Всероссийский конкурс
научно-технологических проектов «Большие вызовы»

Направление: Большие данные, искусственный интеллект, финансовые технологии и машинное обучение

ОПРЕДЕЛЕНИЕ ТОНАЛЬНОСТИ КОММЕНТАРИЕВ С ПОМОЩЬЮ НЕЙРОННОЙ СЕТИ

Работу выполнил:
Д. А. Бобров

Наставник:
А. М. Шапенская
педагог доп. образования
ГАНОУ «РЦПД»

Брянск 2023

Нейронные сети – это метод в искусственном интеллекте, который учит компьютеры обрабатывать данные таким же способом, как и человеческий мозг. Они используются для различных задач, включая распознавание образов, прогнозирование и другие.

Цель: разработать программу для определения тональности текстов комментариев, которая занимается нахождением оценочных мнений в тексте и определением их свойств в отзывах.

Задачи:

- изучить литературу по данной теме;
- выбрать язык программирования;
- изучить библиотеки для создания нейронных сетей на выбранном языке;
- найти обучающие данные для программы;
- написать и протестировать нейронную сеть.

Актуальность: в данный момент в интернете много безграмотных и некультурных людей. Необходимо отчищать веб-площадки от их комментариев, чтобы коммуникации в интернете стали этичнее. Предприниматели (производители и продавцы) смогут отобрать отрицательные комментарии и изучить их более детально. Таким образом, они смогут узнать мнение аудитории о своем продукты и проработать проблемные места, улучшить свой продукт и сделать его более удобным для реальной целевой аудитории.

Новизна: «SentiStrength» — система, разработанная M. Thelwall, K. Buckley, G. Paltoglou и D. Cai. Результат выдается в виде двух оценок – оценка позитивной составляющей текста (по шкале от +1 до +5) и оценка негативной составляющей (по шкале от -1 до -5). Недостатки системы: система может быть сконфигурирована для русского языка, но реализованные алгоритмы не учитывают его специфику, в том числе русскую морфологию, что приводит к ряду проблем. Данный проект не имеет подобных проблем и предназначен для русского языка.

Проблема: огромное количество некультурных, оскорбительных и необоснованных комментариев в сети Интернет.

Объект исследования: нейронные сети.

Предмет исследования: тексты и комментарии, взятые с сайтов банков.

Методы исследования:

- Анализ – анализ, имеющихся данных. Изучение уже существующих ранее разработанных способов анализа тональности текста;

- Были изучены аналоги, проведено сравнение с другими нейронными сетями;
- Сравнение – проведено сравнение продукта с аналогами. При разработке не раз были сравнены библиотеки, фреймворки, различные способы написания нейронных сетей и их виды;
- Измерение – измерены параметры нейронной сети, ее эффективность;
- Тестирование – готовый продукт был протестирован на работоспособность.

Продукт: нейронная сеть, которая определяет тональность комментария и отображает “уверенность” в своем предсказании.

1. Общие сведения

Искусственные нейронные сети состоят из нескольких слоев нейронов, связанных друг с другом. Каждый слой принимает некоторые данные и производит выходные данные для следующего слоя. Последний слой производит окончательные выходные данные для задачи. Нейронные сети обучаются с помощью алгоритма обратного распространения ошибки. Этот алгоритм анализирует ошибки, которые происходят на каждом шаге алгоритма, и исправляет их, чтобы сеть могла достичь нужных результатов.

Для выполнения данной работы был выбран python в качестве языка программирования, потому что этот ЯП очень удобный и существует много библиотек для этого языка, с помощью которых можно создавать нейронные сети. Для написания кода будем использовать PyCharm.

Нейронная сеть написана с помощью библиотеки PyTorch. Изначально была использована библиотека Tensorflow, но она очень неудобна, плохо подходит для поставленной задачи и уже устарела. PyTorch – более новый инструмент. Эта библиотека более гибкая, удобная и лаконичная. В ней есть функции, которые очень удобны для определения тональности текстов, и нет ничего лишнего.

2. Обучающие данные

Обучающие данные - это набор данных, используемый для обучения нейронных сетей. Они могут быть изображениями, текстом, аудиофайлами или другими типами данных. Обучающие данные используются для построения модели нейронной сети, которая может предсказывать правильные ответы на новые данные. Они представляют собой набор входных данных, выходных данных и меток для обучения. Эти данные могут быть предоставлены в виде набора примеров или использоваться для генерации собственных примеров. Они могут быть использованы для различных задач, включая распознавание образов, прогнозирование и другие. Потенциальный заказчик предоставил всю необходимую информацию. В обучающей выборке содержится 16 000 отзывов. Для каждого комментария есть оценка (положительный он или отрицательный). Обучающие данные были разделены на обучающую(80% данных) и тестовую(20% данных) выборки. Нейронная сеть на основе этих данных оценила слова, содержащиеся в этих отзывах, и самостоятельно решила, насколько каждое слово положительно или отрицательно. Далее каждый входной комментарий разбивается на слова, и на основе тональности каждого слова оценивается тональность всего комментария.

