Assignment 2

Introduction

Sentiment analysis has become a broadly investigated research area with the recently growing attention from different research communities. Due to a rapidly increasing number of users, Twitter is one of the most appropriate virtual environments for data monitoring and tracking. It contains a massive amount of data, which could be converted into helpful information by applying sentiment analysis.

Preprocessing

Regular language is already extremely complicated, requiring a great deal of preprocessing. This is exaggerated on social media; people usually convey their opinions in an informal way, rendering typical methods useless. The language in Twitter differs from the language used in other settings. The language data can have a lot of noise such as elongated words, emojis, laughs, etc. For this reason, the first step is preprocessing, converting tweets into a particular form which is easier to analyse.

The following steps have been performed during preprocessing:

1. Replacing emojis, expressing happiness, with the word “posemojy”
:) :] :3 :> 8) (: =) =] :’) :-)

2. Replacing emojis, expressing sadness, with the word “negemojy”
:(:[:< 8( ): =(=[:’(:-(

3. Replacing user mentions (with @ at the beginning) with the word “usermention”

4. Replacing laughing e.g. haha, ahaha, lol… with the word “laughintweet”

5. Replacing urls with the word “urlintweet”

6. Deleting all the numbers and words that include numbers

7. Changing elongated words to no more than 2 same letters (for example, Hmmmm -> Hmm)

8. Deleting non-alphanumeric characters

9. Deleting one-letter words

10. Replacing words, expressing negations, with a special word “negationintweet”.

11. All words were converted into lower case.

For tokenisation, TweetTokenizer was used. Words written with non-English symbols were ignored.

WordNetLemmatizer was used for reducing the size of vocabulary. Before it, POS tagging (nltk.tag.pos_tag) was performed to help the lemmatizer. For the purposes of this notebook only nouns and verbs were given their non-default tags. All other words were considered as adjectives.

Feature extraction

For feature extraction both CountVectorizer and TFIDF from sklearn were used separately. By this, I divided my work into two parts. In both cases, after embedding, the matrices were reduced by applying X2 testing (chi-squared testing) in order to find 500 important features. This number of features was chosen after cross-validation.

Three additional features were added: the number of positive words, the number of negative words and the number of bad words occurred in each tweet. The presence of these types of words could be highly correlated with the target label. The sets of positive and negative words were downloaded from https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html. The set of bad words - https://www.cs.cmu.edu/~biglou/resources/. However, I additionally uploaded them on on Figshare.

The dimension of the final feature vector is 503. 500 words chosen by chi-squared testing and 3 features - “PositiveWords”,”NegativeWords”,”BadWords”.

Classifiers

Initially, I used a large set of classifiers, but Multinomial Naive Bayes, Gaussian Naive Bayes, and Passive Aggressive Classifier showed the best accuracies. The following tables show the performance, for each of these classifiers, according to accuracy, F1 score, and macro averaged F1 score.

Accuracy

Test	Multinomial Naive Bayes		Gaussian Naive Bayes		Passive Aggressive Classifier	
	CV	TFIDF	CV	TFIDF	CV	TFIDF
1	0.629	0.6182	0.4594	0.4721	0.6497	0.5817
2	0.6357	0.6141	0.5364	0.5532	0.633	0.6104
3	0.5956	0.5851	0.4704	0.4834	0.5948	0.5976

F1 score

Test	Multinomial Naive Bayes		Gaussian Naive Bayes		Passive Aggressive Classifier	
	CV	TFIDF	CV	TFIDF	CV	TFIDF
1	0.6194	0.5995	0.3519	0.3817	0.6304	0.5621
2	0.6355	0.6077	0.4183	0.4652	0.6302	0.6009
3	0.5914	0.5708	0.3742	0.4071	0.5734	0.5757

Macroaveraged F1 score

Test	Multinomial Naive Bayes		Gaussian Naive Bayes		Passive Aggressive Classifier	
	CV	TFIDF	CV	TFIDF	CV	TFIDF
1	0.513	0.457	0.433	0.507	0.471	0.442
2	0.546	0.467	0.476	0.546	0.522	0.504
3	0.493	0.436	0.442	0.494	0.423	0.464

The tables show that Multinomial Naive Bayes is the best choice among other classifiers.

