2021 Sentiment Analysis Reviews Project Reports

Muhammad Shiraz

January 2021
Information Systems Engineering Department
KOCAELİ UNIVERSITY

Table of contents

Introduction

Sentiment Analysis is the task of identifying whether the opinion expressed in a review is positive or negative or neutral in general, or about a given review. For example: "I love this paper and I think the previous team building strategy behind is worth exploring further" is a general positive text, and the review. and Sentiment Analysis also known as opinion mining refers to the identification, extraction and study of sentiment states by using natural language processing, text analysis, computational linguistics and biometrics.

Keywords: Sentiment Analysis, Reviews, Machine Learning, Supervised Text Classification, Data Visualization.

Datasets

In this project, we used one dataset. Let us now describe the datasets in more details:

1. ICLR 2020 Conference | OpenReview dataset using selenium crawler
   https://openreview.net/group?id=ICLR.cc/2020/Conference

   This set is a collection of paper reviews documents labeled with respect to their overall sentiment polarity (positive or negative or neutral). The set collected in 2020-Dec-20 and it contains 702 positive and 794 negative and 1038 neutral processed reviews. The reviews were preprocessed by the dataset editors so that each review is formatted as a plain tokenized text, containing no structured information that can imply on the polarity of the text (e.g., label – 0/1). The average review size (in words) is 40000. Although this set is considered as a user---generated content the language is relatively grammatically correct in most cases, probably because users are not restricted with the size of the text.

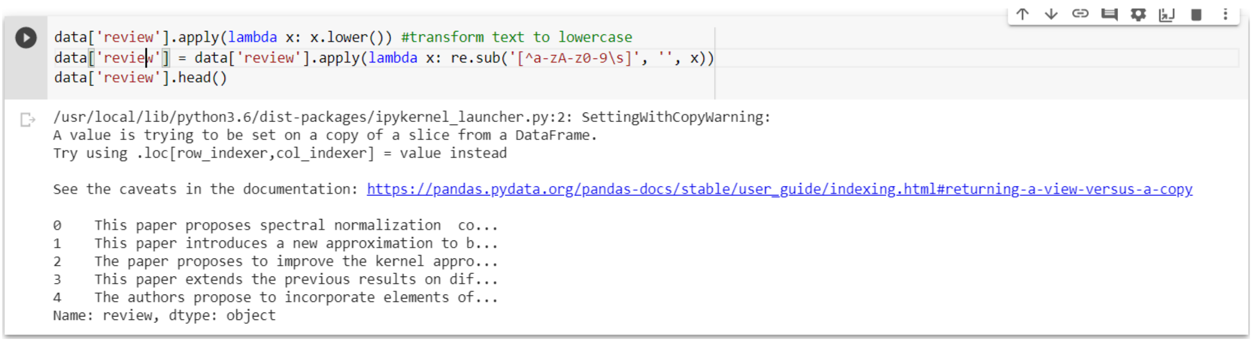
Review	**Sentiments**
I love this paper and I think the previous team building strategy behind is worth exploring further.	positive
The experimental results of the experiment are not made explicitly.	Negative
This paper can be strengthened by discussion and testing of other forms of distribution in the descriptive family.	Neutral

Table 1 --- Example of a multiple review's entry

Preprocessing

Before the feature extractor can use the review to build feature vector, the review text goes through preprocessing step where the following steps are taken. These steps convert plain text of the review into processable elements with more information added that can be utilized by feature extractor. For all these steps, third-party tools were used that were specialized to handle unique nature of review text.

Transform text to lowercase



```
data['review'].apply(lambda x: x.lower()) #transform text to lowercase
data['review'] = data['review'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x))
data['review'].head()
```

