МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ПРИЛОЖЕНИЯ KVAZARONLINE С ВЫЧИСЛИТЕЛЬНЫМ МОДУЛЕМ НА C++

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Быковой Марии Дмитриевны

Научный руководитель
к. ф.-м. н _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2025

ВВЕДЕНИЕ

В современном мире объем цифровой информации растет очень быстро. Организации всех уровней — от государственных учреждений до коммерческих предприятий — сталкиваются с проблемой эффективного поиска и извлечения релевантной информации из огромных массивов неструктурированных данных.

Целью настоящей работы является реализация интеллектуальной поисковой системы, способной автоматически обрабатывать документы различных форматов, структурировать их содержимое, обеспечивать точный семантический поиск с использованием современных технологий обработки естественного языка и машинного обучения.

В рамках работы решаются следующие задачи:

1. Исследовать современные системы поиска информации в документах, определить оптимальные подходы для реализации поставленной цели.
2. Создать модуль для извлечения текста из документов различных форматов с последующим автоматическим разделением на смысловые статьи на основе структурных элементов документа.
3. Разработать алгоритм обработки естественно-языковых запросов пользователя.
4. Реализовать механизм обработки естественного языка.
5. Разработать интуитивно понятный веб-интерфейс для загрузки документов, создания запросов и просмотра результатов поиска.
6. Провести комплексное тестирование системы на реальных данных, оценить точность и полноту поиска.

Структура и объём работы. Выпускная квалификационная работа состоит из введения, 2 разделов, заключения, списка использованных источников и 6 приложений. Общий объем работы — 73 страница, из них 58 страниц — основное содержание, включая 8 рисунков, USB-носитель в качестве приложения, список использованных источников информации — 22 наименования.

Первый раздел «Понятие поисковой системы» посвящен описанию основных определений, используемых в работе, а также теоретической информации, касающейся технологического стека и структуры вычислительного модуля Kvazar.

Второй раздел «Разработка поисковой системы» посвящен детальному описанию кодовой базы инструментов разработки.

Понятие поисковой системы

Понятие информационного поиска

Информационный поиск (Information Retrieval, IR) — это процесс, заключающийся в нахождении и извлечении релевантных данных из обширных массивов информации. Эти данные, как правило, хранятся в цифровом формате в памяти компьютеров или распределённых базах данных.

Ключевая задача информационного поиска — это удовлетворение информационных потребностей пользователя, то есть обеспечение доступа к тем данным, которые соответствуют запросу по смыслу, содержанию или контексту. Для этого применяются различные методы и алгоритмы, включая индексирование, ранжирование, машинное обучение и обработку естественного языка (NLP).

Проблематика IR

Современные системы поиска информации в документах сталкиваются с рядом существенных проблем, которые значительно усложняют процесс эффективного поиска и извлечения релевантных данных. Эти проблемы носят комплексный характер и требуют разработки специализированных подходов для их решения. Основные трудности, существующие в данной области, могут быть сформулированы следующим образом:

- Разнородность форматов документов.
- Отсутствие единой структуры.
- Ограничения традиционного поиска.
- Проблема обработки больших объемов данных.
- Естественно-языковые запросы.

Теория индексирования

Поисковый индекс — это специализированная структура данных, обеспечивающая быстрый доступ к статьям документа по содержащимся в них словам. В отличие от традиционных баз данных, оптимизированных для произвольного доступа, поисковые индексы создаются для решения одной задачи — максимально быстрого нахождения релевантных статей документов по их текстовому содержимому.

- токенизация — разбиение текста на отдельные слова (токены) с учетом языковых особенностей.

— нормализация — приведение слов к единой форме: приведение к нижнему регистру, удаление диакритических знаков (надстрочные, подстрочные, внутристрочные знаки, используемые для уточнения значения буквенных сочетаний), выделение основы слова.
— построение словаря — создание списка уникальных терминов в коллекции статей.
— создание постлистов — для каждого термина из словаря формируется список статей, в которых он встречается, с указанием информации о позиции в статье и частоте встречаемости.