Рис. 1 – Обучающие данные, предоставленные потенциальным заказчиком

3. Процесс выполнения

Запустим программную среду PyCharm т.к. это популярная среда для разработки приложений и она очень удобна в использовании. Для создания нейронной сети был выбран язык python, т.к. на нем написано большинство нейронных сетей, есть готовые библиотеки. Использовал библиотеку torch, потому что она идеально подходит для выполнения поставленной задачи.

В файле neural_network.py инициализируется класс Network, потомок класса torch.nn.Module. У этого класса есть три метода: __init__(конструктор класса), forward, init_hidden_state.

Этот код (Листинг 1) определяет класс “Network”, который является нейронной сетью с рекуррентными слоями LSTM. Конструктор класса принимает несколько параметров, таких как размер словаря (vocab_size), размер выходного слоя (output_size), размерность вектора вложения (embedding_dim), размер скрытого слоя (hidden_dim), количество слоев (number_of_layers) и вероятность отключения (drop).

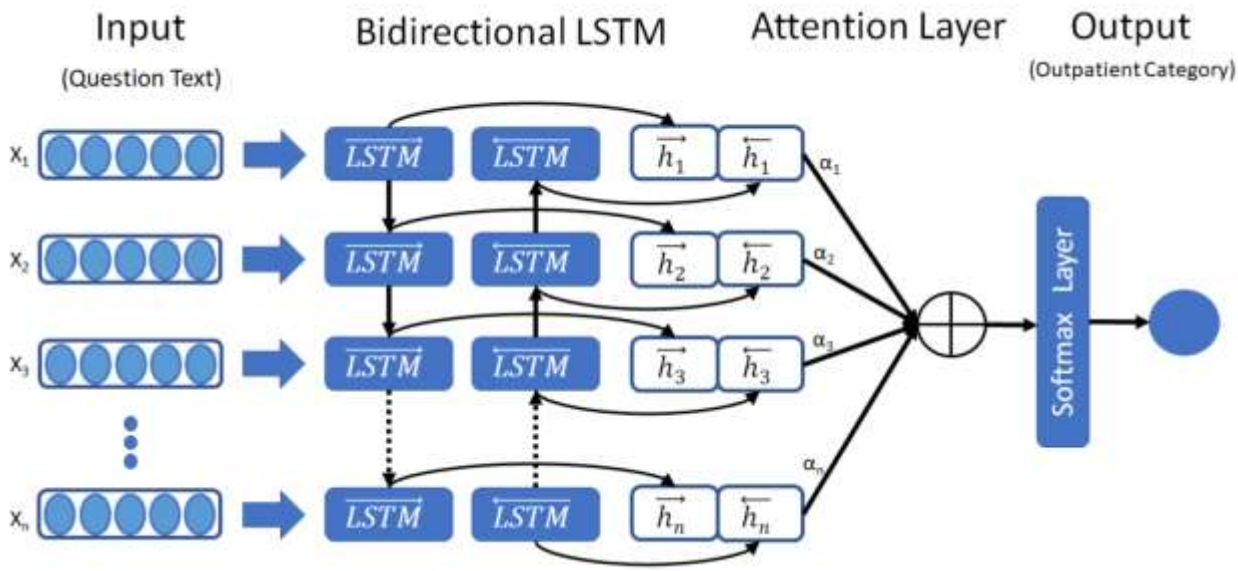

Рис. 2 – Строение рекуррентной нейронной сети (LSTM)

Внутри конструктора класса инициализируются несколько атрибутов, таких как размер выходного слоя, количество слоев, размер скрытого слоя, слой вложения (embedding), слой LSTM, слой dropout, полносвязный слой (fc) и функция активации сигмоид (sigmoid).

Слой вложения (embedding) преобразует входные данные (целочисленные значения) в векторы вложения. Слой LSTM обрабатывает векторы вложения и возвращает выходные данные и новое скрытое состояние. Слой dropout применяется для предотвращения переобучения. Полносвязный слой (fc) преобразует выходные данные LSTM в выходные данные с заданным размером. Функция активации сигмоид (sigmoid) применяется для получения вероятности положительного настроения (Листинг 1).

Листинг 1 – конструктор класса Network.

```
class Network(nn.Module):
    def __init__(self, vocab_size, output_size, embedding_dim,
hidden_dim, number_of_layers, drop=0.5):
        super(Network, self).__init__()
        self.output_size = output_size
        self.number_of_layers = number_of_layers
        self.hidden_dim = hidden_dim
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim,
number_of_layers, dropout=drop, batch_first=True)
        self.dropout = nn.Dropout(0.45)
        self.fc = nn.Linear(hidden_dim, output_size)
        self.sigmoid = nn.Sigmoid()
```

Данный код (Листинг 2) представляет собой метод “forward” класса, который является моделью нейронной сети. Он принимает на вход два аргумента: “x” – входные данные, и “hidden_state” – скрытое состояние LSTM слоя.

