МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра вычислительных технологий

Допустить к защите
Заведующий кафедрой
д-р техн. наук, профессор
_____ Ю.М. Вишняков
(подпись)
20 июня 2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

СРЕДСТВА ТОКЕНИЗАЦИИ НАУЧНО-ТЕХНИЧЕСКИХ ТЕКСТОВ ДЛЯ СЕМАНТИЧЕСКОЙ ИНТЕРПРЕТАЦИИ

Работу выполнил _____ В.И. Шиян
(подпись)

Направление подготовки 02.03.02 «Фундаментальная информатика и информационные технологии»

Направленность (профиль) «Вычислительные технологии»

Научный руководитель
д-р техн. наук, профессор _____ Ю.М. Вишняков
(подпись)

Нормоконтролер
ассистент _____ А.А. Климец
(подпись)

Краснодар
2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время каждый день человек сталкивается с большим количеством научно-технической информации, представленной в том числе в виде текста, которая требует обработки. Часто количество документов настолько велико, что тяжело внимательно изучить их за приемлемое время, поэтому вопрос времени стоит особенно остро.

Анализ содержания большого объема текста проблематичен с точки зрения затрачиваемого времени. Ясно, что чем меньше объем текста, тем меньше времени потребуется на его обработку. Отсюда вытекает задача выделения из текста некоторых структур, которые могли бы представить содержание текста в лаконичном виде с целью минимизации времени, затрачиваемого на обработку этого текста. Одной из таких структур является реферат – краткий текст, который выражает главную мысль исходного текста. Поэтому задача реферирования текста заключается в формировании лаконичного изложения его содержания.

Цель работы – разработка инструмента автоматического реферирования, использующего эффективный алгоритм.

Для достижения этой цели поставлены задачи:

1) провести анализ известных подходов к реферированию текста с использованием вычислительной техники;

2) разработать алгоритм разбиения текста на предложения;

3) разработать алгоритм разбиения предложений на токены;

4) разработать алгоритм составления реферата;

5) написать программу для автоматического реферирования, реализующую разработанные алгоритмы, и провести оценку ее эффективности.

1 Основные понятия компьютерной лингвистики

Компьютерная лингвистика существует уже десятки лет. Возникновение сети интернет и бурный рост доступной текстовой информации существенно ускорило формирование этой научной области. В рамках данной области предложено немало идей по автоматической обработке текстов на естественном языке, которые были воплощены в прикладных системах. В сфере автоматической обработки текстов постоянно возникают новые задачи, требующие решения. Интернет-сайт международной ассоциации компьютерной лингвистики поможет получить представление о достижениях области. Здесь собрано множество работ в данной области. Интернет-сайт представлен на рисунке 1.

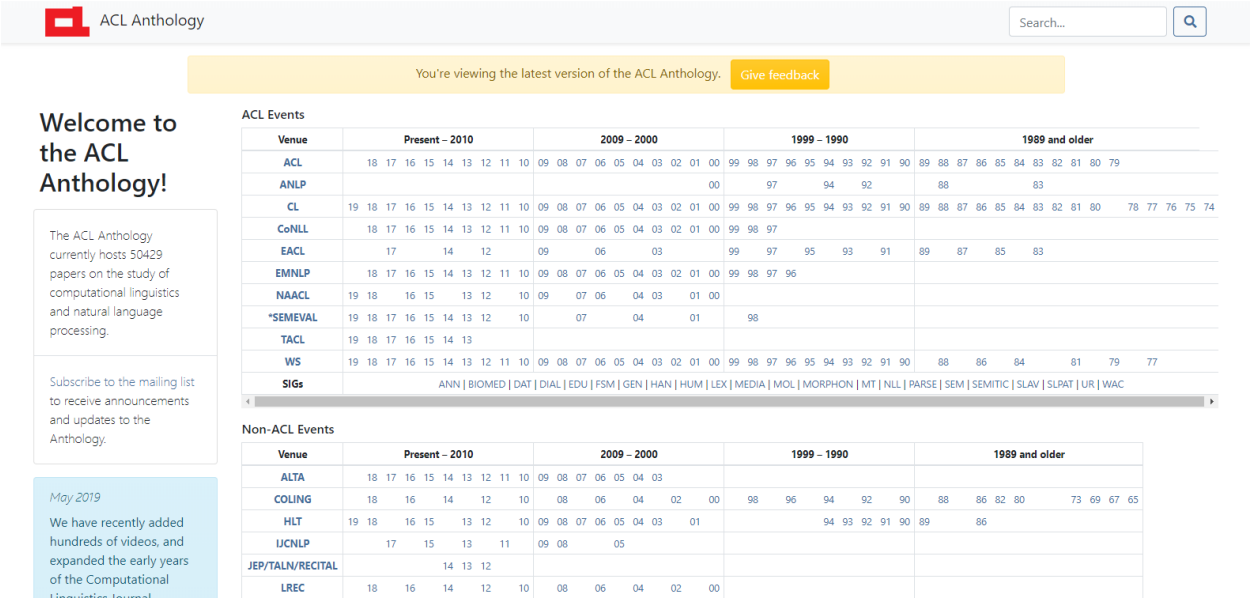

Рисунок 1 – Интернет-сайт международной ассоциации компьютерной лингвистики

Компьютерная лингвистика – междисциплинарная область, которая охватывает науки, которые представлены на рисунке 2. Для решения задач она использует методы и подходы, разработанные в рамках этих наук.

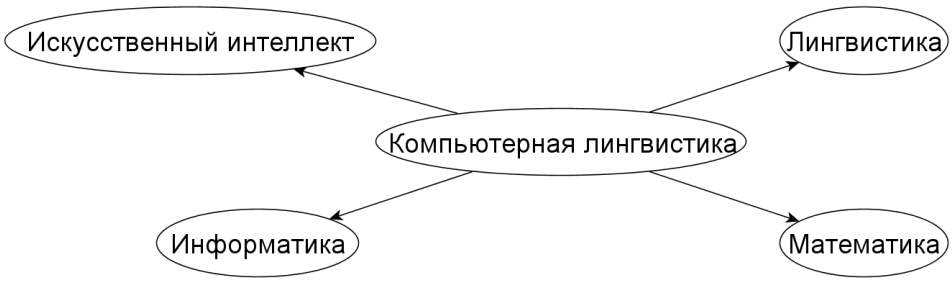

Рисунок 2 – Науки, которые охватывает компьютерная лингвистика

Начало развития науки было положено американским языковедом Н. Хомским, который написал работы по формализации естественного языка, с первых экспериментов в области машинного перевода, произведенных математиками и программистами, а еще с началом разработки программ, направленных на осмысление текста на естественном языке, с использованием технологий искусственного интеллекта.

Так как в компьютерной лингвистике объектом обработки выступают тексты естественного языка, ее формирование невозможно в отсутствии базовых знаний в области общей лингвистики. Лингвистика включает области, которые представлены на рисунке 3.

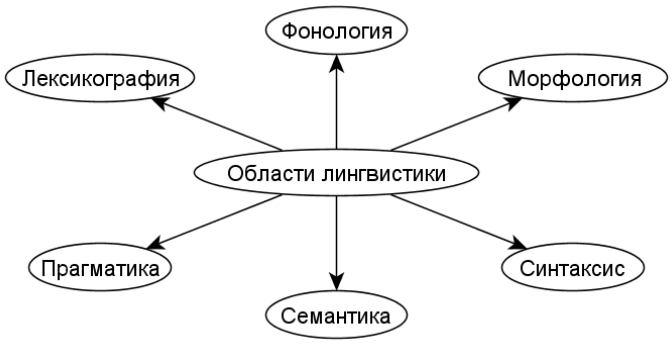

Рисунок 3 – Области лингвистики

Компьютерная лингвистика не может существовать без искусственного интеллекта [1], поскольку именно в этой области ведется разработка специальных интеллектуальных функций. Искусственный интеллект не охватывает всю компьютерную лингвистику, хотя исследования в этих областях имеют много общего, а именно – использование преимущественно

компьютерного моделирования для решения задач, конечная цель исследований, использование эвристических методов.

Задача компьютерной лингвистики – разработка методов и средств построения лингвистических процессоров для решения задач в области автоматической обработки текстов, написанных на естественном языке. При этом создается модель языка, основанная на формальном описании свойств этого языка.

1.1 Задачи компьютерной лингвистики

На практике компьютерная лингвистика применима во многих областях. На рисунке 4 представлены задачи, которые решаются компьютерной лингвистикой.

Рисунок 4 – Задачи, которые решаются компьютерной лингвистикой

Одной из ее задач выступает машинный перевод [2]. Это было одна из первых задач, которая решалась в рамках компьютерной лингвистики.

решения задачи машинного перевода появились в середине прошлого века и реализовывали простейший механизм пословного перевода. Но достаточно скоро стало ясно, что для машинного перевода потребуется больше свойств обрабатываемого текста.

К настоящему времени спроектировано множество систем машинного перевода, как большие международные проекты, так и небольшие платные переводчики. Интересен подход с использованием промежуточного языка, на котором кодируется значение обрабатываемого текста. Такой подход используется в приложениях многоязыкового перевода. Другой подход – статистический. Он основан на использовании статистики переводных пар слов и словосочетаний.

Задача машинного перевода решается уже десятки лет, однако качество получаемого перевода все еще неидеально. Использование нейронных сетей и машинного обучения помогло добиться некоторого прорыва в данной области и улучшить качество машинного перевода.

Информационный поиск [3] также помогает при решении задач индексирования, реферирования, классификации и рубрицирования текстов.

Поиск необходимых документов в огромных базах текстов требует их индексирования. При индексировании производится предобработка текста и выстраивается индексная структура. Одной из самых известных и использующихся подходов – векторная модель. При ее использовании каждому тексту присваивается вектор слов, запросу пользователя также присваивается такой вектор. Релевантность документа определяется по схожести полученных векторов. Такую модель используют, к примеру, современные поисковики. При это индексирование проводится по употребляемым в тексте словам, а затем подходящие документы выдаются в некотором порядке, определяемым алгоритмом ранжирования. Также в области информационного поиска важным является вопрос о многоязыковой поиске.

Под реферированием текста понимают сокращение его размера и получение лаконичного изложения содержащейся в тексте информации – реферата. Реферат позволяет ускорить поиск в наборах документов. Реферат может быть составлен как для одного, так и для нескольких текстов, объединенных общей тематикой. Наиболее распространенной стратегией автоматического реферирования является отбор наиболее важных предложений. Отбор таких предложений производится с использованием статистики по словам и словосочетаниям, а также на основе анализа лингвистических и структурных особенностей текстов.

Задача аннотирования текста напоминает задачу реферирования, однако аннотация, в отличие от реферата, представляет собой не краткую выжимку текста, а список ключевых тем.

Задачи классификации и кластеризации [4] схожи между собой. Их отличие состоит в том, что при проведении классификации текст должен быть отнесён к одному из заранее известных классов с заранее определёнными характеристиками. Кластеризация же заключается в разбиении множества текстов на кластеры – группы документов, близких по некоторым характеристикам. Задачи можно отнести к области Text Mining, которая, в свою очередь, является подобластью направления Data Mining [5].

На практике задача классификации применима, к примеру, для распознавании спама или классификации сообщений.

Задача рубрицирования текста похожа на задачу классификации. Она заключается в отнесении текста к одной из тематических рубрик. Как правило, рубрики образуют иерархическое дерево.

Еще одна важная задача обработки естественного языка – извлечение информации из текстов [6]. Эту задачу относят к направлению Text Mining. Решение подобных задач полезно для финансовой и производственной аналитики. Извлечение информации из текста осуществляется путем выделения из текста конкретных именованных сущностей и связей между ними. Как правило, это реализуется с использованием частичного

синтаксического анализа текста, позволяющего обработать большие коллекции текстов, например, новостей. После извлечения данные структурируются и визуализируются.

Относительно новая задача – формирование ответа на вопросы [7]. Пользователь задаёт системе вопрос на естественном языке, а система должна в своей базе текстов найти ответ на него и выдать его пользователю.

Выделение мнений и анализ тональности текстов – две актуальные задачи, которые также относят к направлению Text Mining [8]. Задача выделения мнений заключается в анализе мнений пользователей о товарах и других объектах в интернет-магазинах, блогах, форумах. Задача анализа тональности подразумевает оценку тональности текста в целом или отдельных его фрагментов.

Более 50 лет назад появилась задача, связанная с развитием сети интернет – поддержка диалога на естественном языке. Первые решения этой задачи реализовывались в рамках отдельной информационной системы, например, для обработки запросов к базе данных. При этом запросы должны были строиться по определенным, достаточно строгим правилам, что позволяло упрощать анализ вопросов, а ответы на них выстраивать согласно некоторому шаблону.

В настоящее время большое распространение получили чат-боты – программы, способные поддерживать разговор на некоторую тему. Одна из первых известных разработок в этой области – система ELIZA. Сейчас появились системы, способные пройти известный тест Тьюринга.

Совсем другое направление в области обработки естественного языка – автоматизация подготовки и редактирования текстов на естественном языке. Развитие этого направления началось с разработки программ, осуществляющих автоматическое определение переносов слов и программ проверки текста на орфографию. В платных системах помимо проверки орфографии реализована проверка некоторых синтаксических ошибок, к примеру, ошибок согласования слов.

В то же время в автокорректорах пока не реализовано распознавание Обнаружение более сложных ошибок, например, некорректное употребление предлогов, опечатки, наверное использование паронимов или похожих слов пока не реализовано в полной мере, однако в компьютерной лингвистике ведутся исследования, направленные на выявление таких ошибок с помощью статистики словоупотребления и словосочетаний [9].

Еще одна важная задача – разработка систем обучения естественному языку. Такие системы создаются для помощи пользователю в изучении отдельных разделов языка, к примеру, морфологии, лексики, синтаксиса.

В рамках этого направления также создаются компьютерные словари, не имеющие печатных аналогов. К таким словарям, например, относят словарь сочетаемости КроссЛексика [10], в котором дополнительно содержится информация о синонимах, антонимах, семантических связях слов.

Другое важное направление связано не с анализом, а синтезом текста на естественном языке [11]. Она используется при осуществлении автоматического перевода, в том числе многоязыкового.

Бурно развивающееся направление – распознавание и синтез звучащей речи. Ошибки распознавания неизбежны, они корректируются системой на основе информации словарей и морфологических моделей. В некоторых случаях применятся машинное обучение.

1.2 Сложности моделирования естественного языка

Естественный язык – большая открытая многоуровневая система знаков. Она появилась в результате деятельности человека и постоянно меняется в процессе данной деятельности . Поэтому моделирование языка – сложная задача.

Для того, чтобы анализировать текст, его разбивают на отдельные единицы, которые относятся к определенным уровням. Эти уровни представлены на рисунке 5.

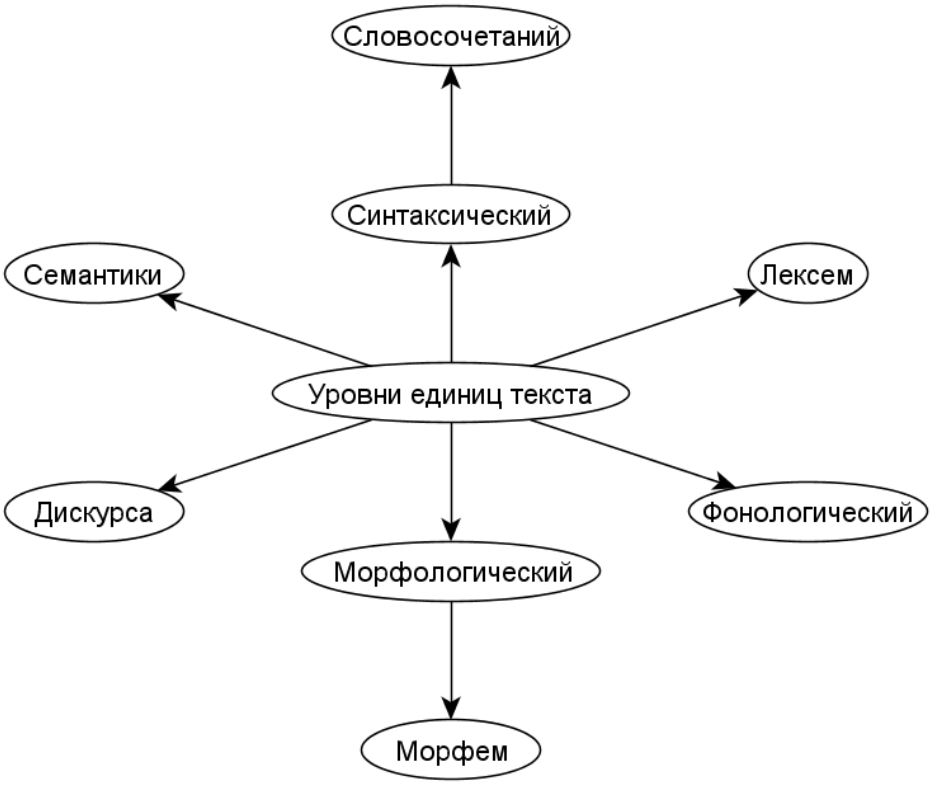

Рисунок 5 – Уровни единиц текста

Для языков, использующих алфавитный способ записи слов, фонологический уровень соответствует уровню символов.

Подсистемы естественного языка называются уровнями. В некоторых из этих уровней также могут быть выделены свои подуровни. Например, в морфологическом уровне может быть отдельно выделен уровень морфем – минимально значащих частей слов таких, как корень, приставка, суффикс, окончание, постфикс.

Иногда выделяют уровень лексем. Лексема – это слово языка, она включает все свои словоформы, которые встречаются в тексте. Начальная форма лексемы – лемма. Например, для глагола леммой будет его инфинитив.

В синтаксическом уровне выделяют уровень словосочетаний. Словосочетание – группа слов, связанных по смыслу и грамматически. В то же время синтаксический уровень является подуровнем сложного синтаксического целого. Сложное синтаксическое целое – последовательность предложений, связанных по смыслу и с помощью

лексико-грамматических средств таких, как лексические повторы, местоимения. Сложному синтаксическому целому примерно соответствует абзац текста.

Единицы более высокого уровня можно разложить на единицы более низкого. К примеру, сложное синтаксическое целое можно разложить на предложения, а их, в свою очередь – на словосочетания.

Можно выделить уровень дискурса, который характеризует текст как совокупность связных предложений, написанных с определённой целью и выполняющих определённую задачу. Некоторые тексты обладают одинаковой структурой, например, технические описания.

Особым уровнем при моделировании естественного языка является уровень семантики. Семантика текста – один из наименее изученных аспектов при моделировании естественного языка. Семантика трудно формализуема.

Кроме многоуровневости системы естественного языка сложность его моделирования связана с постоянно происходящими в нем изменениями. Изменения касаются не только словарного запаса языка, но и синтаксиса, морфологии и фонетики. Поэтому невозможно единожды разработать формальную модель естественного языка и спроектировать соответствующий лингвистический процессор. Требуется постоянное пополнение знаний о языке на всех его уровнях и коррекция модели.

Еще одной сложностью является то, что некоторые единицы языка многозначны. Многозначность может быть на разных уровнях. На рисунке 6 представлены разновидности многозначности.

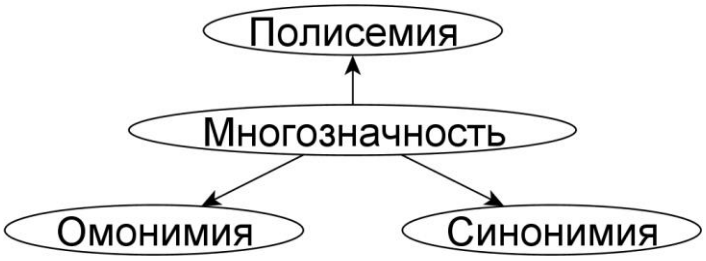

Рисунок 6 – Разновидности многозначности

Полисемия – явление, когда у одной единицы языка есть несколько связанных между собой значений. На рисунке 7 представлен пример полисемии слова.

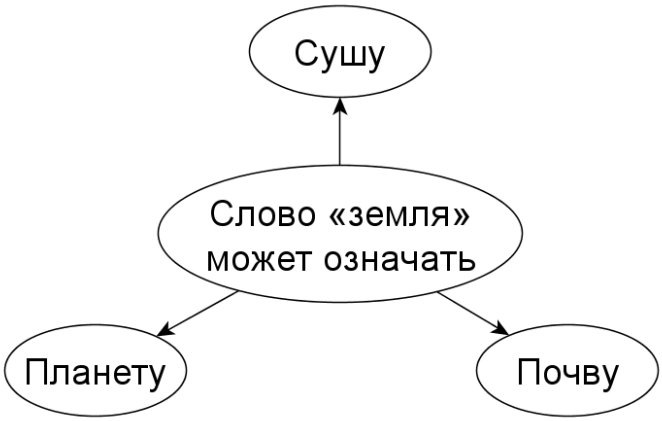

Рисунок 7 – Пример полисемии слова

Синонимия – частичное или полное совпадение значений разных единиц, к примеру, синонимия слов: подлец и негодяй, синонимия приставок пере- и пре-.

Омонимия – совпадение по форме двух разных по смыслу единиц. В таблице 1 представлены виды омонимии.

