**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МОЭВМ**

# ОТЧЕТ

**по индивидуальному домашнему заданию**
**по дисциплине «Искусственные нейронные сети»**
**Тема: «Анализ настроение по твитам»**

Студент гр. 7381 _____ Кортев Ю. В.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

**Цель работы.**

Датасет текста. Включает в себя набор твитов разной эмоциональной окраски, а также информацию о твите.

Задача заключается в определении того, насколько положительным или негативным является текст в твите.

## Задачи

- Модель должна быть разработана на языке Python с использованием Keras API
- Исходный код проекта должен быть быть в формате PEP8
- В исходном коде должны быть поясняющие комментарии
- Модель не должна быть избыточной (должен соблюдаться баланс между размером сети [кол-во слоев и нейронов] и качеством выдаваемого результата)
- Обучение модели должно быть стабильно (для предложенной архитектуры ИНС обучение должно приводить к примерно одним и тем же результатом, то есть не должно быть такого, что при обучении 10 сетей удовлетворительный результат дают только 5 из них)
- *Плюсом будет анализ с использованием callback'а TensorBoard*
- *Плюсом будет разработка собственных callback'ов*
- *Плюсом будет создание модели из ансамбля ИНС*

**Требования**

- В отчете должно быть описание датасета, а также описание решаемой
- В отчете должен быть проведен начальный анализ данных. Проведение статистического анализа, анализ на необходимость нормировки данных, обоснование применения определенного вида нормировки, и.т.д.
- В отчете необходимо отразить весь процесс разработки: с чего началась разработка модели, на основании чего проводились те или иные изменения/корректировки, обоснование выбора тех или иных изменений/корректировок. По сути выполнение должно быть разбито на итерации, каждая итерация должна сопровождаться результатами

модели на итерации, а также краткой выдержкой исходного кода, показывающая изменения на итерации

- В конце отчета должен быть приведен анализ результирующей модели, а также перечислены возникшие проблемы (и как они были решены) и проблемы, которые решить не удалось. Плюсом будет предложение по улучшению модели.
- В отчете должно быть указано, кто в бригаде за что отвечал (написание отчета не является зоной ответственности)
- В приложении должен быть исходный код
- *Плюсом будет сравнение разработанной модели с методами решающими задачу и не относящимися к ИНС.*

**Ход работы.**

**Анализ датасета и задачи.**

Первые элементы выборки показаны на рисунке 1.

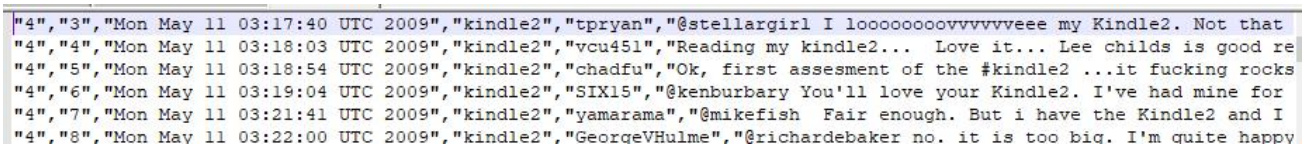


Рисунок 1 - Выборка

Структура выборки следующая:

1ым столбцом идет оценка настроение сообщения, вторым столбцом идет id твита, третьим - дата поста, четвертым - имя пользователя, пятым - запрос в сообщении, шестым - само сообщения. Дополнительная сложность в том, что во многих словах опечатки, много обращений и ссылок. Используются смайлы, а в некоторых словах используется повторение букв с целью предания эмоционального окраса(lllllllllooooooooooovvveeeeee)

Задача анализа настроения - определение полярности настроение в тексте (позитивное, негативное, нейтральное).

Как и во многих других задачах анализа текста, наибольший объем анализа настроение представляет предварительная обработка текста.

Препроцессинг включает в себя токенизацию, исправление опечаток в словах, удаление слов не имеющих смысла(имена, местоимения, ссылки), лемматизацию(приведение слов к нормальной форме). Все это позволяет, как применять словарь полученный на исходном датасете к пользовательским текстам, так и сузить объем словаря, чтобы он включал в себя только слова, имеющие влияние на результат.

Нормализация данных не требуется, тк токены, представляющие слова, будут проиндексированы, и каждое слово будет представлено в виде векторного представления.

**Препроцессинг.**

За предварительную обработку текста отвечает модуль preprocessing.py.

Он загружает выборку, выделяет в ней образцы и метки, токенизирует метки, удаляет ссылки на других людей (@name), удаляет url адреса, удаляет слова не имеющие смысла (местоимения, союзы), производит лемматизацию слов.

Все описанные процессы реализуются с использованием библиотек re и nltk, re - реализует регулярные выражения, а nltk - обработку естественного языка.

В листинге 1 показан основной фрагмент модуля.

```
def gen_tokens(X):
    stop_words = set(stopwords.words("english")) #stopwords like 'i we their'
    tweet = TweetTokenizer()
    lemmat = WordNetLemmatizer()

    for i, x in enumerate(X):
        x = re.sub(r'https?://[^\"\s>]+', ' ', x) #del urls
        x = re.sub(r'@\w+', ' ', x)              #del names
        x = tweet.tokenize(x)          #tokenizer saving smiles
        x = [w.lower() for w in x if not re.match(r"[@&:;$~()<>`\[\]+#/\"*0-9%'.,!?\\-]" , w)]
#del punct
        pos_tag = nltk.pos_tag(x)  #get list of word tags like verb
        for j, w in enumerate(x):
            x[j] = lemmat.lemmatize(w, tag_dict.get(pos_tag[j][1][0].upper(), wordnet.NOUN))
#set words to normal form
        X[i] = [w for w in x if w not in stop_words] #del stopwords

    return X
```

Листинг 1 - Функция токенизации сообщения

Благодаря токенизатору TweetTonenizer библиотеки nltk обученная модель сможет понимать значения смайликов.

На рисунке 2-3 показаны исходные элементы выборки и результаты препроцессинга.

