Министерство цифрового развития, связи и массовых коммуникаций РФ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра вычислительных систем
Допустить к защите
Зав. кафедрой к.т.н., доцент
_____ Перышкова Е.Н.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Реализация моделей машинного обучения для задачи обнаружения мошеннических операций

Пояснительная записка

Студент Григорьев Ю.

Институт ИВТ Группа ИС-142

Руководитель Крамаренко К.Е,

Новосибирск - 2025

Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

КАФЕДРА
вычислительных систем

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

СТУДЕНТА Григорьева Ю. ГРУППЫ ИС-142

«УТВЕРЖДАЮ»
«_____» _____
Зав. кафедрой ВС
к.т.н., доцент
_____ Перышкова Е.Н.

Новосибирск, 2025 г.

1. Тема выпускной квалификационной работы бакалавра: «Реализация моделей машинного обучения для задачи обнаружения мошеннических операций» утверждена приказом СибГУТИ от «__»_____ 2025 г. № _____

2. Срок сдачи студентом законченной работы: 10 июня 2025 г.

3. Исходные данные к работе

1 Специальная литература

2 Материалы сети интернет

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов)	**Сроки выполнения по разделам**
Работа с библиотечными фондами, сбор и анализ материалов по теме выпускной квалификационной работы	03.02.25 – 22.02.25
Введение (постановка цели, задач, актуальности исследования)	24.02.25 – 01.03.25
Анализ известных решений и обоснование выбора моделей	03.03.25 – 15.03.25
Формирование требований к системе классификации и выбор датасета	17.03.25 – 29.03.25
Выбор программных средств и подготовка рабочей среды	19.04.25 – 24.04.25
Предобработка данных (балансировка, масштабирование признаков)	31.03.25 – 05.04.25
Разработка и обучение моделей машинного обучения	07.04.25 – 26.04.25
Оптимизация гиперпараметров моделей и анализ интерпретируемости по методикам SHAP и LIME	28.04.25 – 03.05.25
Сравнительный анализ результатов, тестирование моделей	05.05.25 – 17.05.25
Формулировка выводов	19.05.25 – 06.06.25

Дата выдачи задания: «_____» _____

Руководитель _____ Крамаренко К.Е.

Задание принял к исполнению «_____» _____

Студент _____ Григорьев Ю.

АННОТАЦИЯ

Выпускная квалификационная работа Григорьева Ю.
по теме «Реализация моделей машинного обучения для задачи обнаружения мошеннических операций»

Объем работы 62 страниц, на которых размещены 15 рисунков и 0 таблиц. При написании работы использовалось 12 источников.

Ключевые слова: машинное обучение, балансировка классов, задача бинарной классификации, интерпретируемость, ансамблевые модели, нейронные сети.

Работа выполнена на кафедре ВС СибГУТИ.
Руководитель – ст. преподаватель Крамаренко К.Е.,

Целью бакалаврской работы является исследование и реализация моделей машинного обучения для задачи обнаружения мошеннических операций в финансовых транзакциях. В условиях стремительного роста объема цифровых транзакций и усложнения мошеннических схем разработка эффективных методов их выявления становится ключевым фактором обеспечения безопасности финансовых систем. Особое внимание в работе уделено проблеме дисбаланса классов, характерной для данного типа задач, и поиску подходов к ее преодолению.

В рамках исследования проведен анализ современных методов машинного обучения, включая классические алгоритмы (логистическая регрессия, деревья решений), ансамблевые модели (случайные леса, градиентный бустинг) и модели глубокого обучения (нейронные сети). Изучены подходы к обработке дисбаланса данных, такие как oversampling (SMOTE), undersampling и перевзвешивание классов. Разработаны и обучены несколько моделей, оценка которых выполнена с использованием метрик precision, recall, F1-score, ROC-AUC и PR-AUC.

По итогам проведенных экспериментов определены наиболее эффективные модели и методы обработки данных для обнаружения мошенничества, а также выработаны рекомендации по их применению с учетом интерпретируемости и практической применимости в реальных условиях.

Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

ОТЗЫВ

на выпускную квалификационную работу студента
группы ИС-142 Григорьева Ю.
по теме «Реализация моделей машинного обучения для задачи обнаружения мошеннических операций»

Выпускная квалификационная работа студента группы ИС-142 Григорьева Ю. посвящена актуальной теме — разработке моделей машинного обучения для выявления мошеннических операций в финансовых транзакциях. Работа выполнена на высоком уровне, демонстрируя глубокое понимание предметной области и владение современными методами анализа данных.

Студент провел анализ существующих подходов к обнаружению мошенничества, обосновал выбор моделей (логистическая регрессия, дерево решений, случайный лес, градиентный бустинг, многослойный перцептрон) и реализовал их с использованием Python, scikit-learn и PyTorch. Для решения проблемы дисбаланса классов применен метод SMOTE, что позволило повысить качество классификации. Модель многослойного перцептрона показала наилучший результат метрики ROC-AUC в 0.9998.

Особо стоит отметить раздел интерпретируемости моделей, где использованы методы SHAP и LIME, обеспечившие объяснимость предсказаний, что важно для систем безопасности. Сравнительный анализ моделей, выполненный по метрикам ROC-AUC, Precision, Recall и F1-score, подкреплен наглядными графиками, что отражает научную строгость и внимание к деталям.

Работа отличается четкой структурой, логичным изложением и грамотным оформлением. Теоретическая часть подкреплена авторитетными источниками, практическая — выполнена с применением вычислительных систем. Выводы и рекомендации имеют практическую ценность для систем мониторинга транзакций.

К замечаниям можно отнести ограниченный охват современных нейронных сетей (например, рекуррентных или сверточных). Рекомендуется расширить исследование в направлении анализа временных зависимостей с использованием моделей LSTM или Transformer.

Работа Григорьева Ю. свидетельствует о его высоком уровне подготовки, умении решать сложные задачи и применять теоретические знания на практике. Она заслуживает оценки «отлично».

Оценка уровней сформированности общекультурных и профессиональных компетенций обучающегося:

Компетенции		Уровень сформированности компетенций		
		Высокий	Средний	Низкий
Общекультурные	ОК-5 (способность к коммуникации в устной и письменной формах на русском и иностранном языках для решения задач межличностного и межкультурного взаимодействия)	+		
	ОК-7 (способность к самоорганизации и самообразованию)	+		
Профессиональные	ПК-1 (способность применять естественнонаучные и математические знания для решения профессиональных задач)	+		
	ПК-2 (способность разрабатывать алгоритмы и программы, соответствующие поставленным задачам)	+		
	ПК-3 (способность использовать современные информационные технологии и программные средства для решения задач профессиональной деятельности)	+		
	ПК-5 (способность применять методы обработки и анализа данных в профессиональной деятельности)	+		
	ПК-7 (способность проводить исследования в области информатики и информационных технологий)	+		

Работа имеет практическую ценность	+	Тема предложена предприятием	
Работа внедрена		Тема предложена студентом	+
Рекомендую работу к внедрению		Тема является фундаментальной	
Рекомендую работу к опубликованию		Рекомендую студента в магистратуру	+
Работа выполнена с применением ЭВМ	+	Рекомендую студента в аспирантуру	

Старший преподаватель кафедры
вычислительных систем СибГУТИ

_____ Крамаренко К.Е.
(Крамаренко Константин Евгеньевич)
«_____» _____

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современная цифровая экономика сопровождается ростом числа финансовых транзакций через интернет и мобильные приложения, что ведет к увеличению мошеннических операций, наносящих ущерб банкам и клиентам. Мошенники применяют сложные схемы, такие как использование третьих лиц, маскировка транзакций через «мертвые души», взлом кассовых аппаратов или установка скиммеров на банкоматы, что делает традиционные методы защиты недостаточно эффективными. Сложность мошеннических схем требует современных методов их выявления в реальном времени, способных адаптироваться к новым угрозам.

Традиционные rule-based системы теряют актуальность из-за адаптации мошенников и появления новых типов атак. Методы машинного обучения, включая глубокое обучение, позволяют выявлять скрытые закономерности и адаптироваться к изменениям, но сталкиваются с проблемой дисбаланса классов (мошеннических транзакций — менее 0.17%). Это усложняет нахождение и интерпретацию паттернов в данных, так как стандартные алгоритмы склонны игнорировать редкие случаи мошенничества.

Работа посвящена исследованию и реализации моделей машинного обучения для обнаружения мошеннических операций. **Цель** — проанализировать методы, разработать и сравнить модели с учётом дисбаланса данных, выявив лучшие подходы. **Новизна** заключается в комплексном подходе к анализу методов с акцентом на дисбаланс и практическую применимость, детальную интерпретацию результатов моделей с выявлением конкретных признаков, которые влияют на принимаемое решение.

1 Постановка задачи

Задача обнаружения мошеннических операций в финансовых транзакциях относится к категории задач бинарной классификации. Каждая транзакция должна быть отнесена к одному из двух классов: «нормальная» (легитимная) или «мошенническая». Формально, для заданного набора данных D, содержащего признаки транзакций X и соответствующие метки классов $y \in \{0,1\}$ (где 0 — нормальная транзакция, 1 — мошенническая), требуется построить модель $f(X)$, которая минимизирует ошибки классификации.

Основная сложность — выраженный дисбаланс классов: мошеннические транзакции составляют менее 0.17% данных. Стандартные алгоритмы, оптимизированные для *Accuracy*, переобучаются на преобладающий класс, игнорируя редкие случаи мошенничества, что недопустимо для безопасности финансовых операций.

Для достижения цели исследования поставлены задачи: изучить методы машинного обучения для обнаружения мошенничества (логистическая регрессия, деревья решений, ансамблевые методы, нейронные сети); оценить методы обработки дисбаланса данных (SMOTE, undersampling, перевзвешивание); реализовать и обучить модели; оценить модели по метрикам Precision, Recall, F1-score, ROC-AUC; сравнить результаты и определить лучшие модели; рассмотреть интерпретируемость и применимость полученных моделей.

Итоговая модель должна сочетать высокие Precision и Recall, что подтверждается метриками F1-score и ROC-AUC.

2 Решение

2.1 Датасет и описание данных

Для проведения исследования и экспериментов по обнаружению мошеннических операций был выбран публичный датасет *Credit Card Fraud Detection*, доступный на платформе Kaggle. Данный набор данных содержит информацию о транзакциях, совершенных европейскими держателями кредитных карт в течение двух дней сентября 2013 года. Он представляет собой анонимизированный массив, специально подготовленный для задач бинарной классификации, где требуется разделение транзакций на нормальные и мошеннические. Выбор этого датасета обусловлен его широким признанием в научном сообществе, а также реалистичным отражением проблемы дисбаланса классов, характерной для реальных финансовых систем.

Общий объем датасета составляет 284,807 транзакций, каждая из которых описывается набором из 31 признака. Структура данных включает поля: *Time* — время в секундах, прошедшее от первой транзакции в наборе (диапазон значений отражает двухдневный период); *V1–V28*: 28 анонимизированных признаков, полученных с использованием метода главных компонент (*Principal Component Analysis, PCA*), представляют собой преобразованные исходные данные (такие как номер счета, местоположение, персональная информация), скрытые для обеспечения конфиденциальности; *Amount*: сумма транзакции в денежных единицах (единственный неанонимизированный количественный признак); *Class*: целевая переменная, принимающая значения 0 (нормальная транзакция) или 1 (мошенническая транзакция).

Анонимизация через PCA сохраняет закономерности, но ограничивает интерпретируемость, что делает датасет подходящим для выявления скрытых паттернов. Дисбаланс классов значителен: 284,315 нормальных (99.83%) и 492 мошеннических (0.17%) транзакции, что соответствует реальным финансовым

сценариям и усложняет обучение моделей, склонных переобучаться на доминирующий класс.

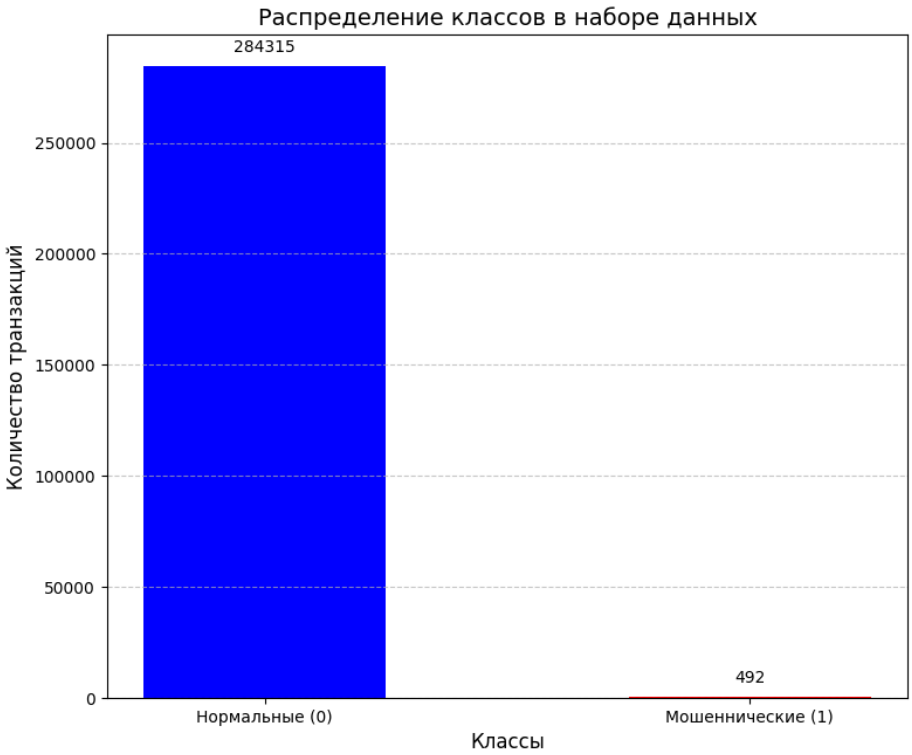

Рисунок 2.1 – Распределение классов в исследуемом наборе данных

Для наглядного представления структуры данных выше приведено распределение классов (см. Рисунок 1). Этот дисбаланс подчеркивает необходимость применения специализированных методов обработки данных и выбора метрик, устойчивых к подобным условиям.

2.2 Основные требования к моделям

При разработке моделей машинного обучения для классификации транзакций на нормальные и мошеннические необходимо учитывать специфику задачи и практические потребности финансовых систем. Основные требования к моделям включают следующие аспекты.

Высокая эффективность классификации

Модель должна обеспечивать оптимальный баланс между точностью (precision) и полнотой (recall). Высокая точность минимизирует количество ложных срабатываний, когда нормальные транзакции ошибочно

классифицируются как мошеннические, что важно для сохранения доверия клиентов и снижения операционных издержек. Одновременно высокая полнота гарантирует выявление максимального числа реальных мошеннических операций, предотвращая финансовые потери. Учитывая приоритет обнаружения редких событий, предпочтение отдается метрикам, устойчивым к дисбалансу, таким как F1-score и ROC-AUC.

Интерпретируемость решений

В финансовой сфере объяснимость результатов классификации имеет ключевое значение для соответствия регуляторным требованиям и обеспечения прозрачности процессов. Модели должны предоставлять возможность анализа факторов, влияющих на принятие решений. Например, традиционные алгоритмы, такие как логистическая регрессия и деревья решений, обладают высокой интерпретируемостью, в то время как методы глубокого обучения (нейронные сети) требуют дополнительных инструментов для интерпретации (например, SHAP или LIME). Выбор модели предполагает компромисс между точностью и объяснимостью.

