МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ
В ГЭК И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
к.т.н., доцент
М. С. Воробьева
24.06. 2019 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**
(магистерская диссертация)

# РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ АНАЛИЗА СЛОВАРНОГО СОСТАВА ТЕКСТОВ ВЫПУСКНЫХ КВАЛИФИКАЦИОННЫХ РАБОТ

02.04.03. Математическое обеспечение и администрирование информационных систем
Магистерская программа «Разработка, администрирование и защита вычислительных систем»


Выполнила работу
Студентка 2 курса
очной формы обучения
Сунгани
Секаи

Научный руководитель
Д.п.н Профессор
Захарова
Ирина
Гелевна

Рецензент
К.н доцент Каф
информационных систем
Бидуля
Юлия
Владмировна.


г. Тюмень, 2019

## Введение

Современные методы обработки и анализа текста позволяет автоматически извлекать и классифицировать информацию, содержащуюся в различных текстовых документах. Существует много видов анализа текста, предназначенных для разных целей. Оценка текстов студенческих работ является примером анализа текста. Это может быть, как просто подсчет количества слов, так и анализ текста, позволяющий определить, были ли написаны несколько документов одним и тем же автором. Из текста может быть извлечена конкретная информация, такая как ключевые слова, имена или учетные данные. Можно также категоризировать текст согласно теме или настроению.  Даже обычная сортировка данных может быть повторяющимся, трудоемким и дорогостоящим процессом, когда выполняется людьми. При этом большие объемы текста могут автоматически анализироваться компьютерами, экономя время и предоставляя больше информации из данных. Анализ текстов направлен на получение описания содержания, структуры и назначения (смысла) отдельных элементов текста в документах. Важно учитывать тип изучаемых текстовых документов, чтобы определить правильный подход к их анализу.

Целью данной исследовательской работы является разработка инструмента для анализа текста для русского языка, который позволяет сравнивать и визуализировать особенности содержания текстов выпускных квалификационных работ студентов бакалавриата и магистратуры.

Для достижения этой цели были поставлены следующие задачи:

1. Изучить тенденции в области обработки естественного языка и области, в которых такая обработка может применяться.

2. Изучить теорию обработки естественного языка и анализа текста. Проанализировать и выбрать язык программирования, библиотеки и методы, необходимые для реализации приложения.
3. Разработать приложение и с его помощью проанализировать особенности текстов выпускных квалификационных работ.

## Глава 1: Введение в Python Текст Обработка

Обработка текста на Python часто используется для Natural Language Processing (NLP,обработка естественного языка). Компоненты NLP– Natural Language Understanding(NLU-понимание естественного языка) и Natural Language Generation(NLG-Генерация естественного языка). NLU включает отображение входных данных в полезные представления и анализ различных аспектов текста.

NLG облегчает извлечение и поиск информации, анализ настроений и т . д. NLG включает в себя планирование текста, планирование предложений и реализацию текста[1].

- Text Planning (планирование текста) - для извлечения контента из источника данных.
- Sentence Planning (планирование предложения)- выбор релевантных слов, фраз со значением и заданным тоном предложения.
- Реализация текста - представление плана предложения в структуре

NLP применяется в поисковых системах, анализе каналов для газет , чатботов, проверки орфографии, подбора рекламы, приложений машинного перевода как Google Translate и для голосовых приложений, таких как Siri , Cortana и Alexa. Python Natural Language Toolkit (NLTK) -

это набор библиотек, которые могут создавать такие системы обработки текста.

Может также применяться для следующих систем:

**Summarization (обобщение)** - для суммирования длинного текста с различными параметрами, например, какой процент от документа должен быть в итоговом резюме[2].

**Web-scrapping(скрапинг)**: извлекает текст из веб-скрапинг и включает в себя обработку текста. Например, если нам нужно извлечь только заголовки, присутствующие на html-странице, тогда мы ищем тег h1 в структуре страницы и извлекаем текст между тегами h1 . Для этого нужна программа обработки текста из python[3].

**Фильтрация спама** : выявляет и устраняет спам, анализируя текст в строке темы и текст в теле письма [4].

**Sentimental Analysis(Анализ настроений)**: анализ положительных и отрицательных отзывов от текста. Измерение частоты обратной связи, чтобы найти общее настроение[5].

В рамках настоящего исследования основное внимание будет уделено приложениям предназначеннным обработки текстового анализа поисковых данных.

### 1.1 Задачи исследования

Анализ настроений в тексте используется во многих приложениях, но обычно текст исследуется на английском языке.К настоящему времени много работы по анализу текста и обработка етественного языка с помощью машинным обучением, но они в основном сосредоточены на английский язык. Текстовая аналитика стала настолько популярной , что ведущие IT-компании теперь

предлагают свои собственные текстовые аналитические продукты. IBM разработала SPSS Text Analytics[6], SAS имеет программное обеспечение Text Miner[7], SAP предлагает HANA Text Analytics[8], а
Oracle предлагает функции интеллектуального анализа текста в своем Data Miner[9].

## 1.2 Цель и задачи

Целью данного исследования является разработка инструмента анализа текста для русского языка. Анализ текста используется во многих приложениях, но обычно текст на английском языке. Цель состоит в том, чтобы построить русский корпус из электронных текстов.[10]определяет (corpus) корпус как совокупность текстовых документов, а (corpora) хранилище - это множество корпусов. Пользовательский корпус - это набор текстовых файлов в каталоге
.

### 1.2.1 Задачи

1. Собрать квалификационных работ в формате PDF для студентов бакалавриата и магистратуры
2. Конвертировать PDF- файлы в TXT
3. Очистить данные ( удалить стоп-слово, токенизировать, удалить цифры и знаки препинания)
4. Создать словарь текстов и их частоты
5. Выполнить кластеризацию
6. Показать результаты в виде облаков слов и графиков

## 1.3 Данные

Количество текстов, необходимых для обучения модели машинного обучения, зависит от сложности того, чего вы хотите достичь, и количества задействованных тегов.

Например:

- Для определения темы требуется около 250 примеров на тег (тему).
- Анализ настроений требует не менее 500 примеров для каждого тега (настроения), чтобы получить хорошие результаты.

PDF-файлы могут хранить текст, но содержат дополнительную информацию, которая может не понадобиться для анализа. Необходимо конвертировать файлы PDF в файлы .txt для более удобного управления

Модуль PyPDF.2 можно использовать для того чтобы конвертировать файлы PDF в файлы .txt. Полученные в результате файлы .txt могут храниться так же, как мы сохраняем файлы PDF .

## 1. 4 Экспериментальные инструменты

**Python** : язык программирования общего назначения, который относится к интерпретируемым и высокоуровневым языками [11].

**Spyder**: научная среда, написанная на Python и предназначенная для ученых, инженеров и аналитиков данных. Он обладает сочетанием расширенных функций редактирования, анализа, отладки и профилирования инструмента разработки. В качестве научного пакета он предлагает исследование данных, интерактивное выполнение кода, глубокую проверку и высококачественную визуализацию [12].

**Anaconda**:программный продкут с открытым исходным кодом, предоставяющии среду разработка для языков программирования Python и R для научных вычислений (наука о данных, приложения машинного обучения, крупномасштабная обработка данных, прогнозная аналитика и т. д.). Это упрощает управление пакетами и их развертывание. Версии пакетов управляются системой управления пакетами conda . Пакеты с открытым исходным кодом, такие как NLTK и Gensim, можно установить отдельно из репозитория Anaconda с помощью команды conda install или с помощью команды pip install, которая устанавливается вместе с Anaconda.[13]

**Natural Language Tool Kit (NLTK):** ведущая платформа для создания программ Python для работы с данными на естественном языке. Она предоставляет библиотеки для обработки текста для классификации, токенизации, определения происхождения, тегов, анализа и семантического анализа.[14]

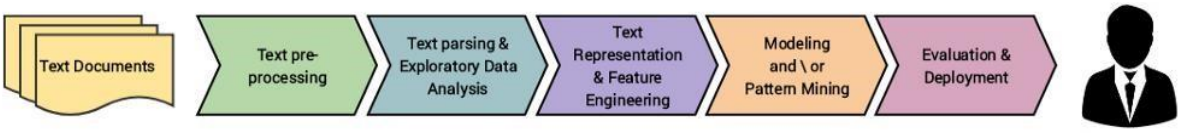
**Scikit -Learn:** машинного обучения библиотека ориентированная на задачи для языка программирования Python . Она включает различные алгоритмы классификации , регрессии и кластеризации.[15]

**Matplotlib:** 2D-графическая библиотека Python, которая создает показатели качества публикации в различных форматах и интерактивных средах [16]

## 1. 5 Межотраслевой стандартный процесс для интеллектуального анализа данных

CRISP- DM *(Cross-Industry Standard Process for Data Mining)* модель является отраслевым стандартом для выполнения каких - либо данных научно -технических проектов[17]. В области Data Science как правило, любая проблема в области NLP может быть решена в рамках

рабочего процесса, который включает последовательность шагов, как показано на рисунке ниже.



