GEEKBRAINS

ДИПЛОМНАЯ РАБОТА

"Разработка и внедрение модели нейронной сети для обнаружения мошенничества в финансовых транзакциях"

Программа:
Data Science. Специалист
Автор - Серов В.В.

Новосибирск

2024

Введение

Тема проекта: Разработка и внедрение модели нейронной сети для обнаружения мошенничества в финансовых транзакциях

Цель: Разработать и внедрить модель нейронной сети для эффективного обнаружения случаев мошенничества в финансовых транзакциях, обеспечивающую высокий уровень точности, минимизацию ложных срабатываний и улучшение общей безопасности финансовых систем.

Какую проблему решает: Решает проблему, заключающуюся в опережающем подходе к анализу и выявлению фактов мошенничества в некоторых операциях. Основные выводы, как и многие другие, показывают, что существующие контрольные меры не «срабатывают» и не позволяют избежать масштабов мошенничества и воздействия на доверие к финансовым учреждениям.

Задачи:

1. Изучить литературу, касающуюся темы исследования.
2. Изучить основные инструменты и методы для разведочного анализа данных
3. Изучить инструменты для создания, обучения, тестирования модели нейронной сети.
4. Найти на просторах интернета подходящий для анализа и обучения модели датасет
5. Загрузить, очистить и проанализировать датасет

6. Создать модель нейронной сети
7. Обучить модель на преобразованных тренировочных данных
8. Протестировать модель на преобразованных тестовых данных
9. Сформировать собственные примеры транзакций и протестировать их

Инструменты:

- Google Colab: облачная платформа для разработке на языке Python, предоставляющая доступ к GPU/TPU для машинного обучения и анализа данных.

- OpenML: сайт с наборами данных(датасеты).

- Scikit-learn: библиотека для машинного обучения в Python, предлагающая широкий спектр алгоритмов для классификации, регрессии, кластеризации и предобработки данных.

- Pandas: библиотека для работы с табличными данными, предоставляющая инструменты для их обработки, анализа и манипуляции.

- NumPy: библиотека для выполнения численных вычислений, работающая с многомерными массивами и предоставляющая множество математических функций.

- PySpark: интерфейс для работы с Apache Spark, предназначенный для распределенной обработки больших данных и выполнения вычислений.

- Matplotlib и Seaborn: библиотеки для визуализации данных, позволяющие создавать графики, диаграммы и другие виды визуального представления информации.

- SMOTE: метод увеличения выборки данных, предназначенный для борьбы с дисбалансом классов путем синтетического создания новых примеров меньшинства, на основе уже существующих в наборе данных.

- PyTorch: библиотека для создания и обучения нейронных сетей, предоставляющая гибкий интерфейс для разработки глубокого обучения.

Глава 1. Теоретическая часть

1.1Обзор предметной области

1.1.1 Понятие финансовых транзакций и виды мошеннической деятельности.

В нынешнее время можно часто услышать из различных сфер слово “транзакция”. Деловые транзакции, технические транзакции, информационные транзакции, социальные транзакции, криптовалютные транзакции, финансовые транзакции и другие, однако в этой работе речь будет идти именно о финансовых транзакциях.

Финансовые транзакции - в общем случае, это любая сделка, с использованием банковского счета.

Но почему же эти транзакции так интересны для анализа? Всё очень просто: часть транзакций является транзакциями, произведенными мошенниками.
Мошеннические транзакции - это незаконное совершение финансовой транзакции мошенником, в случае, когда мошенник получил данные карты или банковского счета нелегальным путем.
Мошенник - человек, который каким-либо нелегальным путём получает доступ к банковским реквизитам другого человека.

Существует несколько видов мошеннической деятельности:

- Кража кредитных карт(использование украденных кредитных карт для совершения покупок)

- Несанкционированный доступ к банковским счетам(получения доступа к банковским счетам, с целью последующей оплаты покупок, услуг, в общем, своих нужд)
- Мошенничество с переводами(подделка платежных поручений или запросов на перевод)
- Создание поддельных счетов(создание фиктивных аккаунтов для получения бонусов, скидок или других преимуществ)
- Фишинг(получение доступа к аккаунтам, данных кредитных карт и прочему за счет перехода жертвы на специально подготовленные мошенником подозрительные сайты)

1.1.2 Методы и алгоритмы обнаружения мошенничества: традиционные подходы и современные методы на основе ИИ.

Обнаружение мошенничества - сложная задача, которая включает в себя применение различных методов, инструментов и подходов для выявления подозрительных и мошеннических действий, и в зависимости от используемой технологии, можно выделить традиционные методы и современные методы на основе искусственного интеллекта для обнаружения признаков мошенничества.

Традиционные методы:

- Принципы и эвристики:

  Этот подход основан на наборе правил, которые заранее определены для того, чтобы помочь выявить подозрительные сделки. Например, если транзакция совершается в количестве, превышающем определенный порог, или если транзакция происходит в необычное время. Тем не менее, такие методы недостаточно гибки для выявления сложных схем мошенничества.

- Исследование временных рядов:

  Использовать статистические методы для анализа временных данных, таких как транзакции. Паттерны поведения могут указывать на мошенничество. Это хорошо работает для выявления фальшивых транзакций, таких как частые и короткие мелкие переводы.

- Опорный векторный метод, также известный как SVM:

  Этот алгоритм широко используется для классификации транзакций как нормальных или мошеннических, хорошо работает с линейно-разделимыми данными и находит идеальную гиперплоскость, разделяющую классы.

Современные методы:

- Нейронные системы:

  Современные нейронные сети, такие как сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN) и многослойные перцептроны (MLP), обладают способностью обрабатывать большие объемы данных, обнаруживать сложные, нелинейные зависимости, хорошо справляются с текстовыми, числовыми и временными данными.

- Генеративные модели, такие как автоэнкодеры:

  Эти модели используются для реконструкции нормальных данных для выявления аномалий. Это может свидетельствовать о мошенничестве, если реконструкция значительно отличается от исходных данных. Особенно полезны для выявления новых или скрытых видов мошенничества, которые не были заранее запрограммированы.

- Методы группировки, такие как Random Forest и XGBoost:
  Эти алгоритмы улучшают точность и устойчивость классификации, принимая решения с помощью нескольких моделей. Например, случайный лес может эффективно работать с большими и сложными данными, что снижает риск переобучения и повышает точность распознавания мошеннических действий.

- Алгоритмы кластеризации (например, DBSCAN):
  Используются для выявления групп транзакций с похожими характеристиками.

- Методы на основе глубокого обучения (например, LSTM):
  Применяются для анализа временных рядов, таких как транзакции, происходящие с течением времени. LSTM (долгосрочная краткосрочная память) может эффективно выявлять скрытые зависимости в данных о транзакциях и обнаруживать подозрительные активности.

1.1.3 Преимущества нейронных сетей для анализа данных в финансовой сфере.

В ряде случаев использование современных методов, а именно нейронных сетей, имеет большие преимущества над традиционными методами анализа данных в финансовой сфере.

Нейронные сети очень хорошо себя показывают в определении скрытых зависимостей в данных. В отличие от традиционных методов, они не используют определенные паттерны, они находят новые зависимости.

Нейронную сеть можно дообучить, а вот с традиционными методами это не так легко.

Также нейронные сети очень хорошо справляются с большими объёмами данных, что не скажешь о традиционных методах

Нейронные сети могут предсказывать данные, чего традиционные методы не умеют делать.

Нейронные сети могут автоматически извлекать признаки из данных, что облегчает жизнь аналитику данных. В традиционных методах, определение параметров - задача аналитика.

Нейронные сети обладают множеством преимуществ для анализа данных в финансовой сфере, включая высокую точность, способность работать с большими объемами данных, автоматическое извлечение признаков и адаптивность к изменениям. Эти характеристики делают нейронные сети эффективным инструментом для решения сложных задач, таких как выявление мошенничества.

1.2 Нейронные сети и их применение для анализа

1.2.1 Основы искусственных нейронных сетей.

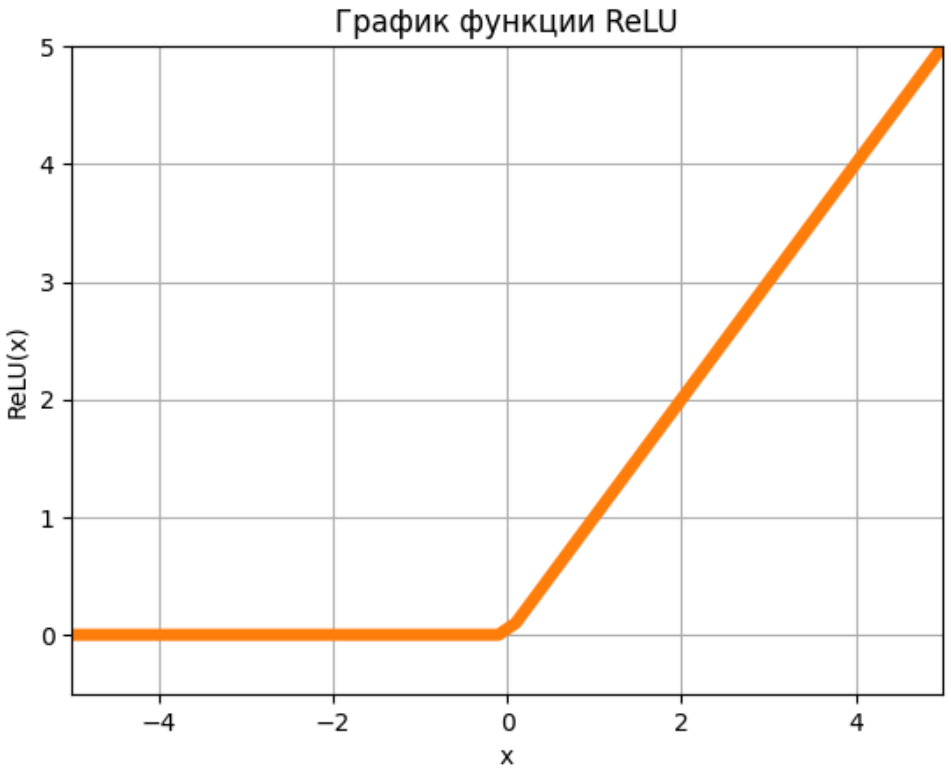
Каждая нейронная сеть состоит из одного входного, одного выходного и скрытых слоёв. Число скрытых слоёв может быть произвольным и зависит от задач и целей нейронной сети.

