CS918 Exercise2 - Report

Nan Liu
Nan.Liu.1@warwick.ac.uk
9th December 2018

1 Introduction

This report describes the process of building sentiment classifiers for Twitter, with a brief explanation of the methodologies used in every stage.

2 Programming

There are seven libraries used in the whole program: **pandas**(create DataFrame to store and process the data), **numpy**(calculate the array and matrix used in exercise), **re**(match and remove certain characters), **nltk**(lemmatize the word, use stopwords to preprocess the text and sentiwordnet to extract the lexicon score), **time**(record the run time of the code), **sklearn**(extract the feature bag of word using CountVectorizer and set up models for classification) and **genism**(extract the feature word embeddings using Word2Vec). The exercise consists of three parts, which will be introduced below.

2.1 Text Preprocessing

The first step of building sentiment classifiers is to preprocess the text which includes replacing some particularities of tweets, removing certain characters and lemmatizing all words. Preprocessing code developed for exercise 1 is reused here.

To parse and clean the texts, **re.compile()** and **re.sub()** are called to define the patterns and replace them with spaces or certain words respectively. Text preprocessing includes the following sections:

- Lowercase all word in the text.
- Replace @user_mentions with string 'USERMENTION'.
- Replace URLs with string 'URLLINK'.
- Remove non-alphanumeric characters except spaces.
- Remove stop words which are meaningless in the text. Here, stopwords, list of stop words in nltk.corpus, is used to compare.
- Remove duplicate characters. For example, 'coooooool' will become 'col' and 'looooooooove' will become 'love' after preprocessing.
- Remove numbers fully made of digits (12) and word including digits (12th).
- Remove words with 2 characters or less. Because some 3-character words are meaningful in the text.
- Lemmatize all word in the text. Here, nltk.pos_tag() is not used which would have impact on feature extraction.

2.2 Feature Extraction and Classifier Training

The second step is to extract features for different classifiers, before training a sentiment classifier. They will be introduced together in this part 2.2.

2.2.1 Classifier 1: Ngram + Naïve Bayes

CountVectorizer() is used for extracting feature unigram and bigram which is helpful in converting a collection of text documents to a matrix of token counts. Parameter *ngram_range* is to select unigram and bigram, *max_features* is to build a vocabulary that consider the top 15000 features ordered by frequency, *stop_words* is to use a built-in stop word list for English. When choosing the method of classification, naïve bayes, logistic regression and SVM are all tried to compare the F1 score. Their performances are not much different. Naïve Bayes (**MultinomialNB()**) is chosen just for diversity of classifiers in this report. While **Pipeline()** can be used to assemble feature extraction and classification by defining a list of transforms and a final estimator. So, it is used to fit the data directly.

2.2.2 Classifier 2: Word2Vec + Logistic Regression

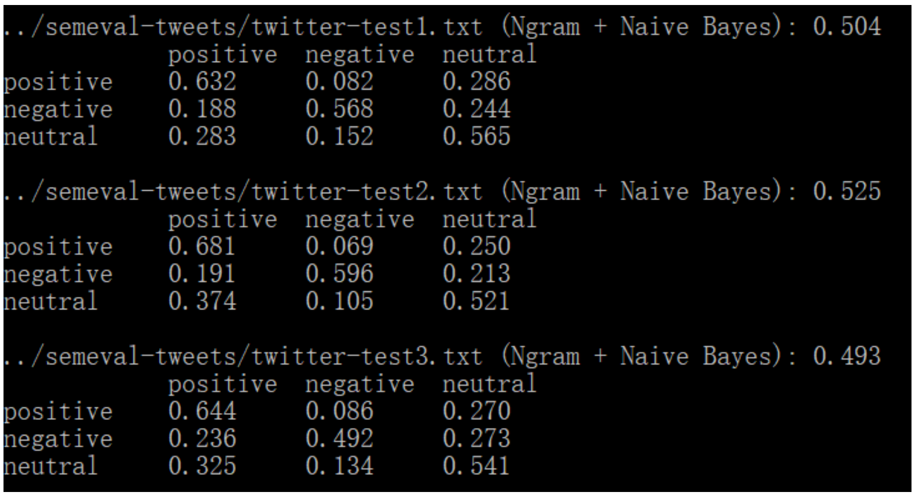
Google's pre-trained Word2Vec model is used for feature extraction (downloaded in https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit) in classifier 2. Vector of every word can be obtained directly through this model. So, the key is how to get the vector of the whole sentence. Here, **get_mean()** is applied to get mean vector of a sentence, and **get_meanvec()** is applied to get an array of all sentences' vectors in given data which is also seen as a extracted feature. But it exist NaN in the array, so **np.nan_to_num()** is needed to apply for the next step. Logistic regression (**LogisticRegression()**) is chosen to apply because of its best performance among three classifiers. As for the parameter, '*newton-cg*' is chosen for solver and '*multinomial*' is chosen for multi_class.