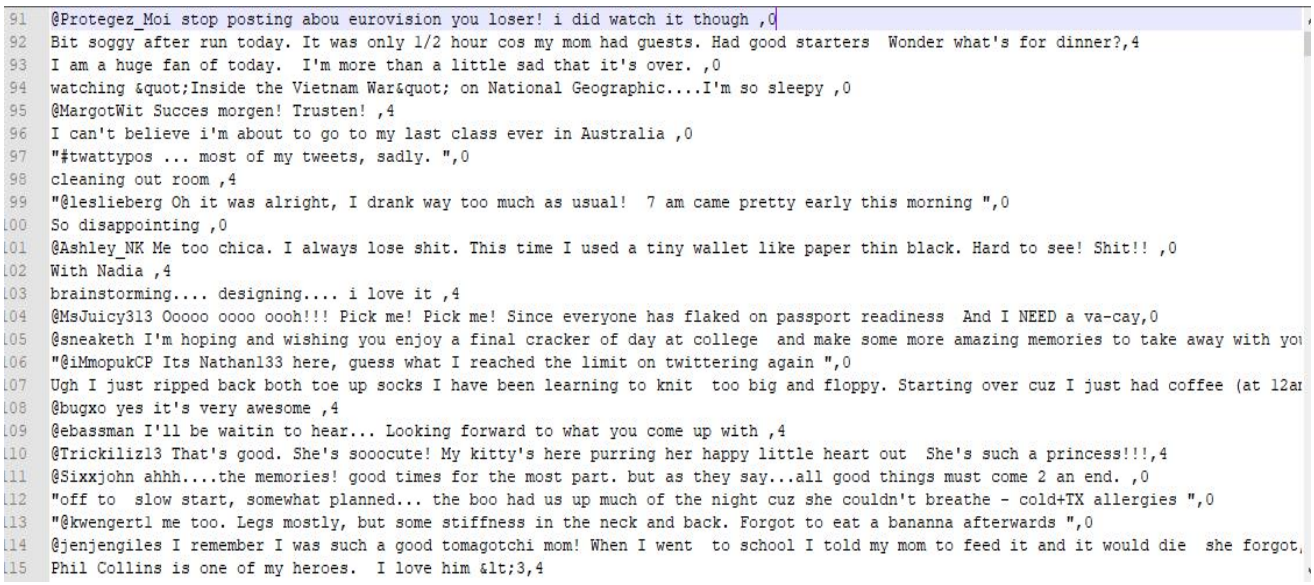


Рисунок 2 - Образцы до предварительной обработки

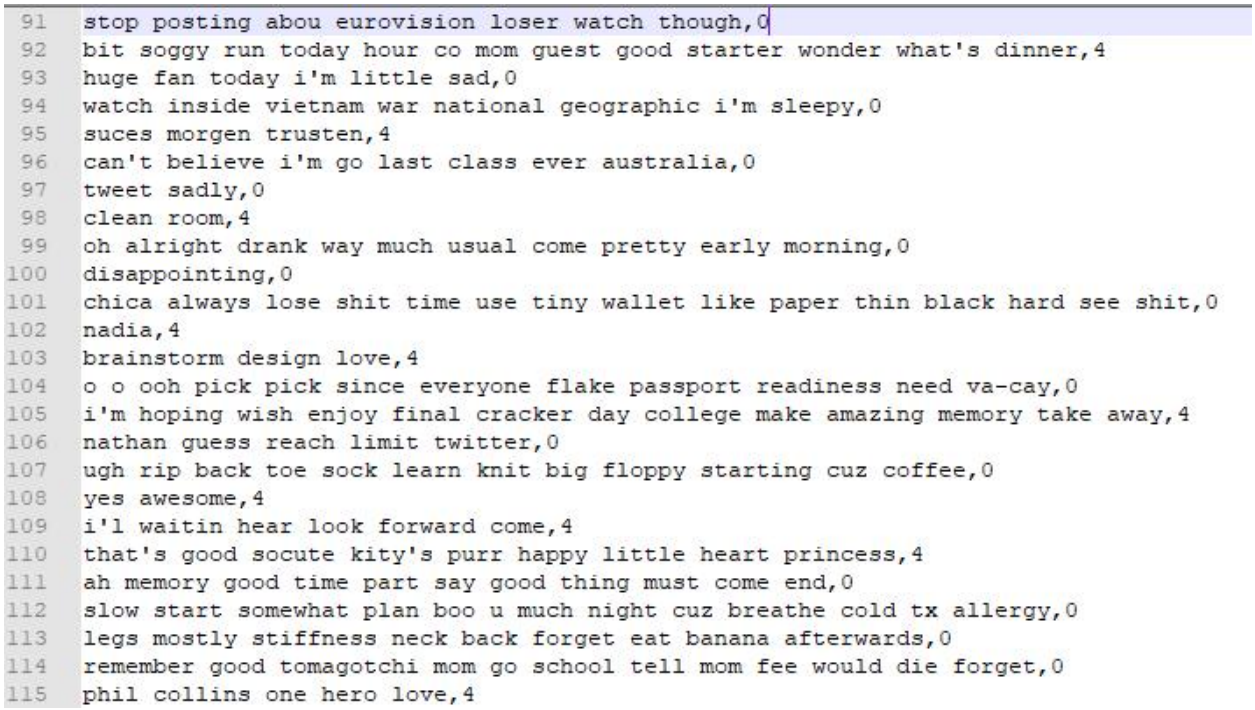


Рисунок 3 - Образцы после предварительной обработки

Генерация словаря показана в листинге 2.

```
def gen_wrd2idx(X):
    wrd2id={}
    for i, sent in enumerate(X):
        for j, token in enumerate(X[i]):
            if token not in wrd2id:
                wrd2id[token] = 1
            else:
                wrd2id[token]+=1  #word freq

    wrd2id={k: v for k, v in sorted(wrd2id.items(), key=lambda item: item[1], reverse=True)}
#sort words by freq
    wrd2id={k: i+1 for i, k in enumerate(wrd2id.keys())} #set indxs
    wrd2id['-UNKNWN-']=0
    return wrd2id
```

Листинг 2 - Функция генерации словаря

В полученном словаре токены отсортированы по популярности, что позволяет обрезать словарь, используя самые популярные слова, исключая те, в которых совершены ошибки, или несуществующие слова.

Модуль train_models.py получает из модуля preprocessing.py датасет и метки и обучает модели. В листинге 3 - 4 показаны инициализация и параметры обучения моделей. Для начала были выбраны полносвязная модель и двунаправленная рекуррентная модель.

```python
def train_dense_model():
    model = Sequential()
    model.add(Embedding(len_dict, 3, input_length=pad,name='emb'))
    model.add(Flatten())
    model.add(Dense(8, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(8, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

    num_epochs = 30;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs, batch_size=512
                  , validation_data=(train_x[train:size_of_dataset], train_y[train:size_of_dataset]),
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.3,
                          patience=3
                      )
                  ])

    model.save('model.h5')

    plot_history(h,num_epochs)
```

Листинг 3 - Архитектура и параметры обучения полносвязной модели

```python
def train_bidir_rnn():
    model = Sequential()
    model.add(Embedding(len_dict, 8, input_length=pad, name='emb'))
    model.add(Bidirectional(LSTM(60,return_sequences=True, dropout=0.3)))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    num_epochs = 30;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs, batch_size=1024
                  , validation_data=(train_x[train:size_of_dataset], train_y[train:size_of_dataset]),
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.2,
                          patience=6
                      )])

    plot_history(h, num_epochs)
    model.save('model_bidir_lstm.h5')
```

Листинг 4 - Архитектура и параметры обучения двунаправленной LSTM модели

Обе модули были обучены на 500 000 отобранных из перемешанной выборки образцах. Обе модели показывают одинаковый результат в примерно 76 процентов точности на тестовой выборке. Из того, что обе модели показываются одинаковый результат, делаю вывод, что архитектура и параметры были подобраны правильно, значит нужно работать над выборкой. Для постановки проблемы, решение которой увеличит точность, воспользуюсь TensorBoard. На рисунке 4 показано векторное пространство, которые формирует слой embedding спроецированное в трехмерное пространство. Как видно пространство слов группируется в 2 кластера слов, с положительным эмоциональным окрасом и отрицательным. На рисунках 5-6 показаны слова, входящие в эти кластеры.

