МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

Направление подготовки/специальность
09.04.02 Информационные системы и технологии

направленность (профиль)/специализация
«Анализ и синтез информационных систем»

Выпускная квалификационная работа

Разработка NLP алгоритмов извлечения дефиниций

Обучающегося 2 курса
очной формы обучения
Першина Сергея Владимировича

Руководитель выпускной квалификационной работы:
доктор технических наук, профессор,
Фомин В.В.

Санкт-Петербург
2024

Содержание

Введение

Растущее количество разнообразной информации, свободно распространяемой посредством различных информационных технологий, значительно осложняет доступ к получению необходимых знаний и поиску данных, необходимых для решения производственных, просветительских, жизненных и других задач. Появляется необходимость извлечения структурированных, важных знаний из общего объема неструктурированной информации. Для решения этой проблемы широко разрабатываются и применяются технологии, облегчающие сортировку, поиск и генерацию информации. Разрабатывается широкий спектр алгоритмов поисковых систем, сортировки информации, чат-боты с генеративным искусственным интеллектом, алгоритмы распознавания и анализа речи, системы голосовых помощников и другие разнообразные системы, на сегодняшний день активно применяемые как в повседневной жизни, так и в производственных процессах и в просветительской деятельности. Большинство таких систем работает на основе современных алгоритмов и методов обработки естественного языка (NLP, Natural Language Processing).

Одной из основных проблем систем на основе NLP является извлечение дефиниций и сущностей из исходного текста. Корректное извлечение сущностей позволяет более точно провести дальнейший анализ текстов и эффективнее достичь целей разрабатываемых систем. Таким образом, исследованию методов автоматического извлечения дефиниций отводится важная роль при решении задач NLP.

Целью выпускной работы является разработка алгоритма извлечения дефиниций, классифицирующих группу из текстов, имеющих общий характер.

Разработка алгоритма включает выполнение таких задач:

- Проектирование алгоритма извлечения дефиниций.
- Разработка современных методов предварительной обработки текстов, методов представления, оценки, классификации токенов.

- Создание модели алгоритма извлечения дефиниций.
- Тестирование и анализ созданной модели.

Результат разработки алгоритма подразумевает создание системы извлечения дефиниций, позволяющей выделить из группы извлекаемых текстов наиболее значимые токены, позволяющие проводить дальнейший анализ и классификацию текстов.

1 Технологии обработки текстов естественного языка

На сегодняшний день существуют различные методы NLP, позволяющие подготовить тексты, написанные на естественном языке, к автоматизированной компьютерной обработке и последующему анализу. Такие методы позволяют взаимодействовать с текстом как с математической моделью, производить над текстом различные операции, проводить анализ свойств и характеристик, сравнивать тексты как модели.

1.1 Предобработка текстов естественного языка

Различные методы предобработки текстов позволяют привести тексты различного содержания и формата к единому виду, используемому в дальнейшей обработке. Хотя данные методы могут как положительно, так и отрицательно сказаться на результатах обработки, их выборочное применение может существенно улучшить скорость работы алгоритма, уменьшить затраты данных на обработку и повысить качество конечного результата [7].

Токенизация

Токенизация — неотъемлемый процесс предобработки текстов естественного языка. Под токенизацией в NLP подразумевают процесс разделения текста на базовые единицы, называемые токенами, для дальнейшей обработки уже массива токенов [8]. Токенами могут быть слова, фразы, символы или другие различные части текста, которые в зависимости от цели обработки должны оставаться неделимыми единицами. Основной целью токенизации является сохранение в токенах их лексического смысла таким образом, чтобы каждый токен содержал как можно больше информации. Разные задачи NLP могут требовать разделения токенов различными способами. Не всегда существует необходимость разделять текст на наименьшие по объему токены, символы или отдельные слова. Такое разделение может привести к потере важных связей между словами, а в следствии и потери смысла слов. Также разделение слов на токены большего

размера, словосочетаний, предложений, абзацев, может привести к потере морфологических и других свойств отдельных слов.

Проверка на орфографию

Зачастую в текстах встречаются различного рода опечатки, которые могут серьезно повлиять на результат обработки текста [9]. Одна из проблем орфографических ошибок и опечаток, встречающихся в тексте, — создание новых уникальных языковых единиц. Появление таких сущностей влияет на частотные характеристики других слов, а сами опечатки и ошибки могут привести к неправильной обработке языковых конструкций, влиять на семантику.

Приведение к одному регистру

Один из основных процессов предобработки текстов — приведение символов к одному единому регистру. Данный процесс может с минимальными затратами значительно ускорить и улучшить процесс обработки текстов за счет объединения одинаковых слов с разным регистром букв и избавления от чувствительности к регистру. Но в то же время различные именованные сущности, которые принято указывать в тексте с заглавной буквой, потеряют свой изначальный смысл и будут объединены с похожими по написанию другими сущностями, что может повлиять на частотные, семантические и другие их характеристики.

Распознавание и связывание именованных сущностей

Распознавание именованных сущностей (Named Entity Recognition, NER) — это процесс распознавания и классификации выражений со специальными значениями в тексте, написанном на естественном языке. Такие выражения могут варьироваться от имени человека и названия организации до дат и часто содержат важную для текста информацию [10]. Распознавание таких сущностей необходимо по причине того, что большинство именованных сущностей имеют одинаковое написание с обычными сущностями, что при приведении текста к одному регистру затрудняет их различие, впоследствии теряя информацию и свойства, которые они хранят.

Связывание именованных сущностей (Named Entity Linking, NEL) имеет схожую задачу. Задача связывания заключается в соотнесении именованной сущности с её корректным значением, позволяющим определить её роль в семантике текста. Для применения NEL обычно необходима база знаний, содержащая данные о связях предполагаемой именованной сущности.

Расширение аббревиатур, сокращений

Аббревиатуры и сокращения широко используются в различных видах текстов для избежания повторений и сокращения его объема. Хотя такие сущности могут обрабатываться в их исходном значении, в некоторых случаях для понимания контекста следует прибегнуть к восстановлению аббревиатур и сокращений до их изначального значения.

Удаление знаков пунктуации, эмодзи, специальных символов, ссылок

Процесс удаления наименее содержащих смысловое содержание языковых единиц — один из самых распространенных способов предобработки текста. Хотя знаки пунктуации и эмодзи иногда несут информацию о тональности текста и его структуре, в большинстве задач NLP они являются лишними и удаляются на этапе предобработки из-за невозможности установления связей между ними и словами текста.

Ссылки и специальные символы обычно не несут какой-либо информации, необходимой для анализа. Обычно доменные имена могут даже помешать распознаванию характеристик текста из-за частого содержания случайных для анализируемого текста слов.

Удаление стоп-слов

Стоп-словом в NLP обычно называют часто встречаемые в тексте слова, не несущие большой смысловой нагрузки. Обычно это артикли, междометия, союзы и т. д., которые сильно влияют на частотные характеристики других слов текста. Для каждой отдельной задачи используют свой список стоп-слов, которые удаляют из текста на этапе предобработки.

Удаление наиболее распространенных слов

Удаление наиболее распространенных слов, так же как и удаление стоп-слов, несет в себе смысл сокращения количества малозначащих сущностей в обрабатываемом тексте. Наиболее распространенными словами принято считать слова, которые являются типичными для конкретных классов текстов и меняются в зависимости от тематики.

Стемминг

Стемминг — это процесс приведения слова к его основе. Цель стемминга — сократить количество лексических различий между термами текста, облегчая процесс его дальнейшей обработки [11]. Сокращение слова до основы позволяет не терять его лексический смысл, при этом существенно уменьшая количество различных сущностей для обработки, объединяя все включения однокоренных слов текста в одну сущность. Хотя стемминг может улучшить обработку текстов естественного языка, в процессе сокращения слова теряются важные смысловые характеристики, такие как время, падеж и др. Без этих характеристик многие методы анализа языка становятся недоступны. Для решения этой проблемы вместо алгоритмов стемминга можно применять другие алгоритмы нормализации, например, лемматизацию слов.

Лемматизация

Лемматизация, так же как и стемминг, представляет собой процесс сокращения слова до его базовой формы. В отличие от стемминга, лемматизация сокращает слово до его нормальной формы, леммы, учитывая и сохраняя часть речи исходного слова. Такой метод позволяет не только, так же как и стемминг, уменьшить количество обрабатываемых сущностей, но и помогает не терять при этом характеристики слова.

Одной из проблем лемматизации является необходимость точного определения части речи и нахождения правильной основы слова. Иначе лемму слова будет невозможно определить, что может привести к ошибке

лемматизации и дальнейшей потере характеристик сущности и неправильному определению связей между сущностями.

1.2 Способы представления текста

В NLP важнейшую роль играет метод представления обрабатываемого текста. От модели представления зависят способы взаимодействия с термами, их хранение и вид. В различных представлениях текст может принимать вид как набора числовых значений, отвечающих за различные характеристики текста, так и словарей и матриц. В зависимости от характера задачи, решаемой при помощи средств NLP, и методов дальнейшей обработки, могут быть использованы разные модели представления текста.

Векторное представление

В большинстве случаев модель векторного представления состоит из набора векторов, соответствующего количеству токенов текста, содержащих последовательность значений, отражающих набор признаков токена. Набор признаков для каждой отдельной задачи NLP определяется отдельно.

В контексте NLP выделяют вектора с разреженными данными и плотные вектора. Плотными векторами называют вектора, все значения которых значимы и используются для классификации токена. Признаки в таких векторах выражены неявно и зависят от всех значений вектора. Разреженные вектора представляют собой набор значений, где каждое значение соответствует одному из признаков токена.

Мешок слов

Мешок слов представляет собой упрощенную модель представления, при которой текст представляется в виде множества токенов с указанием их характеристики, чаще всего количества или частоты. Мешок слов в контексте NLP применяют как упрощенный разреженный вектор или как промежуточный этап для вычисления признаков токена.

N-граммы

В основе N-граммных моделей лежит предположение о том, что вероятность появления последующих слов-токенов текста зависит от предыдущих. В данной модели текст разделен на наборы по N токенов, где вероятность N-го токена зависит от вероятности 1:N–1 токенов [12]. Разделения по два токена называются биграммами, по три токена — триграммами, по четыре и более — называются N-граммами. Таким образом, в биграмме вероятность второго токена зависит только от вероятности предыдущего.

N-граммные модели в основном применяют для предугадывания следующих токенов и их генерации.

Булевая модель

Булевая модель представления текста основана на булевой алгебре. Представляет собой набор текстов и соответствующие им булевы выражения, аргументами которого являются токены, извлеченные из текста. Между текстом и токенами выстраивается полное соответствие.

Матрица документ-терм

Модель представления текста в виде матрицы документ-терм представляет собой матрицу $n \times m$, где m — количество текстов в наборе, а n — общее количество токенов набора. Ячейками матрицы могут являться различные характеристики токенов, например частота, количество токенов в тексте и др.

1.3 Методы оценки токенов

В NLP существуют различные методы оценки извлеченных из текстов токенов. Наиболее важными для алгоритмов классификации текстов являются методы оценки сходства. Оценка сходства позволяет причислять токены к какой-либо группе или сравнивать их, что и является способом классификации.

Частотность токена, количество повторений

Самой простой мерой оценки токенов является оценка их частоты или количества повторений в документе. При помощи показателя частоты можно определить, насколько выбранный токен распространен в тексте, несет ли он значимую для текста информацию, ее объем. Например, одним из признаков незначимого слова является высокая частота его появления в тексте, а очень малое количество какого-либо слова в тексте в задачах классификации говорит о низкой принадлежности этого слова к какому-либо классу.

Мера Жаккара

Мера Жаккара представляет собой показатель сходства каких-либо объектов. В NLP применяется для нахождения сходств или различий между наборами сущностей. Наборы состоят из признаков оцениваемых текстов и, в зависимости от задачи, могут быть словами, предложениями или другими полученными при извлечении токенов признаками.

Коэффициент Жаккара рассчитывается как

$$K_J = \frac{n(A \cap B)}{n(A)+n(B)-n(A \cap B)} = \frac{n(A \cap B)}{n(A \cup B)}, \quad (1)$$

где A и B — наборы извлеченных признаков.

Данный коэффициент не учитывает ни семантическое сходство сущностей, ни порядок их появления. Помимо этого, на меру влияет сам способ выбора наборов для сравнения признаков. Например, в наборах, состоящих из большого количества слов, с большой вероятностью имеется множество распространенных слов, общих для обоих наборов, сильно влияющих на результирующий коэффициент.

Расстояние Левенштейна

Мера, показывающая количество действий, необходимых для преобразования одного набора в другой. Разрешенными операциями для расчета метрики являются: удаление, вставка и замена. Расстояние

Левенштейна рассчитывается как количество таких операций или как сумма весов каждой из операций.

Наиболее часто данная мера применяется для сравнения двух слов между собой с целью поиска недостающих или неправильных букв для проверки грамматики. Помимо этого, данную меру можно применять для сравнения других наборов извлеченных признаков.

Данный коэффициент не учитывает семантическое сходство сущностей, а также требует существенного количества вычислений для расчета коэффициента.

Расстояние Хэмминга

Частный случай расстояния Левенштейна. Расстояние Хэмминга — метрика, применяемая к признакам одинаковой длины, отражающая количество замен, необходимых для преобразования одного набора в другой. В отличие от расстояния Левенштейна требует меньшего количества вычислений, но является более строгой метрикой сходства, не учитывающей свойства признаков.

Евклидово расстояние

Данная мера определяет расстояние между двумя векторами признаков. Расстояние прямо пропорционально схожести оцениваемых сущностей. Чем меньше расстояние, тем ближе друг к другу токены. Евклидово расстояние рассчитывается как

$$d(A,B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}, \quad (2)$$

где A и B — векторы извлеченных признаков, а n — количество извлеченных признаков. При расчете Евклидова расстояния, векторы извлеченных признаков должны иметь одинаковый размер.

В отличие от предыдущих метрик, работающими непосредственно с самими признаками, Евклидово расстояние подразумевает работу с

векторным представлением оцениваемого текста. Так как для процесса преобразования текста в векторное представление необходима предварительная оценка признаков, то данную метрику невозможно применить при первичной оценке и преобразовании в векторную модель.

