МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра информационно-аналитических систем безопасности

К защите допустить:

Зав. кафедрой _____ А.Н. Целых

"_____"_____201_ г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему

«Выявление мошеннических транзакций

методами машинного обучения при несбалансированной выборке»

Научный руководитель _____ Беляков С.Л.

(подпись)

Студент гр. КТбо 4-9 _____ Карпов С.М.

(подпись)

«___» _____ 201_ г.

«Выявление мошеннических транзакций
методами машинного обучения при
несбалансированной выборке»

Карпов Станислав Максимович
Группа КТбо4-9

Выпускная квалификационная работа

ЮФУ, Таганрог 2019.

АННОТАЦИЯ

Данная выпускная квалификационная работа посвящена актуальной проблеме обнаружения мошеннических транзакций на несбалансированных данных.

Актуальность данной темы работы определяется необходимостью существенно повысить эффективность современных методов выявления злоумышленных действий в реальных банковских операциях с помощью машинного обучения.

В рамках поставленной задачи, была реализована избыточная и недостаточная выборка, на основе которых, исследовались модели машинного обучения. В качестве моделей использовались такие классификаторы как метод опорных векторов, логистическая регрессия, алгоритм К-ближайших соседей и случайный лес. Также была реализована нейронная сеть на архитектуре многослойный перцептрон. В завершении проведен сводный анализ результатов работы получившихся моделей.

Система выявления мошеннических транзакций спроектирована на языке программирования Python. В качестве механизма обучения используется подход – обучение с учителем. Реализован случайный поиск параметров моделей машинного обучения по заранее заданной сетке. Также во избежание переобучения модели применяется перекрестная проверка.

«Identify fraudulent transactions using machine learning methods for unbalanced data»

Karpov Stanislav Maximovich

Group KTbo4-9

Final graduation work

SFU, Taganrog 2019.

ANNOTATION

This final graduation work deals with the actual problem of detecting fraud transactions on unbalanced data.

The topicality of this work is determined by the necessity to improve significantly the efficiency of modern methods for identifying malicious actions in real banking operations using machine learning.

To achieve the task, an oversample and undersample was implemented. It was used to study machine learning teaching models. These are the support vector method, logistic regression, the K-nearest-neighbor algorithm, and the random forest models. A neural network on a multilayer perceptron architecture was also implemented. In conclusion, a summary analysis of the results of the working models was done.

The fraud detection system is designed in the Python programming language. A learning with a teacher is used as a learning approach. a random search for the parameters of machine learning models was implemented for a predetermined grid. Also, in order to avoid the model retraining a cross-validation is applied.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Мошенничество известно человечеству с давних времен и может принимать практически бесконечное разнообразие форм. Однако в последние годы разработка и внедрение инновационных технологий, открыло иные пути для злоумышленников. К традиционным формам мошенничества, например, как отмывание денег присоединились и новые виды, такие как мошенничество с банковскими картами.

Спектр направления действий злоумышленников достаточно велик. Это и классические методы такие как встраиваемый вредоносный код, фальсификация документов, кража данных авторизации и CVV-кодов. Так и более нестандартные операции (например, отмыв денег) которые при поверхностном рассмотрении не вызывают подозрений.

В среднем компании ежегодно теряет около 5 процентов своих доходов из-за мошенничества. Так, по прогнозам экспертов, уже к 2020 году объем мошеннических операций с кредитными картами вырастет до 32 миллиардов долларов.

Ныне практически все системы мониторинга мошеннических транзакций, работают в режиме реального времени и способны распознать действия злоумышленников еще на этапе рассмотрения транзакции. Однако функционируют такие комплексы по-разному.

До недавнего времени в качества основного метода в борьбе с мошенниками применялись системы, построенные на специальных правилах и логических условиях. В таких комплексах прежде, чем транзакция будет одобрена или отклонена проводится ряд операций. Сначала осуществляется первичный анализ платежа по таким параметрам как местонахождение клиента, сумма транзакции, история платежей и другие. Если все в порядке, то транзакция подтверждается. Если системе что-то показалось подозрительным, то транзакция отправляется на следующий этап. И так в зависимости от сложности системы, после

выполнения или не выполнения условия транзакциям присваиваются метки уровня риска, например «подозрительная» или «низкий уровень риска». А в завершении все финансовые операции, вызвавшие подозрение у системы, отправляются на ручную проверку.

В настоящее время, все большую популярность набирают системы мониторинга мошеннических транзакций, основанные на машинном обучении. В таких системах, программа работает автоматизировано, и по своей структуре является не набором правил и условий, а искусственным интеллектом. Отличной особенностью которого является способность к обучению. То есть на основе данных прошлых финансовых операций, система мониторинга анализирует поступившую на вход транзакцию и автоматически принимает решение об одобрении или отклонении.

Учитывая нынешние условия бизнеса: жесткую конкурентную среду, снижение маржи и усиление давления со стороны акционеров, все больше и больше компаний рассматривают выявление и предотвращение мошенничества в качестве ключевого стратегического приоритета. Чтобы быть на шаг впереди злоумышленников, которые продолжают совершенствовать свою тактику благодаря новым технологиям (например, интернет вещей), фирмам нужны новые, эффективные и действенные инструменты.

ГЛАВА 1 ОБЗОР СОВРЕМЕННЫХ ПОДХОДОВ К РЕШЕНИЮ ПРОБЛЕМЫ ОБНАРУЖЕНИЯ МОШЕННИЧЕСТВА

1.1 Проблема обнаружения мошенничества

Для начала ясно разграничим понятия предотвращение и обнаружение мошенничества.

Предотвращение мошенничества – это набор приемов, правил и методов, направленные на предотвращение противоправных действий. Например, в сфере банковских карт это надежная система идентификации и аутентификации пользователей. Сюда же относится и специальная конструкция банковской карты и особые правила функционирования терминалов.

Обнаружение мошенничества – это процедура выявления мошенничества в случае, если не удалось предотвратить незаконные действия на первом этапе. На практике, обнаружение мошенничества ведется в режиме реального времени, ведь не всегда достоверно известно, что этап предотвращения мошенничества был преодолен.

Проблема выявления мошеннических транзакций заключается в создании такой системы обнаружения, которая могла бы максимально точно и быстро классифицировать ту или иную транзакцию на основе некоторых данных.

Всякий раз, когда очередная уязвимость становится известна, преступники улучшают свою стратегию и разрабатывают новый подход. А потому постоянно ведется улучшение методов предотвращения и обнаружения мошенничества. Данные понятия неразрывно связаны друг с другом. Так выявление незаконных дел позволяет понять эффективность первого этапа и повысить надежность системы в целом. Потому развитие методов и инструментов обнаружения мошенничества является одной из приоритетных задач для финансовых организаций в мире.

1.2 Способы обнаружения мошеннических транзакций

Существует два подхода к проблеме выявления мошенничества:

– Не контролируемые методы

– Контролируемые методы

Неконтролируемые методы используются в тех случаях, когда отсутствуют предварительные наборы честных и мошеннических транзакций. Используемые здесь алгоритмы обычно представляют собой комбинацию методов профилирования и обнаружения выбросов. То есть моделируется некое базовое распределение, которое принимается как нормальное поведение. А значениям, показывающим наибольшее отклонение, уделяется повышенное внимание.

Пример такого алгоритма представлен на рисунке 1.

Рисунок 1 – Пример алгоритма неконтролируемого метода

Одним из примеров такого метода является цифровой анализ с использованием закона Бенфорда, представленный на рисунке 2. Закон Бенфорда гласит, что распределение первых значащих цифр чисел, взятых из широкого спектра случайных распределений, будет иметь (асимптотически) определенную форму.

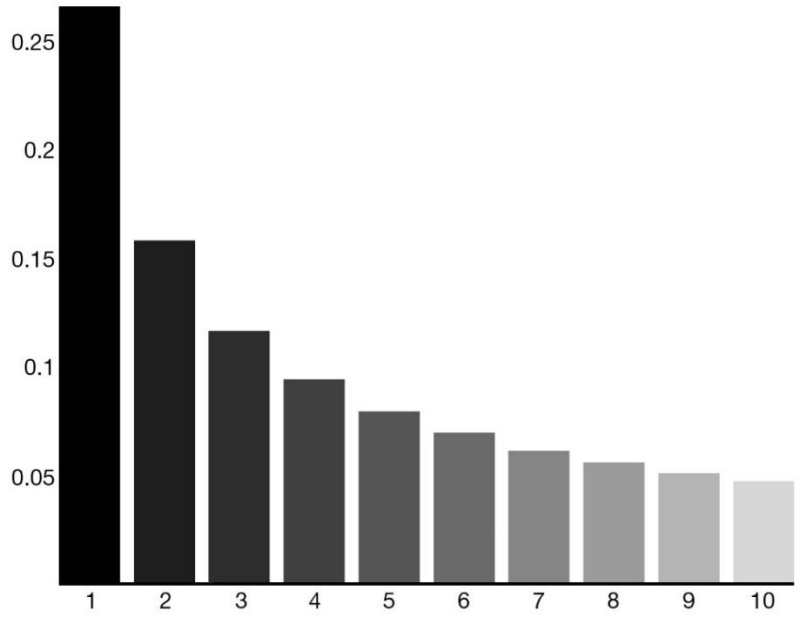


Рисунок 2 – Частота появления первой цифры по Бенфорду

Однако применение этого закона для обнаружения мошенничества зачастую затрудненно в связи со сложностью подготовки данных.

Контролируемые методы используют базу данных известных мошеннических дел, из которой можно построить модель, и получить оценку подозрительности для новых дел. Здесь широкое распространение получили методы, основанные на специальных правилах вида If-Then-Else. То есть, когда задаются определенные условия, при выполнении которых предпринимается дальнейшая цепочка действия. Часто такая цепочка завершается ручной проверкой классифицированных дел. Такой метод имеет ряд недостатков основные из которых: низкая скорость, высокая затратность и вероятность ошибки. До недавнего времени именно алгоритм, основанный на правилах, применялся крупнейшими финансовыми компаниями.

Однако в последнее время все больший оборот набирает применение машинного обучения. Здесь в качестве моделей используются как традиционные методы статистической классификации (деревья решений, k-ближайших соседей), так и более мощные инструменты, например нейронные сети.

1.3 Системы обнаружения мошеннических транзакций основанные на правилах

В прошлом подавляющее большинство платформ для обнаружения мошенничества использовали набор специальных правил и логических выражений для обнаружения подозрительных транзакций, чтобы потом направлять их на ручную проверку. Удивительно, но этот традиционный подход все еще используется в некоторых компаниях.

Такие системы обладают рядом серьезных недостатков:

1. **Дороговизна.** Содержание персонала, занимающегося ручной проверкой, обучение сотрудников ручному анализу – все это требует больших расходов компании.
2. **Время**. Ручная проверка как бы не была грамотно построена, она всегда будет упираться в человеческую производительность. А при увеличении числа клиентов и как следствие транзакций, время ожидания становится критически долгим.
3. **Ложные срабатывания.** Из-за применения жестких правил, порогов, в случае их превышения даже на незначительную величину сотрудник будет вынужден отклонить ту или иную транзакцию. А большое количество ложны срабатывания приведет к оттоку клиентов и снижению репутации компании.

Таким образом, применение ручных проверок, основанных на правилах в качестве основного инструмента в проблеме обнаружения мошенничества нецелесообразно. Нецелесообразно из-за наличия более прогрессивных и действенных методов. Единственное применение описанной выше системы – это дополнительная линия защита в комплексе борьбы с мошенничеством.

1.4 Машинное обучение в системах обнаружения нечестных транзакций

Разнообразие подходов к обнаружению мошеннических транзакций не ограничивается лишь контролируемыми и неконтролируемыми методами. Многие компании разрабатывают свои уникальные инструменты основываясь на предъявляемых в этой организации требованиях и эксплуатационных характеристиках. Однако все чаще финансовые корпорации склоняются к тому, что наиболее эффективной стратегией борьбы со злоумышленниками является применение методов машинного обучения.

Машинное обучение – большой класс методов искусственного интеллекта. Ключевое отличие от классических подходов заключается в том, что модель решения задачи, способна обучаться и предсказывать результаты на новых данных. Применение такого класса методов искусственного интеллекта упрощает обработку крайне больших наборов данных.

Машинное обучение позволяет обнаруживать незаметные человеческому глазу зависимости и паттерны и преобразовывать эту информацию в простой формат, достаточно понятный для человека. Особенно актуально это для случаев, когда результат требуется получить в течении короткого промежутка времени. После обучения на тренировочном наборе данных, модель машинного обучения способна предсказывать результат и на новом наборе данных, который не рассматривался в процессе обучения. Далее, накапливая информацию и опыт, система развивается и становится более совершенной и точной в своих оценках.

Повсеместные исследования и применение такого метода в борьбе с мошенничеством становится очевидным если подробно рассмотреть преимущества, которые дает данный подход.

1.4.1 Скорость

По мере роста скорости торговли очень важно иметь более быстрое решение. Потому что современный бизнес требует быстрых результатов, где счет идет на микросекунды. И только методы машинного обучения позволяют достичь этого с таким высоким уровнем точности. Машинное обучение может обрабатывать огромное количество информации в единицу времени. Это несравнимо с человеческими способностями, особенно в задачах, когда требуется оценивать огромное количество транзакций в режиме реального времени.

1.4.2 Масштаб

Алгоритмы и модели машинного обучения становятся более эффективными с увеличением наборов данных. Важно понимать, что в моделях на основе правил стоимость обслуживания системы обнаружения мошенничества увеличивается с увеличением клиентской базы. Стоимость обслуживания системы на машинном обучении не зависит от объема данных, более того, при увеличении клиентской базы и потока информации такая модель начинает работать более точно. Однако если такая модель при обучении пропустит реальный случай мошенничества, есть риск, что в будущем этот тип мошенничества будет игнорироваться. Поэтому следующим важным фактором является эффективность.

1.4.3 Эффективность

В отличие от людей, методы машинного обучения умеют выявлять различные паттерны, корреляции и другие скрытые данные. Такая система постоянно анализирует и обрабатывает новые данные. Более того, передовые модели, такие как нейронные сети, могут автономно обновлять свои модели, чтобы запомнить последние тенденции и действовать более эффективно.

1.5 Алгоритмы классификации, используемые в качестве моделей машинного обучения

1.5.1 Деревья решений

Данный алгоритм представляет собой иерархичную последовательную структуру с узлами. В каждом таком узлу, присутствует логическое условие, а ответ на это условие является выходом. Деревья решений применяются как для решения задачи регрессии, классификации так и просто для описания данных, представляя собой достаточно компактную форму хранения информации.

Рисунок 3 иллюстрирует примерную схему алгоритма деревьев решения.

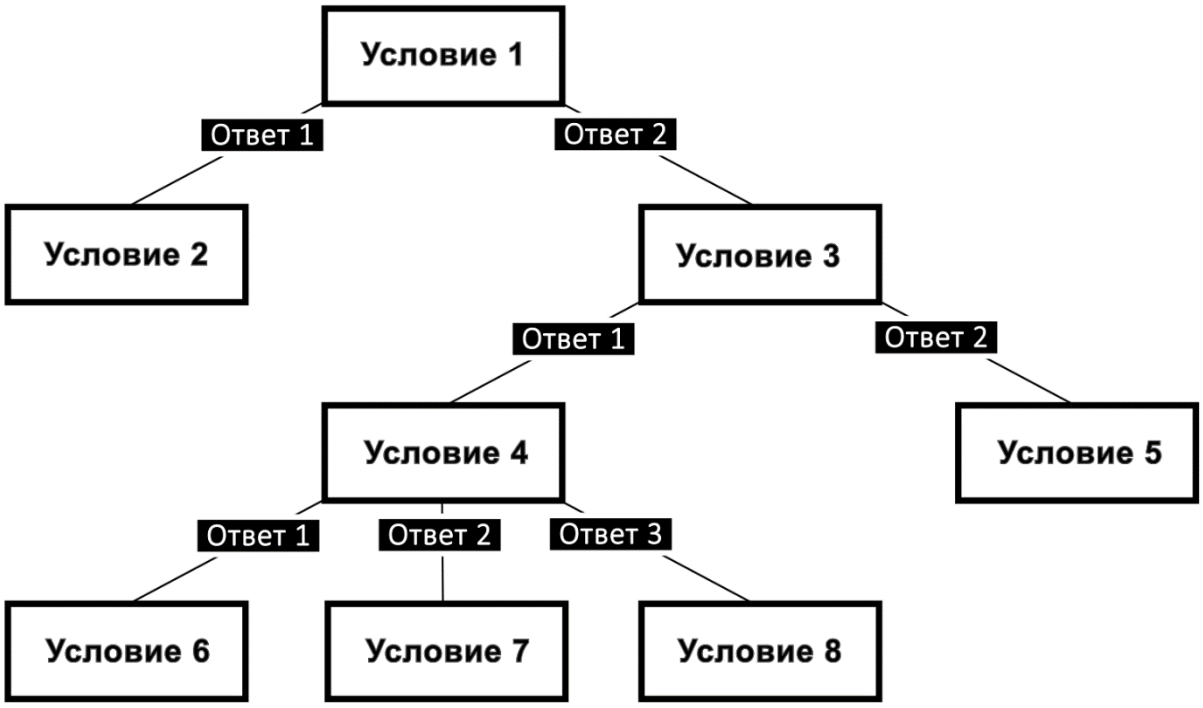


Рисунок 3 – Деревья решений

1.5.2 Случайный лес

Случайный лес также использует как для задач регрессии, так и для задач классификации. На рисунке 4 видно, что данный алгоритм состоит из нескольких других алгоритмов. Такой метод, называется ансамблевым и

представляет собой несколько алгоритмов типа «деревья решения» из чего и образовывается его название – случайный лес. Таким образом объединение позволяет создавать более мощные модели прогнозирования.

Из плюсов данного алгоритма можно выделить то, что он не является предвзятым. Так как существует несколько деревьев, и каждое такое дерево обучается на подмножестве данных. То есть, по сути, случайный лес опирается на силу «толпы», а потому общая предвзятость алгоритма снижается. Пожалуй, единственным минусом данного алгоритма является его вычислительная затратность, из-за чего время обучения получается довольно долгим.

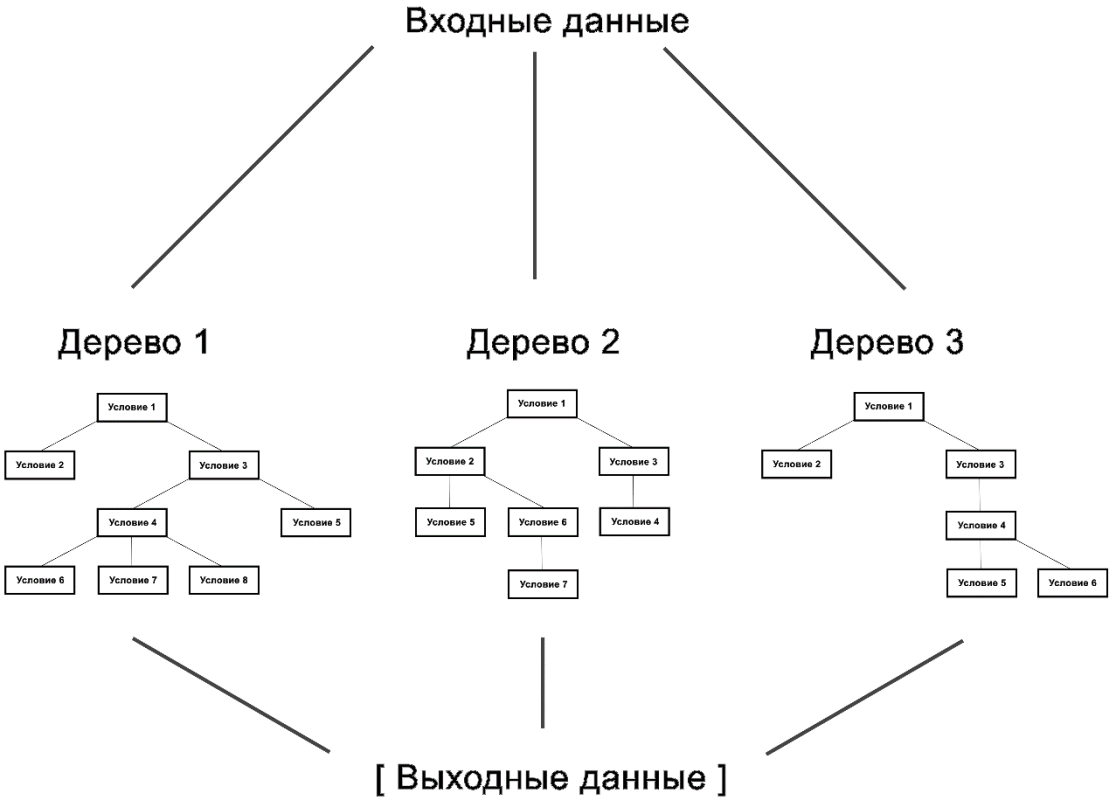


Рисунок 4 – Деревья решений

1.5.3 Метод опорных векторов

Основная идея метода опорных векторов заключается в поиске такого правильного вектора или как говорят гиперплоскости, который разделит имеющиеся данные на два класса. Сначала строятся опорные векторы –

линии разделения, наиболее близко расположенные возле своих классов. Потом вычисляется расстояние между опорными векторами, именуемое «зазором». Чем больше зазор и меньше объектов классов в нем, тем эффективнее модель. Упрощенная схема метода опорных векторов представлена на рисунке 5

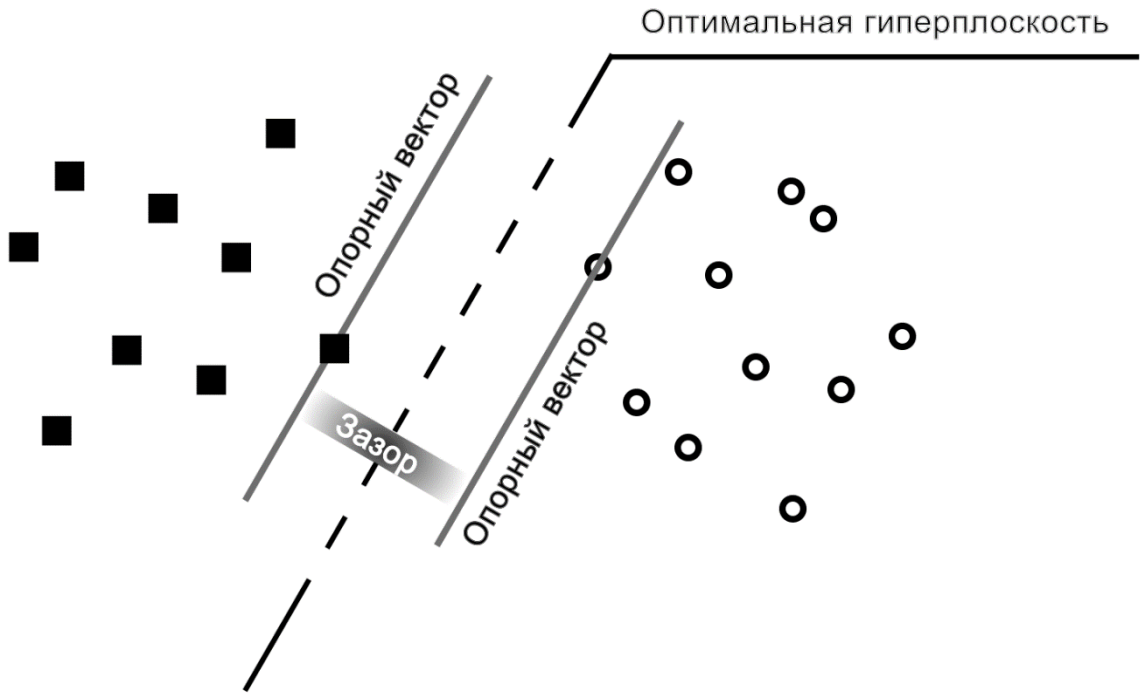


Рисунок 5 – Метод опорных векторов

1.5.4 K-ближайших соседей

Классический, один из наиболее часто встречаемых алгоритмов. Несмотря на то, что метод довольно простой, порой он способен выдавать достойные результаты на уровне самых сложных моделей.

Алгоритм работы метода К-ближайших соседей:

1. Сначала выполняется трассировка расстояния между всеми объектами в обучающем наборе данных.
2. Далее среди этих объектов выбирается такое k – количество объектов, расстояние между которыми минимально.
3. Итоговый класс предсказывается как класс, наиболее часто встречаемый среди выбранных k-объектов.

Более наглядно схема работы данного алгоритма представлена на рисунке 6.

