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 №2

«Автоматическое распознавание языка текста»

Выполнили студенты Лукашевич А.Д.

группы 221703: Веркович Е.В.

Проверил: Крапивин Ю.Б.

Минск 2025

## **Цель работы:**

Изучить и отработать практические навыки применения методов автоматического распознавания языка текстовых документов.

## **Задачи лабораторной работы (Вариант 19):**

* Изучить и реализовать основные методы определения языка текстовых документов (алфавитный, частотных слов, нейросетевой).
* Разработать систему для обработки коллекции HTML-документов на Русский и Английском языках.
* Провести сравнительный анализ реализованных методов по точности и быстродействию.
* Разработать интерфейс для взаимодействия с системой.

## **Язык текста:**

Русский, Английский

## **Формат Документа:**

HTML

## **Текстовая коллекция документов:**

Документы на английском и русском языках были взяты с википедии по случайному поиску и дополнены книгами(“Дракула” Стокера на английском и “Голова профессора Доуля” Беляева на русском) Что позволило сформировать качественную основу базу для выполнения лабораторной работы.

## **Описание инструментов разработки:** В рамках задачи было создано десктопное приложение на языке Python с применением следующих библиотек и технологий:

* **FastAPI + Jinja2** — фреймворки Python для создания веб-приложения и шаблонизаторов интерфейса.
* **BeautifulSoup4** — библиотека для парсинга HTML и XML документов. В проекте была задействована для извлечения чистого текстового содержимого из HTML-файлов, что является ключевым шагом предобработки перед анализом.
* **PyTorch + sklearn** — для реализации нейросетевого метода была реализована нейронная сеть с одним скрытым слоем. Входные и тренировочные тексты были векторизованы встроенным алгоритмом TF-IDF.

## **Структура разработанной системы:**

Система построена по модульному принципу, где каждый компонент отвечает за свою часть функционала:

* **main.py (backend):** основной модуль, отвечающий за веб интерфейс, обработку действий пользователя (нажатие кнопок, открытие файлов) и координацию работы других компонентов.
* **Templates/ :** тут лежат шаблоны файлов интерфейса.
* **Lang\_processors/ :** тут лежат файлы с классами-обработчиками текстов. AlphabetProcessor, FrequencyProcessor и NeuroProcessor. Также в этот папке лежит JupiterNotebook в котором обучалась нейронная сеть.

**Структуры данных:**

## ***AlphabetProcessor*** (Детектор языка на основе частоты букв)

Эта часть кода использует следующие структуры данных для создания профилей языков и сравнения текста.

1. Словарь (dict) - self.profiles

Основное хранилище для профилей частоты букв для каждого языка.Это словарь, где ключом является название языка (строка, например, "english", "russian"), а значением является вложенный словарь (профиль языка). Хранит буквы (строки) как ключи и их относительные частоты (числа с плавающей точкой от 0 до 1) как значения. Пример: self.profiles["english"] = {"a": 0.081, "b": 0.015, ...}

### 2. Объект collections.Counter

Используется внутри метода create\_profile\_from\_file и detect для быстрого подсчета частоты вхождения каждой буквы в текст.По сути, это подкласс словаря, где ключами являются буквы, а значениями — количество их вхождений (целые числа).

### 3. Множество (set) - all\_letters

В методе detect используется для сбора всех уникальных букв, присутствующих как в анализируемом тексте, так и во всех обученных профилях языков. Это необходимо для точного расчета косинусного сходства, чтобы учесть буквы, отсутствующие в одном из сравниваемых профилей (им присваивается частота 0).

### 4. Словарь (dict) - similarities

Используется в методе detect для временного хранения результатов косинусного сходства анализируемого текста с каждым из языковых профилей.Ключ — название языка (строка), значение — показатель сходства (число с плавающей точкой).

## ***WordFrequencyProcessor*** (Детектор языка на основе частоты слов)

### 1. Словарь (dict) - self.profiles

Основное хранилище для профилей частоты слов для каждого языка. Аналогично AlphabetProcessor, ключ — название языка, значение — вложенный словарь.Хранит слова (строки) как ключи и их нормализованную частоту (числа с плавающей точкой, относительно максимальной частоты) как значения.

