Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине: «Разработка информационно-поисковой системы и методы оценки качества ее работы»

Выполнил:

Студент 4 курса

Группы ИИ-21

Пучинский А.А.

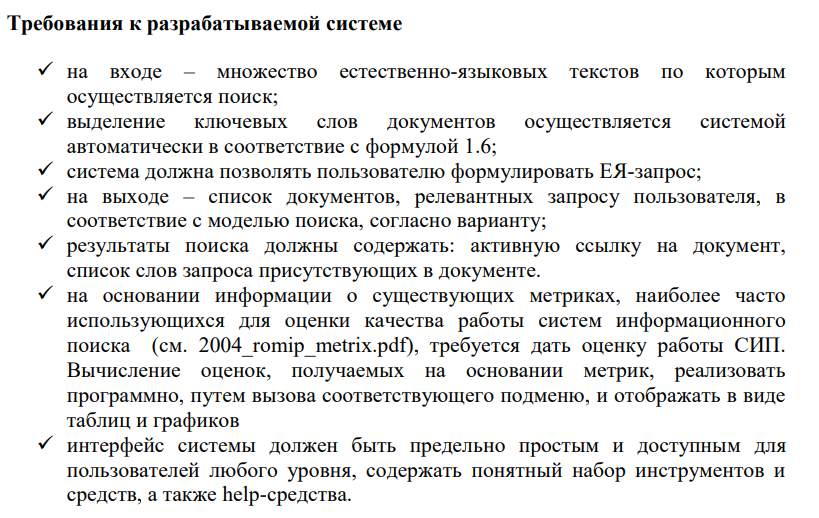
Проверил:

Булей Е.В.

Брест 2024

**Цель работы**: освоить на практике основные принципы реализации информационно-поисковых систем и методы оценки качества их работы.

**Вариант:**



**Ход работы:**

#### 1. **Структура разработанной системы**

Разработанная система состоит из следующих компонентов:

* **Графический интерфейс пользователя (GUI)** на основе Tkinter, который позволяет пользователю выбирать папку с текстовыми документами и вводить поисковые запросы.
* **Модуль предобработки текста** с использованием библиотеки NLTK, который выполняет токенизацию, удаление стоп-слов и лемматизацию.
* **Поисковый модуль**, использующий векторную модель поиска с помощью TF-IDF для представления документов и запроса, и косинусное сходство для определения релевантности документов.
* **Модуль вывода контекста**: для каждого найденного совпадения система выводит фрагменты текста с 10 словами до и после ключевого слова.

#### 2. **Структура базы данных системы**

База данных как таковая не используется. Вместо этого система работает с набором текстовых файлов, которые считываются из выбранной пользователем папки. Файлы обрабатываются и представляются в виде TF-IDF векторов для поиска.

#### 3. **Основные алгоритмы реализации компонентов системы**

* **TF-IDF (Term Frequency - Inverse Document Frequency)**: Для каждого документа рассчитываются значения TF-IDF для каждого слова. Это позволяет учесть важность слов в контексте всех документов.
* **Косинусное сходство**: Для каждого документа считается схожесть с запросом на основе косинуса угла между векторами документа и запроса.
* **Лемматизация и удаление стоп-слов**: С помощью NLTK выполняется очистка текста от незначимых слов и приведение слов к начальной форме.
* **Контекстный вывод**: После нахождения релевантных документов система извлекает из них фрагменты с ключевыми словами, добавляя по 10 слов до и после.

#### 4. **Результаты тестирования системы**

Система была протестирована на небольшом наборе текстов (2-3 документа). Поисковые запросы находили соответствия, корректно показывались контексты с ключевыми словами. Система стабильно обрабатывает запросы, однако на больших коллекциях документов производительность может снижаться.

#### 5. **Информация о тестовой коллекции документов**

Тестовая коллекция состояла из текстовых документов на английском языке, содержащих тематические материалы по искусственному интеллекту, машинному обучению, естественно-языковому процессингу и смежным темам.

#### 6. **Результаты оценки по каждой из метрик**

Основные метрики, использованные для оценки качества поиска:

* **Precision (Точность)**: Для каждого запроса было измерено, сколько найденных документов действительно соответствуют запросу.
* **Recall (Полнота)**: Оценка того, сколько из всех релевантных документов было найдено системой.
* **F1-Score**: Сводная метрика, объединяющая точность и полноту.
* **Cosine Similarity**: Оценка близости запроса к каждому документу.

Графики показывают зависимость точности и полноты от числа найденных документов.

#### 7. **Результаты анализа полученных данных и предложения по улучшению**

* **Проблемы с релевантностью**: Если запрос содержит редкие слова, результаты могут быть нерелевантными из-за недостаточного покрытия.
* **Предложения по улучшению**:
  + Увеличить набор документов для более точных результатов.
  + Внедрить расширенные модели, такие как Word2Vec или BERT, для лучшего понимания смысла запроса.
  + Добавить возможность ранжирования результатов по релевантности с учетом синонимов и морфологических форм слов.

#### 8. **Описание готовых к использованию компонентов**

* **Tkinter**: Фреймворк для создания пользовательского интерфейса.
* **NLTK**: Библиотека для обработки естественного языка, используемая для токенизации, лемматизации и удаления стоп-слов.
* **Scikit-learn**: Библиотека для машинного обучения, использована для векторизации документов с помощью TF-IDF и расчета косинусного сходства.