Внутри метода “forward” происходит следующее:

1. Входные данные “x” преобразуются в эмбеддинги с помощью метода “embedding”.
2. Полученные эмбеддинги и скрытое состояние “hidden_state” передаются в LSTM слой, который возвращает выход “lstm_out” и новое скрытое состояние “hidden_state”.
3. Выход “lstm_out” преобразуется в двумерный тензор с помощью метода “contiguous” и “view”.
4. Преобразованный выход “lstm_out” подвергается дропауту с помощью метода “dropout”.
5. Полученный результат передается в полносвязный слой “fc”.
6. Выход “fc” преобразуется с помощью функции активации “sigmoid”.
7. Размерность выхода “sig_out” изменяется с помощью метода “view”.
8. Измененный выход “sig_out” обрезается таким образом, чтобы получить только последний элемент каждой последовательности в батче.
9. Метод возвращает “sig_out” и новое скрытое состояние “hidden_state”.

(Листинг 2)

Листинг 2 – метод forward класса Network.

```
    def forward(self, x, hidden_state):
        lstm_out, hidden_state = self.lstm(self.embedding(x.long()),
hidden_state)
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)
        out = self.dropout(lstm_out)
        out = self.fc(out)
        sig_out = self.sigmoid(out)
        batch_size = x.size(0)
        sig_out = sig_out.view(batch_size, -1)
        sig_out = sig_out[:, -1]
        return sig_out, hidden_state
```

Этот код (Листинг 3) определяет метод “init_hidden_state” для класса, представляющего модель нейронной сети. Этот метод инициализирует скрытое состояние модели перед началом обучения или предсказания.

Шаги, выполняемые этим методом:

1. Получение весов модели с помощью метода “next(self.parameters()).data”. Этот метод возвращает следующий параметр модели, который является тензором весов.
2. Создание скрытого состояния с помощью метода “new” тензора весов. Этот метод создает новый тензор с теми же размерами и типом данных, что и тензор весов, но заполненный нулями. Скрытое состояние состоит из двух тензоров: первый тензор

представляет скрытое состояние LSTM, а второй тензор представляет состояние ячейки LSTM.

3. Возврат скрытого состояния в качестве результата метода.

В файле model_utils.py мы импортируем нужные библиотеки, выполняем код предобработки данных и инициализируем функции tokenize_text, add_pads и predict. В основном файле мы будем использовать функцию predict, остальные функции нужны для ее корректной работы (Листинг 3).

Листинг 3 – метод `init_hidden_state`.

```
    def init_hidden_state(self, batch_size):
        weight = next(self.parameters()).data
        hidden_state = (weight.new(self.number_of_layers, batch_size,
self.hidden_dim).zero_(),
                        weight.new(self.number_of_layers, batch_size,
self.hidden_dim).zero_())
        return hidden_state
```

Этот код (Листинг 4) выполняет следующие действия:

1. Определение списка “n”, содержащего названия столбцов в CSV-файле.

2. Чтение положительных и отрицательных текстовых данных из CSV-файлов “positive.csv” и “negative.csv” с помощью метода “read_csv” библиотеки Pandas. В качестве разделителя используется символ “;”, а столбцы “id”, “date”, “name”, “text”, “typr”, “rep”, “rtw”, “faw” “stcount”, “foll”, “frien”, “listcount” не используются.

3. Определение размера выборки “sample_size” и объединение положительных и отрицательных текстовых данных в один массив “texts_without_shuffle”.

4. Создание меток для текстовых данных в массиве “labels_without_shuffle”. Метка “1” соответствует положительным текстам, а метка “0” – отрицательным.

5. Перемешивание текстовых данных и меток с помощью функции “shuffle” из библиотеки “sklearn.utils”. Результаты перемешивания сохраняются в массивах “texts” и “labels”.

6. Удаление знаков препинания из текстовых данных и приведение всех символов к нижнему регистру.

7. Разделение текстовых данных на отдельные слова и подсчет количества каждого слова с помощью класса “Counter” из стандартной библиотеки Python.

8. Сортировка слов по убыванию частоты их встречаемости в текстовых данных.

9. Создание словаря “vocab_to_int”, который сопоставляет каждому слову в словаре уникальный целочисленный идентификатор. Идентификаторы начинаются с 1, а не с 0, чтобы можно было использовать 0 в качестве заполнителя (padding) при обучении модели.

Листинг 4.

```
n = ['id', 'date', 'name', 'text', 'typr', 'rep', 'rtw', 'faw',
'stcount', 'foll', 'frien', 'listcount']
data_positive = pd.read_csv('./dataset/positive.csv', sep=';',
error_bad_lines=False, names=n, usecols=['text'])
data_negative = pd.read_csv('./dataset/negative.csv', sep=';',
error_bad_lines=False, names=n, usecols=['text'])

sample_size = 50000
texts_without_shuffle =
np.concatenate((data_positive['text'].values[:sample_size],

data_negative['text'].values[:sample_size]), axis=0)
labels_without_shuffle = np.asarray([1] * sample_size + [0] *
sample_size)
assert len(texts_without_shuffle) == len(labels_without_shuffle)

texts = shuffle(texts_without_shuffle, labels_without_shuffle,
random_state=0)[0]
punctuation = r"!\"#$%&'()*+,-./:;<=>?[\]^_»{|}~"
all_texts = 'separator'.join(texts)
all_texts = all_texts.lower()
all_text = ''.join([c for c in all_texts if c not in punctuation])
texts_split = all_text.split('separator')
all_text = ' '.join(texts_split)
words = all_text.split()
counts = Counter(words)
vocab = sorted(counts, key=counts.get, reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab, 1)}
```

