МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Применение алгоритмов интеллектуального анализа текстовых данных»

Студент Д.Н. Антонов
(И.О. Фамилия) (личная подпись)

Руководитель к.ф.-м.н., доцент, О.В. Лелонд
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

## Аннотация

Тема выпускной квалификационной работы – «Применение алгоритмов интеллектуального анализа текстовых данных».

Как показывает практика текстовой аналитики, наиболее эффективными средствами повышения качества анализа текстов являются методы и алгоритмы интеллектуального анализа данных.

Применение алгоритмов интеллектуального анализа текстовых данных представляет актуальность и научно-практический интерес.

Объектом исследования бакалаврской работы являются методы и алгоритмы интеллектуального анализа текстовых данных.

Предметом исследования бакалаврской работы является применение алгоритмов интеллектуального анализа текстовых данных.

Целью выпускной квалификационной работы является исследование особенностей практического применения алгоритмов интеллектуального анализа для повышения качества анализа текстовых данных.

Методы исследования – текстовая аналитика, Text Mining, методы и технологии проектирования программного обеспечения.

Практическая значимость бакалаврской работы заключается в разработке программы, реализующей эффективные алгоритмы интеллектуального анализа текстовых данных.

Результаты бакалаврской работы могут быть рекомендованы для бизнес-аналитиков и разработчиков программ, использующих для принятия управленческих решений методы и алгоритмы интеллектуального анализа текстовых данных.

Выпускная квалификационная работа состоит из 46 страниц текста, 15 рисунков, 1таблицы и 24 источников.

## Abstract

The topic of the given graduation work is “Utilizing Text mining algorithms”.

As the practice of text analytics shows, the most effective means of improving the quality of text analysis are methods and algorithms for Text mining.

Utilizing Text mining algorithms is of relevance and scientific and practical interest.

The object of study of the graduation work is Text mining methods and algorithms.

The subject of study of the graduation work is utilizing Text mining algorithms.

The aim of the graduation work is the study of the features of the practical utilizing Text mining algorithms to improve the quality of text data analysis.

Research methods: text analytics, Text Mining, software design methods and technologies.

The practical significance of the graduation work lies in the development of a program that implements effective Text mining algorithms.

The results of the graduation work are of scientific and practical interest and can be recommended for business analysts and program developers who use methods and algorithms for text data analysis to make management decisions.

The graduation work consists of an explanatory note on 46 pages including 15 figures, 1 table, the list of 24 references.

**Оглавление**

## Введение

В настоящее многие организации осознали пользу от внедрения аналитических инструментов для поддержки принятия решений.

Одним из новых направлений в этой области является текстовая аналитика.

Текстовая аналитика - это автоматизированный процесс преобразования больших объемов неструктурированного текста в количественные данные для выявления идей, тенденций и закономерностей. В сочетании с инструментами визуализации данных этот метод позволяет компаниям понять суть цифр и принимать более обоснованные решения [11].

Как показывает практика текстовой аналитики, наиболее эффективными средствами повышения качества анализа текстов являются методы и алгоритмы интеллектуального анализа данных.

Применение алгоритмов интеллектуального анализа текстовых данных представляет актуальность и научно-практический интерес.

Объектом исследования бакалаврской работы являются методы и алгоритмы интеллектуального анализа текстовых данных.

Предметом исследования бакалаврской работы является применение алгоритмов интеллектуального анализа текстовых данных.

Целью выпускной квалификационной работы является исследование особенностей практического применения алгоритмов интеллектуального анализа для повышения качества анализа текстовых данных.

Для достижения данной цели необходимо выполнить следующие задачи:

- провести анализ методов и алгоритмов интеллектуального анализа текстовых данных;
- исследовать особенности применения алгоритмов интеллектуального анализа текстовых данных в различных прикладных задачах;
- разработать и протестировать программу, реализующую алгоритмы интеллектуального анализа текстовых данных.

Методы исследования – текстовая аналитика, Text Mining, методы и

технологии проектирования программного обеспечения.

Практическая значимость бакалаврской работы заключается в разработке программы, реализующей эффективные алгоритмы интеллектуального анализа текстовых данных.

Данная работа состоит из введения, трех глав, заключения и списка используемой литературы.

Первая глава посвящена обзору и анализу методов и алгоритмов интеллектуального анализа текстовых данных.

Во второй главе проанализированы особенности применения алгоритмов интеллектуального анализа текстовых данных в различных прикладных задачах.

В третьей главе описан процесс разработки и тестирования программы, реализующей алгоритмы интеллектуального анализа текстовых данных.

В заключении описываются результаты выполнения выпускной квалификационной работы.

Бакалаврская работа состоит из 46 страниц текста, 15 рисунков, 1 таблицы и 24 источников.

# Глава 1 Обзор и анализ методов и алгоритмов интеллектуального анализа текстовых данных

Следует отметить, что методы интеллектуального анализа текстовых данных относятся к области Natural Language Processing (NLP) или обработке естественного языка.

NLP в широком смысле определяется как автоматическая обработка естественного языка, такого как речь и текст, с помощью программного обеспечения.

Обработка естественного позволяет машинам разбирать и интерпретировать человеческий язык.

NLP лежит в основе инструментов, которые мы используем каждый день - от программного обеспечения для перевода, чат-ботов, фильтров спама и поисковых систем до программного обеспечения для исправления грамматики, голосовых помощников и инструментов для мониторинга социальных сетей (рисунок 1) [17].

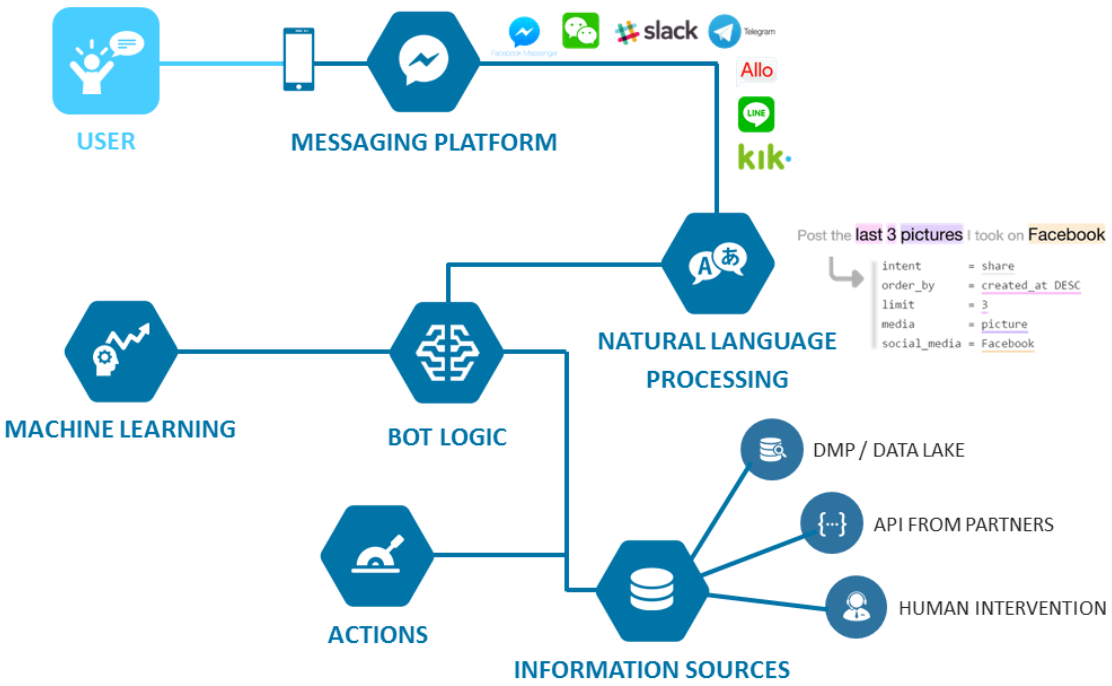


Рисунок 1 – Области применения NLP

Text mining или интеллектуальный анализ текста используется для извлечения интересной информации, знаний или закономерностей из неструктурированных документов из разных источников.

Он преобразует слова и фразы в неструктурированной информации в числовые значения, которые могут быть связаны со структурированной информацией в базе данных и проанализированы с помощью известных методов интеллектуального анализа данных. Это анализ данных, содержащихся в тексте на естественном языке.

Если методы интеллектуального анализа текста используются для решения бизнес-задач, это называется текстовой аналитикой.

Основные этапы интеллектуального анализа текста показаны на рисунке 2.