Таблица 1 – Виды омонимии

Вид омонимии	Что означает	Пример
Лексическая	Одинаково звучащие и пишущиеся слова, но обладающие разным смыслом	Лук – овощ и лук – оружие
Морфологическая	Совпадение разных форм одного и того же слова	Совпадение формы именительного и винительного падежа у некоторых существительных

Продолжение таблицы 1

Лексико-морфологическая	Совпадение словоформ двух разных лексем	Слово «стали» может быть словоформой слова «сталь» или слова «стать»
Синтаксическая	Неоднозначность синтаксической структуры, что может приводить к различным интерпретациям одного и того же предложения	В предложении «Я встретил кота Игоря» слово «Игоря» может означать кличку кота, а может означать хозяина кота

1.3 Этапы обработки текста на естественном языке

Естественный язык трудно формализуем, поэтому разумно разить процесс его обработки на отдельные этапы, каждый из которых соответствует одному уровню языка. Поэтому большая часть передовых лингвистических процессоров выполняет определенные этапы обработки текста, которые представлены на рисунке 8.

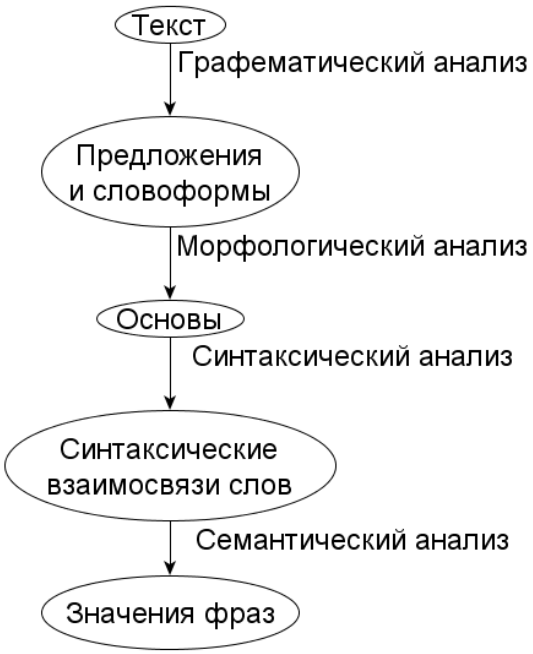

Рисунок 8 – Этапы обработки текста

Лингвистический процессор поэтапно транслирует каждое предложение текста в некоторую внутреннюю структуру, содержащую информацию о семантике этого предложения, и наоборот – по заданному смыслу генерирует соответствующее предложение.

Модули лингвистического процессора могут по-разному объединяться и взаимодействовать, но, как правило, отдельные уровни языковой модели такие, как морфология, синтаксис, семантика, обрабатываются разными модулями. Более того, решение некоторых задач подразумевает наличие представления в процессоре всех этапов обработки.

На рисунке 9 представлены отличия модулей морфологического анализа.

Рисунок 9 – Отличия модулей морфологического анализа

Этап семантического анализа текста менее проработан в рамках компьютерной лингвистики.

На рисунке 10 представлены формализмы, использующиеся для представления семантики всего текста.

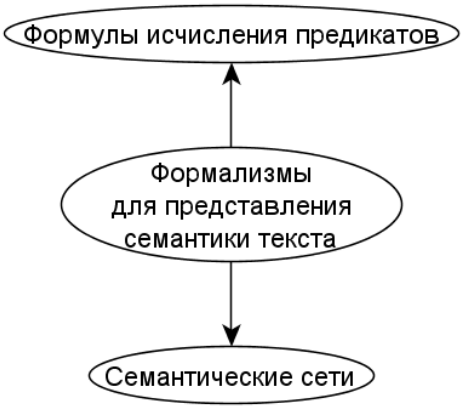

Рисунок 10 – Формализмы для представления семантики всего текста

Модели естественного языка, которые используются в компьютерной лингвистике, традиционно строятся с учетом лингвистических теорий и моделей. На рисунке 11 представлены особенности моделей компьютерной лингвистики.

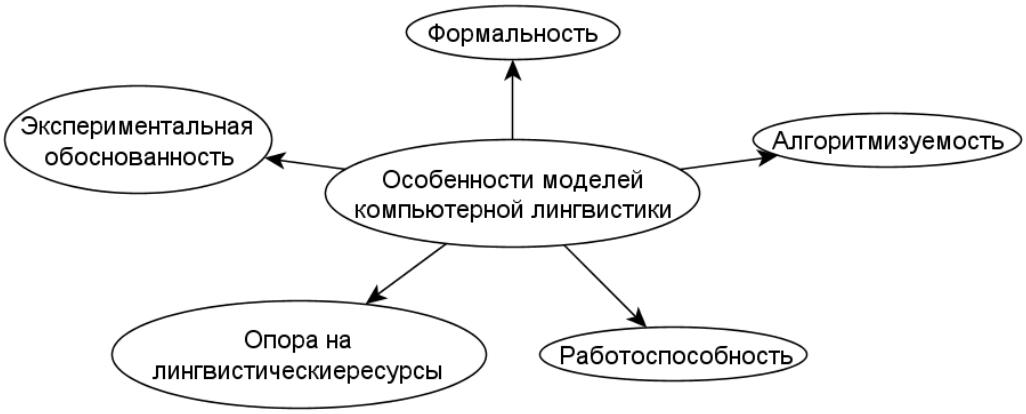

Рисунок 11 – Особенности моделей компьютерной лингвистики

1.4 Подходы к построению модулей компьютерной лингвистики

В настоящее время для создания модулей лингвистических процессоров применяется два основных подхода: основанный на правилах и основанный на машинном обучении.

Первым появился подход, который основан на правилах. Он заключается в описании модели языка в виде набора формальных правил. В ранних системах правила записывались прямо в код программы, но в настоящее время для записи правил обычно используют готовый формальный язык. Иногда такой язык создают специально для разрабатываемого приложения. Правила, описывающие модель языка, создаются специалистами в области лингвистики или в проблемной области обрабатываемых текстов.

Другой подход – машинное обучение. Для составления языковой модели вместо правил используется специально отобранный набор размеченных текстов, на которых проводится обучение модели. Выделяют методы обучения с учителем, без учителя, частичное обучение с учителем.

Оба подхода – с использованием правил и с использованием машинного обучения – имеют свои достоинства и недостатки.

Создание правил трудоемко, требует ручного труда специалиста, как правило, лингвиста. К тому же часто невозможно сразу заранее предусмотреть все возможные случаи. С другой стороны, правила легко понимаемы, их легко модифицировать, добавлять новые, производить отладку языковой модели для обнаружения и корректировки ошибок.

Напротив, при использовании машинного обучения для обучения модели не требуется ручной труд, но здесь важно правильно выбрать методы обучения и корпус размеченных текстов, на котором будет производиться обучение. Не всегда возможно найти подходящий корпус, а создание собственного корпуса требует трудоемкого ручного труда. К тому же полученная в результате обучения модель непрозрачна для понимания, ее трудно отлаживать и модифицировать.

2 Проблема автоматического реферирования текста

2.1 Области применения задачи автоматического реферирования текста

В настоящее время каждый день человек сталкивается с большим количеством научно-технической информации, представленной в том числе в виде текста, которая требует обработки. Часто количество документов настолько велико, что тяжело внимательно изучить их за приемлемое время, поэтому вопрос времени стоит особенно остро.

Анализ содержания большого объема текста проблематичен с точки зрения затрачиваемого времени. Ясно, что чем меньше объем текста, тем меньше времени потребуется на его обработку. Отсюда вытекает задача выделения из текста некоторых структур, которые могли бы представить содержание текста в лаконичном виде с целью минимизации времени, затрачиваемого на обработку этого текста. Одной из таких структур является реферат – краткий текст, который выражает главную мысль исходного текста. Поэтому задача реферирования текста заключается в формировании лаконичного изложения его содержания.

Отклассифицированная структура документа позволяет системе ускорить поиск в ней и улучшить его качество. Также реферат помогает пользователю быстрее определить, соответствует ли документ требуемому запросу, а также узнать основное содержание текста. На рисунке 12 представлены области, в которых ощущается потребность в автоматическом реферировании.

Рисунок 12 – Области, в которых ощущается потребность в автоматическом реферировании

Так, например, в поисковых системах необходимо создание короткого текста при показе ответов на запросы пользователя. На рисунке 13 представлен пример таких ответов.

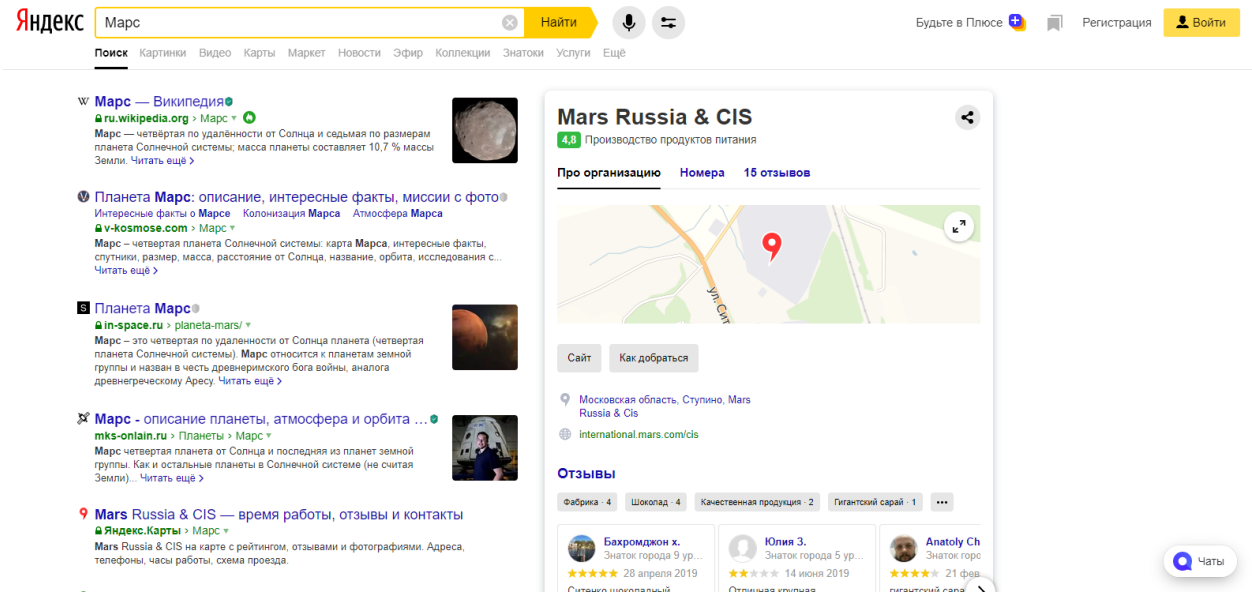

Рисунок 13 – Короткие тексты при показе ответов на запросы пользователя

Вывод текста в формате, удобном для пользователя необходим, например, при чтении новостей. На рисунке 14 представлен пример такого вывода.

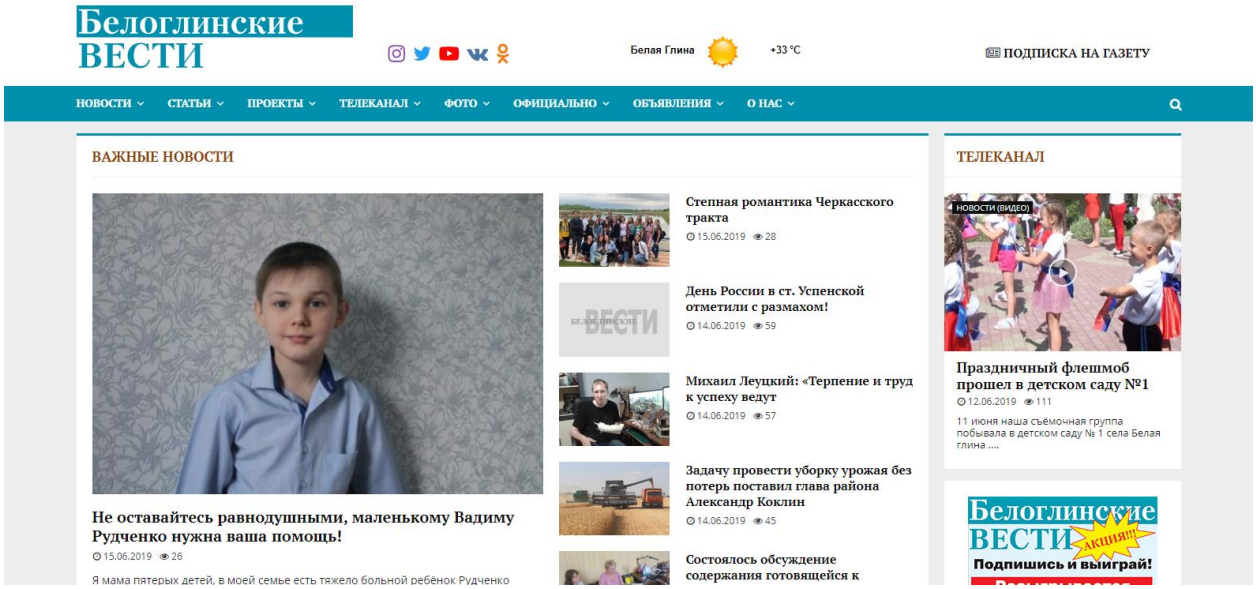

Рисунок 14 – Вывод текста в формате, удобном для пользователя при прочтении новостей

Это также необходимо, например, при выборе товара в интернет-магазине. На рисунке 15 представлен пример такого вывода.

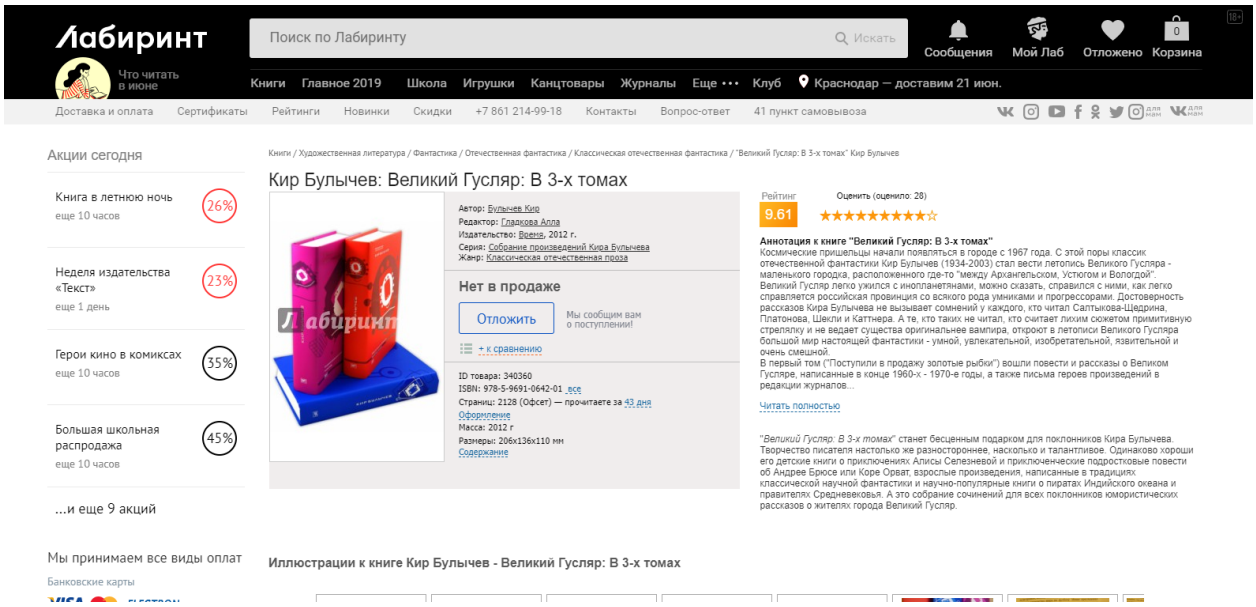

Рисунок 15 – Вывод текста в формате, удобном для пользователя при выборе товара в интернет-магазине

Задача реферирования текста заключается в формировании лаконичного изложения его содержания.

Составление реферата с помощью компьютера предполагает генерацию лаконичного изложения текста, что требует построения алгоритма, которые на вход получает текст на естественном языке, а на выходе возвращает реферат этого текста. Данные тексты характеризуются определенными характеристиками.

Характеристики входного текста представлены в таблице 2.

Таблица 2 – Характеристики входного текста

Характеристики входного текста	Возможные значения
Количество документов	1 Один документ 2 Множество документов
Особенность	1 Определенная предметная область 2 Общего назначения
Язык	1 Одноязыковой 2 Многоязыковой

Характеристики входного текста представлены в таблице 3.

Таблица 3 – Характеристики выходного текста

Параметры выходного текста (реферата)	Возможные значения
Словообразование	1 Экстракция 2 Абстракция
Логичность, последовательность	1 Есть 2 Нет
Информативность	Количественная характеристика
Степень сжатия	Количественная характеристика
Тип реферата	1 Общий 2 Для конкретных пользователей 3 На базе заданного вопроса

Продолжение таблицы 3

Параметры выходного текста (реферата)	Возможные значения
Стиль	1 Указывающая 2 Критика 3 Оценка 4 Информативная
Пользователь	1 Новичок 2 Эксперт
Жанр	1 Технические отчеты 2 Научные отчеты 3 Новости 4 Электронные сообщения 5 Книги
Форма	1 Таблица 2 Текст 3 Структура заголовков 4 Список

Проблема автоматической обработки текстов появилась достаточно давно, почти сразу после появления вычислительной техники. И хотя исследования в области искусственного интеллекта и смежных областей ведутся уже полвека, удовлетворительного решения большинства прикладных задач так и не было найдено.

Одной из важнейших задач является разработка оптимальных способов автоматического сжатия документов. Под сжатием понимают последовательность действий, направленных на получение лаконичного вида текста, сохраняя при этом как можно больше смысла.

Методы общего назначения, работающие с широким классом текстов, не могут показать хорошие результаты реферирования. Поэтому тексты

разбиваются на более мелкие классы, внутри которых тексты обладают похожими свойствами. Функциональный стиль текста считается одним из эффективных параметров классификации текстов русского языка в задаче автоматического реферирования.

В каждой сфере общественной жизни используется некоторый подкласс русского литературного языка, который имеет свой набор отличительных черт на каждом из языковых уровней. Такие подклассы называются функциональными стилями. На рисунке 16 представлены функциональные стили, которые выделяют в современном русском языке:

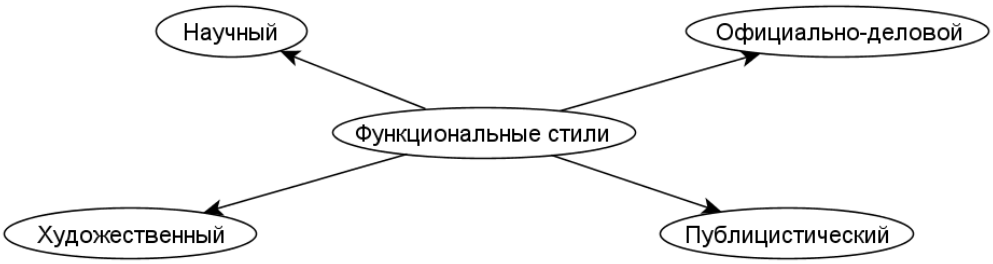

Рисунок 16 – Функциональные стили, которые выделяют в современном русском языке

Тексты, которые относятся к одному стилю, схожи между собой по некоторым характеристикам, что позволяет лучше оценивать информативность слов и фрагментов текста.

Так, для официально-делового стиля характерны четкие формулировки, однозначное толкование, регулятивный и предписывающий характер речи. Нередко встречаются параллельные синтаксические конструкции, которые оформлены в виде нумерованного списка. Такие структуры обычно можно сократить без нарушения синтаксической правильности предложения.

Для текстов публицистического стиля характерны социально-оценочный и информационный характер речи, использование большого количества выразительных языковых средств, использование устаревших слов, слов в переносном значении. Как правило, автор текста стремится сделать текст более образным и эмоционально-насыщенным.

Наиболее информационно значимыми частями речи в текстах этого стиля являются глагол и имя существительное, что соответствует классическому для русского языка распределению информационной нагрузки.

Художественный стиль включает различные по размеру, составу, форме, теме и жанру тексты.

На рисунке 17 представлены особенности научного стиля.

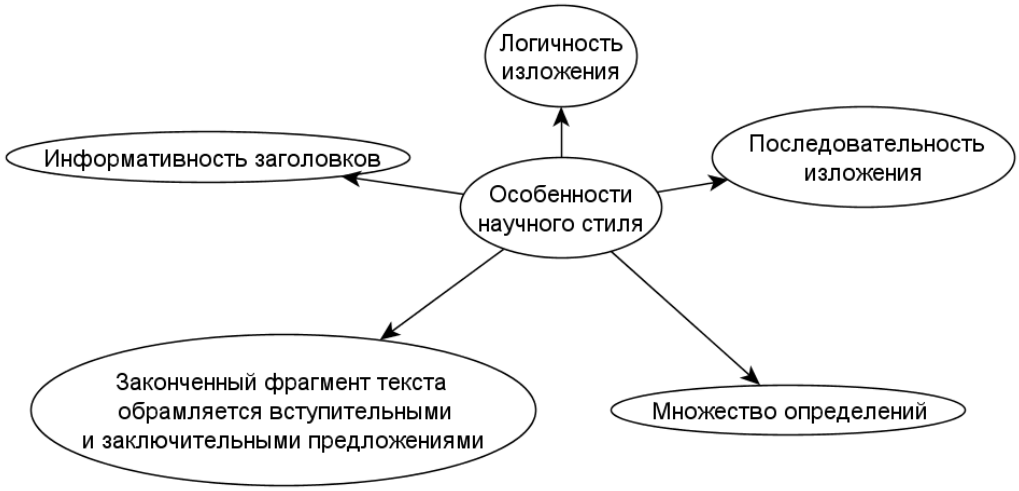

Рисунок 17 – Особенности научного стиля

К отличительным особенностям научно-технического текста можно отнести его стандартизированность, большое количество специальных терминов, формально-логический способ изложения содержания. Как правило, научно-технический текст представляет собой рассуждение, которое должно обосновать и описать научное исследование. При этом шаги такого рассуждения автор указывает, используя шаблонные слова и выражения, называемые речевыми маркерами. Маркеры помечают дискурснвные операции и относятся к дискурсивному уровню текста [27]. К таким операциям относятся обоснование вывода, выдвижение гипотезы, введение термина и понятия, приведение фактов и доказательств, подведение итогов. Такие шаги часто помечаются общенаучными словами и выражениями.