Устойчивость к дисбалансу классов

При перевесе нормальных транзакций (99.83%) нужны методы балансировки (oversampling, undersampling, веса классов) и метрики (ROC-AUC, PR-AUC), исключающие оптимизацию по accuracy.

2.3 Известные решения

Задача обнаружения мошеннических операций с кредитными картами активно изучается в научной среде, и за последние годы было предложено множество решений, основанных на методах машинного обучения. Среди традиционных подходов выделяются алгоритмы supervised learning, такие как логистическая регрессия, случайные леса (Random Forest) и градиентный бустинг (например, XGBoost), которые демонстрируют высокую точность на сбалансированных наборах данных [1], однако их ограничение заключается в

слабой адаптации к новым, ранее не встречавшимся схемам мошенничества, а также в зависимости от качества разметки данных. Для борьбы с дисбалансом классов часто применяются методы ресемплинга (*resampling*), такие как SMOTE (*Synthetic Minority Oversampling Technique*) (*оверсемплинг*, увеличение миноритарного класса) [4], или ансамблевые подходы, повышающие чувствительность к редкому классу мошеннических транзакций.

В последние годы все большее внимание уделяется методам глубокого обучения (deep learning), включая сверточные нейронные сети (CNN) и рекуррентные нейронные сети (RNN), которые способны моделировать сложные временные зависимости и аномалии в данных без необходимости ручного проектирования признаков [7]. В данной работе для исследования были выбраны модели логистической регрессии, дерева решений, случайного леса (Random Forest), градиентный бустинг и многослойный перцептрон, так как они представляют собой комбинацию простых интерпретируемых методов и мощных ансамблевых и нейросетевых подходов, позволяющих оценить как базовые, так и продвинутые возможности классификации. Выбор также обусловлен необходимостью показать разную интерпретируемость, точность и способность моделей справляться с дисбалансом классов, что соответствует целям исследования и практической применимости в условиях реальных финансовых данных.

2.4 Программные и аппаратные средства

Для реализации моделей машинного обучения и проведения экспериментов в рамках данной работы были выбраны следующие программные и аппаратные средства, обеспечивающие высокую производительность, удобство разработки и воспроизводимость результатов.

Разработка моделей осуществлялась на языке программирования Python (версия 3.13.2), который выбран за его широкую популярность в области машинного обучения, наличие обширного набора библиотек и простоту использования. Для реализации традиционных моделей

(логистическая регрессия, дерево решений, случайный лес, градиентный бустинг) применялась библиотека scikit-learn, предоставляющая готовые инструменты для классификации, предобработки данных и оценки моделей. Многослойный перцептрон (MLP) был реализован с использованием фреймворка PyTorch, который выбран за его гибкость в работе с нейронными сетями, поддержку динамических вычислительных графов и возможность ускорения вычислений на GPU через CUDA. Для предобработки данных использовались библиотеки pandas (работа с данными в формате таблиц) и NumPy (вычисления с массивами), а для визуализации результатов — matplotlib. Балансировка классов выполнялась с помощью метода SMOTE из библиотеки imbalanced-learn, что позволило эффективно справиться с дисбалансом в датасете. Разработка велась в среде Visual Studio Code, обеспечивающей удобство отладки и управления проектом.

Вычисления проводились на персональном компьютере с операционной системой Microsoft Windows 11 Pro (версия 10.0.26100). Основные характеристики системы: процессор 11th Gen Intel Core i5-11400F с частотой 2.60 ГГц, 16 ГБ оперативной памяти, SSD-накопитель объемом 360 ГБ. Для ускорения вычислений, особенно при обучении нейронной сети (MLP), использовалась видеокарта NVIDIA GeForce RTX 3060 с 3584 ядрами CUDA, тактовой частотой 1882 МГц, 12 ГБ выделенной видеопамяти GDDR6 и пропускной способностью памяти 360.048 ГБ/с. Общая доступная графическая память составила 20,414 МБ, что позволило эффективно обрабатывать большие объемы данных. Видеокарта использует драйвер версии 572.42 (от 13 февраля 2025 года), обеспечивающий стабильную работу с CUDA.

Выбор программных средств обусловлен их совместимостью, производительностью и распространенностью в задачах машинного обучения. Python и scikit-learn обеспечили простоту реализации традиционных моделей, а PyTorch позволил гибко настраивать архитектуру MLP и использовать GPU

для ускорения обучения. Аппаратная конфигурация была выбрана за ее доступность (имевшийся персональный компьютер) и высокую вычислительную мощность и поддержку CUDA, что существенно сократило время обучения нейронной сети (например, обучение MLP с архитектурой 128–64–32 заняло около 600 секунд). Процессор и 16 ГБ оперативной памяти обеспечили стабильную работу при предобработке данных и обучении ансамблевых моделей, таких как случайный лес и градиентный бустинг. SSD-накопители ускорили загрузку данных и доступ к файлам проекта, что повысило общую эффективность работы. Некоторые методы машинного обучения были реализованы с использованием готовых библиотек и шаблонов, а не разработаны вручную, что обусловлено несколькими факторами. Во-первых, библиотеки, такие как scikit-learn, содержат оптимизированные и проверенные реализации алгоритмов, которые обеспечивают высокую производительность и точность. Во-вторых, использование готовых библиотек минимизирует риск ошибок в коде, так как эти инструменты прошли тестирование в академическом и промышленном сообществе. В-третьих, в рамках ограниченного времени на выполнение работы приоритет был отдан анализу и сравнению моделей, а не разработке алгоритмов с нуля. Однако в некоторых случаях, например, для логистической регрессии и дерева решений, была выполнена собственная реализация (см. Приложение А, Листинги А.1 и А.2), чтобы глубже понять их внутреннюю работу и продемонстрировать навыки реализации модели с нуля. Модель дерева решений была использована далее в ансамблевых методах. Принятый подход позволил сбалансировать исследовательские цели и техническую реализацию, сосредоточив усилия на ключевых аспектах работы.

2.5 Выбранные метрики для оценки моделей

Для оценки качества моделей машинного обучения в задаче обнаружения мошеннических операций были выбраны следующие метрики: Accuracy, Precision, Recall, F1-score и ROC-AUC. Каждая метрика отражает

различные аспекты производительности модели, что позволяет провести комплексный анализ в условиях несбалансированного датасета Credit Card Fraud Detection (99.83% нормальных транзакций, 0.17% мошеннических).

Рассмотрим их определение, результаты и применимость к данной задаче. Accuracy указывает на высокую общую точность предсказаний. Однако в условиях сильного дисбаланса эта метрика может быть обманчивой: модель, предсказывающая все транзакции как нормальные, достигла бы Accuracy 0.9983, но не выявила бы ни одного мошенничества. Precision демонстрирует высокую точность среди предсказанных мошенничеств, но не учитывает пропущенные случаи. Recall не отражает ложных срабатываний на нормальных транзакциях. F1-score, вычисляемый как гармоническое среднее Precision и Recall, балансирует эти две метрики и полезен для оценки компромисса между точностью и полнотой. Например, при настройке порога F1-score помогает найти оптимальное значение, минимизируя как пропуски мошенничеств, так и ложные тревоги.

Основной метрикой для сравнения моделей выбран ROC-AUC (*Receiver Operating Characteristic — Area Under Curve*), который отражает способность модели различать классы, измеряя площадь под кривой, построенной на основе истинных положительных скоростей (TPR, равной Recall) и ложных положительных скоростей (FPR) при варьировании порога. Значение ближе к 1 указывает на идеальное разделение классов, тогда как 0.5 соответствует случайному угадыванию. Выбор ROC-AUC обусловлен спецификой задачи: важно учитывать как пропуск мошенничеств (низкий Recall), так и ложные срабатывания на нормальных транзакциях (высокий FPR), особенно при дисбалансе 0.17%. В отличие от Accuracy, которая искажается преобладанием нормальных транзакций, ROC-AUC независима от порога и дает общую оценку.

3 Описание и реализация моделей

3.1 Логистическая регрессия

Логистическая регрессия — классический метод бинарной классификации [5], который оценивает вероятность принадлежности объекта к положительному классу с использованием логистической функции (сигмоиды):

$$\sigma(x)=\frac{1}{1+e^{-x}},$$

где $x=w_1x_1+w_2x_2+...+w_nx_n+b$ — линейная комбинация признаков $x_1,...,x_n$, весов $w_1,...,w_n$ и смещения b. Цель метода — оптимизация параметров модели путем минимизации функции потерь — бинарной кросс-энтропии:

$$L(y,\hat{y})=\frac{-1}{N}\sum_{i=1}^{N}[y_i\cdot\log(\hat{y}_i)+(1-y_i)\cdot\log(1-\hat{y}_i)],$$

где y — истинные метки, $\hat{y}$ — предсказанные вероятности, а N — число примеров.

Логистическая регрессия отличается простотой реализации и высокой интерпретируемостью: веса модели напрямую указывают на вклад каждого признака в классификацию. Применение регуляризации (L1 или L2) повышает ее устойчивость к переобучению. Однако метод предполагает линейную разделимость классов, что ограничивает его эффективность при наличии сложных нелинейных зависимостей в данных. Кроме того, модель чувствительна к дисбалансу классов, что особенно критично для задачи обнаружения мошенничества, где доля положительного класса мала.

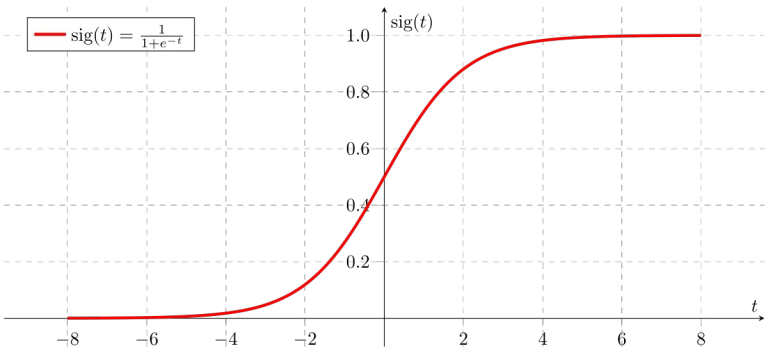

Рисунок 3.1.1 – Логистическая функция (сигмоидная кривая)

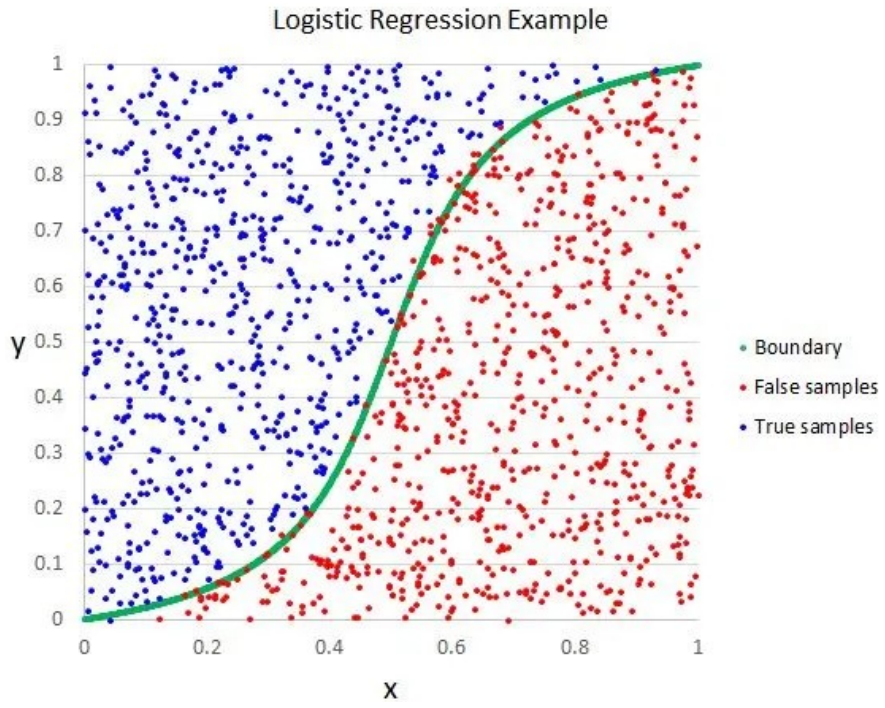

Рисунок 3.1.2 Пример работы модели для задачи бинарной классификации

Для повышения качества обучения числовые признаки были нормализованы с использованием метода StandardScaler из библиотеки scikit-learn. Это позволило привести значения признаков к единому масштабу (среднее 0, стандартное отклонение 1), устранив проблему различий в диапазонах и ускорив сходимость градиентного спуска. Данные методы используются и в моделях реализованных далее.

Учитывая сильный дисбаланс в датасете (99.83% нормальных транзакций против 0.17% мошеннических), были применен метод oversampling с использованием SMOTE, который генерирует синтетические примеры меньшинственного класса, уравновешивая распределение. О различиях разных методов балансировки см. **пункт 4.3 Влияние балансировки данных**.

Модель логистической регрессии реализована вручную с использованием градиентного спуска для минимизации бинарной кросс-энтропии (см. **Приложение А, Листинг А.1**). Обновление параметров выполняется по формулам:

$$w := w - \eta \frac{\partial L}{\partial w},\ b := b - \eta \frac{\partial L}{\partial b},$$

где η — скорость обучения. Параметры подбирались итеративно до достижения сходимости.

Результаты полученных от модели метрик:

- *Accuracy*: 0.9444 — доля правильно классифицированных примеров.
- *Precision*: 0.9830 — доля верно предсказанных мошеннических транзакций среди всех положительных предсказаний.
- *Recall*: 0.9046 — доля обнаруженных мошеннических транзакций из их общего числа.
- *F1-score*: 0.9422 — сбалансированная мера между precision и recall.
- *ROC-AUC*: 0.9848 — показатель качества классификации при разных порогах.

Результаты демонстрируют высокую общую точность, однако ограниченная полнота указывает на необходимость дальнейшей оптимизации для задач с дисбалансом. При внесении новых данных результат может кардинально измениться.