2.2.3 Classifier 3: Lexicon + Logistic Regression

As for the feature lexicon, *sentiwordnet* in *nltk.corpus* can be used to get the negative score, positive score and neutral score of given word. **word_score()** is applied to get the average score of a word. The arrange of score is -1 to 1, which means if it is negative, assign value -1; if it is positive, assign value 1; otherwise, assign value 0. So, the score of a word can be obtained in this way. But the **pos_tag()** is also not used here, so maybe it influence the score of the word. And **data_score()** get a mean score of the whole sentences, which is an extracted feature for classifier 3. Logistic regression (**LogisticRegression()**) is chosen to apply for the same reason, and its default parameters are good for the results.

2.3 Evaluation and Classifier Testing

Provide evaluation script is imported and used to evaluate the classifier above. The F1 score and confusion matrix of every model is showed as below.



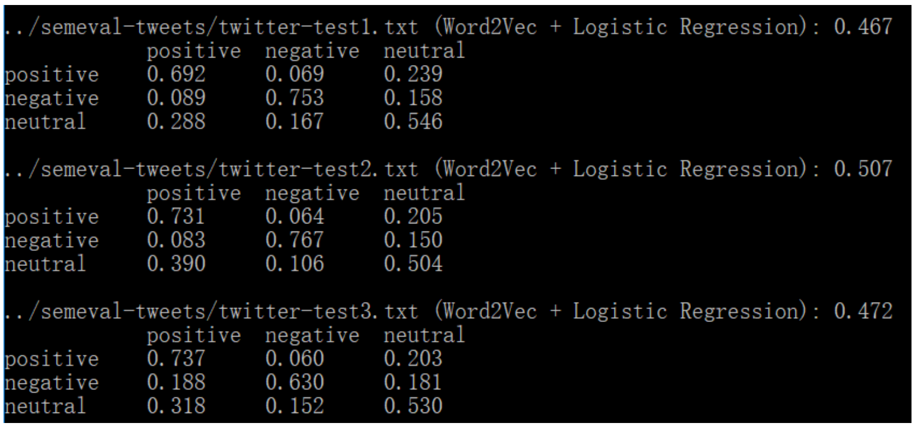
```
../semeval-tweets/twitter-test1.txt (Ngram + Naive Bayes): 0.504
            positive  negative  neutral
positive    0.632     0.082     0.286
negative    0.188     0.568     0.244
neutral     0.283     0.152     0.565

../semeval-tweets/twitter-test2.txt (Ngram + Naive Bayes): 0.525
            positive  negative  neutral
positive    0.681     0.069     0.250
negative    0.191     0.596     0.213
neutral     0.374     0.105     0.521

../semeval-tweets/twitter-test3.txt (Ngram + Naive Bayes): 0.493
            positive  negative  neutral
positive    0.644     0.086     0.270
negative    0.236     0.492     0.273
neutral     0.325     0.134     0.541
```

Figure 1. Evaluation of Classifier 1: Ngram + Naive Bayes

The result of classifier 1 is shown as Figure 1, with an accuracy of around 58%, and the true position rate for positive texts is around 65%, while the true position rate for negative and neutral texts is around 55%.