Наиболее распространенной практикой является предварительная нормализация векторов признаков для более точного сравнения схожести различных исходных векторов.

Косинусная близость

Косинусная близость — мера, которая показывает угол между двумя векторами признаков. Чем меньше величина этого угла, тем более схожи объекты. Косинусная близость рассчитывается как

$$cos(\theta) = \frac{A \cdot B}{|A| * |B|}, \quad (3)$$

где A и B — векторы извлеченных признаков.

Так же, как и мера Евклидова расстояния, косинусная близость применяется к векторному представлению извлеченных признаков и не используется при первичной оценке.

Из-за того, что косинусная близость отражает угол, данная мера является нормализованной. Помимо этого, косинусная близость учитывает все извлеченные признаки.

Расстояние городских кварталов

Метрика, рассчитывающая расстояние между двумя векторами на основе «манхэттенского расстояния». В отличии от Евклидова расстояния, данная метрика может применятся к векторам с признаками различных типов. Метрика рассчитывается как

$$d(A, B) = \sum_{i=1}^{n} |A_i - B_i|, \quad (4)$$

где A и B — векторы извлеченных признаков, а n — количество извлеченных признаков. Так же, как и при расчете остальных мер, применяемых к векторному представлению, размер векторов должен быть одинаковым.

Расстояние Минковского

Обобщенная мера расчета расстояний между векторами — расстояние Минковского. К формуле расчета добавляется изменяемый параметр $p \geq 1$, в зависимости от которого свойства метрики меняются. Например при p =2 метрика становится мерой Евклидова расстояния. Метрика рассчитывается как

$$d(A,B) = (\sum_{i=1}^{n} |A_i - B_i|^p)^{\frac{1}{p}}, \quad (5)$$

где A и B — векторы извлеченных признаков, n — количество извлеченных признаков, p — параметр.

Расстояние перемещения слова

Метрика, основанная на метрике Вассерштейна, подразумевает нахождение минимальных расстояний, на которые токенам одного текста необходимо «переместиться», чтобы добиться семантики токенов другого текста [13].

Данная метрика в наивысшей степени учитывает семантическое сходство сущностей, так как сравнивает не сами расстояния между векторами признаков, а матрицы этих расстояний между сущностями.

TF-IDF

Наиболее распространенная в NLP метрика, которая определяет важность токена в тексте. Состоит из двух компонентов TF (Term Frequency, частота слова) и IDF (Inverse Document Frequency, обратная частота документа). Высокий показатель имеют токены, часто встречающиеся в пределах документа, но редко встречающиеся в других документах коллекции.

Таким образом мера TF-IDF показывает вес токена в пределах коллекции документов.

$$TF(t,d) = \frac{n_t}{\sum_k n_k}, \quad (6)$$

где n_t — число слов входящих в документ, а $\sum_k n_k$ — общее число слов в данном документе.

$$IDF(t,D) = log\left(\frac{|D|}{|\{d_i \in D | t \in d_i\}|}\right), \quad (7)$$

где $|D|$ — число документов в коллекции; $|\{d_i \in D | t \in d_i\}|$ — число документов из коллекции D в которых встречается t.

$$TF{-}IDF(t,d,D) = TF(t,d) \times IDF(t,D) \quad (8)$$

1.4 Методы классификации извлеченных данных

В результате обработки текстовых данных из текста будет получен набор признаков, присущих токенам этого текста. В зависимости от поставленной задачи NLP существует необходимость систематизации этого набора, разделении его на группы, классы, выявлении общих черт, имеющихся у схожих токенов. Процесс классификации строится на нахождении токенов со схожими признаками и объединении их в общую группу.

В NLP выделяют контролируемую, неконтролируемую и частичную виды классификации.

Контролируемая классификация

Контролируемая классификация представляет собой методы классификации, в основе которых лежит распределение токенов по уже готовым размеченным классам. На основе имеющихся данных об этих классах строится обучающая модель, с помощью которой алгоритмы выявляют принадлежность токенов и признаков к этим классам.

Среди наиболее распространенных методов контролируемой классификации выделяют: алгоритм Роккио, наивный Байесовский алгоритм,

метод деревьев решений, метод К-ближайших соседей и методы на основе нейронных сетей.

Алгоритм Роккио

Алгоритм Роккио является методом определения границ размеченных классов. Для определения границ классов посредством алгоритма Роккио необходимо найти центроид класса. Он вычисляется как усредненное значение векторов признаков класса:

$$\mu(\mathrm{c}) = \frac{1}{|D_c|}\sum_{d\in D} \nu(d), \quad (9)$$

где D_c — множество токенов из набора D, класс которых является c: $D_c = \{d: \langle d, \mathrm{c}\rangle \in D\}$, а $\nu(d)$ нормализованный вектор признаков токена d [14].

Границей между классами является прямая, равноудаленная от центроидов этих классов. Таким образом, для классификации при помощи алгоритма Роккио необходимо найти такой центроид класса, который является наименее удаленным от вектора признаков классифицируемого токена.

Наивный Байесовский алгоритм

Алгоритм представляет собой вероятностный классификатор, используя теорему Байеса для определения вероятности принадлежности токена к какому-либо классу. Данный метод основан на предположении о том, что набор признаков токенов какого-либо класса взаимонезависим.

Можно предположить, что вероятность принадлежности i-го признака к классу C является:

$$p(d_i|C) \quad (10)$$

Тогда вероятность встречи токена d, в классе C является:

$$p(d|C) = \prod_i p(d_i|C) \quad (11)$$

Таким образом вероятность принадлежности токена d к классу C, по теореме Байеса:

$$p(C|d) = \frac{p(C)}{p(d)}\prod_i p(d_i|C), \quad (12)$$

где $p(C)$ — априорная вероятность встретить документ класса C, а $p(d)$ — априорная вероятность встретить токен d. Данные величины можно вычислить при помощи уже размеченных классов токенов:

$$p(C) = \frac{d_C}{d} \quad (13)$$

Так как значения вероятности $p(d)$ известны и являются константой, можно не учитывать знаменатель при определении вероятности принадлежности.

Классификация токена посредством данного алгоритма происходит путем выбора класса с наибольшим значением вероятности для этого токена.

Деревья решений

Алгоритм деревьев решений для классификации токенов основан на принципе нахождения таких признаков токена, которые наилучшим образом разделяют выборку классов.

В наиболее простом исполнении данного алгоритма для построения дерева следует выполнить следующие действия:

1. Создать первый узел дерева, отражающий все размеченные классы токенов и все имеющиеся их признаки.
2. Выбрать из имеющихся признаков такой, что значения данного признака для имеющихся классов наиболее различны. Выбрать для этого признака подходящие пороговые значения.
3. На основе полученных пороговых значений признака разделить классы и их признаки на несколько групп, совпадающих с критериями порогового значения, без включения выбранного признака.
4. Обрабатывать полученные узлы дерева в соответствии с алгоритмом, пока в узлы не будут содержать только один класс токенов или не закончатся имеющиеся признаки.

Таким образом классифицируемые токены, двигаясь по дереву, будут причислены к одному единственному классу.

K-ближайших соседей

Метод К-ближайших соседей — метод контролируемой классификации, при котором класс токена определяется посредством определения классов k соседей, находящихся наиболее близко к набору признаков классифицируемого токена.

На этапе обучения алгоритма определяются векторы признаков размеченных токенов классов, каждому вектору присваивается собственная метка класса. На этапе классификации задается параметр k, определяющий количество влияющих на класс токена ближайших соседей. Класс тех соседей, количество которых в k ближайших экземплярах наибольшее, определяет класс классифицируемого токена.

Выбор параметра k зависит от задачи NLP. Большие параметры k уменьшают влияние шума на определение класса, но уменьшают влияние различий между этими классами. Также чётные значения параметра могут привести к неопределённости в выборе класса, классифицируя токен как принадлежащий одновременно нескольким классам. Таким образом, выбор параметра может существенно повлиять на точность работы метода.

Нейронные сети

Алгоритм нейросетевой классификации схож с использованием нейронных сетей в других областях машинного обучения и другой деятельности. Размеченный набор токенов подготавливается таким образом, чтобы набор их признаков был одинаковым. Для обучения нейронной сети на входной слой поступает набор подготовленных признаков, и веса нейронной сети корректируются до получения точного результата в обучающей выборке. Далее точность классификации нейронной сети проверяется на тестовой выборке. Если точность классификации низкая, процесс обучения повторяется до достижения высокого результата. После этого нейронную сеть можно применять как классификатор.

Нейронная сеть подходит не для каждой задачи классификации. Зачастую при обработке текстов набор признаков имеет гигантскую

размерность. Обработка нейронной сетью такого набора может приводить к существенным затратам во времени обработки и требовать высоких мощностей. Поэтому нейронную сеть стоит применять либо к небольшим плотным векторам признаков, либо к набору признаков небольшой размерности.

Неконтролируемая классификация

Неконтролируемая классификация или кластеризация — это подход определения групп токенов или кластеров, имеющих схожие свойства. Данный подход в NLP применяется для текстов и данных, заранее не имеющих меток принадлежности к классу, выполняющийся без использования обучающей выборки.

Также помимо классификации, методы кластеризации в NLP применяют для снижения размерности векторов признаков.

Среди наиболее распространенных методов неконтролируемой классификации выделяют: метод k-средних и иерархическую кластеризацию.

Метод k-средних

Одним из наиболее распространенных методов кластеризации является метод k-средних. Работа данного метода основана на минимизации суммарного квадратичного отклонения векторов признаков токенов от центров k кластеров. Параметр k выбирается заранее, по предполагаемому количеству кластеров.

Алгоритм k-средних следующий:

1. Случайным образом выбирается k случайных начальных векторов являющихся центрами кластеров.
2. Векторное пространство делится на группы в соответствии с выбранными центрами кластеров так, чтобы каждый вектор принадлежал только одному кластеру. Выбирается тот центр кластера, для которого квадрат Евклидова расстояния от вектора является минимальным.

3. Для каждого i кластера вычисляется его новый центр масс:

$$\mu_i = \frac{1}{|S_i|}\sum_{x \in S_i} x \ , \tag{14}$$

где S_i — набор точек i кластера; x — вектор признаков.

Таким образом, этапы 2 и 3 повторяются до того момента, пока центр масс не перестанет меняться.

Как и другие методы кластеризации, данный метод требует существенных затрат по времени и компьютерных мощностей для кластеризации высокоразмерных векторов признаков. Помимо этого, из-за неоднозначности выбора k начальных векторов результат данного метода кластеризации может существенно меняться в зависимости от выбора начальных центров масс.

Иерархическая кластеризация

Иерархическая кластеризация — метод кластеризации, в основе которого лежит объединение токенов с близкими векторами признаков в один кластер. Так, начиная с группы кластеров, количество которых соответствует группе токенов, кластеры с близким расстоянием друг от друга объединяются, создавая новый большой кластер. Выбор ближайших кластеров зависит от метода определения расстояния. Для определения расстояний чаще всего используют расстояния между элементами кластеров, между центрами масс кластеров и другие методы.

По сравнению с методом k-средних данный метод не требует выбора предварительных случайных параметров, поэтому результат для одинакового набора данных всегда будет совпадать, и точность данного метода будет постоянной.

Частичная классификация

Частичная классификация — это метод, который сочетает в себе методы контролируемой и неконтролируемой классификации. В этом методе для обучения алгоритмов контролируемой классификации используются неразмеченные данные или методы кластеризации.

Например, методы кластеризации позволяют очистить выборку признаков от шума. Затем на основе этих очищенных данных обучаются алгоритмы контролируемой классификации. Также методы кластеризации могут быть использованы для предварительного или дополнительного обучения алгоритмов контролируемой классификации, что повышает их точность.

2 Разработка алгоритма извлечения дефиниций

Извлечение дефиниций — это сложный процесс обработки текста с помощью методов обработки естественного языка (NLP). Он включает в себя предобработку текста, представление текста, оценку признаков и классификацию.

В общем смысле, дефиниции — это совокупность термина и его определения, которое отражает свойства и признаки этого термина. В контексте данной работы «извлечение дефиниций» означает извлечение из документа или текста слов, понятий, терминов и сущностей, которые характеризуют этот текст или класс подобных текстов.

Разработка алгоритма извлечения дефиниций предполагает создание программного средства, которое с помощью методов и инструментов NLP сможет извлекать из документов, текстов и других текстовых данных языковые единицы, соответствующие поставленной задаче.

2.1 Постановка задачи разрабатываемого алгоритма

Основной целью разрабатываемого алгоритма служит извлечение из набора текстов, размеченных с указанием классов принадлежности, сущностей, отражающих специфику текста, с помощью которых будет возможно определить принадлежность какого-либо текста к одному из указанных классов.

Задачей разрабатываемого алгоритма является преобразование набора текстовых данных, написанных на русском языке, в матрицу «документ-терм»,

содержащую наиболее значимые для каждого класса текстовых данных n-слов, в ячейках которой должны содержаться их числовые характеристики (например, частоты появления слов в тексте), позволяющие с высокой частотой определить методами классификации принадлежность текста, не участвующего в обучающей выборке, к одному из классов преобразуемых текстов.

Данную задачу в соответствии с изученными методами NLP обработки текста можно разделить на последовательность подзадач:

1. Необходимо предобработать размеченные текстовые данные для дальнейшей их обработки.
2. Провести оценку полученных токенов методами NLP. Вычислить их важность для каждого размеченного класса. Найти наиболее значимые токены.
3. Преобразовать полученные при оценке данные в доступный для классификации способ представления.
4. Оценить полученную модель на предмет применимости к методам классификации и оценить точность классификации на примере полученной модели.

2.2 Анализ средств разработки алгоритма

Основным инструментом разработки алгоритма является интерпретируемый язык программирования Python версии 3.10. Выбор данного инструмента обусловлен большим количеством модулей и фреймворков, имеющих инструментарий для обработки текста средствами NLP, чем других языков программирования.

Обзор NLP модулей Python

Для языка Python было создано множество модулей для токенизации, стемминга, лемматизации, удаления стоп-слов, векторизации и других NLP-методов. Наиболее распространенными из них являются модули «NLTK», «spaCy», «Scikit-learn», «Gensim», «TextBlob», «CoreNLP» и «pymorphy2».

