РЕФЕРАТ

Выпускная квалификационная работа содержит 45 страниц, 10 рисунков, 10 таблиц, 11 использованных источников.

ЗАДАЧА КЛАССИФИКАЦИИ, МАШИННОЕ ОБУЧЕНИЕ, АНАЛИЗ ДАННЫХ, ТЕОРИЯ ГРАФОВ, СЛУЧАЙНЫЙ ЛЕС, МЕТОД K-БЛИЖАЙШИХ СОСЕДЕЙ, ПРОЕКТИРОВАНИЕ ПРИЗНАКОВ, МЕТРИКИ КАЧЕСТВА.

Выпускная квалификационная работа посвящена анализу и разработке модели, которая способна предсказать потенциально мошеннические транзакции. Проектируется и разрабатывается прикладная программа для визуализации графов, на основе которой предполагается выявление дополнительных признаков. Проводится анализ, проектирование и отбор признаков. Проводятся различные способы улучшения качества работы спроектированной модели.

Реализованы модели случайного леса и k-ближайших соседей. Обучение модели и преобразование данных спроектированы и реализованы в виде отдельной программы. Полученная модель имеет высокий процент верных предсказаний.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире информационные технологии проникли во все сферы деятельности человека. Объем информации, который проходит через человека за сутки, невозможно было и представить несколько десятилетий назад. Такой объем человек физически не в состоянии обработать и проанализировать. На помощь ему приходят вычислительные машины, которые стремительно развиваются, покоряя все новые вершины вычислительных мощностей.

Естественно компьютеризация коснулась и финансовой сферы деятельности человечества. Во всем мире активно используются цифровые платежные системы, а бумажные деньги уходят на второй план. У каждого банка есть свое мобильное приложение, через которое можно совершить перевод средств кому угодно, если имеются реквизиты. В некоторых странах, например, в Швеции, доля финансовых операций, совершаемых через электронные системы, от всех финансовых операций, происходящих в стране, достигает 99% по информации статьи «*Will Cash Disappear*?» газеты The New York Times от 14 ноября 2017 года.

Однако, хоть деньги и теряют свой физический облик, люди, желающие нечестным способом их заполучить, никуда не делись. Находятся мошенники, которые ухищряются украсть и на цифровом пространстве. Но как же уследить и отловить мошенников, установить, какие транзакции совершаются злоумышленниками в этом многомилионном потоке информации?

Для анализа больших объемов данных часто прибегают к такому разделу искусственного интеллекта как машинное обучение. С ростом мощностей современных компьютеров, применение алгоритмов машинного обучения становится более реальным. На многие вычисления, которым раньше требовалось большое количество времени, сейчас уходят несколько минут или даже секунд.

Целью данной выпускной квалификационной работы является построение модели, способной с помощью алгоритмов машинного обучения

определять подозрительные финансовые транзакции. На текущий момент решения такой задачи имеются только у крупнейших участников на финансовом рынке.

На тестовом наборе будет происходить обучение спроектированной модели. Для проектирования модели будет произведен анализ и отбор признаков. Частью подзадачи анализа признаков является построение графов для анализа истории взаимодействия клиентов. Для этой подзадачи будет реализована специальное прикладное программное обеспечение, способное переводить историю финансовых переводов в ориентированный граф, показывающий взаимодействия между клиентами, а также визуализировать его.

ОСНОВНАЯ ЧАСТЬ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Графы

1.1.1 Основные определения

В данной работе графы будет использоваться в двух местах: для визуализации и анализа, а также при построении деревьев решений.

Определение 1.1. [4] Графом называется упорядоченная пара $G=(V(G),E(G))$, где $V(G)$- непустое множество вершин (узлов) графа G, а $E(G)$ – множество ребер графа G.

Далее будут обозначены некоторые вспомогательные определения:

- количество вершин графа называется его порядком и обозначается $v=|V(G)|$. В данной работе будут рассматриваться только конечные графы, т.е. множества вершин и ребер принимают конечное число значений;
- количество рёбер графа называется его размером и обозначается $e=|E(G)|$;
- ребро $e=\{u,v\}$ соединяет вершины, называющиеся концевыми (концами) u и v;
- соседними вершинами называются такие концы, которые соединены одним и тем же ребром;
- смежными называются ребра, имеющие общую концевую вершину;
- изолированной называют вершину в случае, если она не является конечной ни для одного из ребер;
- листом, висячей или терминальной вершиной называется вершина, которая является концом ровно одного ребра.

Если не говорится об обратном, то граф считается неориентированным (пример приведен на рис 1.1), это означает, что каждое его ребро имеет два конца, порядок которых не имеет значения.

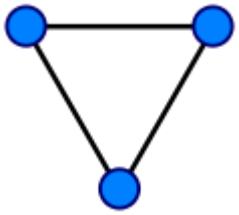

Рис. 1.1. Неориентированный граф

Пусть $(v,w)\in E$ называется дугой. Тогда вершину v называют её началом, а w — концом. Можно сказать, что дуга $v\to w$ ведёт от вершины v к вершине w.

Определение 1.2. Граф называется ориентированным (орграфом), если $G=(V(G),E(G))$ — упорядоченная пара, где $V(G)$— непустое множество вершин (узлов) графа G, а $E(G)$ — множество упорядоченных пар различных вершин, которые называются дугами. Пример такого графа приведен на рис. 1.2.

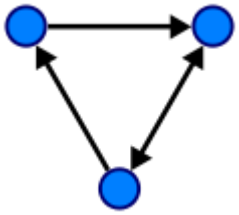

Рис. 1.2. Ориентированнный граф

Вспомогательные определения для орграфов:

- маршрутом в графе называется конечная последовательность вершин, в которой каждая вершина (исключая последнюю) соединяется со следующей в последовательности вершиной ребром;
- путем в графе называют конечную последовательность вершин и дуг, в которой каждый элемент соединен с предыдущем и последующим;
- цепью называется маршрут, в котором любое ребро встречается лишь один раз, то есть отсутствуют повторяющиеся ребра;
- циклом называют такую цепь, в которой начальная вершина также является и конечной;
- цикл называют простым в случае отсутствия повторяющихся ребер;

- цикл называют элементарным, если он простой и вершины в нем не повторяются.

Определение 1.3. Граф H называется подграфом графа G, если $V(H) \subset V(G)$ и $E(H) \subset E(G)$.

Определение 1.4. 1) Вершины a и b графа G называются связанными, если в графе существует путь между ними.

2) Граф называется связным, если любые две его вершины связаны.

Существует много несколько разновидностей графов. В данной работе будет использоваться такие структуры как дерево и лес.

Определение 1.5. 1) Деревом называется связный граф без циклов.

2) Лесом называется упорядоченное множество деревьев.

Определение 1.6. Степенью вершины называется количество инцидентных ей ребер.

Определение 1.7. Двоичным (бинарным) деревом называется:

1. неориентированное дерево, степени вершин которого не превосходят 3;
2. ориентированное дерево, в котором число исходящих из каждой вершины ребер не превосходит 2.

Вершина, из которой выходят ребра называется родительской. А связанные с ней ребрами вершины называются левой и правой дочерней.

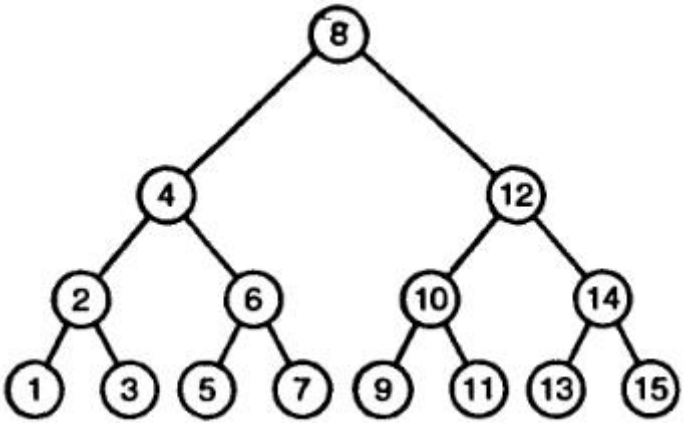

Рис. 1.3. Пример неориентированного бинарного дерева

1.1.2 Простые алгоритмы на графах

В этом разделе приводится несколько основных алгоритмов обхода графов. При обходе графа алгоритм осуществляет проход по всем вершинам через ребра. При этом накапливается довольно много информации, которая полезна для дальнейшей обработки графа.

1.1.2.1 Поиск в ширину

Поиск в ширину (breadth-first search) – один из базисных алгоритмов, составляющий основу многих других.

[6] Пусть задан граф $G = (V(G), E(G))$ и фиксирована начальная вершина s. Алгоритм поиска в ширину перебирает все доступные из s вершины, которые можно достичь при проходе по ребрам. Если последовательно перечислять пройденные вершины, получится отсортированный в порядке возрастания расстояния от s список всех вершин графа. Расстоянием называется длина минимального пути из начальной вершины до рассматриваемой. Алгоритм поиска в ширину можно применять как неориентированным графам, так и к ориентированным.

Название алгоритма обусловлено тем, что процесс обхода совершается вширь, а не вглубь. То есть первыми просматриваются все соединенные с рассматриваемой вершины, затем все соединенные с соединенными и так далее.

Алгоритм 1.1.

1. Начинается обход из фиксированной начальной вершины s. Она помечается как посещенная. Вершина добавляется в изначально пустую очередь.
2. Вершина u извлекается из начала очереди.
3. Если она является уже посещенной повторить шаг 2. Иначе все соединенные с u не посещённые вершины добавляются в очередь и осуществляется переход к шагу 2.