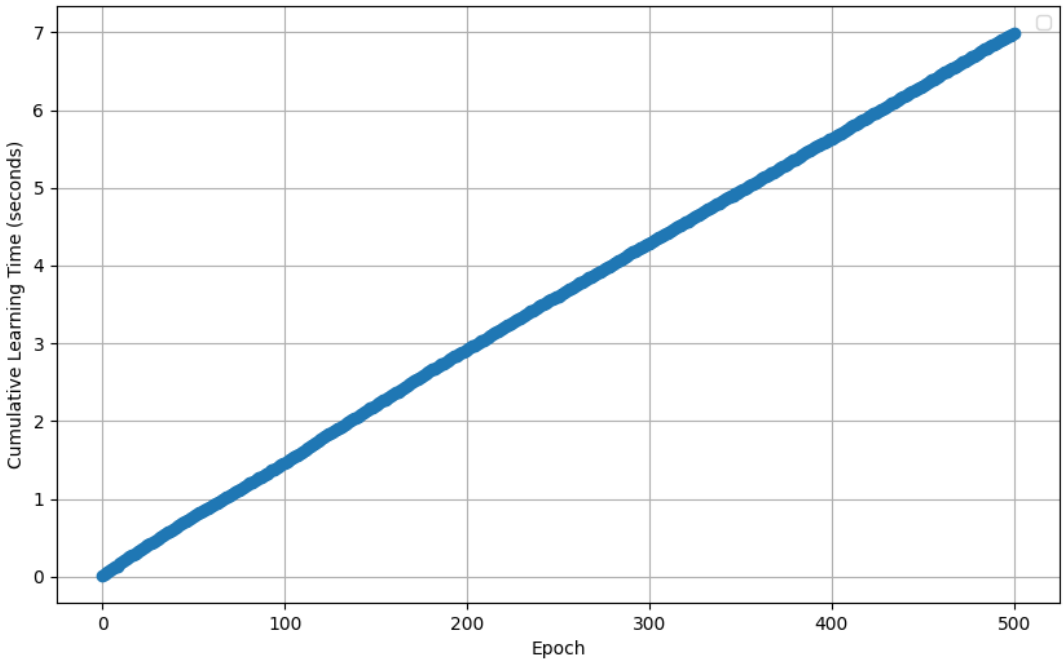

Рисунок 3.1.3 – Зависимость кумулятивного времени обучения от пройденной эпохи

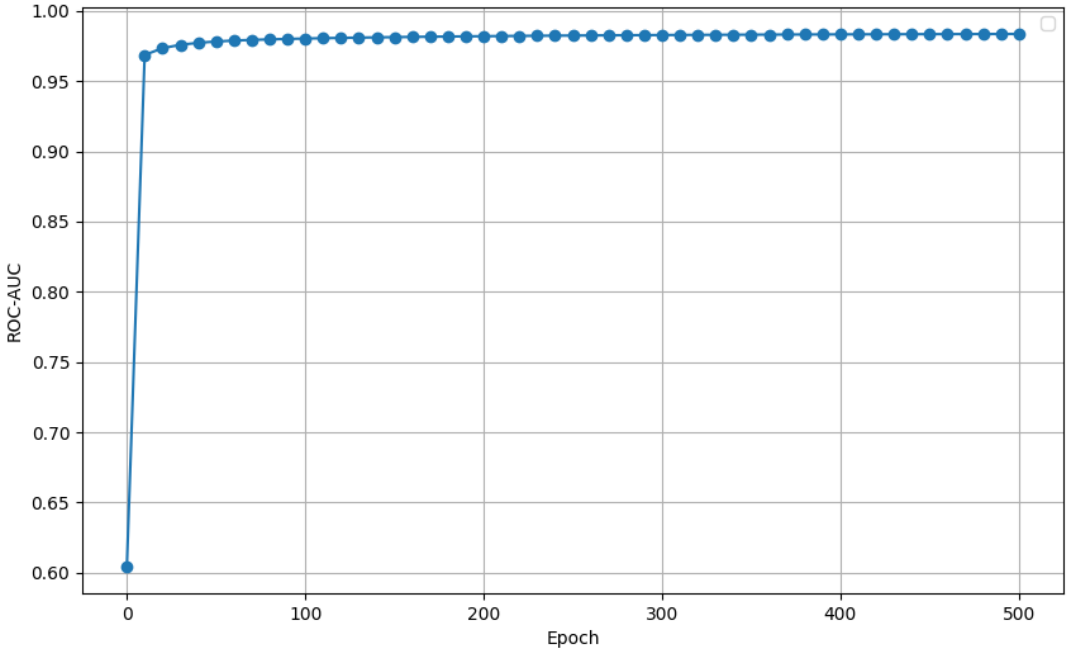

Рисунок 3.1.4 – Зависимость ROC-AUC от числа эпох обучения

3.2 Дерево решений

Дерево решений — это метод машинного обучения, широко применяемый для задач классификации и регрессии [2]. Оно представляет собой иерархическую структуру, где каждый внутренний узел соответствует

проверке значения определенного признака, ветви — возможным исходам этой проверки, а листья — предсказанным классам. Классификация объекта начинается с корневого узла и завершается в листе, определяющем принадлежность к классу 0 (нормальная транзакция) или 1 (мошенническая). В данной работе для выбора оптимальных разбиений использовался критерий Джини, который минимизирует неоднородность подмножеств данных на каждом шаге построения дерева.

Дерево решений отличается простотой интерпретации: структура дерева позволяет визуализировать процесс принятия решений и оценить вклад каждого признака. Метод гибок и способен моделировать как линейные, так и нелинейные зависимости между признаками. Однако он склонен к переобучению, особенно при отсутствии ограничений на глубину, что снижает обобщающую способность на новых данных. Кроме того, дерево нестабильно к шуму: небольшие изменения в обучающей выборке могут существенно изменить его структуру.

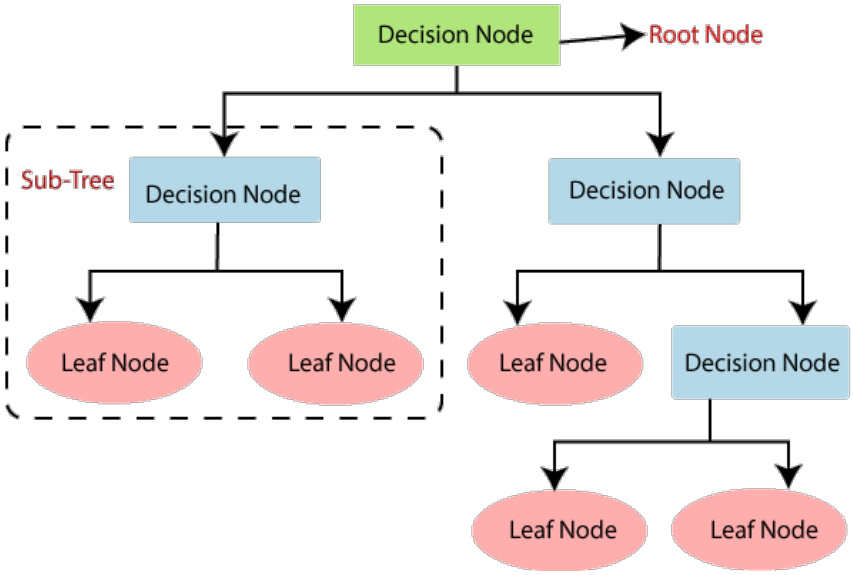

Рисунок 3.2.1 – Пример структуры дерева решений

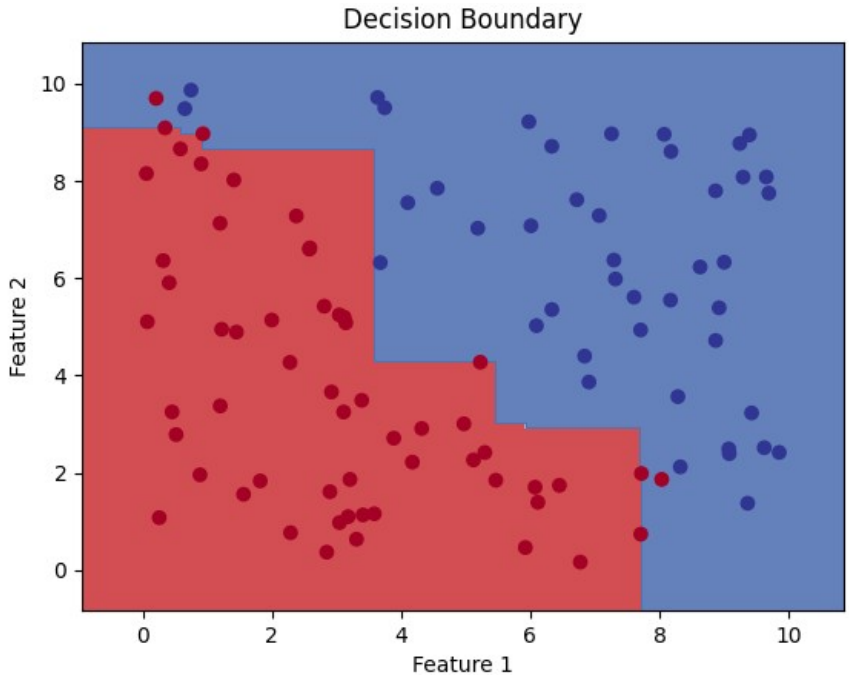

Рисунок 3.2.2 – Пример работы модели для задачи бинарной классификации

Обучение дерева решений проводилось с использованием критерия Джини, который измеряет неоднородность данных в узле по формуле:

$$Gini = 1 - \sum_{k=1}^{K} p_k^2,$$

где p_k — доля объектов класса k в узле, $K = 2$ — число классов (0 и 1). На каждом этапе для всех признаков и возможных пороговых значений рассчитывается значение критерия Джини, далее выбирается разбиение, минимизирующее средневзвешенное значение Джини для дочерних узлов. Процесс продолжается до достижения заданной максимальной глубины или минимального числа объектов в узле. В данной реализации глубина дерева и минимальное число объектов в листе ограничивались (например, *max_depth = 10*, *min_samples_leaf = 5*), чтобы предотвратить переобучение и повысить обобщающую способность модели (см. Приложение А, Листинг А.2).

Результаты метрик для модели Дерева решений:

- *Accuracy*: 0.9349 — общая доля правильных предсказаний.
- *Precision*: 0.9453 — точность выявления мошеннических транзакций.

- *Recall*: 0.9235 — полнота обнаружения мошенничества.
- *F1-score*: 0.9343 — сбалансированная мера precision и recall.
- *ROC-AUC*: 0.9349 — показатель качества классификации при разных порогах.

Модель продемонстрировала хорошую способность к выявлению нелинейных зависимостей, однако ее эффективность ограничена чувствительностью к настройке гиперпараметров. Параметр ROC-AUC оказался значительно ниже модели логистической регрессии.

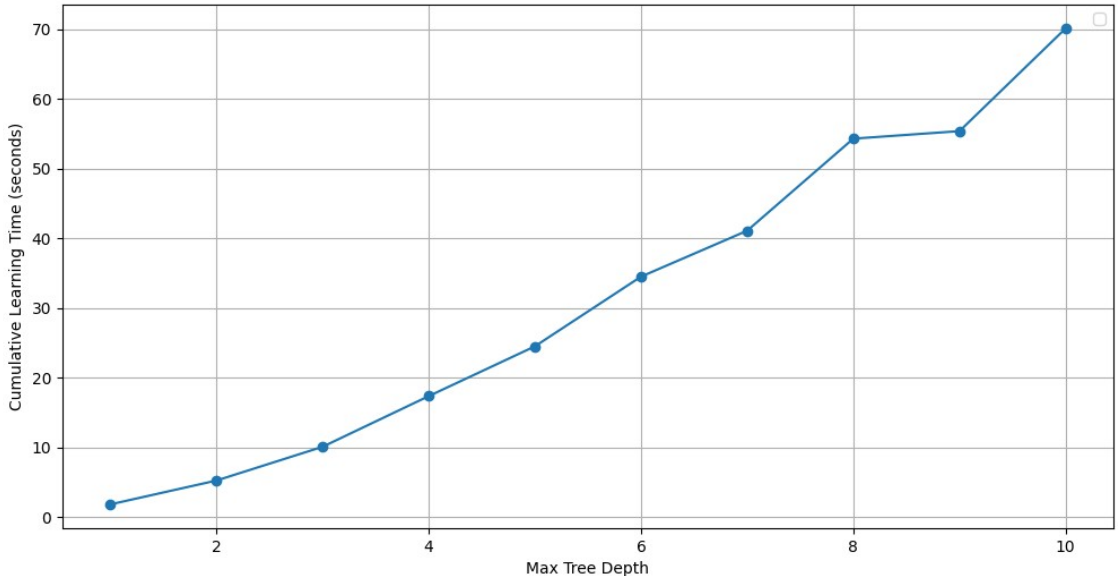

Рисунок 3.2.3 – Зависимость кумулятивного времени обучения от максимальной глубины дерева

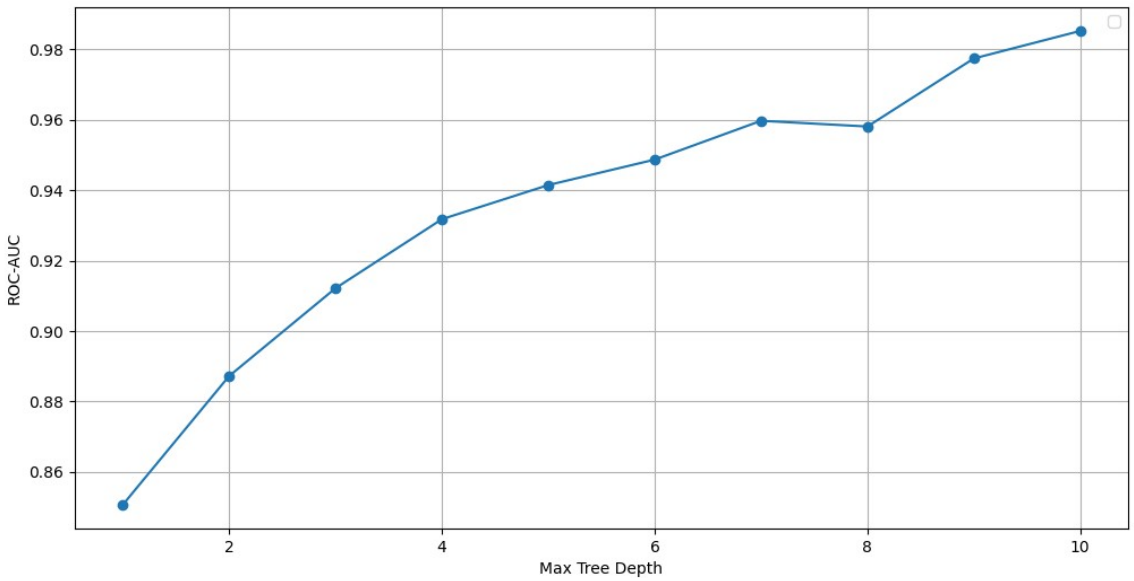

Рисунок 3.2.4 – Зависимость ROC-AUC от максимальной глубины дерева

3.3 Случайный лес

Случайный лес (Random Forest) — это ансамблевый метод машинного обучения, объединяющий множество деревьев решений для повышения точности и устойчивости классификации. Основной принцип заключается в снижении переобучения путем усреднения предсказаний независимых деревьев, каждое из которых обучается на случайной подвыборке данных и признаков. Метод сочетает технику bootstrap-семплирования (выборки с возвращением) и случайный выбор подмножества признаков на каждом этапе разбиения, что делает его эффективным для задач с шумными или несбалансированными данными, таких как обнаружение мошеннических транзакций.

