|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное автономное образовательное  учреждение высшего образования |
| **НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»** |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА**  Направление подготовки 10.05.03  «Информационно-аналитические системы безопасности» |



**Пояснительная записка к учебно-исследовательской**

**работе на тему:**

«Исследование возможности выявления подозрительных транзакций в блокчейне с помощью методов машинного обучения»

Студент гр. C15-501…………………………………………/………………………./

Научный руководитель …………….………………………/………………………./

Москва, 2019

РЕФЕРАТ

Отчет Х с., Y рис., Z источников.

БЛОКЧЕЙН, КРИПТОВАЛЮТА, BITCOIN, API, PYTHON, ТРАНЗАКЦИИ, ПОД/ФТ, МАШИННОЕ ОБУЧЕНИЕ, КЛАСТЕРИЗАЦИЯ

Объекты исследования: блокчейн, язык программирования Python, JSON API.

Цель работы: на языке программирования Python разработать программный код, осуществляющий выгрузку данных о транзакциях на криптовалюте Bitcoin и провести кластеризацию полученных транзакций с целью выявления закономерностей, которые могли бы свидетельствовать о необходимости отнесения определенных транзакций к числу подозрительных.

Были изучены основы блокчейн-технологиий, язык программирования Python и его возможности в разрезе взаимодействия со сторонними JSON API. Были получены навыки реализации различных алгоритмов кластеризации на языке Python, а также навыки интерпретации результатов проведенной кластеризации.

В результате работы был разработан программный код, осуществляющий асинхронную выгрузку данных о транзакциях на криптовалюте Bitcoin из стороннего JSON API, код, формирующий матрицу признаков из полученных сырых данных, а также код, реализующий кластеризацию с помощью алгоритма K-Means.

Изучение криптовалют и блокчейн-технологий способствует грамотному и целостному пониманию особенностей функционирования цифровой экономики. Знания в данной области могут помочь сформулировать основные законодательные положения, которые могли бы урегулировать вопросы применения криптовалют на территории РФ, а также помочь в области разработки своевременных мер по минимизации рисков в сфере ПОД/ФТ.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 4

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 5

ВВЕДЕНИЕ 6

ОСНОВНАЯ ЧАСТЬ 8

1 Изучение принципов функционирования блокчейна 8

2 Формулирование гипотезы 9

3 Работа с API блокчейна 10

3.1 Изучение API 10

3.2 Описание подхода к получению данных 13

3.3 Описание кода для получения данных 13

3.3.1 Библиотеки 13

3.3.2 Вспомогательная функция fetch 14

3.3.3 Функция main 14

3.3.4 Режим get\_info\_by\_transaction\_list 14

3.3.5 Режим get\_info\_by\_address\_list 15

3.3.6 Функция create\_row 16

3.3.7 Описание итогового JSON 17

4 Подходы к проведению кластеризации 19

4.1 Кластеризация транзакций 19

4.2 Кластеризация адресов 19

4.2.1 Формирование матрицы «адрес-признак» для проведения кластеризации 19

5 Применение методов снижения размерности. Проведение кластеризации 24

5.1K-means 24

5.2 TSNE и PCA 24

5.2 Проведение кластеризации и анализ результатов 24

ЗАКЛЮЧЕНИЕ 27

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29

ПРИЛОЖЕНИЕ А 30

ПРИЛОЖЕНИЕ Б 41

ПРИЛОЖЕНИЕ В 53

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете о НИР применяют следующие сокращения и обозначения.

ПОД/ФТ – противодействие отмыванию денег и финансированию терроризма.

API – Application Programming Interface (программный интерфейс приложения)

JSON – JavaScript Object Notation (текстовый формат обмена данными, основанный на JavaScript)

ID – идентификатор

# ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящем отчете о НИР применяют следующие термины и определения.

База данных – это организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации, представленная в виде совокупности строк и столбцов.

Блокчейн – распределенная база данных, где каждая следующая запись регистрируется следом за всеми предыдущими.

Криптовалюта – разновидность цифровых денег, оборот которых основан на различных криптографических методах.

Машинное обучение – обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных к обучению на конкретных данных.

Кластеризация – задача группировки множества объектов на подмножества (кластеры) таким образом, чтобы объекты из одного кластера были более похожи друг на друга, чем на объекты из других кластеров по какому-либо критерию. Задача кластеризации относится к классу задач обучения без учителя.

Асинхронное программирование – реализация программного кода, позволяющая продолжить выполнение следующих частей кода, не ожидая завершения работы той части кода, которая начала исполняться ранее.

Darknet – скрытая сеть интернет-соединений, существующая параллельно обычному Интернету

# ВВЕДЕНИЕ

Традиционная банковская система развивалась и формировалась на протяжении многих столетий и все это время не имела альтернатив. Однако, начиная с 2008 года, когда впервые была описана технология блокчейн [1], наравне с традиционными валютами начало набирать популярность такое явление, как цифровые деньги. Анонсирование блокчейна сделало возможным существование децентрализованных платформ, продемонстрировало, что хранить и передавать ценности надёжно и безопасно можно и без использования банковских структур.

Уникальность блокчейна заключается том, что функционирование и защита системы основаны на криптографических методах. Децентрализованность технологии и высокий уровень безопасности обеспечил необходимое доверие пользователей. Возможность совершать транзакции анонимно также сделало блокчейн привлекательней по сравнению с традиционными банковским системами.

Однако, вышеописанные уникальные характеристики блокчейна помимо очевидных преимуществ несут в себе также и угрозу злоупотребления анонимностью путем совершения транзакций, связанных с финансированием наркотрафика, незаконной покупкой и продажей оружия, поддельных документов, а также прочей незаконной деятельности. Разумеется, пользователи, осуществляющие подобную деятельность, составляют лишь малую долю среди всех пользователей блокчейна, тем не менее известны случаи, в которых криптовалюта использовалась бы для финансирования терроризма, краж, наркотрафика, мошенничества и вымогательства. Одним из наиболее известных примеров использования криптовалюты в преступной деятельности является Silk Road [2].

Существование возможности злоупотребления уникальными свойствами блокчейна подрывает доверие к технологии со стороны финансовых регуляторов, правоохранительных органов и разведывательных служб. Для правительства факт использования блокчейна для совершения транзакций, связанных с теневой экономикой, представляет серьезную опасность. В связи с этим возникает необходимость изучить возможность выявления подозрительных транзакций в блокчейне.

Целью данной работы является изучение основных принципов функционирования технологии блокчейн, а также применение методов машинного обучения для исследования возможности верхнеуровнего выделения подозрительных транзакций из общей массы с целью их последующего детального рассмотрения.

# ОСНОВНАЯ ЧАСТЬ

## 1 Изучение принципов функционирования блокчейна

Блокчейн – база данных, совместно используемая всеми пользователями, совершающими транзакции с криптовалютой. Эта база данных представлена в виде цепочки блоков, хранящих информацию о всех совершенных операциях. Блоки в цепи нельзя заменить или сфальсифицировать, так как добавление нового блока верифицируется всеми участниками, а в дальнейшем копия сформированной цепи хранится у каждого участника. Полная копия цепочки блоков содержит все когда-либо совершенные и подтвержденные транзакции. Используя эту информацию, можно точно узнать сумму и время совершения транзакций, адреса отправителей и получателей, а также прочие параметры транзакции независимо от того, как давно она была совершена. Таким образом, несмотря на анонимность адресов, блокчейн абсолютно прозрачен, а история транзакций в публичном блокчейне доступна для изучения и анализа любому заинтересованному лицу.

## 2 Формулирование гипотезы

Все вышеописанное подтверждает утверждение о прозрачности блокчейна. Тот факт, что движение криптовалюты между адресами можно с легкостью отследить, позволяет сделать предположение, что существует возможность выявить закономерности в перемещении средств между адресами и на основе этого сделать выводы, насколько различные адреса в сети адреса связаны между собой.

Имея в качестве отправной точки адреса, для которых существуют основания считать их подозрительными, можно получить полную историю транзакций. На основе полученной истории можно сформировать матрицу признаков, которая давала бы некоторое константное описание рассматриваемым адресам. Такая матрица может позволить выявить особенности, которые отличают подозрительные адреса от адресов, используемых обычными пользователями блокчейна для операций, не связанных с незаконной деятельностью. В дальнейшем, сравнивая параметры случайно взятых адресов с аналогичными параметрами, полученными для подозрительных, можно будет делать выводы о степени подозрительности того или иного адреса, а также о уровне опасности, который представляют транзакции, совершенные с его использованием.

Разумеется, существует проблема «одноразовости» адресов. Она заключается в том, что преступник, желающий скрыть свои транзакции, будет использовать каждый адрес не более 1-2 раз, и по таким адресам нельзя будет сформировать достаточно репрезентативную матрицу признаков. Однако, логично предположить, что средства с таких адресов хоть и претерпевают множество переводов, в конечном счете скапливаются на ограниченном количестве итоговых адресов, принадлежащих единственному пользователю с целью их последующего выведения. Не исключено, что по итогам выявления закономерностей в матрице признаков в качестве подозрительных будут выделены также и одноразовые адреса.

## 3 Работа с API блокчейна

Как уже было сказано выше, вся история транзакций по всем когда-либо существующим адресам доступна любому заинтересованному лицу. Технически получение информации выглядит как обращение к API блокчейна. Для каждой криптовалюты транзакции записываются в разные блокчейны. В данной работе рассматривался блокчейн криптовалюты Bitcoin. В силу того, что данная криптовалюта наиболее распространена по сравнению с прочими, вероятность ее использования для совершения подозрительных транзакций гораздо выше, чем для прочих криптовалют.

Существует множество провайдеров, предоставляющих собственное программное решение для доступа к информации из блокчейна. Каждое из таких решений обладает преимуществами для выполнения той или иной практической задачи. В соответствии с целями данного исследования основными критериями при выборе решения были возможность получения максимально полной информации о транзакциях, а также бесплатный доступ и неограниченное количество запросов к API. Таким образом, был выбран API провайдера [5]. Дальнейшее описание данных и процесса их получения будет представлено через призму работы с данным конкретным API.