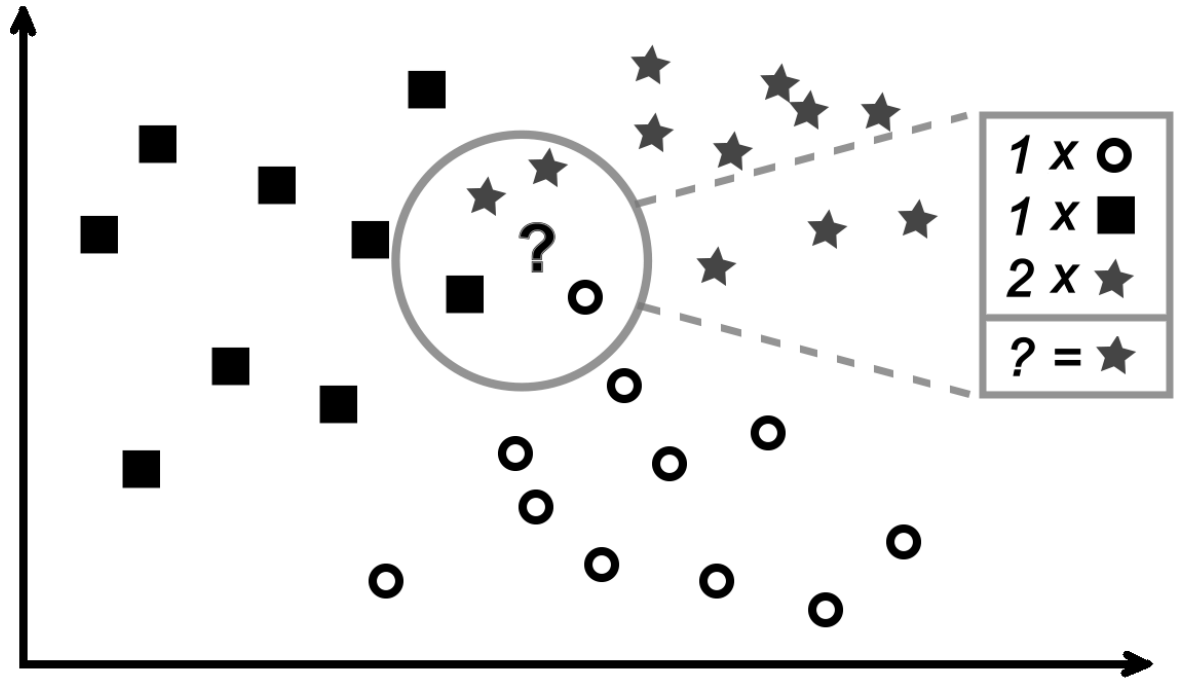


Рисунок 6 – Метод k-ближайших соседей

1.5.5 Наивный байесовский классификатор

Достаточно сильный инструмент в машинном обучении. Байесовский классификатор предсказывает метки классов по векторам признаков. Данный алгоритм основывается на одноименно теореме и представляет собой апостериорную (условную) вероятностную модель. То есть, вероятность предсказывается на основе данных, получаемых после проведения некоторых опытов. При этом, байесовский классификатор не учитывает корреляции между признаками, считая, что они влияют на итоговую оценку независимо. Отсюда и добавляется приставка «наивный».

1.5.6 Нейронные сети

Такие модели машинного обучения представляют собой достаточно большой раздел машинного обучения и насчитывает огромное количество

самых разных типов нейронных сетей. Различие их заключается в количестве, слоев, нейронов, используемых функциях активации и алгоритмах обучения. Преимущество нейронных сетей в том, что они отлично распознают как линейные, так и нелинейные отношения в обучаемом наборе данных.

На рисунке 7 представлена упрощенная архитектура нейронной сети.

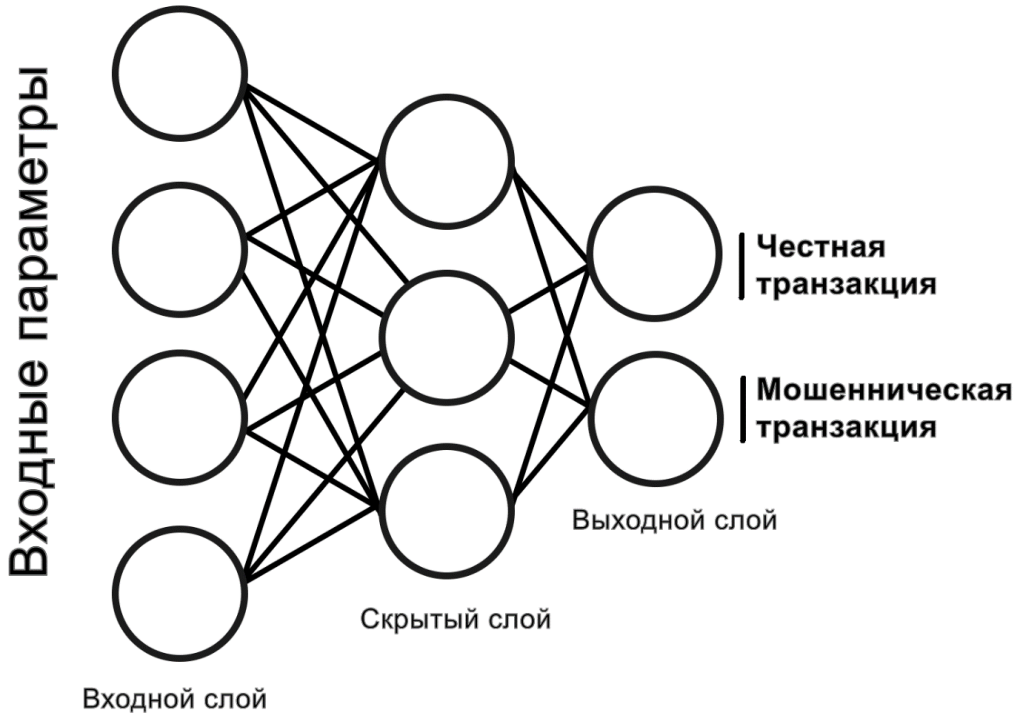


Рисунок 7 – Схема нейронной сети

Каждая транзакция имеет набор признаков, таких как дата, сумма транзакции, данные о клиенте и др.

1) Признак (информация которую он содержит) поступает на вход нейрона. Нейрон представляет собой так называемую вычислительную единицу нейронной сети. Он осуществляет некоторые арифметические операции над информацией.
2) К поступившей информации применяется функция активации. Её предназначение – нормализация данных.
3) Далее через синапс информация передается следующему нейрону.

Синапс (синаптический вес) – это способ связи между нейронами. Главным параметром синапса является его вес. Именно благодаря весам, исходная информация преобразуется в итоговый результат.

Существует 3 парадигмы обучения:

- **Обучение с учителем**. Когда тренировочные данные изначально размечены.
- **Обучение без учителя**. Нейронная сеть сама пытается классифицировать имеющиеся данные по их глубинным параметрам и структуре.
- **Смешанное обучение.** Суть заключается в том, что нейронная сеть на вход получает ту информацию, которая должна быть на выходе. Если модель предсказывает неправильно, значения весов корректируют и пробуют снова.

Из определения парадигм обучения выходит понятие алгоритма обучения.

Вот самые популярные из них:

- **Управляемое обучение**. Данный способ предполагает изменение весов в ручном режиме с постоянным отслеживанием результата.
- **Алгоритм обратного распространения ошибок**. Так, в сети выделятся два потока, по одному из которых движется входной сигнал. По достижении результата становится известна величина ошибки, которая движется по второму пути в обратном направлении, корректируя весовые коэффициенты.

1.6 Ограничения использования машинного обучения для обнаружения мошенничества

Машинное обучение не является панацеей для выявления мошенничества. Это очень полезная технология, которая позволяет находить аномалии в транзакциях и точно их классифицировать. Она

действительно превосходят человеческий анализ и другие методы, которые использовались прежде. Но у этой методики обнаружения мошенничества есть свои ограничения.

1.6.1 Проверяемость результатов

Это ограничение тесно связано с эффективностью машинного обучения. Из-за того, что машина способна видеть зависимости недоступные человеческому разуму, порой трудно понять, почему та или иная транзакция была помечена именно так. А ведь для того, чтобы иметь достаточные аргументы против нечестного клиента требуется объяснить причины, по которым он был помечен как мошенник.

1.6.2 Холодный старт

Чтобы модель машинного обучения была точной требуется значительное количество данных. Без соответствующих данных машины могут усвоить неправильные выводы и сделать ошибочные или неуместные оценки мошенничества. Для крупных организаций это не является проблемой, для других же, лучше сначала применить базовый набор правил и позволить моделям машинного обучения «разогреваться» чтобы потом работать с большим количеством данных.

1.6.3 Несбалансированные данные

Одной из ключевых проблем на пути применения контролируемых методов таких как машинное обучение являются несбалансированные данные. Дело в том, что в реальности число мошеннических транзакций критически мало в сравнении с законными операциями. А большинство классификаторов неспособно правильно работать в условиях дисбаланса данных. Все это приводит к крайне низкой точности предсказаний. Что сводит на нет все предполагаемые преимущества таких систем обнаружения мошенничества.

1.7 Проблема несбалансированных данных в машинном обучении на примере проблемы выявления нечестных транзакций

Суть проблемы несбалансированных данных заключается в том, что большинство классификаторов неспособно правильно работать с неравномерным распределением целевого класса. А в проблеме мошеннических транзакций, реальные данные всегда распределены неравномерно. Это очень хорошо видно если взглянуть на рисунок 8.

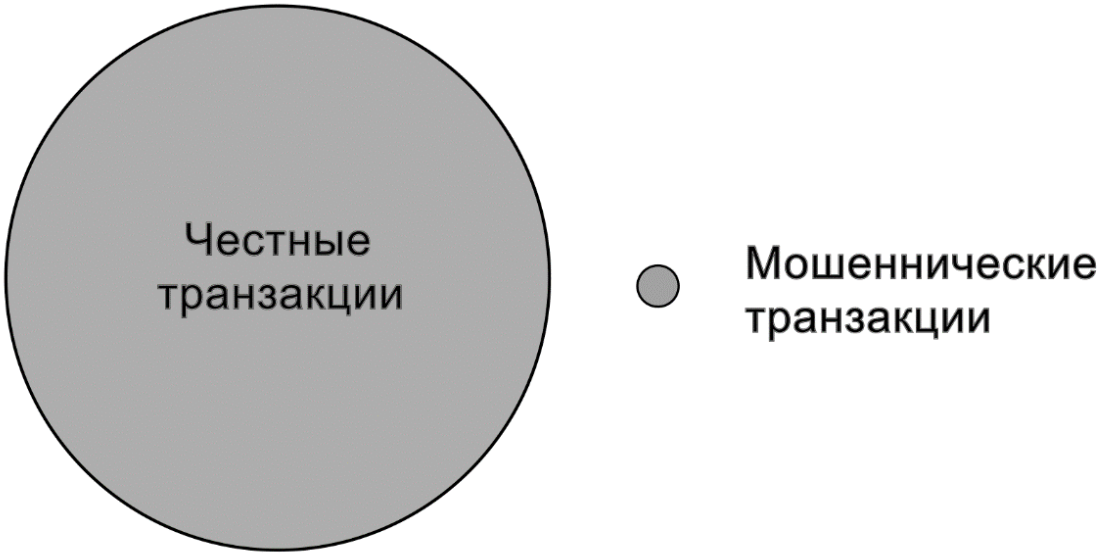


Рисунок 8 – Распределение банковских транзакций

Так, при обучения происходит смещение в сторону мажоритарного класса, потому что функция потерь в любом таком классификаторе старается оптимизировать частоту появления ошибки, не принимая во внимание распределение данных. В худшем случае примеры меньшего класса вообще рассматриваются как выбросы из класса большинства и игнорируются. Если такую модель применить на новых данных, то она отнесет все значения к доминирующему классу, практически абсолютно игнорируя класс меньшинства. А именно точное определение класса меньшинства и является приоритетной целью системы обнаружения мошенничества.

Основным способом решения проблемы несбалансированных данных является повторная выборка с целью компенсировать дисбаланс. Глобально методы повторной выборки делят на две категории: недостаточная и избыточная. Также иногда применяют их сочетание.

1.7.1 Undersampling

Недостаточная выборка (Undersampling) – уменьшение числа примеров мажоритарного класса, до уровня класса меньшинства.

Как правило, такая выборка сильно снижает количество данных для обучения, что может сказаться на результатах.

Иллюстрация изменения соотношения данных представлена на рисунке 9.

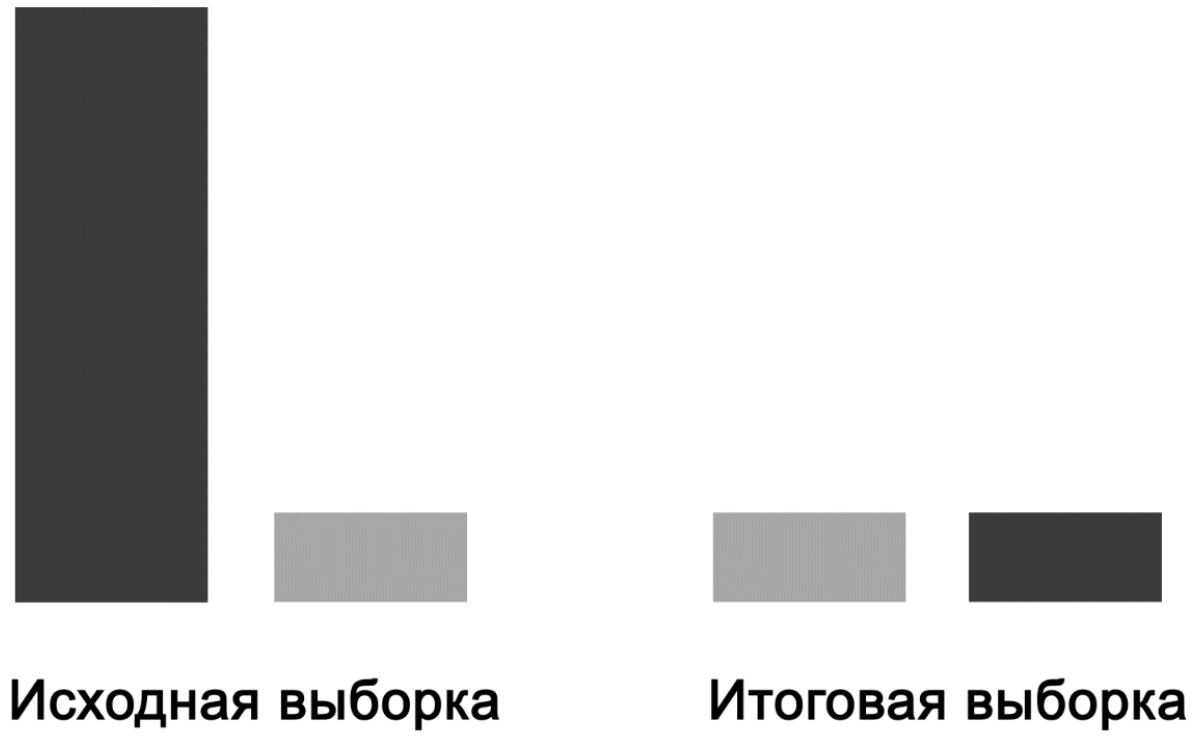


Рисунок 9 – Недостаточная выбора класса большинства

1.7.2 Oversampling

Избыточная выборка (Oversampling) – увеличение числа примеров миноритарного класса до уровня класса большинства[6-8].

Как правило, избыточная выборка дает более точные результаты, но более требовательна к вычислительной мощности, что сказывается на скорости работы.

Пример работы алгоритма избыточной выборки представлен на рисунке 10.

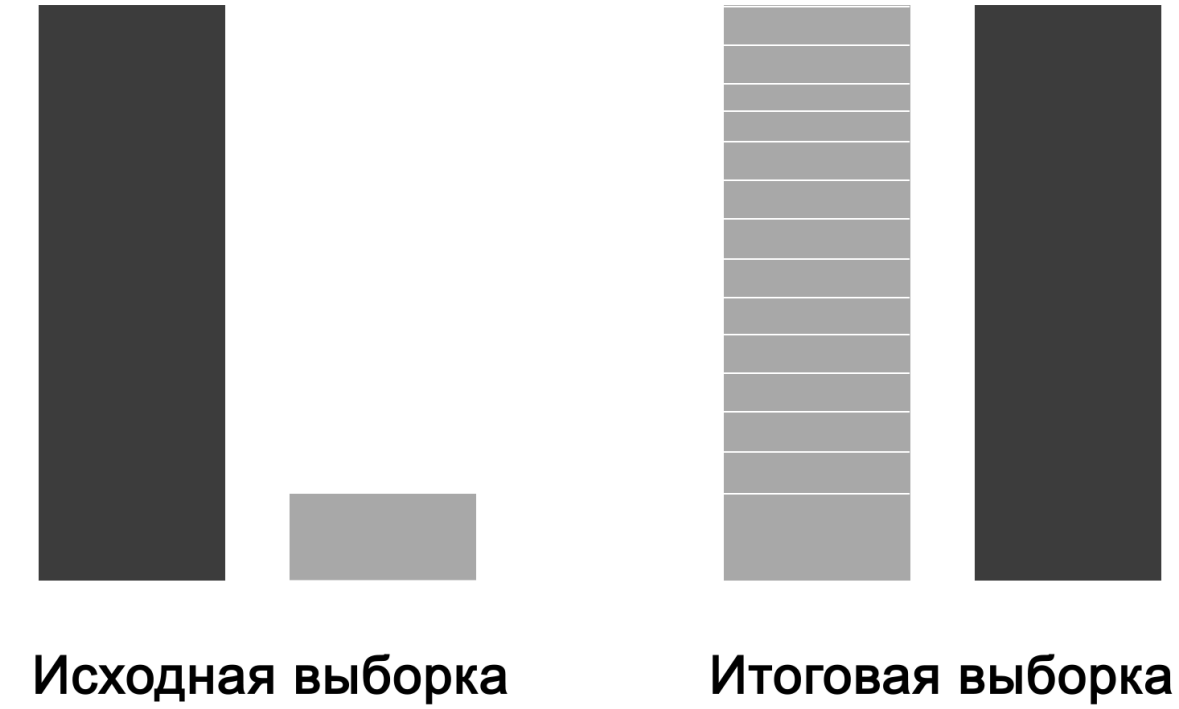


Рисунок 10 – Избыточная выборка класса меньшинства

Каждый из этих подходов содержит в себе отдельные алгоритмы изменения репрезентативности классов. Мы рассмотрим наиболее часто используемые методы.

1.8 Современные подходы к преодолению проблемы несбалансированных классов

1.8.1 Избыточная выборка

SMOTE (Synthetic Minority Over-sampling Technique)

Алгоритм SMOTE заключается в том, что класс меньшинства подвергается избыточной выборке путем создания «синтетических» примеров. Искусственные примеры создаются при помощи алгоритма k-ближайших соседей. При этом синтетические примеры никогда не дублируют объекты класса меньшинства.

Рисунок 11 подробно иллюстрирует схему работы алгоритма SMOTE.

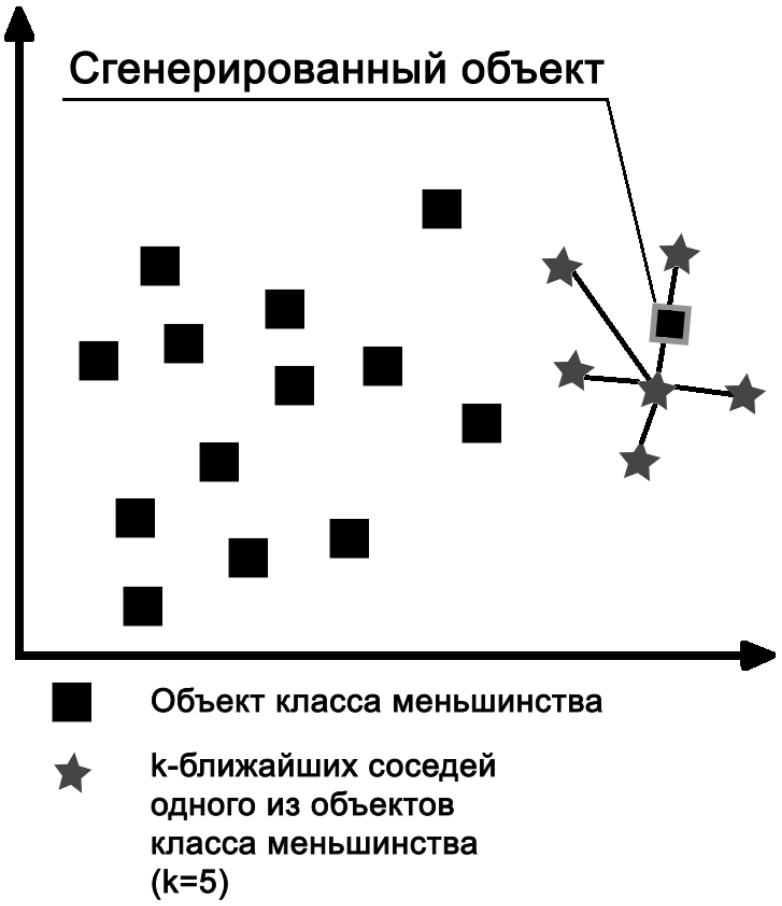


Рисунок 11 – Алгоритм SMOTE (regular)

На практике применяют 4 разновидности SMOTE алгоритма: borderline1, borderline2, regular, SVM.

- Borderline разновидности создают синтетические примеры вблизи объектов на границе классов.
- SVM использует опорный вектор для определения границы.
- Regular использует соотношение соседей принадлежащих разным классам с помощью kNN-алгоритма.

ADASYN (Adaptive Synthetic Sampling)

Основная идея алгоритма ADASYN заключается в использовании параметра распределения плотности данных в качестве критерия для автоматического определения количества синтетических выборок. То есть чтобы компенсировать разницу распределений ADASYN обеспечивает разные веса для разных примеров миноритарного класса. Результирующий набор данных после ADASYN не только обеспечивает сбалансированное представление распределения данных, но также заставляет алгоритм обучения сосредоточиться на этих трудных для изучения примерах. Это является существенным отличием по сравнению с алгоритмом SMOTE, в котором для каждого примера данных меньшинства генерируется равное

1.8.2 Недостаточная выборка

RandomUnderSampler

RandomUnderSampler представляет собой алгоритм случайного выбора данных из класса большинства. Это один из самых простых способов сбалансировать класс. Также данный метод поддерживает независимую повторную выборку с учетом каждого целевого класса.

NearMiss

NearMiss — это метод недостаточной выборки, основанный на алгоритме k-ближайших соседей. В настоящее время используют несколько версий данного метода:

NearMiss-1 выбирает выборки из класса большинства, для которого среднее расстояние до ближайших соседей является наименьшим.

NearMiss-2 выбирает выборки из класса большинства, для которого среднее расстояние до самых дальних соседей является наименьшим.

NearMiss-3 можно разделить на 2 этапа. Сначала создается краткий список значений из мажоритарного класса. Затем из полученного списка выбирается выборка с наибольшим средним расстоянием до *k* ближайших соседей.

1.9 Выводы

Проведенный анализ проблемы обнаружения мошеннических транзакций позволил понять настоящее положение дел в данной области. Стало очевидным превосходство машинного обучения над другими методами обнаружения мошеннических транзакций. Благодаря таким факторам как эффективность, высокая скорость работы, масштабируемость системы, машинное обучение остается наиболее перспективным решением данной проблемы. Алгоритмы, применяемые при выявлении нечестных операций представлены широким спектром математических методов: от простой логистической регрессии до такого мощного инструмента как нейронная сеть. Также были выявлены проблемы стоящие на пути применения машинного обучения в реальных задачах. Основной из них является несбалансированных набор данных. Однако для преодоления этого ограничения, разработаны и активно применяются методы изменения репрезентативности классов, такие как избыточная и недостаточная выборки.

ГЛАВА 2 ПОСТАНОВКА ЗАДАЧИ

Наименование задачи: Разработка и реализация решения в области выявления мошеннических транзакций на основе несбалансированных данных.

Цель разработки: Эффективное выявление мошеннических транзакций при несбалансированной выборке.

2.1 Предъявляемые требования

- Масштабируемость;
- Эффективная работа в условиях несбалансированных данных;
- Способность точно классифицировать как мошеннические, так и законные транзакции.

2.2 Входные данные

Таблица формата .csv, которая содержит 284807 транзакций и 31 признак [4].

Формат данных признаков:

V1-V28 – (float64);

Time – время (float64);

Amount – сумма транзакции (float64);

Class – целевая метка транзакции (int64).

2.3 Выходные данные

Сводная таблица с оценками recall и precision метрик для каждого статистического классификатора по каждой реализованной выборке.

2.4 Критерии оценки результата

- Процентное отношение числа распознанных мошеннических транзакций к числу всех мошеннических транзакций исходной выборки;
- Процентное отношение числа распознанных честных транзакций к числу всех честных транзакций исходной выборки.

2.5 Методы контроля качества

Для статистических классификаторов мы будем использовать метрики из библиотеки sklearn.metrics:

- Recall;
- Precision.

Для нейронной сети воспользуемся встроенными в библиотеку keras средствами оценки точности и величины ошибки обучения. Также используем матрицу путаницы для наглядной демонстрации результатов.

Во время работы нейронной сети мы будем анализировать кривые тренировки и теста как для функции потерь, так и для точности. Чем меньше отклонения между кривыми, тем выше шанс что переобучение модели не произошло.

ГЛАВА 3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ ОБНАРУЖЕНИЯ МОШЕННИЧЕСКИХ ТРАНЗАКЦИЙ МЕТОДАМИ МАШИННОГО ОБУЧЕНИЯ

3.1 Теоретическое обоснование предлагаемого решения

3.1.1 Выбор средств реализации проектируемой системы

Для создания системы выявления мошеннических транзакции мы будем использовать язык программирования Python в среде Anaconda3.

Данный выбор обоснован наличием огромного числа специализированных библиотек и ресурсов с технической документацией [9-10]. Мы будем использовать следующие библиотеки:

Pandas, matpolib, seaborn – построение графиков и схем;

Imblearn – алгоритмы избыточной и недостаточной выборки;

Sklearn – статистические классификаторы и перекрестная проверка.

3.1.2 Выбор средств масштабирования значений признаков

В качестве функции масштабирования мы будем использовать RobustScaler из библиотеки sklearn.preprocessing. Данная функция удаляет медиану и масштабирует данные в соответствии с квантильным диапазоном. Центрирование и масштабирование выполняются независимо для каждой функции. Медианный и межквартильный диапазоны затем сохраняются для последующего использования [2].

3.1.3 Выбор модели машинного обучения

Очевидно, что нет единого универсального алгоритма для модели машинного обучения [15]. А потому мы будем использовать несколько классификаторов, чтобы оценить их эффективность и выбрать лучший.