Модель строит n деревьев решений. Каждое дерево обучается на подвыборке данных и использует случайное подмножество признаков для выбора оптимального разбиения в каждом узле. На этапе предсказания итоговый результат определяется голосованием большинства для задачи классификации:

$$y = mode(y_1, y_2, \ldots, y_n),$$

где y_i — предсказание i-го дерева. Вероятность принадлежности к классу вычисляется как среднее:

$$P(\text{class}=1)=\frac{1}{n}\sum_{i=1}^{n} P_i(\text{class}=1)$$

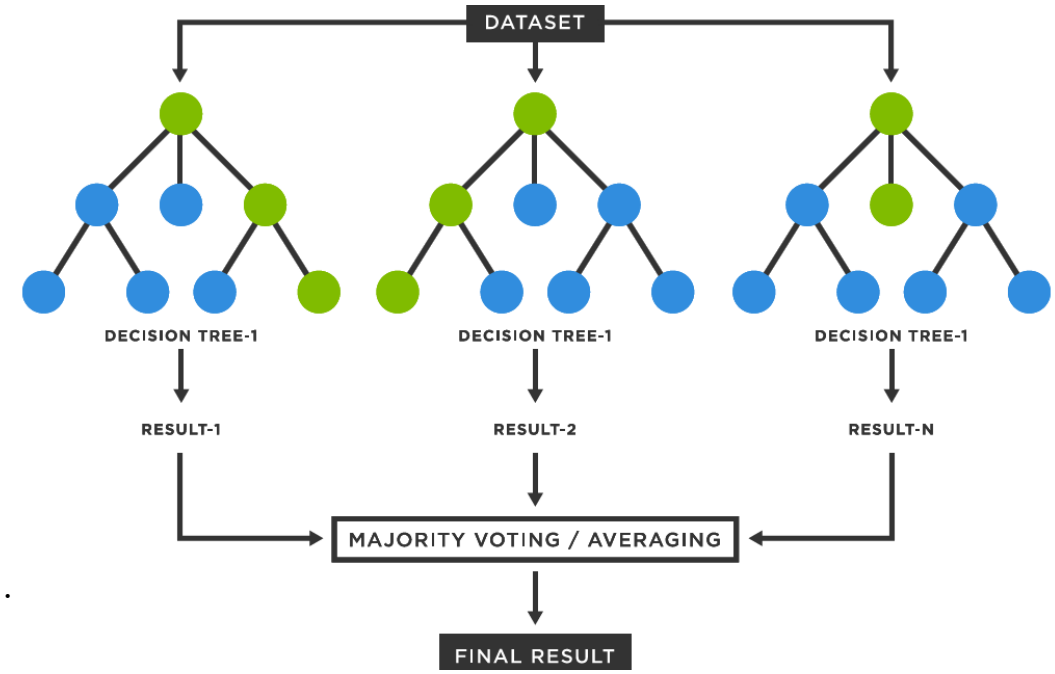

Рисунок 3.3.1 – Структура модели Random Forest

Каждое дерево строится с использованием критерия Джини (см. Дерево решений):

$$Gini=1-\sum_{k=1}^{K} p_k^2,$$

где p_k — доля объектов класса k в узле, $K=2$ — число классов.

На каждом уровне выбирается случайное подмножество признаков (например, $m=\sqrt{M}$, где M — общее число признаков), далее определяется разбиение, минимизирующее критерий Джини. Гиперпараметры, такие как максимальная глубина (*max_depth*) и минимальное число объектов для разбиения (*min_samples_split*), ограничивались (например, *max_depth = 10*, *min_samples_split = 5*) для предотвращения переобучения.

Число деревьев (n) задавалось с некоторым шагом до ограничения (например, $max = 100$, $step = 5$) (см. Приложение А, Листинг А.3).

Результаты метрик:

- *Accuracy*: 0.9535 — доля правильных предсказаний.
- *Precision*: 0.9884 — точность выявления мошенничества.
- *Recall*: 0.9181 — полнота.
- *F1-score*: 0.9519 — сбалансированная мера.
- *ROC-AUC*: 0.9536 — показатель разделимости классов.

Модель показала высокую точность и устойчивость, эффективно справляясь с нелинейными зависимостями и дисбалансом.

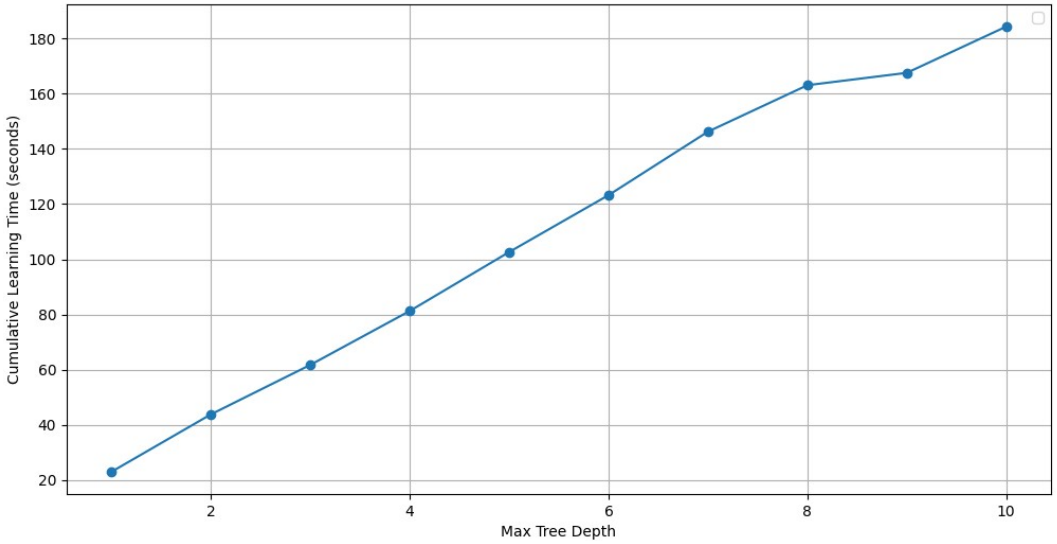

Рисунок 3.3.2 – Зависимость времени обучения от максимальной глубины деревьев

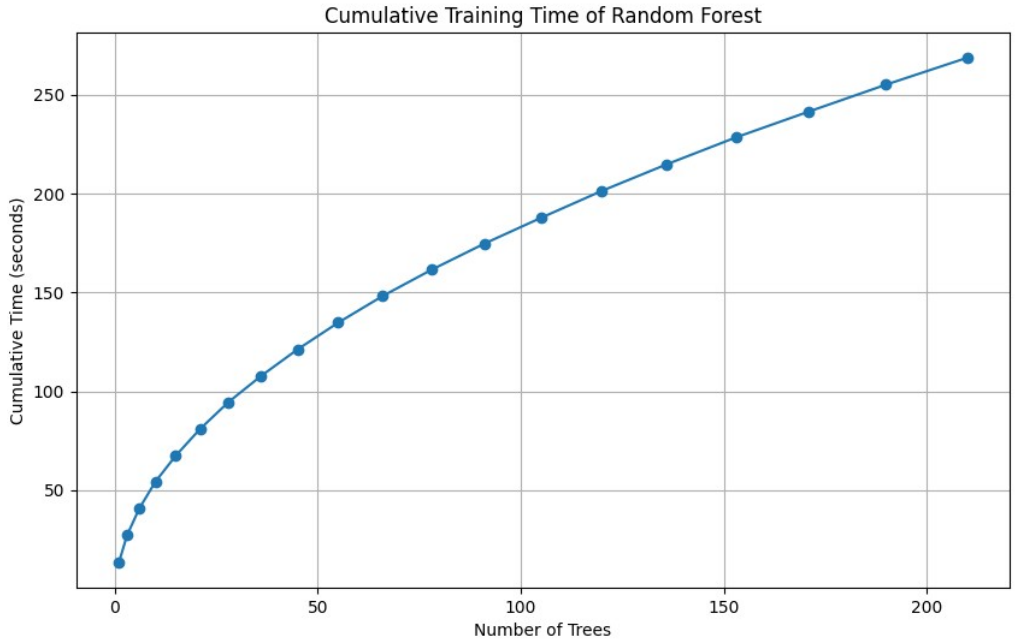

Рисунок 3.3.3 – Зависимость кумулятивного времени обучения от количества деревьев решений

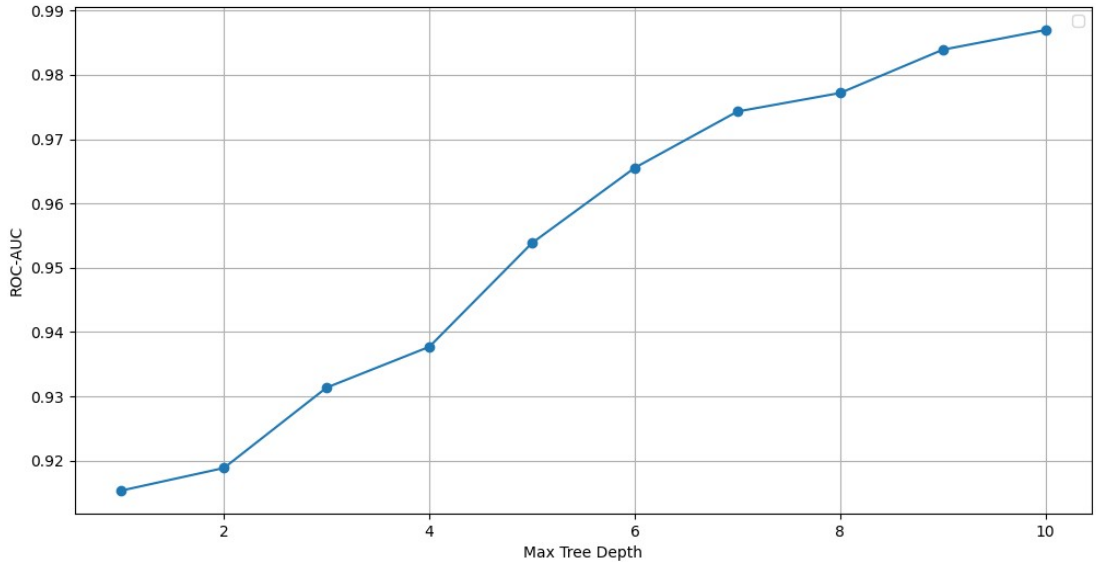

Рисунок 3.3.4 – Зависимость ROC-AUC от максимальной глубины деревьев

3.4 Градиентный бустинг

Градиентный бустинг (Gradient Boosting) — это ансамблевый метод машинного обучения, который последовательно объединяет деревья решений для повышения точности классификации [11]. В отличие от случайного леса, где деревья обучаются независимо, в градиентном бустинге каждое следующее дерево корректирует ошибки предыдущих, минимизируя

заданную функцию потерь. Это делает метод особенно эффективным для задач с высокой требовательностью к точности, таких как обнаружение мошеннических транзакций.

Начальное предсказание инициализируется константой (например, логарифмом отношения классов), а затем каждое дерево обучается на остатках ошибок предыдущего ансамбля. Вклад каждого дерева регулируется параметром скорости обучения η (*learning_rate*), а для повышения устойчивости используется подвыборка данных (*subsample*).

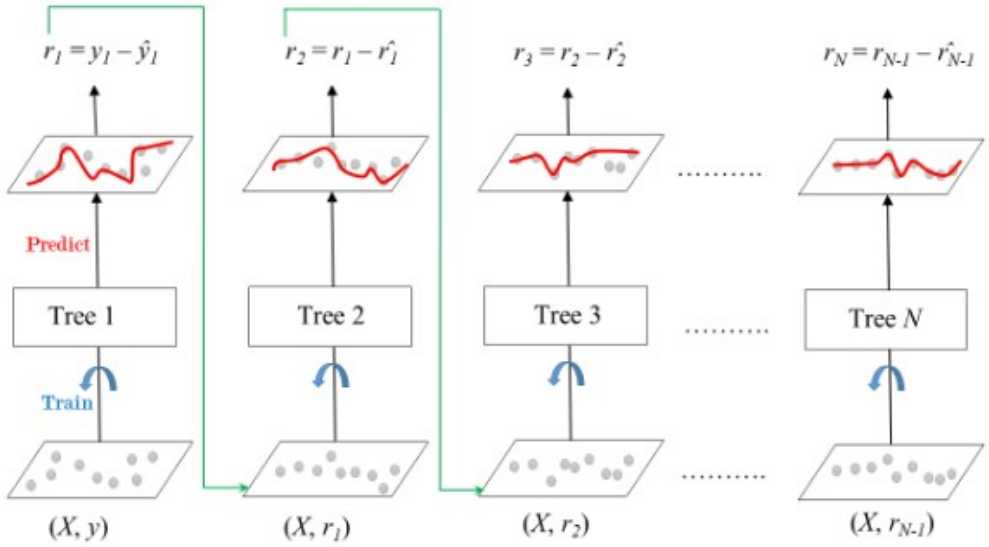

Рисунок 3.4.1 – Визуализация работы градиентного бустинга

Процесс обучения включает в себя инициализацию предсказания логарифмом отношения классов и построение деревьев, минимизирующих бинарную кросс-энтропию, которая вычисляется по следующей формуле:

$$L(y, \hat{y}) = \frac{-1}{N} \sum_{i=1}^{N} [y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)].$$

Далее выполняется обновление остатков с учетом градиента функции потерь и параметра η. Для разбиения узлов используется критерий уменьшения ошибки, а регуляризация достигается через ограничение глубины дерева (*max_depth*), минимального числа объектов в узле (*min_samples_split*) и

доли подвыборки (*subsample*). Например, применялись значения *max_depth = 5, min_samples_split = 5, subsample = 0.8* (см. Приложение А, Листинг А.4).

Итоговое предсказание вычисляется как сумма взвешенных вкладов всех деревьев:

$$\hat{y} = \sum_{i=1}^{n} eta \cdot T_i(x),$$

где n — число деревьев, $T_i(x)$ — предсказание i-го дерева, η — скорость обучения. Вероятность преобразуется через сигмоиду, а класс определяется по порогу (например, *0.5*).

Метрики:

- *Accuracy*: 0.9971 — доля правильных предсказаний.
- *Precision*: 0.9984 — точность выявления мошенничества.
- *Recall*: 0.9957 — полнота обнаружения.
- *F1-score*: 0.9971 — сбалансированная мера.
- *ROC-AUC*: 0.9970 — высокая разделимость классов.

Результаты подтверждают превосходство градиентного бустинга над предыдущими моделями благодаря последовательной коррекции ошибок.