### 3.1 Изучение API

Документация для рассматриваемого API описана в [6].

Согласно документации, API позволяет делать следующие запросы:

− запрос для получения информации о блоках (с возможность указания нескольких номеров блоков)

− запрос для получения информации о конкретной транзакции по ее id

− запрос для получения информации о всех транзакциях по конкретному адресу

− запрос для получения информации о курсе валюты в определенный момент времени

Для целей настоящей работы представляет интерес получение информации по id транзакции, а также по адресам участников транзакций.

Для любого совершенного запроса ответ возвращается в формате, описанном на рисунке 1.

{

"data": ...,

"err\_no": 0,

"err\_msg": null

}

Рисунок 1 – Формат ответа сервера

Здесь *data* – тело ответа, *error\_no*– код ошибки (0 – OK, 1 - Resource Not Found, 2 - Parameter Error), а *error\_msg* – текст ошибки.

Такой формат позволяет корректно обрабатывать ошибки, возвращаемые сервером, с целью совершения повторного, но уже скорректированного запроса.

На запрос по id транзакции в поле data возвращается информация, в соответствии со структурой, представленной на рисунке 2.

data: {

confirmations: int

block\_height: int

block\_hash: string

block\_time: int

created\_at: int

fee: int

hash: string

inputs\_count: int

inputs\_value: int

is\_coinbase: boolean

is\_double\_spent: boolean

is\_sw\_tx: boolean

outputs\_count: int

outputs\_value: int

weight: int

vsize: int

witness\_hash: string

lock\_time: int

size: int

sigops: int

version: int

inputs: [

{

"prev\_addresses": Array<String>

"prev\_position": int

"prev\_tx\_hash": string

"prev\_value": int

"script\_asm": string

"script\_hex": string

"sequence": int

},

],

outputs: [

{

addresses: Array<String>

value: int

type: string

spent\_by\_tx: string

spent\_by\_tx\_position: int

}

],

}

Рисунок 2 – Содержания поля data при совершении запроса по

id транзакции

На запрос по адресу в поле data возвращается информация, в соответствии со структурой, представленной на рисунке 3.

data: {

total\_count: int

page: int

page\_size: int

list: [

/список транзакций, каждый элемент которого имеет структуру, описанную на рисунке 2/

]

}

Рисунок 3 – Содержания поля data при совершении

запроса по адресу

Таким образом, можно реализовать код, который бы делал запросы к API, передавая в качестве параметров адрес (или id транзакции), а также формировал структуру, удобную для последующей обработки.

### 3.2 Описание подхода к получению данных

Как было сказано ранее, имея в качестве отправной точки адреса, для которых существуют основания считать их подозрительными, можно получить полную историю транзакций по данным адресам и сформировать на основе этой истории матрицу признаков, которая давала бы константное описание рассматриваемому адресу.

Для того, чтобы по результатам кластеризации можно было делать определенные выводы, необходимо иметь некоторый список адресов, для которых есть основания считать их подозрительными, чтобы в дальнейшем изучить, в какой кластер они попали, а также детально изучить их соседей по кластеру. Ограниченный список таких адресов был получен путем исследования просторов сети Darknet.

Далее был реализован код для асинхронного получения JSON-файлов с полной историей транзакций по таким адресам, который одновременно записывал в отдельную структуру все адреса, связанные с рассматриваемыми, и далее повторял рекурсивно аналогичные действия для собранных адресов (столько раз, сколько указано при вызове скрипта). Необходимость написания кода с использованием библиотеки для асинхронного программирования объясняется тем, что объем данных значительный, а алгоритм запросов имеет сложную структуру (будет описана в разделе 3.3), и выполнение последовательных запросов не позволило бы в разумные сроки получить всю необходимую информацию.

### 3.3 Описание кода для получения данных

Код, описанный далее, представлен в приложении А.

#### 3.3.1 Библиотеки

Для реализации кода использовались следующие библиотеки: os, ujson, json, requests, pandas, sys, re, math, random, itertools, aiohttp\_socks, asyncio, aiohttp, aiofiles.

#### 3.3.2 Вспомогательная функция fetch

Была реализована вспомогательная функция fetch, которая делала в цикле запрос по url, обрабатывала ошибки (список ошибок был получен эмпирическим путем) и повторяла запрос.

Для обхода ограничений на большое число запросов к API за малый промежуток времени был собран список proxy-серверов. Далее при каждом вызове функции fetch из данного списка случайным образом выбирался адрес сервера.

#### 3.3.3 Функция main

В функции main реализовано два режима выгрузки информации из блокчейна: get\_info\_by\_transaction\_list и get\_info\_by\_address\_list (выбранный режим необходимо передавать в качестве параметра при запуске скрипта). Первый режим отвечает за получение информации по заранее собранному списку транзакций, а второй – по списку адресов. Данные списки можно рассматривать в качестве первого уровня выгрузки. Далее рекурсивно выгружалась информация по связанным адресам и транзакциям, полученным на предыдущем уровне (столько раз, сколько указано при запуске скрипта).

#### 3.3.4 Режим get\_info\_by\_transaction\_list

Если при запуске скрипта в качестве параметра указан режим get\_info\_by\_transaction\_list, то в коде вызывается функция get\_info\_by\_transaction\_list.

Данная функция принимает на вход следующие параметры:

− список транзакций curr\_tr\_list для выгрузки первого уровня (в жтом списке указаны транзакции, которые считаем подозрительными)

− путь до файла, в который будет записываться структура, сформированная из получаемого JSON-файла (так как нас интересуют не все поля в возвращаемом JSON, представляется разумным сразу же извлекать из него полезную информацию и записывать ее в структуру, удобную для последующей работы с собранными данными)

− параметр n, отвечающий за число уровней для совершения рекурсивных запросов (по умолчанию равен 3)

Основной цикл отвечает за выполнение набора действий для указанного числа n. Тело цикла состоит из следующего набора действий:

− для списка curr\_tr\_list формируются чанки tx\_chunks. Для всех транзакций, попавших в чанк, запросы будут выполняться одновременно

− для каждой транзакции в чанке вызывается функция get\_tx, возвращающая информацию по транзакции (в свою очередь использует функцию fetch, описанную выше)

− код ожидает выполнение запросов для всех транзакций в чанке, записывая результаты запросов в структуру data (представляет собой список, элементами которого являются результаты работы функции get\_tx для каждой транзакции)

− для списка data снова формируются чанки create\_row\_chunks

− для каждого элемента в create\_row\_chunks одновременно выполняется функция create\_row, которая формирует из сырого JSON структуру, с которой далее будет удобно работать

− эта структура записыватся в конец файла, путь до которого передается в качестве параметра path\_fin в функцию get\_info\_by\_transaction\_list

#### 3.3.5 Режим get\_info\_by\_address\_list

Алгоритм выгрузки для этого режима аналогичен алгоритму, описанному в пункте 3.3.4. Однако, здесь есть свои особенности.

Основным отличием является то, что за все время существования адреса с ним могут быть связаны сотни транзакций. В таком случае при стандартном запросе (без указания дополнительных параметров) API возвращает только первую страницу со списком транзакций. Разумеется, этот список не полный. Таким образом, сначала необходимо сначала сделать первый запрос, который вернет количество страниц по рассматриваемому адресу, а далее реализовать цикл, который бы вернул информацию по каждой из страниц.

Получение всех страниц для адреса также реализовано с помощью чанков. Для каждой полученной страницы собираются все транзакции, которые были совершены с участием рассматриваемого адреса. А далее для полученного списка транзакций выполняется набор действий, описанный в разделе 3.3.4.

#### 3.3.6 Функция create\_row

Вне зависимости от выбранного режима выгрузки (по адресам или по транзакциям) основным шагом является формирование интересующей нас структуры из полученных сырых данных. За формирование такой структуры отвечает функция create\_row.

Помимо очевидных признаков транзакции таких как id, размер транзакции, ее сумма, число входящих и исходящих адресов, комиссия, время совершения и прочих, представляется разумным также сформировать признак, который бы отражал время задержки данной транзакции относительно предыдущих и последующих (как известно, транзакции в блокчейне неразрывно связаны одна с другой).

Сложность заключается в том, что если у транзакции несколько входов и/или несколько выходов, то однозначно определить время совершения предыдущей/следующей транзакции невозможно, так как на входящие адреса средства поступили в разный момент времени, и, аналогично, были переведены с исходящих адресов в разный момент времени. Таким образом, для подобных случаев можно получить время совершения предыдущих транзакций (а также их сумму) для каждого входящего адреса, время следующих транзакций (если они уже совершены) для каждого исходящего адреса, а далее сформировать признаки max/min/avg.

Для формирования признаков, связанных с определением задержки транзакции относительно предыдущей/следующей вызывается функция create\_time\_features.

