CDF_REAL4 [1440, 256]
```

```
E_VOLTAGE_RAD2: CDF_REAL4 [1440, 256]
E_VOLTAGE_TNR: CDF_REAL4 [1440, 96]
Epoch: CDF_EPOCH [1440]
Epoch2: CDF_EPOCH [1]
Frequency_RAD1: CDF_INT2 [256] NRV
Frequency_RAD2: CDF_INT2 [256] NRV
Frequency_TNR: CDF_REAL4 [96] NRV
Minimum_voltage_RAD1: CDF_REAL4 [1, 256]
Minimum_voltage_RAD2: CDF_REAL4 [1, 256]
Minimum_voltage_TNR: CDF_REAL4 [1, 96]
```

В данном случае:

1. E_VOLTAGE_RAD1 — эта переменная содержит значения электрического напряжения, зарегистрированные инструментом RAD1 (Radio Receiver Band 1) за каждый момент времени. Первая размерность (1440) соответствует количеству временных отсчётов в течение дня, а вторая (256) — количеству частотных каналов. Данные представлены в формате CDF_REAL4, что означает числа с плавающей точкой (4 байта). Они используются для анализа сигналов в определённом диапазоне частот;
2. E_VOLTAGE_RAD2 — аналогично переменной E_VOLTAGE_RAD1, эта переменная содержит данные о напряжении, зарегистрированном инструментом RAD2 (Radio Receiver Band 2), работающем в другом диапазоне частот. Формат и структура совпадают с E_VOLTAGE_RAD1;
3. E_VOLTAGE_TNR — эта переменная представляет напряжение, зарегистрированное инструментом TNR (Thermal Noise Receiver), который фиксирует тепловой шум. Размерность массива отличается: 96 частотных каналов и 1440 временных отсчётов. Инструмент TNR используется для более точного анализа сигналов в нижнем диапазоне частот;
4. Переменная Epoch представляет временные метки для каждого из 1440 измерений в течение дня. Формат CDF_EPOCH используется для представления временных данных в формате UTC с высокой точностью, что позволяет сопоставить данные с глобальным временем наблюдений;
5. Переменная Epoch2 содержит единственную временную метку, представляющую общее время начала записи данных или другой важный временной момент, относящийся ко всему файлу;
6. Переменная Frequency_RAD1 представляет массив частотных каналов, соответствующих измерениям инструмента RAD1. Формат CDF_INT2 означает, что данные хранятся в виде 16-битных целых чисел. Маркер

NRV (No Record Variance) указывает, что эти значения постоянны для всех временных отсчётов;

7. Frequency_RAD2, аналогично Frequency_RAD1, содержит частоты для инструмента RAD2. Они остаются неизменными для всех временных меток;
8. Переменная Frequency_TNR содержит частоты, использованные инструментом TNR. Данные представлены в формате CDF_REAL4 и имеют более низкое разрешение (96 каналов). Они также не зависят от временных отсчётов;
9. Переменная Minimum_voltage_RAD1 содержит минимальные значения напряжения для каждого частотного канала, зарегистрированные инструментом RAD1. Первая размерность (1) указывает на то, что данные фиксированы для всех временных отсчётов;
10. Данные Minimum_voltage_RAD2, аналогично Minimum_voltage_RAD1, отражают минимальные значения напряжения для инструмента RAD2 по каждому из частотных каналов;
11. Переменная Minimum_voltage_TNR содержит минимальные значения напряжения для инструмента TNR. Меньшее количество частотных каналов (96) соответствует специфике работы этого инструмента.

2.2 Описание инструментов и библиотек программной реализации

В процессе выполнения данной работы использовались современные программные инструменты и библиотеки, которые обеспечивают высокую эффективность при работе с данными, машинным обучением и глубоким обучением. Ниже приведено краткое описание основных из них.

Python является универсальным и широко используемым языком программирования, идеально подходящим для анализа данных, статистической обработки и машинного обучения. Его обширная экосистема библиотек делает его незаменимым инструментом для научных исследований. [26]

Pandas предоставляет удобный интерфейс для работы с табличными данными. С помощью этой библиотеки можно легко обрабатывать, фильтровать и агрегировать данные, а также преобразовывать их в удобные форматы, такие как датафреймы. Pandas использовалась для обработки данных, извлечённых из формата CDF, и для подготовки входных данных для машинного обучения. [27]

Библиотека SpacePy, а точнее её модуль pycdf, позволяет работать с файлами в формате Common Data Format (CDF). Данная библиотека использовалась

для извлечения временных рядов из данных NASA и их последующей конвертации в датафреймы. [28]

NumPy предоставляет средства для работы с массивами данных и числовыми вычислениями. Эта библиотека была использована для выполнения линейной алгебры, обработки многомерных массивов и подготовки данных для визуализации спектрограмм. [29]

Scikit-learn — библиотека для классических методов машинного обучения. Она использовалась для реализации алгоритма метода k ближайших соседей (kNN), а также для оценки качества моделей с помощью метрик, таких как F1-score. [30]

Библиотека PyTS предоставляет набор инструментов для работы с временными рядами, включая различные методы классификации, преобразования и анализа. Одним из важных для данной работы компонентов этой библиотеки является класс TimeSeriesForest, который представляет собой алгоритм классификации для временных рядов, основанный на случайных лесах. Этот метод предназначен для решения задач классификации, где данные представлены в виде временных рядов. [31]

CatBoost — библиотека для обучения градиентного бустинга. Она эффективно работает с категориальными признаками и была применена для построения моделей классификации временных рядов. [32]

Pickle — стандартная библиотека Python для сериализации объектов. Она применялась для сохранения и загрузки обученных моделей, чтобы обеспечить их повторное использование без необходимости повторного обучения. [33]

SciPy предоставляет множество инструментов для научных вычислений. Модуль scipy.signal.spectrogram использовался для построения спектрограмм временных рядов с помощью оконного преобразования Фурье. [34]

Matplotlib — основная библиотека для построения графиков и визуализации данных. Seaborn — её надстройка, которая упрощает создание эстетичных и информативных визуализаций. [35] Эти библиотеки использовались для анализа данных, визуализации результатов и создания спектрограмм. [36]

PyTorch — одна из наиболее популярных библиотек для глубокого обучения. Она предоставляет гибкий интерфейс для построения и обучения нейронных сетей. PyTorch использовалась для реализации модели ResNet34, а также для обучения других нейронных сетей. [37]

Torchvision — библиотека для работы с изображениями, входящая в состав экосистемы PyTorch. Она предоставляет предобученные модели, включая ResNet34, а также инструменты для обработки изображений. [38]

Timm (PyTorch Image Models) — библиотека, предоставляющая доступ к большому количеству архитектур нейронных сетей для задач компьютерного зрения. В данном исследовании с её помощью была использована модель Xception. [39]

Transformers — библиотека от Hugging Face для работы с трансформерными архитектурами. Она предоставляет готовые реализации моделей, таких как ViT (Vision Transformer), которые были использованы для классификации временных рядов. [40]

2.3 Обработка набора данных

Авроральное километровое радиоизлучение (АКР) — это интенсивное природное радиоизлучение в диапазоне частот 30–800 кГц, создаваемое в околоземной плазме и распространяющееся от Земли. Необходимый диапазон частот был указан в параметре Frequency_RAD1. Так как ему соответствует массив E_-VOLTAGE_RAD1 значений нормализованного напряжения электрического поля, дальнейшее исследование и обработка осуществлялось относительно него. Все остальные массивы данных помимо выше упомянутых и параметра Epoch использованы не были.

С помощью библиотеки SpacePy и Pandas было осуществлено преобразование исходных массивов данных в более удобный формат для обработки средствами машинного обучения и нейросетями. Далее были вручную выделены временные промежутки, на которых наблюдалось АКР. Каждому моменту времени было сопоставлено значение, которое равнялось единице, если в этот момент времени наблюдалось АКР, или равнялось нулю, если в этот момент времени АКР не наблюдалось.

В результате получился мультивариативный временной ряд, состоящий из 256 временных рядов, разделенных на интервалы, где фиксировалось наличие или отсутствие аврорального километрового излучения (см. Рисунок 8 и 9).

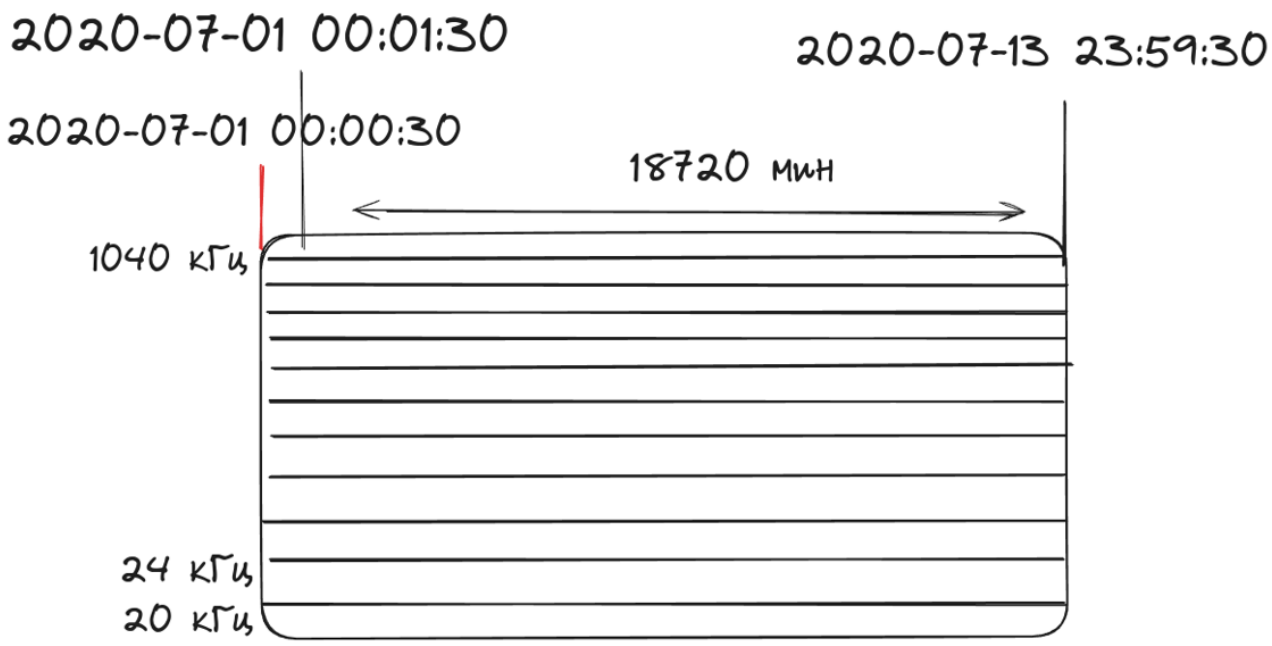

Рисунок 8 – Схематичное изображение набора данных (в диапазоне частот от 20 кГц до 1040 кГц)

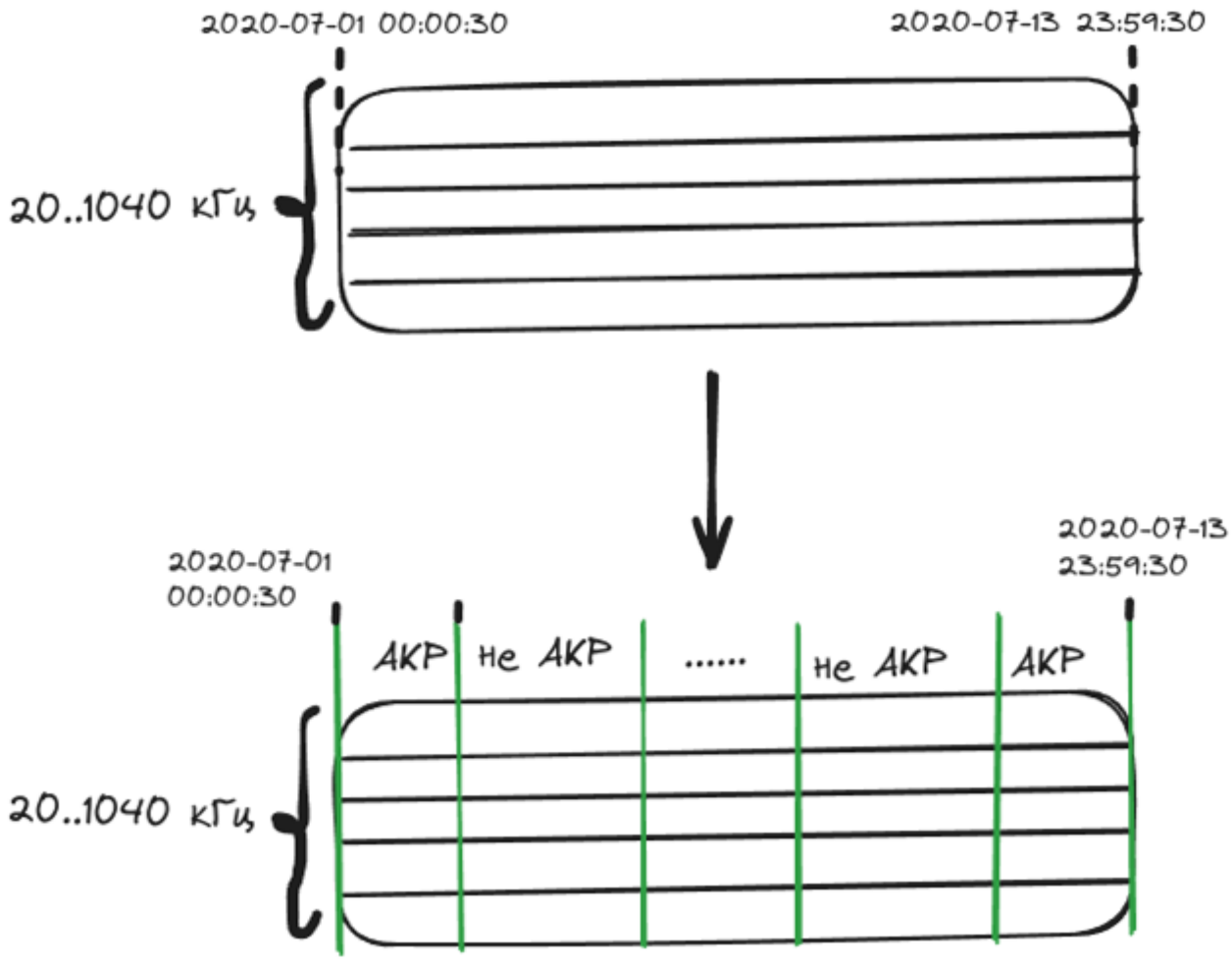

Рисунок 9 – Схематичное изображение состояния набора данных после разметки

Так как используемые мною для классификации нейронные сети и алгоритмы машинного обучения предполагают одинаковую размерность всех входных образцов, на втором этапе обработки данных стояла задача разделения мультивариативного временного ряда на интервали/блоки одинаковой размерности.

Для выбора оптимальной размерности были проанализированы длительности интервалов с АКР и интервалов без АКР. Минимальная длительность промежутка с АКР составила 4 минуты, максимальная — 1193 минуты. Минимальная длительность промежутка без АКР составила 2 минуты, максимальная — 2433 минуты. Было построено и визуализированно нормальное распределение значений длительности АКР, на основании которого осуществлялся выбор размерности блоков (см. Таблица 1 и Рисунок 10).

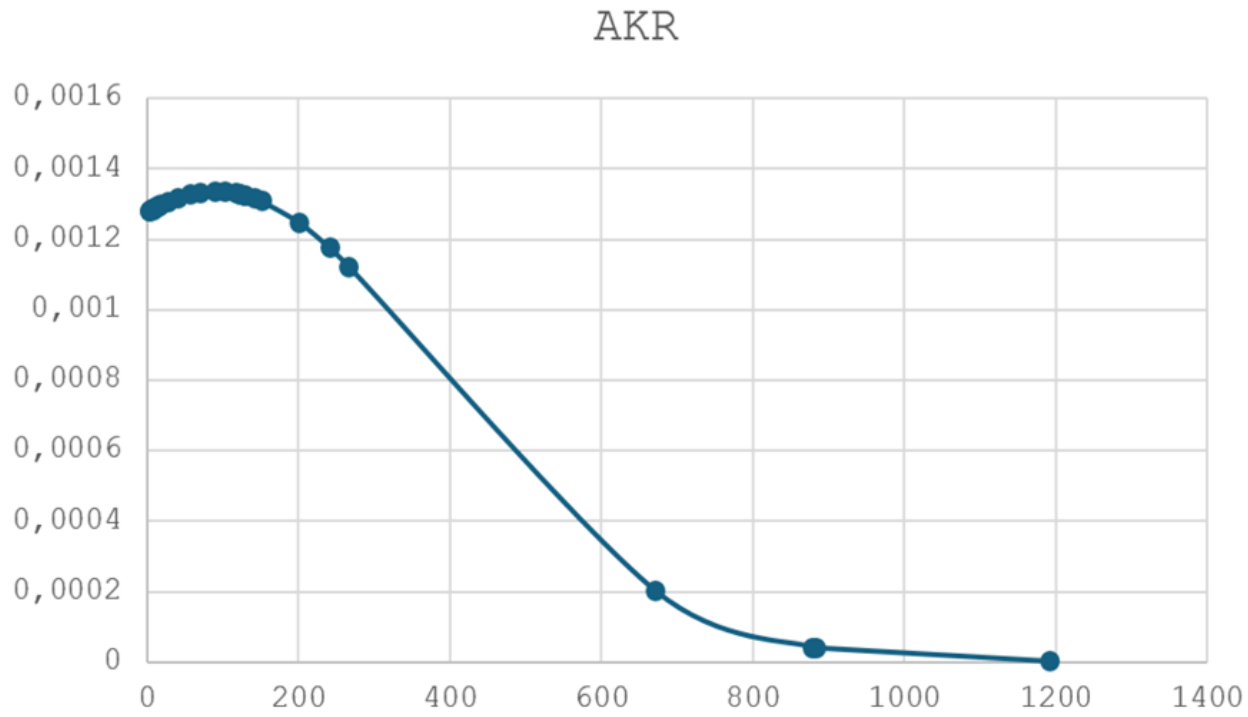

Рисунок 10 – Нормальное распределение значений длительности АКР. Ось X — значения длительности промежутков АКР, ось Y - плотность вероятности

Для блоков были выбраны размеры длиной в 4, 16 и 32 моментов времени по следующим причинам:

1. Размер 4 обусловлен тем, что 4 - это минимальная длительность АКР; самый длинный интервал АКР без перекрытия можно представить как 298 блоков длины 4, что количественно больше, чем при использовании блоков больших размеров;

Длит-ть	Плотность	Длит-ть	Плотность	Длит-ть	Плотность
4	0.001279747	8	0.001284630	58	0.001327042
4	0.001279747	9	0.001285817	71	0.001332173
6	0.001282215	12	0.001289300	91	0.001335160
8	0.001284630	14	0.001291555	104	0.001333897
15	0.001292662	17	0.001294836	118	0.001329721
19	0.001296955	28	0.001305810	123	0.001327526
41	0.001316597	130	0.001323836	130	0.001323836
142	0.001315853	152	0.001307625	201	0.001247681
242	0.001175105	267	0.001122518	672	0.000201611
879	0.0000412351	883	0.0000398011	1193	0.00000148541

Таблица 1 – Длительность (в минутах) и плотность вероятности АКР

2. Размер 32 обусловлен тем, что значения длительности АКР в диапазоне [28, 41] имеют большую плотность вероятности, чем значения длительности АКР в диапазонах [4, 28], и при этом блоки длительности 32 требуют меньше вычислительной мощности, чем блоки большей размерности;
3. Размер 16 был выбран как промежуточный шаг между 4 и 32: это ближайшее значение к среднему арифметическому между 4 и 32 (18). При этом 16 является степенью двойки, что ускоряет быстрое преобразование Фурье.

В соответствии с выбранными значениями были созданы три выборки, состоящие из последовательностей одинаковой длины. Преобразование в интервалы (блоки) одной длины осуществлялось с помощью скользящего окна (см. Рисунок 11).

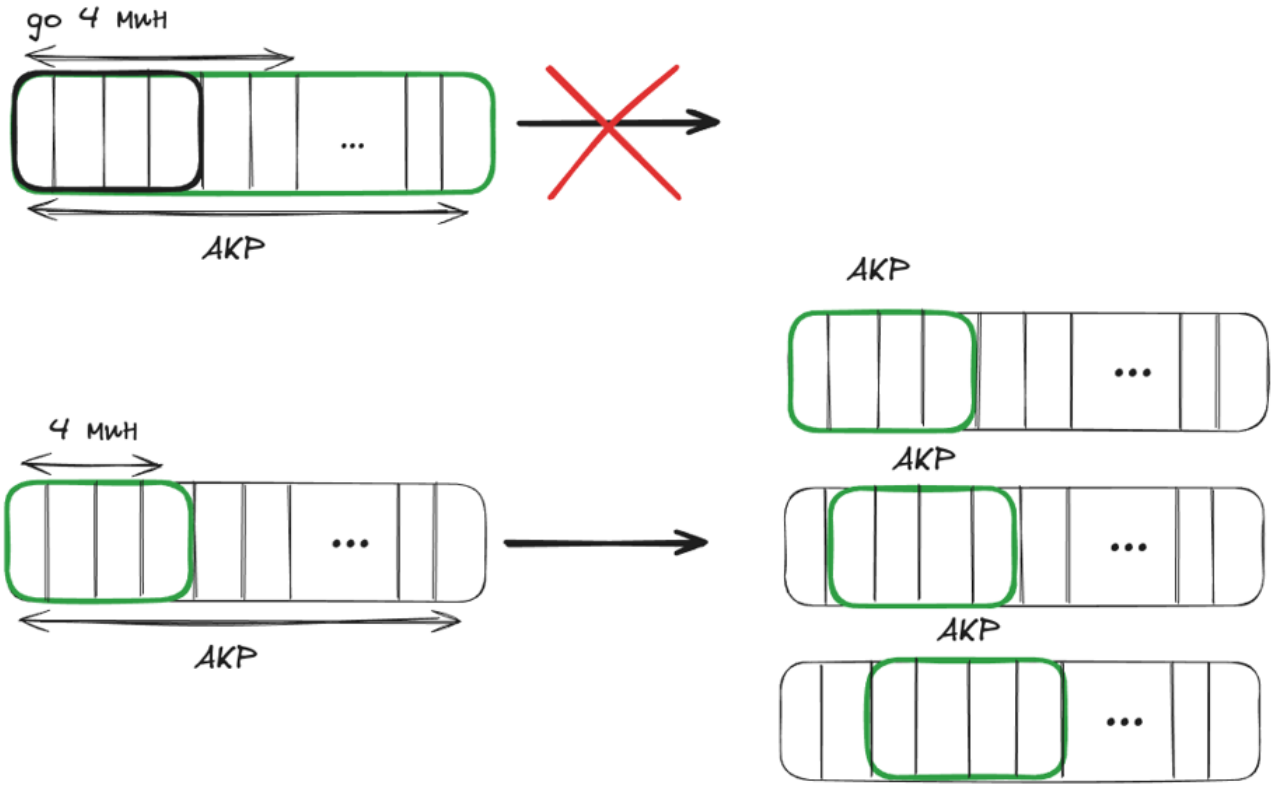

Рисунок 11 – Схематичное изображение примера применения скользящего окна к набору данных

Механизм преобразования интервалов данных с наличием и отсутствием АКР в блоки одинаковой длины с использованием скользящего окна заключается в последовательном выделении подмассивов фиксированной длины из исходного блока данных. Если длина блока превышает заданную целевую длину, из него формируются новые блоки путём ”скольжения” окна фиксированной длины по исходному массиву данных. Начальная позиция окна сдвигается на определённое количество элементов (шаг скользящего окна) после каждого извлечения подмассива. Если же длина блока меньше целевой длины, то такой блок не используется при формировании нового набора данных.

