

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:
ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА АНАЛИЗА ТОНАЛЬНОСТИ
ТЕКСТА

Направление: 10.03.01 Информационная безопасность

Профиль: Безопасность компьютерных систем

Выполнили:
студенты группы УИБ-313
Бороденков А.И., Гущин Д.Е.,
Канатов Н.В., Макеев Д.В.
Метельков А.Н., Павлов А.Р.

Наставники:
Цыганова Н. А.
Малинский С.В.
(должность, ФИО)

МОСКВА
2023

АННОТАЦИЯ

Пояснительная записка 69 с., 24 рис., 4 приложения.

АНАЛИЗ ТОНАЛЬНОСТИ ТЕКСТА, 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 компьютерной техники.

В ходе работы было проделано: создание и настройка собственного ПО, сравнение, определение наиболее эффективного ПО среди разработанных, создание базы данных, разработка интеллектуальной поисковой системы

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1 ПРОБЛЕМА АНАЛИЗА ТОНАЛЬНОСТИ ТЕКСТОВ.....	6
2 ФОРМИРОВАНИЕ И ПОДГОТОВКА ДАННЫХ ДЛЯ МАШИННОГО ОБУЧЕНИЯ ПРОГРАММЫ АНАЛИЗА ТОНАЛЬНОСТИ ТЕКСТА.....	8
3 СОЗДАНИЕ МОДЕЛИ МАШИННОГО ОБУЧЕНИЯ НА ОСНОВЕ НЕЙРОННОЙ СЕТИ.....	10
3.1 Алгоритм распознавания. Рекуррентная нейронная сеть.....	10
3.2 Разработка рекуррентной нейронной сети (RNN).....	12
3.3 Анализ результатов работы программы.....	18
3.4 Разработка нейронной сети на основе искусственного нейрона.....	19
3.5 Разработка собственной нейронной сети. Искусственный нейрон.....	20
3.6 Процесс обучения нейронной сети на основе искусственного нейрона.....	22
3.7 Анализ результата работы программы.....	23
4 РАЗРАБОТКА И НАСТРОЙКА ИНТЕЛЕКТУАЛЬНОЙ ПОИСКОВОЙ СИСТЕМЫ.....	24
5 СРАВНЕНИЕ ТОЧНОСТИ ОПРЕДЕЛЕНИЯ ТОНАЛЬНОСТИ ТЕКСТА МОДЕЛЯМИ МАШИННОГО ОБУЧЕНИЯ НА ОСНОВЕ РЕКУРРЕНТНОЙ НЕЙРОСЕТИ И ИСКУССТВЕННОГО НЕЙРОНА.....	27
ЗАКЛЮЧЕНИЕ.....	28
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	29
ПРИЛОЖЕНИЕ А.....	30
ПРИЛОЖЕНИЕ Б.....	33
ПРИЛОЖЕНИЕ В.....	36
ПРИЛОЖЕНИЕ Г.....	39

ОБЩЕЕ ЗАДАНИЕ НА ПРОЕКТНУЮ ДЕЯТЕЛЬНОСТЬ

- Подготовка данных;
- Определение информативности слов;
- Создание собственных ПО;
- Настройка собственных ПО;
- Сравнение собственных ПО
- Создание базы данных
- Разработка интеллектуальной поисковой системы

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.

1 ПРОБЛЕМА АНАЛИЗА ТОНАЛЬНОСТИ ТЕКСТОВ

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.

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

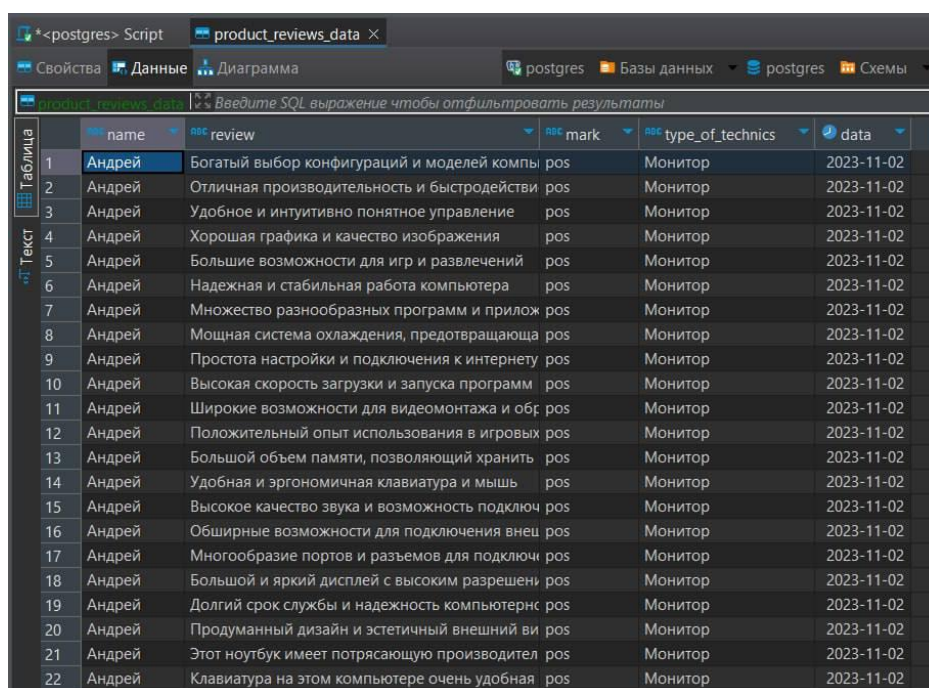
Несмотря на достаточно большое количество работ и систем, проведенных в сфере анализа тональности текста, точность определения сентимент анализа не находится на совершенном уровне. Ниже представлены проблемы, которые часто возникают при разработке и использовании автоматических алгоритмов определения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

- 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;
- 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;
- 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;
- Противительные союзы, могут изменить всю тональность предложения. В русском языке существует небольшое количество союзов такого типа: а, да (в значении, но), зато, но, однако;
- 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 поисковой системе (Далее ИПС).



	name	review	mark	type_of_technics	data
1	Андрей	Богатый выбор конфигураций и моделей компы	pos	Монитор	2023-11-02
2	Андрей	Отличная производительность и быстродействи	pos	Монитор	2023-11-02
3	Андрей	Удобное и интуитивно понятное управление	pos	Монитор	2023-11-02
4	Андрей	Хорошая графика и качество изображения	pos	Монитор	2023-11-02
5	Андрей	Большие возможности для игр и развлечений	pos	Монитор	2023-11-02
6	Андрей	Надежная и стабильная работа компьютера	pos	Монитор	2023-11-02
7	Андрей	Множество разнообразных программ и прилож	pos	Монитор	2023-11-02
8	Андрей	Мощная система охлаждения, предотвращающа	pos	Монитор	2023-11-02
9	Андрей	Простота настройки и подключения к интернету	pos	Монитор	2023-11-02
10	Андрей	Высокая скорость загрузки и запуска программ	pos	Монитор	2023-11-02
11	Андрей	Широкие возможности для видеомонтажа и обр	pos	Монитор	2023-11-02
12	Андрей	Положительный опыт использования в игровых	pos	Монитор	2023-11-02
13	Андрей	Большой объем памяти, позволяющий хранить	pos	Монитор	2023-11-02
14	Андрей	Удобная и эргономичная клавиатура и мышь	pos	Монитор	2023-11-02
15	Андрей	Высокое качество звука и возможность подклю	pos	Монитор	2023-11-02
16	Андрей	Обширные возможности для подключения внеш	pos	Монитор	2023-11-02
17	Андрей	Многообразие портов и разъемов для подклю	pos	Монитор	2023-11-02
18	Андрей	Большой и яркий дисплей с высоким разрешени	pos	Монитор	2023-11-02
19	Андрей	Долгий срок службы и надежность компьютернс	pos	Монитор	2023-11-02
20	Андрей	Продуманный дизайн и эстетичный внешний ви	pos	Монитор	2023-11-02
21	Андрей	Этот ноутбук имеет потрясающую производител	pos	Монитор	2023-11-02
22	Андрей	Клавиатура на этом компьютере очень удобная	pos	Монитор	2023-11-02

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 ИПС. 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 МОДЕЛИ МАШИННОГО ОБУЧЕНИЯ НА ОСНОВЕ НЕЙРОННОЙ СЕТИ

3.1 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 2.

