

**РЕФЕРАТ**

Курсовая работа: 28 стр., 5 ч., 15 рис., 3 источн., 1 прил.

НОРМАЛИЗАЦИЯ ТЕКСТА, PYTHON, ОБРАБОТКА ТЕКСТОВ, УДАЛЕНИЕ ДУБЛИКАТОВ,СТАТИСТИКА ТЕКСТОВ, АЛГОРИТМЫ ЕСТЕСТВЕННО-ЯЗЫКОВОЙ ОБРАБОТКИ.

Цель работы: Разработка и обучение модели машинного обучения для автоматической классификации спам-сообщений на русском языке, с последующей интеграцией в веб-приложение для практического применения.

Объектом исследования является процесс автоматической классификации текстовых сообщений и его влияние на фильтрацию информационного шума.

Предметом исследования являются алгоритмы машинного обучения и обработки естественного языка, применяемые для классификации спам-сообщений, включая использование модели RuBERT и методов дообучения на специализированных датасетах.

Методологической основой исследования является использование алгоритмов трансформеров и предобработки текста, в частности, дообучение модели RuBERT на собственных данных, а также интеграция обученной модели в веб-приложение с визуализацией результатов и возможностью ввода пользовательских сообщений.

Научная новизна работы заключается в создании прикладного решения на основе модели RuBERT, способного эффективно распознавать спам-сообщения на русском языке. Разработано веб-приложение, позволяющее проводить классификацию в реальном времени с возможностью тестирования и сохранения истории классификации.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc197992485)

[1 Теоретическая часть 5](#_Toc197992486)

[1.2 Применение методов машинного обучения 5](#_Toc197992487)

[1.3 Архитектура модели BERT и её русскоязычные версии 5](#_Toc197992488)

[1.4 Особенности обработки текстов перед обучением моделей 6](#_Toc197992489)

[1.5 Выбор и подготовка датасета 6](#_Toc197992490)

[1.6 Дообучение модели трансформера 6](#_Toc197992491)

[1.7 Практическая реализация классификатора 6](#_Toc197992492)

[1.8 Заключение 7](#_Toc197992493)

[2. Основы построения модели классификации текста 8](#_Toc197992494)

[2.1 Основные принципы классификации текстов 8](#_Toc197992495)

[2.2 Введение в трансформеры 8](#_Toc197992496)

[2.3 Архитектура BERT 8](#_Toc197992497)

[2.4 RuBERT — русскоязычная версия BERT 9](#_Toc197992498)

[2.5 Принцип работы модели на практике 9](#_Toc197992499)

[2.6 Преимущества использования трансформеров 10](#_Toc197992500)

[2 Области применения изученных методов 11](#_Toc197992501)

[3 Разработка приложения 12](#_Toc197992502)

[4.1 Архитектура и используемые технологии 12](#_Toc197992503)

[4.2 Описание приложения 13](#_Toc197992504)

[4 Исследование 13](#_Toc197992505)

[5.1 Приведение к нижнему регистру 13](#_Toc197992506)

[5.2 Удаление пунктуации 14](#_Toc197992507)

[5.3 Лемматизация 15](#_Toc197992508)

[5.4 Удаление стоп-слов 15](#_Toc197992509)

[5.5 Удаление дубликатов 16](#_Toc197992510)

[5.6 Нормализация даты 17](#_Toc197992511)

[ЗАКЛЮЧЕНИЕ 20](#_Toc197992512)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21](#_Toc197992513)

[ПРИЛОЖЕНИЕ А 22](#_Toc197992514)

# ВВЕДЕНИЕ

Актуальность темы исследования обусловлена постоянным ростом объёма текстовой информации, поступающей через электронную почту, мессенджеры и социальные сети. Существенную долю этой информации составляют спам-сообщения, которые не только затрудняют восприятие полезных данных, но и могут содержать вредоносный контент. Автоматическая классификация таких сообщений становится критически важной задачей для повышения информационной безопасности и удобства пользователей.

Современные методы машинного обучения, особенно основанные на трансформерных архитектурах, таких как BERT и его русскоязычная версия RuBERT, позволяют достигать высокой точности в задачах анализа текста. Использование таких моделей даёт возможность не только эффективно распознавать спам, но и адаптировать их под специфические задачи путём дообучения на пользовательских данных.

Цель данной работы — разработка и обучение модели для автоматической классификации сообщений по признаку «спам / не спам» на русском языке, с последующей интеграцией решения в веб-приложение. Приложение должно обеспечивать удобный интерфейс для ввода текстов, отображения результата и тестирования модели на реальных примерах.

Объектом исследования является процесс классификации текстовых сообщений, а предметом — алгоритмы и модели машинного обучения, применяемые для решения этой задачи.

Научная новизна работы заключается в адаптации и дообучении модели RuBERT под задачу спам-фильтрации, а также создании функционального веб-интерфейса, реализующего классификацию в интерактивном режиме.

# 1 Теоретическая часть

**1.1 Актуальность задачи классификации текстов**

Классификация текстовых сообщений представляет собой важное направление в области обработки естественного языка. В условиях стремительного увеличения объёма информации возрастает необходимость в автоматической фильтрации сообщений, особенно в отношении спама. Спам-сообщения не только перегружают пользователей ненужной информацией, но и могут содержать ссылки на вредоносные ресурсы. Это делает задачу их своевременного обнаружения особенно актуальной.

## 1.2 Применение методов машинного обучения

Современные методы машинного обучения позволяют достигать высокой точности в задачах классификации текстов. Особенно эффективными оказались нейросетевые модели трансформерного типа, такие как BERT. Благодаря возможности дообучения на новых датасетах эти модели могут быть адаптированы под конкретные задачи, включая фильтрацию спама на русском языке.

## 1.3 Архитектура модели BERT и её русскоязычные версии

Модель BERT построена на архитектуре трансформеров и предназначена для обработки контекстуальных зависимостей в тексте. Её русскоязычная модификация RuBERT была создана на основе исходной модели и обучена на большом корпусе текстов на русском языке. Это позволяет использовать её для решения различных задач NLP, включая классификацию, с высокой точностью.

## 1.4 Особенности обработки текстов перед обучением моделей

Для достижения стабильной работы модели необходимо предварительно подготовить текстовые данные. Предобработка включает в себя перевод на нужный язык (если необходимо), очистку текста, нормализацию регистра, а также токенизацию. В задачах классификации текстов также важно наличие разметки данных — наличие меток, указывающих на принадлежность текста к классу спам или не спам.

## 1.5 Выбор и подготовка датасета

Качество модели напрямую зависит от используемого датасета. Для задач классификации спама могут использоваться как открытые наборы данных, так и собственные коллекции сообщений. В данном проекте использован англоязычный датасет, который был переведён на русский язык для последующего обучения модели RuBERT. Данные представляли собой пары: текст сообщения и его метка (спам или не спам).

## 1.6 Дообучение модели трансформера

Дообучение модели заключается в обучении предобученной нейросети на новой задаче классификации с использованием размеченного датасета. Процесс включает настройку гиперпараметров, таких как количество эпох, размер батча и скорость обучения. После дообучения модель способна проводить классификацию новых текстов с высокой точностью.

## 1.7 Практическая реализация классификатора

Результатом обучения модели становится возможность её использования в приложении, позволяющем пользователю вводить текст и получать классификацию. Для этой цели может быть создано веб-приложение с простым интерфейсом. Это делает разработку не только исследовательским проектом, но и прикладным инструментом, пригодным для использования на практике.