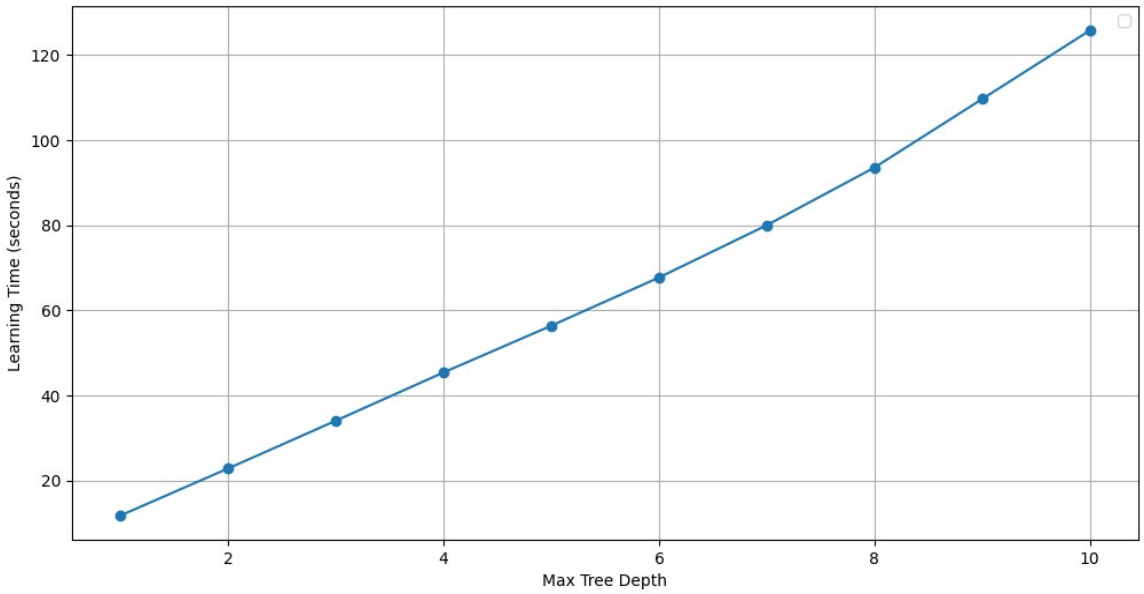

Рисунок 3.4.2 – Зависимость времени обучения от максимальной глубины деревьев

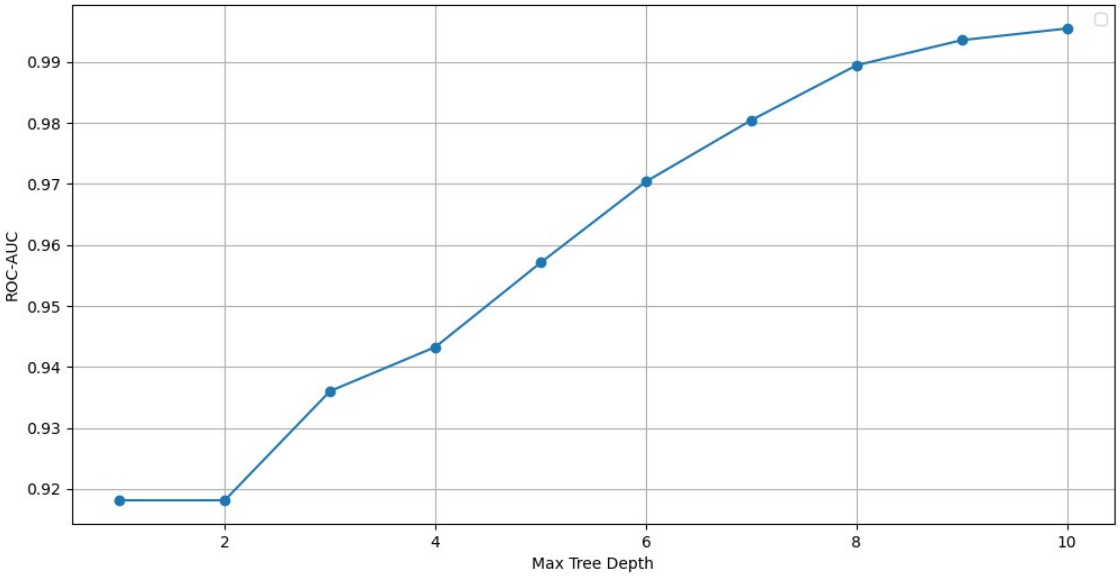

Рисунок 3.4.3 – Зависимость ROC-AUC от глубины деревьев

3.5 Многослойный перцептрон

Многослойный перцептрон (MLP) — это тип искусственной нейронной сети, подходящий для задач классификации и регрессии [6]. Он состоит из входного слоя, одного или нескольких скрытых слоев и выходного слоя. Каждый нейрон выполняет линейное преобразование входных данных с последующим применением нелинейной функции активации, что позволяет моделировать сложные зависимости в данных, включая нелинейные.

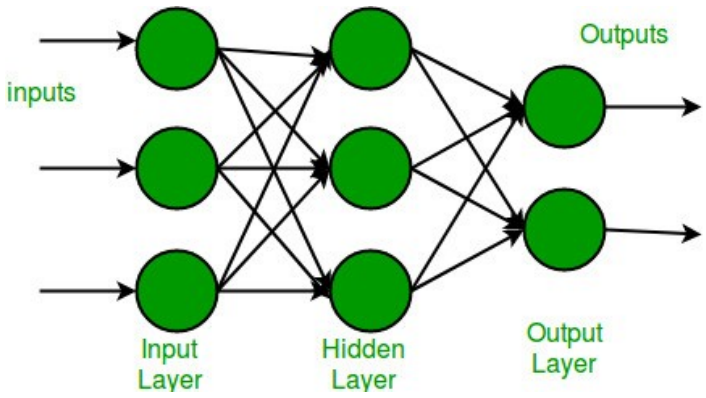

Рисунок 3.5.1 – Структура многослойного перцептрона

Для балансировки данных, как и в предыдущих моделях, использован метод SMOTE для устранения дисбаланса путем генерации синтетических примеров меньшинственного класса. Признаки нормализованы с помощью StandardScaler (среднее 0, стандартное отклонение 1) для обеспечения стабильного обучения нейронной сети.

Модель реализована с использованием библиотеки PyTorch (см. Приложение А, Листинг А.5) и включает в себя *Входной слой*: 30 нейронов, соответствующих числу признаков датасета, *Скрытые слои*: первый (128 нейронов, активация ReLU, dropout (0.3) для регуляризации), второй (64 нейрона, активация ReLU, dropout (0.3)) третий (32 нейрона, активация ReLU), и *Выходной слой*: 1 нейрон с сигмоидной активацией, возвращающий вероятность класса 1 (мошенничество).

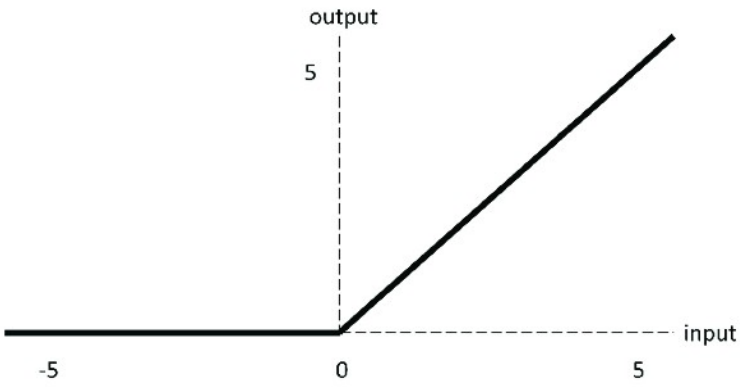

Рисунок 3.5.2.1 – Функция активации ReLU

Функция активации ReLU, определенная как $f(x)=max(0,x)$, была выбрана по следующим причинам [8]: в отличие от сигмоиды или гиперболического тангенса, ReLU предотвращает затухание градиентов во время обратного распространения ошибки. Это обеспечивает более стабильное и быстрое обучение, что критично для сети с тремя скрытыми слоями (128–64–32), где накопление градиентов могло бы замедлить сходимость. ReLU также вводит нелинейность, позволяя модели эффективно улавливать сложные зависимости в данных, что особенно важно для задачи

классификации, где паттерны часто имеют нелинейный характер. Довольно важная характеристика для не самого производительного тестового стенда: ReLU проще в вычислении по сравнению с сигмоидой или гиперболическим тангенсом (кандидаты на роль функции активации), так как ее производная равна 0 для отрицательных значений и 1 для положительных, что снижает нагрузку на вычисления и ускоряет процесс обучения, что подтверждено умеренным временем обучения (около 600 секунд для архитектуры 128–64–32, см. Рисунок 3.5.2.2). Комбинация ReLU с dropout усиливает регуляризацию, минимизируя риск переобучения на сбалансированных данных, полученных с помощью SMOTE, что особенно важно, так как искусственно сгенерированные примеры могут вводить шум, который ReLU помогает фильтровать, сохраняя только положительные активации. В завершение обоснования, на практике ReLU зарекомендовала себя как стандартный выбор для глубоких нейронных сетей в задачах классификации, включая обнаружение мошенничества, демонстрируя высокую производительность и низкую итоговую потерю (менее 0.005), что также видно на Рисунках 3.5.2.2-3.

Выбор архитектуры слоев обоснован экспериментальным сравнением четырех конфигураций: 1 скрытый слой (64 нейрона), 2 скрытых слоя (128–64), 3 скрытых слоя (128–64–32) и 4 скрытых слоя (256–128–64–32). Анализ проводился по метрикам ROC-AUC, итоговой функции потерь и времени обучения (см. Рисунки 3.5.2.2-3). Архитектура с тремя скрытыми слоями (128–64–32) оказалась оптимальной: она обеспечила низкую потерю (около 0.005) и приемлемое время обучения (около 600 секунд). Простая модель с одним слоем (64) продемонстрировала значительную потерю (около 0.025), что указывает на недостаточную выразительную способность для сложных зависимостей в данных. Добавление второго слоя (128–64) улучшило результаты (потеря ~0.01), но не достигло пика эффективности. Увеличение до четырех слоев (256–128–64–32) повысило время обучения до 800 секунд и не сильно улучшило потерю (около 0.004) по сравнению с конфигурацией 128-64-

32, что свидетельствует об избыточной глубине и риске переобучения для данного датасета.

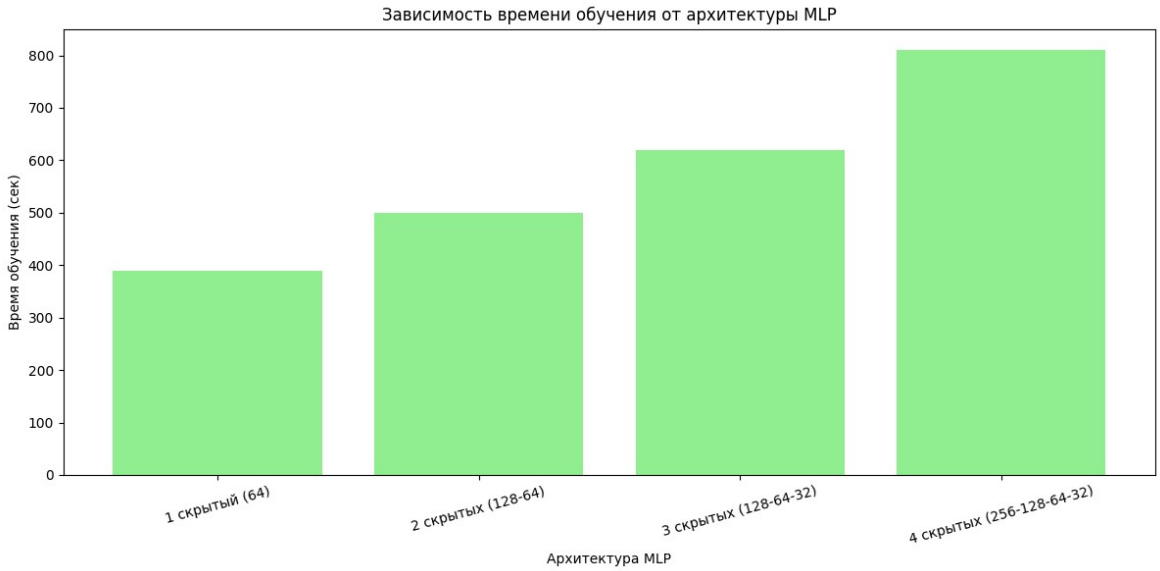

Рисунок 3.5.2.2 – Зависимость времени обучения от архитектуры MLP

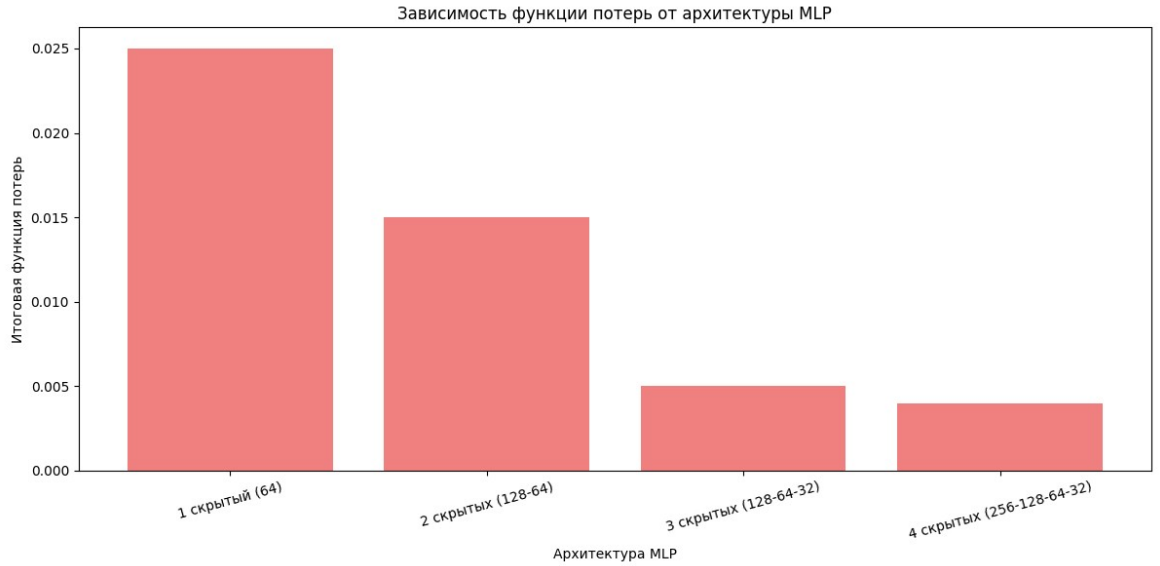

Рисунок 3.5.2.3 – Зависимость функции потерь от архитектуры MLP

Обучение проводилось в течение 100 эпох с функцией потерь — бинарной кросс-энтропией:

$$L(y,\hat{y})=\frac{-1}{N}\sum_{i=1}^{N}[y_i\cdot\log(\hat{y}_i)+(1-y_i)\cdot\log(1-\hat{y}_i)],$$

и оптимизатором Adam (скорость обучения 0.001). Adam адаптивно корректирует шаг оптимизации, используя моменты градиента, что ускоряет сходимость на задачах с переменной динамикой параметров.

Результаты на тестовой выборке:

- *Accuracy*: 0.9996 — доля правильных предсказаний.
- *Precision*: 0.9992 — точность выявления мошенничества.
- *Recall*: 1.0000 — полнота обнаружения.
- *F1-score*: 0.9996 — сбалансированная мера.
- *ROC-AUC*: 0.9998 — выдающаяся разделимость классов.

MLP показал лучшие результаты среди рассмотренных моделей благодаря способности к глубокому анализу данных.

