МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

*А.В. БЛАГОВ, И.А. РЫЦАРЕВ*

# АНАЛИЗ СОЦИАЛЬНЫХ СЕТЕЙ

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве учебного пособия для обучающихся по основной образовательной программе высшего образования по направлению подготовки 01.04.02 Прикладная математика и информатика

С А М А Р А
Издательство Самарского университета
2020

УДК 004.738.5(075)+316.472(075)
ББК 32.937.202я7+60.56я7
Б681

Рецензенты: д-р физ.-мат. наук, проф. СамГТУ А. И. Ж д а н о в,
канд. техн. наук, доц. Самарского университета
П. Ю. Я к и м о в

***Благов, Александр Владимирович***

**Б681 Анализ социальных сетей:** учебное пособие / *А.В. Благов, И.А. Рыцарев*. – Самара: Издательство Самарского университета, 2020. – 104 с.: ил.

**ISBN 978-5-7883-1556-0**

В данном пособии рассмотрены особенности анализа данных, а также связей в социальных сетях. Проведен обзор существующих инструментов по сбору, обработке и анализу данных. Исследованы вопросы визуализации, кластерного анализа и моделирования социальных сетей. Рассмотрены методы и алгоритмы анализа текстовой информации.

В целях практического применения описанных сведений в пособии приводится цикл разработанных лабораторных работ.

Предназначено для обучающихся по направлению подготовки 01.04.02 Прикладная математика и информатика.

Подготовлено на кафедре технической кибернетики.

УДК 004.738.5(075)+316.472(075)
ББК 32.937.202я7+60.56я7

ISBN 978-5-7883-1556-0 

## ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

В современном мире одним из наиболее активно развивающихся направлений в информационных технологиях в настоящее время являются, так называемые, «большие данные» или BigData. Данное понятие, как правило, применимо для данных колоссального объёма, для которых традиционные методы и программные средства их обработки являются неприемлемыми. Термин «большие данные» используется практически повсеместно: в государственном и частном секторе, науке и технологии, здравоохранении и социальных сетях.

Аналитическое агентство Gartner в 2012 году опубликовало отчёт под названием "Цикл ажиотажа для развивающихся технологий". Согласно отчёту, технологии "Социальная аналитика" и "Большие данные" находились на так называемом "пике завышенных ожиданий". В настоящее время их актуальность и востребованность только возросли. В частности, исследованиями социальных данных активно занимаются такие университеты как Стэнфорд, Оксфорд, а также компании Google, Amazon, Facebook, Yahoo!, LinkedIn и многие другие. Компании-владельцы сервисов онлайновых социальных сетей (Facebook, Twitter) активно инвестируют в разработку усовершенствованных инфраструктурных и алгоритмических (новые алгоритмы поиска и рекомендации пользователей, товаров и услуг) решений для обработки больших массивов пользовательских данных. Появляются и успешно развиваются коммерческие компании, предоставляющие услуги по доступу к хранилищам социальных данных, сбору социальных данных по заданным сценариям, социальной аналитике, а также расширению существующих платформ с помощью социальных данных.

Анализ социальных данных стремительно набирает популярность во всём мире благодаря появлению в начале 2000-х онлайновых сервисов социальных сетей (Facebook, Twitter, YouTube, ВКонтакте и другие). С этим связан феномен социализации персональных данных: стали публично доступными события и факты биографии, переписка, дневники, фото-, видео-, аудиоматериалы, заметки о путешествиях и т.д. Таким образом, социальные сети являются уникальным источником данных о личной жизни и интересах реальных людей. Это открывает беспрецедентные возможности для решения исследовательских и бизнес-задач (многие из которых до этого невозможно было решать эффективно из-за недостатка данных), а также создания вспомогательных сервисов и приложений для пользователей социальных сетей. Кроме того, этим обуславливается повышенный интерес к сбору и анализу социальных данных со стороны компаний и исследовательских центров.

За последнее десятилетие социальные сети стали играть огромную роль в жизни общества. Они, будучи предметом социализации людей, занимают одну из лидирующих позиций по производству «больших данных». Возможность выкладывать и делиться сообщениями, фотографиями, музыкой, видео с друзьями, возможность создавать и проводить различные мероприятия, возможность продвижения бизнеса – всё это колоссальные объёмы постоянно генерирующихся, устаревающих, обновляющихся данных.

Значимость социальных сетей также обусловлена тем, что они являются наиболее мощным и доступным политическим, идеологическим и экономическим инструментом.

Таким образом, специалисты из исследовательских центров и компаний по всему миру используют данные социальных сетей для

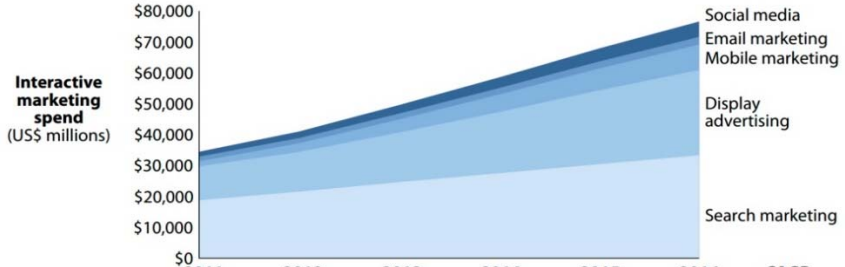
моделирования социальных, экономических, политических и других процессов от персонального до государственного уровня с целью разработки механизмов воздействия на эти процессы, а также создания инновационных аналитических и бизнес-приложений и сервисов.

С другой стороны социальные сети становятся все более привлекательной средой для реализации программ маркетинга. Однако все же руководители маркетинговых служб и специалисты-практики никак не могут оценить эффективность этого канала и сравнить результаты маркетинга в социальных сетях с другими маркетинговыми программами в сети Интернет. Особенно сложно, например, определить охват и воздействие социальных сетей и понять, как маркетинг в социальных сетях влияет на ключевые показатели бизнеса, такие как объем продаж, число заказов и количество событий конверсии посетителей в потребителей. Не зная, каким образом маркетинг в социальных сетях изменяет итоговые показатели прибыли, специалисты в области маркетинга рискуют принимать решения о программе мероприятий маркетинга и бюджетных ассигнованиях вслепую.

Необходимо понимать, что социальная сеть направлена на построение сообществ в Интернете из людей со схожими интересами и/или деятельностью. Связь осуществляется посредством сервиса внутренней почты или мгновенного обмена сообщениями. И отыскание пользователей социальной сети с необходимыми интересами становится практически важным для многих компаний, ведущих коммерческую деятельность.

Важно уметь измерять результаты, так как в целом ряде отраслей маркетологи увеличивают долю расходов на маркетинг в социальных сетях. По данным Forrester Research, в США расходы на социальный маркетинг в период с 2011 по 2016 гг. возросли

более чем вдвое и составили порядка 5 млрд. долларов США. Социальные сети с совокупными среднегодовыми темпами роста 26% – второй из самых быстро развивающихся маркетинговых каналов после мобильного, как показывает диаграмма на рис. 1.




| | 2011 | 2012 | 2013 | 2014 | 2015 | 2016 | CAGR |
|---|---|---|---|---|---|---|---|
| Social media | $1,590 | $2,119 | $2,760 | $3,453 | $4,217 | $4,995 | 26% |
| Email marketing | $1,510 | $1,694 | $1,875 | $2,066 | $2,262 | $2,468 | 10% |
| Mobile marketing | $1,652 | $2,777 | $4,238 | $5,697 | $7,057 | $8,237 | 38% |
| Display advertising | $10,949 | $12,860 | $16,085 | $19,783 | $23,919 | $27,600 | 20% |
| Search marketing | $18,756 | $21,553 | $24,613 | $27,515 | $30,433 | $33,319 | 12% |
| **Total** | **$34,457** | **$41,003** | **$49,571** | **$58,515** | **$67,888** | **$76,620** | 17% |
| Percent of all ad spend | 16% | 18% | 20% | 22% | 24% | 26% | |

Рис. 1. Динамика развития маркетинговых каналов в США

С учётом колоссального роста объёмов мобильных устройств (смартфоны, планшеты и т.д.), а также разработки приложений для использования социальных сетей, количество массивов данных социальных сетей возрастает с каждым днём ещё в большей степени. Отсюда следует, что и оказываемое влияние социальных сетей практически на все сферы деятельности человека также растёт, что обуславливает актуальность анализа данных социальных сетей.

# 1. СОЦИАЛЬНЫЕ СЕТИ И ИСПОЛЬЗОВАНИЕ ИХ ДАННЫХ

В настоящее время используется большое множество социальных сетей, самыми популярными из них являются: Facebook, Twitter, LinkedIn, Instagram, ВКонтакте и т.д. Наибольший интерес при этом представляют следующие данные: контент и связи в социальных сетях.

## 1.1. Контент социальных сетей

В части контента имеет место следующее. Данные социальных сетей в основном являются неструктурированными и представляют собой различную сущность. Социальные сети можно условно классифицировать по типу данных следующим образом:

- содержащие преимущественно текстовые данные – Twitter, Telegram, LiveJournal и т.д.;
- содержащие преимущественно изображения – Instagram, Snapchat и т.д.;
- содержащие преимущественно видео – YouTube, Vimeo и т.д.;
- содержащие данные смешанного типа видео – FaceBook, ВКонтакте и т.д.

Для анализа различных типов данных существуют различные подходы и методы. Зачастую неструктурированные «сырые» данные подлежат предварительной обработке. Схема работы с данными социальных сетей в обобщенном виде представлена на рис. 2.

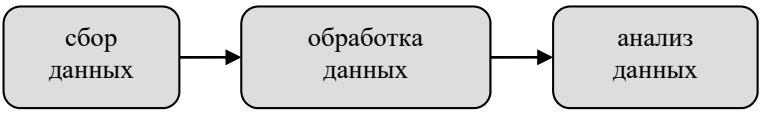



Рис. 2. Схема обработки неструктурированных данных

Первый этап работы с данными социальных сетей обусловлен, прежде всего, наличием необходимого инструментария, а также возможностью предоставления сбора (открытого API) самой социальной сетью. Сбор может осуществляться как уже размещенных в социальных сетях данных, так и в режиме on-line. При этом еще на этапе сбора можно ставить определенные фильтры, к примеру, собрать лишь те сообщения, в которых присутствует то или иное слово или хештег.

Такие компании как IBM, Cloudera, Apache Software Foundation, Hortonworks и другие предоставляют различные платформы для сбора и обработки данных.

Объём собранных (собираемых) данных может быть в достаточной степени велик. Более того, потоковые данные, полученные из социальных сетей обычно содержат в себе множество служебной информации. При этом для дальнейшего анализа бывают важны лишь те данные, которые представляют интерес, поэтому необходимо отделить служебную информацию от нужной. В описанных выше платформах, как правило, используется технология, основанная на модели распределенных вычислений MapReduce. С помощью этой технологии производится структуризация путем компоновки и исключения служебных и не представляющих практический интерес данных. В подходе, основанном на данной модели, производятся вычисления некоторых наборов распределенных задач с использованием большого количества компьютеров (называемых «нодами»), образующих кластер. Работа MapReduce состоит из двух шагов: Map и Reduce. На Map-шаге происходит предварительная обработка входных данных. Для этого один из компьютеров (называемый главным узлом — master node) получает входные данные задачи, разделяет их на части и передает другим компьютерам (рабочим узлам — worker node) для

предварительной обработки. Название данный шаг получил от одноименной функции высшего порядка.

На Reduce-шаге происходит свёртка предварительно обработанных данных. Главный узел получает ответы от рабочих узлов и на их основе формирует результат – решение задачи, которая изначально формулировалась. Пример работы технологии MapReduce показан на рис. 3.

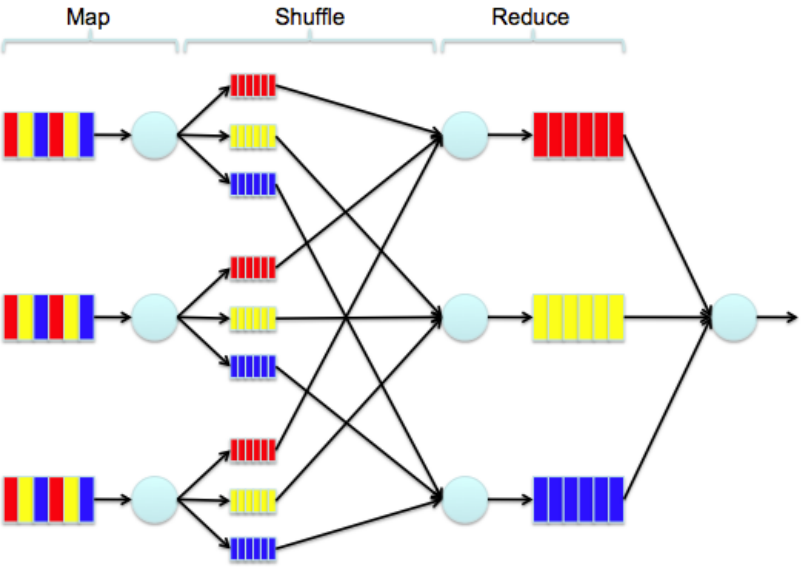



Рис. 3. Алгоритм работы технологии MapReduce

Преимущество MapReduce заключается в том, что она позволяет распределено производить операции предварительной обработки и свёртки. Операции предварительной обработки работают независимо друг от друга и могут производиться параллельно (хотя на практике это ограничено источником входных данных и/или количеством используемых процессоров). Аналогично множество рабочих узлов могут осуществлять свёртку – для этого необходимо,

чтобы все результаты предварительной обработки с одним конкретным значением ключа обрабатывались одним рабочим узлом в один момент времени. Хотя этот процесс может быть менее эффективным по сравнению с более последовательными алгоритмами, технология MapReduce может быть применена к большим объёмам данных, которые могут обрабатываться большим количеством серверов. Так, MapReduce может быть использована для сортировки петабайта данных, что займёт всего лишь несколько часов. Параллелизм также даёт некоторые возможности восстановления после частичных сбоев серверов: если в рабочем узле, производящем операцию предварительной обработки или свёртки, возникает сбой, то его работа может быть передана другому рабочему узлу (при условии, что входные данные для проводимой операции доступны).

Обработанные и структурированные данные анализируются в соответствие с их типом. Отдельные примеры подобной аналитики представлены в следующих разделах данного учебного пособия.

### 1.2. Связи в социальных сетях

Не менее важными, чем контент, данными являются связи в социальных сетях. Они могут характеризовать знакомство, влияние, а также принадлежность тому или иному сообществу. Исследуя связи в социальных сетях можно решать задачи определения предпочтений и интересов пользователей. Это может, в том числе, использоваться в работе ряда рекомендательных систем.

Социальные сети, в части связей, могут быть обычно представлены в виде графов (рис. 4). Это обусловлено тем, что аккаунты в социальных сетях отождествляются с вершинами в графе, а связи – с рёбрами, при этом в зависимости от вида связи

(дружба, либо подписка) граф может быть, как неориентированным, так и ориентированным.

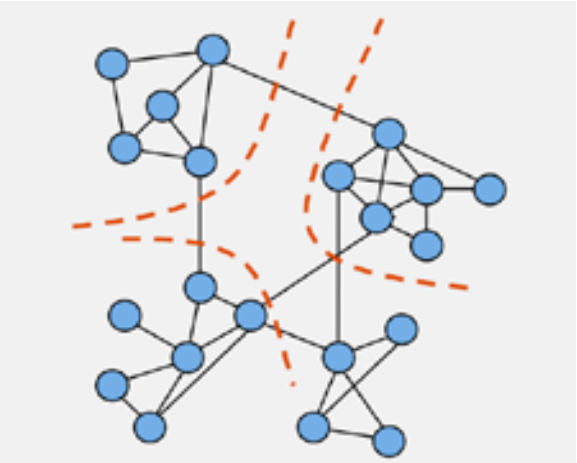


Рис. 4. Фрагмент социальной сети, описаный графом

С помощью инструментов теории графов можно решать различные задачи, такие как, например:

- определение сообществ;
- принадлежность к конкретному сообществу;
- относительная роль узлов и рёбер в графе и т.д.

Граф, описывающий социальную сеть или ее фрагмент, ввиду значительного количества профилей и связей между ними, является графом большой размерности (рис. 5). При этом актуальной задачей становится визуализация такого графа, возможная сегментация, либо кластеризация, позволяющая, в том числе уменьшить, его размерность для более удобного восприятия, но без потерь значимой информации.

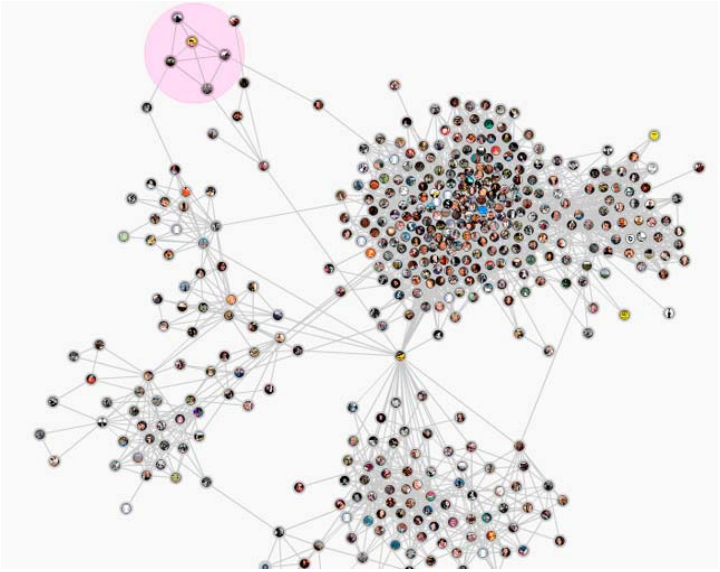


Рис. 5. Фрагмент социальной сети, описаный графом
большой размерности

Кластеризация графа большой размерности также может быть использована для отыскания сетевых сообществ, с возможной дальнейшей их классификацией.

# 2. СБОР ДАННЫХ, ИНСТРУМЕНТЫ ПО СБОРУ И ОБРАБОТКЕ ДАННЫХ

Для проведения анализа любых данных, в том числе полученных из социальных сетей, необходимо организовать их сбор и дальнейшую обработку.