На рисунке 18 представлены средства организации научного текста.

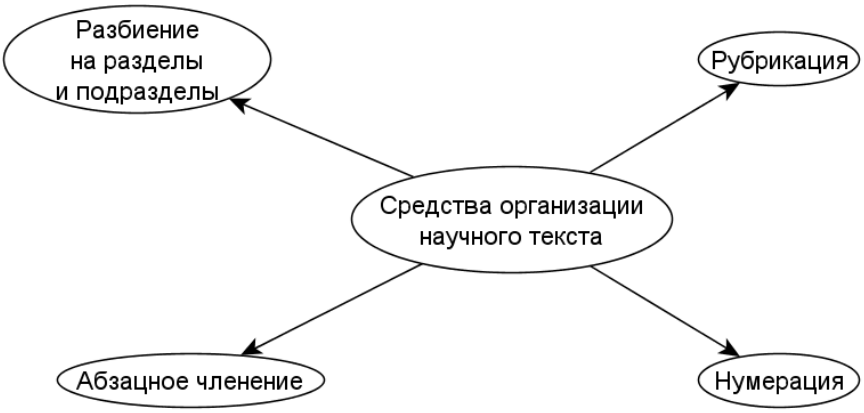

Рисунок 18 – Средства организации научного текста

Дальше рассмотрены основные методы автоматического реферирования текста.

2.2 Методы автоматического реферирования текста

ЭВМ стали применяться для решения интеллектуальных задач практически сразу после своего появления.

На рисунке 19 представлены два основных подхода к реферированию:

Рисунок 19 – Основные подходы к реферированию

При использовании подхода экстракции главная трудность – поиск самых важных предложений в тексте и объединение их в единый связный текст.

На рисунке 20 представлены этапы генерации реферата при использовании подхода абстракции.

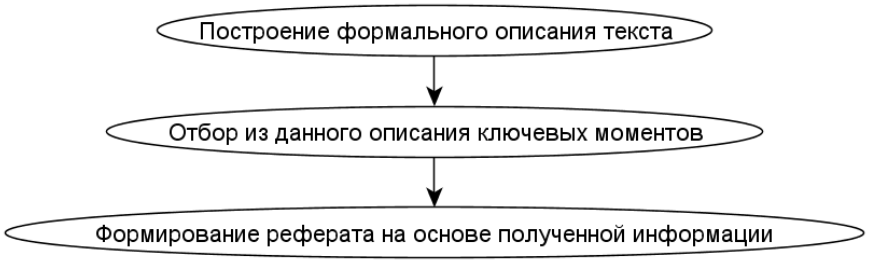

Рисунок 20 – Этапы генерации реферата при использовании подхода абстракции

В подавляющем большинстве случаев при практической реализации используются только статистические методы.

Яцко В.А. предложил метод симметричного реферирования [13], который заключается в том, что вес предложения определяется количеством связей с предыдущим и последующим предложениями.

На рисунке 21 представлены основные методы реферирования на основе экстракции.

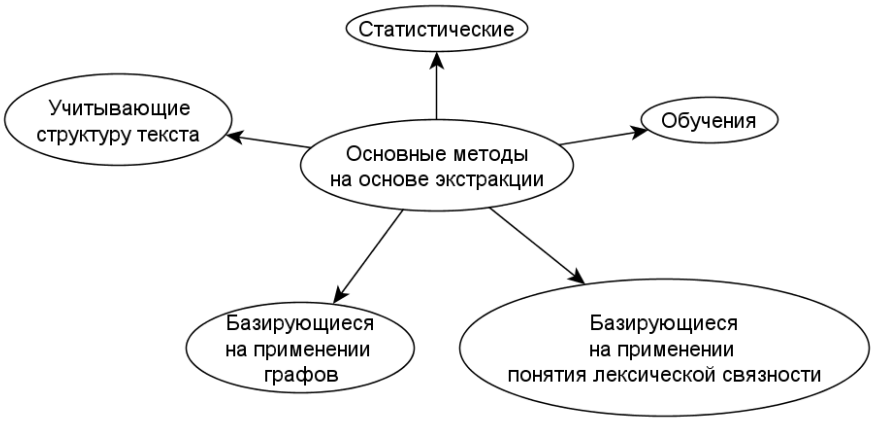

Рисунок 21 – Основные методы реферирования на основе экстракции

Структура системы реферирования без опоры на знания представлена на рисунке 22.

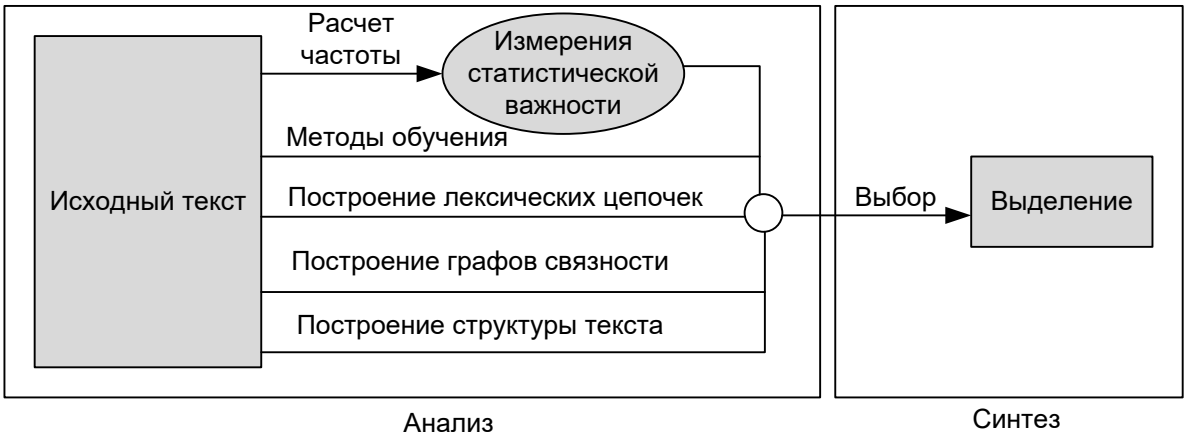

Рисунок 22 – Структура системы реферирования без опоры на знания

Подавляющее большинство современных систем реферирования используют подход экстракции, хотя в последние годы появились достойные внимания работы, которые развивают второе направление [14, 15].

Более сложные алгоритмы используются при реферировании путем абстракции. При этом реферат представляет собой не просто подмножество предложений исходного текста, а новый текст, который содержательно обобщает первичные документы.

При использовании этого подхода для подготовки реферата системе требуется значительная вычислительная мощность. Также необходимо наличие грамматик и словарей, чтобы система могла проводить синтаксический разбор и синтезировать предложения на естественном языке. Более того, для принятия решения о том, какая информация наиболее важна и как она должна быть представлена в реферате согласно здравому смыслу, необходимо использование онтологических справочников.

Подход на основе абстракции основан на понимании семантики естественного языка и использует искусственный интеллект.

Структура системы для реферирования на основе абстракции представлена на рисунке 23.

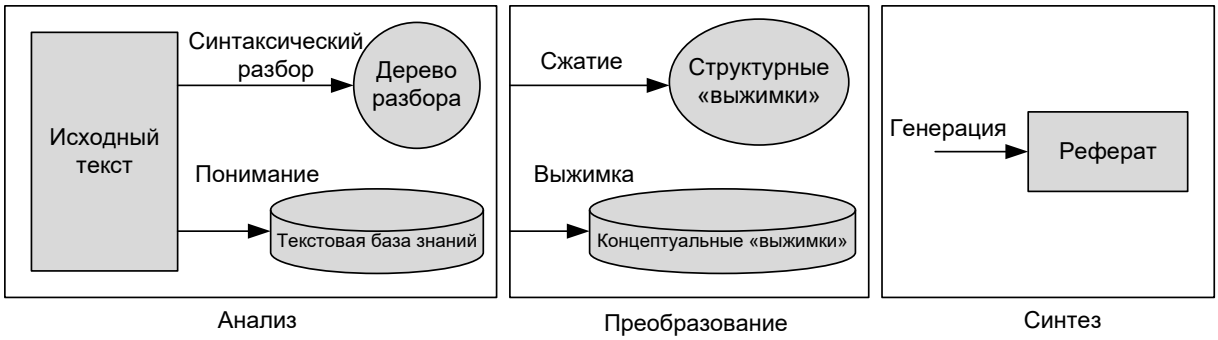

Рисунок 23 – Структура системы реферирования на основе абстракции

В процессе преобразования представление испытывает несколько изменений. Лишняя информация удаляется. Это может осуществляться с помощью ликвидации некоторых концептуальных подграфов либо удалением поверхностных суждений. После этого путем слияния графов или обобщения преобразованная информация еще больше агрегируется. В результате всех этих преобразований по существу, создаются концептуальные «выжимки» из текста.

Оба подхода реализуют схожий метод синтеза, а именно: модуль генерации из структурного или концептуального представления информации получает конструкцию на естественном языке. Некоторые системы дают возможность пользователю давать указания системе и влиять на получаемые «выжимки», при этом этапа генерации как такового не предусмотрено, однако исходные тексты должны быть предоставлены вместе с их кратким содержанием.

Системе заранее указывается, какая структура наиболее характерна.

Методы экстракции проще в реализации и позволяют работать с большими объемами текстов, однако конечный результат – реферат – может быть несвязным.

Методы абстракции выдают наиболее сложные рефераты, часто содержащие информацию, которая дополняет исходный текст. Такие методы, которые подразумевают опору на знания, как правило, требуют полноценных источников знаний, исчерпывающих словарей.

Все это является препятствием для широкого распространения данных методов.

2.3 Готовые программные решения

В настоящее время почти все лидеры рынка разработки программного обеспечения создают собственные системы автоматической обработки текста на естественном языке, в том числе и в области реферирования. В таких системах, как правило, используются разные математические и лингвистические методы работы с текстом. Они оснащены хорошим интерфейсом и позволяют легко манипулировать данными и визуализировать их. Такие продукты, как правило, разработаны согласно клиент-серверной архитектуре, обеспечивая доступ к различным данным.

В качестве примеров таких систем можно привести следующие:

На рисунке 24 представлены современные системы автоматического реферирования:

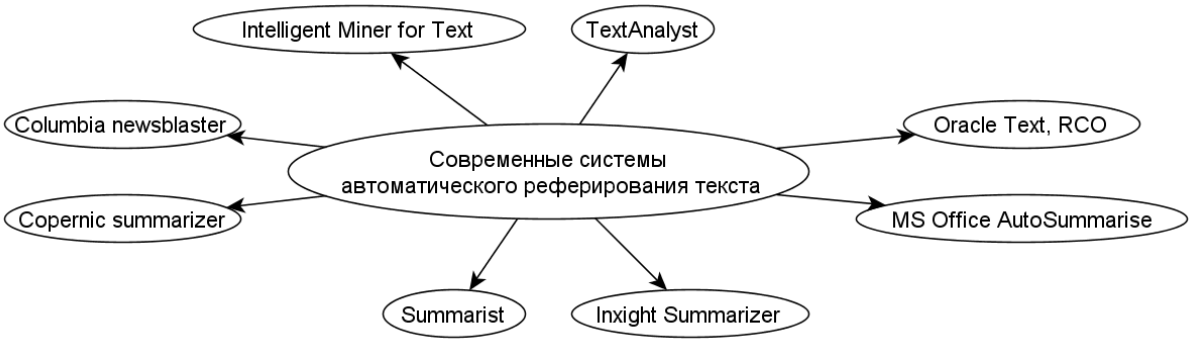

Рисунок 24 – Современные системы автоматического реферирования

Традиционные системы реферирования, использующие статистические методы реферирования или слова-маркеры, не слишком распространены на рынке программного обеспечения. Одна из наиболее узнаваемых коммерческих систем – Inxight Summarizer. Была создана в Исследовательском центре Ксерокса в Пало-Альто.

Система является успешной, поскольку:

1) реализует один из наиболее совершенных методов определения качества получаемого реферата;

2) использует несколько наиболее применяемых алгоритмов автоматического реферирования параллельно;

3) алгоритмы реферирования и оценки качества непосредственно связаны между собой;

4) распространение программного обеспечения в виде отдельных модулей реферирования, к примеру, динамических библиотек для платформ Win32 и Solaris.

Еще одной заслуживающей внимания платной системой является система Prosum, которая разработана British Telecommunications Laboratories как часть платной платформы TranSend, представляющей встроенный в веб страницу скрипт.

Система Extractor использует выделение именных групп.

Фирма IBM Intelligent Miner for Text разработала программный продукт, состоящий из отдельных утилит. Одна из таких утилит – Annotation Tool, которая позволяет выявить смысл текста и составить с его помощью реферат.

Система TextAnalyst разработана российской компанией Мегапьютер Интеллидженс. На рисунке 25 представлены задачи, которые решает TextAnalyst.

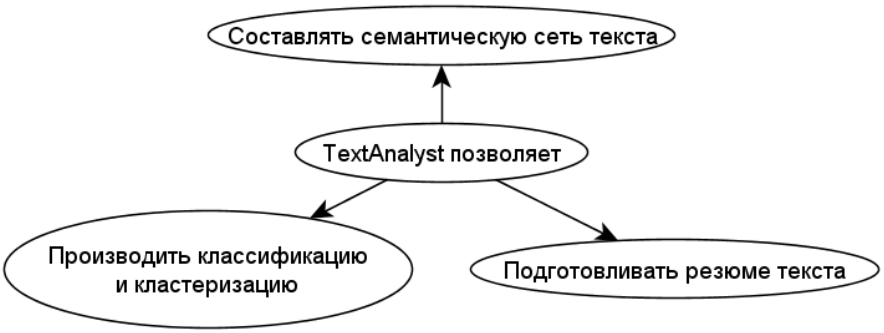

Рисунок 25 – Задачи, которые решает TextAnalyst

Система OracleText отличается тем, что для работы с текстом используется реляционная СУБД, к которой документы ищутся по ключевым слова и фразам. В Oracle Text небольшое количество функций для работы с текстами на русском языке, поэтому компанией «Гарант-Парк-Интернет» был разработан модуль Russian Context Optimizer. Модуль включает информацию о русской морфологии, содержит средства нечёткого поиска, реферирования документов и тематического анализа.

MS Office AutoSummarize – функция автоматического получения краткого реферата, которая встроена в офисный пакет. Однако полученный реферат будет невысокого качества.

Inxight Summarizer использует систему взвешенных показателей для каждого предложения, в основном, больший вес получают первые и последние предложения параграфа, исходя из параметров запроса пользователя.

Summarist – это попытка разработать систему, объединяющую методы экстракции и абстракции. На сегодняшний день SUMMARIST производит рефераты на пяти языках. На данный момент производятся работы, направленные на расширение системы и создание базы знаний, чтобы использовать метод абстракции.

Copernic summarizer выделяет релевантные предложения на основе статистических методов. Умеет анализировать текст на одном из четырёх языков. Может сгенерировать краткое содержание, опираясь на желания пользователя. Реферат может генерироваться по тексту, содержащемуся в текстовых документах, электронных письмах, веб-страницах, PDF-файлах.

Columbia newsblaster – система для автоматической обработки ежедневных новостей без использования ручного труда человека.

Сравнительная характеристика описанных систем представлена в таблице 4.

Таблица 4 – Сравнительная характеристика систем автоматического реферирования

Наименование системы	Характеристики системы		
	Метод реферирования	Поддержка русского языка	Качество реферирования
Intelligent Miner for Text (IBM)	Статистические методы	Отсутствует	Высокое
TextAnalyst		Есть	Среднее
Oracle Text, RCO		Отсутствует	Среднее
MS Office AutoSummarise		Есть	Низкое
Inxight Summarizer		Отсутствует	Среднее
Summarist		Отсутствует	Высокое
Copernic summarizer	Статистические методы, лингвистические методы	Отсутствует	Среднее

Что касается русского языка, известны только две системы, которые позволяют получать рефераты на русском языке: TextAnalyst и функция Autosummarize, которая встроена в пакет Microsoft Office. Эти системы относятся к группе систем, которые используют какие-либо из статистических методов. Согласно наблюдениям, использование статистических методов для анализа текста на естественном языке достигло своего предела. Дальнейшее усложнение математики, не учитывающее особенности структуры текста, не сможет заметно улучшить системы, работающие только на базе статистических методов. Поэтому, с целью повышения эффективности

решения задачи автоматического реферирования русскоязычных научно-технических текстов, предлагается использовать подход, который учитывает особенности их структуры.

3 Алгоритм реферирования текста

3.1 Общая схема алгоритма автоматического реферирования

Процесс автоматического реферирования текста на основе разработанных алгоритмов должен состоять из нескольких этапов, основными из которых являются следующие: разбиение текста, разбиение предложений и составление реферата.

Общая схема алгоритма автоматического реферирования представлена на рисунке 26.

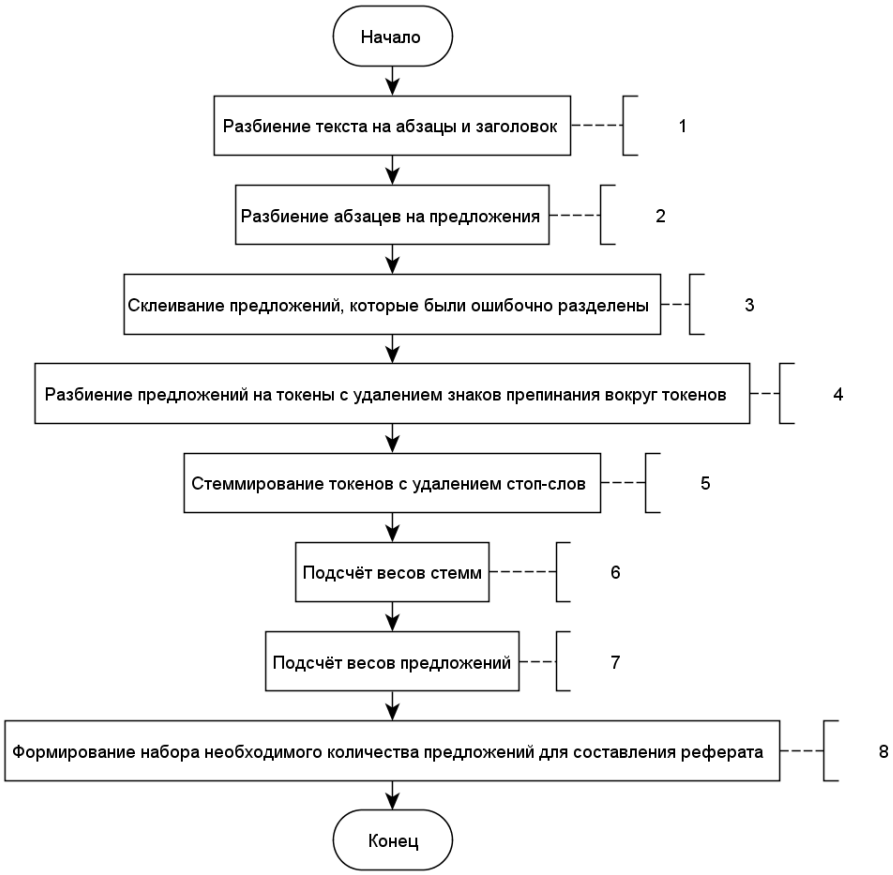

Рисунок 26 – Общая схема алгоритма автоматического реферирования

Этапами алгоритма автоматического реферирования являются:

1) разбиение текста на предложения (блоки 1-3);

2) разбиение предложений на токены (блоки 4, 5);

3) составление реферата (блоки 6-8).

Далее подробно рассмотрены данные этапы и алгоритмы, использующиеся на данных этапах.

3.2 Разработка алгоритма разбиения текста на предложения

Алгоритм разбиения текста на предложения предполагает наличия на входе исходного текста, и включает в себя три этапа:

1) разбиение текста на абзацы и заголовок;

2) разбиение абзацев на предложения;

3) склеивание предложений, которые были ошибочно разделены.

Блок-схема алгоритма разбиения текста на абзацы и заголовок представлена на рисунке 27.