#### 3.3.7 Описание итогового JSON

В результате работы кода, описанного выше, формируется большое количество файлов, которые складываются в папку add\_jsons. Каждый файл содержит информацию об отдельной транзакции и имеет следующую структуру:

− txid (id транзакции)

− confirmations (число подтверждений блока, в который попала текущая транзакция)

− time (время совершения текущей транзакции)

− is\_coinbase

− is\_double\_spend

− is\_sw\_tx

− weight (вес блока)

− vsize

− inputs\_count (число входящих адресов)

− outputs\_count (число исходящих адресов)

− inputs\_value (сумма на входе)

− outputs\_value (сумма на входе)

− max\_prev\_time (максимальное, т.е. самое последнее время совершения транзакции среди всех предыдущих)

− min\_prev\_time (минимальное, т.е. самое раннее время совершения транзакции среди всех предыдущих)

− avg\_prev\_time (среднее время совершения транзакции среди всех предыдущих)

− max\_prev\_value (максимальная сумма среди всех предыдущих транзакций)

− min\_prev\_value (минимальная сумма транзакции среди всех предыдущих транзакций)

− avg\_prev\_value (средняя сумма транзакции среди всех предыдущих транзакций)

− max\_next\_time (максимальное, т.е. самое последнее время совершения транзакции среди всех последующих)

− min\_next\_time (минимальное, т.е. самое ближайшее время совершения транзакции среди всех последующих)

− avg\_next\_time (среднее время совершения транзакции среди всех последующих)

− max\_next\_value (максимальная сумма среди всех последующих транзакций)

− min\_next\_value (минимальная сумма среди всех последующих транзакций)

− avg\_next\_value (средняя сумма среди всех последующих транзакций)

− list\_of\_input (список входящих адресов для текущей транзакции и их параметры)

− prev\_address (адрес)

− curr\_value (сумма, которая пришла на этот адрес)

− prev\_time (время совершения предыдущей транзакции для данного адреса)

− total\_prev\_val (общая сумма, транзакции в рамках которой на адрес пришла сумма curr\_value)

− list\_of\_output (список исходящих адресов для текущей транзакции и их параметры)

− next\_address (адрес)

− curr\_value (сумма, которая ушла с этого адреса)

− next\_time (время совершения следующей транзакции для данного адреса)

− total\_next\_val (общая сумма, транзакции в рамках которой на адрес была потрачена сумма curr\_value)

Далее работа с сформированными файлами ведется в Jupiter Notebook.

## 4 Подходы к проведению кластеризации

Задачу кластеризации можно сформулировать двумя способами: можно кластеризовать транзакции, а можно провести кластеризацию адресов.

## 4.1 Кластеризация транзакций

Для данного подхода необходимо сформировать матрицу «транзакция-признак». Формирование такой матрицы не требует больших усилий в силу того, что данные изначально группировались в структуры, удобные для подобного преобразования. Таким образом, для формирования матрицы «транзакция-признак» достаточно в цикле считать JSON-файлы, полученные по каждой транзакции (процедура получения описана в разделе 3), а затем провести первичную предобработку, т.е. заменить переменные логического типа на 1/0, удалить строковые признаки (например, id транзакции), а также нормализовать данные.

## 4.2 Кластеризация адресов

Для выполнения задачи кластеризации адресов необходимо сформировать новую матрицу, в которой по строкам были бы адреса (а не транзакции, как в 4.1). То есть необходимо некоторым образом агрегировать все транзакции по каждому адресу, и на основе проведенной агрегации сформировать признаки, которые могли бы характеризовать каждый рассматриваемый адрес с разных сторон.

### 4.2.1 Формирование матрицы «адрес-признак» для проведения кластеризации

Код для формирования матрицы «адрес-признак» представлен в приложении Б.

Подход к формированию матрицы «адрес-признак» напрямую зависит от типа транзакции. Все собранные транзакции можно разделить на 4 группы: 1 – транзакция имеет один входной и один выходной адрес; 2 – транзакция имеет более одного входного адреса и один выходной; 3 – транзакция имеет один входной и более одного выходного адреса; 4 – транзакция имеет более одного входного и более одного выходного адреса.

В зависимости от выделенных типов будем по-разному формировать матрицу «адрес-признак».

На первом шаге хотим получить матрицу, где по строкам будут транзакции относительно адресов (для каждой строки только один входной и один выходной адрес): если у транзакции было n входящих (исходящих) адресов, то данная транзакция преобразовывается в n строк; если у транзакции было n входящих и m исходящих адресов, то данная транзакция преобразовывается в n+m строк.

В полученной матрице каждая строка будет состоять из следующих полей:

− 'txid' - id текущей транзакции

− 'confirmations' - число подтверждений

− 'time' - время совершения данной транзакции

− 'is\_coinbase' - флаг

− 'is\_double\_spend' - флаг

− 'is\_sw\_tx' - флаг

− 'weight' - вес

− 'vsize' - вес

− 'inputs\_count' - число входящих адресов

− 'outputs\_count'- число исходящих адресов

− 'inputs\_value' - общая сумма текущей транзакции (отличается от curr\_value, если более 1 входа)

− 'outputs\_value' - общая сумма исходящей транзакции (отличается от curr\_value-fee, если более 1 выхода)

− 'max\_prev\_val' - максимальная сумма среди предыдущих транзакций (отличается от tot\_prev\_tx\_value, если более 1 входа)

− 'min\_prev\_val' - минимальная сумма среди предыдущих транзакций (отличается от tot\_prev\_tx\_value, если более 1 входа)

− 'avg\_prev\_val' - средняя сумма предыдущих транзакций (отличается от tot\_prev\_tx\_value, если более 1 входа)

− 'max\_next\_val' - максимальная сумма среди следующих транзакций (отличается от tot\_next\_tx\_value, если более 1 входа)

− 'min\_next\_val' - минимальная сумма среди следующих транзакций (отличается от tot\_next\_tx\_value, если более 1 входа)

− 'avg\_next\_val' - средняя сумма следующих транзакций (отличается от tot\_next\_tx\_value, если более 1 входа)

− 'diff\_max\_prev\_time' - максимальная разница во времени текущей транзакции относительно всех предыдущих (отличается от diff\_prev\_time, если более 1 входа)

− 'diff\_min\_prev\_time' - минимальна разница во времени текущей транзакции относительно всех предыдущих (отличается от diff\_prev\_time, если более 1 входа)

− 'diff\_avg\_prev\_time' - средняя разница во времени текущей транзакции относительно всех предыдущих (отличается от diff\_prev\_time, если более 1 входа)

− 'diff\_max\_next\_time' - максимальная разница во времени текущей транзакции относительно всех следующих (отличается от diff\_next\_time, если более 1 входа)

− 'diff\_min\_next\_time' - минимальная разница во времени текущей транзакции относительно всех следующих (отличается от diff\_next\_time, если более 1 входа)

− 'diff\_avg\_next\_time' - средняя разница во времени текущей транзакции относительно всех следующих (отличается от diff\_next\_time, если более 1 входа)

− 'is\_suspicious' - флаг, является ли транзакция подозрительной (1, если транзакция есть в первоначальном списке tr\_list)

− 'transaction\_type' - тип пранзакции (1,2,3,4)

− 'input\_add' - входящий адрес

− 'output\_add' - исходящий адрес

− 'curr\_value' - сумма транзакции для данного входящего/исходящего адреса (в зависимости от transaction\_type)

− 'tot\_prev\_tx\_value' - размер предыдущей транзакции для данного input\_add (если transaction\_type=4, то берется avg\_prev\_val)

− 'tot\_next\_tx\_value' - размер следующей транзакции для данного input\_add (если transaction\_type=4, то берется avg\_next\_val)

− 'diff\_prev\_time' – задержка относительно предыдущей транзакций

− 'diff\_next\_time' – задержка относительно следующей транзакций

Для каждого типа транзакции выполняются свои преобразования, чтобы получить матрицу с признаками, описанными выше. Далее все 4 полученных датафрейма конкатенируются.

После конкатенации необходимо провести группировку по адресам, чтобы получить для каждого адреса только одну строку. В итоговой матрице будут следующие признаки:

− count\_tx - число транзакций, в которых участвовал данный адрес

− sum\_is\_coinbase (если больше нуля, то в какой-либо транзакции в этом поле была единица)

− sum\_is\_double\_spend (если больше нуля, то в какой-либо транзакции в этом поле была единица)

− sum\_is\_sw\_tx (если больше нуля, то в какой-либо транзакции в этом поле была единица)

− nunique\_transaction\_type – число уникальных типов транзакций, в которых участвовал данный адрес

− sum\_the\_same\_input\_output – число транзакций, в которых адрес участвует как входной и выходной

− weight (min, max, avg) – минимальный, максимальный и средний вес блоков, среди всех транзакций для этого адреса

− weight (min, max, avg) – минимальный, максимальный и средний вес блоков, среди всех транзакций для этого адреса

− inputs\_count (min, max, avg) – минимальное, максимальное и среднее число входов, среди всех транзакций для этого адреса

− outputs\_count (min, max, avg) – минимальное, максимальное и среднее число входов, среди всех транзакций для этого адреса

− inputs\_value (min, max, avg) – минимальная, максимальная и средняя сумма транзакций, среди всех транзакций для этого адреса

− max\_prev\_val (min, max, avg) – минимальная, максимальная и средняя сумма среди максимальных сумм предыдущих транзакций, среди всех транзакций для этого адреса

− min\_prev\_val (min, max, avg) – минимальная, максимальная и средняя сумма среди минимальных сумм предыдущих транзакций, среди всех транзакций для этого адреса

− avg\_prev\_val (min, max, avg) – минимальная, максимальная и средняя сумма среди средних сумм предыдущих транзакций, среди всех транзакций для этого адреса

− max\_next\_val (min, max, avg) – минимальная, максимальная и средняя сумма среди максимальных сумм следующих транзакций, среди всех транзакций для этого адреса

− min\_next\_val (min, max, avg) – минимальная, максимальная и средняя сумма среди минимальных сумм следующих транзакций, среди всех транзакций для этого адреса

− avg\_next\_val (min, max, avg) – минимальная, максимальная и средняя сумма среди средних сумм следующих транзакций, среди всех транзакций для этого адреса

− diff\_max\_prev\_time (min, max, avg) – минимальная, максимальная и средняя разница во времени между временем совершения текущей транзакции и самой последней среди предыдущих, среди всех транзакций для этого адреса

− diff\_min\_prev\_time (min, max, avg) – минимальная, максимальная и средняя разница во времени между временем совершения текущей транзакции и самой ранней среди предыдущих, среди всех транзакций для этого адреса

− diff\_avg\_prev\_time (min, max, avg) – минимальная, максимальная и средняя разница во времени между временем совершения текущей транзакции и средним временем совершения предыдущих, среди всех транзакций для этого адреса

− diff\_max\_next\_time (min, max, avg) – минимальная, максимальная и средняя разница во времени между временем совершения текущей транзакции и самой последней среди следующих, среди всех транзакций для этого адреса