4. Если очередь пустая, алгоритм заканчивается. В противном случае переходит к шагу 2.

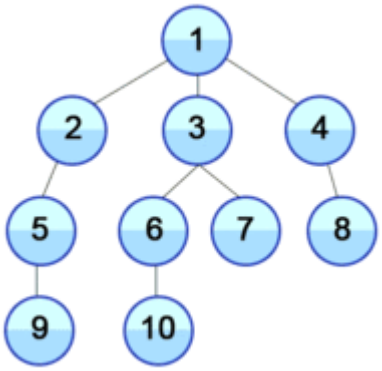

Рис. 1.4. Порядок обхода графа при поиске в ширину

На рис. 1.4. приведен пример работы алгоритма поиска в ширину. Цифры являются порядковым номером посещения вершины в ходе работы алгоритма.

1.1.2.2 Поиск в глубину

[6] Поиск в глубину (depth-first search) наряду с вышеописанным алгоритмом также является одним из базисных методов обхода графа.

Он имеет следующую стратегию: как и в поиске в ширину, фиксируется начальная вершина *s* и от нее начинается проход «вглубь», пока имеется такая возможность, т.е. пока существуют не пройдённые ребра, затем возвращается и ищет иной путь, в случае, когда таких ребер не осталось. Алгоритм работает, пока не обнаружит все вершины, достижимые из исходной.

Алгоритм 1.2.

1. Начинается обход из фиксированной начальной вершины *s*.
2. Текущая вершина помечается как посещенная.
3. Если есть соседние не посещенные вершины, осуществляется переход к одной из них и для нее выполняется алгоритм начиная с шага 2. Если все соседние вершины посещены, либо отсутствуют вовсе алгоритм заканчивается.

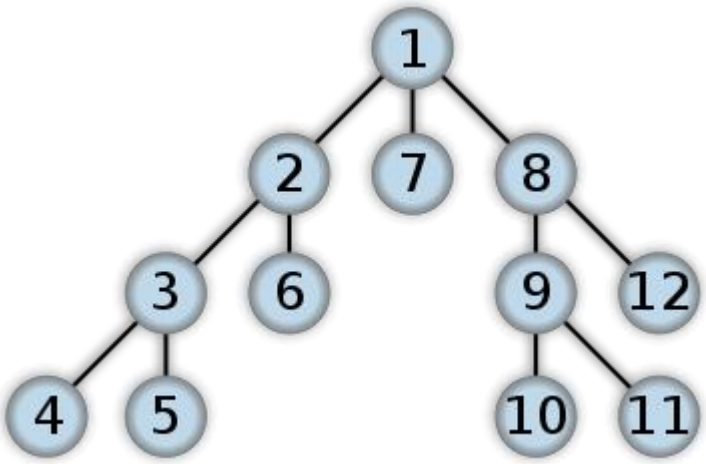

Рис. 1.5. Порядок обхода графа при поиске в глубину

На рис. 1.5. представлен пример работы алгоритма поиска в глубину. Аналогично рис. 1.4. цифры соответствуют порядковому номеру вершины при обходе графа алгоритмом 1.2.

1.2 Классификаторы

Классификацией называют задачу разделения объектов на классы по заданному набору признаков. Допустим существует конечное множество объектов, которое неким образом делится на классы.

В машинном обучении существует такой тип задач как обучение с учителем. Это означает, что исследователю известна выборка с уже заданными значениями целевой переменной. Как раз к нему и относится задача классификации. В этом случае целевой переменной является один из элементов множества с дискретным количеством значений. Необходимо построить алгоритм, классифицирующий произвольный объект из исходного множества.

Классифицировать объект — значит, указать номер (или наименование) класса, к которому относится данный объект.

Классификация объекта — номер или наименование класса, выдаваемый алгоритмом классификации в результате его применения к конкретному объекту.

Существует несколько типов классификации:

- двухклассовая, число классов равно двум;
- многоклассовая, когда число классов достигает многих тысяч (при распознавании иероглифов или слитной речи).

А также существует несколько типов классов:

- непересекающиеся классы;
- пересекающиеся классы. Объект может относиться одновременно к нескольким классам;
- нечёткие классы. Требуется определять степень принадлежности объекта каждому из классов, обычно это действительное число от 0 до 1.

В данной работе будет рассматриваться случай двухклассовой классификации для непересекающихся классов.

Введем некоторые базовые определения, которые будут использованы в дальнейшем.

Определение 1.8. [11] Признаком (feature) называется результат измерения некоторой характеристики объекта. Можно сказать, что признак есть отображение $f : X \to D_f$, где D_f – множество допустимых значений признака.

В зависимости от природы этого множества, признаки делятся на нижеперечисленные типы:

- бинарный признак $D_f = \{0,1\}$;
- номинальный признак D_f — конечное множество;
- порядковый признак: D_f — конечное упорядоченное множество;
- количественный признак: $D_f = \mathbb{R}$.

Если все признаки имеют одинаковый тип, то исходные данные называются однородными, в ином случае – разнородными.

Определение 1.9 [11]. Пусть имеется набор признаков $f_1,\ldots, f_n$. Признаковым описанием объекта $x \in X$ называют вектор $((f_1(x),\ldots, f_n(x))$, составленный из значений фиксированного набора признаков на данном объекте.

В задачах машинного обучения не делается различия между объектом и его признаковым описанием. Полагается, что $X = D_{f_1} \times \ldots \times D_{f_n}$

Постановка задачи классификации выглядит следующим образом. Пусть X — множество признаков объектов. Чем является объект, определяется спецификой предметной области. Например, в задачах спортивного менеджмента объектами являются спортсмены.

Пусть Y — конечное множество номеров (имён, меток) классов. Существует неизвестная целевая зависимость — отображение $y^*: X \to Y$, значения которой известны только на объектах конечной обучающей выборки $X^m = \{(x_{1,}y_1),...,(x_{m,}y_m)\}$.

Ставится задача построить алгоритм $a(x): X \to Y$, способный классифицировать произвольный объект $x \in X$.

1.2.1 Нормализация входных данных

Зачастую, чтобы достичь адекватности работы модели, необходимо нормализовать (масштабировать) входные данные. Как будет видно далее работоспособность некоторых моделей зависит от расстояния между объектами, вследствие чего возникает необходимость проведения данной процедуры. Проблема заключается в разных измерениях признаков. Например, если рассматривать погоду, то такие ее признаки как температура, давление, скорость ветра и т.д. измеряются в различных физических величинах, а их числовые значения могут на порядки отличаться.

Нормализовать данные можно разными способами, вот два основных.

Минимаксная нормализация:

$$x_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}. \quad (1)$$

Данный метод осуществляет переход от абсолютных значений признаков к относительным. Новые переменные будут принимать значения в диапазоне от 0 до 1.

Z-нормализация (масштабирование):

$$x_i = \frac{x_i - \bar{x}}{s}, \quad (2)$$

где $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i -$ выборочное среднее, $s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$ – выборочное среднеквадратичное отклонение.

Поскольку не все признаки имеют количественные значения, может применяться создание фиктивных переменных (dummy coding). В этом случае категориальные признаки заменяются бинарными. Например, заменяется признак «пол» на два новых: «пол мужской», «пол женский» со значения 0 и 1.

1.2.2 Метод k-ближайших соседей

[1] Метрическими называют алгоритмы, основанные на анализе сходства объектов.

Метрическим классификатором (similarity-based classifier) называют алгоритм классификации, основанный на вычислении оценок сходства между объектами. Чтобы формализовать понятие сходства вводится функция расстояния между объектами $\rho(x, x')$ в пространстве объектов *X*. Следует заметить, что данная функция может не всегда удовлетворять всем аксиомам метрики. Например, довольно часто не выполняется неравенство треугольника.

Метрические классификаторы опираются на гипотезу компактности. Она, в свою очередь, предполагает, что схожие объекты гораздо чаще лежат в одном классе, чем в разных. Можно сказать, что классы образуют компактно локализованные подмножества в пространстве объектов. То есть граница между классами имеет довольно простую форму.

Метод ближайшего соседа позиционируется как один из простейших метрических классификаторов. Классифицируемый объект x относится к тому классу y_i, к которому принадлежат ближайший к нему объект обучающей выборки x_i.

Метод *k* ближайших соседей (k-nearest neighbors algorithm, k-NN) для повышения надёжности классификации относит объект к тому классу, которому принадлежит большинство из его соседей, то есть k ближайших к нему объектов обучающей выборки x_i. В задачах с двумя классами число соседей берут нечётным, чтобы не возникало ситуаций неоднозначности, когда одинаковое число соседей принадлежат разным классам.

Пусть задана обучающая выборка $X^m = \{(x_1, y_1), ..., (x_m, y_m)\}$ и на множестве объектов задана функция расстояния $\rho(x, x')$. Данная функция должна быть достаточно адекватной моделью сходства объектов, для этого можно провести

процедуру нормализации, описанную в разделе 1.2.1. Чем больше значение этой функции, тем менее схожими являются два объекта x, x'.

Для произвольного объекта u расположим объекты обучающей выборки x_i в порядке возрастания расстояний до u: $\rho(u, x_{1;u}) \le \rho(u, x_{2;u}) \le ... \rho(u, x_{m;u})$, где $x_{i;u}$ обозначает объект обучающей выборки, который является i-ым соседом объекта u. Аналогичное обозначение введём и для ответа на i-ом соседе: $y_{i;u}$. Таким образом, произвольный объект u порождает свою перенумерацию выборки. В наиболее общем виде алгоритм ближайших соседей выглядит так:

$$a(u) = \arg\max_{y \in Y} \sum_{i=1}^{m} [y(x_{i;u}) = y] w(i,u), \qquad (3)$$

где $w(i,u)$ – заданная весовая функция, которая оценивает степень важности i-го соседа для классификации объекта u. Эта функция неотрицательна и не возрастает по i.