## 1.8 Заключение

Методы трансформеров и архитектура BERT, а также её адаптация для русского языка в виде модели RuBERT, позволяют эффективно решать задачу автоматической классификации текстов. Предобработка данных, выбор качественного датасета и правильное дообучение модели являются ключевыми этапами в создании классификатора спам-сообщений.

# Основы построения модели классификации текста

## 2.1 Основные принципы классификации текстов

Классификация текстов — это задача машинного обучения, при которой каждому текстовому фрагменту присваивается одна из заранее определённых категорий. В случае спам-фильтрации категориями являются «спам» и «не спам». Классификация выполняется на основе содержимого текста, с учётом распределения слов, фраз и их контекста. Важную роль в такой задаче играет предварительная обработка текста и выбор подходящей модели.

## 2.2 Введение в трансформеры

Трансформеры — это архитектура нейросетей, предназначенная для работы с последовательностями, прежде всего текстом. В отличие от рекуррентных сетей (RNN), трансформеры обрабатывают весь текст одновременно, используя механизм внимания (self-attention), который позволяет модели учитывать контекст каждого слова по отношению ко всем остальным словам в предложении.

Главное преимущество трансформеров — это способность учитывать длинные зависимости в тексте, что особенно важно для понимания смысла и контекста сообщений.

## 2.3 Архитектура BERT

BERT (Bidirectional Encoder Representations from Transformers) — это модель, разработанная компанией Google, которая обучается предсказывать пропущенные слова в тексте и учитывать контекст слева и справа одновременно. Она состоит из нескольких энкодеров трансформеров, каждый из которых обрабатывает входной текст, представленный в виде токенов, и формирует векторные представления (эмбеддинги), которые далее используются для выполнения конкретных задач.

Особенностью BERT является предобучение на огромных объемах текстов, после чего модель может быть дообучена под конкретную задачу (например, классификацию).

## 2.4 RuBERT — русскоязычная версия BERT

RuBERT — это модификация BERT, адаптированная для работы с русским языком. Модель была обучена на русском корпусе текстов (включая Wikipedia, новостные сайты и соцсети) и является наиболее популярной трансформерной моделью для русского NLP.

Для дообучения RuBERT на задачу классификации спама используются размеченные примеры сообщений. После дообучения модель может с высокой точностью предсказывать, является ли новое сообщение спамом.

## 2.5 Принцип работы модели на практике

На вход модель получает текст сообщения, который предварительно токенизируется — разбивается на составные части (токены). Эти токены преобразуются в числовые векторы с помощью эмбеддингов и подаются на вход трансформеру. На выходе получается вектор признаков, который затем подаётся на линейный классификатор, обученный отличать спам от не спама.

В процессе дообучения модель настраивает веса нейронной сети таким образом, чтобы минимизировать ошибку классификации. После завершения обучения модель сохраняется и используется в приложении для обработки новых сообщений.

## 2.6 Преимущества использования трансформеров

Использование моделей на базе трансформеров, таких как RuBERT, обеспечивает высокую точность классификации, устойчивость к неструктурированным данным и возможность адаптации под специфические задачи. Благодаря поддержке библиотек transformers и torch такие модели легко интегрируются в Python-приложения и позволяют решать задачи анализа текста на практике.

.

# Области применения изученных методов

Методы нормализации текста находят применение в различных задачах обработки текстовой информации.

В информационном поиске они используются для приведения запросов и текстов к единому формату, что повышает релевантность результатов.

В анализе тональности нормализация помогает очищать текст от лишних символов и повторов, делая анализ более точным, особенно в данных из социальных сетей и отзывов.

Для классификации текста нормализация играет важную роль в предварительной обработке, например, при определении категорий или фильтрации спама.

В коммерческой сфере нормализация используется для обработки клиентских отзывов, заявок и запросов, облегчая анализ уникальных сообщений и ключевых терминов.

Методы также востребованы в системах фильтрации контента для выявления нежелательных сообщений.

# Разработка приложения

## 4.1 Архитектура и используемые технологии

Приложение для классификации спам-сообщений реализовано с использованием архитектурного шаблона MVC (Model-View-Controller), который обеспечивает логическое разделение логики работы, отображения и управления данными. Это упрощает сопровождение и масштабирование проекта.

Язык программирования: Python

Основной фреймворк: Dash (на базе Flask)

Дополнительные библиотеки: transformers, torch, pandas, plotly, dash-bootstrap-components

Компоненты приложения:

1. Файл model.py содержит реализацию загрузки и использования дообученной модели на базе RuBERT. Модель получает текст на вход, проводит токенизацию и возвращает метку — «спам» или «не спам».
2. Файл controller.py реализует логику взаимодействия между моделью и интерфейсом. Он получает текст, передаёт его в модель и возвращает предсказание для отображения пользователю.
3. Файл view.py содержит элементы пользовательского интерфейса, включая поле ввода текста, кнопку запуска классификации, и область для отображения результата. Используются компоненты Dash и Bootstrap для стилизации.
4. Файл app.py объединяет все модули, запускает сервер и обеспечивает маршрутизацию. Он подключает компоненты интерфейса и настраивает обратные вызовы (callback-функции), чтобы обновлять результат при вводе текста.

## 4.2 Описание приложения

После запуска пользователь получает доступ к веб-интерфейсу. В интерфейсе доступно поле для ввода сообщения и кнопка запуска классификации. При нажатии кнопки сообщение передаётся в модель, которая определяет, является ли оно спамом. Результат отображается под полем ввода. Также в приложении может вестись лог запросов и отображаться история.

# Исследование

Для проверки работоспособности программы мы проведем нормализацию по пунктам, применяя каждый метод отдельно.

Исходный текст: «Кто из вас знает, как решить эту задачу? Это очень важно для следующей встречи на конференции.»

Для уточнения стоит отметить, что текст берется из файла JSON, который был подвергнут парсингу, в ходе которого из него был изъят текст сообщения.

Файл JSON включает в себя id, date, sender\_id, text.

## Приведение к нижнему регистру

Ожидаемый результат: «кто из вас знает, как решить эту задачу? это очень важно для следующей встречи на конференции».

Фактический результат показан на рисунке 7.

