

МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»
(РУТ(МИИТ))

Институт управления и цифровых технологий

Кафедра «Вычислительные системы, сети и информационная безопасность»

ОТЧЕТ ПО ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ
НА ТЕМУ:
ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗАТОР ТОНАЛЬНОСТИ ОТЗЫВОВ

Направление: 10.03.01 Информационная безопасность

Профиль: Безопасность компьютерных систем

Выполнили:
студенты группы УИБ-311
Макеев Д.В.
Павлов А.Р.

Наставник:
Малинский С.В.
(должность, ФИО)

МОСКВА 2024

АННОТАЦИЯ

Пояснительная записка 44 с., 32 рис., 4 приложения.

АНАЛИЗ ТОНАЛЬНОСТИ ОТЗЫВОВ, ОСВОЕНИЕ И НАСТРОЙКА ПО, ИНФОРМАТИВНОСТЬ СЛОВ, СОЗДАНИЕ СОБСТВЕННОЙ НЕЙРОННОЙ СЕТИ.

Объектом исследования являются нейронные сети.

Предмет исследования – анализ тональности текста.

В процессе поиска информации, для дальнейшего принятия решения по отношению к выбранному субъекту или объекту, важную роль играет мнение других людей. С большим темпом развиваются и создаются новые интернет-ресурсы: социальные сети, блоги, мобильные приложения и многие другие источники информации, где люди могут делиться своим мнением.

В рамках данной работы рассматривается проблема автоматического определения тональности текста, исследование существующих систем, подходов и методов для выявления положительной или отрицательной окраски текста.

Целью данной работы является разработка программного обеспечения для определения и анализа тональности текстов отзывов о кинофильмах.

В ходе работы было освоено и настроено ПО сторонних разработчиков, созданы и настроены собственные ПО, произведено сравнение, определение наиболее эффективной ПО среди разработанных, а также определены пути их совершенствования.

ОБЩЕЕ ЗАДАНИЕ НА ПРОЕКТНУЮ ДЕЯТЕЛЬНОСТЬ

1. Подготовка данных;
2. Определение информативности слов;
3. Создание собственных ПО;
4. Настройка собственных ПО;
5. Сравнение собственных ПО;
6. Разработка интеллектуального анализатора

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 ПРОБЛЕМА АНАЛИЗА ТОНАЛЬНОСТИ ОТЗЫВОВ.....	7
2 ФОРМИРОВАНИЕ И ПОДГОТОВКА ДАННЫХ ДЛЯ МАШИННОГО ОБУЧЕНИЯ ПРОГРАММЫ АНАЛИЗА ТОНАЛЬНОСТИ ТЕКСТА	9
3 СОЗДАНИЕ И НАСТРОЙКА НЕЙРОННЫХ СЕТЕЙ.....	10
3.1 Многослойный персептрон	10
3.2 Разработка нейронной сети на основе многослойного персептрона	12
3.3 Анализ результата работы программы	14
3.4 Алгоритм распознавания. Рекуррентная нейронная сеть.	15
3.5 Разработка нейронной сети (RNN)	15
3.6 Анализ результатов работы программы	21
3.7 Искусственный нейрон	23
3.8 Разработка собственной нейронной сети на основе искусственного нейрона	24
3.9 Анализ результата работы программы	27
4 СРАВНЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ ТРЕХ МОДЕЛЕЙ НЕЙРОННЫХ СЕТЕЙ	28
5 РАЗРАБОТКА ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗАТОРА НА ОСНОВЕ ЛУЧШЕЙ МОДЕЛИ	29
ЗАКЛЮЧЕНИЕ	31
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	33
ПРИЛОЖЕНИЕ А	34
ПРИЛОЖЕНИЕ Б.....	36
ПРИЛОЖЕНИЕ В	38
ПРИЛОЖЕНИЕ Г.....	41

ВВЕДЕНИЕ

Автоматическая классификация эмоциональной окраски текстов, также известная под термином «анализ тональности», с каждым годом становится все более актуальной задачей и с теоретической, и с практической точек зрения. В первую очередь, это связано с развитием интернета и изменением формата коммуникаций в современном мире – для подавляющего большинства людей социальные сети стали занимать лидирующее положение среди остальных источников информации и площадок для дискуссий.

Пользователями социальных сетей ежедневно генерируются значительные объемы текстовой информации. Многие компании стали использовать социальные сети для продвижения своих продуктов и услуг, а также оценки репутации своего бренда путем анализа комментариев своих клиентов в сети. Однако увеличивающиеся с каждым годом развития интернета объемы данных, создаваемые пользователями сети, поставили перед исследователями в области обработки естественного языка вопрос автоматизации анализа текстов, написанных человеком.

Текстам в социальных сетях более характерен разговорный стиль речи. Как следствие, это вызывает серию существенных трудностей при автоматической обработке, так как в разговорном стиле чаще встречаются сленг, фразеологизмы, авторская пунктуация, опечатки и ошибки, а также другие стилистические особенности, которые сложно обрабатывать в автоматическом режиме.

Учитывая вышеперечисленное, есть потребность в разработке программного обеспечения, для выполнения автоматического анализа предложенной обществом информации для выявления отношения людей к данным товарам и услугам. Чтобы распознать мнение, изложенное пользователем в своем тексте, то есть выявить отрицательную или положительную окраску текста, требуется выполнить анализ тональности текста. Сентимент анализ (англ. sentimentanalysis, анализ тональности текста,

эмоциональная окраска текста) - обработка естественного языка (англ. NLP, natural language processing), цель которого является извлечение эмоционального содержания из текста.

1 ПРОБЛЕМА АНАЛИЗА ТОНАЛЬНОСТИ ОТЗЫВОВ

Отзывы занимают основное место почти во всех областях человеческой деятельности. Отзывы и связанные с ними понятия, такие как чувства, оценки, отношения и эмоции являются предметом изучения анализа настроений (sentiment analysis). Зарождение и быстрое развитие этой области связано с интересами людей в Интернете, как правило, спрос всегда порождает предложение. В качестве примера можно привести различные отзывы, форумы, обсуждения, блоги, социальные сети. Впервые в человеческой истории мы имеем огромный объем отзывов, записанных в цифровом формате. С начала XXI века сфера анализа тональности данных стала одной из наиболее активно развивающихся и исследуемых направлений в области обработки естественных языков.

Анализ тональности текста – обработка естественного языка, классифицирующая тексты по эмоциональной окраске. Такой анализ можно рассматривать как метод количественного описания качественных данных, с присвоением оценок настроения. Целью является нахождение мнений в тексте и определение их свойств. Например, необходимо выявить то, о чем ведется речь – объект разговора и отношение субъекта к нему – определение тональности. Для этого нужно понять смысл текста, что является весьма непростой задачей.

Задача определения эмоциональной окраски текста является задачей классификации, она может быть бинарной (негативный, позитивный), тернарной (негативный, нейтральный, позитивный) или n-арной (например: сильно негативный, умеренной негативный, нейтральный, умеренно позитивный, сильно позитивный).

При решении задачи возникает ряд трудностей, он может быть связан с порядком слов в предложении, омонимичностью слов, орфографическими и синтаксическими ошибками, все это может в корне менять смысл, а

следовательно, и эмоциональную окраску. В общем случае тональность текста весьма субъективна, но ее анализ находит много полезных применений.

Несмотря на достаточно большое количество работ и систем, проведенных в сфере анализа тональности текста, точность определения сентимент анализа не находится на совершенном уровне. Ниже представлены проблемы, которые часто возникают при разработке и использовании автоматических алгоритмов определения тональности русскоязычного текста:

- 1 При проведении анализа, необходимо учитывать предметную область. Для получения высоких результатов требуются более высокоточные алгоритмы. Например, провести анализ тональности продукции или услуги намного легче, нежели анализировать область политики, т.к. там существенно больше различной терминологии и словосочетаний;
- 2 В процессе выполнения анализа исследователи сталкиваются с рядом таких проблем как: синтаксические и грамматические, ошибки, сокращения слов, сленги, которые в дальнейшем играют большую роль на качество определения тональности;
- 3 Выявление сарказма и иронии – одна из больших проблем сентимент анализа, т. к. текста и предложения, где содержатся вышеперечисленные составляющие, могут быть по смыслу отрицательными, но сам текст, написан в положительном ключе;
- 4 Противительные союзы, могут изменить всю тональность предложения. В русском языке существует небольшое количество союзов такого типа: а, да (в значении но), зато, но, однако;
- 5 Использование смайликов, при написании. Зачастую, смайлики не соответствуют тексту, что может привести к неправильной оценке тональности текста;
- 6 Применение машинного перевода, может привести к искажению смысла исходного текста, что повлияет на качество определения тональности.