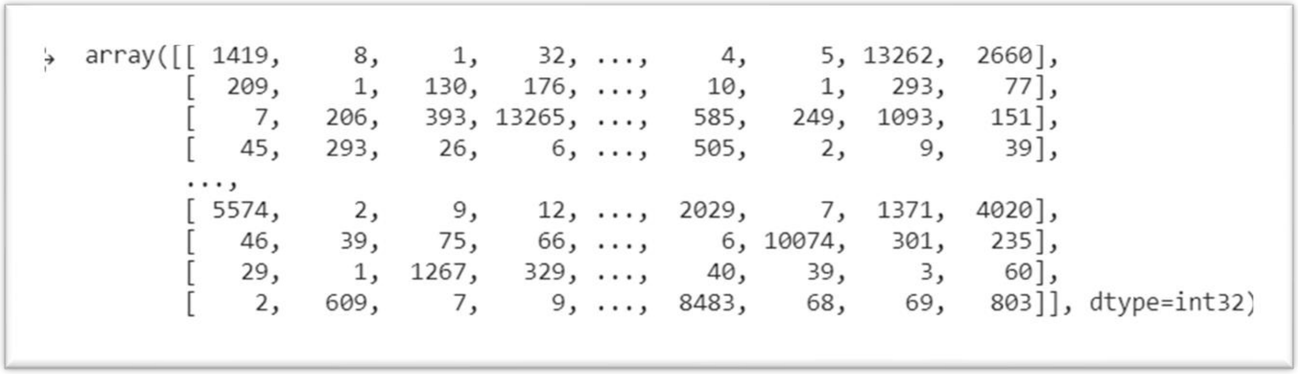
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

0    This paper proposes spectral normalization  co...
1    This paper introduces a new approximation to b...
2    The paper proposes to improve the kernel appro...
3    This paper extends the previous results on dif...
4    The authors propose to incorporate elements of...
Name: review, dtype: object
```

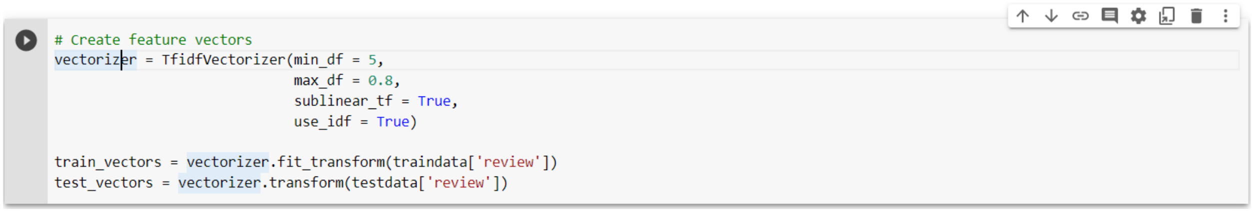
Tokenization

Tokenization is the process of converting text as a string into processable elements called tokens. In the context of a reviews, these elements can be words, emoticons, url links, hashtags or punctuations.



```
array([[ 1419,     8,     1,    32, ...,     4,     5, 13262,  2660],
       [  209,     1,   130,   176, ...,    10,     1,   293,    77],
       [    7,   206,   393, 13265, ...,   585,   249,  1093,   151],
       [   45,   293,    26,     6, ...,   505,     2,     9,    39],
       ...,
       [ 5574,     2,     9,    12, ...,  2029,     7,  1371,  4020],
       [   46,    39,    75,    66, ...,     6, 10074,   301,   235],
       [   29,     1,  1267,   329, ...,    40,    39,     3,    60],
       [    2,   609,     7,     9, ...,  8483,    68,    69,   803]], dtype=int32)
```

Vectorizing the data

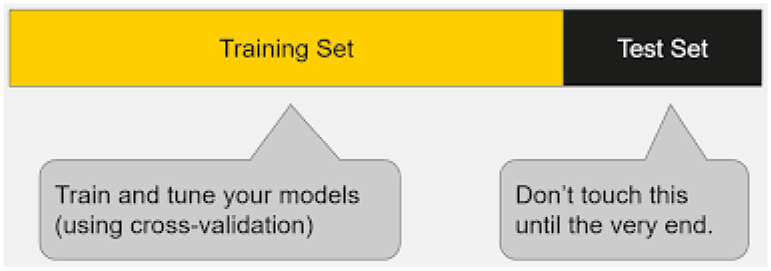