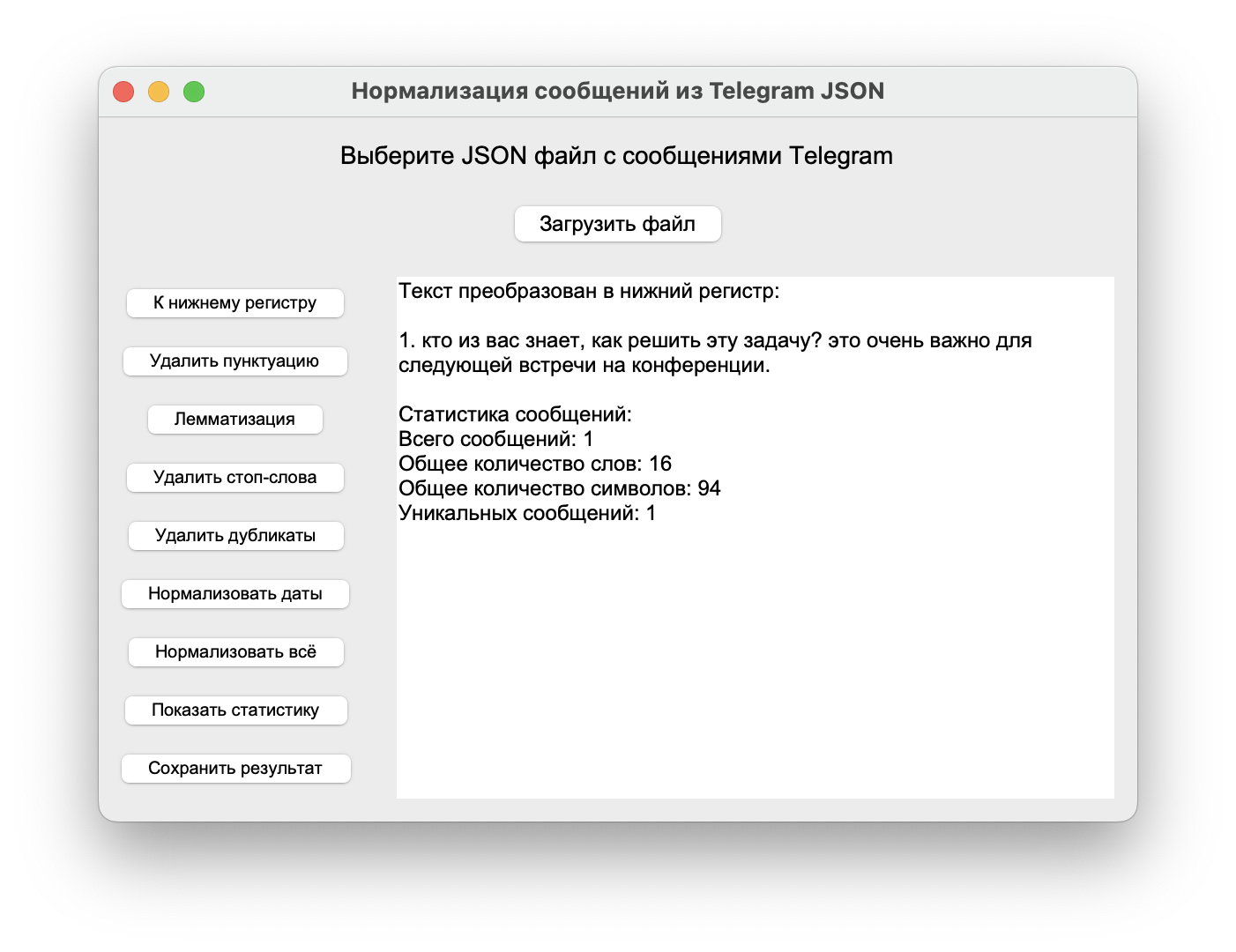


Рисунок 7 – Тест 1

## Удаление пунктуации

Ожидаемый результат: удаляться все знаки препинания.

Фактический результат показан на рисунке 8.

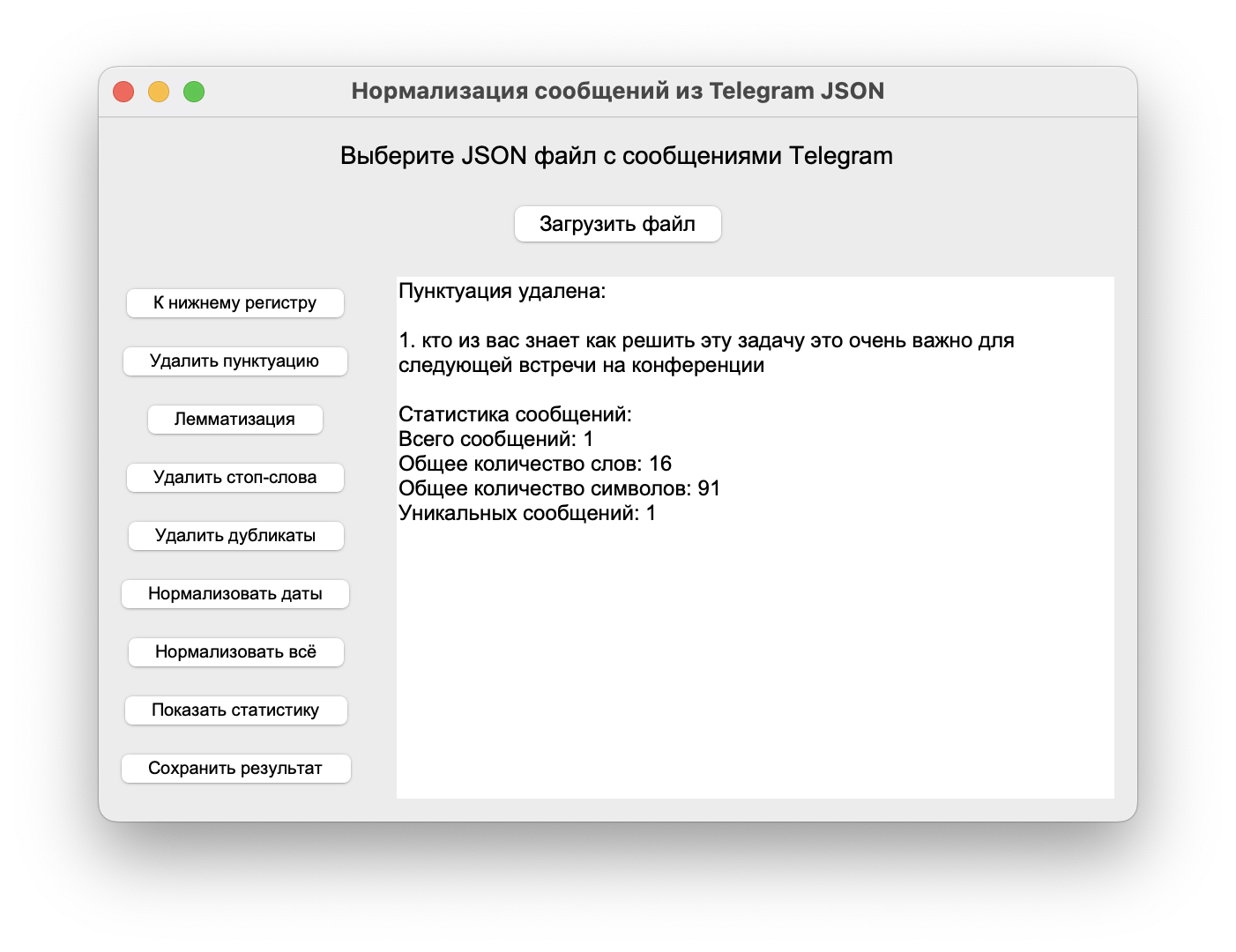


Рисунок 8 – Тест 2

## Лемматизация

Ожидаемый результат: слова преобразуются в свою начальную форму.

Фактический результат показан на рисунке 9.

