Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине: «Естественно-языковой интерфейс ИС»

Тема: «Разработка системы автоматического реферирования документов»

Выполнил:

Студент 4 курса

Группы ИИ-21

Пучинский А.А.

Проверил:

Булей Е.В.

Брест 2024

**Цель:** освоить на практике основные принципы автоматического реферирования документов.

**Ход работы:**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Язык текста** | **Методика** | **Предметная область** |
| 9 | Русский, Английский | Sentence extraction+ ML | Научные статьи по computer science, Сочинения по литературе |



* на входе – на входе – текстовые документы одинакового размера (например, 10 страниц формата А4), содержащие тексты из предметных областей на естественных языках согласно варианту подлежащие процедуре автоматического реферирования;
* на выходе – активная ссылка на исходный документ и построенный, в соответствии с вариантом реферат документа, состоящий из 2-х разделов: 1 - классического реферата и реферата в виде списка ключевых слов (по методу Sentence extraction); 2 – реферата, построенного с применением машинного обучения (ML).
* наличие средств сохранения в файл и распечатки полученной на выходе информации;
* интерфейс системы должен быть предельно простым и доступным для пользователей любого уровня, содержать понятный набор инструментов и средств, а также helpсредства.

**Код программы:**

import re

import nltk

import tkinter as tk

from tkinter import filedialog, Text, Frame, Label, Button, Radiobutton

from sklearn.feature\_extraction.text import TfidfVectorizer

from transformers import BartTokenizer, BartForConditionalGeneration

from docx import Document

import threading

nltk.download('punkt')

nltk.download('stopwords')

from nltk.corpus import stopwords

from nltk.tokenize import sent\_tokenize, word\_tokenize

# Функция для чтения текста из .docx

def read\_docx(docx\_file\_path):

    document = Document(docx\_file\_path)

    text\_content = [paragraph.text for paragraph in document.paragraphs]

    return '\n'.join(text\_content)

# Функция для очистки текста

def prepare\_text(text):

    text = re.sub(r'\s+', ' ', text)  # Удаление лишних пробелов

    text = re.sub(r'\d+', '', text)   # Удаление чисел

    return text

# TF-IDF метод для извлечения предложений

def extract\_sentences\_with\_tfidf(text):

    sentences = sent\_tokenize(text)

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

    cleaned\_sentences = [' '.join([word for word in word\_tokenize(sentence.lower())

                          if word.isalpha() and word not in stop\_words]) for sentence in sentences]

    vectorizer = TfidfVectorizer()

    tfidf\_matrix = vectorizer.fit\_transform(cleaned\_sentences)

    return sentences, tfidf\_matrix.sum(axis=1)  # type: ignore

def create\_summary(sentences, sentence\_weights, max\_sentences=10):

    ranked\_sentences = sorted(((weight, sentence) for sentence, weight in zip(sentences, sentence\_weights)), reverse=True)

    summary = ' '.join([sentence for weight, sentence in ranked\_sentences[:max\_sentences]])

    return summary

def create\_ml\_summary(text):

    if not isinstance(text, str):  # Убедимся, что это строка

        raise ValueError("Входные данные должны быть строкой")

    model\_name = "facebook/bart-large-cnn"  # Модель BART для сводки текста

    model = BartForConditionalGeneration.from\_pretrained(model\_name)

    tokenizer = BartTokenizer.from\_pretrained(model\_name)

    # Преобразуем текст в формат, понятный модели BART

    inputs = tokenizer(text, return\_tensors="pt", max\_length=1024, truncation=True)

    # Генерируем сводку

    summary\_ids = model.generate(inputs['input\_ids'], max\_length=150, min\_length=40, length\_penalty=2.0, num\_beams=4, early\_stopping=True)

    # Декодируем сводку обратно в текст

    summary = tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)

    return summary

# Обработка документа и вывод результата

def process\_docx(file\_path, chosen\_method):

    text\_content = read\_docx(file\_path)

    cleaned\_text = prepare\_text(text\_content)

    if chosen\_method == "TF":

        sentences, sentence\_weights = extract\_sentences\_with\_tfidf(cleaned\_text)

        summary = create\_summary(sentences, sentence\_weights)

    elif chosen\_method == "ML":

        summary = create\_ml\_summary(cleaned\_text)

    return text\_content, summary

# Выбор файла

def select\_file():

    file\_path = filedialog.askopenfilename(filetypes=[("Document files", "\*.docx")])

    if file\_path:

        file\_selector.set(file\_path)

        text\_content = read\_docx(file\_path)

        original\_text\_area.delete(1.0, tk.END)

        original\_text\_area.insert(tk.END, text\_content)

# Запуск процесса

def start\_summarization():

    threading.Thread(target=summarize).start()

def summarize():

    if file\_selector.get():

        method\_choice = summarization\_method.get()

        original\_text\_area.delete(1.0, tk.END)

        summary\_area.delete(1.0, tk.END)

        try:

            original\_text, summary = process\_docx(file\_selector.get(), method\_choice)