Например, если целевая длина блока равна n, то первое окно захватывает элементы с 1-го по n-й, затем сдвигается на заданный шаг и захватывает элементы с 2-го по $(n+1)$-й и так далее, пока не будет достигнут конец исходного блока. Каждому выделенному подмассиву присваивается уникальный идентификатор, чтобы они рассматривались как отдельные новые блоки. Этот метод позволяет эффективно разбить исходные данные на множество частей одинаковой длины, сохраняя при этом локальную структуру данных и их временную последовательность.

Для длины 4 получилась выборка в 18534 интервалов, для 16 - 17912 интервалов, для 32 - 17280 интервалов. Затем для классических моделей машинного обучения осуществлялась балансировка количества элементов в обоих классах. Механизм балансировки выборок в рамках данной работы направлен на уравнивание количества блоков данных относительно классов, определяющих наличие или отсутствие АКР. Процесс начинается с разделения исходного набора данных на две подвыборки: одну, содержащую блоки с АКР, и другую, содержащую блоки без АКР. После этого определяется минимальное количество блоков между этими двумя группами. Если уже наблюдается равенство размеров групп, данные остаются неизменными. В противном случае, применяется метод ”undersampling” к избыточной группе, то есть из неё случайным образом выбирается подмножество блоков, равное по размеру меньшей группе. Это достигается с использованием специального метода sample у объекта библиотеки Pandas, который случайным образом выбирает строки из группы, при этом фиксируется генератор случайных чисел через параметр random_state для обеспечения воспроизводимости результатов. Наконец, обе группы объединяются обратно в один набор данных, при этом количество блоков с наличием и отсутствием АКР становится одинаковым. Такой подход позволяет получить сбалансированный набор данных, который улучшает работу алгоритмов машинного обучения, особенно в случаях, когда дисбаланс классов мог бы привести к перекосу в сторону более многочисленного класса. Каждая из выборок была в дальнейшем разделена на подвыборки для обучения и для теста классических моделей МО (см. Таблица 2). Каждый из экспериментов проводился относительно блоков одной длины.

	Обучающая выборка	Тестовая выборка
Блоки с АКР	4512	1150
Блоки без АКР	4547	1115

Таблица 2 – Информация о количестве блоках с АКР и без АКР в обучающей и тестовой выборках

2.4 Параметры классических алгоритмов машинного обучения

Важно упомянуть, что относительно каждого временного ряда (т.е. для каждой частоты) создавалась отдельная модель машинного обучения, которая обучалась и тестировалась на данных своей определенной частоты. То есть

процесс определения того, является ли приведенный интервал времени размерности 4, 16 или 32 блоком, в котором присутствует или отсутствует авроральное километровое радиоизлучение, состоял в следующем: для каждой частоты определенная модель прогнозировала класс, после чего выбиралось наиболее популярное значение класса среди всех моделей.

Для обучения и тестирования моделей машинного обучения TimeSeriesForest, kNN и CatBoost были использованы следующие подходы. Модели TimeSeriesForest и kNN запускались с параметрами по умолчанию. В случае с TimeSeriesForest, алгоритм использует случайные подвыборки признаков для построения ансамбля деревьев решений, и по умолчанию используется количество деревьев, равное 500, а также другие параметры, такие как максимальная глубина деревьев и критерий расщепления, устанавливаемые внутренними значениями, оптимальными для большинства задач. Алгоритм kNN (k-Nearest Neighbors) использует стандартные параметры, такие как число соседей (по умолчанию 5) и метрику Евклида для расчета расстояния между точками, что делает модель простой, но эффективной для классификации в задачах с небольшими данными.

Для модели CatBoost использовался подход с подбором гиперпараметров с помощью GridSearchCV. Для этого был определен список параметров с различными значениями для количества итераций (iterations), темпа обучения (learning_rate) и глубины деревьев (depth). Конкретно, для параметра iterations были протестированы значения от 1 до 5, для learning_rate — значения 0.01, 0.1 и 1, а для depth — значения от 1 до 5. Использование GridSearchCV позволило найти оптимальные комбинации этих параметров и выбрать наилучшую модель CatBoost для предсказания.

2.5 Создание спектрограмм

Затем, на основе выборок, полученных до балансировки, были созданы наборы данных для обучения алгоритмов компьютерного зрения (ResNet34, Xception, ViT), состоящие из спектрограмм. Отсутствие балансировки классов для ViT, ResNet и Xception обосновано спецификой работы данных моделей и характером задачи. Эти модели, основанные на глубоком обучении, обладают высокой способностью извлекать сложные паттерны из данных, что делает их менее чувствительными к дисбалансу классов по сравнению с более простыми алгоритмами. Во-первых, глубокие нейронные сети способны адаптироваться к

дисбалансу классов за счёт использования взвешенных функций потерь. Например, при наличии дисбаланса можно назначать больший вес ошибкам на классе с меньшим количеством объектов, что позволяет компенсировать влияние преобладающего класса. Это устраняет необходимость прямой балансировки данных. Во-вторых, объём данных, используемый для обучения этих моделей, имеет достаточно элементов обоих классов для эффективного обучения. При этом модели обучаются учитывать соотношение между классами, представленное в данных, что делает выборку реалистичной и ближе к реальным условиям, в которых предполагается использовать модель. В-третьих, архитектуры ViT, ResNet и Xception обладают большой ёмкостью модели, что позволяет им изучать даже редкие шаблоны, характерные для уступающего в объеме образцов класса. Это контрастирует с алгоритмами меньшей сложности, которые могут быть склонны к переобучению на преобладающем классе, если дисбаланс не был устранён. Кроме того, в задачах, связанных с изображениями, балансировка данных может не всегда быть необходимой, так как модели могут извлекать важную информацию из структурных особенностей данных, независимо от их распределения. Таким образом, отсутствие балансировки позволяет моделям сохранять исходное распределение классов и лучше отражать реальные данные, что особенно важно при тестировании и развертывании модели.

В данной работе для построения спектрограмм использовался алгоритм с использованием функции spectrogram из библиотеки scipy.signal. Спектрограммы строились для временных блоков данных с использованием различных параметров оконного преобразования в зависимости от длины блоков. Основной задачей было преобразовать временные ряды в набор спектральных изображений, пригодных для дальнейшего анализа.

Для каждого блока, в зависимости от его длины, задавались определенные параметры оконного преобразования: длина окна и степень перекрытия между окнами. Для блоков длиной 4 использовалось окно длиной 2 с перекрытием 1; для блоков длиной 16 — окно длиной 4 с перекрытием 2; для блоков длиной 32 — также окно длиной 4 с перекрытием 2. Эти параметры выбирались таким образом, чтобы обеспечить оптимальное разрешение спектра при сохранении информативности временных характеристик сигнала.

Для каждого блока спектрограмма вычислялась отдельно для всех частотных каналов. Среднее значение спектральной мощности по всем каналам

суммировалось и усреднялось для получения итоговой спектрограммы блока. Это позволило учесть вклад всех каналов и снизить влияние возможных шумов или выбросов в отдельных каналах.

Если итоговая спектрограмма имела крайне низкую мощность сигнала (что могло указывать на отсутствие полезной информации), она исключалась из дальнейшего анализа. Готовые спектрограммы визуализировались с помощью библиотеки Matplotlib в виде изображений, где оси времени и частоты были убраны для сохранения чистоты данных. Каждая спектрограмма сохранялась в виде изображения без полей, а информация о соответствующих блоках временного ряда и времени их начала сохранялась в виде структуры данных Pandas. Примеры изображений спектрограмм с учетом осей представлены ниже. Ось Y это нормализованные частоты (в долях от частоты дискретизации, при Частоте дискретизации равной 1 Гц). Ось X — индексы временных окон (номера отсчётов). Временные метки t представляют центры временных окон. Каждая точка на оси X соответствует временному интервалу, в котором вычислялась спектральная плотность мощности. Центры временных окон — это моменты времени, которые соответствуют серединам каждого окна анализа в спектрограмме. Цветовая шкала определяет мощность спектра в логарифмическом масштабе (дБ).

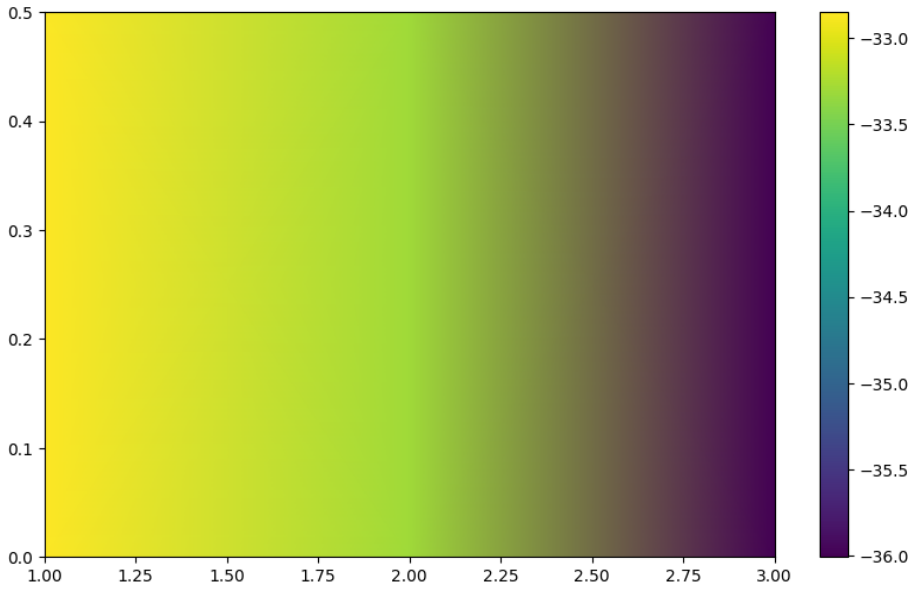

Рисунок 12 – Пример спектрограммы, построенной на основе блока длиной 4

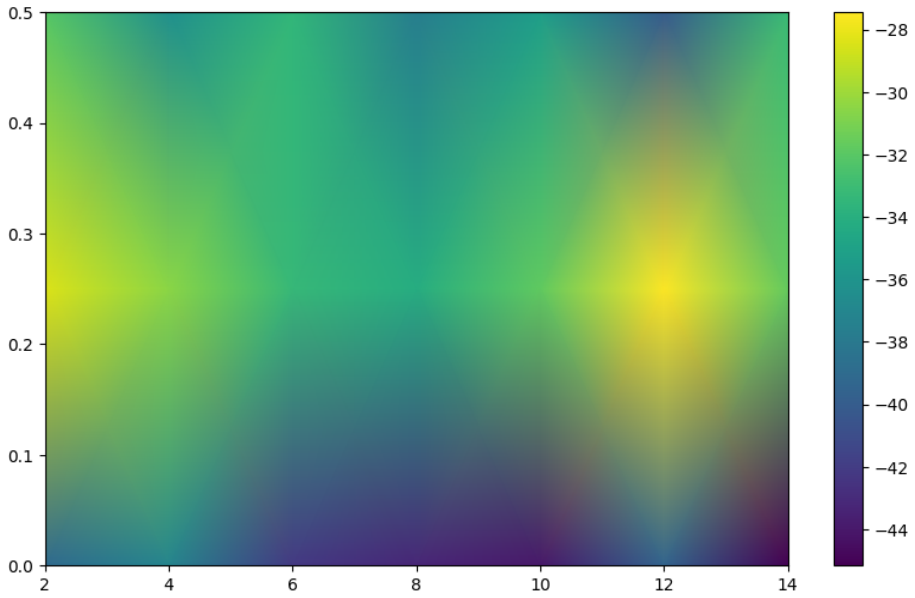

Рисунок 13 – Пример спектрограммы, построенной на основе блока длиной 16

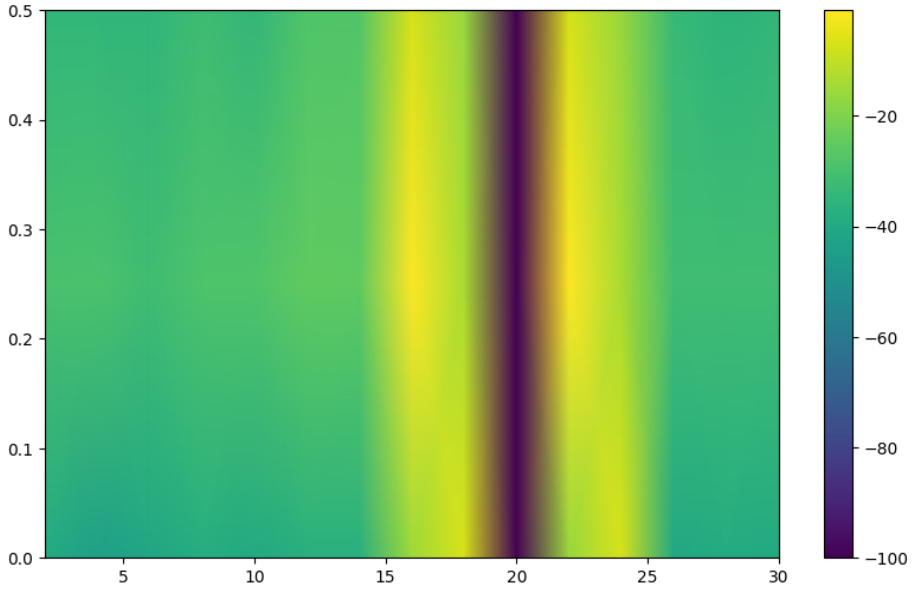

Рисунок 14 – Пример спектрограммы, построенной на основе блока длиной 32

Подход обеспечил гибкость построения спектрограмм и возможность работы с временными рядами разной длины, создавая основу для дальнейшего применения спектральных изображений в рамках экспериментов. Информация про соотношение количества спектрограмм того или иного класса с длинами 4, 16 и 32 содержится в таблице 3. Выборка изображений делилась на обучающую и тестовую в соотношении 4 к 1.

2.6 Параметры алгоритмов компьютерного зрения

Для обучения с последующей проверкой качества классификации спектрограмм каждая из моделей ViT, ResNet34 и Xception настраивалась с исполь-

	Длина 4	Длина 16	Длина 32
Кол-во спектрограмм с АКР	5662	5345	5022
Кол-во спектрограмм без АКР	12873	12567	12258

Таблица 3 – Количество спектрограмм с АКР и без АКР для блоков разной длины

зованием соответствующих гиперпараметров и алгоритмов оптимизации.

Для экспериментов с ViT использовалась предобученная архитектура google/vit-base-patch16-384, предоставляемая библиотекой transformers. Модель была модифицирована для задачи бинарной классификации с установкой параметра num_labels=2 и включением опции ignore_mismatched_sizes, которая позволяет корректно адаптировать размеры слоев к измененной задаче. В процессе обучения применялся оптимизатор AdamW с фиксированной скоростью обучения, равной 5×10^{-5}. Регулирование скорости обучения осуществлялось с помощью линейного планировщика (linear scheduler). Полный процесс обучения включал 5 эпох, что соответствует числу шагов обучения, равному произведению числа эпох и количества батчей в тренировочном наборе данных.

Для эксперимента с ResNet34 была использована архитектура ResNet34 с модификацией последнего полносвязного слоя. Вместо стандартного слоя был добавлен каскад из линейного слоя с 512 нейронами, функции активации ReLU, слоя дропаута с вероятностью 0.2, и финального линейного слоя с двумя выходами. Для расчета вероятностей классов использовалась функция LogSoftmax, обеспечивающая логарифмическую шкалу выходных вероятностей. В качестве функции потерь применялась отрицательная логарифмическая правдоподобность (NLLLoss), а оптимизация проводилась с использованием алгоритма Adam с коэффициентом скорости обучения 1×10^{-4}. Процесс обучения включал 10 эпох, обеспечивающих достаточную сходимость параметров модели.

Модель Xception была реализована на основе предобученной версии, предоставляемой библиотекой timm. Архитектура адаптирована для задачи бинарной классификации. Для вычисления функции потерь использовалась отрицательная логарифмическая правдоподобность (NLLLoss), а оптимизация параметров осуществлялась алгоритмом Adam с фиксированной скоростью обучения 1×10^{-4}. Обучение проводилось в течение 10 эпох, что обеспечивало стабильную производительность модели.

Все вычисления выполнялись с использованием графического процессо-