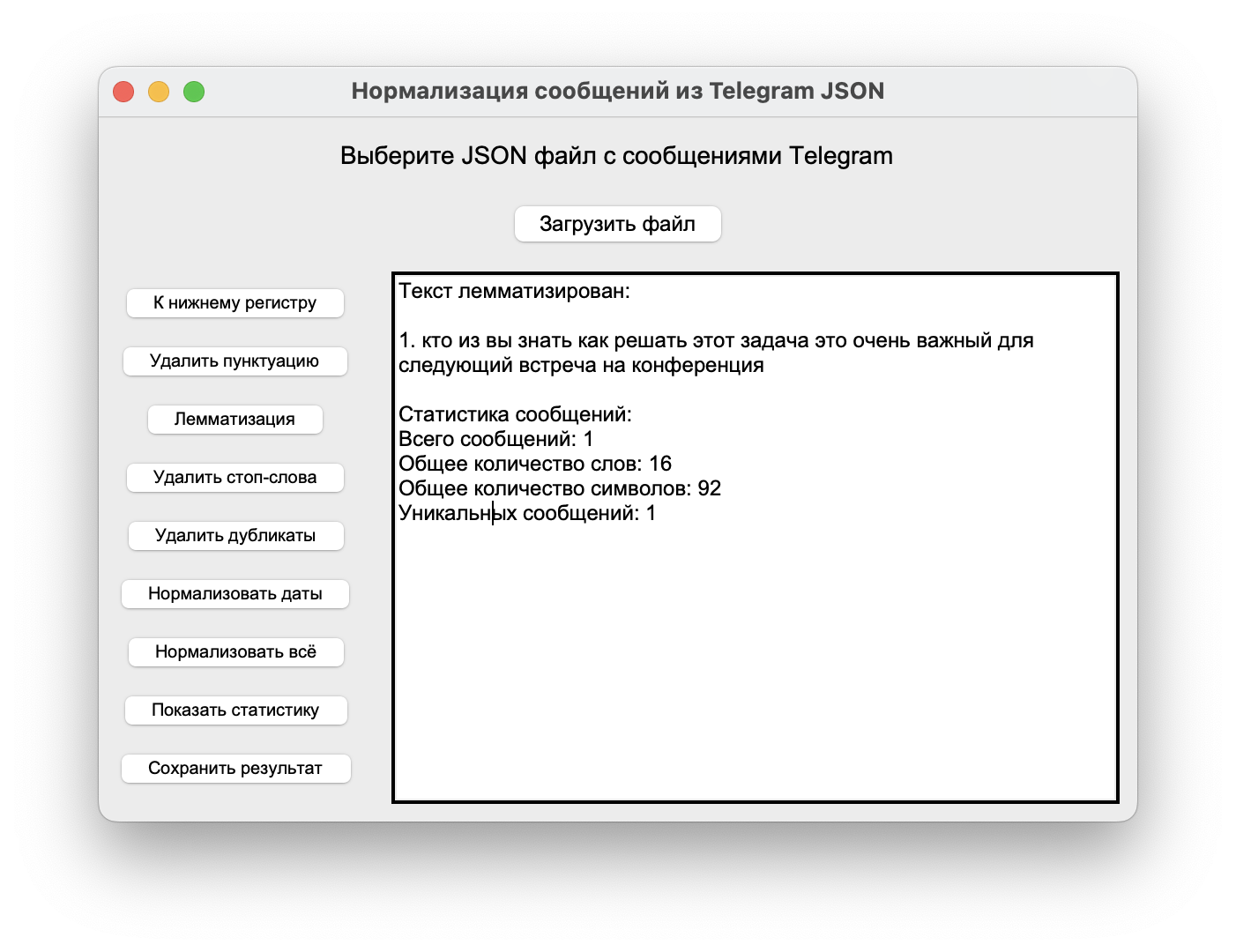


Рисунок 9 – Тест 3

## Удаление стоп-слов

Ожидаемый результат: удаляться стоп-слова.

Фактический результат показан на рисунке 10.

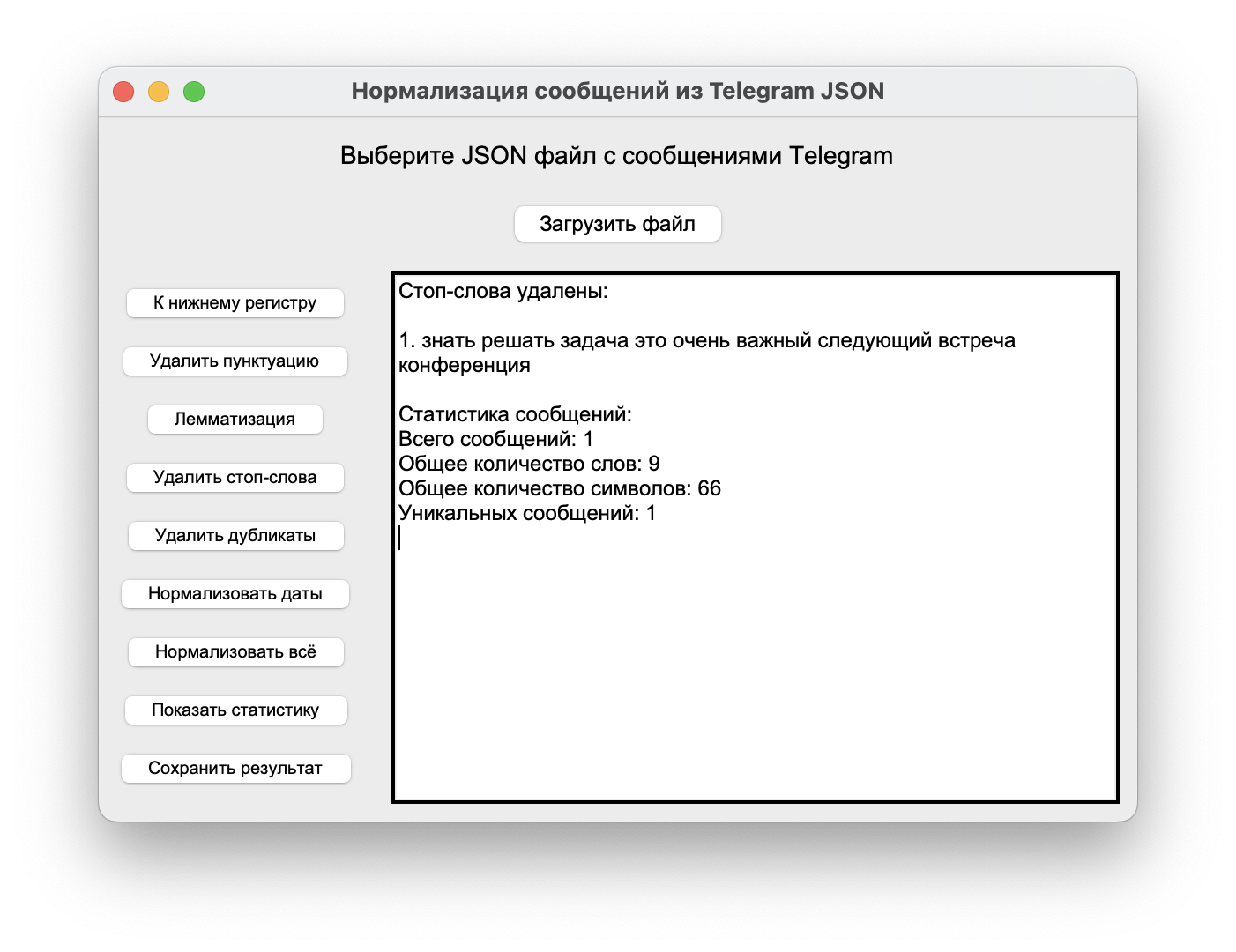


Рисунок 10 – Тест 4

## Удаление дубликатов

Ожидаемый результат: повторяющиеся слова удалятся.

Фактический результат показан на рисунке 11.