Эта функция (Листинг 5) выполняет токенизацию текста, то есть разбивает текст на отдельные слова и преобразует их в числовой формат. Рассмотрим шаги, которые выполняет данный код:

1. Приводит текст к нижнему регистру с помощью метода “lower()”.

2. Удаляет все знаки препинания из текста с помощью глобальной переменной “punctuation” и метода “join()” и “not in”.

3. Разбивает текст на отдельные слова с помощью метода “split()”.

4. Создает новый список “new_text”, в который добавляет только те слова, которые не начинаются с символа "@" и не содержат подстроку "http" и не являются числом.

5. Преобразует каждое слово из списка “new_text” в соответствующий ему числовой идентификатор, используя словарь “vocab_to_int”.

6. Возвращает список числовых идентификаторов слов в виде списка списков, где каждый вложенный список содержит идентификаторы слов для одного текста.

Например, если на вход функции “tokenize_text()” подать строку "Это примерное предложение", то на выходе мы получим список списков [[14, 6, 3]], где каждый числовой идентификатор соответствует одному слову из исходной строки.

Листинг 5 – функция tokenize_text.

```
def tokenize_text(text):
    global punctuation
    text = text.lower()
    text = ''.join([c for c in text if c not in punctuation])
    test_words = text.split()
    new_text = []
    for word in test_words:
        if (word[0] != "@") & ("http" not in word) & (~word.isdigit()):
            new_text.append(word)
    mas_to_int = [vocab_to_int[word] for word in new_text if word in
vocab_to_int]
    return [mas_to_int]
```

Функция “add_pads” (Листинг 6) выполняет добавление паддингов к последовательностям числовых идентификаторов слов, чтобы все последовательности имели одинаковую длину. Рассмотрим шаги, которые выполняет данный код:

1. Создает нулевой массив “features” размера “(len(texts_ints), sequence_length)”, где “len(texts_ints)” – количество текстов, “sequence_length” – максимальная длина последовательности.

2. Проходит по каждому тексту из списка “texts_ints” и добавляет его числовые идентификаторы в массив “features”. Если длина последовательности превышает “sequence_length”, то обрезает ее до “sequence_length”. Если длина последовательности меньше “sequence_length”, то добавляет нулевые элементы в начало массива “features”, чтобы длина последовательности стала равной “sequence_length”.

3. Возвращает массив “features”, содержащий последовательности числовых идентификаторов слов с добавленными паддингами.

Например, если на вход функции “add_pads()” подать список списков [[14, 6, 3, 15], [2, 4, 6], [8, 10, 12, 14, 16, 18]], а “sequence_length” равно 6, то на выходе мы получим массив размера “(3, 6)”:

```
array([
    [14, 6, 3, 15, 0, 0],
    [ 0, 0, 2, 4, 6, 0],
    [ 8, 10, 12, 14, 16, 18],
  ])
```

где нулевые элементы добавлены в начало первой и второй последовательности, чтобы их длина стала равной “sequence_length”.

Листинг 6.

```
def add_pads(texts_ints, sequence_length):
    features = np.zeros((len(texts_ints), sequence_length), dtype=int)

    for i, row in enumerate(texts_ints):
        if len(row):
            features[i, -len(row):] = np.array(row)[:sequence_length]

    return features
```

Функция “predict” (Листинг 7) принимает на вход нейронную сеть “net”, текст “text” и необязательный параметр “sequence_length”, который по умолчанию равен 30.

Рассмотрим шаги, выполняемые этим кодом:

1. “net.eval()” – переводит нейронную сеть в режим оценки (evaluation mode), что означает, что сеть не будет обучаться и не будет изменять свои веса.
2. “features = add_pads(tokenize_text(text), sequence_length)” – вызывает функцию “tokenize_text”, которая токенизирует текст, то есть разбивает его на отдельные слова и преобразует их в числовые идентификаторы. Затем вызывается функция “add_pads”, которая добавляет заполнение (padding) к последовательности числовых идентификаторов, чтобы все последовательности имели одинаковую длину. Результат сохраняется в переменной “features”.
3. “tensor = torch.from_numpy(features)” – преобразует массив “features” в тензор PyTorch.
4. “batch_size = tensor.size(0)” – определяет размер пакета (batch size) как размер первого измерения тензора “tensor”.
5. “output = net(tensor, net.init_hidden_state(batch_size))[0]” – передает тензор “tensor” в нейронную сеть “net”, чтобы получить выходные данные. “net.init_hidden_state(batch_size)” инициализирует скрытое состояние нейронной сети, которое передается вместе с входными данными. “[0]” используется для извлечения только первого элемента из выходных данных.
6. “prediction = torch.round(output.squeeze())” – округляет выходные данные до ближайшего целого числа и сохраняет результат в переменной “prediction”.
7. “positive_probability = output.item()” – извлекает вероятность положительного класса из выходных данных и сохраняет результат в переменной “positive_probability”.
8. “if prediction.item() == 1:” – проверяет, является ли предсказание положительным классом.
9. “result = "Негативное сообщение"” – если предсказание является отрицательным классом, то результатом будет строка "Негативное сообщение".