            # Выводим исходный текст

            original\_text\_area.insert(tk.END, original\_text)

            # Выводим сводку

            summary\_area.insert(tk.END, summary)

        except Exception as e:

            summary\_area.insert(tk.END, f"Ошибка при создании сводки: {e}")

# Создание главного окна приложения

main\_window = tk.Tk()

main\_window.title("Конспект")

main\_window.configure(bg="#F0F0F0")

file\_selector = tk.StringVar()

summarization\_method = tk.StringVar(value="TF")

# Поля для текста

original\_text\_label = Label(main\_window, text="Исходный текст:", bg="#F0F0F0", fg="#333333", font=("Arial", 12, "bold"))

original\_text\_label.pack()

original\_text\_area = Text(main\_window, height=10, width=80, bg="#E8F0FE", fg="#333333", font=("Arial", 10))

original\_text\_area.pack(pady=5)

summary\_label = Label(main\_window, text="Сводка:", bg="#F0F0F0", fg="#333333", font=("Arial", 12, "bold"))

summary\_label.pack()

summary\_area = Text(main\_window, height=10, width=80, bg="#E8F0FE", fg="#333333", font=("Arial", 10))

summary\_area.pack(pady=5)

# Создаём фреймы

frame2 = Frame(main\_window, bg="#F0F0F0")

frame2.pack(pady=10)

frame1 = Frame(main\_window, bg="#F0F0F0")

frame1.pack(pady=10)

# Кнопка выбора файла

select\_file\_button = Button(frame1, text="Выбрать файл (docx-файл)", command=select\_file, bg="#4A90E2", fg="white", font=("Arial", 10))

select\_file\_button.pack(side="top", padx=5, pady=5)

file\_label = Label(frame1, textvariable=file\_selector, bg="#F0F0F0", fg="#333333", font=("Arial", 10))

file\_label.pack(side="top")

# Переключатели для выбора метода

tfidf\_radio = Radiobutton(frame2, text="TF", variable=summarization\_method, value="TF", bg="#F0F0F0", fg="#333333",

                          activebackground="#F0F0F0", selectcolor="#4A90E2", font=("Arial", 10))

tfidf\_radio.pack(anchor="w")

ml\_radio = Radiobutton(frame2, text="ML", variable=summarization\_method, value="ML", bg="#F0F0F0", fg="#333333",

                       activebackground="#F0F0F0", selectcolor="#4A90E2", font=("Arial", 10))

ml\_radio.pack(anchor="w")

# Кнопка для создания сводки

start\_button = Button(main\_window, text="Создать сводку", command=start\_summarization, bg="#4A90E2", fg="white", font=("Arial", 10))

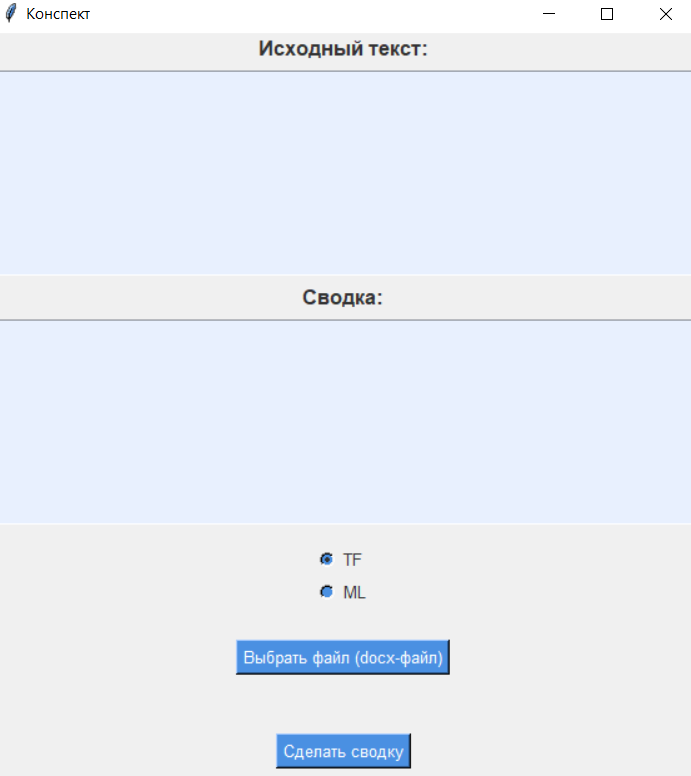
start\_button.pack(pady=5)

