МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

Индивидуальному домашнему заданию
по дисциплине «Искусственные нейронные сети»
Тема: «Sentiment140»

Студентка гр. 7383 _____ Иолшина В.
Студентка гр. 7383 _____ Прокопенко Н.
Студентка гр. 7383 _____ Чемова К.А.
Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

Цель.

В рамках выполнения ИДЗ необходимо разработать и реализовать модель ИНС решающую определенную задачу на заданном датасете.

Задачи.

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

Требования.

- 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*

Распределение ролей.

Чемова Ксения – препроцессинг датасета, улучшение сети

Прокопенко Надежда – построение нейронной сети

Иолшина Валерия – построение байесовского классификатора, тестирование сети

Описание датасета.

Датасет текста. Включает в себя набор твитов разной эмоциональной окраски, а также информацию о твите. Данные хранятся в .csv файлах. Был использован файл без смайликов training.1600000.processed.noemoticon.csv. В файле содержатся данные о твите, пользователе его написавшем, эмоциональной окраске твита.

В фале 6 полей:

- sentiment – оценка твита
- id
- date – дата публикации
- query_string – запрос
- user – пользователь
- text – тест твита

Sentiment может быть отрицательной (0), нейтральной (2), положительной (4). Если запроса нет, то в поле query_string стоит NO_QUERY. На рис. 1 показаны исходные данные.

	sentiment	id	date	query_string	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

Рисунок 1 – Пример исходных данных

Ход работы.

1. Начальный анализ текста

Сначала был обработан датасет для удобства работы. Поскольку оценка положительности зависит только от теста твита, то для обучения были выбраны только два поля sentiment и text, остальные были удалены. Далее текст твитов был токенизирован на слова с помощью `TweetTokenizer()`. Полученные данные были записаны в файл train_data.csv. На рис. 2 представлены обработанные данные.

	sentiment	text
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire
4	0	@nationwideclass no, it's not behaving at all....

Рисунок 2 – Обработанные данные

Для выбора метрики качества датасет был проверен на соотношение твитов разной оценки. Оно показало, что датасет содержит только положительные и отрицательные твиты причем в равном количестве:

```
df['sentiment'].value_counts()

4    800000
0    800000
```

Поэтому в качестве метрики выберем точность.

Код обработки текста представлен в приложении А.

2. Построение Байесовского классификатора

В качестве не относящегося к ИНС метода решения данной задачи был выбран наивный Байесовский классификатор. Сначала были разбиты исходные данные на тестовые и обучающие в соотношении 2/8, установив

SEED для воспроизводимости экспериментов. Далее были переписаны целевые показатели (target) в формат 0/1, где 1 соответствует 4 (позитивная окраска твита), а 0 – 0, т.е. негативной. Поскольку работать сеть может только с числами, то векторизируем слова. Вместо простого подсчета частоты встречаемости каждого слова была использована мера TF-IDF (Term Frequency Inverse Document Frequency). Это статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

Был создан байесовский классификатор с использованием библиотеки Scikit-learn. Был выбран `naive_bayes.MultinomialNB` поскольку он хорошо подходит для решения задач классификации текста.

```
# Создаём классификатор
bayes_classifier = naive_bayes.MultinomialNB()

# Обучим его
bayes_classifier.fit(train_x, train_y)
```

После обучения классификатора была оценена эффективность на тестовом наборе. Полученная точность составила 77%.

Код обучения Байесовского классификатора представлен в приложении Б.

3. Построение нейронной сети

Сначала были разбиты исходные данные на тестовые и обучающие в соотношении 2/8, установив SEED для воспроизводимости экспериментов. Далее были переписаны целевые показатели (target) в формат 0/1, где 1 соответствует 4 (позитивная окраска твита), а 0 – 0, т.е. негативной.

Создаем и обновляем внутренний словарь, после чего выравниваем все тексты до одного размера.

```
tokenizer = Tokenizer(oov_token="<unk>")
tokenizer.fit_on_texts(df_train['text'])

vocab_size = len(tokenizer.word_index) + 1

# Выровняем все тексты до одного размера
maxlen = 50

train_x = pad_sequences(tokenizer.texts_to_sequences(df_train.text),
maxlen=maxlen, padding="post")
test_x = pad_sequences(tokenizer.texts_to_sequences(df_test.text),
maxlen=maxlen, padding="post")
```

После этого была построена и обучена модель нейронной сети. Был выбран оптимизатор `Adam` поскольку он совмещает в себе лучшее от `RSMProp` и `Adagrad`.