− diff\_min\_next\_time (min, max, avg) – минимальная, максимальная и средняя разница во времени между временем совершения текущей транзакции и самой ранней среди следующих, среди всех транзакций для этого адреса

− diff\_avg\_next\_time (min, max, avg) – минимальная, максимальная и средняя разница во времени между временем совершения текущей транзакции и средним временем совершения следующих, среди всех транзакций для этого адреса

− tot\_prev\_tx\_val (min, max, avg) – минимальная, максимальная и средняя разница сумма среди общих сумм предыдущих транзакций, среди всех транзакций для этого адреса

− diff\_avg\_next\_time (min, max, avg) – минимальная, максимальная и разница сумма среди общих сумм следующих транзакций, среди всех транзакций для этого адреса

− tot\_prev\_tx\_val (min, max, avg) – минимальная, максимальная и средняя сумма среди общих сумм предыдущих транзакций, среди всех транзакций для этого адреса

− diff\_avg\_next\_time (min, max, avg) – минимальная, максимальная и средняя сумма среди общих сумм следующих транзакций, среди всех транзакций для этого адреса

− diff\_prev\_time (min, max, avg) – минимальная, максимальная и средняя разница задержка по времени, среди всех транзакций для этого адреса

− diff\_next\_time (min, max, avg) – минимальная, максимальная и средняя задержка по времени, среди всех транзакций для этого адреса

## 5 Применение методов снижения размерности. Проведение кластеризации

В разделе 4 были описаны два варианта постановки задачи. Проведение кластеризации транзакций результатов для целей данного исследования подходит меньше, чем кластеризация адресов, т.к. имеется недостаточное количество id транзакций, которые априорно можно считать подозрительными. Таким образом, далее будет решаться задача кластеризации адресов.

### 5.1K-means

Теория

### 5.2 TSNE и PCA

Теория

…

Вывод: применение описанных методов в настоящем исследовании не дало ожидаемых результатов.

### 5.2 Проведение кластеризации и анализ результатов

По результатам работы кода, описанного в разделе 3 были получены данные по 100267 транзакциям. Это соответствует полной истории транзакций по 679 адресам. Среди данных 679 адресов 6 размечены как подозрительные.

Далее на этих 679 адресах проводилась кластеризация методом K-Means.

Программный код для проведения кластеризации представлен в приложении В.

Первым шагом было определение числа кластеров. Для этого был построен график зависимости функционала ошибки от числа кластеров, представленный на рисунке 4.

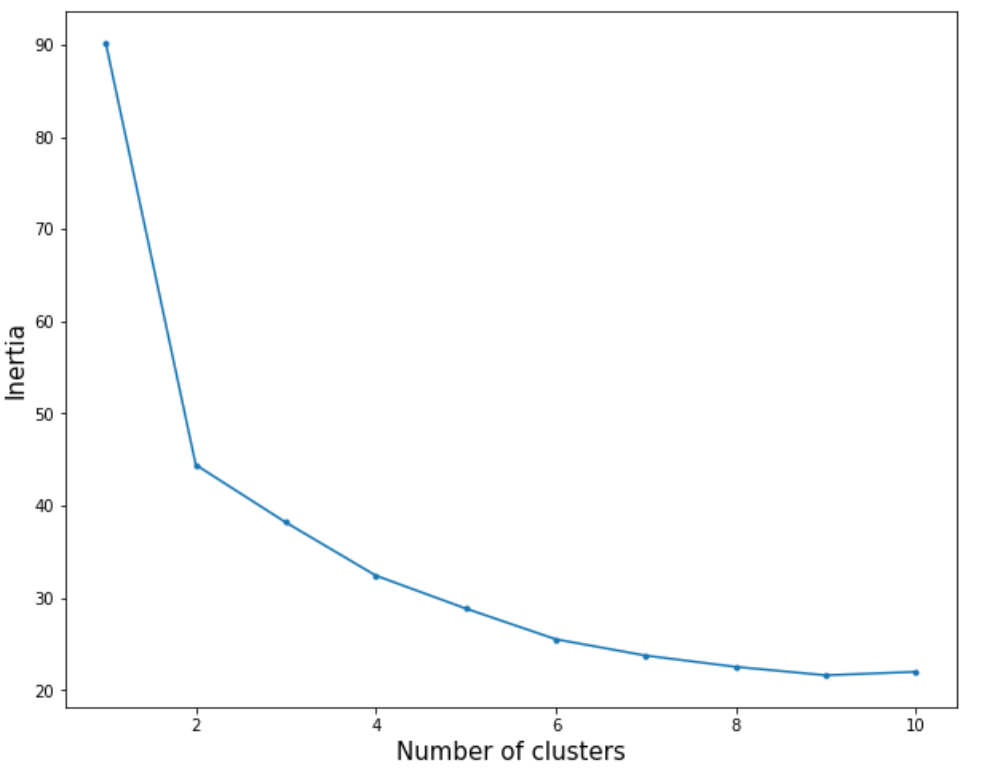


Рисунок 4 – График зависимости функционала ошибки от числа кластеров

По графику было выбрано оптимальное число кластеров, равное 9.

При дальнейшем проведении кластеризации был получен состав кластеров, указанный в таблице 1.

Таблица 1 – Состав кластеров

|  |  |
| --- | --- |
| Номер кластера | Количество элементов в кластере |
| 0 | 187 |
| 1 | 47 |
| 2 | 255 |
| 3 | 7 |
| 4 | 53 |
| 5 | 62 |
| 6 | 4 |
| 7 | 1 |
| 8 | 63 |

При детальном анализе состава кластеров требовалось определить, в какие кластера попали исходные 6 подозрительных транзакций. Результаты анализа представлены на рисунке 5.

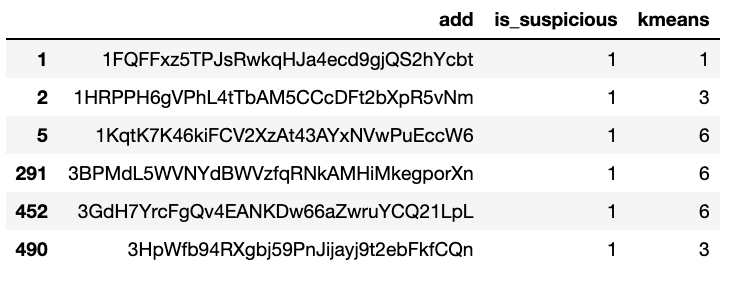


Рисунок 5 – Распределение по кластерам

подозрительных транзакций

Таким образом, в результате анализа получили, что все транзакции, отмеченных в качестве подозрительных, попали в кластеры с наименьшим числом элементов. На основе этого можно сделать вывод, что в рамках имеющейся выборки выделенные адреса имеют некоторые признаки, которые отличают их от общей массы адресов в рассматриваемом датасете.

# ЗАКЛЮЧЕНИЕ

Целью данной работы было изучение возможности выявления подозрительных транзакций в блокчейне с помощью методов машинного обучения.

Первым шагом был поиск адресов, которые можно было бы отнести к числу подозрительных. После сбора таких адресов необходимо было выгрузить историю транзакций по этим адресам из блокчейна, а также историю по случайным адресам, для которых не имелось оснований считать их подозрительными.

Для целей получения истории транзакций был подготовлен программный код на языке Python, реализующий выгрузку данных о транзакциях на криптовалюте Bitcoin. Работа кода построена на взаимодействии с API блокчейна. Были реализованы два режима выгрузки данных: по исходному списку id транзакций, а также по исходному списку адресов. Далее проводилось преобразование сырых исходных данных в структуру, которая являлась наиболее удобной для целей дальнейшего проведения исследования.

После сбора и первичной обработки данных был подготовлен программный код, формирующий матрицу признаков транзакций и агрегирующий строки этой матрицы по адресам. Далее была проведена кластеризация адресов с помощью метода K-Means. По результатам кластеризации было выявлено, что все транзакции, отмеченных в качестве подозрительных, попали в кластеры с наименьшим числом элементов. На основе этого можно сделать вывод, что в рамках имеющейся выборки выделенные адреса имеют некоторые признаки, которые отличают их от общей массы адресов в рассматриваемом датасете.

Разумеется, малый объем исходных данных не позволяет распространить сделанный вывод на всю генеральную совокупность. Однако, полученные результаты дают почву для проведения углубленного исследования с целью получения более репрезентативных и конкретных результатов, которое, к сожалению, не представлялось возможности провести в рамках данной работы в силу нехватки вычислительных мощностей и прочих ресурсов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
2. N. Christin, “Traveling the silk road: A measurement analysis of a large anonymous online marketplace,” in Proceedings of the 22Nd International Conference on World Wide Web, WWW ’13, (New York, NY, USA), pp. 213–224, ACM, 2013.
3. [Bitcoin](https://en.bitcoin.it/wiki/Bitcoin" \o "Bitcoin) Wiki [Электронный ресурс]. – URL: https://en.bitcoin.it/wiki/Transaction (дата обращения 01.05.2018)
4. [Bitcoin](https://en.bitcoin.it/wiki/Bitcoin" \o "Bitcoin) Wiki [Электронный ресурс]. – URL: https://en.bitcoin.it/wiki/Block\_chain (дата обращения 01.05.2018)
5. Провайдер API [Электронный ресурс]. – URL: https://btc.com/ (дата обращения 01.05.2018)
6. Документация к API [Электронный ресурс]. – URL: https://btc.com/api-doc (дата обращения 01.05.2018)

# ПРИЛОЖЕНИЕ А

Программный код для получения информации о транзакциях из блокчейна

import time, os, sys, re, math

import ujson, json

import requests

import pandas as pd

from statistics import mean

from proxy\_list import USER\_AGENTS, PROXIES, tr\_list, addr\_list

import asyncio, aiohttp, aiofiles, random, itertools, sys, json, os

from aiohttp\_socks import SocksConnector, SocksVer

from aiohttp\_socks.errors import SocksError, SocksConnectionError

PROXIES['HTTP'] = [tuple(line.split(':')) for line in PROXIES['HTTP'].strip().split('\n')]