10. “result = "Позитивное сообщение"” – если предсказание является положительным классом, то результатом будет строка "Позитивное сообщение".

11. “return result, positive_probability” – возвращает результат предсказания и вероятность положительного класса.

Листинг 7 – функция predict.

```
def predict(net, text, sequence_length=30):
    net.eval()
    features = add_pads(tokenize_text(text), sequence_length)
    tensor = torch.from_numpy(features)

    batch_size = tensor.size(0)
    output = net(tensor, net.init_hidden_state(batch_size))[0]

    prediction = torch.round(output.squeeze())
    positive_probability = output.item()

    if prediction.item() == 1:
        result = "Негативное сообщение"
    else:
        result = "Позитивное сообщение"

    return result, positive_probability
```

Файл app.py – это главный файл проекта, он запускает выполнение нейронной сети, используя для этого остальные файлы.

Код (Листинг 8) включает в себя Flask веб-приложение, которое позволяет пользователю отправлять текстовые комментарии и получать результат анализа тональности комментария из браузера, используя красивый и удобный интерфейс.

Здесь мы импортируем необходимые библиотеки и устанавливаем параметры для модели нейронной сети. В частности, мы загружаем модель, создаем экземпляр класса Network, который представляет нейронную сеть, создаем словарь размером VOCAB_SIZE, определяем размерность вектора EMBEDDING_DIM, размерность скрытого слоя HIDDEN_DIM и количество слоев LAYERS_NUMBER.

Затем мы создаем маршрут приложения Flask, который вызывается при обращении к корневому каталогу. Если метод запроса POST, то мы получаем текст из формы, передаем его в функцию predict, чтобы получить результат анализа тональности комментария и уверенность нейронной сети в предсказании.

Если метод запроса GET, то мы просто отображаем главную страницу приложения.

В конце мы запускаем приложение Flask.

Функция predict, которая вызывается внутри маршрута Flask, выполняет следующие действия:

1. Выполняется forward pass нейронной сети.

2. Текст токенизируется, и каждому слову сопоставляется уникальный идентификатор.

3. Создается последовательность числовых идентификаторов и добавляется заполнение, чтобы все последовательности имели одинаковую длину.

4. Нейронная сеть используется для предсказания того, является ли комментарий позитивным или негативным, и возвращается вероятность, что комментарий является позитивным.

Листинг 8 – файл app.py.

```
from math import ceil

import torch
from flask import Flask, render_template, request

from neural_network import Network
from model_utils import predict

DEBUG = True
app = Flask(__name__)
app.config.from_object(__name__)
app.config["SECRET_KEY"] = "anything"

VOCAB_SIZE = 164192
OUTPUT_SIZE = 1
EMBEDDING_DIM = 100
HIDDEN_DIM = 128
LAYERS_NUMBER = 2
model = Network(VOCAB_SIZE, OUTPUT_SIZE, EMBEDDING_DIM, HIDDEN_DIM,
LAYERS_NUMBER)
model.load_state_dict(torch.load("model",
map_location=torch.device("cpu")))
sequence_length = 30


@app.route("/", methods=["GET", "POST"])
def main():
    if request.method == "POST":
        text = request.form["text"]
        result, positive_probability = predict(model, text,
sequence_length)
        if result == "Негативный комментарий":
            percent = ceil((1 - positive_probability) * 100)
        else:
            percent = ceil(positive_probability * 100)

        return render_template("main.html", result=result,
percent=percent, text=text)
    return render_template("main.html")


if __name__ == "__main__":
    app.run()
```