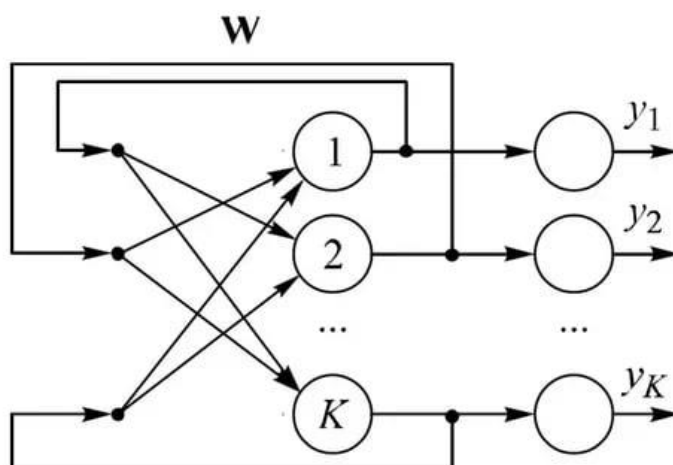


Рисунок 2 - Модель рекуррентной нейронной сети

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.

Рекуррентные нейронные сети (RNN) – это вид искусственных нейронных сетей, которые позволяют обрабатывать последовательности данных разной длины. Основным принципом работы является использование обратной связи, которая позволяет передавать информацию о предыдущих входах в следующие шаги. Рекуррентная нейронная сеть-пример работы с последовательными данными различного вида, такими как тексты, речь, временные ряды, музыкальные последовательности и другие. В отличие от сверточных нейронных сетей, RNN могут сохранять информацию о предыдущих состояниях и использовать ее для дальнейшей обработки входных данных. Рекуррентные нейронные сети применяются практически в любой сфере деятельности и могут выполнять множество задач.

Например:

- Распознавание речи: использование для обработки и распознавания речи. Они могут принимать на вход звуковые данные и выдавать текстовый результат. Примером может быть Siri или Google Assistant.
- Машинный перевод: преобразование предложений на одном языке в другой язык. Примером может быть Google Translate.
- Генерация текста: написание текстов наподобие человеческих. Это может быть полезно для создания автоматических ответов на электронные письма или создания автоматических сообщений в чатах.
- Распознавание рукописного текста: классификация записей, написанных от руки, что полезно, например, для систем распознавания подписей.
- Прогнозирование временных рядов: например, цены на акции, температура или количество продаж.
- Распознавание образов: например, в распознавании лиц или диагностике медицинских изображений.
- Музыкальное творчество: создание новой музыки, имитируя стиль и композиционные приемы известных музыкантов.

Анализ текста: определение тональности текста

В целом, применение RNN в повседневной жизни может помочь автоматизировать рутинные задачи, улучшить качество и точность работы в различных областях и даже создать новые возможности для творческой деятельности. В ходе работы была выбрана именно эта модель нейронных сетей.

3.2 Разработка рекуррентной нейронной сети (RNN)

В начале разработки мы подготовили обучающую выборку (датасет), состоящую из двух файлов, содержащих позитивные и негативные отзывы. Первым этапом разработки является подача на вход тензора со строго определёнными размерами. Так как в датасете содержатся рецензии разных размеров, короткие рецензии 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', '')

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

Рисунок 3 - Подключение файлов с положительными и отрицательными рецензиями

Для разработки собственной нейронной сети использовался язык программирования – Python, библиотеки numpy os (функций для работы с операционной системой), Tokenizer (лексический сканер для исходного кода на Python) и слои, необходимые для создания нейронных сетей.

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

import numpy as np
import re

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 4 - 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`.

```
maxWordsCount = 160
tokenizer = Tokenizer(num_words=maxWordsCount, filters='!"#$%&*+,-./:;<=>@[\\]^_`{|}~\t\n\r«»', lower=True, split=' ', char_level=False)
tokenizer.fit_on_texts(texts)

dist = list(tokenizer.word_counts.items())
#print(dist[:10])
#print(texts[0][:100])
```

Рисунок 5 - Использование метода `Tokenizer`

`Tokenizer` выполняет токенизацию, 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 ... 0 43 43]
 [855 443 274 ... 242 128 81]
 ...
 [8 0 0... 0 0 0]
 [[0 0 ... 0 0 GI]
 [0 0 0... 0 0 0]]

```

Рисунок 6 - Процесс токенизации

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

```

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

```

Рисунок 7 - 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]`.

```

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

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

```

Рисунок 8 - Создание обучающей выборки требуемых значений на выходе нейронной сети

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

```

texts = texts_true + texts_false
count_true = len(texts_true)
count_false = len(texts_false)
total_lines = count_true + count_false
#print(count_true, count_false, total_lines)

```

Рисунок 9 - 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 10 - 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 1/58
15/15 [=====] - 11s 30ms/step - loss: 0.9471 -
accuracy: 0.5110
Epoch 2/50
15/15 [=====] - 0s 31ms/step - Loss: 0.7964-
accuracy: 0.5055
Epoch 3/50
15/15 [=====] - 0s 29ms/step - loss: 0.7762-
accuracy: 0.5000
Epoch 4/50
15/15 [=====] - 0s 29ms/step - loss: 0.6532 -
accuracy: 0.6099
Epoch 5/58
15/25 [=====] - 0s 31ms/step - Loss: 0.6463-
accuracy: 0.5962
Epoch 6/50
15/15 [=====] - 0s 28ms/step - Loss: 0.5910-
accuracy: 0.6593
Epoch 7/58
15/15 [=====] - 0s 30ms/step - Loss: 0.6845-
accuracy: 0.6841
```

Рисунок 11 - Процесс самообучения

Конечным этапом разработки является определение эмоциональной окраски тестируемой выборки на основе самообучения, проходящего при помощи обучающей выборки. Для тестирования разработанного ПО использовалось создание обычной строки, куда с клавиатуры вводятся рецензии.

```
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 12 - Блок для тестирующей выборки

3.3 Анализ результатов работы программы

На основе обучающей выборки результаты проверяющей лежат в пределах от 60% до 95%. Результат зависит от количества значимых и неинформативных слов в проверяющей выборке (от 160 до 1000). Ниже представлен график зависимости точности распознавания от количества слов.

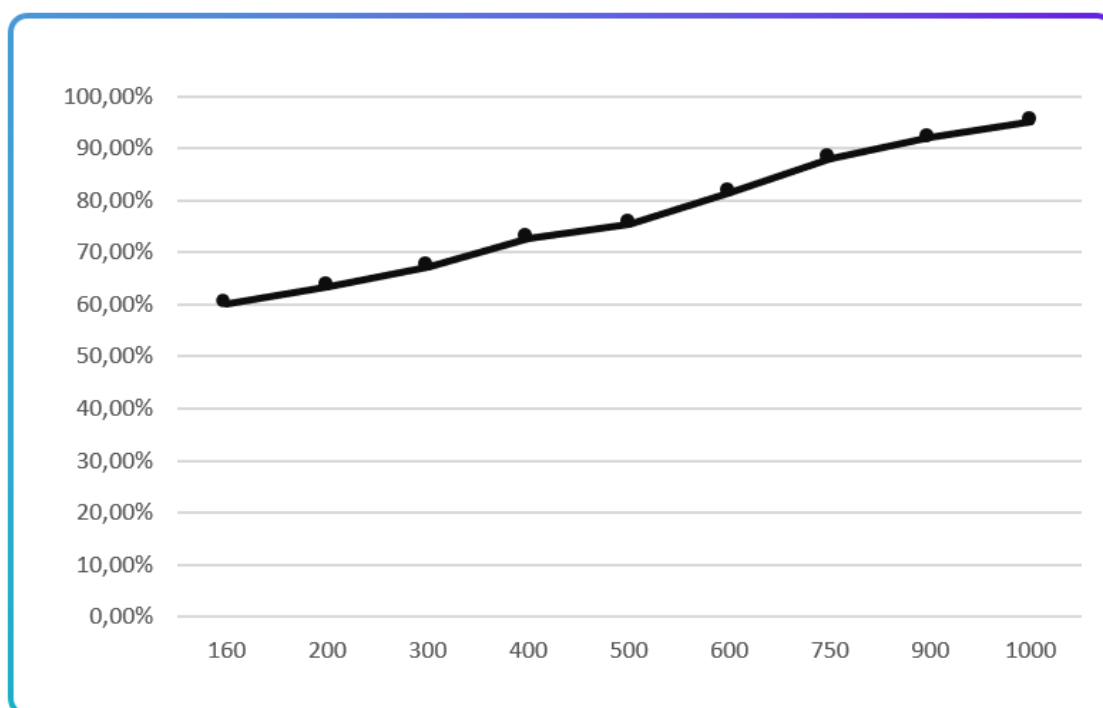


Рисунок 15 - График зависимости точности распознавания от количества слов

Анализируя приведенный график можно сделать вывод о том, что качество распознавания прямо пропорционально увеличивается с ростом количества информативных слов. Однако важно заметить, что результат является улучшаемым по средствам дообучения модели нейронной сети.

3.4 Разработка нейронной сети на основе искусственного нейрона

В качестве второго 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

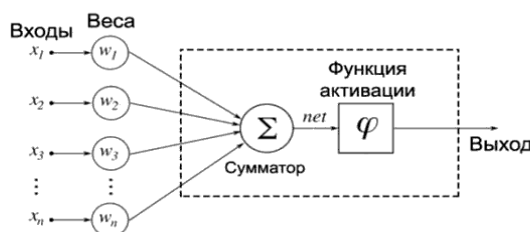


Рисунок 16 - Модель Искусственный нейрон

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: $X_1W_1 + X_2W_2 + X_3W_3 + \dots X_nW_n = net$

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

3.5 Разработка собственной нейронной сети. Искусственный нейрон.

В начале работы был освоен алгоритм распознавания Искусственный нейрон. Используемый язык программирования – Python. Для разработки использовались: библиотеки **Scikit-learn** и **nlTK** (Natural Language Toolkit) – ведущая платформа для создания NLP-программ на Python.