2 ФОРМИРОВАНИЕ И ПОДГОТОВКА ДАННЫХ ДЛЯ МАШИННОГО ОБУЧЕНИЯ ПРОГРАММЫ АНАЛИЗА ТОНАЛЬНОСТИ ТЕКСТА

Процессом подготовки данных для машинного обучения, реализующего анализ тональности текста, является сбор данных. Для начала был собран большой набор данных, состоящий из текстовых сообщений. В случае с анализом тональности текста отзывов о компьютерной технике были использованы сервисы Яндекс.Маркет, DNS, МВидео для сбора информации. На основе данных отзывов была составлена база данных, включающая в себя тысячи отзывов и десятки тысяч слов, на основе которых модель машинного обучения может быть настроена и обучена.

Для реализации была выбрана база данных PostgreSQL, которая содержит весь необходимый функционал, является простой в настройке и модификации, что было использовано при подключении к интеллектуальной системе анализа тональности.

id	user_name	review	tech_name
1	Андрей	Этот смартфон имеет потрясающее качество камеры.	Смартфон
2	Григорий	Принтер очень быстро принимает запрос на печать.	Принтер
3	Оксана	Ноутбук имеет быструю и отзывчивую клавиатуру.	Ноутбук
4	Антон	Планшет имеет яркий и четкий дисплей.	Планшет
5	Даниил	У этих наушников просто потрясающее звучание	Наушники

Рисунок 1 – Скриншот базы данных PostgreSQL

База данных разделена на несколько столбцов: Имя, Отзыв, Тональность, Тип техники, Дата написания отзыва. Перечисленные параметры необходимы для корректной работы системы анализа. При записи нового отзыва с помощью интеллектуальной системы пользователь указывает необходимые данные, а при поиске отзыва ИПС представляет искомый отзыв и параметры, записанные в базу данных.

3 СОЗДАНИЕ И НАСТРОЙКА НЕЙРОННЫХ СЕТЕЙ

3.1 Многослойный персептрон

Для реализации задачи анализа тональности текста первым типом нейронных сетей был выбран многослойный персептрон со скрытыми слоями и функцией активации сигмоида.

Многослойный персептрон (MLP) - это класс нейронных сетей прямого распространения, состоящий из нескольких слоев нейронов: входного слоя, одного или нескольких скрытых слоев и выходного слоя. Он является одним из наиболее распространенных типов нейронных сетей и используется в широком спектре приложений, включая распознавание образов, классификацию, регрессию и многое другое.

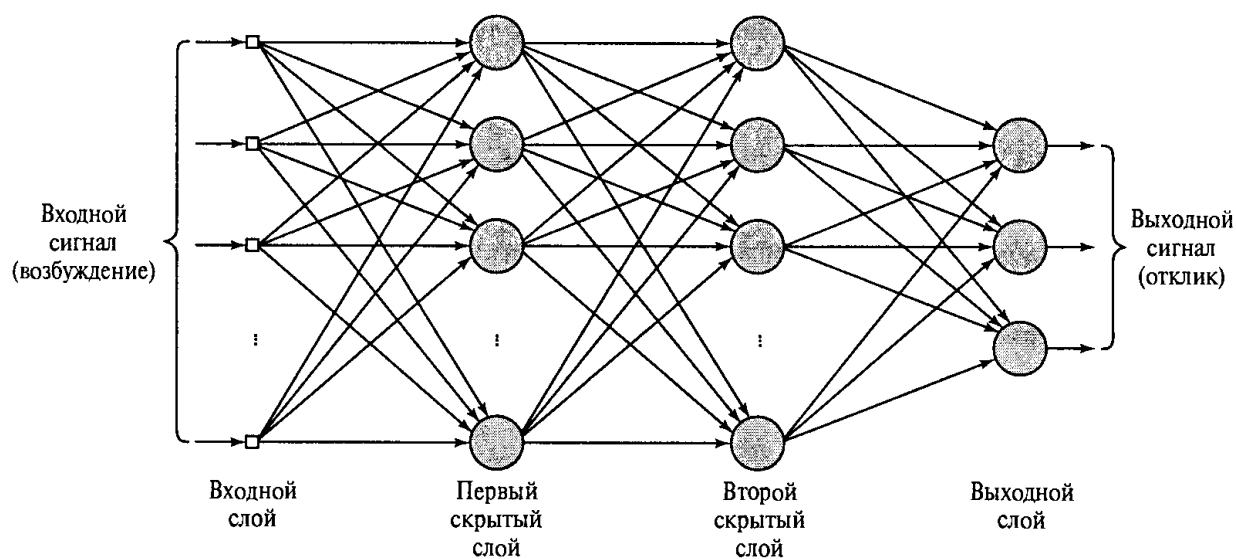


Рисунок 2 – Многослойный персептрон

Основные характеристики многослойного персептрона:

Скрытые слои: Многослойный персептрон имеет один или несколько скрытых слоев между входным и выходным слоями. Каждый нейрон в скрытом слое получает входные сигналы от всех нейронов предыдущего слоя и вычисляет свой выход, который затем передается на вход следующего слоя.

Функция активации: Для нелинейного преобразования входных данных каждый нейрон в сети использует функцию активации. Распространенными функциями активации для скрытых слоев являются сигмоида, гиперболический тангенс (\tanh), ReLU (Rectified Linear Unit) и другие.

Обратное распространение ошибки (Backpropagation): Для обучения многослойного персептрона обычно используется алгоритм обратного распространения ошибки. Этот алгоритм вычисляет градиент функции потерь по отношению к весам сети и использует его для обновления весов в направлении уменьшения ошибки.

Функция потерь: Для оценки того, насколько хорошо модель выполняет свою задачу, используется функция потерь. В задачах классификации часто используется кросс-энтропия, а в задачах регрессии - среднеквадратичная ошибка.

Регуляризация: Для предотвращения переобучения и улучшения обобщающей способности модели может применяться регуляризация, такая как L1 или L2 регуляризация.

Архитектура: Архитектура многослойного персептрона, включая количество скрытых слоев, количество нейронов в каждом слое и типы функций активации, может быть настроена в зависимости от конкретной задачи и данных.

Общая структура MLP делает его мощным инструментом для аппроксимации сложных нелинейных функций и решения различных задач машинного обучения.

3.2 Разработка нейронной сети на основе многослойного персептрона

Процесс разработки нейронной сети на языке программирования Python всегда начинается с подключения необходимых библиотек.

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet')
```

Рисунок 3 – Подключение библиотек

Следующим шагом является процесс очистки датасэта (положительных и отрицательных отзывов) от стоп-слов, а также их объединение.

```
# Лемматизация и удаление стоп-слов
lemmatizer = WordNetLemmatizer()
def preprocess_text(text):
    tokens = text.lower().split()
    tokens = [lemmatizer.lemmatize(token) for token in tokens if not token in ENGLISH_STOP_WORDS]
    return " ".join(tokens)

with open("pos.txt", "r", encoding='utf-8') as f:
    positive_texts = f.readlines()
    positive_texts = [preprocess_text(text) for text in positive_texts]

with open("neg.txt", "r", encoding='utf-8') as f:
    negative_texts = f.readlines()
    negative_texts = [preprocess_text(text) for text in negative_texts]

# Объединяем положительные и отрицательные отзывы для анализа тональности
all_reviews = positive_texts + negative_texts
```

Рисунок 4 – Очистка от стоп-слов

После завершения подготовки к работе нейросети начинается основной процесс обучения