Случайный лес

Данный алгоритм обладает практически нулевой ошибкой на любой выборке, что будет полезно в нашей задачи с таким большим дисбалансом классов.

Метод опорных векторов

Данный алгоритм считается одним из самых эффективных в машинном обучении. В методе опорных векторов используется понятие гиперплоскости, построение которой разделяет входные переменные по классам. Поэтому предполагаем, что данный алгоритм должен дать хорошие результаты в задачи бинарной классификации.

К-ближайших соседей

Несмотря на то, что многомерные данные плохо сказываются на работе данного алгоритма, зачастую данный метод показывает отличные результаты.

Логистическая регрессия

Логистическая функция выглядит как сигмоида и преобразовывает любое значение в диапазоне от 0 до 1. Благодаря тому как обучается модель логистической регрессии она отлично подходит для задач бинарной классификации.

Нейронная сеть

Для построения нейронной сети мы будем использовать открытую библиотеку keras. Модульность и компактность данного решения позволяет очень удобно реализовывать модели машинного обучения.

Для нейронной сети мы выбрали модель перцептрона, как наиболее простое и в тоже время эффективное решение.

3.1.4 Поиск гиперпараметров по сетке

Практически все выбранные статистические классификаторы обладают набором параметров, значения которых лежит в определенном диапазоне. С цель получения эффективных результатов от модели

машинного обучения, мы будем применять оптимизацию этих параметров. Конкретно, случайный поиск гиперпараметров по сетке. Данный алгоритм подразумевает, что оптимизатор, используя заданный пользователем словарь параметров будет случайно перебирать комбинации этих параметров n-е число раз [1].

Однако, при оптимизации гиперпараметров существует риск переобучения модели, если поиск параметров и конечное тестирование будет происходить на одних и тех же данных. Чтобы избежать такой методической ошибки мы будем использовать перекрестную проверку.

3.1.5 Перекрестная проверка

Чтобы избежать переобучения, можно использовать 3 набора данных:

- Набор для обучения;
- Набор для тренировки;
- Набор для тестов.

При таком подходе, набор для тестов изначально откладывается и не принимает участия в обучении. Однако в условиях сильного дисбаланса классов это приведет к тому, что итоговая оценка будет крайне случайной величиной.

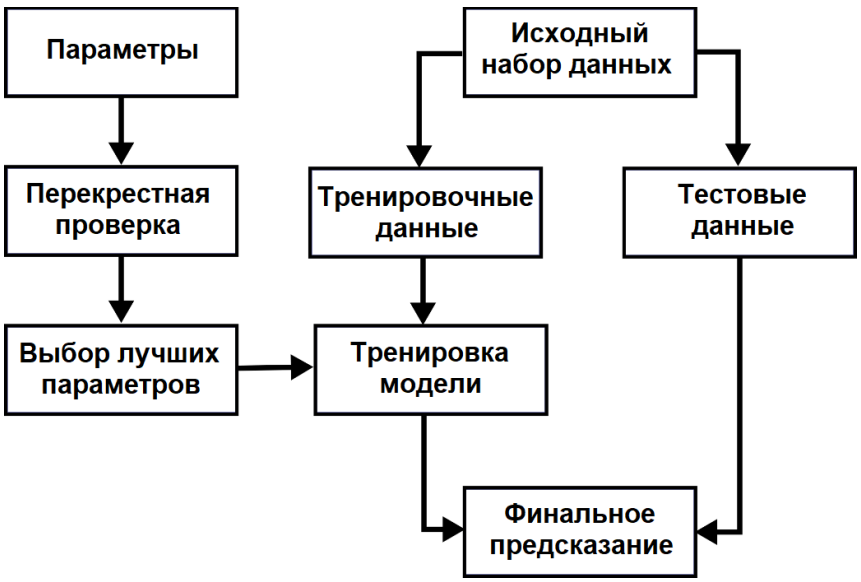


Рисунок 12 – Перекрестная проверка

Для того, чтобы исключить описанную проблемы мы будем использовать перекрестную проверку [10-11]. Набор для тестирования все также не будет принимать участия ни в обучении, ни в тренировке модели. Вместо этого, тренировочный набор данных разбивается на k наборов данных, затем модель обучается на k-1 наборах данных и тестируется на оставшейся выборке. Данная операция выполняется до тех пор, пока все данные не побывают как на месте наборов для обучения, так и на месте наборов для теста. Таким образом оценка перекрестной проверки будет представлять собой среднюю оценку по k операциям.

В завершении обученная модель проверяется на первоначально отложенной тестовой выборке.

Также стоит отметить, во избежание утечки данных в процессе обучения, мы будем использовать методы изменения репрезентативности классов прямо во время перекрестной проверки.

3.1.6 Выбор методов изменения репрезентативности классов

В решении задачи мы воспользуемся как методами избыточной, так и недостаточной выборки. Мы

Для недостаточной выборки мы будем использовать алгоритмы:

- RandomUnderSampler;
- NearMiss-1.

Для избыточной выборки:

- SMOTE;
- ADASYN.

3.1.7 Выбор средств оценки модели машинного обучения

Для того чтобы оценить производительность модели в задачи классификации, мы будем использовать матрицу путаницы [3,12]. Пример такой матрицы изображен на рисунке 13.

Рисунок 13 – Матрица путаницы

Приведенная выше матрица отображает 4 характеристики:

- Мошеннические транзакции, которые распознаны как мошеннические (TP);
- Мошеннические транзакции, которые не распознанные как мошеннические (FN);
- Честные транзакции, распознанные как честные транзакции (TN);
- Честные транзакции, распознанные как мошеннические (FN).

Перечислим все метрики, которые мы будем вычислять.

Accuracy

Данная метрика отражает долю всех правильных ответов относительно всех ответов алгоритма.

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

Precision

Precision позволяет понять долю истинных мошеннических транзакций от числа всех транзакций предсказанных классификатором как мошеннические.

$$precision = \frac{TP}{TP+FP} \quad (2)$$

Recall

Данную метрику еще именуют как «полнота». Она отражает долю истинных мошеннических транзакций, распознанных моделью из числа всех истинных мошеннических транзакций.

$$recall = \frac{TP}{TP+FN} \quad (3)$$

F1

Позволяет найти среднее гармоническое между метриками recall и precision. То есть если любая из величин стремится к нулю, то и F1 тоже стремится к нулю.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision+recall} \quad (4)$$

ROC AUC.

Еще одной метрикой которой мы воспользуемся является кривая ROC AUC (Рисунок 14).

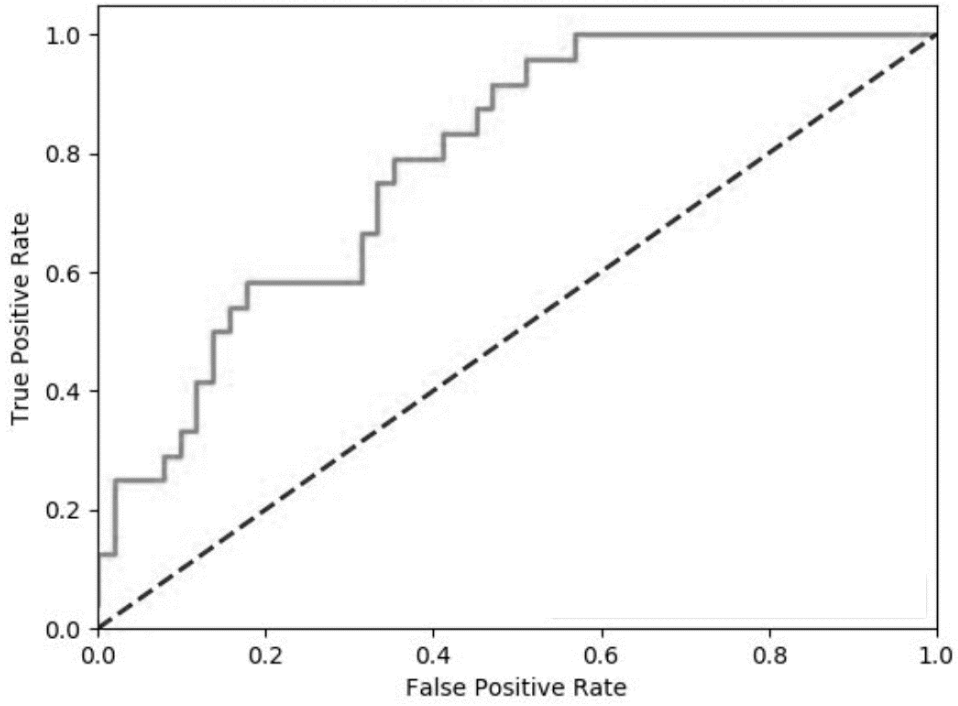


Рисунок 14 – Кривая ROC AUC

Данная кривая лежит в координатной плоскости между True Positive Rate и False Positive Rate.

$$TRP = \frac{TP}{TP+FN} \quad (5)$$

$$FPR = \frac{FP}{TN+FP} \quad (6)$$

Как видно исходя из формулы TRP идентичная метрики recall. FRP же можно интерпретировать как долю честных транзакций предсказанных неверно.

Прежде чем выбрать приоритетную метрику для нашей задачи, выделим два рода ошибок предсказания модели:

Ошибка первого рода – когда мошенническая транзакция распознана как честная.

Ошибка второго рода – когда честная транзакция распознана как мошенническая.

В рамках задачи по выявлению мошенничества приоритетным является распознавание настоящих злоумышленников. А потому главной оценкой, на которую мы будем опираться будет метрика **recall**.

Но если рассматривать систему выявления нечестных транзакций как продукт для внедрения на реальном предприятии, то ошибка второго рода имеет такое же важное значение. Потому обозначим дополнительную оценку для оптимизации – метрику **precision**.

3.1.8 Проектирование функции для оценки классификаторов.

В целях автоматизации обучения, тренировки и тестирования классификаторов мы реализуем специальную функцию. Целью которой является объединение перекрестной проверки, поиска параметров по сетке, функции изменения репрезентативности классов и тренировки модели. Для этого мы будем использовать конвейер данных.

Конвейер данных – абстрактная структура позволяющая преобразовывать данные из одного представления в другое посредством последовательности шагов. Данный подход позволяет объединить несколько шагов преобразований и итоговую оценку в одну конструкцию с единым интерфейсом. В языке Python, концепция конвейера данных реализована через понятие pipeline [13-14].

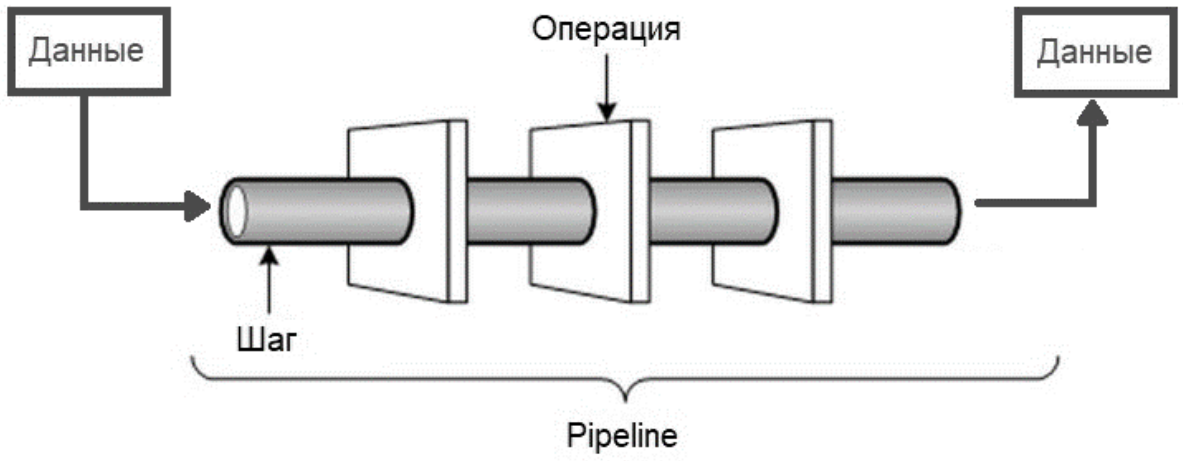


Рисунок 15 – Pipeline

Выделим несколько важных принципов при построении конвейера:

- Каждый компонент конвейера отделен от других и получает определенный вход и возвращает определенный выход;
- Результаты могут быть сохранены и использованные далее;
- Каждый компонент конвейера передает данные в другой компонент.

Таким образом в проектируемой функции, мы осуществим разбиение оригинального тренировочного набора данных на субтренировочный и субтестовый. Далее мы будем производить перекрестную проверку в ходе которой будет происходить поиск гиперпараметров по сетке, обучение и тренировка классификаторов на лучших подобранных параметрах модели. Реализуем вывод и иллюстрацию метрик. Наконец выполним предсказание тестовой выборки оригинальных данных. И в финале построим матрицу путаницы и график кривой ROC AUC.

Такую функцию мы реализуем для 4-х видов классификаторов и 4-х видов изменения репрезентативности классов.

3.1.9 Проектирование нейронной сети

Для тестирования различных выборок на нейронной сети мы реализуем собственную функцию.

Входные данные функции:

- Имя алгоритма создания выборок;
- Инициализация алгоритма с заданными параметрами.

Выходные данные:

- Оценка функции потерь
- Оценка точности
- Матрица путаницы
- Кривые тренировки и тестирования функции потерь
- Кривые тренировки и тестирования точности.

Чтобы достоверно знать, не произошло ли переобучение или недообучение модели мы будем анализировать графики кривых. Наша цель добиться минимального отклонения между кривыми тренировки тестов по всем оценкам [16].

Алгоритм работы функции:

- Инициализация указанного алгоритма создания выборки;
- Создание сбалансированной выборки из тренировочных данных оригинального набора данных;
- Инициализация модели нейронной сети;
- Добавление слов;
- Инициализация нормализации между слоями;
- Компилирование модели;
- Обучение нейронной сети;
- Предсказание исходных тестовых данных;
- Подсчет оценок;
- Инициализация и вывод матрицы путаницы;
- Вывод кривых функции потерь;
- Вывод кривых точности.

В качестве модели нейронной сети мы будем использовать перцептрон с 2-мя промежуточными слоями. Модель будет иметь плотную структуру, то есть каждый нейрон связан со всеми нейронами следующего слоя. В качестве функции активации для выходного слоя будем использовать сигмоиду, для промежуточных – функцию relu. Также между скрытыми слоями мы реализуем нормализацию значений с помощью метода BatchNormalization. Из-за того, что сигнал проходя между слоями может сильно исказиться, мы и будем использовать нормализацию. Суть данной процедуры заключается в таком преобразовании входных данных слоя, чтобы получить единичную дисперсию и нулевое матожидание.

Для функции потерь мы будем использовать binary_crossentropy. В роли оптимизатор – метода стохастической оптимизации Adam. В качестве оптимизируемой метрики используем accuracy, по той причине, что нейронные сети на библиотеке Keras не поддерживают оптимизацию метрик recall и precision. Можно написать свою метрику и применить её в модели, но существует вероятность неправильного обучения и как следствие некорректных результатов работы нейронной сети. Исходный код спроектированной функции представлен в приложении Д.

3.2 Практическая реализация системы выявления мошеннических транзакций

3.2.1 Первичный анализ данных

Для начала загрузим данные и реализуем вывод шапки таблицы (первые 20 столбцов и 60 строк). Результат представлен на рисунке 15.

	Time	V1	V2	V3	...	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	...	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	...	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	...	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	...	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	...	0.219422	0.215153	69.99	0

Рисунок 16 – Шапка таблицы

Далее осуществим вывод общей информации по исходным данным: число признаков, типы данных, количество записей. Важно учесть, что классификаторы могу работать некорректно если в наборе данных, присутствуют пустые значения. Как видно из приложения 2, в наших данных таковы отсутствуют.

Так как реальный смысл практических всех признаков зашифрован, наибольшую ценность для анализа представляют столбцы «Class» и «Amount».

Чтобы понять соотношение транзакций, реализуем подсчет транзакций (столбец «Class») в процентном отношении (Рисунок 17).

```
Честные транзакции 99.83 % от всех операций
Мошеннические транзакции 0.17 % от всех операций
```

Рисунок 17 – Процентное отношение транзакций

Чтобы нагляднее понять, насколько большой дисбаланс построим график (Рисунок 18)

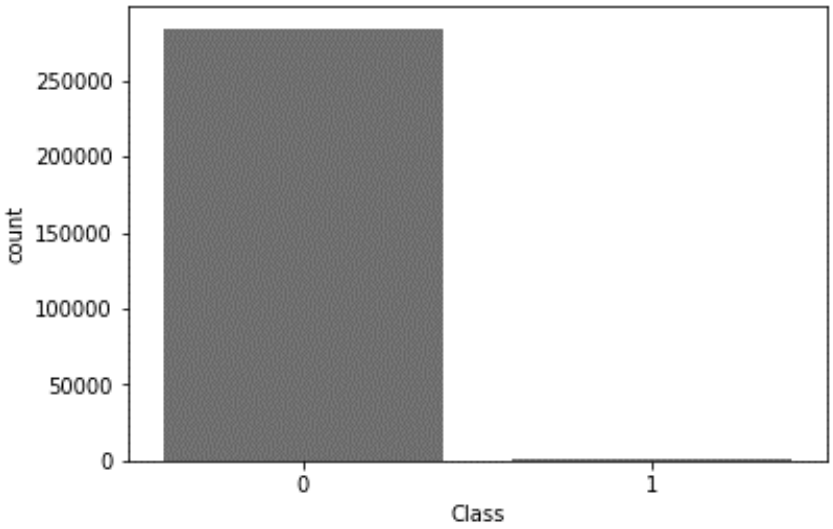


Рисунок 18 – Распределение транзакций

Далее на рисунке 19 проиллюстрируем распределения числа транзакций в единицу времени.

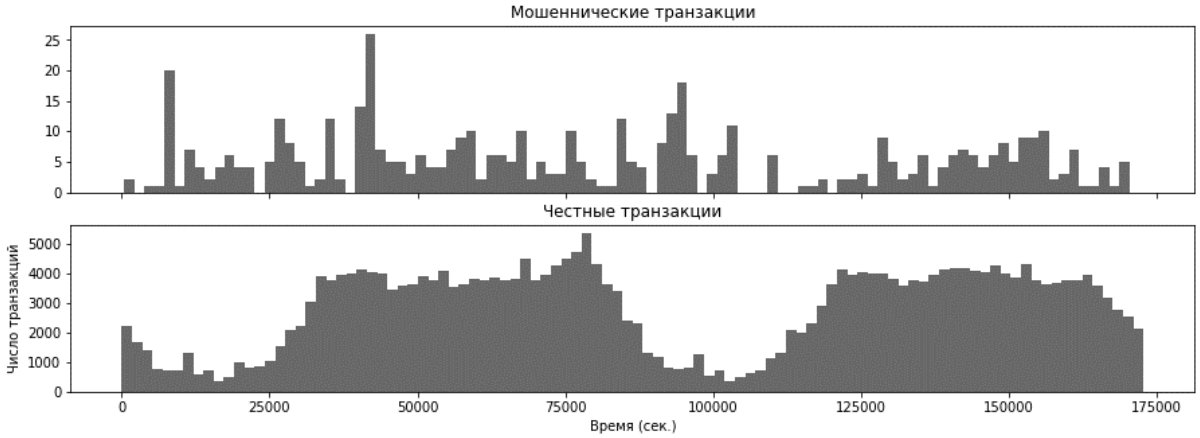


Рисунок 19 – Распределение транзакций в единицу времени

Как видно, данные достаточно неоднородны. Наибольшая «яма» среди честных операций наблюдается на отметке 100000 сек. По графику нечестных транзакций можно понять, что в среднем в единицу времени совершается от 4 до 7 операций, но также наблюдаются значительные выбросы.

На рисунке 20 показано на какую сумму чаще всего совершаются транзакции.

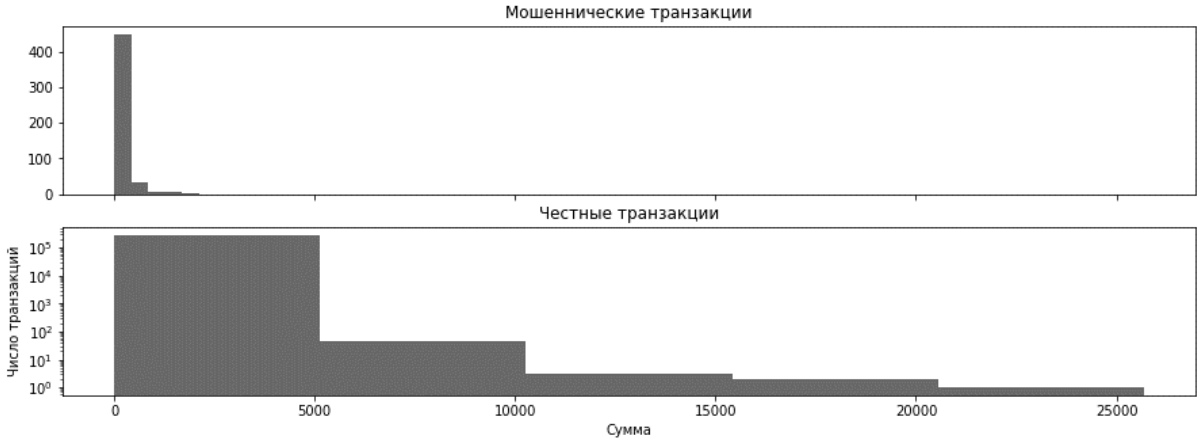


Рисунок 20 – Распределение мошеннических транзакций по сумме

Как видно из распределения, наибольшее количество случаев мошенничества происходит на сумму 100$.

3.2.2 Подготовка данных

В исходном наборе данных присутствуют признаки, значение которых не соответствует единому диапазону остальных функций. Чтобы устранить эту проблему применим масштабирование.

На рисунке 21 приведена иллюстрация получившегося набора данных.

	robust_amount	rubust_time	V1	...	V27	V28	Class
0	1.783274	-0.994983	-1.359807	...	0.133558	-0.021053	0
1	-0.269825	-0.994983	1.191857	...	-0.008983	0.014724	0
2	4.983721	-0.994972	-1.358354	...	-0.055353	-0.059752	0
3	1.418291	-0.994972	-0.966272	...	0.062723	0.061458	0
4	0.670579	-0.994960	-1.158233	...	0.219422	0.215153	0

Рисунок 21 – Исходный набор данных после масштабирования значений

Разделим исходный набор данных на тренировочную и тестовую выборки в соотношении 80% и 20% от исходных данных соответственно.

Результат приведен на рисунке 22.

```
Распределение тренировочных данных: Counter({0: 227451, 1: 394})
Распределение тестовых данных: Counter({0: 56864, 1: 98})
```

Рисунок 22 – Распределение данных для тренировочной и тестовой выборки

3.2.3 Реализация функции

В целях упрощения и автоматизации обучения и тестирования классификаторов, и алгоритмов реализуем специальную функцию. Целью которой является стандартизация входных и выходных данных для каждого вида модели и выборки.

На вход функции поступают такие данные как:

- Имя статистического классификатора;
- Инициализация статистического классификатора;
- Словарь параметров для указанного статистического классификатора;
- Инициализация алгоритма изменения репрезентативности классов.

В ходе выполнения функция возвращает следующие данные:

- Средние значения метрик после перекрестной проверки;
- Значения метрик после предсказания на исходном тестовом наборе данных;
- Матрица путаницы;
- Кривая ROC AUC.

Алгоритм работы функции:

1) Объявляются переменные списков;
2) Инициализируется функция случайного поиска по сетке RandomizedSearchCV;
3) Тренировочные данные разбиваются на субтренировочные и субтестовые и далее внутри цикла осуществляется перекрестная проверка. Алгоритм изменения выборки осуществляется таким образом, чтобы избежать утечки данных и как следствие переобучения;
4) Реализуется обучение на субтренировочных данных;
5) Выбирается лучший набор параметров для модели;
6) Выполняется предсказание на субтестовом наборе данных;
7) Полученные значения метрик добавляются в список и цикл повторяется до тех пор, пока перекрестная проверка не будет завершена;
8) Вывод среднего значения по полученным данным метрик;
9) Списки метрик обнуляются;
10) Выполняется предсказание модели на исходных тестовых данных;
11) Результат записывается в списки;
12) Вывод записанных метрик;
13) Инициализируется функция построения матрицы путаницы;
14) Вывод получившегося графика;
15) Инициализируется функция построения ROC AUC кривой;
16) Вывод получившегося графика.

Блок схема реализованной функции проиллюстрирована в приложении Б.