ра (GPU), что ускоряло процесс обработки данных и обеспечивало эффективное обучение моделей. Применение разнообразных архитектур, включая трансформерные и свёрточные нейронные сети, позволило провести сравнительный анализ их эффективности для задачи классификации спектрограмм, а также выявить преимущества и недостатки каждой из них.

2.7 Программная реализация

Перед обработкой данных было осуществлено преобразование исходных данных формата CDF в формат csv. Данный этап исследования содержится в файле ”preprocessing.py”. Этап обработки данных, балансирования выборок и разделения на обучающую и тестовую выборки для алгоритмов машинного обучения содержится в скрипте ”processing.py”. Процесс создания спектрограмм с последующим созданием набора данных для алгоритмов компьютерного зрения указан в файле ”spectrogramms.py”. Процесс обучения и теста классических моделей машинного обучения содержится в ”classic.py”. Аналогично классическим моделям, для моделей компьютерного зрения программный код размещен в файле ”cv.py”.

2.8 Результаты

На основе проведенных экспериментов с алгоритмами машинного обучения, а также с алгоритмами компьютерного зрения, были получены результаты относительно каждой из выборок. Результаты представлены в Таблице 4. Все модели оценивались по трем основным метрикам: TPR (True Positive Rate), TNR (True Negative Rate) и F1-score. Метрика TPR, также известная как чувствительность, отражает долю правильно классифицированных положительных примеров среди всех положительных наблюдений, или же вероятность корректного определения блока с АКР в рамках конкретной выборки. Метрика TNR, также называемая специфичностью, измеряет долю правильно классифицированных отрицательных примеров среди всех отрицательных наблюдений, или же вероятность корректного определения блока без АКР в рамках конкретной выборки. F1-score, в свою очередь, представляет собой гармоническое среднее точности (Precision) и полноты (Recall), что делает его полезным для оценки моделей в условиях несбалансированных классов.

Размер блока	Модель	TPR	TNR	F1-score
4	CatBoost	0.00	1.00	0.00
4	kNN	0.74	0.90	0.80
4	TimeSeriesForest	0.75	0.89	0.80
4	ViT	0.30	0.90	0.76
4	Xception	0.91	0.30	0.70
4	ResNet34	0.91	0.28	0.68
16	CatBoost	0.00	1.00	0.00
16	kNN	0.70	0.83	0.76
16	TimeSeriesForest	0.71	0.81	0.77
16	ViT	0.74	0.88	0.84
16	Xception	0.88	0.73	0.82
16	ResNet34	0.66	0.84	0.77
32	CatBoost	0.00	1.00	0.00
32	kNN	0.69	0.80	0.75
32	TimeSeriesForest	0.70	0.79	0.76
32	ViT	0.97	0.99	0.98
32	Xception	0.97	0.98	0.97
32	ResNet34	0.87	0.86	0.86

Таблица 4 – Значения метрик обучения моделей для различных размеров блоков

На основании значений метрик также были построены графики зависимости значения каждой из метрик от длины блока (см. рисунки ниже).

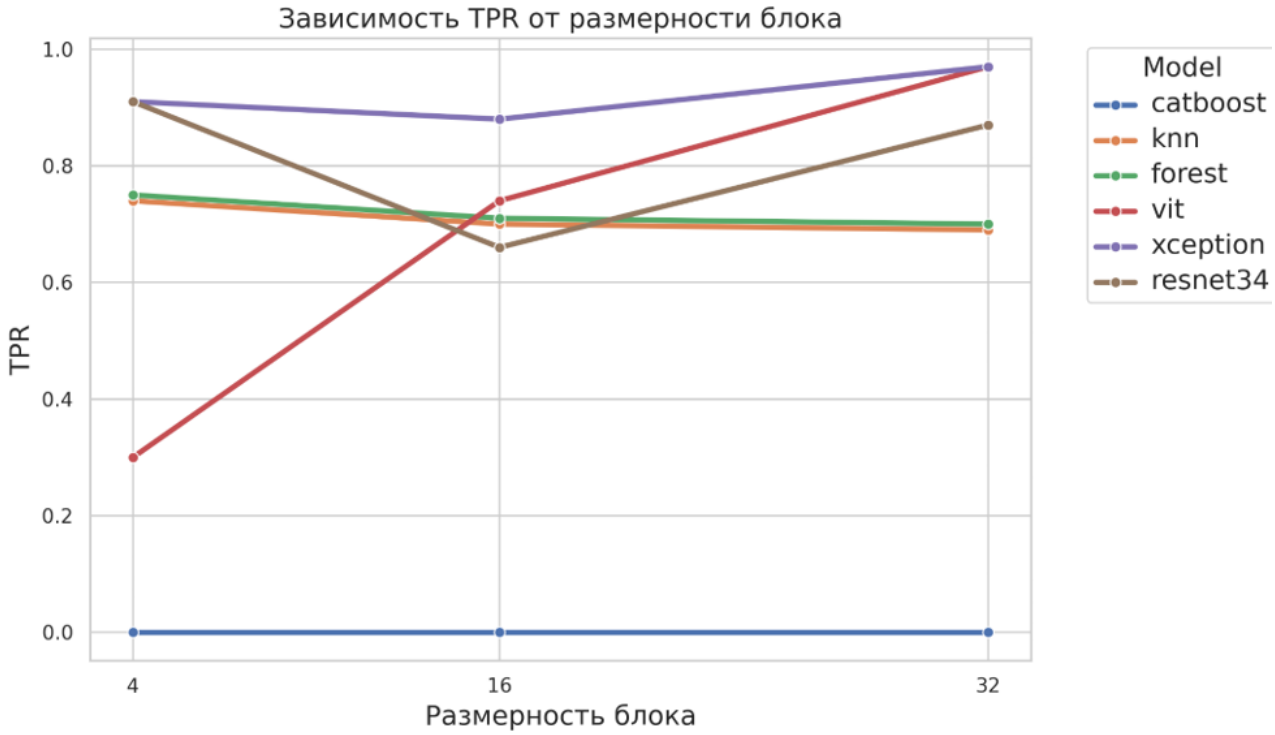

Рисунок 15 – График зависимости метрики TPR от длины блока

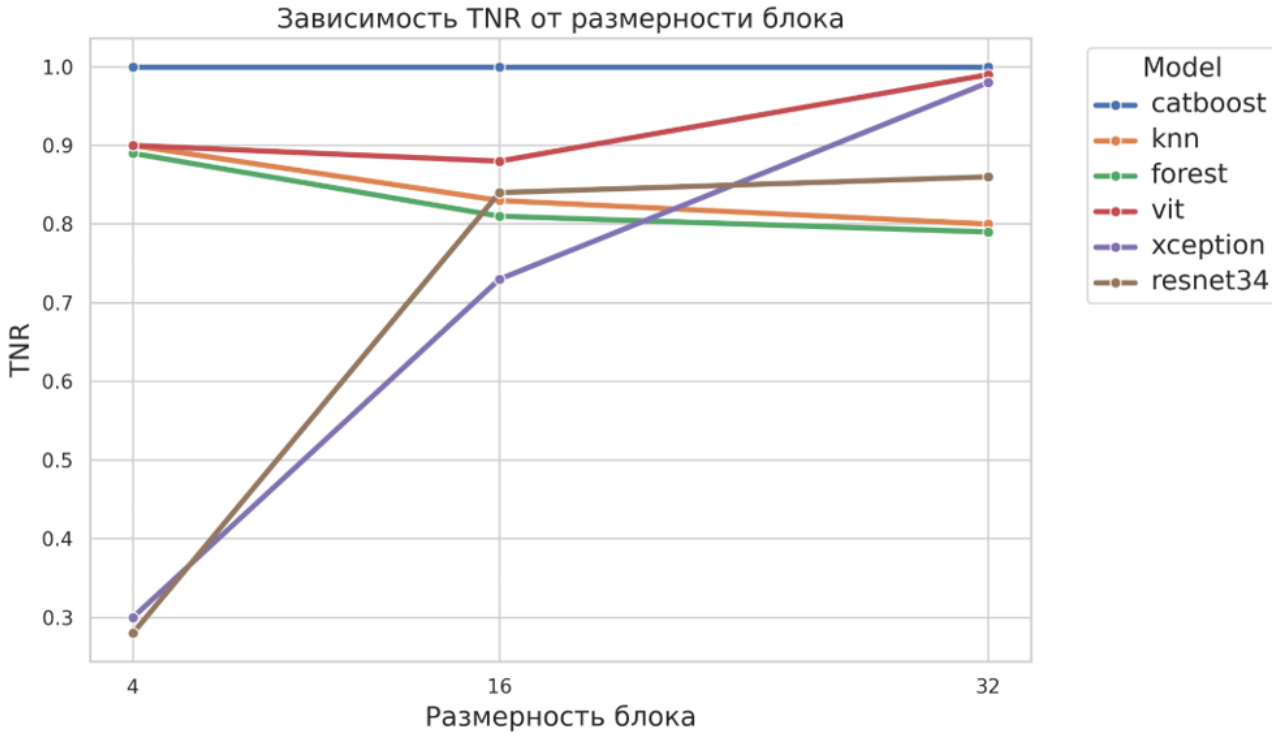

Рисунок 16 – График зависимости метрики TNR от длины блока

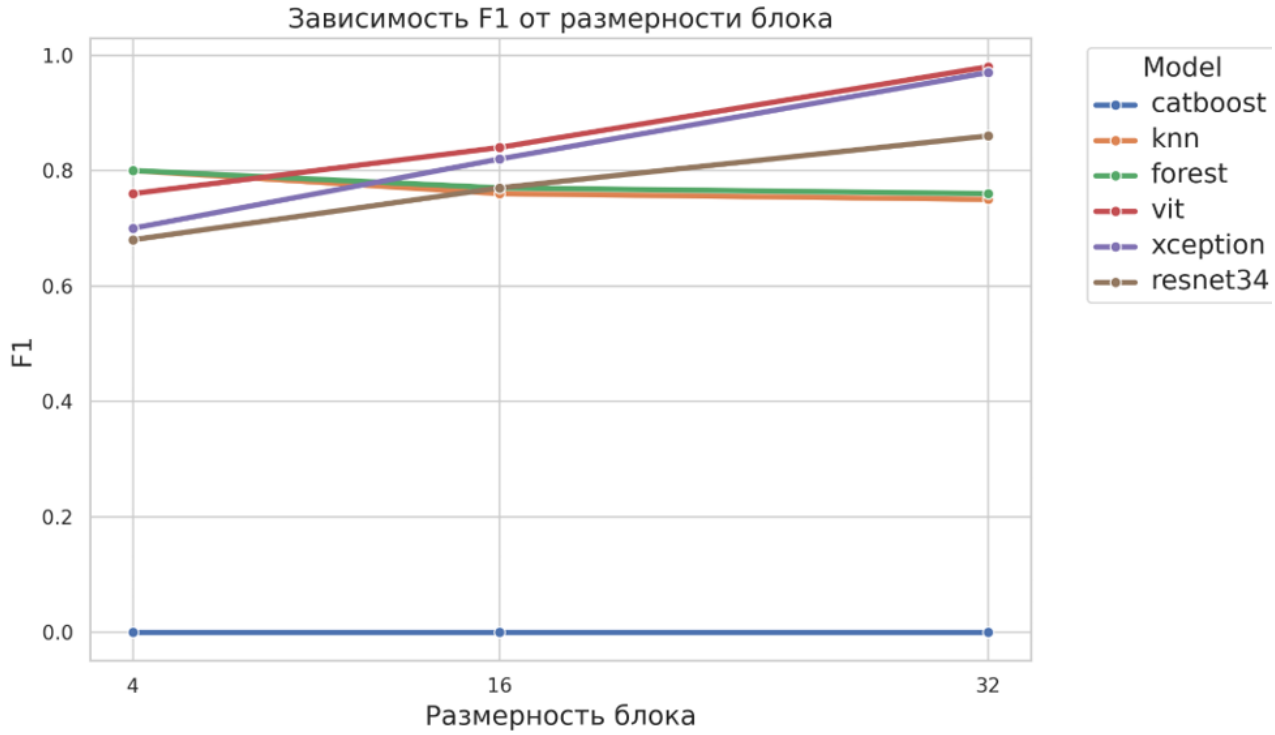

Рисунок 17 – График зависимости метрики F1-score от длины блока

ЗАКЛЮЧЕНИЕ

На основании полученных результатов можно отметить несколько ключевых выводов:

1. Предобработка данных при решении задачи существенно влияет на полученные результаты;
2. Модели ViT и Xception демонстрируют наилучшие результаты при использовании блоков длиной 32, достигая значений F1-score 0.98 и 0.97 соответственно;
3. При увеличении размера блоков качество классификации всех алгоритмов компьютерного зрения растет;
4. При увеличении размера блоков качество классификации всех алгоритмов классического машинного обучения, за исключением CatBoost, незначительно снижается;
5. Модель CatBoost показала неспособность к успешной классификации, независимо от размера блоков, что может быть связано с её ограниченной способностью обрабатывать временные ряды в предложенной форме;
6. Случайный лес и метод k ближайших соседей продемонстрировали приемлемую производительность (F1 Score = 0.8) на блоках длиной 4.

Таким образом, выбор архитектуры модели и размера блока существенно влияет на процесс обнаружения аномалий (в рамках данного исследования в качестве аномальных данных мультивариативного временного ряда выступали интервалы, на которых наблюдалось авроральное километровое радиоизлучение). Также, в ходе экспериментов модели глубокого обучения, такие как ViT и Xception, продемонстрировали наиболее стабильную и высокую производительность. В дальнейшем следует изучить возможность применения более современных архитектур для решения задачи обнаружения аномалий с помощью классификации в мультивариативных временных рядах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Zhang, Y. Human activity recognition based on time series analysis using U-Net / Y. Zhang, Z. Zhang, J. Bao, Y. Song // arXiv preprint arXiv:1809.08113. — 2018.

2 Schmidl, S. Anomaly detection in time series: a comprehensive evaluation / P. Wenig, T. Papenbrock // Proceedings of the VLDB Endowment. — 2022. — T. 15. — С. 1779-1797.

3 Cheng, H. Detection and characterization of anomalies in multivariate time series / P. N. Tan, C. Potter, S. Klooster // Proceedings of the 2009 SIAM international conference on data mining. — 2009. — С. 413-424.

4 Gamboa, J. C. B. Deep learning for time-series analysis [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/1701.01887 (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

5 Nussbaumer, H. J. The fast Fourier transform / H. J. Nussbaumer // Springer Berlin Heidelberg. — 1982. — С. 80-111.

6 Nason, G. P. Wavelet methods for time series analysis [Электронный ресурс] : [статья] / URL https://www.researchgate.net/publication/5068418_Wavelets_-in_Time_Series_Analysis (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

7 Ye, L. Time Series Shapelets: A New Primitive for Data Mining [Электронный ресурс] : [статья] / URL https:// www.cs.ucr.edu/ẽamonn/shaplet.pdf (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

8 Tsay, R. S. Outliers in multivariate time series. / D. Pena, A. E. Pankratz // Biometrika. — 2000. — T. 87. — С. 789-804.

9 Oppenheim, A. V. Speech spectrograms using the fast Fourier transform // IEEE spectrum. — 1970. — T. 7. — С. 57-62.

10 Nisar, S. An efficient adaptive window size selection method for improving spectrogram visualization. / O. U. Khan, M. Tariq // Computational intelligence and neuroscience. — 2016.

11 Mateo, C. Short-time Fourier transform with the window size fixed in the frequency domain. / J. A. Talavera // Digital Signal Processing. — 2018. — T. 77. — C. 13-21.

12 Allen, J. Short term spectral analysis, synthesis, and modification by discrete Fourier transform. // IEEE transactions on acoustics, speech, and signal processing. — 1977. — T. 25. — C. 235-238.

13 Roy, T. K. Performance analysis of low pass FIR filters design using Kaiser, Gaussian and Tukey window function methods. / M. Morshed // In 2013 2nd international conference on advances in electrical engineering. — 2013. — C. 1-6.

14 Borges, L. F. Multifaceted DDoS Attack Prediction by Multivariate Time Series and Ordinal Patterns [Электронный ресурс] : [статья] / URL https://www.researchgate.net/publication/382523397_Multifaceted_DDoS_-Attack_Prediction_by_Multivariate_Time_Series_and_Ordinal_Patterns (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

15 Tu, F. STFT-TCAN: A TCN-attention based multivariate time series anomaly detection architecture with time-frequency analysis for cyber-industrial systems [Электронный ресурс] : [статья] / URL https://www.sciencedirect.com/science/article/pii/S0167404824002669 (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

16 Liu, Y. New machine learning algorithm: Random forest. / Y. Wang, J. Zhang // In Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, September 14-16. — 2012. — C. 246-252.

17 Prokhorenkova, L. CatBoost: unbiased boosting with categorical features. / G. Gusev, A. Vorobev, A. V. Dorogush, A. Gulin // Advances in neural information processing systems. — 2018.

18 Guo, G. KNN model-based approach in classification. / H. Wang, D. Bell, Y. Bi, K. Greer // In On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy — 2003. — C. 986-996.

19 Zhuang, Q. Human-computer interaction based health diagnostics using ResNet34 for tongue image classification. / S. Gan, L. Zhang // Computer Methods and Programs in Biomedicine. — 2022. — T. 226.

20 Wu, X. An xception based convolutional neural network for scene image classification with transfer learning / R. Liu, H. Yang, Z. Chen // In 2020 2nd international conference on information technology and computer application. — 2020. — C. 262-267.

21 Han, K. A survey on vision transformer. / Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, D. Tao // IEEE transactions on pattern analysis and machine intelligence. — 2022. — T. 45. — C. 87-110.

22 Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/2010.11929 (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

23 Flach P. Performance evaluation in machine learning: the good, the bad, the ugly, and the way forward // Proceedings of the AAAI conference on artificial intelligence. – 2019. – Т. 33. – №. 01. – С. 9808-9814.

24 Düntsch I. Confusion matrices and rough set data analysis / G. Gediga // Journal of Physics: Conference Series. – IOP Publishing, 2019. – Т. 1229. – №. 1. – С. 012055.

25 Kurtz, M. J. The NASA astrophysics data system: Overview. / G. Eichhorn, A. Accomazzi, C. S. Grant, S. S. Murray, J. M. Watson // Astronomy and astrophysics supplement series. — 2000. — T. 143. — C. 41-59.

26 DeVito, Z. Using Python for model inference in deep learning, [Электронный ресурс] : [статья] / URL https:// arxiv.org/pdf/2104.00254.pdf (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

27 Pandas [Электронный ресурс] : [сайт] / URL: https:// pandas.pydata.org/ (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

28 SpacePy [Электронный ресурс] : [сайт] / URL: https:// spacepy.github.io/ (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

29 NumPy. The fundamental package for scientific computing with Python [Электронный ресурс] : [сайт] / URL: https:// numpy.org/ (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

30 scikit-learn [Электронный ресурс] : [сайт] / URL: https:// scikit-learn.org/stable/ (дата обращения 18.12.2024) — Загл. с экрана.

31 PyTS. A Python Package for Time Series Classification [Электронный ресурс] : [сайт] / URL: https:// pyts.readthedocs.io/en/stable/ (дата обращения 18.12.2024) — Загл. с экрана.

32 CatBoost [Электронный ресурс] : [сайт] / URL: https:// catboost.ai/ (дата обращения 18.12.2024) — Загл. с экрана.

33 pickle — Python object serialization [Электронный ресурс] : [сайт] / URL: https:// docs.python.org/3/library/pickle.html (дата обращения 18.12.2024) — Загл. с экрана.

34 SciPy [Электронный ресурс] : [сайт] / URL: https:// scipy.org/ (дата обращения 18.12.2024) — Загл. с экрана.

35 Matplotlib: Visualization with Python [Электронный ресурс] : [сайт] / URL: https:// matplotlib.org/ (дата обращения 18.12.2024) — Загл. с экрана.

36 seaborn: statistical data visualization [Электронный ресурс] : [сайт] / URL: https:// seaborn.pydata.org/ (дата обращения 18.12.2024) — Загл. с экрана.

37 PyTorch [Электронный ресурс] : [сайт] / URL: https:// pytorch.org/ (дата обращения 18.12.2024) Загл. с экрана. Яз. англ.

38 torchvision [Электронный ресурс] : [сайт] / URL: https:// pytorch.org/vision/stable/index.html (дата обращения 18.12.2024) — Загл. с экрана.

39 Pytorch Image Models (timm) [Электронный ресурс] : [сайт] / URL: https:// timm.fast.ai/ (дата обращения 18.12.2024) — Загл. с экрана.

40 Transformers [Электронный ресурс] : [сайт] / URL: https:// huggingface.co/docs/transformers/en/index (дата обращения 18.12.2024) — Загл. с экрана.

ПРИЛОЖЕНИЕ А

Листинг preprocessing.py

