МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Самарский национальный исследовательский университет

имени академика С.П. Королева»

Институт информатики и кибернетики

Факультет информатики

Кафедра программных систем

ОТЧЁТ

по лабораторной работе № 3

«Анализ тональности (sentiment analysis)»

по курсу «Нейронные сети»

Выполнил:

Портнов К.А. Хорина В.И.

гр. 6132-020402D

Проверила:

Жданова А.Н.

Самара 2022

Оглавление

[Математическая постановка задачи 3](#_Toc122621695)

[Обоснование выбора нейронной сети 3](#_Toc122621696)

[Описание архитектуры сети, параметров, метода обучения 4](#_Toc122621697)

[Структурная схема алгоритма 7](#_Toc122621698)

[Описание вычислительных экспериментов с целью оценки эффективности модели 8](#_Toc122621699)

[Результаты работы 10](#_Toc122621700)

[Заключение 10](#_Toc122621701)

[Ссылки 10](#_Toc122621702)

[Листинг программы 11](#_Toc122621703)

# Математическая постановка задачи

Требуется реализовать нейронную сеть, позволяющую определить по тексту его тональность, то есть позитивное ли отношение несет этот текст или негативное.

Набор данных содержит 226834 строчек данных взятых и основан на сообщениях российского сегмента twitter.

Описание данных:

1. id: уникальный номер сообщения в системе twitter;
2. tdate: дата публикации сообщения (твита);
3. tmane: имя пользователя, опубликовавшего сообщение;
4. ttext: текст сообщения (твита);
5. ttype: поле в котором в дальнейшем будет указано к кому классу относится твит (положительный, отрицательный, нейтральный);
6. trep: количество реплаев к данному сообщению. В настоящий момент API твиттера не отдает эту информацию;
7. tfav: число сколько раз данное сообщение было добавлено в избранное другими пользователями;
8. tstcount: число всех сообщений пользователя в сети twitter;
9. tfol: количество фоловеров пользователя (тех людей, которые читают пользователя);
10. tfrien: количество друзей пользователя (те люди, которых читает пользователь);
11. listcount: количество листов-подписок в которые добавлен твиттер-пользователь.

# Обоснование выбора нейронной сети

Обычно другие типы нейронных сетей, такие как плотно связанные сети или сверточные сети, не имеют памяти, это означает, что каждый вход обрабатывается независимо, не имея никакого отношения к другим. Это противоположно тому, что люди обычно делают при чтении текста, а именно сохранение в памяти того, что было прочитано в предыдущих строках. В следствии чего появляется ощущение всего смысла теста, и это точно такой же принцип, который принимают RNN. RNN обрабатывают последовательности, итерируя вдоль элементов последовательности и сохраняя информацию относительно того, что она обрабатывала до сих пор.

Рекуррентные нейронные сети (RNN) - это класс нейронных сетей, который является мощным для моделирования последовательности данных, таких как временные ряды или естественный язык.

Схематически уровень RNN использует for цикл для перебора временных шагов последовательности, сохраняя при этом внутреннее состояние, которое кодирует информацию о временных шагах, которые он видел до сих пор.

# Описание архитектуры сети, параметров, метода обучения

Для выполнения работы использованы:

API Keras RNN для создания и обучения сети.

API Keras RNN разработан с акцентом на:

* Простота использования;
* Простота настройки.

Nltk – платформа для работы с данными на человеческом языке с набором библиотек обработки текста для классификации, токенизации, выделения, тегов, синтаксического анализа и т.д.

Gensim – библиотека для представления документов в виде семантических векторов

Pandas – для работы с dataFrame и т.д.

И прочие библиотеки python.

Была реализована рекуррентная нейронная сеть типа двунаправленная LSTM.

Слой встраивания – Embedding с размерность и количеством токенов.