Первым слоем был выбран `Embedding` поскольку превращает индексы в плотные вектора.

Для борьбы с переобучением и увеличением устойчивости сети третьим слоем был выбран слой `Dropout`. Последним слоем был выбран `Dense` с сигмоидной функцией активации для бинарной классификации.

Изначально в качестве второго слоя был выбран простой `GRU`, так как его скорость обучения выше, чем у `LSTM`. Была получена точность 68%. Это плохой показатель эффективности, на 10% уступающий наивному Байесовскому классификатору.

Для улучшения показателей была использована двунаправленная обертка bidirectional над `GRU`. Это улучшение тоже не дало сильного увеличения точности, которая составила всего 74%.

Вместо GRU был взят слой LSTM. Данный слой дает высокую эффективность в различных задачах. Простой слой LSTM дал точность 78%, которая сопоставима с наивным Байесовским классификатором (77%).

Так же как и у GRU для улучшения LSTM была использована bidirectional. Это позволило получить 88%, что значительно превосходит наивный Байесовский классификатор.

```
def build_model(max_len=maxlen, emb_dim=128, dropout_proba=0.1,
                gru_state_size=128, bidirectional=False):
    model = Sequential()
    model.add(Embedding(len(tokenizer.index_word) + 1, emb_dim,
trainable=True, input_length=max_len))
    if bidirectional:
        model.add(Bidirectional(LSTM(gru_state_size)))
    else:
        model.add(LSTM(gru_state_size))
    model.add(Dropout(dropout_proba))
    model.add(Dense(1, activation='sigmoid'))
    optimizer = Adam
    opt = optimizer(lr=1e-3)
    loss = "binary_crossentropy"
    model.compile(loss=loss, optimizer=opt, metrics=["accuracy"])
    return model
```

Полный код обучения нейронной сети представлен в приложении В.

4. Использование callback’ов

Был использован callback ReduceLROnPlateau, который уменьшает скорость обучения модели при выходе на плато. В качестве метрики мы используем val_acc, по умолчанию скорость обучения уменьшается в 10 раз.

Callback EarlyStoppin останавливает обучения и сохраняет лучшие веса, когда val_loss перестает уменьшаться.

Callback `keras.callbacks.TensorBoard` используется для логгирования модели.

```
callbacks = [ReduceLROnPlateau(monitor="val_acc"),  # Уменьшаем lr, если выходим на плато
             EarlyStopping(monitor="val_loss", restore_best_weights=True),  # Регуляризация -- метод ранней остановки
             keras.callbacks.TensorBoard(log_dir=logdir),  # Логгирование
             ]
```

5. Анализ наилучшей модели

Результирующая модель побила бейзлайн. Как часто бывает в нейронных сетях, ценой больших усилий мы можем немного побить не нейросетевые подходы. Точность на валидационных данных – 89%, на тестовых данных – 82% $\Rightarrow$ модель немного переобучились.

На рис. 3 и 4 показаны точность и ошибка результирующей модели. Графики получены при помощи Tensorboard.