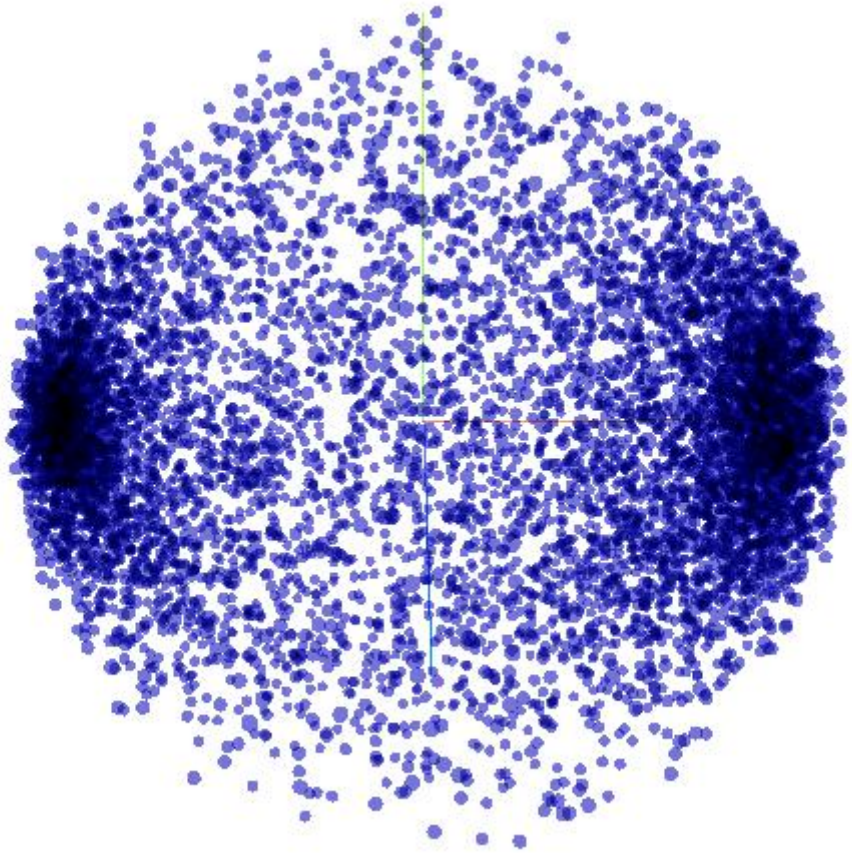

Рисунок 4 - Пространство слов

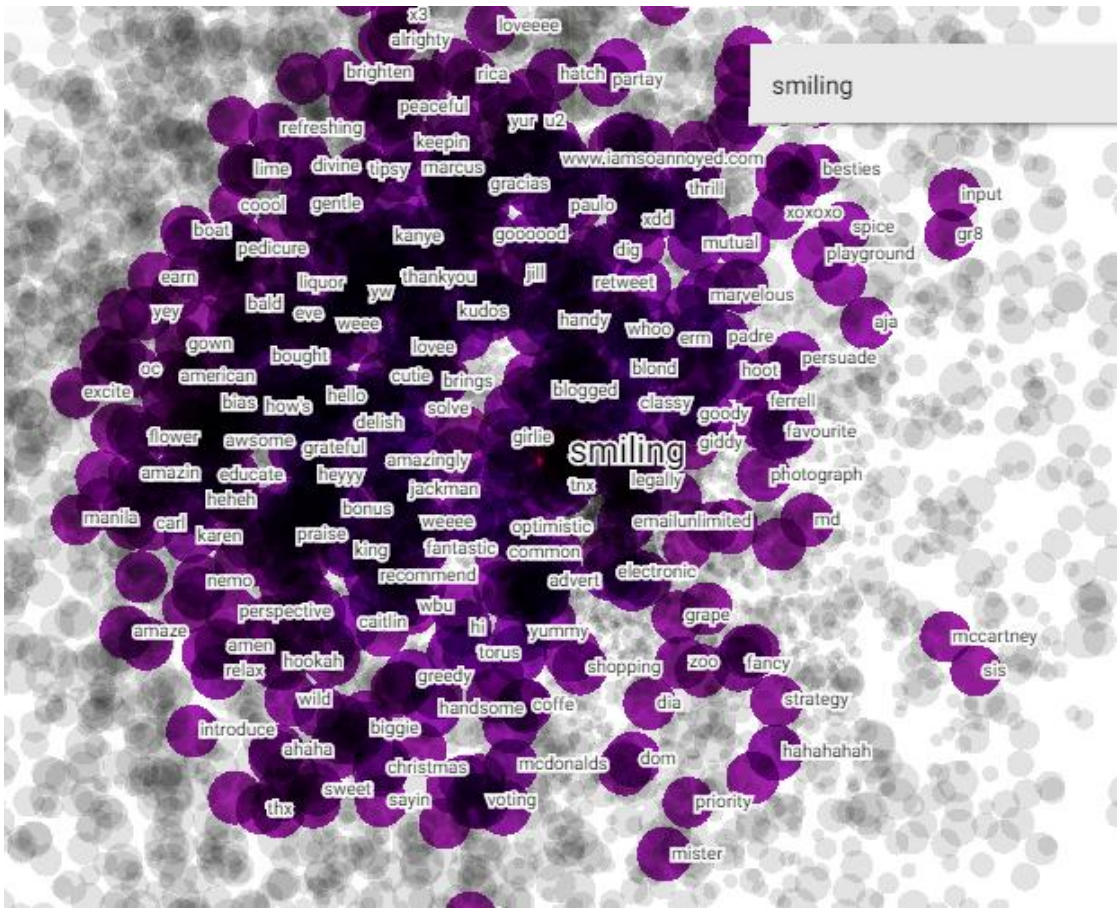


Рисунок 5 - Позитивный кластер

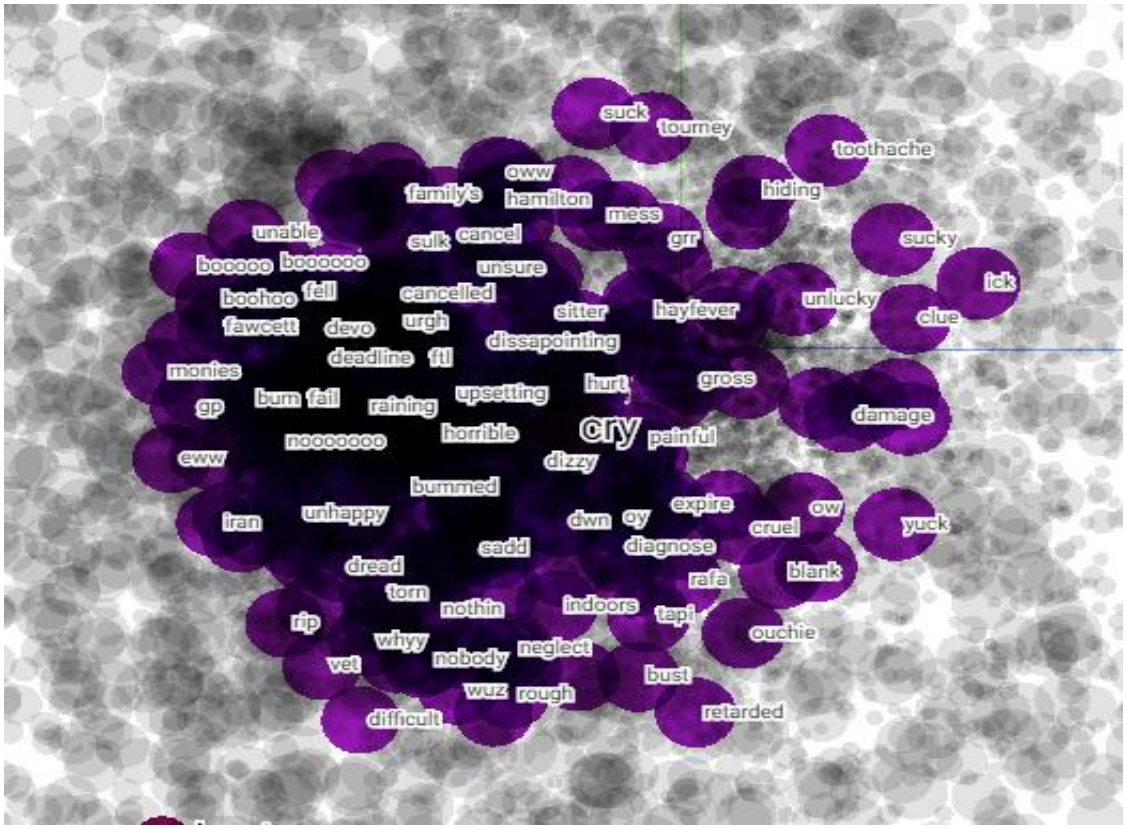


Рисунок 6 - Негативный кластер

Проанализировав рисунки 5-6 можно увидеть слова в которых повторяются буквы (noooo, booo,lovee). Слой embedding правильно обучился понимать положительный окрас несет такое слово или отрицательной, но если он увидит например слово loveeee в тестовой выборке, он не узнает его и не сможет определить положительное оно или отрицательное, для модели это будет новое слово. Для решение этой проблемы в модуле preprocessing.py был добавлен класс RepeatReplacer, реализация класса показана в листинге 5. Класс использует регулярное выражание для удаления повторяющихся букв, а также словал wordnet для раннего выхода из рекурсии.

```
class RepeatReplacer(object):
    def __init__(self):
        self.repeat_regexp=re.compile(r'(\w*)(\w)\2(\w*)')
        self.repl=r'\1\2\3'

    def replace(self, word):
        if wordnet.synsets(word):
            return word
        repl_word=self.repeat_regexp.sub(self.repl,word)
        if repl_word!=word:
            return self.replace(repl_word)
        else:
            return repl_word
```

Листинг 5 - Класс удаления повторяющихся букв в словах

Теперь модель должна встречать больше знакомых слов в тестовой выборке.

На рисунке 7 показано векторное пространство в котором в 2 раза больше слов. Как видно при увеличении пространство слов, объем кластеров также увеличивается, что должно положительно сказаться на точности. При этом тк словарь отсортирован по популярности слов, чрезмерное увеличение пространства слов не будет оказывать большого эффекта.

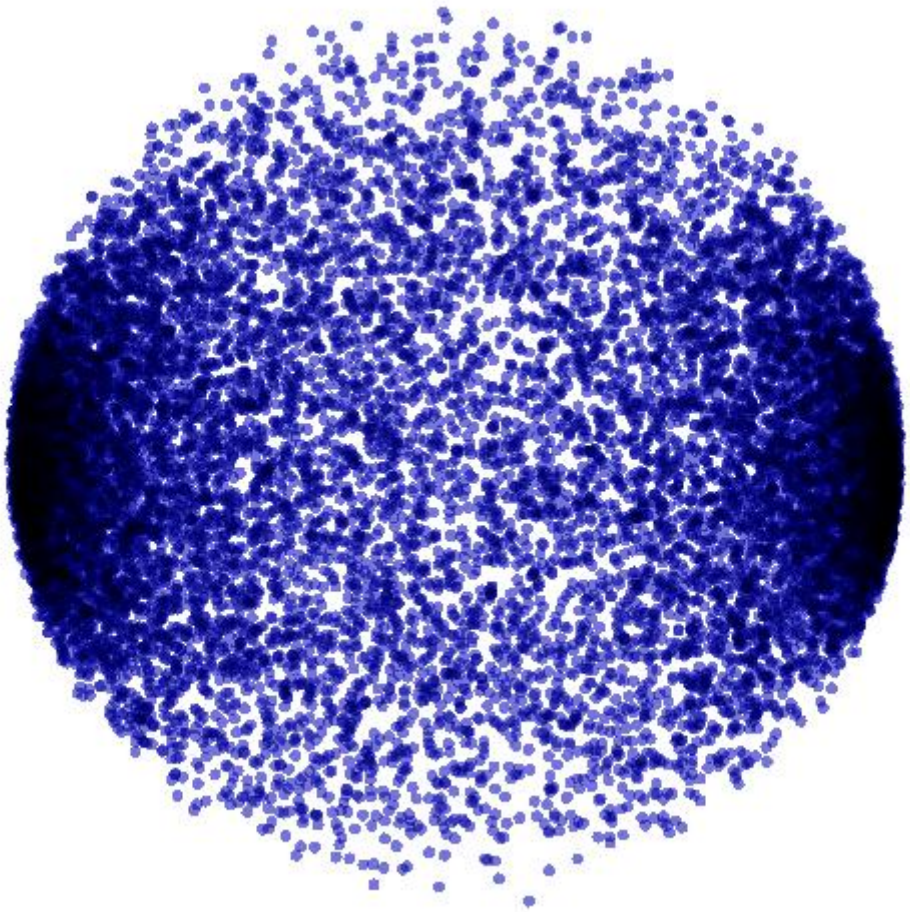

Рисунок 7 - Пространство из 15 000 слов

На рисунках 8-11 показаны графики точности и потерь для полносвязной и двунаправленной модели.