В настоящее время существует множество программных инструментов для сбора данных. Многие ведущие компании мира, такие как IBM, Oracle, Microsoft и другие имеют свои решения в этой области. Из свободных и сравнительно недорогих чаще всего используются следующие продукты:

- 1010data;
- Apache Hadoop;
- Apache Ambari;
- Jaspersoft;
- LixisNexis Risj Sokutions HPCC Systems;
- Revolution Analytics (на базе языка R для мат. статистики);
- Hortonworks;
- Biginsights;
- Cloudera;
- Teradata и др.

Многие из них имеют модули для работы с данными социальных сетей. К примеру, используя решения от Apache Ambari, IBM Biginsights, Hortonworks, либо Cloudera можно довольно просто организовать потоковый on-line сбор данных социальной сети Twitter в режиме реального времени по любой выбранной геолокации.

Общая схема сбора потоковых данных в режиме реального времени (на примере модуля Flume решения Apache Ambari) описывается следующим образом. Поток начинается с клиента (приложения, передающего данные), который передает событие

(набор данных) агенту (независимый процесс, в котором осуществляется хранение событий, и выполняются такие компоненты, как источники, каналы и стоки). Источник (интерфейс, принимающий сообщения через различные протоколы передачи данных), получивший событие, передает его на один или несколько каналов. Из каналов событие передается на стоки, входящие в состав того же агента. Он может передать ее другому агенту, или (если это конечный агент) – на узел назначения.

Так как источник может передавать события на несколько каналов, потоки могут направляться на несколько узлов назначения. Это наглядно показано на рис. 6: Агент считывает событие в два канала (Канал 1 и Канал 2), и затем передает их в независимые стоки.

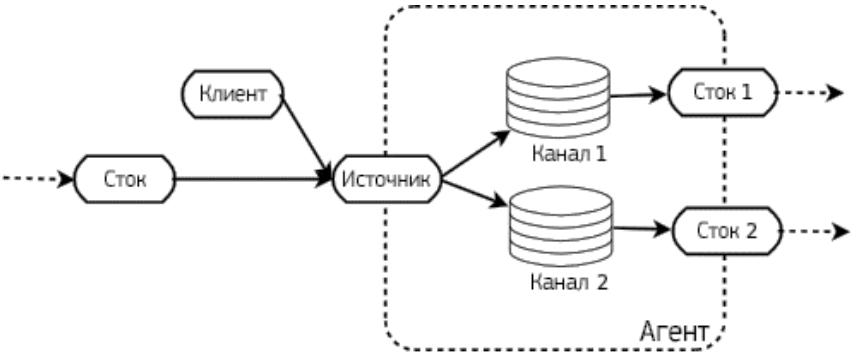



Рис. 6. Схема работы потока в модуле Flume

Несколько потоков можно объединить в один. Для этого нужно, чтобы несколько источников в составе одного и того же агента передавали данные на один и тот же канал. Схема взаимодействия компонентов при объединении потоков показана на рис. 7. В этом случае каждый из трех агентов, включающий несколько источников, передает данные на один и тот же канал и затем на сток.

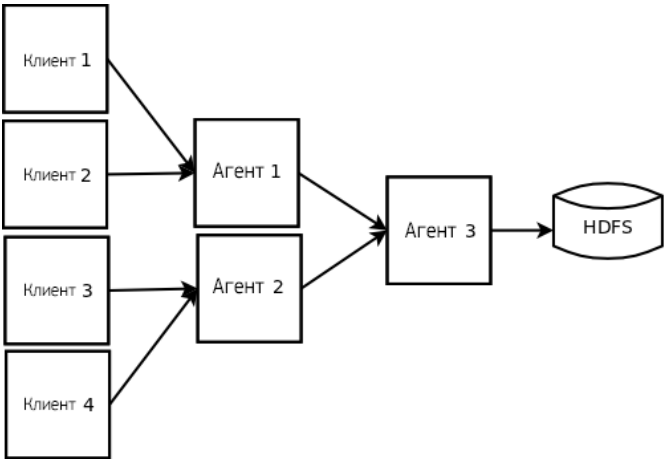



Рис. 7. Схема взаимодействия компонентов при объединении потоков в модуле

Передача данных между источниками и каналами, а также между агентами, осуществляется с использованием транзакций, благодаря чему обеспечивается сохранность данных.

Обработка ошибок также осуществляется на базе транзакционного механизма. Когда поток проходит через несколько различных агентов, и при прохождении возникают проблемы связи, события буферизуются на последнем доступном в потоке агенте. Более наглядно схема обработки ошибок представлена на рис. 8.

Обработка ошибок в потоке происходит следующим образом:

- события движутся от клиента к центральному хранилищу без проблем связи;

- возникает проблема связи на участке между Агентом 2 и центральным хранилищем, тогда события буферизуются на Агенте 2;

- после устранения проблем связи события, буферизованные на Агенте 2, отправляются в центральное хранилище.

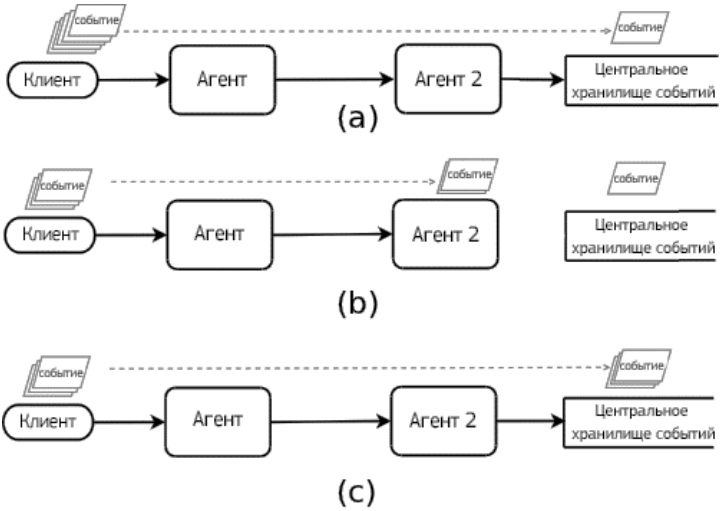



Рис. 8. Схема обработки ошибок в модуле Flume

Потоковые данные, полученные из социальных сетей, содержат в себе множество служебной информации. Для дальнейшего анализа важны лишь те данные, которые представляют интерес, поэтому необходимо отделить служебную информацию от нужной.

Структуризация путем компоновки и исключения служебных и не представляющих практический интерес данных может быть произведена с помощью описанной выше в первом разделе технологии MapReduce, реализованной во многих программных решениях, либо с помощью иных разработанных методов и алгоритмов. Как правило, MapReduce может быть применен к большим объёмам данных, которые могут обрабатываться большим количеством серверов. Например, эта технология может быть использована для сортировки петабайта данных, что займет лишь несколько часов. Также параллельная работа операций обработки и операции свертки дает возможность восстановления после частичных сбоев в работе серверов. Например, если в рабочем узле, производящем операцию предварительной обработки или свёртки,

возникает сбой, то его работа может быть делегирована другому рабочему узлу, при условии, что входные данные для данной проводимой операции доступны.

В социальной сети Twitter собранные текстовые данные состоят из сведений о шрифтах, размерах текста, ссылок на профиль пользователя и многих других. Однако, для анализа зачастую требуются лишь некоторые из них, а именно, поля sentiment, location, text, language и, возможно, некоторые другие. В этой связи обработку данных с целью извлечения необходимой информации можно организовать как на уровне сбора этих данных, наложив определенные фильтры на каналы передачи данных, так и после на этапе обработки.

Необходимой представляющей интерес информацией, например, могут быть следующие параметры.

Общее количество «твиттов» $K_j$ для каждой L локации (географической зоны):

$$K_L = \sum_i (k_i \in L),$$

где $k_\text{i}$ – каждый следующий твитт из обрабатываемого потока.

Частота употребления $Count(w)$ каждого уникального слова $w$ определяется из общего множества $S$ текстовых данных:

$$Count(w) = \sum_i (w_i \in S).$$

Настроение каждого твитта $sp(w, d)$ определяется из словаря – $d$, в котором прописано настроение (отношение):

$$sp(w,d) = \begin{cases} 0, & \text{если } w \text{ имеет негативный окрас,} \\ 1, & \text{если } w \text{ имеет нейтральный окрас,} \\ 2, & \text{если } w \text{ имеет положительный окрас.} \end{cases}$$

Ряд рекомендаций по сбору и обработке данных приводится в лабораторной работе №1 (в конце данного учебного пособия).

# 3. АНАЛИЗ ТЕКСТОВОГО КОНТЕНТА В СОЦИАЛЬНЫХ СЕТЯХ, АЛГОРИТМЫ КЛАССИФИКАЦИИ ТЕКСТОВЫХ СООБЩЕНИЙ

Одним из самых распространённых типов данных социальных сетей являются текстовые данные. Для одних социальных сетей, таких как, например, Instagram этот вид данных является сопутствующим, в других сетях: Facebook, Vkontakte – это один из основных типов данных, а в третьих, таких как Twitter – это основной тип. В любом случае текстовые данные представляют огромный интерес, а их анализ при этом очень востребован. Помимо определения семантического окраса текстовых сообщений, а также вычисления различных числовых метрик, приведенных в конце предыдущего раздела, имеет место задача классификации текстовых данных.

Классификация текстовых данных социальных сетей обладает некоторой спецификой по отношению к простой классификации текстовых данных, поскольку в социальных сетях существует ряд ограничений. Наиболее значимым ограничением является ограничение по объему текстовой информации. Для большинства социальных сетей это является определенным правилом хорошего тона, а некоторых (Twitter) ограничение на количество символов в сообщении задается на программном уровне. Это ограничение делает практически невозможным классификацию текстов сообщений традиционными методами. В качестве одного из решений данной задачи можно использовать обработку текстов методами k-means с использованием метрики TF-IDF, а также LDA алгоритм.

**K-means** – это один из наиболее распространённых алгоритмов кластеризации, который относится к неиерархическим итеративным алгоритмам.

Алгоритм k-средних строит k кластеров, расположенных на возможно больших расстояниях друг от друга. Основной тип задач, которые решает алгоритм k-средних – наличие предположений (гипотез) относительно числа кластеров, при этом они должны быть различны настолько, насколько это возможно. Выбор числа k может базироваться на результатах предшествующих исследований, теоретических соображениях или интуиции.

Общая идея алгоритма: заданное фиксированное число k кластеров наблюдения сопоставляются кластерам так, что средние в кластере (для всех переменных) максимально возможно отличаются друг от друга.

Первоначальное распределение объектов по кластерам происходит следующим образом. Выбирается число k, и на первом шаге эти точки считаются "центрами" кластеров. Каждому кластеру соответствует один центр. Этот выбор часто может осуществляется согласно такой логике:

- выбор k-наблюдений для максимизации начального расстояния;
- случайный выбор k-наблюдений;
- выбор первых k-наблюдений.

В результате каждый объект назначен определенному кластеру.

Вычисляются центры кластеров, которые затем и далее считаются покоординатными средними кластеров. Объекты опять перераспределяются.

Процесс вычисления центров и перераспределения объектов продолжается до тех пор, пока не выполнено одно из условий:

- кластерные центры стабилизировались, т.е. все наблюдения принадлежат кластеру, которому принадлежали до текущей итерации;
- число итераций равно максимальному числу итераций.

Другими словами, в k-means, каждый из k кластеров представлен одной точкой в пространстве. Обозначим этот набор представителей кластера как множество C=$\{c_j \mid j=1,\ldots,k\}$. Эти представители k кластеров также называются состояние кластера или центройды кластера. В алгоритмах кластеризации точки группируются некоторым понятием «близости» или «подобия». В k-means мера близости по умолчанию Евклидово расстояние. В частности можно показать, что k-means пытается минимизировать следующую неотрицательную функцию стоимости.

$$Cost = \sum_{i=1}^{N} \left( \arg min_j \left\| x_i - c_j \right\|_2^2 \right).$$

Таким образом, алгоритм k-means минимизирует итоговый квадрат Евклидова расстояния между каждой точкой $x_i$ и ее самым близким представителем кластера $c_j$. Приведенное выше часто упоминается как целевая функция k-means.

Алгоритм сходится, когда значения $c_j$. больше не изменяются.

Алгоритм k-means достаточно прост и удобен, однако у него имеются два недостатка:

- алгоритм слишком чувствителен к выбросам, которые могут искажать среднее. Возможным решением этой проблемы является использование модификации алгоритма – алгоритм k-медианы;

- алгоритм может медленно работать на больших базах данных.

Возможным решением данной проблемы является использование выборки данных.

**TF-IDF** – статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален количеству употребления этого слова в документе, и обратно пропорционален частоте употребления слова в других документах коллекции.

Мера TF-IDF часто используется в задачах анализа текстов и информационного поиска, например, как один из критериев релевантности документа поисковому запросу, при расчёте меры близости документов при кластеризации.

TF (term frequency – частота слова) – отношение числа вхождения некоторого слова к общему количеству слов документа. Таким образом, оценивается важность слова в пределах отдельного документа.

$$tf(t,d) = \frac{n_i}{\sum_k n_k},$$

где $n_i$ – число вхождений слова в документ. В знаменателе формулы – общее число слов в данном документе.

IDF (*inverse document frequency* – обратная частота документа) – инверсия частоты, с которой некоторое слово встречается в документах коллекции. Учёт IDF уменьшает вес широкоупотребительных слов. Для каждого уникального слова в пределах конкретной коллекции документов существует только одно значение IDF.

$$idf(t,D) = \log \frac{|D|}{|(d_i \ni t_i)|},$$

где $|D|$ – количество документов в корпусе;

$|(d_i \ni t_i)|$ – количество документов, в которых встречается $t_i$ (когда $n_i \neq 0$).

Выбор основания логарифма в формуле не имеет значения, поскольку изменение основания приводит к изменению веса каждого слова на постоянный множитель, что не влияет на соотношение весов.

Таким образом, мера TF-IDF является произведением двух сомножителей:

$$tf\ idf(t,d,D) = \mathrm{tf}(t,d) \times idf(t,D).$$

Большой вес в TF-IDF получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

**LDA** алгоритм, в свою очередь, основан на определении наиболее употребляемых топиках (темах), которые могут образовывать кластеры.

Модель LDA решает классическую задачу анализа текстов: создать вероятностную модель большой коллекции текстов (например, для information retrieval или классификации).

Очевидно, что у одного документа может быть несколько тем. Подходы, которые кластеризуют документы по темам, никак этого не учитывают. LDA – это иерархическая байесовская модель, состоящая из двух уровней:

- на первом уровне – смесь, компоненты которой соответствуют «темам»;
- на втором уровне – мультиномиальная переменная с априорным распределением Дирихле, которое задаёт «распределение тем» в документе.

Граф модели можно увидеть на рис. 9.

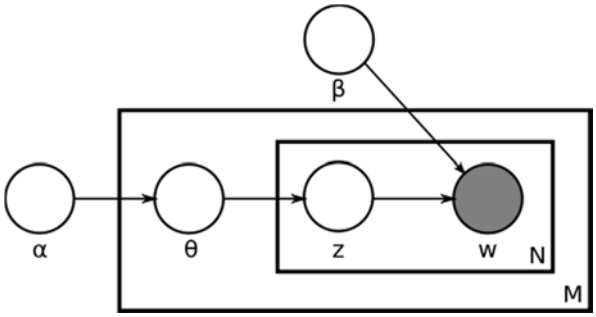



Рис. 9. Граф LDA модели

Модель будет генерировать новый документ исходя из следующих действий:

- выбор длины документа $N$;
- выбор вектора $\theta \sim (\alpha)$ – вектора «степени выраженности» каждой темы в этом документе.

Для каждого из $N$ слов $w$:

- выбор темы $z_n$ по распределению $Mult(\theta)$;
- выбор слова $w_n \sim p(w_n \mid z_n, \beta)$ с вероятностями, заданными в $\beta$.

Для простоты фиксируем число тем $k$ и будем считать, что $\beta$ – это набор параметров $\beta_{ij} = p(w^j = 1 \mid z^i = 1)$, которые нужно оценить, при этом явно не обозначая распределение по $N$. Совместное распределение выглядит следующим образом

$$p(\theta,..,N \mid \alpha,\beta) = p(N \mid \zeta)p(\theta \mid a)\prod_{n=1}^{N} p(z_n \mid \theta)p(w_n \mid z_n,\beta).$$

В отличие от обычной кластеризации с априорным распределением Дирихле или обычным распределением Байеса, мы не выбираем кластер один раз, а затем подбираем слова из этого кластера, а для каждого слова сначала выбираем по распределению $\theta$ тему, а уже потом определяем слово по этой теме.

Задача классификации текстовых данных может решаться и с помощью алгоритмов, основанных на методах **машинного обучения**.

Системы классификации основанные на машинном обучении, как правило, состоят из двух основных частей: частотный анализатор со словарём и нейросетевой классификатор. На вход системы поступает текст, на выходе получаем тему, которой посвящен этот текст (спорт, политика, медицина и т.п.).

Перед началом работы пользователю необходимо определить классы, с которыми будет работать система и подобрать множество

учебных текстов. Далее из множества учебных текстов, определённым образом, выделяются слова – формируется словарь. На последнем этапе инициализации системы, используя учебные тексты и полученный словарь, необходимо обучить нейросетевой классификатор. Классификатор необходимо обучить на множестве учебных текстов. Эту процедуру можно разбить на четыре этапа:

1. Определение количества классов, с которыми будет работать система. Подбор тематических учебных текстов, в одинаковом объеме для каждого класса. Тексты должны в точности соответствовать своей теме.

2. Составление словаря из множества учебных текстов. Для решения этой задачи можно воспользоваться обобщенным законом Бредфорда. В следующих разделах эта задача рассматривается подробнее.

3. Частотный анализ множества учебных текстов, с использованием полученного словаря. Получаем множество учебных векторов для нейронной сети.

4. Обучение нейросетевого классификатора на наборе векторов частотных характеристик учебных текстов. Далее рассматривается решение этой задачи.

После процедуры обучения классификатор текстов готов к работе.

Метод коллективного принятия решения, предлагаемый авторами данного пособия, основывается на уникальности каждого слова. В его основу положен словарь, в котором хранятся слова и их категории к которым они принадлежат.

Словарь формируется при участии эксперта, который обучает систему, а так же имеет возможность вносить изменения в словарь с целью его уточнения.