Каждый нейрон принимает входные данные, применяет веса и передаёт сигнал дальше, через функцию активации.

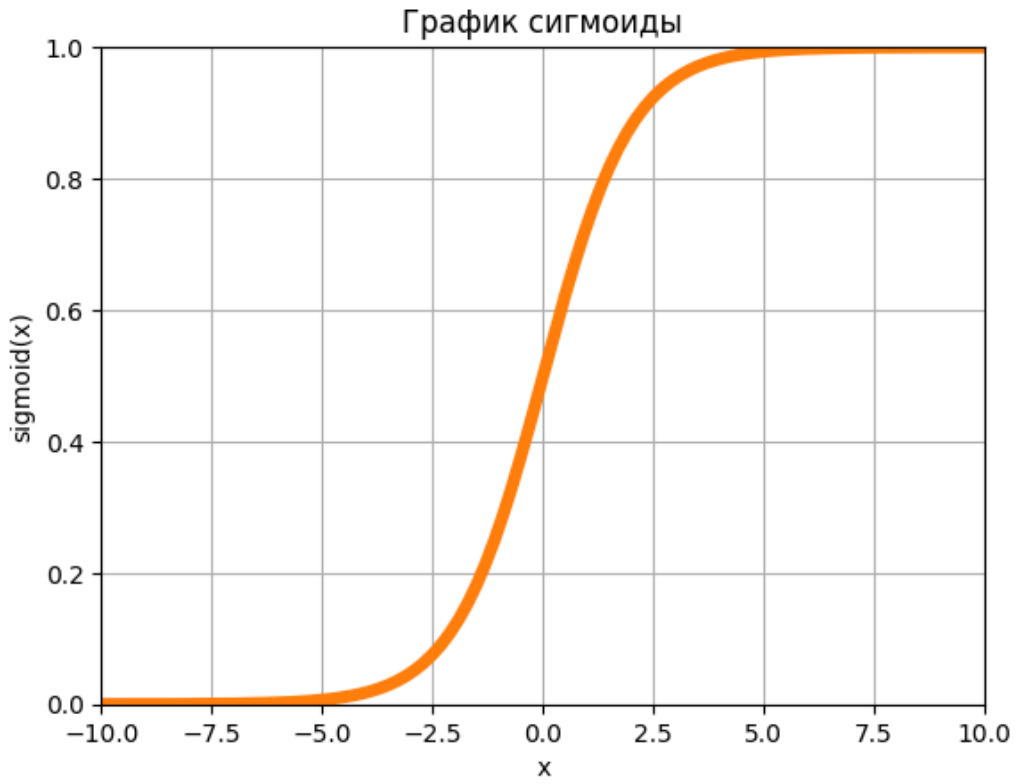
Функция активации - функция, определяющая выходной сигнал. В зависимости от функции активации получаются различные выходные сигналы.

Функция активации помогает представить выходные данные в промежутке от 0 до 1.

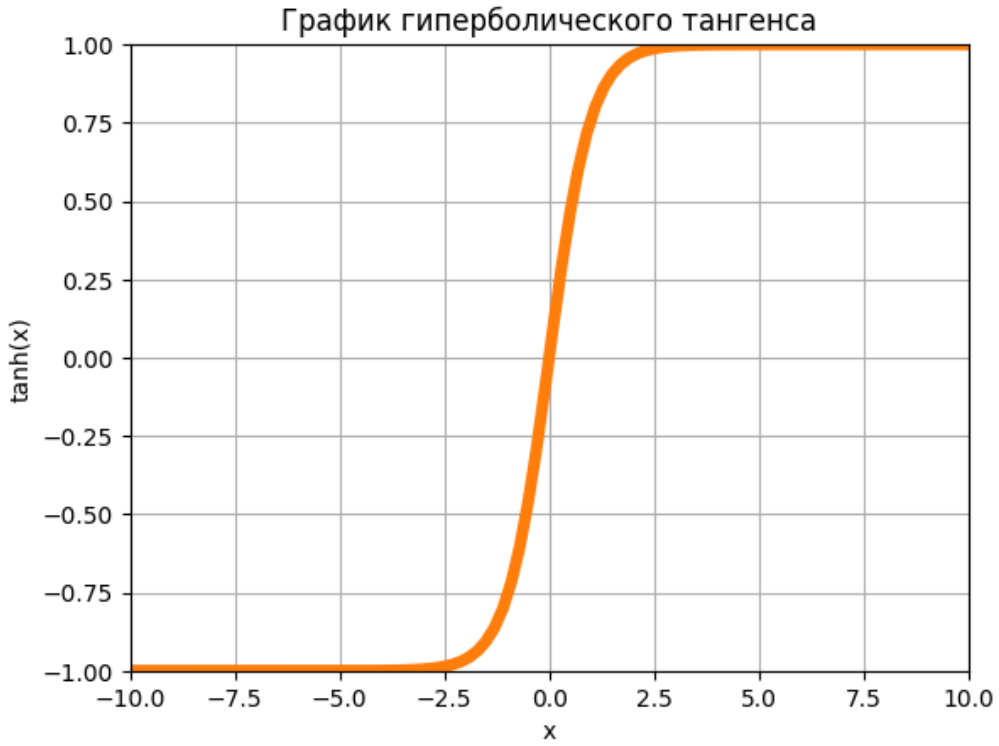
Виды основных функций активации:



- ReLU - функция активации (max(0,x)), которая преобразует все отрицательные значения в 0. Это позволяет оставить только положительные значения выходного сигнала. Самая простая и эффективная. Однако страдает проблемой “умирающих” нейронов, когда нейрон получает отрицательное значение и остаётся таковым до конца обучения.



- Sigmoid - гладкая монотонно возрастающая нелинейная функция активации($\sigma(x) = \frac{1}{1+e^{-x}}$) . Преобразует входные значения от 0 до 1, отрицательные значения по x очень близко находятся к 0, а положительные к 1. Данная функция активации используется в задачах бинарной классификации. Имеет недостаток затухающего градиента, то есть градиенты могут стать очень маленькими, что затрудняет обучение.



- Tanh - гиперболический тангенс $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Преобразует выходные значения в промежуток от -1 до 1. Лучше, чем Sigmoid в плане того, что значения центрированы относительно 0, что ускоряет обучение, однако также подвержена затуханию градиента.

- Softmax - используется для преобразования вектора значений в вероятностное распределение, которое суммируется до 1. Используется в выходном слое для задач многоклассовой классификации.

1.2.2 Типы нейронных сетей, применяемых для анализа транзакций.

Полносвязные сети (MLP):

- Используются для обработки структурированных данных. Хорошо подходят для задач классификации, например, определение мошенничества в транзакциях.

Рекуррентные сети (RNN, LSTM):

- Предназначены для анализа временных данных, таких как последовательность транзакций.
- LSTM эффективны в учёте долгосрочных зависимостей, что важно для выявления мошеннических действий, проявляющихся с течением времени.

Сверточные сети (CNN):

- Применяются для анализа временных рядов, преобразованных в изображения или векторы.
- Позволяют выявлять сложные паттерны и корреляции в последовательности финансовых операций.

1.3 Особенности данных о финансовых транзакциях

1.3.1 Структура и свойства данных: дисбаланс классов, наличие выбросов и аномалий.

Данные о финансовых транзакциях имеют различные структуры и параметры. Если рассматривать данные о финансовых транзакциях в банках, которые скрыты от посторонних, то структура там такова:
id операции, идентификатор отправителя, идентификатор получателя, дата, сумма и различные атрибуты. На просторах интернета очень сложно найти именно такую структуру, так как банки не распространяют данные о пользователях. Банки могут предоставлять датасеты, но уже анонимизированные, то есть имя(идентификатор) пользователя были скрыты. Сумма транзакции тоже может быть скрыта или преобразована в отношение к средней цене.

Если говорить о мошеннических и обычных транзакциях, то можно заметить, что первых будет много меньше, чем вторых. Отсюда и возникает дисбаланс классов. Это не очень хорошо для обучения моделей. Ведь если учить модель только на большинстве, то при встрече данных и класса меньшинства, она сможет не понять, куда отнести эти данные.

Выбросы и аномалии практически отсутствуют в транзакциях, из-за принципа атомарности(“всё или ничего”). Исключения могут возникнуть только тогда, когда часть данных теряется при передаче по каналу связи или из-за проблем с оборудованием.

1.3.2 Подготовка данных для анализа: очистка, нормализация.

В большинстве случаев данные приходят “грязными” и не читаемыми. Чтобы исправить это, используют очистку и обработку данных. К примеру пришла почти пустая строка(или несколько) всего с одним значением. Её проще удалить из данных, ведь никакой смысловой нагрузки она не несёт. Но можно попытаться восстановить эти данные с помощью различных методов(заполнение мат.ожиданием, медианой, предсказанием нейронной сети и т.п.). Когда данные были очищены, можно приступать к обработке самих признаков. Текстовые признаки в финансовых транзакциях порождают не очень эффективный анализ данных, особенно обучение модели. Поэтому нужно по максимуму избавляться от текста и переходить на метки. К примеру “да/нет” можно заменить “1/0”, категориальные признаки тоже можно заменить на числа. А поля, которые не так важны для анализа, можно и вовсе не учитывать.

После этапа обработки, можно приступать к анализу. А после анализа данные нужно подать на вход модели. Для того, чтобы модель лучше понимала зависимости, ей нужно подавать нормализованные данные.
Делается это разными путями: делением всех значений на максимальное, разностью значения и среднего и поделенного всего на среднеквадратичное отклонение.

1.3.3 Проблема дисбаланса классов и методы ее решения.

Как уже было упомянуто выше, данные о мошеннических транзакциях всегда в меньшинстве. Чтобы это исправить можно использовать разные методы:

1. Oversampling - увеличение класса меньшинства. Дублирование или синтезирование новых данных на основе уже существующих с помощью(SMOTE)
2. Undersampling - уменьшение класса большинства.