4. Тестирование

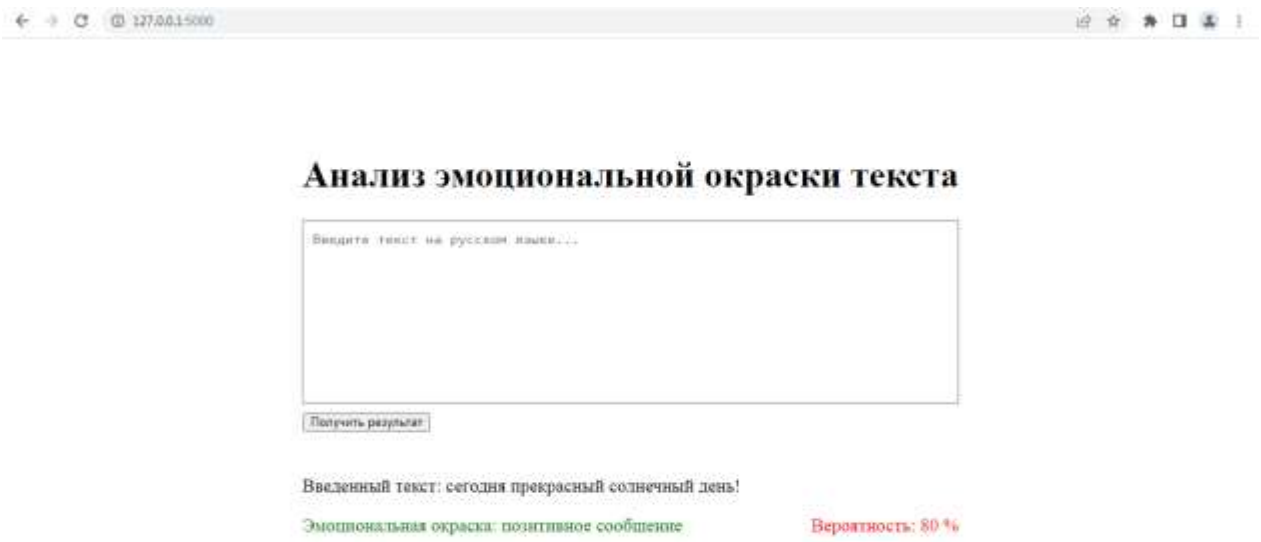

Рис. 3 – Тестирование нейронной сети

Вывод: нейронная сеть успешно распознает положительные комментарии

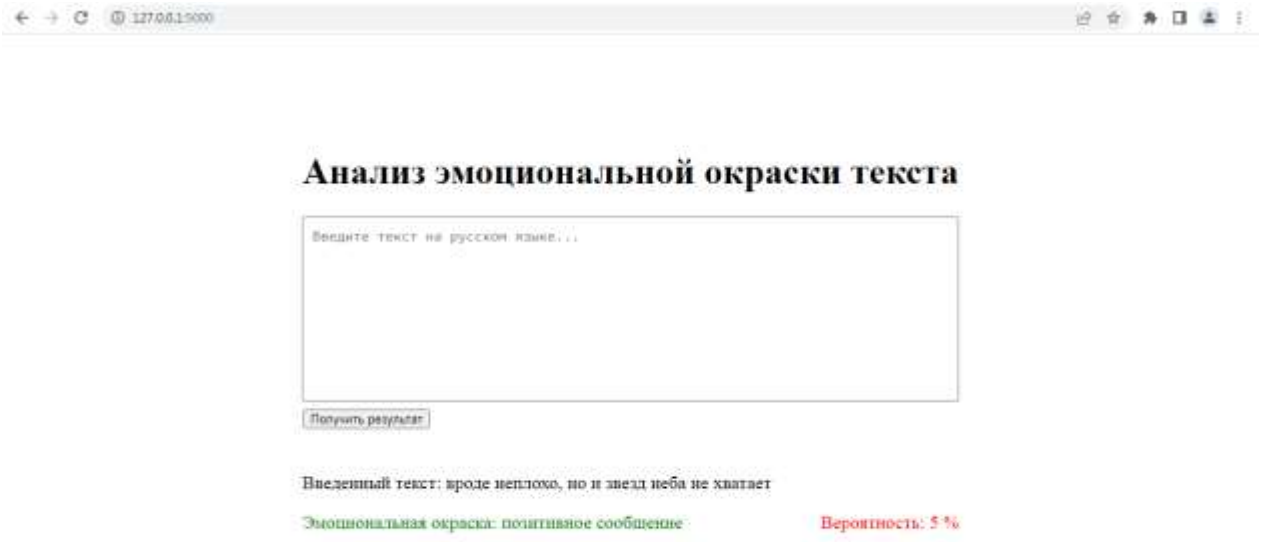

Рис. 4 – Тестирование нейронной сети

Вывод: нейронная сеть успешно распознает нейтральные комментарии

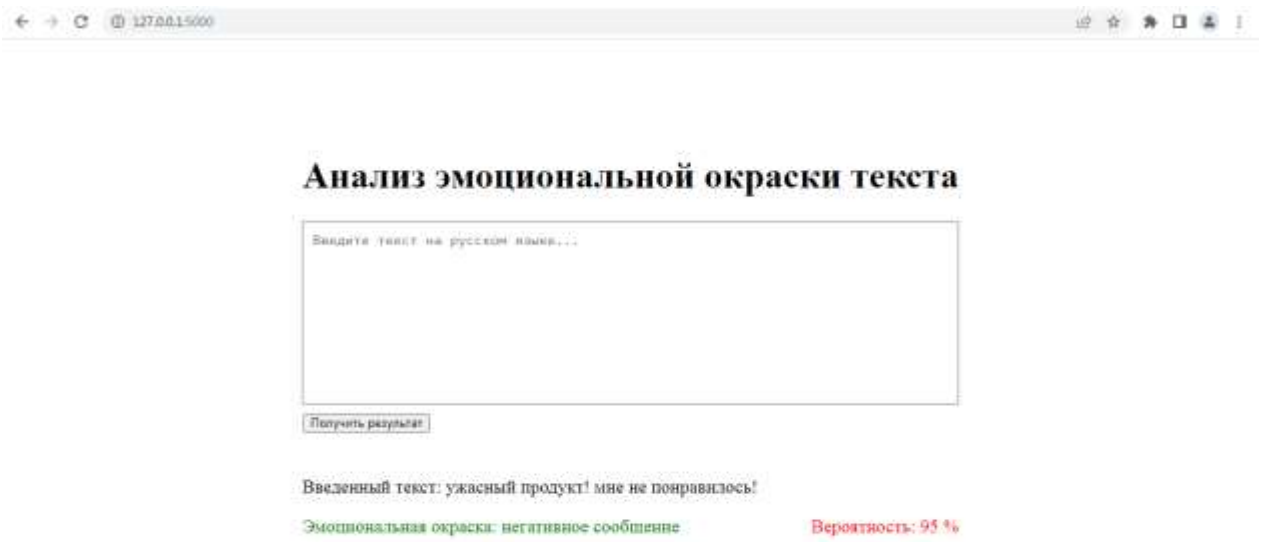

Рис. 5 – Тестирование нейронной сети

Вывод: нейронная сеть успешно распознает отрицательные комментарии

Перспективы развития:

1) определение сарказма и иронии – даже люди не всегда могут распознать сарказм и иронию, а если это не сделать, то можно неправильно распознать тональность комментария;

2) определение темы – о чем говорится во мнении;

3) определение автора комментария – боты могут манипулировать мнением людей, создавать ложные впечатления.

Заключение

В результате работы над проектом была разработана программа, которая определяет тональность комментария. Нейронная сеть может иногда ошибаться, но в таком случае процент вероятности будет маленьким, т.е. нейронная сеть будет не уверена в предсказании. Таким образом, можно в автоматическом режиме отсеивать очень позитивные либо очень негативные комментарии, а остальные обрабатывать уже вручную.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Анализ тональности в русскоязычных текстах, часть 1: введение [Электронный ресурс] – Режим доступа: https://habr.com/ru/company/vk/blog/516214/ (Дата обращения: 15.12.22)
2. Обучаем компьютер чувствам (sentiment analysis по-русски) [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/149605/ (Дата обращения: 13.01.23)
3. Анализ тональности текстов с помощью сверточных нейронных сетей [Электронный ресурс] – Режим доступа: https://habr.com/ru/company/vk/blog/417767/ (Дата обращения: 28.01.23)