Таблица сравнительного анализа данных модулей по основным критериям представлена в приложении А.

Наиболее важным критерием при определении инструментов работы с текстом является поддержка модулем русского языка. Из представленных модулей поддержкой русского языка обладают модули «NLTK», «spaCy» и «pymorphy2», причем модуль «pymorphy2» специализируется на работе с русским текстом. Так как он имеет все необходимые для выполнения задачи функции, для предобработки текста был выбран данный модуль.

Модуль «pymorphy2» предоставляет инструменты по быстрой лемматизации и стемминга текста, морфологического разбора и в со словами, отсутствующими в словаре.

Обзор модулей для работы с файлами Python

Большинство модулей для работы с различными форматами файлов в Python — встроенные.

Для работы со строками и их форматирования применяется встроенный модуль «string».

Для создания промежуточных данных, таблиц, матриц и словарей применяются встроенные модули «json», «csv».

Для эффективного хранения и быстрого доступа к данным, их сериализации применяется модуль «pickle», позволяющий преобразовывать словари и другие типы данных в бинарный код и хранить эти данные в файлах.

Обзор остальных модулей Python

Для отслеживания времени работы частей алгоритма и анализа скорости выполнения применяется модуль «time» встроенный в Python.

Для обработки текстовых данных, удаления знаков препинания и других не смысловых знаков применяется встроенный модуль регулярных выражений «re».

Так как работа алгоритма основана на частом взаимодействии с файловой системой, в которой хранятся текстовые документы и

промежуточные файлы обработки, для взаимодействия с файловой системой применяется модуль «pathlib».

2.3 Методы NLP, применяемые в алгоритме

В соответствии с задачами разрабатываемого алгоритма можно выделить три группы методов, применяемых в алгоритме: методы предобработки, методы оценки токенов и методы представления текста.

Характер задачи накладывает определенные ограничения на категорию используемых методов NLP-обработки текста. Процесс обучения классификатора, а соответственно, извлечения сущностей, должен быть интерпретируем и детерминирован для одинаковых групп текстов. Таким образом, неинтерпретируемые методы, а также методы, использующие случайные величины, например, применяющие нейронные сети, не будут получать точный результат обработки.

Помимо этого, обработка групп текстов часто требует взаимодействия с большими объемами данных. Поэтому необходимо использовать методы уменьшения размерности обрабатываемых данных.

Методы предобработки в алгоритме

Задача состоит в том, чтобы выполнить предобработку текста, оставив только наиболее значимые слова. Поэтому следует избавиться от всех незначащих единиц текста: удалить пунктуацию, числа, представленные в цифровом формате, опечатки.

Помимо этого, слова, имеющие одинаковое написание, написанные с буквы в верхнем и в нижнем регистре, следует принимать как одно и то же слово, поэтому необходимо привести все слова к единому нижнему регистру.

Кроме этого, для дальнейшей обработки и оценки слов текста их необходимо токенизировать, составив для каждого текста последовательность из слов, встречаемых в тексте.

Тексты могут содержать важные слова, часто встречающиеся в тексте в разных формах, для того чтобы увеличить значимость каждого такого слова и

уменьшить количество обрабатываемых сущностей, следует применить лемматизацию, сохраняя только начальные формы слов. Выбор лемматизации обусловлен неспособностью метода стемминга корректно определять общего представителя, что часто приводит к изменению семантики слов, подвергшихся данному методу. Процесс лемматизации же включает в себя использование словаря и морфологического анализа слов, что позволяет корректно определять нормальную форму слова, сохраняя его смысл.

Методы оценки токенов в алгоритме

Оценка токенов в алгоритме с учетом специфики задачи производится несколькими методами: нахождением частоты обрабатываемых токенов в тексте, их количества, а также нахождением метрик TF-IDF. Каждая из этих оценок помогает определить важность того или иного слова на разных этапах анализа текста. Например, оценка количества повторений слова в тексте помогает выявить те слова, которые не влияют на определение категории текста, встречаются редко или, наоборот, слишком часто. Также оценка частоты позволяет исключить слова, которые часто являются стоп-словами и не несут смысловой нагрузки. Оценка TF-IDF может применяться не только к текстам одного класса, но и определять важность токенов между классами. Такую оценку можно комбинировать с другими методами оценки для определения наиболее значимых слов.

Методы представления текста в алгоритме

В соответствии с задачей результирующим представлением должно являться представление «матрица документ-терм», содержащая частоты отобранных слов. Помимо этого, следует применять наиболее распространенный вид представления текста в виде мешка слов для хранения промежуточных результатов обработки.

2.4 Структура алгоритма

Структура алгоритма состоит из последовательности операций над текстовыми данными и термами, выполняемыми для получения результата. Последовательность применяемых методов NLP следующая:

1. Вычисляется уникальность терма в его классе: Создается словарь слов, содержащий оценку частот текстов, для каждого слова класса. Таким образом, каждому слову присваивается значение отношения текстов, в которых встречается данное слово, к общему количеству текстов.
2. Вычисляется частота терма в тексте: Создается словарь слов, содержащий оценку частот термов, для каждого текста группы. Таким образом, каждому тексту присваиваются значения отношения количества появлений каждого слова текста к общему количеству слов текста.
3. Вычисляется частота терма в классе: Создается словарь слов, содержащий оценку частот термов, для каждого класса. Каждому классу присваиваются значения отношения количество появлений каждого слова класса к общему количеству слов класса.
4. Важность терма для каждого класса текстов: К каждому терму применяется метод оценки важности TF-IDF, но частота термов (TF) рассчитывается относительно всех текстов каждого класса, а IDF рассчитывается относительно всех классов текстов.

На разных этапах работы алгоритма применяются оценка характеристик извлекаемых термов, для извлечения наиболее значимых. Оценка производится по критериям: уникальность терма в классе, величина значения TF-IDF.

Алгоритм представляет собой последовательность выполняемых функций, каждая из которых выполняет один из этапов обработки текстовых данных:

1. Обработка текстов классов:
   На данном этапе текстовые данные проходят процесс предобработки. Из

каталогов классов извлекаются тексты. Далее они проходят лемматизацию, удаление знаков препинания, включений неизвестных слов, опечаток. Все буквы приводятся к нижнему регистру. Создаются промежуточные файлы, содержащие словари уникальности терма в классе и количества включений терма в текст.

2. Оценка токенов на основе расчета метрик:
   На данном этапе отсеиваются слова, не прошедшие установленный в виде параметра порог уникальности терма в классе. Далее для прошедших отбор слов рассчитывается метрика важности терма для каждого класса текстов, основанная на метрике TF-IDF.
3. Формирование результирующего набора термов:
   Формируется результирующий набор термов, состоящий из n термов, количество которых определяется установленным параметром, содержащий токены, имеющие наибольшее значение модифицированного TF-IDF.
4. Формирование словаря источников и частот:
   Для каждого терма из результирующего набора определяются все тексты, в которых он содержится, и определяются частоты появления данного терма в этих текстах.
5. Формирование результирующей матрицы:
   На основании словаря источников и частот формируется результирующая матрица документ-терм размера $m \times n$, где m — общее количество текстов, содержащихся во всех размеченных классах, а n — установленный параметр количества извлекаемых термов.

2.5 Реализация алгоритма

Реализация алгоритма на языке Python указана в приложении Б. Скрипт имеет глобальные параметры. Их обозначение указано в таблице 1.

Таблица 1 — глобальные параметры разработанного алгоритма

Название параметра	Описание
RAW_DATA	Относительный путь к директории, содержащей директории классов текстов с текстами внутри.
DATA_SET	Относительный путь к директории, содержащей промежуточные файлы обработки текста.
RESULT	Относительный путь к директории содержащей результирующие файлы.
POS_FILTER	Фильтр используемых для обработки частей речи.
LEMMATIZE_TEXT	Булевый параметр определяющий использование лемматизации библиотеки «pymorphy2».
LINE_NUMBER_FOR_EXTRACTION	Параметр количества строк извлекаемых из результирующей таблицы весов TF-IDF.
WORD_USAGE_THRESHOLD	Параметр определяющий минимальную частоту появления слова в текстах класса для дальнейшей его обработки.

CSV_DELIMITER_OPTION	Опциональный разделитель для CSV файлов, изменяющий метод разделения ячеек.
USE_CSV_DELIMITER	Булевый параметр определяющий использование опционального разделителя CSV файлов.
GENERATE_READABLE_SOURCE_LIST	Булевый параметр, определяющий возможность генерации читаемого списка источников результирующих слов.
TO_REMOVE_PATHS	Указание путей к файлам, которые будут удалены по завершению работы программы.
MORPH	Указатель на метод анализа морфологических признаков.

Программа состоит из последовательности выполняемых функций, перечисленных в функции «main». Блок схема последовательности функций представлена на рисунке 1. Функционал реализованных функций соответствует описанию последовательности структуры алгоритма. Описание каждой функции указано в таблице 2. Выполнение основных методов предобработки, расчета метрик и оценки токенов производится в функциях «group_processing» и «group_tf_dict». Блок-схемы алгоритмов, выполняемых в данных функциях, изображены на рисунке 2 и 3 соответственно.

Таблица 2 — описание функций разработанного алгоритма.

Название функции	Описание функции

group_processing	Обработка текстов классов.
group_tf_dict	Оценка токенов на основе расчета метрик.
result_to_csv	Формирование результирующего набора термов.
dictionary_sources	Формирование словаря источников и частот.
final_constructor	Формирование результирующей матрицы.
readable_dictionary_sources	Генерация читаемого списка источников для слов и их показателей TF-IDF.
garbage_removal	Удаление файлов путь к которым указан в глобальном параметре TO_REMOVE_PATHS.

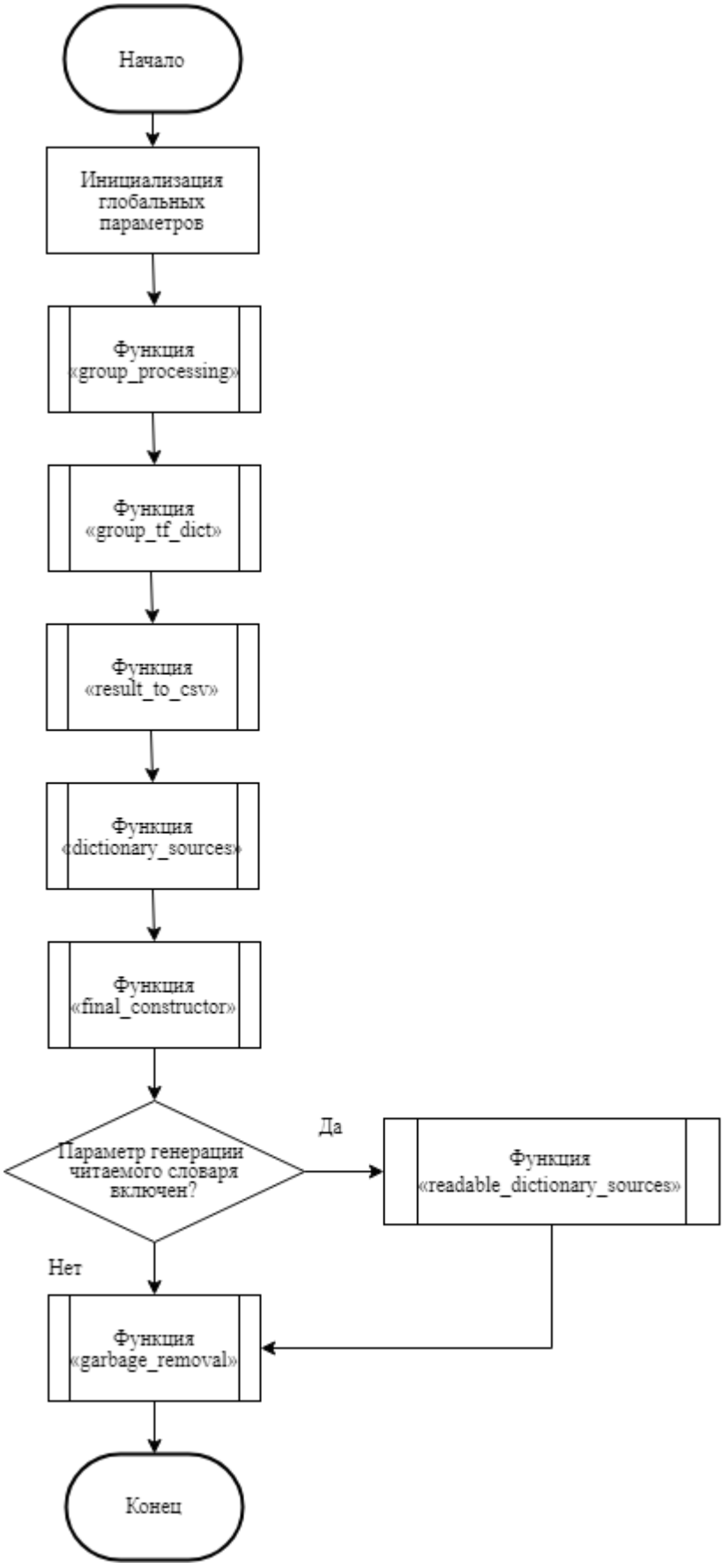

Рисунок 1 — блок-схема алгоритма функции «main»