```
# Основной цикл обучения
learning_rate = 0.01
decay_rate = 0.01
batch_size = 25
num_epochs = 100
optimizer = GradientDescentOptimizer()
optimizer.init(learning_rate, decay_rate)

for epoch in range(num_epochs):
    for i in range(0, len(training_inputs), batch_size):
        input_batch = training_inputs[i:i+batch_size]
        output_batch = training_outputs[i:i+batch_size]

        outputs = sigmoid(np.dot(input_batch, synaptic_weights))
        error = output_batch - outputs
        adjustments = error * sigmoid_derivative(outputs)

        synaptic_weights += optimizer.update_weights(input_batch, adjustments, epoch)
```

Рисунок 5 – Процесс обучения

Подытоживая, разработанная программа реализующая работу нейросети на основе многослойного персептрона работает следующим образом.

Предобработка текста: Исходные тексты (положительные и отрицательные отзывы) подвергаются предобработке, включая лемматизацию и удаление стоп-слов (слова, которые не несут смысловой нагрузки, например, "и", "в", "на" и т.д.).

Векторизация текста: Используется CountVectorizer из библиотеки scikit-learn, чтобы преобразовать текст в числовые признаки, считая количество вхождений каждого слова в текст.

Настройка входных и выходных данных: Подготовленные данные используются в качестве входных (матрица X) и выходных данных (матрица Y) для обучения сети. Входные данные представляют собой матрицу векторов признаков, а выходные данные представляют собой столбец, где "1" обозначает положительные отзывы, а "0" - отрицательные.

Инициализация весов: Веса синапсов (параметры модели) инициализируются случайным образом.

Оптимизация градиентным спуском: Обучение модели выполняется с использованием градиентного спуска. В коде реализовано обновление весов в цикле обучения с помощью градиентного спуска с уменьшением скорости обучения (learning rate) с течением времени.

Функция активации и обратная функция активации: Используется сигмоидальная функция активации и ее производная.

Проверка модели на новых данных: После обучения модель анализирует тональность нового предложения. Полный код программы представлен в приложении А.

3.3 Анализ результата работы программы

В ходе проделанной работы можно смело сказать, что ПО работает корректно. Модель определяет точность распознавания, которая варьируется от 70 до 84%, а также оценивает введённый текст и присваивает ему тональность. На рисунке 6 представлен график зависимости точности распознавания от количества слов.

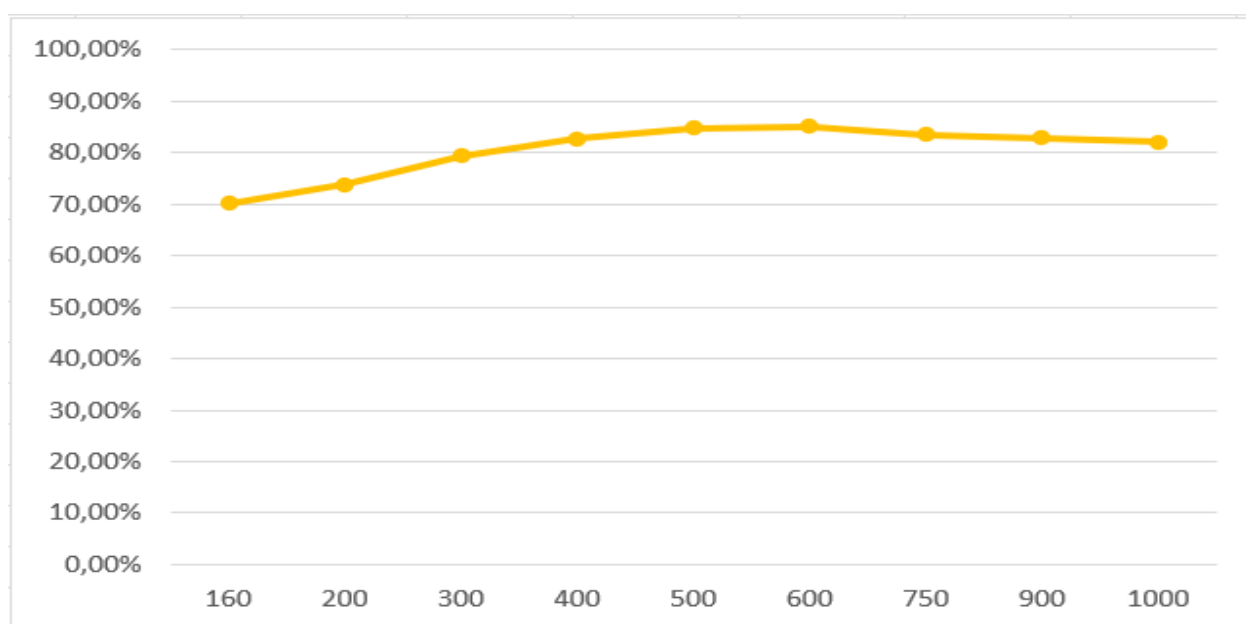


Рисунок 6 - График зависимости точности распознавания от количества слов

3.4 Алгоритм распознавания. Рекуррентная нейронная сеть.

В качестве алгоритма распознавания была выбрана рекуррентная нейронная сеть. Рекуррентной нейронной сетью называют вид нейронных сетей, где связи между элементами образуют направленную последовательность. Модель сети представлена на рисунке 7.

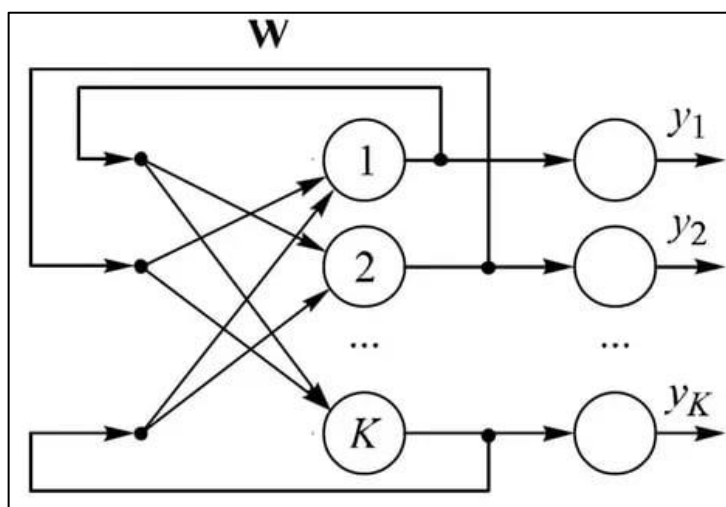


Рисунок 7 - Модель рекуррентной нейронной сети

Одно из важных свойств нейронных сетей – способность обучаться на основе входных данных для повышения своей производительности. Обучение – это процесс, в котором свободные параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена.

3.5 Разработка нейронной сети (RNN)

В начале разработки мы подготовили обучающую выборку (датасет), состоящую из двух файлов, содержащих позитивные и негативные отзывы. Подключение данных файлов показано на рисунке 8. Первым этапом

разработки является подача на вход тензора со строго определёнными размерами. Так как в датасете содержатся отзывы разных размеров, короткие отзывы будем дополнять нулями, а длинные обрезать. Размер каждой фразы определяется заданным значением в переменной `max_text_len`, то есть задаётся определённое количество слов в каждой фразе. Следовательно, на входе мы получим двумерный тензор, состоящий из количества фраз (`bath_size`) и длины каждого вектора (`max_text_len`). Таким образом мы конвертируем тексты в численные значения.

```
with open('pos.txt', 'r', encoding='utf-8') as f:
    texts_true = f.readlines()
    texts_true[0] = texts_true[0].replace('\uffff', '')

with open('neg.txt', 'r', encoding='utf-8') as f:
    texts_false = f.readlines()
    texts_false[0] = texts_false[0].replace('\uffff', '')
```

Рисунок

8 - Подключение файлов с положительными и отрицательными отзывами

Для разработки собственной нейронной сети использовался язык программирования – Python, библиотеки `numpy` (поддержка многомерных массивов; поддержка высокоуровневых математических функций), `os` (функций для работы с операционной системой), `Tokenizer` (лексический сканер для исходного кода на Python) и слои, необходимые для создания нейронных сетей, отвечают за архитектуру модели глубокого обучения - `Dense`, `LSTM`, `Embedding`, `Sequential`, `Adam`, рисунок 9.


```

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import numpy as np
import re
from collections import Counter
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')

from tensorflow.keras.layers import Dense, LSTM, Input, Dropout, Embedding
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer, text_to_word_sequence
from tensorflow.keras.preprocessing.sequence import pad_sequences

```

Рисунок 9 - Подключение библиотек

Следующим этапом является создание переменной `texts`, в которой перемешиваются позитивные и негативные отзывы. Далее мы разбиваем наши рецензии на отдельные слова используя метод `Tokenizer`, рисунок 10.