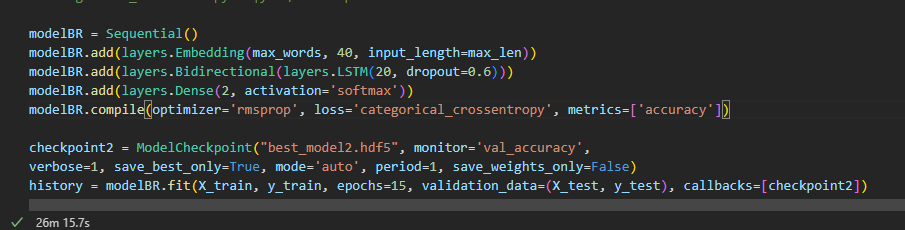
Двунаправленный слой с LSTM с 20 скрытыми единицами внутри слоя и 0,6 в качестве Dropout для регуляризации сети.

Выходной слой – Densy с 2 выходам (2 категории) и softmax в качестве функции активации.

В качестве оптимизатора использован алгоритм RMSprop и categorical\_crossentropy как функция потерь.

Обучение проводилось на 15 эпохах с сохранением состояния.

Сама модель выглядит следующим образом:



И лог обучения:

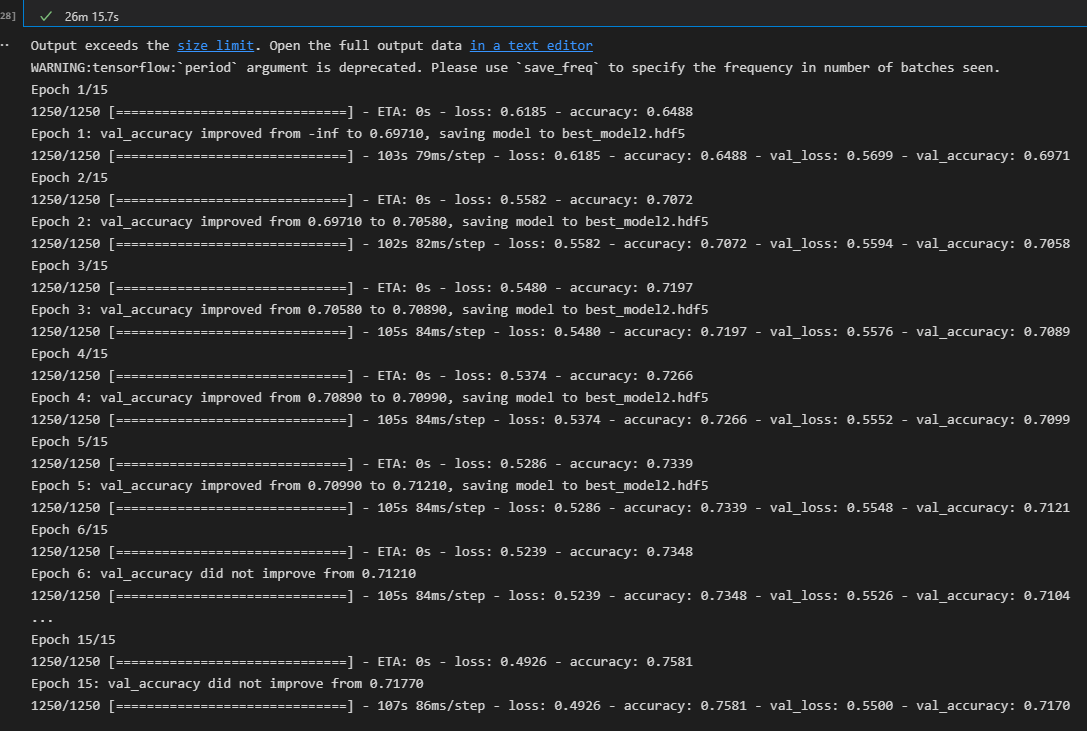


График точности при обучении:

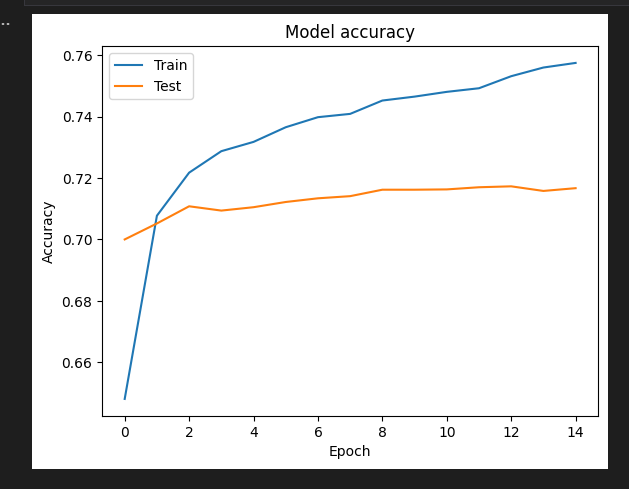
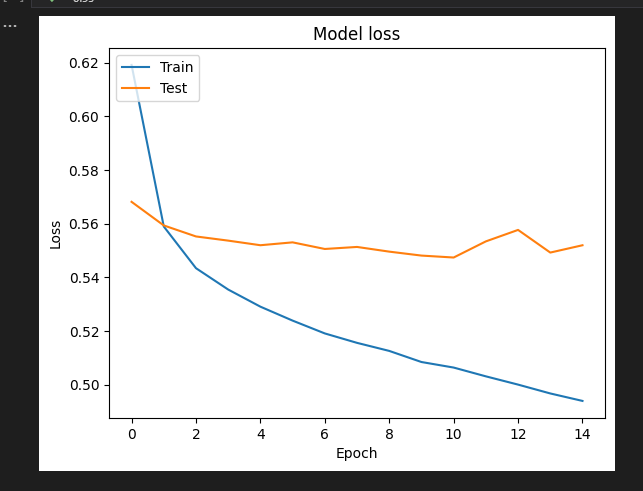


График потерь при обучении:



# Структурная схема алгоритма

Выполнение работы начинается с импортирования всех необходимых библиотек в проект.

Импортируем датасет и превращаем его в DataFrame.

Для более быстрого обучения сети, обрезаем выборку по 25000 строк каждой категории сообщений.

Т.к. в выборке гораздо больше полей, чем нам нужно в работе, обрезаем только по необходимым полям.

Все пустые сообщения заменяем на заглушку “пусто”.

В цикле проводим очистку текста:

* Удаляем URL
* Удаляем Email
* Удаляем символы новой строки
* Удаляем одинарные кавычки

Преобразуем строки в список токенов.

Удаляем повторяющиеся слова из набора данных, следуя идее, что самые короткие и длинные обычно бесполезны.

Производим детокенизацию всех предложений – превращаем обратно в строки.

Создаём список лэйблов, состоящий из преобразованных признаков тональности (0 –негативный, 1- позитивный).

Преобразовываем текстовые данные в двумерные данные с плавающей запятой.

Разбиваем набор данных на обучающую и тестовую выборку в соотношении 4:1.

Создаём и обучаем сеть с сохранением состояния.

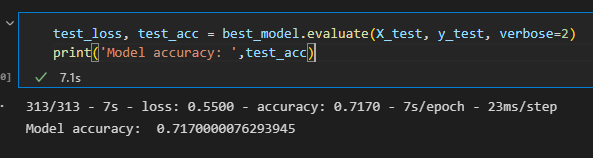
Строим графики точности и потерь при обучении.

Запускаем сеть на тестовом наборе данных.

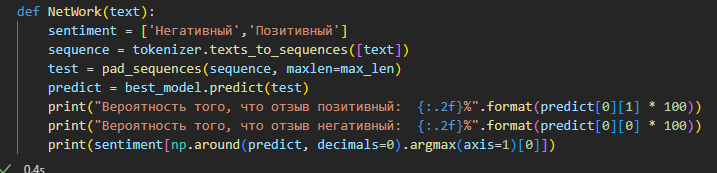
Проводим ручные проверки работы, обученной сети.