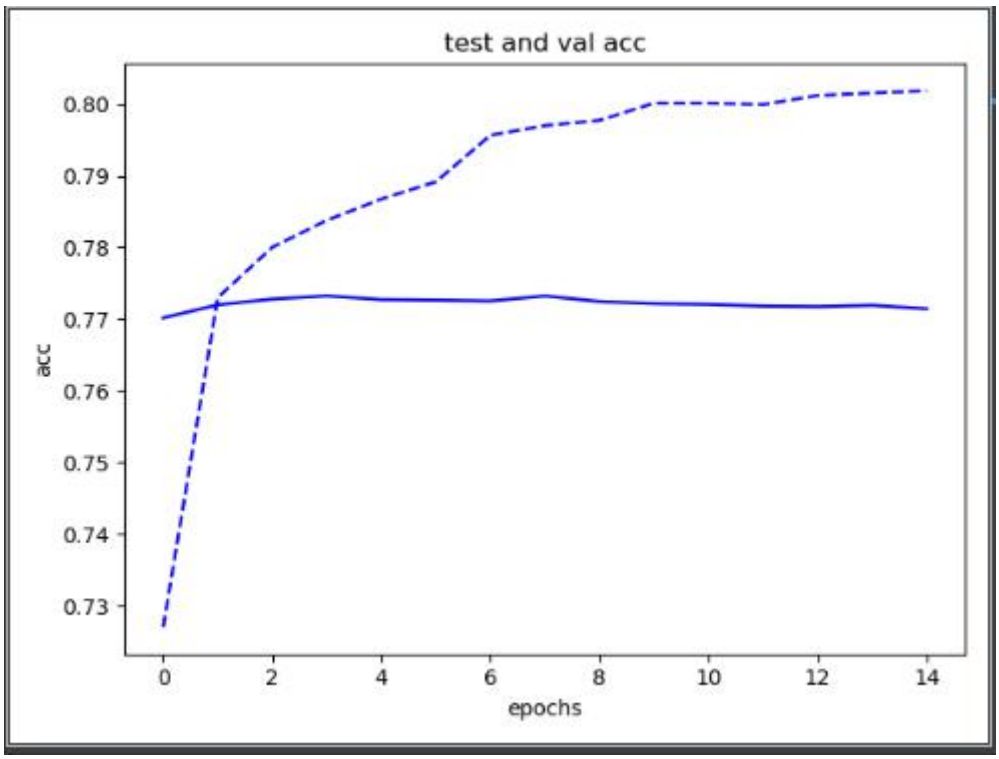


Рисунок 8 - Точность полносвязной модели

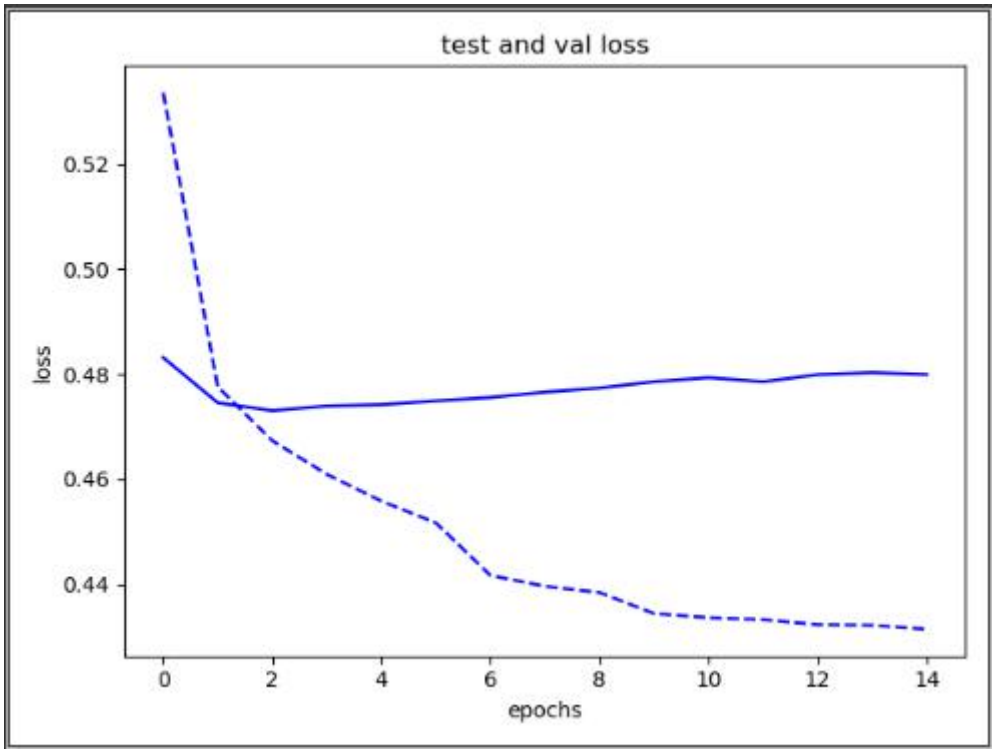


Рисунок 9 - Потери полносвязной модели

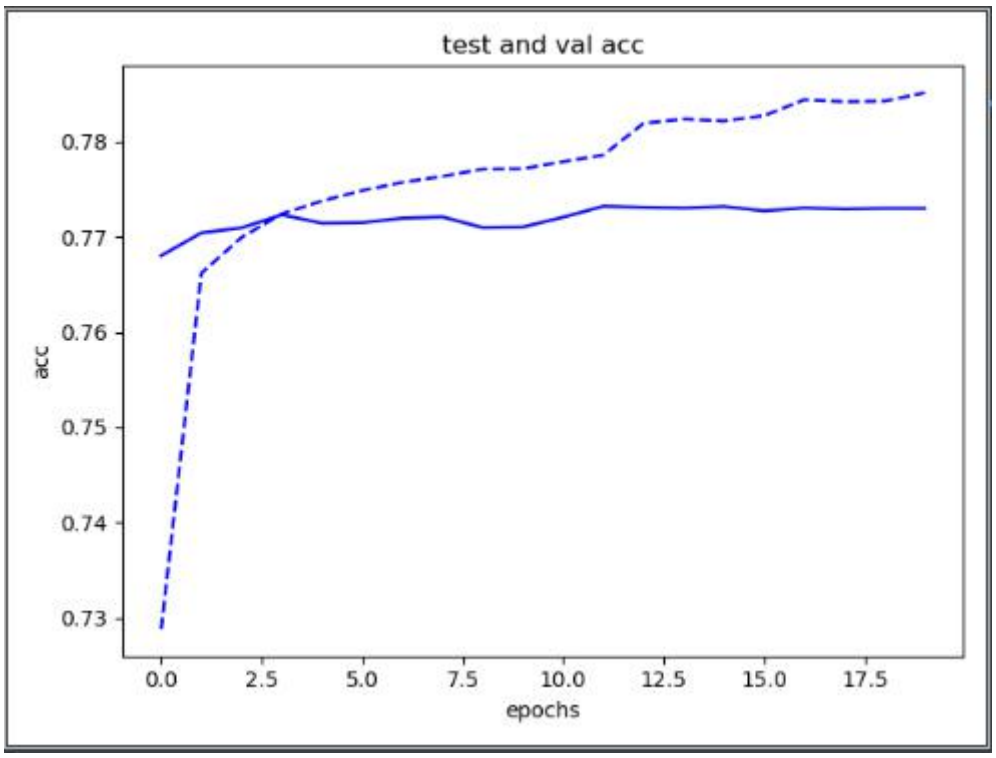


Рисунок 10 - Точность двунаправленной LSTM модели

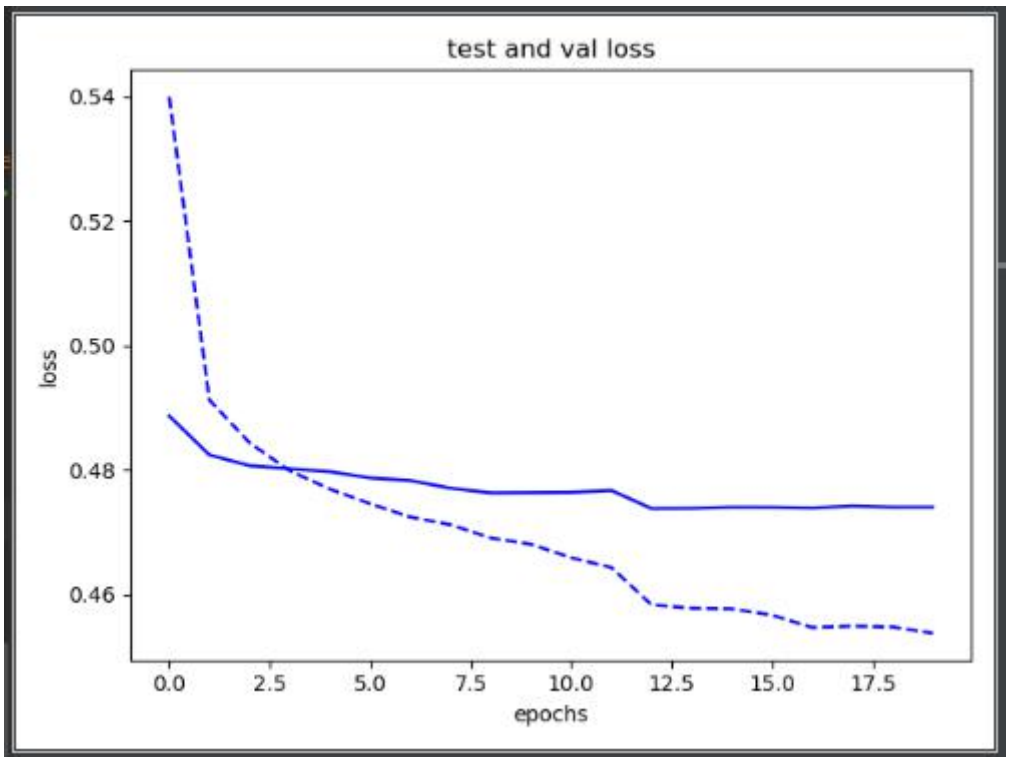


Рисунок 11 - Потери двунаправленной LSTM модели

Проделанные действия улучшили результат, теперь для увеличения точности, увеличу датасет до 1 000 000 образцов и увеличу емкость моделей. Также обучу модель с 2мя слоями lstm и сверточную модель со слоем lstm.

Архитектуры всех 4х полученных моделей показаны в листингах, графики точности и потерь показаны на рисунках.

```
def train_dense_model():
    model = Sequential()
    model.add(Embedding(len_dict, 8, input_length=pad,name='emb'))
    model.add(Flatten())
    model.add(Dense(160, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(120, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

    num_epochs = 3;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs,
batch_size=1024
                  , validation_data=(train_x[train:size_of_dataset],
train_y[train:size_of_dataset]), verbose=2,
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.3,
                          patience=3
                      )
                  ])

    model.save('model.h5')
```