```
1  from spacepy import pycdf
2  import datetime
3  import pandas as pd
4  from matplotlib import pyplot as plt
5  import numpy as np
6
7
8  def get_cdf_list():
9      """
10         Функция для считывания спутниковых данных в Python-объект
11     """
12     cdf1 = pycdf.CDF('data/wi_h1_wav_20200701_v01.cdf')
13     cdf2 = pycdf.CDF('data/wi_h1_wav_20200702_v01.cdf')
14     cdf3 = pycdf.CDF('data/wi_h1_wav_20200703_v01.cdf')
15     cdf4 = pycdf.CDF('data/wi_h1_wav_20200704_v01.cdf')
16     cdf5 = pycdf.CDF('data/wi_h1_wav_20200705_v01.cdf')
17     cdf6 = pycdf.CDF('data/wi_h1_wav_20200706_v01.cdf')
18     cdf7 = pycdf.CDF('data/wi_h1_wav_20200707_v01.cdf')
19     cdf8 = pycdf.CDF('data/wi_h1_wav_20200708_v01.cdf')
20     cdf9 = pycdf.CDF('data/wi_h1_wav_20200709_v01.cdf')
21     cdf10 = pycdf.CDF('data/wi_h1_wav_20200710_v01.cdf')
22     cdf11 = pycdf.CDF('data/wi_h1_wav_20200711_v01.cdf')
23     cdf12 = pycdf.CDF('data/wi_h1_wav_20200712_v01.cdf')
24     cdf13 = pycdf.CDF('data/wi_h1_wav_20200713_v01.cdf')
25     cdf_list = [cdf1, cdf2, cdf3, cdf4, cdf5, cdf6, cdf7, cdf8, cdf9, cdf10,
26                 cdf11, cdf12, cdf13]
27     return cdf_list
28
29
30 def create_AKR_df():
31     """
32         Функция для создания датафрейма с размеченными вручную отрезками
33         времени, которые соответствуют всплескам АКР
34     """
35     detected_akr = pd.read_excel('data/WINDacr2020.xlsx')
36     AKR_intervals = {}
37
38     upper_bound = detected_akr[["конец", "Unnamed: 8", "Unnamed: 9",
39                                 "Unnamed: 10", "Unnamed: 11", "Unnamed: 12"
```