### 2. Объект collections.Counter

Используется в create\_profile\_from\_file для подсчета частоты вхождения слов (а не букв).

### 3. Множество (set) - document\_words

В методе detect используется для хранения уникальных слов из анализируемого текста. Это позволяет избежать повторного подсчета веса слова, если оно встречается в тексте несколько раз, поскольку метод подсчета баллов основан на наличии слова в профиле, а не на его частоте в документе.

### 4. Словарь (dict) - scores

Временное хранилище для суммарного веса (балла) анализируемого текста по отношению к каждому языковому профилю.

## ***NeuroProccessor*** (Детектор языка на основе нейронной сети)

Эта часть кода использует структуры данных для загрузки предварительно обученной модели и вспомогательных данных.

### 1. Словарь (dict) - self.code\_to\_label\_map

Словарь для сопоставления числовых кодов классов (выход нейронной сети) с их текстовыми названиями языков. Загружается из JSON-файла. Ключ — код класса (строка, например, "0", "1"), значение — название языка (строка, например, "Русский", "Английский").

### 2. Тензор PyTorch (torch.tensor)

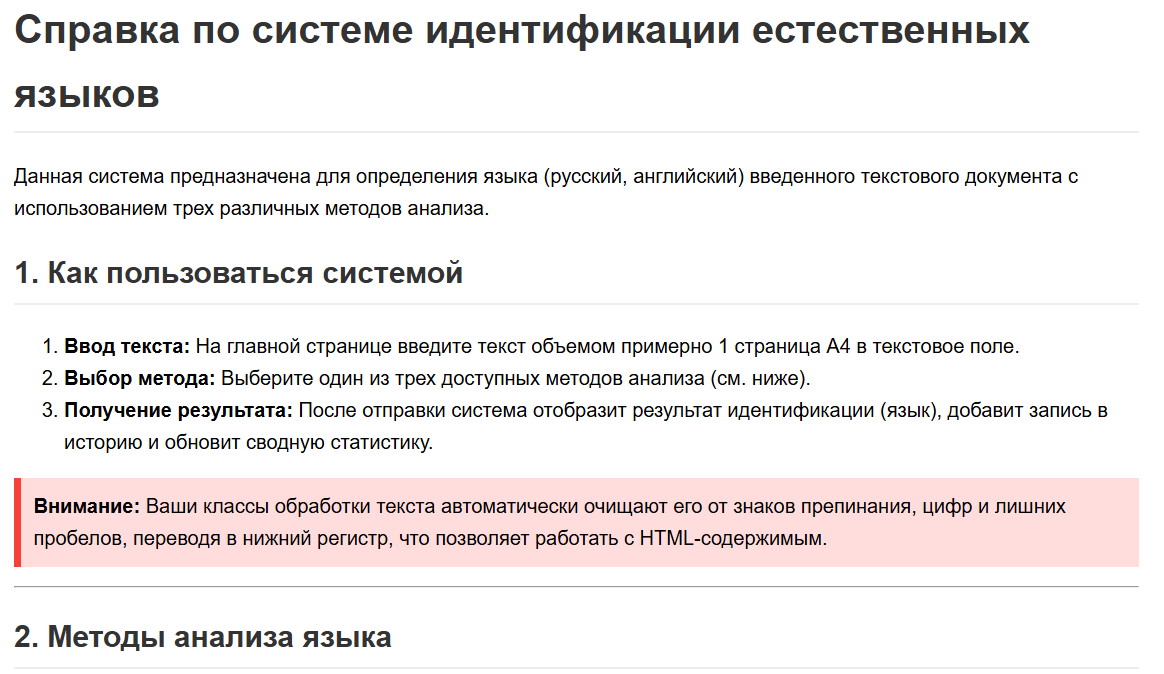
Основная структура данных для работы с нейронной сетью. Используется для: преобразования разреженного вектора TF-IDF (из библиотеки joblib) в плотную тензорную форму, которую может принять нейронная сеть, хранения выходных значений нейронной сети перед применением функции активации (необработанные оценки для каждого класса).