```
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
```

Рисунок 17 - Подключение библиотек

Следующим этапом следует подключить два файла с положительными и отрицательными отзывами.

```
# 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)
```

Рисунок 18 - Подключение файлов с рецензиями

Далее рецензии очищаются от пустых строк и пронумерованных строк.

```
# 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
```

Рисунок 19 - Процесс очистки лишних строк

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

```
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)
```

Рисунок 20 - Создание вектора

3.6 Процесс обучения нейронной сети на основе искусственного нейрона

Процесс обучения нейронной сети представляет собой исполнение большого количества итерация или “эпох”, во время которых программа с каждым разом лучше и лучше определяет тональность предложения. Ниже приведена таблица процесса обучения

Таблица 1 – процесс обучения

Кол-во Информативных слов	Итерация	Процент успешности распознавания
1000	№ 1	11%
1000	№ 2	14%
1000	№ 99	88%
1000	№ 100	89%
400	№ 1	14%
400	№ 2	15%
400	№ 99	71%
400	№ 100	72%
160	№ 1	14%
160	№ 2	15%
160	№ 99	59%
160	№ 100	73%

Анализируя полученные значения, был сделан вывод о том, что искусственный нейрона обучается и определяет тональность текста успешно. Помимо этого, данные значения являются улучшаемыми по средствам дополнительного обучения. Подробный процесс обучения приведен в Приложении Г.

3.7 Анализ результата работы программы

В ходе проделанной работы можно сделать вывод, что ПО работает корректно. Модель определяет точность распознавания, которая варьируется от 72% до 93%, а также оценивает введённый текст и присваивает ему тональность.

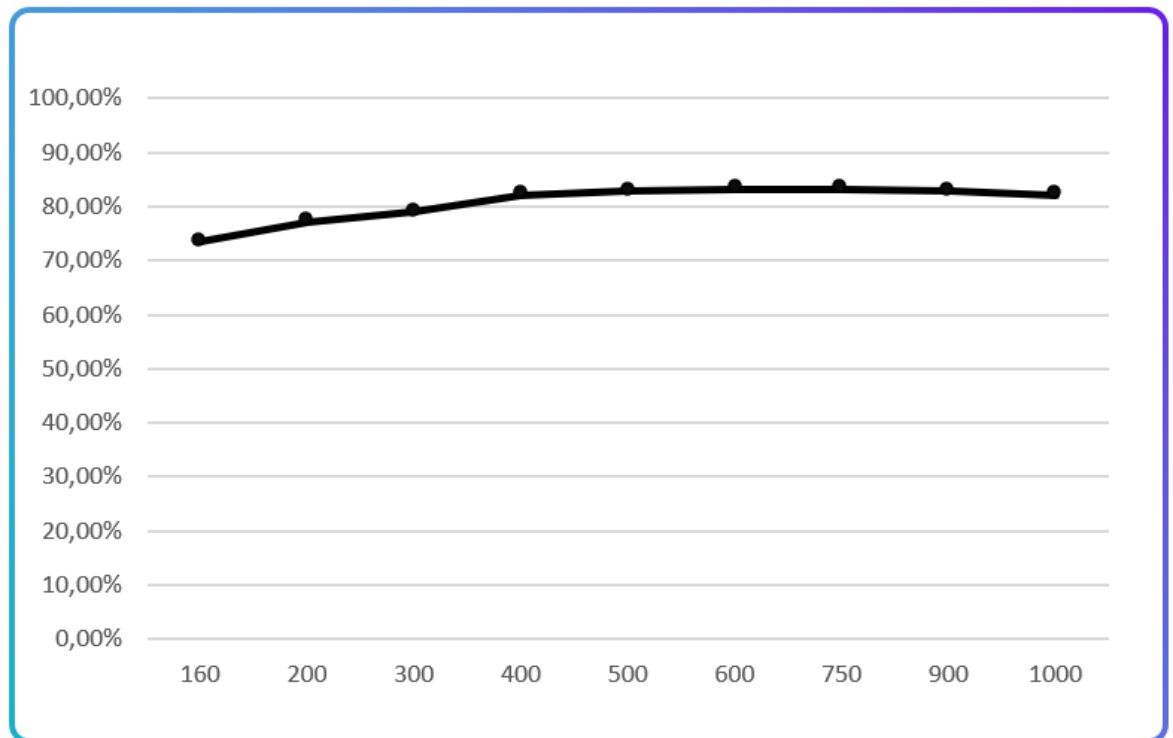


Рисунок 21 - График зависимости точности распознавания от количества слов

4 РАЗРАБОТКА И НАСТРОЙКА ИНТЕЛЛЕКТУАЛЬНОЙ ПОИСКОВОЙ СИСТЕМЫ

Для удобного взаимодействия с разработанными моделям машинного обучения была создана система интеллектуального поиска. Разработка осуществлялась на языке программирования Python с использованием ранее созданной базы данных.

The image shows a web application window with the title "Поиск отзывов на комплектующие". The interface is designed for searching and adding reviews. It features the following elements:

- Автор:** A text input field for the author's name.
- Оценка:** A text input field for the rating.
- Отзыв:** A large text area for the review content.
- Добавить отзыв**: A button to submit the review.
- Поиск отзывов:** A text input field for search queries.
- Поиск**: A button to execute the search.
- A large empty rectangular box at the bottom, intended for displaying the search results.

Рисунок 22 – Поиск отзыва

Интерфейс ИПС 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 ИПС 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 23 – Поиск отзыва

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

5 СРАВНЕНИЕ ТОЧНОСТИ ОПРЕДЕЛЕНИЯ ТОНАЛЬНОСТИ ТЕКСТА МОДЕЛЯМИ МАШИННОГО ОБУЧЕНИЯ НА ОСНОВЕ РЕКУРЕНТНОЙ НЕЙРОСЕТИ И ИСКУССТВЕННОГО НЕЙРОНА

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 это является отличным результатом, модель на основе рекуррентной нейросети уступает на 12%, что является существенной разницей. Однако при увеличении количества слов модель на основе искусственного нейрона уступает лишь на 2%, 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.

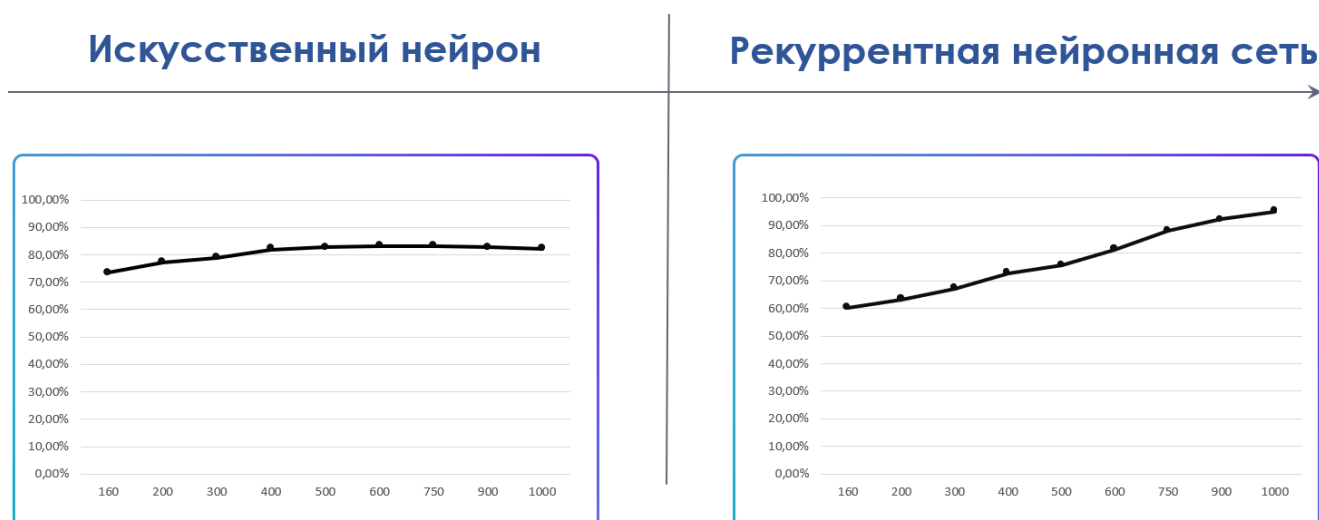


Рисунок 24 - График точности распознавания с собственным датасетом

ЗАКЛЮЧЕНИЕ

Перед моделями была поставлена задача определения тональности текста на русском языке. Для тестирования использовались отзывы о компьютерной технике с различных порталов таких, как: Яндекс.Маркет, DNS, МВидео. Результаты распознавания удовлетворили поставленную задачу. Модели успешно определяли тональность отзывов.

В ходе работы была подробно рассмотрена задача анализа тональности текстов. Была составлена база данных, состоящая из полученных отзывов. В базе данных к каждому отзыву были определены параметры для работы ИПС.

Созданы и настроены собственные ПО, произведено сравнение собственных ПО.

Была разработана и настроена система интеллектуального поиска по базе данных для удобства работы с информацией.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- Рекуррентные слои – русскоязычная документация Keras [Электронный ресурс] URL: <https://ru-keras.com/recurrent-layers/> Дата обращения [20.10.2022]
- Анализ тональности [Электронный ресурс] URL: https://nlpub.ru/Анализ_тональности Дата обращения [20.10.2022]
- 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>
- Анализ тональности текста [Электронный ресурс] URL: https://github.com/alcatraz-rm/Text_tone_analyzer

ПРИЛОЖЕНИЕ А