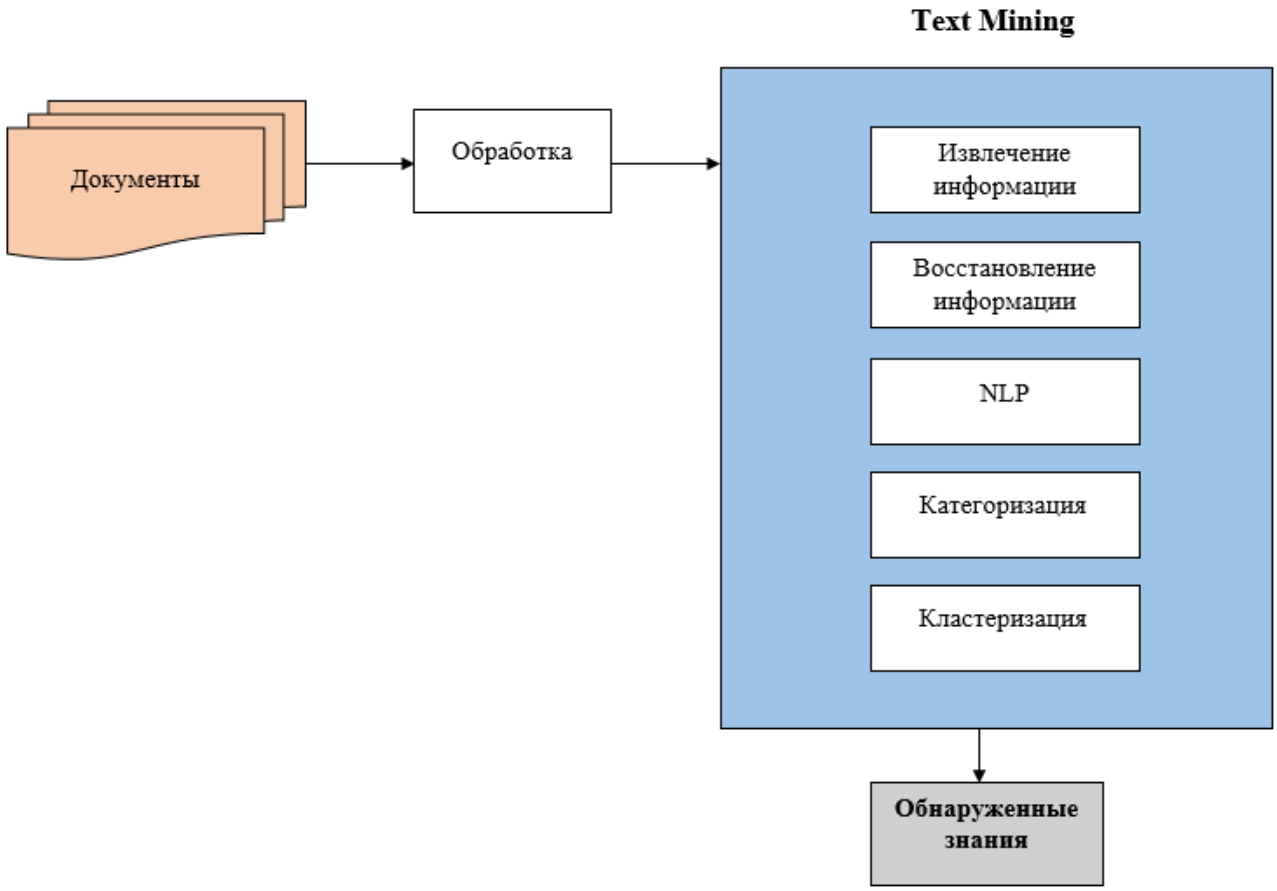


Рисунок 2 – Этапы интеллектуального анализа текста

На основании анализа литературы и источников по проблеме были выбраны для анализа следующие методы и алгоритмы интеллектуального анализа текстовых данных:

- метод токенизации;
- частота термина в документе;
- метод лемматизации;
- стоп-листинг

Проанализируем и сравним указанные методы и алгоритмов на предмет эффективного решения задач интеллектуального анализа текстовых данных.

### 1.1 Метод токенизации

Токенизация - это процесс сегментации потока текстового контента на слова, термины, символы или некоторые другие значимые элементы, называемые токенами [23].

Список токенов превращается во входные данные для дополнительной обработки, в том числе для парсинга или интеллектуального анализа текста (Text Mining).

Токенизация полезна как в лингвистике, так и в информатике, где она является частью лексического анализа. Обычно процесс токенизации происходит на уровне слов. Но иногда бывает сложно определить, что подразумевается под словом. Обычно токенизатор выполняет простые эвристические операции, например:

- знаки пунктуации и пробелы могут быть включены или не включены в итоговый список токенов;
- все непрерывные строки буквенных символов являются частью одного токена. То же справедливо и для чисел.
- токены разделяются пробелами, такими как пробел или разрыв строки, или знаками пунктуации.

Основное использование токенизации - определение значимых ключевых

слов.

Рассмотрим математическое описание токенизации на примере задачи нечёткого сопоставления пары токенизаций [1].

Пусть имеются токенизации $T_1$ и $T_2$ (набор слов $\{w_1^1, w_2^1,..., w_{|T1|}^1\}$ и $\{w_2^1, w_2^2,..., w_{|T2|}^2\}$).

Определение: выравнивание токенизаций означает построение таких отображений:

$$A_1 : \mathrm{N} \to \mathrm{N} \cup \{0\} \text{ и } A_2 : \mathrm{N} \to \mathrm{N} \cup \{0\}, \quad (1)$$

что:

$$A_1(\{1,2,...,|T_1|\}) \Delta A_2(\{1,2,...,|T_2|\}) \subset \{0\} \quad (2)$$

и выполнено следующее условие:

$$\forall i_1 < i_2 \forall j \in \{1,2\} : (A_j(i_1) \neq 0 \wedge A_j(i_2) \neq 0) \Rightarrow (A_j(i_1) \leq A_j(i_2)). \quad (3)$$

Задача нечёткого сопоставления состоит в построении такого выравнивания ($A_1$, $A_2$), которое минимизирует некоторый функционал потерь $\zeta(T_1, T_2, A_1, A_2)$.

Полный перебор имеет экспоненциальную сложность.

Функция потерь была выбрана на основании того замечания, что внешний модуль не изменяет и не удаляет символы латинских букв.

Она складывается из трёх функций:

- функция штрафа за нарушение измельчения (сопоставление одного слова $T_2$ нескольких словам $T_1$);
- функция штрафа за нарушение инъективности сопоставления латинских слов;
- функция штрафа за искажение латинских букв.

Рассмотрим следующую функцию потерь:

$$\varsigma_{\text{strict}}(T_1, T_2, A_1, A_2) = \sum_{i=1}^{3} \varepsilon_i l_i(T_1, T_2, A_1, A_2), \quad (4)$$

$$\varsigma(T_1, T_2, A_1, A_2) = \varsigma_{\text{strict}}(T_1, T_2, A_1, A_2) + \varepsilon_4 l_4(T_1, T_2, A_1, A_2), \quad (5)$$

где:

$\varepsilon_1$, $\varepsilon_2$, $\varepsilon_3$ и $\varepsilon_4$ — некоторые положительные коэффициенты.

Несложно видеть, что в связи с тем, что функции $_{li}$ принимают только целый значения, подбор коэффициентов $\varepsilon_i$ позволяет минимизировать только функционал $\zeta\,(T_1,T_2, A_1, A_2)$ для достижения того же результата.

Действительно, достаточно взять $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 1$ и $\varepsilon_4 = 1/(M+1)$, где M — максимальное допустимое возможное количество токенов в тексте.

При описанном наборе коэффициентов выполняется следующее соотношение

$$\zeta\,(T_1,T_2, A_1, A_2) < 1 \Leftrightarrow \zeta_{strict}\,(T_1,T_2, A_1, A_2) = 0 \qquad (6)$$

Для данного случая был предложен алгоритм построения выравнивания, про который доказаны следующие теоремы.

Определение: пара наборов слов $T_1$, $T_2$ является допустимой, если существует выравнивание, задаваемое $A_1$, $A_2$, такое что $\zeta_{strict}(T_1,T_2, A_1, A_2) = 0$.

Теорема: для допустимой пары наборов слов $T_1$, $T_2$ описанный выше алгоритм строит оптимальное выравнивание $A_1$, $A_2$.

Алгоритм имеет сложность O(|T1| × |T2|).

«Лексер (токенизатор, сегментатор) - это часть анализатора текста на естественном языке. В более широком смысле лексер также участвует в анализе устной речи. Задача лексера - выделить в письменной или устрой речи основные структурные единицы - лексемы и распознать их, сопоставив со словарными формами или другими морфологическими образцами.

Письменная речь может быть представлена печатным текстом (цепочки символов) или графически (отсканированный текст).

Лексер разработан для обработки реальных текстов, содержащих различные грамматические ошибки и опечатки. Поэтому кроме пассивного распознавания слов в исходной цепочке лексер может также активно модифицировать исходную цепочку, сливая и расщепляя лексемы, а также добавляя новые» [1].

В результате работы лексера получается сложная структура данных -

граф токенизации.

Граф токенизации - направленный граф, не имеющий циклических путей.

Он является исходным материалом для работы синтаксического парсера.

Пример графа токенизации изображен на рисунке 3.

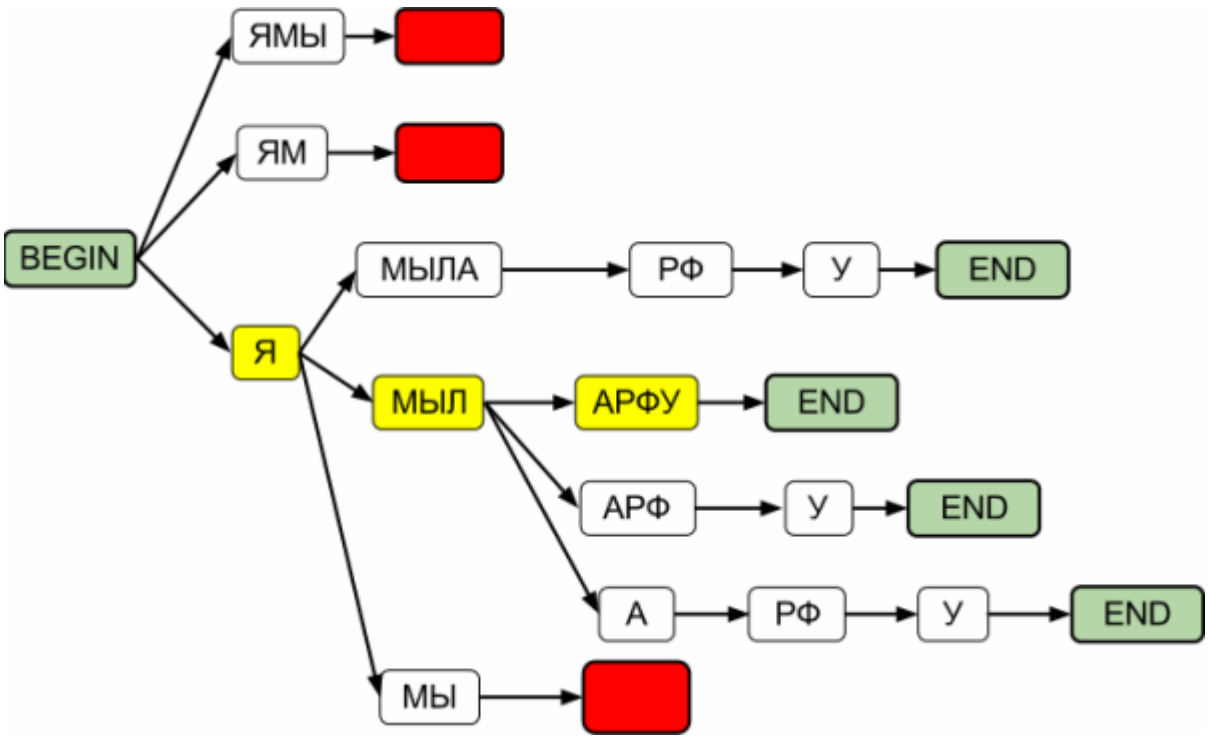


Рисунок 3 – Пример графа токенизации

Рассмотрим пример токенизации текста.

Вход: *Data Mining is the process to extract hidden predictive information from database and transform it into understandable structure for future use*.

Выход: *Data, Mining, is, the, process, to, extract, hidden, predictive, information, from, database, and, transform, it, into, understandable, structure, for, future, use*.