Вторая часть системы классифицирует текст основываясь на имеющимся словаре. Другими словами каждое слово из текста

$T = t_1 t_2 t_3 .. t_n$ сопоставляется со словарем $U = u_1 u_2 u_3 .. u_m$ с целью получения списка категорий:

$$\delta(u_i) = \{\underbrace{k_j ... k_h}_{z}\},$$

где $\delta(u_i)$ – функция извлечения категорий слова $u_i$ из словаря;

$\{\underbrace{k_j ... k_h}_{z}\}$ – вектор из $z$ элементов $k$ (элемент списка категорий $K = k_1 k_2 k_3 .. k_L$), к которым принадлежит слово $u_i$;

$1 \le j, h, z \le L$.

Если $\gamma(t_i) = u_j$ ($\gamma(t_i)$ – функция стемминга текста $t_i$, то $\delta(t_i) = \delta(u_j) = \overline{k}$. В случае отсутствия слова в словаре оно отдаётся эксперту на классификацию с последующим добавлением в словарь.

Отличительной особенностью алгоритма является наличие динамического порогового значения: при определении порогового значения учитывается количество и вес категорий слов, упоминающихся в тексте.

В результате обработки текста имеем перечень категорий (к которым относятся слова) на осонове которого алгоритм высчитывает пороговое значение «веса» тематики и классифицирует исходный текст.

# 4. ВИЗУАЛИЗАЦИЯ ДАННЫХ СОЦИАЛЬНЫХ СЕТЕЙ

Анализ данных социальных сетей включает в себя, в том числе, корректное и удобное их представление. Визуальное представление данных часто может являться не тривиальной задачей, ввиду их разнообразия и большого объема.

Рассмотрим наиболее распространенные способы визуализации данных социальных сетей.

В конце второго раздела шла речь о различных числовых метриках произведённых сообщений, либо иного текстового контента, содержащего информацию о геолокации. Удобным инструментом визуализации данной информации являются картограммы. На картограмме может быть показано количество сообщений определенного заданного типа (например, содержащего то или иное слово, объект либо хештег) произведенных в течение заданного временного интервала (рис. 10).


Рис. 10. Картограмма, содержащая информацию о частоте употреблений сообщений

При этом размер кружков на картограмме соответствует количеству подобных сообщений. В визуализацию таких данных можно добавить и семантический окрас сообщений, например, положительный, отрицательный и нейтральный (рис. 11).

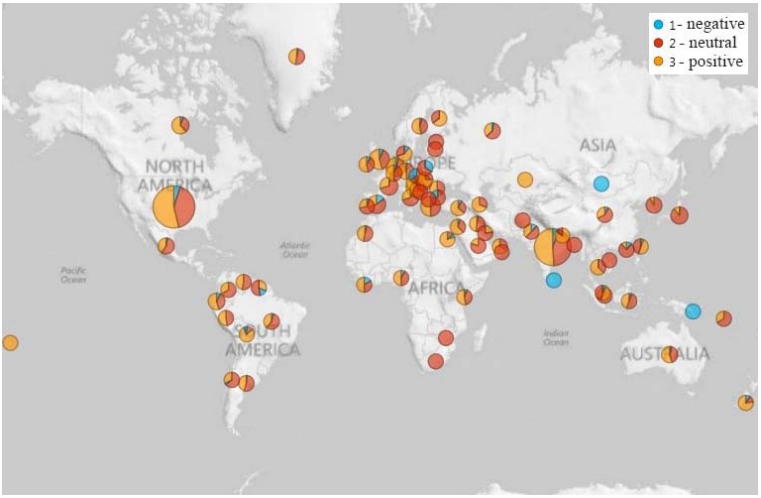



Рис. 11. Картограмма, содержащая информацию о частоте употреблений сообщений и об их семантическом окрасе

Анализ данных социальных сетей может быть направлен и на определение частоты употребления слов, фраз, иных объектов. В данном случае одним из наиболее удобных способов представления данных может служить облако тегов. На рис. 12 приводится облако тегов наиболее употребляемых слов (за исключением местоимений, союзов и предлогов) в текстовых сообщениях социальной сети Twitter.

Для визуализации различных аналитических выкладок и метрик полученных по социальным сетям могут использоваться стандартные способы визуализации: графики, диаграммы, таблицы и т.д. (примеры приведены на рис. 13, 14).

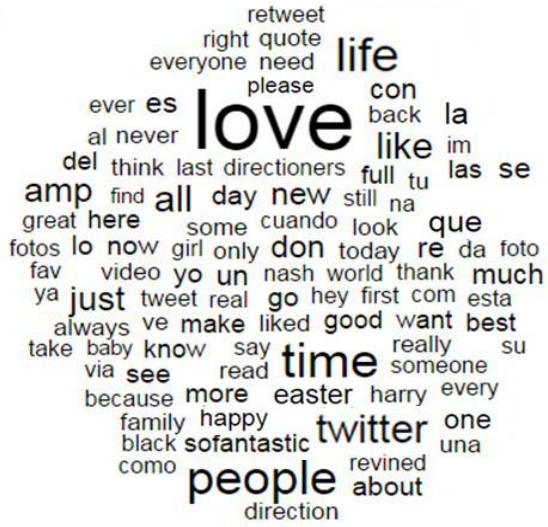



Рис. 12. Облако тегов наиболее употребляемых слов
в текстовых сообщениях социальной сети Twitter

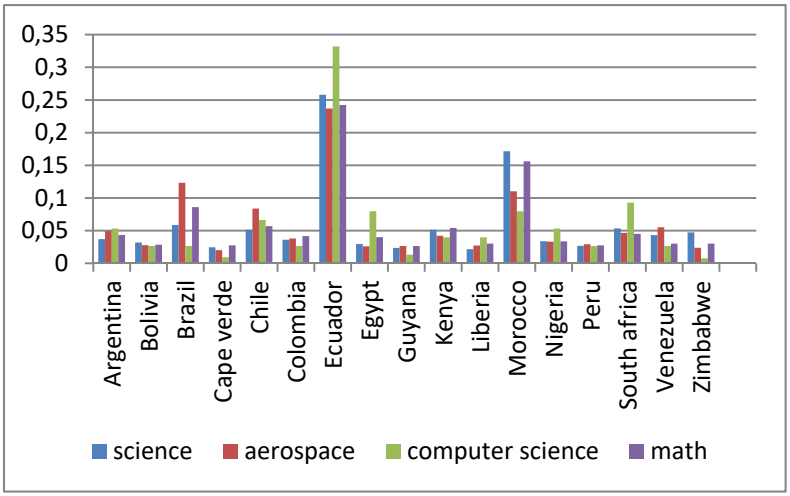



Рис.13. Распределение употребления определенных тем
в сообщениях сети Twitter по кластерам в группе стран

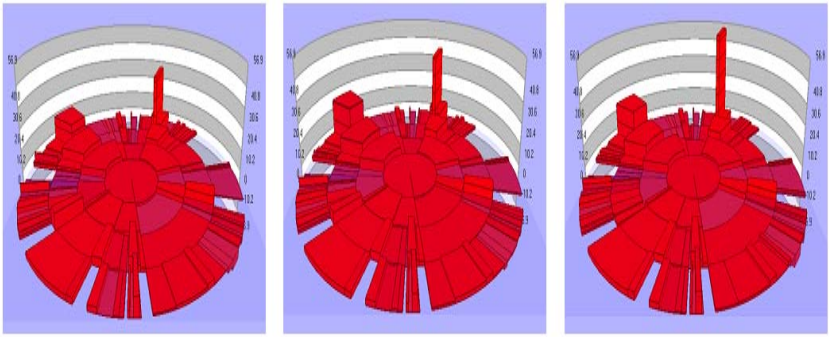


Рис. 14. Анимированное круговое представление сюжетов тем, затронутых в социальной сети

При всем разнообразии вариантов визуализации данных социальных сетей самым распространенным инструментом, описывающим и представляющим связи в социальной сети, является граф. При этом основной проблемой представления в виде графа часто является его большая размерность (рис. 15).

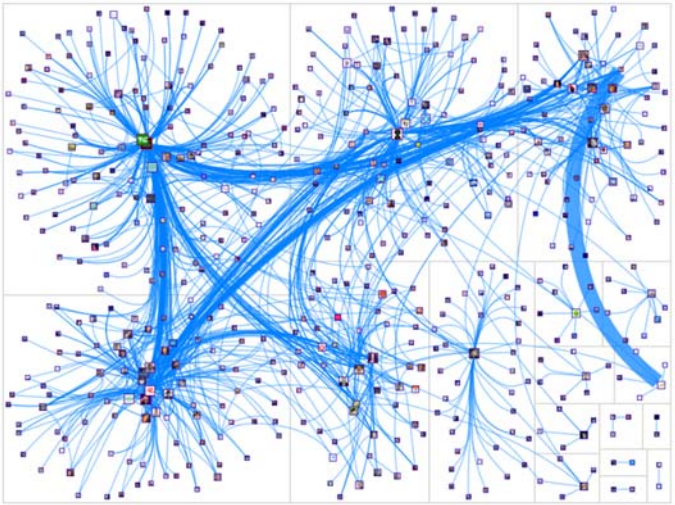


Рис. 15. Граф, описывающий сегмент социальной сети

Граф большой размерности с точки зрения визуального представления крайне нечитабелен. Для решения данной проблемы используется выделение сегментов и кластеров (рис. 16) с возможным дальнейшим уменьшением размерности.

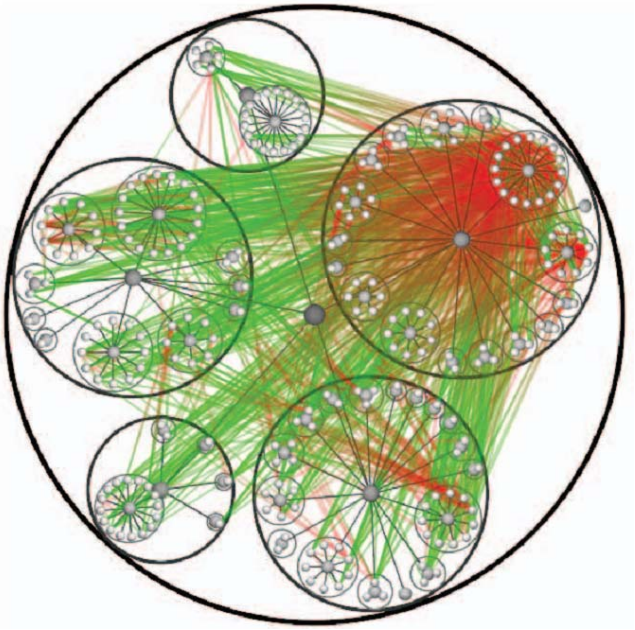


Рис. 16. Граф большой размерности
с выделенными отдельными кластерами

Задача кластеризации графа большой размерности будет рассмотрена ниже.

# 5. ПРЕДСТАВЛЕНИЕ СОЦИАЛЬНОЙ СЕТИ ГРАФОМ БОЛЬШОЙ РАЗМЕРНОСТИ, АНАЛИЗ СВЯЗЕЙ

Как уже было отмечено выше социальные сети, их сегменты, а также связи в них принято представлять в виде графа. Рассмотрим основные понятия теории, необходимые для введения в анализ социальной сети, представленной графом.

Граф – это упорядоченная пара $G=(V,E)$, где $V$ *(vertices)* – множество вершин (узлов) графа, а $E$ *(edges)* – множество ребер. Граф может быть как ориентированным, так и неориентированным (рис. 17).

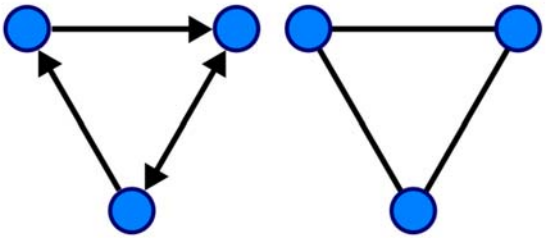


Рис. 17. Ориентированный и неориентированный графы

С точки зрения социальных сетей ребра в неориентированном графе могут описывать такую связь как «дружба», а в ориентированном – «подписку». Граф, представляющий социальную сеть, также может быть и смешанным – имеющим как ориентированные ребра, так и неориентированные.

Граф $G=(V,E)$, с $|V|=n$ вершинами может быть представлен в виде симметричной бинарной матрицы размерности $n$ x $n$:

$$A(i,j)=\begin{cases}1, \textit{если } v_i \textit{ связана ребром с } v_j,\\ 0, \textit{в иных случаях}.\end{cases}$$

Если граф ориентированный матрица $A$ не будет симметричной.

Если граф взвешенный (ребру которого поставлено в соответствие некое число – вес) матрица $A$ имеет вид

$$A(i,\ j)=\begin{cases} w_{ij},\ \textit{если}\ v_i\ \textit{связана}\quad \textit{ребром}\ \ c\ \ v_j, \\ 0,\ \textit{в иных случаях}, \end{cases}$$

где $w_{ij}$ – вес ребра, соединяющего вершины $v_i$ и $v_j$.

Опираясь на матричное представление графа множество наборов данных, социальных сетей и не только можно представить в виде графа.

Пусть $D=\{x_i\}_{i=1}^{n}$ – исходный набор данных. Определим взвешенный граф $G=(V,E)$ с весом ребер

$$w_{ij}=sim(x_i,\ x_j),$$

где $sim(x_i, x_j)$ описывает зависимость между $x_i$ и $x_j$.

Таким образом, исходный набор данных $D=\{x_i\}_{i=1}^{n}$ будет представлен взвешенным графом. В случае с социальной сетью, отыскание зависимостей является тривиальной задачей.

Граф $H=(V_H, E_H)$ называется подграфом графа $G=(V,E)$, если $V_H \subseteq V$ и $E_H \subseteq E$. Такой граф может использоваться для выделения сообщества в социальной сети.

Наиболее распространёнными задачами в рамках анализа социальных сетей, решаемых с помощью теории графов, являются:

1. Подсчёт пользователей и связей при помощи определения количества вершин и рёбер.
2. Поиск зависимостей и связей через нахождение компонент связности в графе.
3. Наличие сообществ в социальной сети, как правило, при помощи кластерного анализа графа.
4. Принадлежность профиля (вершины графа) к тому или иному сообществу (кластеру или подграфу).

5. Роль и влияние конкретных пользователей друг на друга и на сообщества (роль и значимость каждой вершины в графе).

6. Моделирование социальных сетей, в том числе посредством определения средней плотности графа.

Вершины и рёбра графа называются также элементами графа, число вершин в графе $|V|$ – *порядком*, число рёбер $|E|$ – *размером* графа.

*Маршрутом* в графе называют конечную последовательность вершин, в которой каждая вершина (кроме последней) соединена со следующей в последовательности вершиной ребром.

*Цепью* принято называть маршрут, ребра которого различны. В ориентированном графе цепь является *путём*. Длина цепи – это количество входящих в неё рёбер.

*Циклом* называют цепь, первая и последняя вершины которой совпадают. Длиной пути (цикла) называют число составляющих его рёбер.

Граф называется *связным*, если любые две его вершины связаны маршрутом.

*Компонента связности* графа $G=(V,E)$ – его подграф $H=(V_H,E_H)$, образованный на подмножестве всех вершин $V_H$, которые можно соединить произвольным маршрутом.

Связный граф состоит из единственной компоненты связности. На компоненте связности можно ввести понятие расстояния между вершинами как минимальную длину пути, соединяющего эти вершины. Ребро графа называется *мостом*, если его удаление увеличивает число компонент.

*Расстояние* между вершинами графа – длина кратчайшей цепи, соединяющей эти вершины.

Эксцентриситет вершины графа − это максимальное расстояние от неё до других вершин (по количеству ребер или их весу).

Диаметром графа $d(G)$ является максимальное расстояние между всеми парами вершин в графе $x_i$ и $x_j$ : $d(G)=\max d(x_i, x_j)$. Таким образом, диаметр графа − это максимальный из эксцентриситетов вершин.

*Радиусом графа* называется минимальный эксцентриситет среди всех вершин графа.

*Центральной вершиной* графа является вершина, чей эксцентриситет равен радиусу графа.

*Периферийной вершиной* графа является вершина, чей эксцентриситет равен диаметру графа.

При выделении сообщества принято сравнивать среднюю плотность связей внутри кластера и между кластерами, т.е. сообщество – это та часть графа, в которой средняя плотность между узлами превышает плотность связей между сообществами. Другими словами, внутри сообщества связей должно быть больше, чем между сообществами. Алгоритмы, описывающие выделение сообществ приводятся в соответствующем разделе данного учебного пособия.

*Степенью вершины* графа называется количество инцидентных ей рёбер. Последовательностью степеней вершин неориентированного графа является список степеней вершин, отсортированный по убыванию.

Пусть $N_k$ – количество вершин графа со степенью *k*. Распределение частот степеней графа при этом $(N_0, N_1, ..., N_t)$, где *t* – максимальная степень вершины в графе.

Пусть *X* – случайная величина, обозначающая степень вершины. Распределение степеней в графе – есть функция вероятности *f* случайной величины *X*, обозначенная как

$$(f(0), f(1), ..., f(t)),$$

где $f(k) = P(X = k) = \frac{N_k}{n}$ – вероятность вершины графа со степенью $k$.

В качестве примера рассмотрим граф, изображенный на рис.18. Последовательность степеней вершин этого графа: $(4, 4, 4, 3, 2, 2, 2, 1)$.

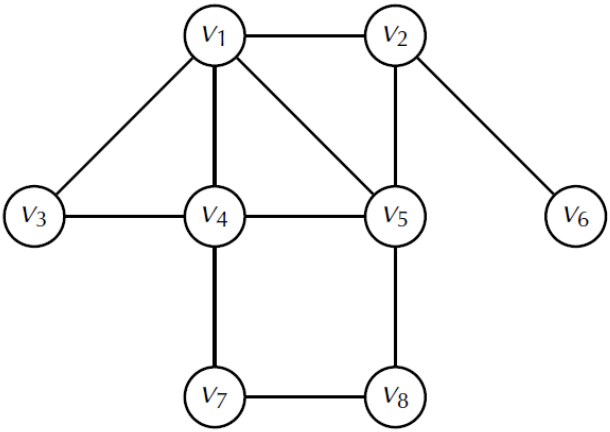



Рис. 18. Граф с восемью вершинами

Распределение частот степеней графа

$$(N_0, N_1, N_2, N_3, N_4) = (0, 1, 3, 1, 3).$$

Распределение степеней в графе при этом

$$(f(0), f(1), f(2), f(3), f(4)) = (0, 0.125, 0.375, 0.125, 0.375).$$

Распределение степеней в графе может использоваться в моделировании социальной сети, представленной данным графом.