В каждом методе есть как свои преимущества, так и недостатки

	Oversampling	Undersampling
Преимущества	**Улучшение баланса данных**: увеличивает количество примеров редкого класса, что помогает модели лучше распознавать этот класс. **Сохранение всей информации**: данные доминирующего класса остаются неизменными, что минимизирует потерю информации. **Применимость к малым наборам данных**: хорошо работает, если исходный набор данных небольшой и удаление примеров большинства неприемлемо.	**Улучшение баланса данных**: уменьшает количество примеров большего класса, что помогает модели лучше распознавать редкий класс. **Снижение объёма данных**: уменьшает размер выборки, ускоряя процесс обучения модели. **Простота реализации**: легко реализовать методом случайного удаления примеров.
Недостатки	**Риск переобучения**: дублирование примеров может заставить модель запомнить данные вместо обучения общим закономерностям. **Увеличение времени обучения**: добавление данных увеличивает объём выборки, что замедляет	**Потеря информации**: удаление данных из доминирующего класса может лишить модель важных закономерностей. **Риск недообучения**: уменьшенный объём данных может привести к недостаточному обучению модели.

	процесс обучения. **Сложность генерации синтетических данных**: такие методы, как SMOTE, могут создавать нерепрезентативные примеры или добавлять шум, ухудшая качество модели.	**Неэффективность для сложных данных**: при высоком уровне разнообразия внутри доминирующего класса удаление данных может исказить распределение.

1.4 Методы оценки качества моделей для обнаружения мошеннических операций

1.4.1 Метрики оценки: точность, полнота, F1-мера, ROC-AUC.

Метрики помогают оценить, насколько модель хорошо/плохо справляется с поставленной задачей. Существует несколько метрик для оценки:

- Precision - это доля верных положительных предсказаний среди всех предсказанных положительных

  $$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

- Recall - это доля правильно классифицированных объектов из всех объектов, принадлежащих положительному классу.

  $$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

- F1-score - гармоническое среднее между точностью (Precision) и полнотой (Recall)

  $$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

- Accuracy - доля правильных предсказаний от общего числа наблюдений.

  $$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Negative + False\ Positive}$$

- ROC-AUC (Receiver Operating Characteristic – Area Under Curve) - измеряет способность модели различать классы, оценивая площадь под кривой зависимости True Positive Rate (TPR) от False Positive Rate (FPR).

Каждую метрику нужно применять, отталкиваясь от задачи:

- Точность (Accuracy): при сбалансированных классах и одинаковой важности всех ошибок.

- Полнота (Recall): при приоритете на минимизацию пропущенных случаев положительного класса.
- Точность(Precision): при необходимости минимизировать ложноположительные результаты (например, когда важно быть уверенным в каждом положительном предсказании).
- F1-мера: при необходимости баланса между точностью и полнотой.
- ROC-AUC: для сравнения моделей и работы с несбалансированными данными.

1.4.2 Проблемы с метриками в условиях дисбаланса данных.

При наличии дисбаланса классов (например, в задаче обнаружения мошенничества, где мошеннические транзакции составляют лишь малую долю всех данных), традиционные метрики, такие как точность (Accuracy), становятся менее полезными, поскольку они могут давать ложные положительные результаты, преувеличивая эффективность модели. Рассмотрим основные проблемы с метриками в условиях дисбаланса данных:

1. Проблема точности (Accuracy):

В случае дисбаланса классов модель может просто предсказывать большинство классов, минимизируя количество ошибок, но при этом пропуская важные случаи из редкого класса. Например, если 95% транзакций не являются мошенническими, модель может предсказать все как "нормальные", получив высокую точность, но не будет распознавать мошенничество.

2. Недооценка редких классов:

В задачах с дисбалансом, как в случае обнаружения мошенничества, важность полноты (Recall) значительно возрастает. Модель должна быть

настроена на выявление редких, но критически важных случаев (например, мошенничество). Если метрика ориентирована только на точность, то модель может игнорировать мошеннические транзакции.

3. Высокая зависимость от ложноположительных и ложноотрицательных ошибок:

В условиях дисбаланса модели могут сильно искажать Precision и Recall, особенно если они склонны к большому числу ложноположительных (когда нормальная транзакция классифицируется как мошенничество) или ложноотрицательных ошибок (когда мошенничество классифицируется как нормальная транзакция).

4. Неэффективность простых метрик (например, Accuracy) при многоклассовых задачах:

Если задача классификации включает более двух классов (например, несколько типов мошенничества), метрики вроде точности и полноты могут быть еще менее информативными, так как они не учитывают специфические особенности каждого класса.

1.4.3 Особенности интерпретации результатов модели в финансовой сфере.

1. Риск и последствия неудачи:

В финансовой сфере последствия ошибок модели могут быть довольно серьезными. Например, ложная тревога обычной транзакции как мошеннической может вызвать неудобства для клиентов и финансовые потери для бизнеса. С другой стороны, пропуск мошеннической транзакции может привести к потенциальным финансовым потерям, репутационному риску и возможным юридическим последствиям.

2. Зависит от контекста:

Интерпретация результатов модели в финансах должна проводиться с учетом контекста транзакции. Например, анализ мошенничества будет включать дополнительные данные, такие как поведение пользователя, аномалии в платежах, географические и временные закономерности, которые требуют надлежащего учета моделью для получения лучших прогнозов.

3. Решения на основе модели:

В финансовых системах результаты модели становятся основой для решений в реальном времени, поэтому модели должны быть объяснимыми, помимо точности. Например, в случаях, когда система обнаружения мошенничества решает заблокировать транзакцию, причины такого решения должны быть объяснены пользователю или регулирующему органу.

4. Баланс риска и выгоды:

В финансовых приложениях следует рассмотреть компромисс между риском ложных отрицательных результатов, например, пропуска мошеннической транзакции, и риском ложных положительных результатов, включая блокировку законных транзакций. Это необходимо для того, чтобы система работала правильно, неудобства для пользователей уменьшались, а финансовые потери предотвращались.

5. Важность регулярной переоценки моделей:

Финансовая сфера постоянно развивается, и модели прогнозирования мошенничества потенциально быстро устаревают, поскольку мошенники осваивают новые способы уклонения от обнаружения. Это означает, что результаты моделей следует периодически пересматривать, а модели переобучать на новых данных, чтобы результаты оставались актуальными.

1.5 Обзор существующих технологий

1.5.1 Анализ инструментов и библиотек для построения моделей (Scikit-learn, PyTorch, TensorFlow).

1. Scikit-learn:

   Описание: Scikit-learn — это одна из наиболее популярных библиотек для машинного обучения в Python. Она предоставляет простой и универсальный интерфейс для различных алгоритмов, таких как классификация, регрессия, кластеризация, а также для предобработки данных и оценки моделей.

2. PyTorch:

   Описание: PyTorch — это фреймворк для глубокого обучения, который был разработан компанией Facebook. Он используется для построения и обучения нейронных сетей с возможностью гибкой настройки.

3. TensorFlow:

   Описание: TensorFlow — это фреймворк с открытым исходным кодом для создания и обучения машинных моделей, особенно глубоких нейронных сетей. TensorFlow широко используется в промышленности для обработки больших данных и построения сложных моделей

	Scikit-learn	Pytorch	TensorFlow
Преимущества	- Легкость в использовании. - Множество встроенных алгоритмов (например, деревья решений, логистическая регрессия, SVM). - Хорошая документация и активное сообщество. - Поддержка различных метрик и методов кросс-валидации.	- Отлично подходит для динамических вычислений и исследования. - Поддержка автодифференцирования, что упрощает разработку сложных моделей. - Более интуитивно понятный и гибкий, чем TensorFlow. - Расширенные возможности для работы с нейронными сетями, включая сверточные и рекуррентные сети.	- Отлично масштабируется, что делает его удобным для работы с большими данными. - Поддерживает распределенное обучение и работу на разных устройствах (GPU, TPU). - Обширное сообщество и множество готовых моделей.

Недостатки	- Меньшая гибкость для глубокого обучения, по сравнению с TensorFlow или PyTorch. - Не поддерживает распределенные вычисления для обработки больших объемов данных.	- Меньше готовых инструментов и библиотек для работы с большими данными, чем у TensorFlow. - Для некоторых задач может потребоваться больше вычислительных ресурсов.	- Более сложный и менее интуитивно понятный по сравнению с PyTorch, особенно для новичков. - Меньше гибкости при настройке модели для конкретных исследований.

1.5.2 Сравнение существующих моделей обнаружения мошенничества.

Существует несколько подходов к построению моделей для обнаружения мошенничества в финансовых транзакциях. Рассмотрим основные из них:

Таблицу с описанием методов и их преимуществами и недостатками можно посмотреть в Приложении 1

1.5.3 Использование распределенных технологий для обработки больших данных (PySpark).

PySpark — это интерфейс, который работает с Apache Spark и обеспечивает распределенную обработку данных для больших наборов данных.
Обработка транзакций, машинное обучение и анализ больших объемов данных — все это его функции.

Преимущества:

1) Поддерживает распределенную обработку данных, что позволяет работать с огромными количествами данных.

2) Быстрая обработка кластерных данных с возможностью параллельного выполнения задач

3) Поддержка интеграции с MLlib и другими известными инструментами для машинного обучения.

4) Возможности обработки информации в режиме реального времени.

Недостатки:

1) Требует значительного количества вычислительных ресурсов, таких как кластеры.

2) По сравнению с простыми библиотеками, такими как Scikit-learn, настройка и управление более сложны.

Применение PySpark в анализе мошенничества:

- Большие объемы данных о транзакциях, например, в режиме реального времени, могут быть обработаны и анализированы PySpark.
- Распределение вычислений по нескольким узлам кластера помогает эффективно масштабировать модели, работающие с миллионами транзакций.

Глава 2 Практическая часть

2.1 Поиск датасета

Для поиска подходящего датасета были проанализированы такие сайты, как Kaggle.com, huggingface.co, openml.org.

Именно на последнем сайте я решил остановить свой взгляд. там был найден подходящий датасет с финансовыми транзакциями(ссылка на него в Приложении 2).