```
40                                  ]
41                                 ][1:]
42      timestamp = []
43      for j in range(upper_bound.shape[0]):
44          i = j + 1
45          timestamp.append(f'{upper_bound["конец"][i]}-{upper_bound["Unnamed:
            ↪ 8"][i]}-{upper_bound["Unnamed: 9"][i]} {upper_bound["Unnamed:
            ↪ 10"][i]}:{upper_bound["Unnamed: 11"][i]}:{upper_bound["Unnamed:
            ↪ 12"][i]}')
46      upper_bound["timestamp"] = [datetime.datetime.strptime(datestamp,
        ↪ "%Y-%m-%d %H:%M:%S") for datestamp in timestamp]
47
48      lower_bound = detected_akr[["начало", "Unnamed: 1", "Unnamed: 2",
        ↪ "Unnamed: 3", "Unnamed: 4", "Unnamed: 5"]][1:]
49      timestamp = []
50      for j in range(upper_bound.shape[0]):
51          i = j + 1
52          timestamp.append(str(lower_bound["начало"][i]) + "-" +
            ↪ str(lower_bound["Unnamed: 1"][i]) + "-" +
            ↪ str(lower_bound["Unnamed: 2"][i]) + " " +
            ↪ str(lower_bound["Unnamed: 3"][i]) + ":" +
            ↪ str(lower_bound["Unnamed: 4"][i]) + ":" +
            ↪ str(lower_bound["Unnamed: 5"][i]))
53      lower_bound["timestamp"] = [datetime.datetime.strptime(datestamp,
        ↪ "%Y-%m-%d %H:%M:%S") for datestamp in timestamp]
54
55      AKR_intervals["begin"] = lower_bound["timestamp"]
56      AKR_intervals["end"] = upper_bound["timestamp"]
57      AKR_intervals = pd.DataFrame(AKR_intervals)
58      return AKR_intervals
59
60
61  def get_total_df(cdf_list, AKR_intervals):
62      """
63          Создание общего датафрейма мультивариативного ряда, где каждый ряд
64          соответствует показателям напряжения в единицу времени на конкретной
65          частоте
66      """
67      cdf_df_list = []
68
69      for cdf in cdf_list:
```