# 6. СЕТЕВЫЕ СООБЩЕСТВА И КЛАСТЕРНЫЙ АНАЛИЗ ДАННЫХ СОЦИАЛЬНЫХ СЕТЕЙ

Сетевое сообщество представляет собой группу пользователей социальной сети, объединенных по тому или иному признаку. Как правило, связи внутри группы сильнее между собой, чем с внешней средой. При представлении социальной сети в виде графа сообщества могут выделяться кластерами.

Строго математического определения сообществ не существует. По этой причине выделение кластеров происходит исходя из задаваемых исследователями параметров. При этом существует целый ряд подходов для выделения сообществ.

На рис. 19 изображено два графа с выделенными кластерами, описывающими сообщества. На графе слева подобное выделение кластеров является очевидным даже визуально, однако это скорее идеальный случай. В реальности же отыскание сообществ в графе – задача далеко не тривиальная, как например, в случае графа расположенного слева на рис. 19. Разными цветами обозначены наиболее вероятные сообщества, полученные при проведении кластерного анализа.

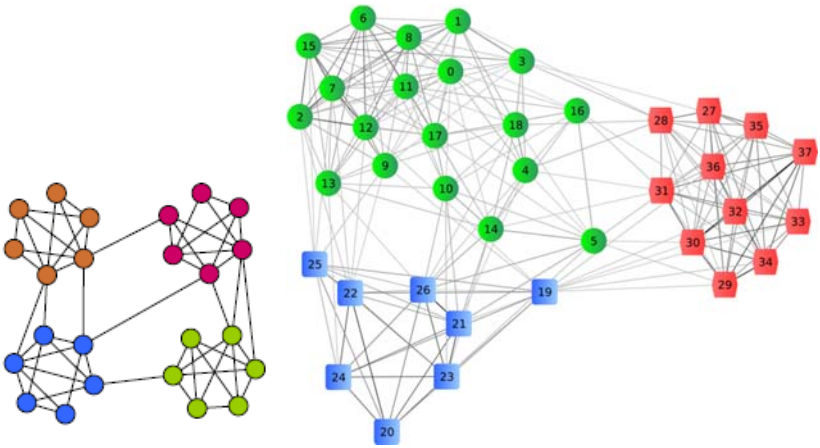



Рис. 19. Примеры графов с выделенными сообществами

Одним из возможных методов кластерного анализа является расчёт на основе *кластерного коэффициента*.

Кластерным коэффициентом вершины $v_i$ называется мера плотности связей данной вершины с соседними. Пусть $G_i = (V_i, E_i)$ – подграф графа $G$, включающий вершины, связанные с $v_i$ (назовем их соседями вершины $v_i$). Обозначим $|V_i| = n_i$ – количество соседей вершины $v_i$, а $|E_i| = m_i$ – количество ребер между соседями вершины $v_i$. Кластерный коэффициент вершины $v_i$ определяется следующим образом

$$C(v_i) = \frac{\text{число ребер в } G_i}{\text{максимальное число ребер в } G_i} = \frac{m_i}{\binom{n_i}{2}} = \frac{2\,m_i}{n_i(n_i-1)}.$$

Таким образом, кластерный коэффициент графа $G$ – представляет собой среднее значение кластерного коэффициента по всем своим вершинам:

$$C(G) = \frac{1}{n}\sum_i C(v_i).$$

Опираясь на определение кластерного коэффициента можно ввести понятие плотности связей в графе

$$\rho = \frac{2m}{n(n-1)},$$

где $m$ – количество рёбер в графе; $n$ – количество вершин.

Тогда для определенного кластера $C$ (сообщества) плотность связей равна количеству рёбер в этом сообществе $m_c$, делённому на максимально возможное количество рёбер в сообществе

$$\delta_{\text{int}}(C) = \frac{2m_c}{n_c(n_c-1)}.$$

А внешняя для данного кластера $C$ плотность аналогично получается при вычислении

$$\delta_{ext}(C) = \frac{2m_{ext}}{n_c(n_c - 1)},$$

где $m_{ext}$ – полное число внешних связей (рёбер).

Для выделения сообщества плотность внутренних связей должна быть больше плотности внешних связей, при этом плотность внутренних связей должна быть больше, чем средняя плотность графа, а плотность внешних связей должна быть меньше, чем средняя плотность графа. Таким образом, получаем следующие условия

$$\delta_{ext} < \rho < \delta_{\text{int}}.$$

Таким образом, можно находить кластеры путем анализа максимума разницы внутренних и внешних плотностей, вводя следующий параметр

$$\max = (\delta_{\text{int}} - \delta_{ext}).$$

При этом имеется ряд сложностей. Во-первых, для максимизации такой формулы нужно вначале выделить необходимое сообщество, что сопряжено с большим количеством переборов. А во-вторых задача отыскания сообществ при помощи плотности внутренних связей в графе на основе кластерного коэффициента решается в определенной степени приближенно:

– используются эвристики (нет гарантии схождения, однако на практике это, как правило, работает);

– используется жадный алгоритм (нет гарантии нахождения глобального минимума);

– приближенно ищется глобальный минимум (это задача комбинаторной оптимизации);

– часто используется рекурсивное разбиение графа на 2 части.

Рассмотрим иные подходы, использующиеся для выявления сетевых сообществ.

Наиболее распространенными подходами к разбиению графов на кластеры являются следующие алгоритмы:

– алгоритмы основанные на *модулярности* (жадный алгоритм, а также спектральная максимизация модулярности);

– алгоритмы разреза графов;

– эвристические алгоритмы (случайные блуждания и алгоритмы использующие степень посредничества).

Один из способов отыскания сетевых сообществ, также связанный с соотношением внутренней и внешней плотности кластера в графе, основывается на понятии *модулярности*.

Работа данного метода заключается в следующем. Пусть граф можно разбить на кластеры, и у каждого кластера есть своя метка (например, разные цвета). Тогда, чем больше отличается подграф, соответствующий сообществу, от случайного подграфа, тем лучше разбиение (рис. 20).

Вводится понятие модулярности, под которой понимается скалярная величина из отрезка [-1, 1], выражаемая следующей формулой

$$Q = \frac{1}{2m}\sum_{i,j}\left(A_{ij} - \frac{d_i d_j}{2m}\right)\delta(C_i, C_j),$$

где $A$ – матрица смежности графа;

$A_{ij}$ – $(i, j)$ элемент матрицы $A$;

$d_i$ – степень $i$-й вершины графа;

$C_i$ – метка вершины (номер сообщества, к которому относится вершина);

$m$ – общее количество ребер в графе;

$\delta(C_i, C_j)$ – дельта-функция (единица, если $C_i = C_j$, ноль иначе).

В этой формуле просчитывается количество связей, находящихся внутри одного кластера, и потом они складываются.

Стоит отметить, что в формуле присутствует член $\frac{d_i d_j}{2m}$. Его роль в следующем: сравнивается данное разбиение на кластеры (сообщества) со случайным графом. Если имеет место случайный граф, то в таком случае нет хороших кластеров. Таким образом, плотность в таком графе можно использовать как относительную метрику: у сообщества плотность должна быть строго больше, чем в случайном графе.

Если имеется узел со степенью $d_i$, то к нему присоединено $d_i$ рёбер. Вероятность того, что произвольное ребро присоединено к узлу, равна $\frac{d_i}{2m}$, а вероятность связи между ребрами равна $\frac{d_i d_j}{2m}$. Иначе говоря, мы вычисляем разницу между реальным количеством рёбер и ожидаемым. При этом сумма не равна нулю, если мы работаем в пределах одного класса.

Таким образом, при хорошем разбиении на кластеры модулярность высокая, если класс один, то модулярность равна 0.

Задача поиска выделения сообществ в графе сводится к поиску таких $C_i$, которые будут максимизировать значение модулярности. Примеры графов с различными значениями модулярности представлены на рис. 20.

Поскольку модулярность описывает качество разделения графа на группы, к решению задачи отыскания оптимального разбиения графа можно подойти, решая задачу максимизации. Однако простым перебором решить эту задачу практически невозможно, так как число вариантов разделения $n$ узлов на $k$ групп растёт, экспоненциально с ростом $n$.

Вопрос о количестве кластеров в графе обычно решается некоторой метрикой, в качестве которой в том числе может быть модулярность.

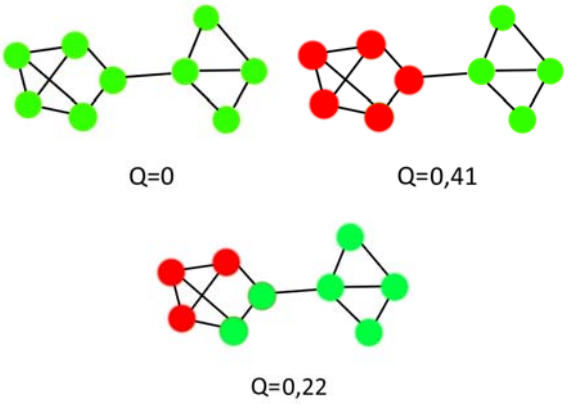



Рис. 20. Значение коэффициента модулярности
при выделении различных кластеров

Отметим, что модулярность можно не только использовать как критерий, но и отслеживать как показатель развития сообщества.

Рассмотрим жадный алгоритм оптимизации функции модулярности, имеющий в своём основании пошаговое объединение двух групп, дающих наибольший прирост модулярности.

Рассмотрим некое разбиение узлов из $N$ на $k$ групп ($N$ – множество узлов с числом элементов $n$). Функция модулярности будет равна

$$Q_1 = \frac{1}{m}\sum_{l=1,l\neq i,l\neq j}^{k} + \frac{1}{m}\left(m_i + m_j - \frac{\left(d(N_i)\right)^2 + \left(d(N_j)\right)^2}{4m}\right).$$

Теперь объединим группы $i$ и $j$ в одну, которую обозначим как $N_{i\cup j} = N_i \cup N_j$. Функция модулярности для нового графа будет иметь следующий вид

$$Q_2 = \frac{1}{m}\sum_{l=1,l\neq i,l\neq j}^{k}\left(m_l - \frac{\left(d(N_l)\right)^2}{4m}\right) + \frac{1}{m}\left(m_{i\cup j} - \frac{\left(d(N_{i\cup j})\right)^2}{4m}\right).$$

Число дуг внутри группы $N_{i\cup j}$ равно сумме дуг внутри групп $N_i$ и $N_j$ плюс число дуг между ними. Иными словами

$$m_{i\cup j} = m_i + m_j + m_{i,j}.$$

Степень объединённой группы $N_{i\cup j}$ равна сумме степеней групп $N_i$ и $N_j$, то есть

$$d\left(N_{i\cup j}\right) = d(N_i) + d\left(N_j\right).$$

Следовательно

$$\left(d\left(N_{i\cup j}\right)\right)^2 = \left(d(N_i)\right)^2 + \left(d\left(N_j\right)\right)^2 + 2d(N_i)d\left(N_j\right).$$

Учитывая это, получаем

$$\Delta Q = Q_2 - Q_1 = \frac{1}{m}\left(m_{i,j} - \frac{2d(N_i)d\left(N_j\right)}{4m}\right) =$$

$$= \frac{1}{m}\left(m_{i,j} - \frac{d(N_i)d\left(N_j\right)}{2m}\right).$$

Отсюда получаем, что наибольший рост модулярности происходит при объединении таких групп $N_i$ и $N_j$, для которых максимальна величина

$$\Delta\left(N_i, N_j\right) = m_{i,j} - \frac{d(N_i)d\left(N_j\right)}{2m}.$$

Также видно, что объединение групп, между которыми нет дуг ($m_{i,j} = 0$), не может дать увеличения модулярности.

К достоинствам алгоритма модулярности можно отнести следующее:

1. Модулярность достаточно просто интерпретируется и её значение равно разности между долей рёбер внутри сообществ, и ожидаемой долей связей, если бы рёбра были размещены случайно.

2. Модулярность возможно эффективно пересчитывать при небольших изменениях в кластерах.

В то же время, явными недостатками данного алгоритма являются:

1. Функционал не является непрерывным, и задача его оптимизации – дискретная. Для поиска глобального оптимума используют приближённые схемы. Некоторые из них действительно оптимизируют значение функционала, другие же по значению модулярности выбирают наилучшее решение из найденных, то есть без гарантий локальной оптимальности решения.

2. Существует проблема разрешающей способности (грубо говоря, функционал не видит маленькие сообщества). Эта проблема решается путем использования модифицированного функционала, который сохраняет все достоинства и добавляет параметр масштаба.

Рассмотрим еще один алгоритм выделения сообществ из разряда эвристических алгоритмов. Он основывается на понятии степени посредничества (в англоязычной литературе, как правило, обозначается – edge betweenness) – количество кратчайших путей, проходящих через ребро:

$$C(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

где $\sigma_{st}$ – общее число кратчайших путей из вершины *s* в вершину *t*, $\sigma_{st}(v)$ – число этих кратчайших путей, проходящих через вершину *v*.

Идея алгоритма заключается в том, что рёбра, через которые проходит наибольшее количество кратчайших путей, являются «мостами» между кластерами в графе. Поэтому для этих рёбер описанная метрика будет высокой. Можно найти эти рёбра, удалить их, и тогда граф распадется на сообщества.

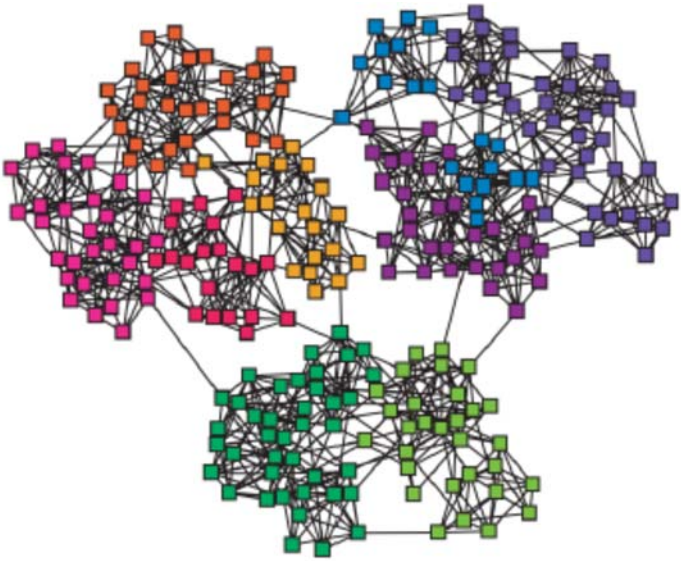


Рис. 21. Разделение графа на кластеры с помощью алгоритма, использующего метрику степени посредничества

В ходе выполнения алгоритма вычисляем степень посредничества и удаляем ребро с максимальным значением этой метрики. Каждый раз метрика должна быть пересчитана, так как при удалении ребер она может меняться. Делаем это до тех пор, пока в графе остаются ребра. Если хотим поделить граф на 2 части, то останавливаем процесс в тот момент, когда у нас есть 2 компоненты. Несмотря на то, что алгоритм жадный, он очень интуитивен и даёт неплохие результаты.

Пример представлен на рис. 21. При первом разбиении отделятся красный и оранжевый кластеры, потом отделятся все зеленые и т.д.

Алгоритм имеет и недостатки: для не очень больших графов он может работать не очень хорошо, а также отличается сложностью вычислений.

Еще одним алгоритмом, с помощью которого можно осуществлять кластеризацию графа является SCAN (Structural Clustering Algorithm for Networks) алгоритм. Данный алгоритм представляет большой интерес в первую очередь тем, что помимо кластеризации графа, он также осуществляет классификацию вершин этого графа.

Прежде чем описывать процесс классификации вершин графа по SCAN, необходимо ввести ряд следующих связанных использующихся понятий.

*Структура вершин* – величина

$$\Gamma(x) = \{y \in X | (x, y) \in U\} \cup \{x\},$$

где выше, $X$ – множество вершин графа, $U$ – множество рёбер графа.

*Структурное сходство* – величина

$$\sigma = \frac{|\Gamma(x) \cap \Gamma(y)|}{\sqrt{|\Gamma(x)||\Gamma(y)|}}.$$

Когда член кластера имеет похожую структуру с одним из его соседей, их расчетное структурное сходство будет большим. Мы используем *пороговое значение* $\varepsilon$, применяя его к вычисленному структурному подобию, когда определяем принадлежность к тому или иному кластеру.

$\varepsilon$ *– окрестность ядра* ($N_\varepsilon(x)$) – множество вершин (больше параметра $\mu$), чья структурная схожесть больше параметра $\varepsilon$.

*Core (ядро)* – это вершина, содержащая в $\varepsilon -$ окрестности, по крайней мере $\mu$ вершин.

*Прямая структурная достижимость* – две вершины напрямую структурно-достижимы, тогда и только тогда, когда одна из вершин является ядром, а другая вершина находится в $\varepsilon -$ окрестности данного ядра. Иными словами

$$DirReach_{\varepsilon,\mu}(x, y) \leftrightarrow Core_{\varepsilon,\mu}(x) \wedge y \in N_\varepsilon(x).$$

*Структурно-достижимые вершины* $(x, y)$ – две вершины структурно-достижимы, если существует последовательность вершин $x_1..x_n$ таких, что $x_1 = x$ и $x_n = y$ и $\forall i \in \{1..n-1\}$ $x_i$ и $x_{i+1}$ являются напрямую структурно достижимыми:

$$Reach_{\varepsilon,\mu}(x, y) \leftrightarrow \exists\, x_1..x_n \in X : x_1 = x \wedge x_n = y \wedge ,$$
$$\forall i \in \{1..n-1\}: DirReach_{\varepsilon,\mu}(x_i, x_{i+1}).$$

*Структурно-связанные вершины* – две вершины являются структурно-связанными тогда и только тогда, когда существует третья вершина, с которой указанные вершины являются попарно структурно-достижимыми:

$$Connect_{\varepsilon,\mu}(x, y) \leftrightarrow \exists\, u \in X: Reach_{\varepsilon,\mu}(u, x) \wedge Reach_{\varepsilon,\mu}(u, y).$$

*Hub («хаб»)* – это отдельная вершина, соседи которой принадлежат двум или более различным кластерам.

*Outlier (посторонний, «аутлаер»)* – это отдельная вершина, все соседи которой принадлежат одному и тому же кластеру, или не принадлежат никакому кластеру.

Пример графа с вершинами типа: ядро, «хаб», «аутлаер» – представлены на рис. 22.