```
# Create feature vectors
vectorizer = TfidfVectorizer(min_df = 5,
                             max_df = 0.8,
                             sublinear_tf = True,
                             use_idf = True)

train_vectors = vectorizer.fit_transform(traindata['review'])
test_vectors = vectorizer.transform(testdata['review'])
```

Train Test Split

To measure the accuracy of the model we are creating, the data needs to split into 2 parts. A training set to fit and tune our model and a testing set to create predictions on and evaluate the model at the very end.


Training Set
Test Set
Train and tune your models (using cross-validation)
Don't touch this until the very end.

Classifiers:

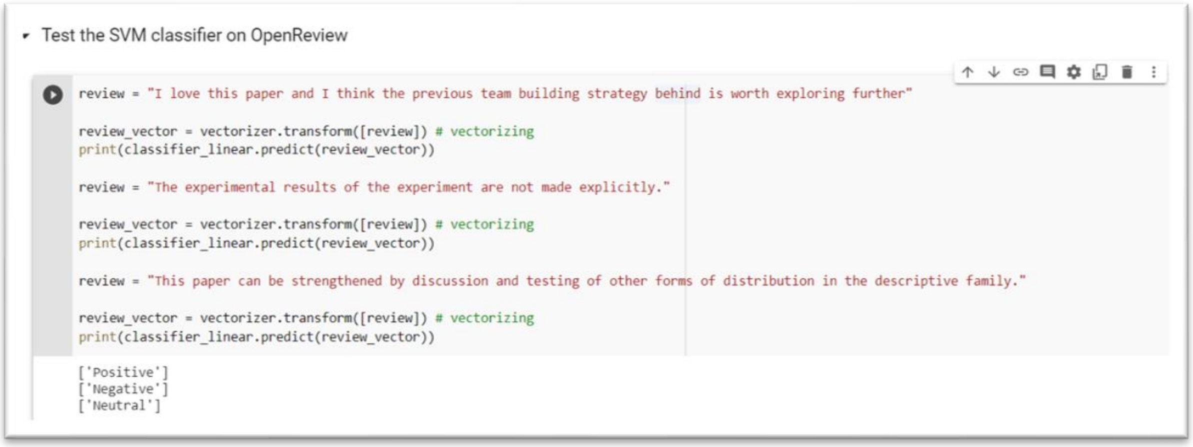
We used five different classifiers: RNN, LSTM, kNN, and SVM, ULMFit. Intuitively, the features seem mutually dependent.

1: Creating a Linear SVM Model

SVM is a supervised(feed-me) machine learning algorithm that can be used for both classification or regression challenges. Classification is predicting a label/group and Regression is predicting a continuous value. SVM performs classification by finding the hyper-plane that differentiate the classes we plotted in n-dimensional space. SVM draws that hyperplane by transforming our data with the help of mathematical functions called “Kernels”. Types of Kernels are linear, sigmoid, RBF, non-linear, polynomial, etc.,

Results:

precision	recall	f1-score	support
0.95	0.90	0.93	794
0.86	0.97	0.91	1038
0.97	0.85	0.91	702



```
review = "I love this paper and I think the previous team building strategy behind is worth exploring further"

review_vector = vectorizer.transform([review]) # vectorizing
print(classifier_linear.predict(review_vector))

review = "The experimental results of the experiment are not made explicitly."

review_vector = vectorizer.transform([review]) # vectorizing
print(classifier_linear.predict(review_vector))

review = "This paper can be strengthened by discussion and testing of other forms of distribution in the descriptive family."

review_vector = vectorizer.transform([review]) # vectorizing
print(classifier_linear.predict(review_vector))