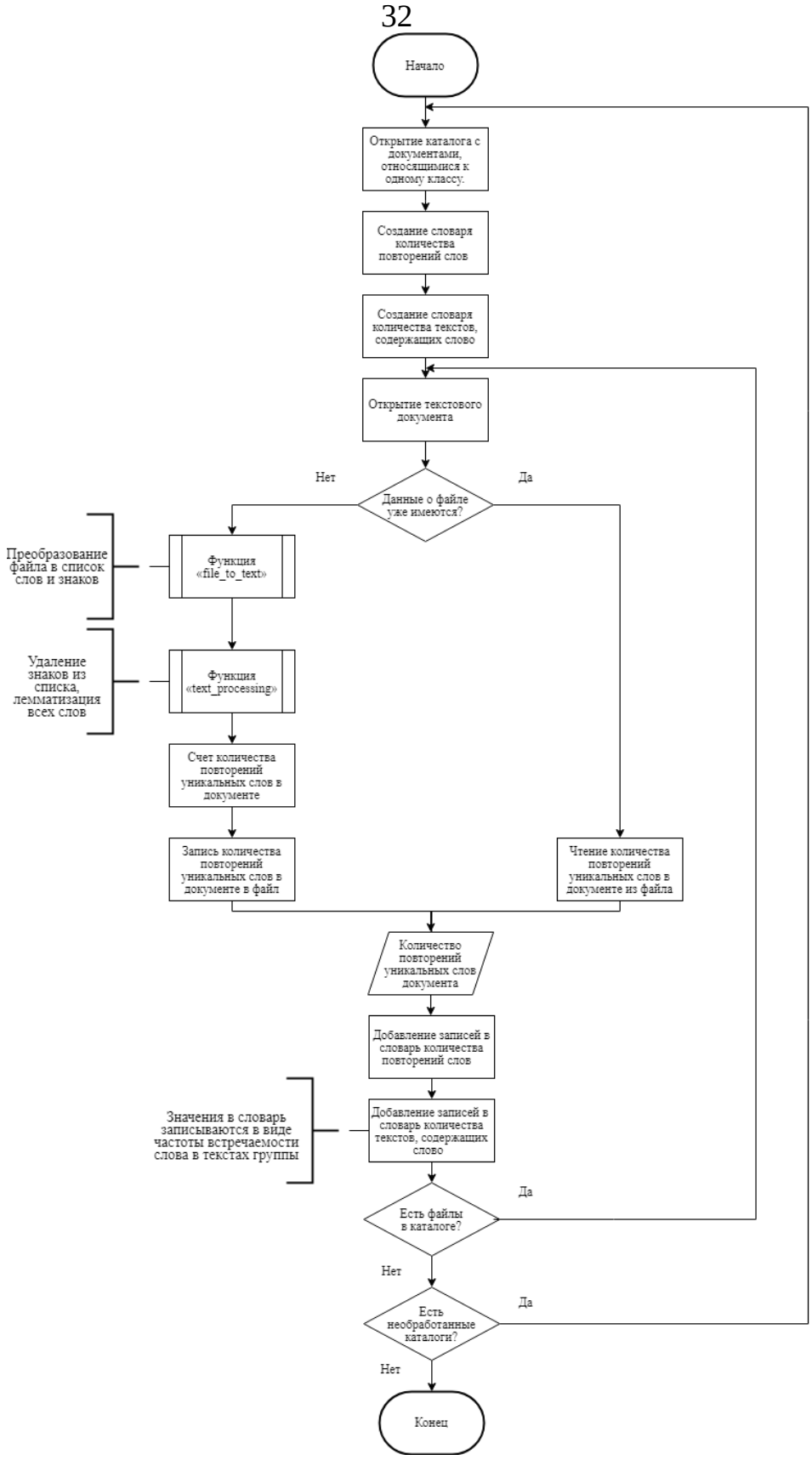

Рисунок 2 — блок-схема алгоритма функции «group_processing»

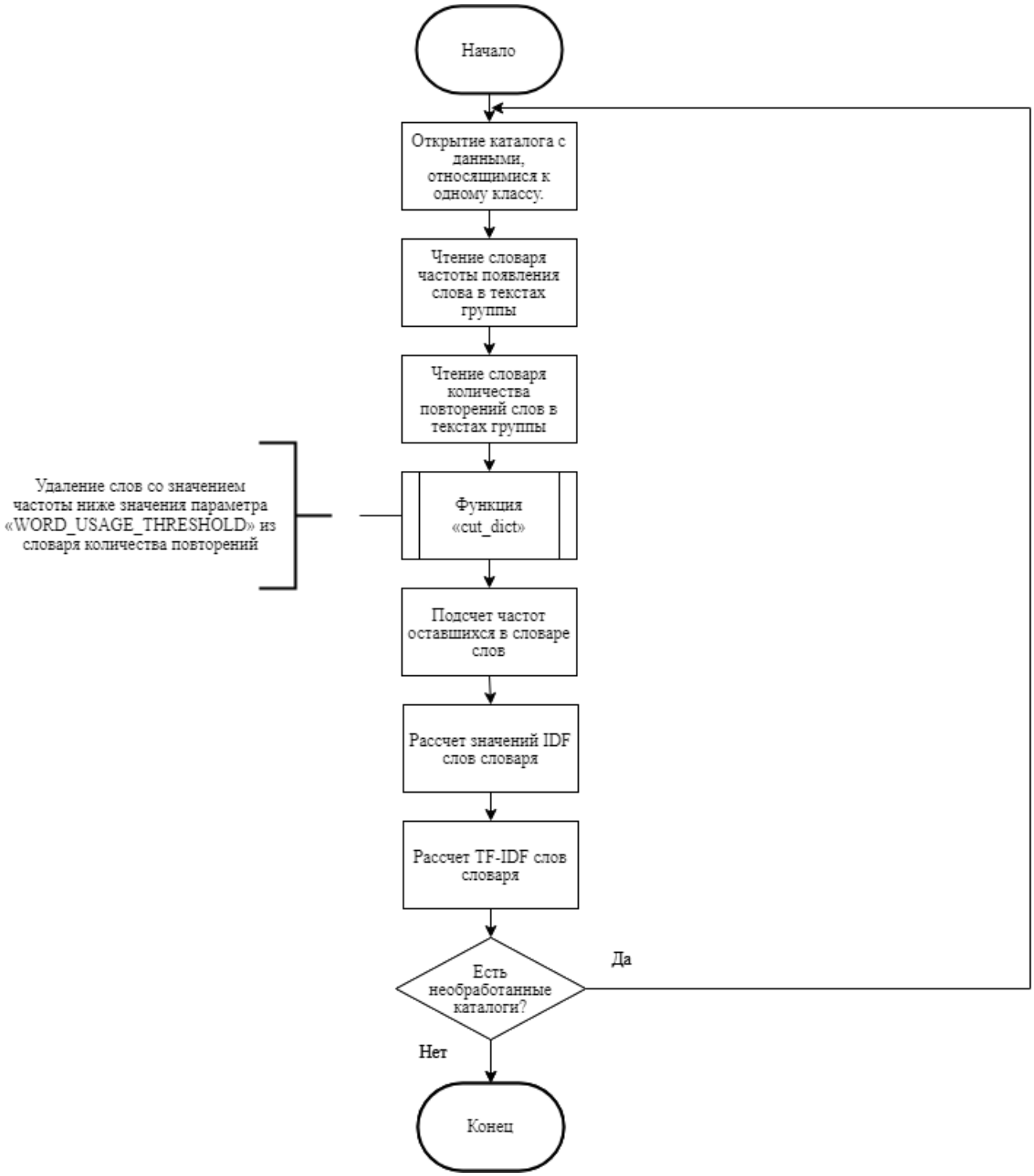

Рисунок 3 — блок-схема алгоритма функции «group_tf_dict»

Пример результата работы функции «readable_dictionary_sources» представлен на рисунке 4. В результате работы функции в .JSON файл помещается список названий текстовых документов и список токенов,

соответствующий данному документу. У каждого токена указывается рассчитанная величина его TF-IDF.


```
"Nepomnyaschaya Dina. Rozovyy surok": [
    [
        "прыгнуть",
        0.0006385696040868455
    ],
    [
        "снять",
        0.0006385696040868455
    ],
    [
        "утром",
        0.0006385696040868455
    ],
    [
        "кричать",
        0.0006385696040868455
    ],
    [
        "тёплый",
        0.0006385696040868455
    ],
    [
        "удивиться",
        0.0012771392081736911
    ],
    [
        "упасть",
        0.0006385696040868455
    ],
    [
        "весна",
        0.0006385696040868455
    ],
    [
        "дождь",
        0.0038314176245210726
    ],
```


Рисунок 4 — блок-схема алгоритма функции «readable_dictionary_sources»

Помимо основных функций программа состоит из вспомогательных функций, не перечисленных в функции «main», но выполняющихся по ходу работы программы. Описание этих функций указано в таблице 3.

Таблица 3 — описание вспомогательных функций разработанного алгоритма.

Название функции	Описание функции
text_processing	В зависимости от параметра LEMMATIZE_TEXT лемматизирует текст при помощи функции lemmatize_sentence или удаляет из текста знаки препинания и разбивает его на токены.
file_to_text	Преобразует текстовый файл в список слов с буквами нижнего регистра.
lemmatize_sentence	Преобразует текст в разбитый на токены лемматизированный текст.
word_frequency	Определяет все частоты слов словаря, состоящего из слова и количества его появлений в тексте.
cut_dict	В соответствии с параметром WORD_USAGE_THRESHOLD отсекает токены, значение частоты появления которых в текстах класса — ниже указанного.
columns_to_rows	Меняет ориентацию списка с горизонтальной в вертикальную.
file_word_frequency	Определяет частоту выбранного слова в тексте.

selected_word_dict	Функция выбирающая верхние n строк результирующей таблицы значений TF-IDF в соответствии с указанным параметром.

Результатом работы программы является получение файлов «result.csv» и «formatted_result.csv»

Пример результата работы функции «result_to_csv» показан на рисунке 5. Результат содержит столбцы слов и их значений TF-IDF для каждого класса текстов.

	A	B	C	D	E	F
1	Слова из каталога 1_RawData\БЕЗ ЭМОЦИЙ	TF-IDF	Слова из каталога 1_RawData\УЖАСЫ	TF-IDF	Слова из каталога 1_RawData\ЮМОР	TF-IDF
2	философия	0.00282298623134636	дверь	0.0008845603825314258	ну	0.001428287995042468
3	бытие	0.0024931981202077663	ну	0.0007144526166599977	иван	0.0006641519715947311
4	развитие	0.0014972380245692142	дед	0.0007134068382850982	дядя	0.0005534599763289426
5	теория	0.0014972380245692142	словно	0.0007011398349831033	сидеть	0.00048395236795718393
6	истина	0.001259790584549427	голос	0.0005709704141423584	ах	0.0004512827499297532
7	понятие	0.0011938329623217084	ладонь	0.0005410669840926307	врач	0.00044309924598677226
8	г	0.0011839393189875505	жёлтый	0.0005250353697491453	дверь	0.00042267268500156665
9	представление	0.00110479017231428 8	туман	0.00047694052671868925	дама	0.0003874219834302598
10	изменение	0.0009596834034133069	окно	0.0004363633994093154	сыр	0.0003618776768304624
11	объект	0.0009431939978563773	стена	0.0004363633994093154	сад	0.00034059075466396466
12	общественный	0.0009398961167449913	тварь	0.0004288456836882332	доктор	0.00032525370184135414
13	основа	0.0009168109489652898	комната	0.0004097378360555266	рубль	0.0003235612169307664
14	сущность	0.0009135130678539038	куртка	0.0004088061657588765	здоровье	0.00031930383249746685
15	определённый	0.000910215186742518	палец	0.00040086264827093036	голос	0.0003063984147780872
16	политический	0.0008838321378514304	спина	0.0003934666584504335	пиво	0.0003022742947642686
17	современный	0.0008805342567400446	кровь	0.0003905082625222347	супруг	0.0003022742947642686
18	философский	0.0008739384945172726	дождь	0.00037273503348603444	гражданин	0.0003022742947642686
19	государство	0.0008673427322945008	замок	0.00037273503348603444	ой	0.00029801691033096904
20	исторический	0.0008574490889603429	ударить	0.0003647192263142918	водка	0.00029801691033096904

Рисунок 5 — пример результата работы функции «result_to_csv»

На рисунке 6 изображен пример результирующей матрицы документ-терм. Матрица содержит частоты слов, полученные с помощью алгоритма, для каждого текста в каждом классе.

Группа	Имя Файла	очнуться	платон	опасность	давить	аналогичный	тёплый	внезапный	вырастать
БЕЗ ЭМОЦИЙ	Green Growth Зеленая революция	0	0	0.00107138074	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Адам Смит Теория нравственных чувств	0	0	0.000157753588	0.000157753588	0	0	0	0
БЕЗ ЭМОЦИЙ	Аксенов ГП Причина времени	0	0.002689979825	9.607070804111	0	0.000192141416	0	0	0
БЕЗ ЭМОЦИЙ	Алексей Турчин Футурология	0	0	0.00056481219	0	0	0	0.000282406099	0
БЕЗ ЭМОЦИЙ	Алфавит жизни Роман Куклин	0	0	0	0.000243190661	0	0	0	0
БЕЗ ЭМОЦИЙ	Американские просвятители	0	0	0.000117882824	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Античная метафизика Бутина-Шабаль	0	0.000175039383	0.000175039383	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Артур Шопенгауэр Афоризмы житейской мудрости (фрагм)	0	0.000106360348	0.000425441395	0	0.000106360348	0.000106360348	0	0.00010636034
БЕЗ ЭМОЦИЙ	Архетипы и коллективное бессознательное. Карл Густав Юнг	0	0	0.000297176820	0	0.000594353640	0	0	0
БЕЗ ЭМОЦИЙ	Базалук Теория эволюции	0	0	0	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Базалук Философия образования	0	0	0	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Беседы и размышления Сёрен Кьеркегор	0	0	0	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Бондарев Хаос Основа реальности	0	0	0	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Борис Бирюков Жар холодных чисел и пафос бесстрастной логики	0	0.000431406384	8.628127696289	0	0	0	0	8.62812769628
БЕЗ ЭМОЦИЙ	Введение в социальное проектирование. Андрей Захаров	0	0	0	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Воин Философия и глобальный кризис	0	0	0.000536768652	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Гриффен Диалектика общественного развития	0	0	0.000263019463	0	0.000526038926	0	0	0
БЕЗ ЭМОЦИЙ	Гриффен Феномен техники	0	0	0	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Гуревич Средневековый мир	0	0	0	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Давид против Голиафа. Гейдар Джахидович Джемаль	0	0.000356531660	7.130633200228	0	0	7.130633200228	0	7.13063320022
БЕЗ ЭМОЦИЙ	Джозеф Кэмпбелл Мифы и личностные изменения	0	0	0	0.000126582278	0	0	0	0
БЕЗ ЭМОЦИЙ	Джулиан Баджини Свинья которая хотела чтоб ее съели	0	0	0	0	0	0.000291375291	0	0
БЕЗ ЭМОЦИЙ	Диктатура Карл Шмитт	0	0	0.000472478147	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Дугин в поисках темного логоса	0	0.000398883127	0	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Дугин Основы геополитики	0	0	0.000568343279	0	0.000284171639	0	0	0
БЕЗ ЭМОЦИЙ	Жиль Делез Эмпиризм и субъективность	0	0	0	0	9.905894006934	0	0	0
БЕЗ ЭМОЦИЙ	Жирар Рене Вещи сокрытые от создания мира	0	0.005604857543	0.000700607192	0	0.000233535730	0	0	0
БЕЗ ЭМОЦИЙ	Законы диалектики. Всеобщая мировая ирония Гегель	0	0	0	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Захаров-Гезехус Гены судьба духовность	0	0.000307219662	0	0	0	0	0	0.00030721966
БЕЗ ЭМОЦИЙ	Ильенков Об идолах и идеалах	0	0	0.000162892979	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Имре Лакатос Бесконечный регресс и основания математики	0	0	0.000149611011	0	0	0	0	0
БЕЗ ЭМОЦИЙ	Искусство и философия. Строева	0	0.000335345405	0.000111781801	0	0	0	0	0.00022356360
БЕЗ ЭМОЦИЙ	Истинная жизнь Ален Бадью	0	0	0	0	0	0	0	0

Рисунок 6 — пример результата работы функции final_constructor

На рисунке 7 изображен пример вывода консоли. Вывод консоли содержит указание обработанных текстовых документов, обработанных групп документов и время выполнения алгоритма.

```
 |Уникальных слов в тексте: 1087|
97. Эго. Жан-Люк Марион
 |Уникальных слов в тексте: 1073|
98. Юлиус Эвола Восстание против современного мира
 |Уникальных слов в тексте: 2199|

|Уникальных слов в дирректории "Старшая школа++": 31691|
------------------------------------------------------------
------------------------------------------------------------

--- Program works for 545.137330532074 seconds ---
```

Рисунок 7 — пример вывода работы консоли.