## **Описание алгоритма определения языка**

В системе реализовано четыре метода для сравнения их эффективности.

1. Алфавитный метод - для входного текста создается его собственный профиль частоты букв. Определяется объединенное множество всех уникальных букв, присутствующих как в документе, так и во всех эталонных профилях. Для сравнения профиля документа с профилем каждого эталонного языка используется косинусное сходство (Cosine Similarity). Профили рассматриваются как многомерные векторы, где каждое измерение соответствует определенной букве. Частота буквы — это координата вектора. Язык, с которым косинусное сходство максимально, объявляется наиболее вероятным.
2. Метод частотных слов - входной текст очищается, и из него извлекается множество уникальных слов. Для каждого эталонного языкового профиля рассчитывается суммарный балл: алгоритм итерирует по уникальным словам из входного текста и если слово присутствует в эталонном профиле языка, его нормализованный вес (из профиля) добавляется к общему баллу этого языка.Язык, набравший наибольший суммарный балл (наибольший "общий вес" совпавших слов), объявляется наиболее вероятным.
3. Нейросетевой метод входной текст преобразуется в вектор признаков TF-IDF с помощью загруженного векторизатора. Этот вектор представляет важность каждого слова или n-граммы в тексте. Разреженный вектор преобразуется в тензор PyTorch (torch.tensor). Тензор подается на вход нейронной сети (self.neuro). Сеть вычисляет вероятностные оценки для каждого класса языка. Выбирается индекс класса с максимальным значением логита (torch.argmax). Найденный индекс сопоставляется с названием языка с помощью словаря.