['Positive']
['Negative']
['Neutral']
```

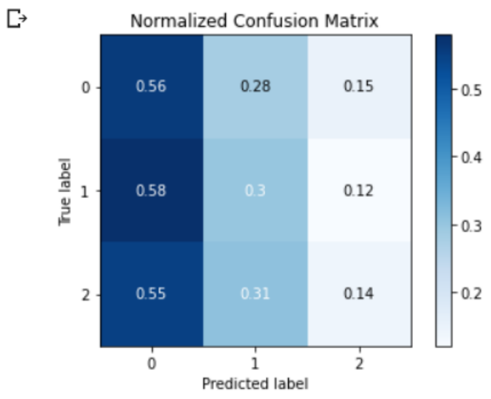
2: Multi-layer Perceptron

MLP is a supervised machine learning algorithm for regression or classification. Within the input and output, there might be some nonlinear layers, called hidden layers. Hence, MLP has the ability to learn nonlinear models. Each of the hidden layers comprises of neurons, which is the number of nodes in the hidden layer. MLP is sensitive to unscaled features, thus it might be important to scale the feature - [0, 1].

Results:

precision	recall	f1-score	support
0.35	0.39	0.37	241
0.40	0.41	0.40	316
0.24	0.30	0.27	204

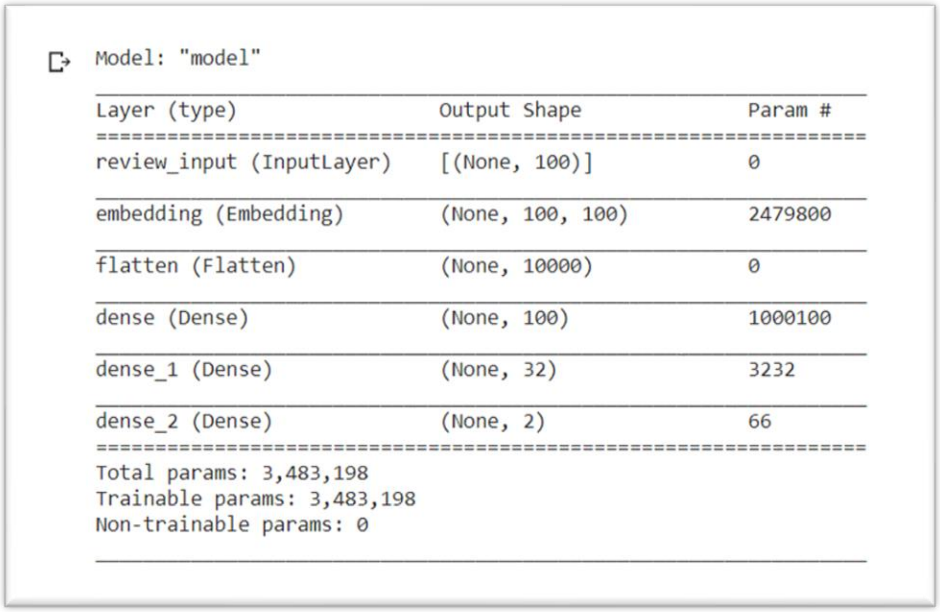
- **Confusion Matrix**



3: Word2Vec Pre-Trained Embeddings Conversion

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

<u>**Results:**</u>



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
review_input (InputLayer)    [(None, 100)]             0
_________________________________________________________________
embedding (Embedding)        (None, 100, 100)          2479800
_________________________________________________________________
flatten (Flatten)            (None, 10000)             0
_________________________________________________________________
dense (Dense)                (None, 100)               1000100
_________________________________________________________________
dense_1 (Dense)              (None, 32)                3232
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 66
=================================================================
Total params: 3,483,198
Trainable params: 3,483,198
Non-trainable params: 0
_________________________________________________________________
```

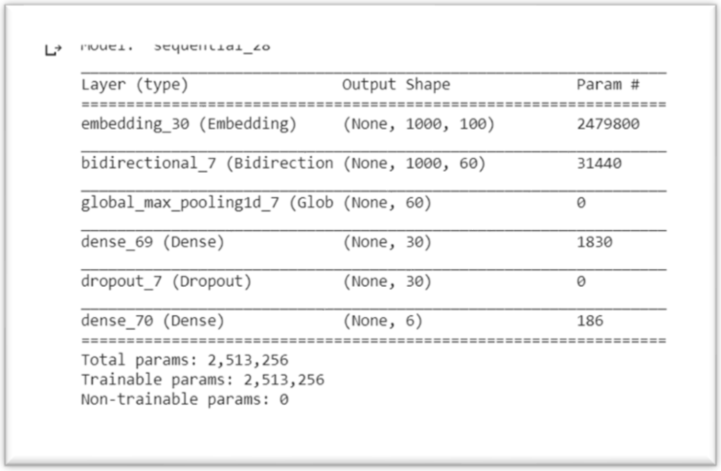
4: GloVe Pre-trained Word Embeddings

The training data is not so large, the model might not be able to learn good embeddings for the sentiment analysis. Alternatively, we can load pre-trained word embeddings built on a much larger training data.

The GloVe database contains multiple pre-trained word embeddings, and more specific embeddings trained on tweets. So, this might be useful for the task at hand.

First, we put the word embeddings in a dictionary where the keys are the words and the values the word embeddings.