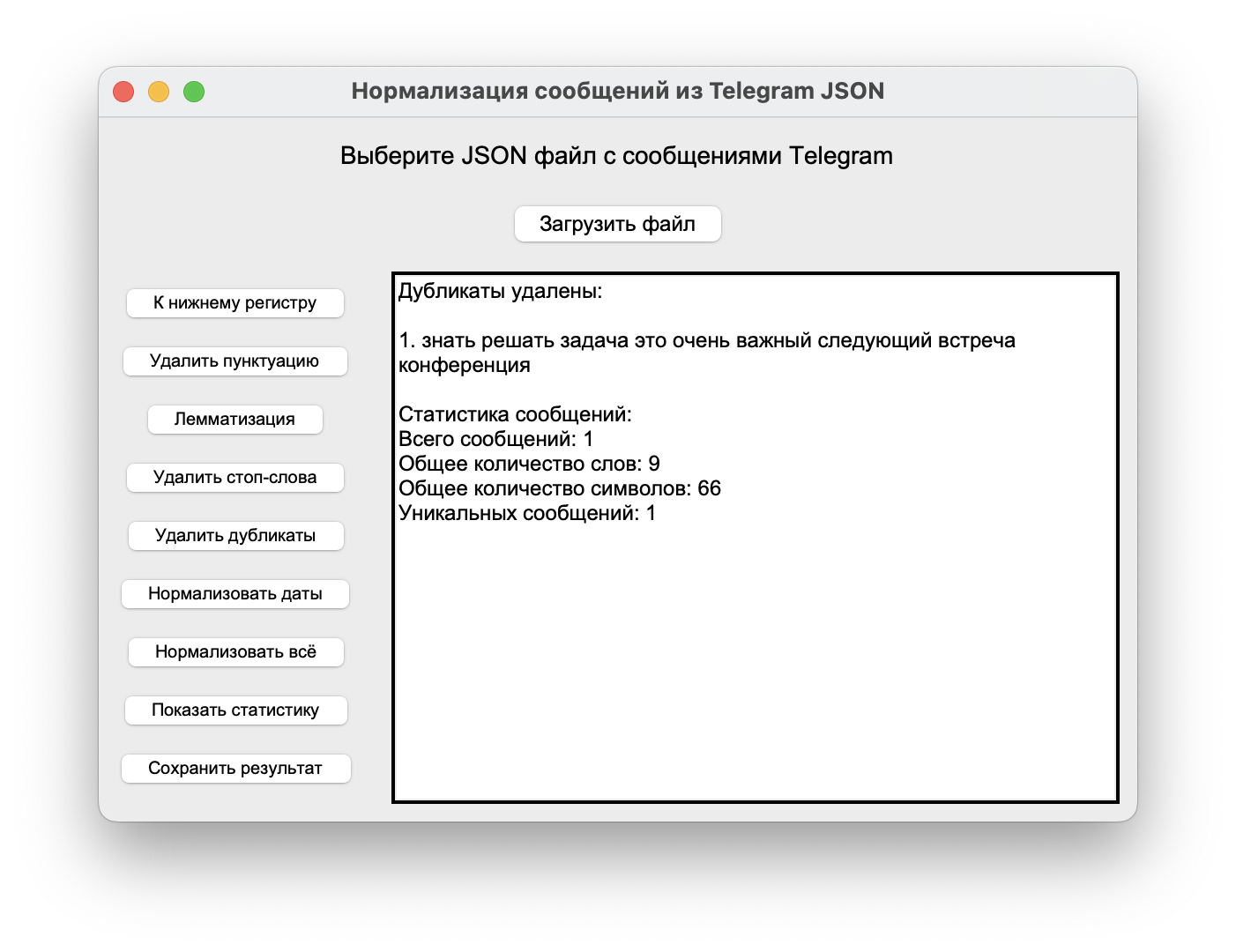


Рисунок 11 – Тест 5

## Нормализация даты

Для проверки данного метода, загрузим файл с разными форматами записи дат

Исходный текст показан на рисунке 12.

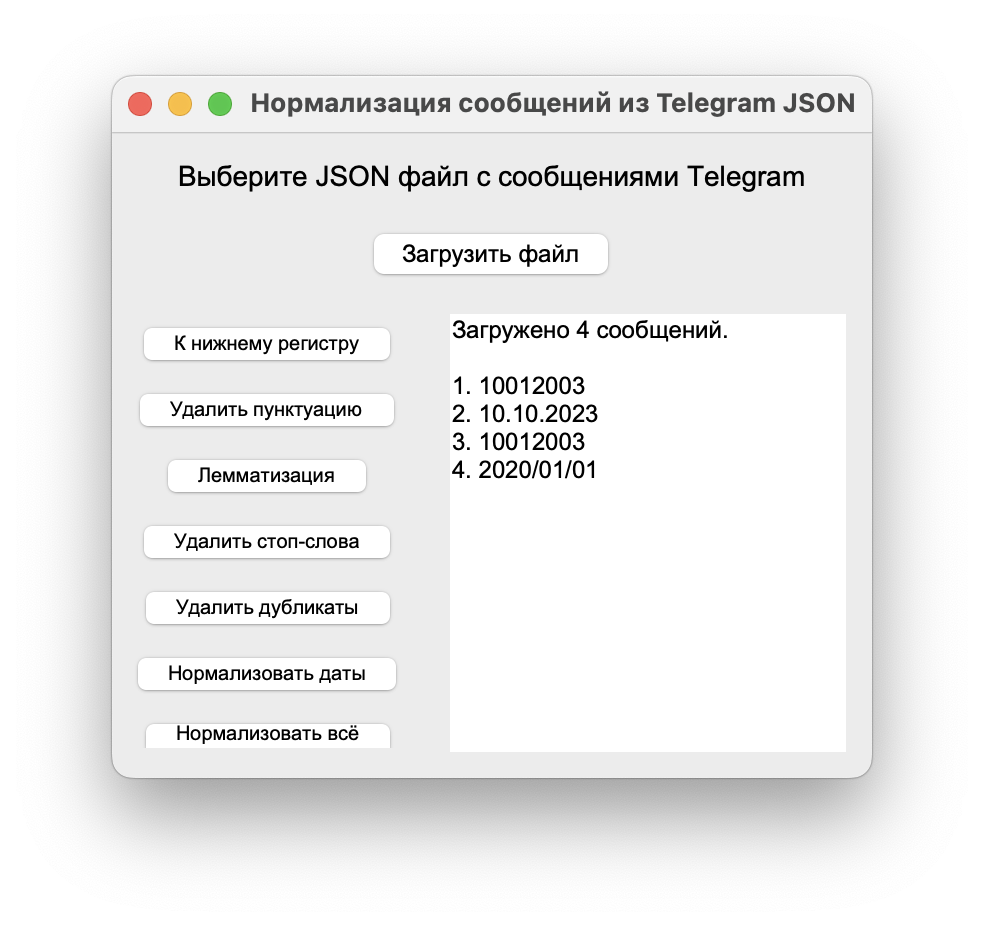


Рисунок 12 – Загруженные даты

Фактический результат показан на рисунке 13.