Листинг 6 - Инициализация и обучение полносвязной модели

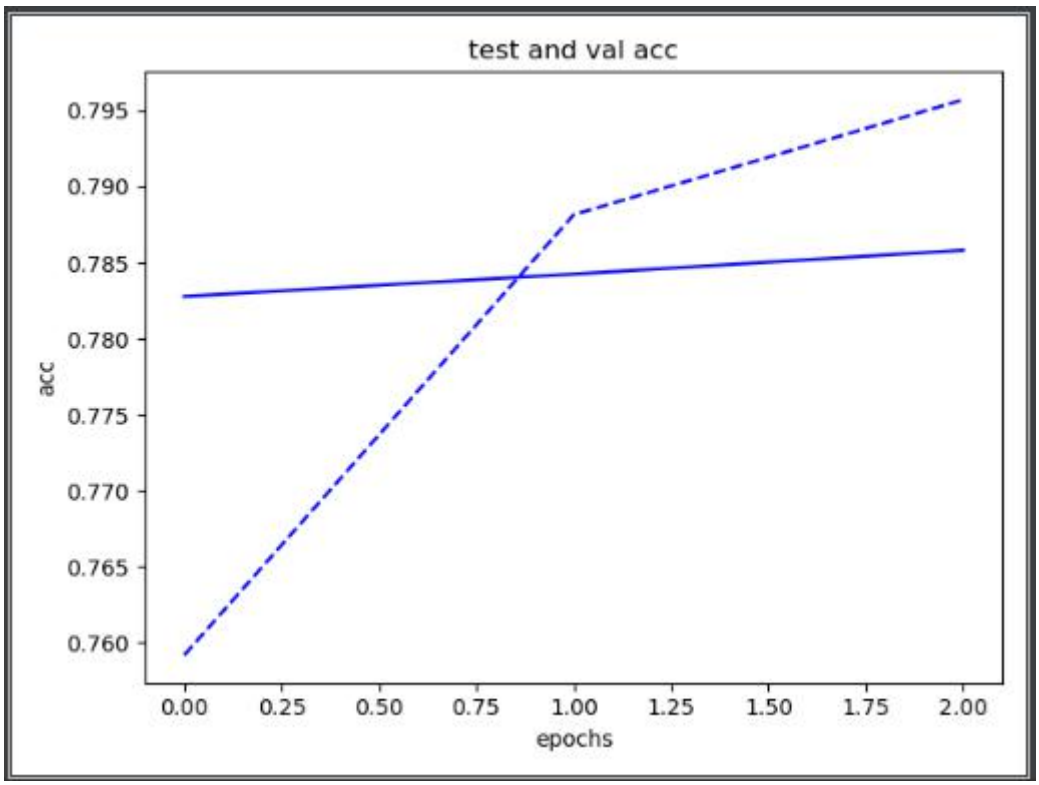


Рисунок 12 - Точность полносвязной модели

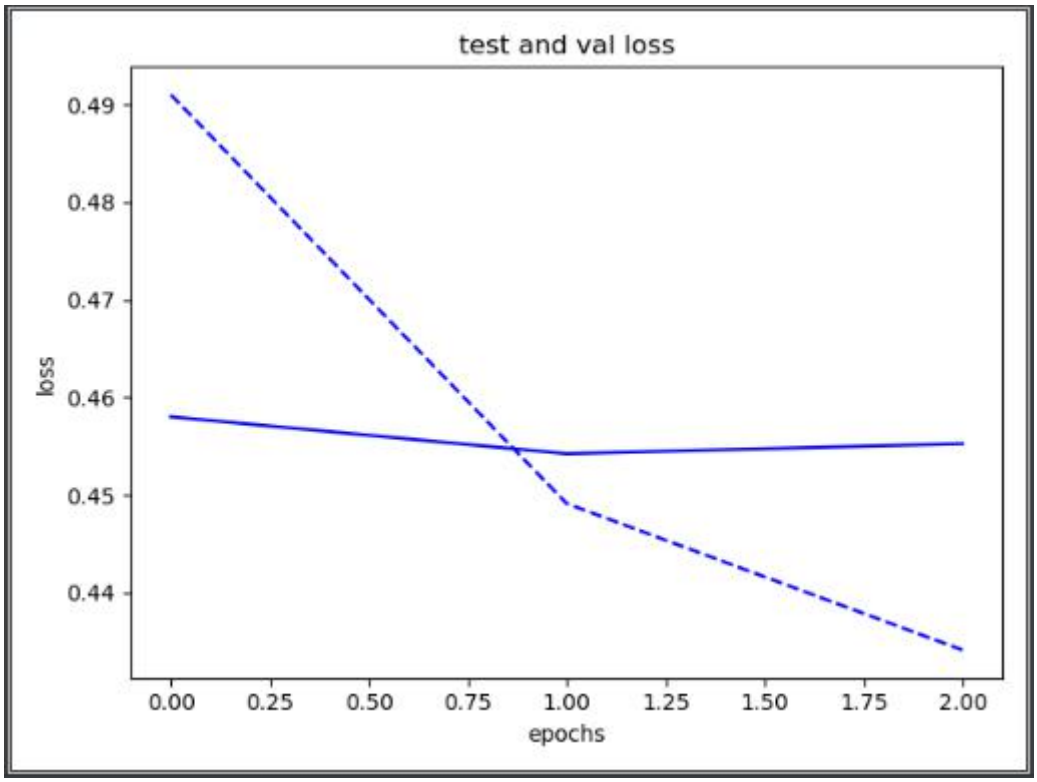


Рисунок 13 - Потери полносвязной модели

```
def train_2lstm():
    model = Sequential()
    model.add(Embedding(len_dict, 18, input_length=pad, name='emb'))
    model.add(LSTM(80, dropout=0.3,return_sequences=True))
    model.add(LSTM(150, dropout=0.3))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])

    num_epochs = 30;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs,
batch_size=1024
                  , validation_data=(train_x[train:size_of_dataset],
train_y[train:size_of_dataset]), verbose=2,
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.3,
                          patience=6
                      )
                  ])

    plot_history(h,num_epochs)
```

Листинг 7 - Инициализация и обучение сверточной модели со слоем lstm

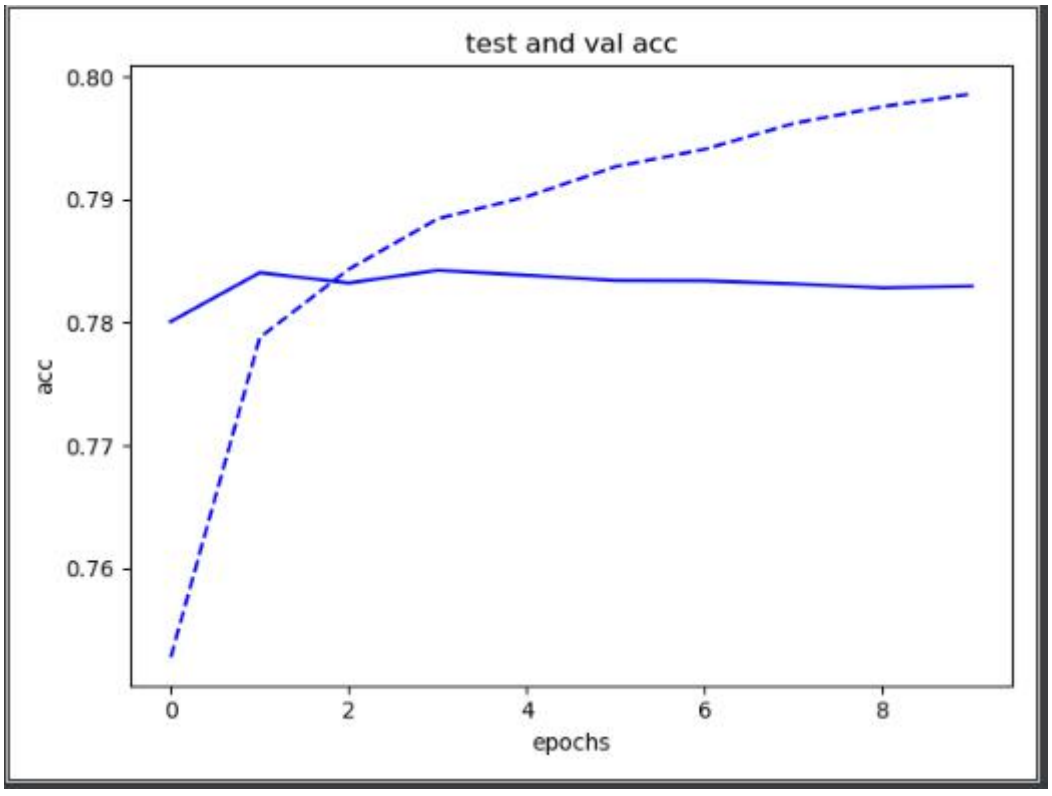


Рисунок 14 - Точность сверточной модели со слоем lstm

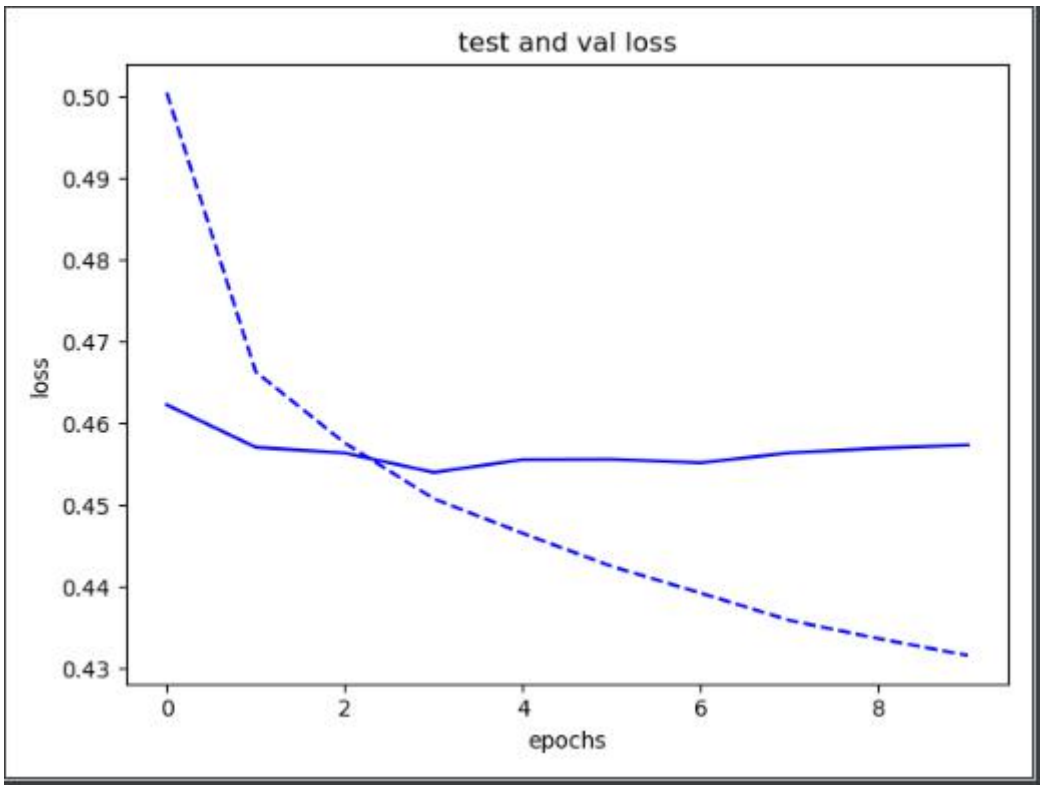


Рисунок 15 - Потери сверточной модели со слоем lstm

```
def train_bidir_rnn():
    model = Sequential()
    model.add(Embedding(len_dict, 18, input_length=pad, name='emb'))
    model.add(Dropout(0.3))
    model.add(Bidirectional(LSTM(90, dropout=0.5)))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    num_epochs = 12;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs, batch_size=1024
                  , validation_data=(train_x[train:size_of_dataset],
train_y[train:size_of_dataset]), verbose=2,
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.2,
                          patience=6
                      )])
```