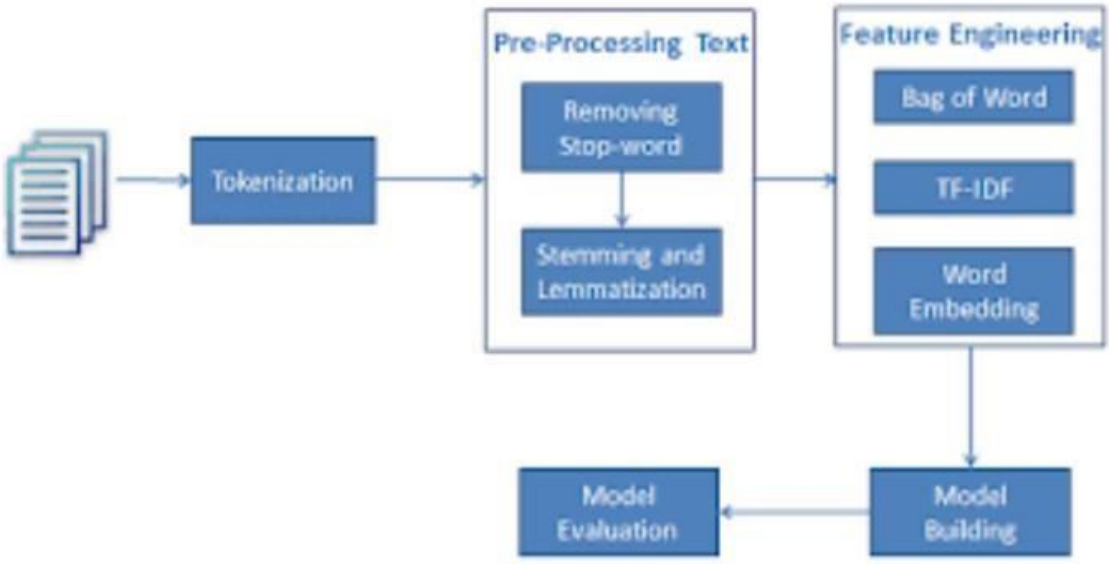




**Рис 1.1 CRISP-DM для проблемы в области NLP**

Проект начинается с формирования  корпуса текстовых документов и следует стандартным процессам для предварительной обработки текста и выполнения основного исследовательского анализа данных. Текст затем представляется с использованием  методов выделения и отбора признаков. В зависимости от проблемы, мы строим модель решения. Наконец, мы оцениваем модель и общие критерии успеха проекта и внедряем модель.

### 1.5.1 Базовая предварительная обработка

[18]выделяет следующие ключевые моменты и термины для работы с текстом:

- Тексты представлены в Python с использованием списков: ['Monty', 'Python'].
- Термин «токен» - это появление данного слова в тексте.
- Термин «словоформа» - это уникальная форма слова как последовательности букв. Мы подсчитываем токены слов, используя функцую `len(text)`, и словоформа, используя функцую `len(set(text))`.
- Чтобы получить словарь текста t, используем функцую `sorted(set(t))`.
- Для работы с каждым элементом текста используем функцую `[f(x) for x in text]`.
- Чтобы получить словарь, сворачивая различия в регистре и игнорируя знаки препинания, используем `set([ w.lower () for w in text, if w.isalpha ()])`.
- Для обработки каждого слова в тексте используется оператор for, например, для w в t: или для word в тексте :. За этим должен следовать символ двоеточия

Предобработка преобразует исходные тексты в смысловые единицы , которые содержат важные лингвистические особенности , прежде чем будет решается задач NLP . [19] отмечает, что строковые методы являются основными инструментами обработки текста в Python. Эти методы позволяют разделить строку на подстроки, выполнит преобразования регистра, проверить содержания строки и удалить пробельные символы из строки.

- Перевод текста в *нижний регистр* позваляют избежать несколько копий одних и тех же слов, как . Например, при подсчете слов « Слово » и « слово » считаются разными словами.

- *Удаление знаков пунктуации* уменьшит размер текста и удалит все отметки, которые не добавляют никакой информации для анализа.
- *Токенизация предложений* разбивает документ на предложения. Есть несколько способов сделать это, например с использованием регулярных выражений.Также для это можно использовать библиотеки NLTK и SpaCy. Тем не менее, большинство разработчиков утверждают , что для предложения токенизации, NLTK предпочтительнее Spacy. NLTK поддерживает различные языки, тогда как SpaCy поддерживает английский, немецкий, испанский, французский, португальский, итальянский и голландский языки но не подходит для токенизации русского текста. Мы можем маркировать предложения в абзаце, используя метод NLTK `sent_tokenize` .
- *Токенизация слов* разбивает большую часть текста на слова. Это требуется в задачах обработки естественного языка, где каждое слово должно быть взято и проанализировано путем классификации и подсчета настроения и т. д. Метод `word_tokenize` используется для разделения абзаца на отдельные слова.

На рисунке 2 показаны некоторые из доступных инструментов Stemming . Полный список доступен по адресу[1]

1. https://docs.google.com/spreadsheets/d/19rMhfcmxFv2V2Q5ZWn1FfLDZZYsuwb1eoSp9CiEEOg/edit?usp=sharing

| Name | Developer, Initial release | Features | Programming languages | License |
|---|---|---|---|---|
| Natural Language Toolkit (NLTK) | The University of Pennsylvania, 2001 | Mac/Unix/Windows support<br>Contains many corpora, toy grammars, trained models, etc [1]. | Python | Apache License Version 2.0. |
| TextBlob | Steven Loria, 2013 | Splitting text into words and sentences<br>WordNet integration [2] | Python | http://textblob.readthedocs.io/en/dev/license.html |
| Spacy | Explosion AI, 2016 | Runs on Unix/Linux, MacOS/OS X, and Windows.<br>Neural network models<br>multi-language support [3] | Python | MIT License |
| Gensim | RaRe Technologies, 2009 | Can process large, web-scale corpora<br>Runs on Linux, Windows and OS X<br>Vector space modeling and topic modeling [4] | Python | GNU LGPLv2.1 license |
| Apache OpenNLP | Apache Software Foundation, 2004 | Contains a large number of pre-built models for a variety of languages<br>Includes annotated text resources [5] | Java | Apache License, Version 2.0 |
| OpenNMT | Yoon Kim, harvardnlp, 2016 | Is a generic deep learning framework mainly specialized in sequence-to-sequence models<br>Can be used either via command line applications, client-server, or libraries. [6]<br>Has currently 3 main implementations (OpenNMT-lua, OpenNMT-py, OpenNMT-tf) | Python<br>Lua | MIT License |

**Рис 1.2. Инструменты токенизации**

- [20] описывает *стоп-слова* как обычные слова, которые не помогают в выборе документов, которые нужны пользователю. Библиотека Python NLTK предоставляет нам корпус стоп-слов по умолчанию, который мы можем использовать для остановки слов в тексте . Каждый язык предоставляет много списков для стоп-слов . В зависимости от вашего текста и необходимого анализа вы можете добавлять слова в эти списки или удалять их .

Русские слова сопряжены в предложениях с помощью окончаний. Например , следующие слова все означают то же самое (большой): Большой, больше, больший, Большим, большой, большая, Больш ие. Учитывая это, просто подсчет количества уникальных слов не имеет особого смысла.

- *Стеммиг* удаляет грамматические оканчания и суффиксы слов, в результате чего получаются строки, которые короче словарных слов.