```

maxWordsCount = 1000

tokenizer = Tokenizer(num_words=maxWordsCount, filters='!-~#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n\r«»', lower=True, split=' ', char_level=False)
tokenizer.fit_on_texts(texts)

```

Рисунок 10 - Использование метода `Tokenizer`

`Tokenizer` выполняет токенизацию, рисунок 11, то есть разделение текстов на предложения, а предложения на слова, исключая символы, которые не влияют на эмоциональную окраску. Также токенизация в нашем программном обеспечении происходит с разделением текстов пробелами и сведением символов к нижнему регистру.

```

[[ 0  0  0 ... 31  1 47]
 [ 0  0  0 ...  1 274 439]
 [ 0  0  0 ...  1  15 130]
 ...
 [ 51 560  9 ... 760  88  18]
 [  0 203  7 ... 126  29 354]
 [ 35 308 50 ... 310  37  2]]

```

Рисунок 11 - Процесс токенизации

Затем происходит подсчёт количества появлений каждого слова в тексте. Результат подсчёта представлен на рисунке 8.

`('подключения', 17), ('скорость', 17), ('ноутбук', 17), ('делает', 17), ('Качество', 17), ('легко', 17)`

Рисунок 12 - Процесс подсчета повторений слов

Преобразование текстов в последовательность чисел в соответствии с полученным словарём является следующим шагом. Данное преобразование выполняется при помощи функции `texts_to_sequences`, рисунок 13.

```
max_text_len = 10
data = tokenizer.texts_to_sequences(texts)
data_pad = pad_sequences(data, maxlen=max_text_len)
```

Рисунок 13 - Использование функции `texts_to_sequences`

Преобразование каждой фразы происходит независимо друг от друга и его результат присваивается переменной `data`. После данного преобразования выполняется функция заполнения коротких рецензий нулями и обрезания длинных (`pad_sequences`) и приравнивается переменной `pad_data`. Далее создаётся обучающая выборка требуемых значений на выходе нейронной сети. Нейронная сеть представляет собой два нейрона, один из которых отвечает за положительный текст, а второй за отрицательный. На выходе для нейрона, отвечающего за положительный текст, выдаётся вектор равный `[1,0]`, а для нейрона, отвечающего за отрицательный текст, выдаётся вектор равный `[0,1]`, рисунок 14.

```

X = data_pad
Y = np.array([[1, 0]]*count_true + [[0, 1]]*count_false)

indices = np.random.choice(X.shape[0], size=X.shape[0], replace=False)
X = X[indices]
Y = Y[indices]

```

Рисунок 14 - Создание обучающей выборки требуемых значений на выходе нейронной сети

Количество положительных текстов определяется переменной count_true, а для отрицательных используется переменная count_false, рисунок 15.

```

texts = texts_true + texts_false
count_true = len(texts_true)
count_false = len(texts_false)
total_lines = count_true + count_false

```

Рисунок 15 - Использование переменных count_true, count_false

Чтобы получить обучающую выборку требуемых значений на выходе нейронной сети происходит сложение произведений значений положительного вектора на count_true и значений отрицательного вектора на count_false. Далее для лучшего обучения перемешиваем полученные значения. После чего переходим к созданию рекуррентной нейронной сети.

Архитектурой нашей нейронной сети будет являться Embedding. Слой Embedding представляет собой матрицу размерностью [maxWordscount, input_length], где maxWordscount – размер так называемого словаря уникальных слов, input_length – размер вектора word embeddings.

Словарь в машинном обучении – перечень уникальных слов, встречающихся в тексте. Зачастую в словаре также ведется учет того, насколько часто встречается то или иное слово: при объемных массивах исходных данных для уменьшения затрат на вычисления мы можем

уменьшить словарь, убрав из него наиболее часто встречающиеся слова, не имеющие высокой ценности.

Рекуррентная нейронная сеть создаётся из двух LSTM слоёв идущих последовательно друг за другом.

```
model = Sequential()
model.add(Embedding(maxWordsCount, 128, input_length = max_text_len))
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(64))
model.add(Dense(2, activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer=Adam(0.1))

history = model.fit(X, Y, batch_size=25, epochs=100)

reverse_word_map = dict(map(reversed, tokenizer.word_index.items()))

def sequence_to_text(list_of_indices):
    words = [reverse_word_map.get(letter) for letter in list_of_indices]
    return(words)
```

Рисунок 16 - Создание рекуррентной нейронной сети

На выходе получаем полно-связный слой из двух нейронов, после чего происходит компилирование нейронной сети с подсчётом потерь, точностью метрик и оптимизацией сходимости 0,001. Далее начинается процесс самообучения.

```
Epoch 45/100
41/41 [=====] - 1s 27ms/step - loss: 0.7769 - accuracy: 0.6229
Epoch 46/100
41/41 [=====] - 1s 25ms/step - loss: 0.7672 - accuracy: 0.6024
Epoch 47/100
41/41 [=====] - 1s 25ms/step - loss: 0.7228 - accuracy: 0.6396
Epoch 48/100
41/41 [=====] - 1s 27ms/step - loss: 0.7525 - accuracy: 0.6082
Epoch 49/100
41/41 [=====] - 1s 29ms/step - loss: 0.7310 - accuracy: 0.6239
Epoch 50/100
41/41 [=====] - 1s 34ms/step - loss: 0.8184 - accuracy: 0.6014
```

Рисунок 17 - Процесс самообучения

Конечным этапом разработки является определение эмоциональной окраски тестируемой выборки на основе самообучения, проходящего при помощи обучающей выборки, рисунок 18. Для тестирования разработанного ПО использовалось создание обычной строки, куда с клавиатуры вводятся рецензии. Результаты тестирования представлены на рисунках 18.1 и 18.2. Полный код программы представлен в приложении Б.

```
data = tokenizer.texts_to_sequences([otziv])
data_pad = pad_sequences(data, maxlen=max_text_len)

res = model.predict(data_pad)
print(res, sep='\n')
if (np.argmax(res) == 1):
    print("негативный(точность указана справа)")
else:
    print("положительный(точность указана слева)")
```

Рисунок 18 - Блок для тестирующей выборки

```
[[0.4515319 0.5484681]]
негативный(точность указана справа)
```

Рисунок 18.1 - Вывод результата (негативный отзыв)

```
[[0.95451814 0.04548191]]
положительный(точность указана слева)
```

Рисунок 18.2 - Вывод результата (положительный отзыв)

3.6 Анализ результатов работы программы

На основе нашей обучающей выборки результаты проверяющей лежат в пределах от 55 до 96%, рисунки 19.1 и 19.2. Результат зависит от количества значимых и неинформативных слов в проверяющей выборке (от 160 до 1000). На рисунке 19 представлен график зависимости точности распознавания от количества слов.

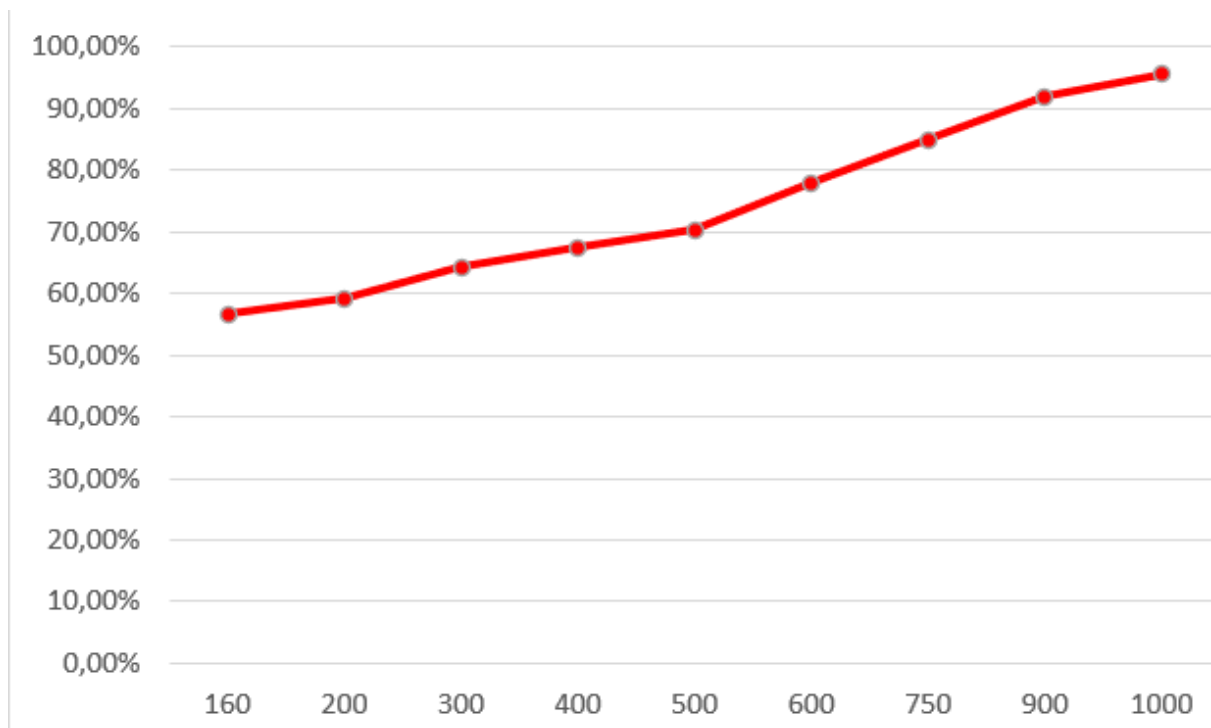


Рисунок 19 - График зависимости точности распознавания от количества слов