Листинг 8 - Инициализация и обучение двунаправленной lstm модели

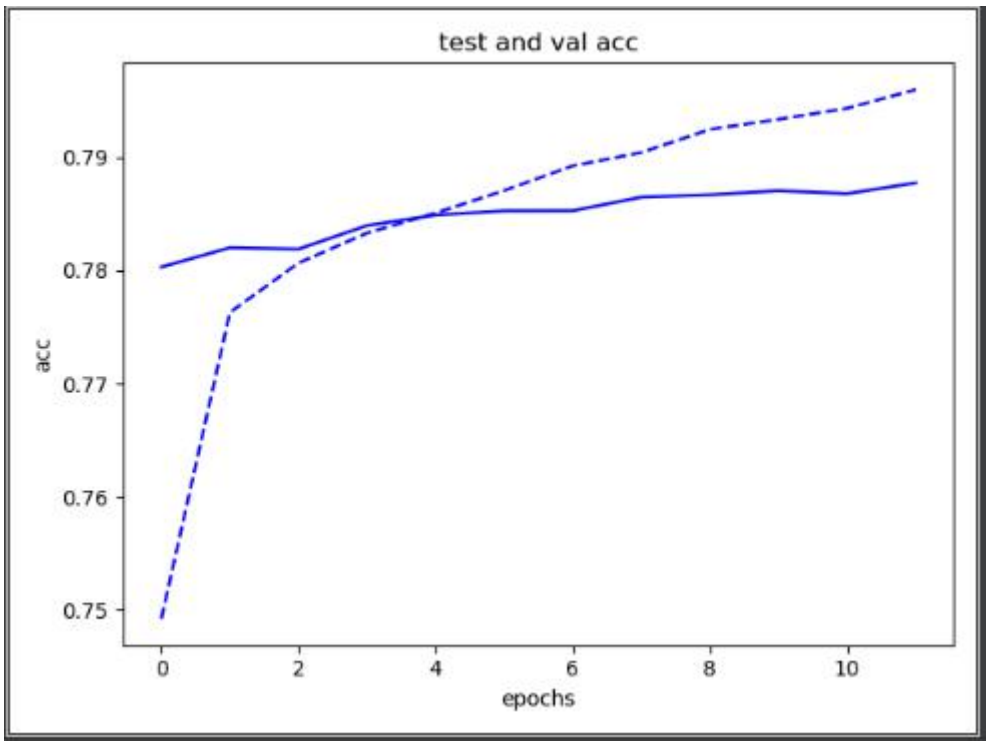


Рисунок 16 - Точность двунаправленной lstm модели

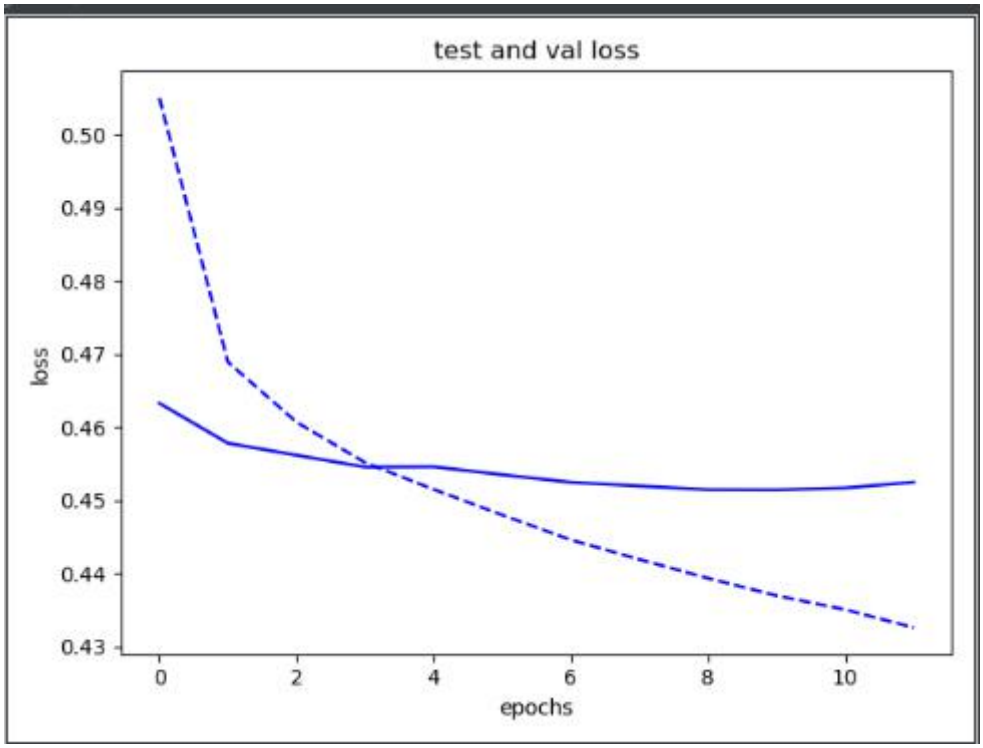


Рисунок 17 - Потери двунаправленной lstm модели

```
def train_2lstm():
    model = Sequential()
    model.add(Embedding(len_dict, 18, input_length=pad, name='emb'))
    model.add(LSTM(80, dropout=0.3,return_sequences=True))
    model.add(LSTM(150, dropout=0.3))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])

    num_epochs = 30;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs,
batch_size=1024
                  , validation_data=(train_x[train:size_of_dataset],
train_y[train:size_of_dataset]), verbose=2,
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.3,
                          patience=6
                      )
                  ])
```