Преимущества применения токенизации:

– обычно текстовые данные - это только набор символов на начальном этапе. Все процессы анализа текста потребуют слов, имеющихся в

наборе данных. По этой причине требованием к синтаксическому анализатору является токенизация документов. Этот процесс может быть тривиальным, поскольку текст уже хранится в машиночитаемых форматах. Но некоторые проблемы все же остаются, например, удаление знаков препинания, удаление дефиса в конце строки. Но такие символы, как скобки, дефисы и т. д. обрабатываются достаточно хорошо;

- токенизаторы также обеспечивают надежность документов.

Недостаток метода токенизации - сложность обработки документа без пробелов, специальных символов или других знаков.

### 1.2 Частота термина в документе (TF-IDF)

«TF-IDF (TF - term frequency, IDF — inverse document frequency) — статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса.

Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех 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» [8].

Для определения величины TF используется формула:

$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k}, \tag{7}$$

где:

$n_t$ - число вхождений слова t в документ, а в знаменателе общее число слов в данном документе.

IDF **(**инверсия частоты, с которой некоторое слово встречается в документах коллекции) определяется с помощью формулы:

$$\mathrm{itf}(t, D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|}, \quad (8)$$

где:

$|D|$ - количество документов в коллекции;

$|\{d_i\} \in D \mid t \in d_i\}|$ - число документов из коллекции $D$, в которых встречается $t$ (когда $n_t \neq 0$).

«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:

$$\mathrm{tf-itf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D). \quad (9)$$

Большой вес получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах» [8].

Разновидностью формулы (8) является BM25 - поисковая функция на неупорядоченном множестве термов.

Пусть задан запрос $Q$, содержащий слова $q_1$, …, $q_2$.

Функция BM25 дает следующую оценку релевантности документа $D$ по запросу $Q$:

$$\mathrm{score}(D, Q) = \sum_{i=1}^{n} \mathrm{IDF}(q_i) \frac{f(q_i, D)(k+1)}{f(q_i, D) + k_1(1 - b + b\frac{|D|}{\mathrm{avgdl}})}, \quad (10)$$

где:

$f(q_i, D)$ - частота слова $q_i$ в документе $D$;

$|D|$ - длина документа (количество слов в нём);

avgdl - документа в коллекции;

$k_1$ и b - свободные коэффициенты, обычно их выбирают как $k_1$=0,25 и b=0,75.

Преимущества метода TF-IDF:

- простота вычисления;
- простота извлечения наиболее информативных ключевые слов из документа;
- измеряет уникальность и актуальность исследуемого контента;
- повышает ваш пользователя в Google.

К недостаткам метода TF-IDF можно отнести:

- TF-IDF основан на модели пакета слов (BoW), поэтому он не фиксирует позицию в тексте, семантику, совпадения в разных документах и т. д.;
- по этой же причине TF-IDF полезен только как функция лексического уровня;
- невозможно зафиксировать семантику (например, по сравнению с тематическими моделями, встраиваниями слов и т.п.)

Метод TF-IDF  используется в качестве индикатора важности термина для документа.

Индикатор важности увеличивается пропорционально количеству раз, когда слово появляется в документе, но компенсируется частотой слова в корпусе.

### 1.3 Методы стемминга и лемматизации

Целью как стемминга, так и лемматизации является сокращение флективных форм, а иногда и словообразовательных форм слова до общей базовой формы.

Стемминг – это частица алгоритма поиска, чтобы найти основу словоформы. Такой поиск предназначен искать слова в морфологическом изменении. Беря за основу определенное слово, поиск происходит во всей грамматике слова, без окончаний или суффиксов [20].

Это похоже на срезание ветвей дерева до стеблей. Например, основа слов

eating, eats, eaten – это eat.

Неизменяемая часть слова называется стеммой.

Принцип работы стемминга показан на рисунке 4.

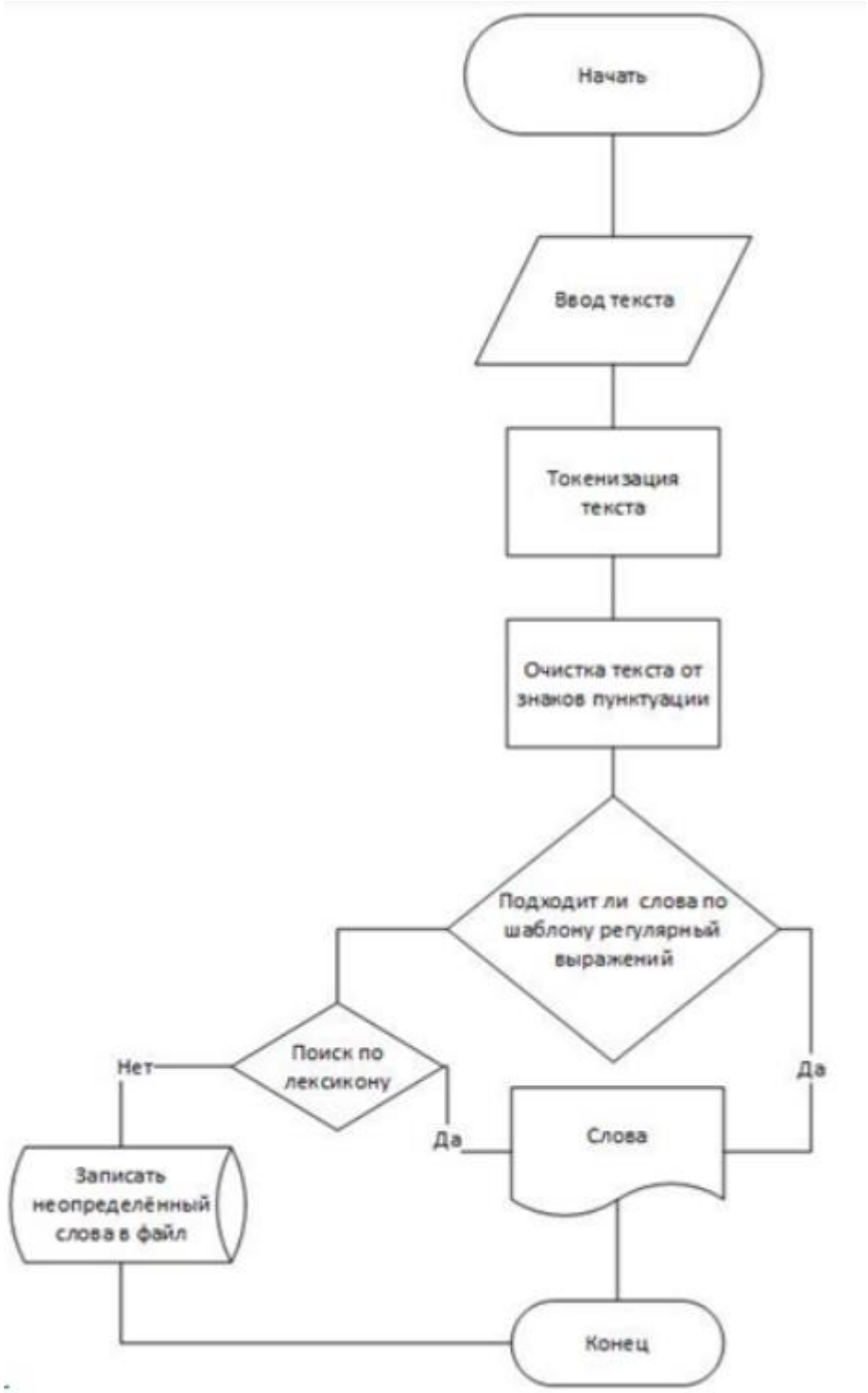


Рисунок 4 – Блок-схема общего алгоритма стемминга

Поисковые системы используют стемминг для индексации слов.

Вот почему вместо того, чтобы сохранять все формы слова, поисковая система может хранить только основы. Таким образом, выделение уменьшает размер индекса и повышает точность.

Программы, реализующие стемминг, называются стеммерами.

Преимущества стемминга, очевидно, включают сокращение словарного пространства, что значительно улучшает размер индекса (или пространства признаков). Стеммеры, основанные исключительно на словарях или правилах (например, стеммер Портера), работают очень быстро.

К недостаткам стемминга относятся:

- популярные реализации стемминга по-прежнему основаны на правилах. Не все слова можно связать с их правым корневым словом. Есть несколько недостатков, которые мы должны упустить из-за скорости процесса забоя.
- основание не может связывать слова, которые имеют разные формы на основе грамматических конструкций, например, is, am, be, все представляют один и тот же корень глагола be. Но стебель не может обрезать их до общей формы. Слово «лучше» следует преобразовать в «хорошо», но стеммеры этого не сделают.

Лемматизация – это приведение каждого слова в документе к его нормальной форме – лемме [4].

В русском языке нормальными формами считаются:

- для существительных и прилагательных – именительный падеж, единственное число, мужской род;
- для глаголов, причастий и деепричастий – глагол в неопределенный форме.

Программы, реализующие лемматизацию, называются лемматизаторами.

Для приведения слова к нормальной форме используется метод, основанный на расстоянии Левенштейна.

В компьютерной лингвистике расстояние Левенштейна - это процедура, использующая динамическое программирование для измерения наименьшего числа одного символа, который объявляется для редактирования или изменения между двумя словами (то есть вставками, удалениями или заменами).

В приложениях NLP оно широко применяется для правильного написания слов.

«В теории информации, лингвистике и информатике расстояние Левенштейна является строковой метрикой для измерения разницы между двумя последовательностями.

Иначе расстояние Левенштейна называется расстоянием редактирования и рассматривается как способ количественной оценки того, насколько разнородны две строки (например, слова), путем подсчета минимального количества операций, необходимых для преобразования одной строки в другую.

Рассмотрим определение расстояния Левенштейна.

Расстояние Левенштейна между двумя строками *a,b* (длиной |a| и |b|, соответственно) $lev_{a,b}(|a|,|b|)$ определяется из выражения:

$$\mathrm{lev}_{a,b}(i,j)=\begin{cases}\max(i,j) & \text{если } \min(i,j)=0\\ \min\begin{cases}\mathrm{lev}_{a,b}(i-1,j)+1\\ \mathrm{lev}_{a,b}(i,j-1)+1\\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{a_i\neq b_j}\end{cases} & \text{в противном случае,}\end{cases} \quad (11)$$

где:

$\mathbf{1}_{a_i \neq b_j}$ – индикаторная или характеристическая функция, равная 0, если $a_i=b_j$ и равна 1 в противном случае;

$lev_{a,b}(i,j)$ – расстояние между первыми *i* символами строки *a* и первыми *j* символами строки *b* (*i* и *j* – индексы индикаторной функции).

