Министерство образования Республики Беларусь

Учреждение образования “Белорусский государственный университет информатики и радиоэлектроники”

Факультет информационных технологий и управления

Кафедра интеллектуальных информационных технологий

Отчёт по дисциплине

“Естественно-языковой интерфейс интеллектуальных систем”

Лабораторная работа №1

“Разработка автоматизированной системы формирования словаря естественного языка”

Выполнили студенты группы 221702: Кветко Е.Д.

Багдасаров И.Е.

Проверил: Крапивин Ю.Б.

Минск 2025

### 1. Цель работы

Освоить на практике основные принципы реализации информационно-поисковых систем и методы оценки качества их работы.

### 2. Задачи лабораторной работы (Вариант 11)

* Изучить основы построения информационно-поисковых систем.
* Реализовать модель поиска документов по запросу.
* Оценить качество работы системы с использованием метрик.
* Разработать веб-интерфейс для взаимодействия с системой.

Язык текста: Английский

Предметная область: Локальная вычислительная сеть

### 3. Описание и особенности применения готовых компонентов

В рамках проекта была разработана поисковая система на языке Python с использованием фреймворка Flask. Система предназначена для индексации и поиска текстовых документов, а также интеграции с локальной языковой моделью для ответов на запросы, по которым не найдено результатов.

Для реализации были использованы следующие ключевые компоненты:

* **Flask**: Микрофреймворк для создания веб-приложений на Python. Обеспечивает простую маршрутизацию, обработку запросов и рендеринг шаблонов.
* **PostgreSQL**: Реляционная база данных для хранения индексов документов, терминов и их статистик. Обеспечивает надежное хранение и быстрый доступ к данным.
* **NLTK (Natural Language Toolkit)**: Библиотека для обработки естественного языка. В проекте используется для:
  + Токенизации текста (`word\_tokenize`)
  + Удаления стоп-слов (`stopwords`)
  + Стемминга (`PorterStemmer`)
* **Ollama Python Client**: Клиент для взаимодействия с локальной нейросетью Ollama. Позволяет генерировать ответы на запросы, для которых не найдено документов.
* **Watchdog**: Библиотека для мониторинга файловой системы. Обеспечивает автоматическую переиндексацию при добавлении, изменении или удалении файлов.
* **Psycopg2**: Адаптер PostgreSQL для Python, используемый для взаимодействия с базой данных.

### 4. Структура разработанной системы

Система построена на основе классической трехслойной архитектуры:

**1. Слой представления (View):**

- HTML-шаблоны: index.html, search.html, metrics.html, documents.html, upload.html, stats.html, preview.html

- Шаблонизатор Jinja2

- Обработка HTTP-запросов в app.py

**2. Слой сервисов (Service):**

- SearchService - ядро поисковой системы (индексация, поиск BM25)

- DataService - управление взаимодействием с БД

- FileWatcher - мониторинг файловой системы

**3. Слой данных (Data):**

- PostgreSQL база данных с таблицами документов, терминов и связей

### 5. Структура базы данных системы

Таблица **documents** (Документы)

Поля:

- id - уникальный идентификатор документа (первичный ключ)

- title - название документа (имя файла без расширения)

- content - полное текстовое содержимое документа

- date\_added - дата добавления документа в систему

- file\_path - полный путь к файлу на диске (уникальный)

- last\_modified - временная метка последнего изменения файла

- doc\_length - длина документа в токенах (после предобработки)

Ответственность:

- Хранение метаинформации о документах

- Содержание полного текста для поиска и предпросмотра

- Отслеживание изменений файлов через last\_modified

- Расчет статистики через doc\_length

Таблица **terms** (Термины)

Поля:

- term - сам термин (слово в основе, после стемминга) - первичный ключ

- doc\_count - количество документов, содержащих данный термин

Ответственность:

- Расчет IDF (Inverse Document Frequency) для алгоритма BM25

- Статистика распространенности терминов в коллекции

Таблица **document\_terms** (Связь документов и терминов)

Поля:

- doc\_id - ссылка на документ (внешний ключ к documents.id)

- term - ссылка на термин (внешний ключ к terms.term)

- frequency - частота встречаемости термина в конкретном документе

Ответственность:

- Реализация инвертированного индекса

- Хранение TF (Term Frequency) для алгоритма BM25

- Связывание документов с их терминологическим составом

### 6. Основные алгоритмы реализации

**1. Алгоритм индексации документа**

Этап 1: Чтение файла

- Открытие текстового файла по указанному пути

- Считывание всего содержимого в строковую переменную

- Обработка ошибок доступа и кодировки

Этап 2: Токенизация текста

- Разбиение текста на отдельные слова (токены)

- Использование NLTK word\_tokenize для английского языка

- Сохранение порядка слов для возможного фразового поиска

Этап 3: Приведение к нижнему регистру

- Нормализация всех токенов к единому регистру

- Обеспечение регистронезависимого поиска

- Сохранение оригинального текста для отображения

Этап 4: Удаление стоп-слов

- Фильтрация общеупотребительных слов (the, and, is, etc.)