```

41/41 [=====] - 2s 39ms/step - loss: 0.5471 - accuracy: 0.7346
Epoch 46/50
41/41 [=====] - 1s 37ms/step - loss: 0.5218 - accuracy: 0.7375
Epoch 47/50
41/41 [=====] - 2s 43ms/step - loss: 0.5697 - accuracy: 0.7228
Epoch 48/50
41/41 [=====] - 2s 45ms/step - loss: 0.5796 - accuracy: 0.7258
Epoch 49/50
41/41 [=====] - 1s 35ms/step - loss: 0.5736 - accuracy: 0.7062
Epoch 50/50
41/41 [=====] - 1s 36ms/step - loss: 0.5299 - accuracy: 0.7532
1/1 [=====] - 6s 6s/step
[[0.73835045 0.26164955]]
положительный(точность указана слева)

```

Рисунок 19.1 - Точность и оценка рецензии (положительный отзыв)

```

41/41 [=====] - 1s 33ms/step - loss: 0.8114 - accuracy: 0.6082
Epoch 96/100
41/41 [=====] - 1s 34ms/step - loss: 0.8004 - accuracy: 0.5818
Epoch 97/100
41/41 [=====] - 1s 36ms/step - loss: 0.7142 - accuracy: 0.5896
Epoch 98/100
41/41 [=====] - 1s 30ms/step - loss: 0.7397 - accuracy: 0.5867
Epoch 99/100
41/41 [=====] - 1s 28ms/step - loss: 0.7568 - accuracy: 0.5935
Epoch 100/100
41/41 [=====] - 1s 31ms/step - loss: 0.7213 - accuracy: 0.6121
1/1 [=====] - 2s 2s/step
[[0.39673606 0.603264 ]]
негативный(точность указана справа)

```

Рисунок 19.2 - Точность и оценка рецензии (негативный отзыв)

3.7 Искусственный нейрон

В качестве третьего алгоритма распознавания была выбрана модель Искусственного нейрона. **Искусственный нейрон** – простейший вид нейронных сетей. В основе лежит математическая модель восприятия информации мозгом, состоящая из сенсоров, ассоциативных и реагирующих элементов.

Искусственный нейрон

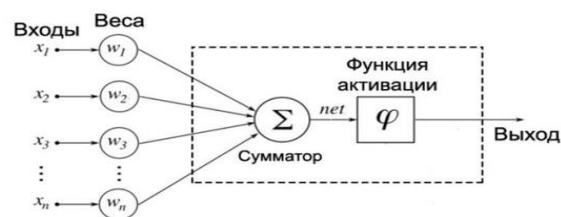


Рисунок 20 - Модель Искусственный нейрон

У каждого нейрона, в том числе и у искусственного, должны быть какие-то входы, через которые он принимает сигнал. Сигналы, проходящие по связям, умножаются на соответствующие веса. На картинке веса изображены кружками.

Поступившие на входы сигналы умножаются на свои веса. Сигнал первого входа X_1 умножается на соответствующий этому входу вес W_1 . В итоге получаем X_1W_1 . И так до n -го входа. В итоге на последнем входе получаем X_nW_n .

Теперь все произведения передаются в сумматор. Уже исходя из его названия можно понять, что он делает. Он просто суммирует все входные сигналы, умноженные на соответствующие веса: $X_1W_1 + X_2W_2 + X_3W_3 + \dots + X_nW_n = \text{net}$

Результатом работы сумматора является число, называемое взвешенной суммой (net).

3.8 Разработка собственной нейронной сети на основе искусственного нейрона

В начале работы был освоен алгоритм распознавания Искусственный нейрон. Используемый язык программирования – Python. Для разработки использовались: библиотеки **Scikit-learn** (Это библиотека, предназначенная для машинного обучения, написанная на языке программирования Python и распространяемая в виде свободного программного обеспечения.), **nlTK** ((Natural Language Toolkit) – ведущая платформа для создания NLP-программ на Python. У нее есть легкие в использовании интерфейсы для многих языковых корпусов, а также библиотеки для обработки текстов для классификации, токенизации, стемминга, разметки, фильтрации и семантических рассуждений.). На рисунках 21, 22 продемонстрировано подключение библиотек. Полный код программы представлен в приложении А.


```
import numpy as np
import re
from nltk.corpus import stopwords
```

Рисунок 21 - Подключение библиотек nltk и numpy

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
```

Рисунок 22 - Подключение библиотеки sklearn

Следующим этапом следует подключить два файла с положительными и отрицательными отзывами. На рисунке 23 показано внедрение датасета в ПО.

```
# Читаем положительные и отрицательные тексты из файлов
with open("pos.txt", "r", encoding='utf-8') as f:
    positive_texts = f.readlines()
with open("neg.txt", "r", encoding='utf-8') as f:
    negative_texts = f.readlines()
```

Рисунок 23 - Подключение файлов с положительными и отрицательными рецензиями

Далее рецензии собираются в обучающую выборку, также происходит процесс формирования матрицы признаков. Весь процесс изображён на рисунке 24.

```
# Создаем тренировочную выборку
X_train = positive_texts + negative_texts
y_train = ['positive'] * len(positive_texts) + ['negative'] * len(negative_texts)

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, stratify=y_train)

# Формируем матрицу признаков из тренировочной выборки
X_train_vectorized = vectorizer.fit_transform(X_train)
```

Рисунок 24 - Процесс создания выборки и матрицы признаков

После того как произошла чистка происходит процесс деления рецензий на обучающую и проверяющую выборку, затем подключение библиотек, чтобы реализовать вектор. Это показано на рисунке 25.

```
max_feature = 1000  
vectorizer = CountVectorizer(token_pattern=r'\b[a-яА-ЯёЁ]+\b', max_features=max_feature)
```

Рисунок 25 - Создание вектора

Обучение модели Искусственный нейрон изображено на рисунке 26.

```
# Создаем и обучаем нейронную сеть  
clf = MLPClassifier(hidden_layer_sizes=(50,), max_iter=100, verbose = True, learning_rate_init=0.0001, batch_size=25, alpha=0.1)  
clf.fit(X_train_vectorized, y_train)
```

Рисунок 26 - Обучение модели

В итоге обученная модель вывела результаты, которые продемонстрированы на рисунках 27, 28.

```
Текст: 1. "Этот компьютер имеет невероятную производительность - он запускает игры без каких-либо задержек".  
Тональность:94.89% положительный  
Общая точность = 4.7445%
```

Рисунок 27 - Пример позитивного отзыва

```
Текст: 1. "Этот компьютер просто ужасен".  
Тональность:67.65% негативный  
Общая точность = 3.3825000000000003%
```

Рисунок 28 - Пример негативного отзыва

3.9 Анализ результата работы программы

В ходе проделанной работы можно смело сказать, что ПО работает корректно. Модель определяет точность распознавания, которая варьируется от 72 до 83%, а также оценивает введенный текст и присваивает ему тональность. На рисунке 29 представлен график зависимости точности распознавания от количества слов.