```
70          cdf_dict = {}
71          cdf_dict["Epoch"] = cdf["Epoch"]
72
73          for freq_j in range(cdf["Frequency_RAD1"].shape[0]):
74              voltage_list = cdf["E_VOLTAGE_RAD1"][:, freq_j]
75              cdf_dict[cdf["Frequency_RAD1"][freq_j]] = voltage_list
76
77          cdf_df = pd.DataFrame(cdf_dict)
78          cdf_df = cdf_df.set_index('Epoch')
79
80          # создание столбца с АКР
81          is_AKR_list = []
82          for i in range(cdf_df.shape[0]):
83              cur_time = cdf_df.index[i]
84              is_in_range = None
85              for cur_interval in AKR_intervals.values:
86                  begin, end = cur_interval
87                  if begin <= cur_time <= end:
88                      is_in_range = True
89                      break
90              if is_in_range is None:
91                  is_in_range = False
92              is_AKR_list.append(is_in_range)
93
94          cdf_df["is_AKR"] = is_AKR_list
95          cdf_df_list.append(cdf_df)
96
97      final_df = cdf_df_list[0]
98      for cdf_df_i in range(1, len(cdf_df_list)):
99          final_df = final_df.append(cdf_df_list[cdf_df_i])
100     return final_df
101
102
103 ##############################################################################
104
105 def split_into_blocks_by_class(df):
106     """
107         Функция делит временные ряды на блоки, где каждый блок соответствует
108         всплеску АКР или его отсутствию. Блоки не равномерные.
109     """
110     block_number = 0
```

```
111     block_number_list = [block_number]
112     for i in range(1, df.shape[0]):
113         cur_value = df["is_AKR"][i]
114         prev_value = df["is_AKR"][i - 1]
115         if prev_value == cur_value:
116             block_number_list.append(block_number)
117         else:
118             block_number += 1
119             block_number_list.append(block_number)
120     df["block"] = block_number_list
121     return df
122
123
124 ##############################################################################
125
126 def main():
127     # step 1
128     # получение и первичная предобработка спутниковых данных
129     cdf_list = get_cdf_list()
130     AKR_df = create_AKR_df()
131     df = get_total_df(cdf_list, AKR_df)
132
133     # step 2
134     # обработка спутниковых данных для задачи классификации по блокам
135     df = split_into_blocks_by_class(df)
136     df.to_csv("processed_data.csv")
137
138
139 if __name__ == "__main__":
140     main()
```

ПРИЛОЖЕНИЕ Б

Листинг processing.py

```
import pandas as pd
import numpy as np
import pickle


def adjust_blocks_to_length(df, target_length, step=1):
    """
    Приводит блоки в датафрейме к указанной длине. Блоки с длиной меньше удаляются,
    а блоки с длиной больше формируются с помощью скользящего окна с заданным шагом.

    Параметры:
    - df (pd.DataFrame): Исходный датафрейм с блоками.
    - target_length (int): Желаемая длина блоков.
    - step (int): Шаг скользящего окна. По умолчанию 1.

    Возвращает:
    - pd.DataFrame: Новый датафрейм с блоками указанной длины.
    """
    # Список для хранения новых блоков
    new_blocks = []
    new_block_id = 0

    # Группируем по столбцу 'block'
    grouped = df.groupby('block')

    for block_id, group in grouped:
        # Если длина блока меньше целевой, пропускаем его
        if len(group) < target_length:
            continue

        # Если длина блока равна целевой, добавляем его как есть
        elif len(group) == target_length:
            group['block'] = new_block_id
            new_blocks.append(group)
            new_block_id += 1

        # Если длина блока больше целевой, применяем скользящее окно
```

```
38          else:
39              for start in range(0, len(group) - target_length + 1, step):
40                  # Формируем новое окно
41                  new_block = group.iloc[start:start + target_length].copy()
42                  new_block['block'] = new_block_id  # Присваиваем новый ID
                    ↪ блока
43                  new_blocks.append(new_block)
44                  new_block_id += 1
45
46      # Объединяем все блоки обратно в один DataFrame
47      if new_blocks:
48          new_df = pd.concat(new_blocks)
49      else:
50          new_df = pd.DataFrame(columns=df.columns)  # Пустой DataFrame, если
            ↪ нет подходящих блоков
51
52      return new_df
53
54
55  def cleared_test_ds(X_dict, y, threshold=1):
56      """
57          Функция, которая удаляет из тренировочной выборки последовательности,
    ↪ в
58          которых значение напряжения равно нулю больше чем threshold секунд
59      """
60      new_X_dict = {}
61      bad_idx = []
62      # находим для каждой частоты индексы с данными, в которых больше одного 0
63      freqs = list(X_dict.keys())
64      for freq in freqs:
65          cur_freq_test = X_dict[freq]
66          for i in range(len(cur_freq_test)):
67              if cur_freq_test[i].count(0) > threshold:
68                  bad_idx.append(i)
69      bad_idx = np.unique(np.array(bad_idx)).tolist()
70      new_y = pd.DataFrame(y).drop(bad_idx, axis=0).values.tolist()
71      new_y = [y[0] for y in new_y]
72      # удаляем для каждой частоты объекты с этими индексами (в т.ч. в 'y')
73      for freq in freqs:
74          cur_freq_test = X_dict[freq]
75          new_X = pd.DataFrame(cur_freq_test).drop(bad_idx,
            ↪ axis=0).values.tolist()
```

```
76          new_X_dict[freq] = new_X
77      return new_X_dict, new_y
78
79
80  def balance_blocks_by_is_AKR(df):
81      """
82      Сбалансировать количество блоков по значению столбца `is_AKR`.
83      Уравнивает количество блоков с `is_AKR = True` и `is_AKR = False` путем
    ↪   рандомного удаления избыточных блоков.
84
85      Параметры:
86      - df (pd.DataFrame): Датафрейм, содержащий столбцы `block` и `is_AKR`.
87
88      Возвращает:
89      - pd.DataFrame: Сбалансированный датафрейм с равным количеством блоков
    ↪   `is_AKR = True` и `is_AKR = False`.
90      """
91      # Разделение на группы по значению `is_AKR`
92      true_blocks = df[df["is_AKR"] == True]
93      false_blocks = df[df["is_AKR"] == False]
94
95      # Определение минимального количества блоков
96      min_count = min(len(true_blocks), len(false_blocks))
97
98      # Если количество блоков уже сбалансировано, ничего не делаем
99      if len(true_blocks) == len(false_blocks):
100         return df
101
102     # Если блоков с is_AKR=True больше или меньше, делаем undersampling
103     true_blocks_balanced = true_blocks.sample(n=min_count, random_state=42)
104     false_blocks_balanced = false_blocks.sample(n=min_count, random_state=42)
105
106     # Объединяем сбалансированные блоки
107     balanced_df = pd.concat([true_blocks_balanced,
        ↪ false_blocks_balanced]).reset_index(drop=True)
108
109     return balanced_df
110
111
112 def main():
113     df = pd.read_csv("processed_data.csv", index_col=0)
```

```
114     ethalon_block_sizes = [4, 16, 32]
115     for bs in ethalon_block_sizes:
116         df_bs = adjust_blocks_to_length(df, bs)
117         df_bs.to_csv(f"windowed_df_{bs}.csv")
118         freqs = df_bs.columns[:-2]
119 
120         dfs_for_freqs = []
121         for cur_freq in freqs:
122             kekw = df_bs[[cur_freq, "is_AKR", "block"]]
123 
124             grouped = kekw.groupby("block").apply(lambda x: pd.Series({
125                 cur_freq: x[cur_freq].tolist(),
126                 "is_AKR": x["is_AKR"].iloc[0],
127                 "block": x["block"].iloc[0]
128             })).reset_index(drop=True)
129 
130             grouped[cur_freq] = grouped[cur_freq].apply(lambda x: x)
131             dfs_for_freqs.append(grouped.copy())
132 
133         balanced_dfs_for_freqs = [balance_blocks_by_is_AKR(df) for df in
        ↪ dfs_for_freqs]
134         with open(f'balanced_dfs_for_freqs_{bs}.pkl', 'wb') as file:
135             pickle.dump(balanced_dfs_for_freqs, file)
136 
137 
138 if __name__ == "__main__":
139     main()
```

ПРИЛОЖЕНИЕ В

Листинг spectrogramms.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import spectrogram
import os
import shutil


def make_spectrogramms(df, block_size, window_length, overlap):
    spectrogram_info = []
    time_series = df.iloc[:, 1:-2].to_numpy()
    for start_idx in range(0, len(time_series), block_size):
        cur_time_series = time_series[start_idx:start_idx + block_size]

        Sxx_accumulated = None

        for i in range(cur_time_series.shape[1]):
            f, t, Sxx = spectrogram(cur_time_series[:, i],
             ↪  nperseg=window_length, noverlap=overlap)

            if Sxx_accumulated is None:
                Sxx_accumulated = np.zeros_like(Sxx)

            Sxx_accumulated += Sxx

        Sxx_mean = Sxx_accumulated / cur_time_series.shape[1]

        if np.all(Sxx_mean <= 1e-10):
            print(f"Warning: Low signal power at index {start_idx}")
            continue

        plt.figure(figsize=(10, 6))
        plt.pcolormesh(t, f, 10 * np.log10(Sxx_mean + 1e-10),
         ↪  shading='gouraud')

        plt.gca().set_xticks([])
        plt.gca().set_yticks([])
        plt.colorbar().remove()