# Описание вычислительных экспериментов с целью оценки эффективности модели

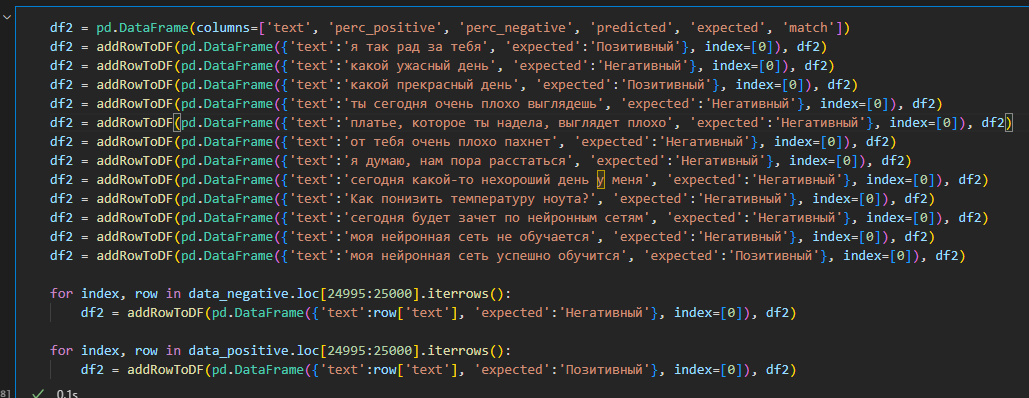
Результаты проверки обученной сети на тестовых данных представленный ниже.



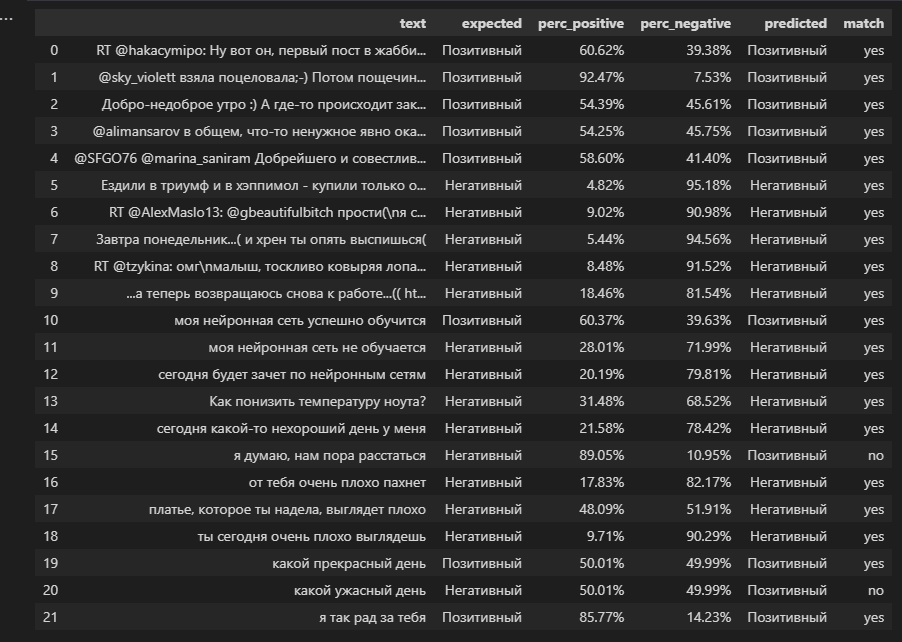
Для проверки и вывода результата прогноза сети реализована функция:



Для проверки работы нейронной сети был подготовлен дата фрейм с вручную заданными фразами и ожидаемым результатом, так же были включены некоторые фразы из обучаемого набора данных.



Результат:



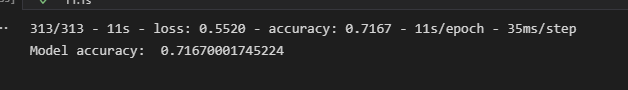
# Результаты работы

Таким образом были разработаны рекуррентная нейронная сеть типа двунаправленная LSTM.

Набора данных, состоящих их 50000 записей и 15 эпох обучения было достаточно для получения точности в 0,7575 и потерь 0,49.

Разработанная сеть хорошо себя показала на пользовательской выборке, предсказав верно в 90,9%.

Результаты проверки на тестовой выборке оказались несколько хуже, чем на пользовательской



# Заключение

В ходе выполнения лабораторной работы была создана и обучена рекуррентная нейронная сеть, позволяющая определить по тексту его тональность, то есть позитивное ли отношение несет этот текст или негативное.