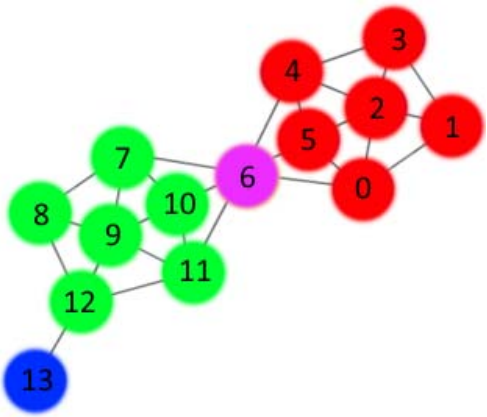



Рис. 22. Пример графа с различными типами вершин

Процесс работы SCAN-алгоритма поэтапно описан ниже.

1. Поиск начинается с начального посещения каждой вершины один раз, с целью нахождения структурно-связных кластеров, а затем посещения изолированных вершин, чтобы идентифицировать их (hub или outlier).

2. SCAN выполняет один проход сети и находит все структурно-связанные кластеры для заданного параметра. В начале все вершины помечены как неклассифицированные. Алгоритм SCAN классифицирует каждую вершину либо как являющуюся членом кластера, либо как не являющуюся. Для каждой вершины, которая еще не классифицирована, SCAN проверяет, является ли эта вершина ядром. Если вершина является ядром, новый кластер расширяется из этой вершины. В противном случае вершина помечается как не являющаяся членом кластера.

3. Чтобы найти новый кластер, SCAN начинается с произвольной ядра V и ищет все вершины, которые структурно-достижимы из V. Этого вполне достаточно, чтобы найти полный кластер, содержащий вершину V. Генерируется новый ID кластера, который будет назначен всем найденным вершинам.

4. SCAN начинается, постановкой всех вершин в $\varepsilon$ – окрестности вершины V в очередь. Для каждой вершины в очереди вычисляются все непосредственно достижимые вершины, и в очередь вставляются те вершины, которые до сих пор не классифицированы. Это повторяется до тех пор, пока очередь не опустеет.

5. Вершины, не являющиеся членами кластеров, могут быть дополнительно классифицированы как «хабы» или посторонние. Если отдельная вершина имеет рёбра на два или более кластеров, она может быть классифицирована как «хаб». В противном случае это «аутлаер».

SCAN-алгоритм лишен недостатков алгоритмов, использующих модулярность. Отличительной особенностью SCAN-алгоритма является наличие параметров $\mu$ и $\varepsilon$, которые могут задаваться пользователем или экспертом, что позволяет избежать проблемы определения маленьких сообществ. Иными словами, пользователь или эксперт может сам определять размеры сообществ, которые необходимо определять и члены которых необходимо классифицировать. При этом нахождение оптимального значения данных параметров можно провести при помощи машинного обучения системы, используя определённые сегменты сети.

Также в пользу SCAN-алгоритма выступают результаты исследования времени работы, которое значительно меньше, чем в алгоритмах, базирующихся на понятии модулярности.

Кластеризация графа, описывающего социальную сеть или ее фрагмент может использоваться не только для отыскания сообществ, но также и для удобного визуального представления (выделение сегментов и уменьшение размерности). При этом могут использоваться описанные выше алгоритмы, а также их модификации, зачастую направленные на работу с графами большой размерности.

# 7. МОДЕЛИРОВАНИЕ СОЦИАЛЬНЫХ СЕТЕЙ

Ранее в пятой главе приводилось определение распределения степеней в графе. Говоря о реальных социальных сетях, стоит отметить некоторые особенности распределений степеней их вершин. В частности, большинство вершин имеют довольно низкую степень, в то же время иногда встречаются узлы с достаточно высокой степенью. Наличие такого распределения является характеристическим свойством социальных сетей и одной из необходимых характеристик для классификации сети как сложной или социальной.

Функция распределения степеней вершин социальной сети, как правило, выглядит как показано на рис. 23.

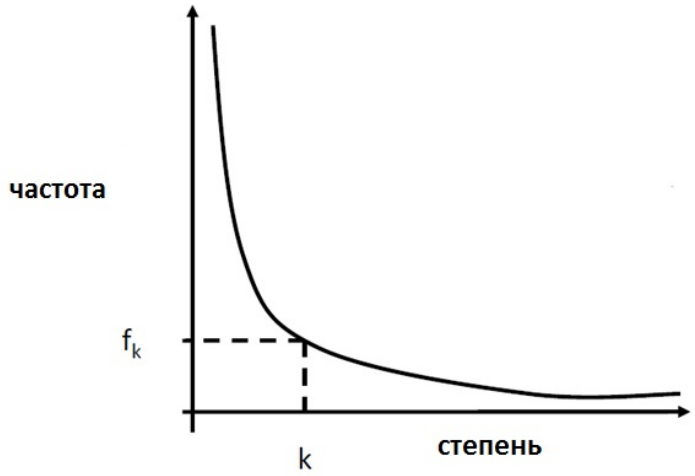



Рис. 23. Пример функции распределения степеней вершин социальной сети

На рисунке $f_k$ – вероятность случайно выбранной вершины, имеющий степень $k$.

В социальных сетях большинство пользователей имеют довольно маленькое количество связей, и наоборот, большое количество связей имеют малое количество пользователей.

Для аналитического описания степеней распределения социальных сетей больше всего подходят степенные распределения или распределения с «тяжелым хвостом».

Общий вид степенного распределения

$$f(x) = Cx^{-a},$$

где $C$ – нормировочная константа.

Данное обстоятельство играет важную роль в моделировании социальных сетей.

Стоит отметить, что довольно много явлений (как и социальные сети) обладают степенной функцией распределения, например, частота встречаемости слов в тексте или число цитирований научных публикаций.

Важными отличительными особенностями реальных социальных сетей являются:

– степенной закон распределения степеней вершин (очень много узлов с малым количеством соседей и в то же время существуют узлы с очень большим количеством соседей);

– маленький диаметр сетей (небольшое среднее расстояние между вершинами);

– высокий коэффициент кластеризации;

– наличие тесно связанного ядра (достаточно удалить из сети 10% вершин самой высокой степени, чтобы она распалась на мелкие компоненты; средний кратчайший путь 3,5 шага делает внутри 10% наиболее активных вершин).

Как правило, при моделировании социальных сетей необходимо учитывать все эти особенности.

Основной, ставшей в какой-то степени классической, группой моделей социальных сетей являются модели случайных графов, а именно:

– модель Эрдоша–Реньи (Erdős–Rényi model);
– модель Барабаши–Альберта (Barabási–Albert model);
– модель Ваттса–Строгатца (Watts–Strogatz model).

Рассмотрим данные модели.

## 7.1. Модель Эрдоша–Реньи

Модель Эрдоша–Реньи – это одна из моделей генерации случайных графов. Пусть $G\{V, E\}$ – граф с множеством вершин $V$ и рёбер $E$. Существуют два тесно связанных варианта модели: $G(n, m)$ – модель и $G(n, p)$ – модель. У $G(n, m)$ – модели два параметра: $n$ – число вершин (узлов), $m$ – число ребер. При этом:

- $n(n-1)/2$ – количество пар вершин, которые могут быть соединены ребром (количество паросочетаний);
- $C^m_{n(n-1)/2}$ – количество способов выбора $m$ ребер из $n(n-1)/2$ числа рёбер.

О случайном графе можно думать как об одном графе, выбранном случайным образом из этого множества $C^m_{n(n-1)/2}$.

У модели $G(n, p)$ имеются 2 параметра – $n$ (число узлов) и $p$ – вероятность того, что любая случайно выбранная пара узлов соединена ребром. Если $p = 0$, то нет рёбер вообще в графе, $p = 1$, то получаем полный граф, все узлы соединены рёбрами. Тогда число рёбер – это случайное число, которое имеет математическое ожидание:

$$\langle m \rangle = p\frac{n(n-1)}{2}.$$

При этом средняя степень вершины

$$\langle k \rangle = \frac{1}{n}\sum_i k_i = \frac{2\langle m \rangle}{n} = p(n-1) \approx pn.$$

Для случайных графов легко посчитать функцию распределения степени узлов: она описывается формулой Бернулли:

$$P(k_i = k) = P(k) = C_{n-1}^{k} p^k (1-p)^{n-k-1}.$$

В случае $n \to \infty$ и фиксированном среднем значении:

$$\langle k \rangle = pn = \lambda,$$

наша функция распределения степени узлов описывается распределением Пуассона:

$$P(k) = \frac{\langle k \rangle^k e^{-\langle k \rangle}}{k!} = \frac{\lambda^k e^{-\lambda}}{k!}.$$

Это распределение дискретно, т.е. $k$ имеет целочисленное значение.

В данной модели все узлы графа будут иметь степень, близкую к средней степени, и очень мало сильно-отличающихся узлов. В таком графе найти узлы с маленькой степенью почти невозможно.

Интересное свойство модели – наличие фазового перехода, при котором плавное изменение параметра приводит к скачкообразным изменениям свойств системы.

В модели $G(n, p)$ при $p = 0$, как было сказано выше, связи отсутствуют, а при $p = 1$ имеет место полный граф. Начнем понемногу увеличивать $p$ от значения $p = 0$. В промежутке от 0 до 1 происходит структурное изменение – появляется возможность попасть из одного узла в любой другой за какое-то количество шагов. Другими словами, происходит фазовый переход, который заключается в том, что появляется связность. В какой-то момент времени возникает большая связная компонента. Момент скачка из несвязанного состояния к появлению большой связной компоненты и есть фазовый переход.

Размер большой связной компоненты можно определить, решая уравнение:

$$1 - s = e^{-\lambda s},$$

где $s$ – это доля узлов, которые принадлежат большой связной компоненте.

Если $0 < \lambda < 1$, то большая связная компонента отсутствует, если $\lambda > 1$ то гигантская связная компонента резко возрастает. Выбирая критическое значение $p$, мы можем это контролировать.

У модели $G\ (n, p)$ есть еще ряд свойств. Если $p$ ведет себя как степенная функция от $n$ в степени, то это отражается на структуре графа. Меняя $p$ и $n$, можно гарантировать не только то, что возникает гигантская связная компонента, но и то, что возникают определенные структуры (циклы, полные подграфы различных порядков и т.д.). Другими словами, можно определить те моменты, когда в графе появляется определенного типа поведение. Преимущество модели заключается в том, что эти моменты математически хорошо просчитывается.

Модель случайного графа легко видоизменить и подстроить под нее функцию распределения, тем самым подгоняя свойства модели под имеющиеся экспериментальные данные.

Модель Эрдоша–Реньи – это модель, впервые предложенная для анализа сетей. Она предусматривает всего два параметра – заданное (зафиксированное изначально) количество узлов и вероятность наличия связи между парами этих узлов. У этой модели есть интересные свойства, такие как, например, наличие большой связанной компоненты и эффект малого диаметра. Однако её недостатками являются нестепенное распределение степеней узлов и маленький кластерный коэффициент.

### 7.2. Модель Барабаши–Альберта

Модель Барабаши–Альберта – модель с предпочтительным присоединением генерирующая безмасштабные сети (сети, которые развиваются со временем и имеющие степенное распределение вершин). Примерами безмасштабных сетей могут быть: сети цитирования, сети взаимодействия (коллаборации) и,

конечно же, социальные сети. Модель Барабаши–Альберта прежде всего подходит для моделирования сетей цитирования, однако применима и для моделирования остальных вышеназванных примеров.

Формирование сетей цитирование выглядит следующим образом. Имеется статья некоторого автора, затем другой автор пишет статью, цитируя данную (в конце каждой статьи есть целый список авторов, потому как автор цитирует много статей), а потом проходит время, и уже его начинают цитировать. Таким образом, цитируемость работы зависит от времени: чем раньше работа была написана, тем больше у нее шансов быть цитированной. В то же время сами узлы не будут заданы изначально, они появляются с течением времени. Имеет место характерная динамика: появляется новый узел и появляются связи, при этом сначала появляются связи, входящие в этот узел, затем появляются связи, от этого узла идущие дальше.

Модель Барабаши–Альберта характеризуется следующими параметрами:

– в начальный момент времени $t = 0$ есть $m$ несвязанных вершин;

– на каждом шаге $(t = 1, 2, 3, \dots)$ будем добавлять новую вершину с $m$ ребрами;

– количество связей, с которым приходит новая вершина в граф, фиксировано (равно $m$), однако к какому именно узлу он присоединяется – выбирается случайным образом, т.е. рост идет случайно.

При этом степень $i$-й вершины равна

$$k_i(t) = m\left(1 + log\frac{t}{i}\right),$$

где $i$ – индекс вершины соответствующий тому времени, когда узел присоединился.

На рис. 24 приводится поведение модели как функцию времени для $m = 20, i = 10, 20, 40$. Вершины, которые присоединились раньше, имеют большую степень с течением времени, чем позже присоединившиеся вершины. Кривые на рисунке не пересекаются, что и означает преимущество присоединившегося ранее. Когда вершина только приходит в систему, у неё конкуренция за новые связи очень небольшая, т.е. любая новая приходящая вершина в любом случае с ней свяжется; когда же сеть (граф) становится большой, то при появлении новой вершины она может установить с другими вершинами только ограниченное количество связей. Другими словами, шанс случайной вершины получить связь становится все меньше и меньше.

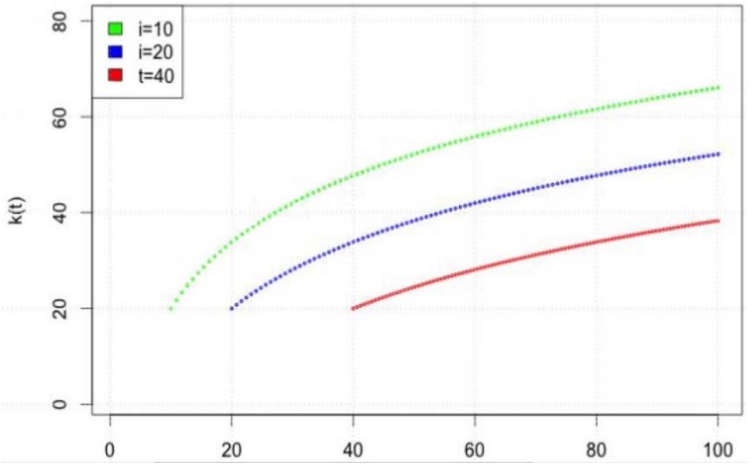



Рис. 24. Поведение модели Барабаши–Альберта как функция времени

Рис. 25 показывает поведение этой же модели как функцию от $i$ для $m = 20,\ t = 50, 100, 200$. В данном случае посмотрим на эти вершины (на поведение/степень вершин) как функцию от $i$ (от номера вершины). Например, когда нужно найти все те вершины, которые в момент времени $t$ имеют степень меньше какой-то данной степени, т.е. в конечном счете посчитать для такого графа

функцию распределения степеней узлов. Можно показать, что функция распределения степеней узлов в этом случае равна

$$P(k) = \frac{1}{m} e^{-\frac{m-k}{m}}.$$

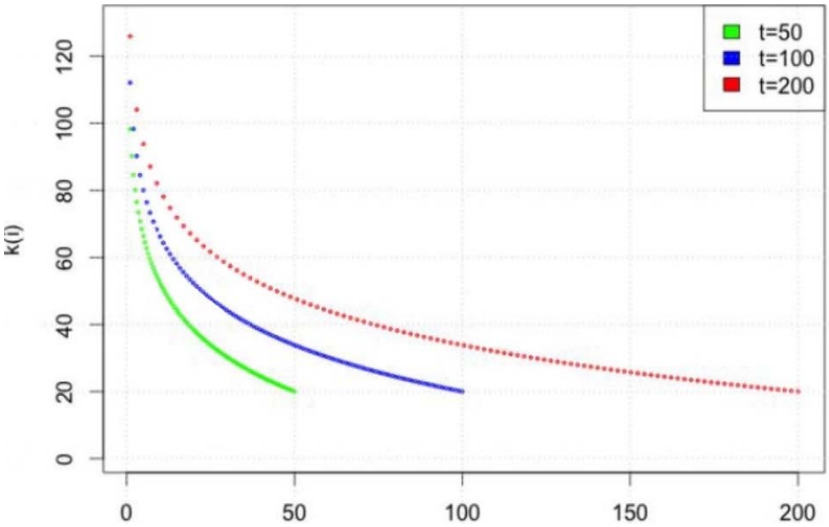



Рис. 25. Поведение модели Барабаши–Альберта
как функция номер вершины

В модели Барабаши–Альберта имеется еще одно условие. Вначале есть некоторое количество вершин и дальше с течением времени присоединяются новые вершины. Вместо того, чтобы присоединение к новым вершинам происходило просто случайным образом, вводится метод предпочтительного присоединения. Таким образом, вероятность у новой вершины присоединиться к существующей вершине пропорциональна степени этой существующей вершины.

У более ранних вершин и так было преимущество перед новыми вершинами – они с течением времени имели степень большую, чем новые. Добавление предпочтительного присоединения только усилило этот эффект. Наглядно это представлено на рис. 26.

С введением предпочтительного присоединения функция распределения степеней узлов становится равной

$$P(k) = \frac{2m^2}{k^3}.$$

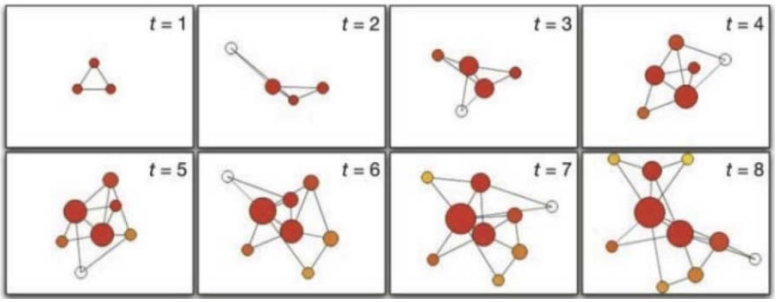



Рис. 26. Работа модели Барабаши–Альберта с предпочтительным присоединением

В модели Барабаши–Альберта присутствует больше узлов с малой степенью, чем в модели случайного графа. Эта модель демонстрирует гораздо большую вероятность иметь узлы с высокой степенью, чем модели просто случайно растущего графа. Однако определенным недостатком модели Барабаши-Альберта является маленький коэффициент кластеризации.

### 7.3. Модель Ваттса–Строгатца

Модель Ваттса–Строгатца или модель «малого мира» – модель с маленьким диаметром и большим коэффициентом кластеризации. Модель достаточно простая, но аналитически просчитывается только в очень простом случае.