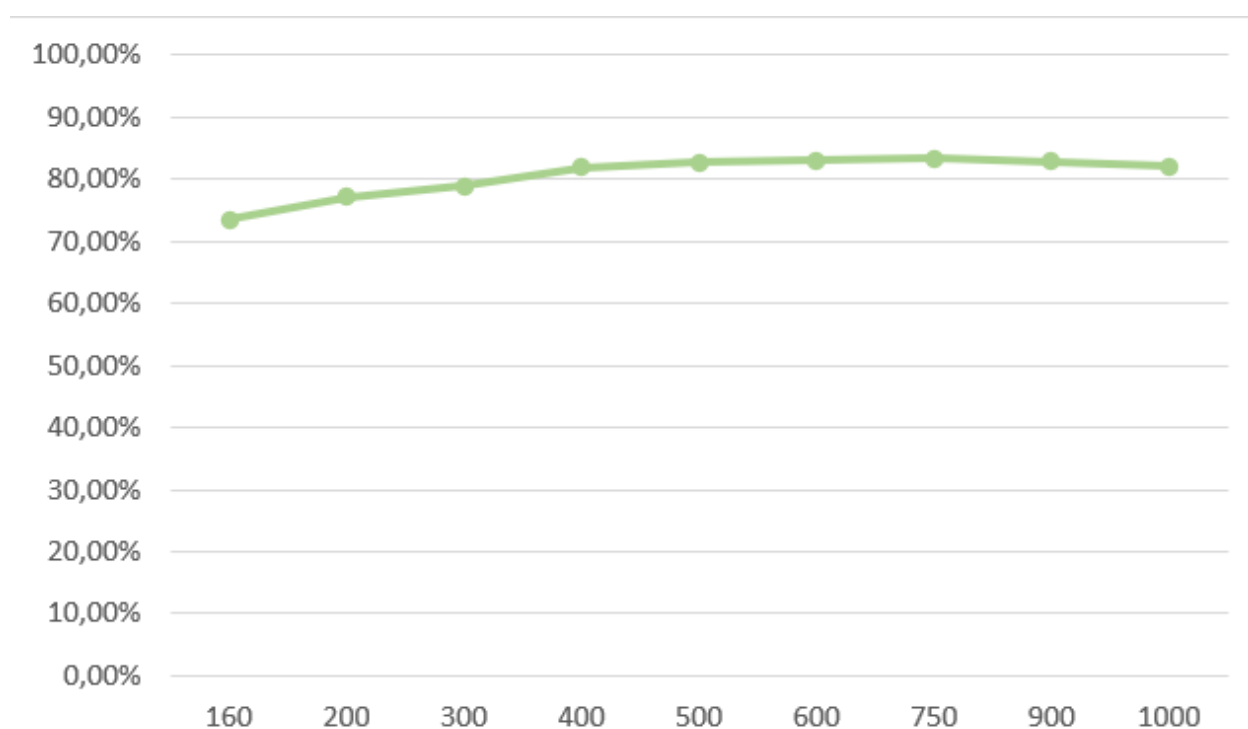


Рисунок 29 - График зависимости точности распознавания от количества слов

4 СРАВНЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ ТРЕХ МОДЕЛЕЙ НЕЙРОННЫХ СЕТЕЙ

Анализируя полученные результаты при разработке и обучении моделей машинного обучения, можно сделать вывод о том, что нейросеть на основе искусственного нейрона точнее определяет тональность текста при меньшем количестве слов. При выборке в 160 слов процент распознавания 70-73% это является отличным результатом, модель на основе рекуррентной нейросети уступает на 16%, что является существенной разницей. Однако при увеличении количества слов модель на основе искусственного нейрона уступает лишь на 2%, в пике многослойному персептрону, что может являться погрешностью. Подводя итог, был сделан вывод, что для реализации поставленной задачи нейросеть, реализованная на искусственном нейроне является наиболее эффективной и разработка более сложного алгоритма нецелесообразна. Ниже приведен точный график зависимости точности распознавания от количества слов.



Рисунок 30 - График точности распознавания с собственным датасетом

5 РАЗРАБОТКА ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗАТОРА НА ОСНОВЕ ЛУЧШЕЙ МОДЕЛИ

Для удобного взаимодействия была создана система интеллектуального анализа отзывов. Система была разработана на основе результатов сравнений лучшей модели, а именно на основе искусственного нейрона. Разработка осуществлялась на языке программирования Python с использованием ранее созданной базы данных. Окно с интерфейсом показано на рисунке 31.

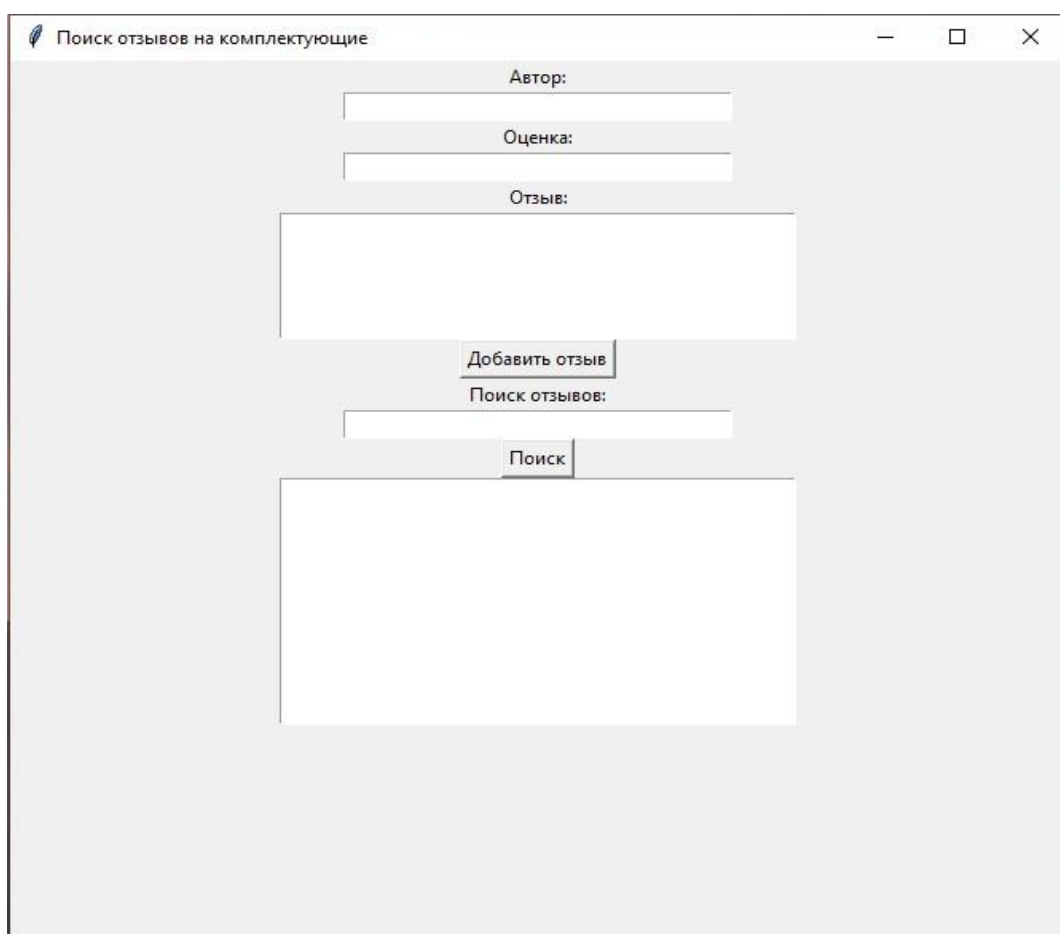


Рисунок 31 – Интерфейс анализатора

Интерфейс интеллектуального анализатора состоит из поля “Автор”, в которое пользователь вносит свое имя, поля “Оценка” для определения является ли отзыв негативным или положительным, поля “Отзыв” для написания отзыва о продукте и кнопки “Добавить отзыв” для записи отзыва в базу данных.

Помимо возможности написания отзыва о каком-либо продукте интеллектуальный анализатор предоставляет возможность поиска среди всех отзывов в базе данных с помощью поиска по словам и кнопки “Поиск”. В поле ниже будет предоставлен искомый пользователем отзыв.