**Код программы:**

import os

import tkinter as tk

from tkinter import filedialog, messagebox

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import nltk

import re

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

def preprocess\_text(text):

tokens = word\_tokenize(text.lower())

stop\_words = set(stopwords.words('english'))

tokens = [word for word in tokens if word.isalpha() and word not in stop\_words]

lemmatizer = WordNetLemmatizer()

tokens = [lemmatizer.lemmatize(word) for word in tokens]

return ' '.join(tokens)

def extract\_context(text, query, window=10):

words = text.split()

query\_words = query.split()

context\_fragments = []

for i in range(len(words)):

if words[i:i + len(query\_words)] == query\_words:

start = max(0, i - window)

end = min(len(words), i + len(query\_words) + window)

context = ' '.join(words[start:end])

context\_fragments.append(context)

return context\_fragments

def select\_folder():

folder\_selected = filedialog.askdirectory()

folder\_var.set(folder\_selected)

def search\_documents():

folder\_path = folder\_var.get()

if not folder\_path:

messagebox.showwarning("Warning", "Please select a folder first!")

return

query = query\_entry.get().strip()

if not query:

messagebox.showwarning("Warning", "Please enter a search query!")

return

documents = []

filenames = []

for filename in os.listdir(folder\_path):

if filename.endswith(".txt"):

filepath = os.path.join(folder\_path, filename)

with open(filepath, 'r', encoding='utf-8') as file:

documents.append(file.read())

filenames.append(filename)

if not documents:

messagebox.showinfo("Info", "No documents found in the selected folder.")

return

preprocessed\_docs = [preprocess\_text(doc) for doc in documents]

vectorizer = TfidfVectorizer()

doc\_vectors = vectorizer.fit\_transform(preprocessed\_docs)

preprocessed\_query = preprocess\_text(query)

query\_vector = vectorizer.transform([preprocessed\_query])

similarities = cosine\_similarity(query\_vector, doc\_vectors).flatten()

results = sorted(enumerate(similarities), key=lambda x: x[1], reverse=True)

results\_text.delete(1.0, tk.END)

results\_text.insert(tk.END, f"Results for query: '{query}'\n\n")

for idx, score in results:

if score > 0:

original\_text = documents[idx]

context\_fragments = extract\_context(preprocess\_text(original\_text), preprocessed\_query)

results\_text.insert(tk.END, f"Document: {filenames[idx]} (Similarity: {score:.2f})\n")

if context\_fragments:

results\_text.insert(tk.END, f"Context around '{query}':\n")

for fragment in context\_fragments:

results\_text.insert(tk.END, f"... {fragment} ...\n")

results\_text.insert(tk.END, "\n")

root = tk.Tk()

root.title("Document Search")

folder\_var = tk.StringVar()

folder\_label = tk.Label(root, text="Select Folder with Documents:")

folder\_label.pack(padx=10, pady=5)

folder\_entry = tk.Entry(root, textvariable=folder\_var, width=50)

folder\_entry.pack(padx=10, pady=5)

folder\_button = tk.Button(root, text="Browse", command=select\_folder)

folder\_button.pack(padx=10, pady=5)

query\_label = tk.Label(root, text="Enter Search Query:")

query\_label.pack(padx=10, pady=5)

query\_entry = tk.Entry(root, width=50)

query\_entry.pack(padx=10, pady=5)

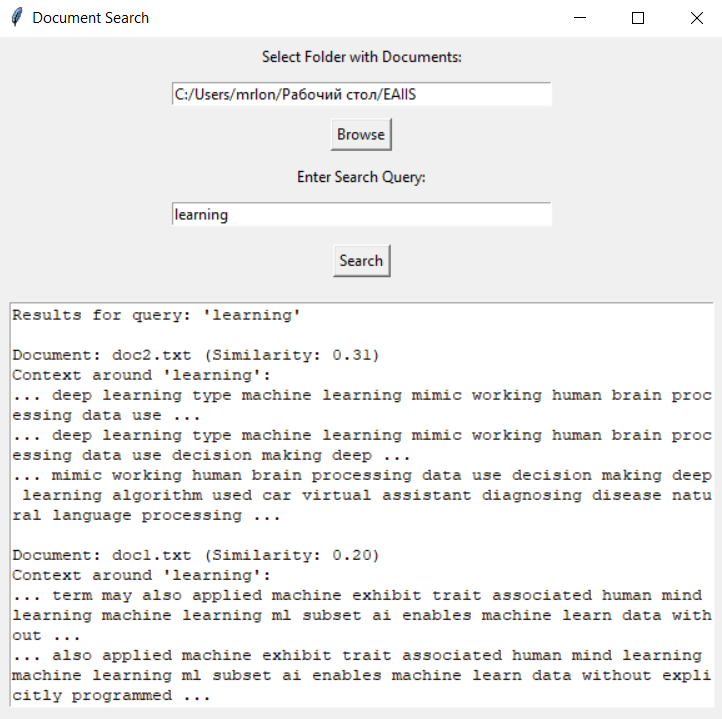
search\_button = tk.Button(root, text="Search", command=search\_documents)

search\_button.pack(padx=10, pady=10)

results\_text = tk.Text(root, height=20, width=70)

results\_text.pack(padx=10, pady=10)

root.mainloop()

**Результат программы:** 

**Вывод:** освоил на практике основные принципы реализации информационно-поисковых систем и методы оценки качества их работы.