PROXIES['SOCKS'] = [tuple(line.split(':')) for line in PROXIES['SOCKS'].strip().split('\n')]

# перемешиваем список прокси для равномерного распределения

random.shuffle(PROXIES['HTTP'])

random.shuffle(PROXIES['SOCKS'])

# создаем итератор, который будет возвращать при вызове next (PROXY\_ITER) каждый раз разный прокси

PROXY\_ITER = {

'HTTP': itertools.cycle(PROXIES['HTTP']),

'SOCKS': itertools.cycle(PROXIES['SOCKS']),

}

USE\_PROXY = 'SOCKS' # False, 'HTTP', 'SOCKS'

FETCH\_ATTEMPTS = 5

FETCH\_TIMEOUT = 10

def chunkify(seq, size):

return (seq[pos:pos + size] for pos in range(0, len(seq), size))

def get\_random\_proxy\_params(use\_proxy):

if not use\_proxy:

connector = None

proxy\_url = None

elif use\_proxy == 'HTTP':

connector = None

proxy = next(PROXY\_ITER['HTTP'])

if len(proxy) == 2:

host, port = proxy

proxy\_url = f'http://{host}:{port}'

elif len(proxy) == 4:

host, port, username, password = proxy

proxy\_url = f'http://{username}:{password}@{host}:{port}'

elif use\_proxy == 'SOCKS':

proxy = next(PROXY\_ITER['SOCKS'])

if len(proxy) == 2:

host, port = proxy

username = password = None

elif len(proxy) == 4:

host, port, username, password = proxy

connector = SocksConnector(SocksVer.SOCKS5, host, int(port), username=username, password=password)

proxy\_url = None

return connector, proxy\_url

async def fetch(url, params=None, attempts=FETCH\_ATTEMPTS, use\_proxy=USE\_PROXY):

""" обёртка над aiohttp.get() с повтором запроса при ошибке + работа с прокси """

for attempt in range(attempts):

try:

connector, proxy\_url = get\_random\_proxy\_params(use\_proxy)

async with aiohttp.ClientSession(connector=connector) as session:

ua = random.choice(USER\_AGENTS)

# отправляем запрос

response = await session.get(

url,

params=params,

headers={"User-Agent": ua},

proxy=proxy\_url,

timeout=aiohttp.ClientTimeout(total=FETCH\_TIMEOUT),

)

if response.status != 200: # получили ошибку

data = await response.text()

print(f'{response.url} returned HTTP {response.status}: {data}')

await asyncio.sleep(1.0)

continue # пробуем еще раз (следующий attempt в цикле)

else: # успешно

data = await response.json()

return data

except (asyncio.TimeoutError, SocksError, SocksConnectionError, ConnectionResetError, BrokenPipeError, \

aiohttp.client\_exceptions.ClientError) as e:

print(f'exception @ {params}:', e, str(e))

continue # пробуем еще раз

# тут мы оказываемся, если все попытки пройдут безуспешно, и функция по умолчанию вернет return None

CACHE\_DIR\_TX = '/tmp/btc\_txids'

CACHE\_DIR\_ADDS = '/tmp/btc\_adds'

os.makedirs(CACHE\_DIR\_TX, exist\_ok=True)

os.makedirs(CACHE\_DIR\_ADDS, exist\_ok=True)

curr\_tr\_list = tr\_list

curr\_add\_list = addr\_list

path\_tx\_fin = './tx\_json.json'

path\_add\_fin = './add\_jsons'

async def get\_tx(txid, attempts=5):

path = f'{CACHE\_DIR\_TX}/{txid}.json'

if os.path.exists(path):

for attempt in range(attempts):

try:

try:

async with aiofiles.open(path, 'r') as f:

data = ujson.loads(await f.read())

return data

except (ValueError) as e:

return {"data": {"hash": txid}, "err\_no": 666}

except (OSError) as e:

print(f'exception:', e, str(e))

continue # пробуем еще раз

else:

for attempt in range(attempts):

#print(f'{txid} is new')

data = await fetch(f'https://chain.api.btc.com/v3/tx/{txid}')

if not data or data.get('err\_no', 0) == 2:

#print(f'{txid}: {data}')

continue

else:

async with aiofiles.open(path, 'w') as f:

await f.write(ujson.dumps(data))

return data

if data is None or not data:

return {"data": {"hash": txid}, "err\_no": 666}

return data

async def get\_transactions\_by\_address(addr, page=1):

data = await fetch(f'https://chain.api.btc.com/v3/address/{addr}/tx?page={page}')

if data and 'data' in data:

txs = data['data']['list']

total\_count = data['data']['total\_count']

pagesize = 50

pages = math.ceil(total\_count/pagesize)

return data['data']

else:

return {"data": {"hash": None}, "err\_no": 666}

async def get\_all\_transactions\_by\_address(addr):

print(f'addr {addr}')

data = await get\_transactions\_by\_address(addr)

if not data:

return

total\_count = data['total\_count']

pagesize = 50

pages = math.ceil(total\_count/pagesize)

all\_txs = []

all\_txs += data['list']

tasks = [get\_transactions\_by\_address(addr, page=page) for page in range(2, pages+1)]

chunks = chunkify(tasks, 8)

for chunk in chunks:

chunk\_results = await asyncio.gather(\*chunk)

for result in chunk\_results:

if not result:

print(f"Got no data (out of attempts/timeout/etc) :(")

continue

all\_txs += result['list']

return all\_txs

def tr\_features(row, tx):

row['txid'] = tx['hash']

row['confirmations'] = tx['confirmations']

row['time'] = tx['block\_time']

row['is\_coinbase'] = tx['is\_coinbase']

row['is\_double\_spend'] = tx['is\_double\_spend']

row['is\_sw\_tx'] = tx['is\_sw\_tx']

row['weight'] = tx['weight']

row['vsize'] = tx['vsize']

return row

def inp\_outp\_features(row, tx):

row['inputs\_count'] = tx['inputs\_count']

row['outputs\_count'] = tx['outputs\_count']

row['inputs\_value'] = tx['inputs\_value']

row['outputs\_value'] = tx['outputs\_value']

inp\_adds = []

for inp in tx['inputs']:

inp\_adds.extend(inp['prev\_addresses'])

outp\_adds = []

for outp in tx['outputs']:

outp\_adds.extend(outp['addresses'])

return row

async def create\_time\_features(err, row, addr\_list, addr\_type, attempts=3):

txs = []

times = []

vals = []

meta\_tr\_dict = []

if addr\_type == 'input':

field\_type = 'inp\_address'

add\_type = 'prev\_addresses'

value\_type = 'prev\_value'

hash\_type = 'prev\_tx\_hash'

meta\_addr\_field = 'prev\_address'

meta\_time\_field = 'prev\_time'

meta\_value\_field = 'total\_prev\_val'

meta\_agg\_value\_field = 'prev\_val'

elif addr\_type == 'output':

field\_type = 'outp\_address'

add\_type = 'addresses'

value\_type = 'value'

hash\_type = 'spent\_by\_tx'

meta\_addr\_field = 'next\_address'

meta\_time\_field = 'next\_time'

meta\_value\_field = 'total\_next\_val'

meta\_agg\_value\_field = 'next\_val'

meta\_txs\_tasks = [get\_tx(itx[hash\_type]) for itx in addr\_list]

try:

addresses = [{field\_type: itx[add\_type][0], 'curr\_value': itx[value\_type]} for itx in addr\_list]

except:

chunks\_size = 16

txs\_chunks = list(chunkify(meta\_txs\_tasks, chunks\_size))

address\_chunks = list(chunkify(addresses, chunks\_size))

for chunk\_i, (txs\_chunk, address\_chunk) in enumerate( list(zip(txs\_chunks,address\_chunks)), start=1):

tmp\_txs = await asyncio.gather(\*txs\_chunk)

for j, tmp\_tx in enumerate(tmp\_txs):

try:

txid = tmp\_tx['data']['hash']

txs.append(txid)

if tmp\_tx.get('err\_no', '') != 0:

err.append(txid)

time = None

val = None

else:

time = tmp\_tx['data']['block\_time']

val = tmp\_tx['data']['inputs\_value']

times.append(time)

vals.append(val)

meta\_tr\_dict.append({meta\_addr\_field: address\_chunk[j][field\_type],

'curr\_value': address\_chunk[j]['curr\_value'],

meta\_time\_field: time,

meta\_value\_field: val

})

except TypeError as e:

continue # пробуем еще раз

times = [x for x in times if x]

if times:

row[f'max\_{meta\_time\_field}'] = max(times)

row[f'min\_{meta\_time\_field}'] = min(times)

row[f'avg\_{meta\_time\_field}'] = round(mean(times))

else:

row[f'max\_{meta\_time\_field}'] = None

row[f'min\_{meta\_time\_field}'] = None

row[f'avg\_{meta\_time\_field}'] = None

vals = [x for x in vals if x]

if vals:

row[f'max\_{meta\_agg\_value\_field}'] = max(vals)

row[f'min\_{meta\_agg\_value\_field}'] = min(vals)

row[f'avg\_{meta\_agg\_value\_field}'] = round(mean(vals))

else:

row[f'max\_{meta\_agg\_value\_field}'] = None

row[f'min\_{meta\_agg\_value\_field}'] = None

row[f'avg\_{meta\_agg\_value\_field}'] = None

row[f'list\_of\_{addr\_type}'] = meta\_tr\_dict

return err, row, txs

async def time\_features(row, tx):

new\_txs = []

err = []

err, row, txs = await create\_time\_features(err=err, row=row, addr\_list=tx['inputs'], addr\_type='input')

new\_txs += txs

err, row, txs = await create\_time\_features(err=err, row=row, addr\_list=tx['outputs'], addr\_type='output')