Поиск отзывов на комплектующие

Автор:

Оценка:

Отзыв:

Добавить отзыв

Поиск отзывов:

ZXC

Поиск

Автор: Слава
Оценка: Отрицательная
Дата добавления: 2023-11-07 05:19:05
Отзыв: Совсем не понравился монитор ZXC-322LS!
Очень сильные засветы и плохой угол обзора

Рисунок 32 – Поиск отзыва

Данная версия интеллектуального анализатора является гибко настраиваемой и легко модифицируется при необходимости. При необходимости можно как добавлять поля, так и удалять, менять отображаемую и запрашиваемую информацию.

ЗАКЛЮЧЕНИЕ

Анализ тональности является ценным инструментом для понимания общественного мнения и извлечения полезных сведений из текстовых данных.

Интеллектуальные методы анализа тональности, такие как машинное обучение, превосходят традиционные методы по точности и эффективности.

Будущие исследования могут быть направлены на расширение словаря для анализа тональности, включение обработки сарказма и иронии, а также разработку новых алгоритмов для повышения точности анализа.

Анализ тональности может быть интегрирован в различные приложения, такие как системы обработки клиентских запросов, инструменты исследования рынка и системы рекомендаций.

Область анализа тональности постоянно развивается, поскольку появляются новые методы и технологии, которые способствуют более глубокому пониманию человеческого языка.

Усовершенствованные методы анализа тональности могут способствовать принятию обоснованных решений на основе данных и улучшению взаимодействия с клиентами.

Перед моделями была поставлена задача анализа тональности текста на русском языке. Решение описанной проблемы удовлетворительно реализовано разработанным анализатором тональности отзывов. Для тестирования использовались отзывы о компьютерной технике с различных порталов таких, как: Яндекс.Маркет, DNS, МВидео. Результаты распознавания удовлетворили поставленную задачу. Модели успешно анализировали тональность отзывов.

В ходе работы была подробно рассмотрена задача анализа тональности текстов. Была составлена база данных, состоящая из полученных отзывов. Были подготовлены обучающая и проверяющая выборки и определена информативность слов.

Созданы и настроены собственные ПО, произведено сравнение собственных ПО.

Также на основе лучшего ПО была разработана интеллектуальный анализатор тональности отзывов. Разработанный анализатор тональности отзывы может быть использован для мониторинга репутации бренда, оценки удовлетворенности клиентов и получения информации о тенденциях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рекуррентные слои – русскоязычная документация Keras [Электронный ресурс] URL: <https://ru-keras.com/recurrent-layers/> Дата обращения [15.04.2024]
2. Анализ тональности [Электронный ресурс] URL: https://nlpub.ru/Анализ_тональности Дата обращения [15.04.2024]
3. Алгоритмическое и программное обеспечение для анализа тональности текстовых сообщений с использованием машинного обучения [Электронный ресурс] URL: <https://cyberleninka.ru/article/n/algoritmicheskoe-i-programmnoe-obespechenie-dlya-analiza-tonalnosti-tekstovyh-soobscheniy-s-ispolzovaniem-mashinnogo-obucheniya> Дата обращения [15.04.2024]
4. Анализ тональности текста [Электронный ресурс] URL: https://github.com/alcatraz-rm/Text_tone_analyzer Дата обращения [15.04.2024]

ПРИЛОЖЕНИЕ А

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet')

# Функция активации - сигмоида
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Обратная функция активации для вычисления градиента
def sigmoid_derivative(x):
    return x * (1 - x)

# Лемматизация и удаление стоп-слов
lemmatizer = WordNetLemmatizer()
def preprocess_text(text):
    tokens = text.lower().split()
    tokens = [lemmatizer.lemmatize(token) for token in tokens if not token in
ENGLISH_STOP_WORDS]
    return " ".join(tokens)

with open("pos.txt", "r", encoding='utf-8') as f:
    positive_texts = f.readlines()
    positive_texts = [preprocess_text(text) for text in positive_texts]

with open("neg.txt", "r", encoding='utf-8') as f:
    negative_texts = f.readlines()
    negative_texts = [preprocess_text(text) for text in negative_texts]

# Объединяем положительные и отрицательные отзывы для анализа тональности
all_reviews = positive_texts + negative_texts

# Создаем экземпляр класса CountVectorizer с заданным максимальным количеством
слов
max_feature = 100
vectorizer = CountVectorizer(token_pattern=r'\b[a-яA-ЯёЁ]+\b',
max_features=max_feature)
X = vectorizer.fit_transform(all_reviews).toarray()

# Установка входных и выходных данных для обучения сети
training_inputs = X
training_outputs = np.array([[1] * len(positive_texts) + [0] *
len(negative_texts)]).T

# Настройка случайных весов
```

```

np.random.seed(1)
synaptic_weights = 2 * np.random.random((len(training_inputs[0]), 1)) - 1

# Оптимизатор градиентного спуска с уменьшением скорости обучения
class GradientDescentOptimizer:
    def init(self, learning_rate, decay_rate):
        self.learning_rate = learning_rate
        self.decay_rate = decay_rate

    def update_weights(self, inputs, adjustments, epoch):
        lr = self.learning_rate / (1 + self.decay_rate * epoch)
        return lr * np.dot(inputs.T, adjustments)

# Основной цикл обучения
learning_rate = 0.01
decay_rate = 0.01
batch_size = 25
num_epochs = 100
optimizer = GradientDescentOptimizer()
optimizer.init(learning_rate, decay_rate)

for epoch in range(num_epochs):
    for i in range(0, len(training_inputs), batch_size):
        input_batch = training_inputs[i:i+batch_size]
        output_batch = training_outputs[i:i+batch_size]

        outputs = sigmoid(np.dot(input_batch, synaptic_weights))
        error = output_batch - outputs
        adjustments = error * sigmoid_derivative(outputs)

        synaptic_weights += optimizer.update_weights(input_batch, adjustments,
epoch)

# Проверка модели на новых данных
new_sentence = ["Этот компьютер имеет невероятную производительность - он
запускает игры без каких-либо задержек"]
new_sentence = [preprocess_text(sentence) for sentence in new_sentence]
X_new = vectorizer.transform(new_sentence).toarray()
output = sigmoid(np.dot(X_new, synaptic_weights))
print("Результат анализа тональности текста:", output)

# Вывод результата анализа тональности текста
result = "Положительный" if output >= 0.5 else "Отрицательный"
print(f"Результат анализа тональности текста: {result}")

```

ПРИЛОЖЕНИЕ Б

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import numpy as np
import re
from collections import Counter
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')

from tensorflow.keras.layers import Dense, LSTM, Input, Dropout, Embedding
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer, text_to_word_sequence
from tensorflow.keras.preprocessing.sequence import pad_sequences

with open('pos.txt', 'r', encoding='utf-8') as f:
    texts_true = f.readlines()
    texts_true[0] = texts_true[0].replace('\uffff', '')

with open('neg.txt', 'r', encoding='utf-8') as f:
    texts_false = f.readlines()
    texts_false[0] = texts_false[0].replace('\uffff', '')

with open('test_texts.txt', 'r', encoding='utf-8') as file:
    otziv = file.readlines()
    otziv[0] = otziv[0].replace('\uffff', '')

texts = texts_true + texts_false
count_true = len(texts_true)
count_false = len(texts_false)
total_lines = count_true + count_false

maxWordsCount = 1000

tokenizer = Tokenizer(num_words=maxWordsCount, filters='!-#$$%&()*+,-
./:;<=>?@[\\]^_`{|}~\t\n\r«»', lower=True, split=' ', char_level=False)
tokenizer.fit_on_texts(texts)

all_words = ' '.join(texts).split()
word_freq = Counter(all_words)
sorted_word_freq = sorted(word_freq.items(), key=lambda x: x[1], reverse=True)
print("Самые популярные слова:", sorted_word_freq[:100])

stop_words = set(stopwords.words('russian'))
print("Стоп-слова:", stop_words)
```

```

word_scores = {}
for word, freq in word_freq.items():
    if word not in stop_words:
        word_scores[word] = np.log(total_lines / (1 + freq))

sorted_word_scores = sorted(word_scores.items(), key=lambda x: x[1],
reverse=True)
print("Самые информативные слова:", sorted_word_scores[:10])

max_text_len = 10
data = tokenizer.texts_to_sequences(texts)
data_pad = pad_sequences(data, maxlen=max_text_len)
print(data_pad)

X = data_pad
Y = np.array([[1, 0]]*count_true + [[0, 1]]*count_false)

indeces = np.random.choice(X.shape[0], size=X.shape[0], replace=False)
X = X[indeces]
Y = Y[indeces]

model = Sequential()
model.add(Embedding(maxWordsCount, 128, input_length = max_text_len))
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(64))
model.add(Dense(2, activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy', metrics=['accuracy'],
optimizer=Adam(0.1))

history = model.fit(X, Y, batch_size=25, epochs=50)

reverse_word_map = dict(map(reversed, tokenizer.word_index.items()))

def sequence_to_text(list_of_indices):
    words = [reverse_word_map.get(letter) for letter in list_of_indices]
    return(words)

data = tokenizer.texts_to_sequences([otziv])
data_pad = pad_sequences(data, maxlen=max_text_len)

res = model.predict(data_pad)
print(res, sep='\n')
if (np.argmax(res) == 1):
    print("негативный(точность указана справа)")
else:
    print("положительный(точность указана слева)")