Следует отметить, что первый элемент минимума соответствует удалению (от *a* к *b*), второй – вставке, а третий – совпадению или несовпадению, в зависимости от того, совпадают ли соответствующие символы» [7].

Например, чтобы найти лемму, можно использовать следующее формулу [16].

$$Lemma(word) = min\ (edit\ (word, root\ words_i)$$
$$Wh\ ere\ i = one\ to\ th\ e\ number\ of\ th\ e\ root\ word$$

Рассмотрим различия между указанными методами.

Стемминг обычно относится к грубому эвристическому процессу, который обрезает концы слов в надежде на правильное достижение этой цели большую часть времени и часто включает удаление деривационных аффиксов.

Лемматизация обычно относится к правильному выполнению действий с использованием словарного запаса и морфологического анализа слов, обычно направленного на удаление только флективных окончаний и возвращение леммы.

Очевидное преимущество лемматизации состоит в том, что она более точна. Поэтому, если вы имеете дело с приложением NLP, таким как чат-бот или виртуальный помощник, где понимание смысла диалога имеет решающее значение, лемматизация будет полезна. Но за такую точность приходится платить.

Поскольку лемматизация предполагает получение значения слова из чего-то вроде словаря, на это уходит очень много времени. Таким образом, большинство алгоритмов лемматизации медленнее по сравнению с их аналогами с выделением корней. Также существуют накладные расходы на вычисления для лемматизации, однако в проблеме машинного обучения вычислительные ресурсы редко вызывают беспокойство.

Лингвистическая обработка для стемминга или лемматизации часто выполняется с помощью дополнительного подключаемого компонента к процессу индексации, и существует ряд таких компонентов, как коммерческих, так и с открытым исходным кодом.

### 1.4 Стоп-листинг

Важным аспектом всех задач машинного обучения, связанных с обработкой документов, является список «стоп-слов», также известный как «стоп-лист». Текстовые документы содержат огромную долю таких данных, которые не являются полезными для обработки. Поэтому желательно разработать какой-то автоматический метод идентификации таких данных и удаления их из набора данных до его обработки.

«Стоп-листинг применяется для удаления стоп-слов и отдельных символов, которые появляются в текстовом документе, и, вероятно, частота стоп-слов может быть больше, чем другие ключевые слова в этом тексте. Вот почему этот шаг необходимо использовать для получения более четкого результата в машинном обучении.

Основные принципы метода:

- стоп-слова - это слова с низкой дискриминационной ценностью;
- конкретные существительные, глаголы или другие грамматические типы могут не относится к стоп-словам, а такие элементы, как артикли, предлоги и союзы, обычно присутствуют в списке стоп-слов;
- стоп-слова служат только синтаксической функции и никогда не имеют какой-либо предсказательной способности. Они не указывают на предмет;
- стоп-слова имеют очень высокую частоту, поэтому они могут повлиять на эффективность процесса обработки.

Рассмотрим метод «Термин на основе случайной выборки (TBRS)».

Это метод, в котором стоп-слова обнаруживаются вручную из документов.

Этот метод используется путем перебора случайно выбранных отдельных фрагментов данных и ранжирования объектов в каждом фрагменте на основе их значений в формате с использованием меры дивергенции Кульбака-Лейблера, как показано в следующем уравнении» [11]:

$$d\_x(t) = Px(t).\log\_2\left[\frac{Px(t)}{P(t)}\right], \quad (12)$$

где *Px(t)* - нормализованная частота термина *t* в пределах веса *x*;

*P(t)* - нормализованная частота термина *t* во всей коллекции.

Окончательный стоп-лист строится, принимая наименее информативные термины во всех документах, удаляя все возможные дубликаты.

## Выводы к главе 1

Первая глава ВКР посвящена постановке обзору и анализу методов интеллектуального анализа текстовых данных.

Результаты проделанной работы позволили сделать следующие выводы:

- методы интеллектуального анализа текстовых данных относятся к области NLP или обработке естественного языка;
- основное использование токенизации - определение значимых ключевых слов. Главным преимуществом токенизации заключается в том, что помогает интерпретировать значение текста, анализируя последовательность слов. Недостаток метода токенизации - сложность обработки документа без пробелов, специальных символов или других знаков;
- метод TF-IDF часто используется в задачах анализа текстов и информационного поиска. Преимущество метода заключается в простоте вычислений. Однако он не фиксирует позицию в тексте, семантик и совпадения в разных документах;
- стемминг обычно относится к грубому эвристическому процессу, который обрезает концы слов в надежде на правильное достижение этой цели большую часть времени и часто включает удаление деривационных аффиксов;
- лемматизация обычно относится к правильному выполнению действий

с использованием словарного запаса и морфологического анализа слов, обычно направленного на удаление только флективных окончаний и возвращение леммы;

- стоп-слова имеют очень высокую частоту, поэтому они могут повлиять на эффективность процесса интеллектуального анализа текста.

# Глава 2 Обзор и анализ алгоритмов интеллектуального анализа текстовых данных

Рассмотрим и проанализируем алгоритмы интеллектуального анализа текстовых данных, построенных на основе методов, описанных в главе 1.

## 2.1 Алгоритмы токенизации

Рассмотрим токенизацию на примере алгоритмов обработки подслов.

«Подслово - это некоторая строгая последовательность символов слова, начинающаяся в определенной позиции. Для ее получения берется часть слова, представляющая собой последовательность символов определенной длины. Если представить слово набором букв, то подслово представляет собой содержимое некоторого окна заданной длины, наложенного на это слово.

Корпусом называют собрание текстов или фрагментов текстов в электронной форме, отобранных в соответствии с внешними критериями, чтобы наиболее полно представлять язык или вариацию языка. Кроме текстовых данных корпус составляет программное обеспечение системы управления и анализа текстов» [22].

Рассмотрим алгоритм кодирования пар байтов (Byte Pair Encoding, BPE).

Пошаговое представление алгоритма BPE имеет вид:

Шаг 1. Подготовьте достаточно большие обучающие данные (например,

корпус);

Шаг 2. Определите желаемый размер словарного запаса подслов

Шаг 3. Разделите слово на последовательность символов и добавить суффикс «</w>» к концу слова с частотой слов. Итак, на этом этапе основной единицей является символ. Например, частота «low» равна 5, затем мы перефразируем его на «l o w </w>»: 5.

Шаг 4. Генерация нового подслова в соответствии с высокой частотой появления.

Шаг 5. Повторяем шага 4 до тех пор, пока не будет достигнут размер словарного запаса подслов, который определен на шаге 2, или следующая самая высокая пара частот не будет равна 1.

Псевдокод алгоритма кодирования пар байтов представлен на рисунке 5.

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

Рисунок 5 - Псевдокод алгоритма кодирования пар байтов

Достоинством BPE является возможность эффективного сбалансирования размера словарного списка и количества шагов (количество токенов, необходимых для кодирования предложения).

К недостаткам относится то, BPE является жадным алгоритмом и основан на детерминированной замене символов. Кроме того, результаты множественной фрагментации с заданной вероятностью не могут быть предоставлены.

Алгоритм BPE может использоваться как простая форма сжатия данных, в которой наиболее распространенная пара последовательных байтов данных заменяется байтом, которого нет в этих данных.

«Недавно его начали использовать для токенизации текстовых данных. В таких моделях, как BERT и GPT-2, тексты токенизировались с помощью BPE.

Оптимизированный BPE работает как минимум в два раза быстрее существующих альтернатив.

Две основные характеристики оптимизированного алгоритма:

- линейное время работы, которое зависит от размера обучающего корпуса текстов;
- использование нескольких потоков для обучения и токенизации. Это сокращает время работы алгоритма в несколько раз.

Как показал анализ, скорость работы для некоторых текстов выросла в 10 раз» [3].

WordPiece - это еще один алгоритм сегментации слов, похожий на BPE.

Его разработчики - Шустер и Накадзима представили WordPiece, решив проблему голоса для японцев и Кореи в 2012 году.

По сути, WordPiece схож с BPE, а разница в части вероятности формирует новое подслово, но не следующую по частоте пара.

Пошаговое представление алгоритма WordPiece имеет вид:

Шаг 1. Подготовьте достаточно большие обучающие данные (например, корпус).

Шаг 2. Определите желаемый размер словарного запаса подслов.

Шаг 3. Разделите слово на последовательность символов.

Шаг 4. Создайте языковую модель на основе данных шага 3

Шаг 5. Выберите новую единицу слова из всех возможных, которая больше всего увеличивает вероятность обучающих данных при добавлении в модель.

Шаг 6. Повторяйте шаг 5 до достижения размера словаря подслов, который определен на шаге 2, или увеличение вероятности упадет ниже определенного порогового значения.

По сравнению с BPE WordPie не выбирает наиболее часто встречающуюся пару символов, а выбирает пару символов, которая максимизирует возможность добавления обучающих данных в словарь.

Возможность максимизации обучающих данных эквивалентна нахождению пары символов во всех символах, первый символ - это наибольшая вероятность того, что второй символ будет максимальным.

Интуитивно понятный WordPiece немного отличается от BPE, который оценивает потерянный контент путем объединения двух символов.

Модель языка Unigram (Unigram Language Model, ULM) - это алгоритм сегментации подслов, разработанный Т. Кудо [21]. Он основан на предположении, что все вхождения подслов происходят независимо, и последовательность подслов создается как произведение вероятностей появления подслов.

И WordPiece, и Unigram Language Model используют языковую модель для создания словарного запаса подслов.

ULM можно рассматривать как комбинацию нескольких конечных автоматов с одним состоянием. Он разделяет вероятности различных терминов в контексте, например, из выражения:

$$P(t_1t_2t_3) = P(t_1)P(t_2|t_1)P(t_3|t_1t_2) \quad (13)$$

получим:

$$P_{\text{ulm}}(t_1 t_2 t_3) = P(t_1)P(t_2)P(t_3). \quad (14)$$

В этой модели вероятность каждого слова зависит только от собственной вероятности этого слова в документе, поэтому в качестве единиц у нас есть только конечные автоматы с одним состоянием. Сам автомат имеет распределение вероятностей по всему словарю модели, в сумме равное 1.

Пошаговое представление алгоритма ULM имеет вид:

Шаг 1. Подготовьте достаточно большие обучающие данные (например, корпус).

Шаг 2. Определите желаемый размер словарного запаса подслов.

Шаг 3. Оптимизируйте вероятность появления слова, задав последовательность слов.