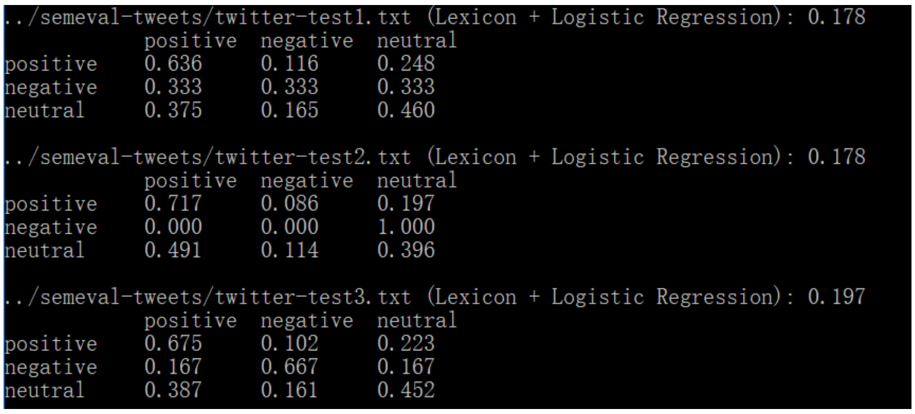
```
../semeval-tweets/twitter-test1.txt (Word2Vec + Logistic Regression): 0.467
            positive  negative  neutral
positive    0.692     0.069     0.239
negative    0.089     0.753     0.158
neutral     0.288     0.167     0.546

../semeval-tweets/twitter-test2.txt (Word2Vec + Logistic Regression): 0.507
            positive  negative  neutral
positive    0.731     0.064     0.205
negative    0.083     0.767     0.150
neutral     0.390     0.106     0.504

../semeval-tweets/twitter-test3.txt (Word2Vec + Logistic Regression): 0.472
            positive  negative  neutral
positive    0.737     0.060     0.203
negative    0.188     0.630     0.181
neutral     0.318     0.152     0.530
```

Figure 2. Evaluation of Classifier 2: Word2Vec + Logistic Regression

The result of classifier 2 is shown as Figure 2, with an accuracy of around 66%, and the true position rate for positive and negative texts is around 70%, while the true position rate for neutral texts is around 52%.



```
../semeval-tweets/twitter-test1.txt (Lexicon + Logistic Regression): 0.178
            positive  negative  neutral
positive    0.636     0.116     0.248
negative    0.333     0.333     0.333
neutral     0.375     0.165     0.460

../semeval-tweets/twitter-test2.txt (Lexicon + Logistic Regression): 0.178
            positive  negative  neutral
positive    0.717     0.086     0.197
negative    0.000     0.000     1.000
neutral     0.491     0.114     0.396

../semeval-tweets/twitter-test3.txt (Lexicon + Logistic Regression): 0.197
            positive  negative  neutral
positive    0.675     0.102     0.223
negative    0.167     0.667     0.167
neutral     0.387     0.161     0.452
```

Figure 3. Evaluation of Classifier 3: Lexicon + Logistic Regression

The result of classifier 3 is shown as Figure 3, with an accuracy of around 50%, and the true position rate for positive texts is around 67%, while the true position rate for neutral texts is around 43%. In fact, the accuracy of this classifier is not low, but the f1 score is very low compared with other two models. It is due to the high recall rate especially in test2, all negative tweets are predicted to be neutral.

3 Conclusion

In conclusion, the first classifier: Ngram + Naïve Bayes performs best among these classifiers. This may be because the sentiment analysis needs a complicated method to keep as much feature as possible to be accurate. On the other hand, the F1 score of the second classifier: Word2Vec + Logistic Regression is close to the first one, but when using the Word2Vec model trained by myself, the score and the accuracy are much smaller than this. So, the Word2Vec model is very important for sentiment analysis. However, the performance of the third classifier: Lexicon + Logistic Regression is much worse than the first two. Maybe the method for extracting feature 'lexicon' is not suitable, for the score of word not in the sentiwordnet is assigned value 0 when processing.

The whole program takes about 300 seconds. After completing this exercise, I have acknowledged a better understanding of sentiment analysis for Twitter, including text preprocessing, feature extraction, model training and evaluation, with the skills of processing natural language using Python and related libraries.