3.2.4 Результат тестирование классификаторов с применением алгоритма недостаточной выборки

Алгоритм RandomUnderSampler

```
_____________________________
Метод опорных векторов

_____________________________
Результат кроссвалидации:
accuracy:96.848 %
precision:4.999 %
recall:89.367 %
f1:9.441 %
Roc Auc:93.114 %

_____________________________
_____________________________
Результат тестирования:
accuracy:97.814 %
precision:6.782 %
recall:91.837 %
f1:12.632 %
Roc Auc:94.831 %

_____________________________
```

Рисунок 23 – Оценки для метода опорных векторов

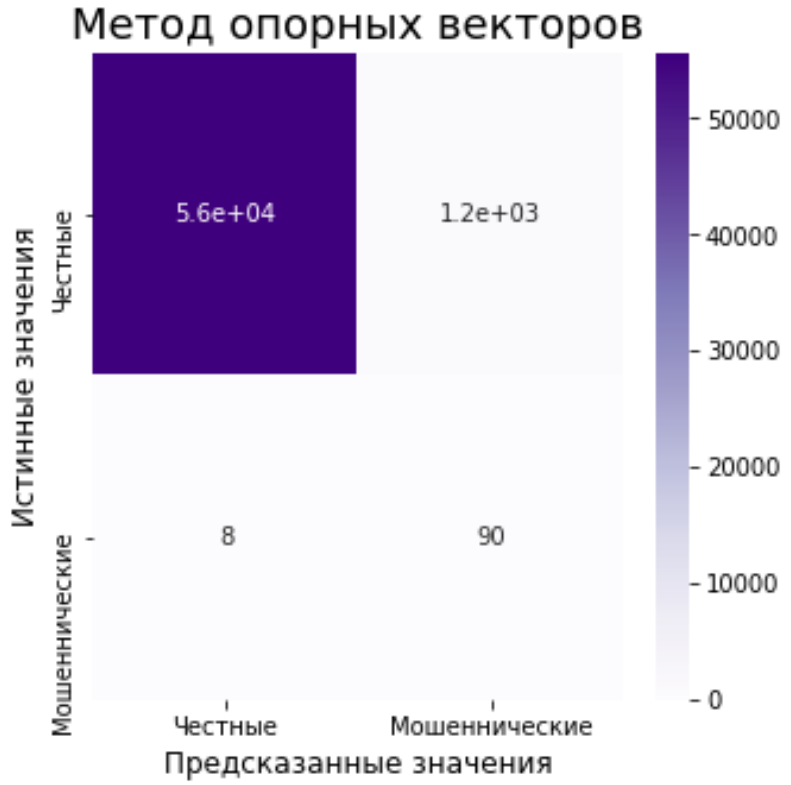


Рисунок 24 – Матрица путаницы для метода опорных векторов

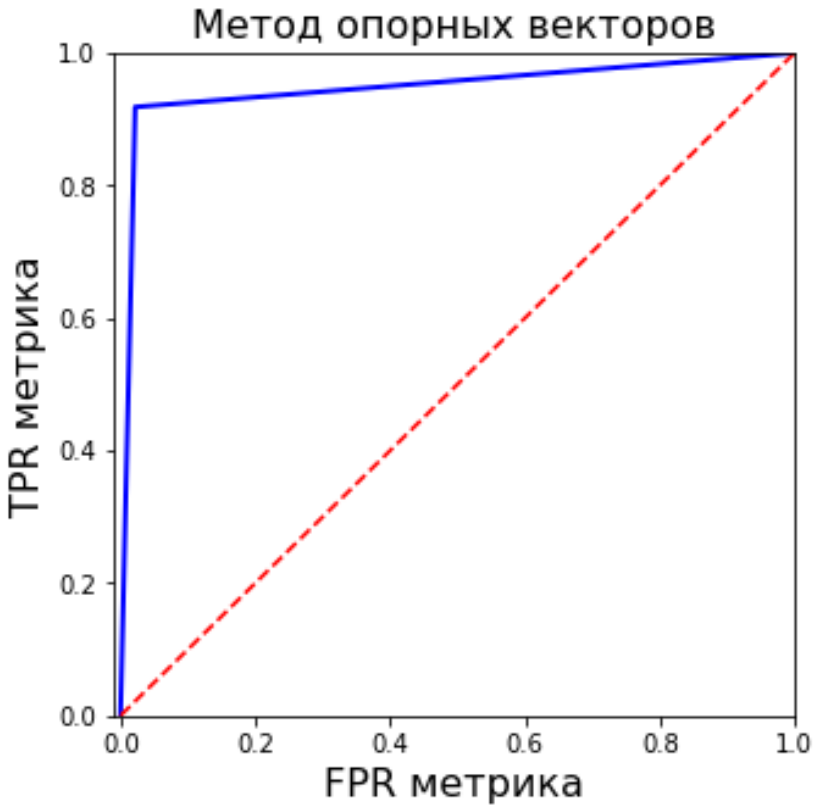


Рисунок 25 – ROC AUC кривая для метода опорных векторов

```
_________________________
Случайный лес

_________________________
Результат кроссвалидации:
accuracy:97.208 %
precision:5.700 %
recall:88.481 %
f1:10.656 %
Roc Auc:92.852 %

_________________________
_________________________
Результат тестирования:
accuracy:97.811 %
precision:6.642 %
recall:89.796 %
f1:12.368 %
Roc Auc:93.810 %

_________________________
```

Рисунок 26 – Оценки для алгоритма «случайный лес»

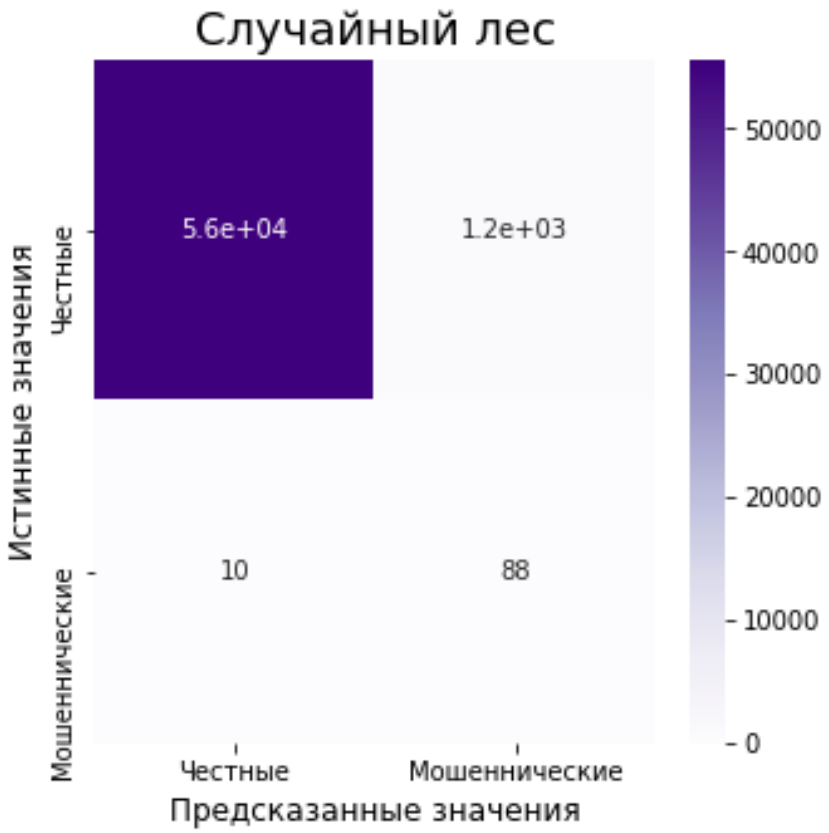


Рисунок 27 – Матрица путаницы для алгоритма «случайный лес»

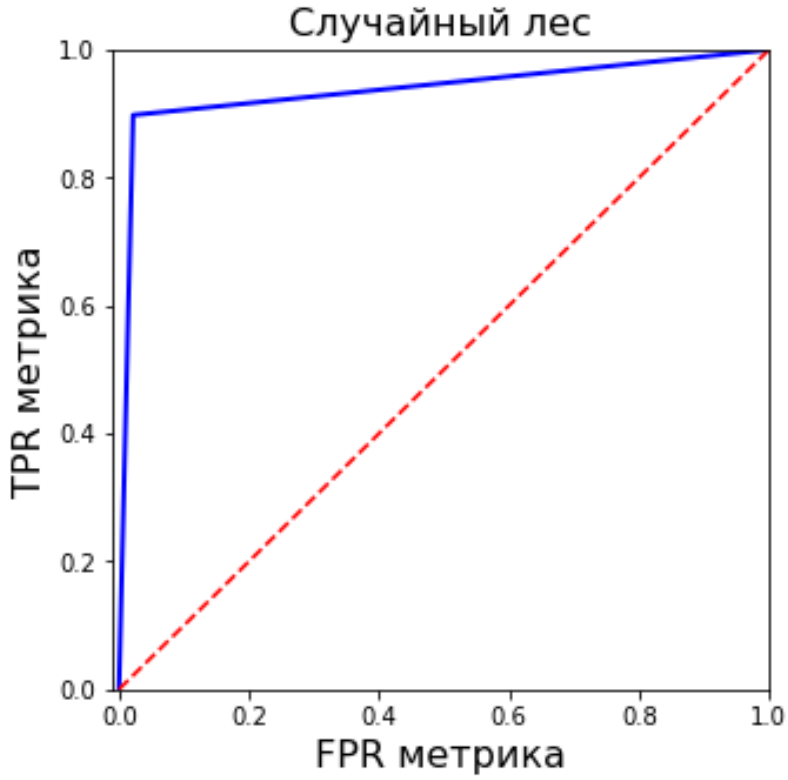


Рисунок 28 – Кривая ROC AUC для алгоритма «случайный лес»

```
_____________________________
K-ближайших соседей

_____________________________
Результат кроссвалидации:
accuracy:97.103 %
precision:5.482 %
recall:89.620 %
f1:10.295 %
Roc Auc:93.368 %

_____________________________
_____________________________
Результат тестирования:
accuracy:97.839 %
precision:6.656 %
recall:88.776 %
f1:12.384 %
Roc Auc:93.315 %

_____________________________
```

Рисунок 29 – Оценки алгоритма «k-ближайших соседей»

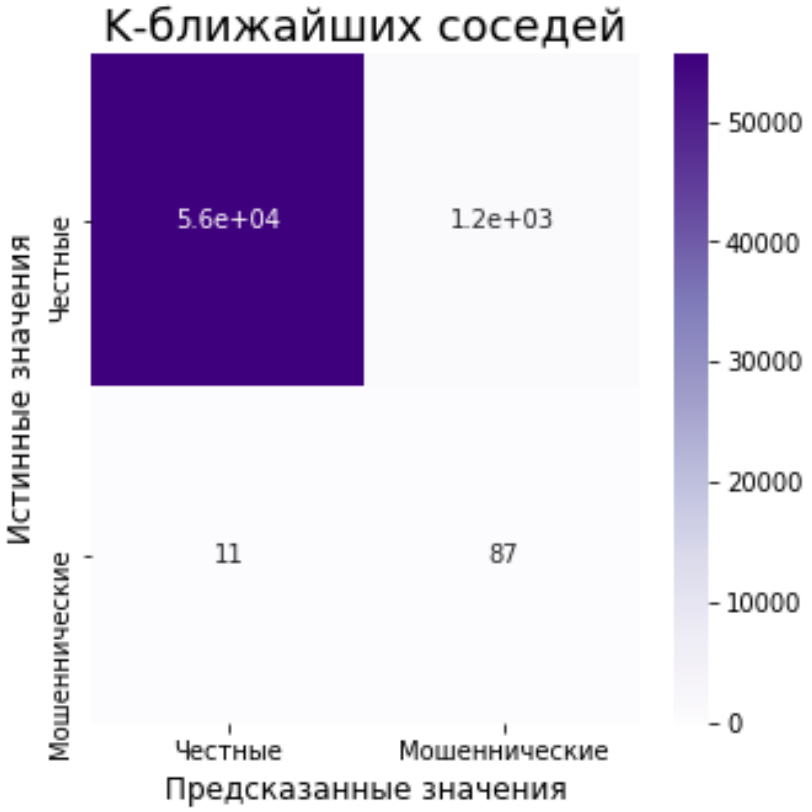


Рисунок 30 – Матрица путаницы для алгоритма «k-ближайших соседей»

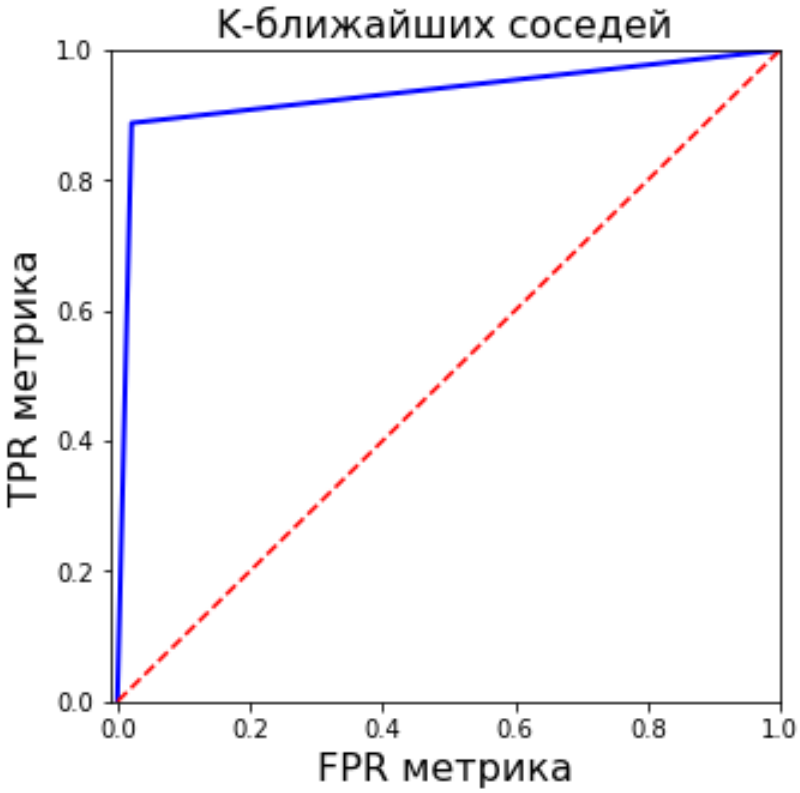


Рисунок 31 – ROC AUC кривая для алгоритма «к-ближайших соседей»

```
_____________________________
Логистическая регрессия

_____________________________
Результат кроссвалидации:
accuracy:97.101 %
precision:5.885 %
recall:89.114 %
f1:10.949 %
Roc Auc:93.114 %

_____________________________
_____________________________
Результат тестирования:
accuracy:98.645 %
precision:10.165 %
recall:87.755 %
f1:18.220 %
Roc Auc:93.209 %

_____________________________
```

Рисунок 32 – Оценки для алгоритма логистической регрессии

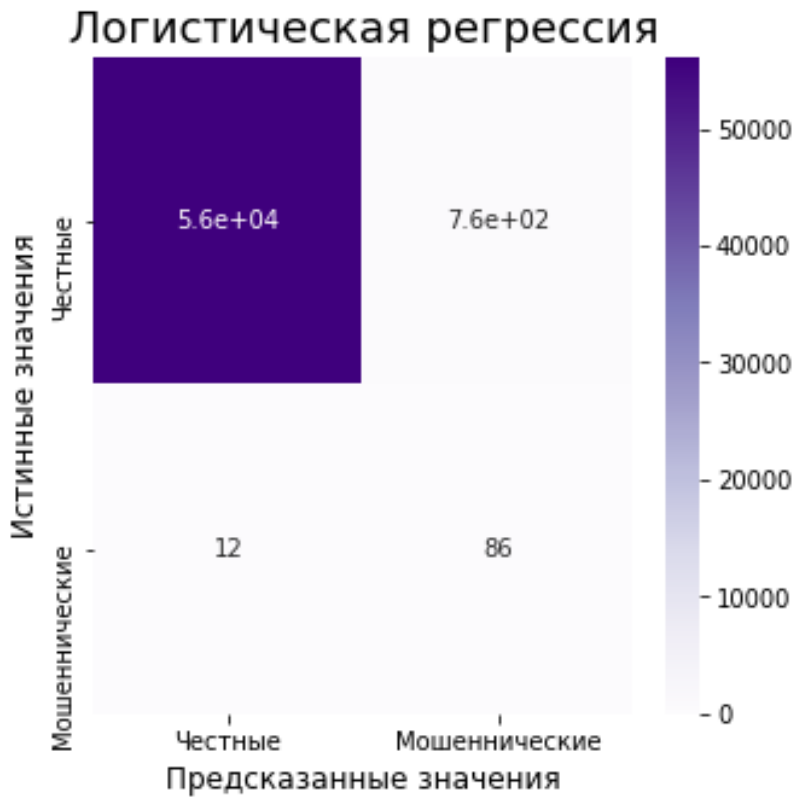


Рисунок 33 – Матрица путаницы для логистической регрессии

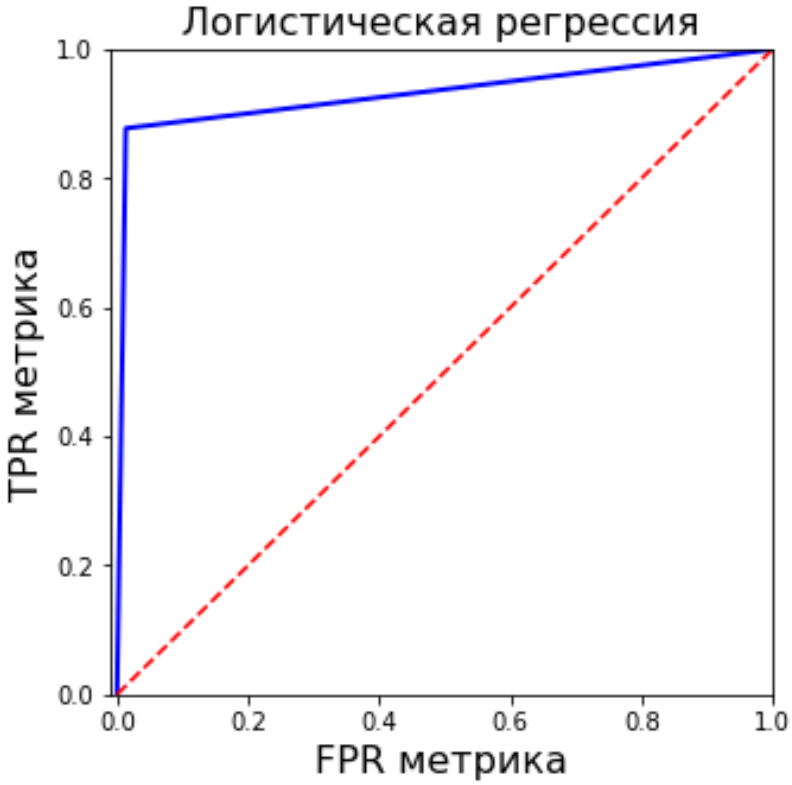


Рисунок 34 – ROC AUC кривая для алгоритма логистической регрессии

Алгоритм NearMiss

```
_____________________________
Метод опорных векторов

_____________________________
Результат кроссвалидации:
accuracy:29.285 %
precision:0.266 %
recall:94.937 %
f1:0.531 %
Roc Auc:62.054 %

_____________________________
_____________________________
Результат тестирования:
accuracy:16.973 %
precision:0.194 %
recall:93.878 %
f1:0.388 %
Roc Auc:55.359 %

_____________________________
```

Рисунок 35 – Оценки для метода опорных векторов

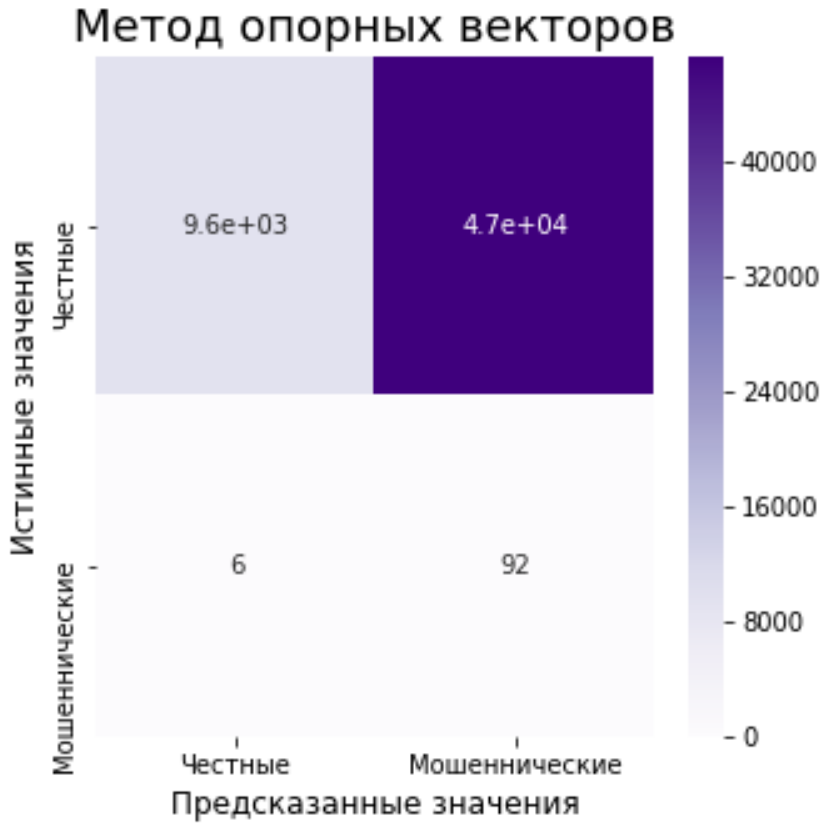


Рисунок 36 – Матрица путаницы для метода опорных векторов

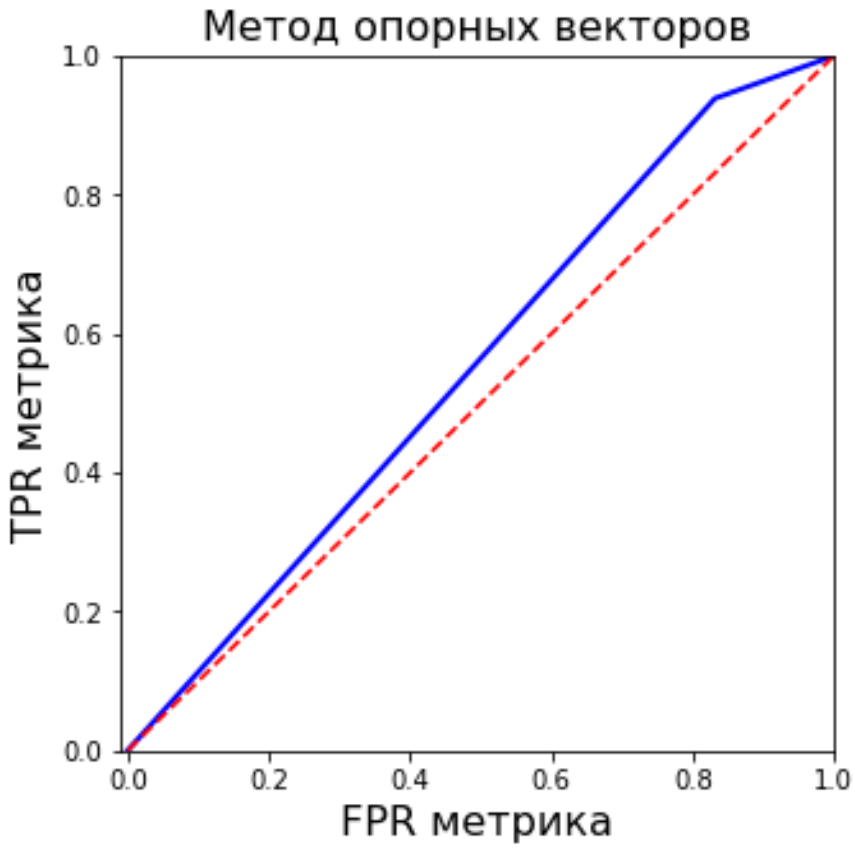


Рисунок 37 – ROC AUC кривая для метода опорных векторов

```
_____________________________
Случайный лес

_____________________________
Результат кроссвалидации:
accuracy:22.191 %
precision:0.217 %
recall:96.076 %
f1:0.432 %
Roc Auc:59.070 %

_____________________________
_____________________________
Результат тестирования:
accuracy:30.280 %
precision:0.236 %
recall:95.918 %
f1:0.471 %
Roc Auc:63.043 %

_____________________________
```