Цель стеммига объяснения, как объяснил Мэннинг (2009), состоит в том, чтобы свести слова к общей базовой форме.

Python поддерживает следующие стеммеры:

**Porter Стеммер** - алгоритм не очень агрессивен при переводе слов в их корневую форму.

**Lancaster Стеммер** - Это более агрессивно, чем Ланкастер

**Snowball Стеммер** - улучшение по сравнению с Porter, быстрее, чем Porter с точки зрения вычислительное время.

На рис. 1.3 перечислены алгоритмы стемминга, поддерживающие Python.

| Name | Developer, Initial release | Features | Programming languages | License |
|---|---|---|---|---|
| Natural Language Toolkit (NLTK) | The University of Pennsylvania, 2001 | Includes Porter stemmer, Snowball stemmer, and Lancaster stemmer<br>Contains a stemmer that uses regular expressions to identify morphological affixes<br>multiple languages support [15] | Python | Apache License Version 2.0. |
| Snowball | Martin Porter, 2002 | Multiple languages support<br>Small string processing language designed for creating stemming algorithms [16] | Java<br>Python Wrappers are provided | the 3-clause BSD License |
| PyStemmer | Richard Boulton, 2006 | Efficient algorithms for calculating a "stemmed" form of a word<br>Provides algorithms for several (mainly European) languages<br>Porter stemming algorithm for English [17] | MPython | MIT license |

**Рис 1.3. Инструменты для стемминга**

Базовый поток преобразования текста токенизации, удаления стоп-слов и определения происхождения текста на русском языке показан на рисунке 1.4.

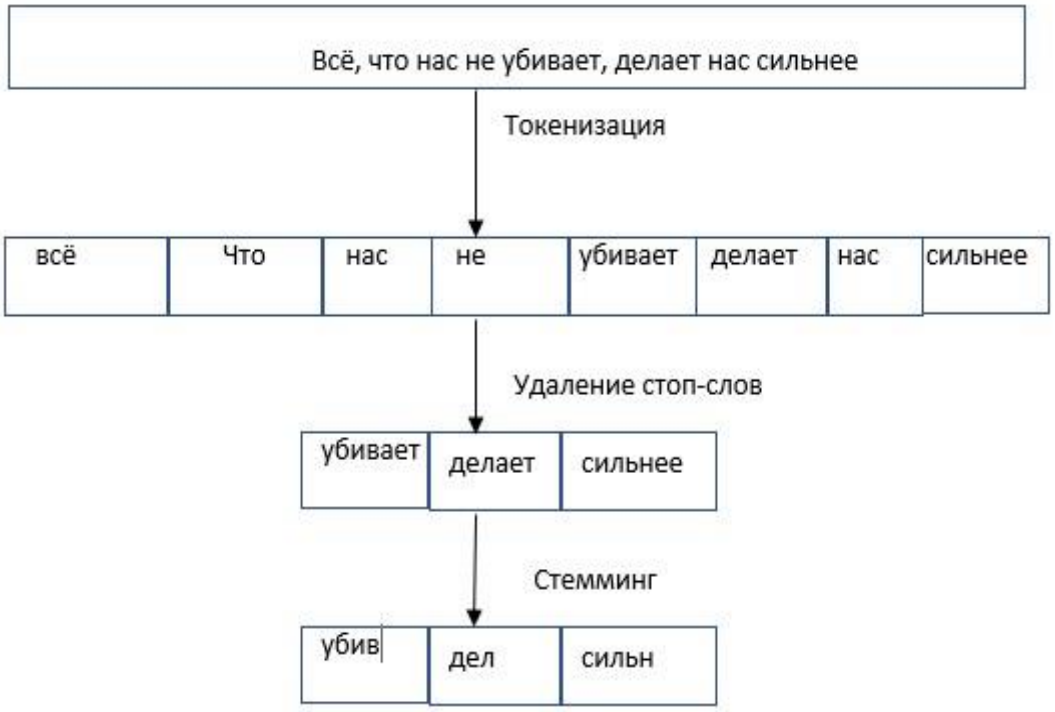



**Рис 1.4. Токенизация, удаление стоп слов и стемминг для русского текста**

- Некоторые слова, возникающие в результате стемминга, не являются реальными словами, которые можно найти в словаре. *Лемматизация* похожа на стемминга, но дает вывод реальных слов.
- Установка *тегов* грамматических категорий слов. Токенизация и функция `pos_tag` используются для создания тегов для слов. NLTK может представлять слова как глаголы, существительные, прилагательные и многое другое одним из следующих классов:

| | | | | | |
|---|---|---|---|---|---|
| CC | Coordinating conjunction | NNS | Noun, plural | UH | Interjection |
| CD | Cardinal number | NNP | Proper noun, singular | VB | Verb, base form |
| DT | Determiner | NNPS | Proper noun, plural | VBD | Verb, past tense |
| EX | Existential there | PDT | Predeterminer | VBG | Verb, gerund or present participle |
| FW | Foreign word | POS | Possessive ending | VBN | Verb, past participle |
| IN | Preposition or subordinating conjunction | PRP | Personal pronoun | VBP | Verb, non-3rd person singular present |
| JJ | Adjective | PRP$ | Possessive pronoun | VBZ | Verb, 3rd person singular present |
| JJR | Adjective, comparative | RB | Adverb | WDT | Wh-determiner |
| JJS | Adjective, superlative | RBR | Adverb, comparative | WP | Wh-pronoun |
| LS | List item marker | RBS | Adverb, superlative | WP$ | Possessive wh-pronoun |
| MD | Modal | RP | Particle | WRB | Wh-adverb |
| NN | Noun, singular or mass | SYM | Symbol | | |
| | | TO | to | | |

Рис 1.5. Part Of Speech tagging

- *Bag of words(BOW)* преобразует текст в числовые векторы. Это набор всех слов, найденных во всех документах. Каждое предложение отображается в виде вектора длины, равной размеру словаря. Каждая координата вектор представляет собой одно из слов в словаре.Значение координаты вектора показывает, сколько раз это слово появилось в предложении. Имена столбцов являются базовыми словами, а значения могут быть следующими:

1. Двоичный, который указывает, присутствует ли слово / отсутствует в данном документе
2. Частота, которая указывает количество появлений слова в данном документе
3. TFIDF, специальная оценка уникальность слова

*Bag of words* представляет тексты, игнорируя порядок слов и только проверяет

присутствие/отсутствие слов . Одним из основных недостатков использования представления bag of words является то, что слово порядок полностью отбрасывается. Таким образом, две строки «это плохо, совсем не хорошо» и

«Это хорошо, совсем не плохо» имеют одинаковое представление, хотя смисль их противоно. Bag of words не правильно отражает отрицания. Это также терпит неудачу со словами с ошибками, например, это будет рассчитывать «база данных» и « базы данных » как разные слова. Представляя предложения в качествевекторы, мешок слов теряет упорядоченный характер текста, что приводит к потере очень важной информации .

**Улучшения по сравнению с Bag of Words**

*Tf- IDF Взвешивание*

В больших текстах часто встречающиеся слова (например, а, или, но,и) не содержат много значимой информации о конкретных предложениях. Это может быть более ценно для повышение веса слов, которые встречаются реже, чем тяжелее слов, которые встречаются чаще.

Метод Tf -IDF преобразует количество слов в значения с плавающей точкой. Это достигается с помощью следующего соотношения:

$$TFIDF\,(t, d) = tf\,(t, d) \times idf\,(t) \qquad (1)$$

Term Frequency (Частота Термина,TF) означает, сколько раз термин встречается в документе (который в данном случае является предложением). Это отношение количества слов в предложении к длине предложения.

$$TF = \frac{\text{Количество раз,когда термин T появляется в строке}}{\text{количество терминов в этой строке}} \qquad (2)$$

Матрица частоты терминов (TF) :

Это техника, используемая для определения релевантности слова в документе. Слово , которое появляется частое Ly является более актуальным в контексте.