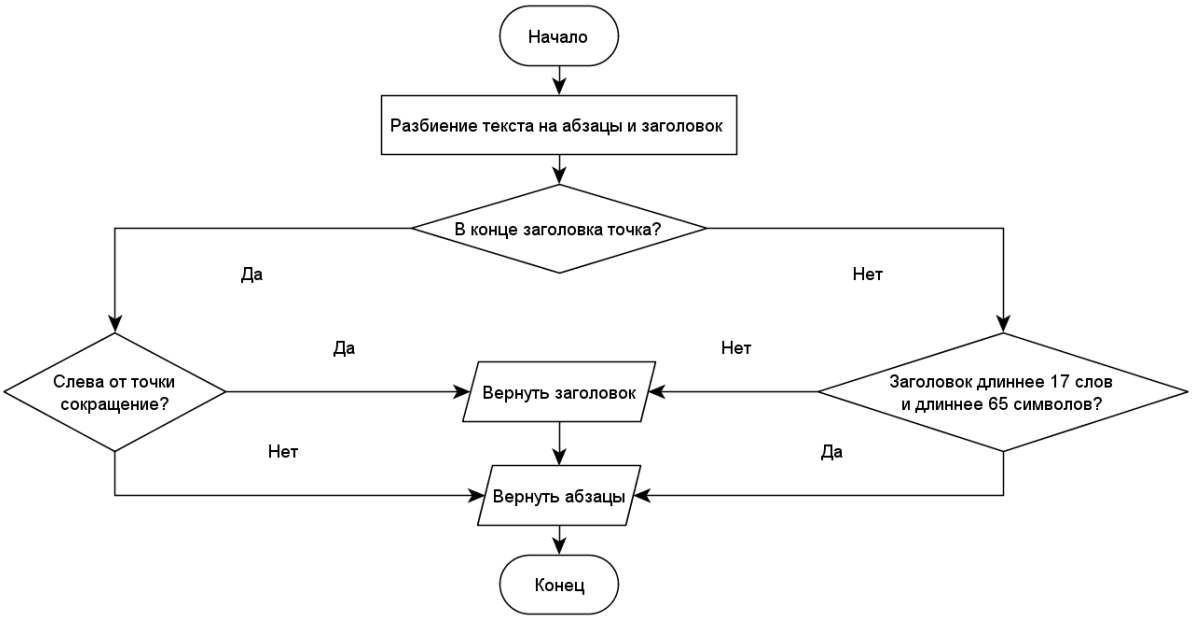

Рисунок 27 – Блок-схема алгоритма разбиения текста на абзацы и заголовок

Блок-схема алгоритма разбиения абзацев на предложения, склеивания предложений, которые были ошибочно разделены, представлена на рисунке 28.

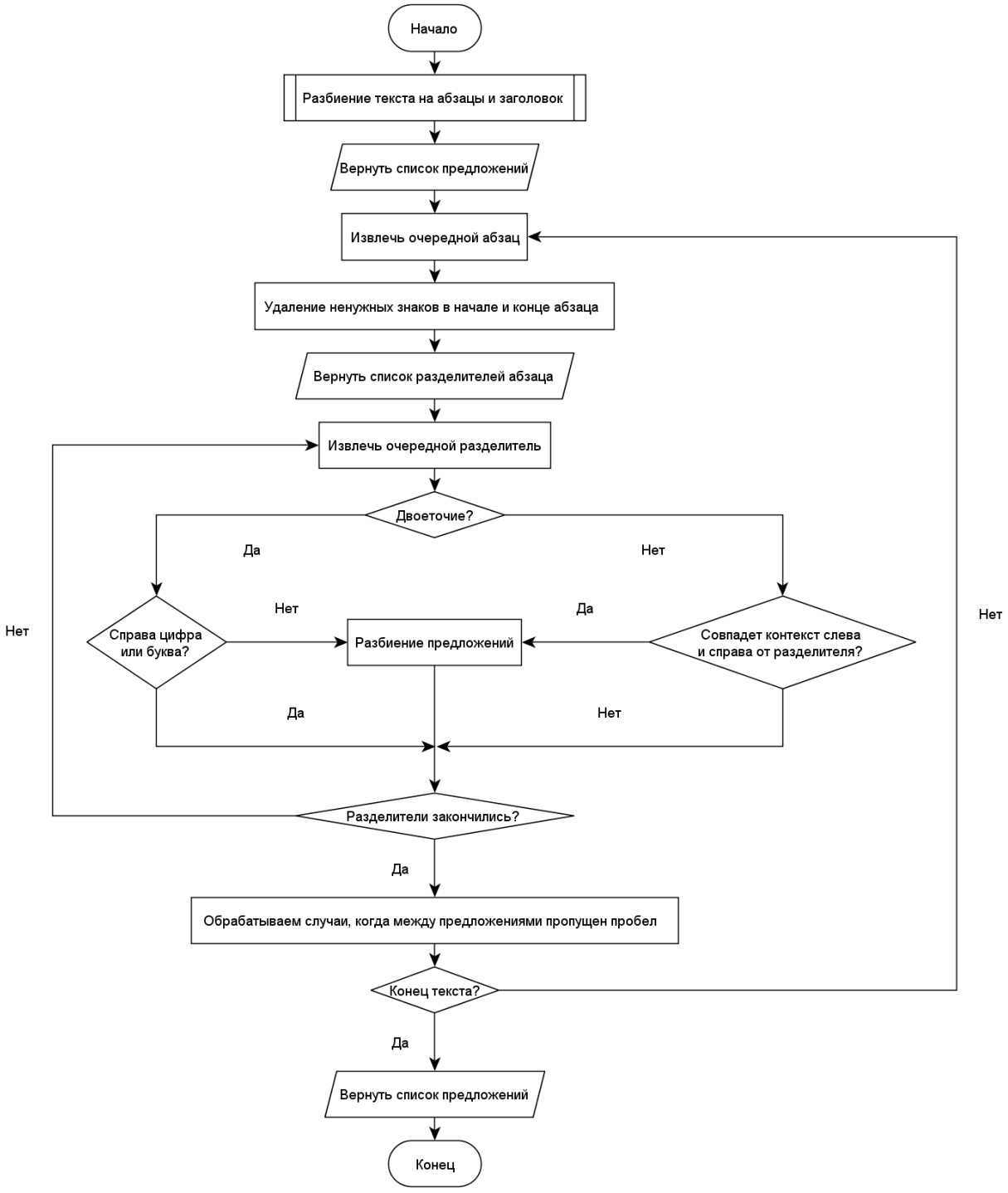

Рисунок 28 – Блок-схема алгоритма разбиения абзацев на предложения, склеивания предложений, которые были ошибочно разделены

Набор регулярных выражений для определения правильного контекста приведен в таблице 5.

Таблица 5 – Список регулярных выражений для определения правильного контекста

Регулярное выражение	Семантика
+	«...конец. Начало...»
[\"\'”“”„«»‘)\]\}>`′″*¹³²]+[\s.!?]*\s	«...конец.")! Начало...»
\[*[а-яА-Я0-9,.\-:]+ *\]+\s+	«... конец. Начало...»
[•\"\'’„“«‘(\[<{`′″]*[А-Я\d]	Слово с заглавной буквы или цифра, перед которыми может идти открывающая скобка или кавычка или маркер списка
[А-Я]?[а-я_\'»)\]\}>]+	Слово с заглавной буквы, после которого идет подчеркивание, обратный слэш, закрывающая скобка или кавычка
[\"“„«‘(\[<{`]?[А-Я]([А-Яа-я.\']+)?	Скобка или кавычка, после которых идут заглавная буква, слово или цифра
[0-9а-яА-Я]+([.,:][0-9а-яА-Я]+)*	После двоеточия идет цифра или слово

3.3 Разработка алгоритма разбиения предложений на токены

Алгоритм разбиения предложений на токены предполагает выполнение двух основных этапов:

1) разбиение предложений на токены с удалением знаков препинания вокруг токенов;

2) стеммирование токенов с удалением стоп-слов.

Блок-схема алгоритма разбиения предложений представлена на рисунке 29.

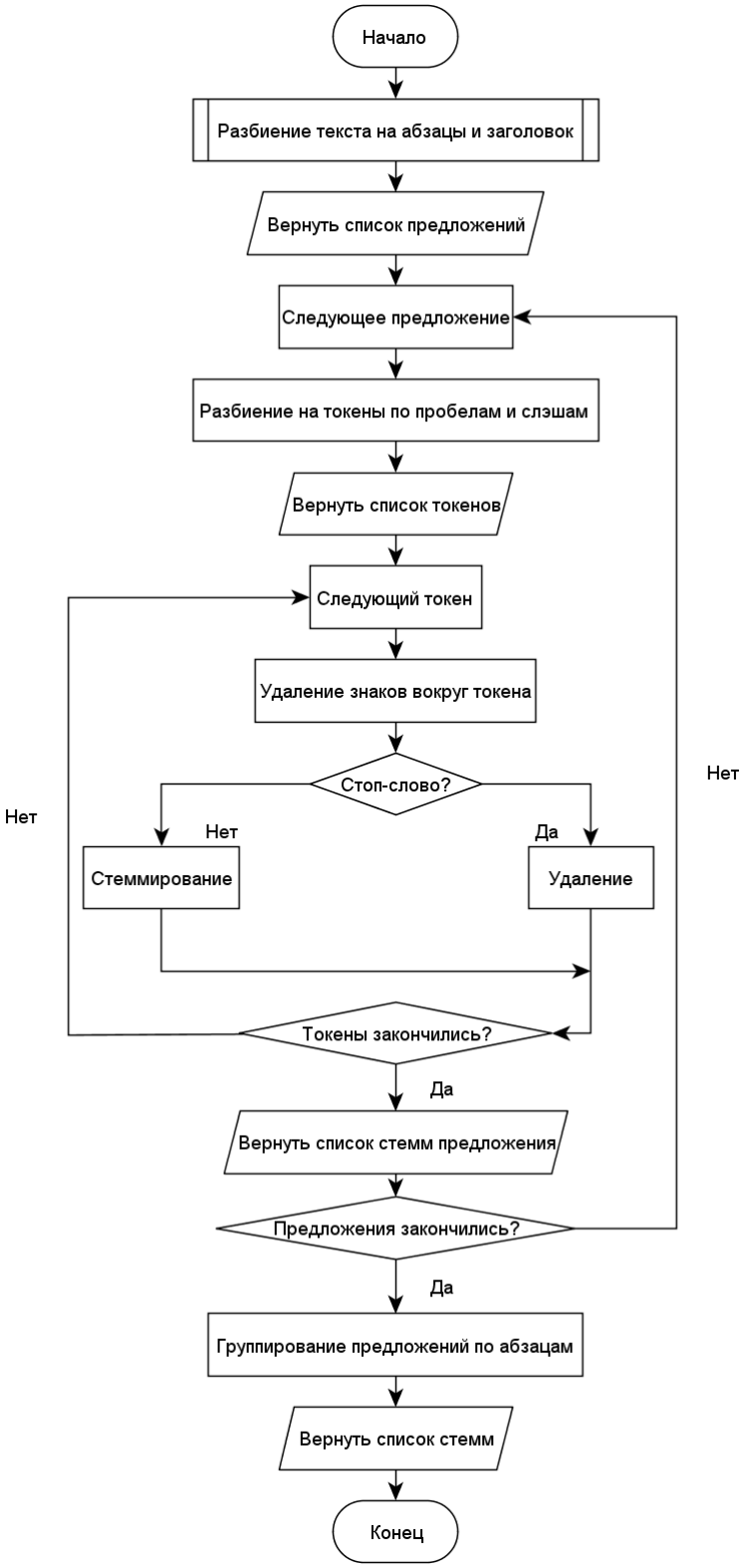

Рисунок 29 – Блок-схема алгоритма разбиения предложений

3.4 Отбор предложений для включения в реферат

Алгоритм составления реферата предполагает выполнение трех основных этапов:

1) подсчет весов стемм;

2) подсчет весов предложений;

3) формирование набора необходимого количества предложений для составления реферата.

Блок-схема алгоритма подсчета весов стемм и предложений представлена на рисунке 30.

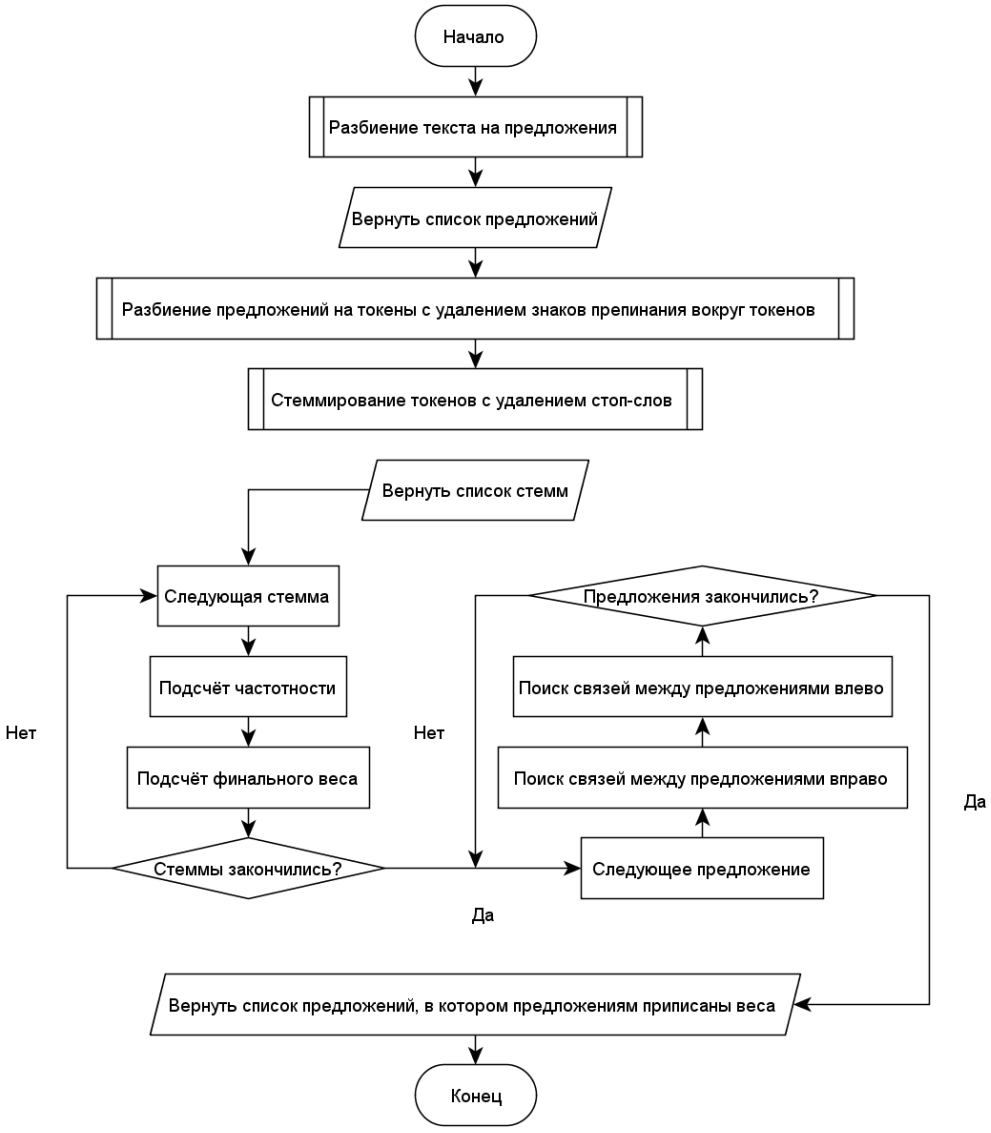

Рисунок 30 – Блок-схема алгоритма подсчета весов стемм и предложений

Блок-схема алгоритма составление реферата представлена на рисунке 31.

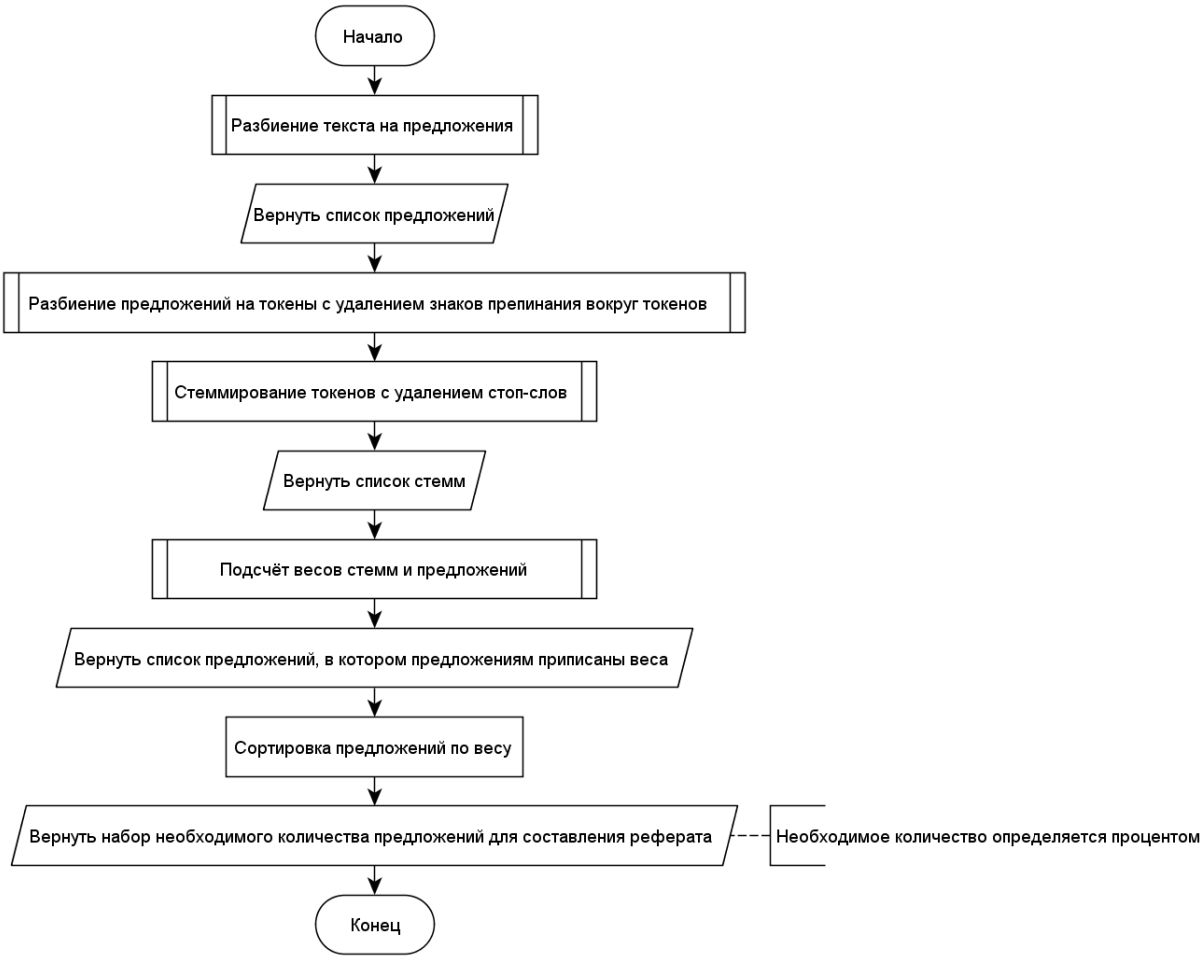

Рисунок 31 – Блок-схема алгоритма составления реферата

4 Реализация и анализ эффективности системы реферирования

4.1 Общая структура программы

Структура системы представлена на рисунке 32.

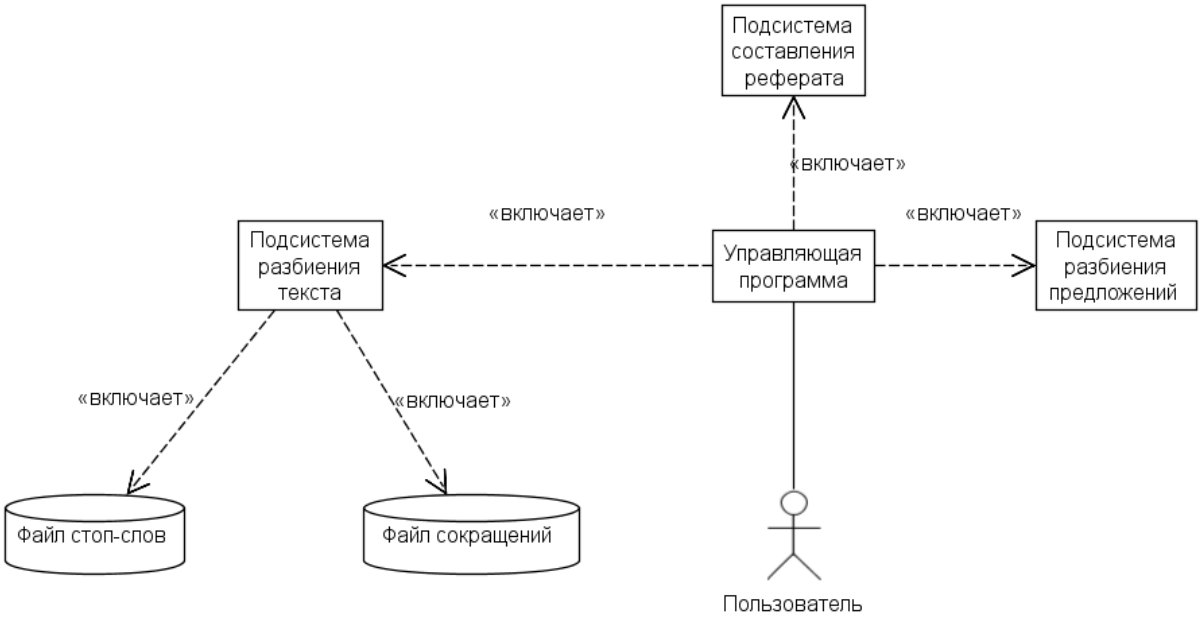

Рисунок 32 – Структура системы автоматического реферирования

Согласно алгоритму, первой активируется подсистема разбиения текста, которая на вход получает текст в виде последовательности символов, а в качестве результата – список предложений. После этого начинает работу подсистема разбиения предложений, которая из полученных системой разбиения текста предложений выделяет токены и стеммирует их. Подсистема реферирования присваивает веса предложениям и выбирает предложения для составления реферата.