Листинг 9 - Инициализация и обучение рекуррентной модели с 2мя слоями lstm

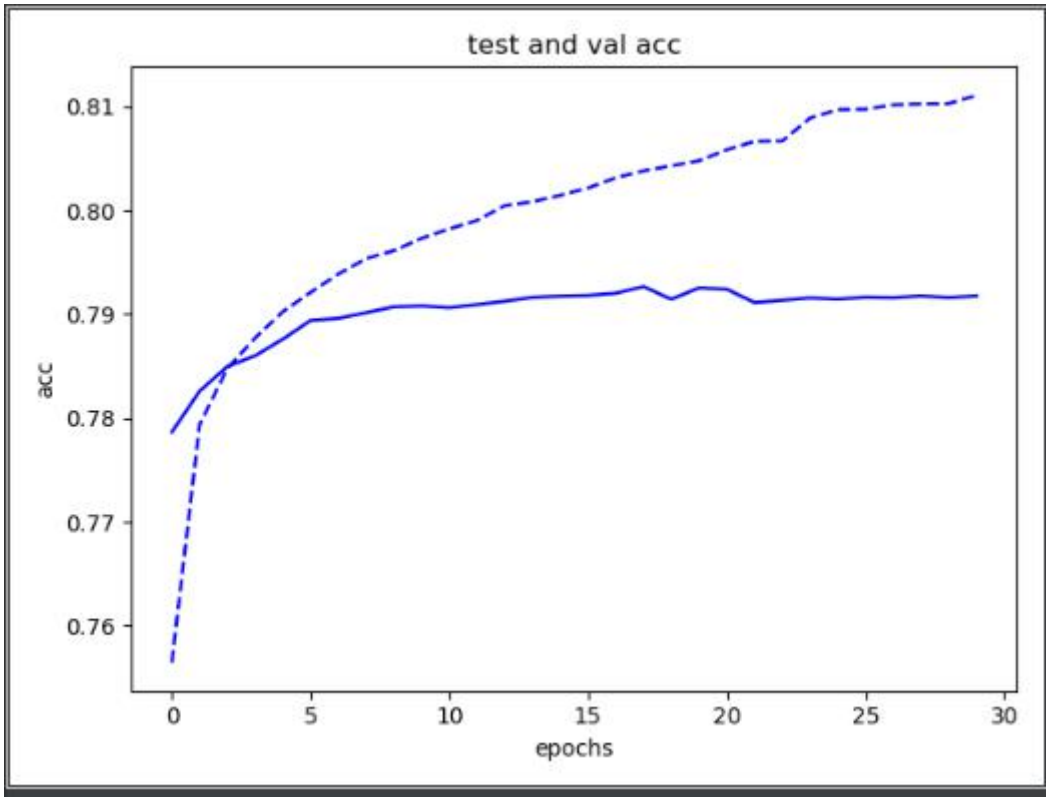


Рисунок 18 - Точность рекуррентной модели

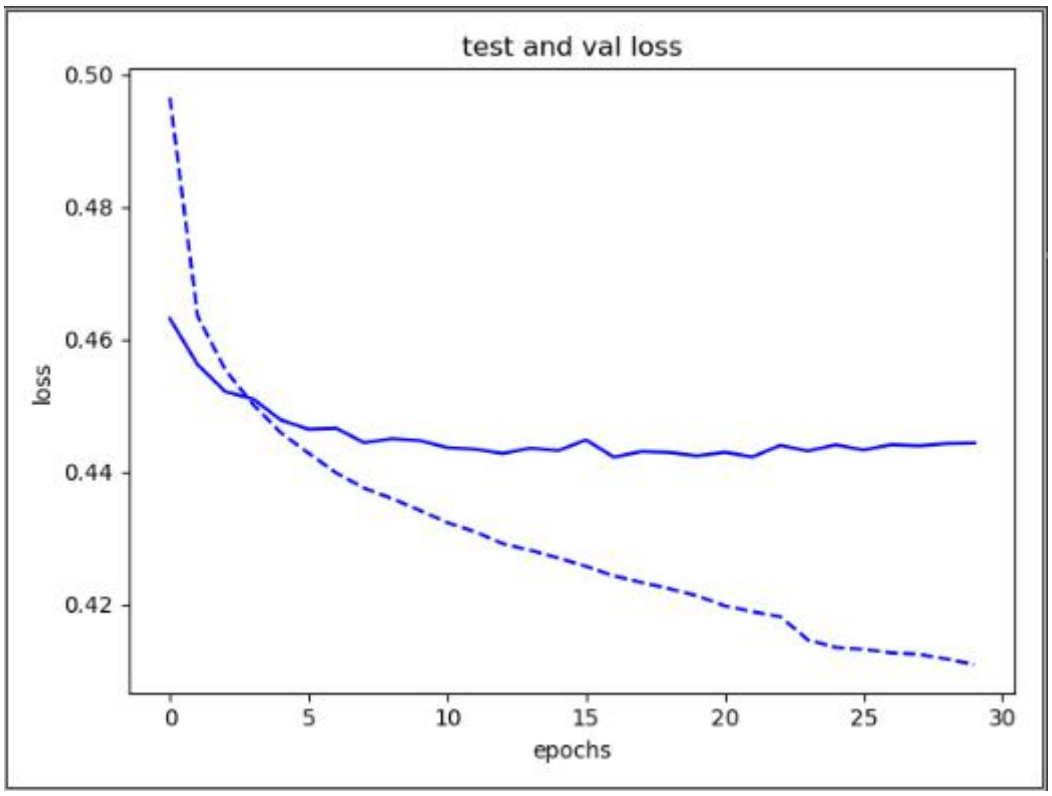


Рисунок 19 - Потери рекуррентной модели

Все модели примерно одинаково хорошо справляются с задачей, из этого делаю вывод, что подобраны правильные архитектуры и параметры обучения, реализующие максимум информации, заключенной в датасете. К тому же каждая модель имеет свой подход к получению результата, поэтому логично будет объединить модели в ансамбль нейронных сетей. Точность модели с 2мя последовательными слоями lstm примерно на 1 процент больше других моделей, поэтому ей будет присвоен немного больший вес в ансамбле. Ансамблирование будет использоваться для классификации пользовательских сообщений.

Реализация ансамблированной модели показана в листинге 10.

```
class PredictSentiment(object):
    def __init__(self):
        self.cnn_lstm = load_model('model_cnn_lstm.h5')
        self.model_2lstm = load_model('model_2lstm.h5')
        self.fnn = load_model('model.h5')
        self.bi_cnn = load_model('model_bidir_lstm.h5')

    def predict(self,x):
        cnn_lstm_pred = self.cnn_lstm.predict(x)
        fnn_pred = self.fnn.predict(x)
        bi_cnn_pred = self.bi_cnn.predict(x)
        lstm_pred = self.model_2lstm.predict(x)

        pred=0.3*lstm_pred+0.24*bi_cnn_pred+0.23*fnn_pred+0.23*cnn_lstm_pred

        for i, pr in enumerate(pred):
            if 0.0<=pr[0]<0.4:
                print('negativ'.format(i + 1), end=' ')
            elif 0.4<=pr[0]<0.6:
                print('neitral'.format(i + 1), end=' ')
            elif 0.6 <= pr[0] <= 1.0:
                print('positiv'.format(i + 1), end=' ')

            print('({:.2f}=fnn: {:.2f} + cnn_lstm: {:.2f} + bi_lstm: {:.2f} + 2lstm:
{:.2f})'.format(pr[0], fnn_pred[i][0],

cnn_lstm_pred[i][0],

bi_cnn_pred[i][0],
```

Листинг 10 - Класс реализующий ансамблирование

Для обработки пользовательских сообщений создан модуль predict_users.py. Он загружает файлы с пользовательскими обзорами, используя модуль preprocessing, обрабатывает их и передает на вход ансамблю. Пример работы модуля показан на рисунке 20.

```
tokens:
 [['movie', 'surprising', 'plenty', 'unsettle', 'plot', 'twist'], ['dislike', 'cabin', 'cruiser']]
word indexing:
 [[104, 8095, 2040, 0, 4354, 3105], [2837, 5660, 12781]]
1: positiv (0.95=fnn: 0.97 + cnn_lstm: 0.96 + bi_lstm: 0.96 + 2lstm: 0.93)
2: negativ (0.15=fnn: 0.08 + cnn_lstm: 0.18 + bi_lstm: 0.23 + 2lstm: 0.13)
```

Рисунок 20 - Предсказание  настроения пользовательских сообщений

Полученная модель хорошо справляется с поставленной задачей. Модель с 2мя последовательными lstm слоями справляется немного лучше.
Совет по улучшению модели:
Очень часто пользователи совершают опечатки или пытаются создать какие-то новые слова, производные от других. Во время предварительной обработки при проверке такие признаки исключаются, хотя имеют значение. Если реализовать или найти механизм приведения таких слов к изначальной форме и использовать, модель может показать лучшие результаты. Также возможно использование предварительно натренированного embedding слоя может положительно сказаться на точности.

Была проблема при которой предварительная обработка пропускала слова с повторяющимся буквами (lllllloooooooooovvvvvvveeeeeee), из-за чего такие слова во время проверки помечались как неизвестные, хотя имели большое значение. Как уже говорилось проблема решена созданием класса, который исправляет такие слова. Решение проблемы немного повысило итоговую точность.
Осталась проблема при которой некоторые конструкции слов не разделялись (I’m,  I’d), из-за чего такие ничего не значащие слова оставались в словаре.

Сравнения с другими методами:
Данная задача также может решаться rule-based методом. Он заключается в разделении последовательностей на токены, затем над каждым токеном производится множество лингвистических операций, таких как лемматизация, морфологический разбор с целю приведения слов к нормальной форме. Используя заранее заготовленный словарь, слова разделяются на положительно направленные и отрицательные. Дальше сравниваются размеры полученных списков, если размер положительного списка больше, то сообщение позитивное, наоборот - отрицательное.

Если сравнивать этот метод с решением, применяющим инс, то отличие лишь в том, что правила, по которым текст классифицируется генерируется не человеком, а моделью в процессе обучения. Словарь например не создается человеком или написанной программой, а генерируется в слое embedding, а процесс подсчета слов заключен в активации нейронов.

**Вывод**

В итоге индивидуального домашнего задания была построена ансамблированная модель, решающая задачу анализа текста по настроению. Были получены навыки в предварительной лингвистической обработке текста.