Различно задавая весовую функцию, получаются варианты метода ближайших соседей.

- $w(i,u) = [i = 1]$ – простейший метод ближайшего соседа;
- $w(i,u) = [i \le k]$ – метод k ближайших соседей;
- $w(i,u) = [i \le k] q^i$ – метод k экспоненциально взвешенных ближайших соседей, где предполагается $q < 1$ (обычно используется в случае 3-х и более классов).

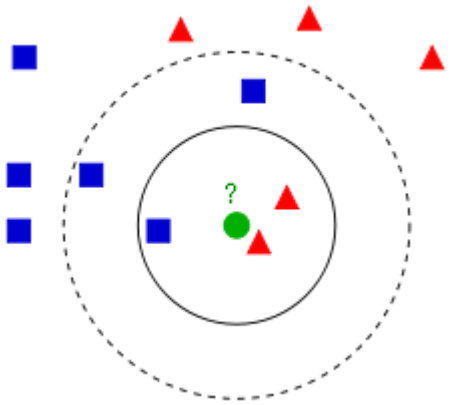

Рис. 1.6. Метод k ближайших соседей

На рис. 1.6. приведена ситуация классификации объекта, в данном случае зеленого круга. Круг должен быть классифицировать как синий квадрат, либо как красный треугольник (класс 1 и класс 2 соответственно). На иллюстрации видно две окружности.

Окружность, обведенная сплошной линией, показывает поведение алгоритма при $k=3$. В этом случае объект будет классифицирован как 2-ой класс, так как внутри окружности находятся 2 треугольника и 1 квадрат, треугольников больше, а значит и решение принимается в сторону этого класса.

В окружности, обведенной штрихом, $k=5$. Тогда ситуация меняется, потому что количество квадратов начало превалировать над треугольниками. Соответственно и объект будет классифицирован как синий квадрат, то есть 1-й класс.

1.2.3 Дерево решений

[2] Решающими деревьями называется семейство моделей, которые позволяют восстанавливать нелинейные зависимости произвольной сложности. Они воспроизводят логические схемы, позволяющие получить окончательное решение о классификации объекта с помощью ответов на иерархически организованную систему вопросов. Причем вопрос, задаваемый на последующем иерархическом уровне, зависит от ответа, полученного на предыдущем.

Каждой из вершин дерева за исключением листьев соответствует некоторый вопрос, подразумевающий несколько вариантов ответов, соответствующих выходящим ребрам. В зависимости от выбранного варианта ответа осуществляется переход к вершине следующего уровня. Листьям поставлены в соответствие метки, указывающие на отнесение распознаваемого объекта к одному из классов.

Решающее дерево называется бинарным, если каждая внутренняя или корневая вершина инцидентна только двум выходящим рёбрам.

Определение 1.10. [8] Рассмотрим бинарное дерево, в котором каждой внутренней вершине v приписана функция (или предикат) $\beta_v : X \to \{0,1\}$, а каждому листу v приписан прогноз $c_v \in Y$. В случае классификации листу может быть приписан вектор вероятностей. При классификации объекта $x \in X$ он проходит путь от корня дерева до некоторой концевой вершины, в соответствии с алгоритмом *a(x).*

Теперь рассмотрим алгоритм *a(x),* который стартует из корневой вершины v_0 и вычисляет значение функции β_{v_0}.

Алгоритм 1.3.

1. Начинается с начальной вершины v_0.
2. Вычисляется значение функции β_{v_0} для v_0.
3. Если текущая вершина является листом, возвращается класс, который приписан данной вершине и алгоритм заканчивается.

4. Если вычисленное значение равно нулю, то алгоритм переходит в левую дочернюю, в ином случае в правую. Для новой вершины выполняется шаг 2.

Такой алгоритм называется бинарным решающим деревом.

На практике в большинстве случаев используются одномерные предикаты β_v, которые сравнивают значение одного из признаков с порогом: $\beta_v(x; j,t) = [x_j < t]$.

Объект x доходит до вершины v тогда и только тогда, когда выполняется конъюнкция $K_v(x)$, составленная из всех предикатов, приписанных внутренним вершинам дерева на пути от корня v_0 до вершины v.

Пусть T — множество всех терминальных вершин дерева. Множества объектов $\Omega_v = \{x \in X : K_v(x) = 1\}$, выделяемых терминальными конъюнкциями $v \in T$, попарно не пересекаются, а их объединение совпадает со всем пространством X. Данное утверждение легко доказывается индукцией по числу вершин дерева. Отсюда следует, что решающее дерево никогда не отказывается от классификации. А также, что алгоритм классификации $a(x): X \to Y$, реализуемый бинарным решающим деревом, можно представить в виде простого голосования конъюнкций:

$$a(x) = \arg\max_{y \in Y} \sum_{v \in T} [c_v = y] K_v(x), \qquad (4)$$

причем для любого $x \in X$ одно и только одно слагаемое во всех этих суммах равно единице.

Можно увидеть, что для любой выборки можно реализовать решающее дерево, которое не допустит на ней ни одной ошибки. Даже с простыми одномерными предикатами можно сформировать дерево, в каждом листе которого находится ровно по одному объекту выборки. Вероятнее всего, такое дерево будет переобученным и не сможет показать хорошее качество классификации на новых данных.

1.2.3.1 Построение дерева

Опишем базовый жадный алгоритм построения бинарного решающего дерева.

Алгоритм 1.4

1. В качестве начальной берется вся обучающая выборка X. Находится ее наилучшее разбиение на две части $R_1(j,t)=\{x \mid x_j < t\}$ и $R_2(j,t)=\{x \mid x_j \geq t\}$ с точки зрения заранее заданного функционала качества $Q(x, j, t)$.
2. На основе найденных наилучших значения j и t создается корневая вершина дерева v_0 и принимается за текущую.
3. В текущей вершине проверяется, не выполнилось ли некоторое условие останова. Если выполнилось, то рекурсия прекращается, эта вершина объявляется листом. Осуществляется переход к шагу 6. Если же не выполнилось, то к шагу 4.
4. В соответствие текущей вершине ставится предикат $[x_j < t]$. Объекты разобьются на две части — одни попадут в левое поддерево, другие в правое.
5. Для каждой из подвыборок, получившихся на шаге 4, рекурсивно повторяется шаг 2, строятся дочерние вершины для корневой, и так далее.
6. В построенном дереве каждому листу приписывается ответ. В случае классификации – класс, к которому больше всего относится объектов в листе или вектор вероятности. Алгоритм завершен.

Выбор конкретной функции зависит от функционала качества в исходной задаче. Таким образом, конкретный метод построения решающего дерева определяется:

- видом предикатов в вершинах;
- функционалом качества $Q(x, j, t)$;

- критерием останова.

Далее подробнее рассматривается каждый пункт из вышеперечисленного списка.

[8] При построении дерева необходимо задать функционал качества, на основе которого будет осуществляться разбиение выборки на каждом шаге. Пусть R_m – множество объектов, которые попали в вершину, разбиваемую на текущем шаге, а R_l и R_r – подмножества R_m, состоящие из объектов, попадающих при заданном предикате в левое и правое поддерево соответственно. Будет использован функционал следующего вида:

$$Q(R_m, j, s) = H(R_m) - \frac{|R_l|}{|R_m|} H(R_l) - \frac{|R_r|}{|R_m|} H(R_r), \quad (5)$$

где $H(R)$ есть критерий информативности (impurity criterion), который оценивает качество распределения целевой переменной среди объектов множества R. Чем меньше разнообразие целевой переменной, тем меньше должно быть значение критерия информативности — и, соответственно, ставится задача минимизировать его значение. Функционал качества $Q(R_m, j, s)$ при этом необходимо максимизировать.

Как уже обсуждалось выше, в каждом листе дерево будет выдавать константу — вещественное число, вероятность или класс. Исходя из этого, можно предложить оценивать качество множества объектов R тем, насколько хорошо их целевые переменные предсказываются константой (при оптимальном выборе этой константы):

$$H(R) = \min_{c \in Y} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} L(y_i, c) \quad (6)$$

где $L(y_i, c)$ — некоторая функция потерь. Далее рассматривается, какие именно критерии информативности часто используют в задачах классификации.

1.2.3.2 Критерии информативности

Через p_k обозначается доля объектов класса k ($k \in \{1,...,K\}$), попавших в вершину R:

$$p_k = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i = k], \tag{7}$$

Через k_* обозначается класс, представителей которого нашлось больше остальных среди объектов, попавших в данную вершину:

$$k_* = \arg\max_k p_k . \tag{8}$$

В пример приводятся два критерия информативности.

Первым будет ошибка классификации. Индикатор ошибки рассматривается, как функция потерь:

$$H(R) = \min_{c \in Y} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i \neq c], \tag{9}$$

Заметно, что оптимальным предсказанием тут будет наиболее популярный класс k_*. Это означает, что критерий будет равен следующей доле ошибок:

$$H(R) = \min_{c \in Y} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i \neq k_*] = 1 - p_{k_*}, \tag{10}$$

Данный критерий считается довольно грубым, так как учитывает частоту p_{k_*} лишь одного класса.