Шаг 4. Вычислите потерю каждого подслова.

Шаг 5. Отсортируйте символы по убыванию и оставьте верхние X% слова (например, X может быть 80). Чтобы избежать исчерпания словарного запаса, рекомендуется включать уровень символа как подмножество подслова.

Шаг 6. Повторяйте шаги 3–5 до достижения размера словарного запаса подслов, определенного на шаге 2, или без изменений на шаге 5.

Следует отметить, что представленные не решают многие проблемы токенизации, что особенно заметно в машинном переводе, поскольку многие языки являются синтетическими и словообразование в них более сложное по сравнению с аналитическим английским языком.

### 2.2 Алгоритм TF-IDF

TF-IDF статистика – это общий инструмент для извлечения не только векторов слов, но и извлечения ключевых слов. В данном подходе ключевые слова извлекаются не из конкретного документа, а из набора.

Также, если произвести предварительную кластеризацию, то имеется возможность извлекать документы из кластера, которые должны являться

однородными по смыслу (рисунок 6).


Рисунок 6 - Извлечение ключевых слов кластера с помощью TF-IDF

TF-IDF извлекает ключевые слова из документа методом выбора N слов с наибольшим значением TF-IDF документа. Формула TF – отношение числа вхождений некоторого слова к общему числу слов документа. Таким образом, оценивается важность слова в пределах отдельного документа.

«IDF – это инверсия частоты, с которой некоторое слово встречается в коллекции документов. Учёт IDF позволяет уменьшить вес широкоупотребительных слов. Для каждого отдельного уникального слова в пределах конкретной коллекции документов существует только одно значение IDF. Таким образом получается, что наибольший вес будут иметь слова, часто употребляемые в конкретном документе и реже встречаемые в других, что позволяет в рамках алгоритма отфильтровать стоп-слова.

Для реализации метода TF-IDF используются алгоритмы кластеризации, основанные на классическом EM-алгоритме» [14].

EM (Expectation-maximization) – популярный алгоритм кластеризации, позволяющий эффективно работать с большими объемами данных.

Блок-схема EM-алгоритма представлена на рисунке 7.

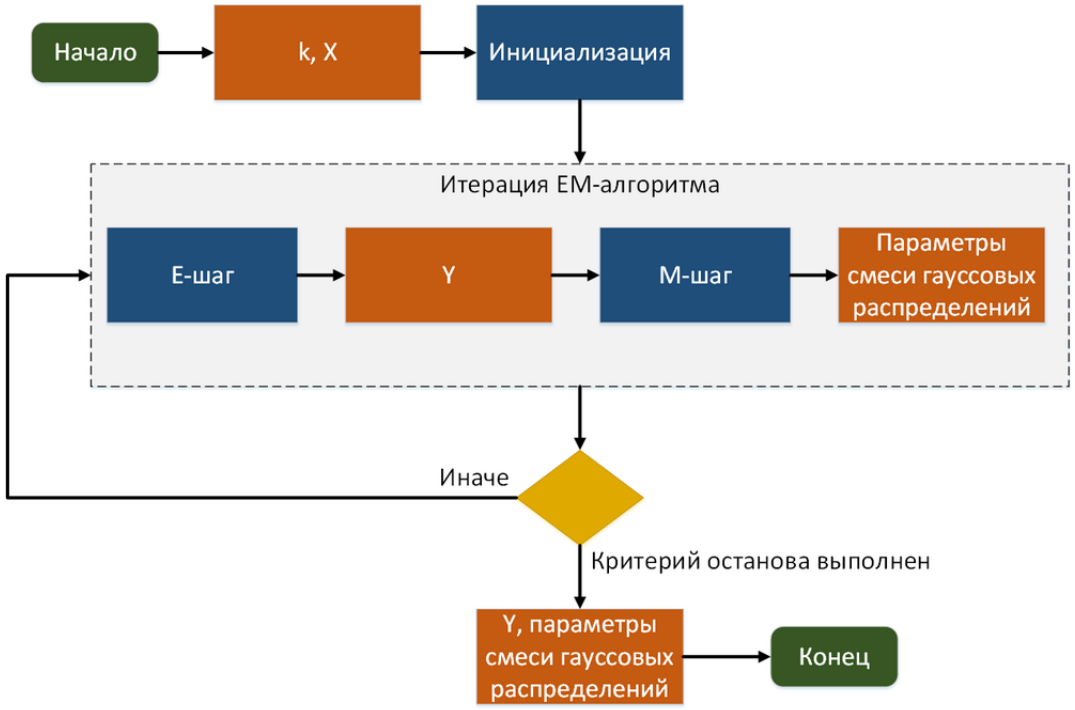


Рисунок 7 – Блок-схема EM-алгоритма

В основе идеи EM-алгоритма лежит предположение, что любое наблюдение принадлежит ко всем кластерам, но с разной вероятностью.

Поэтому на выходе формируются два дополнительных столбца: номер кластера и вероятность принадлежности.

Объект должен быть отнесен к тому кластеру, для которого данная вероятность выше.

Примером алгоритма данного типа является алгоритм k-means.

Преимущества алгоритма TF-IDF - простота и скорость, результат больше соответствует реальной ситуации.

Вместе с тем недостаточно просто использовать «частоту слов» в качестве стандарта измерения. Иногда важные слова могут не появляться много раз.

Следует отметить, что алгоритм TF-IDF можно эффективно использовать совместно с классическими алгоритмами кластеризации, например, k-means, для оценки влиянии частотности терминов при кластеризации текстовых

документов [9].

### 2.3 Алгоритм лемматизации WordNet

WordNet - это лексическая база данных семантических отношений между словами более чем 200 языков.

WordNet связывает слова в семантические отношения, включая синонимы, гипонимы и меронимы. Синонимы сгруппированы в синсеты с краткими определениями и примерами использования. Таким образом, WordNet можно рассматривать как комбинацию и расширение словаря и тезауруса. Хотя он доступен для пользователей через веб-браузер, его основное применение - автоматический анализ текста и приложения искусственного интеллекта

Основа для группировки слов в WordNet - их значения.

WordNet использует алгоритм внутренней лемматизации для нормализации слов.

Алгоритм лемматизации WordNet использует два ресурса, набор правил, которые определяют флективные окончания, которые могут быть отделены от отдельных слов, и список исключений для неправильных форм слов. Сначала алгоритм проверяет наличие исключений, а затем применяет правила отсоединения.

После каждого преобразования по правилам в базе данных WordNet выполняется поиск существования выходной формы. Этот инструмент используется для реализации алгоритма машинного обучение.

Псевдокод алгоритма лемматизации WordNet показан на рисунке 8 [12].

```
G(V, E)
V := vertices arranged in breadth first order
E := set of edges
|V| := m |E| := n
for v_i ∈ V do
   create node v'_i for each sense of v_i
   distribute edges across senses
end for
v' := new sense vertex set; k := |v'|
for i := 0 → k do
   if Edge set v'_i == 0 then
      delete v'_i
   end if
   for j := 0 → k do
      if Edge set v'_i == Edge set v'_j then
         merge v'_i and v'_j
      end if
   end for
end for
```

Рисунок 8- Псевдокод алгоритма лемматизации WordNet

Пример результирующего графа лемматизации WordNet показан на рисунке 9.

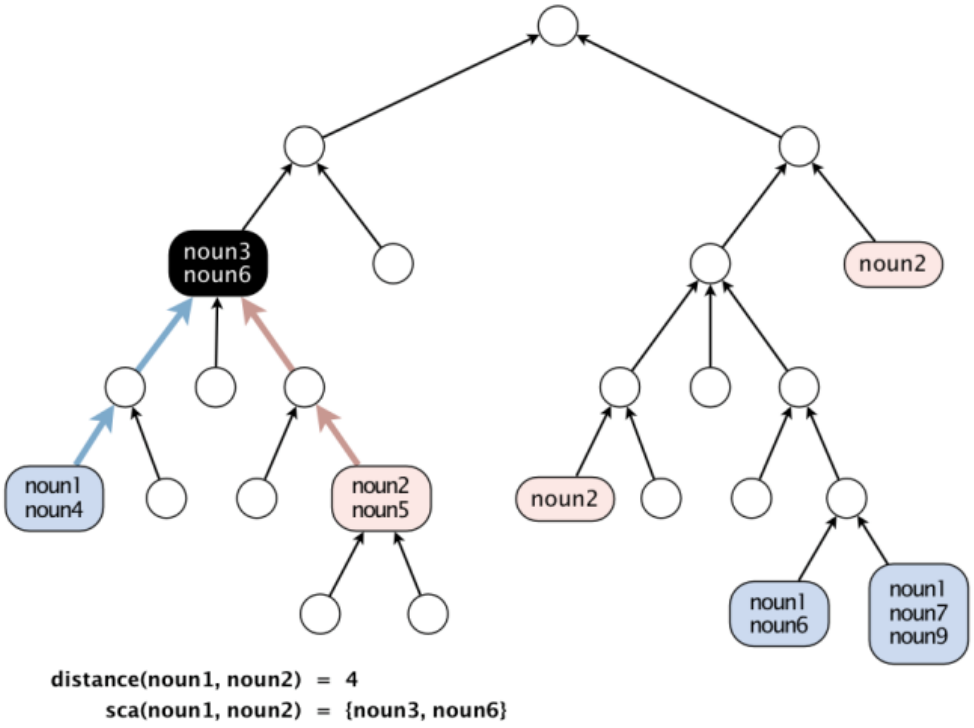


Рисунок 9 - Пример результирующего графа лемматизации WordNet

«WordNet можно свободно использовать в коммерческих и научных целях.

Для работы с ним существует несколько программ, множество интерфейсов и API, реализуемых на большинстве возможных языков, так и с помощью протокола DICT, программы GoldenDict и других. Также, пакеты WordNet присутствуют в некоторых репозиториях ПО для GNU и Linux и их дистрибутивов» [24].

### 2.4 Алгоритмы стоп-листинга

Для реализации стоп-листинга используются алгоритмы классификации Data mining, например, K-ближайших соседей (KNN) [15].

Формальное описание задачи классификации по данному методу имеет вид:

$$\hat{y} = \arg\max_{y} \sum_{k=1}^{K} \mathrm{I}(y_k == y) \qquad (15)$$

Алгоритм может быть применим к многомерным выборкам. Для этого перед применением нужно определить метрику расстояний.

При реализации алгоритма k ближайших соседей первым шагом является преобразование точек данных в векторы признаков или их математические значения.

Затем алгоритм работает, находя расстояние между математическими значениями этих точек.