Рассмотрим пример, изображенный на рис. 27.

В графе (в форме круга) изображенном на рис. 27 (в случае *а*) связи в форме треугольников создают высокий кластерный коэффициент. В данном случае у произвольного узла на графе 4 соседа. Максимальное количество связей между ними равно 4×3/2=6, а реальное число связей между соседями – 3, т.е.

кластерный коэффициент для произвольного узла равен 3/6=1/2. Так как граф абсолютно симметричен, то полный кластерный коэффициент для него также равен 1/2. Диаметр этого графа при этом высокий. Поэтому необходимо его уменьшить. Для этого достаточно в этот граф добавить некоторые длинные связи – например, как показано на рис. 27 (в случаях *б* и *в*). При наличии нескольких таких связей резко уменьшается диаметр графа.

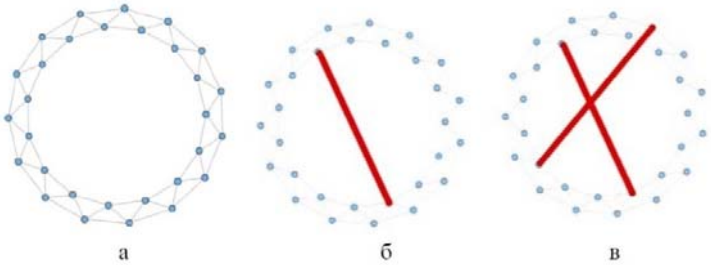



Рис. 27 – построение модели «малого мира»:
а – исходный граф; б, в – добавление длинных связей

Однако, при увеличении таких длинных связей, изменяется средняя степень узла в графе (добавляются новые связи). Можно попробовать сохранить эту среднюю степень узла фиксированной. Для этого, вместо того, чтобы просто добавлять какие-то связи, можно попробовать взять одну связь, локально оторвать ее и перенаправить в другое место. Тогда полное количество связей не меняется, поэтому средняя степень узла сохраняется, но получатся связи на большом расстоянии.

Рассмотрим этапы построения модели Ваттса–Строгатса.

– имеется граф, представляющий собой регулярную решетку с некоторым количеством ближайших соседей;

– введем параметр $p$, при котором модель становится однопараметрической;

– когда $p = 0$ имеет место регулярная решетка (идеальное кольцо), а когда $p = 1$ – получается случайный граф.

Параметр $p$ отображает долю добавленных рёбер. Наглядный пример приводится на рис. 28.

Если пересоединить все рёбра, то получится случайный граф.

Когда все связи регулярны, коэффициент кластеризации большой, но диаметр графа также высокий, а в случайном графе он близок к нулю. Поэтому следует ожидать, что существует золотая середина, где, с одной стороны, сохраняется высокий кластерный коэффициент, а с другой – появляется момент возникновения малого мира.

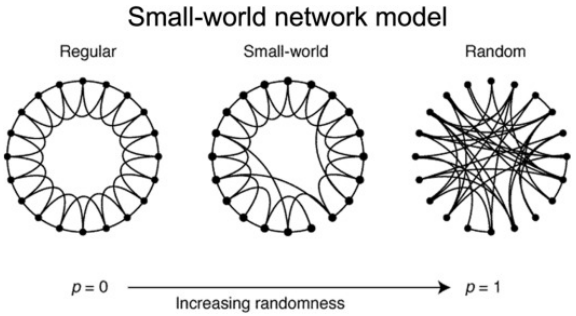



Рис. 28. Эволюция модели с добавлением ребер

В модели Ваттса–Строгатца виден переход от регулярного мира к случайному состоянию через состояние малого мира. Иначе говоря, одним параметром (параметром добавления случайных связей) можно регулярную решетку превратить в малый мир.

Авторы модели провели численные расчёты характеристик:

– распределение степеней вершин Пуассоновское;
– средняя длина пути $\langle L(p)\rangle$:

  $p \to 0\ \langle L(0)\rangle = 2n/k$ – круговая решетка;

  $p \to 1\ \langle L(1)\rangle = \log(n)\ /\log(k)$ – случайный граф;
– кластерный коэффициент $C(p)$:

  $p \to 0\ C(0) = 3/4 = \text{const}$ – круговая решетка;

  $p \to 1\ C(1) = k/n$ – случайный граф.

В модели Ваттса–Строгатца не заложен никакой контроль над функцией распределения степеней узлов. Эта функция изменяется от пуассоновской для случайного графа к отдельной точке (значению степени узла) для регулярной решетки. Ни тот, ни другой крайний случай не соответствует эмпирическим распределениям степенного закона, наблюдаемым на практике, поэтому не стоит ожидать, что, смешав их, можно получить степенной закон. Поэтому функцию распределения по этой модели не вычисляют, а основное внимание уделяют вычислению средней длины пути и кластерного коэффициента.

Интересно, что существует некоторый интервал значений $p$, в котором уже кластерный коэффициент не нулевой (недостаточно упал), а в то же время средняя длина пути уже становится достаточно маленькой. То есть можно с помощью одного параметра перейти из полного порядка к случайности и пройти через некоторую зону, где появляются эффекты малого мира.

Запишем основные особенности рассмотренных моделей (табл. 1).

Таблица 1

| **Модель** | **Особенности** |
|---|---|
| Эрдоша–Реньи | Функция распределения является распределением Бернулли. Коэффициент кластеризации низкий |
| Барабаши–Альберта | Степенной закон функции распределения. Коэффициент кластеризации низкий |
| Ваттса–Строгатца | Функция распределения близка к распределению Пуассона. Коэффициент кластеризации может быть большим |

Рассмотренные модели являются базовыми, они отличаются очевидностью и простотой. К настоящему времени создан целый ряд других моделей, но они получались более сложными и не настолько очевидными, как эти три основные.

# 8. ССЫЛКИ МЕЖДУ ПРОФИЛЯМИ ПОЛЬЗОВАТЕЛЕЙ В РАЗНЫХ СОЦИАЛЬНЫХ СЕТЯХ

Обычно пользователи сети Интернет имеют аккаунты сразу в нескольких популярных социальных сетях. При этом каждый из них, как правило, контактирует с одними и теми же людьми в различных социальных сетях. Основываясь на гипотезе, что человек состоит в одних и тех же сообществах во всех социальных сетях, которыми пользуется, можно установить связь между профилями разных социальных сетей, находящихся в одном сообществе и с определённой вероятностью предположить, что они принадлежат одному человеку. Данная задача является нетривиальной и имеет немалую практическую ценность.

Для решения поставленной задачи можно воспользоваться алгоритмом предварительной обработки данных:

1. Генерация графа на основе имеющихся данных. Вершины представляют собой сущность “человек” и хранят идентификаторы профилей, принадлежащих конкретному пользователю. Связь между вершинами устанавливается по следующему принципу: две вершины смежны, если связны профили соответствующих социальных сетей. Определим вес ребра между вершинами как

$$w(A,B) = \left|\left\{i : i \in [0, n-1] \cap \exists A_i, B_i \cap rel(A_i, B_i)\right\}\right|,$$

где *rel(A, B)* – функция, которая истина тогда и только тогда, когда профили *A* и *B* взаимосвязаны (установлено отношение дружбы или проявление активности).

2. К полученному графу применяется алгоритм машинного обучения Label Propagation, реализованный внутри программного решения GraphX API, который решает задачу кластеризации и находит сообщества в графе.

3. Для каждого сообщества генерируется RDD (resilient distributed dataset, отказоустойчивый распределенный набор данных) с набором профилей ВКонтакте, Facebook, Instagram и Twitter, которые связаны с одним или несколькими членами исходного сообщества

Таким образом, формируется набор данных (dataset), на основе которого мы можем строить предположения о принадлежности группы аккаунтов разных социальных сетей (без явного указания взаимосвязей) одному человеку.

Далее к каждому набору данных из датасета применяется следующая процедура:

1. Построение полного многодольного графа, в котором хранится информация о профилях социальных сетей и потенциал, характеризующий вероятность их принадлежности одному человеку.

В вершинах графа содержится информация о профилях, которая используется при их сравнении.

Для сравнения двух профилей используется многослойная нейронная сеть. На входной слой сети подаётся вектор размерности 12, содержащий следующие данные:

- Name ↔ Name'
- max(Name → Username', Name' → Username)
- max(Name → E-mail', Name' → E-mail)
- max(Name → Skype', Name' → Skype)
- Username ↔ Username'
- max(Username → E-mail', Username' → E-mail)
- Username ↔ Skype'
- max(Skype → Username', Skype' → Username)
- max(Skype → E-mail', Skype' → E-mail)
- E-mail ↔ E-mail'

– Phone ↔ Phone'
– Website ↔ Website'

Определим операции $a \to b$ и $a \leftrightarrow b$:

1.1. Полнота вхождения $a$ в $b$:

$$a \to b = 1 - \frac{d + r + s}{len(a)} \in [0, 1],$$

где $d$ – количество операций удаления для преобразования $a$ в $b$;

$r$ – количество операций замены для преобразования $a$ в $b$;

$s$ – количество операций транспозиции для преобразования $a$ в $b$;

*len(x)* – функция вычисления длины аргумента.

1.2. Сравнение $a$ и $b$:

$$\forall i \in [1, len(a)], j \in [1, len(b)] d[i, j] = 1 - \frac{dist(a[i], b[j])}{len(b[j])} \in [0, 1],$$

$$a \leftrightarrow b = \frac{\sum_1^{len(a)} d[i, fit(i)]}{min(len(a), len(b))} \in [0, 1],$$

где *dist(a, b)* – функция, вычисляющая расстояние Дамерау-Левенштейна для строк $a$ и $b$;

*fit(i)* – функция, возвращающая индекс слова строки $b$, поставленного в соответствие слову $a[i]$.

Операция сравнения не учитывает порядок слов. Все слова исходных строк попарно сравниваются, а затем, при помощи алгоритма Куна-Манкреса, каждому слову строки $a$ ставится в соответствие слово строки $b$ так, чтобы сумма схожести по всем парам слов была максимальной. Также не учитываются знаки препинания и прочие символы (за исключением букв и цифр).

Перед обработкой все символы входных данных приводятся к латинским по правилам транслитерации. Для этого используется сводная таблица основных алфавитов (русский, украинский, болгарский, индийский, арабский).

Обучающая и контрольная выборки собраны на основе первичных данных.

В качестве отрицательных примеров могут использоваться как случайные пары профилей, так и пары, найденные при помощи полнотекстового поиска по разным параметрам (*name, username, email, skype*).

2. В сгенерированной графе для каждой пары долей выполняется следующий алгоритм:

2.1. Рёбра сортируются в порядке убывания весов;

2.2. Удаляются рёбра, вес которых меньше порогового значения или одна из инцидентных вершин уже связана с какой-либо вершиной противоположной доли.

В результате этих преобразований получается граф, в котором каждая компонента связности представляет собой группу аккаунтов из разных социальных сетей, которые принадлежат одному человеку.

В связи с тем, что человек может одновременно принадлежать нескольким сообщества, а также если одна и та же группа была сформирована в нескольких сообществах, то можно полагать, что аккаунты этой группы действительно принадлежат одному пользователю.

Данные, полученные на последнем этапе, записываются в ту же базу, где хранятся первичные данные. Однако они не используются в качестве входных данных для реализованного алгоритма ввиду своей недостоверности.

# 9. СУЩЕСТВУЮЩИЕ РЕШЕНИЯ ПО АНАЛИЗУ СОЦИАЛЬНЫХ СЕТЕЙ

В связи с большой актуальностью задач, связанных с анализом данных социальных сетей, в настоящее время существует довольно много их программных решений.

В части языков программирования и существующих библиотек можно отметить язык высокого уровня Python, имеющий существенные преимущества в части анализа больших и разнородных данных, а также пакет networkX. В этом пакете имеется большинство стандартных сетевых метрик, работа с различными типами связей и узлов, а также широкий функционал, простота и вычислительная скорость, присущие решениям языка Python. Отличительным преимуществом данной библиотеки является хорошая визуализация графов, в том числе большой размерности. Помимо networkX, основные пакеты для сетевого анализа в Python – это igraph и graphtool. Igraph также есть в R (и в C/C++). В нём можно считать многие сетевые статистики и визуализировать графы. Graphtool обещает очень высокую скорость, особый подход к большим графам и качественную визуализацию.

К альтернативным решениям можно отнести статистическую среду R. Для расчёта базовой описательной статистики в R можно использовать пакеты sna, igraph, network. Для построения статистических моделей сетей будут полезны пакеты statnet, ergm и RSiena. К примеру, RSiena, используется для изучения динамики социальных сетей, выявления ключевых структурных механизмов формирования сетей, изучения взаимосвязи структурных и не структурных характеристик. Для изучения эволюции социальной сети все чаще используют стохастическое акторно-ориентированное моделирование (CAOM), реализованное в пакете RSiena.

Помимо решений, требующих навыков владения программированием, существуют следующие аналитические инструменты: UCINET, ORA, Pajek и Gephi. Первые две программы распространяются по лицензии, а Pajek и Gephi являются свободно распространяемыми. К примеру, Gephi используется, прежде всего, для визуализации социальных сетей, представленных графами.

Для более глубокого погружения в анализ данных социальных сетей рекомендуется использовать информационный ресурс Awesome Network Analysis, доступный по следующей ссылке: https://github.com/briatte/awesome-network-analysis. На данном ресурсе имеется как современная литература по данной тематике (книги, учебники, статьи), так и описания и реализация алгоритмов, а также ссылки на программные средства.

# 10. ЛАБОРАТОРНЫЕ РАБОТЫ ПО АНАЛИЗУ СОЦИАЛЬНЫХ СЕТЕЙ

В целях практического применения описанных в пособии сведений, а также овладения существующим инструментарием в области анализа социальных сетей в данном учебном пособии авторами приводятся следующие четыре лабораторные работы.

## 10.1. Лабораторная работа № 1 «Сбор данных социальных сетей»

При выполнении данной лабораторной работы приобретаются навыки взаимодействия с открытыми API социальных сетей VKонтакте и Twitter.

Для выполнения данной лабораторной работы необходимы аккаунты в этих социальных сетях.

***Задание № 1. Инсталляция и настройка Python окружения***

Лабораторная работа выполняется с использованием языка программирования Python. В случае отсутствия данного программного обеспечения на компьютере необходимо перейти на официальный сайт и скачать последнюю версию Python. Далее установить Python на компьютер следуя указаниям мастера установки.

Для выполнения также необходимы пакеты ***vk_api*** и ***tweepy***.

Для установки этих пакетов необходимо выполнить в командной строке с правами администратора команду вида:

```
pip install <имя_пакета>
```

***Задание № 2. Подключение к социальной сети VK***

Необходимо открыть Python Shell и запустить к исполнению следующий код.

```
import vk_api

def auth_handler():
```

```
    """ При двухфакторной аутентификации вызывается
эта функция.
    """

    # Код двухфакторной аутентификации
    key = input("Enter authentication code: ")
    # Если: True - сохранить, False - не сохранять.
    remember_device = True

    return key, remember_device

def stop_f(items):
    print (items)


def main():
    login, password = '<ВАШ ЛОГИН>', '<ВАШ ПАРОЛЬ>'
    vk_session = vk_api.VkApi(
            login, password,
            auth_handler=auth_handler  # функция
для обработки двухфакторной аутентификации
        )

    try:
        vk_session.auth()
    except vk_api.AuthError as error_msg:
        print(error_msg)

    tools = vk_api.VkTools(vk_session)
    vk_app = vk_session.get_api()
    print(vk_app.wall.post(message='Hello world!'))

if __name__ == '__main__':
    main()
```

Этот код отправит запрос на авторизацию в VK (для получения доступа к Вашей учетной записи) и опубликует на Вашей странице запись от Вашего имени с содержанием «Hello world!».

Далее можно попробовать скачать все посты (записи) на странице сообщества. Для этого нужно выбрать произвольное

сообщество (к примеру, группу «Аспирантов Самарского университета») и выполнить следующий код:

```
import vk_api
import json

group_id = -43938013

def main():
    """ Пример получения всех постов со стены """

    login, password = '<ВАШ ЛОГИН>', '<ВАШ ПАРОЛЬ>'
    vk_session = vk_api.VkApi(login, password)

    try:
        vk_session.auth(token_only=True)
    except vk_api.AuthError as error_msg:
        print(error_msg)
        return

    tools = vk_api.VkTools(vk_session)

    wall = tools.get_all('wall.get', 100,
{'owner_id': group_id})

    print('Posts count:', wall['count'])

    f = open(r" wall_asp.txt", 'a')
    f.write(json.dumps(wall))
    f.close()

if __name__ == '__main__':
    main()
```

В файле ***wall_asp.txt*** будет находиться дамп всех сообщений на стене выбранного сообщества (в нашем примере, группа «Аспирантов Самарского университета»). Можно увидеть, в полученном файле достаточно много непонятных слов и цифр. Это различные служебные и информационные поля. В них содержится

информация о том, когда был сделан пост, кем, сколько лайков, сколько комментариев и многое.

У Api социальной сети VKонтакте много возможностей. Подробнее о них можно прочитать на странице документации (https://vk.com/dev/manuals).

***Задание № 3. Подключение к социальной сети Twitter***

Алгоритм подключения к Twitter практически не отличается от подключения к социальной сети VKонтакте.

Для начала необходимо получить секретный токен. Для этого нужно перейти по ссылке (https://developer.twitter.com/en/apps) и нажать кнопку Create an app. После ввода всех требуемых данных, приложение сгенерирует API key и API key (рис. 29). Также стоит поменять права доступа приложения на чтение и запись.