Как говорилось выше, подсистема разбиения текста имеет на входе исходный текст. Главными задачами этой подсистемы являются:

1) разбиение текста на абзацы и заголовок;
2) разбиение абзацев на предложения;
3) склеивание предложений, которые были ошибочно разделены.

Входом подсистемы разбиения предложений является заголовок, если он есть, и набор предложений. Результатом работы подсистемы разбиения предложений является набор стемм.

Для получения результата подсистема выполняет следующие действия:

1) разбиение предложений на токены с удалением знаков препинания вокруг токенов;

2) стеммирование токенов с удалением стоп-слов.

Подсистема составления реферата имеет на входе набор стемм, а также размер предполагаемого реферата. Выходом подсистемы является реферат, составленный в соответствии с заданным размером. Основными действиями подсистемы являются:

1) подсчет весов стемм;

2) подсчет весов предложений;

3) формирование набора необходимого количества предложений для составления реферата.

Далее рассмотрена программная реализация системы автоматического реферирования текста.

4.2 Детали реализации программы

Реализация системы осуществлена на языке программирования Python 3.6. Python – язык программирования высокого уровня, являющийся кроссплатформенным, интерпретируемым языком общего назначения. К основной особенности можно отнести понятный синтаксис, позволяющий помочь разработчику в чтении программного кода и увеличивающий производительность его написания. Python поддерживает различные высокоуровневые структуры данных, несколько парадигм программирования, в том числе функциональную и объектно-ориентированную.

Стандартная библиотека языка программирования, входящая в комплект интерпретатора языка, позволяет решать большинство прикладных задач. Кроме стандартных библиотек присутствует обширный каталог свободно разрабатываемых библиотек. Среди них можно отметить библиотеки для научных исследований – numpy и scipy, которые можно

рассматривать как альтернативу MATLAB – пакету прикладных программ для технических расчетов. Также стоит отметить интерактивную веб-оболочку для научных исследований – Jupyter Notebook [16], которая предоставляет писать и распространять тексты программ, а также визуализировать различные данные. Аналитики данных используют язык Python и вспомогательные средства, а также активно развивают их.

Кроме решения исследовательских задач Python используется для создания оконных и веб-приложений. Для создания оконных приложений был адаптирован для языка Python графический фреймворк Qt. Но язык наиболее активно применяется при веб-разработке. Популярным веб-фреймворком в языке Python является Django, который решает большинство задач, а также упрощает создание веб-приложений [17].

Эти особенности языка делают его эффективным инструментом для написания сценариев, научных исследований и для разработки различных приложений. Для исследований, проводимых в текущей работе, язык программирования Python может быть применен. Последующая реализация продукта также возможна с использованием Python, например, в виде веб-сервиса. В языке Python также есть дополнительные пакеты для обработки естественного языка.

Согласно рисунку 32, для реализации системы автоматического реферирования, а именно подсистем разбиения текста и предложений, необходимо использовать методы обработки естественного языка. Основным пакетом для работы с естественным языком в языке программирования Python является NLTK [18]. Этот пакет включает более 50 корпусов текста и лексических ресурсов, а также набор библиотек для обработки текста, классификации, токенизации, тегирования. Однако данное решение в некоторых случаях не поддерживает русский язык, например, при морфологическом анализе текста.

Для морфологического анализа текста на русском языке сотрудником компании Scrapinghub был создан морфологический анализатор pymorphy2 [19, 20].

Данный пакет позволяет:

1) приводить слова к нормальной форме, например, слово люди к форме человек, или слово гулял к форме гулять;

2) ставить слово в нужную форму;

3) возвращать грамматическую информацию о слове, например, число, род, падеж, часть речи.

Далее в таблице 6 представлены основные классы системы автоматического реферирования.

Таблица 6 – Описание основных классов системы автоматического реферирования

Наименование класса	Описание
LoadExternalLists	Загружает в память файл стоп-слов и файл сокращений
TextSegmentor	Разбивает текст на абзацы, определяет заголовок текста, если он есть, разбивает абзацы на предложения
FindProperNouns	Ищет имена собственные
CountTermWeights	Подсчитывает веса стемм
SymmetricalSummarizationWeightCount	Производит начисление весов предложениям, сортирует предложения в порядке убывания весов, формирует набор необходимого количества предложений для составления реферата

Продолжение таблицы 6

Наименование класса	Описание
SentenceSplitter	Разбивает отдельные предложения на токены, формирует набор стеммированных токенов с удалением стоп-слов, сохраняя при этом структуру текста

Диаграмма классов системы автоматического реферирования представлена на рисунке 33.

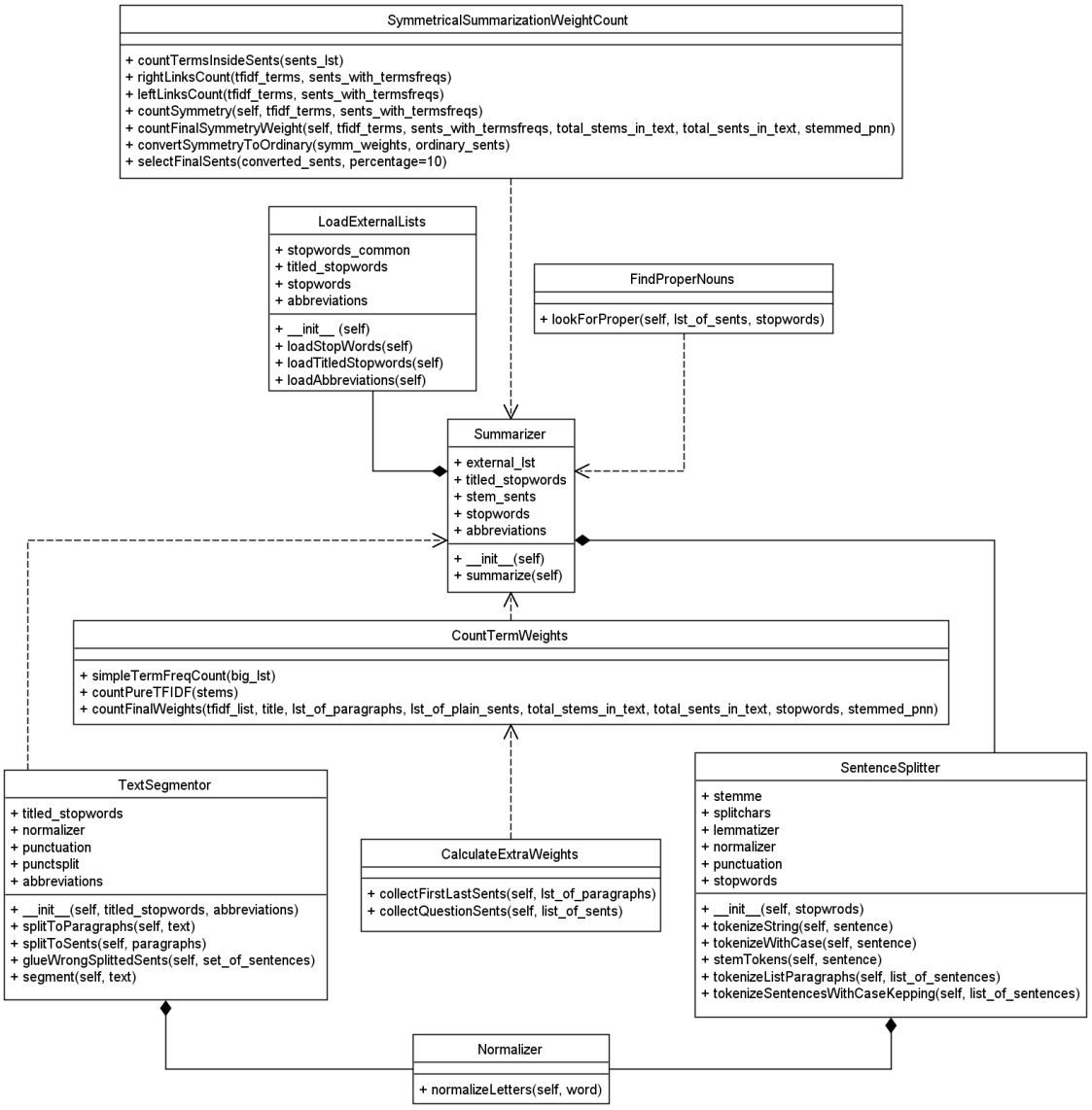

Рисунок 33 – Диаграмма классов системы автоматического реферирования

Основные функции подсистемы разбиения текста представлены в таблице 7.

Таблица 7 – Описание основных функций подсистемы разбиения текста

Наименование функции	Описание
splitToParagraphs	Разбивает текст на абзацы и определяет заголовок текста, если он есть
splitToSents	Разбивает абзацы на предложения
glueWrongSplittedSents	Склеивает предложения, которые были ошибочно разделены

Основные функции подсистемы разбиения предложений представлены в таблице 8.

Таблица 8 – Описание основных функций подсистемы разбиения предложений

Наименование функции	Описание
normalizeLetters	Конвертирует «ё» в «е»
tokenizeString	Разбивает отдельные предложения на токены
stemTokens	Формирует набор стеммированных токенов с удалением стоп-слов

Основные функции подсистемы составления реферата представлены в таблице 9.

Таблица 9 – Описание основных функций подсистемы составления реферата

Наименование функции	Описание
lookForProper	Ищет имена собственные

Продолжение таблицы 9

Наименование функции	Описание
collectFirstLastSents	Формирует набор первых и последних предложений
collectQuestionSents	Формирует набор вопросительных и восклицательных предложений
simpleTermFreqCount	Считает частотности стемм
countFinalWeights	Подсчитывает веса стемм
countSymmetry	Производит начисление весов предложениям
countFinalSymmetryWeight	Добавляет к весам предложений дополнительный коэффициент, чтобы длинные предложения не набрали больший вес
convertSymmetryToOrdinary	Сортирует предложения в порядке убывания весов
selectFinalSents	Формирует набор необходимого количества предложений для составления реферата

На вход поступает текст, считываемый из файла. Затем текст делится на заголовок и абзацы по символу перевода строки. Заголовком считается строка без точки слева от знака переноса и содержащая не более 17 слов или 65 символов. Разбиение на слова осуществляется по пробелам. Если точка все же есть, то проверяется, не является ли она элементом сокращения с помощью регулярного выражения. Все допустимые сокращения прописаны в регулярном выражении в явном виде.

Затем абзацы разбиваются на предложения. Конец предложения определяется согласно регулярным выражениям из таблицы. Формируется

список разделителей. При этом если текст слева от разделителя соответствует регулярному выражению левого контекста, а текст справа от разделителя соответствует регулярному выражению правого контекста, то разделитель представляет собой конец предложения, и осуществляется разбиение, в противном случае разделитель входит в предложение.

Далее осуществляется склеивание ошибочно разделённых предложений.

Предложения разбиваются на токены, которые подвергаются стеммированию, при этом буква «ё» конвертируется в букву «е». Токены помещаются в структуру, представляющую собой трёхмерный список. Первый уровень списка соответствует абзацу, в который помещаются списки, соответствующие предложениям, и уже в этот список помещаются стеммы данного предложения. Абзацы в тексте, предложения в абзаце и токены в предложении идут в том же порядке, что и в исходном тексте. Удаляется пунктуация. Удаляются стоп-слова. Это слова, не несущие смысловой нагрузки – союзы, предлоги, частицы, вводные слова, слова-паразиты, общепринятые сокращения.

В полученном списке токенов выбираются имена собственные – они будут влиять на вес предложения, в котором находятся. Именем собственным считается слово с большой буквы, не стоящие в начале предложения.

Формируется список первых и последних предложений абзаца, а также восклицательных и вопросительных предложений (определяются по знаку в конце предложения).

Для каждой стеммы считается частота появлений и общее количества стемм. Высчитывается доля каждой стеммы в тексте. Считается среднее арифметическое долей стемм в тексте. Стеммы, доля которых меньше найденного значения, удаляются. Для оставшихся стемм высчитываются веса. За основу берется частотность, также используются дополнительные коэффициенты.

Формируется список слов первых и последних предложений каждого абзаца, общее количество слов в этих предложениях и количество слов, доля которых выше среднего арифметического, также в этих предложениях.

Для каждого абзаца вычисляется доля отобранных слов среди всех слов первого и последнего предложения.

Также считается доля слов в первых и последнего предложений всех абзацев к общему количеству слов в тексте.

Далее формируется список слов в вопросительных и восклицательных предложениях и считается количество таких предложений.

Если отобранные стеммы есть в заголовке, то их вес удваивается.

Если отобранные стеммы есть в первом или последнем предложении абзацев, то вес умножается на частное доли отобранных стемм в первом и последнем предложении этого абзаца и доли слов первого и последнего предложения в абзаце.

Если стемма встретилась в вопросительном или восклицательном предложении, то вес умножается на долю количества таких предложений в тексте.

Если стемма имя собственное, то умножается на частное доли отобранных стемм в первом и последнем предложении этого абзаца и доли слов первого и последнего предложения в абзаце.

После этого производится начисление весов предложениям. Подчитывается частота всех входящих в него стемм.

Осуществляется поиск вправо связи между предложениями.

Если стемма из данного предложения вошла в какое-то из предложений справа, считается количество отобранных стемм в предложениях, расположенных справа и слева от данного, их сумма и будет весом текущего предложения.

Параллельно для текущего предложения вычисляется сумма весов отобранных стемм в данном предложении, которая прибавляется в весу предложения, который был получен на предыдущем шаге. параллельно

вычисляется позиционный коэффициент – чем раньше предложение встречается в тексте, тем больше его вес, который тоже прибавляется к весу предложения.

Еще один коэффициент вводится для того, чтобы большие предложения не набрали наибольший вес.

Вес первого предложения в тексте удваивается. Также вес каждого предложения умножается на количество имен собственных и чисел.

На заключительном этапе выбирается требуемый процент предложений, имеющим наибольший вес с сохранением порядка, в котором они шли в тексте.

4.3 Анализ эффективности

Оценка эффективности разработанного алгоритма проводилась по двум критериям:

1) оценка зависимости времени работы от длины текста;
2) оценка качества получаемого реферата.

Оценка производительности алгоритма выполнялась с помощью разработанной системы на компьютере с процессором Intel Pentium 3550M 2,3 ГГц. Время работы алгоритма оценивалось на множестве научно-технических текстов различной длины от 130 слов до 800 слов.

График зависимости времени составления реферата от объема текста представлен на рисунке 9.

Рисунок 34 – График зависимости времени составления реферата от объема текста

Анализ полученных результатов показал, что время составления реферата прямо пропорционально длине документа и имеет близкую к линейной зависимость от объема текста.

Объективная оценка качества автоматического реферирования текста является непростой задачей. Само понятие реферирования заведомо предполагает субъективный характер оценки получаемых результатов.

Оценка качества реферата, подготовленного программой субъективна и требует работы эксперта.

На рисунке 35 представлены основные этапы подготовки и проведения оценки качества рефератов.

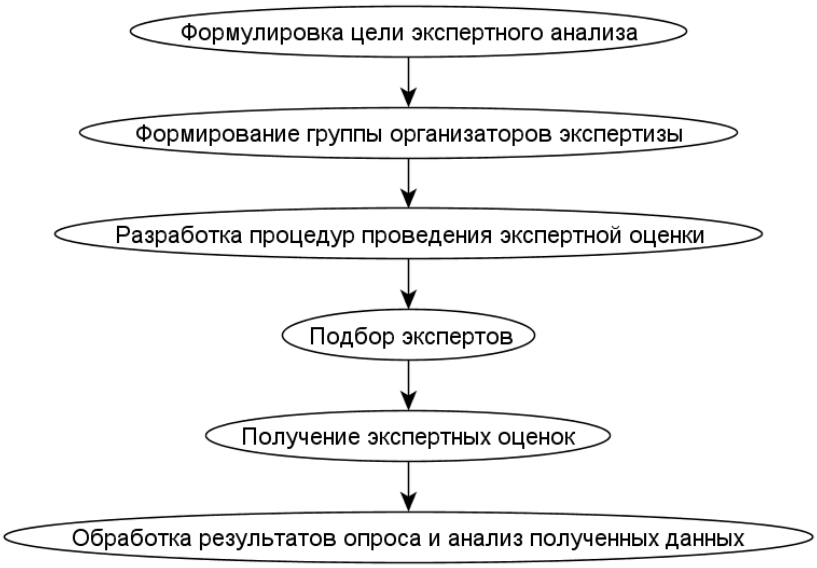

Рисунок 35 – Основные этапы подготовки и проведения оценки качества рефератов

Цель экспертного анализа – оценить эффективность разработанного метода и алгоритмов формирования рефератов. Это возможно сделать опосредованно на основе оценки качества получаемых рефератов. Проблема подбора экспертов является одной из наиболее значимых в практике экспертных исследований.

Очевидно, в качестве экспертов необходимо использовать тех людей, чьи суждения помогут принятию адекватного решения. Процедура проведения экспертной оценки при выборе фрагментов текста для составления реферата состояла в следующем.

Эксперту было предъявлено 5 одних и тех же научно-технических текстов разного объема с количеством слов от 130 до 800.

При составлении реферата экспертом оценка предложения для включения его в реферат проводилась по интервальной шкале методом непосредственной оценки.

Каждому предложению эксперт присваивает одну из следующих оценок:

1) 0 – неважные предложения;
2) 1 – относительно важные предложения;
3) 2 – самые важные предложения.

Основной задачей оценки получаемого реферата является установление смыслового соответствия или, иными словами, семантического тождества реферата и первоисточника. Для решения данной задачи традиционно используются критерий семантической адекватности и критерий семантической эквивалентности. Для количественной оценки критерия полноты используется отношение полученных в реферате релевантных предложений к общему количеству релевантных предложений. Кроме этого используется гармоническое среднее параметров полноты и точности, которое вычисляется по формуле:

$$F = \frac{2 \cdot Точность \cdot Полнота}{Точность + Полнота} \quad (1)$$

В настоящее время известны две системы автоматического реферирования для русского языка: TextAnalyst, которая позволяет получать реферат для текста на русском языке на основе построения семантической сети основных понятий и встроенная функция пакета Microsoft Office AutoSummarizer, использующая статистическое методы, основанные на анализе частоты повтора слов в тексте.

В таблице 9 представлены результаты сравнительной оценки качества рефератов, получаемых с помощью разработанной системы, системы TextAnalyst и системы Microsoft AutoSummirize.

Таблица 9 – Средние значения показателей качества методов автоматического реферирования

Система	Полнота	Точность	F-параметр
Разработанная система	65%	67%	66%
Система TextAnalyst	48%	45%	46%
Система Microsoft AutoSummirize	35%	33%	34%

Анализ полученных результатов показывает, что качество рефератов, полученных с помощью разработанной системы, значительно выше по сравнению с рефератами, полученными с помощью систем TextAnalyst и Microsoft AutoSummirize.

Далее необходимо выяснить, как влияет объем текста на качество рефератов. Такое исследование было проведено для текстов разного объема. Результаты проведенного исследования представлены в таблице 10.

Таблица 10 – Зависимость качества рефератов от объема текста

Текст	Количество слов	Количество предложений	Полнота	Точность	F-параметр
1	130	8	68%	74%	70%
2	207	14	62%	72%	67%
3	365	21	69%	64%	67%
4	515	32	65%	68%	67%
5	750	45	63%	65%	62%

Анализ полученных результатов показывает, что качество рефератов практически не зависит от объема текста.

Полученные результаты согласуются с оценками полноты и точности, приведенными ранее, и подтверждают более высокую эффективность разработанной системы.

Проведенные экспериментальные исследования показали, что качество рефератов, полученных на основе разработанного метода, в среднем на 20% выше по сравнению с рефератами, полученными с помощью традиционных методов, реализованных в системе TextAnalyst и встроенной функции пакета Microsoft Office – Autosummarize.

ЗАКЛЮЧЕНИЕ

Таким образом, в процессе написания работы проведен анализ современных подходов к автоматическому реферированию текстов.

Разработаны алгоритмы:

1) разбиения текста на предложения;

2) разбиения предложений на токены;

3) составления реферата.

На основе разработанных алгоритмов разработана система автоматического реферирования.

Проведена оценка эффективности разработанных методов и алгоритмов.

Практическая ценность работы заключается в том, что разработанное программное обеспечение позволяет строить системы автоматического реферирования научно-технического текста для русского языка, что делает более быстрым поиск в коллекциях документов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Люгер Дж. Искусственный интеллект: стратегии и методы решения сложных проблем. М., 2005.

2 Somers H. Machine Translation: Latest Developments. In: The Oxford Handbook of Computational Linguistics. Mitkov R. (ed.). Oxford University Press, 2003, p. 512-528.

3 Маннинг К., Рагхаван П., Шютце Ч. Введение в информационный поиск – М.: Вильямс, 2011.

4 Васильев В.Г., Кривенко М.П. Методы автоматизированной обработки текстов. – М.: ИПИ РАН, 2008.

5 Барсегян А.А. и др. Технологии анализа данных: Data Mining, Visual Mining, Text Mining, OLAP – 2-е изд. – СПб.: БХВ-Петербург, 2008.

6 Grishman R., Information Extraction. In: The Handbook of Computational Linguistics and Natural Language Processing. A. Clark, C. Fox, and S. Lappin (Eds), Wiley-Blackwell, 2010, p. 515-530.