[20] подробно описывает следующий двухэтапный процесс представленого для текста, в виде bag of words:

1. Каждому слову в документе присваивается целое число и сохраняется в словаре

2. Для каждого документа создается вектор. Столбцы векторов соотвествуют фактическим словам в тексте. Они формируют признаки. Значения элементов векторов определяются через TF-IDF TFIDF.

$$idf(t) = \log \frac{1 + n_d}{1 + df(d,t)} + 1 \quad (3)$$

$\mathbf{N_d}$ - общее количество документов, df ( d, t ) - количество документов, содержащих слово т . Эти векторы затем нормализуются евклидовой нормой :

$$v_{norm} = \frac{v}{\|v\|^2} = \frac{v}{\sqrt{v_1{}^2 + v_2{}^2 + \cdots + v_n{}^2}} \quad (4)$$

Полученные значения с плавающей точкой затем используются в векторах представляющих докумены

*N-граммовые модели*

Эта модель решает проблемупотери порядка слов в bag of words, включая nграммы. N-граммы объединяют несколько слов, используемых вместе. Nграммы с N = 1 называются униграммами. Точно так же можно использовать биграммы (N = 2), триграммы (N = 3) и т. д. N-граммы отражают структуру языка, например, какая буква или слово могут следовать за данным. Чем длиннее n-грамм (чем выше n), тем большебконтекста можно учесть.

Библиотека Scikit -Learn содержит очень эффективный пакет представления слов в виде класса CountVectorizer . Класс позволяет учесть любое Nграммовое представление информации. Представление текста с помощью TFIDF осуществляется с использованием класса TfidfVectorizer .

**1.5 .2 Анализ данных**

Исследовательский анализ данных - это процесс выявления закономерностей, аномалий, проверки гипотез и предположений, с использованием сводной статистики и графических представлений. В этом проекте Exploratory Data Analysis (исследовательский анализ данных, EDA) будет выполнять следующие задачи:

- Загрузка текстовых данных в корпус.
- Очистка корпуса
- Создание n-граммовых токенов.
- Обнаружение наиболее общих особенностей корпуса.

С очищенными данными теперь можно анализировать на основе различных параметров. Например, кто-то может быть заинтересован в знании слов,

которые в основном используются студентами в своих диссертациях. Также можно найти, словарный запас, то есть количество уникальных слов, используемых в тексте,

Пакет wordcloud создает список из 200 найболее характерных слов для корпуса и список нормализованных количеств слов для каждого слова. Затем используется библиотека изображений Python для рисования облака слов . Полученные результаты можно визуализировать с помощью WordClouds .Из облаков слов  мы идей представление о том, что представляет собой текст , и пытаемся получить вывод, который имеет смысл.

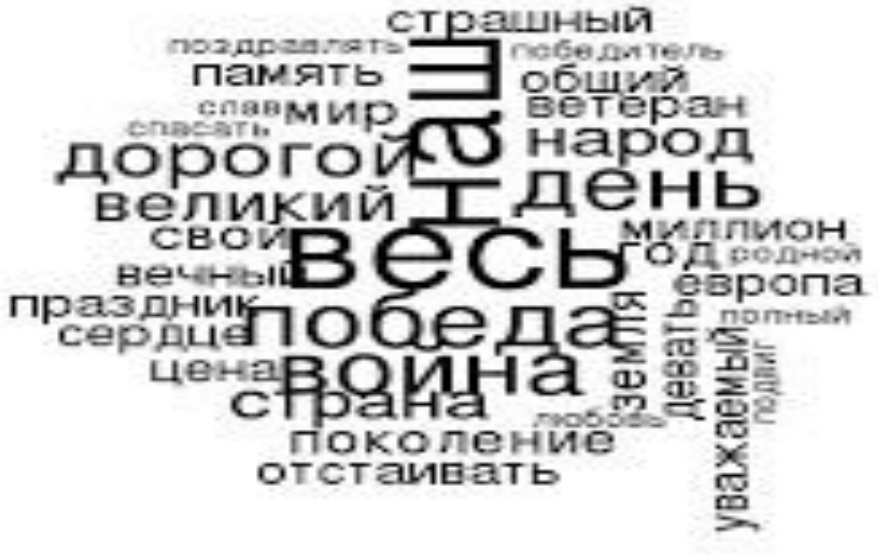



**Рис 1.6. Русское облако слов**

## 1. 6 Заключение

В этой главе освещены известные методы и приемы работы по обработкой текстов в Python . Это дает представление о том, как работать с корпусом текстовых документов выполнять предварительную обработку, анализ для понимания структуры и сущностей текста .

# Глава 2: Методы интеллектуального анализа текстов

Далее освещаются новые разработки и приложения в области интеллектуального анализа данных и обнаружения знанийб ф также обобщаются инструменты и методы, полезные для анализа данных. Анализ данных стал популярным в последние годы. Теперь он опирается на более сложные и разнообразные источники данных; пользователи хотят интегрировать данные из разных источников, изучая данные разных типов.

## 2.1 Анализ текста vs Text Mining vs текстовая аналитика

Термины « Text Mining» и «анализ текста» часто используются для объяснения одного и того же процесса получения данных с помощью изучения статистических шаблонов. Итак, *анализ текста* против *анализа текста* : в чем разница?

[21] объясняет, что текстовая аналитика является качественной, а *анализ текста* количественный. Если машина выполняет анализ текста, она идентифицирует важную информацию в тексте . Текстовая аналитика выявляет закономерности по тысячам текстовых данных, в результате чего появляются графики, отчеты, таблицы и т. д.

Различные типы данных в текстовых записях дают возможность проверить, существуют ли базовые шаблоны в текстовых полях. В этой главе представлены шаги, которые будут предприняты для того, чтобы достигнуть цели исследования. Они включают:

- Сбор данных в виде PDF-файлов и преобразование в TXT
- Очистка данных
- Визуализация данных и исследование
- Кластеризация

### 2.1.1 Сбор данных

Исходуый текст он может содержать акцентированные символы или символы вне латинского алфавита; Кириллица, китайский мандарин и арабский. Работа с таким текстом может дать неожиданный вывод. Эти типы выходных данных также могут возникать при чтении файлов. Причиной этого является кодировка символов, и необходимо явно указать кодировку, например , ('UTF-8', ASCII). Кодировка по умолчанию - UTF-8, если не указано иное.

**Частоты слов**

Частоту слов в тексте можно посчитать с помощью цикла for и счетчика из коллекций. Счетчик оказался быстрее, чем при использовании цикла. Функция частоты слова вернет словарь уникальных слов и их частоту в виде пары ключ-значение.

```
from collections import Counter  def

count_words_fast(text):

    text = text.lower()      skips='''!()-

[]{};:'"\,<>./?@#$%^&*_~'''     for ch in skips:

        result = ''.join(i for i in text if not
i.isdigit())         text = result.replace(ch, "")
word_counts = Counter(text.split(" "))      return
word_counts
```

**Чтение текстов**

Мы создаем функцию `read_text()`, которая будет читать текст, сохранять его как длинную строку в переменной и возвращать его. Параметром функции будет местоположение текста.txt, который нужно прочитать, и он будет передан при вызове функции.

```
def read_text(title_path):        with open(title_path, "r",

encoding ="utf-8") as current_file:

        text = current_file.read()          result =
''.join(i for i in text if not i.isdigit())         text =
result.replace("\n", "").replace("\r", "")      return text
```

Мы рассмотрим свойства текста отдельных текстов в коллекции текстов от студентов на уровне бакалавров и магистратуры и различных программных кодов. Мы рассмотрим длину текста, количество уникальных слов и то, как эти атрибуты группируются по уровню обучения и коду программы.