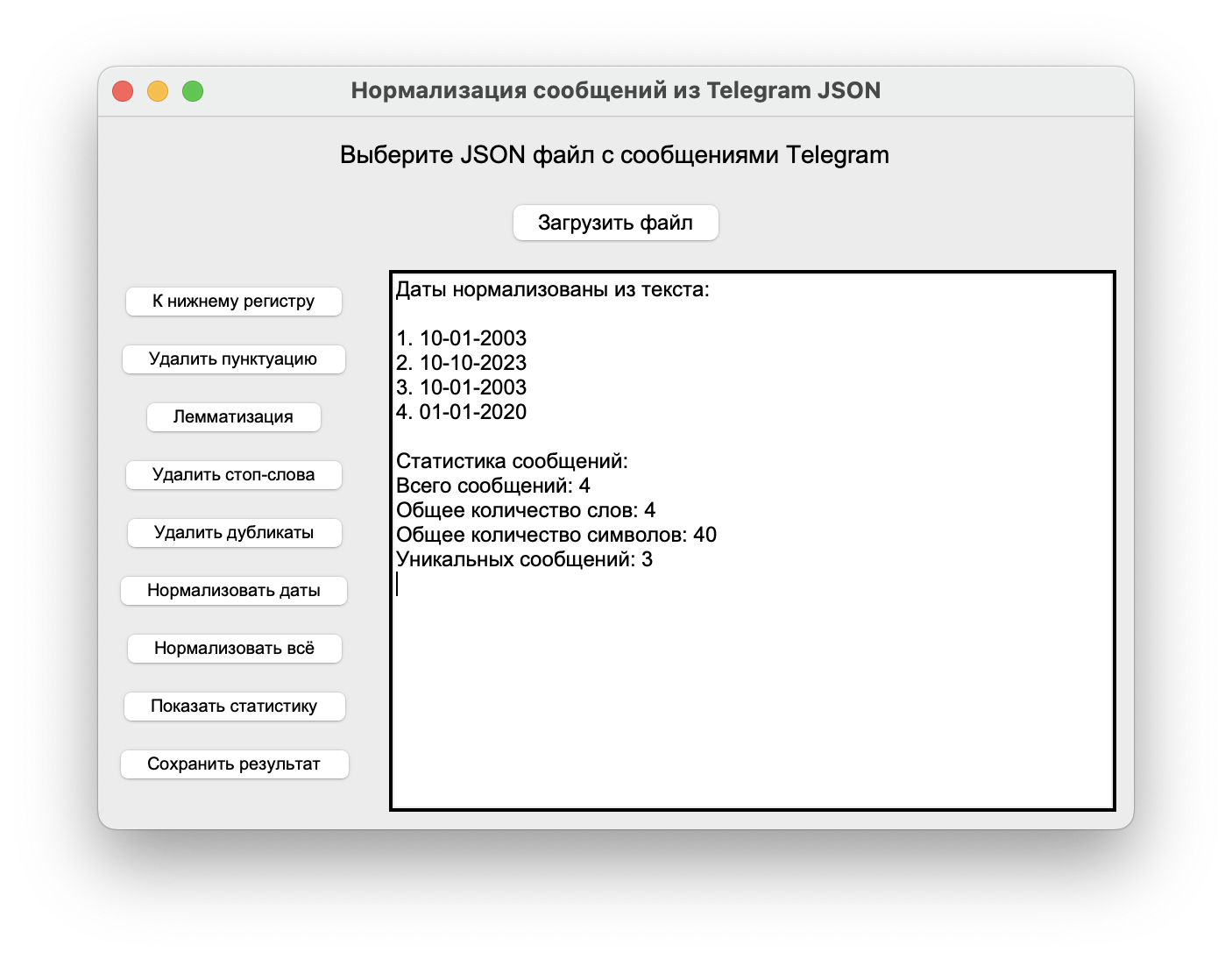


Рисунок 13 – Тест 6

В ходе проверок методов нормализации стало понятно, что все методы работают корректно.

Далее проверим работу программы на разных языках и текстах, включающих в себя ссылки, эмодзи, ненужные символы, HTML теги.

Исходные данные на рисунке 14:

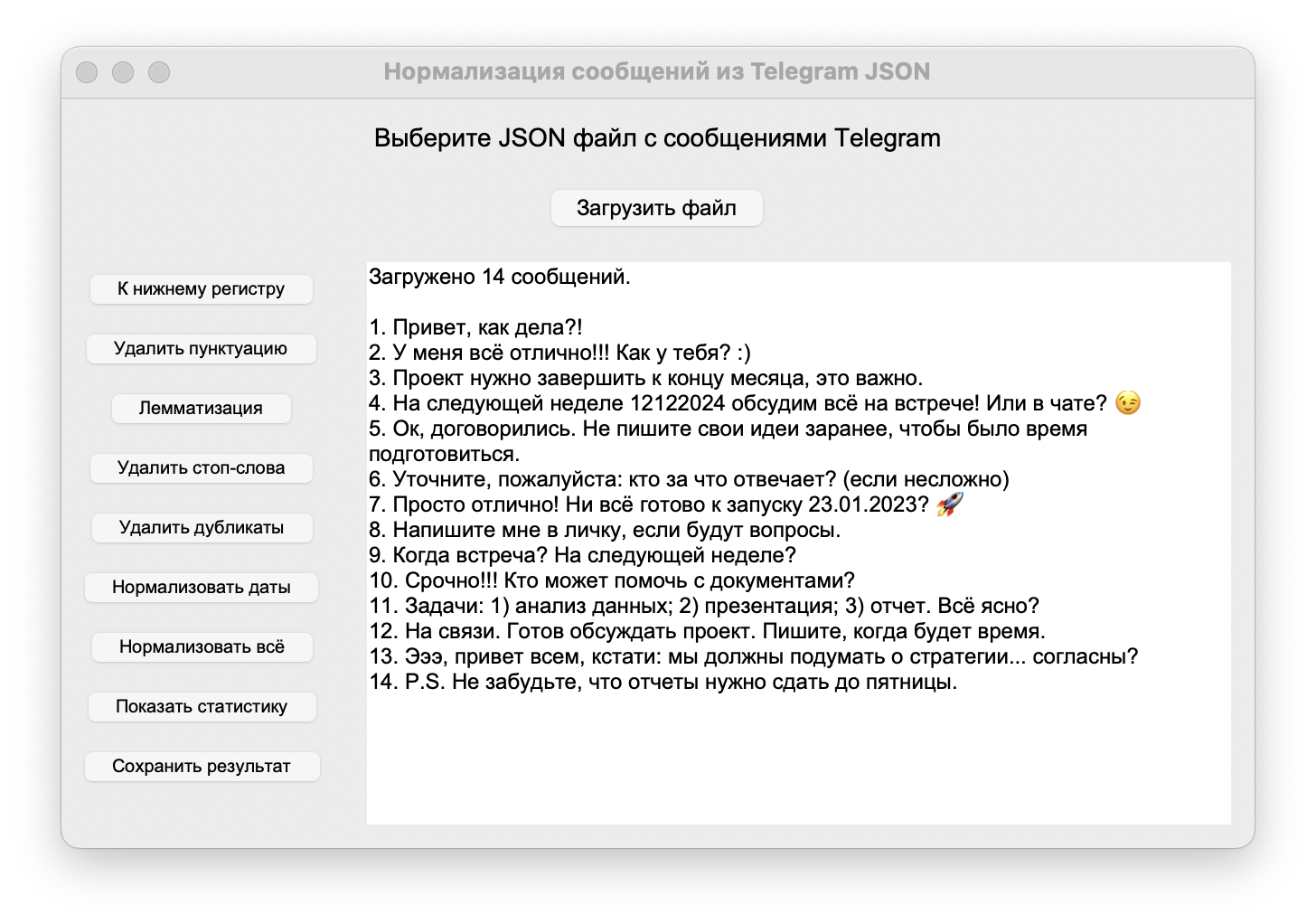


Рисунок 14 – Исходные данные

Фактический результат показан на рисунке 15.