3. Анализ и сравнение результатов алгоритма

Для оценки реализованного алгоритма и определения его эффективности при помощи чат-бота с генеративным искусственным интеллектом «ChatGPT-4o» был создан классификатор, использующий деревья решений, реализованный при помощи инструментария языка программирования Python. Программный код классифицирующего алгоритма представлен в приложении В. Структура директории программы представлена на рисунке 8. При помощи данного алгоритма была протестирована эффективность классификации текстов при использовании полученного в результате работы реализованного алгоритма (программный код алгоритма указан в приложении Б) набора признаков и набора признаков, полученного при помощи нахождения слов с наибольшими значениями TF-IDF (программный код алгоритма указан в приложении Г), выбранного как контрольный набор. Примеры наборов признаков указаны на рисунках 9 и 10 соответственно.

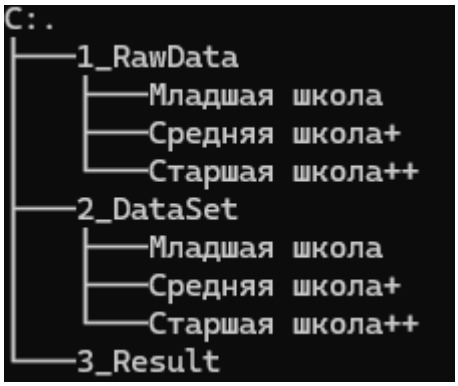

Рисунок 8 — структура директории программы

Для работы предложенного программного кода требуется установить дополнительные модули. Команда установки модулей:

pip install pandas numpy scikit-learn pymorphy2 (15)

Class_Name	Text_Name	век	богатство	поведение	искусственный
Младшая школа	T Pamela, Мери Поппенс (Заходер)	9.75609756097561e-05	0	0.0001951219512195122	0
Младшая школа	Yakovlev Yuriy, Umka	0	0	0	0
Младшая школа	Акимов Пройти но вернуться	0	0	0	0
Младшая школа	Аксаков Аленький цветочек	0.0004362367311327614	0.0014541224371092047	0	0
Младшая школа	Андерсен Гадкий утенок	0	0	0	0
Младшая школа	Андерсон Дикие лебеди	0	0	0	0
Младшая школа	Волк и семеро козлят	0	0	0	0
Младшая школа	Городок в табакерке Одоевский	0.0009643201542912247	0	0	0
Младшая школа	Грим Бременские музыканты	0	0	0	0
Младшая школа	Грим Госпожа метелица	0	0	0	0
Младшая школа	Грим Сказка Белоснежка и семь гномов	0	0	0	0
Младшая школа	Грим Соломинка уголёк и боб	0	0	0	0
Младшая школа	Джеймс Боуэн Уличный кот по имени Боб	0.0001631321370309951	0	0	0
Младшая школа	Дикий помещик СалтыковЩедрин	0	0	0	0
Младшая школа	Драгунский Зеленчатые леопарды	0	0	0	0
Младшая школа	Драгунский Англичанин Павля	0	0	0	0
Младшая школа	Драгунский Арбузный переулок	0	0	0	0
Младшая школа	Драгунский Белые амадины	0	0	0	0
Младшая школа	Драгунский Где это виданно	0	0	0	0
Младшая школа	Житков Белый домик	0	0	0	0
Младшая школа	Житков Беспризорная кошка	0	0	0	0
Младшая школа	Житков Про обезьянку	0	0	0	0
Младшая школа	Зайкин кораблик Сутеев	0	0	0	0
Младшая школа	Золушка или хрустальная туфелька Шарль Пьеро	0	0	0	0
Младшая школа	Зощенко Бабушкин Подарок	0	0	0	0
Младшая школа	Зощенко Великие Путешественники	0	0	0	0
Младшая школа	Калиф аист Гауф	0	0	0	0
Младшая школа	Каменный цветок Бажов	0	0	0	0
Младшая школа	Карлик Нос Гауф	0	0	0	0
Младшая школа	Киплинг Братья Маугли	0	0	0	0
Младшая школа	Киплинг Дикие собаки	0.00014287755393627662	0	0	0
Младшая школа	Киплинг Кошка которая гуляла сама по себе	0	0	0	0
Младшая школа	Киплинг Рикки Тикки Тави	0	0	0	0
Младшая школа	Красная шапочка Шарль Пьеро	0	0	0	0
Младшая школа	Лигред Мирабель	0	0	0	0
Младшая школа	Линдгред Карлсон который живёт на крыше читать	8.868786306593942e-05	0	0	0
Младшая школа	Лиса и тетерев	0	0	0	0
Младшая школа	Лягушка путешественница Гаршин	0.0008771929824561404	0	0	0
Младшая школа	Максим Горикий воробьишко	0	0	0	0
Младшая школа	Малахитовая шкатулка Бажов	0.0001562011871290221 8	0.0001562011871290221 8	0	0
Младшая школа	Маленький Мук Гауф	0	0.0006432246998284735	0	0

Рисунок 9 — пример набора признаков, полученных с помощью собственного алгоритма

Class_Name	Text_Name	эраст	вселенной	образом	мама
Младшая школа	Nepomnyaschaya Dina, Rozovyy surok - BooksCafe.Net.txt	0.0	0.0	0.0	0.001154068090017311
Младшая школа	T Pamela, Мери Поппенс (Заходер).txt	0.0	0.0	0.0	0.0001836884643644379
Младшая школа	Yakovlev Yuriy, Umka.txt	0.0	0.0	0.0	0.0
Младшая школа	Акимов Пройти но вернуться.txt	0.0	0.0	0.0006231306081754736	0.0001246261216350947
Младшая школа	Аксаков Аленький цветочек.txt	0.0	0.0	0.0	0.0
Младшая школа	Андерсен Гадкий утенок.txt	0.0	0.0	0.0	0.0
Младшая школа	Андерсон Дикие лебеди.txt	0.0	0.0	0.0	0.0
Младшая школа	Волк и семеро козлят.txt	0.0	0.0	0.0	0.0
Младшая школа	Городок в табакерке Одоевский.txt	0.0	0.0	0.0	0.0
Младшая школа	Грим Бременские музыканты.txt	0.0	0.0	0.0	0.0
Младшая школа	Грим Госпожа метелица.txt	0.0	0.0	0.0	0.0
Младшая школа	Грим Сказка Белоснежка и семь гномов.txt	0.0	0.0	0.0	0.0
Младшая школа	Грим Соломинка уголёк и боб.txt	0.0	0.0	0.0	0.0
Младшая школа	Джеймс Боуэн Уличный кот по имени Боб.txt	0.0	0.0	0.0001623640201331385	0.000324728040266277
Младшая школа	Дикий помещик СалтыковЩедрин.txt	0.0	0.0	0.0	0.0
Младшая школа	Драгунский Зеленчатые леопарды .txt	0.0	0.0	0.0	0.0010330578512396695
Младшая школа	Драгунский Англичанин Павля.txt	0.0	0.0	0.0	0.0029585798816568047
Младшая школа	Драгунский Арбузный переулок.txt	0.0	0.0	0.0	0.007390983000739098
Младшая школа	Драгунский Белые амадины.txt	0.0	0.0	0.0	0.002129925452609159
Младшая школа	Драгунский Где это виданно.txt	0.0	0.0	0.0	0.0
Младшая школа	Житков Белый домик.txt	0.0	0.0	0.0	0.0037688442211055275
Младшая школа	Житков Беспризорная кошка.txt	0.0	0.0	0.0	0.0
Младшая школа	Житков Про обезьянку.txt	0.0	0.0	0.0	0.0006069802731411229
Младшая школа	Зайкин кораблик Сутеев.txt	0.0	0.0	0.0	0.003215434083601286
Младшая школа	Золушка или хрустальная туфелька Шарль Пьеро.txt	0.0	0.0	0.0004411116012351125	0.0
Младшая школа	Зощенко Бабушкин Подарок.txt	0.0	0.0	0.0	0.0024968789013732834
Младшая школа	Зощенко Великие Путешественники.txt	0.0	0.0	0.0	0.000591715976331361
Младшая школа	Калиф аист Гауф.txt	0.0	0.0	0.0	0.0
Младшая школа	Каменный цветок Бажов.txt	0.0	0.0	0.0	0.0
Младшая школа	Карлик Нос Гауф.txt	0.0	0.0	0.0	0.0
Младшая школа	Киплинг Братья Маугли.txt	0.0	0.0	0.0	0.0
Младшая школа	Киплинг Дикие собаки.txt	0.0	0.0	0.0	0.0
Младшая школа	Киплинг Кошка которая гуляла сама по себе.txt	0.0	0.0	0.0	0.0
Младшая школа	Киплинг Рикки Тикки Тави .txt	0.0	0.0	0.00020173492031470649	0.0
Младшая школа	Красная шапочка Шарль Пьеро.txt	0.0	0.0	0.0	0.0019230769230769232
Младшая школа	Лигред Мирабель.txt	0.0	0.0	0.0	0.0072585147962032385
Младшая школа	Линдгред Карлсон который живёт на крыше читать.txt	0.0	0.0	0.0	0.003002126506275278
Младшая школа	Лиса и тетерев.txt	0.0	0.0	0.0	0.0
Младшая школа	Лягушка путешественница Гаршин.txt	0.0	0.0	0.0	0.0
Младшая школа	Максим Горикий воробьишко.txt	0.0	0.0	0.0	0.003316749585406302
Младшая школа	Малахитовая шкатулка Бажов.txt	0.0	0.0	0.0	0.0
Младшая школа	Маленький Мук Гауф.txt	0.0	0.0	0.0	0.0

Рисунок 10 — пример набора признаков, полученных с помощью контрольного алгоритма

3.1 Структура оценки эффективности алгоритма

Оценка реализованного алгоритма проводится посредством сравнения точности определения класса текстов x случайно выбранных документов из

подготовленного блока классификатора с обучением на основе полученных данных алгоритма извлечения дефиниций и классификатора с обучением на основе полученных данных алгоритма подбора признаков с помощью TF-IDF и тесте соответствия Хи-квадрат.

Был определен следующий алгоритм оценки:

1. На вход алгоритму оценки поступает набор предварительно размеченных текстов, для которых путем экспертной самостоятельной оценки был определен один из классов принадлежности. Каждый класс соответствует текстам, изучаемым на соответствующем этапе обучения.
2. Из общего набора текстовых файлов разного класса случайным образом выбираются x текстовых файлов от каждого класса текстов. Их класс сохраняется, и затем данные документы помещаются в отдельный каталог для оценки. Данные документы не участвуют в обучении моделей.
3. Каждая из сравниваемых моделей на основе оставшихся в каталогах размеченных текстов создает матрицу документ-терм с выделенными из текста m наиболее значимыми словами, указывая их соответствующий оригинальный класс, название текстового файла и частоту данного слова в нем. Для алгоритма извлечения дефиниций параметр «WORD_USAGE_THRESHOLD», определяющий минимальную частоту появления слова в классе текста, варьируется.
4. Алгоритм классификации обучается на полученной матрице документ-терм. Затем для каждого случайно отобранного документа при помощи алгоритма деревьев решений для классификации определяется предполагаемый класс текста. Для всего каталога случайно выбранных текстов определяется точность, с которой алгоритм предсказал класс документа.
5. Этапы 1-4 повторяются n раз. Вычисляется средняя точность для каждого из алгоритмов.

3.2 Процесс проведения экспериментов

В качестве размеченного набора текстовых документов был выбран блок из 307 текстов, изучаемых в школьной программе. Путем экспертной самостоятельной оценки, данный блок текстов был разделен на три класса: «Младшая школа», «Средняя школа» и «Старшая школа», соответствующие изучаемым на каждом периоде обучения произведениям.

Количество извлекаемых текстов из каждого класса вычисляется случайным образом и находится в промежутке от 5 до 10 документов. Для обоих алгоритмов количество извлекаемых слов одинаково и равняется 100. Для полной оценки алгоритма извлечения дефиниций, параметр «WORD_USAGE_THRESHOLD» был определен в значениях 0.1,0.2,0.3,0.4,0.5,0.6,0.7и0.8. Было выбрано количество повторений циклов алгоритмов, равное 20. Результат проведения экспериментов указан в таблице 4. Диаграмма зависимости точности от параметра «WORD_USAGE_THRESHOLD» представлена на рисунке 11.