Вторым будет Критерий Джини. Рассматривается ситуацию, когда в вершине выдается не один класс, а распределение по всем классам $c = (c_1, \ldots, c_n), \sum_{k=1}^{K} c_k = 1$. Качество такого распределения можно измерять с помощью критерия Бриера (Brier score):

$$H(R) = \min_{\sum_k c_k = 1} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} (c_k - [y_i = k])^2 , \tag{11}$$

Можно показать, что оптимальный вектор вероятностей состоит из долей классов p_k: $c_* = (p_1, \ldots, p_K)$.

Если подставить эти вероятности в исходный критерий информативности (6), провести ряд преобразований, то на выходе получается критерий Джини:

$$H(R = \sum_{k=1}^{K} p_k (1 - p_k). \quad (12)$$

1.2.3.3 Критерии останова

Можно придумать большое количестве критериев останова. Ниже приведены некоторые ограничения и критерии:

- ограничение максимальной глубины дерева;
- ограничение минимального числа объектов в листе;
- ограничение максимального количества листьев в дереве;
- останов в случае, если все объекты в листе относятся к одному классу;
- требование, чтобы функционал качества при дроблении улучшался как минимум на s процентов.

С помощью грамотного выбора подобных критериев и их параметров можно существенно повлиять на качество дерева.

1.2.4 Случайный лес

В основе алгоритма случайного леса лежит идея использования ансамбля (комитета) решающих деревьев. Сами по себе входящие в ансамбль решающие деревья дают невысокий результат классификации, но за счет большого их количества, решение, принимаемое голосованием, получается достаточно хорошим.

Задается обучающую выборку, количество образцов в которой равно N. Пусть размерность пространства признаков будет равна M. Параметр $m \approx \sqrt{M}$ (такое значение используется в задачах классификации) равен количеству неполных признаков, используемых для обучения.

Рассматривается наиболее распространенный способ построения отдельных экземпляров деревьев для случайного леса.

Алгоритм 1.5.

5. Генерируется случайная подвыборку с возвращением размером n. Некоторые образцы попадут в выборку более одного раза, а некоторые не попадут вовсе.
6. Формируется решающее дерево в соответствии с алгоритмом 1.4. При создании очередного узла дерева случайным образом выбирается m признаков, на основе которых будет производится разбиение. Для выбора лучшего из этих признаков используется критерий Джини.
7. Дерево строится до срабатывания выбранного критерия останова, которые были описаны в разделе 1.2.2. Чаще всего этим критерием является полное исчерпание подвыборки.

Данный алгоритм повторяется K (количество деревьев в лесу) раз. Получившийся лес классифицирует объекты путём голосования. Процесс голосования выглядит следующим образом: каждое дерево в комитете относит объект к одному из классов. Побеждает класс, которой набрал наибольшее количество голосов.

1.3 Метрики качества

После того как модель построена и дает результаты, возникает вопрос о качестве работы данной модели и правдоподобности результатов. Прежде чем говорить о подборе той или иной метрики для оценки качества классификатора, необходимо ввести понятие матрицы ошибок.

Пусть дана обучающая выборка, над ней произведена процедура бинарной классификации, в результате которой получен вектор ответов $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_n)$. Поскольку выборка обучающая, то вектор ожидаемых результатов $y = (y_1, \ldots, y_n)$ известен. Класс объектов равных 1, будем называть C_1, а объектов равных 0 – C_0.

Тогда матрица ошибок будет выглядеть следующим образом, который показан в таблице 1.1.

Таблица 1.1 Матрица ошибок

	$\hat{y}_i = 0$	$\hat{y}_i = 1$
$y_i = 0$	*TN*	*FP*
$y_i = 1$	*FN*	*TP*

Получившиеся на пересечении значения означают:

- *TN* – количество верно предсказанных классификатором объектов класса C_0.
- *TP* – количество верно предсказанных классификатором объектов класса C_1.
- *FP* – количество ошибок первого рода, которые означают, что классификатор определил класс C_0, как класс C_1.
- *FN* – количество ошибок второго рода, которые означают, что классификатор определил класс C_1, как класс C_0.

Первой очевидной метрикой, которая интуитивно приходит на ум является доля правильных ответов:

$$accuracy = \frac{TP + TN}{TP + TN + FN + FP}. \quad (13)$$

На практике такая метрика очень редко используется, потому что является бесполезной в задачах, где количество объектов одного класса сильно превалирует на другим.

Далее введем метрики для оценки работы модели на каждом из классов по отдельности. Первой будет точность (precision):

$$precision = \frac{TP}{TP + FP}, \quad (14)$$

Она показывает долю объектов, определенных классификатором как класс C_1, которые в действительности относятся к классу C_1.

Второй метрикой будет полнота (recall):

$$recall = \frac{TP}{TP + FN}, \quad (15)$$

она показывает, какую долю объектов класса C_1 из всех его объектов нашел классификатор.

Объединяет последние две метрики F_1-мера. Которая является средним гармоническим точности и полноты:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (16)$$

Очевидно, что F_1-мера будет максимальной при максимальных значениях полноты и точности, и будет близка к нулю, если один из аргументов близок к нулю.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

Практическим результатом работы является обученная модель, способная определять потенциально мошеннические транзакции. Также было спроектировано прикладное программное обеспечение, позволяющее визуализировать в виде ориентированных графов взаимодействие клиентов посредством финансовых транзакций.

2.1 Постановка задачи

Ставится задача анализа финансовых транзакций с целью предотвращения мошеннических операций. Данная задача разбивается на две подзадачи:

1. Реализовать классификатор способный отличить мошеннические транзакции от не являющихся таковыми.
2. Построить графы на основе финансовых переводов. На графах посчитать новые параметры (расстояние до ближайшего мошенника, количество мошенников в круге радиуса n), добавить их в модель и проверить значимость.

2.1.1 Описание данных

Данные, на которых будут проводится исследования, представляют собой файл с расширением *.csv*, в котором содержится информация о 6362620 финансовых транзакциях. Мошеннические транзакции в нем помечены. Следует отметить, что этот набор данных сгенерирован синтетически (по соображениям конфиденциальности) на основе реального, предоставленного африканским сервисом мобильных переводов. Временное окно данных – 1 месяц. Для отслеживания времени введена условная переменная, равная 1 часу.

В таблице 2.1 приведены признаки, которые заданы для каждой транзакции в выборке.

Таблица 2.1 Описание признаков

Название в *.csv* файле	**Смысл**	**Принимаемые значения**
step	Аналог времени (1 step = 1 час)	1–744 (30 дней)
type	тип транзакции	CASH-IN, CASH-OUT, DEBIT, PAYMENT, TRANSFER
amount	сумма перевода	Действительное число больше нуля
nameOrig	ID пользователя-отправителя	ID в формате *‘C165564’*
oldbalanceOrg	баланс отправителя до транзакции	Действительное неотрицательное число
newbalanceOrig	баланс отправителя после транзакции	Действительное неотрицательное число
nameDest	ID пользователя-получателя	ID в формате *‘C165564’* (магазин, если первая буква ID – *M)*
oldbalanceDest	баланс получателя до транзакции	Действительное неотрицательное число
newbalanceDest	баланс получателя после транзакции	Действительное неотрицательное число
isFraud	пометка о мошеннической транзакции	1 – мошенническая 0 – в противном случае

Пример строки в файле:

184,CASH_IN,90687.53,C594792269,14620285.23,14710972.76,C1485172763,166355.91,75668.37,0,0,16,1,1,0,0,0,0

2.2 Предварительный анализ

Прежде, чем перейти к построению модели, было принято решение посчитать некоторые очевидные вещи, которые лежат на поверхности.

В результате была получена информация, приведенная в таблице 2.2.

Таблица 2.2 Данные, полученные при предварительном анализе

Название	**Значение**
Всего транзакций	6362620
Мошеннических	8213
Средняя сумма перевода мошеннических транзакций	1467967.29
Средняя сумма перевода не мошеннических транзакций	178197.04
Уникальных клиентов	4777844
Количество магазинов	2151495
Уникальных магазинов	2150401
Клиенты, взаимодействующие друг с другом более 1 раза	0
Мошеннических транзакций с магазинами (из них мошеннических)	0
Количество типов CASH-IN	1399284 (0)
Количество типов CASH-OUT	2237500 (4116)
Количество типов DEBIT	41432 (0)
Количество типов PAYMENT	2151495 (0)
Количество типов TRANSFER	532909 (4097)

Анализируя полученные данные, можно сделать выводы:

- присутствует сильный дисбаланс между классом мошеннических и не мошеннических транзакций. Было принято решение сократить выборку до ~500.000 транзакций, причем количество мошеннических оставить равным исходному. Преимущество этого шага еще и в том, что теперь для обучения модели будет требоваться меньше памяти. Факт нехватки памяти сильно тормозил (иногда и вовсе не давал посчитать результаты) процесс исследования;
- средняя сумма мошеннических транзакций на порядок выше не мошеннических. Значит сумма является важным критерием для классификации в этом наборе данных;
- количество уникальных клиентов достаточно велико. Данный факт говорит о том, что взаимосвязь между клиентами весьма слабая;
- неправдоподобным выглядит отсутствие клиентов, взаимодействующих друг с другом более 1 раза. В жизненных условиях всегда найдутся клиенты, активно переводящие друг другу средства. Скорее всего этот недостаток, обусловлен синтетической природой данной выборки;
- уникальных магазинов также очень много, почти столько же, сколько и всего транзакций, производимых с их участием. Количество магазинов совпадает с количеством транзакций с типом PAYMENT. А также мошеннические операции не проводятся в рамках контактов с магазинами;
- мошеннические транзакции производятся в пределах двух типов: CASH-OUT и TRANSFER. Наряду с суммой перевода, этот факт будет играть одну из ключевых ролей при классификации.