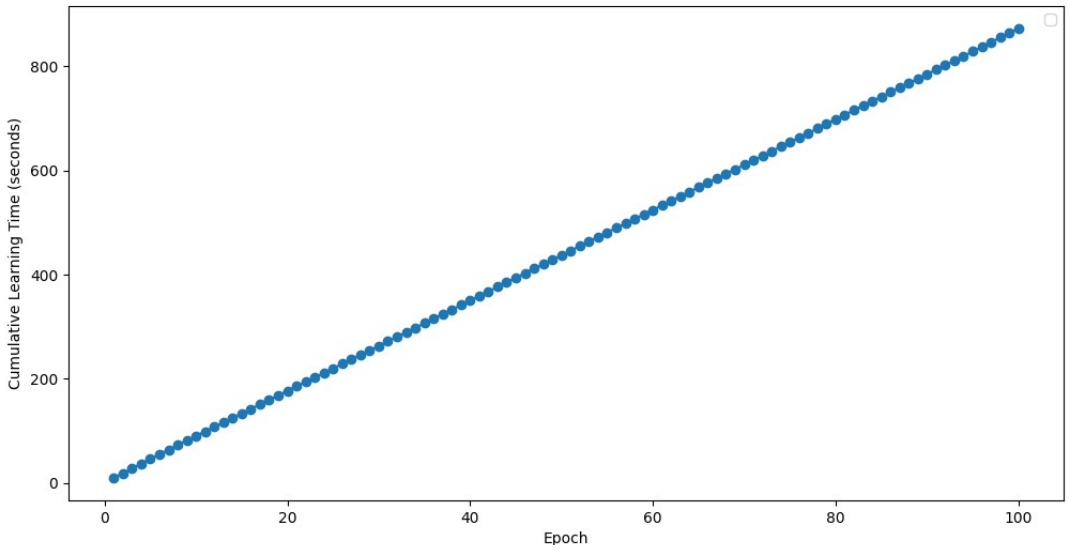

Рисунок 3.5.3 – Зависимость кумулятивного времени обучения от числа эпох

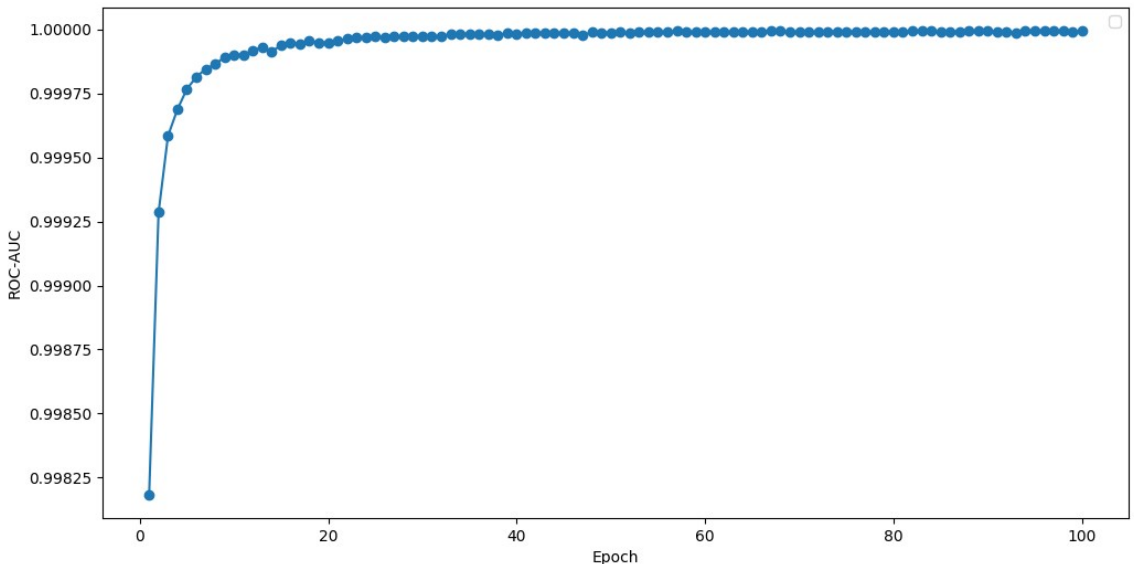

Рисунок 3.5.4 – Зависимость ROC-AUC от числа эпох

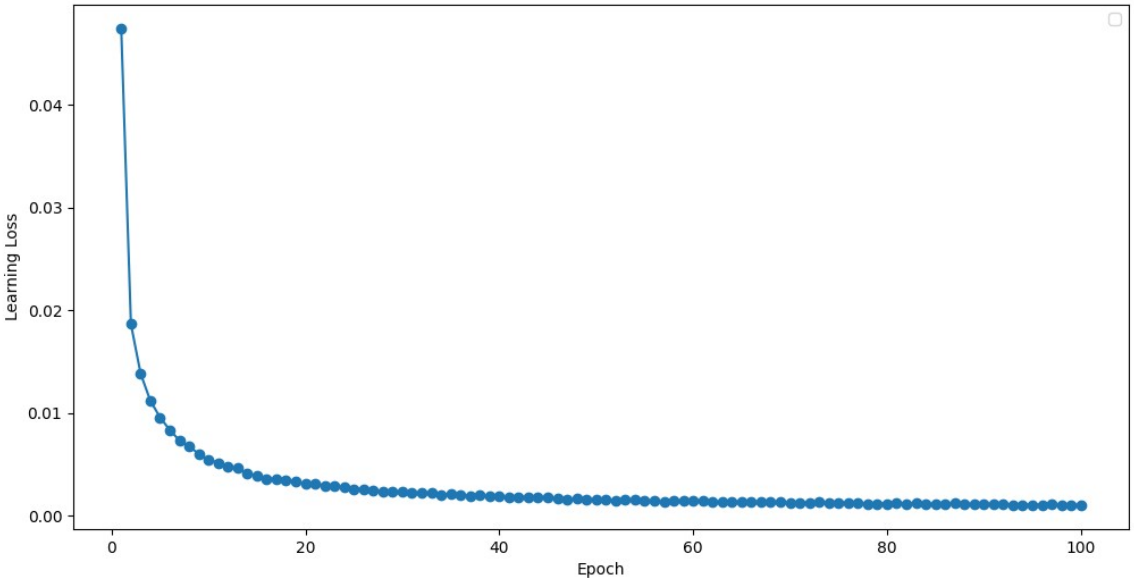

Рисунок 3.5.5 – Зависимость функции потерь от числа эпох

4 Сравнение полученных результатов

Для оценки эффективности моделей в задаче обнаружения мошеннических транзакций ключевой метрикой выбран ROC-AUC (площадь под кривой ROC) [10]. Эта метрика отражает способность модели разделять классы независимо от порога классификации, что критически важно для несбалансированных данных, где ложные пропуски (false negatives) могут иметь серьезные последствия. Ниже приведен сравнительный анализ преимуществ и недостатков каждой модели на основе их производительности и применимости.

4.1 Преимущества и недостатки каждой модели

Логистическая регрессия

ROC-AUC: 0.9848. Модель показала умеренную полноту (recall: 0.9046), что указывает на пропуск части мошеннических транзакций, но приемлемую общую разделимость классов. Ее линейная природа ограничивает способность улавливать сложные зависимости, однако обеспечивает высокую интерпретируемость и скорость обучения.

Вывод: Подходит для быстрого анализа и задач с линейными закономерностями, но уступает более сложным методам в точности на нелинейных данных.

Дерево решений

ROC-AUC: 0.9349. Высокая полнота (recall: 0.9235) компенсируется низкой точностью (precision: 0.9453), что свидетельствует о большом числе ложных срабатываний. Качество сильно зависит от глубины дерева (см. Рисунок 3.4), а склонность к переобучению снижает обобщающую способность.

Вывод: Обеспечивает интерпретируемость и базовый анализ данных, но требует ансамблевых методов для повышения устойчивости и качества.

Случайный лес

ROC-AUC: 0.9536. Модель демонстрирует высокий уровень обобщения и стабильности благодаря ансамблевому подходу, что подтверждается близким к 0.99 значением ROC-AUC (см. Рисунок 3.6). Однако обучение и предсказание замедляются при увеличении числа деревьев.

Вывод: Эффективна для задач, требующих устойчивости к шуму, но ограничена вычислительными затратами и сложностью интерпретации.

Градиентный бустинг

ROC-AUC: 0.9970. Модель достигла выдающихся результатов по всем метрикам (F1-score: 0.9971), обеспечивая точное разделение классов (см. Рисунок 3.8). Последовательная коррекция ошибок делает ее гибкой, но обучение требует значительного времени и тщательной настройки гиперпараметров.

Вывод: Оптимальна для задач с высокими требованиями к точности, несмотря на длительное обучение и чувствительность к параметрам.

Многослойный перцептрон (MLP)

ROC-AUC: 0.9998. MLP показал наилучшие результаты (recall: 1.0000, F1-score: 0.9996), эффективно моделируя нелинейные зависимости (см. Рисунок 3.11). Колебания качества на ранних эпохах стабилизировались благодаря dropout и оптимизации Adam.

Вывод: Идеально подходит для сложных задач с большим объемом данных, но требует значительных ресурсов и теряет в интерпретируемости.

4.2 Интерпретируемость моделей

Интерпретируемость моделей в задаче обнаружения мошеннических транзакций играет ключевую роль для объяснения предсказаний и поддержки принятия решений. Простые модели, такие как логистическая регрессия и дерево решений, обладают встроенной интерпретируемостью: веса в логистической регрессии указывают на вклад признаков, а структура дерева решений отражает пороговые значения и логику классификации. Для сложных

моделей (Random Forest, градиентный бустинг, MLP) применяются универсальные методы интерпретации, такие как SHAP и LIME [9].

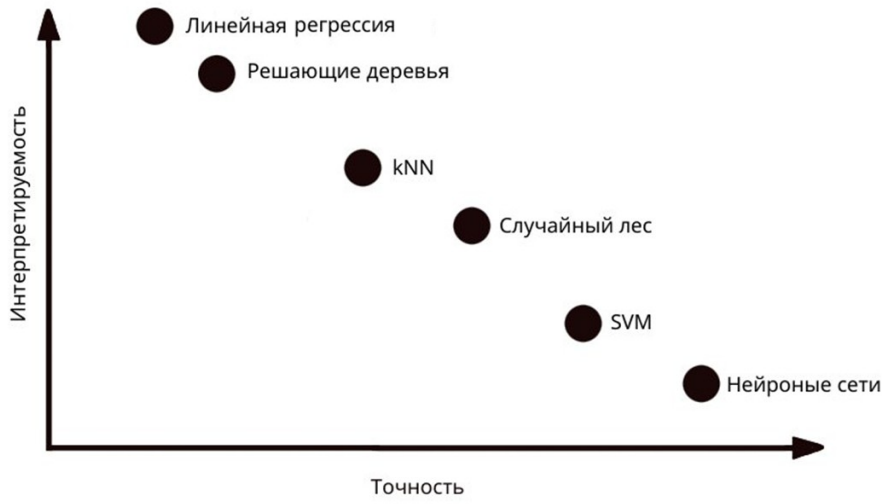

Рисунок 4.2.1 – Условная зависимость интерпретируемости моделей от точности их решений

4.2.1 Методы интерпретации

SHAP (SHapley Additive Explanations): Оценивает вклад каждого признака в предсказание на основе теории игр, обеспечивая глобальную и локальную интерпретацию.

LIME (Local Interpretable Model-agnostic Explanations): Объясняет предсказания путем аппроксимации сложной модели локально интерпретируемой функцией, выявляя ключевые зависимости для отдельных примеров.

4.2.2 Анализ интерпретируемости

Дерево решений (SHAP)

На графике SHAP (Рисунок 4.1) представлены вклады признаков в предсказание класса 1 (мошенничество): по вертикали — признаки, отсортированные по убыванию важности; по горизонтали — значения SHAP: положительные увеличивают вероятность мошенничества, отрицательные — снижают; цвет: красный (высокие значения признака), синий (низкие).

Результаты: Признак V14 имеет наибольший разброс SHAP-значений, указывая на его высокую важность: высокие значения (красные точки) значительно повышают вероятность мошенничества, V4 и V12 также влиятельны: низкие значения V4 (синие точки) склоняют модель к классу 1, тогда как V12 демонстрирует смешанное воздействие. Признаки V8, V26, V20 в том же графике оказывают умеренное влияние с меньшим разбросом.

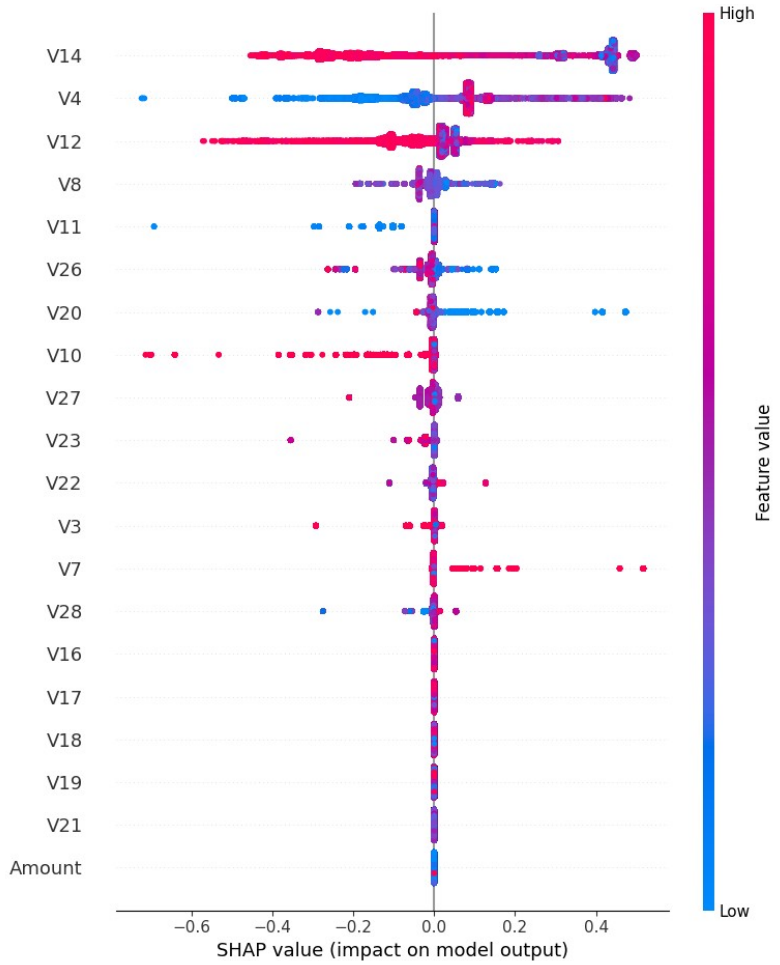

Рисунок 4.1. График SHAP для дерева решений: вклад признаков в предсказание мошенничества

Логистическая регрессия (SHAP)

Перейдем к графику зависимости SHAP-значений признака V14 от его величины с наложением V4 (Рисунок 4.2). Ось X здесь показывает значения признака V14, ось Y — SHAP-значения (положительные — за мошенничество,

отрицательные — против); цвет — значения V4 (синий — низкие, красный — высокие).

Результаты: При росте V14 SHAP-значения снижаются, указывая на уменьшение вероятности мошенничества при высоких значениях. Низкие значения V4 (синий) усиливают отрицательное влияние V14, тогда как высокие V4 (красный) смягчают этот эффект, демонстрируя корреляцию между признаками.