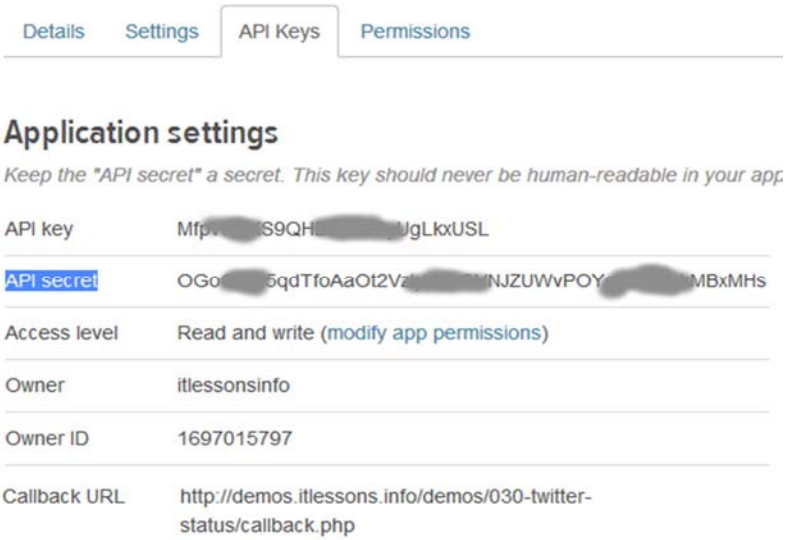



Рис. 29. Создание приложения по подключению к Twitter API

Затем следует подключиться к социальной сети Twitter и запросить информацию о количестве подписчиков пользователя данной сети.

```
import tweepy

API_KEY = "YOUR_API_KEY"
API_SECRET = "YOUR_API_SECRET"

ACCESS_TOKEN = "YOUR_ACCESS_TOKEN"
ACCESS_TOKEN_SECRET = "YOUR_ACCESS_TOKEN_SECRET"

auth = tweepy.OAuthHandler(API_KEY, API_SECRET)
auth.set_access_token(ACCESS_TOKEN,
ACCESS_TOKEN_SECRET)

api = tweepy.API(auth)

user = api.get_user('twitter')

print(user.screen_name, user.followers_count)
```

Данный код вызывает метод API, который называется get_user и возвращает информацию о пользователе, имя которого мы указали (в примере – ‘twitter’). Можно указать другое имя (своё, например) и посмотреть результат. Затем нужно вывести user.screen_name и user.followers_count – это отображаемое имя и количество тех, кто подписался на данного пользователя.

Далее попробуем что-либо написать на своей странице. Для этого запустим на исполнение следующий код:

```
import tweepy

API_KEY = "YOUR_API_KEY"
API_SECRET = "YOUR_API_SECRET"

ACCESS_TOKEN = "YOUR_ACCESS_TOKEN"
ACCESS_TOKEN_SECRET = "YOUR_ACCESS_TOKEN_SECRET"
```

```
auth = tweepy.OAuthHandler(API_KEY, API_SECRET)
auth.set_access_token(ACCESS_TOKEN,
ACCESS_TOKEN_SECRET)

api = tweepy.API(auth)

api.update_status('Просто так твит ни о чём!')
```

В результате получается следующее (рис. 30).

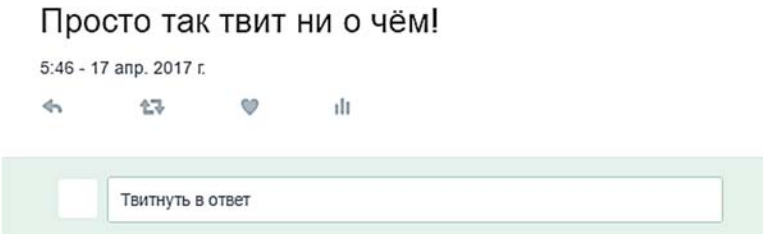



Рис. 30. Вывод сообщения в результате исполнения кода

Подробнее про Twitter API можно прочитать по ссылке (https://developer.twitter.com/en/docs/api-reference-index).

Для сбора данных подключимся к StreamingaAPI следующим образом.

```
import tweepy

API_KEY = "YOUR_API_KEY"
API_SECRET = "YOUR_API_SECRET"
ACCESS_TOKEN = "YOUR_ACCESS_TOKEN"
ACCESS_TOKEN_SECRET = "YOUR_ACCESS_TOKEN_SECRET"
class MyStreamListener(tweepy.StreamListener):

    def on_status(self, status):

        f = open('messages.txt' , 'a')
        f.write ("@%s, сообщение номер %s\n %s\n"
%(status.author.screen_name, status.id_str,
status.text))
        f.close()
```

```
        print ("@%s, сообщение номер %s\n %s\n"
%(status.author.screen_name, status.id_str,
status.text))

    def on_error(self, status_code):
        if status_code == 420:
            return False

def monitoring_tweets(query):

        GEOBOX_SAMARA_BIG=
[48.9700523344,52.7652295668,50.7251182524,53.66483
29274]

        myStreamListener = MyStreamListener()
        myStream = tweepy.Stream(auth = api.auth,
listener=myStreamListener)

myStream.filter(locations=GEOBOX_SAMARA_BIG)

if __name__ == '__main__':

    query = [" "]
    while True:
        try:
            auth = tweepy.OAuthHandler(API_KEY,
API_SECRET)
            auth.set_access_token(ACCESS_TOKEN,
ACCESS_TOKEN_SECRET)
            api = tweepy.API(auth)

            monitoring_tweets(query)

        except Exception as error_msg:
            print (error_msg)
```

Данный код обращается к серверам Twitter с запросом всех сообщений, у которых точка отправки сообщения находится в пределах указанного геобокса (GEOBOX_SAMARA_BIG).

Подробнее о StreamingAPI можно почитать здесь (https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data.html).

## 10.2. Лабораторная работа № 2 «Предобработка и классификация данных социальных сетей»

При выполнении данной лабораторной работы обучающиеся приобретают навыки обработки и кластеризации данных.

***Задание № 1. Предварительная обработка данных***

Лабораторная работа выполняется с использованием языка программирования Python. В случае отсутствия данного программного обеспечения на компьютере необходимо перейти на официальный сайт и скачать последнюю версию Python. Далее установить Python на компьютер следуя указаниям мастера установки.

Первый этапом данной лабораторной работы является удаление всех сторонних полей с оставлением лишь полезной информации.

Для этого необходимо загрузить скаченный в предыдущей лабораторной работе файл данных из социальной сети VK и извлечь необходимое поле text.

```
all_wall = []
file = open(r" wall_asp.txt")
for line in file.readlines():
    string = line

    wall = json.loads(string)
    json_decode = json.JSONDecoder()

    parsed_response =
json_decode.decode(json.dumps(wall))
    nodes = parsed_response.get('items')
    for node in nodes:
        all_wall.append(node.get("text"))
```

***Задание № 2. Подсчет слов в собранных текстовых данных***

Задание включает в себя реализацию алгоритма WordCount. Это одна из самых простых и тривиальных задач машинного обучения. Она заключается в разбиении текста на слова, а также подсчёт их количества.

Добавим в имеющийся код следующие строки

```
wordcount={}
for wall in all_wall:
    for word in wall.split():
        if word not in wordcount:
            wordcount[word] = 1
        else:
            wordcount[word] += 1
try:
    for wc in wordcount:
        print(wc, wordcount[wc])
except Exception:
    k=1
```

В результате перезапуска программы можно увидеть в консоли множество сочетаний вида «значение, ключ».

***Задание № 3. Кластеризация текстовых данных***

Следующей задачей является кластеризация текстовых данных.

В области анализа данных широко распространена задача разделения множества объектов на подмножества таким образом, чтобы все объекты каждого подмножества имели больше сходства друг с другом, чем с объектами других подмножеств.

Данная задача находит широкое практическое применение. Например, в области медицины алгоритм кластеризации может помочь идентифицировать центры клеток на изображении группы клеток. Используя GPS-данные мобильного устройства, можно определить наиболее посещаемые пользователем места в пределах определенной территории.

В рамках текущей задачи создадим новый файл и импортируем необходимые библиотеки.

```
import numpy as np
import pandas as pd
import nltk
import re
import os
import codecs
import matplotlib.pyplot as plt
import matplotlib as mpl
```

Необходимо считать данные в массив, а далее приступить к нормализации – приведению слова к начальной форме. Это можно сделать несколькими способами, используя стеммер Портера, стеммер MyStem и PyMorphy2. Стоит отметить – MyStem работает через wrapper, поэтому скорость выполнения операций очень медленная. Предлагается использовать стеммер Портера, хотя можно использовать другие и комбинировать их с друг другом (например, сначала пройтись PyMorphy2, а после стеммером Портера).

```
print(str(len(all_wall)) + ' запросов считано')

from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("russian")
#nltk.download()

def token_and_stem(text):
    tokens = [word for sent in
nltk.sent_tokenize(text) for word in
nltk.word_tokenize(sent)]
    filtered_tokens = []
    for token in tokens:
        if re.search('[а-яА-Я]', token):
            filtered_tokens.append(token)
    stems = [stemmer.stem(t) for t in
filtered_tokens]
    return stems
```

```
def token_only(text):
    tokens = [word.lower() for sent in
nltk.sent_tokenize(text) for word in
nltk.word_tokenize(sent)]
    filtered_tokens = []
    for token in tokens:
        if re.search('[а-яА-Я]', token):
            filtered_tokens.append(token)
    return filtered_tokens

#Создаем словари (массивы) из полученных основ
totalvocab_stem = []
totalvocab_token = []
for i in all_wall:
    allwords_stemmed = token_and_stem(i)
    #print(allwords_stemmed)
    totalvocab_stem.extend(allwords_stemmed)

    allwords_tokenized = token_only(i)
    totalvocab_token.extend(allwords_tokenized)
```

Создадим матрицу весов TF-IDF. Будем считать каждый поисковой запрос за документ (так делают при анализе постов в Twitter, где каждый твит – это документ). tfidf_vectorizer возьмем из пакета sklearn, а стоп-слова выберем из корпуса ntlk (изначально придется скачать через nltk.download()). Параметры можно подстроить от верхней и нижней границы до количества n-gram (в данном случае возьмем 3).

```
stopwords = nltk.corpus.stopwords.words('russian')
#можно расширить список стоп-слов
stopwords.extend(['что', 'это', 'так', 'вот',
'быть', 'как', 'в', 'к', 'на'])

from sklearn.feature_extraction.text import
TfidfVectorizer, CountVectorizer

n_featur=200000
```

```
tfidf_vectorizer = TfidfVectorizer(max_df=0.8,
max_features=10000,
                                   min_df=0.01,
stop_words=stopwords,
                                   use_idf=True,
tokenizer=token_and_stem, ngram_range=(1,3))
tfidf_matrix =
tfidf_vectorizer.fit_transform(all_wall)
print(tfidf_matrix.shape)
```

Над полученной матрицей начинаем применять различные методы кластеризации.

```
num_clusters = 5

# Метод к-средних - KMeans
from sklearn.cluster import KMeans

km = KMeans(n_clusters=num_clusters)
km.fit(tfidf_matrix)
idx = km.fit(tfidf_matrix)
clusters = km.labels_.tolist()

print(clusters)
print (km.labels_)

# MiniBatchKMeans
from sklearn.cluster import MiniBatchKMeans

mbk  = MiniBatchKMeans(init='random',
n_clusters=num_clusters) #(init='k-means++',
'random' or an ndarray)
mbk.fit_transform(tfidf_matrix)
mbk.fit(tfidf_matrix)
miniclusters = mbk.labels_.tolist()
print (mbk.labels_)

# DBSCAN
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.3,
min_samples=10).fit(tfidf_matrix)
labels = db.labels_
```

```
labels.shape
print(labels)

# Аггломеративная класстеризация
from sklearn.cluster import AgglomerativeClustering

agglo1 =
AgglomerativeClustering(n_clusters=num_clusters,
affinity='euclidean') #affinity можно выбрать любое
или попробовать все по очереди: cosine, l1, l2,
manhattan
answer = agglo1.fit_predict(tfidf_matrix.toarray())
answer.shape
```

Полученные данные можно сгруппировать в dataframe и посчитать количество запросов, попавших в каждый кластер.

```
#k-means
clusterkm = km.labels_.tolist()
#minikmeans
clustermbk = mbk.labels_.tolist()
#dbscan
clusters3 = labels
 #agglo
#clusters4 = answer.tolist()

frame = pd.DataFrame(all_wall, index = [clusterkm])

#k-means
out = { 'title': all_wall, 'cluster': clusterkm }
frame1 = pd.DataFrame(out, index = [clusterkm],
columns = ['title', 'cluster'])

#mini
out = { 'title': all_wall, 'cluster': clustermbk }
frame_minik = pd.DataFrame(out, index =
[clustermbk], columns = ['title', 'cluster'])

frame1['cluster'].value_counts()
frame_minik['cluster'].value_counts()
```

Из-за большого количества запросов не совсем удобно смотреть таблицы и хотелось бы больше интерактивности для понимания. Поэтому сделаем графики взаимного расположения запросов друг относительного друга. Сначала необходимо вычислить расстояние между векторами. Для этого можно применить косинусное расстояние. При этом можно использовать вычитание из единицы, чтобы не было отрицательных значений и находиться в пределах от 0 до 1.

```
from sklearn.metrics.pairwise import
cosine_similarity
dist = 1 - cosine_similarity(tfidf_matrix)
dist.shape
```

Так как графики будут двух- и трехмерные, а исходная матрица расстояний n-мерная, то придется применять алгоритмы снижения размерности. На выбор есть много алгоритмов (MDS, PCA, t-SNE), но остановим выбор на Incremental PCA. Этот выбор сделан вследствие выигрыша данного алгоритма в ресурсоемкости.

Алгоритм Incremental PCA используется в качестве замены метода главных компонентов (PCA), когда набор данных, подлежащий разложению, слишком велик, чтобы разместиться в оперативной памяти. IPCA создает низкоуровневое приближение для входных данных, используя объем памяти, который не зависит от количества входных выборок данных.

```
# Метод главных компонент - PCA

from sklearn.decomposition import IncrementalPCA
icpa = IncrementalPCA(n_components=2,
batch_size=16)
icpa.fit(dist)
demo2 = icpa.transform(dist)
xs, ys = demo2[:, 0], demo2[:, 1]

# PCA 3D
```

```
from sklearn.decomposition import IncrementalPCA
icpa = IncrementalPCA(n_components=3,
batch_size=16)
icpa.fit(dist)
ddd = icpa.transform(dist)
xs, ys, zs = ddd[:, 0], ddd[:, 1], ddd[:, 2]


#Можно сразу примерно посмотреть, что получится в
итоге
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs, ys, zs)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```

## 10.3. Лабораторная работа № 3 «Кластеризация изображений»

В рамках данной лабораторной работы будет производится кластеризация собранных из социальной сети изображений.

***Задание № 1. Скачивание фотографий из социальной сети***

При выполнении данной лабораторной работы следует использовать пакет VK (не VK_api).

Установим данный пакет с помощью следующей команды.

```
pip install vk
```

Необходимо создать приложение VK для того, чтобы получить доступ к данным. Для этого перейдем на страницу разработчиков VK и в разделе «Мои приложения» создадим новое приложение. Для дальнейшей работы нам потребуется ID приложения и Защищённый ключ.

Войдем в VK.

```
from urllib.request import urlretrieve
import vk, os, time, math

# Авторизация

login = ''
password = ''
vk_id = 'ID_ВАШЕГО_ПРИЛОЖЕНИЯ'

session = vk.AuthSession(app_id=vk_id,
user_login=login, user_password=password)

vkapi = vk.API(session)
```

Для удобства, входными данными можно указать ссылки на альбомы. Только целиком url не подойдёт, понадобится id хозяина альбома (группы или человека) и id самого альбома, которые и можно достать из ссылки. К примеру, в https://vk.com/album-54530371_212428070 id владельца (в данном случае сообщества) это -54530371, а id альбома – 212428070. Следует обратить внимание, если загружать из альбома сообщества, то «-» (дефис) перед id владельца обязателен.

```
#url = input("Введите url альбома: ")
url = "https://vk.com/album223007487_199949846"
# Разбираем ссылку
album_id = url.split('/')[-1].split('_')[1]
owner_id = url.split('/')[-
1].split('_')[0].replace('album', '')
```

Получаем на вход ссылку на альбом, затем разбираем её и раскладываем по переменным album_id и owner_id соответствующие id.

Далее нужно получить количество фото, а также инициализировать переменные для статистики.

```
photos_count =
vkapi.photos.getAlbums(owner_id=owner_id,
album_ids=album_id)[0]['size']
```

```
counter = 0 # текущий счетчик
prog = 0 # процент загруженных
breaked = 0 # не загружено из-за ошибки
time_now = time.time() # время старта
```

Проблема при загрузке большого количества фотографий в том, что за один запрос нельзя забрать больше 1000 штук, в то время как в альбоме их может быть десяток тысяч.

Процесс загрузки описывается следующим образом.

```
#Создадим каталоги
    if not os.path.exists('saved'):
        os.mkdir('saved')
    photo_folder =
'saved/album{0}_{1}'.format(owner_id, album_id)
    if not os.path.exists(photo_folder):
        os.mkdir(photo_folder)

    for j in range(math.ceil(photos_count / 1000)):
# Подсчитаем сколько раз нужно получать список
фото, так как число получится не целое - округляем
в большую сторону
        photos =
vkapi.photos.get(owner_id=owner_id,
album_id=album_id, count=1000, offset=j*1000,
v=5.95)['items'] # Получаем список фото
        for photo in photos:
            counter += 1

            sizes = photo['sizes']
            s = photo['sizes'][0]
            value_x = 0
            value_y = 0
            for size in sizes: #выбираем самый
большой размер
                if value_x < size['width']:
                    value_x = size['width']
                    s = size
                if value_y < size['height']:
                    value_y = size['height']
```

```
                    s = size

            print(s['url'])
            url_ = s['url'] # Получаем адрес
изображения
            print('Загружаю фото № {} из {}.
Прогресс: {} %'.format(counter, photos_count,
prog))
            prog =
round(100/photos_count*counter,2)
            try:
                urlretrieve(url_, photo_folder +
"/" + os.path.split(url_)[1]) # Загружаем и
сохраняем файл
            except Exception:
                print('Произошла ошибка, файл
пропущен.')
                breaked += 1
                continue
    time_for_dw = time.time() - time_now
    print("\nВ очереди было {} файлов. Из них
удачно загружено {} файлов, {} не удалось
загрузить.")
```

***Задание № 2. Кластеризация цветов на изображении***

Кластерный анализ (англ. Data clustering) – задача разбиения заданной выборки объектов (ситуаций) на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из схожих объектов, а объекты разных кластеров существенно отличались.

В рамках данной лабораторной работы будет решаться задача кластерного анализа изображений, собранных из задания №1.

Для начала попробуем обработать одну картинку, а именно, решим задачу определения доминирующих цветов.

Американский веб-разработчик Чарльз Лейфер (Charles Leifer) использовал метод k-средних для кластеризации цветов на изображении. Идея метода при кластеризации любых данных заключается в том, чтобы минимизировать суммарное

квадратичное отклонение точек кластеров от центров этих кластеров. На первом этапе выбираются случайным образом начальные точки (центры масс) и вычисляется принадлежность каждого элемента к тому или иному центру. Затем на каждой итерации выполнения алгоритма происходит перевычисление центров масс – до тех пор, пока алгоритм не сходится.

В применении к изображениям каждый пиксель позиционируется в трёхмерном пространстве RGB, где вычисляется расстояние до центров масс. Для оптимизации картинки уменьшаются до 200x200 с помощью библиотеки PIL. Она же используется для извлечения значений RGB.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import cv2

#read image
img = cv2.imread('colors.jpg')

#convert from BGR to RGB
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#get rgb values from image to 1D array
r, g, b = cv2.split(img)
r = r.flatten()
g = g.flatten()
b = b.flatten()

#plotting
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(r, g, b)
plt.show()
```

Данный код обрабатывает изображение и определяет пять самых доминирующих цветов на изображении.