- Использование NLTK stopwords для английского языка

- Уменьшение размера индекса и шума при поиске

Этап 5: Стемминг (PorterStemmer)

- Приведение слов к их основе (running → run, computers → comput)

- Группировка различных форм одного слова

- Улучшение полноты поиска

Этап 6: Подсчет частот терминов

- Создание словаря "термин → частота"

- Учет повторяющихся слов в документе

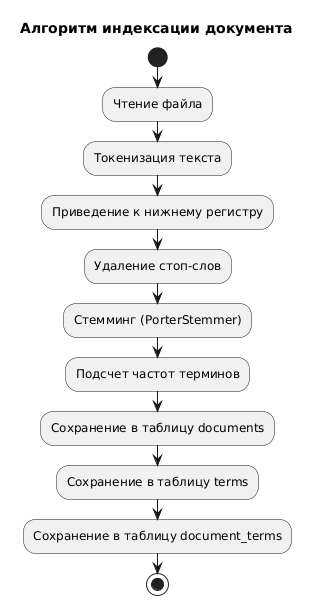
- Расчет TF (Term Frequency) для алгоритма BM25

Этап 7: Сохранение в БД

- documents: метаинформация и полный текст

- terms: глобальная статистика терминов

- document\_terms: связи документов с терминами и частотами



**2. Алгоритм поиска с gpt-фолбэком**

Этап 1: Получение поискового запроса

- Прием строки запроса от пользователя через веб-интерфейс

- Валидация входных данных (проверка на пустоту, длина)

Этап 2: Предобработка запроса

- Применение той же цепочки обработки, что и при индексации

- Токенизация, приведение к нижнему регистру, удаление стоп-слов, стемминг

- Получение нормализованных терминов для поиска

Этап 3: Выполнение BM25 поиска

- Расчет релевантности всех документов коллекции

- Сортировка результатов по убыванию score

- Выбор топ-N наиболее релевантных документов

Этап 4: Проверка наличия результатов

- Если найдены документы с score > 0 → возврат обычных результатов

- Если документы не найдены → переход к GPT-фолбэку

Этап 5: Формирование промпта для Ollama

- Создание контекстного запроса на основе исходного вопроса

- Добавление инструкций для получения качественного ответа

- Настройка параметров генерации (temperature, length)

Этап 6: Отправка запроса к LLM

- Использование Ollama Python Client для взаимодействия

- Обработка таймаутов и ошибок сети

Этап 7: Получение и обработка ответа

- Извлечение сгенерированного текста из ответа API

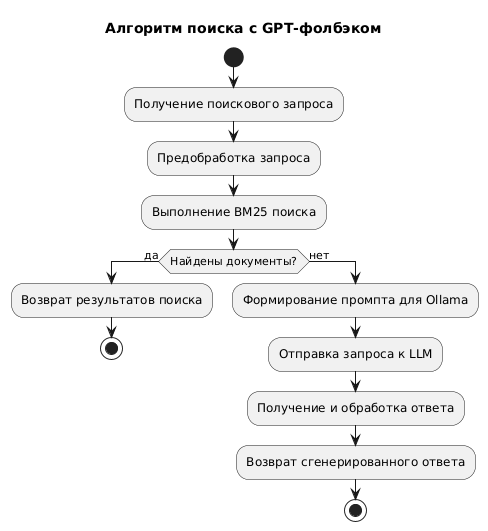
- Очистка и форматирование результата

- Создание структуры, совместимой с результатами поиска

Этап 8: Возврат сгенерированного ответа

- Представление ответа LLM как "виртуального документа"

- Сохранение единого формата ответа для клиента



**3. Алгоритм расчета BM25**

Этап 1: Инициализация входных данных

- Документ D: текущий оцениваемый документ

- Запрос Q: предобработанные термины поискового запроса

- Параметры: k1=2.0, b=0.75

Этап 2: Получение статистик документа

- Частоты терминов документа из таблицы document\_terms

- Длина документа |D| в токенах (после предобработки)

- Средняя длина документов коллекции avgdl

Этап 3: Цикл по терминам запроса

- Для каждого уникального термина в поисковом запросе

- Пропуск терминов, отсутствующих в документе

- Последовательный расчет вклада каждого термина

Этап 4: Расчет IDF (Inverse Document Frequency)

- N - общее количество документов в коллекции

- n\_t - количество документов, содержащих термина t

- Формула: IDF = log((N + 1) / (n\_t + 1))