Следующая функция `word_stats()` будет принимать словарь частоты слов

(`count_words_fast()` / `count_words())` в качестве параметра. Функция возвращает сумму уникальных слов ( сумма / общее количество ключей в словаре частоты слов) и `dict_values`, хранящие общее количество их вместе, в виде кортежа.

```
def word_stats(word_counts):

   num_unique = len(word_counts)    counts =

word_counts.values()     return (num_unique, counts)
```

Мы пишем цикл, который поможет загружать тексты из их местоположения на компьютере в Python. Текстовые файлы упорядочены в папке с именем « диплом ». В папке диплом, у нас есть две папки для студентов бакалавриата (бакалаврь) и другой для (МАГ) магистрантов. В папках бакалавра и магистра мы создаем папку для каждого программного кода, например, код 144

```
import os  import

pandas as pd

book_dir =

"./диплом"

os.listdir(book_dir)

stats = pd.DataFrame(columns =("level", "code", "title", "length",
"unique"))  title_num = 1 for language in

os.listdir(book_dir):      for author in

os.listdir(book_dir+"/"+level):          for title in

os.listdir(book_dir+"/"+level+"/"+code):

inputfile = book_dir+"/"+ level +"/"+code+"/"+title

text = read_text(inputfile)

            (num_unique, counts) = word_stats(count_words_fast(text))

            stats.loc[title_num]= level, code.capitalize(),
title.replace(".txt", ""), sum(counts), num_unique
title_num+= 1
```

### 2.1.2 Очистка данных

Текст, собранный из текстовых файлов, имеет различные типы данных и может содержать информацию, которая не относится к нашему исследованию. Перед тем, как выполнить проверку текста, вызвав функции, созданные выше, нам нужно очистить данные, чтобы они содержали значимые данные, которые полезны для идентификации шаблонов.

Мы используем методы предварительной обработки, выделенные в первой главе:

1. Преобразование текста в нижний регистр

```
lower=text.lower()
```

**2. Удаление цифр**

Удаление чисел из текста, таких как «1,2,3,4,5…». Это особенно полезно при кластеризации текста или получении ключевых фраз, так как числа не придают большого значения. Чтобы удалить цифры, можно использовать :.`isnumeric()` или. `isdigit ()`

```
output = ''.join(c for c in lower if not c.isdigit())
```

**3. Токенизация слов**

```
word_tokens = nltk.word_tokenize(output)
```

**4. Удалить стоп-слова**

```
import nltk

from nltk.corpus import stopwords

stopword = stopwords.words('russian')

removing_stopwords = [word for word in word_tokens if word not in stopwords
```

**5. Стемминг**

```
from nltk.stem import SnowballStemmer

snowball_stemmer = SnowballStemmer('russian')

stemmed_word = [snowball_stemmer.stem(word) for word in
removing_stopwords]
```

**6. Удаление пунктуации**

```
from string import punctuation def

strip_punctuation(s):

    return ' '.join(c for c in s if c not in punctuation)

punct=strip_punctuation(stemmed_word)
```

### 2.1.3 Визуализация и исследование данных

Мы вызываем функцию count_words_ fast ( ) для отображения количества слов в тексте:

```
count_words_fast(punct)
```

Вывод представляет собой словарь ключей ( слов в текстах) и их значений (частота слова в тексте):

```
Counter({'': 1280, 'тестирован': 154, 'i': 109, 'фактор': 107, 'набор': 106, 'тестов': 105, '…': 93, '-': 90, 'j': 87,
'метод': 80, 'l': 76, 'f': 74, 'тест': 61, 'count': 61, 'l': 59, 'количеств': 58, 'алгоритм': 56, 'кажд': 55, 'котор':
53, 'перемен': 50, 'значен': 49, 't': 48, 'int': 48, 'o': 47, '```': 46, 'приложен': 44, 'дан': 44, 'эт': 40, 'n': 38,
'm': 34, 'подход': 33, 'образ': 32, 'цикл': 32, 'имеет': 31, 'k': 31, 'for': 31, 'проверок': 30, 'selenium': 29, 'li':
29, 'numberoftests': 29, 'так': 28, 'перв': 28, 'нахожден': 27, 'сложност': 27, 'работ': 27, 'продукт': 27, 'сценар': 26,
'рис': 26, 'программн': 25, 'след': 25, 'см': 25, 'list': 25, 'string': 24, 'testing': 23, 'ящик': 22, 'параметр': 22,
'пользовател': 22, 'класс': 21, 'автоматическ': 21, 'c': 21, 'равн': 20, 'i++': 20, 'проверк': 19, 'создан': 19,
'необходим': 19, 'помощ': 19, 'код': 19, 'lN': 19, 'driverfindelement': 19, 'обеспечен': 18, 'оценк': 18, 'imag': 18,
'одн': 18, 'созда': 18, 'возможн': 18, 'выбра': 18, 'click': 18, 'программ': 17, 'явля': 17, 'числ': 17, 'име': 17,
'вебприложен': 16, 'такж': 16, 'максимальн': 16, 'браузер': 16, 'step': 16, 'nm': 16, 'bytagnam': 16, 'полн': 15,
'атомарн': 15, 'loader': 15, 'таблиц': 15, 'выполня': 15, 'скорост': 15, 'перебор': 14, 'содерж': 14, 'factors': 14,
'if': 14, 'ручн': 13, 'разработк': 13, 'минимальн': 13, 'процесс': 13, 'результат': 13, 'част': 13, 'друг': 13, 'lN-':
13, '…l': 13, 'втор': 13, 'new': 13, 'вид': 12, 'бел': 12, 'сравнен': 12, 'использован': 12, 'разработа': 12, 'модул':
12, 'представля': 12, 'лиш': 12, 'fn': 12, 'c': 12, 'проведен': 11, 'черн': 11, 'сер': 11, 'различн': 11, 'компан': 11,
'времен': 11, 'могут': 11, 'команд': 11, 'fcount': 11, 'глав': 10, 'автоматизац': 10, 'попарн': 10, 'creation': 10,
'customization': 10, 'сервис': 10, 'разрабатыва': 10, систем': 10, 'позволя': 10, 'соб': 10, 'провод': 10, 'сам': 10,
'имен': 10, 'fi': 10, 'max': 10, 'модульн': 9, 'задач': 9, 'генерац': 9, 'test': 9, 'случа': 9, 'врем': 9, 'поэт': 9,
'вход': 9, 'использ': 9, 'запуска': 9, 'r': 9, 'вложен': 9, 'return': 9, 'add': 9, 'системн': 8, 'интеграцион': 8,
'описан': 8, 'пример': 8, 'цел': 8, 'schlumberger': 8, 'качеств': 8, 'функц': 8, 'получ': 8, 'обычн': 8, 'функциональн':
8, '–': 8, 'завис': 8, 'доступ': 8, 'наход': 8, '«': 8, '»': 8, 'сво': 8, 'вебсервер': 8, 'состо': 8, 'скольк': 8, 'and':
8, 'файл': 8, 'values': 8, 'свободн': 8, 'j++': 8, 'links': 8, 'определен': 7, 'приемочн': 7, 'автоматизирова': 7,
```

**Рис 2.1 Словарь с частотой слов**

**Облака слов**

WordClouds(Облако слов) - это метод, используемый для представления текстовых данных. Размер каждого слова указывает на его частоту или важность.

```
from wordcloud import WordCloud import

matplotlib.pyplot as plt  import pandas as pd

wordcloud = WordCloud(width = 800, height = 800,

background_color ='black',

min_font_size = 10).generate(punct)

plt.figure(figsize = (8, 8), facecolor = None)

plt.imshow(wordcloud)  plt.axis("off")

plt.tight_layout(pad = 0)  plt.show()
```