```
Import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np
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('C:\\Users\\Саша\\Downloads\\Telegram Desktop\\положительные.txt', 'r',
encoding='utf-8') as f:
    texts_true = f.readlines()
    texts_true[0] = texts_true[0].replace("\uffff", "")
with open('C:\\Users\\Саша\\Downloads\\Telegram Desktop\\негативные.txt', 'r',
encoding='utf-8') as f:
    texts_false = f.readlines()
    texts_false[0] = texts_false[0].replace("\uffff", "")
texts = texts_true + texts_false
count_true = len(texts_true)
count_false = len(texts_false)
total_lines = count_true + count_false
print(count_true, count_false, total_lines)
maxWordsCount = 1000
tokenizer = Tokenizer(num_words=maxWordsCount, filters='!—"—
#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n\r«»', lower=True, split=' ',
char_level=False)
tokenizer.fit_on_texts(texts)
dist = list(tokenizer.word_counts.items())
print(dist[:10])
```

```

print(texts[0][:100])
max_text_len = 10
data = tokenizer.texts_to_sequences(texts)
data_pad = pad_sequences(data, maxlen=max_text_len)
print(data_pad)
print( list(tokenizer.word_index.items()) )
X = data_pad
Y = np.array([[1, 0]]*count_true + [[0, 1]]*count_false)
print(X.shape, Y.shape)
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)
t = "".lower()
data = tokenizer.texts_to_sequences([t])
data_pad = pad_sequences(data, maxlen=max_text_len)
res = model.predict(data_pad)
print(res, sep='\n')

```

```
if (np.argmax(res) == 1):  
    print("положительный(точность указана справа)")  
else:  
    print("негативный(точность указана слева)")
```


ПРИЛОЖЕНИЕ Б

```
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
vectorizer = CountVectorizer(token_pattern=r"\b[a-яА-ЯёЁ]+\b")

# 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))
```

```

        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, test_size=0.1)

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

# Создаем и обучаем нейронную сеть
clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=10000)
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)

# Выводим результаты
for text, pred in zip(test_texts, y_pred):
    print(f"Текст: {text.strip()}")
    print(f"Тональность: {pred}")

# 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)[:10]
print("\nНаиболее информативные слова:")
for weight, word in top_words:
    print(f"{word}: {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 В

```
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()

```

ПРИЛОЖЕНИЕ Г

1000

Iteration 1, loss = 0.89159563

Iteration 2, loss = 0.86955278

Iteration 3, loss = 0.84971789

Iteration 4, loss = 0.83105314

Iteration 5, loss = 0.81304009

Iteration 6, loss = 0.79568272

Iteration 7, loss = 0.77882150

Iteration 8, loss = 0.76227710

Iteration 9, loss = 0.74591884

Iteration 10, loss = 0.72979169

Iteration 11, loss = 0.71361738

Iteration 12, loss = 0.69747437

Iteration 13, loss = 0.68144573

Iteration 14, loss = 0.66544252

Iteration 15, loss = 0.64969829

Iteration 16, loss = 0.63400033

Iteration 17, loss = 0.61846267

Iteration 18, loss = 0.60329456

Iteration 19, loss = 0.58831825

Iteration 20, loss = 0.57381826

Iteration 21, loss = 0.55956350

Iteration 22, loss = 0.54556219

Iteration 23, loss = 0.53195503

Iteration 24, loss = 0.51890238

Iteration 25, loss = 0.50628056

Iteration 26, loss = 0.49403732

Iteration 27, loss = 0.48238346

Iteration 28, loss = 0.47095544

Iteration 29, loss = 0.46020700
Iteration 30, loss = 0.44972053
Iteration 31, loss = 0.43983939
Iteration 32, loss = 0.43025261
Iteration 33, loss = 0.42110857
Iteration 34, loss = 0.41245415
Iteration 35, loss = 0.40413835
Iteration 36, loss = 0.39621613
Iteration 37, loss = 0.38865133
Iteration 38, loss = 0.38136172
Iteration 39, loss = 0.37445955
Iteration 40, loss = 0.36788662
Iteration 41, loss = 0.36160723
Iteration 42, loss = 0.35553718
Iteration 43, loss = 0.34983044
Iteration 44, loss = 0.34425885
Iteration 45, loss = 0.33901264
Iteration 46, loss = 0.33390821
Iteration 47, loss = 0.32907259
Iteration 48, loss = 0.32450197
Iteration 49, loss = 0.31995873
Iteration 50, loss = 0.31567564
Iteration 51, loss = 0.31163061
Iteration 52, loss = 0.30767829
Iteration 53, loss = 0.30391512
Iteration 54, loss = 0.30027645
Iteration 55, loss = 0.29668165
Iteration 56, loss = 0.29340102
Iteration 57, loss = 0.29010788
Iteration 58, loss = 0.28684336

Iteration 59, loss = 0.28393992
Iteration 60, loss = 0.28093404
Iteration 61, loss = 0.27806252
Iteration 62, loss = 0.27533234
Iteration 63, loss = 0.27266009
Iteration 64, loss = 0.27011674
Iteration 65, loss = 0.26758803
Iteration 66, loss = 0.26519054
Iteration 67, loss = 0.26278044
Iteration 68, loss = 0.26058772
Iteration 69, loss = 0.25837269
Iteration 70, loss = 0.25621914
Iteration 71, loss = 0.25418426
Iteration 72, loss = 0.25217028
Iteration 73, loss = 0.25006912
Iteration 74, loss = 0.24817133
Iteration 75, loss = 0.24630588
Iteration 76, loss = 0.24462083
Iteration 77, loss = 0.24269897
Iteration 78, loss = 0.24094418
Iteration 79, loss = 0.23932644
Iteration 80, loss = 0.23769732
Iteration 81, loss = 0.23613581
Iteration 82, loss = 0.23454602
Iteration 83, loss = 0.23308006
Iteration 84, loss = 0.23162569
Iteration 85, loss = 0.23009615
Iteration 86, loss = 0.22883121
Iteration 87, loss = 0.22746128
Iteration 88, loss = 0.22606367