Семантические аспекты поиска. Работа с синонимами

Автоматическое выявление синонимичных и семантически близких терминов представляет собой сложную задачу, для которой на сегодняшний день разработано несколько эффективных методов, каждый из которых обладает своими преимуществами и ограничениями, а их выбор зависит от конкретной предметной области, объема данных и требуемой точности:

1. Анализ совместной встречаемости — данный метод основывается на предположении, что если два слова часто появляются в одних и тех же документах, статьях или располагаются рядом в тексте, то между ними существует семантическая связь.
2. Использование тезаурусов и онтологий — данный подход предполагает применение существующих словарей синонимов, таких как WordNet, BabelNet и других лингвистических ресурсов, в которых зафиксированы семантические отношения между словами.
3. Векторные модели слов (word2vec, GloVe) — этот метод основан на идее перевода слов в многомерные векторные пространства, где семантически близкие термины располагаются на небольшом расстоянии друг от друга.
4. Контекстные языковые модели (BERT, ELMo) — это более современный и продвинутый подход, который учитывает не только общую семантику слов, но и их конкретное значение в зависимости от контекста употребления.

Оценка качества поиска

Все существующие метрики оценки качества поиска можно разделить на три большие группы, каждая из которых отражает определенный аспект

эффективности поисковой системы:

1. Точность (Precision) — это одна из ключевых метрик, которая определяется как доля релевантных документов среди всех документов, возвращенных поисковой системой в ответ на пользовательский запрос.
2. Полнота (Recall) — метрика, которая показывает, какая доля релевантных документов из всей коллекции была успешно найдена и возвращена поисковой системой.
3. F-мера — это комплексный показатель, представляющий собой гармоническое среднее между точностью и полнотой.

Разработка поисковой системы

Загрузка и обработка документов

Процесс обработки документов можно разделить на несколько ключевых этапов:

- загрузка и валидация загружаемых файлов;
- извлечение текстового содержимого с сохранением структурных элементов документа;
- анализ и нормализация текста;
- разделение текста на смысловые блоки и формирование статей.

Модуль загрузки документов спроектирован с соблюдением принципов RESTful архитектуры, что обеспечивает простоту интеграции с клиентским приложением. Основной точкой входа в систему является REST-контроллер, реализующий API для приёма файлов.

Класс DocumentUploadController является основным компонентом модуля и реализует логику обработки входящих запросов.

Аннотация RestController указывает, что данный класс является контроллером Spring MVC, а возвращаемые им значения должны быть записаны непосредственно в тело HTTP-ответа. Аннотация RequestMapping задаёт базовый путь для всех методов контроллера. Конструктор контроллера использует механизм внедрения зависимостей Spring (Dependency Injection) для получения экземпляров сервисов обработки документов и преобразования в JSON. Такой подход делает код более тестируемым и упрощает возможную модификацию системы в будущем.

Основной метод контроллера uploadDocuments обрабатывает POST-запросы по пути, который задаётся аннотацией PostMapping и принимает массив файлов в параметре files. Spring автоматически связывает этот параметр с данными multipart-формы, отправленной клиентом.

После успешной валидации файлы передаются в сервис обработки документов DocumentProcessingService. Метод processDocuments сервиса выполняет извлечение и стуктурирование содержимого документов. В случае успешной обработки возвращается список ProcessedDocument, каждый из которых содержит информацию об исходном файле и выделенных из него статьях. Полученные данные преобразуются в JSON-формат с помощью JsonConverter.

В случае успеха возвращается ответ с кодом состояния 200 (OK), который

содержит JSON-представление обработанных документов.

Весь код обработки документов заключён в блок try-catch для перехвата возможных исключений. При возникновении любой ошибки (например, проблемы с чтением файла или его форматом) клиент получит ответ с кодом 500 (Internal Server Error) и сообщением, содержащим описание ошибки.

Анализ структуры документа. Деление докумета на статьи

Для анализа структуры документа реализован класс DocumentStructureAnalyzer. Он решает комплексную задачу преобразования сплошного текста документа в набор структурированных статей:

1. выявление структурных элементов документа:
   - заголовки разделов,
   - границы смысловых блоков,
   - особые элементы (списки, таблицы);
2. разделение текста на статьи:
   - определение границ статей,
   - связывание заголовков с соответствующим текстом,
   - обработка документов без чёткой структуры;
3. формирование метаданных статей:
   - извлечение первого предложения,
   - сохранение оригинального имени файла,
   - учёт порядка документа в списке загрузки.