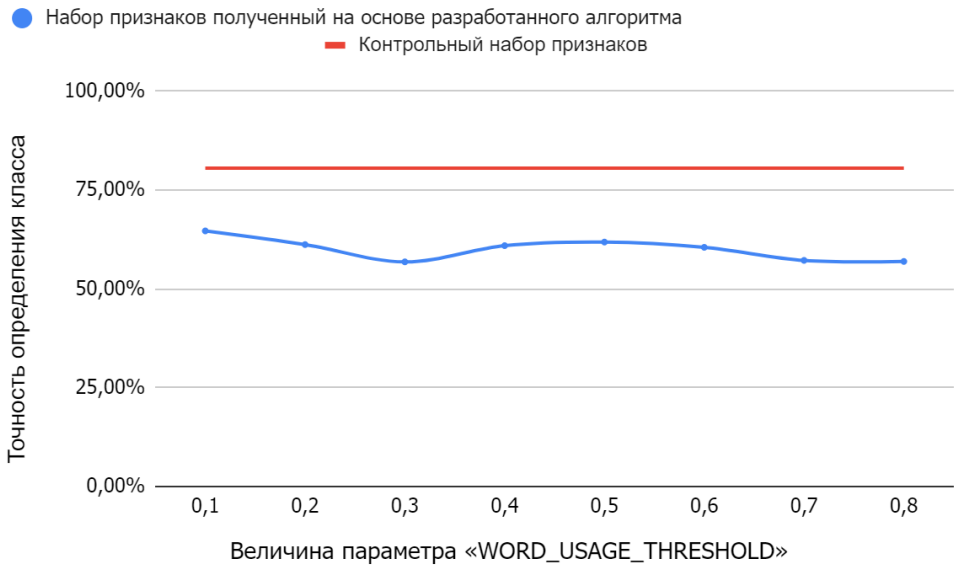

Рисунок 11 — диаграмма зависимости точности определения класса от параметра порога

Таблица 4 — точность определения класса текста

Цикл	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**Контр.**
1	66,67%	66,67%	66,67%	47,62%	62,96%	83,33%	58,33%	77,78%	60%
2	58,33%	62,50%	70,83%	66,67%	44,44%	46,67%	50%	66,67%	88,89%
3	60%	44,44%	44,44%	61,11%	51,85%	50%	61,90%	48,15%	79,17%
4	60%	66,67%	61,90%	83,33%	55,56%	70,37%	66,67%	59,26%	92,59%
5	66,67%	60%	51,58%	55,56%	66,67%	37,50%	40%	80%	83,33%
6	71,43%	50%	66,67%	61,11%	57,14%	66,67%	66,67%	60%	75%
7	46,67%	57,14%	66,67%	70,83%	45,83%	46,67%	61,11%	33,33%	83,33%
8	54,17%	60%	60%	70,83%	47,62%	62,50%	62,96%	62,96%	90,48%
9	60%	55,56%	66,67%	53,33%	66,67%	77,78%	55,56%	66,67%	87,50%
10	66,67%	70,83%	59,26%	58,33%	79,17%	77,78%	54,17%	38,10%	73,33%
11	59,26%	58,33%	57,14%	73,33%	47,62%	58,33%	57,14%	37,04%	70,83%
12	59,26%	66,67%	57,14%	45,83%	70,37%	81,48%	73,33%	59,26%	86,67%
13	62,96%	57,14%	51,85%	61,90%	66,67%	46,67%	72,22%	52,38%	85,19%
14	75%	50%	53,33%	50%	76,19%	66,67%	37,04%	66,67%	83,33%
15	80%	62,96%	44,44%	59,26%	55,56%	60%	38,89%	44,44%	93,33%
16	73,33%	91,76%	61,11%	70,37%	66,67%	45,83%	66,67%	50%	88,89%
17	66,67%	71,43%	44,44%	53,33%	77,78%	70,37%	66,67%	62,96%	71,43%
18	62,50%	55,56%	55,56%	60%	70,83%	55,56%	45,83%	72,22%	62,50%
19	73,33%	54,17%	44,44%	50%	66,67%	54,17%	37,50%	47,62%	88,89%
20	70,83%	62,50%	52,38%	66,67%	61,11%	52,38%	71,43%	53,33%	66,67%
Сред.	64,69%	61,22%	56,83%	60,97%	61,87%	60,54%	57,20%	56,94%	80,57%
Медиан.	64,82%	60,00%	57,14%	60,56%	64,82%	59,17%	59,72%	59,26%	83%

3.3 Анализ результатов экспериментов

Для проведения анализа результатов экспериментов за базовый показатель точности определения класса возьмем показатель контрольного эксперимента, полученного с помощью алгоритма подбора признаков на основе TF-IDF и тесте соответствия Хи-квадрат. Общие результаты проведения экспериментов представлены в таблице 4.

Как можно заметить, изменения параметров разработанного алгоритма могут существенно повлиять на точность конечного результата определения класса текста. Среднее отклонение от среднего значения точности определения класса текста составляет 2,28%. Наибольший коэффициент точности был получен у модели со значением параметра «WORD_USAGE_THRESHOLD» равным 0.1. Вероятно, данный показатель точности обусловлен низким количеством шума в обучающей выборке, что способствовало более качественному обучению. Помимо этого, модель со значением параметра 0,5 имеет такой же медианный результат, что также показывает высокую точность модели.

Средний показатель отклонения точности от контрольного результата составил 20,54%. Однако наименьший показатель отклонения был у модели с параметром «WORD_USAGE_THRESHOLD», равным 0.1 . Он составил 15,88%, что на 4,66% ниже среднего значения.

Помимо этого были определены максимальные и минимальные показатели определения точности. Наибольшие значения были получены при помощи контрольного набора признаков, а также набора признаков со значением параметра «WORD_USAGE_THRESHOLD» равным 0.2 , с показателями, равными 93,33% и 91,76% соответственно. Наименьшие же показатели точности были получены при помощи набора признаков со значением параметра «WORD_USAGE_THRESHOLD» равным 0.8 — 33,33%. Минимальный и максимальный размах между максимальным и минимальным

значением показателя были получены при помощи набора признаков со значениями параметра «WORD_USAGE_THRESHOLD» равными 0.3 и 0.2 — 26,39% и 47,32% соответственно.

Данные анализа результатов работы алгоритмов представлены в таблице 5.

Таблица 5 — характеристики результатов работы алгоритмов

X	Сред.	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	Контр.
Макс.	80,22%	80,00%	91,76%	70,83%	83,33%	79,17%	83,33%	73,33%	80,00%	93,33%
Мин.	41,71%	46,67%	44,44%	44,44%	45,83%	44,44%	37,50%	37,04%	33,33%	60%
Размах	38,51%	33,33%	47,32%	26,39%	37,50%	34,73%	45,83%	36,29%	46,67%	33,33%
Разница Контр. - X	20,54%	15,88%	19,35%	23,74%	19,60%	18,70%	20,03%	23,36%	23,63%	0,00%
Отклонение от сред.	2,28%	4,66%	1,18%	3,21%	0,94%	1,84%	0,50%	2,83%	3,09%	-

Как можно заметить по размаху, определение текстов для обучения алгоритма классификации имеет существенное значение для показателя точности определения. Данный эффект, вероятно, имеет место потому, что изначальный набор обучающих текстов был самостоятельно классифицирован и изначально имел неточности в принадлежности текста к классу. Наиболее стабильными оказались модели со значениями параметра «WORD_USAGE_THRESHOLD», равными 0.1 и 0.3.

Заключение

В результате исследования методов NLP, а также проектирования алгоритма извлечения дефиниций был получен алгоритм, решающий такую задачу, как извлечение наиболее значимых сущностей из текста для дальнейших задач классификации.

В ходе работы были разработаны такие методы обработки текста на естественном языке, как:

- принципы предобработки текстов, позволяющие существенно улучшить точность выполнения задач NLP, избавиться от лишних шумов и подготовить текстовые данные к дальнейшей обработке;
- принципы представления текста, которые могут варьироваться в зависимости от специфики задач и методов их решений;
- принципы оценки текста, позволяющие подготовить набор признаков к дальнейшей классификации;

также были определены различия в подходах к классификации текстов, их особенности.

Была проведена оценка полученного алгоритма путем сравнения, оценена его эффективность.

В ходе анализа полученных результатов оценки разработанного алгоритма было получено, что наиболее точными моделями являются модели на основе набора признаков со значением параметра «WORD_USAGE_THRESHOLD», равным 0.1 и 0.5. Данные модели имеют высокие показатели средней точности, а также одни из самых низких показателей размаха из всех анализируемых алгоритмов.

Было установлено, что классификатор, обученный на основе признаков, полученных с помощью разработанного алгоритма извлечения дефиниций,

позволяет извлекать сущности из размеченных текстов в среднем на 20,54% и на 15,88% с параметром 0.1 близко к точности контрольного набора признаков.

В результате проведенной работы были решены следующие задачи:

1. Спроектирован алгоритм извлечения дефиниций, соответствующий требованиям.
2. Были разработаны необходимые для алгоритма методы NLP обработки текста.
3. При помощи языка Python был создан алгоритм извлечения дефиниций.
4. Разработанный алгоритм путем сравнения был протестирован, была проанализирована его эффективность.

Цель работы полностью достигнута.

Список литературы

1. Yeganova L. Measuring the relative importance of full text sections for information retrieval from scientific literature. / L. Yeganova, Won Kim, D. C. Comeau, W. J. Wilbur, Z. Lu // Proceedings of the BioNLP. - Association for Computational Linguistics, 2021. - P. 247–256.
2. R. Sonbol The Use of NLP-Based Text Representation Techniques to Support Requirement Engineering Tasks: A Systematic Mapping Review / R. Sonbol, G. Rebdawi, N. Ghneim // IEEE Access. - 2022. - №10. - P. 62811-62830.
3. Liu Y. Learning Structured Text Representations / Y. Liu, M. Lapata // Transactions of the Association for Computational Linguistics. - 2018. - №6. - P. 63–75.
4. Dogra V. A Complete Process of Text Classification System Using State-of-the-Art NLP Models / V. Dogra, S. Verma, Kavita, P. Chatterjee, J. Shafi, J. Choi, M. F. Ijaz // Computational Intelligence and Neuroscience. - 2022. - 26 p.
5. Кузнецова Ю. М. Дефиниции в научном тексте: функции, виды, способы выражения и возможности идентификации/ Кузнецова Ю. М. // Искусственный интеллект и принятие решений. - 2015. - № 3. - С. 70-82.
6. Camacho-Collados J. On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis / J. Camacho-Collados, M. T. Pilehvar - Cardiff: Cardiff University 2017. - 7 p.
7. IŞIK M. The impact of text pr The impact of text preprocessing on the processing on the prediction of review ratings / M. IŞIK, H. DAĞ // Turkish Journal of Electrical Engineering and Computer Sciences. - 2020. - №18. - P. 1405-1421.
8. J. J. Webster Tokenization as the initial phase in NLP / J. J. Webster, C. Kit// Proceedings of the 14th conference on Computational linguistics. - 4-th vol. - 1992. - P. 1106-1110.
9. Bertoldi N. Statistical Machine Translation of Texts with Misspelled Words / N. Bertoldi, M. Cettolo, M. Federico // Human Language Technologies:

The 2010 Annual Conference of the North American Chapter of the ACL,. - Los Angeles: Association for Computational Linguistics, 2010. - P. 412–419.

10. Konkol M. Named Entity Recognition / M. Konkol, M. Konopík // Text, Speech and Dialogue 2014. - 36 p.
11. Wang S. Large Language Models for Stemming: Promises, Pitfalls and Failures / S. Wang, S. Zhuang, G. Zuccon // Association for Computing Machinery. - 2024. - P. 1-8.
12. Jurafsky D. Speech and Language Processing. / D. Jurafsky, J. H. Martin - 3-rd vol. - 2024. - 569 p.
13. Kusner M. J. From Word Embeddings To Document Distances / M. J. Kusner, Y. Sun, N. I. Kolkin, K. Q. Weinberger // Proceedings of the 32nd International Conference on Machine Learning. - 2015. - P. 957-966.
14. Manning C. D. Introduction to Information Retrieval / C. D. Manning, P.Raghavan, H. Schütze // Cambridge University Press. - 2008.
15. Miyato T. Adversarial training methods for semi-supervised text classification / T. Miyato, A. M. Dai, I. Goodfellow // ICLR 2017. - 2017. - P. 11.
16. Батура Т. В. Методы автоматической классификации текстов / Батура Т. В. // Программные продукты и системы. - 2017. - Т. 30. - № 1. - С. 85–99; DOI: 10.15827/0236-235X.030.1.085-099.
17. Автоматическая обработка текстов на естественном языке и анализ данных: учеб. пособие / Большакова Е. И., Воронцов К. В., Ефремова Н. Э. и другие. — Москва: Изд-во НИУ ВШЭ, 2017. — 269 с.
18. Боярский К. К. Извлечение низкочастотных терминов из специализированных текстов / Боярский К. К., Арчакова Н. А., Каневский Е. А. // Data Analytics and Management in Data Intensive Domains - Selected Papers . – Ершово, Московская область, 2016. – С. 142-147.
19. Обзор методов автоматической обработки текстов на естественном языке / Белов С. Д., Зрелова Д. П., Зрелов П. В., Кореньков В. В.// Системный анализ в науке и образовании: сетевое научное издание. – 2020. – № 3. – С. 8–22. – URL: http://sanse.ru/download/401 (дата обращения: 23.04.2024). – Режим доступа: открытый доступ.

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

21. Васильев Ю. Обработка естественного языка. Python и spaCy на практике / Васильев Ю. — Санкт-Петербург: Питер, 2021. — 256 с.: ил. — (Серия «Библиотека программиста»).

22. Бенгфорт Б. Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка / Бенгфорт Б., Билбро Р., Охеда Т. — Санкт-Петербург: Питер, 2019. — 368 с.

23. Park Y. Hybrid text mining for finding abbreviations and their definitions / Y. Park, R. J. Byrd // IBM Thomas J. Watson Research Center – URL: https://aclanthology.org/W01-0516.pdf (дата обращения: 15.12.2023)

24. Гольдберг Й. Нейросетевые методы в обработке естественного языка / Гольдберг Й., пер. с анг. Слинкина А. А.. – Москва: ДМК Пресс, 2019. – 282 с.