## **Скриншоты разработанной системы:**



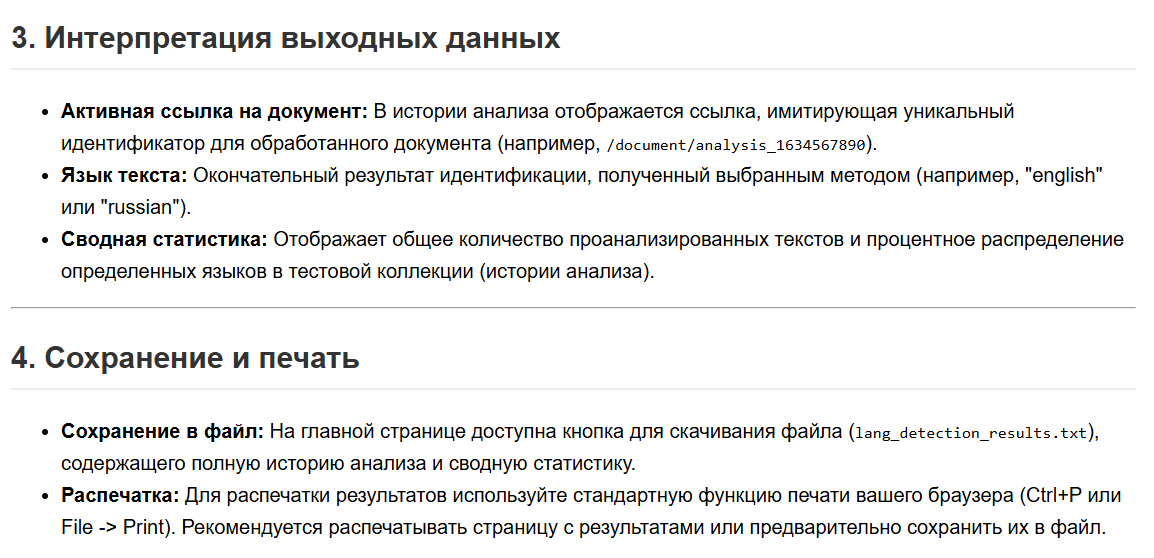


Рис. 6 – Скриншот №1

