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 №3

“Автоматическое реферирование документов”

Выполнили студенты группы 221702: Кветко Е.Д.

Багдасаров И.Е.

Проверил: Крапивин Ю.Б.

Минск 2025

### 1. Цель работы

Цель работы – освоить на практике основные принципы автоматического реферирования документов.

### 2. Задачи лабораторной работы (Вариант 1)

Язык текста: Английский, русский

Предметная область: Научные статьи по computer science, сочинения по литературе

### 3. Описание структуры разработанной системы

В рамках проекта была разработана система реферирования на языке Python с использованием фреймворка Flask. Система предназначена для получения классического реферата текста а также ключевых слов.

Для реализации были использованы следующие ключевые модули:

**Модуль app.py (основное Flask-приложение)**

Цель и задачи:

* Организация веб-интерфейса для взаимодействия с пользователем
* Управление файловыми операциями и сессиями
* Координация работы между модулями системы

Основные функции:

* index() - отображение главной страницы
* upload\_file() - обработка загруженных файлов
* create\_summary() - генерация реферата
* history\_page() - управление историей операций
* download\_summary() - экспорт результатов

Библиотеки:

* Flask - веб-фреймворк
* werkzeug - обработка файлов и безопасность
* reportlab - генерация PDF
* uuid - создание уникальных идентификаторов

**Модуль file\_processor.py**

Цель и задачи:

* Извлечение текста из различных форматов файлов
* Определение языка текста
* Предварительная обработка и очистка текста

Основные функции:

* read\_file() - чтение файлов разных форматов
* detect\_language() - определение языка
* clean\_text() - очистка текста
* split\_into\_sentences() - разделение на предложения

Библиотеки:

* langdetect - определение языка
* chardet - определение кодировки
* re - регулярные выражения

**Модуль summarizer.py**

Цель и задачи:

* Анализ семантической структуры текста
* Выделение ключевых понятий и терминов
* Генерация сжатого реферата
* Извлечение ключевых слов

Основные функции:

* preprocess\_text() - подготовка текста
* calculate\_term\_weights() - расчет весов терминов
* generate\_classic\_summary() - создание реферата
* generate\_keyword\_summary() - извлечение ключевых слов

Библиотеки:

* nltk - обработка естественного языка
* pymorphy3 - морфологический анализ русского языка
* collections.Counter - статистический анализ

### 5. Структуры данных

Входные данные:

* 'file\_path': str, - путь к файлу
* 'filename': str, - оригинальное имя файла
* 'text': str, - извлеченный текст
* 'language': str - определенный язык ('ru'/'en')

Промежуточные данные:

* 'processed\_words': list, - обработанные слова (леммы)
* 'original\_words': list, - оригинальные слова
* 'term\_weights': dict, - словарь {термин: вес}
* 'sentence\_scores': list - список (предложение, оценка)

Выходные данные:

* 'classic\_summary': str, - классический реферат
* 'keyword\_summary': list, - список ключевых слов
* 'language': str, - язык текста
* 'original\_length': int, - длина исходного текста
* 'summary\_length': int, - длина реферата
* 'compression\_ratio': float - степень сжатия (%)

### 6. Основные алгоритмы реализации

Текстовое описание алгоритма:

1. Предобработка текста
   * Токенизация на слова и предложения
   * Приведение к нижнему регистру
   * Удаление стоп-слов и коротких слов (<3 символов)
   * Лемматизация (приведение к нормальной форме)

2. Вычисление весов терминов

* Подсчет частоты встречаемости терминов
* Нормализация частот относительно общего объема текста
* Создание словаря весов {термин: значимость}

3. Оценка важности предложений

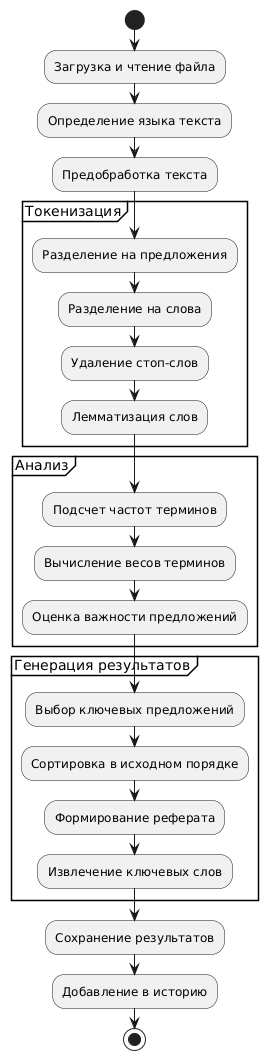
* Для каждого предложения вычисляется суммарный вес содержащихся терминов
* Нормализация по количеству значимых терминов в предложении
* Учет позиционного бонуса (первые предложения важнее)

4. Генерация реферата

* Выбор топ-N предложений с наивысшими оценками
* Восстановление исходного порядка выбранных предложений
* Объединение в связный текст

5. Извлечение ключевых слов

* Выбор топ-M терминов с наибольшими весами
* Формирование списка ключевых понятий



### 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. Динамическая конфигурация и автоматическая переиндексация делают систему гибкой и удобной в администрировании.

**Слабые стороны:**

1. **Качество поиска:** Текущая реализация использует стандартный ClassicSimilarity 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.
4. **Единая точка отказа:** Если сервис Ollama не запущен, пользователь получит сообщение об ошибке.

**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. Это можно реализовать с помощью класса Highlighter из библиотеки Apache Lucene, который уже используется в проекте.
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 (например, через технологию AJAX или Fetch API).
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 (например, через Google Search API или DuckDuckGo Search API).
   * **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**.
4. **Переход на альтернативный интерфейс с встроенным доступом к сети:** Вместо ручной реализации метода RAG можно интегрировать проект с готовыми решениями, которые уже умеют это делать. Важно понимать, что локальные модели сами по себе не выходят в интернет. Эту функцию выполняет приложение-обертка.
   * **Решение:** Можно заменить стандартное обращение к API Ollama на взаимодействие с таким инструментом, как **Open WebUI** (ранее известный как Ollama WebUI).
   * **Как это работает:** Open WebUI — это продвинутый веб-интерфейс для Ollama, который устанавливается отдельно. Он имеет встроенную функцию веб-поиска. Ваше приложение может обращаться уже не напрямую к Ollama, а к API Open WebUI. Если в запросе указать необходимость поиска в сети, Open WebUI сам выполнит поиск, сформирует RAG-промпт и передаст его модели Ollama, а вашему приложению вернет уже готовый, актуальный ответ. Это переносит сложность реализации RAG на сторону готового инструмента.
5. **Логирование и мониторинг:** Добавить полноценное логирование (например, с помощью фреймворка Logback, который идет в комплекте со Spring Boot) для отслеживания работы системы, поисковых запросов и ошибок. Это поможет в дальнейшей диагностике, оптимизации и анализе того, как пользователи взаимодействуют с системой.