Рисунок 38 – Оценки для случайный лес

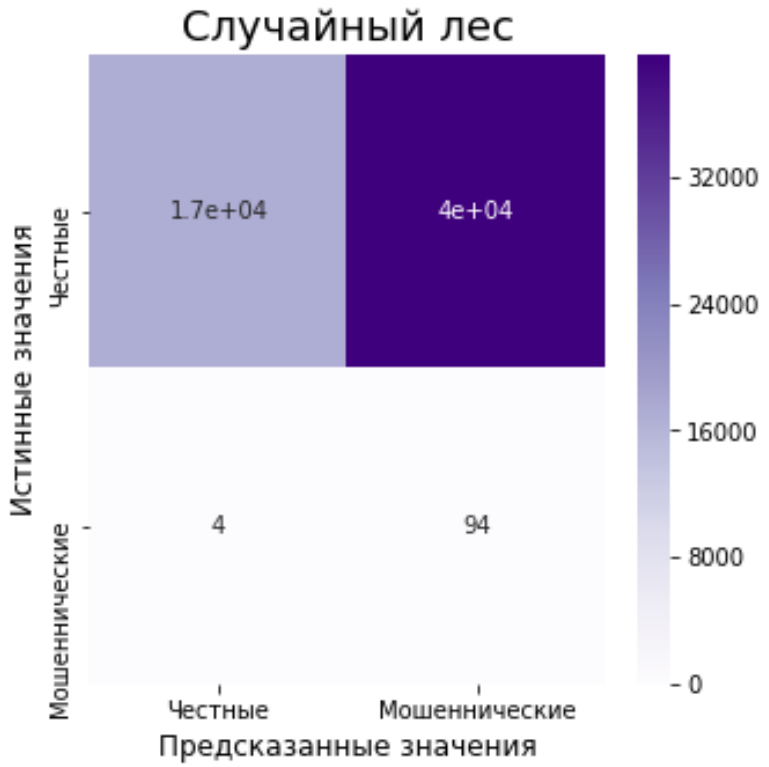


Рисунок 39 – Матрица путаницы для алгоритма случайный лес

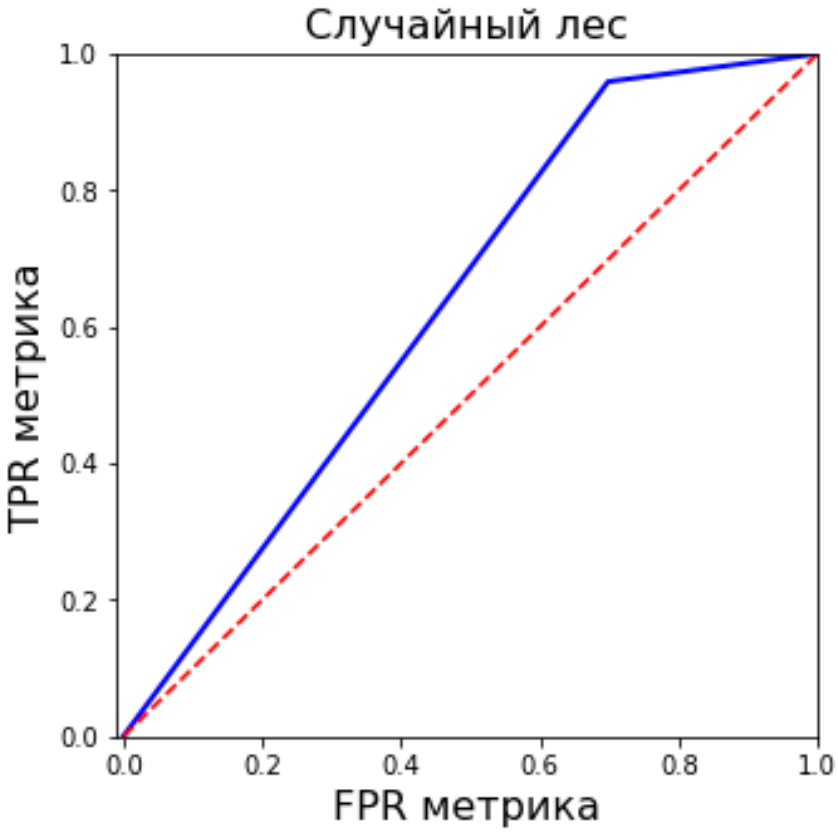


Рисунок 40 – ROC AUC кривая для алгоритма случайный лес

```
_____________________________
K-ближайших соседей

_____________________________
Результат кроссвалидации:
accuracy:68.119 %
precision:0.664 %
recall:91.519 %
f1:1.315 %
Roc Auc:79.799 %

_____________________________

_____________________________
Результат тестирования:
accuracy:58.232 %
precision:0.369 %
recall:89.796 %
f1:0.734 %
Roc Auc:73.987 %

_____________________________
```

Рисунок 41 – Оценки для метода k-ближайших соседей

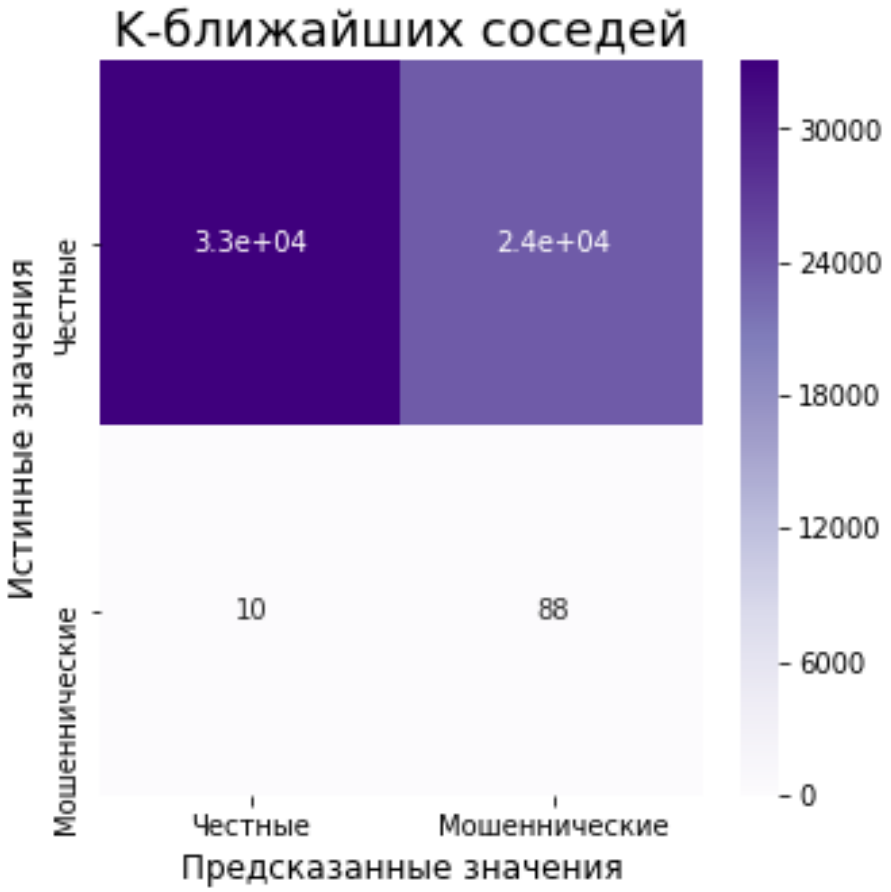


Рисунок 42 – Матрица путаницы для алгоритма k-ближайших соседей

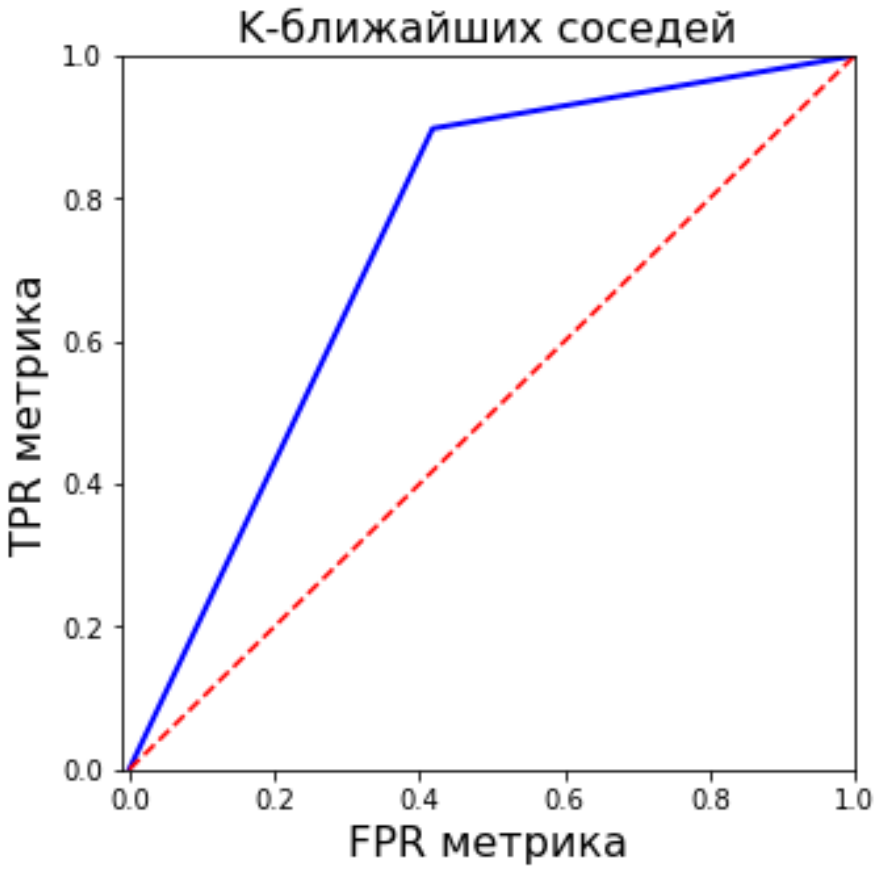


Рисунок 43 – ROC AUC кривая для алгоритма k-ближайших соседей

```
_____________________________
Логистическая регрессия

_____________________________
Результат кроссвалидации:
accuracy:60.900 %
precision:0.429 %
recall:92.658 %
f1:0.853 %
Roc Auc:76.752 %

_____________________________

_____________________________
Результат тестирования:
accuracy:58.962 %
precision:0.388 %
recall:92.857 %
f1:0.773 %
Roc Auc:75.880 %

_____________________________
```

Рисунок 44 – Оценки для метода логистической регрессии

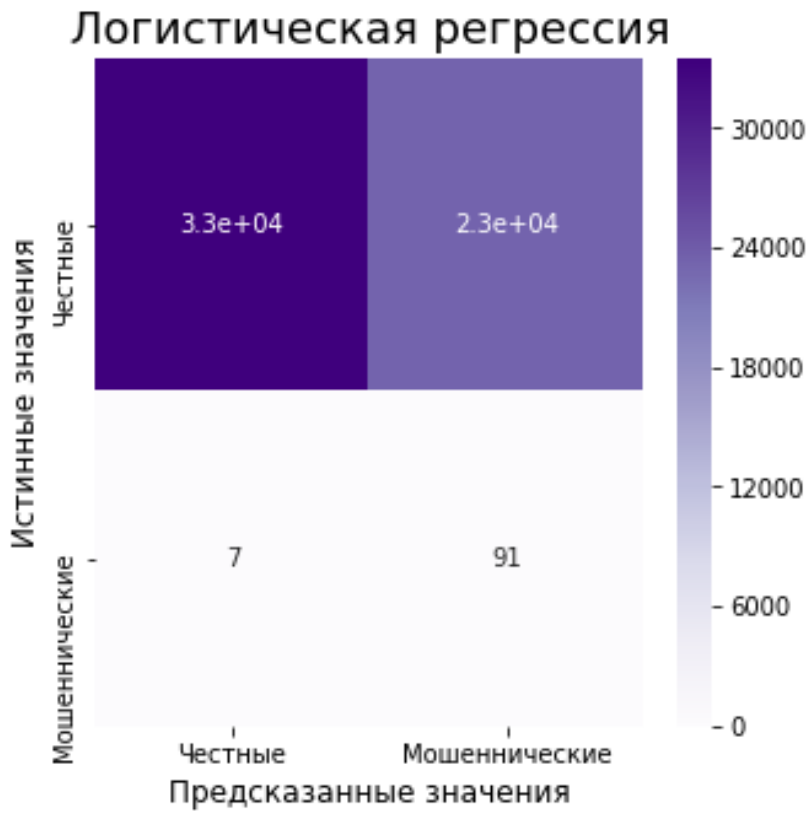


Рисунок 45 – Матрица путаницы для алгоритма логистической регрессии

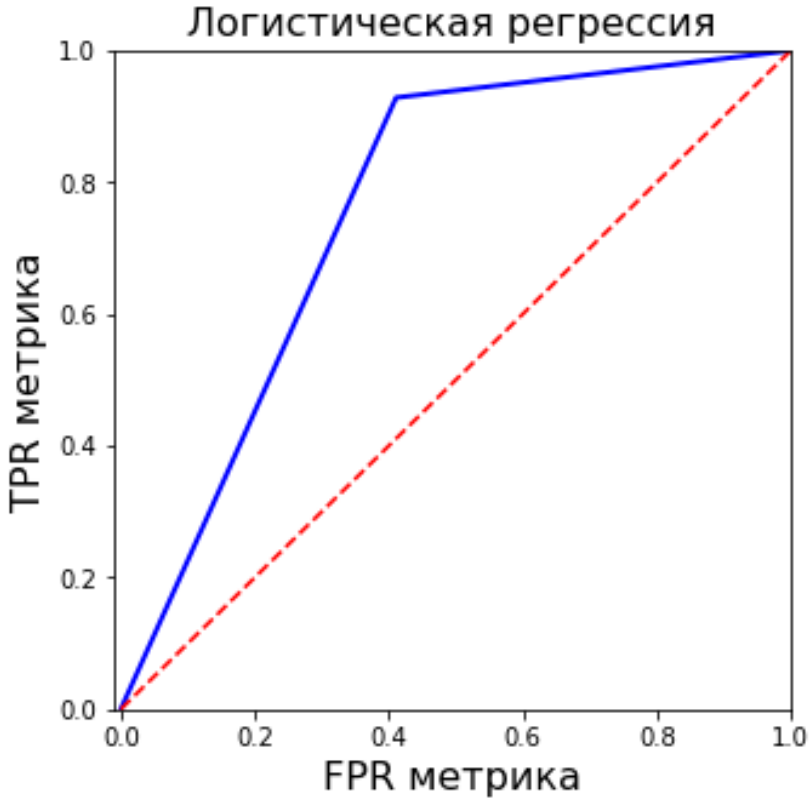


Рисунок 46 – ROC AUC кривая для метода логистической регрессии

3.2.5 Результат тестирования классификаторов с применением алгоритма избыточной выборки

SMOTE

```
_____________________________
Логистическая регрессия

_____________________________
Результат кроссвалидации:
accuracy:97.917 %
precision:7.147 %
recall:92.132 %
f1:13.265 %
Roc Auc:95.029 %

_____________________________

_____________________________
Результат тестирования:
accuracy:97.814 %
precision:6.652 %
recall:89.796 %
f1:12.386 %
Roc Auc:93.812 %

_____________________________
```

Рисунок 47 – Оценки для метода логистической регрессии

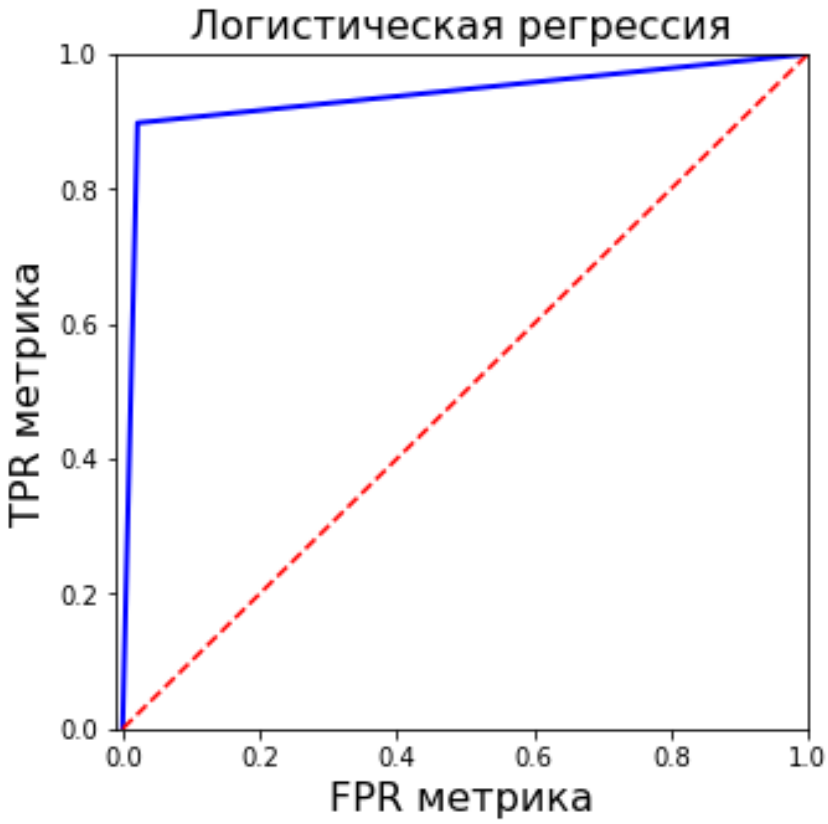


Рисунок 48 – ROC AUC кривая для метода логистической регрессии

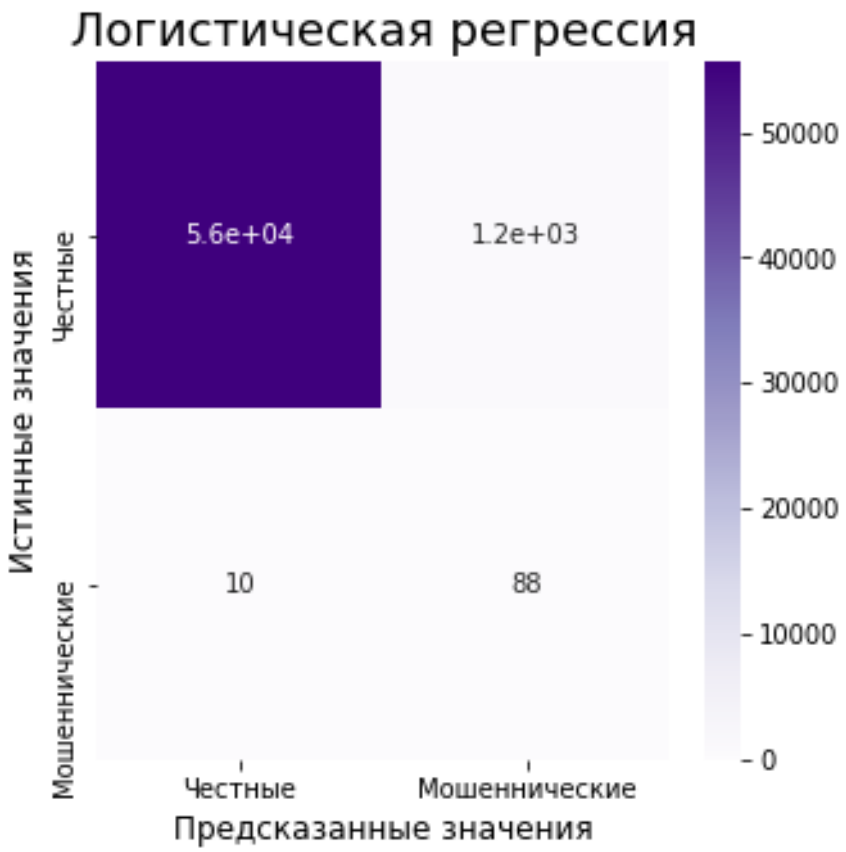


Рисунок 49 – Матрица путаницы для алгоритма логистической регрессии

```
_________________________________
Метод опорных векторов

_________________________________
Результат кроссвалидации:
accuracy:97.023 %
precision:5.115 %
recall:92.386 %
f1:9.694 %
Roc Auc:94.709 %

_________________________________
_________________________________
Результат тестирования:
accuracy:96.942 %
precision:4.636 %
recall:85.714 %
f1:8.796 %
Roc Auc:91.338 %

_________________________________
```

Рисунок 50 – Оценки для метода опорных векторов

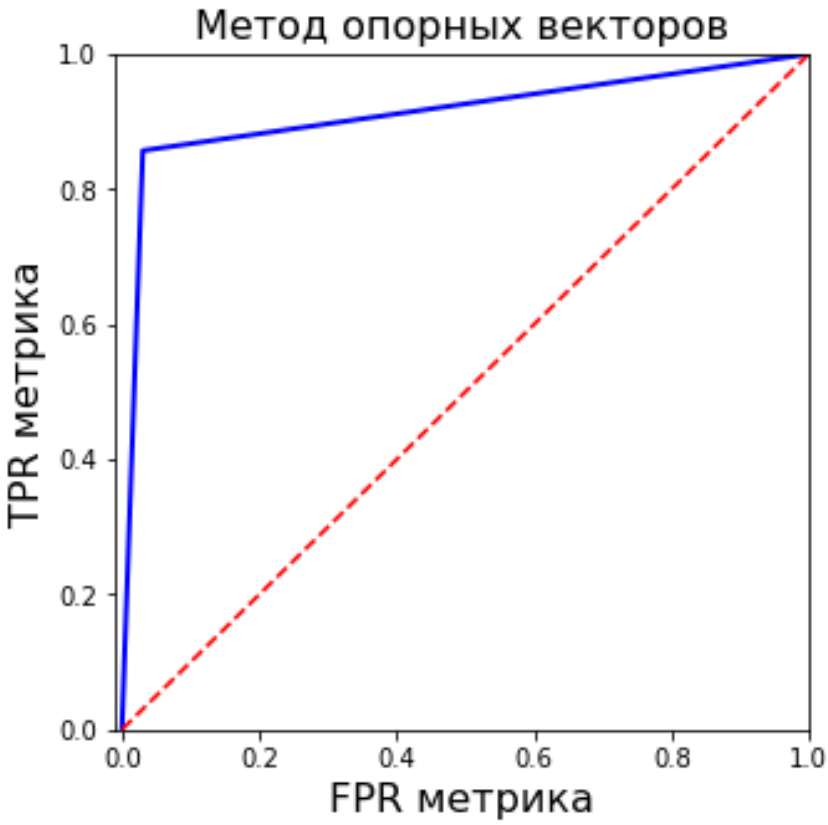


Рисунок 51 – ROC AUC кривая для метода опорных векторов

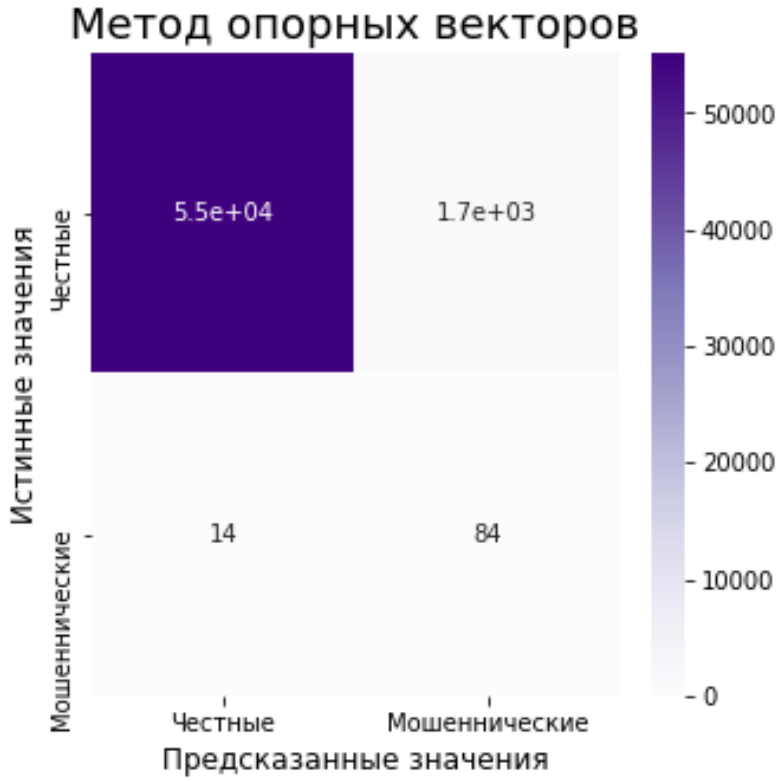


Рисунок 52 – Матрица путаницы для метода опорных векторов

```
_____________________________
K-ближайших соседей

_____________________________
Результат кроссвалидации:
accuracy:97.291 %
precision:5.499 %
recall:90.609 %
f1:10.369 %
Roc Auc:93.956 %

_____________________________
_____________________________
Результат тестирования:
accuracy:97.177 %
precision:4.844 %
recall:82.653 %
f1:9.153 %
Roc Auc:89.928 %

_____________________________
```

Рисунок 53 – Оценки для метода К-ближайших соседей

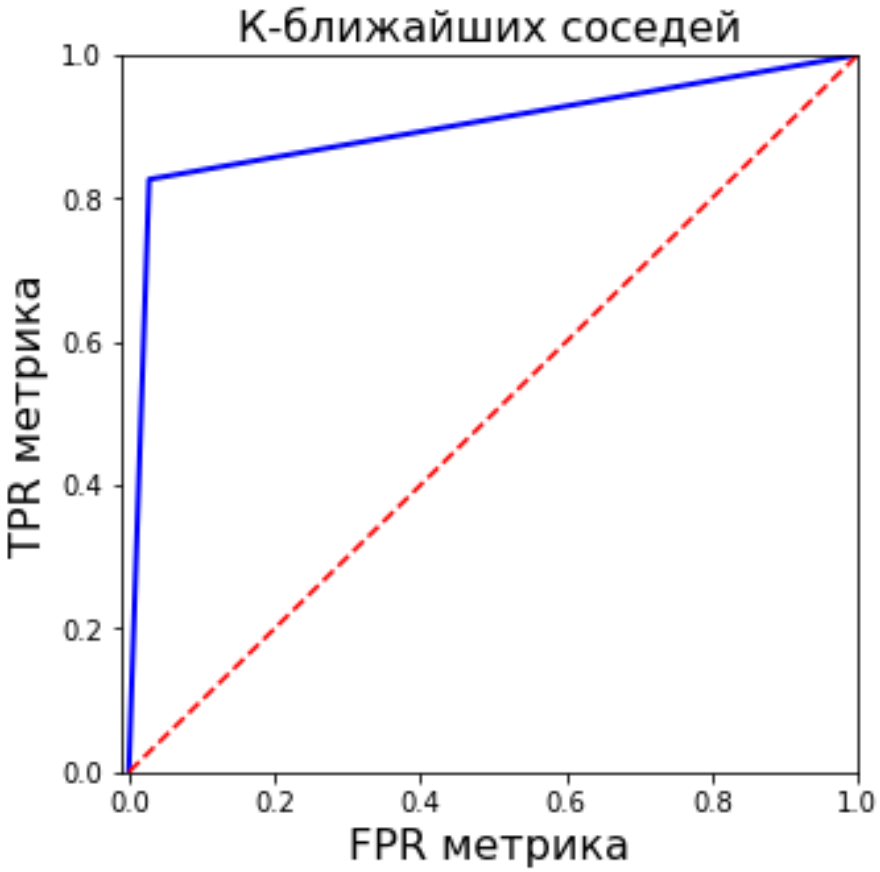


Рисунок 54 – ROC AUC кривая для метода К-ближайших соседей

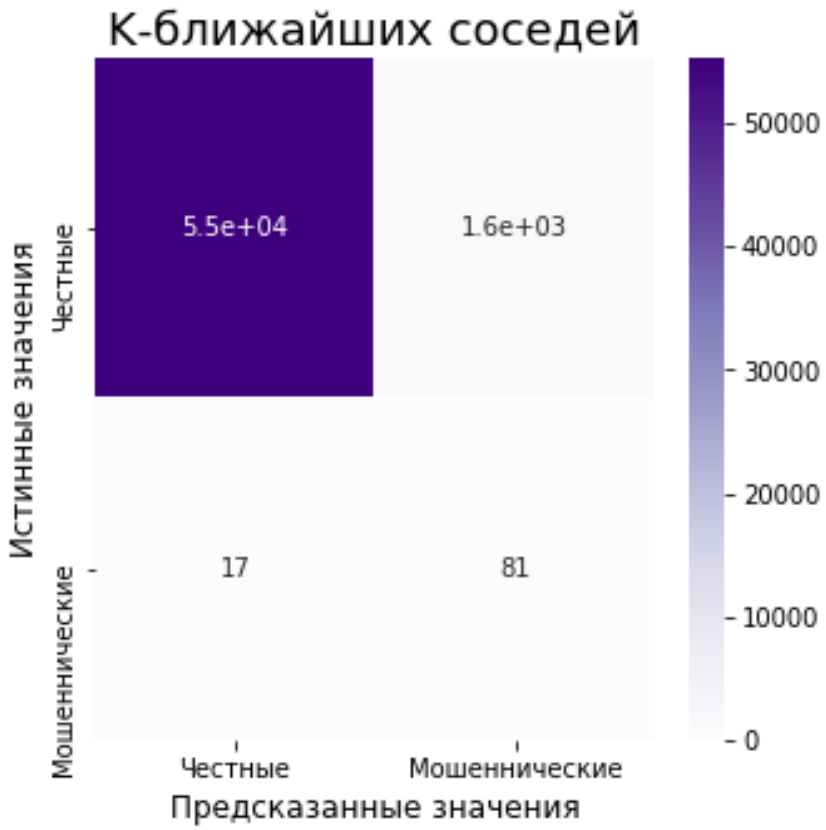


Рисунок 55 – Матрица путаницы для метода К-ближайших соседей

```
_____________________________
Случайный лес

_____________________________
Результат кроссвалидации:
accuracy:92.297 %
precision:2.057 %
recall:93.401 %
f1:4.024 %
Roc Auc:92.848 %

_____________________________

_____________________________
Результат тестирования:
accuracy:92.163 %
precision:1.895 %
recall:87.755 %
f1:3.710 %
Roc Auc:89.963 %

_____________________________
```