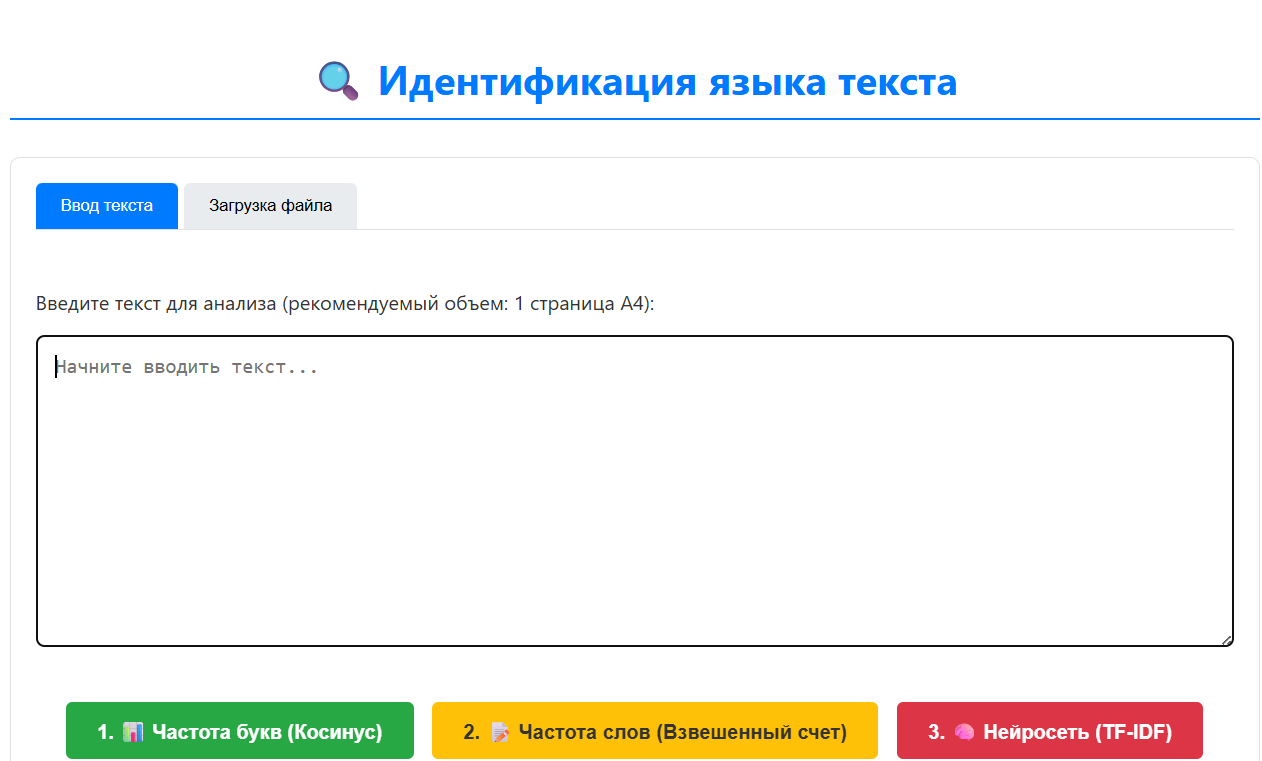


Рис. 7 – Скриншот №2

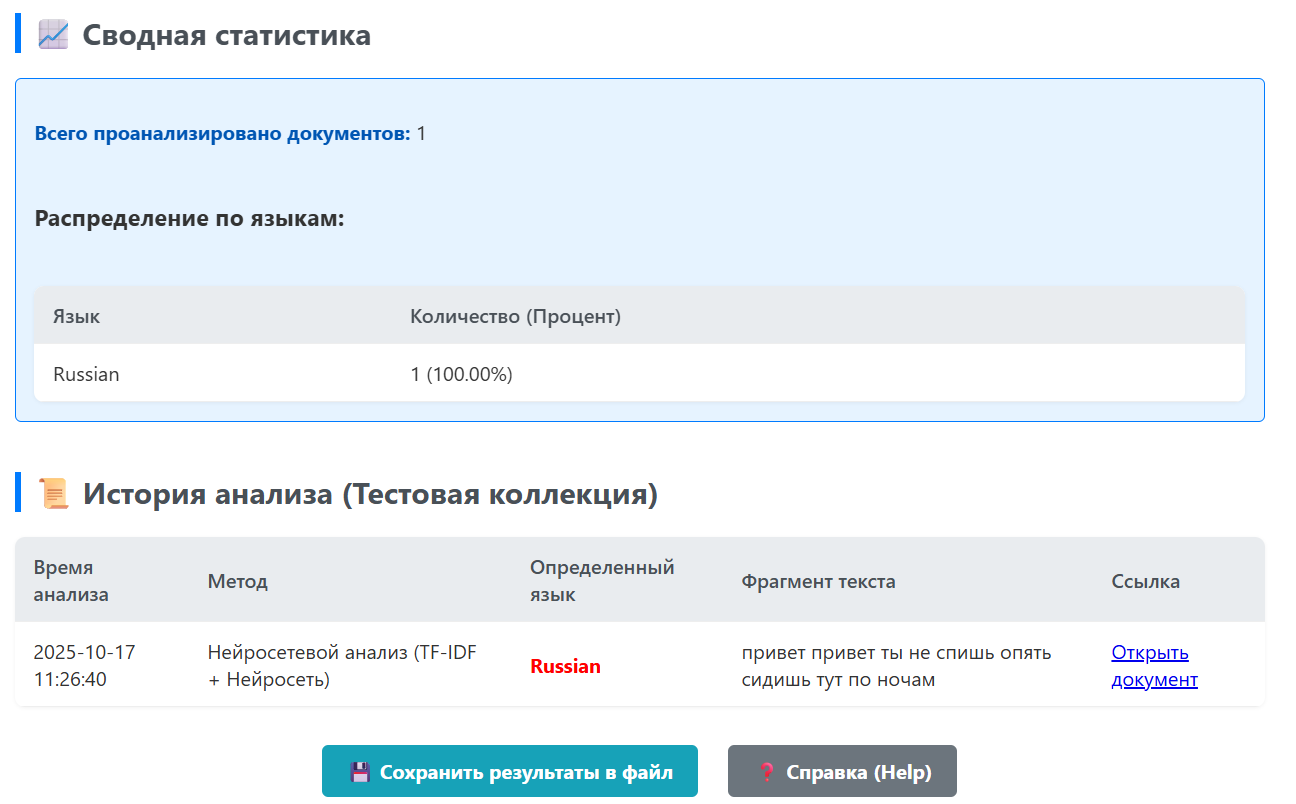
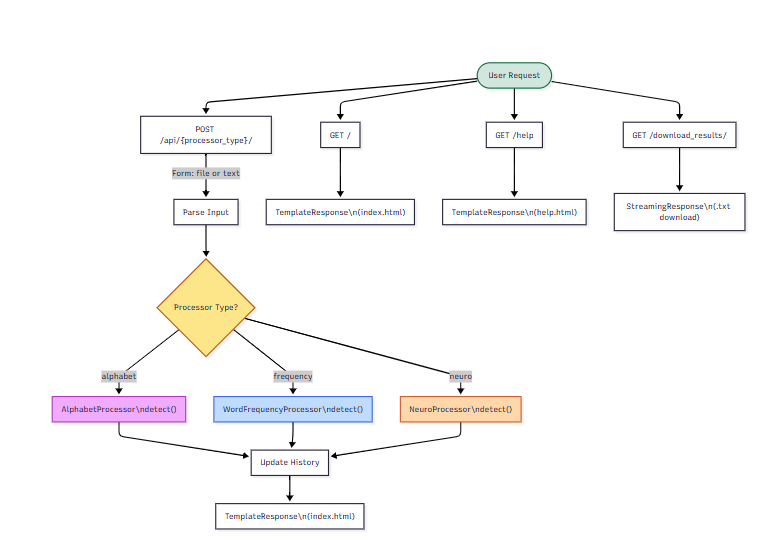