# Запуск основного цикла приложения

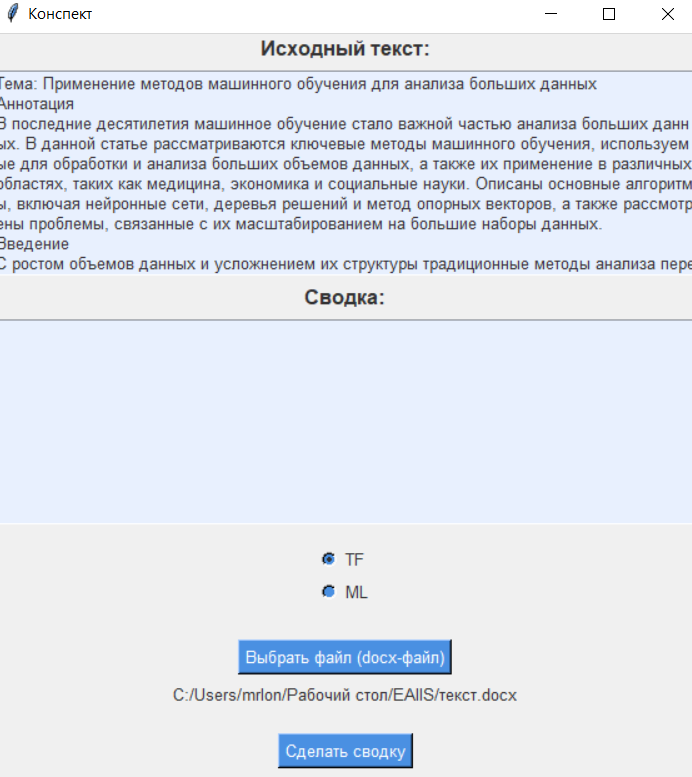
main\_window.mainloop()

**Результат:**

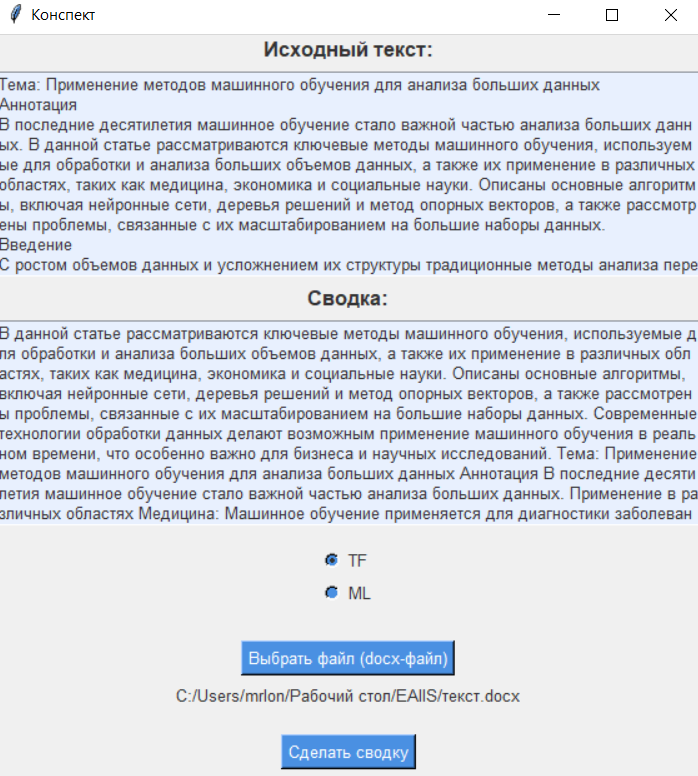
**Исходное главное меню:**



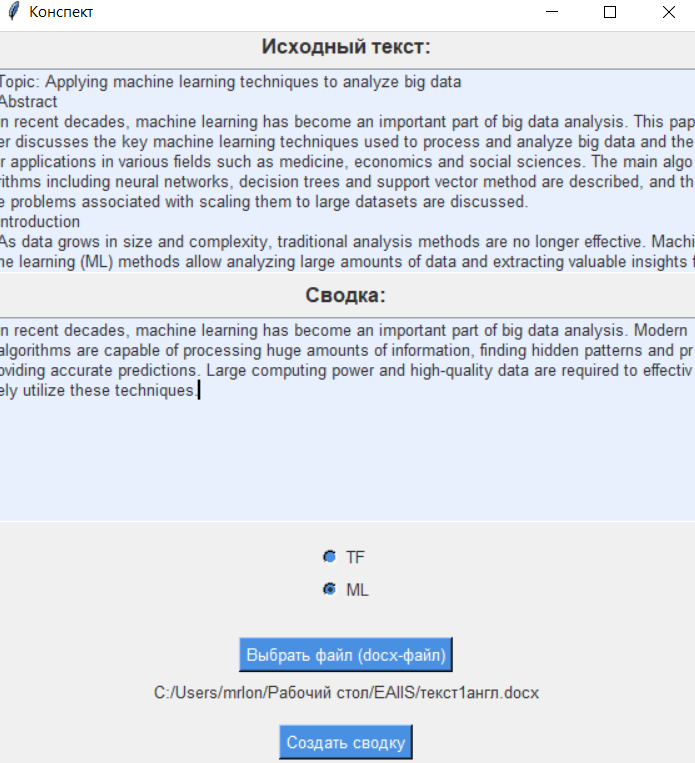
**Выбран файл:**



**Метод Sentence extraction:**



**Метод ML:**



**Вывод:** в ходе выполнения лабораторной работы освоил на практике основные принципы автоматического реферирования документов.