Рисунок 56 – Оценки для метода случайный лес

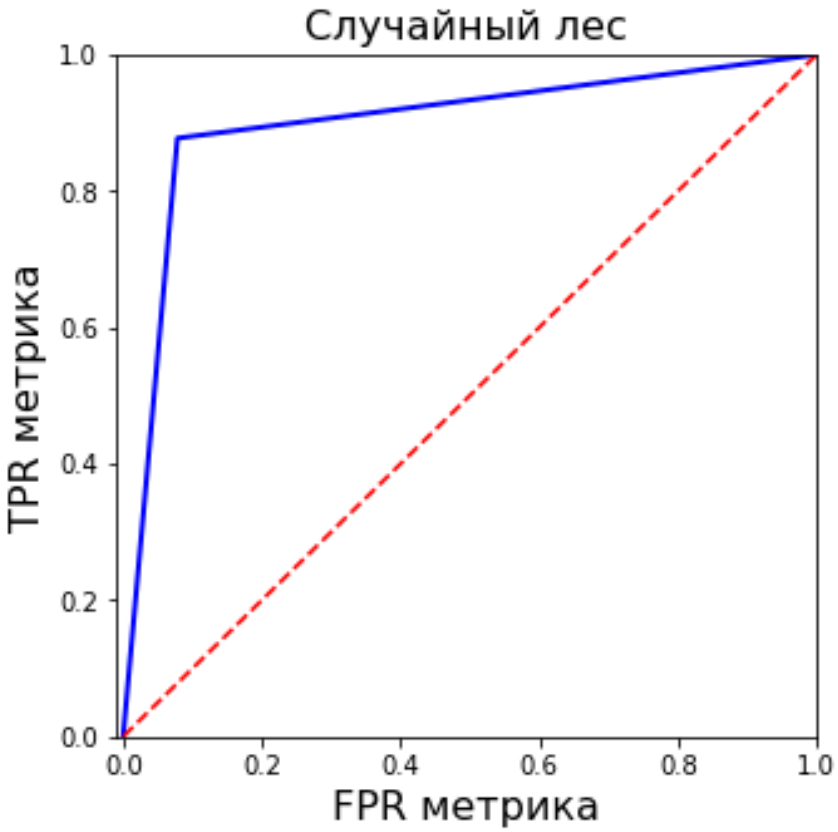


Рисунок 57 – ROC AUC кривая для алгоритма случайный лес

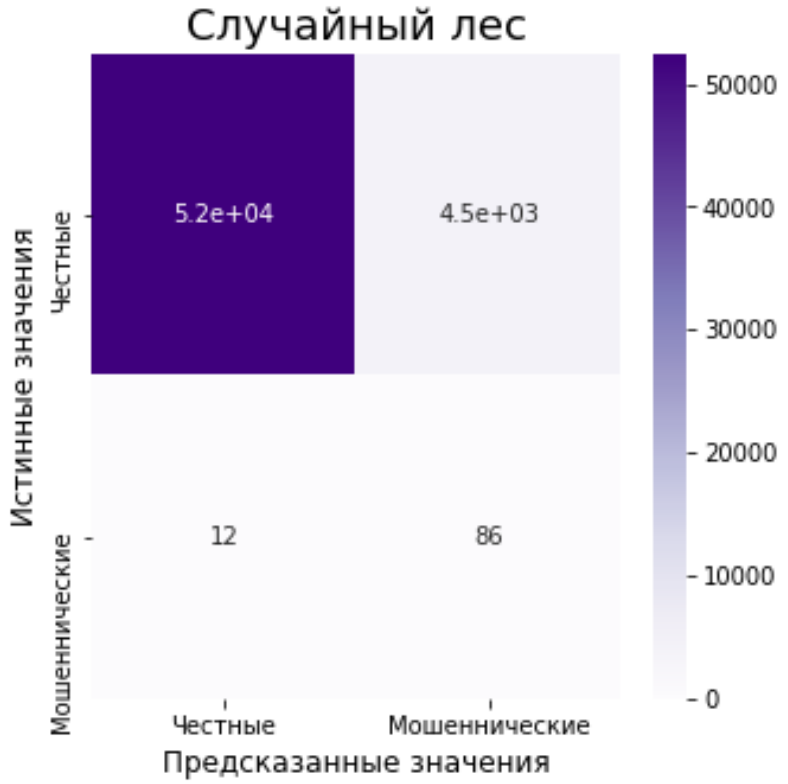


Рисунок 58 – Матрица путаницы для алгоритма случайный лес

ADASYN

```
_____________________________
Логистическая регрессия

_____________________________
Результат кроссвалидации:
accuracy:90.989 %
precision:1.809 %
recall:95.939 %
f1:3.552 %
Roc Auc:93.460 %

_____________________________
_____________________________
Результат тестирования:
accuracy:90.878 %
precision:1.742 %
recall:93.878 %
f1:3.420 %
Roc Auc:92.375 %

_____________________________
```

Рисунок 59 – Оценки для метода логистической регрессии

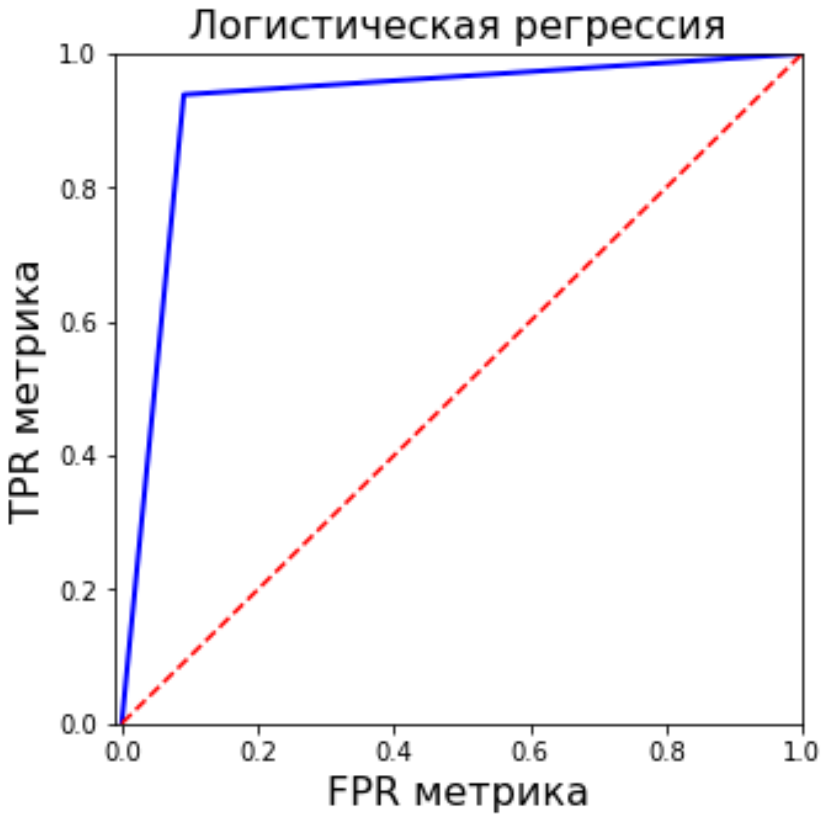


Рисунок 60 – ROC AUC кривая для метода логистической регрессии

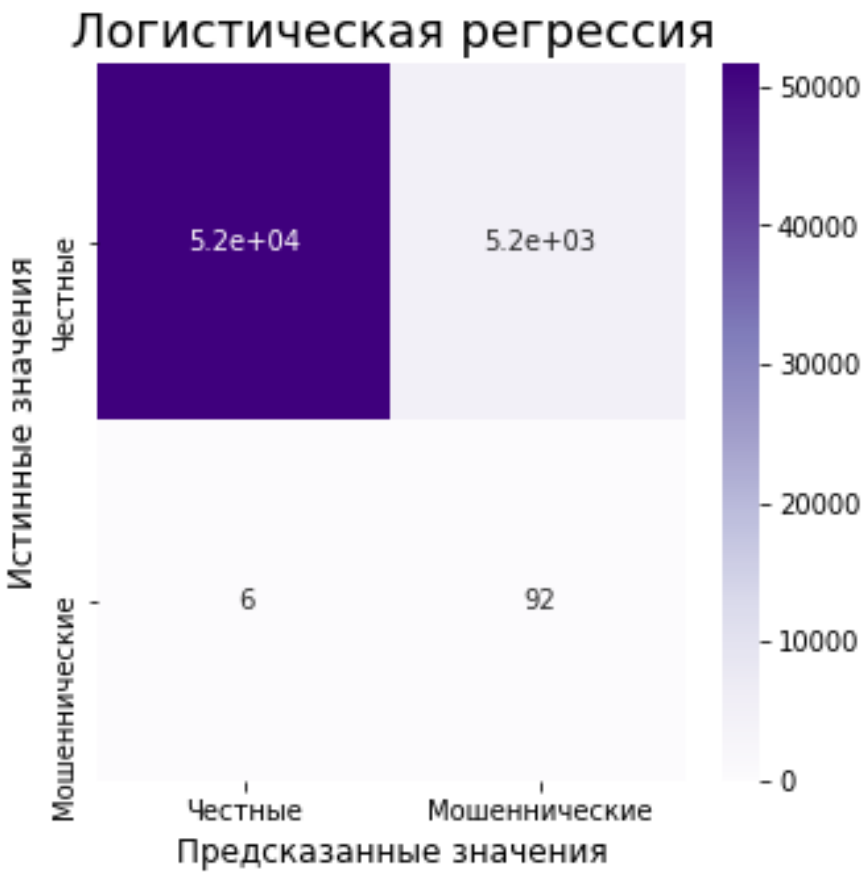


Рисунок 61 – Матрица путаницы для алгоритма логистической регрессии

```
_____________________________
Метод опорных векторов

_____________________________
Результат кроссвалидации:
accuracy:96.901 %
precision:4.911 %
recall:92.132 %
f1:9.324 %
Roc Auc:94.521 %

_____________________________

_____________________________
Результат тестирования:
accuracy:97.012 %
precision:5.145 %
recall:93.878 %
f1:9.756 %
Roc Auc:95.447 %

_____________________________
```

Рисунок 62 – Оценки для метода опорных векторов

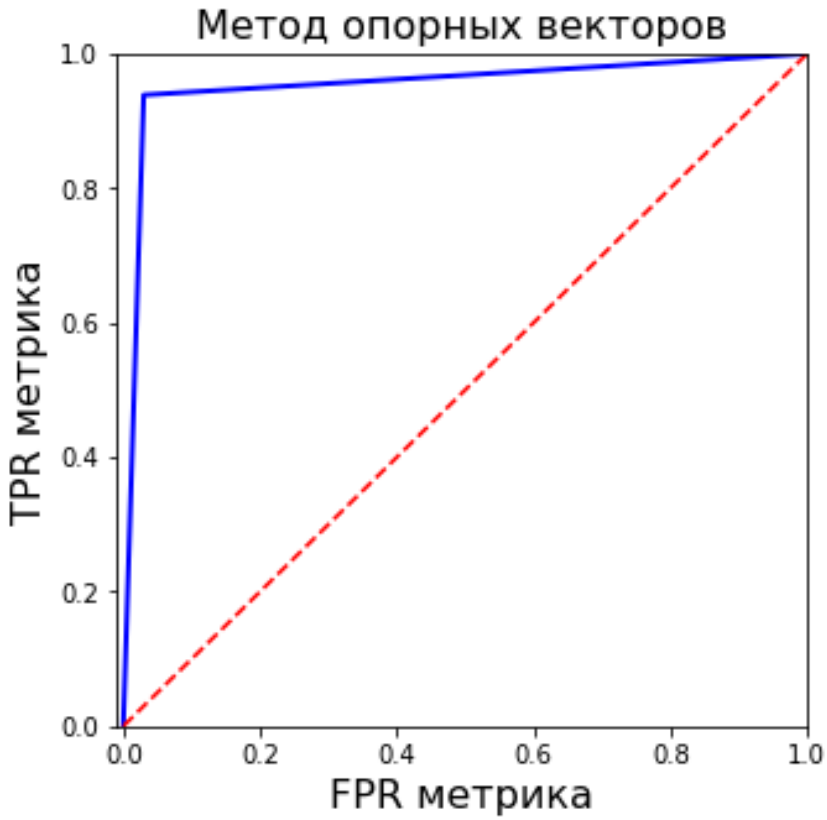


Рисунок 63 – ROC AUC кривая для метода опорных векторов

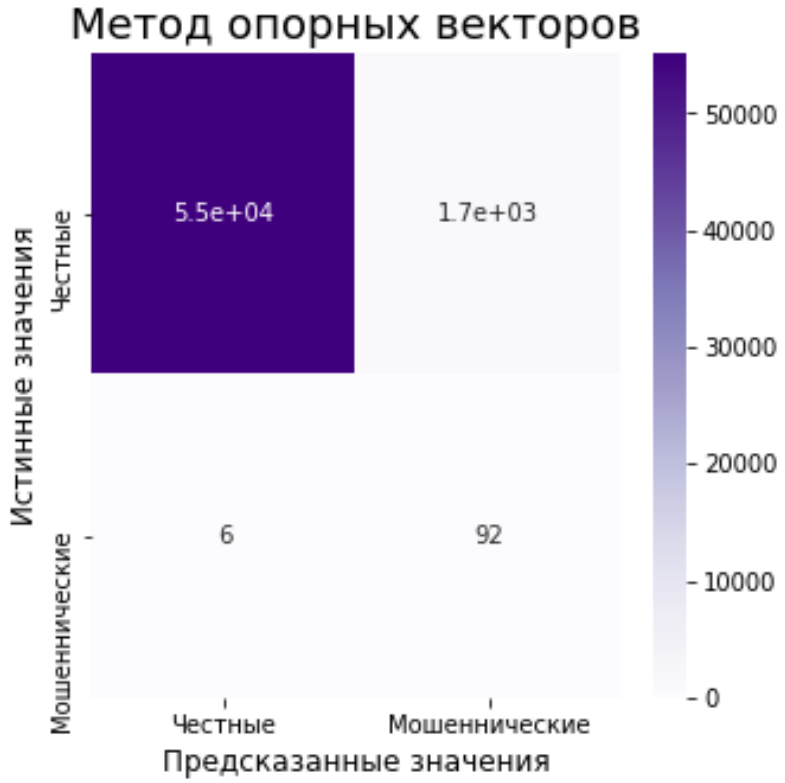


Рисунок 64 – Матрица путаницы для метода опорных векторов

```
_____________________________
K-ближайших соседей

_____________________________
Результат кроссвалидации:
accuracy:98.761 %
precision:11.232 %
recall:89.340 %
f1:19.955 %
Roc Auc:94.058 %

_____________________________
_____________________________
Результат тестирования:
accuracy:98.847 %
precision:12.076 %
recall:90.816 %
f1:21.317 %
Roc Auc:94.838 %

_____________________________
```

Рисунок 65 – Оценки для метода К-ближайших соседей

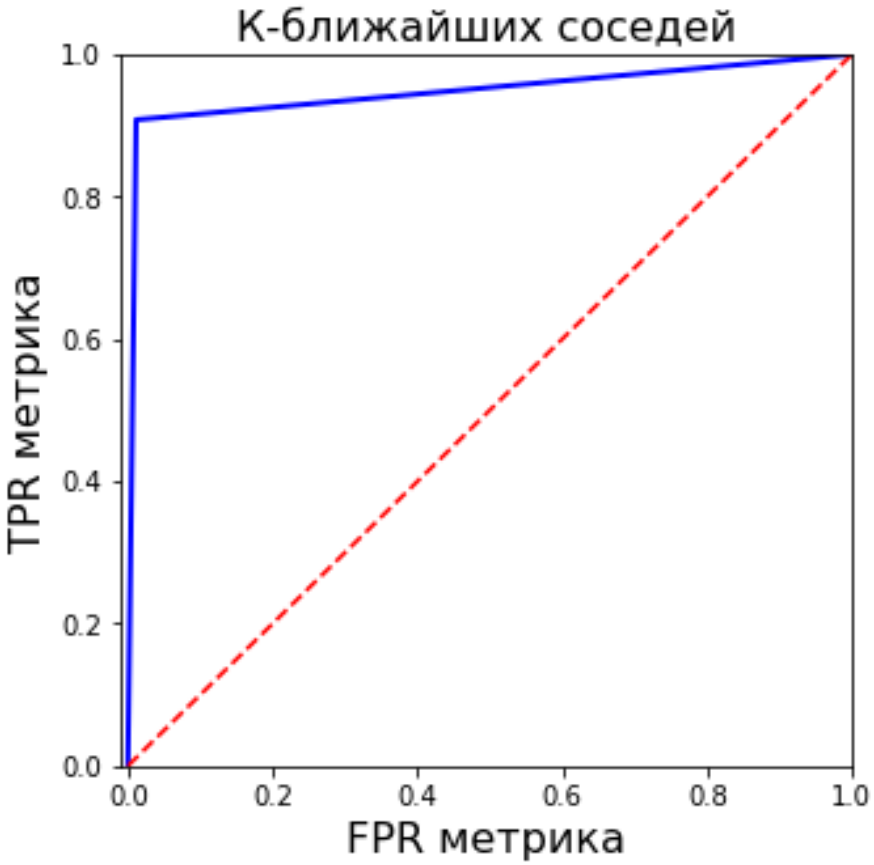


Рисунок 66 – ROC AUC кривая для метода К-ближайших соседей

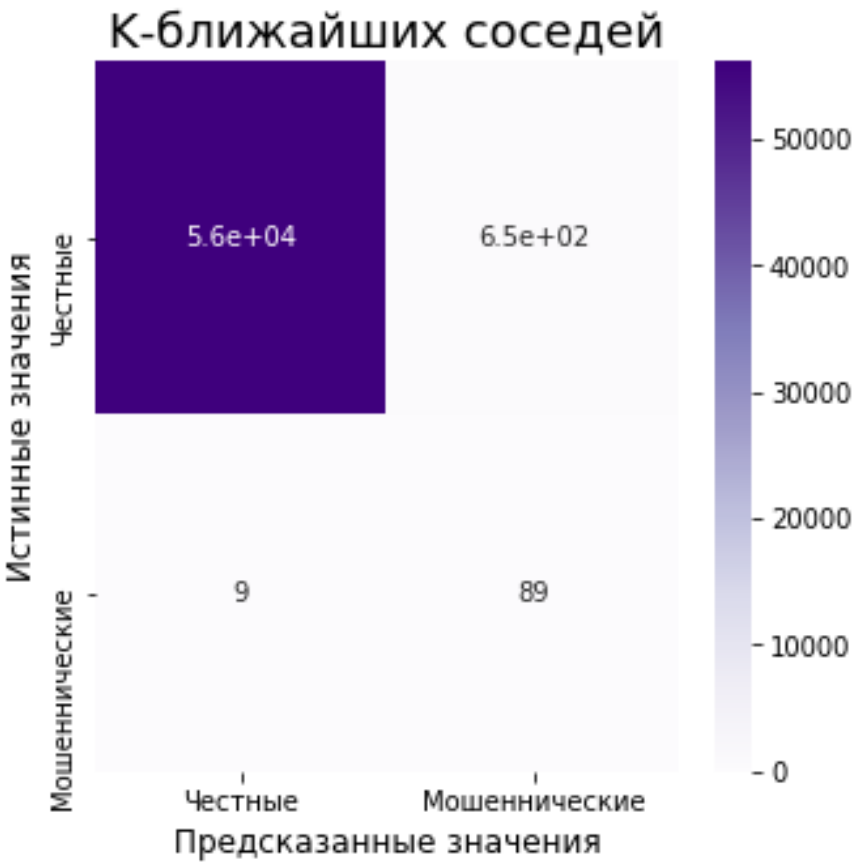


Рисунок 67 – Матрица путаницы для метода К-ближайших соседей

```
_____________________________
Случайный лес

_____________________________
Результат кроссвалидации:
accuracy:98.463 %
precision:9.191 %
recall:88.832 %
f1:16.659 %
Roc Auc:93.656 %

_____________________________
_____________________________
Результат тестирования:
accuracy:98.469 %
precision:9.177 %
recall:88.776 %
f1:16.635 %
Roc Auc:93.631 %

_____________________________
```

Рисунок 68 – Оценки для метода случайный лес

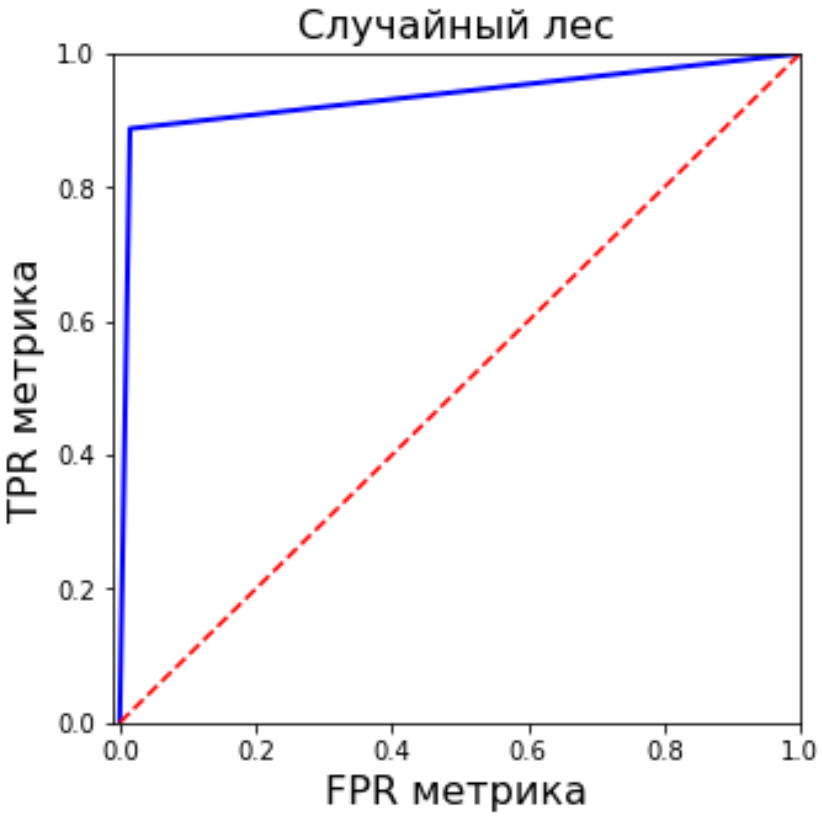


Рисунок 69 – ROC AUC кривая для алгоритма случайный лес

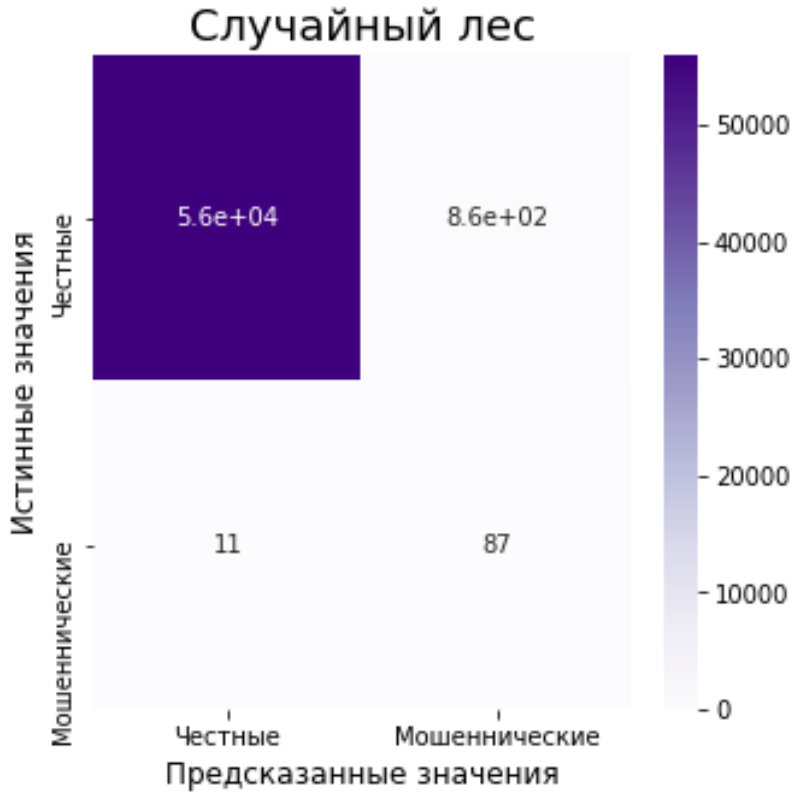


Рисунок 70 – Матрица путаницы для алгоритма случайный лес

3.2.6 Результат тестирование нейронной сети с применением алгоритма недостаточной выборки