```

```
38          # Сохранение изображения без полей
39          filename = f"spectrogram_{start_idx}_{block_size}.png"
40          plt.savefig(filename, bbox_inches='tight', pad_inches=0)
41          plt.close()
42          spectrogram_info.append({'filename': filename, 'start_time':
            ↪ start_idx})
43
44      # Создание DataFrame из накопленной информации
45      spectrogram_df = pd.DataFrame(spectrogram_info)
46      # Сохранение DataFrame в CSV файл
47      spectrogram_df.to_csv(f'spectrogram_info_{block_size}.csv', index=False)
48      print(f"Спектрограммы созданы и информация сохранена в
        ↪ 'spectrogram_info_{block_size}.csv'.")
49      filename = np.array([[filename] * block_size for filename in
        ↪ spectrogram_df["filename"]]).flatten()
50      df["filename"] = filename.tolist() + [None] * (df.shape[0] -
        ↪ len(filename))
51      cv_df = df[["filename", "is_AKR"]].dropna()
52      cv_df.to_csv(f"cv_dataset_{block_size}.csv", index=False)
53      df = cv_df.drop_duplicates()
54
55      # создание директорий со спектрограммами
56      base_dir = f'cv_classification_{block_size}'
57      akr_dir = os.path.join(base_dir, 'AKR')
58      not_akr_dir = os.path.join(base_dir, 'not_AKR')
59
60      os.makedirs(akr_dir, exist_ok=True)
61      os.makedirs(not_akr_dir, exist_ok=True)
62
63      # Копирование файлов в соответствующие директории
64      for _, row in df.iterrows():
65          src_path = row['filename']
66          dest_dir = akr_dir if row['is_AKR'] else not_akr_dir
67          shutil.copy(src_path, dest_dir)
68
69
70  def main():
71      df = pd.read_csv(f"windowed_df_{4}.csv")
72      make_spectrogramms(df, 4, 2, 1)
73      df = pd.read_csv(f"windowed_df_{16}.csv")
74      make_spectrogramms(df, 16, 4, 2)
```

```
75      df = pd.read_csv(f"windowed_df_{32}.csv")
76      make_spectrogramms(df, 32, 4, 2)
77
78
79  if __name__ == "__main__":
80      main()
```

ПРИЛОЖЕНИЕ Г

Листинг classic.py

```
from pyts.classification import TimeSeriesForest
from catboost import CatBoostClassifier
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.metrics import precision_score, recall_score, confusion_matrix,
↪ f1_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV


windowed_df = pd.read_csv("windowed_df.csv", index_col=0)
freqs = windowed_df.columns[:-2]


def cleared_train_ds(X, y, threshold=1):
    """
        Функция, которая удаляет из обучающей выборки последовательности, в
        которых значение напряжения равно нулю больше чем threshold секунд
    """
    cleared_X = []
    cleared_y = []
    for i in range(len(X)):
        if not (X[i].count(0) > threshold):
            cleared_X.append(X[i])
            cleared_y.append(y[i])
    return cleared_X, cleared_y


def split_data_and_train_ensembles_catboost(dfs, threshold=1):
    # Используем первый DataFrame для инициализации
    cur_df = dfs[0]
    X = cur_df["20"]
    y = cur_df["is_AKR"].apply(int).to_list()

    indexed_y = list(zip(y, np.arange(len(y)).tolist()))
```

```
39      _, _, y_train, y_test = train_test_split(X, indexed_y, test_size=0.2,
        ↪ random_state=42)
40      i_train = [i for _, i in y_train]
41      i_test = [i for _, i in y_test]
42      y_train = [y for y, _ in y_train]
43      y_test = [y for y, _ in y_test]
44
45      freq_clf_list = {}
46      X_test_df = {}
47
48      # Параметры для поиска
49      param_grid = {
50          'iterations': [1, 2, 3, 4, 5],
51          'learning_rate': [0.01, 0.1, 1],
52          'depth': [1, 2, 3, 4, 5]
53      }
54
55      for i in range(len(dfs)):
56          cur_df = dfs[i]
57          freq = cur_df.columns[0]  # Получаем частоту
58          freq_X = cur_df[freq]
59          df_X = pd.DataFrame(np.array(freq_X))
60
61          # Формирование обучающей и тестовой выборки
62          train_freq_X = df_X.iloc[i_train]
63          train_freq_X = [train_freq_X.iloc[j].tolist() for j in
            ↪ range(train_freq_X.shape[0])]
64
65          test_freq_X = df_X.iloc[i_test]
66          test_freq_X = [test_freq_X.iloc[j].tolist() for j in
            ↪ range(test_freq_X.shape[0])]
67
68          X_test_df[freq] = test_freq_X
69
70          # Очистка данных
71          train_freq_X, freq_y_train = cleared_train_ds(train_freq_X, y_train,
            ↪ threshold=threshold)
72
73          # Преобразование данных в строки
74          train_freq_X = [[str(cur_num) for cur_num in cur_list] for cur_list in
            ↪ train_freq_X]
```

```
75
76          # Определение категориальных признаков в зависимости от размера данных
77          if len(train_freq_X[0]) > 1:
78              cat_features = list(range(len(train_freq_X[0])))  # Индексы всех
                ↪ столбцов
79          else:
80              cat_features = [0]  # Единственный столбец
81
82          model = CatBoostClassifier(cat_features=cat_features)
83          grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
            ↪ cv=3, scoring='accuracy')
84          grid_search.fit(train_freq_X, freq_y_train)
85          best_model = grid_search.best_estimator_
86          freq_clf_list[freq] = best_model
87
88      return X_test_df, freq_clf_list, y_train, y_test
89
90
91  def split_data_and_train_ensembles_classic(dfs, threshold=1, algo="forest"):
92      # Используем первый DataFrame для инициализации
93      cur_df = dfs[0]
94      X = cur_df["20"]
95      y = cur_df["is_AKR"].apply(int).to_list()
96
97      indexed_y = list(zip(y, np.arange(len(y)).tolist()))
98      _, _, y_train, y_test = train_test_split(X, indexed_y, test_size=0.2,
        ↪ random_state=42)
99      i_train = [i for _, i in y_train]
100     i_test = [i for _, i in y_test]
101     y_train = [y for y, _ in y_train]
102     y_test = [y for y, _ in y_test]
103
104     freq_clf_list = {}
105     X_test_df = {}
106
107     for i in range(len(dfs)):
108         cur_df = dfs[i]
109         freq = cur_df.columns[0]  # Получаем частоту
110         freq_X = cur_df[freq]
111         df_X = pd.DataFrame(np.array(freq_X))
112
```

```
113         # Формирование обучающей и тестовой выборки
114         train_freq_X = df_X.iloc[i_train]
115         train_freq_X = [train_freq_X.iloc[j].tolist() for j in
            ↪ range(train_freq_X.shape[0])]
116
117         test_freq_X = df_X.iloc[i_test]
118         test_freq_X = [test_freq_X.iloc[j].tolist() for j in
            ↪ range(test_freq_X.shape[0])]
119
120         X_test_df[freq] = test_freq_X
121         train_freq_X = [underlist[0] for underlist in train_freq_X]
122         train_freq_X, freq_y_train = cleared_train_ds(train_freq_X, y_train,
            ↪ threshold=threshold)
123         if algo == "forest":
124             cur_clf = TimeSeriesForest(500, random_state=43)
125         else:
126             cur_clf = KNeighborsClassifier()
127
128         cur_clf.fit(train_freq_X, freq_y_train)
129         freq_clf_list[freq] = cur_clf
130
131     return X_test_df, freq_clf_list, y_train, y_test
132
133
134 def eval_test(X_test_df, forest_freq_clf_list, is_catboost=True):
135     X_test_df = pd.DataFrame(X_test_df)
136     y_pred = []
137     for i in range(X_test_df.shape[0]):
138         y_cur_pred = []
139         for freq in freqs:
140             cur_freq_X = X_test_df.iloc[i][freq]
141             if is_catboost:
142                 cur_freq_X = [str(cur_elem) for cur_elem in cur_freq_X]
143             cur_freq_y = forest_freq_clf_list[freq].predict([cur_freq_X])[0]
144             y_cur_pred.append(cur_freq_y)
145         pred_mode = stats.mode(np.array(y_cur_pred))[0]
146         y_pred.append(pred_mode)
147     return y_pred
148
149
150 def get_data_stats(train, test):
```

```
151     print("(train) АКР:", train.count(1))
152     print("(train) не АКР:", train.count(0))
153 
154     print("(test) АКР:", test.count(1))
155     print("(test) не АКР:", test.count(0))
156 
157 
158 def build_conf_matrix(labels, predict, class_name):
159     lab, pred = [], []
160     for i in range(len(labels)):
161         if predict[i] == class_name:
162             pred.append(0)
163         else:
164             pred.append(1)
165         if labels[i] == class_name:
166             lab.append(0)
167         else:
168             lab.append(1)
169     return confusion_matrix(lab, pred, normalize='true')
170 
171 
172 def eval_metrics(y_pred, y_test, id=""):
173     print("f1_score: ", f1_score(y_test, y_pred))
174     print("precision_score: ", precision_score(y_test, y_pred))
175     print("recall_score: ", recall_score(y_test, y_pred))
176 
177     get_res = {0: "нет АКР", 1: "АКР"}
178 
179     for i in range(2):
180         heatmap = sns.heatmap(build_conf_matrix(y_test, y_pred, i),
            ↪  annot=True, cmap='YlGnBu')
181         heatmap.set_title(get_res[i], fontdict={'fontsize':14}, pad=10)
182         plt.xlabel('Предсказанный класс')
183         plt.ylabel('Истинный класс')
184         plt.savefig(f"Класс {i} ({id}).png", dpi=300)
185         plt.show()
186 
187 
188 def cleared_test_ds(X_dict, y, threshold=1):
189     """
190         Функция, которая удаляет из тренировочной выборки последовательности,
    ↪  в
```

```
191         которых значение напряжения равно нулю больше чем threshold секунд
192     """
193     new_X_dict = {}
194     bad_idx = []
195     # находим для каждой частоты индексы с данными, в которых больше одного 0
196     freqs = list(X_dict.keys())
197     for freq in freqs:
198         cur_freq_test = X_dict[freq]
199         for i in range(len(cur_freq_test)):
200             if cur_freq_test[i].count(0) > threshold:
201                 bad_idx.append(i)
202     bad_idx = np.unique(np.array(bad_idx)).tolist()
203     new_y = pd.DataFrame(y).drop(bad_idx, axis=0).values.tolist()
204     new_y = [y[0] for y in new_y]
205     # удаляем для каждой частоты объекты с этими индексами (в т.ч. в 'y')
206     for freq in freqs:
207         cur_freq_test = X_dict[freq]
208         new_X = pd.DataFrame(cur_freq_test).drop(bad_idx,
            ↪ axis=0).values.tolist()
209         new_X_dict[freq] = new_X
210     return new_X_dict, new_y
211
212
213 def make_results(X_test_df, freq_clf_list, y_train, y_test):
214     get_data_stats(y_train, y_test)
215     new_X_test_df, new_y_test = cleared_test_ds(X_test_df, y_test)
216     y_pred = eval_test(new_X_test_df, freq_clf_list)
217     print(confusion_matrix(new_y_test, y_pred, normalize='true'))
218     eval_metrics(y_pred, new_y_test)
219
220
221 def main():
222     ethalon_block_sizes = [4, 16, 32]
223     for bs in ethalon_block_sizes:
224         with open(f'balanced_dfs_for_freqs_{bs}.pkl', 'rb') as file:
225             balanced_dfs_for_freqs = pickle.load(file)
226
227         print("CatBoost:")
228         X_test_df, freq_clf_list, y_train, y_test =
            ↪ split_data_and_train_ensembles_catboost(balanced_dfs_for_freqs)
229         make_results(X_test_df, freq_clf_list, y_train, y_test)
```

```
230
231         print("RandomForest:")
232         X_test_df, freq_clf_list, y_train, y_test =
            ↪ split_data_and_train_ensembles_classic(balanced_dfs_for_freqs,
            ↪ algo="forest")
233         make_results(X_test_df, freq_clf_list, y_train, y_test)
234
235         print("kNN:")
236         X_test_df, freq_clf_list, y_train, y_test =
            ↪ split_data_and_train_ensembles_classic(balanced_dfs_for_freqs,
            ↪ algo="knn")
237         make_results(X_test_df, freq_clf_list, y_train, y_test)
238
239
240 if __name__ == "__main__":
241     main()
```