# Ссылки

Датасет: <http://study.mokoron.com/>

GitHub: <https://github.com/PKkDev/nn_lab3>

# Листинг программы

import pandas as pd

from pandas import DataFrame

import matplotlib.pyplot as plt

import nltk

from nltk.tokenize.treebank import TreebankWordDetokenizer

import numpy as np

import re

import gensim

import keras

from keras.models import Sequential

from keras import layers

from keras.optimizers import RMSprop,Adam

from keras.preprocessing.text import Tokenizer

from keras.utils import pad\_sequences

from keras import regularizers

from keras import backend as K

from keras.callbacks import ModelCheckpoint

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

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

data\_positive = pd.read\_csv('datasets/positive.csv', sep=';', names=n)

data\_negative = pd.read\_csv('datasets/negative.csv', sep=';', names=n)

data\_negative = data\_negative[0:25000]

print(data\_negative.shape[0])

data\_negative.head()

data\_positive = data\_positive[0:25000]

print(data\_positive.shape[0])

data\_positive.head()

data\_total = pd.concat([data\_negative,data\_positive])

data\_total.head()

# длина датасета

len(data\_total)

# есть ли какое-либо другое значение, кроме отрицательного и положительного?

data\_total['typr'].unique()

# распределение набора данных

data\_total.groupby('typr').nunique()

# оставляем только те столбцы, которые будут использоваться

data\_total = data\_total[['text','typr']]

data\_total.head()

# есть ли нулевое значение?

data\_total["text"].isnull().sum()

# заполняем нулевые значения

data\_total["text"].fillna("пусто", inplace = True)

# очистки текста

def depure\_data(data):

# удаляем URL

url\_pattern = re.compile(r'https?://\S+|www\.\S+')

data = url\_pattern.sub(r'', data)

# удаляем Email

data = re.sub('\S\*@\S\*\s?', '', data)

# удалем симолы новой строки

data = re.sub('\s+', ' ', data)

# удаляем одинарные ковычки

data = re.sub("\'", "", data)

# приведение к нижнему регистру + замена «ё» на «е»

# data = data.lower().replace("ё", "е")

# замена ссылок на токен «URL»

# data = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', 'URL', data)

# замена упоминания пользователя на токен «USER»

# data = re.sub('@[^\s]+', 'USER', data)

# удаление знаков пунктуации

# data = re.sub('[^a-zA-Zа-яА-Я1-9]+', ' ', data)

# data = re.sub(' +', ' ', data)

return data

temp = []

# разделение pd.Series на список

data\_to\_list = data\_total['text'].values.tolist()

for i in range(len(data\_to\_list)):

temp.append(depure\_data(data\_to\_list[i]))

list(temp[:5])

# удаляем повторяющиеся слова из набора данных, следуя идее, что самые короткие и длинные обычно бесполезны

# преобразование строки в список токенов.

# deacc=True удаляет знаки препинания

def sent\_to\_words(sentences):

for sentence in sentences:

yield(gensim.utils.simple\_preprocess(str(sentence), deacc=True))

data\_words = list(sent\_to\_words(temp))

print(data\_words[:10])

len(data\_words)

# функция для детокенизации всех предложений

def detokenize(text):

return TreebankWordDetokenizer().detokenize(text)

data = []

for i in range(len(data\_words)):

data.append(detokenize(data\_words[i]))

print(data[:5])

data = np.array(data)

labels = np.array(data\_total['typr'])

y = []

for i in range(len(labels)):

#negative

if labels[i] == -1:

y.append(0)

#positive

if labels[i] == 1:

y.append(1)

y = np.array(y)

labels = tf.keras.utils.to\_categorical(y, 2, dtype="float32")

len(labels)

# преобразование текстовых данных в двумерные данные с плавающей запятой

max\_words = 5000

max\_len = 200

tokenizer = Tokenizer(num\_words=max\_words)

tokenizer.fit\_on\_texts(data)

sequences = tokenizer.texts\_to\_sequences(data)

tweets = pad\_sequences(sequences, maxlen=max\_len)

print(tweets)

print(labels)

# разбиение набора данных на обучающую и тестовую выборку в соотношении 4:1

#X\_train, X\_test, y\_train, y\_test = train\_test\_split(tweets, labels, random\_state=0)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tweets, labels, test\_size=0.2, random\_state=1)

print (len(X\_train),len(X\_test),len(y\_train),len(y\_test))