RandomUnderSampler

```
Loss: 60.69540453235075 %
Accuracy: 70.41360907271515 %
```

Рисунок 71 – Оценки функции потерь и точности для нейронной сети

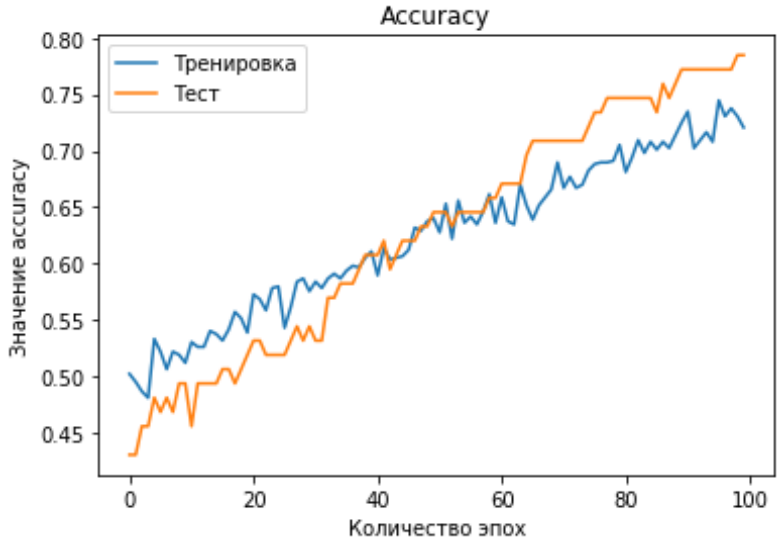


Рисунок 72 – Кривые точности обучения и тестирования нейронной сети

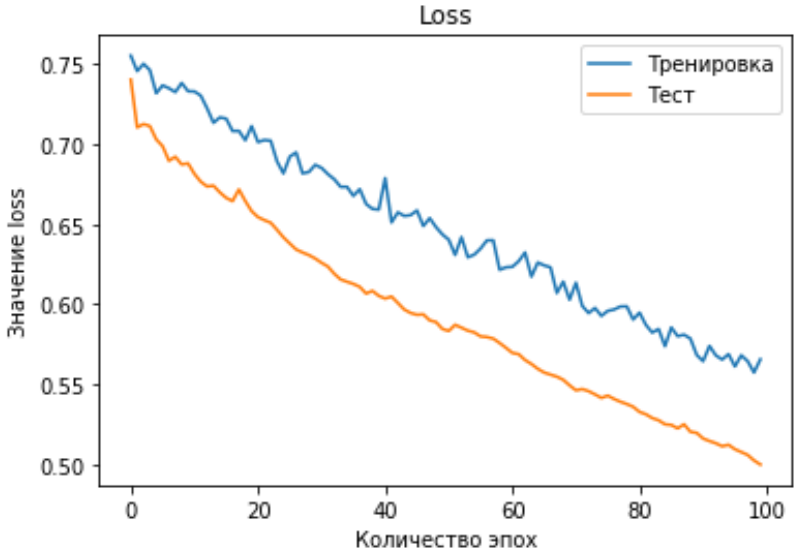


Рисунок 73 – Кривые функции потерь обучения и тестирования нейронной сети

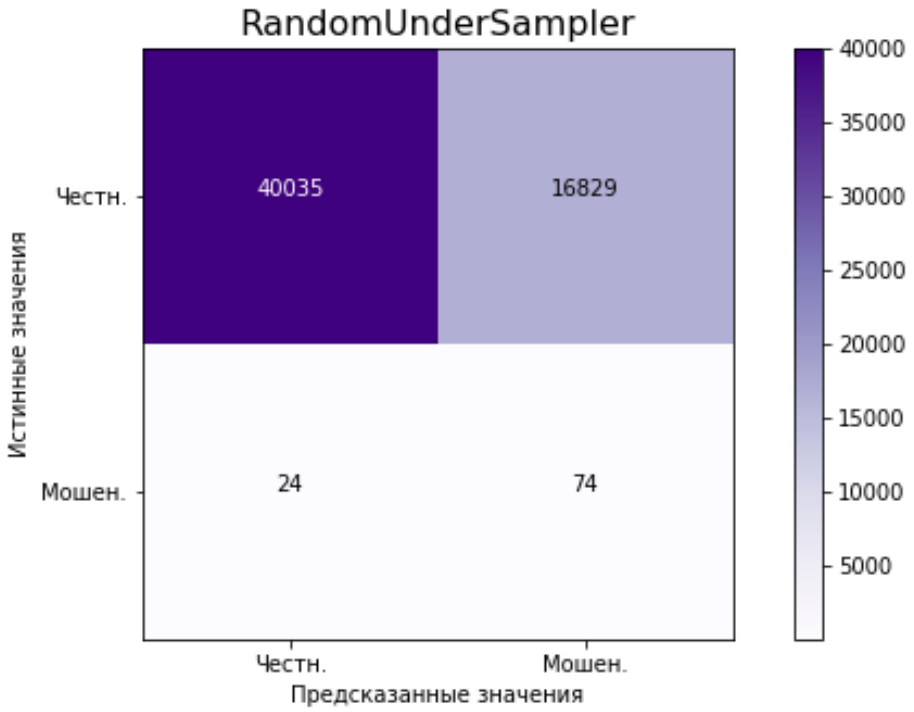


Рисунок 74 – Матрица путаницы для нейронной сети

Используя матрицу путаницы, подсчитаем значения требуемых метрик.

Precision = 0.44%

Recall = 75.51%

NearMiss

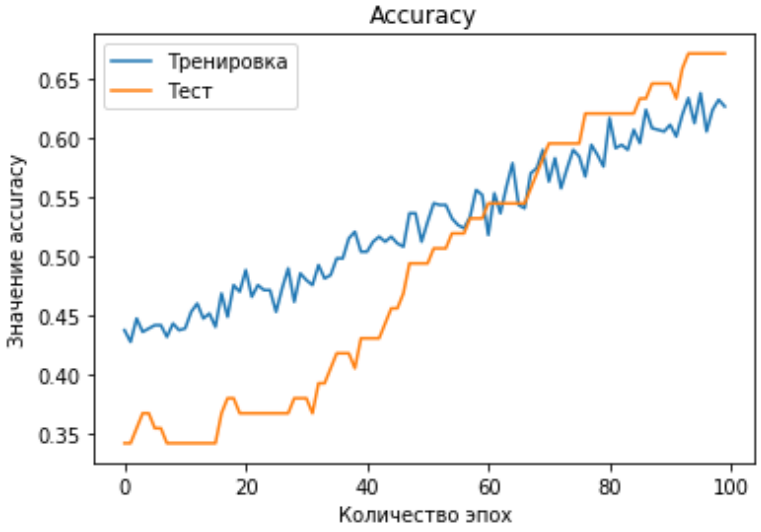


Рисунок 76 – Кривые точности обучения и тестирования нейронной сети

```
Функция потерь: 69.1586739990317 %
Точность (accuracy): 61.084582704258985 %
```

Рисунок 75 – Оценки функции потерь и точности для нейронной сети

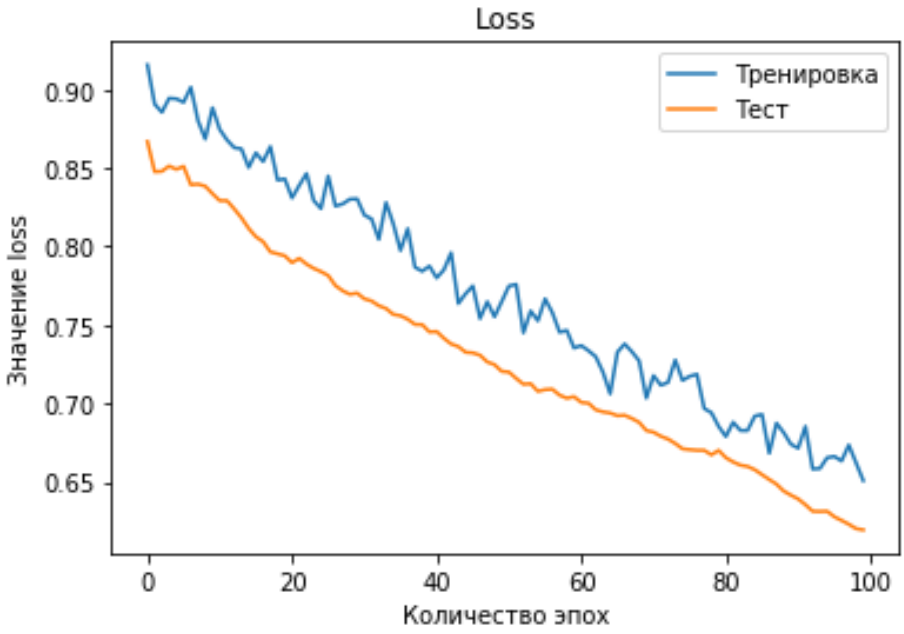


Рисунок 77 – Кривые функции потерь обучения и тестирования нейронной сети

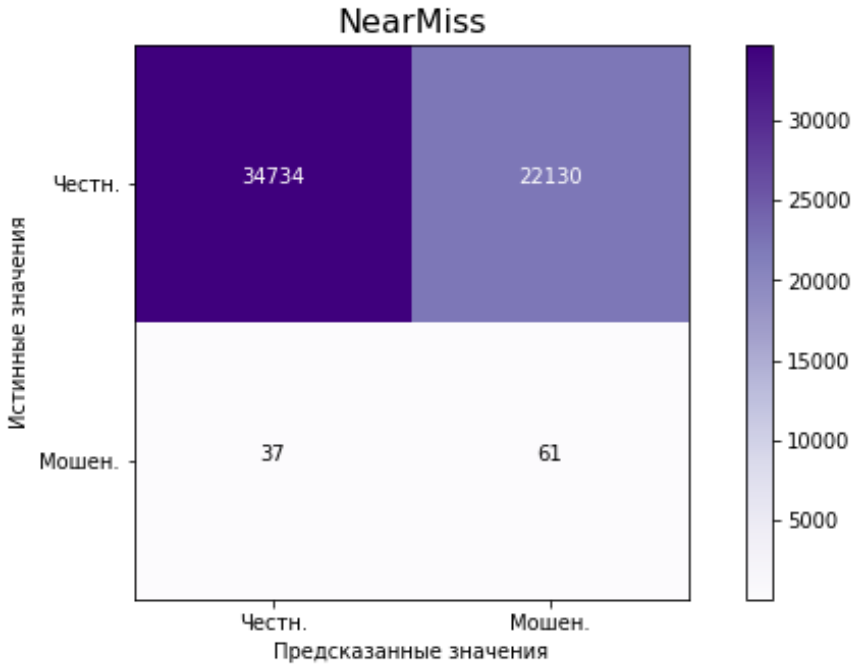


Рисунок 78 – Матрица путаницы для нейронной сети

Используя матрицу путаницы, подсчитаем значения требуемых метрик.

Precision = 0.28%

Recall = 62.24%

3.2.7 Результат тестирование нейронной сети с применением алгоритма избыточной выборки

SMOTE

```
Функция потерь: 0.7617562985137819 %
Точность (accuracy): 83.7997210284049 %
```

Рисунок 79 – Оценки функции потерь и точности для нейронной сети

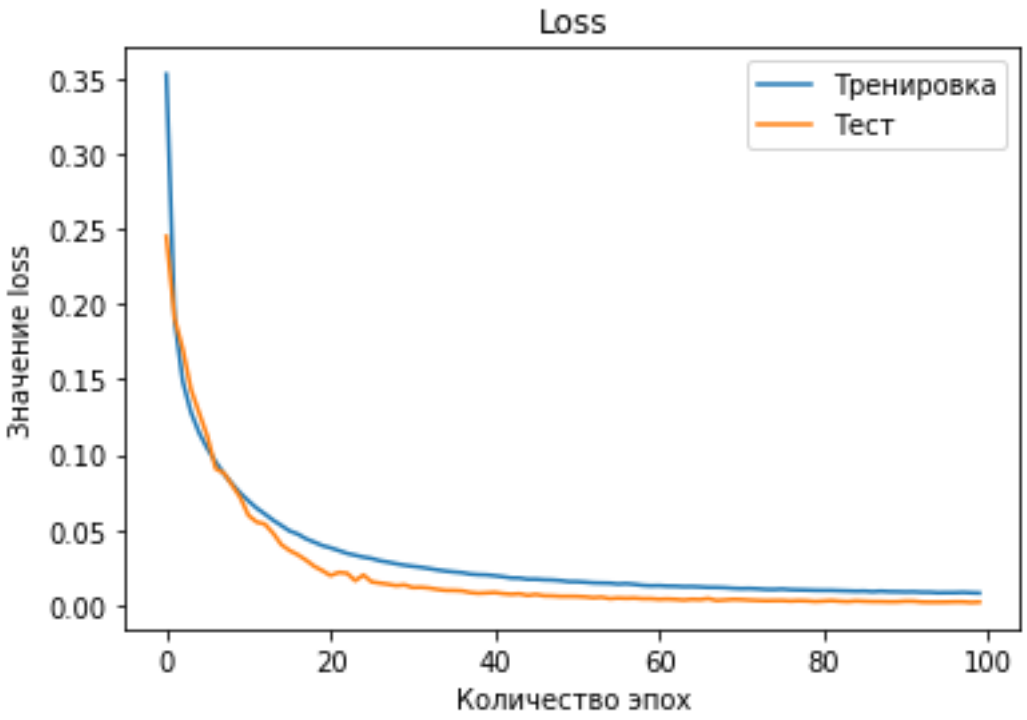


Рисунок 80 – Кривые функции потерь обучения и тестирования нейронной сети

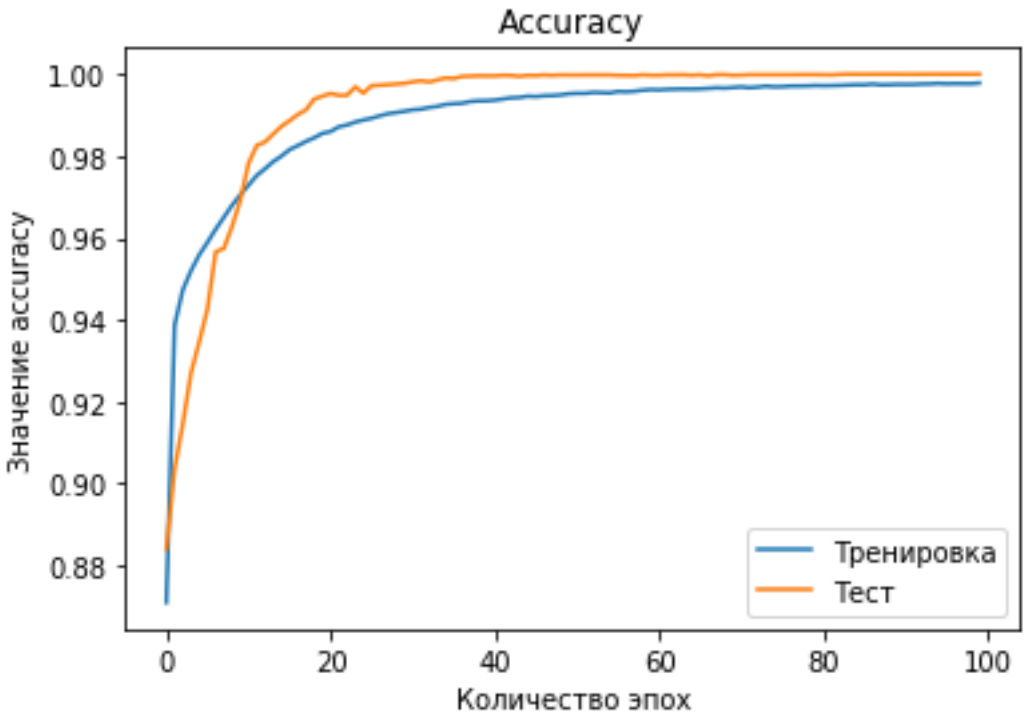


Рисунок 81 – Кривые точности обучения и тестирования нейронной сети

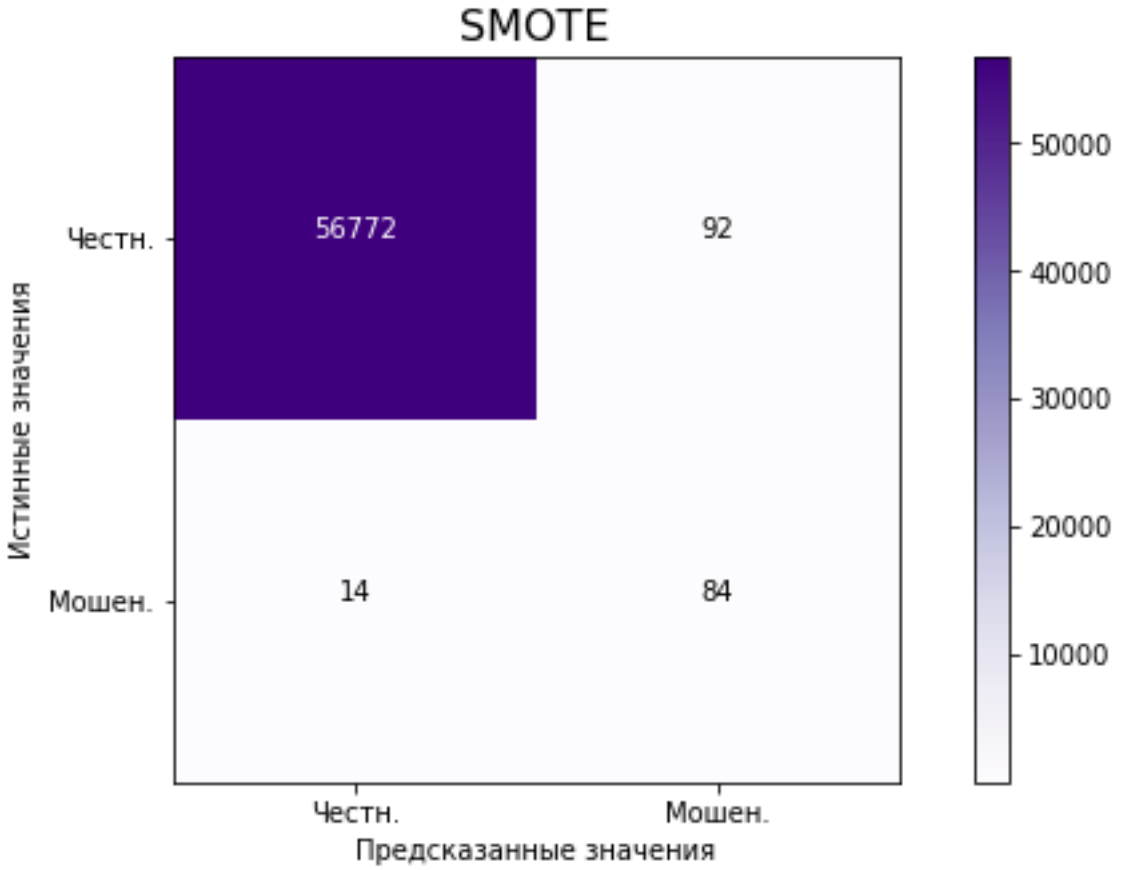


Рисунок 82 – Матрица путаницы для нейронной сети

Используя матрицу путаницы, подсчитаем значения требуемых метрик.

Precision = 47.72%

Recall = 85.71%

ADASYN

```
Функция потерь: 0.8101102842928735 %
Точность (accuracy): 99.76475545100242 %
```

Рисунок 83 – Оценки функции потерь и точности для нейронной сети

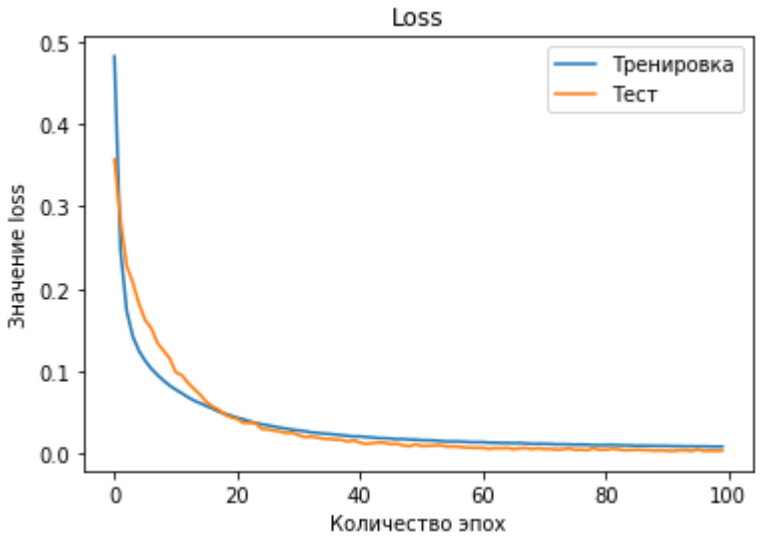


Рисунок 84 – Кривые функции потерь обучения нейронной сети

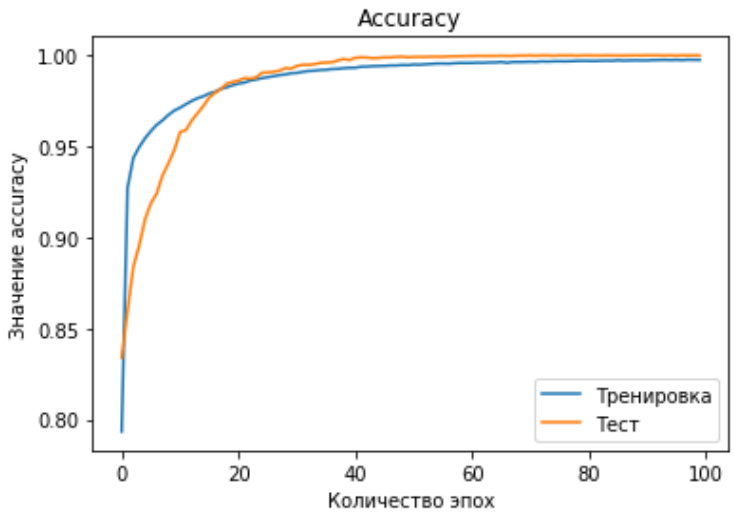


Рисунок 85 – Кривые точности обучения и тестирования нейронной сети

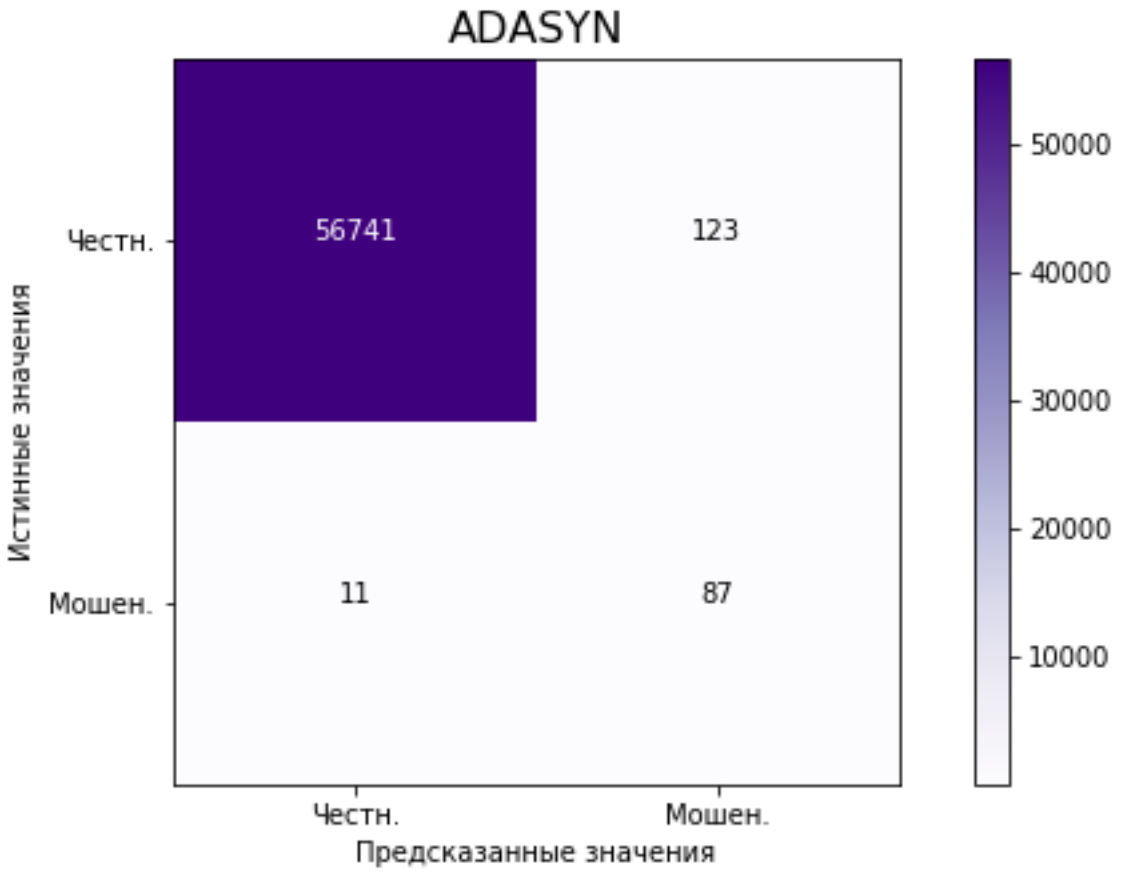


Рисунок 86 – Матрица путаницы нейронной сети

Используя матрицу путаницы, подсчитаем значения требуемых метрик.