7 Harabagiu S., Moldovan D. Question Answering. In: The Oxford Handbook of Computational Linguistics. Mitkov R. (ed.). Oxford University Press, 2003, p. 560-582.

8 Pang Bo, Lee L. Opinion Mining and Sentiment Analysis. In: Foundations and Trends ○R in Information Retrieval. Now Publishers, 2008.

9 Wu J., Yu-Chia Chang Y., Teruko Mitamura T., Chang J. Automatic Collocation Suggestion in Academic Writing. In: Proceedings of the ACL 2010 Conference Short Papers, 2010.

10 Большаков И.А. КроссЛексика – большой электронный словарь сочетаний и смысловых связей русских слов. // Комп. лингвистика и интеллект. технологии: Труды межд. Конф. «Диалог 2009». Вып. 8 (15) М.: РГГУ, 2009, с. 45-50.

11 Bateman J., Zock M. Natural Language Generation. In: The Oxford Handbook of Computational Linguistics. Mitkov R. (ed.). Oxford University Press, 2003, p. 304.

12 Рождественская Н.В. Дискурс как высшая единица коммуникативного акта. [Электронный ресурс]. URL: https://zsu.zp.ua/herald/articles/1955.pdf (дата обращения 27 мая 2019).

13 Яцко В.А. Симметричное реферирование: теоретические основы и методика [Текст] / В.А. Яцко // НТИ. Сер. 2. - 2002. - №5. - с. 18-28.

14 Алыгулиев Р.М. Автоматическое реферирование документов с извлечением информативных предложений [Текст] / Р.М. Алыгулиев // Вычислительные технологии. – 2007. – Т. 12, № 5. - с. 5-15.

15 Ефименко И.В. Лингвистические аспекты кросс-языкового реферирования: синтез текстов под управлением предметных онтологий [Текст] И.В. Ефименко // Труды 10-ой конф. по искусственному интеллекту. - М.: Физматлит, 2006, - Т 1. - с. 81-87.

16 About Jupyter [Электронный ресурс]. URL: https://jupyter.org/ (дата обращения 27 мая 2019).

17 Why Django? [Электронный ресурс]. URL: https://www.djangoproject.com/start/overview/ (дата обращения 27 мая 2019).

18 NLTK 3.3 Documentation [Электронный ресурс]. URL: https://nltk.org/ (дата обращения 27 мая 2019).

19 Документация pymorphy2 [Электронный ресурс]. URL: https://pymorphy2.readthedocs.io/en/latest/ (дата обращения 27 мая 2019).

20 Korobov M. Morphological Analyzer and Generator for Russian and Ukrainian Languages // Analysis of Images, Social Networks and Texts, 2015. - с. 320-332.

ПРИЛОЖЕНИЕ А

Программная реализация системы автоматического реферирования

Файл main.py:

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

from ResListsLoaderClass import LoadExternalLists
from SymmetricalSummarizingClass import *
from TextSegmentorClass import TextSegmentor


class Summarizer:
    def __init__(self):
        self.external_lst = LoadExternalLists()
        self.stopwords = self.external_lst.loadStopWords
        self.titled_stopwords = self.external_lst.loadTitledStopwords
        self.abbreviations = self.external_lst.loadAbbreviations
        self.stem_sents = SentenceSplitter(self.stopwords)

    def summarize(self):
        global q
        with open('text.txt', 'r') as file:
            opentext = file.read()
        # разбиваем в список предложений входной текст
        textsegmentor = TextSegmentor(self.titled_stopwords, self.abbreviations)
        list_of_sentences, ttl = textsegmentor.segment(opentext)
        # склеиваем все списки в один простой список всех предложений для статистики по
предложениям
        all_sentences = list(itertools.chain.from_iterable(list_of_sentences))
        # для методики симметричного реферированияя нужно не менее трёх предложений
        if len(all_sentences) >= 3:
            # стеммируются предложения
            # список предложений из основ слов, предложения сгруппированны по абзацам
            stemmed_sentences = self.stem_sents.tokenizeListParagraphs(list_of_sentences)
            # стеммируется заголовок
            if len(ttl) > 0:
                title_pairs =
list(itertools.chain.from_iterable(self.stem_sents.tokenizeListSentences(ttl)))
                title = [pair[0] for pair in title_pairs]
            else:
                title = []
            # список предложений без границ абзацев, предложения разбиты на основы
            no_paragraphs = list(itertools.chain.from_iterable(stemmed_sentences))
            # большой список всех основ слов для подсчета частотности (TF/IDF)
            big_list_of_pairs =
list(itertools.chain.from_iterable(itertools.chain.from_iterable(stemmed_sentences)))
            big_list_of_stems = [pair1[0] for pair1 in big_list_of_pairs]
            # общее количество стем в тексте
            total_stems_in_text = len(big_list_of_stems)
```

```
            # общее количество предложений в тексте
            total_sents_in_text = len(all_sentences)
            if total_stems_in_text > 0:
                w_count = CountTermWeights()
                # список кортежей (слово, его частота), усечённый по средней частоте
                total_stem_count, absolute_count =
w_count.simpleTermFreqCount(big_list_of_stems)
                # список имён собственных
                proper_nouns, stemmed_pnn = FindProperNouns().lookForProper(all_sentences,
self.stopwords)
                # список терминов с весовыми коэффициентами (кортежи)
                sorted_tfidf = w_count.countPureTFIDF(total_stem_count)
                final_sorted_tfidf = w_count.countFinalWeights(sorted_tfidf, title,
stemmed_sentences, all_sentences,
                                                total_stems_in_text, total_sents_in_text, self.stopwords,
                                                stemmed_pnn)
                # объект класса для вычисления симметричной связи предложений
                # вычисляем вес каждого предложения
                symmetry = SymmetricalSummarizationWeightCount()
                # словари каждого предложения с частотностью по словам
                s_with_termfreqs = symmetry.countTermsInsideSents(no_paragraphs)
                symmetrical_weights = symmetry.countFinalSymmetryWeight(final_sorted_tfidf,
s_with_termfreqs,
                                                    total_stems_in_text, total_sents_in_text,
                                                    stemmed_pnn)
                original_sentences = symmetry.convertSymmetryToOrdinary(symmetrical_weights,
all_sentences)
                q, rate = symmetry.selectFinalSents(original_sentences)
            else:
                print('Нет слов для обработки!')
        else:
            q = ''
            print('В тексте должно быть не менее 3 предложений.')
        with open('output.txt', 'w', encoding='utf-8') as output_file:
            for sent3 in range(len(q)):
                output_file.write(q[sent3][0] + '\n')
                #print(q[sent3][0])


def main():
    Summarizer().summarize()


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

Файл ResListsLoaderClass.py:

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

```
# stopwords содержит список стоп-слов.
# titled_stopwords содержит список стоп-слов с заглавной бувы (необходим для разбиения на предложения).
# abbreviations содержит список аббревиатур с точкой, после которых не разбивать на предложения.

class LoadExternalLists(object):
    """
    Загружаем в память файл стоп-слов (+ с заглавной буквы) и файл сокращений для разделителя.
    """

    def __init__(self):
        self.stopwords = set()
        self.titled_stopwords = set()
        self.abbreviations = set()

    @property
    def loadStopWords(self):
        print('Загружаем в память файл стоп-слов')
        with open(r'./txt_resources/stopwords.txt', 'r', encoding='utf-8') as file_openstopw:
            self.stopwords = set(file_openstopw.read().split('\n'))
        return self.stopwords

    @property
    def loadTitledStopwords(self):
        print('Загружаем в память файл стоп-слов с заглавной буквы')
        self.titled_stopwords = tuple([word.title() for word in self.stopwords])
        return self.titled_stopwords

    @property
    def loadAbbreviations(self):
        print('Загружаем в память файл сокращений для разбиения')
        with open(r'./txt_resources/abbrevs_common.txt', 'r', encoding='utf-8') as file_openabbrev:
            self.abbreviations = set(file_openabbrev.read().split('\n'))
        return self.abbreviations
```

Файл TextSegmentorClass.py:

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

from SentenceSplitterClass import *
import re


class TextSegmentor(object):
    """
    Разделение текста на абзацы, предложения и заголовок, если он есть.
    """

    def __init__(self, titled_stopwords, abbreviations):
```

```
        self.titled_stopwords = titled_stopwords
        self.abbreviations = abbreviations
        self.normalizer = Normalizer()

    @staticmethod
    def splitToParagraphs(text):
        """
        Функция разбивает текст на абзацы по новой строке (\n) и определяет заголовок
текста.
        У строки не должно быть точки слева от первого \n,
        при этом такая строка должна быть не длиннее 17 слов (разбиение по пробелам) или
не длиннее 65 символов.
        Тогда это считается заголовком. Если точка все-таки стоит, то проверяется,
        нет ли слева от точки акронима или сокращения. Если есть, то это заголовок, если нет
- абзац.
        """
        title = []
        # заголовок заканчивается акронимом типа т.е.
        acronim = re.compile(r'([А-Яа-я0-9]\.([А-Яа-я0-9]\.)*)')
        # заголовок заканчивается аббревиатурами типа др.
        abbrev =
re.compile(r'(др.|пр.|см.|ср.|напр.|вв.|гг.|тт.|обл.|оз.|гр.|стр.|акад.|доц.|проф.|им.)')
        newline = text.find('\n')
        splitted_line = text[:newline].split(' ')
        acronim_match = acronim.match(splitted_line[len(splitted_line) - 1])
        abbrev_match = abbrev.match(splitted_line[len(splitted_line) - 1])
        if len(splitted_line) <= 17 or len(text[:newline]) <= 65:
            if newline != -1 and text[newline - 1] != '.' and newline != -1 and text[newline - 2] != '.':
                title.append(text[:newline])
                paragraphs = re.split(r'[\r\n]+', text[newline + 1:])
            else:
                if acronim_match or abbrev_match:
                    title.append(text[:newline])
                    paragraphs = re.split(r'[\r\n]+', text[newline + 1:])
                else:
                    paragraphs = re.split(r'[\r\n]+', text)
                    title = []
        else:
            paragraphs = re.split(r'[\r\n]+', text)
            title = []
        return paragraphs, title

    def splitToSents(self, paragraphs):
        """
        Разбиение на предложения. Функция принимает на вход список абзацев,
        возвращает список с вложенными списками предложений.
        """
        set_of_sentences = []
        re_terminators = re.compile(r'[.!?:;…]')
        # Для rightcontext: если после терминатора стоит:
        # 1) пробел ( +) --> ("...конец. Начало...")
        # 2) один из знаков в скобках плюс возможный \s.!?
```

```
        # и пробел ([\"\'“”„«»‘)\]\}>`′″*¹³²]+[\s.!?]*\s) --> ("...конец.")! Начало...")
        # 3) возможный пробел плюс конструкция [1]
        # или (Автор:2000) плюс пробел (\[ *[а-яА-Я0-9,.\-:]+ *\]+\s+) --> ("... конец.[1] Начало...")
        # 4) возможный знак из скобок плюс слово с заглавной буквы или цифра [•\"\'„“«‘(\[<{`′″]*[А-Я\d]
        rightcontext = re.compile(
            r'( +|[\"\'“”„«»‘)\]\}>`′″*¹³²]+[\s.!?]*\s|\[ *[а-яА-Я0-9,.\-:]+ *\]+\s+)[•\"\'„“«‘(\[<{`′″]*[А-Я\d]')
        # если левое от терминатора слово начинается с заглавной буквы (опционно) + слово с _ или \'
        leftcontext_2 = re.compile(r'[А-Я]?[а-я_\'»)\]\}>]+')
        # если правый от терминатора контекст (слово) = (опционно скобка или кавычка) +
        # заглавная буква + слово (опционно) или цифра
        rightcontext_2 = re.compile(r'[\"“„«‘(\[<{`]?[А-Я]([А-Яа-я.\']+)?')
        # если после ':' идет цифра или слово
        for_colon = re.compile(r'[0-9а-яА-Я]+([.,:][0-9а-яА-Я]+)*')
        for paragraph in paragraphs:
            start = 0
            sentences = []
            # удаление висячих знаков в конце и начале абзаца
            paragraph = paragraph.strip()
            # проверка, если нет знаков препинания --> скинуть в список
            if not re_terminators.search(paragraph):
                sentences.append(paragraph[start:])
            else:
                # список всех терминаторов абзаца
                all_terminators = re_terminators.finditer(paragraph)
                for terminator in all_terminators:
                    # начальная позиция терминатор - это индекс в абзаце
                    i = terminator.start()
                    # если совпадает контекст справа от терминатора, запоминаем его
                    if rightcontext.match(paragraph[i + 1:]):
                        match = rightcontext.match(paragraph[i + 1:])
                        # ищем индекс пробела как разделителя между предложениями,
                        # чтобы по нему складывать предложения и устанавливать итератор start
                        space_index = match.group().index(' ')
                        if paragraph[i] == '.':
                            # список слов абзаца от start до первого терминатора
                            # фильтр не пускает пустые строки в список
                            s = list(filter(None, self.punctsplit.split(paragraph[start:i])))
                            # если последнее слово слева от точки не аббревиатура, складываем:
                            if len(s) > 0 and self.normalizer.normalizeLetters(s[len(s) - 1]) not in self.abbreviations:
                                sentences.append(paragraph[start:i + space_index + 1])
                                start = i + space_index + 2
                            # иначе:
                            # если слева слово из аббревиатур, а справа стоп-слово с заглавной буквы,
                            # складываем предложение
                            else:
                                # отфильтровываем, так как в списке первым может стоять пустая строка
```

```
                        sentence_to_right = list(filter(None, self.punctsplit.split(paragraph[i + 2:])))

                        if len(sentence_to_right) > 0 and self.normalizer.normalizeLetters(
                                sentence_to_right[0].strip(self.punctuation)) in self.titled_stopwords:
                            sentences.append(paragraph[start:i + space_index + 1])
                            start = i + space_index + 2
                    # если не точка:
                    else:
                        # а двоеточие
                        if paragraph[i] == ':':
                            # если справа цифра или буква - не разбивать
                            if not for_colon.match(
                                    self.punctsplit.split(paragraph[i + 2:])[0].strip(self.punctuation)):
                                sentences.append(paragraph[start:i + space_index + 1])
                                start = i + space_index + 2
                        else:
                            sentences.append(paragraph[start:i + space_index + 1])
                            start = i + space_index + 2
                # rightcontext не совпал
                else:
                    k = list(filter(None, self.punctsplit.split(paragraph[start:i])))
                    try:
                        lastword = len(k[len(k) - 1])
                    except IndexError:
                        lastword = 2
                    # обработка случаев, когда между предложениями пропущен пробел
                    if leftcontext_2.match(paragraph[i - lastword:i]) and
rightcontext_2.match(paragraph[i + 1:]):
                        if paragraph[i] == '.':
                            if k[len(k) - 1] not in self.abbreviations:
                                sentences.append(paragraph[start:i + 1])
                                start = i + 1
                        else:
                            sentences.append(paragraph[start:i + 1])
                            start = i + 1
            sentences.append(paragraph[start:])
        set_of_sentences.append(sentences)
    return set_of_sentences

@staticmethod
def glueWrongSplittedSents(set_of_sentences):
    """
    Склеивание предложений, которые были ошибочно разделены.
    Функция принимает на вход список вложенных списков с предложениями,
    возвращает список вложенных списков со склеенными предложениями.
    """
    last_sentence_in_glued = ""
    set_of_gluedsentences = []
    for sentences in set_of_sentences:
        all_sentences = []
        for s in range(len(sentences)):
            if len(sentences[s]) > 0:
```

```
                # склейка через двоеточие, если слева от двоеточия не больше 3 слов
                if sentences[s][len(sentences[s]) - 1] == ':' and s != len(sentences) - 1:
                    if sentences[s] == last_sentence_in_glued:
                        continue
                    if len(sentences[s].split()) <= 3 or len(sentences[s + 1].split()) <= 3:
                        glued_sent = sentences[s] + ' ' + sentences[s + 1]
                        last_sentence_in_glued = sentences[s + 1]
                        all_sentences.append(glued_sent)
                    else:
                        all_sentences.append(sentences[s])
                else:
                    if sentences[s] != last_sentence_in_glued:
                        all_sentences.append(sentences[s])
            set_of_gluedsentences.append(all_sentences)
        return set_of_gluedsentences

    def segment(self, text):
        """
        Функция вызывает последовательно три функции:
        1) разбиение на абзацы и заголовок;
        2) разбиение на предложения;
        3) склеивание предложений.
        Возвращает заголовок и список вида [[[], []], [[], []], [[]]].
        Вторая вложенность - абзацы, внутренние списки - предложения.
        """
        # заменяем множественные пробелы на один
        paragraphs, title = self.splitToParagraphs(re.sub(r' {2}', ' ', text))
        set_of_sentences = self.splitToSents(paragraphs)
        set_of_readysents = self.glueWrongSplittedSents(set_of_sentences)
        return set_of_readysents, title
```

Файл SentenceSplitterClass.py:

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

import re

import pymorphy2
from nltk.stem.snowball import RussianStemmer


class Normalizer(object):

    @staticmethod
    def normalizeLetters(word):
        """
        Конвертация ё --> е.
        """
        if 'e' in word:
            return word.replace('ё', 'е')
        return word
```

```
class SentenceSplitter(object):
    """
    Разбиваем отдельные предложения на токены, токены стеммируем. При этом сохраняется структура текста.
    На вход принимаем список предложений, на выходе возвращаем список стемм в виде [[[], [], []], [[], []]],
    где второй уровень вложенности - это абзацы, третий - сами преложения.
    """

    def __init__(self, stopwords):
        self.stopwords = stopwords
        # знаки, которые будут удаляться в начале и конце токена
        self.stemmer = RussianStemmer()
        # объект pymorphy2.MorphAnalyzer(), будем использовать атрибут normal_form
        self.lemmatizer = pymorphy2.MorphAnalyzer()
        self.normalizer = Normalizer()

    def tokenizeString(self, sentence):
        """
        Функция последовательной обработки каждого слова.
        Разбивает полученное на вход предложение на токены по пробелам и слэшам,
        "отрезая" пунктуацию с концов слова и понижая регистр.
        """
        # генератор списка токенов: по циклу: разбиваем строку на токены по пробелам и слэшам,
        # удаляем знаки вокруг токена, приводим к нижнему регистру
        tokens = (self.normalizer.normalizeLetters(token.strip(self.punctuation).lower()) for token in
                  self.splitchars.split(sentence))
        return tokens

    def tokenizeWithCase(self, sentence):
        """
        Такая же функция токенизации, только без приведения слов к нижнему регистру.
        """
        # генератор списка токенов: по циклу: разбиваем строку на токены по пробелам и слэшам,
        # удаляем знаки вокруг токена, приводим к нижнему регистру
        tokens = (self.normalizer.normalizeLetters(token.strip(self.punctuation)) for token in
                  self.splitchars.split(sentence))
        tokens_with_case = [token for token in tokens if token.lower() not in self.stopwords]
        return tokens_with_case

    def stemTokens(self, sentence):
        """
        Функция формирует список стеммированных терминов с удалением стоп-слов.
        Возвращает список кортежей, в которых содержатся стеммы значимых слов и сами слова.
        Это необходимо для последующего извлечения ключевых слов.
        """
```

```
        # генератор списка терминов: если термин не в списке стоп-слов, то стеммируем его.
        stemmed_sentence = ((self.stemmer.stem(self.lemmatizer.parse(term)[0].normal_form),
term) for term in
                            self.tokenizeString(sentence) if term not in self.stopwords)
        if not stemmed_sentence:
            return []
        return stemmed_sentence

    def tokenizeListParagraphs(self, list_of_sentences):
        """
        Получает список предложений, сгруппированных по абзацам.
        Каждое слово из списка стеммированных токенов складывает в новый список с
сохранением структуры абзацев.
        """
        tokenized_sentences = []
        for sentences in list_of_sentences:
            terms_list = []
            for s in sentences:
                terms_in_sentence = []
                for term_pair in self.stemTokens(s):
                    if len(term_pair[0]) > 0:
                        terms_in_sentence.append(term_pair)
                terms_list.append(terms_in_sentence)
            tokenized_sentences.append(terms_list)
        return tokenized_sentences

    def tokenizeListSentences(self, list_of_sentences):
        """
        Получает список предложений (без абзацев).
        """
        tokenized_sentences = []

        for s in list_of_sentences:
            terms_in_sentence = []

            for term_pair in self.stemTokens(s):
                if len(term_pair[0]) > 0:
                    terms_in_sentence.append(term_pair)
            tokenized_sentences.append(terms_in_sentence)
        return tokenized_sentences

    def tokenizeSentencesWithCaseKeeping(self, list_of_sentences):
        """
        Получает список предложений с сохранением регистра (без абзацев).
        """
        tokenized_sentences = []
        for s in list_of_sentences:
            terms_in_sentence = []
            for term in self.tokenizeWithCase(s):
                if len(term) > 0:
                    terms_in_sentence.append(term)
```

```
            tokenized_sentences.append(terms_in_sentence)
        return tokenized_sentences
```

Файл SymmetricalSummarizingClass.py:

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

import itertools
import math
import re
from collections import defaultdict

from SentenceSplitterClass import SentenceSplitter


class FindProperNouns(object):
    @staticmethod
    def lookForProper(lst_of_sents, stopwords):
        """
        Наивный метод поиска имен собственных. Они нужны при добавлении
дополнительных весов предложениям и словам.
        Выбираются все слова с большой буквы, если они стоят не в начале предложения.
        """
        proper_nouns = set()
        sp_object = SentenceSplitter(stopwords)
        s_set = sp_object.tokenizeSentencesWithCaseKeeping(lst_of_sents)
        for s in range(len(s_set)):
            for w in range(len(s_set[s])):
                if s_set[s][w][0].istitle() and w != 0:
                    proper_nouns.add(s_set[s][w])
        stemmed_pnn = set(
            [sp_object.stemmer.stem(sp_object.lemmatizer.parse(pnn.lower())[0].normal_form) for
pnn in proper_nouns])
        return proper_nouns, stemmed_pnn

class CalculateExtraWeights(object):
    """
    Методы класса возвращают списки, которые нужны для подсчета дополнительных
весов словам.
    """