Параметрами алгоритма являются значения *k* и метрика расстояний.

Самый распространенный способ найти это расстояние - это евклидово расстояние, вычисляемое по следующей формуле:

$$d_E(x, y) = \sqrt{\sum_i (x_i - y_i)^2}\,, \qquad (16)$$

где:

$x = (x_1, x_2, \ldots, x_m)$, $y = (y_1, y_2, \ldots, y_m)$ – векторы значений двух записей.

«KNN широко используется для классификации текстов. Для начала считаем координаты текстов в пространстве. Размер пространства есть количество терминов в корпусе (объем словаря).

Считая TF-IDF для всех текстов в корпусе, получаем представления текста в виде векторов, каждый компонент вектора – «важность» соответствующего слова для данного текста. Координаты текстов используются для решения различных задач, в том числе классификация.

Один из недостатков KNN – медленная скорость, для классификации нового текста нужно вычислять расстояния между этим текстом со всеми текстами в корпусе, а их количество может быть миллионы.

Для оптимизации алгоритма KNN используются различные подходы, одним из которых является уменьшение количества потенциальных ближайших соседей» [6].

Для каждого термина храним $id$ текстов, содержащих этот термин. Для

быстрого поиска $id$ текста сортируем координаты этих текстов по этому термину по возрастанию.

Для каждого термина $t$ есть список $(id, x)$, где $x$ – значение координаты по данному термину:

$$X(t) = [(id1, x1),(id2, x2)...(idm, xm)], \text{ где } x1 <= x2... <= xm. \quad (17)$$

«Оптимизированный алгоритм KNN для классификации нового текста состоит из следующих шагов:

Шаг 1. Вычисляем координаты (TF-IDF) этого текста.

Шаг 2. Для каждого термина $(t, x)$ ищем ближайшее к $x$ значение на $X(t)$ = = $[(id1, x1),(id2, x2)...(idm, xm)]$, используя метод двоичного поиска.

Шаг 2-1. $i = 1$; $j = m$. Шаг 2-2. Пока $i < j$.

Шаг 2-2-1. $k = (i + j)$div2.

Шаг 2-2-2. Если $xk < x$, то $i = k + 1$. Шаг 2-2-2. Если $xk >= x$, то $j = k$. После шага 2-2 $xk$ есть ближайшее значение к $x$ на $X(t)$.

Шаг 3. Рассмотрим 2*К текстов вокруг $(idk, xk)$ на $X(t)$, считая расстояние между текстами, проведем сравнение и обновление К ближайших соседей данного текста.

Сложность оптимизированного алгоритма KNN:

$$O(K*\log(N)*|L|), \quad (18)$$

где:

N – количество текстов в корпусе;

|L| – средняя длина текста в корпусе.

Очевидно, что оптимизированный алгоритм KNN имеет преимущество по скорости работы при больших корпусах текстов» [6].

## Выводы к главе 2

Вторая глава ВКР посвящена обзору и анализу алгоритмов интеллектуального анализа текстовых данных.

Результаты проделанной работы позволили сделать следующие выводы:

Достоинством алгоритма BPE является возможность эффективного

сбалансирования размера словарного списка и количества шагов (количество токенов, необходимых для кодирования предложения). К недостаткам относится то, BPE является жадным алгоритмом и основан на детерминированной замене символов. Кроме того, результаты множественной фрагментации с заданной вероятностью не могут быть предоставлены.

Преимущества алгоритма TF-IDF - простота и скорость, результат больше соответствует реальной ситуации. Вместе с тем недостаточно просто использовать «частоту слов» в качестве стандарта измерения. Иногда важные слова могут не появляться много раз. Следует отметить, что алгоритм TF-IDF можно эффективно использовать совместно с классическими алгоритмами кластеризации, например, k-means, для оценки влиянии частотности терминов при кластеризации текстовых документов.

Лексическая база данных WordNet использует алгоритм внутренней лемматизации для нормализации слов. Алгоритм лемматизации WordNet использует два ресурса, набор правил, которые определяют флективные окончания, которые могут быть отделены от отдельных слов, и список исключений для неправильных форм слов. Сначала алгоритм проверяет наличие исключений, а затем применяет правила отсоединения.

Одним из недостатков алгоритма KNN является медленная скорость, что снижает его эффективность при решении задач классификации больших текстов.

Для оптимизации алгоритма KNN используются различные подходы, одним из которых является уменьшение количества потенциальных ближайших соседей

# Глава 3 Разработка программы интеллектуального анализа текстовых данных

Для разработки программы интеллектуального анализа текстовых данных используем язык программирования Python и технологию Integrated Development Environment (IDE).

## 3.1 Выбор среды разработки программы

IDE – интегрированная среда разработки представляет собой многофункциональную программу, которую можно использовать для различных аспектов разработки программного обеспечения.

Интегрированная среда разработки позволяет программистам объединять различные задачи написания компьютерной программы.

Интегрированные среды разработки повышают продуктивность программиста за счет объединения общих действий по написанию программного обеспечения в одном приложении: редактирование исходного кода, создание исполняемых файлов и отладка.

В состав типовой интегрированной среды разработки входят:

- текстовый редактор;
- транслятор – компилятор или интерпретатор;
- средства автоматизации сборки;
- отладчик.

Рассмотрим функциональные и архитектурные особенности популярных интегрированных сред разработки.

### 3.1.1 Интегрированная среда разработки Visual Studio + Python Tools for Visual Studio

Интегрированная среда разработки или IDE Visual Studio– это стартовая площадка для написания, отладки и сборки кода, а также последующей

публикации приложений.

«Помимо стандартного редактора и отладчика, которые существуют в большинстве сред IDE, Visual Studio включает компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для упрощения процесса разработки.

Python Tools for Visual Studio (PTVS) позволяет писать на Python в Visual Studio и включает в себя Intellisense для Python, отладку и другие инструменты (рисунок 10).

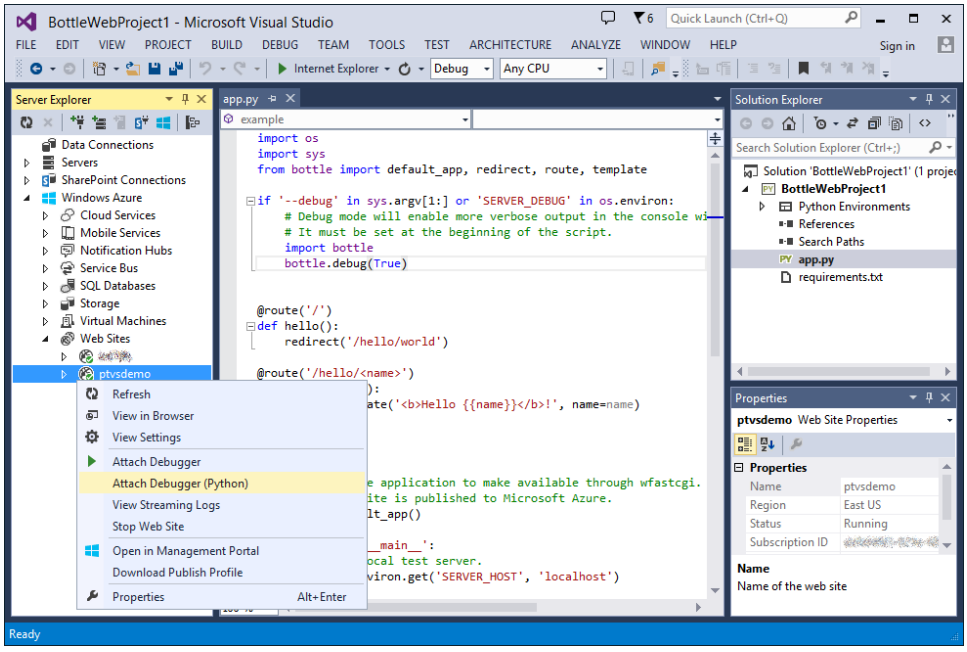


Рисунок 10 – Экран редактора  Python Tools for Visual Studio

Основными инструментами разработчика является следующие:

- обозреватель решений, который позволяет просматривать файлы кода, перемещаться по ним и управлять ими. Обозреватель решений позволяет упорядочить код путем объединения файлов в решения и проекты;
- редактор, отображающий содержимое файла. Здесь можно редактировать код или разрабатывать пользовательский интерфейс, например, окно с кнопками или текстовые поля;» [5]

– командный обозревать, который позволяет отслеживать рабочие элементы и использовать код совместно с другими пользователями с помощью технологий управления версиями, таких как Git и система управления версиями Team Foundation.

Доступная на Windows и Mac OS, Visual Studio представлена как в бесплатном (Community), так и в платном (Professional и Enterprise) вариантах. Visual Studio позволяет разрабатывать приложения для разных платформ и предоставляет свой собственный набор расширений.

Возможности Visual Studio оптимизированы для разработки кроссплатформенных и мобильных приложений.

### 3.1.2 Интегрированная среда разработки PyCharm

Одной из лучших полнофункциональных IDE, предназначенных именно для Python, является PyCharm. Существует как бесплатный open-source (Community), так и платный (Professional) варианты IDE.

PyCharm доступен на Windows, Mac OS X и Linux (рисунок 11).