2.2 Выбор инструментов

Для выполнения данной работы мной были выбраны следующие средства:

- Google Colab(для разработки на языке Python кода)
- Модули для языка Python:
    1. openml(для загрузки датасета)
    2. pandas(для работы с датасетом)
    3. pyspark(для работы с датасетом, используя sql запросы)
    4. matplotlib и seaborn(визуализация графиков)
    5. numpy(вычисление математических функций)

2.3 Загрузка датасета

После того, как датасет был найден, его нужно загрузить в среду Colab и уже использовать там.

```
import openml
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark import SparkContext, SparkConf

# 45955 - id нужного датасета на сайте OpenML
id = 45955

# Загружаем датасет с сайта в объект dataset
dataset = openml.datasets.get_dataset(id)

print(type(dataset))

# Переносим датасет из объекта OpenMLDataset
df, _, _, _ = dataset.get_data(dataset_format="dataframe")
df = df.drop(columns='used_chip')
```

Данной частью кода я загрузил датасет(id 45955) с сайта openml в объект Pandas DataFrame. При этом в данных есть колонка, отвечающая за использование чипа, однако мне не удалось понять, что она значит, ведь чип используется тогда, когда оплата была оффлайн, а бывало так, что чип был использован онлайн. Поэтому я удалил этот столбец функцией **drop**

```
conf = SparkConf().setAppName("FraudDetection").setMaster("local[*]")
sc = SparkContext(conf=conf)

spark = SparkSession.builder.getOrCreate()
spark_df = spark.createDataFrame(df)
```

Далее создаём датафрейм на Spark. Задаём имя “FraudDetection” и задаём, что спарк будет запущен локально с тем количеством рабочих, сколько позволяет процессор.

```
# Выведем первые 5 строк, чтобы посмотреть структуру датасета
df.head()
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_pin_number	online_order	fraud
0	57.877857	0.311140	1.945940	1	0	0	0
1	10.829943	0.175592	1.294219	1	0	0	0
2	5.091079	0.805153	0.427715	1	0	1	0
3	2.247564	5.600044	0.362663	1	0	1	0
4	44.190936	0.566486	2.222767	1	0	1	0

```
# Со спарком сделаем то же самое
spark_df.show(5)
```

```
+------------------+-----------------------------+----------------------------+---------------+---------------+------------+-----+
|distance_from_home|distance_from_last_transaction|ratio_to_median_purchase_price|repeat_retailer|used_pin_number|online_order|fraud|
+------------------+-----------------------------+----------------------------+---------------+---------------+------------+-----+
|     57.8778565839|                  0.311140008|           1.9459399776000001|              1|              0|           0|    0|
|     10.8299426993|                0.1755915023|           1.2942188105999999|              1|              0|           0|    0|
|      5.0910794906|                0.8051525946|                0.4277145612|              1|              0|           1|    0|
|      2.2475643283|                 5.600043547|                0.3626625781|              1|              0|           1|    0|
|     44.1909360026|                0.5664862681|                2.2227672978|              1|              0|           1|    0|
+------------------+-----------------------------+----------------------------+---------------+---------------+------------+-----+
only showing top 5 rows
```

Описание атрибутов:

- distance_from_home: это числовой признак, представляющий географическое расстояние в километрах между местом транзакции и домашним адресом владельца карты.
- distance_from_last_transaction: этот числовой атрибут измеряет расстояние в километрах от места последней транзакции до текущего местоположения транзакции.
- Ratio_to_median_purchase_price: числовое соотношение, которое сравнивает цену транзакции со средней ценой покупки в истории транзакций пользователя.
- repeat_retailer: двоичный атрибут, где «1» означает, что транзакция была проведена у розничного продавца, ранее использовавшегося держателем карты, а «0» указывает на нового розничного продавца.

- used_pin_number: еще одна двоичная функция, где «1» означает использование PIN-кода для транзакции, а «0» означает, что PIN-код не использовался.
- online_order: этот атрибут определяет, была ли покупка совершена онлайн («1») или оффлайн («0»).
- fraud: двоичная целевая переменная, указывающая, была ли транзакция мошеннической («1») или нет («0»).

2.4 Обработка данных и EDA

```
# Выведем информацию о типах значений в полях, пропущенных значениях
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 7 columns):
 #   Column                          Non-Null Count    Dtype  
---  ------                          --------------    -----  
 0   distance_from_home              1000000 non-null  float64
 1   distance_from_last_transaction  1000000 non-null  float64
 2   ratio_to_median_purchase_price  1000000 non-null  float64
 3   repeat_retailer                 1000000 non-null  uint8  
 4   used_pin_number                 1000000 non-null  uint8  
 5   online_order                    1000000 non-null  uint8  
 6   fraud                           1000000 non-null  uint8  
dtypes: float64(3), uint8(4)
memory usage: 26.7 MB
```

Видим, что первые 3 поля имеют тип float64, а остальные имеют всего “0” или “1”, соответственно можно использовать uint8(unsigned int 8-bit). также сразу видно, сколько всего записей в датасете - 1.000.000. Этот датасет использует 26.7 МБ памяти.

Сразу отсюда видно, что данные все целые, нет пропусков ни по какому полю.

2.4.1 Предобработка

Так как данные все целые, то удалять строки или восполнять значения каким-либо способом нам не понадобится.

Посмотрим, все ли данные корректные.

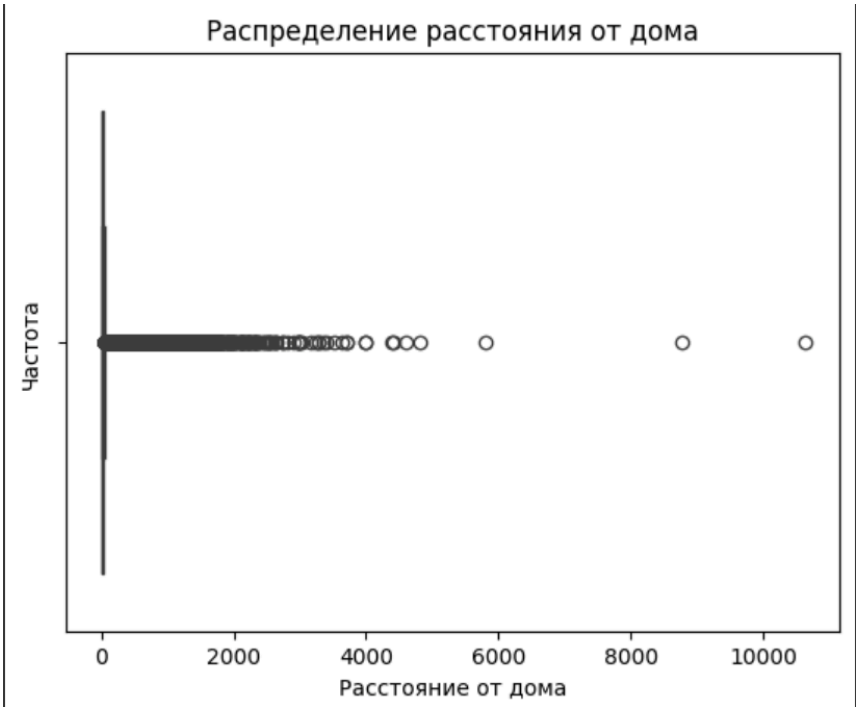
```
# Выведем статистику по каждому столбцу(минимальное, максимальное, среднее
значения, перцентили)
```

```
df.describe()
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_pin_number	online_order	fraud
count	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000
mean	26.628792	5.036519	1.824182	0.881536	0.100608	0.650552	0.087403
std	65.390784	25.843093	2.799589	0.323157	0.300809	0.476796	0.282425
min	0.004874	0.000118	0.004399	0.000000	0.000000	0.000000	0.000000
25%	3.878008	0.296671	0.475673	1.000000	0.000000	0.000000	0.000000
50%	9.967760	0.998650	0.997717	1.000000	0.000000	1.000000	0.000000
75%	25.743985	3.355748	2.096370	1.000000	0.000000	1.000000	0.000000
max	10632.723672	11851.104565	267.802942	1.000000	1.000000	1.000000	1.000000

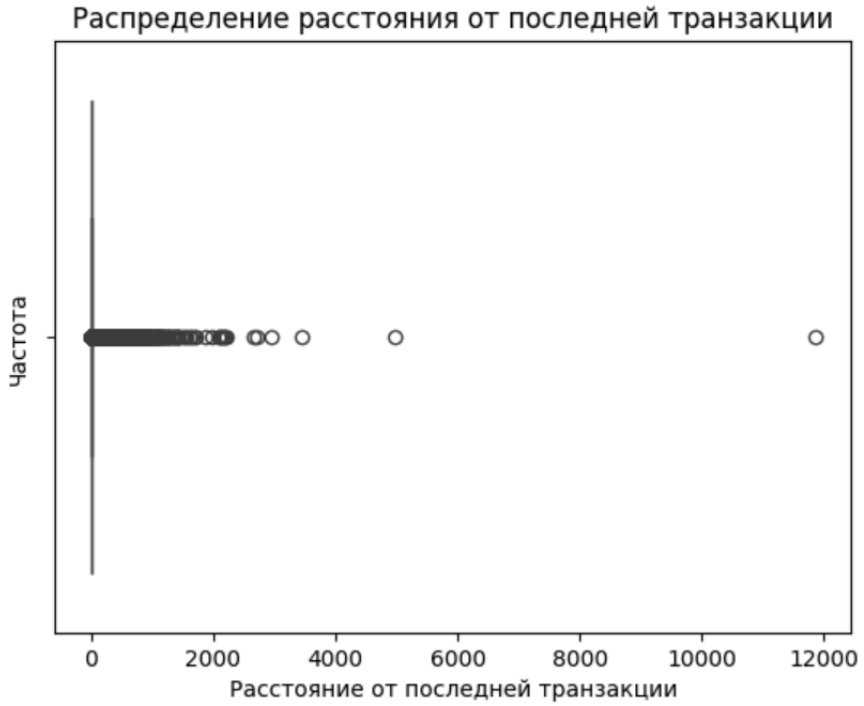
Отсюда видно статистику по каждому столбцу. Можно точно сказать, что никакое поле не принимает отрицательных значений, что вполне логично. Все остальные значения колеблются в пределах нормы, однако немного выбиваются из общей картины максимальные значения в первых трёх столбцах.