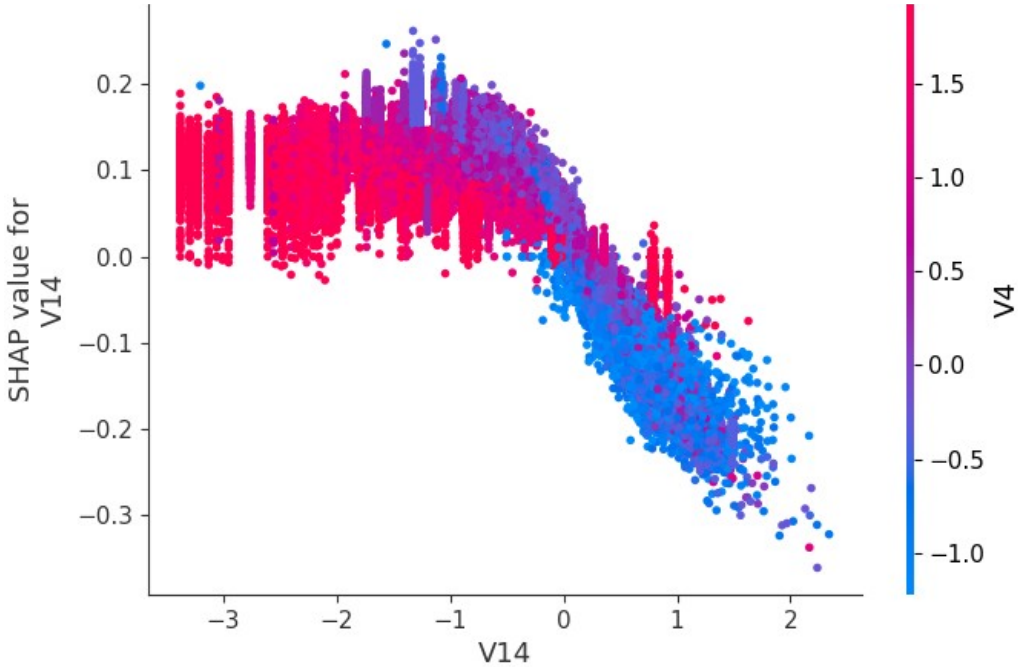

Рисунок 4.2. SHAP-зависимость для V14 в логистической регрессии с наложением V4

Многослойный перцептрон (LIME)

LIME-анализ для одного примера (Рисунок 4.3) показывает вклад признаков в предсказание следующим образом: слева расположены признаки с положительным (оранжевый) и отрицательным (синий) влиянием на класс "Fraud", справа – интервалы значений признаков и их воздействие на оба класса.

Результаты: V14 и V12 склоняют модель к "Non-Fraud" (отрицательный вклад), тогда как V8 (≤ -0.09) и V18 увеличивают вероятность "Fraud".

Интервал 0.50 < V14 ≤ 0.80 усиливает "Non-Fraud", демонстрируя локальную специфику влияния.

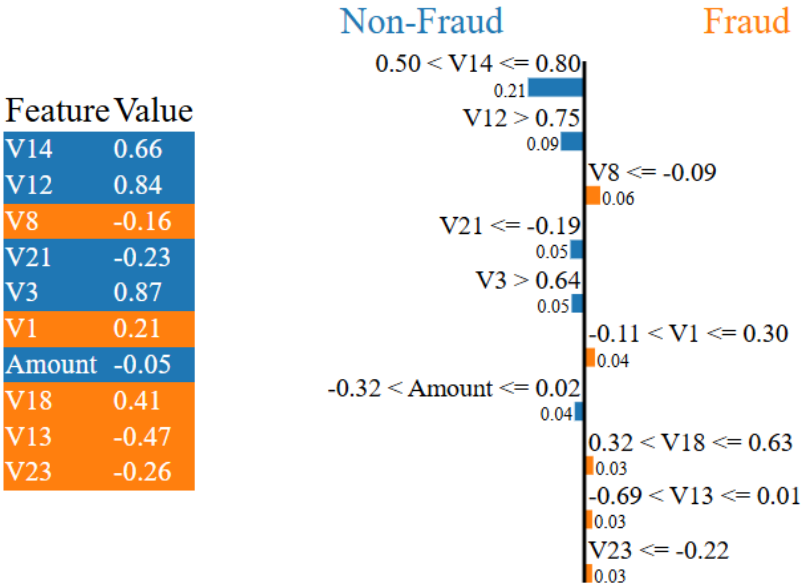

Рисунок 4.3 LIME-анализ для MLP: вклад признаков в предсказание

4.2.3 Выводы

Признаки V14, V4, V12, V8 выделяются как наиболее значимые для всех моделей, SHAP эффективен для глобального анализа (дерево решений, логистическая регрессия), LIME — для локального (MLP). Интерпретируемость позволяет не только обосновывать предсказания, но и выявлять аномалии для ручного анализа, что будет особенно важно в системах безопасности.

4.3 Влияние балансировки данных

Балансировка классов существенно влияет на качество моделей в задаче обнаружения мошеннических транзакций, где дисбаланс (99.83% нормальных против 0.17% мошеннических) может привести к игнорированию миноритарного класса [4]. Без коррекции модели оптимизируют общую точность (accuracy), недооценивая редкие случаи мошенничества, что недопустимо с точки зрения практической значимости.

В работе исследовались три подхода к балансировке:

Oversampling (SMOTE): Генерация синтетических примеров миноритарного класса улучшила ROC-AUC для всех моделей, особенно для

деревьев решений, случайного леса и градиентного бустинга. Это связано с увеличением числа обучающих примеров редкого класса, что позволило моделям лучше выделять его паттерны.

Undersampling: Уменьшение числа примеров мажоритарного класса показало снижение точности (precision) из-за потери части информации, что ограничило обобщающую способность моделей.

Перевзвешивание классов: Присвоение большего веса миноритарному классу в функции потерь также повысило чувствительность моделей, но уступило SMOTE по метрике precision, как видно на примере логистической регрессии (см. Рисунок 4.4).

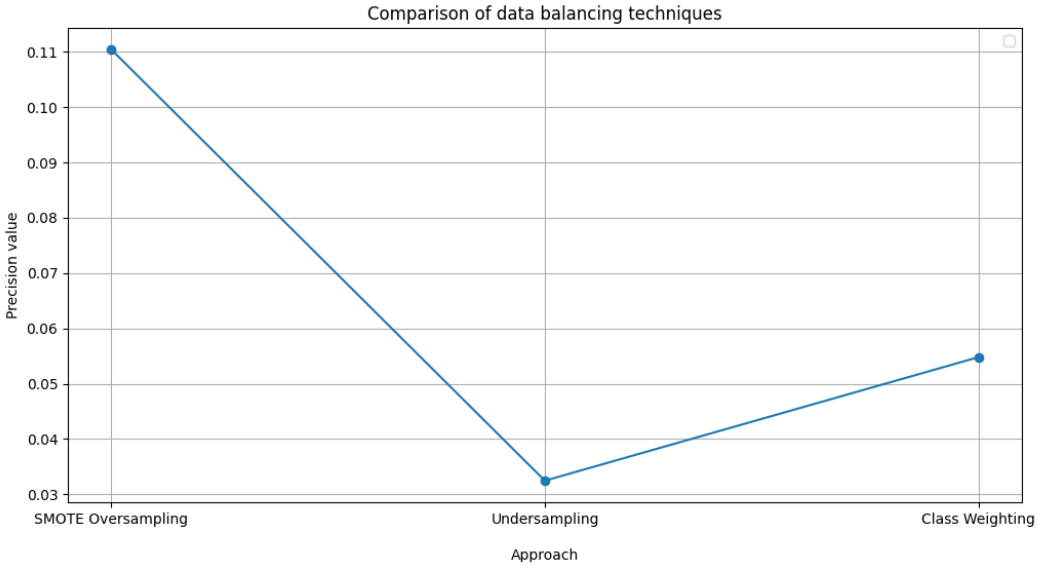

Рисунок 4.4. Зависимость Precision от метода балансировки для логистической регрессии

Метод SMOTE оказался наиболее эффективным, обеспечивая баланс между полнотой (recall) и точностью (precision), что критично для задач с дисбалансом. Undersampling менее предпочтителен из-за сокращения объема данных, а перевзвешивание классов служит компромиссным решением при ограниченных ресурсах.

ЗАКЛЮЧЕНИЕ

В рамках исследования проведен анализ и реализация моделей машинного обучения для обнаружения мошеннических транзакций с акцентом на их эффективность, интерпретируемость и применимость в реальных условиях. Работа включала сравнение методов классификации, оптимизацию гиперпараметров и применение техник балансировки данных для борьбы с выраженным дисбалансом классов.

Выводы

На основе метрик ROC-AUC, F1-score, Precision и Recall лучшие результаты продемонстрировал Многослойный перцептрон, обеспечивая высокую точность и обобщающую способность даже в условиях несбалансированных данных. Метод oversampling с использованием SMOTE значительно повысил производительность всех моделей, особенно логистической регрессии и деревьев решений, улучшив их чувствительность к редкому классу мошенничества. Градиентный бустинг (ROC-AUC: 0.997) также показал выдающиеся результаты. Полученные модели применимы в системах мониторинга транзакций для минимизации финансовых потерь, хотя их внедрение требует учета вычислительных ограничений.

Ограничения исследования

Зависимость от данных: Модели тестировались на одном датасете (*Credit Card Fraud Detection*), что может ограничить их обобщение на данные с иными характеристиками.

Синтетические данные: Использование SMOTE, несмотря на эффективность, вводит искусственные примеры, потенциально искажающие реальное распределение.

Охват методов: Основное внимание уделено классическим и ансамблевым алгоритмам; современные нейронные сети рассмотрены ограниченно.

Интерпретируемость: Сложные модели (градиентный бустинг, MLP) остаются менее «прозрачными», несмотря на применение SHAP и LIME.

Рекомендации для дальнейших исследований

1. Провести тестирование на дополнительных датасетах для оценки универсальности моделей.
2. Исследовать современные нейронные сети (например, RNN, CNN) для анализа временных зависимостей в транзакциях [12].
3. Разработать ансамбли (стекинг, блендинг) для повышения качества классификации.
4. Интегрировать временные признаки и модели (LSTM, Transformer) для учета последовательностей операций [3].
5. Оптимизировать модели под реальные условия, минимизируя вычислительные затраты и ложные срабатывания, а также обеспечивая их внедрение в промышленные системы.

Исследование подтвердило возможность точного обнаружения мошенничества с использованием продвинутых методов классификации и балансировки данных. Результаты могут служить основой для систем мониторинга транзакций, однако дальнейшее развитие требует адаптации к реальным ограничениям и расширения методологии для достижения максимальной эффективности.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В пояснительной записке применяются следующие сокращения и обозначения:

MLP – Multi-Layer Perceptron (многослойный перцептрон)

SMOTE – Synthetic Minority Oversampling Technique (метод синтетического увеличения миноритарного класса)

SHAP – SHapley Additive Explanations (метод интерпретации моделей на основе теории игр)

LIME – Local Interpretable Model-agnostic Explanations (метод локальной интерпретации моделей)

ROC-AUC – Receiver Operating Characteristic – Area Under Curve (площадь под кривой ROC, метрика качества классификации)

F1-score – мера гармонического среднего между точностью и полнотой

Precision – точность (доля верно классифицированных положительных примеров)

Recall – полнота (доля найденных положительных примеров)

Accuracy – точность (доля верно классифицированных примеров)

ReLU – Rectified Linear Unit (функция активации, $f(x)=\max(0,x)$)

BatchNorm – Batch Normalization (метод нормализации входных данных в слоях нейронной сети)

Dropout – метод регуляризации в нейронных сетях, отключающий случайные нейроны во время обучения

Adam – Adaptive Moment Estimation (оптимизатор, использующий адаптивные моменты градиента)

CNN – Convolutional Neural Network (сверточная нейронная сеть)

RNN – Recurrent Neural Network (рекуррентная нейронная сеть)

LSTM – Long Short-Term Memory (долгая краткосрочная память, тип рекуррентной нейронной сети)

XGBoost – Extreme Gradient Boosting (метод градиентного бустинга)

η – скорость обучения (learning rate)

$L(y, \hat{y})$ – функция потерь (в данном случае бинарная кросс-энтропия)

p_k – доля объектов класса k в узле дерева решений (для критерия Джини)

K – общее число классов (в задаче $K = 2$)

n – количество деревьев в ансамбле (для случайного леса и градиентного бустинга)

$T_i(x)$ – предсказание i-го дерева в ансамбле

$Gini$ – критерий Джини для оценки качества разбиения в дереве решений

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Хэсти Т., Тибширани Р., Фридман Дж. Элементы статистического обучения: данные, выводы и прогнозы. – М.: Мир, 2017. – 745 с.

2 Рассел С., Норвиг П. Искуственный интеллект: современный подход. – 4-е изд. – М.: Вильямс, 2021. – 1136 с.

3 Саттон Р., Барто Э. Обучение с подкреплением. – 2-е изд. – М.: ДМК Пресс, 2018. – 528 с.

4 Виттен И. Х., Франк Э., Холл М. А. Data Mining: Практическое руководство по анализу данных. – 3-е изд. – М.: ДМК Пресс, 2011. – 664 с.

5 Нильсон Н. Дж. Введение в машинное обучение. – М.: Мир, 1998. – 536 с.

6 Траск Э. Грокаем глубокое обучение. – СПб.: Питер, 2019. – 384 с.

7 Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение. 2017. URL: http://deeplearningbook.org (Дата обращения 08.02.2025)

8 Шолле Ф. Глубокое обучение на Python. 2018. URL: http://manning.com/books/deep-learning-with-python (Дата обращения 08.02.2025)

9 Жерон О. Прикладное машинное обучение с помощью SciKit-Learn и TensorFlow. 2018. URL: http://oreilly.com/library/view/hands-on-machine-learning/9781492032632 (Дата обращения 09.02.2025)

10 Абу-Мостафа Я., Магдон-Исмаил М., Линь С.-Т. Learning From Data. 2012. URL: http://amlbook.com (Дата обращения 09.02.2025)

11 Бурков А. The Hundred-Page Machine Learning Book. 2019. URL: http://themlbook.com (Дата обращения 10.02.2025)

12 Лапань М. Deep Reinforcement Learning Hands-On. 2018. URL: http://packtpub.com/product/deep-reinforcement-learning-hands-on/9781788834247 (Дата обращения 11.02.2025)

ПРИЛОЖЕНИЕ А

Листинг программы

Листинг А.1 — Программа логистической регрессии

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score
import matplotlib.pyplot as plt
import time

# Загрузка данных
data = pd.read_csv('./data/creditcard.csv')

# Проверка на пропущенные значения
if data.isnull().values.any():
    data.fillna(data.mean(), inplace=True)

# Балансировка классов с помощью oversampling для класса 1
data_class_0 = data[data['Class'] == 0]
data_class_1 = data[data['Class'] == 1]
data_class_1_oversampled = resample(data_class_1, replace=True,
n_samples=len(data_class_0), random_state=42)
data_balanced = pd.concat([data_class_0,
data_class_1_oversampled])

# Масштабируем все признаки
scaler = StandardScaler()
X = scaler.fit_transform(data_balanced.drop('Class', axis=1))
y = data_balanced['Class'].values

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Инициализация весов и параметров модели
np.random.seed(0)
num_features = X_train.shape[1]
weights = np.random.uniform(-0.01, 0.01, num_features)
bias = 0.0
learning_rate = 0.1

# Сигмоидная функция
def sigmoid(z):
    return 1 / (1 + np.exp(-np.clip(z, -500, 500)))
```

```
# Функция потерь
def compute_loss(y, y_hat, class_weights, epsilon=1e-9):
    y_hat = np.clip(y_hat, epsilon, 1 - epsilon)
    weight = np.where(y == 1, class_weights[1], class_weights[0])
     return -np.mean(weight * (y * np.log(y_hat) + (1 - y) *
np.log(1 - y_hat)))