Для анализа текста используются регулярные выражения и эвристические алгоритмы. Данный шаблон распознаёт нумерованные заголовки, главы и разделы. Флаг MULTILINE позволяет искать совпадения в каждой строке текста отдельно.

Процесс поиска заголовков и разделение текста релизован по следующему алгоритму:

1. находим все заголовки в тексте;
2. сохраняем текст между заголовками как отдельные разделы;
3. добавляет текст после последнего заголовка.

Если документ не содержит заголовков, весь текст становится одной статьёй с названием файла. В противном случае для каждого раздела создаётся отдельная статья.

Реализация системы информационного поиска

Основные компоненты поискового модуля:

- Модуль анализа и обработки текста (MyAnalyzer) — отвечает за нормализацию и обработку текста перед индексированием и поиском.
- Поисковый движок (Searcher) — ядро системы, реализующее функции индексирования и поиска документов.
- Инициализатор индекса (SearchIndexInitializer) — управляет созданием и обновлением поискового индекса.
- Модель поискового запроса (SearchQuery) — инкапсулирует параметры поиска.
- Диалоговые интерфейсы (SearchDialogue и т. д.) — обеспечивают взаимодействие с пользователем.
- Вспомогательные утилиты (DocumentParser, DocumentPrinter) — выполняют дополнительные функции обработки документов.

Последовательность обработки запроса в инофрмационной системе:

- клиент загружает документы через DocumentUploadController;
- контроллер делегирует обработку DocumentProcessingService;
- сервис использует TikaParser для извлечения текста;
- DocumentStructureAnalyzer выделяет статьи из текста;
- результаты преобразуются в JSON и сохраняются;
- SearchIndexInitializer обновляет поисковый индекс;
- пользователь выполняет поиск через SearchDialogue;
- Searcher обрабатывает запрос и возвращает результаты.

Класс MyAnalyzer является наследником стандартного Analyzer из библиотеки Apache Lucene и реализует цепочку обработки текста, включая такие этапы как:

- токенизация — разбиение текста на слова;
- приведение к нижнему регистру;
- стемминг — поиск основы слова;
- генерация shingle (последовательностей из нескольких слов);
- замена слов на их синонимы;

Анализатор используется как при индексировании документов, так и при обработке поисковых запросов, что обеспечивает согласованность результатов и повышает их релевантность.

Метод createComponents определяет цепочку обработчиков токенов. Каждый фильтр в цепочке выполняет определённые преобразования:

- WhitespaceTokenizer — разбивает текст на токены по пробельным символам
- LowerCaseFilter — приводит все токены к нижнему регистру
- RussianLightStemFilter — применяет стемминг для русского языка
- hingleFilter — создаёт биграммы (последовательности из 2 токенов)
- SynonymGraphFilter — заменяет слова на их синонимы согласно загруженному словарю

Для корректной обработки русскоязычных текстов в анализаторе применяется RussianLightStemFilter, который выполняет нормализацию окончаний слов, приведение слов к их основе, учёт морфологических особенностей русского языка. Это позволяет находить документы даже при неполном совпадении словоформ в запросе и документах.

Класс Searcher инкапсулирует всю логику работы с поисковым индексом и предоставляет API для:

- создания и обновления индекса;
- выполнения поисковых запросов;
- управления параметрами поиска;
- обработки нечётких совпадений.

Движок построен на основе Apache Lucene.

Класс SearchIndexInitializer предстваляет собой инициализатор индекса Он отвечает за создание и настройку поискового индекса, заполнение индекса документами, контроль целостности индекса и взаимодействие с файловой системой.

Проверка существования индекса происходит перед созданием его созданием, а также реализовано пакетное добавление документов.