Посмотрим на их распределение:

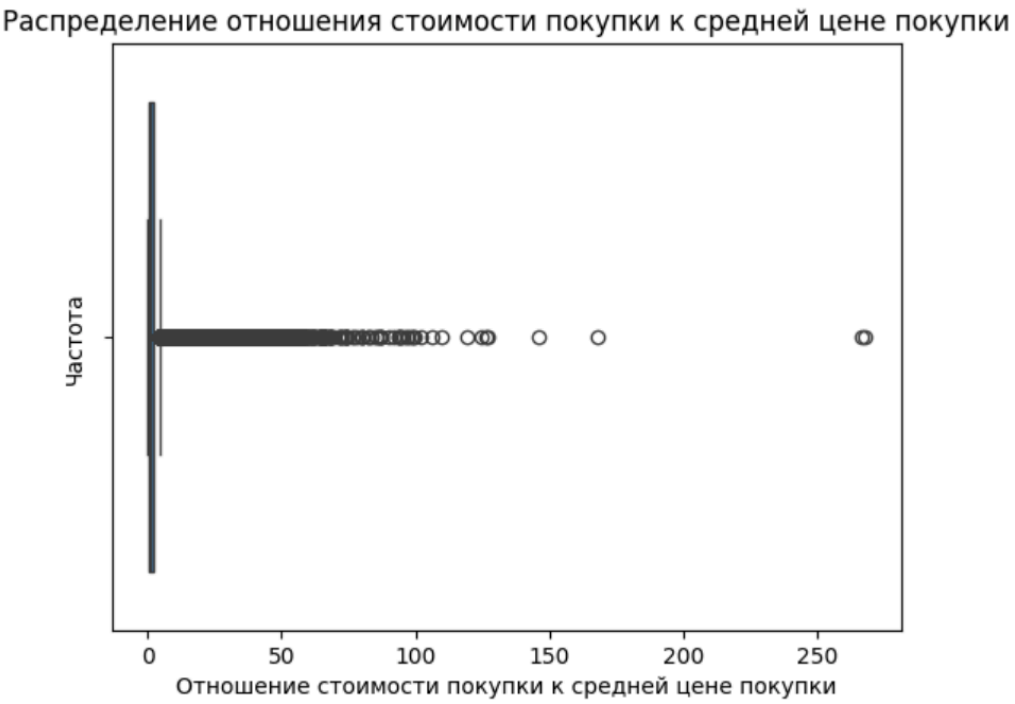



```
# Выведем boxplot распределения расстояния от дома
sns.boxplot(x=df['distance_from_home'])
plt.title('Распределение расстояния от дома')
```

```
plt.xlabel('Расстояние от дома')
plt.ylabel('Частота')
plt.show()
```




```
# Выведем boxplot распределения расстояния от последней транзакции
sns.boxplot(x=df['distance_from_last_transaction'])
plt.title('Распределение расстояния от последней транзакции')
plt.xlabel('Расстояние от последней транзакции')
plt.ylabel('Частота')
plt.show()
```



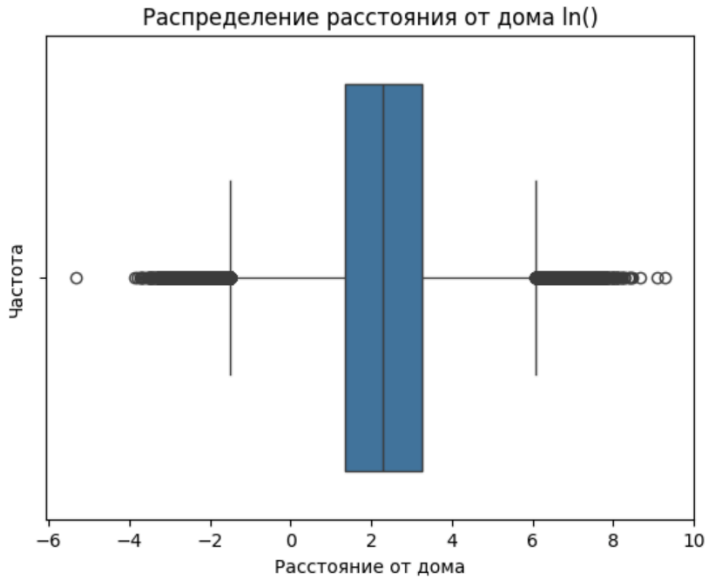

```
# Выведем boxplot распределения расстояния от последней транзакции
sns.boxplot(x=df['ratio_to_median_purchase_price'])
plt.title('Распределение отношения стоимости покупки к средней цене покупки')
plt.xlabel('Отношение стоимости покупки к средней цене покупки')
plt.ylabel('Частота')
plt.show()
```

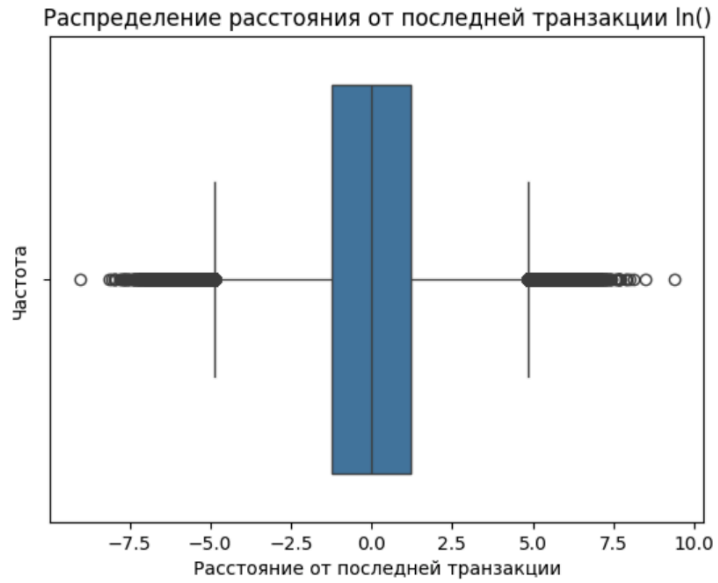
Видно что все 3 распределения имеют длинный правый хвост. Это можно расценить как выбросы, однако эти данные могут быть очень важными для обучения модели в дальнейшем. Так что вместо того, чтобы удалить слишком большие значения, можно воспользоваться трансформацией данных(использовать логарифмирование). Логарифмирование поможет нам лучше локализовать данные.

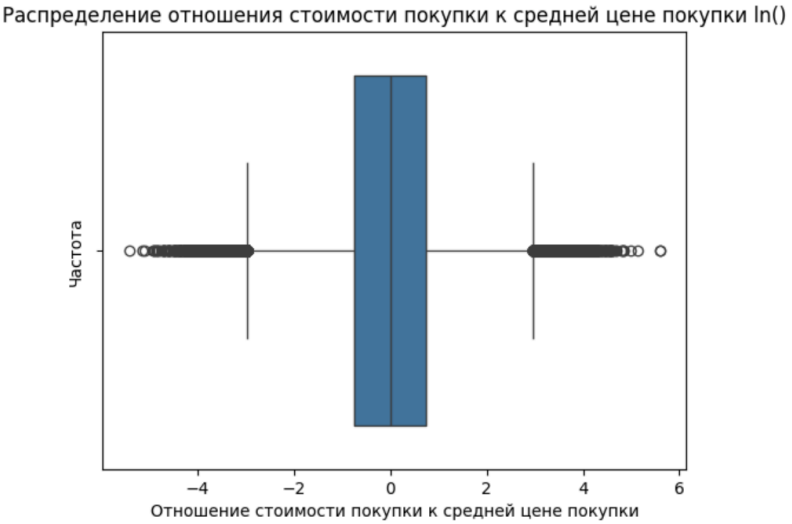
```
import numpy as np
# Вместо "обрезания" хвоста воспользуемся преобразованием данных, используя
натуральный логарифм
# Такое преобразование поможет сохранить все данные и избавит от больших
чисел
```

```
df_ln = df[['distance_from_home','distance_from_last_transaction',
'ratio_to_median_purchase_price']].map(lambda x: np.log(x))
df_ln =
pd.concat([df_ln,df[['repeat_retailer','used_pin_number','online_order','fra
ud']]],axis=1)
```

Выведем теперь распределения этих величин после преобразования







Теперь видно, что данные лучше локализованы и выбросов особо нет. (именно этот датасет будем подавать на обучение модели, а для анализа использовать датасет до преобразования).

2.4.2 EDA

Выведем все бинарные признаки и проанализируем их

```
import pyspark.sql.functions as F
fraud_count =
spark_df.select('repeat_retailer','used_pin_number','online_order','fraud').
groupBy('repeat_retailer','used_pin_number','online_order','fraud').agg(F.co
unt('fraud').alias('count')).orderBy('fraud')
fraud_count.show()
```

```
+---------------+---------------+------------+-----+------+
|repeat_retailer|used_pin_number|online_order|fraud| count|
+---------------+---------------+------------+-----+------+
|              0|              0|           1|    0| 61308|
|              1|              1|           0|    0| 31083|
|              0|              1|           0|    0|  4115|
|              0|              1|           1|    0|  7830|
|              1|              1|           1|    0| 57307|
|              1|              0|           1|    0|441396|
|              1|              0|           0|    0|274825|
|              0|              0|           0|    0| 34733|
|              1|              0|           1|    1| 74442|
|              1|              1|           1|    1|   258|
|              0|              0|           1|    1|  7997|
|              0|              1|           1|    1|    14|
|              0|              0|           0|    1|  2467|
|              1|              0|           0|    1|  2224|
|              1|              1|           0|    1|     1|
+---------------+---------------+------------+-----+------+
```

Исходя из результатов, можно сказать, что

1. Почти все операции, в которых использовался пин-код, являются не мошенническими.
2. Большая часть мошеннических операций приходится на онлайн транзакции
3. Больше всего мошеннических транзакций происходило в том случае, когда покупка совершалась онлайн у "старого" продавца, не используя пин-код
4. И был всего один случай, когда покупка совершённая оффлайн у "старого" продавца, используя пин-код, считалась мошеннической. Быть может это была кража карты вместе с пин-кодом. Либо мошенник до этого знал пин-код к карте и ему оставалось лишь своровать карту.