new\_txs += txs

return err, row, new\_txs

def diff\_time\_features(row):

if row['max\_prev\_time']: row['diff\_max\_prev\_time'] = row['time'] - row['max\_prev\_time']

else: row['diff\_max\_prev\_time'] = None

if row['min\_prev\_time']: row['diff\_min\_prev\_time'] = row['time'] - row['min\_prev\_time']

else: row['diff\_min\_prev\_time'] = None

if row['avg\_prev\_time']: row['diff\_avg\_prev\_time'] = row['time'] - row['avg\_prev\_time']

else: row['diff\_avg\_prev\_time'] = None

if row['max\_next\_time']: row['diff\_max\_next\_time'] = row['max\_next\_time'] - row['time']

else: row['diff\_max\_next\_time'] = None

if row['min\_next\_time']: row['diff\_min\_next\_time'] = row['min\_next\_time'] - row['time']

else: row['diff\_min\_next\_time'] = None

if row['avg\_next\_time']: row['diff\_avg\_next\_time'] = row['avg\_next\_time'] - row['time']

else: row['diff\_avg\_next\_time'] = None

return row

async def create\_row(tx):

print(tx['hash'])

row = {}

row = tr\_features(row, tx)

row = inp\_outp\_features(row, tx)

err, row, new\_txs = await time\_features(row, tx)

row = diff\_time\_features(row)

new\_inp\_adds = [x['prev\_address'] for x in row['list\_of\_input']]

new\_outp\_adds = [x['next\_address'] for x in row['list\_of\_output']]

return row, new\_txs, new\_inp\_adds, new\_outp\_adds

async def get\_info\_by\_transaction\_list(curr\_tr\_list, path\_fin, n=3):

df = {}

finished\_txs = []

for i in range(n):

next\_level\_txs = []

tx\_tasks = [get\_tx(tx) for tx in curr\_tr\_list]

tx\_chunks\_size = 16

tx\_chunks = list(chunkify(tx\_tasks, tx\_chunks\_size))

print('transactions:', len(curr\_tr\_list), 'chunks:', len(tx\_chunks))

for chunk\_i, tx\_chunk in enumerate(tx\_chunks, start=1):

data = await asyncio.gather(\*tx\_chunk)

create\_row\_tasks = [create\_row(tx['data']) for tx in data]

create\_row\_chunks\_size = 8

create\_row\_chunks = list(chunkify(create\_row\_tasks, create\_row\_chunks\_size))

print('txs to create new rows:', len(data), 'chunks:', len(create\_row\_chunks))

for chunk\_j, create\_row\_chunk in enumerate(create\_row\_chunks, start=1):

new\_rows = await asyncio.gather(\*create\_row\_chunk)

for step in new\_rows:

row, new\_txs, new\_inp\_adds, new\_outp\_adds = step[0], step[1], step[2], step[3]

df[row['txid']] = row

next\_level\_txs += new\_txs

finished\_txs.extend(curr\_tr\_list)

curr\_tr\_list = next\_level\_txs

async with aiofiles.open(path\_fin, 'a') as f:

await f.write(ujson.dumps(df))

async def get\_info\_by\_address\_list(curr\_add\_list, path\_fin, n=1):

with open('./finished\_adds', 'r') as f:

finished\_adds = f.read()

with open('./1\_level\_adds', 'r') as f:

curr\_add\_list\_read = f.read()

curr\_add\_list\_read = re.sub('[\[\]\"]', '', curr\_add\_list\_read).split(',')

finished\_adds = re.sub('[\[\]\"]', '', finished\_adds).split(',')

add\_errors = []

curr\_add\_list\_new = [add for add in curr\_add\_list\_read if add not in finished\_adds]

for i in range(n):

next\_level\_adds = []

add\_tasks = [get\_all\_transactions\_by\_address(add) for add in curr\_add\_list\_new]

addresses = [add for add in curr\_add\_list\_new]

add\_chunks\_size = 5

add\_info\_chunks = list(chunkify(add\_tasks, add\_chunks\_size))

add\_list\_chunks = list(chunkify(addresses, add\_chunks\_size))

print('adds:', len(curr\_add\_list\_new), 'chunks:', len(add\_info\_chunks))

for chunk\_i, add\_chunk in enumerate(add\_info\_chunks, start=1):

data = await asyncio.gather(\*add\_chunk)

for add in data:

print(f'add1: {len(add)}')

flat\_data = [item for sublist in data for item in sublist]

create\_row\_tasks = [create\_row(tx) for tx in flat\_data]

row\_chunks\_size = 5

row\_chunks = list(chunkify(create\_row\_tasks, row\_chunks\_size))

print('rows:', len(flat\_data), 'chunks:', len(row\_chunks))

for chunk\_j, row\_chunk in enumerate(row\_chunks, start=1):

print(f'[row chunk #{chunk\_j}/{len(row\_chunks)}]: fetching...')

rows = await asyncio.gather(\*row\_chunk)

for new\_row in rows:

row, new\_txs, new\_inp\_adds, new\_outp\_adds = new\_row[0], new\_row[1], new\_row[2], new\_row[3]

next\_level\_adds += new\_inp\_adds

next\_level\_adds += new\_outp\_adds

txid = row['txid']

path = path\_fin+f'/{txid}.json'

if os.path.exists(path) == False:

async with aiofiles.open(path, 'w') as f:

await f.write(ujson.dumps(row))

finished\_adds.extend(add\_list\_chunks[chunk\_i-1])

async with aiofiles.open('./finished\_adds', 'w') as f:

await f.write(ujson.dumps(finished\_adds))

async with aiofiles.open(f'./{i+2}\_level\_adds', 'w') as f:

await f.write(ujson.dumps(next\_level\_adds))

curr\_add\_list\_new = [add for add in next\_level\_adds if add not in finished\_adds]

async def main():

global curr\_tr\_list

global curr\_add\_list

global path\_tx\_fin

global path\_add\_fin

if len(sys.argv) != 2:

print('Usage:')

print(' python3 titles.py <command>\n')

print('Available commands:')

print(' - get\_info\_by\_transaction\_list')

print(' - get\_info\_by\_address\_list')

sys.exit(1)

if sys.argv[1] == 'get\_info\_by\_transaction\_list':

await get\_info\_by\_transaction\_list(curr\_tr\_list=curr\_tr\_list, path\_fin=path\_tx\_fin)

elif sys.argv[1] == 'get\_info\_by\_address\_list':

await get\_info\_by\_address\_list(curr\_add\_list=curr\_add\_list, path\_fin=path\_add\_fin)

if \_\_name\_\_ == '\_\_main\_\_':

asyncio.get\_event\_loop().run\_until\_complete(

main()

)

# ПРИЛОЖЕНИЕ Б

Программный код для формирования матрицы «объект-признак»

import time, os, re, json, requests

import numpy as np

import pandas as pd

import networkx as nx

from statistics import mean

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

from sklearn.decomposition import PCA

from sklearn.manifold import TSNE

import matplotlib.pyplot as plt

%matplotlib inline

#подозрительные транзакции

tr\_list = ['c7a3673f869fc9c5c5e8f032af97546efed0435762e4ef728f99b6725c027f9b',

'34ed08b17fff9cbe51ce33c62c6c1bb115f82aeec60997bfb216126bf747b5c8']

#подозрительные адреса

addr\_list = ['1HRPPH6gVPhL4tTbAM5CCcDFt2bXpR5vNm',

'3BPMdL5WVNYdBWVzfqRNkAMHiMkegporXn',

'3HpWfb94RXgbj59PnJijayj9t2ebFkfCQn',

'3GdH7YrcFgQv4EANKDw66aZwruYCQ21LpL',

'1FQFFxz5TPJsRwkqHJa4ecd9gjQS2hYcbt',

'1KqtK7K46kiFCV2XzAt43AYxNVwPuEccW6']

#READ

data = {}

%%time

path = './add\_jsons'

for file in os.listdir(path):

try:

with open(path + '/{}'.format(file), 'r') as f:

results = json.load(f)

txid = results['txid']

data[txid] = results

except:

print(file)

df = []

trs = list(data.keys())

for tr in trs:

df.append(data[tr])

cols = list(data[trs[0]].keys())

data = pd.DataFrame(df, columns=cols)

data['is\_coinbase'] = data['is\_coinbase'].map({False: 0, True: 1}).astype(int)

data['is\_double\_spend'] = data['is\_double\_spend'].map({False: 0, True: 1}).astype(int)

data['is\_sw\_tx'] = data['is\_sw\_tx'].map({False: 0, True: 1}).astype(int)

#Feature Matrix

def transaction\_type(row):

if row[0]==1 and row[1]==1: return 1

elif row[0] >1 and row[1]==1: return 2

elif row[0]==1 and row[1] >1: return 3

elif row[0] >1 and row[1] >1: return 4

else: return 0

data['transaction\_type'] = [transaction\_type(x) for x in data[['inputs\_count', 'outputs\_count']].values]

#кол-во транзакций каждого типа в датасете

data.groupby('transaction\_type').count()[['txid']].T

cols = ['txid',

'confirmations',

'time',

'is\_coinbase',

'is\_double\_spend',

'is\_sw\_tx',

'weight',

'vsize',

'inputs\_count',

'outputs\_count',

'inputs\_value',

'outputs\_value',

'max\_prev\_val',

'min\_prev\_val',

'avg\_prev\_val',

'max\_next\_val',

'min\_next\_val',

'avg\_next\_val',

'diff\_max\_prev\_time',

'diff\_min\_prev\_time',

'diff\_avg\_prev\_time',

'diff\_max\_next\_time',

'diff\_min\_next\_time',

'diff\_avg\_next\_time',

'is\_suspicious',

'transaction\_type',

'input\_add',

'output\_add',

'curr\_value',

'tot\_prev\_tx\_value',

'tot\_next\_tx\_value',

'diff\_prev\_time',

'diff\_next\_time']

#transaction\_type = 1

data\_1 = data[data['transaction\_type'] == 1]