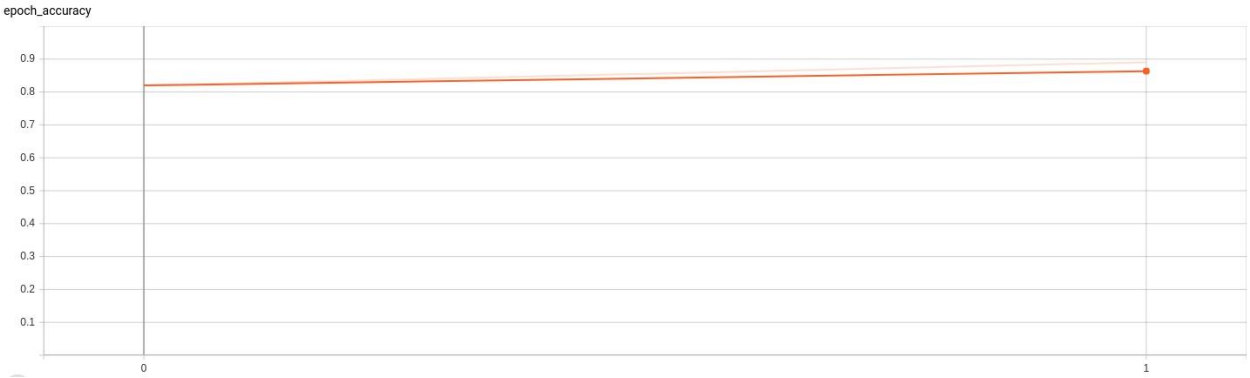

Рисунок 3 – Точность результирующей модели

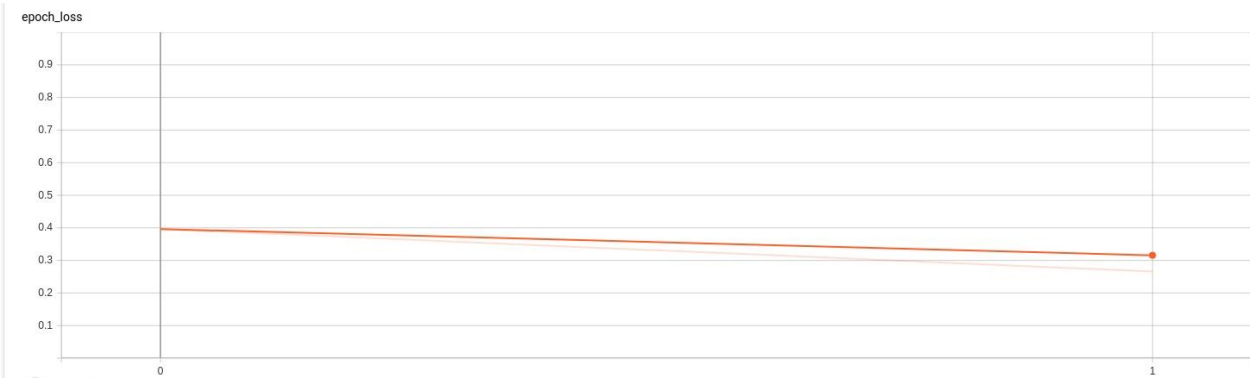

Рисунок 4 – Ошибка результирующей модели

6. Проблемы и их решение

На этапе подготовки данных была удалена информация, не имеющая к задаче отношения. На этапе обучения модель могла попасть в плато, следовательно, выдавала не самый лучший результат. Также модель начинала переобучаться. Эти проблемы были решены при помощи использования callback'ов.

Предложение по улучшению модели:

1) Стемминг и лемматизация, удаление стоп-слов (артикли, вспомогательные глаголы, упоминания (через знак собаки)) при предобработке текста
2) Сильнее регуляризировать модель для избавления от переобучения
3) Добавление слоя `Attention` для обработки последовательных данных с учётом контекста каждой метки времени. Он чаще всего применяется для работы с текстовой информацией. Одно из самых известных применений Attention это применение его для того, чтобы отказаться от рекуррентной сети и перейти к полносвязной модели. С помощью механизма Attention мы можем выстроить сеть способную к изучению последовательностей на базе полносвязной сети, обучить её на GPU, использовать droput.

Выводы.

В ходе выполнения индивидуального задания был обработан датасет Sentiment140. Для задачи классификации были использован Байесовский классификатор и построена и обучена рекуррентная нейронная сеть.

Байесовский классификатор, не учитывающий контекст и не являющийся лучшим для решения данной задачи, дал хороший показатель с точностью 77%. Этот показатель побила только результирующая модель нейронной сети с использованием bidirectional (двунаправленной сети). Остальные варианты сетей не превосходили данную точность.

Также, были описаны проблемы и их решения, которые возникли при подготовке датасета и обучении сети. Были представлены методы улучшения сети.

ПРИЛОЖЕНИЕ А

Preprocessing.py

```
import pandas as pd
from nltk.tokenize import TweetTokenizer

fname = f"training.1600000.processed.noemoticon.csv"

cols = ['sentiment', 'id', 'date', 'query_string', 'user', 'text']
df = pd.read_csv(fname, header=None, names=cols, encoding="latin-1")
df.head(5)

# Нас интересует колонка sentiment -- то, что мы предсказываем
(target)
# Проверим баланс классов
print(df['sentiment'].value_counts())

# Датасет сбалансирован --> значит хорошей метрикой качества
классификации будет accuracy

# Удалим колонки id, date, quety_string и user, т.к. они не
предоставляют релевантной информации
df.drop(['id','date','query_string','user'], axis='columns',
inplace=True)

# Токенезируем текст
tokenizer = TweetTokenizer()
df['text'] = [tokenizer.tokenize(t.lower()) for t in df['text']]
df.head(5)

# Запишем данные для дальнейшей работы
df.to_csv('train_data.csv', index=False)
```