4. Сентимент-анализ. Определяем эмоциональные сообщения на Хабре [Электронный ресурс] – Режим доступа: https://habr.com/ru/company/simbirsoft/blog/539508/ (Дата обращения: 09.01.23)
5. Анализ тональности в русскоязычных текстах [Электронный ресурс] – Режим доступа: https://habr.com/ru/company/vk/blog/516730/ (Дата обращения: 09.01.23)
6. Что такое нейронная сеть? [Электронный ресурс] – Режим доступа: https://aws.amazon.com/ru/what-is/neural-network/ (Дата обращения: 09.01.23)
7. «Люблю» и «ненавижу»: анализ эмоциональной окраски текста с помощью Python [Электронный ресурс] – Режим доступа: https://proglib.io/p/lyublyu-i-nenavizhu-analiz-emocionalnoy-okraski-teksta-s-pomoshchyu-python-2020-11-13 (Дата обращения: 09.01.23)
8. Нейронные сети для начинающих [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/312450/ (Дата обращения: 09.01.23)
9. Создаем прототип для Sentiment Analysis с помощью Python и TextBlob [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/457168/ (Дата обращения: 09.01.23)
10. Sentiment Analysis: A Definitive Guide [Электронный ресурс] – Режим доступа:https://monkeylearn.com/sentiment_analysis/#:~:text=Sentiment%20analysis%20(or%20opinion%20mining,feedback%2C%20and%20understand%20customer%20needs

(Дата обращения: 09.01.23)
11. Sentiment analysis (opinion mining) [Электронный ресурс] – Режим доступа: https://www.techtarget.com/searchbusinessanalytics/definition/opinion-mining-sentiment-mining (Дата обращения: 09.01.23)
12. What is Sentiment Analysis? An Ultimate Guide for 2023 [Электронный ресурс] – Режим доступа: https://brand24.com/blog/sentiment-analysis/ (Дата обращения: 09.01.23)

13. Sentiment Analysis Explained [Электронный ресурс] – Режим доступа: https://www.lexalytics.com/technology/sentiment-analysis/ (Дата обращения: 09.01.23)

