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):
```

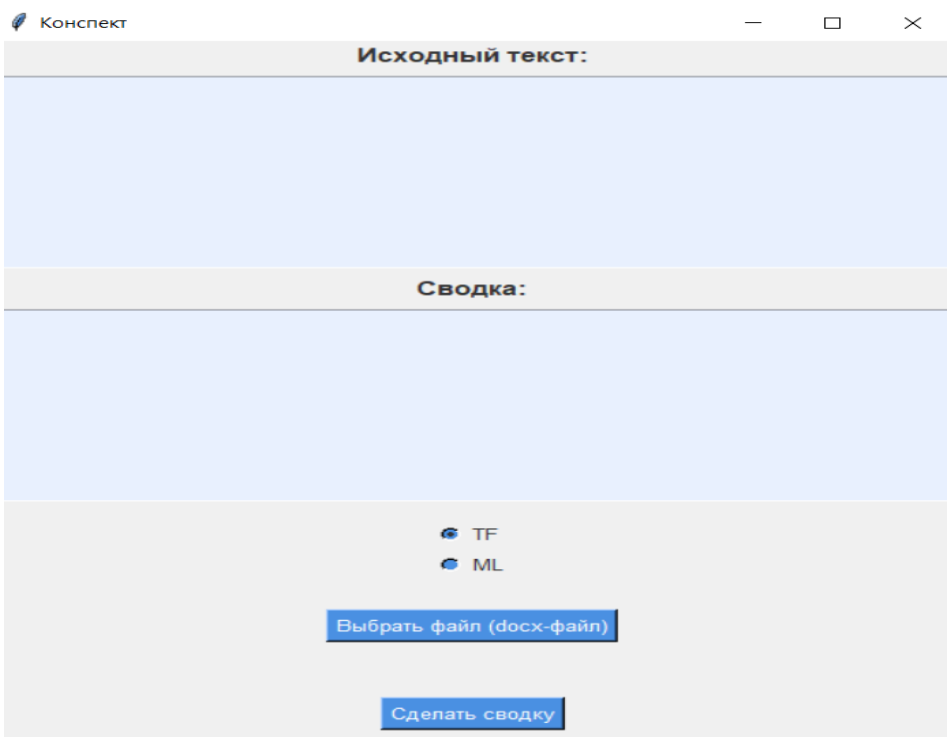
```
    text = re.sub(r'\s+', ' ', text)  # Удаление лишних пробелов
    text = re.sub(r'\d+', '', text)   # Удаление чисел
    return text