2.3 Построение графов

Для построения и визуализации графов использовалась интегрированная среда разработки Visual Studio 2017. Прикладная программа была разработана на языке C# при помощи технологии WPF (Windows Presentation Foundation).

2.3.1 Структура программы

Необходимо перевести данные из построчного представления в *.csv* файле в объектный вид, чтобы использовать преимущества объектно-ориентированной разработки.

Для этого были реализованы следующие классы:

- *Transaction* –данный класс является объектным представлением одной строки исходного файла. В нем абсолютно те же поля, что и набор признаков в выборке. Но вместо строчного представления имени клиента в нем хранится класс *Client* как для отправителя, так и для получателя.
- *Client* – в этом классе записывается ID клиента, список его транзакций в качестве отправителя, а также в качестве получателя, пометка о том, был ли замечен этот клиент в мошеннических переводах. Также тут может содержаться дополнительная информация, например, максимальная длина цепочки, в которой замечен клиент, и расстояние до ближайшего мошенника.

При запуске программы происходит чтение данных, преобразование их к объектному виду. В результате этого формируется список клиентов, список транзакций и граф на основе словаря.

Можно упрощенно описать работу программы алгоритмом 2.1.

Алгоритм 2.1.

Перед началом инициализируется пустой список транзакций и клиентов.

1. Читается каждая строга исходного файла. Из нее создаются экземпляры классов транзакции и клиента, которые добавляются в свои списки соответственно.

2. Итеративно продвигаясь по всему списку транзакций формируется словарь, в котором ключом выступает клиент, а значением – список клиентов, с которыми он взаимодействует.
3. На сформировавшемся на шаге 2 графе, итеративно двигаясь по ключам, с помощью алгоритма 1.2 находим вспомогательные признаки (максимальная цепочка, в которой участвует клиент и расстояние до ближайшего мошенника).

2.3.2 Описание интерфейса и возможностей

После обработки данных, описанной в алгоритме 2.1 перед пользователем появляется интерфейсное окно с визуализированным графом (рис. 2.1).

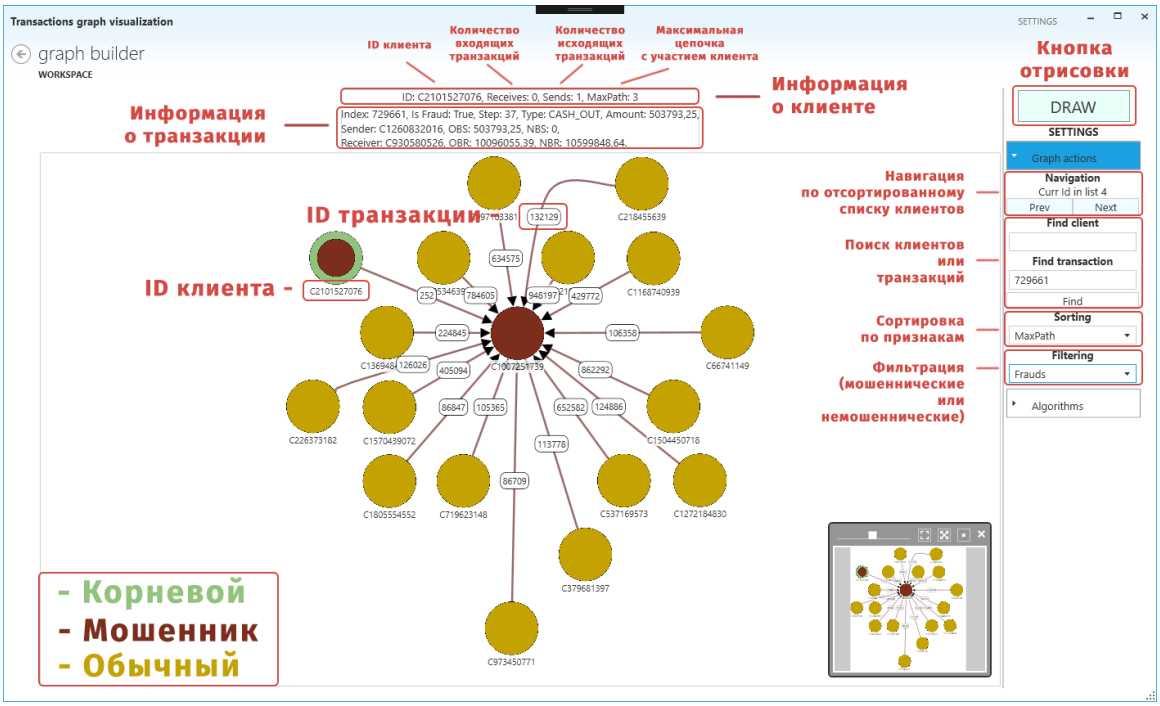

Рис. 2.1 Интерфейсное окно прикладной программы

В центральной части окна располагается поле визуализации графа. Зеленым цветом помечен клиент, с которого началось построение графа. Будем называть таких клиентов корневыми. Желтым цветом помечены обычные клиенты, не замеченные в мошеннических транзакциях. Красным помечены мошенники. Если клиент является одновременно и корневым и

мошенником, то тогда он отмечается красно-зеленым цветом. Строка под кругом – ID клиента.

Направление стрелок на визуализации указывает от отправителя к получателю. Цифры посередине стрелки означают ID транзакции, который соответствует номеру строки в исходном файле.

В верхней части окна выводится информация по выбранному клиенту и транзакции. Про клиента пишется ID, количество транзакций в роли получателя, количество транзакций в роли отправителя и максимальная цепочка, в которой он был замечен. Про транзакцию также пишется ее ID и вся информация, которая доступна из строки исходного файла, то есть все признаки.

В правой части окна располагается интерфейс взаимодействия. В нем можно изменить сортировку клиентов, отфильтровать их, найти клиента или транзакцию по ID. Все изменения вступают в силу после нажатия кнопки «DRAW».

По умолчанию клиенты отсортированы по убыванию максимальной цепочки транзакций. Можно также отсортировать по числу выступлений получателем или отправителем, а также по расстоянию до ближайшего мошенника.

Фильтрация по умолчанию отключена. Но можно отфильтровать клиентов, оставив только мошенников.

2.3.3 Анализ результатов

Построив граф на отфильтрованных данных (как было предложено в разделе 2.2). Оказалось, что длина максимальной цепи равна четырем. Предположения о разреженности графа из раздела 2.2 подтвердились. На раннем этапе исследования на полном наборе данных максимальная длина цепи была равна 9.

Соответственно предложенные для добавления в модель признаки из постановки задачи (раздел 2.1.1) не имеют смысла. Дистанция до ближайших мошенников почти у всех клиентов отсутствует, либо равна 1 или 0 (случай, когда сам клиент мошенник). Максимальной зафиксированной дистанцией была цепь длиной в 2 транзакции. Поиск количества мошенников в круге радиуса *n*, также не принесет особых результатов, поскольку общая картина такова: граф разбит на кучу маленьких подграфов. Один из таких представлен на рис 2.2.

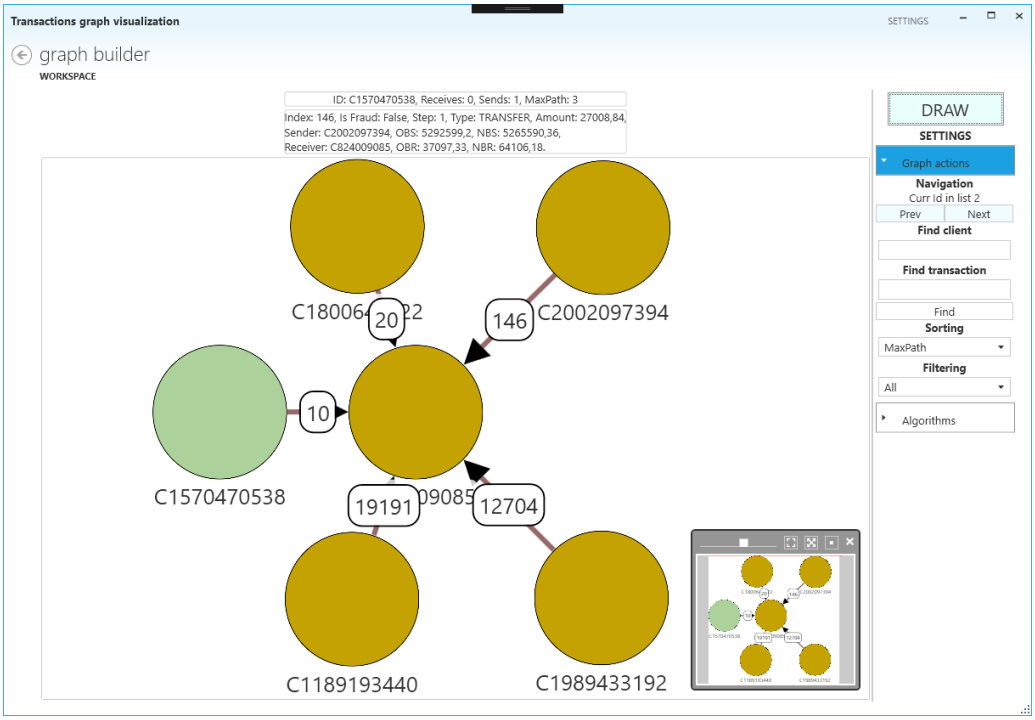

Рис. 2.2 Типичная ситуация для отдельно взятого клиента

Визуально анализируя полученные результаты, удалось установить, что типичная картина мошеннических клиентов выглядит следующим образом: клиент выступает получателем, к которому стекаются несколько транзакций, одна из которых мошенническая (рис. 2.3).