ПРИЛОЖЕНИЕ Д

Листинг cv.py

```
import os
from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm
from transformers import ViTForImageClassification, AdamW, get_scheduler
import torch
from torch import nn
from torch import optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
from transformers import ViTFeatureExtractor
import PIL
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    ↪ confusion_matrix, f1_score
import numpy as np
import timm
import pickle


def build_conf_matrix(labels, predict, class_name):
    lab, pred = [], []
    for i in range(len(labels)):
        if predict[i] == class_name:
            pred.append(0)
        else:
            pred.append(1)
        if labels[i] == class_name:
            lab.append(0)
        else:
            lab.append(1)
    return confusion_matrix(lab, pred, normalize='true')


def vit_experiment(data_dir):
    classes = os.listdir(data_dir)
    feature_extractor =
        ↪ ViTFeatureExtractor.from_pretrained("google/vit-base-patch16-384")
    imgs = []
    lbls = []
    cnt = 0
```

```
38      class2lbl = {"AKR": 1, "not_AKR": 0}
39      for cl in classes:
40          class_path = data_dir + '/' + cl
41          images = os.listdir(class_path)
42          for image in images:
43              try:
44                  input = PIL.Image.open(class_path + '/' +
                    ↪  image).convert("RGB")
45                  imgs.append(feature_extractor(input,
                    ↪  return_tensors="pt")['pixel_values'][0])
46                  lbls.append(class2lbl[cl])
47              except:
48                  cnt += 1
49
50      train_images, val_images, train_labels, val_labels =
        ↪  train_test_split(imgs, lbls, test_size=.2, stratify=lbls)
51
52      class AKRDataset(torch.utils.data.Dataset):
53          def __init__(self, images, labels):
54              self.images = images
55              self.labels = labels
56
57          def __getitem__(self, idx):
58              item = {}
59              item['inputs'] = self.images[idx]
60              item['labels'] = torch.tensor(self.labels[idx])
61              return item
62
63          def __len__(self):
64              return len(self.labels)
65
66      train_data = AKRDataset(train_images, train_labels)
67      val_data = AKRDataset(val_images, val_labels)
68      train_loader = DataLoader(train_data, batch_size=8, shuffle=True)
69      val_loader = DataLoader(val_data, batch_size=8, shuffle=True)
70
71
72      if torch.cuda.is_available():
73          device = torch.device("cuda")
74          print("Running on the GPU")
75      else:
```

```
76          device = torch.device("cpu")
77          print("Running on the CPU")
78
79      model =
        ↪ ViTForImageClassification.from_pretrained("google/vit-base-patch16-384",
        ↪ num_labels=2, ignore_mismatched_sizes=True)
80
81      model.to(device)
82      model.train()
83      optim = AdamW(model.parameters(), lr=5e-5)
84      num_epoch = 5
85
86      losses_for_train = []
87      losses_for_val = []
88      num_training_steps = num_epoch * len(train_loader)
89      lr_scheduler = get_scheduler('linear',
90                                   optimizer=optim,
91                                   num_warmup_steps=0,
92                                   num_training_steps=num_training_steps)
93
94      for epoch in range(num_epoch):
95          loop = tqdm(train_loader, leave=True)
96          model.train()
97          for batch in loop:
98              inputs = batch['inputs'].to(device)
99              labels = batch['labels'].to(device)
100             outputs = model(inputs, labels=labels)
101             loss = outputs[0]
102             loss.backward()
103             loop.set_description(f'Epoch {epoch}')
104             loop.set_postfix(loss=loss.item())
105             optim.step()
106             losses_for_train.append(loss.item())
107             lr_scheduler.step()
108             optim.zero_grad()
109         model.eval()
110
111         for batch in val_loader:
112             inputs = batch['inputs'].to(device)
113             labels = batch['labels'].to(device)
114             with torch.no_grad():
```

```
115                 outputs = model(inputs, labels=labels)
116             loss = outputs[0]
117             losses_for_val.append(loss.item())
118
119
120     train_loss = []
121     val_loss = []
122     tl = losses_for_train[0]
123     for i in range(1, len(losses_for_train)):
124         if i % (len(train_loader) - 1):
125             tl+=losses_for_train[i]
126         else:
127             train_loss.append(tl / len(train_loader))
128             tl = 0
129     tl = losses_for_val[0]
130     for i in range(1, len(losses_for_val)):
131         if i % (len(val_loader) - 1):
132             tl+=losses_for_val[i]
133         else:
134             val_loss.append(tl / len(val_loader))
135             tl = 0
136
137
138     all_predictions = []
139     all_references = []
140     for batch in val_loader:
141         inputs = batch['inputs'].to(device)
142         labels = batch['labels'].to(device)
143         with torch.no_grad():
144             outputs = model(inputs, labels=labels)
145         logits = outputs.logits
146         predictions = torch.argmax(logits, dim= -1)
147         all_predictions.extend(predictions.cpu().numpy())
148         all_references.extend(batch['labels'].cpu().numpy())
149
150     print('accuracy: {} '.format(accuracy_score(all_predictions,
        ↪ all_references)))
151     print('precision: {} '.format(precision_score(all_predictions,
        ↪ all_references ,average='weighted')))
152     print('recall: {} '.format(recall_score(all_predictions, all_references,
        ↪ average='weighted')))
```

```
153     print('f1-score: {} '.format(f1_score(all_predictions, all_references,
        ↪ average='weighted')))
154     print(build_conf_matrix(all_references, all_predictions, 1))
155     model.save_pretrained(f"vit_{data_dir}.pth")
156
157
158 def load_split_train_test(datadir, valid_size = 0.2):
159     train_trainsforms = transforms.Compose([transforms.Resize((224,224)),
        ↪ transforms.ToTensor()])
160     train_data = datasets.ImageFolder(datadir,transform=train_trainsforms)
161
162     num_train = len(train_data)
163     indices = list(range(num_train))
164
165     split = int(np.floor(valid_size * num_train))
166     np.random.shuffle(indices)
167
168     from torch.utils.data.sampler import SubsetRandomSampler
169     train_idx, test_idx = indices[split:], indices[:split]
170     print(len(train_idx), len(test_idx))
171     train_sampler = SubsetRandomSampler(train_idx)
172     test_sampler  = SubsetRandomSampler(test_idx)
173
174     train_loader = DataLoader(train_data,sampler=train_sampler,batch_size=64)
175     test_loader = DataLoader(train_data,sampler=test_sampler,batch_size=64)
176     return train_loader,test_loader
177
178
179 def resnet34_experiment(data_dir):
180     train_loader,test_loader = load_split_train_test(data_dir, 0.2)
181
182     if torch.cuda.is_available():
183         device = torch.device("cuda")  # здесь вы можете продолжить,
            ↪ например,cuda:1 cuda:2... и т. д.
184         print("Running on the GPU")
185     else:
186         device = torch.device("cpu")
187         print("Running on the CPU")
188     model = models.resnet34(pretrained=True)
189
190     for param in model.parameters():
```

```
191         param.requires_grad = False
192 
193     model.fc = nn.Sequential(nn.Linear(512,512),
194                              nn.ReLU(),
195                              nn.Dropout(0.2),
196                              nn.Linear(512,7),
197                              nn.LogSoftmax(dim=1))
198     criterion = nn.NLLLoss()
199     optimizer = optim.Adam(model.fc.parameters(),lr=0.0001)
200     model.to(device)
201 
202     epochs = 10
203     steps = 0
204     running_loss = 0
205     train_losses, test_losses = [],[]
206 
207     for epoch in tqdm.tqdm(range(epochs)):
208         for inputs,labels in train_loader:
209             inputs,labels = inputs.to(device),labels.to(device)
210             optimizer.zero_grad()
211             out = model(inputs)
212             loss = criterion(out,labels)
213             loss.backward()
214             optimizer.step()
215             running_loss +=loss.item()
216             steps +=1
217 
218         test_loss = 0
219         accuracy = 0
220         model.eval()
221         with torch.no_grad():
222             for inputs,labels in test_loader:
223                 inputs, labels = inputs.to(device), labels.to(device)
224                 out2 = model(inputs)
225                 batch_loss = criterion(out2,labels)
226                 test_loss +=batch_loss.item()
227 
228                 ps = torch.exp(out2)
229                 top_pred, top_class = ps.topk(1,dim=1)
230                 equals = top_class == labels.view(*top_class.shape)
231                 accuracy += torch.mean(equals.type(torch.FloatTensor)).item()
```

```
232
233         train_losses.append(running_loss/len(train_loader))
234         test_losses.append(test_loss/len(test_loader))
235
236         print(f"Epoch {epoch+1}/{epochs}"
237                     f"Train loss: {running_loss/len(train_loader):.3f}",
238                     f"Test loss: {test_loss/len(test_loader):.3f} "
239                     f"Test accuracy: {accuracy/len(test_loader):.3f}")
240         running_loss = 0
241         model.train()
242
243     l = []
244     c = []
245     model.eval()
246     with torch.no_grad():
247         for inputs,labels in test_loader:
248             inputs, labels = inputs.to(device), labels.to(device)
249             out2 = model(inputs)
250
251             ps = torch.exp(out2)
252             top_pred, top_class = ps.topk(1,dim=1)
253             for i in range(len(top_class)):
254                 l.append(labels[i].item())
255                 c.append(top_class[i].item())
256
257     with open(f"resnet34_{data_dir}.pth", 'wb') as file:
258         pickle.dump(model, file)
259
260     print(build_conf_matrix(l, c, 1))
261     print('accuracy:{}'.format(accuracy_score(l, c)))
262     print('precision:{}'.format(precision_score(l, c,average='weighted')))
263     print('recall:{}'.format(recall_score(l, c,average='weighted')))
264     print('f1-score:{}'.format(f1_score(l, c,average='weighted')))
265
266
267 def xception_experiment(data_dir):
268     model = timm.create_model('xception', pretrained=True, num_classes=2)
269     model.eval()
270     train_loader, test_loader = load_split_train_test(data_dir, 0.2)
271     if torch.cuda.is_available():
272         device = torch.device("cuda")  # здесь вы можете продолжить,
          ↪ например,cuda:1 cuda:2... и т. д.
```

```
273         print("Running on the GPU")
274     else:
275         device = torch.device("cpu")
276         print("Running on the CPU")
277     for param in model.parameters():
278         param.requires_grad = True
279 
280     criterion = nn.NLLLoss()
281     optimizer = optim.Adam(model.parameters(),lr=0.0001)
282     model.to(device)
283 
284     epochs = 5
285     steps = 0
286     running_loss = 0
287     train_losses,test_losses = [],[]
288 
289 
290     for epoch in tqdm.tqdm(range(epochs)):
291         for inputs,labels in train_loader:
292             inputs,labels = inputs.to(device),labels.to(device)
293             optimizer.zero_grad()
294             out = model(inputs)
295             log_out = torch.nn.functional.log_softmax(out)
296             loss = criterion(log_out,labels)
297             loss.backward()
298             optimizer.step()
299             running_loss +=loss.item()
300             steps +=1
301 
302         test_loss = 0
303         accuracy = 0
304         model.eval()
305         with torch.no_grad():
306             for inputs,labels in test_loader:
307                 inputs, labels = inputs.to(device), labels.to(device)
308                 out2 = model(inputs)
309                 log_out2 = torch.nn.functional.log_softmax(out2)
310                 batch_loss = criterion(log_out2,labels)
311                 test_loss += batch_loss.item()
312 
313                 ps = torch.exp(out2)
```

```
314                 top_pred,top_class = ps.topk(1,dim=1)
315                 equals = top_class == labels.view(*top_class.shape)
316                 accuracy += torch.mean(equals.type(torch.FloatTensor)).item()
317 
318         train_losses.append(running_loss/len(train_loader))
319         test_losses.append(test_loss/len(test_loader))
320 
321         print(f"Epoch {epoch+1}/{epochs}"
322                     f"Train loss: {running_loss/len(train_loader):.3f}",
323                     f"Test loss: {test_loss/len(test_loader):.3f} "
324                     f"Test accuracy: {accuracy/len(test_loader):.3f}")
325         running_loss = 0
326         model.train()
327 
328     l = []
329     c = []
330     model.eval()
331     with torch.no_grad():
332         for inputs,labels in test_loader:
333             inputs, labels = inputs.to(device), labels.to(device)
334             out2 = model(inputs)
335 
336             ps = torch.exp(out2)
337             top_pred, top_class = ps.topk(1,dim=1)
338             for i in range(len(top_class)):
339                 l.append(labels[i].item())
340                 c.append(top_class[i].item())
341 
342     with open(f"xception_{data_dir}.pth", 'wb') as file:
343         pickle.dump(model, file)
344 
345     print(build_conf_matrix(l, c, 1))
346     print('accuracy:{}'.format(accuracy_score(l, c)))
347     print('precision:{}'.format(precision_score(l, c,average='weighted')))
348     print('recall:{}'.format(recall_score(l, c,average='weighted')))
349     print('f1-score:{}'.format(f1_score(l, c,average='weighted')))
350 
351 
352 def main():
353     ethalon_block_sizes = [4, 16, 32]
354     for bs in ethalon_block_sizes:
```

```
355         data_dir = f'cv_classification_{bs}'
356         vit_experiment(data_dir)
357         resnet34_experiment(data_dir)
358         xception_experiment(data_dir)
359
360
361 if __name__ == "__main__":
362     main()
```