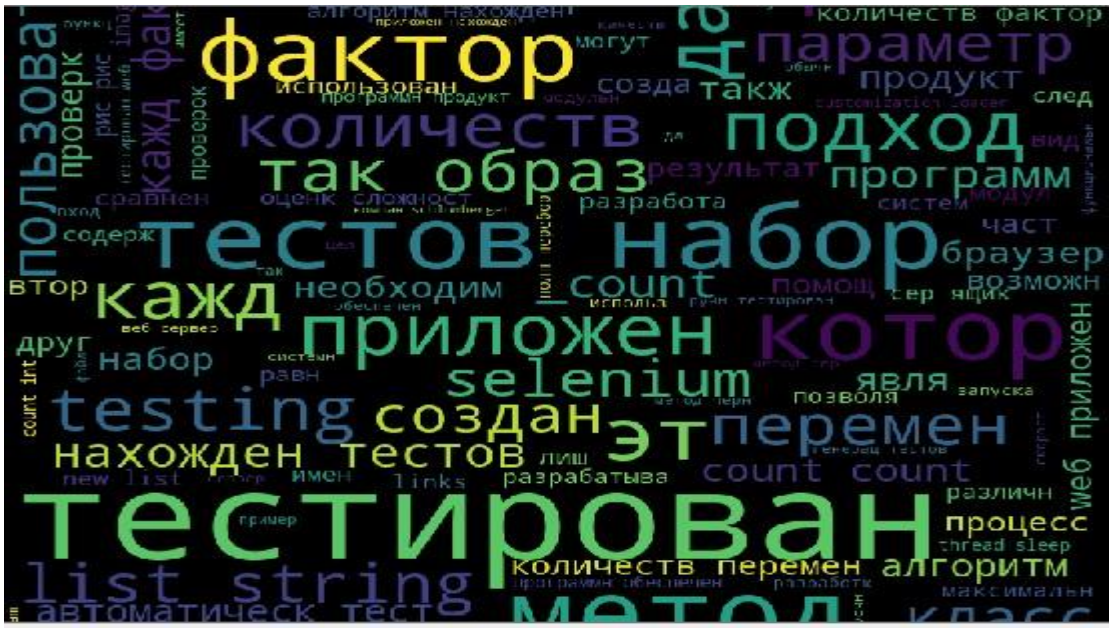



**Рис 2.2 Облако слов из текста дипломной работы**

Мы строим график показывающий сравнение длины текста в зависимости с количеством уникальных слов для всех текстов учебных уровней, используя matplotlib . Мы используем библиотеку pandas для созданиядata frame. Информация о текстах из дипломной работы будет храниться в

информационной рамке в виде столбцов. Мы будем классифицировать эти столбцы по различным категориям, таким как «уровень», «код», «заголовок», «длина» и «уникальный » . Чтобы отобразить длину текста по оси x и количество уникальных слов по оси y, мы используем:

```
import matplotlib.pyplot as plt

plt.plot(stats.length, stats.unique, "bo-")

plt.loglog(stats.length, stats.unique,

"ro")    stats[stats.level =="маг"]

plt.figure(figsize =(10, 10))  subset =

stats[stats.level =="маг"]

plt.loglog(subset.length, subset.unique, "o", label ="маг", color
="orange")  subset = stats[stats.level

=="Бакалаврь"]

plt.loglog(subset.length, subset.unique, "o", label ="Бакалаврь",
color ="forestgreen")  plt.legend()  plt.xlabel("Длина текста ")
plt.ylabel("Количество уникальных слов ")  plt.savefig("fig.pdf")
plt.show()
```

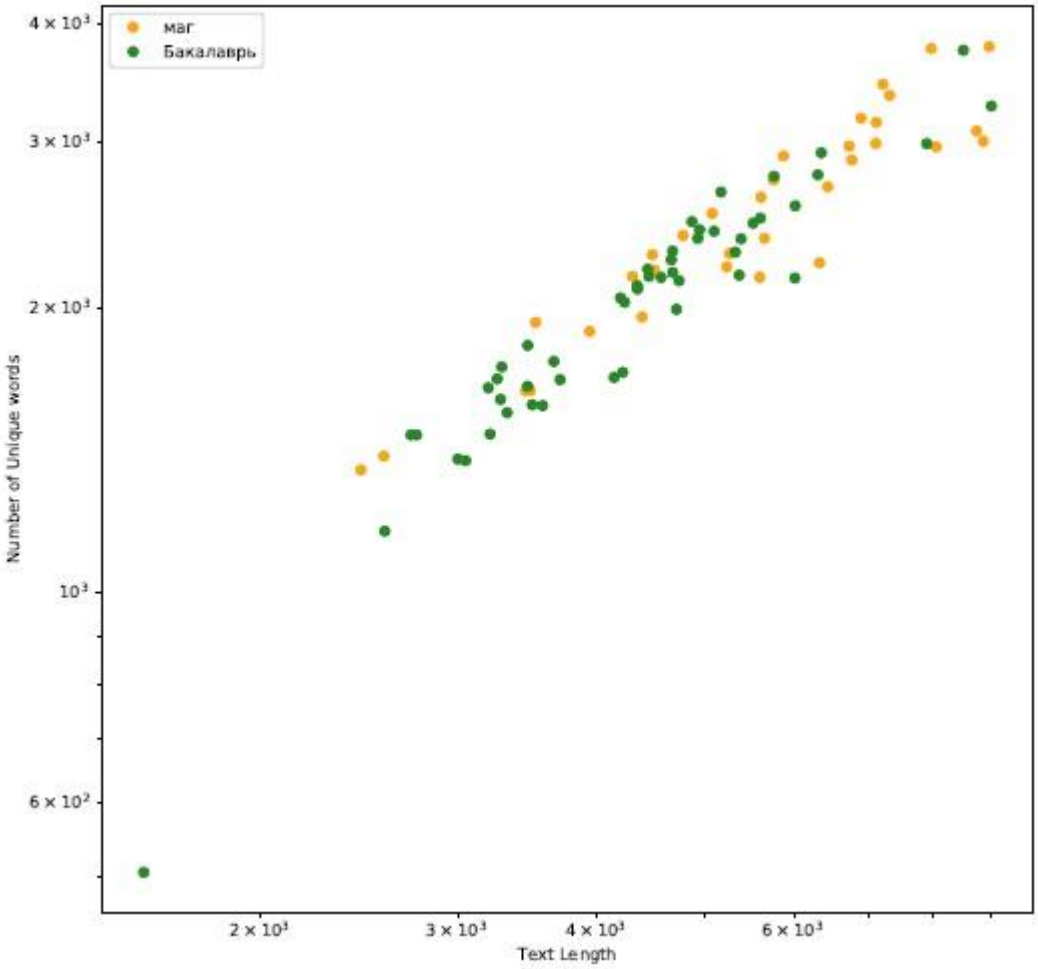



**Рис 2.2 График уникальных слов в сравнении с длинои текста для студентов магистратуры и бакалавров.**

## Гистограммы

Гистограммы представляют собой графическое представление распределения числовых данных. Они показывают оценку распределения вероятностей непрерывной переменной [22]. Гистограмма - это своего рода гистограмма. Чтобы построить гистограмму, разделите весь диапазон значений на ряд интервалов, а затем подсчитайте, сколько значений попадает в каждый интервал. Bins обычно задаются как последовательные, неперекрывающиеся интервалы переменной. Bins (интервалы) должны быть смежными и часто одинакового размера

Гистограммы для подсчетов слов для текстов магистрантов и бакалавров строятся с помощью matplotlib. Также проиллюстрирована общая гистограмма, включающая работу обеих групп.