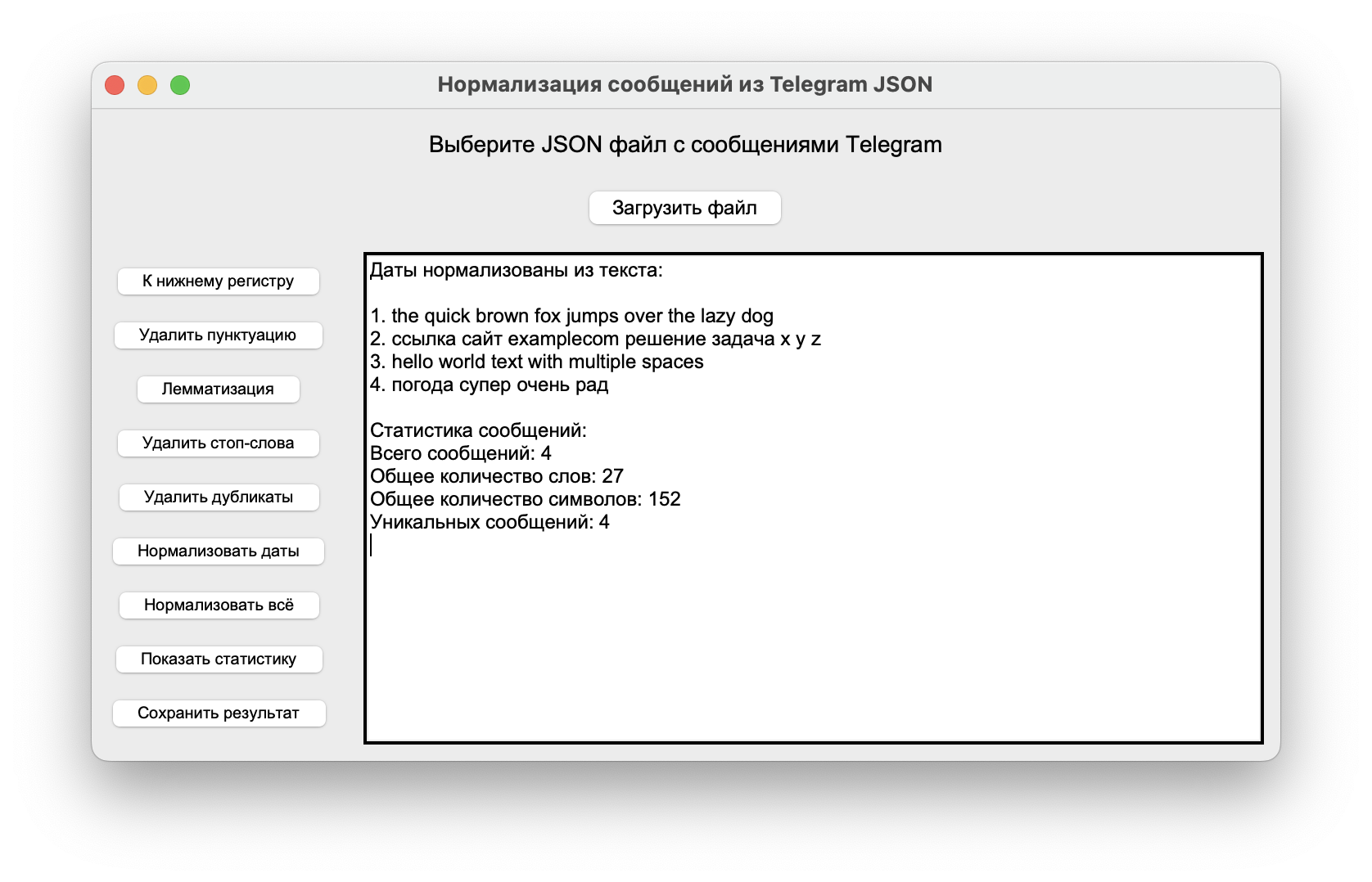


Рисунок 15 – Тест 7

Можно заметить, что удаление стоп-слов на английском языке не выполняется, так как для работы был выбран русский язык. Однако при необходимости удалять стоп-слова английского языка можно использовать ту же библиотеку NLTK, добавив соответствующий список.

Также были удалены HTML-теги, эмодзи, ненужные символы и знаки препинания. Из этого можно сделать вывод, что использование методов, таких как приведение к нижнему регистру, удаление пунктуации, нормализация данных и удаление лишних символов, не зависит от языка, на котором написаны сообщения.

Каждый метод нормализации текста успешно прошел тестирование. Ожидаемые результаты совпали с фактическими. Некоторые методы показали универсальность и применимость к текстам на разных языках и с различными форматами записи данных.

Все методы нормализации способствовали улучшению структуры текста, снижению объёма данных и упрощению их анализа. Это подтверждено тестированием на реальных данных.

# ЗАКЛЮЧЕНИЕ

В рамках работы был изучен принцип нормализации информации с использованием различных методов обработки естественного языка. Особое внимание было уделено языку программирования Python и его библиотекам, предназначенным для работы с текстами.

По результатам исследования было разработано приложение, которое позволяет пользователю нормализовать текст из сообщений различных чатов

В дальнейшем планируется расширение функциональности приложения, интеграцию дополнительных методов анализа, а также оптимизацию работы с большими объёмами данных.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 **Вишняков, Ю.М** Об "измерении" смысловой близости текстов в рамках вычислительной теории семантической интерпретации / Ю.М. Вишняков, Р.Ю. Вишняков . —­ Текст: непосредственный.

2 **Вишняков, Ю.М.** «Вычислительное» представление смысла текстовых фрагментов на основе обратной польской записи / Ю.М. Вишняков, Р.Ю. Вишняков // Известия ЮФУ. Технические науки. Тематический выпуск «Интеллектуальные САПР». – Таганрог: Изд-во ЮФУ. – 2013. – №7 (144). – с.141-147. / Ю.М. Вишняков, Р.Ю. Вишняков. —­ Текст: непосредственный.

3 **Курейчик, В. В.** Основные подходы к извлечению текстовой информации / В. В. Курейчик. — Известия ЮФУ. Технические науки. — Таганрог: Южный федеральный университет. — 2024. — с. 6-14. — Текст: непосредственный.

# ПРИЛОЖЕНИЕ А

**Основная программа**

Файл normalize\_main.py

from collections import OrderedDict  
import re  
import json  
from datetime import datetime  
from tkinter import Tk, filedialog, Text, Label, Button, Scrollbar, END, Frame  
from nltk.corpus import stopwords  
from pymystem3 import Mystem  
  
mystem = Mystem()  
stop\_words = set(stopwords.words('russian'))  
  
*# Функции нормализации*def to\_lowercase(text):  
 return text.lower() if text else ""  
  
def remove\_punctuation(text):  
 return re.sub(r'[^\w\s]', '', text) if text else ""  
  
def lemmatize\_text(text):  
 if not text:  
 return ""  
 lemmatized\_words = mystem.lemmatize(text)  
 return ''.join(lemmatized\_words).strip()  
  
def remove\_stopwords(text):  
 if not text:  
 return ""  
 words = text.split()  
 filtered\_words = [word for word in words if word not in stop\_words]  
 return ' '.join(filtered\_words)  
  
  
def normalize\_date\_from\_text(text):  
 *# Поиск различных форматов даты* patterns = [  
 r'\b(\d{4})[-/](\d{2})[-/](\d{2})\b', *# Формат YYYY-MM-DD или YYYY/MM/DD* r'\b(\d{2})[.](\d{2})[.](\d{4})\b', *# Формат DD.MM.YYYY* r'\b(\d{8})\b' *# Формат DDMMYYYY* ]  
  
 for pattern in patterns:  
 match = re.search(pattern, text)  
 if match:  
 try:  
 if len(match.groups()) == 3: *# Если дата в формате с разделителями* if pattern == r'\b(\d{4})[-/](\d{2})[-/](\d{2})\b':  
 date\_obj = datetime.strptime(f"{match.group(1)}-{match.group(2)}-{match.group(3)}", '%Y-%m-%d')  
 elif pattern == r'\b(\d{2})[.](\d{2})[.](\d{4})\b':  
 date\_obj = datetime.strptime(f"{match.group(1)}.{match.group(2)}.{match.group(3)}", '%d.%m.%Y')  
 elif len(match.groups()) == 1: *# Если дата в формате DDMMYYYY* date\_obj = datetime.strptime(match.group(0), '%d%m%Y')  
  
 return text.replace(match.group(0), date\_obj.strftime('%d-%m-%Y'))  
 except ValueError:  
 continue  
  
 return text *# Если дата не найдена или не удалось преобразовать*class TelegramChatApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Нормализация сообщений из Telegram JSON")  
 self.root.geometry("800x600")  
  
 self.label = Label(root, text="Выберите JSON файл с сообщениями Telegram", font=("Arial", 14))  
 self.label.pack(pady=10)  
  