Теперь рассмотрим влияние отношения цены транзакции к средней цене

Вместо того, чтобы выводить все значения, создадим функцию, которая создаст категории для этого признака. И далее сгруппируем по этим категориям.

```
def set_category(x):
    if x < 2:
      return "normal"
    elif x < 4:
      return "medium"
    else:
      return "high"

from pyspark.sql.types import StringType

set_category_udf = F.udf(set_category, StringType())
```

```
df_with_caterоties_of_ratio =
spark_df.withColumn('Category',set_category_udf(F.col('ratio_to_median_purch
ase_price')))

df_with_caterоties_of_ratio.select('Category','fraud').groupBy('Category','f
raud').agg(F.count('fraud').alias('count')).orderBy('fraud').show()
```

```
+--------+-----+------+
|Category|fraud| count|
+--------+-----+------+
|  medium|    0|156547|
|  normal|    0|717841|
|    high|    0| 38209|
|  medium|    1|  4036|
|  normal|    1| 18416|
|    high|    1| 64951|
+--------+-----+------+
```

Больше всего мошеннических транзакций было произведено, когда отношение стоимости покупки к средней цене составляло более 4, меньше всего от 2 до 4. Это может быть связано с тем, что мошенники в большинстве случаев тратят чужие деньги на нечто крупное(по стоимости).

Далее рассмотрим влияние расстояния на мошеннические операции. Также создадим функцию для задания категорий для расстояния.

```
def set_distance_category(x):
  if x < 5:
    return "low"
  elif x < 40:
    return "normal"
  elif x < 100:
    return "medium"
  else:
    return "high"
```

```
set_distance_category_udf = F.udf(set_distance_category, StringType())
df_with_distance_from_home_category =
spark_df.withColumn('distance_from_home_category',set_distance_category_udf(
F.col('distance_from_home')))
df_with_distance_from_home_category =
df_with_distance_from_home_category.withColumn('distance_from_last_transacti
on_category',set_distance_category_udf(F.col('distance_from_last_transaction
')))

df_with_distance_from_home_category.select('distance_from_home_category','di
stance_from_last_transaction_category','fraud').groupBy('distance_from_home_
category','distance_from_last_transaction_category','fraud').agg(F.count('fr
aud').alias('count')).orderBy('fraud').show(50)
```

```
+------------------------+-----------------------------------+-----+------+
|distance_from_home_category|distance_from_last_transaction_category|fraud| count|
+------------------------+-----------------------------------+-----+------+
|                     low|                             medium|    0|  2945|
|                  normal|                             normal|    0| 81800|
|                  normal|                             medium|    0|  5431|
|                     low|                                low|    0|236313|
|                  normal|                               high|    0|  1580|
|                     low|                               high|    0|   774|
|                     low|                             normal|    0| 47896|
|                  medium|                             medium|    0|  1202|
|                    high|                             medium|    0|   205|
|                  medium|                               high|    0|   344|
|                  medium|                             normal|    0| 17302|
|                  medium|                                low|    0| 85138|
|                    high|                             normal|    0|  4557|
|                    high|                               high|    0|    24|
|                    high|                                low|    0| 22796|
|                  normal|                                low|    0|404290|
|                    high|                             normal|    1|  3551|
|                  normal|                             medium|    1|  2326|
|                  normal|                               high|    1|  1210|
|                     low|                               high|    1|   833|
|                     low|                             medium|    1|  1630|
|                    high|                             medium|    1|   557|
|                    high|                               high|    1|   246|
|                  medium|                             normal|    1|  1087|
|                  normal|                             normal|    1|  5204|
|                  medium|                                low|    1|  5441|
|                     low|                             normal|    1|  3427|
|                  medium|                               high|    1|   249|
|                     low|                                low|    1| 17392|
|                  normal|                                low|    1| 25887|
|                  medium|                             medium|    1|   510|
|                    high|                                low|    1| 17853|
+------------------------+-----------------------------------+-----+------+
```

Получилось, что больше всего мошеннических операций приходится на очень маленькие расстояния от последней покупки, но на разные расстояния от дома.

Дополним датасет данными об онлайн и оффлайн транзакциях

```
df_with_distance_from_home_category\

.select('distance_from_home_category','distance_from_last_transaction_category','online_order','fraud')\

.groupBy('distance_from_home_category','distance_from_last_transaction_category','online_order','fraud')\
                                    .agg(F.count('fraud').alias('count'))\
                                    .orderBy('fraud')\
                                    .filter(F.col('fraud')==1)\
                                    .show(50)
```

```
+--------------------------+---------------------------------------+------------+-----+-----+
|distance_from_home_category|distance_from_last_transaction_category|online_order|fraud|count|
+--------------------------+---------------------------------------+------------+-----+-----+
|                      high|                                 medium|           1|    1|  393|
|                    normal|                                   high|           1|    1| 1121|
|                    medium|                                   high|           1|    1|  229|
|                      high|                                 normal|           1|    1| 3278|
|                    medium|                                   high|           0|    1|   20|
|                       low|                                 medium|           1|    1| 1524|
|                       low|                                 normal|           1|    1| 3049|
|                       low|                                    low|           1|    1|15366|
|                       low|                                   high|           1|    1|  779|
|                    medium|                                    low|           1|    1| 5441|
|                       low|                                   high|           0|    1|   54|
|                       low|                                    low|           0|    1| 2026|
|                    medium|                                 medium|           0|    1|   36|
|                    normal|                                 medium|           0|    1|  162|
|                       low|                                 medium|           0|    1|  106|
|                    medium|                                 normal|           1|    1| 1087|
|                    medium|                                 medium|           1|    1|  474|
|                      high|                                   high|           0|    1|   80|
|                      high|                                 medium|           0|    1|  164|
|                      high|                                    low|           0|    1| 1304|
|                    normal|                                 normal|           1|    1| 5204|
|                       low|                                 normal|           0|    1|  378|
|                    normal|                                   high|           0|    1|   89|
|                      high|                                   high|           1|    1|  166|
|                    normal|                                 medium|           1|    1| 2164|
|                      high|                                 normal|           0|    1|  273|
|                    normal|                                    low|           1|    1|25887|
|                      high|                                    low|           1|    1|16549|
+--------------------------+---------------------------------------+------------+-----+-----+
```

Теперь имеем более подходящую для анализа картину.

Рассмотрим три самых больших по количеству мошеннических транзакций. Как видно из таблицы это значения 25887, 16549 и 15366

Рассмотрим остальные характеристики этих значений. Все эти операции были сделаны онлайн. А теперь рассмотрим их по отдельности:

1. normal-low - то есть очень маленькое расстояние от последней транзакции и чуть большее от дома. Быть может мошенник находился в чертах одного города и совершил транзакцию где-то рядом с последней покупкой. Например: Человек О(обладатель карты) был в общественном месте, к примеру, в торговом центре вдали от дома. Человек М(Мошенник) увидел, как человек О расплачивался картой за еду. М успел сфотографировать данные карты и уехал из торгового центра на расстояние меньше 5 километров. М Совершил перевод с карты или оплатил покупку с этого места.

2. high-low - то есть очень маленькое расстояние от последней транзакции и большое от дома. Быть может М произвёл две транзакции. Одну транзакцию произвели "для вида". Показать, будто О находится в другом городе, далеко от своего дома. А вторую транзакцию М произвёл уже для того, чтобы украсть деньги или совершить большую покупку.

3. low-low - то есть очень маленькое расстояние от последней транзакции и от дома. В пределах маленького города очень даже реальная ситуация. Быть может даже М использовал прокси сервер таковой, что его местоположение почти совпало с адресом О. Даже если взять город

радиусом 5 километров, а в центре размещён сервер, то вполне можно подтвердить данные значения по количеству мошеннических транзакций

Глава 3 Практическая часть

3.1 Выбор инструментов

Для создания, обучения и тестирования модели были выбраны PyTorch и Scikit-learn(для нормализации данных, разделения датасета на тренировочный и тестовый датасеты, вывод метрик)

3.2 Решение проблемы несбалансированных данных

```
value =
df_ln['fraud'].value_counts()[1]/df_ln['fraud'].value_counts()[0]*100
print(f'{round(value,1)}%')
```

Сами по себе данные не сбалансированы. Доля мошеннических операций составляет всего 9.6%

Этого может не хватить, чтобы обучить модель правильно различать данные. Поэтому можно применить операцию oversampling над мошенническими транзакциями.

Можно было пойти простым путём и просто дублировать транзакции, однако модель могла бы легко начать переобучаться, так как каждый раз может встречать одни и те же данные, а при встрече новых может не понять,что делать.

Поэтому воспользуемся SMOTE(Synthetic Minority Oversampling Technique). То есть воспользуемся функцией, которая синтезирует новые мошеннические транзакции, опираясь на уже существующие. То есть

выполнит аугментацию данных. Данные будут похожи на существующие, но не будут являться дубликатами.

```
from imblearn.over_sampling import SMOTE

data = df_ln.copy()
# Разделяем на признаки и целевую переменную
X = data.drop(columns=['fraud'])
y = data['fraud']

# Применяем SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Создаём новый DataFrame
data_resampled = pd.concat([pd.DataFrame(X_resampled),
pd.DataFrame(y_resampled, columns=['fraud'])], axis=1)
```

3.3 Создание модели нейронной сети

После того, как данные для обучения были подготовлены, можно начинать создание модели.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Загрузка данных
data = data_resampled.copy()
```

```
# Разделение на фичи и целевую переменную
X = data.drop(columns=["fraud"])
y = data["fraud"]
```

Подключаем необходимые модули, а именно torch и sklearn. копируем наш датасет в новую переменную. Убираем колонку, соответствующую метке мошеннической операции, и сохраняем в другую переменную.