    @staticmethod
    def collectFirstLastSents(lst_of_paragraphs):
        """
        Формируется список первых и последних предложений абзацев.
        """
        first_last_sents = []
        for par in range(len(lst_of_paragraphs)):
            if len(lst_of_paragraphs[par]) > 0:
                first_last_sents.append(lst_of_paragraphs[par][0])
                if len(lst_of_paragraphs[par]) > 1:
```

```
                first_last_sents.append(lst_of_paragraphs[par][len(lst_of_paragraphs[par]) - 1])
        return first_last_sents

    @staticmethod
    def collectQuestionSents(lst_of_sents):
        """
        Формируется список вопросительных и восклицательных предложений.
        """
        lst_of_quest_exclum_sents = []
        for s in range(len(lst_of_sents)):
            if lst_of_sents[s][len(lst_of_sents[s]) - 1] == '?' or lst_of_sents[s][len(lst_of_sents[s]) - 1]
== '!':
                lst_of_quest_exclum_sents.append(lst_of_sents[s])
        return lst_of_quest_exclum_sents


##### TF-IDF #####
class CountTermWeights(object):
    """
    Вычисление tf-idf для терминов текста.
    """

    @staticmethod
    def simpleTermFreqCount(big_lst):
        """
        Метод получает на вход список стемм. В словаре stemfreqs считаются частотности
стемм.
        Подсчитывается общее количество стемм в словаре (total_stems_in_text).
        Вычисляется среднее арифметическое (mean_freq).
        Стеммы с частотностью выше среднего арифметического выбираются в список
termsfreq.
        Список termsfreq - это кортежи с парами (слово, относительная частота)
        [(word1, 0.34646), (word2, 0.46785), (word3, 0.09786)].
        """
        stemfreqs = defaultdict(int)
        for stem in big_lst:
            stemfreqs[stem] += 1
        total_stems_in_text = float(len(big_lst))
        mean_freq = sum(stemfreqs.values()) / float(len(stemfreqs))
        termsfreq = [(word, freq / total_stems_in_text) for word, freq in stemfreqs.items() if freq >=
mean_freq]
        termsfreq_0 = [(word, freq) for word, freq in stemfreqs.items() if freq >= mean_freq]
        return termsfreq, termsfreq_0

    @staticmethod
    def countPureTFIDF(stems):
        """
        На вход функция получает список стемм с частотами для вычисления tf-idf.
        """
        weighted_terms = []
        for st in range(len(stems)):
            weighted_terms.append((stems[st][0], stems[st][1]))
```

```
        sorted_tfidf = sorted(((term, tfidf) for term, tfidf in weighted_terms), key=lambda w: w[1],
reverse=True)
        return sorted_tfidf

    @staticmethod
    def countFinalWeights(tfidf_list, title, lst_of_paragraphs, lst_of_plain_sents,
total_stems_in_text,
                          total_sents_in_text, stopwords, stemmed_pnn):

        """
        Подсчитывается финальный вес стеммы с накруткой дополнительных
коэффициентов.
        """
        weighted_terms2 = []
        weighted_terms3 = []
        weighted_terms4 = []
        weighted_terms5 = []
        # список первых и последних предложений
        collection_of_first_last_sents =
CalculateExtraWeights().collectFirstLastSents(lst_of_paragraphs)
        # список слов первых и последних предложений абзацев
        pairs_collection_of_first_last_sents =
list(itertools.chain.from_iterable(collection_of_first_last_sents))
        stems_collection_of_first_last_sents = [pair[0] for pair in
pairs_collection_of_first_last_sents]
        # количество слов в первых и последних предложениях
        total_stems_in_first_last = len(stems_collection_of_first_last_sents)
        # количество слов из словаря в первых и последних предложениях
        total_dictwords_in_first_last = 0
        for t0 in range(len(tfidf_list)):
            for s1 in stems_collection_of_first_last_sents:
                if tfidf_list[t0][0] == s1:
                    total_dictwords_in_first_last += 1
        # среднее количество слов из словаря в первых и последних предложениях абзацев
        avg_dictwords_in_first_last = total_dictwords_in_first_last / float(total_stems_in_first_last)
        # среднее количество слов в первых и последних предложениях абзацев
        avg_stems_in_first_last = total_stems_in_first_last / float(total_stems_in_text)
        # список вопросительных и восклицательных предложений
        collection_of_q_excl_sents =
CalculateExtraWeights().collectQuestionSents(lst_of_plain_sents)
        # список слов из вопросительных и восклицательных предложений
        sp_object = SentenceSplitter(stopwords)
        pairs_collection_of_q_excl_sents = list(

itertools.chain.from_iterable(sp_object.tokenizeListSentences(collection_of_q_excl_sents)))
        stems_collection_of_q_excl_sents = set([pair[0] for pair in
pairs_collection_of_q_excl_sents])
        # количество вопросительных и восклицательных предложений в тексте
        num_of_q_excl_sents = len(collection_of_q_excl_sents)
        # если термины из словаря есть в заголовке, то удваиваем их вес
        for t1 in range(len(tfidf_list)):
            if tfidf_list[t1][0] in title:
```

```
            m = tfidf_list[t1][1] * 2
            weighted_terms2.append((tfidf_list[t1][0], m))
        else:
            weighted_terms2.append((tfidf_list[t1][0], tfidf_list[t1][1]))
    # если термины есть в первых и последних предложениях абзацев,
    # то вес термина умножаем на частное среднего количества терминов из словаря в первых и последних предложениях
    # и среднего количества терминов в первых и последних предложениях

    for t2 in range(len(weighted_terms2)):
        if weighted_terms2[t2][0] in set(stems_collection_of_first_last_sents):
            m2 = weighted_terms2[t2][1] * (avg_dictwords_in_first_last / avg_stems_in_first_last)
            weighted_terms3.append((weighted_terms2[t2][0], m2))
        else:
            weighted_terms3.append((weighted_terms2[t2][0], weighted_terms2[t2][1]))
    # если термины есть в вопросительных и восклицательных предложениях,
    # то умножаем вес термина на частное от количества таких предложений и общего количества предложений текста
    for t3 in range(len(weighted_terms3)):
        if weighted_terms3[t3][0] in stems_collection_of_q_excl_sents:
            m3 = weighted_terms3[t3][1] * (num_of_q_excl_sents / float(total_sents_in_text))
            weighted_terms4.append((weighted_terms3[t3][0], m3))
        else:
            weighted_terms4.append((weighted_terms3[t3][0], weighted_terms3[t3][1]))
    # если термины из словаря - это имена собственные,
    # то умножаем вес термина на частное среднего количества терминов из словаря в первых и последних предложений
    # и среднего количества терминов в первых и последних предложениях
    for t4 in range(len(weighted_terms4)):
        if weighted_terms4[t4][0] in stemmed_pnn:
            m4 = weighted_terms4[t4][1] * (avg_dictwords_in_first_last / avg_stems_in_first_last)
            weighted_terms5.append((weighted_terms4[t4][0], m4))
        else:
            weighted_terms5.append((weighted_terms4[t4][0], weighted_terms4[t4][1]))
    mean_weight = sum([(weighted_terms5[s][1]) for s in range(len(weighted_terms5))]) / float(len(weighted_terms5))
    sorted_tfidf2 = sorted(((term, weight) for term, weight in weighted_terms5 if weight > mean_weight),
                           key=lambda w: w[1], reverse=True)
    return sorted_tfidf2

##### Symmetrical Summarizing #####
class SymmetricalSummarizationWeightCount(object):
    """
    Проводится начисление весов предложениям по методике симметричного реферирования.
    """

    @staticmethod
    def countTermsInsideSents(sents_lst):
        """
        Метод получает список предложений вида [[sentence1], [sentence2]].
```

```
        Для каждого предложения подсчитывается частота входящих в него стемм.
        Возвращается список,
        содержащий в себе словари стемм с их частотами в каждом предложении [{word1:2,
word2:5}, {word1:5, word2:6}].
        """
        sents_with_termsfreqs = []
        for sentence in sents_lst:
            stem_f = defaultdict(int)

            for pair in sentence:
                stem_f[pair[0]] += 1
            sents_with_termsfreqs.append(stem_f)
        return sents_with_termsfreqs

    @staticmethod
    def rightLinksCount(tfidf_terms, sents_with_termsfreqs):
        """
        Метод производит поиск связей между предложениями вправо.
        Принимает на вход список tf-idf, и список словарей с частотами для каждого
предложения.
        Берется предложение (словарь), если в нем есть термин из tf-idf,
        то ищется вхождение этого термина в предложениях справа.
        Если термин встретился в предложении справа, то выбирается его наибольшая
частота,
        то есть либо из исходного предложения, либо из правого. Эта частота суммируется с
общим весом предложения.
        Последнее предложение скидывается в список с нулевым весом, так как справа от
него ничего нет.
        Возвращается список кортежей,
        в котором предложениям (словарям) приписаны веса [({sentence1}, вес), ({sentence2},
вес)].
        Параллельно для текущего предложения суммируются веса входящих в него
ключевых слов,
        сумма прибавляется к весу предложения. Дополнительно вычисляется позиционный
коэффициент, то есть,
        чем выше предложение, тем больше вес. Тоже прибавляется к общему весу.
        """
        w_sent_r = []
        line = 0
        for s in range(len(sents_with_termsfreqs)):
            c = 0
            cw = 0
            line += 1
            # позиционный коэффициент
            pscore = (1 / line) * 10
            if s != len(sents_with_termsfreqs) - 1:
                slice1 = sents_with_termsfreqs[s + 1:]
                for t in range(len(tfidf_terms)):
                    if tfidf_terms[t][0] in sents_with_termsfreqs[s]:
                        cw += tfidf_terms[t][1]
                        for s2 in range(len(slice1)):
                            if tfidf_terms[t][0] in slice1[s2]:
```

```
                        if sents_with_termsfreqs[s][tfidf_terms[t][0]] > slice1[s2][tfidf_terms[t][0]]:
                            c += sents_with_termsfreqs[s][tfidf_terms[t][0]]
                        else:
                            c += slice1[s2][tfidf_terms[t][0]]

            w_sent_r.append((sents_with_termsfreqs[s], c + cw + pscore))


        else:
            for t in range(len(tfidf_terms)):
                if tfidf_terms[t][0] in sents_with_termsfreqs[len(sents_with_termsfreqs) - 1]:
                    cw += tfidf_terms[t][1]
                else:
                    cw = 0
            w_sent_r.append((sents_with_termsfreqs[s], cw + pscore))
    return w_sent_r

@staticmethod
def leftLinksCount(tfidf_terms, sents_with_termsfreqs):
    """
    Метод производит поиск связей между предложениями влево. Тот же алгоритм, что и при поиске вправо,
    только с нулевым весом скидывается первое предложение. Чтобы реализовать поиск влево,
    список предложений переворачивается. Возвращается список кортежей,
    в котором предложениям (словарям) приписаны веса [({sentence1}, вес), ({sentence2}, вес)].
    """
    w_sent_l = []
    for s in reversed(range(len(sents_with_termsfreqs))):
        c = 0
        slice1 = sents_with_termsfreqs[:s]
        if s != 0:
            for t in range(len(tfidf_terms)):
                if tfidf_terms[t][0] in sents_with_termsfreqs[s]:
                    for s2 in reversed(range(len(slice1))):
                        if tfidf_terms[t][0] in slice1[s2]:
                            if sents_with_termsfreqs[s][tfidf_terms[t][0]] > slice1[s2][tfidf_terms[t][0]]:
                                c += sents_with_termsfreqs[s][tfidf_terms[t][0]]
                            else:
                                c += slice1[s2][tfidf_terms[t][0]]
            w_sent_l.append((sents_with_termsfreqs[s], c))
        else:
            w_sent_l.append((sents_with_termsfreqs[0], 0))
    return w_sent_l

def countSymmetry(self, tfidf_terms, sents_with_termsfreqs):
    """
    Метод складывает два списка, полученных при поиске вправо и влево. Принимает на вход так же список tf-idf,
    и список предложений-словарей. Внутри явно вызываются функции установления правых и левых связей,
```

```
        порядок  предложений в двух списках выравнивается и их веса складываются.
        Возвращается список кортежей предложений-словарей с весами.
        """
        w_sent = []
        right_links = self.rightLinksCount(tfidf_terms, sents_with_termsfreqs)
        left_links = self.leftLinksCount(tfidf_terms, sents_with_termsfreqs)
        for s_r, s_l in zip(range(len(right_links)), reversed(range(len(left_links)))):
            w_sent.append((right_links[s_r][0], right_links[s_r][1] + left_links[s_l][1]))
        return w_sent

    def countFinalSymmetryWeight(self, tfidf_terms, sents_with_termsfreqs, total_stems_in_text,
total_sents_in_text,
                                 stemmed_pnn):
        """
        Метод добавляет к весам предложения дополнительный коэффициент ASL (average
sentence length),
        чтобы длинные предложения не набрали большой вес.
        Принимает на вход список tf-idf, список предложений-словарей,
        общее количество стемм в тексте и общее количество предложений в тексте.
        Явно вычисляется функция countSymmetry(),
        затем asl и в цикле весу каждого предложения добавляется дополнительный
коэффициент.
        Вес первого предложения удваивается. Также каждый вес умножается на количество
имен собственных и цифр.
        """
        w_sent = self.countSymmetry(tfidf_terms, sents_with_termsfreqs)
        w_sent2 = []
        w_sent3 = []
        w_sent4 = []
        # average sentence length
        asl = total_stems_in_text / total_sents_in_text
        # список для хранения частот имен собственных в каждом предложении
        n_p = []
        f_digits = re.compile(r'[0-9]+([.,:][0-9]+)*')
        digits = []
        for s in range(len(w_sent)):
            p = 0
            for pnn in stemmed_pnn:
                if pnn in w_sent[s][0]:
                    p += 1
            n_p.append(p)
        for sd in range(len(w_sent)):
            dig = len(f_digits.findall(' '.join([sk for sk in w_sent[sd][0]])))
            digits.append(dig)
        for s0 in range(len(w_sent)):
            if len(w_sent[s0][0]) > 0:
                if n_p[s0] != 0:
                    score0 = w_sent[s0][1] * (1 + math.log(n_p[s0], 2))
                    w_sent2.append((w_sent[s0][0], score0))

                else:
                    w_sent2.append((w_sent[s0][0], w_sent[s0][1]))
```

```
            else:
                w_sent2.append((w_sent[s0][0], w_sent[s0][1]))
        for s_0 in range(len(w_sent2)):
            if len(w_sent2[s_0][0]) > 0:
                if digits[s_0] != 0:
                    score_0 = w_sent2[s_0][1] * (1 + math.log(digits[s_0], 2))
                    w_sent3.append((w_sent2[s_0][0], score_0))

                else:
                    w_sent3.append((w_sent2[s_0][0], w_sent2[s_0][1]))
            else:
                w_sent3.append((w_sent2[s_0][0], w_sent2[s_0][1]))
        for s1 in range(len(w_sent3)):
            if len(w_sent3[s1][0]) > 5:
                word_count = len(w_sent3[s1][0])
                score = (asl * w_sent3[s1][1]) / word_count
                if s1 != 0:
                    w_sent4.append((w_sent3[s1][0], score))
                else:
                    w_sent4.append((w_sent3[s1][0], score * 1))
            else:
                w_sent4.append((w_sent3[s1][0], w_sent3[s1][1]))
        return w_sent4

    @staticmethod
    def convertSymmetryToOrdinary(symm_weights, ordinary_sents):
        """
        Метод получает на вход список кортежей предложений-словарей с весами и список
оригинальных предложений,
        то есть не подвергшихся токенизации и стеммингу. Из последнего списка
выбираются предложения,
        которые соответствуют словарям с весами. Стоит ограничение на длину
показываемого предложения.
        Она должна быть не меньше 6 (и не больше 50 токенов). Если выходной список
пустой, то берутся все предложения,
        вне зависимости от длины.
        Возвращается отсортированный по убыванию список предложений из оригинального
текста с весом
        и его позицией в тексте [(sentence, weight), (sentence, weight)].
        """
        prefinal_sentences = []
        ordinary_sents_with_freqs = [(ordinary_sents[s], symm_weights[s][1], s) for s in
                                     range(len(symm_weights))]  # if symm_weights[s][1] > mean_weight]
        s_ordinary_sents_with_freqs = sorted(ordinary_sents_with_freqs, key=lambda w: w[1],
reverse=True)
        for sent in range(len(s_ordinary_sents_with_freqs)):
            # if 6 < len(s_ordinary_sents_with_freqs[sent][0].split()) < 50:
            if len(s_ordinary_sents_with_freqs[sent][0].split()) > 6:
                prefinal_sentences.append((s_ordinary_sents_with_freqs[sent][0],
s_ordinary_sents_with_freqs[sent][1],
                                           s_ordinary_sents_with_freqs[sent][2]))
        if len(prefinal_sentences) == 0:
```

```
            for sent in range(len(s_ordinary_sents_with_freqs)):
                prefinal_sentences.append((s_ordinary_sents_with_freqs[sent][0],
s_ordinary_sents_with_freqs[sent][1],
                                           s_ordinary_sents_with_freqs[sent][2]))
        return prefinal_sentences