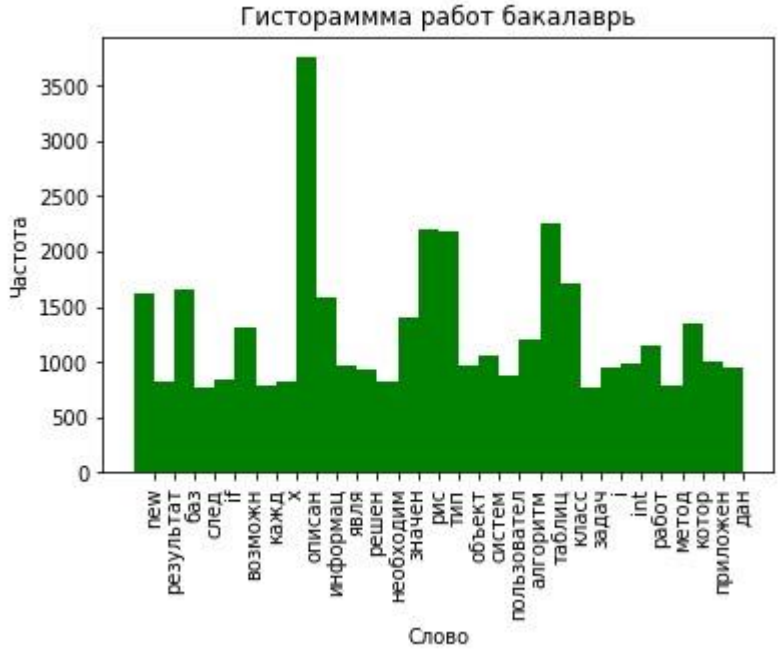



**Рис 2.3 Гистограмма для бакалавриата**

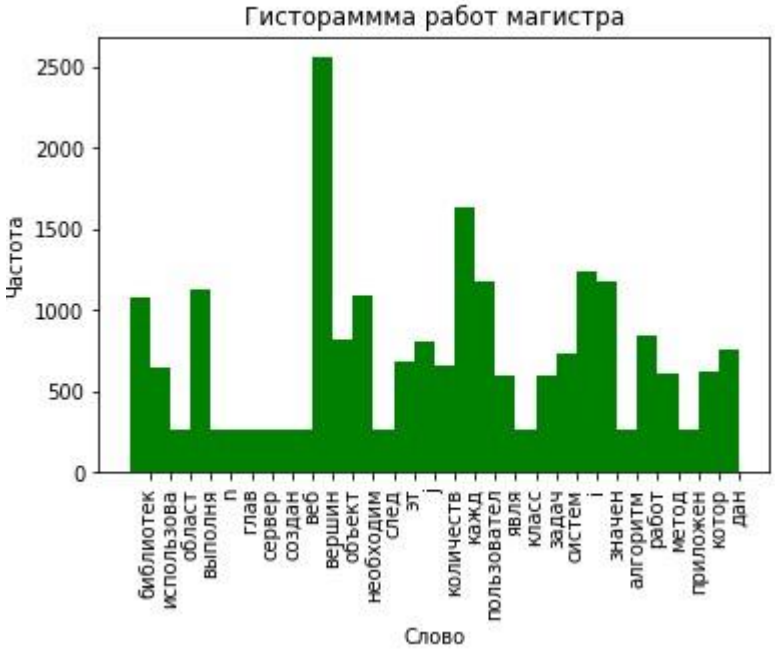



**Рис 2.4 Гистограмма для магистерской работы**

### 2.1.4 Кластеризация

Текстовые кластеры группируют большие объемы неструктурированных данных. Алгоритмы кластеризации менее точны, чем алгоритмы классификации, но быстрее реализуются . Они извлекают информацию и делают прогнозы без использования обучающих данных. Это форма неконтролируемого машинного обучения.

Кластеризация представляет собой способ группировки элементов , так что подобные детали в один кластер элементов и ди FFE г Ent находятся в отдельные кластеры. Алгоритмы могут быть плоскими или иерархическими. Плоская кластеризация  разделяет текст,не связывая кластеры с друг с другом. Многие плоские алгоритмы кластеризации требуют, чтобы количество кластеров было указано заранее.

В иерархической кластеризации количество кластеров указывать не нужно. Такие алгоритми кластеризации создают иерархию кластеров. В то время как похожие тексты сгруппированы в один кластер, похожие кластеры снова сгруппированы в один более крупный кластер.

Scikit предлагает несколько подходов к кластеризации в пакете sklearn.cluster . KMeans - это широко используемый алгоритм плоской кластеризации, который поддерживает начальное количество кластеров, указанных как центроиды кластеров .

Алгоритм работает следующим образом:

1. Инициализировать k точек
2. Распределять по категориям все элементы до его ближайшего среднего
3. Обновить координаты к точек ( по средним значениям уже категоризированым  элементов одой из k точек),

4. Повторить процесса для заданного числа итераций , и в конце концов, будут построены кластеры.

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

from sklearn.metrics import adjusted_rand_score

class TFVectorizer(TfidfVectorizer):

def build_analyzer(self):

        analyzer = super(CountVectorizer, self).build_analyzer()

        return lambda doc: (stemmer.stem(w) for w in analyzer(doc))

vectorizer = TFVectorizer(min_df = 1, stop_words = stopwords) X

= vectorizer.fit_transform(posts)

true_k = 10

model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100,
n_init=1)

model.fit(X)

print("Top terms per cluster:")

order_centroids = model.cluster_centers_.argsort()[:, ::-1]

terms = vectorizer.get_feature_names()

for i in range(true_k):

    print("Cluster %d:" % i),

    for ind in order_centroids[i, :10]:
print(' %s' % terms[ind]),print
```

```
Cluster 0: путешеств, get, this, match, ap,

Cluster 1: дан, таблиц, int, приложен, задан,

Cluster 2: дан, расписан, задач, событ, работ,

Cluster 3: метод, решен, задач, котор, рис,

Cluster 4: x, свертк, блик, rect, y,

Cluster 5: self, slot, услуг, инцидент, rl,

Cluster 6: луч, звук, звуков, волн, doubl,

Cluster 7: i, распознаван, растен, изображен, doubl,

Cluster 8: фильм, дан, кластер, устройств, пользовател,

Cluster 9: rdr, cmd, билет, getvalu, зал,
```

**Рис 2.5 Кластеры бакалавров**

```
Cluster 0: self, датчик, робот, address, connectbot,

Cluster 1: otabl, информатик, cell, суперкомпьютерн, анкет,

Cluster 2: сниппет, пользовател, социальн, дан, сет,

Cluster 3: изображен, фильтр, пиксел, пищев, котор,

Cluster 4: страниц, тестирован, ym, сайт, веб,

Cluster 5: дан, должност, сотрудник, граф, котор,

Cluster 6: помещен, здан, теплов, го, температурн,

Cluster 7: давлен, крив, col, капиллярн, пласт,

Cluster 8: частиц, j, i, алгоритм, метод,

Cluster 9: робот, задан, работ, робототехник, студент,
```

**Рис 2.6 Кластеры магистра**

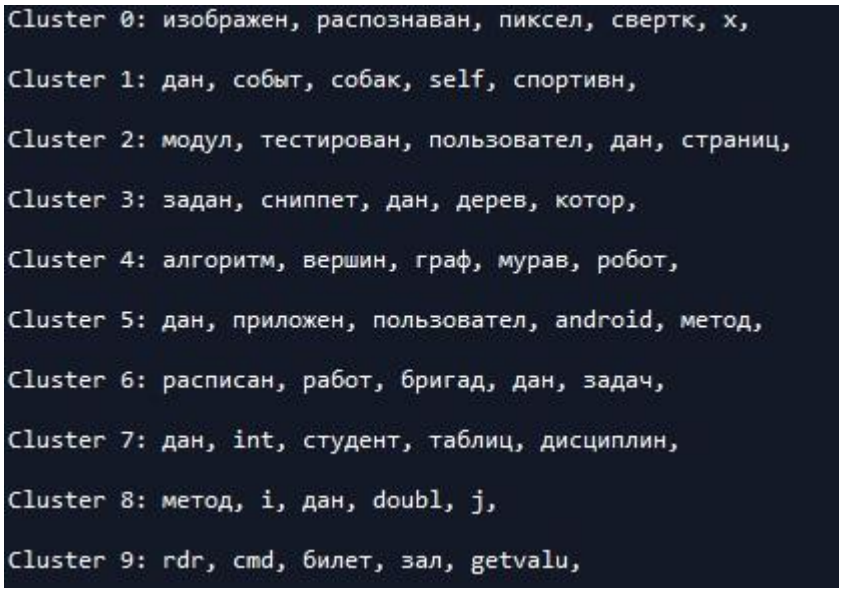



**Рис 2.7 Общие кластеры**

Кластеры показывают 5 наиболее часто используемых слов, связанных с кластером.Из этих слов мы можем получить идеи о том, какие документы входят в кластер. Например, кластеризация магистрант показывает, что кластер 3 имеет документы об изображениях, а кластер 5 может быть связан с компанией или сотрудником. Общий кластер для магистрантов и бакалавров показывает кластер 5 о разработке мобильных приложений и кластер 7 о научной работе

## Заключение

В настоящем исследовании были разработаны программные инструменты для анализа текста выпускных квалификационных работ. Полученные решения могут быть расширены для книг и научно-технических отчетов. Были проанализированы 92 текстовых документа, в том числе, 59 работ студентов бакалавриата и 33 работы студентов магистратуры.

Представленное исследование показало, что магистранты обычно пишут более длинные диссертации, в то время как студенты бакалавриата представляют относительно короткие тексты ВКР. Студенты-бакалавры, как правило, используют меньшее число уникальных слов в своих текстах, в среднем, около 2000 уникальных слов, в то время как студенты магистратуры используют в среднем около 3000 уникальных слов. Результаты были представлены в виде облаков слов для обеих групп, а также гистограмм. В работе приведена общая зависимость между длиной текста в зависимости от количества уникальных слов.

Будущие исследования могут быть распространены на большие данные в компаниях, где данные неструктурированы, а их источниками могут быть твиты, электронные письма, потоки веб-кликов, информация CRM, цепочка поставок.Facebook анонсировал анализ данных, который использует текстовую аналитику, чтобы показать, что аудитория говорит на Facebook о событиях, брендах, темах и действиях. Подобные инструменты могут быть разработаны и для предприятий, чтобы найти полезную информацию о мнениях клиентов и клиентов относительно их продуктов и услуг.

## Код:

```
# -*- coding: utf-8 -*-

import sys

import os

import numpy as np

import re, codecs

from os import listdir

posts = [open(os.path.join('C:\\Users\\sekai\\Desktop\\Final\\TXT',
f)).read() for f in
os.listdir('C:\\Users\\sekai\\Desktop\\Final\\TXT')]

#print(posts[0])



import nltk;

from nltk.corpus import stopwords

stopwords = set(stopwords.words('russian'))



import nltk.stem

stemmer = nltk.stem.SnowballStemmer('russian')

from sklearn.feature_extraction.text import CountVectorizer

class StemmedCountVectorizer(CountVectorizer):
```

```
    def build_analyzer(self):
        analyzer = super(CountVectorizer, self).build_analyzer()
        return lambda doc: (stemmer.stem(w) for w in analyzer(doc))
vectorizer = StemmedCountVectorizer(min_df = 1, stop_words = stopwords,
ngram_range = (1,2)) X = vectorizer.fit_transform(posts)
num_samples, num_features = X.shape
print ('ns= %d, nf= %d' % (num_samples, num_features))
print ('--------------------------------------------------------')
# словарь
freq = np.ravel(X.sum(axis=0)) # sum each columns to get total counts
for each word
import operator
# get vocabulary keys, sorted by value
vocab = [v[0] for v in sorted(vectorizer.vocabulary_.items(),
key=operator.itemgetter(1))]
fdict = dict(zip(vocab, freq)) # return same format as nltk
print ("Fdict: ",fdict)
print ('--------------------------------------------------------')
sorted_fdict = sorted(fdict.items(), key=operator.itemgetter(1))
print ("Sorted_dict: ",sorted_fdict)
# Clustering
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
```

```
from sklearn.metrics import adjusted_rand_score


class TFVectorizer(TfidfVectorizer):

def build_analyzer(self):

        analyzer = super(CountVectorizer, self).build_analyzer()

        return lambda doc: (stemmer.stem(w) for w in analyzer(doc))


vectorizer = TFVectorizer(min_df = 1, stop_words = stopwords) X

= vectorizer.fit_transform(posts)

true_k = 10

model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100,
n_init=1)

model.fit(X)


print("Top terms per cluster:")

order_centroids = model.cluster_centers_.argsort()[:, ::-1]

terms = vectorizer.get_feature_names()

for i in range(true_k):

    print("Cluster %d:" % i),

    for ind in order_centroids[i, :10]:

        print(' %s' % terms[ind]),

    print
```

```
# check frequency of each word
from collections import Counter
def count_words_fast(text):
    text = text.lower()
    skips='''!()-[]{};:'"\,<>./?@#$%^&*_~'''
    for ch in skips:
        result = ''.join(i for i in text if not i.isdigit())
        text = result.replace(ch, "")
    word_counts = Counter(text.split(" "))
    return word_counts #print (word_counts)


#read a book and return it as a string
def read_text(title_path):
    with open(title_path, "r", encoding ="utf-8") as current_file:
        text = current_file.read()
        result = ''.join(i for i in text if not i.isdigit())
        text = result.replace("\n", "").replace("\r", "")
    return text


# number of unique words
def word_stats(word_counts):     # word_counts = count_words_fast(text)
```

```
    num_unique = len(word_counts)
    counts = word_counts.values()
    return (num_unique, counts)
############
import os
import pandas as pd


book_dir = "./диплом"
os.listdir(book_dir)
stats = pd.DataFrame(columns =("language", "author", "title",
"length", "unique"))
title_num = 1


for language in os.listdir(book_dir):
    for author in os.listdir(book_dir+"/"+language):
        for title in os.listdir(book_dir+"/"+language+"/"+author):
inputfile = book_dir+"/"+language+"/"+author+"/"+title
            text = read_text(inputfile)
            (num_unique, counts) = word_stats(count_words_fast(text))
            stats.loc[title_num]= language, author.capitalize(),
title.replace(".txt", ""), sum(counts), num_unique
            title_num+= 1
```

```
##########


import nltk

#def to_lower(text):

#    return ' '.join([w.lower() for w in nltk.word_tokenize(text)])

lower=text.lower()

output = ''.join(c for c in lower if not c.isdigit())

#print(output)

word_tokens = nltk.word_tokenize(output)

#print (word_tokens)


from nltk.corpus import stopwords

stopword = stopwords.words('russian')

removing_stopwords = [word for word in word_tokens if word not in
stopword]

#print (removing_stopwords)


from nltk.stem import SnowballStemmer

snowball_stemmer = SnowballStemmer('russian')

stemmed_word = [snowball_stemmer.stem(word) for word in
removing_stopwords]

#print (stemmed_word)
```

```
from string import punctuation

def strip_punctuation(s):

    return ' '.join(c for c in s if c not in punctuation)

punct=strip_punctuation(stemmed_word)

#print (punct)

clust=count_words_fast(punct)

import operator

sorted_fdict = sorted(clust.items(), key=operator.itemgetter(1))

print(sorted_fdict) #show

word counts

#count_words_fast(punct)

# #wordclouds

from wordcloud import WordCloud

import matplotlib.pyplot as plt

import pandas as pd

wordcloud = WordCloud(width = 800, height = 800,

                background_color ='black',

                min_font_size = 10).generate(punct)


#plot the WordCloud image

plt.figure(figsize = (8, 8), facecolor = None)  plt.imshow(wordcloud)
```

```
plt.axis("off")

plt.tight_layout(pad = 0)

plt.show()



## plot graphs

import matplotlib.pyplot as plt

plt.plot(stats.length, stats.unique, "bo-")

plt.loglog(stats.length, stats.unique, "ro")

stats[stats.language =="маг"] #to check information on english books

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

subset = stats[stats.language =="маг"]

plt.loglog(subset.length, subset.unique, "o", label ="маг", color
="orange")

subset = stats[stats.language =="Бакалаврь"]

plt.loglog(subset.length, subset.unique, "o", label ="Бакалаврь", color
="forestgreen")

plt.legend()

plt.xlabel("Text Length")

plt.ylabel("Number of Unique words")

plt.savefig("fig.pdf")

plt.show()
```

# СПИСОК ЛИТЕРАТУРЫ


1. https://newageinformers.blogspot.com/2016/06/natural-languageprocessing-components.html
2. https://towardsdatascience.com/a-quick-introduction-to-textsummarization-in-machine-learning-3d27ccf18a9f
3. https://en.wikipedia.org/wiki/Web_scraping
4. https://www.kaggle.com/ngyptr/python-nltk-sentiment-analysis
5. https://www.datacamp.com/community/tutorials/simplifying-sentimentanalysis-python
6. https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/com.ibm.spss.ta.help/tmfc_intro.htm
7. https://www.sas.com/en_us/software/text-miner.html
8. https://blogs.sap.com/2017/05/21/sap-hana-ta-text-analysis/
9. https://www.oracle.com/technetwork/database/options/odm/dataminerworkflow-168677.html
10. Perkins,J. Python Text Processing with NLTK 2.0 Cookbook, Packt Publishing, 2010
11. https://www.python.org/
12. https://www.spyder-ide.org/
13. https://www.anaconda.com/
14. https://www.nltk.org/
15. https://scikit-learn.org/
16. https://matplotlib.org/
17. https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/com.ibm.spss.crispdm.help/crisp_overview.htm
18. Bird,S , Klein,E , Loper,E, Natural Language Processing with Python, O'Reilly Media, Inc., 2009

19. Lutz, M, Learning Python, O'Reilly & Associates, Inc., Sebastopol, CA, 2009
20. Subramanian,G, Python Data Science Handbook: Essential Tools for Working with Data, O'Reilly Media, Inc., 2016
21. https://monkeylearn.com/text-analysis/