Results:

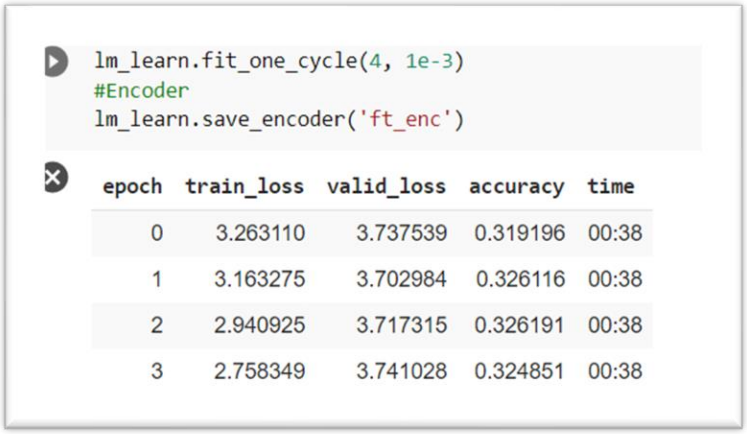


```
Model: sequential_28
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_30 (Embedding)     (None, 1000, 100)         2479800
_________________________________________________________________
bidirectional_7 (Bidirection (None, 1000, 60)          31440
_________________________________________________________________
global_max_pooling1d_7 (Glob (None, 60)                0
_________________________________________________________________
dense_69 (Dense)             (None, 30)                1830
_________________________________________________________________
dropout_7 (Dropout)          (None, 30)                0
_________________________________________________________________
dense_70 (Dense)             (None, 6)                 186
=================================================================
Total params: 2,513,256
Trainable params: 2,513,256
Non-trainable params: 0
```

5: ULMFit classification model

ULMFiT, is an architecture and transfer learning method that can be applied to NLP tasks. It involves a 3-layer AWD-LSTM architecture for its representations. The training consists of three steps: 1) general language model pre-training on a Wikipedia-based text, 2) fine-tuning the language model on a target task, and 3) fine-tuning the classifier on the target task.

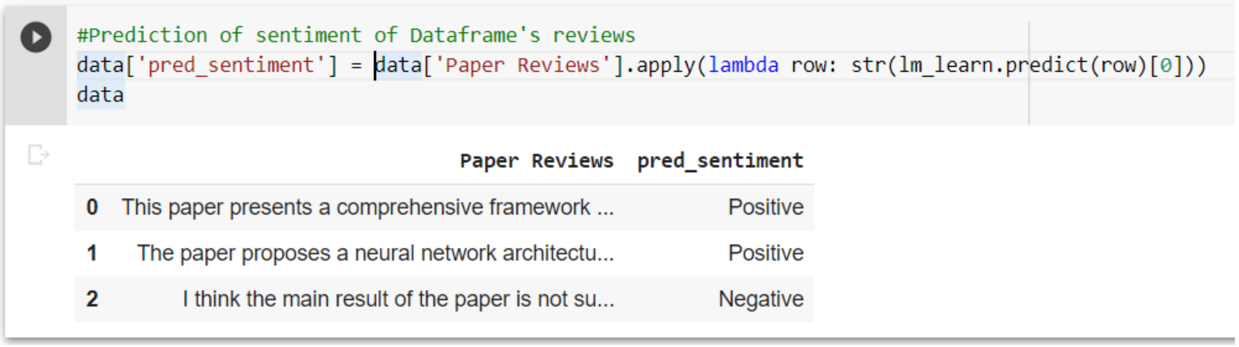
Results:



```
lm_learn.fit_one_cycle(4, 1e-3)
#Encoder
lm_learn.save_encoder('ft_enc')
```

epoch	train_loss	valid_loss	accuracy	time
0	3.263110	3.737539	0.319196	00:38
1	3.163275	3.702984	0.326116	00:38
2	2.940925	3.717315	0.326191	00:38
3	2.758349	3.741028	0.324851	00:38

- **Prediction of sentiment of Dataframe's reviews**

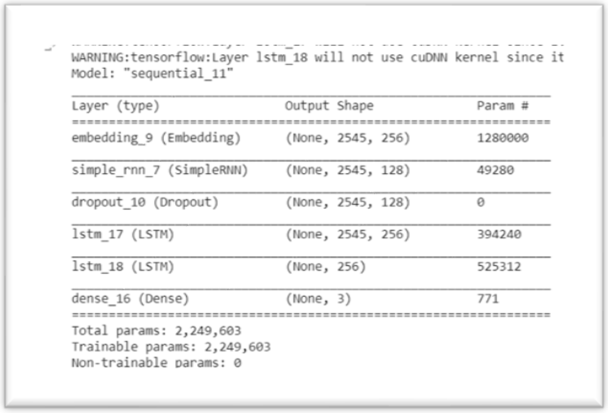


```
#Prediction of sentiment of Dataframe's reviews
data['pred_sentiment'] = data['Paper Reviews'].apply(lambda row: str(lm_learn.predict(row)[0]))
data
```