Precision = 41.43%

Recall = 88.78%

3.2.8 Подведение итогов

Структурируем полученные данные по всем моделям и алгоритмам в сводную таблицу. Результат представлен в приложении Г.

Алгоритмы недостаточной выборки довольно хорошо себя показали при распознавании истинных мошеннических транзакций. Однако точность предсказания честных транзакций крайне низка как среди классификаторов, так и при работе нейронной сети. И это очевидно – слишком высокая потеря данных при создании недостаточной.

Лучшие показатели распознавания мошенничества среди алгоритмов несбалансированной выборки демонстрирует алгоритм NearMiss. Здесь наиболее точным в предсказании мошенничества оказался классификатор «случайный лес» (95.92%). Однако крайне низким, является показатель ошибки при распознавании мошенничества. Так значения метрики precision для алгоритма NearMiss не превышают 0.5% процента, в то время как при методе RandomUnderSampling данная цифра находится в среднем выше 6%.

Наилучший показатель распознавания действительных мошеннических транзакций продемонстрировал метод логистической регрессии при использовании алгоритма RandomUnderSampler (10.17%).

Алгоритмы избыточной выборки вполне сопоставимы с методами андерсэмплинга в точности определения мошеннических транзакций. Так наилучший показатель по метрике recall получили метод логистической регрессии и метод опорных векторов (93.88%)

Несмотря на то, что в среднем показатель precision для методов избыточной выборки выше, чем для недостаточной (лучшее значение – 12.08%) – это по-прежнему крайне низкий показатель. И он означает, что сотни и тысячи честных клиентов получат неудобство в ход работы такой системы.

Нейронная сеть в условиях недостаточной выборки показала самые худшие результаты по обеим метрикам. Оценка precision для нейронной сети не превышает и процента. Метрика recall оценивается как 75.51% и 62.24% для алгоритмов RandomUnderSampling и NearMiss соответственно.

При использовании алгоритмов избыточной выборки нейронная сеть демонстрирует самые высокие результаты по оценке precision. Для алгоритма SMOTE составляет 47.72%, для алгоритма ADASYN 41.43%. Оценка precision тоже показывает хорошие результаты на уровне свыше 85%.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы был проведен анализ проблемы обнаружения мошеннических транзакций в банковских операциях. Были изучены современные способы противодействия злоумышленникам. Выявлены преимущества и проблемы при использовании машинного обучения в борьбе с мошенничеством. Большое внимание было уделено проблеме несбалансированной выборки данных, неизбежно возникающей при работе с реальными банковскими сведениями.

Проведенное практическое исследование позволило выявить сильные и слабые стороны рассматриваемых классификаторов. Так большинство статистических классификаторов показало хороший процент распознавания истинных мошеннических транзакций в среднем на уровне 89%. Существенным недостатком таких моделей является процент ложных срабатываний. Так из числа распознанных мошеннических операций, в среднем реально мошенническими оказались лишь 4.8%.

Исходя из полученных результатов можно заключить, что наиболее эффективной моделью, соответствующей всем поставленным требованиям, является модель, основанная на нейронной сети с применением алгоритмов избыточной выборки.

Также важно отметить, что наиболее эффективной стратегией выявления мошенничества в банковских операциях является комплексный подход. Именно применение различных методик и алгоритмов при создании модели машинного обучения позволит эффективно противостоять злоумышленникам.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Tuning the hyper-parameters of an estimator// Scikit Learn [Электронный ресурс]. URL: https://scikit-learn.org/stable/modules/grid_search.html#grid-search (дата обращения 03.04.2019).

2. Основы анализа данных на python с использованием pandas+sklearn// Хабр [Электронный ресурс]. URL: https://habr.com/ru/post/202090/ (дата обращения 06.03.2019).

3. How to Use Metrics for Deep Learning with Keras in Python// Machine Learning Mastery [Электронный ресурс]. URL: https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/ (дата обращения 25.04.2019).

4. Credit Card Fraud Detection // Kaggle [Электронный ресурс]. URL: https://www.kaggle.com/mlg-ulb/creditcardfraud (дата обращения 25.02.2019).

5. Метрики в задачах машинного обучения // Хабр [Электронный ресурс]. URL: https://habr.com/ru/company/ods/blog/328372/ (дата обращения 16.04.2019).

6. N. V. Chawla, SMOTE: synthetic minority over-sampling technique [Текст] / N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer // Journal of artificial intelligence research. – 2016. – С. 321-357

7. H. Han, “Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning [Текст] / H. Han, W. Wen-Yuan, M. Bing-Huan // Advances in intelligent computingю. – 2015. – С. 878-887

8. H. M. Nguyen Borderline over-sampling for imbalanced data classification [Текст] / H. M. Nguyen, E. W. Cooper, K. Kamei // International Journal of Knowledge Engineering and Soft Data Paradigms. – 2010. – С. 4-21

9. Д. Грас Data Science. Наука о данных с нуля [Текст]: учебно-методическое пособие / Д.Грас. – СПБ.: БХВ-Петербург, 2017. – 478 с.

10. Д. Силен Основы Data Science и Big Data. Python и наука о данных [Текст]: учебно-методическое пособие / Д.Силен, А. Мейсман, М. Али – СПБ.: Питер, 2017. – 376 с.

11. Дж. Вандер Плас Python для сложных задач наука о данных и машинное обучение [Текст]: учебное пособие / Дж. Вандер Плас – СПБ.: Питер, 2018. – 576 с.

12. А.И. Орлов Прикладная статистика [Текст]: учебное пособие / А.И. Орлов – М.: Экзамен, 2014. – 656 с.

13. K. McCarthy Does Cost-Sensitive Learning Beat Sampling for Classifying Rare Classes? / K. McCarthy, Zabar B., Weiss G.M. // Proceedings of the First International Workshop on Utility-Based Data Mining (at KDD-05), ACM Press, 2015. – С. 69-75.

14. Ch. Elchan The Foundations of Cost-Sensitive Learning / Ch. Elchan // Proc. of the 17th International Joint Conference on Artificial Intelligence, 2016. – С. 973-978.

15. M. Kubat Addressing the curse of imbalanced training sets: one-sided selection / M. Kubat, S. Matwin // In: Proc. 14th International Conference on Machine Learning, 2017. – С. 179-186.

16. С.В. Аксенов Организация и использование нейронных сетей [текст] / Аксенов С.В., Новосельцев В.Б.; под ред. В.Б. Новосельцева. – Томск: Изд-во НТЛ, 2016. – 128 с.

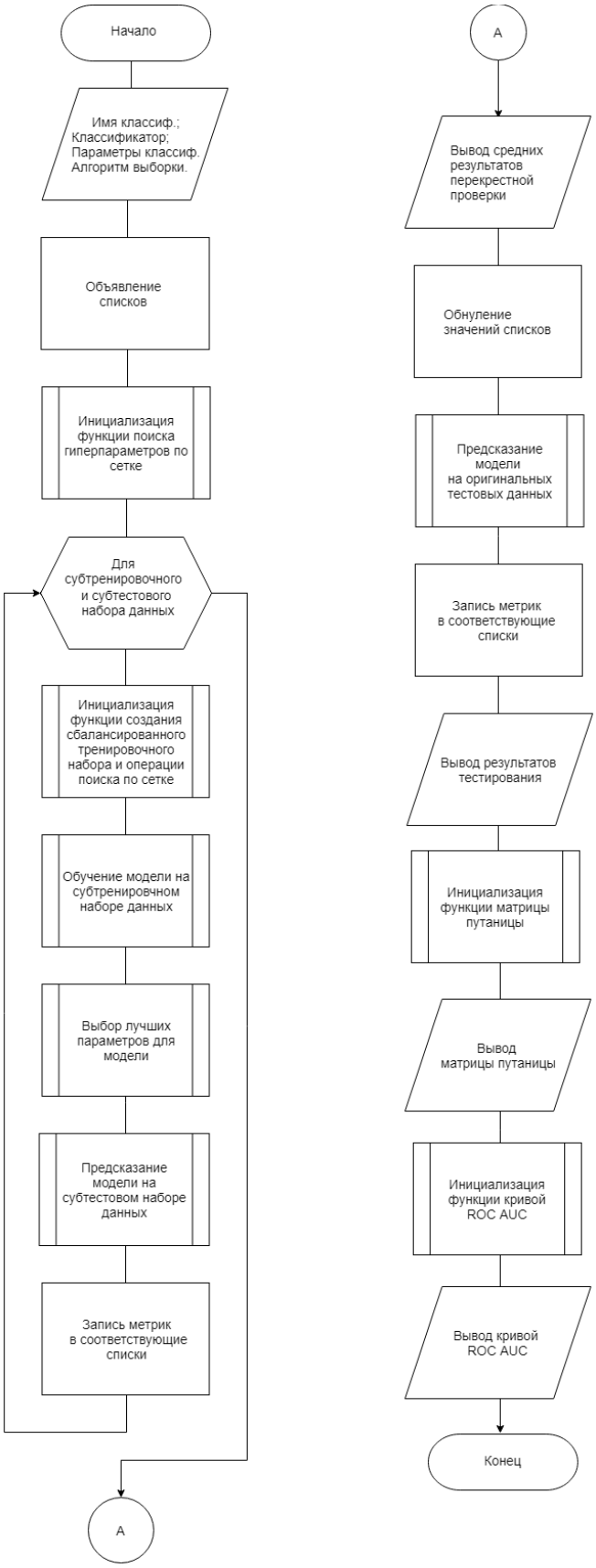
ПРИЛОЖЕНИЕ А

Результат запроса общей информации по данным

```
[5 rows x 31 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Time      284807 non-null float64
V1        284807 non-null float64
V2        284807 non-null float64
V3        284807 non-null float64
V4        284807 non-null float64
V5        284807 non-null float64
V6        284807 non-null float64
V7        284807 non-null float64
V8        284807 non-null float64
V9        284807 non-null float64
V10       284807 non-null float64
V11       284807 non-null float64
V12       284807 non-null float64
V13       284807 non-null float64
V14       284807 non-null float64
V15       284807 non-null float64
V16       284807 non-null float64
V17       284807 non-null float64
V18       284807 non-null float64
V19       284807 non-null float64
V20       284807 non-null float64
V21       284807 non-null float64
V22       284807 non-null float64
V23       284807 non-null float64
V24       284807 non-null float64
V25       284807 non-null float64
V26       284807 non-null float64
V27       284807 non-null float64
V28       284807 non-null float64
Amount    284807 non-null float64
Class     284807 non-null int64
dtypes: float64(30), int64(1)
```

ПРИЛОЖЕНИЕ Б

Блок схема функции для классификаторов:



ПРИЛОЖЕНИЕ В

Блок-схема функции для нейронной сети:

ПРИЛОЖЕНИЕ Г

Таблица 1 – Результаты тестирования моделей машинного обучения

	Random Under Sampler		**Near Miss**		**SMOTE**		**ADASYN**	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
Метод опорных векторов	6.78%	91.84%	0.19%	93.88%	4.64%	85.71%	5.15%	93.88%
К-ближайших соседей	6.66%	88.78%	0.37%	89.79%	4.84%	82.65%	12.08%	90.82%
Случайный лес	6.64%	89.79%	0.24%	**95.92%**	1.89%	87.75%	9.18%	88.78%
Логистическая регрессия	10.17%	87.76%	0.49%	92.86%	6.65%	89.80%	1.74%	93.88%
Нейронная сеть	0.44%	75.51%	0.28%	62.24%	**47.72%**	85.71%	41.43%	88.78%

ПРИЛОЖЕНИЕ Д

Fraud_detection_system.py:

```
import warnings
import itertools
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.preprocessing import RobustScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import recall_score, roc_curve, accuracy_score
from sklearn.metrics import precision_score, roc_auc_score, f1_score,
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import ADASYN, SMOTE
from imblearn.under_sampling import RandomUnderSampler, NearMiss
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers.core import Dense
from keras.layers.normalization import BatchNormalization
from imblearn.pipeline import make_pipeline as
imbalanced_make_pipeline

# Фильтр предупреждений
warnings.simplefilter("ignore")
```

```
original_df =
pd.read_csv("C:/Users/karpo/Desktop/VKR/Datasets/CreditCardOne/DiplomDa
taset.csv")
original_df.head
print('Честные транзакции',
round(original_df['Class'].value_counts()[0]/len(original_df) * 100,2), '% от
всех операций')
print('Мошеннические транзакции',
round(original_df['Class'].value_counts()[1]/len(original_df) * 100,2), '% от
всех операций')

colors = ["#FFA500", "#20B2AA"]

sns.countplot('Class', data=original_df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=16)

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(15,5))

ax1.hist(original_df.Time[original_df.Class == 1], bins = 100)
ax1.set_title('Мошеннические транзакции')
ax2.hist(original_df.Time[original_df.Class == 0], bins = 100)
ax2.set_title('Честные транзакции')

plt.xlabel('Время (сек.)')
plt.ylabel('Число транзакций')
plt.show()

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(15,5))
ax1.hist(original_df.Amount[original_df.Class == 1], bins = 5)
```

```
ax1.set_title('Мошеннические транзакции')
ax2.hist(original_df.Amount[original_df.Class == 0], bins = 5)
ax2.set_title('Честные транзакции')
plt.xlabel('Сумма')
plt.ylabel('Число транзакций')
plt.yscale('log')
plt.show()

# Масштабирование
rob_scaler = RobustScaler()

original_df['rubust_time'] =
RobustScaler().fit_transform(original_df['Time'].values.reshape(-1,1))
original_df['robust_amount'] =
RobustScaler().fit_transform(original_df['Amount'].values.reshape(-1,1))

original_df.drop(['Time','Amount'], axis=1, inplace=True)

rubust_time = original_df['rubust_time']
robust_amount = original_df['robust_amount']

original_df.drop(['robust_amount', 'rubust_time'], axis=1, inplace=True)

original_df.insert(0,  'robust_amount', robust_amount)
original_df.insert(1,  'rubust_time', rubust_time)

original_df.head
```

```
# Случайное разбиение оригинальных данных на обучающие и тестовые выборки
X = original_df.drop('Class', axis=1)# Данные без целевого признака
y = original_df['Class']# Целевой признак

strat_cross_val = StratifiedShuffleSplit(n_splits=10,test_size= 0.2, random_state=123)

for index_train, index_test in strat_cross_val.split(X, y):
    original_train_data, original_test_data = X.iloc[index_train], X.iloc[index_test]
    original_train_index, original_test_index = y.iloc[index_train], y.iloc[index_test]

# Переведем в массив
original_train_data = original_train_data.values
original_test_data = original_test_data.values
original_train_index = original_train_index.values
original_test_index = original_test_index.values

def custom_confusion_matrix(cm, classes, normalize=False,title=''):

    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Purples)
    plt.title(title, fontsize=16)
    plt.colorbar()

    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)
```

```
        fmt = '.2f' if normalize  else 'd'
        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center",
color="white" if cm[i, j] > thresh else "black")

        plt.ylabel('Истинные значения')
        plt.xlabel('Предсказанные значения')

    def custom_roc_curve(FPR, TPR, roc_name):
        plt.figure(figsize=(5,5))
        plt.title(roc_name, fontsize=16)
        plt.plot(FPR, TPR, 'b-', linewidth=2)
        plt.plot([0, 1], [0, 1], 'r--')# диагональ
        plt.xlabel('FPR метрика', fontsize=16)
        plt.ylabel('TPR метрика', fontsize=16)
        plt.axis([-0.01,1,0,1])
        plt.show()

    rf_prop = {'criterion': ['entropy', 'gini'], 'max_depth': [10, 20, 100],
'n_estimators': [10, 20, 100]}
    svc_prop = { 'C': [0.4, 0.5, 0.6, 0.7, 0.9, 1], 'kernel': ['rbf', 'poly', 'sigmoid',
'linear'] }
    knn_prop = {"n_neighbors": [2, 3, 4, 5, 6, 7, 8, 8, 9, 10] , 'algorithm':
['auto', 'ball_tree', 'kd_tree', 'brute']}
    lr_prop = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

    # Реализация функции тестирования классификаторов
```

```
    def ClassifierTesting(method_name,method,  method_prop,
sampling_algoritm):

        acc_lst = []
        prec_lst = []
        rec_lst = []
        f1_lst = []
        auc_lst = []

        random_search = RandomizedSearchCV(method, method_prop,
n_iter=20)

        for train,  test in strat_cross_val.split(original_train_data,
original_train_index):

            cross_val_model = imbalanced_make_pipeline(sampling_algoritm,
random_search) # SMOTE happens during Cross Validation  not before..
            cross_val_model.fit(original_train_data[train],
original_train_index[train])

            best_est = random_search.best_estimator_
            prediction = best_est.predict(original_train_data[test])

            acc_lst.append(cross_val_model.score(original_train_data[test],
original_train_index[test]))
            prec_lst.append(precision_score(original_train_index[test],
prediction))
            rec_lst.append(recall_score(original_train_index[test],  prediction))
            f1_lst.append(f1_score(original_train_index[test],  prediction))
```

```
    auc_lst.append(roc_auc_score(original_train_index[test],  prediction))

print('_' * 50)
print(method_name)
print('_' * 50)
print('Результат перекрестной проверки:')
print("accuracy:{0:.3f}".format(np.mean(acc_lst)*100),'%')
print("precision:{0:.3f}".format(np.mean(prec_lst)*100),'%')
print("recall:{0:.3f}".format(np.mean(rec_lst)*100),'%')
print("f1:{0:.3f}".format(np.mean(f1_lst)*100),'%')
print("Roc Auc:{0:.3f}".format(np.mean(auc_lst)*100),'%')
print('_' * 50)
print('_' * 50)

acc_lst = []
prec_lst = []
rec_lst = []
f1_lst = []
auc_lst = []

prediction = best_est.predict(original_test_data)

acc_lst.append(accuracy_score(original_test_index, prediction))
prec_lst.append(precision_score(original_test_index, prediction))
rec_lst.append(recall_score(original_test_index, prediction))
f1_lst.append(f1_score(original_test_index, prediction))
auc_lst.append(roc_auc_score(original_test_index, prediction))

print('Результат тестирования:')
```

```
            print("accuracy:{0:.3f}".format(np.mean(acc_lst)*100),'%')
            print("precision:{0:.3f}".format(np.mean(prec_lst)*100),'%')
            print("recall:{0:.3f}".format(np.mean(rec_lst)*100),'%')
            print("f1:{0:.3f}".format(np.mean(f1_lst)*100),'%')
            print("Roc Auc:{0:.3f}".format(np.mean(auc_lst)*100),'%')
            print('_' * 50)

            cm = confusion_matrix(original_test_index,  prediction)

            fig, ax = plt.subplots(1, 1,figsize=(5,5))
            sns.heatmap(cm, ax=ax, annot=True, cmap=plt.cm.Purples)
            ax.set_title(method_name,  fontsize=18)
            ax.set_xticklabels(['Честные', 'Мошеннические'], fontsize=10,
rotation=0)
            ax.set_yticklabels(['Честные', 'Мошеннические'], fontsize=10,
rotation=90)
            ax.set_xlabel(('Предсказанные значения'), fontsize=12, rotation=0)
            ax.set_ylabel(('Истинные значения'), fontsize=12, rotation=90)
            plt.show()

            FPR, TPR, none = roc_curve(original_test_index,  prediction)
            custom_roc_curve(FPR, TPR, method_name)

        # Реализация функции нейронной сети
        def NEURO(sampling_name,sampling_method):
            sm = sampling_method
            resample_date, resample_index = sm.fit_sample(original_train_data,
original_train_index)
```

```
        n_inputs = resample_date.shape[1]
        neuro_model = Sequential()
        neuro_model.add(Dense(n_inputs, input_shape=(n_inputs, ),
activation='relu'))
        neuro_model.add(BatchNormalization())
        neuro_model.add(Dense(32, activation='relu'))
        neuro_model.add(BatchNormalization())
        neuro_model.add(Dense(16, activation='relu'))
        neuro_model.add(BatchNormalization())
        neuro_model.add(Dense(2, activation='sigmoid'))

        neuro_model.compile(Adam(lr=0.00001),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        neuro_model_history=neuro_model.fit(resample_date, resample_index,
validation_split=0.2, batch_size=100, epochs=100, shuffle=True, verbose=0)
        neuro_model_predictions
=neuro_model.predict_classes(original_test_data, batch_size=100, verbose=0)

        loss, acc = neuro_model.evaluate(original_test_data,
original_test_index)

        print('Функция потерь:', loss*100,'%')
        print('Точность (accuracy):', acc*100,'%')

        neuro_model_cmr = confusion_matrix(original_test_index,
neuro_model_predictions)

        labels = ['Честн.', 'Мошен.']
        fig = plt.figure(figsize=(10,5))
```

```
            custom_confusion_matrix(neuro_model_cmr, labels,
title=sampling_name)

            # Вывод динамики изменения ошибки и точности.
            plt.figure()
            plt.plot(neuro_model_history.history['loss'])
            plt.plot(neuro_model_history.history['val_loss'])
            plt.title('Функция потерь')
            plt.ylabel('Значение функции потерь')
            plt.xlabel('Количество эпох')
            plt.legend(['Тренировка', 'Тест'], loc='best')
            plt.show()

            plt.figure()
            plt.plot(neuro_model_history.history['acc'])
            plt.plot(neuro_model_history.history['val_acc'])
            plt.title('Accuracy')
            plt.ylabel('Значение accuracy')
            plt.xlabel('Количество эпох')
            plt.legend(['Тренировка', 'Тест'], loc='best')
            plt.show()

        # Алгоритм RandomUnderSampler.
        ClassifierTesting('Метод опорных векторов',SVC(), svc_prop,
RandomUnderSampler(sampling_strategy='majority'))
        ClassifierTesting('Случайный лес',RandomForestClassifier(), rf_prop,
RandomUnderSampler(sampling_strategy='majority'))
        ClassifierTesting('K-ближайших соседей',KNeighborsClassifier(),
knn_prop, RandomUnderSampler(sampling_strategy='majority'))
```

```
ClassifierTesting('Логистическая регрессия',LogisticRegression(),
lr_prop, RandomUnderSampler(sampling_strategy='majority'))

# Алгоритм NearMiss.
ClassifierTesting('Метод опорных векторов',SVC(), svc_prop,
NearMiss(version=1,sampling_strategy='majority',n_jobs=-1))
ClassifierTesting('Случайный лес',RandomForestClassifier(), rf_prop,
NearMiss(version=1,sampling_strategy='majority',n_jobs=-1))
ClassifierTesting('K-ближайших соседей',KNeighborsClassifier(),
knn_prop, NearMiss(version=1,sampling_strategy='majority',n_jobs=-1))
ClassifierTesting('Логистическая регрессия',LogisticRegression(),
lr_prop, NearMiss(version=1,sampling_strategy='majority',n_jobs=-1))

# Алгоритм SMOTE.
ClassifierTesting('Метод опорных векторов',SVC(), svc_prop,
SMOTE(sampling_strategy='minority',n_jobs=-1))
ClassifierTesting('Случайный лес',RandomForestClassifier(), rf_prop,
SMOTE(sampling_strategy='minority',n_jobs=-1))
ClassifierTesting('K-ближайших соседей',KNeighborsClassifier(),
knn_prop, SMOTE(sampling_strategy='minority',n_jobs=-1))
ClassifierTesting('Логистическая регрессия',LogisticRegression(),
lr_prop, SMOTE(sampling_strategy='minority',n_jobs=-1))

# Алгоритм ADASYN.
ClassifierTesting('Метод опорных векторов',SVC(), svc_prop,
ADASYN(sampling_strategy='minority',n_jobs=-1))
ClassifierTesting('Случайный лес',RandomForestClassifier(), rf_prop,
ADASYN(sampling_strategy='minority',n_jobs=-1))
```

```
ClassifierTesting('K-ближайших соседей',KNeighborsClassifier(),
knn_prop, ADASYN(sampling_strategy='minority',n_jobs=-1))
ClassifierTesting('Логистическая регрессия',LogisticRegression(),
lr_prop, ADASYN(sampling_strategy='minority',n_jobs=-1))

# Тестирование нейронной сети.
NEURO('RandomUnderSampler',RandomUnderSampler(sampling_strate
gy='majority',random_state=36))
NEURO('NearMiss', NearMiss(sampling_strategy='majority', version =1,
random_state=36,n_jobs=-1))
NEURO('SMOTE', SMOTE(sampling_strategy='minority',
random_state=36,n_jobs=-1))
NEURO('ADASYN', ADASYN(sampling_strategy='minority',
random_state=36,n_jobs=-1))
```