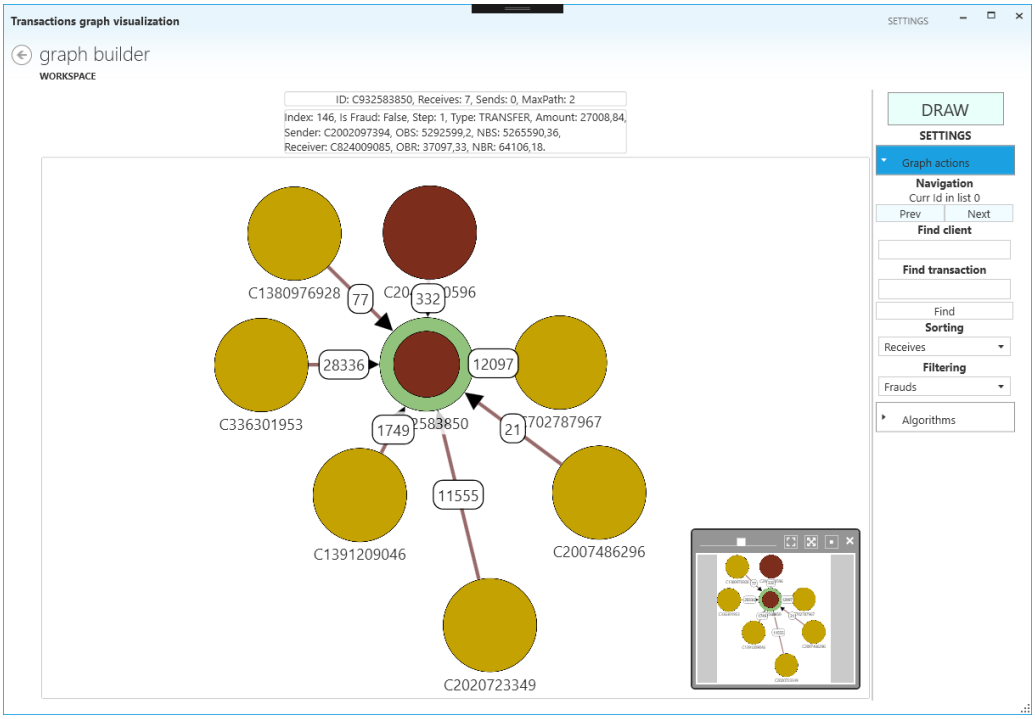

Рис. 2.3 Типичная ситуация для мошенника

Признаки, выявленные на графах, показали свою практическую неприменимость. Поэтому следует уделить большее внимание задаче классификации.

2.4 Проектирование признаков

Проектирование признаков (feature engineering) производится с целью повысить эффективность прогнозирования алгоритмов обучения путем выявления новых характеристик или исключения имеющихся из исходных данных. Данная процедура позволяет упростить процесс обучения.

Для сравнения в список используемых моделей (помимо описанных в разделах 1.2.2 и 1.2.4) добавляется наивный байесовский классификатор. Он основан на применении теоремы Байеса [5] со строгими предположениями о том, что все объекты описываются независимыми признаками.

Проведение экспериментов происходит при помощи языка Python и библиотеки scikit-learn, внутри которой реализация моделей, а также методы расчета метрик, описанных в разделе 1.3.

Отправной точкой будет результат обучения модели на «сырых» данных, то есть никаких добавлений или исключений признаков. Результаты такого эксперимента приведены в таблице 2.3.

Таблица 2.3 Метрики качества на «сырых» данных

	precision	*recall*	F_1
Наивный байесовский	0.595	0.204	0.304
k-ближайших соседей	0.954	0.794	0.867
Случайный лес	0.994	0.831	0.905

Лучше всего себя повел случайный лес и уже показал неплохие результаты. Недавно добавленный наивный байесовский классификатор довольно плохо предсказал результат. А метод ближайших соседей повел себя уверенно, но все же уступает случайному лесу.

Далее производится отбор признаков, затем эксперимент будет еще раз проведен на новом наборе данных.

2.4.1 Отбор признаков

Для устранения сильно коррелирующих между друг другом признаков строится матрица корреляции (рис. 2.4).

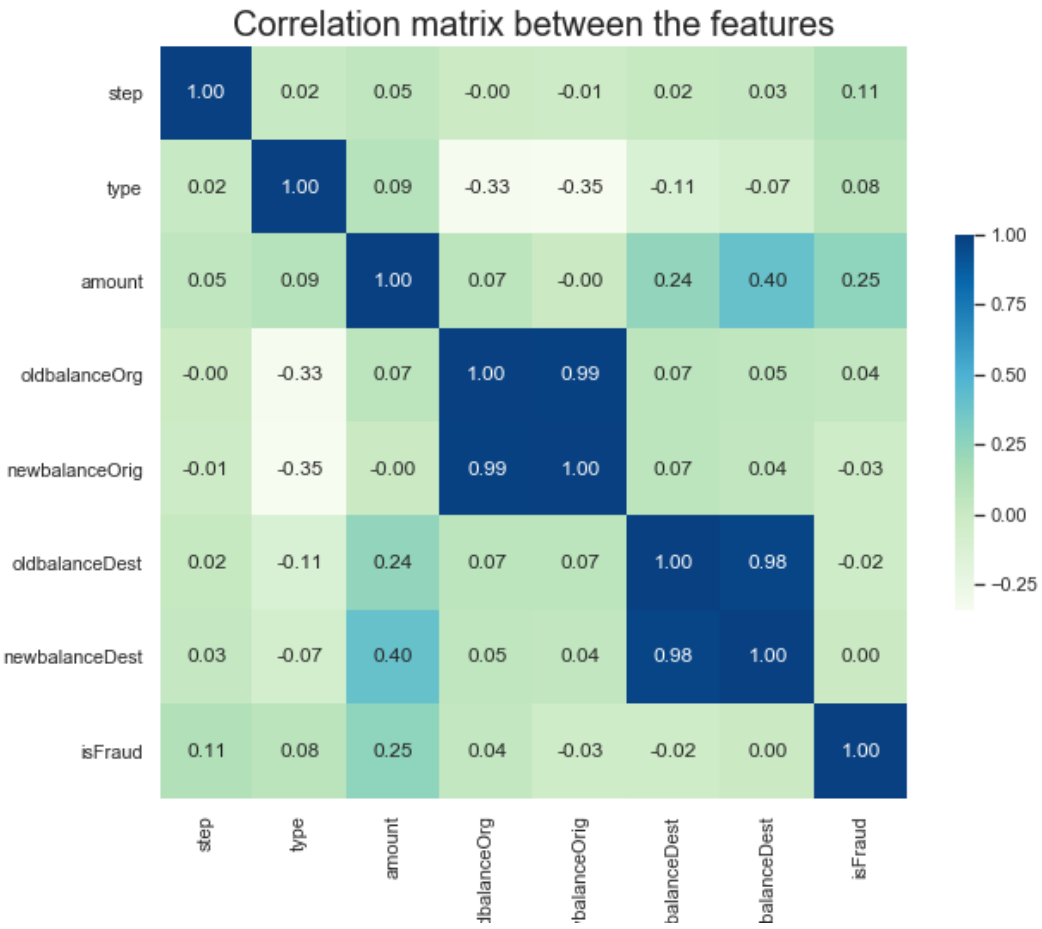

Рис. 2.4 Матрица корреляции исходных признаков

Очевидно, что балансы до и после совершения транзакции будут сильно коррелировать, поэтому можно отбросить балансы после транзакции. Информация по старому балансу получателя тоже практически ни на что не влияет, поэтому отбросим и этот признак.

Далее проектируются новые признаки на основе имеющихся. Создается новый *.csv* файл. Ниже перечисляются добавленные признаки (в скобках указано имя в новом файле):

- время (*hour)*. Условный шаг из исходных данных переведен в 24-чаовой формат. Мошеннические транзакции могут совершаются в определенное время суток;

- новый ли отправитель (*newSender).* Пометка, показывающая выступал ли ранее текущий отправитель в такой же роли;
- новый ли получатель (*newReceiver).* Пометка, показывающая выступал ли ранее текущий получатель в такой же роли;
- является ли получатель магазином (*merchant).* Пометка, показывающая является ли принимающая сторона магазином. Поскольку с магазином не совершаются злоумышленные транзакции, этот пункт может быть полезным;
- был ли хоть один из клиентов, участвующих в текущей транзакции, ранее замечен в нарушениях *(fraudsEarly).* Пометка может быть полезна, если клиент совершает кражи регулярно;
- время, прошедшее с момента последней транзакции в качестве отправителя (*LTS).* Количество часов указывается количество часов, либо ставится -1, если первое появление.
- время, прошедшее с момента последней транзакции в качестве получателя (*LTR).* Аналогично пункту, написанному выше;
- остается ли ноль на балансе отправителя (*IZoB).* Если клиент остается с 0 на балансе, то скорее всего его обчистил мошенник.

Далее с помощью встроенного в библиотеку scikit-learn метода RFE (recursive feature elimination) производится ранжирование признаков по значимости. Название метода переводится как «рекурсивное отсечение признаков». Алгоритм его работы следующий: модель обучается на исходном наборе признаков, оценивает их значимость, отсекает один или несколько наименее значимых, обучается на новых признаках и повторяет эти действия, пока не останется *n* наиболее значимых признаков.