# ПРИЛОЖЕНИЕ А

# ИСХОДНЫЙ КОД PREPROCESSING.PY

```
import pandas as pd
import numpy as np
import  re
import nltk
import spacy
from nltk import word_tokenize, TweetTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.corpus import stopwords
from nltk.corpus import words
import json
import random
from keras.preprocessing import sequence
import math

random.seed(999)

word2idx_path='wrd2idx.txt'

tag_dict = {"J": wordnet.ADJ,    #dictionary for lemmatizer
            "N": wordnet.NOUN,
            "V": wordnet.VERB,
            "R": wordnet.ADV}

class RepeatReplacer(object):
    def __init__(self):
        self.repeat_regexp=re.compile(r'(\w*)(\w)\2(\w*)')
        self.repl=r'\1\2\3'

    def replace(self, word):
        if wordnet.synsets(word):
            return word
        repl_word=self.repeat_regexp.sub(self.repl,word)
        if repl_word!=word:
            return self.replace(repl_word)
        else:
            return repl_word

def gen_tokens(X):
    stop_words = set(stopwords.words("english")) #stopwords like 'i we
their'
    tweet = TweetTokenizer()
    lemmat = WordNetLemmatizer()
    repeat_replacer=RepeatReplacer()

    for i, x in enumerate(X):
        x = re.sub(r'https?://[^\"\s>]+', ' ', x) #del urls
        x = re.sub(r'@\w+', ' ', x)               #del names
```

```
        x = tweet.tokenize(x)          #tokenizer saving smiles
        x = [w.lower() for w in x if len(w)!=1]# and not
re.match(r"[@&:;$~()<>`\[\]+#/\"*0-9%'.,!?\\-]" , w)]# del punct
        pos_tag = nltk.pos_tag(x)  #get list of word tags like verb
        for j, w in enumerate(x):
            x[j] = lemmat.lemmatize(w,
tag_dict.get(pos_tag[j][1][0].upper(), wordnet.NOUN)) #set words to
normal form
        x = [w for w in x if w not in stop_words] #del stopwords
        X[i]=[repeat_replacer.replace(w) for w in x]
    return X


def gen_wrd2idx(X):
    wrd2id={}
    for i, sent in enumerate(X):
        for j, token in enumerate(X[i]):
            if token not in wrd2id:
                wrd2id[token] = 1
            else:
                wrd2id[token]+=1  #word freq

    wrd2id={k: v for k, v in sorted(wrd2id.items(), key=lambda item:
item[1], reverse=True)} #sort words by freq
    wrd2id={k: i+1 for i, k in enumerate(wrd2id.keys())} #set indxs
    wrd2id['-UNKNWN-']=0
    return wrd2id


def indexing(X, wrd2idx):
    for i, sent in enumerate(X):

        try:
            X[i]=sent.split(' ') #split tokens
        except Exception:
            X[i]=[0] #if empty sent

        for j, word in enumerate(X[i]):
            if word in wrd2idx:
                X[i][j]=wrd2idx[word] #set to tokens id
            else:
                X[i][j]=0 #incnwn word
        X[i]=np.array(X[i])
    X=np.array(X)
    return X



def get_dataset(path,word2idx_path,len_of_dir=10000,cut=500):
    dataframe = pd.read_csv(path, header=None, sep=',', encoding='utf-
8')
```

```
    with open(word2idx_path, 'r') as json_file:
        wrd2idx = json.load(json_file)

    wrd2idx = dict(list(wrd2idx.items())[:len_of_dir]) #use len_of_dir
most popular words
    wrd2idx['UNKNWN']=0
    dataset=dataframe.values
    X=dataset[:,0]
    Y=dataset[:,1]

    X=indexing(X,wrd2idx)

    for i, label in enumerate(Y):
        if label == 0:
            Y[i]=0
        elif label == 4:
            Y[i]=1

    Y=np.array(Y)
    X=sequence.pad_sequences(X,maxlen=cut) #set to equal sent size
    return (X,Y), wrd2idx

def preprocess(path,pref):
    size_of_dataset=1000000
    dataframe = pd.read_csv(path, header=None, sep=',',
encoding='latin-1')
    dataset = dataframe.values
    X = dataset[:, 5]
    Y = dataset[:, 0]
    rand = list(range(len(dataset))) #shuffle data
    random.shuffle(rand)
    X = X[rand]
    Y = Y[rand]
    X = X[:size_of_dataset]
    Y = Y[:size_of_dataset]
    pd.DataFrame({'col1': X, "col2": Y}).to_csv(pref+'old_data.csv',
header=False, index=False) #save to csv before preproc

    X=gen_tokens(X)
    wrd2idx=gen_wrd2idx(X)

    for i, sent in enumerate(X):
        string=' '.join([word for word in sent])
        X[i]=string

    pd.DataFrame({'col1': X, "col2": Y}).to_csv(pref+'data.csv',
header=False, index=False,encoding='utf-8') #save ro csv after preproc

    return wrd2idx

if __name__ == "__main__":
```

```
    with open(word2idx_path,'w',encoding='utf-8') as out_file:
        json.dump(preprocess("data/training.csv",'train_'),out_file)
```

# ПРИЛОЖЕНИЕ Б

# ИСХОДНЫЙ КОД TRAIN_MODEL.PY

```
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
#os.environ['TF_FORCE_GPU_ALLOW_GROWTH '] = 'true'
os.system('del /S /Q /F log_dir\*')
from tensorflow.keras.layers import Dense, Dropout, Embedding, Flatten,
Conv1D, MaxPooling1D, LSTM, Bidirectional
from tensorflow.keras.models import Sequential

from tensorflow.keras import callbacks
from matplotlib import pyplot as plt
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from preprocess.preprocess import get_dataset

wrd2idx_path='preprocess/wrd2idx.txt'
pad=20
len_dict=30000
size_of_dataset=1000000
train=int(size_of_dataset*0.9)

(train_x, train_y), wrd2idx=get_dataset('preprocess/train_data.csv',
wrd2idx_path, len_of_dir=len_dict-1, cut=pad)
train_y=train_y.astype('float32')


with open("log_dir/metadata.tsv", 'w',encoding='utf-8') as f:
    f.write("Index\tLabel\n")
    for key, ind in wrd2idx.items():
        f.write("{}\t{}\n".format(key,ind))

tb=callbacks.TensorBoard(
    embeddings_freq=1,
    log_dir='log_dir',
    histogram_freq=3
)


def train_dense_model():
    model = Sequential()
    model.add(Embedding(len_dict, 8, input_length=pad,name='emb'))
    model.add(Flatten())
    model.add(Dense(160, activation="relu"))
    model.add(Dropout(0.3))
    model.add(Dense(120, activation="relu"))
```

```
    model.add(Dropout(0.3))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

    num_epochs = 3;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs,
batch_size=1024
                  , validation_data=(train_x[train:size_of_dataset],
train_y[train:size_of_dataset]), verbose=2,
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.3,
                          patience=3
                      )
                  ])

    model.save('model.h5')

    plot_history(h,num_epochs)


def train_cnn_lstm():
    model = Sequential()
    model.add(Embedding(len_dict, 8, input_length=pad, name='emb'))
    model.add(Dropout(0.25))

model.add(Conv1D(filters=48,kernel_size=5,strides=1,padding='valid',ac
tivation='relu'))
    model.add(MaxPooling1D(pool_size=4))
    model.add(LSTM(130, dropout=0.3))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

    num_epochs = 10;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs,
batch_size=512
                  , validation_data=(train_x[train:size_of_dataset],
train_y[train:size_of_dataset]), verbose=2,
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.3,
                          patience=6
                      )
                  ])
```

```
    plot_history(h,num_epochs)
    model.save('model_cnn_lstm.h5')

def train_2lstm():
    model = Sequential()
    model.add(Embedding(len_dict, 18, input_length=pad, name='emb'))
    model.add(LSTM(80, dropout=0.3,return_sequences=True))
    model.add(LSTM(150, dropout=0.3))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])

    num_epochs = 30;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs,
batch_size=1024
                  , validation_data=(train_x[train:size_of_dataset],
train_y[train:size_of_dataset]), verbose=2,
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.3,
                          patience=6
                      )
                  ])

    plot_history(h,num_epochs)
    model.save('model_2lstm.h5')

def train_bidir_rnn():
    model = Sequential()
    model.add(Embedding(len_dict, 18, input_length=pad, name='emb'))
    model.add(Dropout(0.3))
    model.add(Bidirectional(LSTM(90, dropout=0.5)))
    model.add(Dense(1, activation="sigmoid"))

    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

    num_epochs = 12;
    h = model.fit(train_x[:train], train_y[:train], epochs=num_epochs,
batch_size=1024
                  , validation_data=(train_x[train:size_of_dataset],
train_y[train:size_of_dataset]), verbose=2,
                  callbacks=[
                      callbacks.ReduceLROnPlateau(
                          monitor='val_loss',
                          factor=0.2,
                          patience=6
                      )])
```

```
    plot_history(h, num_epochs)
    model.save('model_bidir_lstm.h5')

def plot_history(h, epochs):
    plt.plot(range(epochs), h.history['val_accuracy'], 'b-',
label='val')
    plt.plot(range(epochs), h.history['accuracy'], 'b--', label='test')
    plt.title('test and val acc')
    plt.xlabel('epochs')
    plt.ylabel('acc')
    plt.show()

    plt.plot(range(epochs), h.history['val_loss'], 'b-', label='val')
    plt.plot(range(epochs), h.history['loss'], 'b--', label='test')
    plt.title('test and val loss')
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.show()


if __name__=='__main__':
    #train_cnn_lstm()
    pass
```

## ПРИЛОЖЕНИЕ В

## ИСХОДНЫЙ КОД PREDICT_USERS.PY

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
from tensorflow.keras.models import load_model
import numpy as np
from sys import argv
from preprocess.preprocess import indexing,gen_tokens
from tensorflow.keras.preprocessing import sequence
import json

class PredictSentiment(object):
    def __init__(self):
        self.cnn_lstm = load_model('model_cnn_lstm.h5')
        self.model_2lstm = load_model('model_2lstm.h5')
        self.fnn = load_model('model.h5')
        self.bi_cnn = load_model('model_bidir_lstm.h5')

    def predict(self,x):
        cnn_lstm_pred = self.cnn_lstm.predict(x)
        fnn_pred = self.fnn.predict(x)
        bi_cnn_pred = self.bi_cnn.predict(x)
        lstm_pred = self.model_2lstm.predict(x)


pred=0.3*lstm_pred+0.24*bi_cnn_pred+0.23*fnn_pred+0.23*cnn_lstm_pred

        for i, pr in enumerate(pred):
            if 0.0<=pr[0]<0.4:
                print('{}: negativ'.format(i + 1), end=' ')
            elif 0.4<=pr[0]<0.6:
                print('{}: neitral'.format(i + 1), end=' ')
            elif 0.6 <= pr[0] <= 1.0:
                print('{}: positiv'.format(i + 1), end=' ')

            print('({:.2f}=fnn: {:.2f} + cnn_lstm: {:.2f} + bi_lstm:
{:.2f} + 2lstm: {:.2f})'.format(pr[0], fnn_pred[i][0],

cnn_lstm_pred[i][0],

bi_cnn_pred[i][0],

lstm_pred[i][0]))
ensemble_predicter=PredictSentiment()
twits= argv[1:]

with open("preprocess/wrd2idx.txt", 'r') as json_file:
    wrd2idx = json.load(json_file)
wrd2idx = dict(list(wrd2idx.items())[:30000])
for i, nof in enumerate(twits):
```

```
    with open(nof,'r') as file:
        twits[i]=file.read()

twits=gen_tokens(twits)
print('tokens: \n', twits)

for i, sent in enumerate(twits):
    for j, word in enumerate(sent):
        if word in wrd2idx:
            twits[i][j]=wrd2idx[word]
        else:
            twits[i][j]=0

print('word indexing: \n', twits)
twits=sequence.pad_sequences(twits, maxlen=20)

ensemble_predicter.predict(twits)
```