В результате исполнения кода получим график примерно такого вида (рис. 31). На нем можно увидеть распределение пикселей в

трехмерном пространстве. Здесь также легко выделить пять доминирующих цветов (рис. 32).

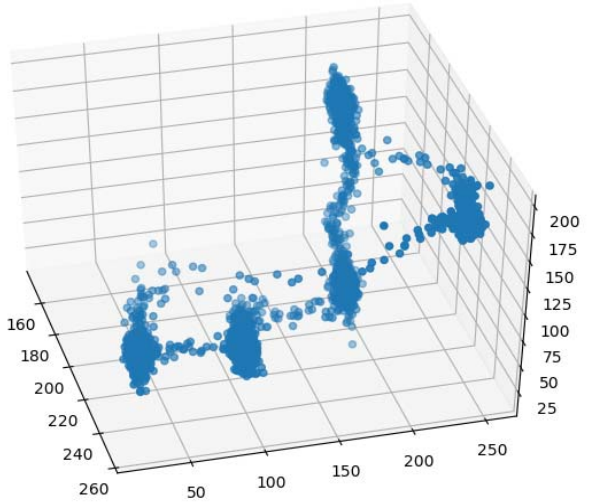



Рис. 31. Распределение пикселей в трехмерном пространстве

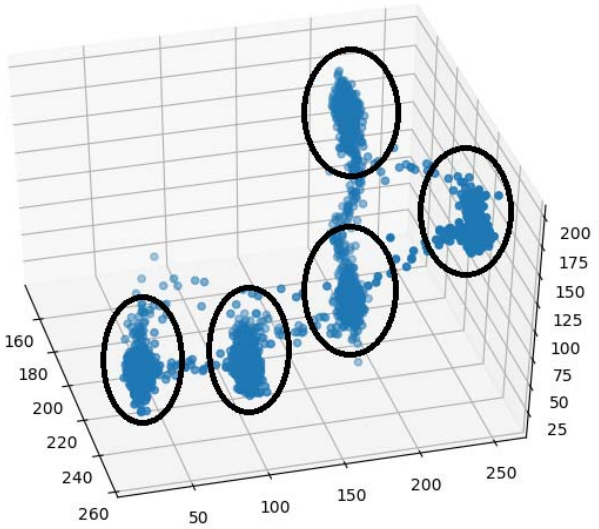



Рис. 32. Выделение доминирующих цветов

Несколько преобразим имеющийся код.

```
import cv2
from sklearn.cluster import KMeans

class DominantColors:

    CLUSTERS = None
    IMAGE = None
    COLORS = None
    LABELS = None

    def __init__(self, image, clusters=3):
        self.CLUSTERS = clusters
        self.IMAGE = image

    def dominantColors(self):

        #read image
        img = cv2.imread(self.IMAGE)

        #convert to rgb from bgr
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        #reshaping to a list of pixels
        img = img.reshape((img.shape[0] *
img.shape[1], 3))

        #save image after operations
        self.IMAGE = img

        #using k-means to cluster pixels
        kmeans = KMeans(n_clusters = self.CLUSTERS)
        kmeans.fit(img)

        #the cluster centers are our dominant
colors.
        self.COLORS = kmeans.cluster_centers_

        #save labels
        self.LABELS = kmeans.labels_
```

```
        #returning after converting to integer from
float
        return self.COLORS.astype(int)

img = 'colors.jpg'
clusters = 5
dc = DominantColors(img, clusters)
colors = dc.dominantColors()
print(colors)
```

На выходе мы получим пять векторов, где будут закодированы 5 основных цветов в формате RGB.

Далее построим гистограмму использования этих пяти основных цветов. Для этого добавим функцию plotHistogramm() в класс DominantColors.

```
def plotHistogram(self):

        #labels form 0 to no. of clusters
        numLabels = np.arange(0, self.CLUSTERS+1)

        #create frequency count tables
        (hist, _) = np.histogram(self.LABELS, bins
= numLabels)
        hist = hist.astype("float")
        hist /= hist.sum()

        #appending frequencies to cluster centers
        colors = self.COLORS

        #descending order sorting as per frequency
count
        colors = colors[(-hist).argsort()]
        hist = hist[(-hist).argsort()]

        #creating empty chart
        chart = np.zeros((50, 500, 3), np.uint8)
        start = 0

        #creating color rectangles
        for i in range(self.CLUSTERS):
```

```
            end = start + hist[i] * 500

            #getting rgb values
            r = colors[i][0]
            g = colors[i][1]
            b = colors[i][2]

            #using cv2.rectangle to plot colors
            cv2.rectangle(chart, (int(start), 0),
(int(end), 50), (r,g,b), -1)
            start = end

        #display chart
        plt.figure()
        plt.axis("off")
        plt.imshow(chart)
        plt.show()
```

Далее вызовем эту функцию.

```
dc.plotHistogram()
```

На выходе получим пропорциональное отображение использования доминирующих цветов на изображении. Пример отображения представлен на рис. 33.

Рис. 33. Отображение использования доминирующих цветов на изображении

***Задание № 3. Кластеризация изображении на основе доминирующих цветов***

Теперь, когда получена характеристика одного изображения, у нас есть инструмент, с помощью которого можно произвести кластеризацию множества изображений на основе их доминирующих цветов. Для этого необходимо обработать каждое изображение в папке, получить его уникальную цветовую

характеристику и сравнить ее с другими изображениями. Поэтому немного модифицируем имеющийся код:

1) добавим глобальную переменную, в которой будем хранить цветовые характеристики каждой картинки;

2) за место конкретной картинки будем указывать директорию и будем итерационно обрабатывать каждую картинку.

```
directory = '<ПУТЬ К ДИРЕКТОРИИ>'

for filename in os.listdir(directory):
    data_iter = []
    filenames.append(filename)
    img = str(directory) + '\\' + filename
    #print (img)
    clusters = 2
    dc = DominantColors(img, clusters)
    colors = dc.dominantColors()
    for i in colors:
        for j in i:
            data_iter.append(j)
    data.append(data_iter)
    print(colors)
```

Преобразуем список в массив данных и добавим код метода главных компонент для того, чтобы понизить размерность вектора характеристик с 15 до 3.

```
np_data = np.asarray(data, dtype=np.float32)

import numpy as np
from sklearn.decomposition import PCA
pca = PCA(n_components = 3)
XPCAreduced = pca.fit_transform(np_data)
print (XPCAreduced)
print(filenames)
```

Далее представим точки изображения на плоскости, где координатами будут элементы вектора характеристик.

```
xs, ys, zs = np_data[:, 0], np_data[:, 1], np_data[:,
2]
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs, ys, zs)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```

## 10.4. Лабораторная работа № 4 «Визуализация связей в виде графов»

При выполнении данной лабораторной работы производится визуализация связей социальных сетей в виде графа.

***Задание № 1. Запрос перечня друзей пользователя***

Для выполнения работы импортируем необходимый пакет.

```
import vk_api
```

Как и при выполнении первой лабораторной работы создаем подключение, а далее находим друзей user1 с помощью вызова метода friends.get.

```
friends_user1 = tools.get_all('friends.get', 100,
{'user_id': user1})
```

На выходе получим перечень id друзей пользователя. API VK позволяет дополнительно сразу запрашивать параметры пользователя такие как:

```
nickname, domain, sex, bdate, city, country, timezone,
photo_50, photo_100, photo_200_orig, has_mobile,
contacts, education, online, relation, last_seen,
status, can_write_private_message, can_see_all_posts,
can_post, universities
```

Необходимые параметры указываются в параметре fields. При этом для выполнения текущей задачи имеет ценность лишь список друзей. Поэтому дополнительный параметры указывать не будем.

Итоговый код будет выглядеть следующим образом.

```
import vk_api
import time
import json
import pickle

user1 = 249771326

def auth_handler():
    """ При двухфакторной аутентификации вызывается эта
функция.
    """

    # Код двухфакторной аутентификации
    key = input("Enter authentication code: ")
    # Если: True - сохранить, False - не сохранять.
    remember_device = True

    return key, remember_device

def stop_f(items):
    print (items)


def get_groups_users(friends_list, tools):
    friends_out = {}
    for friend in friends_list:
        try:
            friends_out[friend] =
tools.get_all('friends.get', 100, {'user_id': friend})
        except Exception:
            friends_out[friend] = []
        time.sleep(1)

    return friends_out
```

```
def main():
    login, password = '<ВАШ ЛОГИН>', '<ВАШ ПАРОЛЬ>'
    vk_session = vk_api.VkApi(
            login, password,
            auth_handler=auth_handler  # функция для
обработки двухфакторной аутентификации
        )

    try:
        vk_session.auth()
    except vk_api.AuthError as error_msg:
        print(error_msg)

    tools = vk_api.VkTools(vk_session)
    friend_list=[]
    friend_list.append(user1)

    friends_out = get_groups_users(friend_list, tools)
    print(friends_out)

if __name__ == '__main__':
    main()
```

В результате исполнения этого данного получится JSON-response со списком друзей пользователя, id которого, был указан в переменной user1.

***Задание № 2. Формирование графа друзей пользователя***

Установим модуль networkX (необходима версия 1.9).

```
pip install networkx==1.9
```

Далее реализуем функции для визуализации графа.

```
import networkx as nx

def make_graph(friends_out, friends_friends):
    graph = nx.Graph()
    graph.add_node(user1, size =
friends_out[user1]['count'])
```

```
    for i in friends_out[user1]['items']:
        try:
            graph.add_node(i, size =
friends_friends[i]['count'])
            intersection =
set(friends_out[user1]['items']).intersection(set(frien
ds_friends[i]['items']))
            graph.add_edge(user1, i,
weight=len(intersection))
        except Exception:
            print("err")

    return graph
```

Данная функция создает граф, вершинами которого являются узлы, содержащие id друзей, а также связывает между собой пользователя и его друга. Так как необходимо связать не только пользователя и друзей, а еще и друзей между собой, то модифицируем код следующим образом.

```
def make_graph(friends_out, friends_friends):
    graph = nx.Graph()
    graph.add_node(user1, size =
friends_out[user1]['count'])


    for i in friends_out[user1]['items']:
        try:
            graph.add_node(i, size =
friends_friends[i]['count'])
            intersection =
set(friends_out[user1]['items']).intersection(set(frien
ds_friends[i]['items']))
            graph.add_edge(user1, i,
weight=len(intersection))
        except Exception:
            print("err")

    for i in range(len(friends_out[user1]['items'])):
```

```
        id1= friends_out[user1]['items'][i]
        for k in range(i+1,
len(friends_out[user1]['items'])):
            id2= friends_out[user1]['items'][k]
            try:
                intersection =
set(friends_friends[id1]['items']).intersection(set(fri
ends_friends[id2]['items']))
                if len(intersection) > 0:
                    graph.add_edge(id1, id2,
weight=len(intersection))
            except Exception:
                print("err friend")
    return graph
```

Здесь дополнительно добавляется переменная friends_friends, которая содержит в себе списки друзей друзей пользователя.

Данная переменная появляется в результате вызова метода get_groups_users куда на вход подаем список id друзей пользователя.

```
friend_friends =
get_groups_users(friends_out[user1]['items'], tools)
```

Количество запросов ограничено, поэтому с целью экономии предлагается использовать модуль Pickle. Данный модуль реализует мощный алгоритм сериализации и десериализации объектов Python. "Pickling" – процесс преобразования объекта Python в поток байтов, а "unpickling" – обратная операция, в результате которой поток байтов преобразуется обратно в Python-объект. Поток байтов легко можно записать в файл, поэтому модуль pickle широко применяется для сохранения и загрузки сложных объектов в Python.

Пример его использования выглядит следующим образом.

```
   with open('friends_friends.pkl', 'wb') as output:
       pickle.dump(friend_friends, output,
pickle.HIGHEST_PROTOCOL)
```

Этот пример для сохранения объекта. Следующий пример – для загрузки объекта.

```
   with open('friends_friends.pkl', 'rb') as input:
        friends_friends = pickle.load(input)
```

Вызовем функцию make_graph в функции main.

```
g = make_graph(friends_out, friend_friends)
```

После формирования связей так же рекомендуется сохранить полученный граф при помощи Pickle.

***Задание № 3. Визуализация графа друзей пользователя***

Сформированный граф необходимо визуализировать. Для этого будем использовать модуль matplotlib (необходимая версия – 2.2.4).

Реализуем функцию визуализации.

```
def plot_graph(graph, adjust_nodesize):
    #pos = nx.drawing.layout.circular_layout(graph)
    pos=nx.spring_layout(graph, k=0.1)
    #нормализуем размер вершины для визуализации.
Оптимальное значение параметра
    #adjust_nodesize - от 300 до 500
    nodesize = [graph.node[i]['size']/adjust_nodesize
for i in graph.nodes()]
    #нормализуем толщину ребра графа. Здесь хорошо
подходит
    #нормализация по Standard Score
    edge_mean =
numpy.mean([graph.edge[i[0]][i[1]]['weight'] for i in
graph.edges()])
    edge_std_dev =
numpy.std([graph.edge[i[0]][i[1]]['weight'] for i in
graph.edges()])
    edgewidth = [((graph.edge[i[0]][i[1]]['weight'] -
edge_mean)/edge_std_dev/2) for i in graph.edges()]
    #создаем граф для визуализации
```

```
    nx.draw_networkx_nodes(graph,
pos,node_size=nodesize, node_color='y', alpha=0.9)

nx.draw_networkx_edges(graph,pos,width=edgewidth,edge_c
olor='b')
    nx.draw_networkx_labels(graph,pos,fontsize=5)
    #сохраняем и показываем визуализированный граф
    plt.savefig('saved')
    plt.show()
```

Вызовем реализованную функцию в main.

```
plot_graph(g, 500)
```

В результате работы кода будет получено изображение графа друзей пользователя.

Данный метод является довольно простым, т.к. при этом не происходит процесс кластеризации графа, для лучшего понимания его структуры. Однако этот пример удобен для понимания процесса работы с графами.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Aizawa A. An information-theoretic perspective of tf–idf measures // Information Processing & Management. 2003. Т. 39. №1. P. 45-65.

2. Bakaev V. A., Blagov A. V. The analysis of profiles on social networks // Information Technologies and Nanotechnologies. 2018. P. 1860-1863.

3. Barabasi A.L., Bonabeau E. Scale Free Networks // Scientific American, 2003, P. 50-59.

4. Blagov A. et al. Big data instruments for social media analysis // Proceedings of the 5th International Workshop on Computer Science and Engineering. 2015. P. 179-184.

5. Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters // Communications of the ACM. 2008. Т.51. №1. P. 107-113.

6. Key Trends to Watch in Gartner 2012 Emerging Technologies Hype Cycle. URL: http://www.forbes.com/sites/gartnergroup/2012/09/18/key-trends-to-watch-in-gartner-2012-emerging-technologies-hype-cycle-2.

7. Leskovec J., Faloutsos C. Sampling from large graphs // Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2006. P. 631-636.

8. Liu X., Murata T. Advanced modularity-specialized label propagation algorithm for detecting communities in networks // Physica A: Statistical Mechanics and its Applications, Vol. 389, Issue 7, P. 1493-1500.

9. Newman, M. E. J. Fast algorithm for detecting community structure in networks // Phys. Rev. E. 2004. T.69. P. 576-587.

10. Newman M. E. J., Girvan M. Finding and evaluating community structure in networks // Phys. Rev. E. 2004. T. 69. P. 426-438.

11. Rytsarev I. A., Blagov A. V. Classification of Text Data from the Social Network Twitter // CEUR Workshop Proceedings. 2016. T. 1638. P. 851-856.

12. Social-network-sourced big data analytics / W. Tan, M.W. Blake, I. Saleh, S. Dustdar // IEEE Internet Computing. 2013. №. 5. P. 62-69.

13. VanBoskirk S., Overby C. S., Takvorian S. US interactive marketing forecast, 2011 to 2016 // Forrester Research. 2011.

14. Xu X. Scan: a structural clustering algorithm for networks // Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2007. P. 824-833.

15. Венгерский алгоритм решения задачи о назначениях. URL: http://e-maxx.ru/algo/assignment_hungary.

16. Гусарова Н. Ф. Анализ социальных сетей. Основные понятия и метрики // Университет ИТМО. СПб., 2016. 70 с.

17. Анализ социальных сетей: методы и приложения / А. Коршунов [и др.] // Труды Института системного программирования РАН. 2014. Т. 26. №. 1.

18. Красников И. А., Никуличев Н. Н. Гибридный алгоритм классификации текстовых документов на основе анализа внутренней связности текста // ИВД. 2013. № 3(26).

19. Сметанин Н. Нечёткий поиск в тексте и словаре. URL: https://habrahabr.ru/post/114997.

20. Рекомендательные системы: LDA. 2014. URL: https://habrahabr.ru/company/surfingbird/blog/150607.

21. Хотилин М. И., Благов А. В. Визуальное представление и кластерный анализ социальных сетей // Материалы Международной конференции ИТНТ. Самара, 2016. С. 1067-1072.

Учебное издание

***Благов Александр Владимирович***
***Рыцарев Игорь Андреевич***

**АНАЛИЗ СОЦИАЛЬНЫХ СЕТЕЙ**

*Учебное пособие*

Редактор А.В. Ярославцева
Компьютерная верстка А.В. Ярославцевой

Подписано в печать 03.12.2020. Формат 60×84 1/16.
Бумага офсетная. Печ. л. 6,5.
Тираж 120 экз. (1-й з-д 1–25). Заказ . Арт. – 13(РЗУ)/2020.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, САМАРА, МОСКОВСКОЕ ШОССЕ 34.

---

Издательство Самарского университета.
443086, Самара, Московское шоссе, 34.