#входящий и исходящий адреса

data\_1['input\_add'] = data\_1['list\_of\_input'].apply(lambda x: x[0]['prev\_address'])

data\_1['output\_add'] = data\_1['list\_of\_output'].apply(lambda x: x[0]['next\_address'] if len(x) else None)

#сумма текущей транзакции

#считаем current value по input'у

data\_1['curr\_value'] = data\_1['list\_of\_input'].apply(lambda x: x[0]['curr\_value'])

#общая сумма предыдущей и следующей транзакций (не сколько на адрес пришло, а именно всей транзакции целиком)

data\_1['tot\_prev\_tx\_value'] = data\_1['list\_of\_input'].apply(lambda x: x[0]['total\_prev\_val'])

data\_1['tot\_next\_tx\_value'] = data\_1['list\_of\_output'].apply(lambda x: x[0]['total\_next\_val'] if len(x) else None)

#дата совершения предыдущей и следующей транзакций

data\_1['prev\_time'] = data\_1['list\_of\_input'].apply(lambda x: x[0]['prev\_time'])

data\_1['next\_time'] = data\_1['list\_of\_output'].apply(lambda x: x[0]['next\_time'] if len(x) else None)

#задержка относительно предыдущей и следующей транзакций

data\_1['diff\_prev\_time'] = data\_1['time'] - data\_1['prev\_time']

data\_1['diff\_next\_time'] = data\_1['next\_time'] - data\_1['time']

data\_1\_upd = data\_1[cols]

#transaction\_type = 2

data\_2 = data[data['transaction\_type'] == 2].reset\_index(drop=True)

#convert input adds

tmp\_inp = data\_2.list\_of\_input\

.apply(pd.Series).merge(data\_2[['txid','list\_of\_input']], left\_index = True, right\_index = True)\

.drop(['list\_of\_input'], axis = 1)\

.melt(id\_vars = ['txid'], value\_name = 'input\_add')\

.drop('variable', axis = 1).dropna().reset\_index(drop=True)

tmp\_inp = tmp\_inp['input\_add'].apply(pd.Series)\

.merge(tmp\_inp[['txid','input\_add']], left\_index = True, right\_index = True)\

.drop(['input\_add'], axis = 1)\

.rename({'prev\_address': 'input\_add','total\_prev\_val': 'tot\_prev\_tx\_value'}, axis=1)

#вытаскиваем исходящий адрес

data\_2['output\_add'] = data\_2['list\_of\_output'].apply(lambda x: x[0]['next\_address'] if len(x) else None)

data\_2['tot\_next\_tx\_value'] = data\_2['list\_of\_output'].apply(lambda x: x[0]['total\_next\_val'] if len(x) else None)

#дата совершения следующей транзакции

data\_2['next\_time'] = data\_2['list\_of\_output'].apply(lambda x: x[0]['next\_time'] if len(x) else None)

data\_2 = data\_2.set\_index('txid')

tmp\_inp = tmp\_inp.set\_index('txid')

cols\_to\_take = ['output\_add', 'tot\_next\_tx\_value', 'next\_time', 'confirmations',

'time', 'is\_coinbase', 'is\_double\_spend', 'is\_sw\_tx', 'weight', 'vsize',

'inputs\_count', 'outputs\_count', 'inputs\_value', 'outputs\_value', 'max\_prev\_val',

'min\_prev\_val', 'avg\_prev\_val', 'max\_next\_val', 'min\_next\_val', 'avg\_next\_val',

'diff\_max\_prev\_time', 'diff\_min\_prev\_time', 'diff\_avg\_prev\_time', 'diff\_max\_next\_time',

'diff\_min\_next\_time', 'diff\_avg\_next\_time', 'is\_suspicious', 'transaction\_type']

data\_2\_upd = tmp\_inp.join(data\_2[cols\_to\_take], how='left').reset\_index()

#задержка относительно предыдущей и следующей транзакций

data\_2\_upd['diff\_prev\_time'] = data\_2\_upd['time'] - data\_2\_upd['prev\_time']

data\_2\_upd['diff\_next\_time'] = data\_2\_upd['next\_time'] - data\_2\_upd['time']

data\_2\_upd = data\_2\_upd[cols]

#transaction\_type = 3

data\_3 = data[data['transaction\_type'] == 3].reset\_index(drop=True)

#convert input adds

tmp\_outp = data\_3.list\_of\_output\

.apply(pd.Series).merge(data\_3[['txid','list\_of\_output']], left\_index = True, right\_index = True)\

.drop(['list\_of\_output'], axis = 1)\

.melt(id\_vars = ['txid'], value\_name = 'output\_add')\

.drop('variable', axis = 1).dropna().reset\_index(drop=True

tmp\_outp = tmp\_outp['output\_add'].apply(pd.Series)\

.merge(tmp\_outp[['txid','output\_add']], left\_index = True, right\_index = True)\

.drop(['output\_add'], axis = 1)\

.rename({'next\_address': 'output\_add','total\_next\_val': 'tot\_next\_tx\_value'}, axis=1)

#вытаскиваем исходящий адрес

data\_3['input\_add'] = data\_3['list\_of\_input'].apply(lambda x: x[0]['prev\_address'])

#вытаскиваем общую сумму следующей транзакции

data\_3['tot\_prev\_tx\_value'] = data\_3['list\_of\_input'].apply(lambda x: x[0]['total\_prev\_val'])

#дата совершения следующей транзакции

data\_3['prev\_time'] = data\_3['list\_of\_input'].apply(lambda x: x[0]['prev\_time'])

data\_3 = data\_3.set\_index('txid')

tmp\_outp = tmp\_outp.set\_index('txid')

cols\_to\_take = ['input\_add', 'tot\_prev\_tx\_value', 'prev\_time', 'confirmations',

'time', 'is\_coinbase', 'is\_double\_spend', 'is\_sw\_tx', 'weight', 'vsize',

'inputs\_count', 'outputs\_count', 'inputs\_value', 'outputs\_value', 'max\_prev\_val',

'min\_prev\_val', 'avg\_prev\_val', 'max\_next\_val', 'min\_next\_val', 'avg\_next\_val',

'diff\_max\_prev\_time', 'diff\_min\_prev\_time', 'diff\_avg\_prev\_time', 'diff\_max\_next\_time',

'diff\_min\_next\_time', 'diff\_avg\_next\_time', 'is\_suspicious', 'transaction\_type']

data\_3\_upd = tmp\_outp.join(data\_3[cols\_to\_take], how='left').reset\_index()

#задержка относительно предыдущей и следующей транзакций

data\_3\_upd['diff\_prev\_time'] = data\_3\_upd['time'] - data\_3\_upd['prev\_time']

data\_3\_upd['diff\_next\_time'] = data\_3\_upd['next\_time'] - data\_3\_upd['time']

data\_3\_upd = data\_3\_upd[cols]

#transaction\_type = 4

data\_4 = data[data['transaction\_type'] == 4].reset\_index(drop=True)

#convert input adds

tmp\_inp = data\_4.list\_of\_input\

.apply(pd.Series).merge(data\_4[['txid','list\_of\_input']], left\_index = True, right\_index = True)\

.drop(['list\_of\_input'], axis = 1)\

.melt(id\_vars = ['txid'], value\_name = 'input\_add')\

.drop('variable', axis = 1).dropna().reset\_index(drop=True)

tmp\_inp = tmp\_inp['input\_add'].apply(pd.Series)\

.merge(tmp\_inp[['txid','input\_add']], left\_index = True, right\_index = True)\

.drop(['input\_add'], axis = 1)\

.rename({'prev\_address': 'input\_add','total\_prev\_val': 'tot\_prev\_tx\_value'}, axis=1)

tmp\_inp['output\_add'] = tmp\_inp['txid']

#convert output adds

tmp\_outp = data\_4.list\_of\_output\

.apply(pd.Series).merge(data\_4[['txid','list\_of\_output']], left\_index = True, right\_index = True)\

.drop(['list\_of\_output'], axis = 1)\

.melt(id\_vars = ['txid'], value\_name = 'output\_add')\

.drop('variable', axis = 1).dropna().reset\_index(drop=True)

tmp\_outp = tmp\_outp['output\_add'].apply(pd.Series)\

.merge(tmp\_outp[['txid','output\_add']], left\_index = True, right\_index = True)\

.drop(['output\_add'], axis = 1)\

.rename({'next\_address': 'output\_add','total\_next\_val': 'tot\_next\_tx\_value'}, axis=1)

tmp\_outp['input\_add'] = tmp\_outp['txid']

data\_4['tot\_next\_tx\_value'] = data\_4['avg\_next\_val']

data\_4['tot\_prev\_tx\_value'] = data\_4['avg\_prev\_val']

data\_4['next\_time'] = data\_4['avg\_next\_time']

data\_4['prev\_time'] = data\_4['avg\_prev\_time']

data\_4 = data\_4.set\_index('txid')

tmp\_inp = tmp\_inp.set\_index('txid')

tmp\_outp = tmp\_outp.set\_index('txid')

cols\_to\_take\_1 = ['tot\_next\_tx\_value', 'next\_time', 'confirmations',

'time', 'is\_coinbase', 'is\_double\_spend', 'is\_sw\_tx', 'weight', 'vsize',

'inputs\_count', 'outputs\_count', 'inputs\_value', 'outputs\_value', 'max\_prev\_val',

'min\_prev\_val', 'avg\_prev\_val', 'max\_next\_val', 'min\_next\_val', 'avg\_next\_val',

'diff\_max\_prev\_time', 'diff\_min\_prev\_time', 'diff\_avg\_prev\_time', 'diff\_max\_next\_time',

'diff\_min\_next\_time', 'diff\_avg\_next\_time', 'is\_suspicious', 'transaction\_type']

cols\_to\_take\_2 = ['tot\_prev\_tx\_value', 'prev\_time', 'confirmations',

'time', 'is\_coinbase', 'is\_double\_spend', 'is\_sw\_tx', 'weight', 'vsize',

'inputs\_count', 'outputs\_count', 'inputs\_value', 'outputs\_value', 'max\_prev\_val',