```
# разделяем датасет на тренировочный и тестовый (метки отдельно)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
stratify=y, random_state=42)
```

Разделяем датасет на тренировочный и тестовый

```
# Стандартизируем данные(приводим их к нормальному виду)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Используем StandardScaler для приведения данных к нормальному виду.

```
# Создаём класс для работы с датасетами PyTorch
class TransactionDataset(Dataset):
    def __init__(self, X, y):
        # Переводим наши данные в тензоры, так как в PyTorch почти все
операции происходят именно с ними
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y.values, dtype=torch.float32)

    # Создаём метод для нахождения количества данных(количество строк)
    def __len__(self):
```

```
        return len(self.y)

    # Создаём метод для получения i-ой строки
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

# Создаём объекты нашего класса, для последующей загрузки в DataLoader
train_dataset = TransactionDataset(X_train, y_train)
test_dataset = TransactionDataset(X_test, y_test)

# Создаём лоадеры для тренировочного и тестового датасетов, чтобы данные
могли подаваться порциями(батчами)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True) #
Разрешаем перемешивание данных
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False) #
Запрещаем перемешивание данных
```

Создание самой модели

```
# Создаём нашу модель
class FraudDetectionNN(nn.Module):
    # Описываем структуру нашей модели
    def __init__(self):
        super(FraudDetectionNN, self).__init__()
        # Модель будет состоять из 5 полносвязных слоёв. На каждом слое,
кроме выходного определим функцию ReLU, чтобы избавиться от отрицательных
значений нейронов.
        # А на последнем используем sigmoid, чтобы выдать вероятность
предсказания.
        self.layers = nn.Sequential(
            nn.Linear(X_train.shape[1], 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 16),
            nn.ReLU(),
            nn.Linear(16, 8),
```

```
            nn.ReLU(),
            nn.Linear(8, 1),
            nn.Sigmoid()
        )

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

# Объявляем модель и переносим её на графический процессор, если есть такая возможность
model = FraudDetectionNN()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
print(device)

# Конвертируем данные из объекта y_train в тензор PyTorch с целочисленным типом данных
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)

# Указываем функцию потерь (Binary-CrossEntropy loss) для бинарной классификации
criterion = nn.BCELoss()
# Указываем оптимизатор Adam, который адаптирует шаг обучения для каждого параметра модели на основе истории градиентов и их моментов (среднего и дисперсии)
optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

На данном этапе модель полностью создана, но ещё ничего не имеет. Необходимо её обучить. Переходим к следующему этапу

3.4 Обучение модели

Обучение модели - процесс, в течение которого модель машинного обучение(в моём случае, нейронной сети) находит скрытые зависимости в данных, считает градиенты, определят веса для нейронов.

Приступим к этому этапу

```
# Задаём количество эпох
epochs = 20
# Переключаем модель в режим обучения
model.train()

for epoch in range(epochs):
    # На каждой эпохе обнуляем ошибку
    epoch_loss = 0
    for X_batch, y_batch in train_loader:
        # Переносим батчи на устройство(cuda или cpu)
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        # Обнуляем градиенты, чтобы избежать их накопления
        optimizer.zero_grad()
        # Пропускаем батч через модель и убираем измерения с размерностью 1
(если такие имеются) во выходных данных
        outputs = model(X_batch).squeeze()
        # Считаем ошибку
        loss = criterion(outputs, y_batch)
        # Обратное распространение ошибок(вычисление градиентов)
        loss.backward()
        # Применение к параметрам модели вычисленных градиентов
        optimizer.step()
        # Суммирование ошибки
        epoch_loss += loss.item()
    # Выводим номер эпохи и значение функции потерь эпохи
    print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss/len(train_loader)}")
```

Результат:

Epoch 1/20, Loss: 0.1204475320487267

Epoch 2/20, Loss: 0.06313513006248195

Epoch 3/20, Loss: 0.05230366611813403

Epoch 4/20, Loss: 0.04577573010517515

Epoch 5/20, Loss: 0.04197559320316196
Epoch 6/20, Loss: 0.039831439267661695
Epoch 7/20, Loss: 0.038347814762905524
Epoch 8/20, Loss: 0.03746843281628276
Epoch 9/20, Loss: 0.03677289658771991
Epoch 10/20, Loss: 0.03617441408904184
Epoch 11/20, Loss: 0.035862836001586934
Epoch 12/20, Loss: 0.03542557101243613
Epoch 13/20, Loss: 0.03508198822489715
Epoch 14/20, Loss: 0.03480483752673259
Epoch 15/20, Loss: 0.0346515532294103
Epoch 16/20, Loss: 0.03445506175309312
Epoch 17/20, Loss: 0.03425801649780313
Epoch 18/20, Loss: 0.03411552795688248
Epoch 19/20, Loss: 0.033972663777121714
Epoch 20/20, Loss: 0.03380394119428783

Как видно из результата, модель с каждой новой эпохой всё лучше и лучше справлялась с поставленной задачей, так как ошибка с каждой эпохой уменьшалась.

На данном этапе наша модель обучена и ей можно подать тестовые данные, чтобы проверить, насколько хорошо она умеет определять класс данных, которые ни разу не встречала.

Переходим к следующему этапу

3.5 Тестирование модели

Тестирование модели - этап разработки модели, при котором модель проверяют на то, насколько хорошо она работает.

Модель может обучиться очень хорошо, но при встрече новых данных может попросту начать угадывать. Если бы я продолжал работать с несбалансированными данными, то модель очень хорошо бы предсказывала класс обычных операций(“0”), а при встрече данных из класса мошеннических(“1”) начала бы просто угадывать.

Для того, чтобы перейти к тестированию, нужно переключить модель в режим тестирования и выключить вычисление градиентов, дабы сэкономить время и память.

```
# Переключаем модель в режим предсказания(тестирования)
model.eval()
y_pred = []
y_true = []
# Отключаем градиенты, чтобы сэкономить память
with torch.no_grad():
   for X_batch, y_batch in test_loader:
       X_batch, y_batch = X_batch.to(device), y_batch.to(device)
       outputs = model(X_batch).squeeze()
       # Конвертируем предсказания в 1 или 0
       y_pred.extend((outputs > 0.5).int().tolist())
       # Конвертируем метки в 1 или 0
       y_true.extend(y_batch.int().tolist())

from sklearn.metrics import f1_score
# Выводим статистику о предсказании модели
print(classification_report(y_true, y_pred))
```

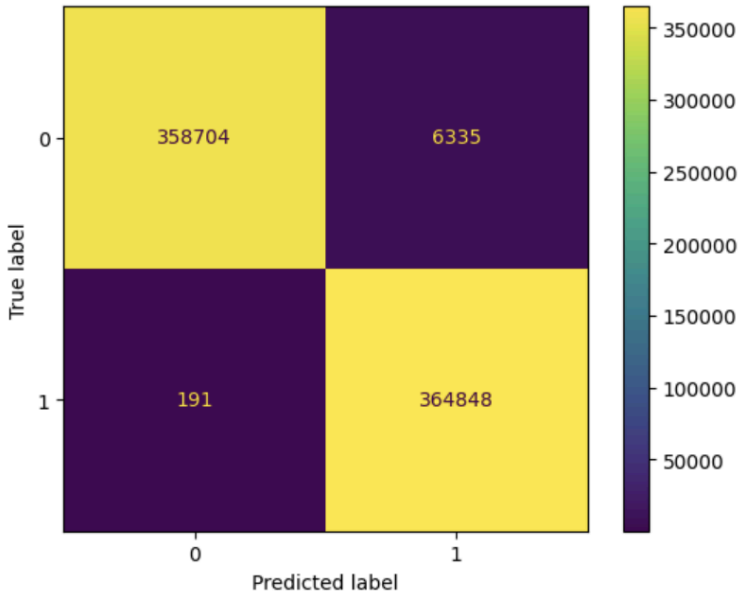
Результат:

	precision	recall	f1-score	support
0	1.00	0.98	0.99	365039
1	0.98	1.00	0.99	365039
accuracy			0.99	730078
macro avg	0.99	0.99	0.99	730078
weighted avg	0.99	0.99	0.99	730078

По результатам видно, что модель почти идеально научилась предсказывать класс транзакции. есть небольшие неточности, но на фоне остальных ответов они нивелируются.

Представим ещё графически эту же информацию

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Создаём матрицу ошибок, сравнивая истинные метки y_test с предсказаниями модели y_pred
cm = confusion_matrix(y_test, y_pred.cpu().numpy())
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
# Выводим матрицу ошибок
disp.plot();
```



Из матрицы видно, что модель очень хорошо справляется с классификацией, но так же есть и ошибки, когда неверно предсказывает метку мошеннической операции. Меньше всего предсказывает ложноотрицательные(по факту мошеннические операции были приняты за не мошеннические), а следом ложноположительные(по факту не мошеннические операции были приняты за мошеннические). Риск ложноположительных не так уж и велик, но всё же присутствует, что может вызывать недовольство у пользователей.
Можно уменьшить количество ложных срабатываний и пропусков путём увеличения данных для обучения.

Модель теперь обучена и протестирована. Она готова предсказывать класс мошеннической или обычной транзакции.

3.6 Моделирование ситуаций

Смоделируем несколько ситуаций, чтобы проверить, как работает нейросеть. Иными словами, придумаем данные под какой-либо небольшой сюжет. Поймёт ли нейросеть класс транзакции
Случай 1:
Пользователь карты уехал в другой город, в котором он уже когда-то бывал. В этом городе он поехал в магазин одежды, где уже покупал что-то не в первый раз. Решил купить там телевизор. При оплате он ввёл пин-код на терминале.
Входные данные:
Расстояние от дома: 50.3
Расстояние от последней покупки: 51.6

Отношение цены покупки к средней цене покупок: 2.3

“Старый” продавец: да

Использование пин-кода: да

Онлайн покупка: нет

Случай 2:

Пользователь карты, будучи на работе, решил заказать своей девушке доставку цветов. Не раз покупал у этого продавца цветы. При покупке в этом магазине у него всегда запрашивают пин-код карты.