ПРИЛОЖЕНИЕ Б

Bayesian approach.py

```
import numpy as np
from sklearn import naive_bayes
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

fname = f"train_data.csv"

cols = ['sentiment', 'text']
df = pd.read_csv(fname, header=None, names=cols, encoding="latin-1")

df.head(5)

# Установим random_seed для воспроизводимости экспериментов
SEED = 666
np.random.seed(SEED)

# Разобьём данные на train и test
df_train, df_test = train_test_split(df, test_size=0.2,
random_state=SEED)


# Перепишем target в формат 0, 1 <--> негативный, позитивный
train_y = np.array(list(map(lambda x: 1 if x == 4 else 0,
df_train['sentiment'])))
test_y = np.array(list(map(lambda x: 1 if x == 4 else 0,
df_test['sentiment'])))


# Векторизируем слова (мы не можем работать со строками, только с
числами)
# Будем использовать tf-idf
vectorizer = TfidfVectorizer(max_features=5000)

vectorizer.fit(df['text'])
train_x = vectorizer.transform(df_train['text'])
test_x = vectorizer.transform(df_test['text'])


# Создаём классификатор
bayes_classifier = naive_bayes.MultinomialNB()
# Обучим его
bayes_classifier.fit(train_x, train_y)

# Проверим качество
```

```
accuracy = accuracy_score(bayes_classifier.predict(test_x), test_y)
print("Качество работы наивного классификатора:
{}%".format(round(accuracy * 100)))
```

ПРИЛОЖЕНИЕ В

NN approach.py

```
import pandas as pd
import numpy as np
import keras
import os
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Bidirectional, LSTM, Dropout,
Dense
from keras.optimizers import Adam
from datetime import datetime
from keras.callbacks import ReduceLROnPlateau, EarlyStopping

fname = f"train_data.csv"

cols = ['sentiment', 'text']
df = pd.read_csv(fname, header=None, names=cols, encoding="latin-1")
df.head(5)

# Установим random_seed для воспроизводимости экспериментов
SEED = 666
np.random.seed(SEED)

# Разобьём данные на train и test
df_train, df_test = train_test_split(df, test_size=0.2,
random_state=SEED)

# Перепишем target в формат 0, 1 <--> негативный, позитивный
train_y = np.array(list(map(lambda x: 1 if x == 4 else 0,
df_train['sentiment'])))
test_y = np.array(list(map(lambda x: 1 if x == 4 else 0,
df_test['sentiment'])))

# Токенезируем текст
tokenizer = Tokenizer(oov_token="<unk>")
tokenizer.fit_on_texts(df_train['text'])

vocab_size = len(tokenizer.word_index) + 1

# Выровняем все тексты до одного размера
maxlen = 50
train_x = pad_sequences(tokenizer.texts_to_sequences(df_train.text),
maxlen=maxlen, padding="post")
test_x = pad_sequences(tokenizer.texts_to_sequences(df_test.text),
maxlen=maxlen, padding="post")
```

```
# Строим сеть
def build_model(max_len=maxlen, emb_dim=128, dropout_proba=0.1,
gru_state_size=128, bidirectional=False):

    model = Sequential()
    model.add(Embedding(len(tokenizer.index_word) + 1, emb_dim,
trainable=True, input_length=max_len))

    if bidirectional:
        model.add(Bidirectional(LSTM(gru_state_size)))
    else:
        model.add(LSTM(gru_state_size))

    model.add(Dropout(dropout_proba))
    model.add(Dense(1, activation='sigmoid'))

    optimizer = Adam
    opt = optimizer(lr=1e-3)
    loss = "binary_crossentropy"

    model.compile(loss=loss, optimizer=opt, metrics=["accuracy"])
    return model

# Обучим bidirectional модель
logdir = "logs/scalars/" + datetime.now().strftime("%Y%m%d-%H%M%S")

callbacks = [ReduceLROnPlateau(monitor="val_acc"),  # Уменьшаем lr,
если выходим на плато
             EarlyStopping(monitor="val_loss",
restore_best_weights=True),  # Регуляризация -- метод ранней остановки
             keras.callbacks.TensorBoard(log_dir=logdir),  #
Логгирование
             ]

model = build_model(bidirectional=True)
history = model.fit(train_x, train_y, epochs=2, batch_size=128,
callbacks=callbacks)

# Сохраним веса используя формат `checkpoint_path` format
checkpoint_path = "bidir.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

model.save_weights(checkpoint_path)

# Проверяем качество
_, acc = model.evaluate(test_x, test_y, verbose=2)
print("Точность на тестовом множестве: {:5.2f}%".format(round(100 *
acc)))
```