 *# Кнопка загрузки файла* self.load\_button = Button(root, text="Загрузить файл", command=self.load\_file, font=("Arial", 12))  
 self.load\_button.pack(pady=5)  
  
 *# Боковая панель с кнопками нормализации* button\_frame = Frame(root)  
 button\_frame.pack(side="left", fill="y", padx=10, pady=10)  
  
 self.lowercase\_button = Button(button\_frame, text="К нижнему регистру", command=self.normalize\_lowercase,  
 state="disabled", font=("Arial", 10))  
 self.lowercase\_button.pack(pady=5)  
  
 self.punctuation\_button = Button(button\_frame, text="Удалить пунктуацию", command=self.normalize\_punctuation,  
 state="disabled", font=("Arial", 10))  
 self.punctuation\_button.pack(pady=5)  
  
 self.lemmatization\_button = Button(button\_frame, text="Лемматизация", command=self.normalize\_lemmatization,  
 state="disabled", font=("Arial", 10))  
 self.lemmatization\_button.pack(pady=5)  
  
 self.stopwords\_button = Button(button\_frame, text="Удалить стоп-слова", command=self.normalize\_stopwords,  
 state="disabled", font=("Arial", 10))  
 self.stopwords\_button.pack(pady=5)  
  
 self.deduplication\_button = Button(button\_frame, text="Удалить дубликаты", command=self.remove\_duplicates,  
 state="disabled", font=("Arial", 10))  
 self.deduplication\_button.pack(pady=5)  
  
 self.date\_button = Button(button\_frame, text="Нормализовать даты", command=self.normalize\_date,  
 state="disabled", font=("Arial", 10))  
 self.date\_button.pack(pady=5)  
  
 self.normalize\_all\_button = Button(button\_frame, text="Нормализовать всё", command=self.normalize\_all,  
 state="disabled", font=("Arial", 10))  
 self.normalize\_all\_button.pack(pady=5)  
  
 self.stats\_button = Button(button\_frame, text="Показать статистику", command=self.show\_statistics,  
 state="disabled", font=("Arial", 10))  
 self.stats\_button.pack(pady=5)  
  
 self.save\_button = Button(button\_frame, text="Сохранить результат", command=self.save\_messages,  
 state="disabled", font=("Arial", 10))  
 self.save\_button.pack(pady=5)  
  
 self.text\_area = Text(root, wrap="word", font=("Arial", 12))  
 self.text\_area.pack(fill="both", expand=True, padx=10, pady=10)  
  
 self.messages = []  
 self.processed\_messages = []  
  
 def load\_file(self):  
 file\_path = filedialog.askopenfilename(filetypes=[("JSON files", "\*.json")])  
 if not file\_path:  
 return  
  
 try:  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 data = json.load(file)  
 self.messages = [item['text'] for item in data if 'text' in item]  
 self.processed\_messages = self.messages[:]  
 self.text\_area.delete(1.0, END)  
 self.text\_area.insert(END, f"Загружено {len(self.messages)} сообщений.\n\n")  
 for idx, message in enumerate(self.messages, 1):  
 self.text\_area.insert(END, f"{idx}. {message}\n")  
  
 self.enable\_buttons()  
 except Exception as e:  
 self.text\_area.delete(1.0, END)  
 self.text\_area.insert(END, f"Ошибка загрузки файла: {e}")  
  
 def enable\_buttons(self):  
 self.lowercase\_button.config(state="normal")  
 self.punctuation\_button.config(state="normal")  
 self.lemmatization\_button.config(state="normal")  
 self.stopwords\_button.config(state="normal")  
 self.date\_button.config(state="normal")  
 self.deduplication\_button.config(state="normal")  
 self.normalize\_all\_button.config(state="normal")  
 self.stats\_button.config(state="normal")  
 self.save\_button.config(state="normal")  
  
 def normalize\_lowercase(self):  
 self.processed\_messages = [to\_lowercase(msg) for msg in self.processed\_messages]  
 self.display\_processed\_messages("Текст преобразован в нижний регистр:")  
  
 def normalize\_punctuation(self):  
 self.processed\_messages = [remove\_punctuation(msg) for msg in self.processed\_messages]  
 self.display\_processed\_messages("Пунктуация удалена:")  
  
 def normalize\_lemmatization(self):  
 self.processed\_messages = [lemmatize\_text(msg) for msg in self.processed\_messages]  
 self.display\_processed\_messages("Текст лемматизирован:")  
  
 def normalize\_stopwords(self):  
 self.processed\_messages = [remove\_stopwords(msg) for msg in self.processed\_messages]  
 self.display\_processed\_messages("Стоп-слова удалены:")  
  
 def remove\_duplicates(self):  
 seen = OrderedDict()  
 for msg in self.processed\_messages:  
 if msg not in seen:  
 seen[msg] = None  
 self.processed\_messages = list(seen.keys())  
 self.display\_processed\_messages("Дубликаты удалены:")  
  
 def normalize\_date(self):  
 self.processed\_messages = [normalize\_date\_from\_text(msg) for msg in self.processed\_messages]  
 self.display\_processed\_messages("Даты нормализованы из текста:")  
  
 def normalize\_all(self):  
 self.normalize\_lowercase()  
 self.normalize\_punctuation()  
 self.normalize\_lemmatization()  
 self.normalize\_stopwords()  
 self.remove\_duplicates()  
 self.normalize\_date()  
  
 def show\_statistics(self):  
 total\_messages = len(self.processed\_messages)  
 total\_words = sum(len(msg.split()) for msg in self.processed\_messages)  
 total\_characters = sum(len(msg) for msg in self.processed\_messages)  
 unique\_messages = len(set(self.processed\_messages))  
  
 self.text\_area.insert(END, "\nСтатистика сообщений:\n")  
 self.text\_area.insert(END, f"Всего сообщений: {total\_messages}\n")  
 self.text\_area.insert(END, f"Общее количество слов: {total\_words}\n")  
 self.text\_area.insert(END, f"Общее количество символов: {total\_characters}\n")  
 self.text\_area.insert(END, f"Уникальных сообщений: {unique\_messages}\n")  
  
 def display\_processed\_messages(self, header):  
 self.text\_area.delete(1.0, END)  
 self.text\_area.insert(END, f"{header}\n\n")  
 for idx, message in enumerate(self.processed\_messages, 1):  
 self.text\_area.insert(END, f"{idx}. {message}\n")  
 self.show\_statistics()  
  
 def save\_messages(self):  
 file\_path = filedialog.asksaveasfilename(defaultextension=".json", filetypes=[("JSON files", "\*.json")])  
 if not file\_path:  
 return  
  
 try:  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 json.dump([{"text": msg} for msg in self.processed\_messages], file, ensure\_ascii=False, indent=4)  
 self.text\_area.insert(END, f"\n\nРезультаты сохранены в файл: {file\_path}")  
 except Exception as e:  
 self.text\_area.insert(END, f"\n\nОшибка сохранения файла: {e}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = Tk()  
 app = TelegramChatApp(root)  
 root.mainloop()