Входные данные:

Расстояние от дома: 3.45

Расстояние от последней покупки: 6.0

Отношение цены покупки к средней цене покупок: 1.5

“Старый” продавец: да

Использование пин-кода: да

Онлайн покупка: да

Случай 3:

Пользователь карты находится в командировке в другом регионе. Во время поездки он решил заказать себе дорогой ноутбук в интернет-магазине, с которым он уже имел дело ранее. Однако через пару часов выясняется, что пользователь не совершал этой покупки. При проверке оказалось, что мошенники завладели данными карты и, зная привычки пользователя, сделали дорогостоящую покупку в знакомом магазине, чтобы не вызывать подозрений.

Входные данные:

Расстояние от дома: 500.10

Расстояние от последней покупки: 100.3

Отношение цены покупки к средней цене покупок: 3.5

“Старый” продавец: да

Использование пин-кода: да

Онлайн покупка: да

Случай 4:

Пользователь карты обнаружил списание за покупку ювелирного украшения в интернет-магазине. Ранее он никогда не совершал покупок у этого продавца, и цена оказалась значительно выше его средних трат. Расследование показало, что мошенники использовали данные карты для покупки дорогого товара в новом магазине, не введя пин-код, и выбрали доставку на адрес, не связанный с владельцем карты.

Входные данные:

Расстояние от дома: 100.05

Расстояние от последней покупки: 120.0

Отношение цены покупки к средней цене покупок: 5.0

“Старый” продавец: нет

Использование пин-кода: нет

Онлайн покупка: да

```
# Создаём функцию для предсказания типа транзакции(мошенническая-1 или
нет-0)
def predict(new_data):
    # Выполняем логарифмирование данных, чтобы улучшить их локализацию в
пространстве
    for row_index in range(len(new_data)):
        new_data[row_index][:2] = list(map(lambda x: np.log(x),
new_data[row_index][:2]))
    # Конвертируем данные в тензор PyTorch
    new_test = scaler.transform(new_data)
    new_data_tensor = torch.tensor(new_test, dtype=torch.float32).to(device)
    # Выполняем предсказание
```

```
    with torch.no_grad():  # Выключаем градиенты
        predictions = model(new_data_tensor).squeeze()  # Получаем
вероятности
        predicted_labels = (predictions >= 0.5).int()  # Преобразуем
вероятности в классы (0 или 1)

    # Выводим результаты
    for i, (data, label, prob) in enumerate(zip(new_data, predicted_labels,
predictions)):
        print(f"Пример {i + 1}: Данные: {data}, Вероятность мошенничества:
{prob:.2f}, Метка: {label.item()}")

# Формируем входные данные
new_data = [[50.3, 51.6, 2.3, 1, 1, 0],
            [3.45, 6.0, 1.5, 1, 1, 1],
            [500.10, 100.03, 3.5, 1 , 1 , 1],
            [100.05, 120.0, 5.0, 0 , 0 , 1]]
# Вызываем функцию для предсказания
predict(new_data)
```

```
Пример 1: Данные: [3.9180050771056933, 3.9435216724875173, 2.3, 1, 1, 0], Вероятность мошенничества: 0.00, Метка: 0
Пример 2: Данные: [1.2383742310432684, 1.791759469228055, 1.5, 1, 1, 1], Вероятность мошенничества: 0.00, Метка: 0
Пример 3: Данные: [6.214808078424858, 4.605470140997089, 3.5, 1, 1, 1], Вероятность мошенничества: 0.99, Метка: 1
Пример 4: Данные: [4.605670061029742, 4.787491742782046, 5.0, 0, 0, 1], Вероятность мошенничества: 1.00, Метка: 1
```

Модель правильно распознала операции. 1 и 2 операции были обычными, а 3 и 4 - мошеннические.

Заключение

В этой дипломной работе были изучены, изучены и применены современные методы обнаружения мошенничества в финансовых транзакциях с использованием нейронных сетей. Основной целью исследования было создание эффективной модели нейронной сети, способной обнаруживать мошеннические транзакции.

1. Анализ предметной области

В теоретической части данной работы я рассмотрел основные элементы финансовых транзакций и классификации мошенничества. Также мной были рассмотрены как традиционные, так и современные методы обнаружения мошеннической активности. Рассмотрены были и методы, использующие нейросети. Преимущества нейронных сетей особо выделяются, поскольку они обеспечивают высокую гибкость и точность при анализе сложных данных и данных больших объёмов.

Кроме того, была проанализирована структура данных о финансовых транзакциях.
Были выделены важные проблемы, такие как выбросы, дисбалансы классов и аномалии. Описаны методы подготовки данных, такие как балансировка, нормализация и очистка классов, которые имеют решающее значение для обучения моделей.

2. Используемые методы оценки моделей и технологии

Методы оценки качества моделей получили особое внимание. При анализе несбалансированных данных представлены основные метрики (точность, полнота, F1-мера, ROC-AUC) и их ограничения. Был проведен обзор уже

существующих библиотек и инструментов, таких как Scikit-learn, PyTorch и TensorFlow, а также распределенных технологий, которые позволяют обрабатывать большие объемы информации.

3. Практический аспект

Поиск и анализ датасета, выбор инструментов для реализации и обработки данных были частью практической части. Этап предварительного анализа данных (EDA), который определяет закономерности структуры транзакций, заслуживает особого внимания.

Для решения проблемы несбалансированных данных использовались методы SMOTE и оверсэмплинг и андерсэмплинг. Модель нейронной сети с обучением и тестированием была создана на основе обработанного датасета. Значения метрик в тестовых данных подтверждают точность и универсальность итоговой модели.

Смоделированные сценарии мошенничества также были использованы для оценки эффективности модели в условиях, максимально приближенных к реальной жизни.

Выводы

В результате исследования можно прийти к следующим выводам:

- Методы обнаружения мошенничества, основанные на нейронных сетях, могут повысить безопасность финансовых операций.

- Для достижения высокой эффективности моделей предварительная обработка данных, решение проблемы дисбаланса классов и выбор подходящих архитектур нейронных сетей являются важными шагами.

- В условиях несбалансированных данных традиционные метрики оценки моделей не всегда адекватно отражают качество, поэтому важно использовать комплексный подход к интерпретации результатов.

Рекомендации

Рекомендуется улучшить модель и адаптировать ее к реальным условиям:

1. Использовать более крупные и разнообразные датасеты, отражающие различные типы мошенничества.
2. Исследовать возможности использования ансамблевых подходов для повышения точности и стабильности модели.
3. Внедрить методы объяснимого ИИ для интерпретации решений модели, что особенно важно для финансовых рынков.

В этом исследовании показано, что использование нейронных сетей для анализа финансовых транзакций открывает возможности для улучшения систем безопасности и предупреждения мошенничества, что может быть полезно как банкам, так и клиентам.

Список используемой литературы

- Книга: Tom Fawcett. Data Science for Business. Sebastopol: O'Reilly Media, 2013.
- Книга: Антон Пухов. Мошенничество в платежной сфере. Бизнес-энциклопедия. Москва: Интеллектуальная Литература, 2015.
- Книга: Ян Гудфеллоу, Иошуа Бенджио, Аарон Курвилль. Глубокое обучение. Москва: ДМК-Пресс, 2018 г.
- Книга: Andreas C. Mueller, Sarah Guido. Introduction to Machine Learning with Python. Sebastopol: O'Reilly Media, 2016.
- PyTorch. Документация по библиотеке для машинного обучения на языке python. URL: https://pytorch.org/docs/stable/index.html (дата обращения 10.09.2024)
- Scikit-learn. Документация по библиотеке для машинного обучения на языке python. URL: https://scikit-learn.org/stable/user_guide.html (дата обращения 3.09.2024)
- Pyspark. Документация по библиотеке для обработки больших данных. URL: https://spark.apache.org/docs/latest/api/python/index.html (дата обращения 29.11.2024)
- Статья: oomoo. “Выбор слоя активации в нейронных сетях: как правильно выбрать для вашей задачи”. URL: https://habr.com/ru/articles/727506/. 2023
- Банковская транзакция**.** Википедия: Свободная энциклопедия.URL:https://ru.wikipedia.org/wiki/%D0%91%D0%B0%

D0%BD%D0%BA%D0%BE%D0%B2%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F (дата обращения 28.07.2024).

- Мошенничество. Википедия: Свободная энциклопедия.URL:https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D1%88%D0%B5%D0%BD%D0%BD%D0%B8%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%BE (дата обращения 28.07.2024).

Приложение

Приложение 1

Название	Описание	Преимущества	Недостатки
Логистическая регрессия	Простая и эффективная модель для бинарной классификации. Обычно используется для предсказания вероятности мошенничества на основе заранее известных признаков.	Быстрая и интерпретируемая модель	Не всегда может справляться с более сложными паттернами данных.
Деревья решений и случайный лес	Модели, которые хорошо работают с категорическими данными и могут легко интерпретироваться. Случайный лес может повысить точность, используя ансамбль решений.	Хорошая интерпретируемость и устойчивость к переобучению.	Могут быть склонны к высокому времени вычислений при обработке больших данных

Методы опорных векторов (SVM)	Часто используются для классификации в случаях с небольшими объемами данных и высокой размерностью.	Отличная производительность для двоичной классификации	Трудности с масштабированием на большие объемы данных.
Глубокие нейронные сети	Модели, использующие несколько слоев нейронов для извлечения сложных паттернов. Применяются в задачах с большим количеством данных и признаков.	Отлично подходят для сложных, высокоразмерных данных.	Требуют значительных вычислительных ресурсов и времени для обучения.
Алгоритмы на основе ансамблей	Например, XGBoost или LightGBM. Эти методы использует ансамбль слабых моделей для построения более точной предсказательной модели.	Высокая точность и способность работать с дисбалансом классов.	Могут быть трудными для интерпретации.

Рекуррентные нейронные сети (RNN), LSTM	Эти сети полезны для анализа временных рядов, таких как транзакции, поскольку могут учитывать зависимость текущего события от предыдущих.	Хорошо справляются с задачами, где данные зависят от предыдущих событий.	Требуют больших вычислительных ресурсов и времени для обучения.

Приложение 2

Исходный датасет:

https://openml.org/search?type=data&status=active&id=45955

ссылка на проект в Google Colab:

https://colab.research.google.com/drive/18bfGbQ6BDR3Czcwzj1_ql1eLZ2DLtUPW?usp=sharing

ссылка на github репозиторий:

https://github.com/ZmanDot/Diploma_GB.git