# Прямой проход
def predict(X, weights, bias):
    z = np.dot(X, weights) + bias
    return sigmoid(z)

# Оценка модели
def evaluate_model(X, y, weights, bias):
    y_pred = predict(X, weights, bias) >= 0.5
    accuracy = accuracy_score(y, y_pred)
    precision = precision_score(y, y_pred, zero_division=1)
    recall = recall_score(y, y_pred, zero_division=1)
    f1 = f1_score(y, y_pred, zero_division=1)
    roc_auc = roc_auc_score(y, predict(X, weights, bias))

    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print(f"ROC-AUC: {roc_auc:.4f}")

# Установка весов классов
class_weights    =    {0:    1,    1:    len(data_class_0)    /
len(data_class_1_oversampled)}

def train_logistic_regression(X, y, weights, bias, learning_rate,
num_epochs, class_weights, tol=0.01, patience=10):
    no_improve_count = 0
    best_loss = float('inf')
    times = []  # Время каждой эпохи
    cumulative_times = []  # Накопленное время
    roc_auc_scores = []  # ROC-AUC метрики
    total_time = 0  # Накопленное время

    for epoch in range(num_epochs):
        start_time = time.time()  # Начало замера времени

        # Прогноз и вычисление ошибки
        y_hat = predict(X, weights, bias)
        error = y_hat - y

        # Градиенты
        dW = np.dot(X.T, error) / len(y)
```

```
        dB = np.sum(error) / len(y)

        # Обновление весов и смещения
        weights -= learning_rate * dW
        bias -= learning_rate * dB

        # Замер времени
        end_time = time.time()
        epoch_time = end_time - start_time
        total_time += epoch_time
        times.append(epoch_time)
        cumulative_times.append(total_time)

        # Вычисление метрики ROC-AUC каждые 10 эпох
        if epoch % 10 == 0:
            loss = compute_loss(y, y_hat, class_weights)
            roc_auc = roc_auc_score(y, y_hat)
            roc_auc_scores.append(roc_auc)

              print(f"Epoch {epoch}, Loss: {loss:.4f}, ROC-AUC:
{roc_auc:.4f}")

            # Раняя остановка
            if loss < best_loss - tol:
                best_loss = loss
                no_improve_count = 0
            else:
                no_improve_count += 1

            if no_improve_count >= patience:
                 print(f"Early stopping at epoch {epoch} due to
minimal loss improvement.")
                break

    return weights, bias, times, cumulative_times, roc_auc_scores

# Обучение модели
weights,  bias,  times,  cumulative_times,  roc_auc_scores  =
train_logistic_regression(
        X_train,  y_train,  weights,  bias,  learning_rate,
num_epochs=2000, class_weights=class_weights
)

# Оценка на тестовой выборке
evaluate_model(X_test, y_test, weights, bias)

# График ROC-AUC
plt.figure(figsize=(10, 6))
plt.plot(range(0, len(roc_auc_scores) * 10, 10), roc_auc_scores,
marker='o')
```

```
#plt.yscale('log')  # Логарифмическая шкала
plt.xlabel('Epoch')
plt.ylabel('ROC-AUC')
plt.legend()
plt.grid()
plt.show()

# График накопленного времени
plt.figure(figsize=(10, 6))
plt.plot(range(len(cumulative_times)),         cumulative_times,
marker='o')
plt.xlabel('Epoch')
plt.ylabel('Cumulative Learning Time (seconds)')
plt.legend()
plt.grid()
plt.show()
```

Листинг А.2 — Программа дерева решений

```
# Узел дерева решений
class DecisionNode:
      def  __init__(self,  feature_index=None,  threshold=None,
left=None, right=None, *, value=None):
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

# Критерий разбиения
def gini(y):
    classes = np.unique(y)
     return 1.0 - sum((np.sum(y == c) / len(y)) ** 2 for c in
classes)

# Разбиение
def split_dataset(X, y, feature_index, threshold):
    left_indices = np.where(X[:, feature_index] <= threshold)
    right_indices = np.where(X[:, feature_index] > threshold)
     return X[left_indices], X[right_indices], y[left_indices],
y[right_indices]

# Измерение времени выполнения для лучшего разбиения
def  best_split(X,  y,  criterion='gini',  num_thresholds=10,
num_features=None):
    best_feature, best_threshold, best_gain = None, None, -1
    n_samples, n_features = X.shape
    parent_impurity = gini(y)
```

```
      features = np.random.choice(n_features, num_features or
n_features, replace=False)

    for feature_index in features:
        thresholds = np.unique(X[:, feature_index])
        if len(thresholds) > num_thresholds:
                    thresholds = np.random.choice(thresholds,
num_thresholds, replace=False)

        for threshold in thresholds:
           X_left, X_right, y_left, y_right = split_dataset(X, y,
feature_index, threshold)
            if len(y_left) > 0 and len(y_right) > 0:
                      p_left, p_right = len(y_left) / len(y),
len(y_right) / len(y)
                   impurity = p_left * gini(y_left) + p_right *
gini(y_right)
                gain = parent_impurity - impurity
                if gain > best_gain:
                     best_feature, best_threshold, best_gain =
feature_index, threshold, gain

    return best_feature, best_threshold

# Построение дерева

def build_tree_with_timing(X, y, depth=0, max_depth=5,
min_samples_split=10, criterion='gini', timing_data=None):
    start_time = time.time()

    if len(np.unique(y)) == 1:
        elapsed_time = time.time() - start_time
        if timing_data is not None:
            timing_data.append((depth, elapsed_time))
        return DecisionNode(value=np.unique(y)[0])

    if depth >= max_depth or len(y) < min_samples_split:
        elapsed_time = time.time() - start_time
        if timing_data is not None:
            timing_data.append((depth, elapsed_time))
        return DecisionNode(value=np.bincount(y).argmax())

      feature_index, threshold = best_split(X, y, criterion,
num_thresholds=10, num_features=5)
    if feature_index is None:
        elapsed_time = time.time() - start_time
        if timing_data is not None:
            timing_data.append((depth, elapsed_time))
        return DecisionNode(value=np.bincount(y).argmax())
```

```
    X_left, X_right, y_left, y_right = split_dataset(X, y,
feature_index, threshold)

    left_subtree = build_tree_with_timing(X_left, y_left, depth +
1, max_depth, min_samples_split, criterion, timing_data)
    right_subtree = build_tree_with_timing(X_right, y_right,
depth + 1, max_depth, min_samples_split, criterion, timing_data)

    elapsed_time = time.time() - start_time
    if timing_data is not None:
        timing_data.append((depth, elapsed_time))

    return DecisionNode(feature_index, threshold, left_subtree,
right_subtree)

# Предсказание
def predict_tree(node, sample):
    if node.value is not None:
        return node.value
    if sample[node.feature_index] <= node.threshold:
        return predict_tree(node.left, sample)
    else:
        return predict_tree(node.right, sample)

def predict(X, tree):
    return np.array([predict_tree(tree, sample) for sample in X])
```

Листинг А.3 — Программа случайного леса

```
class RandomForest:
        def __init__(self, n_estimators=10, max_depth=None,
min_samples_split=2, max_features=None):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.max_features = max_features
        self.trees = []

    def fit(self, X, y):
        self.trees = []
        n_samples = X.shape[0]
        for _ in range(self.n_estimators):
                indices = np.random.choice(n_samples, n_samples,
replace=True)
                tree = DecisionTree(max_depth=self.max_depth,
min_samples_split=self.min_samples_split)
            tree.fit(X[indices], y[indices])
            self.trees.append(tree)
```

```
    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree in
self.trees])
        return np.round(np.mean(tree_preds, axis=0))
```

Листинг А.4 — Программа градиентного бустинга

```
class SimpleDecisionTree:
    def __init__(self, max_depth=3, min_samples_split=10,
max_thresholds=50):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.max_thresholds = max_thresholds  # Максимальное
количество порогов для разбиения
        self.tree = None

    def fit(self, X, y, depth=0):
        print(f"Tree depth {depth}: fitting {len(y)} samples.")

        # Условия остановки
        if depth >= self.max_depth or len(set(y)) == 1 or len(y) <
self.min_samples_split:
            self.tree = np.mean(y)
            print(f"Stopping condition met at depth {depth}. Node
prediction: {self.tree}")
            return

        # Поиск лучшего разбиения
        best_mse, best_idx, best_thr = float('inf'), None, None
        for i in range(X.shape[1]):
            unique_thresholds = np.unique(X[:, i])
            if len(unique_thresholds) > self.max_thresholds:
                    # Ограничиваем количество порогов, выбирая
равномерно распределенные
                                        unique_thresholds    =
np.linspace(min(unique_thresholds),      max(unique_thresholds),
self.max_thresholds)

            print(f"Feature {i}: testing {len(unique_thresholds)}
thresholds.")

            for thr in unique_thresholds:
                left_mask = X[:, i] <= thr
                right_mask = X[:, i] > thr
                if np.sum(left_mask) == 0 or np.sum(right_mask) ==
0:
                    continue
```

```
                mse = (
                    self._mse(y[left_mask]) * np.sum(left_mask)
                  + self._mse(y[right_mask]) * np.sum(right_mask)
                )
                if mse < best_mse:
                    best_mse, best_idx, best_thr = mse, i, thr

         if best_idx is None:  # Если подходящего разбиения не
найдено
            self.tree = np.mean(y)
             print(f"No valid split found at depth {depth}. Node
prediction: {self.tree}")
            return

           print(f"Best  split:  Feature  {best_idx},  Threshold
{best_thr}, MSE {best_mse:.4f}")

        # Сохраняем разбиение
        self.tree = {
            'index': best_idx,
            'threshold': best_thr,
                     'left':  SimpleDecisionTree(self.max_depth,
self.min_samples_split, self.max_thresholds),
                    'right':  SimpleDecisionTree(self.max_depth,
self.min_samples_split, self.max_thresholds),
        }

        left_mask = X[:, best_idx] <= best_thr
        right_mask = X[:, best_idx] > best_thr
        print(f"Splitting: {np.sum(left_mask)} samples go to the
left, {np.sum(right_mask)} to the right.")

       self.tree['left'].fit(X[left_mask], y[left_mask], depth +
1)
           self.tree['right'].fit(X[right_mask],  y[right_mask],
depth + 1)

    def _mse(self, y):
        if len(y) == 0:
            return 0
        return np.mean((y - np.mean(y)) ** 2)

    def predict_single(self, x):
        if not isinstance(self.tree, dict):
            return self.tree
        if x[self.tree['index']] <= self.tree['threshold']:
            return self.tree['left'].predict_single(x)
        return self.tree['right'].predict_single(x)

    def predict(self, X):
```

```
        return np.array([self.predict_single(x) for x in X])


class GradientBoosting:
      def __init__(self, n_estimators=10, learning_rate=0.1,
max_depth=10, min_samples_split=10, max_thresholds=50):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.max_thresholds = max_thresholds
        self.trees = []
        self.init_prediction = None
         self.training_times = []  # Время на обучение каждого
дерева
       self.depth_metrics = []  # Метрики в зависимости от глубины
        self.depth_times = []  # Суммарное время в зависимости от
глубины

    def fit(self, X, y):
        self.trees = []
        self.init_prediction = np.mean(y)
        residuals = y - self.init_prediction
        cumulative_time = 0  # Накопительное время обучения

        for i in range(self.n_estimators):
            start_time = time.time()
             tree = SimpleDecisionTree(max_depth=self.max_depth,
min_samples_split=self.min_samples_split,

max_thresholds=self.max_thresholds)
            tree.fit(X, residuals)
            predictions = tree.predict(X)
            residuals -= self.learning_rate * predictions
            elapsed_time = time.time() - start_time
            cumulative_time += elapsed_time

            # Логируем данные
            self.training_times.append(elapsed_time)
            self.trees.append(tree)

        # Оценка метрик для всех глубин
        for depth in range(1, self.max_depth + 1):
                temp_tree = SimpleDecisionTree(max_depth=depth,
min_samples_split=self.min_samples_split,

max_thresholds=self.max_thresholds)
            start_time = time.time()
            temp_tree.fit(X, residuals)
            elapsed_time = time.time() - start_time
```

```
            predictions = temp_tree.predict(X)
                                roc_auc    =    roc_auc_score(y,
np.round(self.init_prediction     +     self.learning_rate     *
predictions))
                f1  =  f1_score(y,  np.round(self.init_prediction  +
self.learning_rate * predictions))

            self.depth_metrics.append((depth, roc_auc, f1))
            self.depth_times.append((depth, elapsed_time))

    def predict(self, X):
        predictions = np.full(X.shape[0], self.init_prediction)
        for tree in self.trees:
            predictions += self.learning_rate * tree.predict(X)
        return np.round(predictions).astype(int)
```

Листинг А.5 — Программа многослойного перцептрона

```
# Определение многослойного перцептрона
class MLP(nn.Module):
    def __init__(self, input_size):
        super(MLP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_size, 128),
           nn.BatchNorm1d(128),  # Нормализация для стабилизации
            nn.ReLU(),
            nn.Dropout(0.2),

            nn.Linear(128, 64),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.Dropout(0.2),

            nn.Linear(64, 32),
            nn.BatchNorm1d(32),
            nn.ReLU(),

            nn.Linear(32, 1),
            nn.Sigmoid()  # Для предсказания вероятностей
        )

    def forward(self, x):
        return self.model(x)

# Инициализация модели
input_size = X_train.shape[1]
model = MLP(input_size)

# Функция потерь и оптимизатор
```

```
criterion = nn.BCELoss()
optimizer     =     optim.AdamW(model.parameters(),     lr=1e-4,
weight_decay=1e-4)  # AdamW для лучшей регуляризации

# Сборка результатов
num_epochs = 100
accumulation_steps = 4  # Накопление градиентов для уменьшения
шума

train_losses = []
val_roc_auc = []
train_time = []
best_roc_auc = 0
early_stop_counter = 0

# Цикл обучения
for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0.0
    epoch_start = time.time()

    optimizer.zero_grad()
    for step, (X_batch, y_batch) in enumerate(train_loader):
        y_pred = model(X_batch).squeeze()
        loss = criterion(y_pred, y_batch)
          loss = loss / accumulation_steps  # Делим loss на
количество шагов для накопления
        loss.backward()

        if (step + 1) % accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()

        epoch_loss += loss.item()

    train_losses.append(epoch_loss / len(train_loader))
    epoch_end = time.time()
    train_time.append(epoch_end - epoch_start)

    # Оценка на валидации
    model.eval()
    y_test_pred = []
    with torch.no_grad():
        for X_batch, _ in test_loader:
            y_batch_pred = model(X_batch).squeeze()
            y_test_pred.extend(y_batch_pred.tolist())

    # Расчет ROC-AUC
    roc_auc = roc_auc_score(y_test, y_test_pred)
    val_roc_auc.append(roc_auc)
```

```
        print(f"Epoch {epoch + 1}/{num_epochs}, Loss:
{train_losses[-1]:.4f}, ROC-AUC: {roc_auc:.4f}")

    # Early Stopping
    if roc_auc > best_roc_auc:
        best_roc_auc = roc_auc
        early_stop_counter = 0
        best_model = model.state_dict()
    else:
        early_stop_counter += 1

    if early_stop_counter >= 30:
        print("Early stopping triggered!")
        break
```