Поисковый индекс, создаваемый Searcher, имеет следующую структуру:

1. Сегменты индекса: Индекс разбивается на несколько сегментов для оптимизации операций записи и чтения. Каждый сегмент содержит:
   - словарь терминов
   - обратный список документов
   - хранимые поля
   - нормы документов

2. Типы полей:
   — текстовые поля (description, tags, name): полнотекстовый поиск с анализом
   — строковые поля (id, type): точное совпадение
   — числовые поля (rating): диапазонные запросы
3. Метрики индекса:
   — размер индекса: 30-40% от исходных данных
   — время загрузки: 50-100 мс для 10000 документов
   — поддержка инкрементального обновления

Метод initSearchIndex класса SearchIndexInitializer создает индекс для поисковой системы. Сначала проверяется существование индекса; создаётся экземпляр Searche; если индекс не существует, создаётся новый и заполняется документами.

Определение релевантности ответов в поисковой системе

Разработанный класс RetrainableVectorSearch представляет собой компонент системы семантического поиска, который непосредственно определяет релевантные ответы и периодически переобучает модель на актуальных данных.

Компонент начинает работу с загрузки предобученной модели векторных представлений, которая инициализируется при создании экземпляра класса. Размерность векторного пространства определяется автоматически на основе загруженной модели, что гарантирует совместимость с ранее созданными векторными представлениями.

Ключевой особенностью класса является метод fullRetrain, выполняющий полное переобучение модели на всех накопленных статьях. Процесс переобучения начинается с подготовки итератора по предложениям, который последовательно предоставляет тексты для обучения. Новая модель создается с сохранением основных параметров исходной модели, включая размерность векторного пространства, размер окна контекста и другие важные характеристики, что обеспечивает согласованность векторных представлений до и после переобучения.

После создания новой модели происходит ее обучение на актуальном наборе документов, включая как первоначальные, так и все добавленные в процессе работы тексты. Обученная модель атомарно заменяет предыдущую версию, после чего система автоматически пересчитывает векторные представления для всех документов. Так гарантируется, что поисковая система после

переобучения будет работать с согласованными векторными представлениями, учитывающими все изменения в данных.

Оценка качетсва работы поисковой модели

Для проверки корректности работы поисковой модели было проведено ручное тестирование функциональности клиентского интерфейса и семантического поиска по загруженным документам. Основной целью тестирования являлась проверка способности системы обрабатывать документы различных форматов, делить их на статьи, извлекать из них информацию и возвращать релевантные ответы на екстовые запросы пользователя.

Для целей тестирования был сформирован датасет, включающий в себя документы форматов PDF, DOCX, PPTX и TXT. Общий объём тестовой выборки составил 77 файлов (после их обработки получены 6734 статьи) . Документы были отобраны вручную для обеспечения разнообразия структур файлов и тематики информации в них.

Реализация пользовательского интерфейса

Фронтенд данной поисковой системы реализован с использованием библиотеки React. Он предоставляет интуитивно понятный и удобный интерфейс для взаимодействия с пользователем: загрузка документов, запуск поиска, отображение результатов и сообщений от системы.

Фронтенд состоит из двух ключевых компонентов: главной страницы и страницы диалога, которая реализует все функционал загрузки документов, ввода поискового запроса и отображения результатов.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была реализована полноценная система семантического поиска документов с использованием современных технологий. Backend-часть системы построена на Spring Boot с Java, что обеспечило интеграцию с Apache Lucene для эффективного полнотекстового поиска. Для обработки естественного языка и генерации векторных представлений текста использовалась модель Word2Vec (с переобучением), что позволило реализовать семантический поиск по документам.

В ходе данной работы:

1. Исследованы современные системы поиска информации в документах.
2. Создан модуль для извлечения текста из документов различных форматов с последующим автоматическим разделением на смысловые статьи на основе структурных элементов документа.
3. Разработан алгоритм обработки естественно-языковых запросов пользователя.
4. разработан интуитивно понятный веб-интерфейс для загрузки документов, создания запросов и просмотра результатов поиска.
5. Проведено комплексное тестирование системы на реальных данных, оценена точность и полнота поиска.

Таким образом, все поставленные в рамках работы задачи выполнены.