'min\_prev\_val', 'avg\_prev\_val', 'max\_next\_val', 'min\_next\_val', 'avg\_next\_val',

'diff\_max\_prev\_time', 'diff\_min\_prev\_time', 'diff\_avg\_prev\_time', 'diff\_max\_next\_time',

'diff\_min\_next\_time', 'diff\_avg\_next\_time', 'is\_suspicious', 'transaction\_type']

data\_4\_upd\_1 = tmp\_inp.join(data\_4[cols\_to\_take\_1], how='left').reset\_index()

data\_4\_upd\_2 = tmp\_outp.join(data\_4[cols\_to\_take\_2], how='left').reset\_index()

#задержка относительно предыдущей и следующей транзакций

data\_4\_upd\_1['diff\_prev\_time'] = data\_4\_upd\_1['time'] - data\_4\_upd\_1['prev\_time']

data\_4\_upd\_1['diff\_next\_time'] = data\_4\_upd\_1['next\_time'] - data\_4\_upd\_1['time']

#задержка относительно предыдущей и следующей транзакций

data\_4\_upd\_2['diff\_prev\_time'] = data\_4\_upd\_2['time'] - data\_4\_upd\_2['prev\_time']

data\_4\_upd\_2['diff\_next\_time'] = data\_4\_upd\_2['next\_time'] - data\_4\_upd\_2['time']

data\_4\_upd\_1 = data\_4\_upd\_1[cols]

data\_4\_upd\_2 = data\_4\_upd\_2[cols]

#Cобираем все вместе

print(f'data\_1\_upd shape: {data\_1\_upd.shape}')

print(f'data\_2\_upd shape: {data\_2\_upd.shape}')

print(f'data\_3\_upd shape: {data\_3\_upd.shape}')

print(f'data\_4\_upd\_1 shape: {data\_4\_upd\_1.shape}')

print(f'data\_4\_upd\_2 shape: {data\_4\_upd\_2.shape}')

result\_df = pd.concat([data\_1\_upd, data\_2\_upd, data\_3\_upd, data\_4\_upd\_1, data\_4\_upd\_2]).reset\_index(drop=True)

#пометим искуственно сгенерированные inputs and outputs (когда связь многие ко многим, я создаю доп.узел)

result\_df['true\_input'] = result\_df['input\_add'].apply(lambda x: 1 if len(x)<=40 else 0)

result\_df['true\_output'] = result\_df['output\_add'].apply(lambda x: 0 if not x or len(x)>40 else 1)

#метка, если адрес в транзакии и входящиц и исходящий

result\_df.loc[result\_df.input\_add == result\_df.output\_add, 'the\_same\_input\_output'] = int(1)

result\_df.loc[result\_df.input\_add != result\_df.output\_add, 'the\_same\_input\_output'] = int(0)

aggregation = {

'is\_coinbase': {

'sum\_is\_coinbase': 'sum'

},

'is\_double\_spend': {

'sum\_is\_double\_spend': 'sum'

},

'is\_sw\_tx': {

'sum\_is\_sw\_tx': 'sum'

},

'txid': {

'count\_tx': 'count'

},

'transaction\_type': {

'nunique\_transaction\_type': 'nunique'

},

'the\_same\_input\_output': {

'sum\_the\_same\_input\_output': 'sum'

},

'weight': {

'min\_weight': 'min',

'max\_weight': 'max',

'avg\_weight': 'mean',

},

'vsize': {

'min\_vsize': 'min',

'max\_vsize': 'max',

'avg\_vsize': 'mean',

},

'inputs\_count':{

'min\_inputs\_count': 'min',

'max\_inputs\_count': 'max',

'avg\_inputs\_count': 'mean',

},

'outputs\_count':{

'min\_outputs\_count': 'min',

'max\_outputs\_count': 'max',

'avg\_outputs\_count': 'mean',

},

'inputs\_value':{

'min\_inputs\_value': 'min',

'max\_inputs\_value': 'max',

'avg\_inputs\_value': 'mean',

'sum\_inputs\_value': 'sum',

},

'max\_prev\_val':{

'min\_max\_prev\_val': 'min',

'max\_max\_prev\_val': 'max',

'avg\_max\_prev\_val': 'mean'

},

'min\_prev\_val':{

'min\_min\_prev\_val': 'min',

'max\_min\_prev\_val': 'max',

'avg\_min\_prev\_val': 'mean'

},

'avg\_prev\_val':{

'min\_avg\_prev\_val': 'min',

'max\_avg\_prev\_val': 'max',

'avg\_avg\_prev\_val': 'mean'

},

'max\_next\_val':{

'min\_max\_next\_val': 'min',

'max\_max\_next\_val': 'max',

'avg\_max\_next\_val': 'mean'

},

'min\_next\_val':{

'min\_min\_next\_val': 'min',

'max\_min\_next\_val': 'max',

'avg\_min\_next\_val': 'mean'

},

'avg\_next\_val':{

'min\_avg\_next\_val': 'min',

'max\_avg\_next\_val': 'max',

'avg\_avg\_next\_val': 'mean'

},

'diff\_max\_prev\_time':{

'min\_diff\_max\_prev\_time': 'min',

'max\_diff\_max\_prev\_time': 'max',

'avg\_diff\_max\_prev\_time': 'mean'

},

'diff\_min\_prev\_time':{

'min\_diff\_min\_prev\_time': 'min',

'max\_diff\_min\_prev\_time': 'max',

'avg\_diff\_min\_prev\_time': 'mean'

},

'diff\_avg\_prev\_time':{

'min\_diff\_avg\_prev\_time': 'min',

'max\_diff\_avg\_prev\_time': 'max',

'avg\_diff\_avg\_prev\_time': 'mean'

},

'diff\_max\_next\_time':{

'min\_diff\_max\_next\_time': 'min',

'max\_diff\_max\_next\_time': 'max',

'avg\_diff\_max\_next\_time': 'mean'

},

'diff\_min\_next\_time':{

'min\_diff\_min\_next\_time': 'min',

'max\_diff\_min\_next\_time': 'max',

'avg\_diff\_min\_next\_time': 'mean'

},

'diff\_avg\_next\_time':{

'min\_diff\_avg\_next\_time': 'min',

'max\_diff\_avg\_next\_time': 'max',

'avg\_diff\_avg\_next\_time': 'mean'

},

'tot\_prev\_tx\_value':{

'min\_tot\_prev\_tx\_value': 'min',

'max\_tot\_prev\_tx\_value': 'max',

'avg\_tot\_prev\_tx\_value': 'mean'

},

'tot\_next\_tx\_value':{

'min\_tot\_next\_tx\_value': 'min',

'max\_tot\_next\_tx\_value': 'max',

'avg\_tot\_next\_tx\_value': 'mean'

},

'diff\_prev\_time':{

'min\_diff\_prev\_time': 'min',

'max\_diff\_prev\_time': 'max',

'avg\_diff\_prev\_time': 'mean'

},

'diff\_next\_time':{

'min\_diff\_next\_time': 'min',

'max\_diff\_next\_time': 'max',

'avg\_diff\_next\_time': 'mean'

}

}

%%time

input\_agg = result\_df[result\_df['true\_input']==1].groupby(['input\_add'])\

.agg(aggregation)\

.reset\_index()

%%time

output\_agg = result\_df[result\_df['true\_output']==1].groupby(['output\_add'])\

.agg(aggregation)\

.reset\_index()

new\_cols = ['add']+[x[1] for x in list(input\_agg.columns[1:])]

input\_agg\_v2 = pd.DataFrame(input\_agg.values, columns=new\_cols).set\_index('add')

output\_agg\_v2 = pd.DataFrame(output\_agg.values, columns=new\_cols).set\_index('add')

fin = input\_agg\_v2.join(output\_agg\_v2, how='left', rsuffix='\_o', lsuffix='\_i').reset\_index()

# ПРИЛОЖЕНИЕ В

Программный код для проведения кластеризации

#Берем только те адреса, для которых скачена вся история транзакций

with open('./finished\_adds', 'r') as f:

finished\_adds = f.read()

finished\_adds = re.sub('\"','',finished\_adds[1:-1]).split(',')

fin['take\_for\_clustring'] = fin['add'].apply(lambda x: 1 if x in finished\_adds else 0)

train = fin[fin['take\_for\_clustring']==1].set\_index('add')

scaler = MinMaxScaler(feature\_range=(0,1))

feats = train.columns

train\_sc = scaler.fit\_transform(train[feats])

train\_sc = pd.DataFrame(train\_sc, columns=feats, index=train.index)

inertia = []

for k in range(1, 11):

kmeans = KMeans(n\_clusters = k, random\_state=1).fit(train\_sc)

inertia.append(np.sqrt(kmeans.inertia\_)

x = list(range(1, 11))

y = inertia

plt.figure(figsize=(10,8));

plt.plot(x, y, marker='.');

plt.xlabel('Number of clusters', fontsize=15)

plt.ylabel('Inertia', fontsize=15);

#проводим кластеризацию методом K-means

kmeans = KMeans(n\_clusters=9, random\_state=17)

kmeans.fit(train\_sc);

simple\_df = train\_sc

simple\_df['kmeans'] = kmeans.labels\_

means = simple\_df.groupby(['kmeans']).mean()

plt.figure(figsize=(20,17))

for n in range(means.shape[0]):

plt.plot(feats, means.iloc[n, :].values, marker='o', label='cluster {}'.format(n))

plt.legend(loc = 'upper left')

plt.title('K-Means method')

plt.xticks(rotation=90, ha="right");

simple\_df['kmeans'].value\_counts()

simple\_df = simple\_df.reset\_index()

simple\_df['is\_suspicious'] = simple\_df['add'].apply(lambda x: 1 if x in addr\_list else 0)

simple\_df[simple\_df['is\_suspicious']==1][['add','is\_suspicious','kmeans']]