Glove Word Embedding

glove.6B.100d.txt contains a 100-dimensional version of the embedding. It presents a word followed by the weights (100 numbers) on each line. Keras provides a Tokenizer class that can be fit on the training data, converts text to sequences (*texts_to_sequences()* method) and provides a *word_index* attribute.

Categorical cross-entropy was used as loss with the rmsprop optimizer. After testing the softmax, tanh, and sigmoid activation functions, I found that sigmoid provided the best results.

The model layers can be presented as following:

Embedding(5000, 100) -> LSTM(100) -> Dense(3, activation = sigmoid)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 50, 100)           500000
_________________________________________________________________
lstm (LSTM)                  (None, 100)               80400
_________________________________________________________________
dense (Dense)                (None, 3)                 303
=================================================================
Total params: 580,703
Trainable params: 80,703
Non-trainable params: 500,000
```

```
Epoch 1/16
352/352 [==============================] - 75s 214ms/step - loss: 0.9016 - accuracy: 0.5561
Epoch 2/16
352/352 [==============================] - 69s 196ms/step - loss: 0.8550 - accuracy: 0.5947
Epoch 3/16
352/352 [==============================] - 64s 182ms/step - loss: 0.8317 - accuracy: 0.6064
Epoch 4/16
352/352 [==============================] - 74s 209ms/step - loss: 0.8125 - accuracy: 0.6198
Epoch 5/16
352/352 [==============================] - 59s 167ms/step - loss: 0.7987 - accuracy: 0.6282
Epoch 6/16
352/352 [==============================] - 59s 167ms/step - loss: 0.7855 - accuracy: 0.6350
Epoch 7/16
352/352 [==============================] - 59s 167ms/step - loss: 0.7745 - accuracy: 0.6421
Epoch 8/16
352/352 [==============================] - 61s 172ms/step - loss: 0.7636 - accuracy: 0.6458
Epoch 9/16
352/352 [==============================] - 60s 170ms/step - loss: 0.7543 - accuracy: 0.6535
Epoch 10/16
352/352 [==============================] - 59s 168ms/step - loss: 0.7449 - accuracy: 0.6572
Epoch 11/16
352/352 [==============================] - 60s 169ms/step - loss: 0.7361 - accuracy: 0.6624
Epoch 12/16
352/352 [==============================] - 59s 169ms/step - loss: 0.7274 - accuracy: 0.6690
Epoch 13/16
352/352 [==============================] - 59s 167ms/step - loss: 0.7188 - accuracy: 0.6742
Epoch 14/16
352/352 [==============================] - 58s 165ms/step - loss: 0.7110 - accuracy: 0.6759
Epoch 15/16
352/352 [==============================] - 59s 168ms/step - loss: 0.7033 - accuracy: 0.6805
Epoch 16/16
352/352 [==============================] - 59s 168ms/step - loss: 0.6963 - accuracy: 0.6858
```

The following table shows the performance according to accuracy and macro averaged F1 score for epoch = 16.

Test	Accuracy	Macro averaged F1 score
1	0.65	0.521
2	0.65	0.527
3	0.62	0.498

Future Improvements

The data provided for training is heavily biased toward neutral tweets. This was not handled in this work. However, it causes some problems and should be improved.
It is also worth mentioning that for further work it may be useful to create a special processing for hashtags and shortened words (e.g. government = govt; both of them were selected as important). For lexical analysis, according to our selected features, more attention should be paid to special slang, politics and religion related words.

Code

To evaluate my code on additional data, the file locations of testsets should be entered into the list “filenames”. Then, the whole notebook should be run, with the results being output in the final two cells. I also use evaluation.py from skeleton.