Iteration 89, loss = 0.22479805
Iteration 90, loss = 0.22350305
Iteration 91, loss = 0.22227296
Iteration 92, loss = 0.22101978
Iteration 93, loss = 0.21981598
Iteration 94, loss = 0.21868815
Iteration 95, loss = 0.21758916
Iteration 96, loss = 0.21647107
Iteration 97, loss = 0.21539082
Iteration 98, loss = 0.21427759
Iteration 99, loss = 0.21331924
Iteration 100, loss = 0.21223495
900

Iteration 1, loss = 0.89197377
Iteration 2, loss = 0.87038891
Iteration 3, loss = 0.85103922
Iteration 4, loss = 0.83259599
Iteration 5, loss = 0.81492110
Iteration 6, loss = 0.79780636
Iteration 7, loss = 0.78091527
Iteration 8, loss = 0.76456449
Iteration 9, loss = 0.74832736
Iteration 10, loss = 0.73234215
Iteration 11, loss = 0.71645461
Iteration 12, loss = 0.70078437
Iteration 13, loss = 0.68518458
Iteration 14, loss = 0.66986261
Iteration 15, loss = 0.65440860
Iteration 16, loss = 0.63936215
Iteration 17, loss = 0.62434667

Iteration 18, loss = 0.60968383
Iteration 19, loss = 0.59514834
Iteration 20, loss = 0.58106600
Iteration 21, loss = 0.56715879
Iteration 22, loss = 0.55371611
Iteration 23, loss = 0.54047729
Iteration 24, loss = 0.52766115
Iteration 25, loss = 0.51521084
Iteration 26, loss = 0.50317771
Iteration 27, loss = 0.49182049
Iteration 28, loss = 0.48074560
Iteration 29, loss = 0.47014800
Iteration 30, loss = 0.45988862
Iteration 31, loss = 0.45019268
Iteration 32, loss = 0.44077199
Iteration 33, loss = 0.43170612
Iteration 34, loss = 0.42318510
Iteration 35, loss = 0.41489262
Iteration 36, loss = 0.40699972
Iteration 37, loss = 0.39948219
Iteration 38, loss = 0.39222212
Iteration 39, loss = 0.38527489
Iteration 40, loss = 0.37868568
Iteration 41, loss = 0.37240391
Iteration 42, loss = 0.36636698
Iteration 43, loss = 0.36048025
Iteration 44, loss = 0.35493221
Iteration 45, loss = 0.34966598
Iteration 46, loss = 0.34446076
Iteration 47, loss = 0.33954601

Iteration 48, loss = 0.33481662
Iteration 49, loss = 0.33023049
Iteration 50, loss = 0.32594702
Iteration 51, loss = 0.32181382
Iteration 52, loss = 0.31772450
Iteration 53, loss = 0.31376885
Iteration 54, loss = 0.31004030
Iteration 55, loss = 0.30647503
Iteration 56, loss = 0.30298147
Iteration 57, loss = 0.29956116
Iteration 58, loss = 0.29634046
Iteration 59, loss = 0.29323571
Iteration 60, loss = 0.29017960
Iteration 61, loss = 0.28724496
Iteration 62, loss = 0.28437910
Iteration 63, loss = 0.28160640
Iteration 64, loss = 0.27896757
Iteration 65, loss = 0.27634382
Iteration 66, loss = 0.27388340
Iteration 67, loss = 0.27155163
Iteration 68, loss = 0.26907381
Iteration 69, loss = 0.26691843
Iteration 70, loss = 0.26463200
Iteration 71, loss = 0.26250070
Iteration 72, loss = 0.26042637
Iteration 73, loss = 0.25841038
Iteration 74, loss = 0.25639722
Iteration 75, loss = 0.25453868
Iteration 76, loss = 0.25261842
Iteration 77, loss = 0.25087065

Iteration 78, loss = 0.24910284
Iteration 79, loss = 0.24730625
Iteration 80, loss = 0.24562512
Iteration 81, loss = 0.24395590
Iteration 82, loss = 0.24241420
Iteration 83, loss = 0.24085170
Iteration 84, loss = 0.23927040
Iteration 85, loss = 0.23777537
Iteration 86, loss = 0.23638428
Iteration 87, loss = 0.23494823
Iteration 88, loss = 0.23353879
Iteration 89, loss = 0.23219195
Iteration 90, loss = 0.23084927
Iteration 91, loss = 0.22956672
Iteration 92, loss = 0.22829014
Iteration 93, loss = 0.22702878
Iteration 94, loss = 0.22584457
Iteration 95, loss = 0.22467264
Iteration 96, loss = 0.22353158
Iteration 97, loss = 0.22239777
Iteration 98, loss = 0.22124295
Iteration 99, loss = 0.22017134
Iteration 100, loss = 0.21914156

750

Iteration 1, loss = 0.87277230
Iteration 2, loss = 0.85328775
Iteration 3, loss = 0.83559967
Iteration 4, loss = 0.81876518
Iteration 5, loss = 0.80241414
Iteration 6, loss = 0.78647918

Iteration 7, loss = 0.77086398
Iteration 8, loss = 0.75573310
Iteration 9, loss = 0.74075624
Iteration 10, loss = 0.72611409
Iteration 11, loss = 0.71139265
Iteration 12, loss = 0.69698821
Iteration 13, loss = 0.68261093
Iteration 14, loss = 0.66840473
Iteration 15, loss = 0.65436685
Iteration 16, loss = 0.64034370
Iteration 17, loss = 0.62662881
Iteration 18, loss = 0.61296530
Iteration 19, loss = 0.59935685
Iteration 20, loss = 0.58621240
Iteration 21, loss = 0.57318935
Iteration 22, loss = 0.56033219
Iteration 23, loss = 0.54786019
Iteration 24, loss = 0.53564023
Iteration 25, loss = 0.52371927
Iteration 26, loss = 0.51218213
Iteration 27, loss = 0.50107719
Iteration 28, loss = 0.49037111
Iteration 29, loss = 0.47994470
Iteration 30, loss = 0.47006283
Iteration 31, loss = 0.46044818
Iteration 32, loss = 0.45124148
Iteration 33, loss = 0.44243990
Iteration 34, loss = 0.43401106
Iteration 35, loss = 0.42588355
Iteration 36, loss = 0.41807301

Iteration 37, loss = 0.41065991
Iteration 38, loss = 0.40353077
Iteration 39, loss = 0.39666614
Iteration 40, loss = 0.39013398
Iteration 41, loss = 0.38390492
Iteration 42, loss = 0.37791507
Iteration 43, loss = 0.37212765
Iteration 44, loss = 0.36666787
Iteration 45, loss = 0.36133678
Iteration 46, loss = 0.35630181
Iteration 47, loss = 0.35134362
Iteration 48, loss = 0.34676464
Iteration 49, loss = 0.34218181
Iteration 50, loss = 0.33794008
Iteration 51, loss = 0.33369286
Iteration 52, loss = 0.32972304
Iteration 53, loss = 0.32588082
Iteration 54, loss = 0.32202615
Iteration 55, loss = 0.31863131
Iteration 56, loss = 0.31498837
Iteration 57, loss = 0.31166793
Iteration 58, loss = 0.30838749
Iteration 59, loss = 0.30517961
Iteration 60, loss = 0.30218242
Iteration 61, loss = 0.29923495
Iteration 62, loss = 0.29637536
Iteration 63, loss = 0.29358533
Iteration 64, loss = 0.29090546
Iteration 65, loss = 0.28829624
Iteration 66, loss = 0.28571488

Iteration 67, loss = 0.28335628
Iteration 68, loss = 0.28091347
Iteration 69, loss = 0.27862378
Iteration 70, loss = 0.27636904
Iteration 71, loss = 0.27417511
Iteration 72, loss = 0.27206626
Iteration 73, loss = 0.27015800
Iteration 74, loss = 0.26807567
Iteration 75, loss = 0.26608274
Iteration 76, loss = 0.26421134
Iteration 77, loss = 0.26237351
Iteration 78, loss = 0.26054327
Iteration 79, loss = 0.25884508
Iteration 80, loss = 0.25716831
Iteration 81, loss = 0.25543082
Iteration 82, loss = 0.25389785
Iteration 83, loss = 0.25230696
Iteration 84, loss = 0.25075712
Iteration 85, loss = 0.24920962
Iteration 86, loss = 0.24781172
Iteration 87, loss = 0.24634522
Iteration 88, loss = 0.24489292
Iteration 89, loss = 0.24352887
Iteration 90, loss = 0.24220305
Iteration 91, loss = 0.24085598
Iteration 92, loss = 0.23958477
Iteration 93, loss = 0.23837945
Iteration 94, loss = 0.23715403
Iteration 95, loss = 0.23587612
Iteration 96, loss = 0.23475208