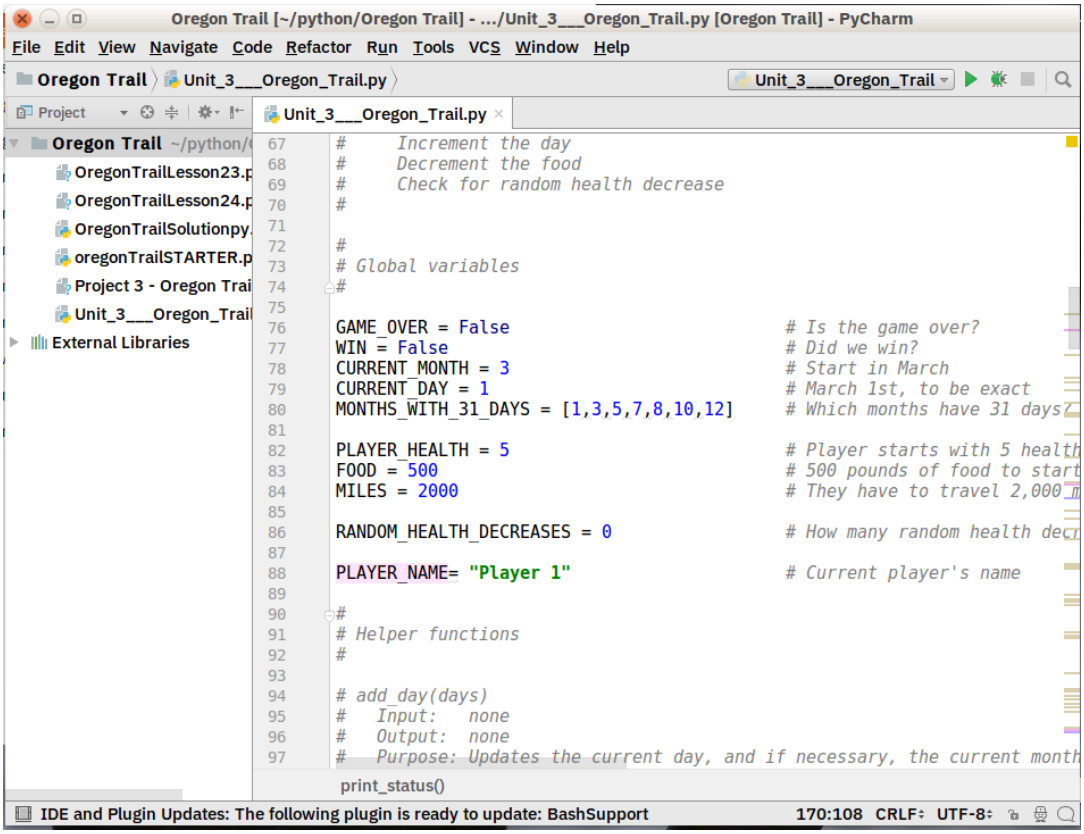


Рисунок 11 – Среда IDE PyCharm

Коробочный вариант PyCharm поддерживает разработку на Python напрямую — достаточно открыть новый файл и начать писать код.

«Разработчик  может запускать и отлаживать код прямо из PyCharm. Кроме того, в IDE есть поддержка проектов и системы управления версиями.

Преимущества: это среда разработки для Python с поддержкой всего и вся и хорошим коммьюнити. В ней «из коробки» можно редактировать, запускать и отлаживать Python-код.

К недостаткам PyCharm можно отнести медленную загрузку. Кроме того, настройки по умолчанию, возможно, придётся подкорректировать для существующих проектов» [19].

### 3.1.3 Интегрированная среда разработки Eclipse + PyDEv

Интегрированная среда разработки Eclipse является бесплатной программной платформой с открытым исходным кодом, контролируется организацией Eclipse Foundation [13].

Среда написана на языке программирования Java.

Основной целью её создания является повышение продуктивности процесса разработки программного обеспечения.

Фрагмент среды представлен на рисунке 12.

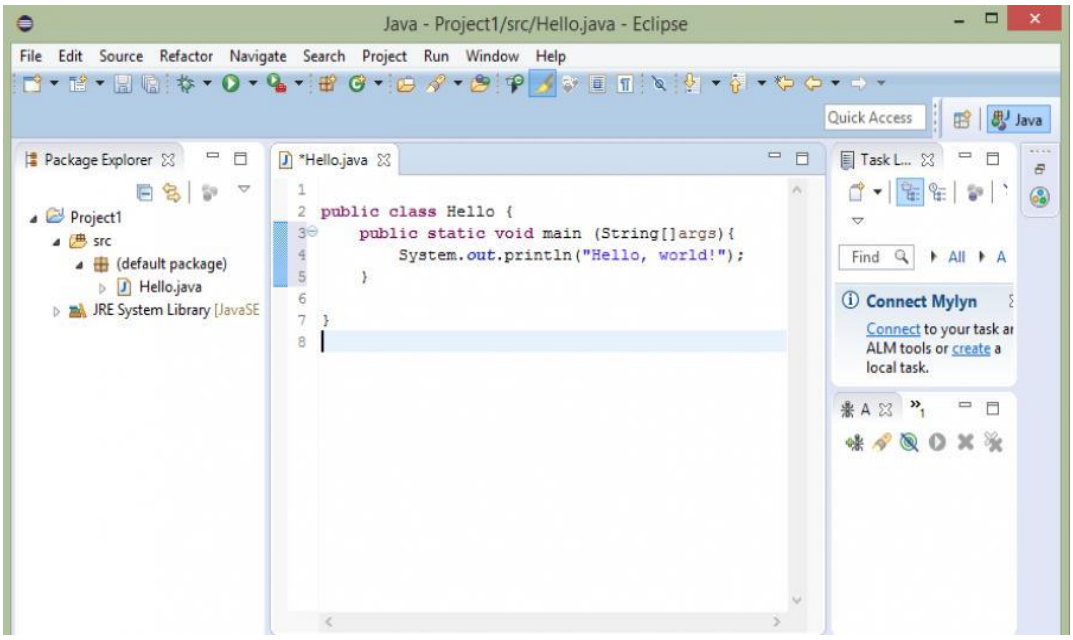

Рисунок 12 – Экран IDE Eclipse

«Основным компонентом является исполняемая среда – Eclipse Runtime, в которой выполняются коды расширений и модулей. Она обеспечивает всю базовую функциональность – управление расширениями и обновлениями, взаимодействие с операционной системой, обеспечение работы системы помощи.

Следующим ключевым компонентом является собственно интегрированная среда разработки – она отвечает за управление элементами программы, управление проектами, отладку и сборку проектов, поиск по файлам и командную программу.

Eclipse претендует на статус наиболее популярной Java IDE и является единственным конкурентом такой мощной платформы, как NetBeans.

Но в отличие от последней, использующей для создания элементов пользовательского интерфейса платформонезависимую библиотеку Swing, Eclipse использует платформозависимую библиотеку SWT.

Интегрированные среды разработки, созданные на платформе Eclipse, широко применяются для создания программного обеспечения на различных языках программирования» [13].

Это обусловлено универсальностью Eclipse, работающей по принципу «Плагины для Eclipse разрабатываются в самой Eclipse».

Следует учесть, что в рамках проекта Eclipse Foundation предлагается несколько платформ для создания подключаемых модулей для настольных инструментов, распределенных сервисов и интерфейсов браузера.

Одним из таких расширений является PyDev, предоставляющий интерактивную консоль Python и возможности для отладки и автодополнения кода.

Установить его просто: запустите Eclipse, выберите Help → Eclipse Marketplace, затем найдите PyDev. Нажмите «Install» и при необходимости перезапустите Eclipse (рисунок 13).

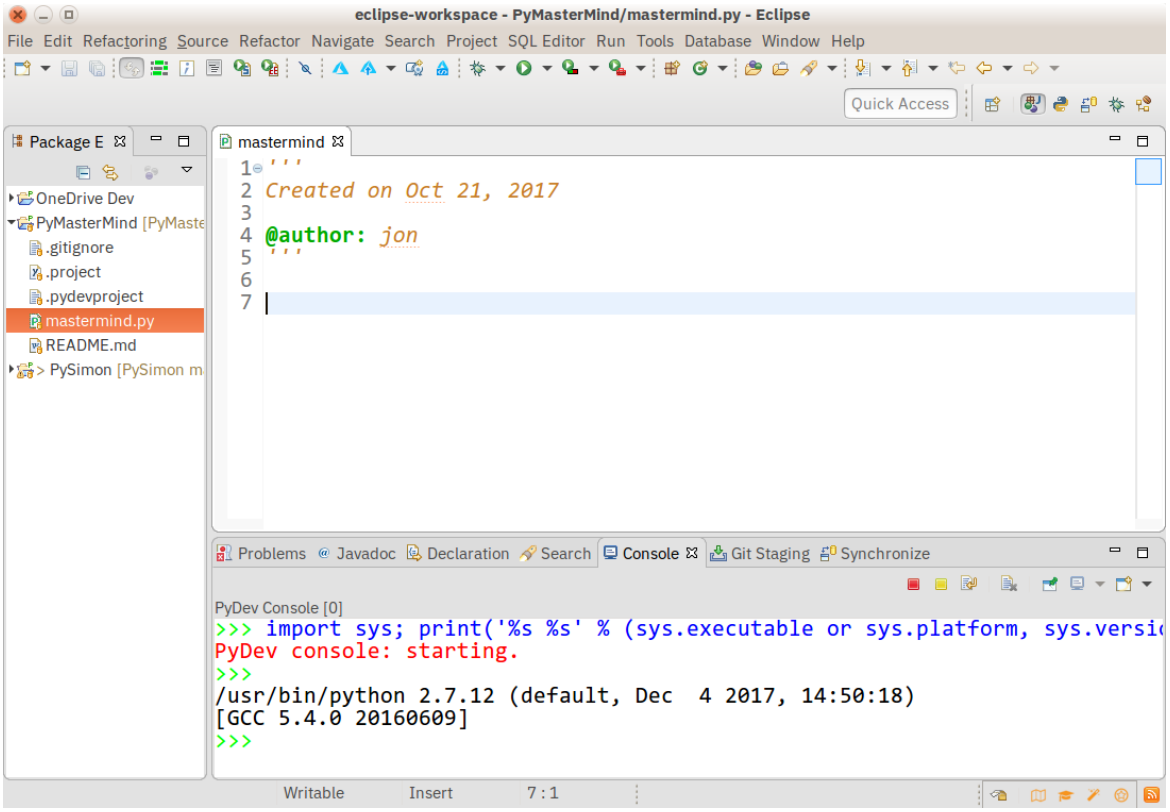


Рисунок 13 – Экран редактора языка Python

Установка PyDev пройдёт быстро и гладко. У опытного пользователя Eclipse не возникнет проблем с изучением этого расширения.

Для выбора интегрированной среды разработки используем таблицу 1, составленную на основе анализа блогов по данной тематике.

Критерии оценивания:

0 – полное несоответствие требованиям;

1 – значительное несоответствие требованиям;

2 – незначительное несоответствие требованиям;

3 – полное соответствие требованиям.

Таблица 1 – Сравнительный анализ интегрированных сред разработки для Python

| Характеристика/балл | Visual Studio | PyCharm | Eclipse+PyDEv |
|---|---|---|---|

| | | | |
|---|---|---|---|
| поддержка языка Python | 2 | 3 | 2 |

Продолжение таблицы 1

| | | | |
|---|---|---|---|
| юзабилити | 3 | 2 | 3 |
| предпочтение разработчика | 1 | 2 | 3 |
| Итого | 6 | 7 | 8 |

Таким образом, наилучшими характеристиками обладает IDE Eclipse+PyDEv.

Выбираем IDE Eclipse+PyDEv в качестве среды для разработки программы.

### 3.2 Реализация и тестирование программы

Для представления программной архитектуры разработана диаграмма компонентов программы, показанная на рисунке 14.