25. Paroubek P. Principles of Evaluation in Natural Language Processing / P. Paroubek, S. Chaudiron, L. Hirschman // TAL, 2007. - P. 7-31.

26. Huang J. AEON: A Method for Automatic Evaluation of NLP Test Cases / J. Huang, J. Zhang, W. Wang, P. He, Y. Su, M. R. Lyu // ACM SIGSOFT International Symposium on Software Testing and Analysis. - 2022. - P. 13.

27. Baghbani S. Text Classification: process and Algorithms / S. Baghbani // 3rd International Conference on Research in Engineering, Science and Technology.. - Batumi, Georgia. - 2016. - P. 11.

28. Ghumade T. G. A Document Classification using NLP and Recurrent Neural Network / T. G. Ghumade, R. A. Deshmukh // International Journal of Engineering and Advanced Technology (IJEAT). - 2019. - №8. - P. 632-636.

29. Orphanos G. Decision Trees and NLP: A Case Study in POS Tagging. / G. Orphanos , D. Kalles , T. Papagelis , D. Christodoulakis // In proceedings of ACAI’99 - 1999. - P. 7.

30. Шевелев О. Г. Классификация текстов с помощью деревьев решений и нейронных сетей прямого распространения/ Шевелев О. Г, Петраков А. В. // Вестн. Том. гос. ун-та. - 2006. - № 290. - URL: https://cyberleninka.ru/article/n/klassifikatsiya-tekstov-s-pomoschyu-

dereviev-resheniy-i-neyronnyh-setey-pryamogo-rasprostraneniya (дата обращения: 06.04.2024).

ПРИЛОЖЕНИЕ А

Сравнительный анализ модулей обработки естественного языка Python

Особенности	**NLTK**	**Scikit-learn**	**spaCy**	**Gensim**	**TextBlob**	**CoreNLP**	**pymorphy2**
Токенизация	Поддерживает токенизацию предложений и слов	Ограниченная поддержка, требует внешних библиотек	Быстрая и точная токенизация, поддержка русского	Базовая токенизация	Простая токенизация предложений и слов	Продвинутая токенизация, включая многословные выражения	Токенизация через интеграцию с другими библиотеками
Лемматизация	Основная лемматизация, не специализирована для русского	Отсутствует встроенная поддержка	Основная лемматизация, поддержка русского	Нет	Базовая лемматизация	Основная лемматизация	Точная лемматизация русского языка
Морфологический анализ	Ограниченные возможности	Отсутствует	Основной морфологический анализ	Отсутствует	Ограниченный	Продвинутый морфологический анализ	Детальный морфологический анализ русского языка
Часть речи (POS-теггинг)	Предлагает POS-теггинг, но требует внешних моделей для русского	Ограниченная поддержка, требует внешних библиотек	Высококачественный POS-теггинг с предобученными моделями	Отсутствует	Базовый POS-теггинг	Высококачественный POS-теггинг	Надежный POS-теггинг с детальной грамматической информацией
Определение именованных сущностей (NER)	Базовые возможности NER, не специализированные для русского	Ограниченная поддержка, требует внешних библиотек	Поддержка NER для русского	Отсутствует	Ограниченный NER	Надежный NER с поддержкой русского	Улучшает NER других библиотек при интеграции
Синтаксический разбор	Синтаксический разбор, но не оптимизирован для русского	Ограниченная поддержка, требует внешних библиотек	Синтаксический разбор предложений	Отсутствует	Базовый синтаксический разбор	Состояние искусства синтаксического и семантического разбора	Поддержка через интеграцию с другими библиотеками
Анализ тональности	Отсутствует встроенная поддержка	Ограниченная поддержка, требует внешних библиотек	Базовый анализ тональности	Отсутствует	Встроенные инструменты для анализа тональности	Встроенный анализ тональности	Отсутствует
Тематическое моделирование	Ограниченные возможности	Отличная интеграция с машинным обучением	Ограниченная поддержка	Отличные возможности моделирования тем (LDA, Word2Vec)	Ограниченные возможности	Ограниченные возможности	Отсутствует
Производительность	Могут быть медленными и требовательными к ресурсам	Высокая производительность, но требует внешних инструментов для NLP	Высокая скорость и эффективность	Эффективная обработка больших текстов	Простая и легкая в использовании, но не для больших объемов	Может быть медленной и требовательной к ресурсам	Эффективна для морфологических задач

ПРИЛОЖЕНИЕ Б

Программный код разработанного алгоритма на языке Python

```
import string
import re
import json
import pymorphy2
import math
import _pickle as pickle
import csv
import time
from pathlib import Path
from collections import Counter

RAW_DATA = '1_RawData'
DATA_SET = '2_DataSet'
RESULT = '3_Result'
POS_FILTER = ['VERB',
'NOUN','ADJF','ADJS','COMP','INFN','PRTF','PRTS','GRND','NUMR','ADVB','NPRO','PRED','PREP','CONJ','PRCL','INTJ']
LEMMATIZE_TEXT = True
LINE_NUMBER_FOR_EXTRACTION = 300 # Количество строк извлекаемых из исходных текстов
WORD_USAGE_THRESHOLD = 0.2; # Порог встречаемости слова в классе, для добавления слова в словарь
CSV_DELIMITER_OPTION = [";","sep=;\n"] #Указание исользуемого разделителя для CSV файла
USE_CSV_DELIMITER = False #Опция использования разделителя CSV файла
GENERATE_READABLE_SOURCE_LIST = False
TO_REMOVE_PATHS = [Path(RESULT, "selected_tf-idf_dict.pickle"), Path(RESULT, "all_groups_external_tf-idf.json"), ]
MORPH = pymorphy2.MorphAnalyzer()

def text_processing(text, lemmatize_text = LEMMATIZE_TEXT):
    if lemmatize_text == True:
        processed_text = lemmatize_sentence(text)
    else:
        unpunctuated_text = text.replace("\n"," ").translate(str.maketrans("", "", TEXT_PUNCTUATION)).strip()
        processed_text = re.findall(r'\w+', unpunctuated_text)
    return Counter(processed_text)

def group_processing(folder_name):
    for group_path in Path(folder_name).iterdir():
        group_dict = {}
        group_uniqueness_dict = {}
        iterator = 1 #Для вывода нумерации
        word_in_class_number = 0
        if not Path(DATA_SET,group_path.name).exists():
            Path(DATA_SET,group_path.name).mkdir(parents = True, exist_ok = True)
        print('{0}\nОбработка текста в дирректории {1}\n{0}'.format(len(list(Path(group_path).name))*'----',Path(group_path).name))
        for text_file in Path(group_path).glob("*.txt"):
            try:
                file_path = Path(text_file)
                file_name = str(file_path.with_suffix("").relative_to(group_path))
                if not Path(DATA_SET, group_path.name, file_name+".pickle").exists():
                    file_dict = dict(text_processing(file_to_text(file_path)))
                    with open(Path(DATA_SET, group_path.name, file_name+".pickle"), 'wb') as file:
                        pickle.dump(file_dict,file)
                else:
                    with open(Path(DATA_SET, group_path.name, file_name+".pickle"), 'rb') as file:
                        file_dict = pickle.load(file)
                word_in_text_number = 0
                for word, value in file_dict.items():
                    if word in group_dict.keys():
                        group_dict.update({word:group_dict.get(word)+value})
                        group_uniqueness_dict.update({word:group_uniqueness_dict.get(word)+1})
                    else:
                        group_dict.update({word:value})
                        group_uniqueness_dict.update({word:1})
                        word_in_class_number+=1
                    word_in_text_number +=1
                print('{0}. {1} \n |Уникальных слов в тексте: {2}|'.format(iterator, file_name, word_in_text_number))
```

```
            except:
                file_path = Path(text_file)
                file_name = str(file_path.with_suffix("").relative_to(group_path))
                print("{0}. {1}\n Ошибка обработки текста".format(iterator, file_name))
                pass
            iterator += 1
        group_uniqueness_dict.update({word:(value/(iterator - 1)) for word, value in group_uniqueness_dict.items()})
        with open(Path(DATA_SET, group_path.name, "group_word_count_dict.pickle"), 'wb') as file:
          pickle.dump(dict(group_dict),file)
        with open(Path(DATA_SET, group_path.name, "group_uniqueness_dict.pickle"), 'wb') as file:
          pickle.dump(dict(group_uniqueness_dict),file)
        print('\n|Уникальных слов в дирректории "{0}": {1}|\n{2}\n{2}\n'.format(Path(group_path).name, word_in_class_number, 60 * '-'))
    return 0

def group_tf_dict():
    number_of_groups = len(list(Path(DATA_SET).iterdir()))
    all_groups_dict = {}
    result_dict = {}
    idf_groups_dict = {}
    for group_folder in Path(DATA_SET).iterdir():
        read_counted_dict_path = Path(group_folder,"group_word_count_dict.pickle")
        read_pattern_dict_path = Path(group_folder,"group_uniqueness_dict.pickle")
        write_file_path = Path(group_folder,"group_word_frequency_dict.pickle")
        with open(read_counted_dict_path, 'rb') as file:
            counted_dict = pickle.load(file)
        with open(read_pattern_dict_path, 'rb') as file:
            pattern_dict = pickle.load(file)
        group_dict = cut_dict(counted_dict, pattern_dict)
        group_frequency_dictionary = word_frequency(group_dict)
        with open(write_file_path, 'wb',) as file:
            pickle.dump(dict(group_frequency_dictionary), file)
        for word, value in group_dict.items():
            if word not in idf_groups_dict.keys():
                idf_groups_dict.update({word: 1})
            else:
                idf_groups_dict.update({word:idf_groups_dict.get(word)+1})
    group_frequencies = []
    for group_folder in Path(DATA_SET).iterdir():
        frequency_file_path = Path(group_folder,"group_word_frequency_dict.pickle")
        with open(frequency_file_path, 'rb') as file:
            group_frequencies.append(dict(pickle.load(file)))
    for word, value in idf_groups_dict.items():
        counter = 0
        value_list = [0] * number_of_groups
        for group in group_frequencies:
            if word not in group.keys():
                value_list[counter] = 0
            else:
                value_list[counter] = group[word] * math.log(number_of_groups/idf_groups_dict[word], number_of_groups)
            all_groups_dict.update({word:value_list})
            counter += 1
    for word, value in all_groups_dict.items():
        if not all_groups_dict[word] == [0] * number_of_groups:
            result_dict.update({word:value})
    if not Path(RESULT).exists():
        Path(RESULT).mkdir(parents = True, exist_ok = True)
    result_path = Path(RESULT, "all_groups_external_tf-idf.json")
    with open(result_path, 'w', encoding='utf-8') as file:
        json.dump(result_dict, file, ensure_ascii=False, indent = 4)
    return 0

def file_to_text(file_path):
    try:
        try:
            file = open(file_path, 'r', encoding = "utf-8")
            text = file.read()
            file.close()
        except:
```

```
            file = open(file_path, 'r', encoding = "cp1251")
            text = file.read()
            file.close()
        except:
            raise Exception("Неподдерживаемая кодировка текстового файла: " + file_path)
        return text.lower()

def lemmatize_sentence(text):
    lemmatized_text = []
    text_to_list = re.findall(r'\w*', text)
    for word in text_to_list:
        morphed_word = MORPH.parse(word)[0]
        if (morphed_word.tag.POS in POS_FILTER) or ('LATN' in morphed_word.tag):
            lemmatized_text.append(morphed_word.normal_form)
    return lemmatized_text

def word_frequency(counted_text):
    total_value = sum(counted_text.values())
    frequency_dictionary = {word: value/ total_value for word, value in counted_text.items()}
    return frequency_dictionary

def cut_dict(orig_dict, pattern_dict,threshold = WORD_USAGE_THRESHOLD):
    result_dict = {}
    for word, value in orig_dict.items():
        if pattern_dict.get(word) >= threshold:
            result_dict.update({word: value})
    return result_dict

def result_to_csv():
    source_path = Path(RESULT, "all_groups_external_tf-idf.json")
    listed_dict = []
    with open(source_path, 'r', encoding='utf-8') as file:
        word_dict = json.load(file)
    for i in range(len(list(Path(RAW_DATA).iterdir()))):
        word_group_value = []
        for word, value in word_dict.items():
            word_group_value.append([word, value[i]])
        word_group_value = sorted(word_group_value, key = lambda x:x[1],reverse = True)
        listed_dict.append(word_group_value)
    selected_word_dict(listed_dict, word_dict) # Процесс отсеивания первых n строк из таблицы
    listed_dict = columns_to_rows(listed_dict)
    result_path = Path(RESULT, "result.csv")
    table_header = []
    for folder_name in list(Path(RAW_DATA).iterdir()):
        table_header += [f'Слова из каталога {folder_name}', "TF-IDF"]
    with open(result_path, 'w', newline='') as csvfile:
        csvfile.write(CSV_DELIMITER_OPTION[1])
        writer = csv.writer(csvfile, delimiter= CSV_DELIMITER_OPTION[0])
        writer.writerow(table_header)
        for row in listed_dict:
            writer.writerow(row)
    return 0

def columns_to_rows(original_list):
    result_list = []
    for i in range(len(original_list[0])):
        new_row = []
        for sub_list in original_list:
            new_row.extend(sub_list[i])
        result_list.append(new_row)
    return result_list

def readable_dictionary_sources():
    dict_path = Path(RESULT, "selected_tf-idf_dict.pickle")
    sources_dict = {}
    possible_sources_paths = list(Path(DATA_SET).iterdir())
    with open(dict_path, 'rb') as file:
        words_dict = pickle.load(file)
    for word, value in words_dict.items():
```

```
        word_sources = []
        for value in words_dict[word]:
            word_value_sources = []
            if value:
                possible_sources = words_dict[word].index(value)
                for dict_file in Path(possible_sources_paths[possible_sources]).glob("*[!dict].pickle"):
                    dict_file_path = Path(dict_file)
                    with open(dict_file_path, 'rb') as file:
                        possible_source = pickle.load(file)
                        if word in possible_source:
                            word_value_sources.append([Path(dict_file).stem, file_word_frequency(word, possible_source)])
            word_sources.append(word_value_sources)
        sources_dict.update({word:word_sources})
    result_path = Path(RESULT, "readable_dict_sources.json")
    with open(result_path, 'w', encoding='utf-8') as file:
        json.dump(sources_dict, file, ensure_ascii=False, indent = 4)
    return 0


def dictionary_sources():
    dict_path = Path(RESULT, "selected_tf-idf_dict.pickle")
    sources_dict = {}
    possible_sources_paths = list(Path(DATA_SET).iterdir())
    with open(dict_path, 'rb') as file:
        words_dict = pickle.load(file)
    for sources_folder in possible_sources_paths:
        for source in Path(sources_folder).glob("*[!dict].pickle"):
            with open(source, 'rb') as file:
                possible_source = pickle.load(file)
            if (possible_source.keys() & words_dict.keys()):
                source_keys = []
                for key in list(possible_source.keys() & words_dict.keys()):
                    source_keys.append([key, file_word_frequency(key, possible_source)])
            sources_dict.update({Path(source).stem:source_keys})
    result_path = Path(RESULT, "dict_sources.json")
    with open(result_path, 'w', encoding='utf-8') as file:
        json.dump(sources_dict, file, ensure_ascii=False, indent = 4)
    return 0

def final_constructor(use_certain_delimeter = USE_CSV_DELIMITER):
    sources_paths = list(Path(RAW_DATA).iterdir())
    sources_dict_path = Path(RESULT, "dict_sources.json")
    with open(sources_dict_path, 'r', encoding='utf-8') as file:
        sources_dict = json.load(file)
    selected_tf_idf_dict_path = Path(RESULT, "selected_tf-idf_dict.pickle")
    with open(selected_tf_idf_dict_path, 'rb') as file:
        selected_tf_idf_dict = pickle.load(file)
    words_list = [0]*2 + list(selected_tf_idf_dict.keys())
    result_path = Path(RESULT, "formatted_result.csv")
    table_header = ["Группа", "Имя Файла", "Количество слов", "Количество предложений"]
    table_header.extend(words_list[2:])
    with open(result_path, 'w', newline='') as csvfile:
        if use_certain_delimeter:
            csvfile.write(CSV_DELIMITER_OPTION[1])
        writer = csv.writer(csvfile, delimiter= CSV_DELIMITER_OPTION[0])
        writer.writerow(table_header)
        group_counter = 0
        for sources_group in sources_paths:
            for text_file in Path(sources_group).glob("*.txt"):
                if Path(text_file).stem in sources_dict:
                    row_content = [Path(sources_group).name, Path(text_file).stem]
                    words_frequencies = [0] * (len(selected_tf_idf_dict.keys())+2)
                    source_words = sources_dict.get(Path(text_file).stem)
                    for word, value in source_words:
                        word_index = words_list.index(word)
                        words_frequencies[word_index] = value
                    row_content.extend(words_frequencies)
                    writer.writerow(row_content)
            group_counter+=1
```

```
    return 0

def file_word_frequency(word, file_dict):
    frequency = file_dict.get(word)/sum(file_dict.values())
    return frequency

def selected_word_dict(result_list, source_dict, number_for_extraction = LINE_NUMBER_FOR_EXTRACTION):
    selected_words = {}
    extracted_words = set()
    if (len(result_list[0]) > number_for_extraction):
        extract_count = number_for_extraction
    else:
        extract_count = len(result_list[0])
    for word_list in result_list:
        for i in range(extract_count):
            extracted_words.add(word_list[i][0])
    for word in list(extracted_words):
        selected_words.update({word:source_dict.get(word)})
    with open(Path(RESULT,"selected_tf-idf_dict.pickle"), 'wb',) as file:
            pickle.dump(dict(selected_words), file)
    return 0

def garbage_removal(to_remove = TO_REMOVE_PATHS):
    for path in to_remove:
        if (path):
            path.unlink()
    return 0

def main():
    start_time = time.time() #Старт расчета затраченного времени
    group_processing("1_RawData") #Сбор слов и их колличества из исходных текстов
    group_tf_dict() #Расчет межклассовых метрик для собранных слов
    result_to_csv()#Перевод json-файла в табличный вид
    dictionary_sources()#Генерация словаря, содержащего частоты выбранных слов относительно источников
    final_constructor()#Сборка данных в обрабатываемый формат
    if (GENERATE_READABLE_SOURCE_LIST):
        readable_dictionary_sources() #Формирование читаемого варианта источников слов с частотами
    garbage_removal()#Удаление ненужных файлов
    print("--- Program works for %s seconds ---" % (time.time() - start_time)) #Окончание расчета затраченного времени
    return 0

main()
```