Iteration 97, loss = 0.23358517
Iteration 98, loss = 0.23248880
Iteration 99, loss = 0.23136684
Iteration 100, loss = 0.23026005
600
Iteration 1, loss = 0.87684171
Iteration 2, loss = 0.85975423
Iteration 3, loss = 0.84418125
Iteration 4, loss = 0.82918273
Iteration 5, loss = 0.81473380
Iteration 6, loss = 0.80055982
Iteration 7, loss = 0.78675513
Iteration 8, loss = 0.77318970
Iteration 9, loss = 0.75983863
Iteration 10, loss = 0.74664584
Iteration 11, loss = 0.73354398
Iteration 12, loss = 0.72046589
Iteration 13, loss = 0.70754893
Iteration 14, loss = 0.69459634
Iteration 15, loss = 0.68169294
Iteration 16, loss = 0.66872126
Iteration 17, loss = 0.65593663
Iteration 18, loss = 0.64318019
Iteration 19, loss = 0.63065371
Iteration 20, loss = 0.61816420
Iteration 21, loss = 0.60581065
Iteration 22, loss = 0.59371382
Iteration 23, loss = 0.58182476
Iteration 24, loss = 0.57009270
Iteration 25, loss = 0.55874909

Iteration 26, loss = 0.54760045
Iteration 27, loss = 0.53665811
Iteration 28, loss = 0.52614724
Iteration 29, loss = 0.51593183
Iteration 30, loss = 0.50601302
Iteration 31, loss = 0.49659433
Iteration 32, loss = 0.48734598
Iteration 33, loss = 0.47858470
Iteration 34, loss = 0.46996624
Iteration 35, loss = 0.46202753
Iteration 36, loss = 0.45410386
Iteration 37, loss = 0.44648033
Iteration 38, loss = 0.43927960
Iteration 39, loss = 0.43224776
Iteration 40, loss = 0.42553603
Iteration 41, loss = 0.41916843
Iteration 42, loss = 0.41296660
Iteration 43, loss = 0.40701888
Iteration 44, loss = 0.40131820
Iteration 45, loss = 0.39582217
Iteration 46, loss = 0.39067044
Iteration 47, loss = 0.38558413
Iteration 48, loss = 0.38067014
Iteration 49, loss = 0.37606514
Iteration 50, loss = 0.37152071
Iteration 51, loss = 0.36728321
Iteration 52, loss = 0.36314010
Iteration 53, loss = 0.35908497
Iteration 54, loss = 0.35521196
Iteration 55, loss = 0.35140367

Iteration 56, loss = 0.34786086
Iteration 57, loss = 0.34442372
Iteration 58, loss = 0.34109661
Iteration 59, loss = 0.33782961
Iteration 60, loss = 0.33468399
Iteration 61, loss = 0.33171410
Iteration 62, loss = 0.32870408
Iteration 63, loss = 0.32574750
Iteration 64, loss = 0.32310862
Iteration 65, loss = 0.32041391
Iteration 66, loss = 0.31780693
Iteration 67, loss = 0.31520005
Iteration 68, loss = 0.31282916
Iteration 69, loss = 0.31035297
Iteration 70, loss = 0.30822910
Iteration 71, loss = 0.30587203
Iteration 72, loss = 0.30371387
Iteration 73, loss = 0.30158435
Iteration 74, loss = 0.29947915
Iteration 75, loss = 0.29748135
Iteration 76, loss = 0.29548281
Iteration 77, loss = 0.29364140
Iteration 78, loss = 0.29175340
Iteration 79, loss = 0.28992663
Iteration 80, loss = 0.28826168
Iteration 81, loss = 0.28652587
Iteration 82, loss = 0.28480522
Iteration 83, loss = 0.28319981
Iteration 84, loss = 0.28156150
Iteration 85, loss = 0.28004224

Iteration 86, loss = 0.27851108
Iteration 87, loss = 0.27708551
Iteration 88, loss = 0.27551690
Iteration 89, loss = 0.27416635
Iteration 90, loss = 0.27271437
Iteration 91, loss = 0.27141632
Iteration 92, loss = 0.27007212
Iteration 93, loss = 0.26867956
Iteration 94, loss = 0.26744816
Iteration 95, loss = 0.26620919
Iteration 96, loss = 0.26496661
Iteration 97, loss = 0.26373867
Iteration 98, loss = 0.26253773
Iteration 99, loss = 0.26137137
Iteration 100, loss = 0.26025775

500

Iteration 1, loss = 0.88409986
Iteration 2, loss = 0.86768489
Iteration 3, loss = 0.85247211
Iteration 4, loss = 0.83794249
Iteration 5, loss = 0.82377857
Iteration 6, loss = 0.80990144
Iteration 7, loss = 0.79635145
Iteration 8, loss = 0.78306302
Iteration 9, loss = 0.76985321
Iteration 10, loss = 0.75689737
Iteration 11, loss = 0.74407996
Iteration 12, loss = 0.73134811
Iteration 13, loss = 0.71869186
Iteration 14, loss = 0.70618306

Iteration 15, loss = 0.69372973
Iteration 16, loss = 0.68151021
Iteration 17, loss = 0.66918067
Iteration 18, loss = 0.65722684
Iteration 19, loss = 0.64511305
Iteration 20, loss = 0.63325963
Iteration 21, loss = 0.62162828
Iteration 22, loss = 0.61005165
Iteration 23, loss = 0.59870678
Iteration 24, loss = 0.58755525
Iteration 25, loss = 0.57666613
Iteration 26, loss = 0.56596418
Iteration 27, loss = 0.55545660
Iteration 28, loss = 0.54523158
Iteration 29, loss = 0.53526157
Iteration 30, loss = 0.52567367
Iteration 31, loss = 0.51623296
Iteration 32, loss = 0.50707967
Iteration 33, loss = 0.49823124
Iteration 34, loss = 0.48975651
Iteration 35, loss = 0.48155469
Iteration 36, loss = 0.47359507
Iteration 37, loss = 0.46615083
Iteration 38, loss = 0.45874642
Iteration 39, loss = 0.45171901
Iteration 40, loss = 0.44497771
Iteration 41, loss = 0.43848255
Iteration 42, loss = 0.43227974
Iteration 43, loss = 0.42613245
Iteration 44, loss = 0.42023326

Iteration 45, loss = 0.41461739
Iteration 46, loss = 0.40924434
Iteration 47, loss = 0.40410847
Iteration 48, loss = 0.39909209
Iteration 49, loss = 0.39427629
Iteration 50, loss = 0.38959156
Iteration 51, loss = 0.38507524
Iteration 52, loss = 0.38068287
Iteration 53, loss = 0.37659090
Iteration 54, loss = 0.37243634
Iteration 55, loss = 0.36856089
Iteration 56, loss = 0.36470468
Iteration 57, loss = 0.36099240
Iteration 58, loss = 0.35738092
Iteration 59, loss = 0.35405747
Iteration 60, loss = 0.35065615
Iteration 61, loss = 0.34743777
Iteration 62, loss = 0.34433213
Iteration 63, loss = 0.34124863
Iteration 64, loss = 0.33826367
Iteration 65, loss = 0.33530174
Iteration 66, loss = 0.33264938
Iteration 67, loss = 0.32992331
Iteration 68, loss = 0.32737714
Iteration 69, loss = 0.32482073
Iteration 70, loss = 0.32228761
Iteration 71, loss = 0.31987222
Iteration 72, loss = 0.31753541
Iteration 73, loss = 0.31525639
Iteration 74, loss = 0.31308470

Iteration 75, loss = 0.31083597
Iteration 76, loss = 0.30878353
Iteration 77, loss = 0.30680839
Iteration 78, loss = 0.30478291
Iteration 79, loss = 0.30283352
Iteration 80, loss = 0.30093736
Iteration 81, loss = 0.29911768
Iteration 82, loss = 0.29738708
Iteration 83, loss = 0.29558199
Iteration 84, loss = 0.29376842
Iteration 85, loss = 0.29206718
Iteration 86, loss = 0.29047962
Iteration 87, loss = 0.28888872
Iteration 88, loss = 0.28726330
Iteration 89, loss = 0.28575156
Iteration 90, loss = 0.28428418
Iteration 91, loss = 0.28293041
Iteration 92, loss = 0.28137886
Iteration 93, loss = 0.27997237
Iteration 94, loss = 0.27857529
Iteration 95, loss = 0.27719760
Iteration 96, loss = 0.27589281
Iteration 97, loss = 0.27462674
Iteration 98, loss = 0.27334488
Iteration 99, loss = 0.27207780
Iteration 100, loss = 0.27083950
400
Iteration 1, loss = 0.86702395
Iteration 2, loss = 0.85187286
Iteration 3, loss = 0.83797709