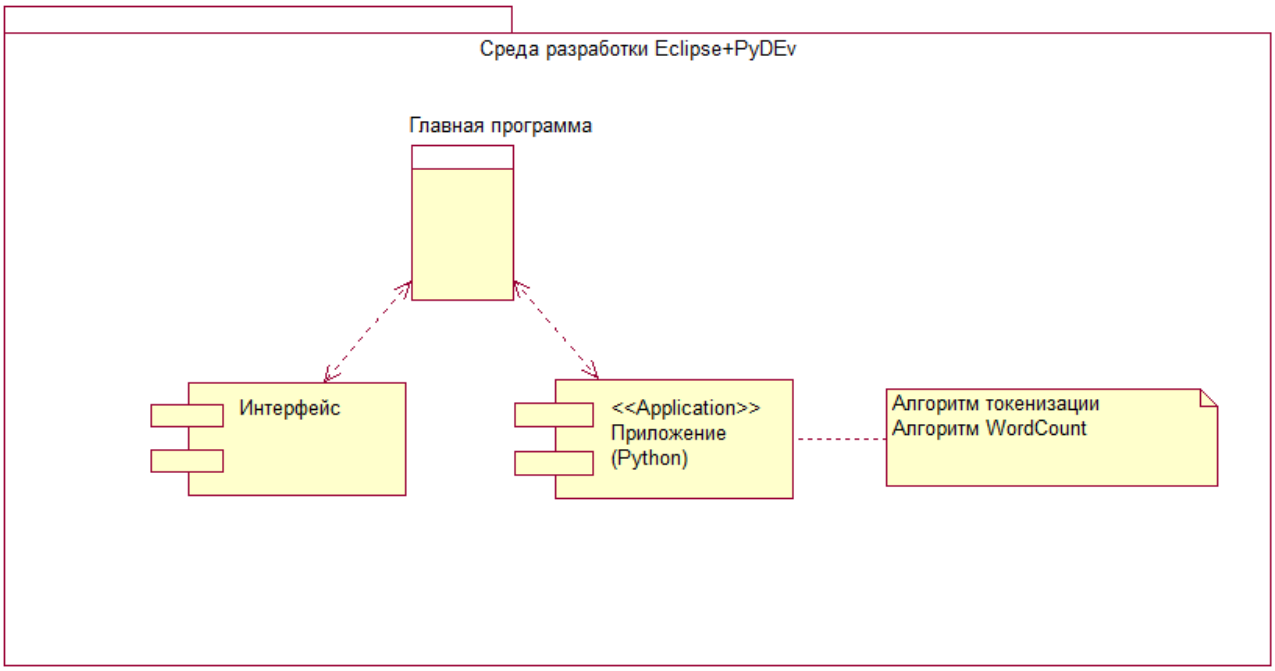


Рисунок 14 - диаграмма компонентов программы

На языке Python реализованы алгоритмы интеллектуального анализа текста [18].

При разработке программного кода использована библиотека NLTK [2].

Листинг программного кода токенизации и результаты его выполнения имеют вид:

```
import redef tokenize(text):
text = text.lower()
text = re.sub(r'[^a-zA-Z ^0-9]', '', str(text))
return text.split()# We are using the text from above as our sample_text
tokens = tokenize(sample_text) tokens[:10]
```

---

```
['a', 'token', 'is', 'a', 'sequence', 'of', 'characters', 'in', 'a', 'document']
```

Листинг алгоритма лемматизации и результаты его выполнения имеют вид:

```
def spacy_lemmatize(text):
doc = nlp.tokenizer(text)
return [token.lemma_ for token in doc]
spacy_lemmas = spacy_lemmatize(sample_text)
spacy_lemmas[:10]
```

---

```
['A', 'token', 'be', 'a', 'sequence', 'of', 'character', 'in', 'a', 'document']
```

«Листинг программного кода подсчета слов имеет вид:

```
from collections import Counter
def word_counter(tokens):
word_counts = Counter()
word_counts.update(tokens)
return word_counts
word_count = word_counter(tokens)
word_count.most_common(5)» [18]
```

[('a', 7), ('of', 4), ('in', 4), ('be', 4), ('an', 3)]

Для визуализации результатов подсчета используется код:

«import matplotlib.pyplot as plt

x = list(word_count.keys())[:10]

y = list(word_count.values())[:10]

plt.bar(x, y)

plt.show();» [18].

Результирующий график представлен на рисунке 15.


Рисунок 15 – График результатов подсчета слов в тексте

Таким образом, тестирование программы подтвердило правильность реализации алгоритмов токенизации и подсчета слов в тексте.

## Выводы к главе 3

Третья глава посвящена разработке программы интеллектуального анализа текстовых данных.

Результаты проделанной работы позволили сделать следующие выводы:

- наилучшими характеристиками для разработки программ на языке Python обладает IDE Eclipse+PyDEv;
- тестирование подтвердило правильность реализации алгоритмов

интеллектуального анализа текстовых данных.

## Заключение

Выпускная квалификационная работа посвящена проблеме исследования особенностей практического применения алгоритмов интеллектуального анализа для повышения качества анализа текстовых данных.

В процессе работы над ВКР решены следующие задачи:

- проведен анализ методов и алгоритмов интеллектуального анализа текстовых данных. Отмечно, что методы интеллектуального анализа текстовых данных относятся к области обработки естественного языка - NLP. Проанализированы методы токенизации, TF-IDF, стемминга,и лемматизации и стоп-листинга. Описаны области применения каждого метода, его достоинства и недостатки;
- исследованы особенности применения алгоритмов интеллектуального анализа текстовых данных в различных прикладных задачах. Описаны характеристики алгоритмов BPE, TF-IDF, WordNet и кластеризации текстовых документов;
- разработана и протестирована программа, реализующая алгоритмы интеллектуального анализа текстовых данных. Как показал анализ, наилучшими характеристиками для разработки программ на языке Python обладает IDE Eclipse+PyDEv. С помощью библитеки NLTK разработаны программные коды алгоритмов токенизации подсчета слов в тексте. Тестирование подтвердило работоспособность разработанной программы и правильность реализации алгоритмов интеллектуального анализа текстовых данных.

Результаты бакалаврской работы представляют научно-практический интерес и могут быть рекомендованы для бизнес-аналитиков и разработчиков программ, использующих для принятия управленческих решений методы и алгоритмы интеллектуального анализа текстовых данных.

## Список используемой литературы

1. Бахтин А.В. Алгоритмы извлечения из неструктурированных текстовых источников метаинформации о научно-технических конференциях. М: МГУ [Электронный ресурс]. URL: https://www.hse.ru/data/2015/06/07/1097438594/presentation_cfp.pdf (дата обращения: 10.06.2021).

2. Библиотека NTLK [Электронный ресурс]. URL: http://www.nltk.org/ (дата обращения: 10.06.2021).

3. ВКонтакте опубликовали библиотеку для предобработки текстовых данных [Электронный ресурс]. URL: https://neurohive.io/ru/novosti/vkontakte-opublikovali-biblioteku-dlya-predobrabotki-tekstovyh-dannyh/ (дата обращения: 10.06.2021).

4. Кластеризация и классификация больших текстовых данных с помощью машинного обучения на Java [Электронный ресурс]. URL: https://itnan.ru/post.php?c=1&p=529548 (дата обращения: 10.06.2021).

5. Краткое руководство. Знакомство с интегрированной средой разработки Visual Studio [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/visualstudio/ide/quickstart-ide-orientation?view=vs-2019 (дата обращения: 10.06.2021).

6. Ле Мань Ха. Оптимизация алгоритма KNN для классификации // ТРУДЫ МФТИ. 2016. Том 8, № 1. С. 92-94.

7. Левенштейн В.И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. АН СССР. 1965. 163 (4). С. 845–848.

8. Метод TF-IDF [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/TF-IDF (дата обращения: 10.06.2021).

9. Отраднов К.К., Раев В.К. Экспериментальное исследование эффективности методик векторизации текстовых документов и алгоритмов их кластеризации. Вестник РГРТУ. 2018. № 64. С. 74-82.

10. Самуйлов С. В. Алгоритмы и структуры обработки данных : учебное

пособие. Саратов : Вузовское образование, 2016. 132 c. [Электронный ресурс]. URL: https://www.iprbookshop.ru/47275.html (дата обращения: 12.06.2021).

11. Что такое «текстовая аналитика»? [Электронный ресурс]. URL: https://www.megaputer.com/ru/what-is-text-analytics/ (дата обращения: 10.06.2021).

12. B. Bhatt, S. Kunnath, P. Bhattacharyya. Graph Based Algorithm for Automatic Domain Segmentation of WordNet, Center for Indian Language Technology Indian Institute of Technology Bombay Mumbai, India, 2014.

13. Eclipse IDE [Электронный ресурс]. URL: https://www.eclipse.org/eclipseide/ (дата обращения: 10.06.2021).

14. EM кластеризация [Электронный ресурс]. URL: https://basegroup.ru/deductor/function/algorithm/em-clustering (дата обращения: 10.06.2021).

15. K-Nearest Neighbors Algorithm for Machine Learning [Электронный ресурс]. URL: https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26 (дата обращения: 10.06.2021).

16. Kowsher Md et al. Lemmatization Algorithm Development for Bangla Natural Language Processing, 2020.

17. Natural Language Processing [Электронный ресурс]. URL: https://monkeylearn.com/natural-language-processing/ (дата обращения: 10.06.2021).

18. Natural Language Processing with Python [Электронный ресурс]. URL: https://medium.com/analytics-vidhya/natural-language-processing-with-python-cb3f83d5a393 (дата обращения: 10.06.2021).

19. Pycharm IDE [Электронный ресурс]. URL: https://www.jetbrains.com/pycharm/ (дата обращения: 10.06.2021).

20. Stemming & Lemmatization [Электронный ресурс]. URL: https://www.tutorialspoint.com/natural_language_toolkit/natural_language_toolkit_stemming_lemmatization.htm (дата обращения: 10.06.2021).

21. T. Kudo Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates, 2018.

22. Three subword algorithms help to improve your NLP model performance [Электронный ресурс]. URL: https://medium.com/@makcedward/how-subword-helps-on-your-nlp-model-83dd1b836f46 (дата обращения: 10.06.2021).

23. Vijayarani S. and Janani R. Text Mining: Open source tokenization tools – and analisys, Advanced Computational Intelligence: An International Journal (ACII), Vol.3, No.1, January 2016.

24. WordNet [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/WordNet (дата обращения: 10.06.2021).