Также у класса случайного леса из библиотеки scikit-learn есть метод, возвращающий значимость каждого признака.

Результаты работы двух этих методов приведены в таблице 2.4.

Таблица 2.4 Результаты работы алгоритма RFE и значимость признаков

	RFE	Значимость признака
step	1	0.085
type	2	0.062
amount	1	0.243
oldbalanceOrg	1	0.389
hour	1	0.110
newSender	7	7.973e-06
newReceiver	4	0.010
merchant	5	0.009
fraudsEarly	6	0.0005
LTS	8	2.28e-08
LTR	3	0.016
IZoB	1	0.071

Результаты удивляют. Самым значимым признаком оказался баланс до транзакции на счету отправителя. Не совсем понятно, с чем связана такая зависимость. Возможно, так получилось из-за синтетической природы исходной выборки. Второй по значимости признак – ожидаемо, сумма. Из новых добавленных признаков наиболее значимыми оказались время в часах, индикатор нуля на балансе, время с последней транзакции. Остальные признаки не проявили себя, их можно отбросить. Также можно отбросить шаг, у 24-часового аналога значимость больше.

Отбросив менее значимые параметры, проводится эксперимент и сравниваются результаты с предыдущей итерацией. Далее отбрасываем еще мало значимые признаки, пока не достигнем максимального результата.

2.5 Результаты

Таким образом, остались следующие признаки: *type, amount, oldBalanceOrg, hour, iZoB.* В процессе анализа, они проявили себя, как представляющие наиболее высокую значимость для обучения модели. Остальные признаки были отброшены по вследствие своих низких показателей.

Результат модели на конечных данных, а также его сравнение с результатом на исходных данных приведены в таблице 2.5.

Таблица 2.5 Результат работы модели на конечных данных

		Наивный байесовский	Случайный лес	k-ближайших соседей
precision	Исходные	0.595	0.994	0.954
	Конечные	0.547	0.985	0.955
	Разница	-0.048	-0.009	0.001
recall	Исходные	0.204	0.831	0.794
	Конечные	0.189	0.915	0.959
	Разница	-0.015	0.084	0.165
F_1	Исходные	0.304	0.905	0.867
	Конечные	0.281	0.949	0.957
	Разница	-0.023	0.044	0.90

Лучше всего себя проявил метод k-ближайших соседей. По сравнению с результатами на исходных данных прогресс явно заметен. Случайный лес тоже выдал хорошее качество предсказаний. Наивный байесовский классификатор ухудшил свои результаты.

Теперь произведем процедуру нормализации данных, описанную в разделе 1.2.1 и проверим ее влияние на работу модели. Результаты работы после совершения нормализации входных данных представлены в таблице 2.6.

Таблица 2.6 Результаты работы модели после процедуры нормализации

	Наивный байесовский	Случайный лес	k-ближайших соседей
precision	0.1953	1.0	0.9976
recall	0.0057	0.9947	0.9747
F_1	0.0111	0.9973	0.9860

Легко видеть, что нормализация пошла на пользу случайному лесу и методу k-ближайших соседей, они показывают очень хорошие результаты. Даже можно сказать, что слишком хорошие, близкие к идеальным. Это выглядит подозрительным, на реальной практике такие ситуации редко встречаются, например, чтобы метрика *precision* была равна 1. Такие обстоятельства можно объяснить, опять же, синтетической природой выборки. Каждая транзакция того или иного класса создается на себе подобной созданной ранее, что и порождает некие зависимости, которые уловил метод случайного леса.

Таким образом, можно сделать вывод, что метод ансамблей решающих деревьев лучше всего подходит для данной задачи. Наравне с ним можно применять метрический метод ближайших соседей. А вот метод, основанный на теореме Байеса, проявил себя хуже всего, его точно не стоит использовать для решения задач такого типа.

Для более наглядного сравнения результатов приводятся матрицы ошибок для всех трех методов в таблицах 2.7, 2.8 и 2.9 соответственно. Стоит отметить, что перед тем как тестировать модель исходная отфильтрованная выборка бьется на две части: обучающую и тестовую. Обучающая составляет 70% исходной, а тестовая соответственно – 30%. По итогу получилось, что обученной модели необходимо классифицировать 115 345 транзакций из тестовой выборки.

Таблица 2.7 Матрица ошибок наивного байесовского классификатора

		Предсказанные значения	
		Не мошенник	Мошенник
Ожидаемые значения	Не мошенник	110884	103
	Мошенник	4333	25

Таблица 2.8 Матрица ошибок случайного леса

		Предсказанные значения	
		Не мошенник	Мошенник
Ожидаемые значения	Не мошенник	110987	0
	Мошенник	24	4334

Таблица 2.9 Матрица ошибок метода k-ближайших соседей

		Предсказанные значения	
		Не мошенник	Мошенник
Ожидаемые значения	Не мошенник	110977	10
	Мошенник	110	4248

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе выполнены все поставленные задачи:

- проведен анализ финансовых транзакций с целью предотвращения мошеннических операций;
- реализована прикладная программа, визуализирующая графы на основе финансовых транзакций;
- реализована модель, способная классифицировать финансовые транзакции;
- проведена процедура проектирования признаков, которая привела к улучшению результатов работы модели;
- проведен анализ эффективности реализованной модели и ее сравнение с другими методами.

Таким образом, потребность в минимизации случаев совершения цифровых краж в финансовой сфере делает настоящую выпускную квалификационную работу актуальной. Предложенные в ней методы помогают выявить нетипичное поведение клиентов платежных систем.

Полученная модель имеет практическую значимость. Она может быть использована в прикладных задачах, например, в классификаторах финансовых переводов, работающих в режиме online. В таких системах модель прямо в момент перевода может не дать ему свершится, если заподозрит нетипичное поведение участников сделки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

4. Айвазян С. А., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. Прикладная статистика: классификация и снижение размерности. — М.: Финансы и статистика, 1989 – 607 с.
5. Воронцов К.В. Лекции по логическим алгоритмам классификации. – Москва, 2010. – 54 с.
6. Загоруйко Н.Г. Прикладные методы анализа данных и знаний. - Новосибирск: ИМ СО РАН, 1999, стр. 270
7. Карпов Д.В. Теория графов – Спб.: 2017, – 420 с.
8. Кибзун А.И., Горяинова Е.Р., Наумов А.В., Сиротин А.Н. Теория вероятностей и математическая статистика. Базовый курс с примерами и задачами, М.: ФИЗМАТЛИТ, 2002. - 224 с.
9. Кормен, Томас Х. и др. Алгоритмы. Построение и анализ, 3-е изд. : Пер с англ. – М. : ООО “И.Д. Вильямс”, 2013 – 1328 с.
10. Ф.Харари ТЕОРИЯ ГРАФОВ М.: Мир, 1973, 300 стр.
11. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. – Springer, 2001. – P 764.
12. Raschka S. Python Machine Learning, 2016 – 429 p.
13. Документация библиотеки scikit-learn [Электронный ресурс]. Электронный адрес: https://scikit-learn.org/stable/documentation.html
14. Machinelearning.ru [Электронный ресурс]. Электронный адрес: *http://www.machinelearning.ru/wiki/*

ПРИЛОЖЕНИЯ

Приложение 1

Основная часть программного кода на языке C#

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows;

namespace DiplomaHelp
{
    class GraphDataManager
    {
        public HashSet<string> IsVisited = new HashSet<string>();

        public List<string> StringsFile = new List<string>();
        public List<Transaction> TransactionsList = new List<Transaction>();
        public Dictionary<string, int> StringTrDict = new Dictionary<string, int>();

        public List<string> ClientsIdList = new List<string>();
        public Dictionary<string, Client> ClientStringClassDict = new Dictionary<string,
Client>();

        public Dictionary<Client, List<Client>> ClientsGraph = new Dictionary<Client,
List<Client>>();

        public Dictionary<string, int> MerchantsDict = new Dictionary<string, int>();
        public List<Client> CurrWorkingList = new List<Client>();
        public int CurrentRootClientInd = 0;
        public bool IsSortingChanged = false;
        public bool IsFilterChanged = false;

        public GraphDataManager(string diplomaDataCsv)
        {
            StringsFile = File.ReadAllLines(diplomaDataCsv).ToList();

            StringsFile.RemoveAt(0); // удаляется строка с описанием столбцов

            //var strCount = StringsFile.Count;
            //var strCount = StringsFile.Count / 2;
            var strCount = 100000;

            for (int i = 0; i < strCount; i++)
            {
                var el = StringsFile[i];

                TransactionsList.Add(new Transaction(el));
                StringTrDict.Add(TransactionsList[i].NameOrig +
TransactionsList[i].NameDest, i);

                var sender = TransactionsList[i].NameOrig;
                var receiver = TransactionsList[i].NameDest;

                if (receiver.Contains("M"))
                {
                    if (MerchantsDict.ContainsKey(receiver))
                        MerchantsDict[receiver]++;
                    else
                        MerchantsDict.Add(receiver, 0);

                    continue;
```

```
                }

                ClientsIdList.Add(sender);
                ClientsIdList.Add(receiver);
            }

            ClientsIdList = ClientsIdList.Distinct().ToList();

            foreach (var el in ClientsIdList)
            {
                var tmpClient = new Client(el);

                ClientsGraph.Add(tmpClient, new List<Client>());

                ClientStringClassDict[el] = tmpClient;
            }

            for (var i = 0; i < strCount; i++)
            {
                var tr = TransactionsList[i];

                if (tr.NameDest.Contains("M"))
                    continue;

                var sender = ClientStringClassDict[tr.NameOrig];
                var receiver = ClientStringClassDict[tr.NameDest];

                if (tr.isFraud)
                {
                    sender.IsFraud = true;
                    receiver.IsFraud = true;
                }

                sender.AddToSender(tr);
                receiver.AddToReceiver(tr);