Iteration 4, loss = 0.82442947
Iteration 5, loss = 0.81133473
Iteration 6, loss = 0.79841336
Iteration 7, loss = 0.78594627
Iteration 8, loss = 0.77339474
Iteration 9, loss = 0.76137259
Iteration 10, loss = 0.74931219
Iteration 11, loss = 0.73741782
Iteration 12, loss = 0.72558737
Iteration 13, loss = 0.71385865
Iteration 14, loss = 0.70228904
Iteration 15, loss = 0.69070326
Iteration 16, loss = 0.67930379
Iteration 17, loss = 0.66786263
Iteration 18, loss = 0.65659253
Iteration 19, loss = 0.64542032
Iteration 20, loss = 0.63431917
Iteration 21, loss = 0.62339234
Iteration 22, loss = 0.61259150
Iteration 23, loss = 0.60210559
Iteration 24, loss = 0.59170076
Iteration 25, loss = 0.58154123
Iteration 26, loss = 0.57175163
Iteration 27, loss = 0.56195130
Iteration 28, loss = 0.55265739
Iteration 29, loss = 0.54359444
Iteration 30, loss = 0.53459148
Iteration 31, loss = 0.52609323
Iteration 32, loss = 0.51768625
Iteration 33, loss = 0.50958187

Iteration 34, loss = 0.50198290
Iteration 35, loss = 0.49422935
Iteration 36, loss = 0.48687948
Iteration 37, loss = 0.47982187
Iteration 38, loss = 0.47299874
Iteration 39, loss = 0.46628209
Iteration 40, loss = 0.45994528
Iteration 41, loss = 0.45372951
Iteration 42, loss = 0.44780701
Iteration 43, loss = 0.44211566
Iteration 44, loss = 0.43651092
Iteration 45, loss = 0.43107093
Iteration 46, loss = 0.42585093
Iteration 47, loss = 0.42079912
Iteration 48, loss = 0.41603018
Iteration 49, loss = 0.41129700
Iteration 50, loss = 0.40672138
Iteration 51, loss = 0.40231730
Iteration 52, loss = 0.39805278
Iteration 53, loss = 0.39394078
Iteration 54, loss = 0.39006227
Iteration 55, loss = 0.38612486
Iteration 56, loss = 0.38240370
Iteration 57, loss = 0.37870752
Iteration 58, loss = 0.37521164
Iteration 59, loss = 0.37190349
Iteration 60, loss = 0.36862274
Iteration 61, loss = 0.36528568
Iteration 62, loss = 0.36222284
Iteration 63, loss = 0.35914931

Iteration 64, loss = 0.35616494
Iteration 65, loss = 0.35340609
Iteration 66, loss = 0.35062150
Iteration 67, loss = 0.34789460
Iteration 68, loss = 0.34529583
Iteration 69, loss = 0.34275542
Iteration 70, loss = 0.34020069
Iteration 71, loss = 0.33783545
Iteration 72, loss = 0.33553405
Iteration 73, loss = 0.33335129
Iteration 74, loss = 0.33103365
Iteration 75, loss = 0.32878184
Iteration 76, loss = 0.32667518
Iteration 77, loss = 0.32460709
Iteration 78, loss = 0.32260355
Iteration 79, loss = 0.32057846
Iteration 80, loss = 0.31867273
Iteration 81, loss = 0.31680641
Iteration 82, loss = 0.31496903
Iteration 83, loss = 0.31323033
Iteration 84, loss = 0.31142200
Iteration 85, loss = 0.30977613
Iteration 86, loss = 0.30804878
Iteration 87, loss = 0.30636840
Iteration 88, loss = 0.30478239
Iteration 89, loss = 0.30327298
Iteration 90, loss = 0.30168330
Iteration 91, loss = 0.30018906
Iteration 92, loss = 0.29863913
Iteration 93, loss = 0.29730606

Iteration 94, loss = 0.29582925
Iteration 95, loss = 0.29442788
Iteration 96, loss = 0.29308796
Iteration 97, loss = 0.29176197
Iteration 98, loss = 0.29040131
Iteration 99, loss = 0.28916720
Iteration 100, loss = 0.28791654
300

Iteration 1, loss = 0.87029262
Iteration 2, loss = 0.85569128
Iteration 3, loss = 0.84205007
Iteration 4, loss = 0.82885687
Iteration 5, loss = 0.81619471
Iteration 6, loss = 0.80372769
Iteration 7, loss = 0.79157661
Iteration 8, loss = 0.77954286
Iteration 9, loss = 0.76807055
Iteration 10, loss = 0.75642293
Iteration 11, loss = 0.74521666
Iteration 12, loss = 0.73406876
Iteration 13, loss = 0.72311670
Iteration 14, loss = 0.71223222
Iteration 15, loss = 0.70152787
Iteration 16, loss = 0.69093912
Iteration 17, loss = 0.68041877
Iteration 18, loss = 0.67017448
Iteration 19, loss = 0.65984045
Iteration 20, loss = 0.64981288
Iteration 21, loss = 0.63975072
Iteration 22, loss = 0.63000447

Iteration 23, loss = 0.62012798
Iteration 24, loss = 0.61052531
Iteration 25, loss = 0.60113556
Iteration 26, loss = 0.59192823
Iteration 27, loss = 0.58266914
Iteration 28, loss = 0.57378010
Iteration 29, loss = 0.56486421
Iteration 30, loss = 0.55621087
Iteration 31, loss = 0.54760720
Iteration 32, loss = 0.53939248
Iteration 33, loss = 0.53141523
Iteration 34, loss = 0.52339651
Iteration 35, loss = 0.51570957
Iteration 36, loss = 0.50811665
Iteration 37, loss = 0.50082642
Iteration 38, loss = 0.49372564
Iteration 39, loss = 0.48685812
Iteration 40, loss = 0.48013553
Iteration 41, loss = 0.47370124
Iteration 42, loss = 0.46730862
Iteration 43, loss = 0.46124307
Iteration 44, loss = 0.45532647
Iteration 45, loss = 0.44954826
Iteration 46, loss = 0.44402842
Iteration 47, loss = 0.43861716
Iteration 48, loss = 0.43350716
Iteration 49, loss = 0.42857811
Iteration 50, loss = 0.42365155
Iteration 51, loss = 0.41905433
Iteration 52, loss = 0.41453380

Iteration 53, loss = 0.41019249
Iteration 54, loss = 0.40604011
Iteration 55, loss = 0.40199784
Iteration 56, loss = 0.39808627
Iteration 57, loss = 0.39426870
Iteration 58, loss = 0.39059852
Iteration 59, loss = 0.38703570
Iteration 60, loss = 0.38357081
Iteration 61, loss = 0.38033411
Iteration 62, loss = 0.37713590
Iteration 63, loss = 0.37399555
Iteration 64, loss = 0.37096833
Iteration 65, loss = 0.36811401
Iteration 66, loss = 0.36531968
Iteration 67, loss = 0.36256493
Iteration 68, loss = 0.35992544
Iteration 69, loss = 0.35734255
Iteration 70, loss = 0.35481798
Iteration 71, loss = 0.35236996
Iteration 72, loss = 0.35011144
Iteration 73, loss = 0.34796325
Iteration 74, loss = 0.34551748
Iteration 75, loss = 0.34341554
Iteration 76, loss = 0.34130893
Iteration 77, loss = 0.33926459
Iteration 78, loss = 0.33723473
Iteration 79, loss = 0.33539390
Iteration 80, loss = 0.33342597
Iteration 81, loss = 0.33162710
Iteration 82, loss = 0.32976353

Iteration 83, loss = 0.32812886
Iteration 84, loss = 0.32635420
Iteration 85, loss = 0.32469585
Iteration 86, loss = 0.32304490
Iteration 87, loss = 0.32145840
Iteration 88, loss = 0.31993111
Iteration 89, loss = 0.31835587
Iteration 90, loss = 0.31698329
Iteration 91, loss = 0.31539560
Iteration 92, loss = 0.31399365
Iteration 93, loss = 0.31267129
Iteration 94, loss = 0.31123319
Iteration 95, loss = 0.30995016
Iteration 96, loss = 0.30863142
Iteration 97, loss = 0.30741758
Iteration 98, loss = 0.30610103
Iteration 99, loss = 0.30487952
Iteration 100, loss = 0.30370693