    ##Процентная выборка
    @staticmethod
    def selectFinalSents(converted_sents, percentage=18):
        """
        Метод выбирает n первых предложений из списка. n определяется указанным
процентом.
        Список сортируется по позиции предложения в оригинальном тексте,
        таким образом возвращается оригинальная последовательность, чтобы хоть как-то
сохранить связность.
        Возвращается список кортежей: [(предложение, вес, порядковый номер), ()]
        """
        compression_rate = int(round(((len(converted_sents) * percentage) / 100) + 0.5))
        salient_sentences = converted_sents[:compression_rate]
        sorted_salient_sentences = sorted(salient_sentences, key=lambda w: w[2])
        return sorted_salient_sentences, compression_rate
```

ПРИЛОЖЕНИЕ Б

Примеры текстов и рефератов

Текст 1:

«Вследствие удаленности от Солнца, климат планеты Марс гораздо суровее земного. Температура на поверхности обычно в среднем достигает -60 градусов Цельсия и может опускаться до -123 градусов Цельсия около полюсов. Только в полдень солнце на тропических широтах достаточно теплое для таяния льда, но любая вода в жидком состоянии испаряется почти мгновенно из-за низкого атмосферного давления.

Хотя в атмосфере содержится небольшое количество воды, и над пустынями Марса образуются облака мелкой пыли, в большинстве случаев сильные ветра смешивают пылевые облака с белыми водяными. Каждую зиму, например, снежная буря углекислого газа поднимается у одного полюса, и в то время как накапливается несколько метров этого снега из сухого льда уже замороженный углекислый газ испаряется на противоположном полюсе. Даже на летнем полюсе, где солнце стоит почти весь день, температура никогда не поднимается до температуры таяния льда.»

Реферат, составленный Microsoft AutoSummarize:

«Температура на поверхности обычно в среднем достигает -60 градусов Цельсия и может опускаться до -123 градусов Цельсия около полюсов. Только в полдень солнце на тропических широтах достаточно теплое для таяния льда, но любая вода в жидком состоянии испаряется почти мгновенно из-за низкого атмосферного давления.»

Реферат, составленный TextAnalyst:

«Только в полдень солнце на тропических широтах достаточно теплое для таяния льда, но любая вода в жидком состоянии испаряется почти мгновенно из-за низкого атмосферного давления. Даже на летнем полюсе, где солнце стоит почти весь день, температура никогда не поднимается до температуры таяния льда.»

Реферат, составленный разработанной системой:

«Вследствие удаленности от Солнца, климат планеты Марс гораздо суровее земного. Температура на поверхности обычно в среднем достигает -60 градусов Цельсия и может опускаться до -123 градусов Цельсия около полюсов.»

Текст 2:

«Смарт-карты становятся все более популярными в последнее время, так как цена на устройства хранения информации стремительно падает. Они имеют два главных преимущества над дискетами. Во-первых, они могут хранить в 100 раз больше информации – и сохранять ее намного надежнее. Во-вторых, они могут выполнять сложные задачи по командам с терминала. К примеру, смарт-карта может проверить ответы на заранее известные вопросы с целью проверить информацию, сохраненную на карте. Карта, использующая данный алгоритм, может оповестить терминал о том, что владелец имеет достаточно средств для оплаты услуги без указания номера счета. В зависимости от важности информации, для безопасности используется персональный идентификационный номер, например, как в банкоматах.

Смарт-карты не являются новым открытием. Они разрабатывались с начала 70-х годов и нашли много применений в Европе, с тех пор было выпущено более четверти миллиарда таких карт. Подавляющее большинство чипов ушло по предоплате, но даже при этих условиях был получен опыт, который впоследствии уменьшил производственные издержки, улучшил надежность и доказал значимость этих смарт-карт. Международные и национальные стандарты для смарт-карт также находятся в стадии разработки для обеспечения уверенности в том, что карты, карт-ридеры и программное обеспечение могут работать вместе надежно и безопасно. Стандарты, установленные Международной Организацией по Стандартам, к примеру, указывают на местоположение контактов на лицевой стороне смарт-карты, чтобы любая карта могла быть соединена с любым карт-ридером.»

Реферат, составленный Microsoft AutoSummarize:

«К примеру, смарт-карта может проверить ответы на заранее известные вопросы с целью проверить информацию, сохраненную на карте. Международные и национальные стандарты для смарт-карт также находятся в стадии разработки для обеспечения уверенности в том, что карты, карт-ридеры и программное обеспечение могут работать вместе надежно и безопасно.»

Реферат, составленный TextAnalyst:

«Международные и национальные стандарты для смарт-карт также находятся в стадии разработки для обеспечения уверенности в том, что карты, карт-ридеры и программное обеспечение могут работать вместе надежно и безопасно. Стандарты, установленные Международной Организацией по Стандартам, к примеру, указывают на местоположение контактов на лицевой стороне смарт-карты, чтобы любая карта могла быть соединена с любым карт-ридером.»

Реферат, составленный разработанной системой:

«Смарт-карты становятся все более популярными в последнее время, так как цена на устройства хранения информации стремительно падает. Они имеют два главных преимущества над дискетами. Смарт-карты не являются новым открытием.»

Текст 3:

«Электронная информация играет все большую роль во всех сферах жизни современного общества. В последние годы объем научно-технической текстовой информации в электронном виде возрос настолько, что возникает угроза обесценивания этой информации в связи с трудностями поиска необходимых сведений среди множества доступных текстов. Развитие информационных ресурсов Интернет многократно усугубило проблему информационной перегрузки. В этой ситуации особенно актуальными становятся методы автоматизации реферирования текстовой информации, то

есть методы получения сжатого представления текстовых документов – рефератов (аннотаций).

Постановка проблемы автоматического реферирования текста и соответственно попытки ее решения с использованием различных подходов предпринимались многими исследователями. История применения вычислительной техники для реферирования насчитывает уже более 50 лет и связана с именами таких исследователей, как Г.П. Лун, В.Е. Берзон, И.П. Севбо, Э.Ф. Скороходько, Д.Г. Лахути, Р.Г. Пиотровский и др. За эти годы выработаны многочисленные подходы к решению данной проблемы, которые достаточно четко подразделяются на два направления: автоматическое реферирование, основанное на экстрагировании из первичных документов с помощью определенных формальных признаков «наиболее информативных» фраз (фрагментов), совокупность которых образует некоторый экстракт; и автоматическое реферирование, основанное на выделении из текстов с помощью специальных информационных языков наиболее существенной информации и порождении новых текстов (рефератов), содержательно обобщающих первичные документы.

В России исследования в области автоматического реферирования в настоящее время, главным образом, ведутся в рамках первого направления с использованием статистических методов, смысл которых заключается в отборе предложений с наибольшим весом, который рассчитывается на основе частоты появления слова в тексте или месторасположения предложения, для включения их в реферат. В настоящее время известны только две системы, позволяющие получать аннотации на русском языке: TextAnalyst и встроенная функция в пакете Microsoft Office – Autosummarize. Обе эти системы относятся к классу систем, использующих различные варианты статистических методов. Согласно исследованиям в области компьютерной лингвистики текст, по своей природе, нелинеен, и его структура определяется особенностями внутренней организации единиц текста и закономерностями взаимосвязи этих единиц в рамках текста как цельного сообщения. Как показала практика,

различные статистические методы недостаточно эффективны, так как они интерпретируют текст в виде набора линейно упорядоченных слов, словосочетаний и предложений, игнорируя при этом лингвистическую взаимосвязанность естественного языка, что приводит к потере значимой информации.

Исследования в области автоматической обработки текстов в Европе и США привлекают внимание крупнейших частных фирм и государственных организаций самого высокого уровня. Существует большое количество систем, разработанных, в основном, специалистами университетских центров и используемых ими для своих нужд. В этих системах предлагаются нетрадиционные решения (отличные от статистических методов), основанные на построении лексических цепочек, концептуальных графов, а также эффективных формализмов описания структуры текста. Однако все эти методы ориентированы на учет особенностей конкретных языков, в основном, английского языка, и не могут быть непосредственно применены для автоматического реферирования текстов на русском языке. Кроме того, большинство разработок носят коммерческий характер, в связи с чем принцип их работы авторами не раскрывается.

Таким образом, актуальным является создание новых эффективных методов и алгоритмов, учитывающих нелинейную и иерархическую природу текста и позволяющих получать сжатое представление текстовых документов на русском языке.»

Реферат, составленный Microsoft AutoSummarize:

«В России исследования в области автоматического реферирования в настоящее время главным образом ведутся в рамках первого направления с использованием статистических методов, смысл которых заключается в отборе предложений с наибольшим весом, который рассчитывается на основе частоты появления слова в тексте или на месторасположении предложения, для включения их в реферат. Исследования в области автоматической

обработки текстов в Европе и США привлекают внимание крупнейших частных фирм и государственных организаций самого высокого уровня.»

Реферат, составленный TextAnalyst:

«В этой ситуации особенно актуальными становятся методы автоматизации реферирования текстовой информации, то есть – методы получения сжатого представления текстовых документов – рефератов, или аннотаций. Постановка проблемы автоматического реферирования текста и соответственно попытки ее решения с использованием различных подходов предпринимались многими исследователями.»

Реферат, составленный разработанной системой:

«В этой ситуации особенно актуальными становятся методы автоматизации реферирования текстовой информации, то есть методы получения сжатого представления текстовых документов – рефератов (аннотаций). В настоящее время известны только две системы, позволяющие получать аннотации на русском языке: TextAnalyst и встроенная функция в пакете Microsoft Office – Autosummarize. Исследования в области автоматической обработки текстов в Европе и США привлекают внимание крупнейших частных фирм и государственных организаций самого высокого уровня. Однако все эти методы ориентированы на учет особенностей конкретных языков, в основном, английского языка, и не могут быть непосредственно применены для автоматического реферирования текстов на русском языке.»

Текст 4:

«Развитие Web-технологий обеспечивает расширение области применения CAD-систем. Эти технологии являются качественным средством коммуникации. Характеризуются двунаправленной интерактивностью, т.е. представляют возможность для двух или более инженеров общаться и объединять свои усилия при конструировании в режиме реального времени. Возможность общения в реальном масштабе времени и манипулирование

конструкторскими и геометрическими данными CAD-системы посредством Web-технологий требуют значительных технических усовершенствований, не только в направлении повышения производительности системы обработки данных, но и коренного изменения в организации процесса проектирования. В данной статье мы рассмотрим распределенные архитектуры, охарактеризуем их плюсы и минусы и остановимся на оптимальной системе для решения нашей задачи.

На предприятии, клиентских рабочих мест всегда во много раз больше, чем серверных или инфраструктурных компонентов корпоративных информационных систем. Поэтому каждый лишний рубль, затраченный на одного клиента, в масштабах большой компании оборачивается большими финансовыми суммами. Вот почему важно точно представлять себе, во что каждый клиент обходится. Данная величина называется полной стоимостью владения (Total Cost of Ownership, TCO). Начальные затраты составляют относительно небольшую долю. Главное – повседневное администрирование, выполняемое штатным системным администратором.

Есть несколько методов для борьбы с толщиной клиентов – стремление уменьшить полную стоимость владения информационной системой, ограничением конфигурационных возможностей на рабочих местах, делая клиентов тонкими, возлагая при этом возможности на сервер. Другим методом является проблема распространения нового программного обеспечения (ПО) или замена ПО, содержащего ошибки. Добиться заметных успехов по совершенствованию этого процесса на классических клиент-серверных системах не удается. Тонкий клиент не хранит у себя ничего − ни данных, ни прикладного ПО, поэтому, смена версий ПО может быть выполнена в течение нескольких минут, просто путем замены соответствующих файлов на сервере.

В настоящее время процессу старения больше подвержено ПО, нежели аппаратная часть компьютера, т.е. новое ПО не работает на старых компьютерах. У тонких клиентов меньше компонентов, подвергающихся такому старению, ведь основная функция у тонких клиентов – интерфейсная.

Если мы перемещаем какие-либо функции рабочего места, то эти функции должны осесть на серверах, создавая дополнительную нагрузку на их аппаратуру и усложняя их администрирование. Хорошо если серверы способны выдержать такие нагрузки без заметного снижения производительности, но если мощности серверов не хватает, постоянно перегружается сеть, зачастую идут на компромисс, «утолщая» клиентов и передавая им соответствующие функции.

Повышение производительности сервера позволит эффективнее использовать ресурсы за счет многопользовательской эксплуатации. Действительно, можно утверждать, что 95% времени персональный компьютер используется на 5%, имея ярко выраженный пиковый характер загрузки. Если встает вопрос повышения производительности, то надо увеличить ресурсы сервера на 50% вместо наращивания ресурсов 50-ти клиентов по 20% на каждого. Данное решение может привести к суммарной экономии более чем на 25%.

Для того чтобы создать действительно многозвенное приложения в качестве промежуточных звеньев задействуются Web-серверы и браузеры. Причем последние все чаще используются для реализации стандартного пользовательского интерфейса. Для поддержки «тонкого» клиента в лице браузера (средства просмотра гипертекстовых страниц), необходимо наличие таких компонентов как Web сервер и сервер приложений. Первый обеспечивает доступ к информационным ресурсам Internet/Intranet, второй − необходимую предварительную обработку данных .

Клиент-серверная архитектура САПР предполагает распределение задач и вычислительной нагрузки по нескольким компьютерам, объединенным в сеть. Программное обеспечение САПР-системы, реализованной в этой архитектуре, логически разделяется на серверное ПО, ПО среднего уровня и ПО клиентских рабочих мест (клиентов). Если в задачи разрабатываемой системы проектирования не входит обеспечение сложных вычислений и многочисленных логических правил, функции среднего уровня могут быть

распределены между сервером и клиентом, что приводит к увеличению быстродействия и уменьшению сетевого трафика. Таким образом, получается двухуровневая архитектура, а клиент, на котором лежит часть функций среднего уровня, называется «толстым».

Серверный и промежуточный модули обычно работают на отдельных немногочисленных производительных серверах и поэтому они легко поддаются настройке и обслуживанию. Клиентское ПО в силу своей специфики должно стоять на каждом рабочем месте. Поэтому с ростом числа рабочих станций подключенных к данной системе, растут затраты на настройку клиентских приложений и стоимость самих компьютеров становятся преобладающими в общих затратах. Естественно, возникает желание упростить клиентское ПО сделать его по возможности «тонким» и установить его на более слабый компьютер. Наиболее «тонкие» клиенты могут быть разработаны при использовании технологии Internet/Intranet. В этом случае клиентское рабочее место может иметь только браузер для просмотра web-страниц.

Определение, каким должен быть клиент «тонким» или «толстым», какую выбрать архитектуру – двух, трехуровневую или Web-центричную, зависит от конкретных задач, для решения которых разрабатывается система проектирования, от предъявляемых к ней требований и от бюджета проекта.»

Реферат, составленный Microsoft AutoSummarize:

«Возможность общения в реальном масштабе времени и манипулирование конструкторскими и геометрическими данными CAD-системы посредством Web-технологий требуют значительных технических усовершенствований, не только в направлении повышения производительности системы обработки данных, но и коренного изменения в организации процесса проектирования. В данной статье мы рассмотрим распределенные архитектуры, охарактеризуем их плюсы и минусы и остановимся на оптимальной системе для решения нашей задачи.»

Реферат, составленный TextAnalyst:

«Хорошо если серверы способны выдержать такие нагрузки без заметного снижения производительности, но если мощности серверов не хватает, постоянно перегружается сеть, зачастую идут на компромисс, «утолщая» клиентов и передавая им соответствующие функции. Клиент-серверная архитектура САПР предполагает распределение задач и вычислительной нагрузки по нескольким компьютерам, объединенным в сеть. Программное обеспечение САПР-системы, реализованной в этой архитектуре, логически разделяется на серверное ПО, ПО среднего уровня и ПО клиентских рабочих мест (клиентов).»

Реферат, составленный разработанной системой:

«Развитие Web-технологий обеспечивает расширение области применения CAD-систем. На предприятии, клиентских рабочих мест всегда во много раз больше, чем серверных или инфраструктурных компонентов корпоративных информационных систем. Есть несколько методов для борьбы с толщиной клиентов – стремление уменьшить полную стоимость владения информационной системой, ограничением конфигурационных возможностей на рабочих местах, делая клиентов тонкими, возлагая при этом возможности на сервер. У тонких клиентов меньше компонентов, подвергающихся такому старению, ведь основная функция у тонких клиентов – интерфейсная. Если встает вопрос повышения производительности, то надо увеличить ресурсы сервера на 50% вместо наращивания ресурсов 50-ти клиентов по 20% на каждого. Для поддержки «тонкого» клиента в лице браузера (средства просмотра гипертекстовых страниц), необходимо наличие таких компонентов как Web сервер и сервер приложений. Наиболее «тонкие» клиенты могут быть разработаны при использовании технологии Internet/Intranet.»

Текст 5:

«В настоящее время происходит насыщение рынка информационных услуг, интенсивные процессы развития сетей голосовой связи и передачи

данных, появление и развитие широкого спектра новых услуг связи (IP-телефония, Интернет, мультимедийные технологии, электронная коммерция и т.д.). Такое развитие в первую очередь способствует удовлетворению стратегических интересов государства направленных на глубокие структурные преобразования телекоммуникационной сферы, увеличению доходности и повышению ликвидности акций базовых операторов связи, что должно способствовать росту их капитализации, увеличению рыночной стоимости, а значит - инвестиционной привлекательности для отечественного и зарубежного крупного бизнеса. Однако внедрение новых технологий неизбежно связано с реорганизацией и реструктуризацией корпоративных моделей ведения бизнеса и управления предприятием, нацеленных на быстрое изменение структурно-организационных форм и принципов построения центральных и региональных подразделений межрегиональных компаний, на разработку и оперативное внедрение новых, более эффективных бизнес-процессов. Связь и телекоммуникации на сегодняшний день являются наиболее бурно развивающейся отраслью отечественной экономики. Очевидно, что столь крупномасштабные преобразования в отрасли не могут не сопровождаться радикальными шагами в направлении создания мощной платформы в виде новейших информационных технологий, необходимой для поддержки вышеуказанных тенденций. Надежную ИТ-платформу для крупного бизнеса возможно обеспечить только путем создания корпоративных информационных систем.

Создание и внедрение корпоративных информационных систем для географически распределённых компаний решает не только проблему формирования единого корпоративного информационного пространства, но и обеспечивает реализацию комплексного подхода к решению в рамках компаний задач ресурсного снабжения, планирования и производства товаров и услуг, организации сбыта, маркетинга и финансового учета. Корпоративная информационная система служит также эффективным инструментом для

выработки и проведения в жизнь согласованной общекорпоративной стратегии и тактики управления всеми предприятиями, входящими в состав корпорации.

Корпоративные информационные системы сейчас рассматриваются как неотъемлемая, важная составляющая современного бизнеса. Невозможно представить себе устойчиво функционирующее крупное предприятие без средств поддержки его основных бизнес-функций в виде мощного центра информационных технологий. Каждое подразделение подобных корпораций имеет в своем распоряжении соответствующую информационную инфраструктуру, например, для поддержки управления, учета материальных ресурсов, биллинговых систем. Поэтому наиболее логичным решением, является необходимость комплексного подхода к системному проектированию архитектуры корпоративных информационных систем. Для решения данной проблемы необходима глобальная переоценка места и значимости информационных технологий в структуре современных компаний корпоративного масштаба.

В основу новых концептуальных представлений должны быть положены как можно более формализованные математические модели и методы анализа и синтеза проектных решений.

Прежде всего, речь идет о базовых архитектурных принципах, на которых основано создание и развитие сложных технических систем. В них используются элементы системного анализа, системотехники, методов оптимизации и теории иерархических многоуровневых систем. Вместе с тем - используя известное положение о том, что наиболее продуктивным является совместное проектирование объекта управления и системы управления им - следует также принимать во внимание ранее разработанные подходы к концептуальному проектированию систем организационного управления, в т.ч. с использованием моделей и методов оптимального управления и поддержки принятия решений. Здесь предстоит решать сложные проблемы создания библиотек системных методов, технологий и приемов

проектирования, а также синтеза базовых моделей разрабатываемых объектов, предназначенных для анализа проектных решений и проведения исследований в рамках заданной предметной области.

Необходимо разработать САПР, обладающую не только возможностями проектирования, но и способную учитывая тенденцию к глобализации процессов проектирования и географическую распределенность современных предприятий, разработать систему, основанную на распределенной архитектуре.

В основе многих современных представлений об архитектуре корпоративных информационных систем лежат несколько принципиальных базовых моделей, среди которых в первую очередь следует отметить модели Д.Хендерсона и Дж.Захмана.

Одной из первых попыток связать архитектуру информационной системы с реальной производственной средой и обозначить их тесное взаимовлияние явилась модель Хендерсона.

Поскольку модель Хендерсона носила в известной степени концептуально абстрактный характер, то в среде разработчиков, занимающихся проблемами проектирования корпоративных информационных систем, большую популярность получила модель Дж.Захман. В данной модели сочетаются простота и концептуальность представлений об общей архитектуре информационных систем, аспектах их реализации, видах обеспечений, взглядах пользователей и разработчиков системы. Эта модель показывает, что построение информационной системы современного крупного предприятия следует начинать с рассмотрения самых главных содержательных аспектов его деятельности. Изучение этих аспектов должно сопровождаться формализацией их представления в виде некоторой схемы, понятной для всех участников процесса разработки информационных систем. Далее эти явно изложенные описания деятельности предприятия должны быть тем или иным образом преобразованы в спецификации проектируемой системы. В силу простоты и наглядности данную модель

целесообразно принять в качестве основной для дальнейшего изложения проблем и задач системного проектирования информационных систем.

Существуют различные сценарии системного проектирования корпоративных информационных систем, среди которых есть успешные попытки решить некоторые из обозначенного комплекса проблем. Одним из примеров могут служить методологии, в основу которых положены хорошо известные идеи и модели Дж.Захмана, Д.Хендерсона и других. Строго сформулированные методологические рекомендации по построению систем масштаба предприятия можно найти также в работах отечественных авторов (Е.Зиндера, Е.Ойхмана, Б.Позина и других).

Исходя из этого, одной из перспектив развития данной области является устранение методологических пробелов в рассмотрении систем проектирования корпоративных информационных систем. В связи с увеличением динамики развития, обусловленной ускоренной сменой платформ и архитектур в области информационных систем. Это обстоятельство объясняется тем, что в современных условиях необходимы и важны не только оптимальное использование имеющихся ресурсов и повышение производительности труда, но и высокая степень управляемости, выражающаяся в гибкости и быстроте реагирования на изменение внешней ситуации при ориентации на постоянное активное взаимодействие с потребителями товаров и услуг.»

Реферат, составленный Microsoft AutoSummarize:

«Создание и внедрение корпоративных информационных систем для географически распределённых компаний решает не только проблему формирования единого корпоративного информационного пространства, но и обеспечивает реализацию комплексного подхода к решению в рамках компаний задач ресурсного снабжения, планирования и производства товаров и услуг, организации сбыта, маркетинга и финансового учета. Корпоративная информационная система служит также эффективным инструментом для выработки и проведения в жизнь согласованной общекорпоративной

стратегии и тактики управления всеми предприятиями, входящими в состав корпорации. В основу новых концептуальных представлений должны быть положены как можно более формализованные математические модели и методы анализа и синтеза проектных решений. В них используются элементы системного анализа, системотехники, методов оптимизации и теории иерархических многоуровневых систем. В основе многих современных представлений об архитектуре корпоративных информационных систем лежат несколько принципиальных базовых моделей, среди которых в первую очередь следует отметить модели Д.Хендерсона и Дж.Захмана.»

Реферат, составленный TextAnalyst:

«Однако внедрение новых технологий неизбежно связано с реорганизацией и реструктуризацией корпоративных моделей ведения бизнеса и управления предприятием, нацеленных на быстрое изменение структурно-организационных форм и принципов построения центральных и региональных подразделений межрегиональных компаний, на разработку и оперативное внедрение новых, более эффективных бизнес-процессов. Корпоративная информационная система служит также эффективным инструментом для выработки и проведения в жизнь согласованной общекорпоративной стратегии и тактики управления всеми предприятиями, входящими в состав корпорации. Поэтому наиболее логичным решением, является необходимость комплексного подхода к системному проектированию архитектуры корпоративных информационных систем. В основе многих современных представлений об архитектуре корпоративных информационных систем лежат несколько принципиальных базовых моделей, среди которых в первую очередь следует отметить модели Д.Хендерсона и Дж.»

Реферат, составленный разработанной системой:

«Надежную ИТ-платформу для крупного бизнеса возможно обеспечить только путем создания корпоративных информационных систем. Корпоративные информационные системы сейчас рассматриваются как

неотъемлемая, важная составляющая современного бизнеса. Поэтому наиболее логичным решением, является необходимость комплексного подхода к системному проектированию архитектуры корпоративных информационных систем. В основе многих современных представлений об архитектуре корпоративных информационных систем лежат несколько принципиальных базовых моделей, среди которых в первую очередь следует отметить модели Д.Хендерсона и Дж.Захмана. Поскольку модель Хендерсона носила в известной степени концептуально абстрактный характер, то в среде разработчиков, занимающихся проблемами проектирования корпоративных информационных систем, большую популярность получила модель Дж.Захман. Исходя из этого, одной из перспектив развития данной области является устранение методологических пробелов в рассмотрении систем проектирования корпоративных информационных систем.»