## RNN типа: двунаправленная LSTM

RNN обрабатывают последовательности, итерируя вдоль элементов последовательности и сохраняя информацию относительно того, что она обрабатывала до сих пор

# Bidirectional LTSM model

modelBR = Sequential()

modelBR.add(layers.Embedding(max\_words, 40, input\_length=max\_len))

modelBR.add(layers.Bidirectional(layers.LSTM(20, dropout=0.6)))

modelBR.add(layers.Dense(2, activation='softmax'))

modelBR.compile(optimizer='rmsprop', loss='categorical\_crossentropy', metrics=['accuracy'])

checkpoint2 = ModelCheckpoint("best\_model2.hdf5", monitor='val\_accuracy',

verbose=1, save\_best\_only=True, mode='auto', period=1, save\_weights\_only=False)

history = modelBR.fit(X\_train, y\_train, epochs=15, validation\_data=(X\_test, y\_test), callbacks=[checkpoint2])

# график точности при обучении и валидации модели

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# график потерь при обучении и валидации модели

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

best\_model = modelBR

test\_loss, test\_acc = best\_model.evaluate(X\_test, y\_test, verbose=2)

print('Model accuracy: ',test\_acc)

def NetWork(text):

sentiment = ['Негативный','Позитивный']

sequence = tokenizer.texts\_to\_sequences([text])

test = pad\_sequences(sequence, maxlen=max\_len)

predict = best\_model.predict(test)

print("Вероятность того, что отзыв позитивный: {:.2f}%".format(predict[0][1] \* 100))

print("Вероятность того, что отзыв негативный: {:.2f}%".format(predict[0][0] \* 100))

print(sentiment[np.around(predict, decimals=0).argmax(axis=1)[0]])

def addRowToDF(newRow, df) -> DataFrame:

df = pd.concat([newRow, df.loc[:]]).reset\_index(drop=True)

return df

df2 = pd.DataFrame(columns=['text', 'perc\_positive', 'perc\_negative', 'predicted', 'expected', 'match'])

df2 = addRowToDF(pd.DataFrame({'text':'я так рад за тебя', 'expected':'Позитивный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'какой ужасный день', 'expected':'Негативный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'какой прекрасный день', 'expected':'Позитивный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'ты сегодня очень плохо выглядешь', 'expected':'Негативный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'платье, которое ты надела, выглядет плохо', 'expected':'Негативный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'от тебя очень плохо пахнет', 'expected':'Негативный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'я думаю, нам пора расстаться', 'expected':'Негативный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'сегодня какой-то нехороший день у меня', 'expected':'Негативный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'Как понизить температуру ноута?', 'expected':'Негативный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'сегодня будет зачет по нейронным сетям', 'expected':'Негативный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'моя нейронная сеть не обучается', 'expected':'Негативный'}, index=[0]), df2)

df2 = addRowToDF(pd.DataFrame({'text':'моя нейронная сеть успешно обучится', 'expected':'Позитивный'}, index=[0]), df2)

for index, row in data\_negative.loc[24995:25000].iterrows():

df2 = addRowToDF(pd.DataFrame({'text':row['text'], 'expected':'Негативный'}, index=[0]), df2)

for index, row in data\_positive.loc[24995:25000].iterrows():

df2 = addRowToDF(pd.DataFrame({'text':row['text'], 'expected':'Позитивный'}, index=[0]), df2)

for index, row in df2.iterrows():

sentiment = ['Негативный','Позитивный']

sequence = tokenizer.texts\_to\_sequences([row['text']])

test = pad\_sequences(sequence, maxlen=max\_len)

predict = best\_model.predict(test)

row['perc\_positive'] = "{:.2f}%".format(predict[0][1] \* 100)

row['perc\_negative'] = "{:.2f}%".format(predict[0][0] \* 100)

row['predicted'] = sentiment[np.around(predict, decimals=0).argmax(axis=1)[0]]

match = A = 'yes' if row['predicted'] == row['expected'] else 'no'

row['match'] = match

df2

NetWork('сегодня ясный день')