200

Iteration 1, loss = 0.86851672
Iteration 2, loss = 0.85600685
Iteration 3, loss = 0.84428840
Iteration 4, loss = 0.83311587
Iteration 5, loss = 0.82220071
Iteration 6, loss = 0.81166135
Iteration 7, loss = 0.80135143
Iteration 8, loss = 0.79141872
Iteration 9, loss = 0.78142624
Iteration 10, loss = 0.77184111
Iteration 11, loss = 0.76231628

Iteration 12, loss = 0.75300048
Iteration 13, loss = 0.74375567
Iteration 14, loss = 0.73477544
Iteration 15, loss = 0.72573932
Iteration 16, loss = 0.71688646
Iteration 17, loss = 0.70810342
Iteration 18, loss = 0.69943382
Iteration 19, loss = 0.69082800
Iteration 20, loss = 0.68242096
Iteration 21, loss = 0.67408259
Iteration 22, loss = 0.66580972
Iteration 23, loss = 0.65773746
Iteration 24, loss = 0.64984530
Iteration 25, loss = 0.64197383
Iteration 26, loss = 0.63433958
Iteration 27, loss = 0.62675678
Iteration 28, loss = 0.61932109
Iteration 29, loss = 0.61218400
Iteration 30, loss = 0.60496483
Iteration 31, loss = 0.59806040
Iteration 32, loss = 0.59123387
Iteration 33, loss = 0.58458519
Iteration 34, loss = 0.57796325
Iteration 35, loss = 0.57162689
Iteration 36, loss = 0.56542908
Iteration 37, loss = 0.55932850
Iteration 38, loss = 0.55342224
Iteration 39, loss = 0.54764355
Iteration 40, loss = 0.54199150
Iteration 41, loss = 0.53647913

Iteration 42, loss = 0.53109187
Iteration 43, loss = 0.52590065
Iteration 44, loss = 0.52074373
Iteration 45, loss = 0.51583369
Iteration 46, loss = 0.51098220
Iteration 47, loss = 0.50625701
Iteration 48, loss = 0.50160068
Iteration 49, loss = 0.49720786
Iteration 50, loss = 0.49279648
Iteration 51, loss = 0.48858579
Iteration 52, loss = 0.48454665
Iteration 53, loss = 0.48048596
Iteration 54, loss = 0.47658322
Iteration 55, loss = 0.47286551
Iteration 56, loss = 0.46928769
Iteration 57, loss = 0.46571704
Iteration 58, loss = 0.46228467
Iteration 59, loss = 0.45891805
Iteration 60, loss = 0.45567043
Iteration 61, loss = 0.45246132
Iteration 62, loss = 0.44936141
Iteration 63, loss = 0.44634022
Iteration 64, loss = 0.44343102
Iteration 65, loss = 0.44047654
Iteration 66, loss = 0.43770941
Iteration 67, loss = 0.43502161
Iteration 68, loss = 0.43233922
Iteration 69, loss = 0.42976532
Iteration 70, loss = 0.42726827
Iteration 71, loss = 0.42486124

Iteration 72, loss = 0.42243395
Iteration 73, loss = 0.42004486
Iteration 74, loss = 0.41785243
Iteration 75, loss = 0.41562572
Iteration 76, loss = 0.41343278
Iteration 77, loss = 0.41134618
Iteration 78, loss = 0.40930909
Iteration 79, loss = 0.40738950
Iteration 80, loss = 0.40534901
Iteration 81, loss = 0.40345508
Iteration 82, loss = 0.40151896
Iteration 83, loss = 0.39972105
Iteration 84, loss = 0.39790598
Iteration 85, loss = 0.39612872
Iteration 86, loss = 0.39445269
Iteration 87, loss = 0.39272860
Iteration 88, loss = 0.39103501
Iteration 89, loss = 0.38944062
Iteration 90, loss = 0.38790579
Iteration 91, loss = 0.38633448
Iteration 92, loss = 0.38486527
Iteration 93, loss = 0.38333640
Iteration 94, loss = 0.38184222
Iteration 95, loss = 0.38047137
Iteration 96, loss = 0.37907935
Iteration 97, loss = 0.37773863
Iteration 98, loss = 0.37638811
Iteration 99, loss = 0.37501669
Iteration 100, loss = 0.37377813

Iteration 1, loss = 0.86319628
Iteration 2, loss = 0.85280443
Iteration 3, loss = 0.84304628
Iteration 4, loss = 0.83358514
Iteration 5, loss = 0.82443685
Iteration 6, loss = 0.81562573
Iteration 7, loss = 0.80692785
Iteration 8, loss = 0.79850358
Iteration 9, loss = 0.79028089
Iteration 10, loss = 0.78208875
Iteration 11, loss = 0.77402974
Iteration 12, loss = 0.76611228
Iteration 13, loss = 0.75824506
Iteration 14, loss = 0.75053754
Iteration 15, loss = 0.74288865
Iteration 16, loss = 0.73526688
Iteration 17, loss = 0.72778638
Iteration 18, loss = 0.72021436
Iteration 19, loss = 0.71279257
Iteration 20, loss = 0.70542067
Iteration 21, loss = 0.69804331
Iteration 22, loss = 0.69072071
Iteration 23, loss = 0.68343142
Iteration 24, loss = 0.67622392
Iteration 25, loss = 0.66908303
Iteration 26, loss = 0.66202849
Iteration 27, loss = 0.65493894
Iteration 28, loss = 0.64806551
Iteration 29, loss = 0.64117933
Iteration 30, loss = 0.63440634

Iteration 31, loss = 0.62775542
Iteration 32, loss = 0.62117421
Iteration 33, loss = 0.61466959
Iteration 34, loss = 0.60835690
Iteration 35, loss = 0.60211957
Iteration 36, loss = 0.59594228
Iteration 37, loss = 0.58986394
Iteration 38, loss = 0.58397832
Iteration 39, loss = 0.57826958
Iteration 40, loss = 0.57263207
Iteration 41, loss = 0.56715477
Iteration 42, loss = 0.56169402
Iteration 43, loss = 0.55647767
Iteration 44, loss = 0.55130243
Iteration 45, loss = 0.54640191
Iteration 46, loss = 0.54144471
Iteration 47, loss = 0.53677448
Iteration 48, loss = 0.53219910
Iteration 49, loss = 0.52778348
Iteration 50, loss = 0.52342346
Iteration 51, loss = 0.51928014
Iteration 52, loss = 0.51516332
Iteration 53, loss = 0.51115782
Iteration 54, loss = 0.50726520
Iteration 55, loss = 0.50351697
Iteration 56, loss = 0.49982772
Iteration 57, loss = 0.49625348
Iteration 58, loss = 0.49283070
Iteration 59, loss = 0.48951657
Iteration 60, loss = 0.48628381

Iteration 61, loss = 0.48304406
Iteration 62, loss = 0.48004193
Iteration 63, loss = 0.47706956
Iteration 64, loss = 0.47419018
Iteration 65, loss = 0.47146536
Iteration 66, loss = 0.46874047
Iteration 67, loss = 0.46605345
Iteration 68, loss = 0.46356803
Iteration 69, loss = 0.46104515
Iteration 70, loss = 0.45867586
Iteration 71, loss = 0.45636590
Iteration 72, loss = 0.45405660
Iteration 73, loss = 0.45194168
Iteration 74, loss = 0.44975182
Iteration 75, loss = 0.44779751
Iteration 76, loss = 0.44567693
Iteration 77, loss = 0.44364107
Iteration 78, loss = 0.44170336
Iteration 79, loss = 0.43979611
Iteration 80, loss = 0.43810407
Iteration 81, loss = 0.43628935
Iteration 82, loss = 0.43454691
Iteration 83, loss = 0.43282636
Iteration 84, loss = 0.43116141
Iteration 85, loss = 0.42950173
Iteration 86, loss = 0.42788654
Iteration 87, loss = 0.42644460
Iteration 88, loss = 0.42495726
Iteration 89, loss = 0.42345326
Iteration 90, loss = 0.42197201

Iteration 91, loss = 0.42060049
Iteration 92, loss = 0.41921235
Iteration 93, loss = 0.41795171
Iteration 94, loss = 0.41657719
Iteration 95, loss = 0.41533300
Iteration 96, loss = 0.41403208
Iteration 97, loss = 0.41279109
Iteration 98, loss = 0.41166430
Iteration 99, loss = 0.41054847
Iteration 100, loss = 0.40932644