- Учет редкости термина в коллекции

Этап 5: Расчет TF-компоненты

- f - частота термина в текущем документе

- |D| - длина текущего документа

- avgdl - средняя длина документов

- Формула: TF = (f \* (k1 + 1)) / (f + k1 \* (1 - b + b \* |D|/avgdl))

Этап 6: Накопление общего score

- Умножение IDF на TF-компоненту для каждого термина

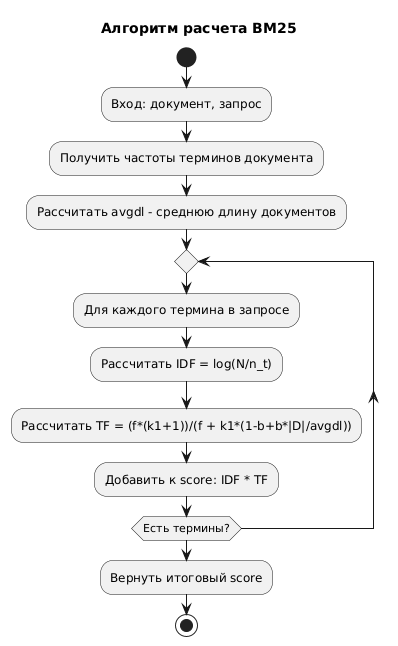
- Суммирование вкладов всех терминов запроса

- Получение итоговой оценки релевантности документа

Этап 7: Возврат результата

- Возврат рассчитанного score для ранжирования

- Использование в алгоритме сортировки результатов



**4. Алгоритм мониторинга файлов**

Этап 1: Запуск FileWatcher

- Инициализация наблюдателя файловой системы

- Настройка отслеживания целевой директории

- Запуск фонового процесса мониторинга

Этап 2: Сканирование существующих файлов

- Рекурсивный обход указанной директории при запуске

- Поиск всех .txt файлов для первоначальной индексации

- Обеспечение актуальности индекса при перезапуске

Этап 3: Цикл активного мониторинга

- Непрерывное отслеживание событий файловой системы

- Обработка событий в реальном времени

- Поддержание работы пока система активна

Этап 4: Обработка создания файлов

- Обнаружение новых .txt файлов в отслеживаемой директории

- Добавление нового документа в поисковый индекс

Этап 5: Обработка изменений файлов

- Отслеживание модификации существующих .txt файлов

- Обновление документа в индексе при изменениях

Этап 6: Обработка удаления файлов

- Обнаружение удаления .txt файлов

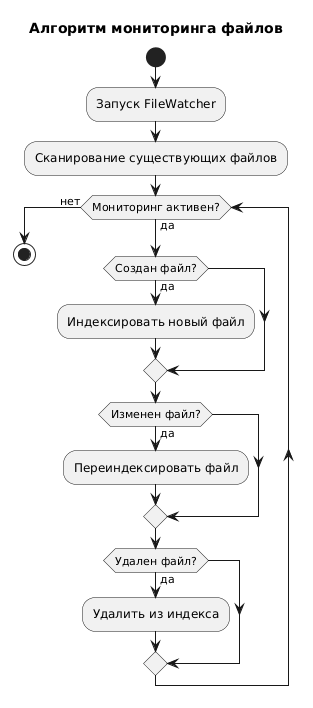
- Удаление соответствующего документа из индекса

- Очистка связанных терминов и статистик

Этап 7: Завершение мониторинга

- Корректная остановка при завершении работы приложения

- Логирование статистики мониторинга



### 7. Тестирование системы

**Тестовая коллекция документов:**

Для тестирования была создана папка documents с текстовыми файлами на 3 различные темы: медицина, технологии и архитектура

**Результаты тестирования:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Входные данные (поисковый запрос) | Ожидаемый результат | Фактический результат | Статус |
| 1 | machine | Найдено два файла, посвященных машинному обучению | Найдено 2 релевантых документа | **Успех** |
| 2 | protein | Найдены три статьи, посвященные медицине | Найдены 3 релевантных документа | **Успех** |
| 3 | structure | Найдены три статьи, посвященных архитектуре | Найдены 4 документа, 3 из которых являются релевантными, последний имеет самый низкий рейтинг, следовательно тест считается успешным | **Успех** |
| 4 | social engineering | Найден документ описывающий социальную инженерию | Найден АРХИТЕКТУРА с высоким рангом. | **Успех** |
| 5 | weather | Локальный поиск не находит документов. Система обращается к Ollama и возвращает сгенерированный ответ. | Результатов по файлам нет. Возвращен ответ от OllamaService с заголовком “AI Answer: weather”. | **Успех** |
| 6 | (пустой запрос) | Оповещение пользователя о пустомм вводе | Предупреждение о пустом поле | **Успех** |

### 8. Оценка качества поиска по стандартным метрикам

Для оценки качества поиска пользователю предлагается самостоятельно определить релевантные документы к конкретному запросу в разделе “Metrics”. Пользователь вводит тестовый запрос, после прилагает список путей к релевантным документам. После система рассчитывает все необходимые метрики.