# 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)
```

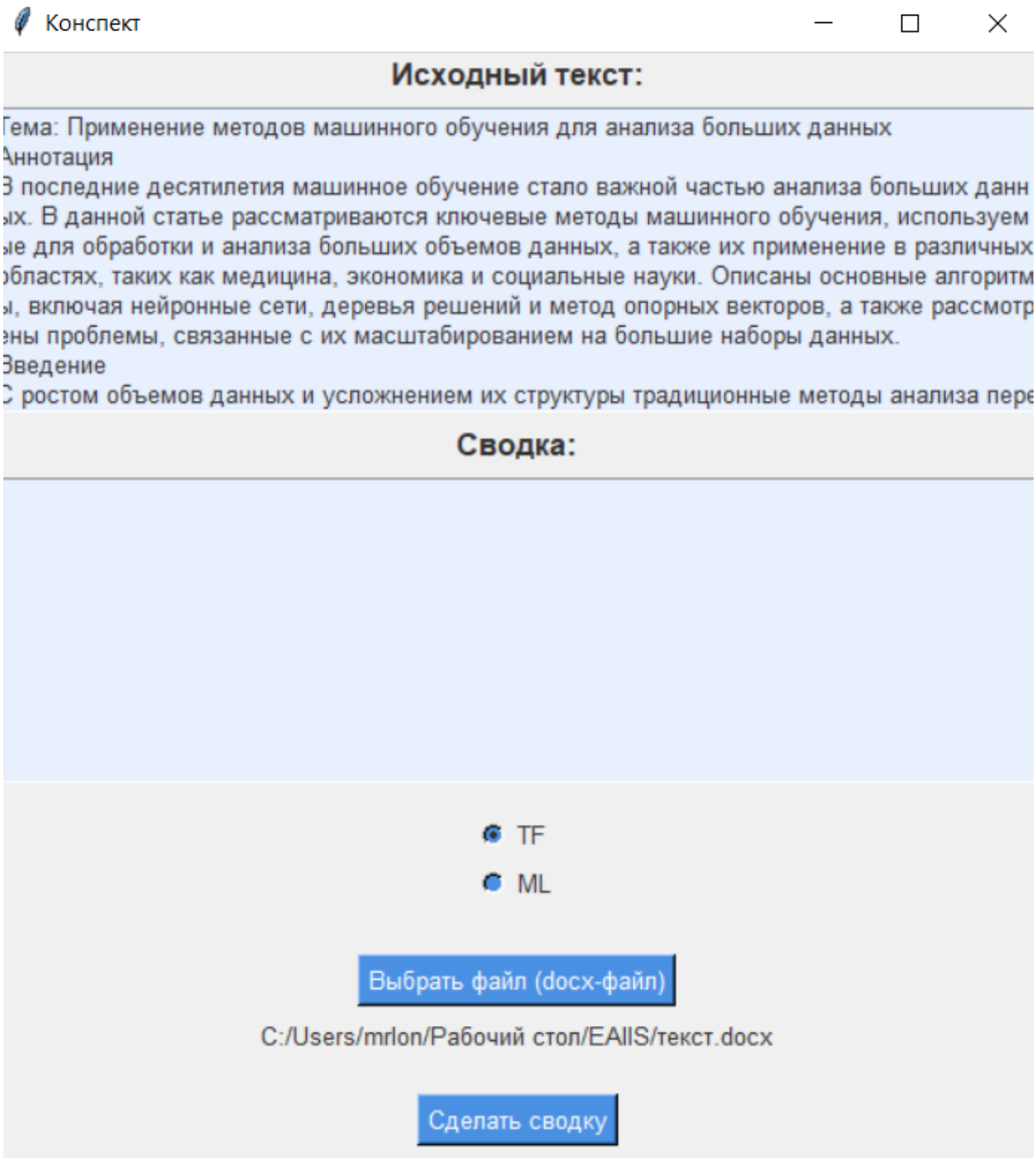
```
summary_label = Label(main_window, text="Сводка:", bg="#F0F0F0", fg="#333333",
font=("Arial", 12, "bold"))
summary_label.pack()

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:

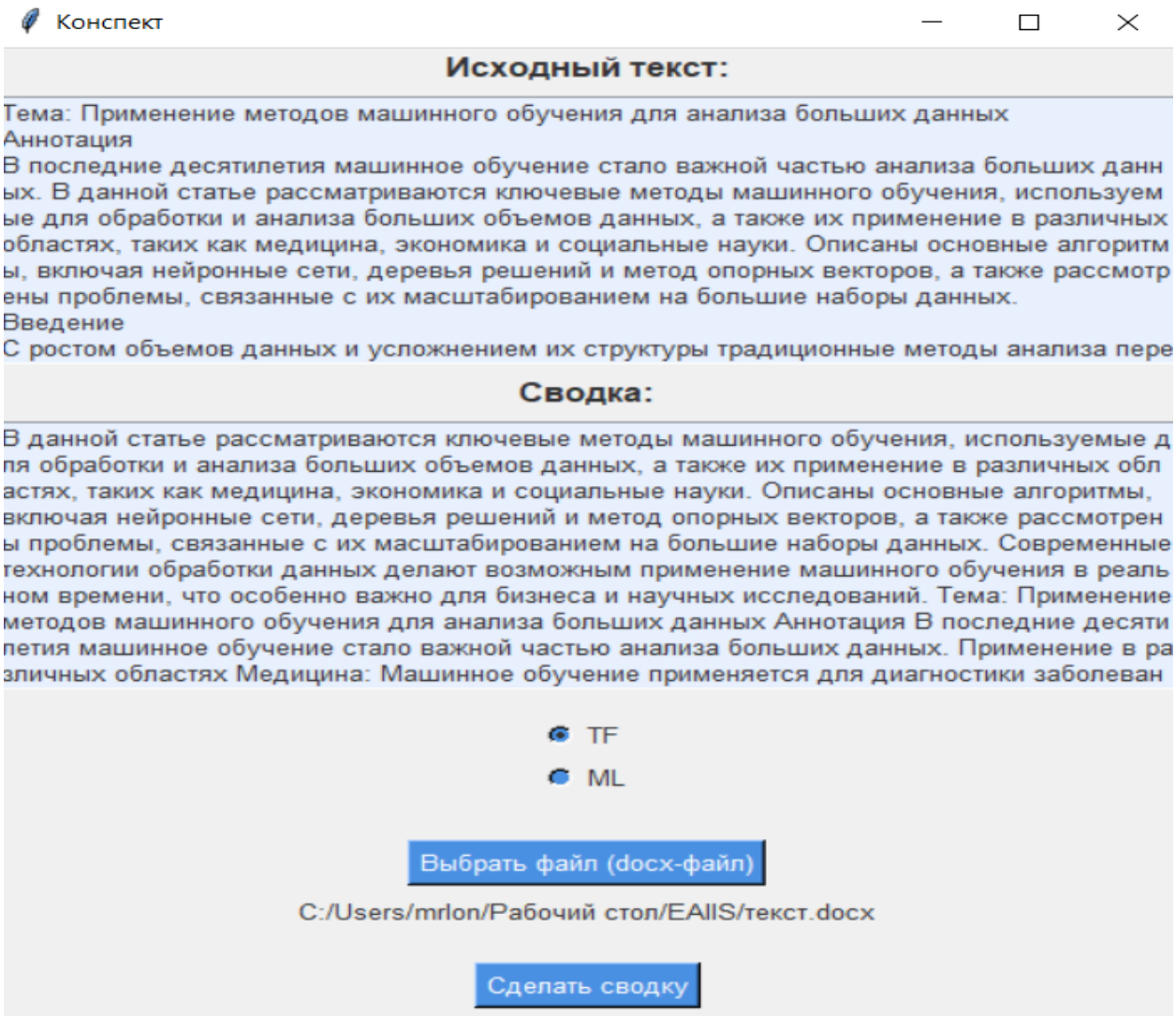
Исходное главное меню:


Конспект
Исходный текст:
Сводка:
TF
ML
Выбрать файл (docx-файл)
Сделать сводку

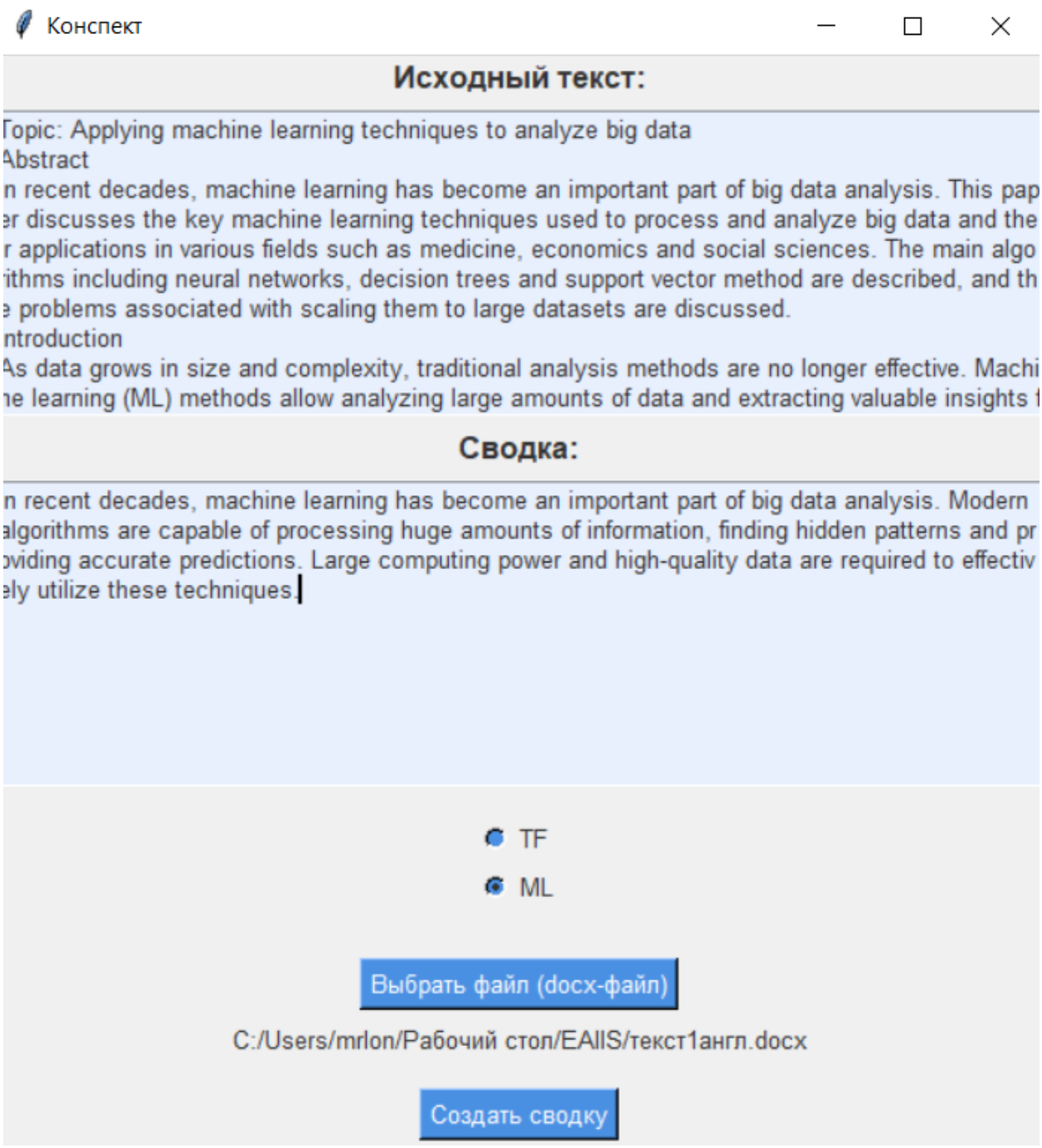
Выбран файл:



Метод Sentence extraction:


Конспект
Исходный текст:
Тема: Применение методов машинного обучения для анализа больших данных
Аннотация
В последние десятилетия машинное обучение стало важной частью анализа больших данн
ых. В данной статье рассматриваются ключевые методы машинного обучения, используем
ые для обработки и анализа больших объемов данных, а также их применение в различных
областях, таких как медицина, экономика и социальные науки. Описаны основные алгоритм
ы, включая нейронные сети, деревья решений и метод опорных векторов, а также рассмотр
ены проблемы, связанные с их масштабированием на большие наборы данных.
Введение
С ростом объемов данных и усложнением их структуры традиционные методы анализа пере
Сводка:
В данной статье рассматриваются ключевые методы машинного обучения, используемые д
ля обработки и анализа больших объемов данных, а также их применение в различных обл
астях, таких как медицина, экономика и социальные науки. Описаны основные алгоритмы,
включая нейронные сети, деревья решений и метод опорных векторов, а также рассмотрен
ы проблемы, связанные с их масштабированием на большие наборы данных. Современные
технологии обработки данных делают возможным применение машинного обучения в реаль
ном времени, что особенно важно для бизнеса и научных исследований. Тема: Применение
методов машинного обучения для анализа больших данных Аннотация В последние десяти
летия машинное обучение стало важной частью анализа больших данных. Применение в ра
зличных областях Медицина: Машинное обучение применяется для диагностики заболеван
TF
ML
Выбрать файл (docx-файл)
C:/Users/mrlon/Рабочий стол/EAIIS/текст.docx
Сделать сводку

Метод ML:



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