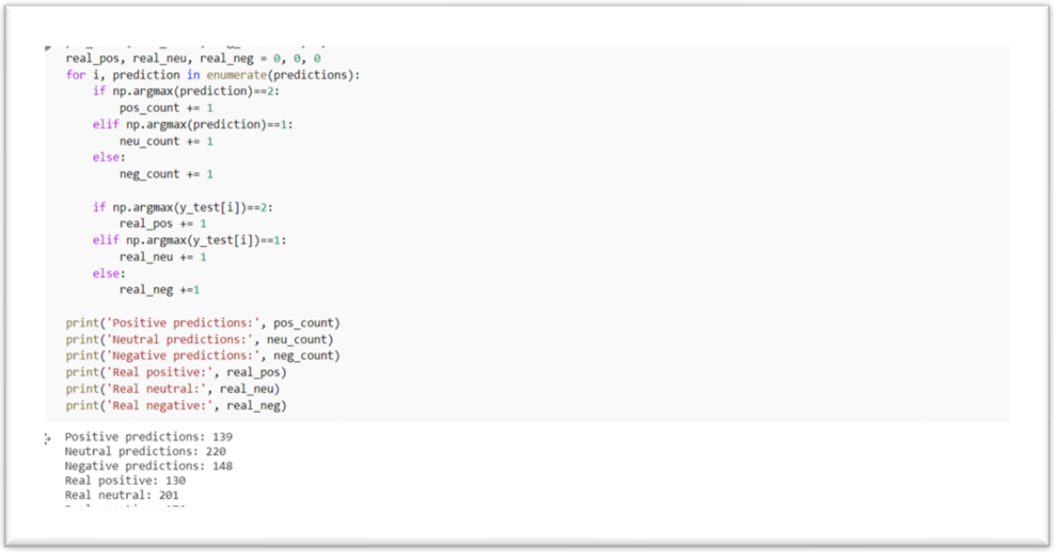
	Paper Reviews	pred_sentiment
0	This paper presents a comprehensive framework ...	Positive
1	The paper proposes a neural network architectu...	Positive
2	I think the main result of the paper is not su...	Negative

6: Creating model RNN and LSTM

Sentiment Analysis using Recurrent Neural Networks (RNN-LSTM) and Google News Word2Vec



```
WARNING:tensorflow:Layer lstm_18 will not use cuDNN kernel since it
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 2545, 256)         1280000
_________________________________________________________________
simple_rnn_7 (SimpleRNN)     (None, 2545, 128)         49280
_________________________________________________________________
dropout_10 (Dropout)         (None, 2545, 128)         0
_________________________________________________________________
lstm_17 (LSTM)               (None, 2545, 256)         394240
_________________________________________________________________
lstm_18 (LSTM)               (None, 256)               525312
_________________________________________________________________
dense_16 (Dense)             (None, 3)                 771
=================================================================
Total params: 2,249,603
Trainable params: 2,249,603
Non-trainable params: 0
```



```
real_pos, real_neu, real_neg = 0, 0, 0
for i, prediction in enumerate(predictions):
    if np.argmax(prediction)==2:
        pos_count += 1
    elif np.argmax(prediction)==1:
        neu_count += 1
    else:
        neg_count += 1

    if np.argmax(y_test[i])==2:
        real_pos += 1
    elif np.argmax(y_test[i])==1:
        real_neu += 1
    else:
        real_neg +=1

print('Positive predictions:', pos_count)
print('Neutral predictions:', neu_count)
print('Negative predictions:', neg_count)
print('Real positive:', real_pos)
print('Real neutral:', real_neu)
print('Real negative:', real_neg)
```

```
Positive predictions: 139
Neutral predictions: 220
Negative predictions: 148
Real positive: 130
Real neutral: 201
```

Conclusions

In this project we have investigated the task of sentiment analysis as a classification problem. We have used one dataset: iclr2020 paper reviews and reviews statuses that differ mostly in their typical length of each document, and the language style. We believe we have shown that using simple word-based features, it is harder to classify reviews statuses, comparing to iclr2020.