```

ПРИЛОЖЕНИЕ В

```
#тут все ок
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import numpy as np
import re
from nltk.corpus import stopwords

# Создаем экземпляр CountVectorizer с указанием языка
max_feature = 1000
vectorizer = CountVectorizer(token_pattern=r'\b[a-яА-ЯёЁ]+\b',
max_features=max_feature)

# Читаем положительные и отрицательные тексты из файлов
with open("pos.txt", "r", encoding='utf-8') as f:
    positive_texts = f.readlines()
with open("neg.txt", "r", encoding='utf-8') as f:
    negative_texts = f.readlines()

stop_words_set = set(stopwords.words('russian'))
stop_words_set.add('это')
stop_words_set.add('оно')
stop_words_set.add('эта')
#print(stop_words_set)

# Функция для проверки однокоренных слов
def is_related(word1, word2):
    return word1[:-2] == word2[:-2]

# Функция для фильтрации информативных слов
def filter_top_words(top_words):
    filtered_words = []
    for weight, word in top_words:
        if (word not in stop_words_set and not re.match(r'^\d', word) and
            not any(is_related(word, filtered_word) for _, filtered_word in
filtered_words)):
            filtered_words.append((weight, word))
    return filtered_words

# Создаем тренировочную выборку
X_train = positive_texts + negative_texts
y_train = ['positive'] * len(positive_texts) + ['negative'] * len(negative_texts)

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,
stratify=y_train)

# Формируем матрицу признаков из тренировочной выборки
```

```

X_train_vectorized = vectorizer.fit_transform(X_train)

# Создаем и обучаем нейронную сеть
clf = MLPClassifier(hidden_layer_sizes=(50,), max_iter=100, verbose = True,
learning_rate_init=0.0001, batch_size=25, alpha=0.1)
clf.fit(X_train_vectorized, y_train)

# Читаем тестовые тексты из файла
with open("test_texts.txt", "r", encoding='utf-8') as f:
    test_texts = f.readlines()

# Формируем матрицу признаков из тестовой выборки
X_test_vectorized = vectorizer.transform(test_texts)

# Предсказываем тональность текстов
y_pred = clf.predict(X_test_vectorized)
y_pred_proba = clf.predict_proba(X_test_vectorized)
total_accuracy = np.mean([1 if true == pred else 0 for true, pred in zip(y_test,
y_pred)])

a = 0
b = 0
# Выводим результаты
for text, pred, prob in zip(test_texts, y_pred, y_pred_proba):
    if pred == 'positive':
        print(f"Текст: {text.strip()}")
        print(f"Тональность:{'%.2f' % (prob[1]*100)}% положительный")
        a = float('%.2f' % (prob[1]*100)) + a
    else:
        print(f"Текст: {text.strip()}")
        print(f"Тональность:{'%.2f' % (prob[0]*100)}% негативный")
        b = float('%.2f' % (prob[0]*100)) + b
print(f"Общая точность = {(a+b)/20}%")
print(f"Общее количество правильно распознанных отзывов: {int(total_accuracy *
len(y_test))} из {len(y_test)}")

# Находим наиболее информативные слова и их веса
feature_names = vectorizer.get_feature_names()

# Фильтруем информативные слова, исключая числовые значения, стоп-слова и
однокоренные слова
top_words = [(weight, word) for weight, word in zip(clf.coefs_[-1],
feature_names)]
top_words = filter_top_words(top_words)
top_words = sorted(top_words, reverse=True)

print("\nНаиболее информативные слова:")
for weight, word in top_words:
    print(f"{word}:  $\omega$ ={weight}")

```

```

# Формируем словарь с количеством вхождений каждого слова в тренировочной выборке
word_counts = {}
for i, word in enumerate(feature_names):
    if word not in stop_words_set and not re.match(r'^\d', word):
        filtered = False
        for _, filtered_word in top_words:
            if is_related(word, filtered_word):
                filtered = True
                break
        if not filtered:
            word_counts[word] = word_counts.get(word, 0) +
X_train_vectorized.getcol(i).sum()

# Находим наиболее частоповторяющиеся слова и их количество среди информативных
слов
top_counts = sorted(word_counts.items(), key=lambda x: x[1], reverse=True)[:30]
print("\nНаиболее часто повторяющиеся слова среди информативных:")
for word, count in top_counts:
    print(f"{word}: {count}")

```


ПРИЛОЖЕНИЕ Г

```
import tkinter as tk from
tkinter import
messagebox from
datetime import datetime

# Функция для добавления отзыва в базу
данных def add_review():
    author = author_entry.get()    rating = rating_entry.get()
    review_text = review_text_entry.get("1.0", "end-1c")    date =
    datetime.now().strftime("%Y-%m-%d %H:%M:%S")    conn
    = sqlite3.connect("reviews.db")    cursor = conn.cursor()
    cursor.execute("INSERT INTO reviews (author, rating,
    review_text, date_added) VALUES (?, ?, ?, ?)",
        (author, rating,
    review_text, date))    conn.commit()
    conn.close()

    messagebox.showinfo("Успех", "Отзыв добавлен
    успешно!")    author_entry.delete(0, "end")
    rating_entry.delete(0, "end")
    review_text_entry.delete("1.0", "end")

# Функция для выполнения
поиска def search_reviews():
    query = search_entry.get()    conn =
    sqlite3.connect("reviews.db")    cursor = conn.cursor()
    cursor.execute("SELECT author, rating, review_text, date_added
    FROM reviews WHERE review_text LIKE ?",
        (f"%{query}%",))
    results = cursor.fetchall()
    conn.close()
```

```

results_text.config(state="normal")
results_text.delete("1.0",
"end")    for result in results:
            author, rating, review_text, date_added = result
results_text.insert("end", f"Автор: {author}\nОценка:
{rating}\nДата добавления: {date_added}\nОтзыв:
{review_text}\n\n")    results_text.config(state="disabled")

# Создание главного
окна root = tk.Tk()

root.title("Поиск отзывов на
комплектующие") # Создание
текстовых полей и меток author_label =
tk.Label(root, text="Автор:")
author_label.pack() author_entry =
tk.Entry(root, width=40)
author_entry.pack()

rating_label = tk.Label(root, text="Оценка:")
rating_label.pack() rating_entry =
tk.Entry(root, width=40) rating_entry.pack()
review_text_label = tk.Label(root,
text="Отзыв:") review_text_label.pack()
review_text_entry = tk.Text(root, width=40,
height=5) review_text_entry.pack() add_button
= tk.Button(root, text="Добавить отзыв",
command=add_review) add_button.pack()
search_label = tk.Label(root, text="Поиск
отзывов:") search_label.pack()

```

```

search_entry = tk.Entry(root, width=40) search_entry.pack()
search_button = tk.Button(root, text="Поиск",
command=search_reviews) search_button.pack() results_text =
tk.Text(root, width=40, height=10) results_text.pack()
results_text.config(state="disabled") conn =
sqlite3.connect("reviews.db") cursor = conn.cursor()
cursor.execute("CREATE TABLE IF NOT EXISTS reviews
(id INTEGER
PRIMARY KEY,
author TEXT, rating
INTEGER,
review_text TEXT,
date_added DATETIME)")
conn.commit() conn.close()

```

```

# Запуск главного цикла приложения root.mainloop()

```