Рис. 8 – Скриншот №2

## **Структурно-функциональная схема приложения:**



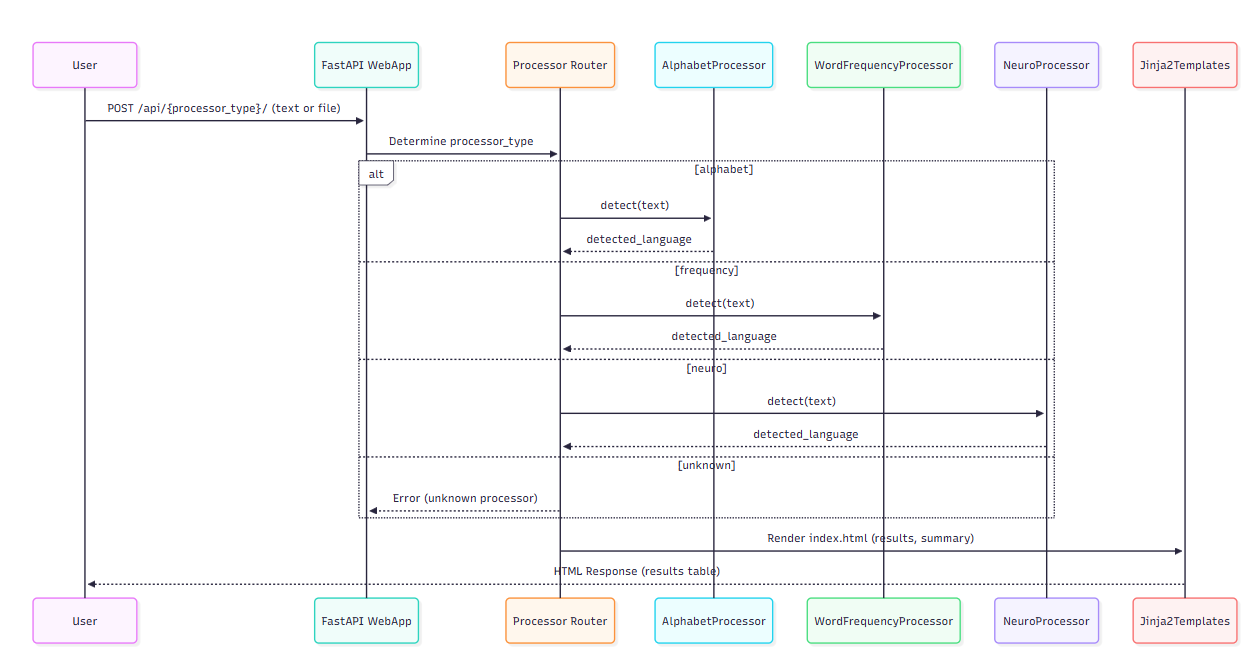


Рис. 9 - структурно-функциональная схема приложения

**Описание и особенности применения готовых компонент:**

Алфавитный метод – алгоритм очень прост, быстр и эффективен для больших объемов текста, поскольку частотные профили букв (например, E в английском, О в русском) стабильны. Хорошо работает даже при наличии опечаток или нелитературного текста, поскольку опечатки слабо меняют общий частотный профиль. Использует косинусное сходство, что является стандартным и надежным способом сравнения векторов.Менее эффективен для коротких текстов. Не различает языки, использующие один и тот же алфавит (например, испанский и итальянский, если не учитывать акценты, которые код игнорирует).

Метод частоты слов - хорошо использует лексические особенности языка, фокусируясь на наиболее характерных для него словах (артиклях, предлогах, союзах). Более точен, чем частота букв, для языков с одинаковым алфавитом, но разными наборами частотных служебных слов. Очень чувствителен к размеру профиля (profile\_size). Может быть менее точным для неформального или очень специфического текста, где не используются слова из топ-300.

Нейронные сети (в данном случае, простой многослойный перцептрон) с векторизацией TF-IDF являются наиболее точным и современным подходом. Они автоматически улавливают сложные закономерности и комбинации признаков (например, специфические n-граммы или сочетания слов), которые вручную трудно учесть.

Система полностью соответствует требованиям задания, включая наличие UI, экспорт результатов и справочную информацию.

## **Перспективы развития приложения:**

1. Расширение языковой поддержки.
2. Поддержка других форматов.