ПРИЛОЖЕНИЕ

Листинг 1 – app.py.

```
from math import ceil

import torch
from flask import Flask, render_template, request

from neural_network import Network
from model_utils import predict

DEBUG = True
app = Flask(__name__)
app.config.from_object(__name__)
app.config["SECRET_KEY"] = "anything"

VOCAB_SIZE = 164192
OUTPUT_SIZE = 1
EMBEDDING_DIM = 100
HIDDEN_DIM = 128
LAYERS_NUMBER = 2
model = Network(VOCAB_SIZE, OUTPUT_SIZE, EMBEDDING_DIM, HIDDEN_DIM,
LAYERS_NUMBER)
model.load_state_dict(torch.load("model",
map_location=torch.device("cpu")))
sequence_length = 30


@app.route("/", methods=["GET", "POST"])
def main():
    if request.method == "POST":
        text = request.form["text"]
        result, positive_probability = predict(model, text,
sequence_length)
        if result == "PќPµPiP°C,PëPIPSPsPµ CЃPsPsP±C‰PµPSPëPµ":
            percent = ceil((1 - positive_probability) * 100)
        else:
            percent = ceil(positive_probability * 100)

        return render_template("main.html", result=result,
percent=percent, text=text)
    return render_template("main.html")


if __name__ == "__main__":
    app.run()
```

Листинг 2 – model_utils.py.

```
from collections import Counter

import numpy as np
import pandas as pd
import torch
from sklearn.utils import shuffle

n = ['id', 'date', 'name', 'text', 'typr', 'rep', 'rtw', 'faw',
'stcount', 'foll', 'frien', 'listcount']
```

```
data_positive = pd.read_csv('./dataset/positive.csv', sep=';',
error_bad_lines=False, names=n, usecols=['text'])
data_negative = pd.read_csv('./dataset/negative.csv', sep=';',
error_bad_lines=False, names=n, usecols=['text'])

sample_size = 50000
texts_without_shuffle =
np.concatenate((data_positive['text'].values[:sample_size],

data_negative['text'].values[:sample_size]), axis=0)
labels_without_shuffle = np.asarray([1] * sample_size + [0] *
sample_size)
assert len(texts_without_shuffle) == len(labels_without_shuffle)

texts = shuffle(texts_without_shuffle, labels_without_shuffle,
random_state=0)[0]
punctuation = r"!\"#$%&'()*+,-./:;<=>?[\]^_`{|}~"
all_texts = 'separator'.join(texts)
all_texts = all_texts.lower()
all_text = ''.join([c for c in all_texts if c not in punctuation])
texts_split = all_text.split('separator')
all_text = ' '.join(texts_split)
words = all_text.split()
counts = Counter(words)
vocab = sorted(counts, key=counts.get, reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab, 1)}


def tokenize_text(text):
    global punctuation
    text = text.lower()
    text = ''.join([c for c in text if c not in punctuation])
    test_words = text.split()
    new_text = []
    for word in test_words:
        if (word[0] != "@") & ("http" not in word) & (~word.isdigit()):
            new_text.append(word)
    mas_to_int = [vocab_to_int[word] for word in new_text if word in
vocab_to_int]
    return [mas_to_int]


def add_pads(texts_ints, sequence_length):
    features = np.zeros((len(texts_ints), sequence_length), dtype=int)

    for i, row in enumerate(texts_ints):
        if len(row):
            features[i, -len(row):] = np.array(row)[:sequence_length]

    return features


def predict(net, text, sequence_length=30):
    net.eval()
    features = add_pads(tokenize_text(text), sequence_length)
    tensor = torch.from_numpy(features)

    batch_size = tensor.size(0)
```

```
    output = net(tensor, net.init_hidden_state(batch_size))[0]

    prediction = torch.round(output.squeeze())
    positive_probability = output.item()

    if prediction.item() == 1:
        result = "PќPµPiP°C,PëPIPSPsPµ CЃPsPsP±C‰PµPSPëPµ"
    else:
        result = "PµPsP·PëC,PëPIPSPsPµ CЃPsPsP±C‰PµPSPëPµ"

    return result, positive_probability
```

Листинг 3 – neural_network.py.

```
import torch.nn as nn


class Network(nn.Module):
    def __init__(self, vocab_size, output_size, embedding_dim,
hidden_dim, number_of_layers, drop=0.5):
        super(Network, self).__init__()
        self.output_size = output_size
        self.number_of_layers = number_of_layers
        self.hidden_dim = hidden_dim
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim,
number_of_layers, dropout=drop, batch_first=True)
        self.dropout = nn.Dropout(0.45)
        self.fc = nn.Linear(hidden_dim, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x, hidden_state):
        lstm_out, hidden_state = self.lstm(self.embedding(x.long()),
hidden_state)
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)
        out = self.dropout(lstm_out)
        out = self.fc(out)
        sig_out = self.sigmoid(out)
        batch_size = x.size(0)
        sig_out = sig_out.view(batch_size, -1)
        sig_out = sig_out[:, -1]
        return sig_out, hidden_state

    def init_hidden_state(self, batch_size):
        weight = next(self.parameters()).data
        hidden_state = (weight.new(self.number_of_layers, batch_size,
self.hidden_dim).zero_(),
                        weight.new(self.number_of_layers, batch_size,
self.hidden_dim).zero_())
        return hidden_state
```