ПРИЛОЖЕНИЕ В

Программный код классифицирующего методом деревьев решений алгоритма

```
import os
import glob
import shutil
import random
import pandas as pd
from collections import Counter
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
import subprocess
import string

class TextClassifier:
    def __init__(self, frequency_table_file, texts_to_classify_dir):
        self.frequency_table_file = frequency_table_file
        self.texts_to_classify_dir = texts_to_classify_dir
        self.valuable_words = []
        self.class_names = []
        self.class_word_frequencies = {}
        self.classifier = DecisionTreeClassifier()
        self.label_encoder = LabelEncoder()

    def load_frequency_table(self):
        df = pd.read_csv(self.frequency_table_file)
        self.valuable_words = df.columns[2:].tolist()
        self.class_names = df['Class_Name'].unique()

        X = df[self.valuable_words].values
        y = self.label_encoder.fit_transform(df['Class_Name'])

        self.classifier.fit(X, y)

    def read_file(self, filepath):
        try:
            with open(filepath, 'r', encoding='utf-8') as file:
                return file.read().lower()  # Convert text to lowercase
        except:
            with open(filepath, 'r', encoding='cp1251') as file:
                return file.read().lower()  # Convert text to lowercase

    def calculate_relative_frequencies(self, text):
        word_counts = Counter(text.split())
        total_words = sum(word_counts.values())
        return [word_counts.get(word, 0) / total_words for word in self.valuable_words]

    def classify_text(self, text):
        text_frequencies = self.calculate_relative_frequencies(text)
        text_frequencies = np.array(text_frequencies).reshape(1, -1)
        predicted_class_index = self.classifier.predict(text_frequencies)[0]
        return self.label_encoder.inverse_transform([predicted_class_index])[0]

    def classify_all_texts(self, text_filepaths, original_classes, attempt_number):
        self.load_frequency_table()
        classification_results = {}
        correct_classifications = 0

        for filepath, original_class in zip(text_filepaths, original_classes):
            text_name = os.path.basename(filepath)
            text = self.read_file(filepath).translate(str.maketrans('', '', string.punctuation))
            predicted_class = self.classify_text(text)
            classification_results[text_name] = (original_class, predicted_class)
            if original_class == predicted_class:
                correct_classifications += 1
```

```
        percent_correct = (correct_classifications / len(text_filepaths)) * 100

        with open('classification_results.txt', 'a', encoding='utf-8') as outfile:
            outfile.write(f"Attempt number {attempt_number}\n")
            for text_name, (original_class, predicted_class) in classification_results.items():
                outfile.write(f"{original_class} {text_name} - {predicted_class}\n")
            outfile.write(f"Percent of correct classification: {percent_correct:.2f}%\n\n")

class CycleRunner:
    def __init__(self, classes_dir, texts_to_classify_dir, frequency_script, num_cycles=20):
        self.classes_dir = classes_dir
        self.texts_to_classify_dir = texts_to_classify_dir
        self.frequency_script = frequency_script
        self.num_cycles = num_cycles
        self.texts_info = []

    def choose_random_texts(self):
        random.seed()
        selected_files = []
        original_classes = []
        num_texts_per_class = random.randrange(5,10)

        for class_dir in os.listdir(self.classes_dir):
            class_path = os.path.join(self.classes_dir, class_dir)
            if os.path.isdir(class_path):
                text_files = glob.glob(os.path.join(class_path, '*.txt'))
                selected = random.sample(text_files, num_texts_per_class)
                for file in selected:
                    shutil.move(file, self.texts_to_classify_dir)
                    selected_files.append(os.path.join(self.texts_to_classify_dir, os.path.basename(file)))
                    original_classes.append(class_dir)

        return selected_files, original_classes

    def move_files_back(self, text_filepaths, original_classes):
        for filepath, original_class in zip(text_filepaths, original_classes):
            class_folder = os.path.join(self.classes_dir, original_class)
            shutil.move(filepath, class_folder)

    def run_cycles(self):
        classifier = TextClassifier('Word_Frequencies.csv', self.texts_to_classify_dir)

        for attempt in range(1, self.num_cycles + 1):
            # Этап 1: Выбор случайных текстов из каталогов классов
            text_filepaths, original_classes = self.choose_random_texts()

            # Этап 2: Запуск алгоритма выделения признаков
            subprocess.run(['python', self.frequency_script])

            # Этап 3: Классификация текстов и запись резульата
            classifier.classify_all_texts(text_filepaths, original_classes, attempt)

            # Этап 4: Перемещение текстов обратно в свои классы
            self.move_files_back(text_filepaths, original_classes)

if __name__ == "__main__":
    classes_dir = 'Texts'
    texts_to_classify_dir = 'TextsToClassify'
    frequency_script = 'Freq_Gen.py'
    num_cycles = 20

    runner = CycleRunner(classes_dir, texts_to_classify_dir, frequency_script, num_cycles)
    runner.run_cycles()
    print(f"Все циклы классификации завершены")
```

ПРИЛОЖЕНИЕ Г

Программный код алгоритма извлечения с помощью TF-IDF и Хи-квадрат

```
import os
import glob
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import chi2
from joblib import Parallel, delayed
from collections import Counter

WORDS_EXTRACTION_COUNT=100

class ValuableWordsExtractor:
    def __init__(self, classified_texts_dir, num_words=WORDS_EXTRACTION_COUNT):
        self.classified_texts_dir = classified_texts_dir
        self.num_words = num_words
        self.texts = []
        self.labels = []
        self.class_names = []

    def load_texts_and_labels(self):
        def read_file(filepath):
            try:
                with open(filepath, 'r', encoding='utf-8') as file:
                    return file.read()
            except:
                with open(filepath, 'r', encoding='cp1251') as file:
                    return file.read()

        texts = []
        labels = []
        class_names = []
        for label in os.listdir(self.classified_texts_dir):
            label_dir = os.path.join(self.classified_texts_dir, label)
            if os.path.isdir(label_dir):
                class_names.append(label)
                filepaths = glob.glob(os.path.join(label_dir, '*.txt'))
                texts += Parallel(n_jobs=-1)(delayed(read_file)(filepath) for filepath in filepaths)
                labels += [label] * len(filepaths)

        self.texts, self.labels, self.class_names = texts, labels, class_names

    def extract_valuable_words(self):
        vectorizer = TfidfVectorizer(max_features=10000)
        X = vectorizer.fit_transform(self.texts)
        y = self.labels
        feature_names = vectorizer.get_feature_names_out()

        top_words = {class_name: [] for class_name in self.class_names}

        for class_name in self.class_names:
            class_mask = [1 if label == class_name else 0 for label in y]
            chi2_scores, _ = chi2(X, class_mask)
            top_indices = chi2_scores.argsort()[-self.num_words:][::-1]
            top_words[class_name] = [feature_names[i] for i in top_indices]

        return top_words

    def save_to_csv(self, top_words, output_file='Valuable_words.csv'):
        max_length = self.num_words
        data = {class_name: top_words[class_name] + [""] * (max_length - len(top_words[class_name])) for class_name in self.class_names}
        df = pd.DataFrame(data)
        df.to_csv(output_file, index=False)

class ValuableWordsFrequency:
```

```
    def __init__(self, classified_texts_dir, valuable_words_file, output_file='Word_Frequencies.csv'):
        self.classified_texts_dir = classified_texts_dir
        self.valuable_words_file = valuable_words_file
        self.output_file = output_file
        self.valuable_words = {}
        self.all_valuable_words = []
        self.results = []

    def load_valuable_words(self):
        df = pd.read_csv(self.valuable_words_file)
        self.valuable_words = {col: df[col].dropna().tolist() for col in df.columns}
        self.all_valuable_words = list(set(word for words in self.valuable_words.values() for word in words))

    def read_file(self, filepath):
        try:
            with open(filepath, 'r', encoding='utf-8') as file:
                return file.read().lower()  # Convert text to lowercase
        except:
            with open(filepath, 'r', encoding='cp1251') as file:
                return file.read().lower()  # Convert text to lowercase

    def calculate_relative_frequencies(self, word_counts):
        total_words = sum(word_counts.values())
        return {word: word_counts[word] / total_words for word in self.all_valuable_words}

    def process_class_texts(self, class_name):
        class_dir = os.path.join(self.classified_texts_dir, class_name)
        filepaths = glob.glob(os.path.join(class_dir, '*.txt'))
        class_results = Parallel(n_jobs=-1)(delayed(self.process_text)(class_name, filepath) for filepath in filepaths)
        self.results.extend(class_results)

    def process_text(self, class_name, filepath):
        text_name = os.path.basename(filepath)
        text = self.read_file(filepath)
        word_counts = Counter(text.split())
        relative_frequencies = self.calculate_relative_frequencies(word_counts)
        return [class_name, text_name] + [relative_frequencies.get(word, 0) for word in self.all_valuable_words]

    def generate_frequencies_table(self):
        self.load_valuable_words()
        for class_name in self.valuable_words.keys():
            self.process_class_texts(class_name)

        # Column headers for the DataFrame
        columns = ['Class_Name', 'Text_Name'] + self.all_valuable_words

        df = pd.DataFrame(self.results, columns=columns)
        df.to_csv(self.output_file, index=False)

if __name__ == "__main__":
    classified_texts_dir = 'Texts'
    valuable_words_file = 'Valuable_words.csv'

    extractor = ValuableWordsExtractor(classified_texts_dir)
    extractor.load_texts_and_labels()
    top_words = extractor.extract_valuable_words()
    extractor.save_to_csv(top_words, 'Valuable_words.csv')
    print(f"Наиболее значимые слова сохранены в таблицу 'Valuable_words.csv'.")
    frequency_generator = ValuableWordsFrequency(classified_texts_dir, valuable_words_file)
    frequency_generator.generate_frequencies_table()
    print(f"Частоты наиболее значимых слов сохранены в матрицу '{frequency_generator.output_file}'
```