Например для данных:

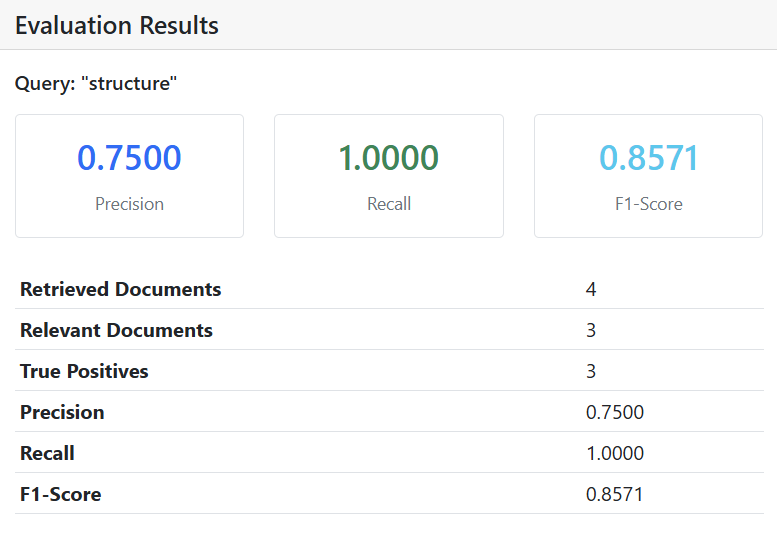
Запрос: “structure”

Релевантные документы:   
 *C:\ME\BSUIR\sem7\EYAZIS\lab1(search\_system)\documents\text7.txt*

*C:\ME\BSUIR\sem7\EYAZIS\lab1(search\_system)\documents\text8.txt*

*C:\ME\BSUIR\sem7\EYAZIS\lab1(search\_system)\documents\text9.txt*

Расчет метрик:



### 9. Вывод и предложения по улучшению

Система успешно выполняет поставленные задачи: индексирует локальные документы и осуществляет по ним быстрый поиск. Интеграция с Ollama является хорошим решением для “запросов-заглушек”, когда в локальной базе знаний нет ответа. Автоматическая переиндексация делает систему гибкой и удобной в администрировании.

**Слабые стороны:**

1. **Качество поиска:** Текущая реализация использует стандартный TF-IDF и базовый анализ текста. Для более сложных запросов релевантность может быть неидеальной.
2. **Отсутствие фрагментов (сниппетов):** В результатах поиска не отображаются фрагменты текста, где были найдены ключевые слова, что снижает удобство для пользователя.
3. **Производительность Ollama:** Длительное ожидание ответа от локальной LLM может негативно сказаться на пользовательском опыте.

**Предложения по улучшению:**

1. **Реализация подсветки и сниппетов:** Для улучшения пользовательского опыта при работе с результатами локального поиска необходимо добавить формирование информативных сниппетов (фрагментов текста) с подсветкой найденных слов.
2. **Асинхронная загрузка ответа от LLM:** Чтобы пользователь не ждал долгого ответа от нейросети в случае отсутствия локальных результатов, можно реализовать асинхронную загрузку. Пользователь сначала увидит сообщение “Локальный поиск не дал результатов, обращаемся к нейросети…”, а затем, когда ответ будет готов, он подгрузится на страницу без перезагрузки с помощью JavaScript.
3. **Интеграция доступа к сети Интернет (метод RAG):** Текущая модель в Ollama работает офлайн. Чтобы дать ей возможность отвечать на вопросы, требующие актуальной информации (новости, погода, курсы валют), можно внедрить механизм **Retrieval-Augmented Generation (RAG)**. Это не потребует замены модели, но изменит логику работы SearchService.
   * **Шаг 1:** Когда локальный поиск не дает результатов, система перед обращением к Ollama выполняет дополнительное действие: делает поисковый запрос во внешнюю систему
   * **Шаг 2:** Получив результаты поиска из интернета (например, несколько текстовых фрагментов), система формирует новый, "обогащенный" промпт для Ollama.
   * **Пример:**
     + *Изначальный промпт:* "Какая погода сегодня в Минске?"
     + *Найденный в сети контекст:* "По данным метеослужбы, сегодня в Минске +15 градусов и солнечно."
     + *Новый промпт для Ollama:* "Используя следующий контекст: 'По данным метеослужбы, сегодня в Минске +15 градусов и солнечно', ответь на вопрос: Какая погода сегодня в Минске?"
   * **Шаг 3:** Этот новый промпт отправляется в OllamaService. Модель, имея всю необходимую информацию, генерирует точный и актуальный ответ. Для упрощения реализации этого механизма можно использовать фреймворки, такие как **LangChain** или **LlamaIndex**.