                ClientsGraph[sender].Add(receiver);
                ClientsGraph[receiver].Add(sender);
            }

            foreach (var pair in ClientsGraph)
            {
                pair.Key.MaxPath = DFS(pair.Key, pair.Value, ClientsGraph,
pair.Key.MaxPath);
            }

            FilterByCriteria(FilterTypes.All);
            SortByCriteria(SortTypes.MaxPath);

            MessageBox.Show("Data downloading completed");
        }

        public void SortByCriteria(SortTypes criteria)
        {
            if (criteria == SortTypes.MaxPath)
                CurrWorkingList = CurrWorkingList.OrderBy(c => -c.MaxPath).ToList();

            if (criteria == SortTypes.Receives)
                CurrWorkingList = CurrWorkingList.OrderBy(c => -
c.Receiver.Count).ToList();

            if (criteria == SortTypes.Sends)
                CurrWorkingList = CurrWorkingList.OrderBy(c => -c.Sender.Count).ToList();
```

```
            var tmpstr = new List<string>();

            foreach (var el in CurrWorkingList)
            {
                tmpstr.Add(el.ToString());
            }

            File.WriteAllLines("path.txt", tmpstr);
        }

        public void FilterByCriteria(FilterTypes criteria)
        {
            if (criteria == FilterTypes.All)
                CurrWorkingList = ClientsGraph.Keys.ToList();

            if (criteria == FilterTypes.Frauds)
                CurrWorkingList = CurrWorkingList.FindAll(el => el.IsFraud);
        }

        private int DFS(Client client, List<Client> connectClients, Dictionary<Client,
List<Client>> clientsGraph, int path)
        {
            var currId = client.Id;

            if (IsVisited.Contains(currId))
                return path;

            if (connectClients == null || connectClients.Count == 0)
                return path + 1;

            IsVisited.Add(currId);
            ++path;

            var lenghtList = new List<int>();

            foreach (var cl in connectClients)
            {
                cl.MaxPath = DFS(cl, clientsGraph[cl], clientsGraph, cl.MaxPath);
                lenghtList.Add(cl.MaxPath);
            }

            return path + lenghtList.Max();
        }
    }
}
```

Приложение 2

Основная часть программного кода на языке Python

```
from pandas import read_csv as read
import pandas as pd
import random

import numpy as np
import seaborn as sns; sns.set(color_codes=True)
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn.metrics import make_scorer, accuracy_score, precision_score,
recall_score, f1_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score


from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split as train

from sklearn.feature_selection import RFE

from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier


from plot import pltPRcurve, pltROCcurve
# # Используемые модели
# In[32]
models = []
models.append(RandomForestClassifier(n_estimators=110, n_jobs=-1))
models.append(KNeighborsClassifier(n_neighbors=3))
models.append(GaussianNB())
#models.append(GradientBoostingClassifier(max_depth=4))
#models.append(DecisionTreeClassifier())

# # Вспомогательные функции
# In[3]:
def plot_corr():
    correlation_matrix = data.corr()
    plt.figure(figsize=(10,8))
    ax = sns.heatmap(correlation_matrix, vmax=1, square=True,
annot=True,fmt='.2f', cmap ='GnBu', cbar_kws={"shrink": .5}, robust=True)
    plt.title('Correlation matrix between the features', fontsize=20)
    plt.savefig("cor.png")
    plt.show()
```

```
# In[4]:

def modExec():
    model.fit(X_train, y_train)
    expected = y_test
    predicted = model.predict(X_test)

    pobas = model.predict_proba(X_test)
    preds = pobas[:, 1]

    report = metrics.classification_report(expected, predicted)
    conf_matrix = metrics.confusion_matrix(expected, predicted)

    print(type(model).__name__)
    print(f"Acc:\t{accuracy_score(expected, predicted)}")
    print(f"Pre:\t{precision_score(expected, predicted)}")
    print(f"Re:\t{recall_score(expected, predicted)}")
    print(f"F1:\t{f1_score(expected, predicted)}")

    print(conf_matrix)

# In[50]:
def modExec1(models, drawPR=False, drawRoc=False):
    for model in models:

        model.fit(X_train, y_train)
        expected = y_test
        predicted = model.predict(X_test)

        pobas = model.predict_proba(X_test)
        preds = pobas[:, 1]

        name = type(model).__name__

        if drawPR:
           pltPRcurve(expected, predicted, preds, name)

        if drawRoc:
           pltROCcurve(expected, predicted, preds, name)

        report = metrics.classification_report(expected, predicted)
        conf_matrix = metrics.confusion_matrix(expected, predicted)

        print(type(model).__name__)
        print(f"Acc: {accuracy_score(expected, predicted)}")
        print(f"Pre: {precision_score(expected, predicted)}")
        print(f"Re: {recall_score(expected, predicted)}")
        print(f"F1: {f1_score(expected, predicted)}")
```

```
        print(conf_matrix)
        print()
# # Анализ на сырых данных
# In[6]:
path = "filtData500k.csv"
data = read(path, delimiter=",")

data.drop('isFlaggedFraud', axis=1, inplace=True)
data.drop('nameOrig', axis=1, inplace=True)
data.drop('nameDest', axis=1, inplace=True)

le = LabelEncoder()
data['type'] = le.fit_transform(data['type'])
# In[8]:
X = data.loc[:, data.columns != 'isFraud'].values
y = data.loc[:, 'isFraud'].values

y=y.astype('int') # не совсем понял, почему
https://stackoverflow.com/questions/45346550/valueerror-unknown-label-type-
unknown

X_train, X_test, y_train, y_test = train(X, y, shuffle=False)
# In[9]:
modExec1(models)
# In[10]:
# In[12]:
data.drop('newbalanceOrig', axis=1, inplace=True)
data.drop('newbalanceDest', axis=1, inplace=True)
data.drop('oldbalanceDest', axis=1, inplace=True)
plot_corr()
# # Проектирование признаков

# In[44]:
path = "newFilt500k.csv"
data = read(path, delimiter=",")
data.drop('isFlaggedFraud', axis=1, inplace=True)
data.drop('nameOrig', axis=1, inplace=True)
data.drop('nameDest', axis=1, inplace=True)
data.drop('newbalanceOrig', axis=1, inplace=True)
data.drop('newbalanceDest', axis=1, inplace=True)
data.drop('oldbalanceDest', axis=1, inplace=True)
# In[45]:
le = LabelEncoder()
data['type'] = le.fit_transform(data['type'])

X = data.loc[:, data.columns != 'isFraud'].values
y = data.loc[:, 'isFraud'].values
```

```
y=y.astype('int') # не совсем понял, почему
https://stackoverflow.com/questions/45346550/valueerror-unknown-label-type-
unknown
X_train, X_test, y_train, y_test = train(X, y, shuffle=False)
# ### Отбор признаков рекурсивным исключением
# In[47]:
# feature extraction

model = RandomForestClassifier(n_estimators=110, n_jobs=-1)

rfe = RFE(model, 5)

fit = rfe.fit(X_train, y_train)

print(fit.support_)
print(fit.ranking_)
# In[48]:
model.fit(X_train, y_train)
print(model.feature_importances_)

# In[19]:

data.drop('newSender', axis=1, inplace=True)
data.drop('fraudsEarly', axis=1, inplace=True)
data.drop('LTS', axis=1, inplace=True)
data.drop('newReceiver', axis=1, inplace=True)
data.head()

# In[20]:

data.drop('merchant', axis=1, inplace=True)
data.drop('LTR', axis=1, inplace=True)

data.head()
# In[21]:
data.drop('step', axis=1, inplace=True)
data.head()
# In[22]:
data.drop('hour', axis=1, inplace=True)
data.head()

# # Нормализация
# In[39]:
path = "newFilt500k.csv"
data = read(path, delimiter=",")

data.drop('isFlaggedFraud', axis=1, inplace=True)
data.drop('nameOrig', axis=1, inplace=True)
data.drop('nameDest', axis=1, inplace=True)
data.drop('newbalanceOrig', axis=1, inplace=True)
```

```
data.drop('newbalanceDest', axis=1, inplace=True)
data.drop('newSender', axis=1, inplace=True)
data.drop('fraudsEarly', axis=1, inplace=True)
data.drop('LTS', axis=1, inplace=True)
data.drop('newReceiver', axis=1, inplace=True)
data.drop('step', axis=1, inplace=True)
data.drop('merchant', axis=1, inplace=True)
data.drop('LTR', axis=1, inplace=True)
data.drop('oldbalanceDest', axis=1, inplace=True)

le = LabelEncoder()
data['type'] = le.fit_transform(data['type'])

X = data.loc[:, data.columns != 'isFraud'].values
y = data.loc[:, 'isFraud'].values

y=y.astype('int') # не совсем понял, почему
https://stackoverflow.com/questions/45346550/valueerror-unknown-label-type-
unknown
# In[25]:
X_train, X_test, y_train, y_test = train(X, y, shuffle=False)
modExec1(models)
# In[41]:
from sklearn import preprocessing

normalized_X = preprocessing.normalize(X)
standardized_X = preprocessing.scale(X)
# In[51]:
X = normalized_X
X_train, X_test, y_train, y_test = train(X, y, shuffle=False)

modExec1(models, True, False)

# In[28]:
X = standardized_X
X_train, X_test, y_train, y_test = train(X, y, shuffle=False)

modExec1(models)
```