CS918

Exercise 2

1853897

1. **Pre-process**

This part contains four stages which are loading data, using regular expressions , lemmatization and removing stop words.

Firstly, using *pd.read_csv()* can easilt load data.

Secondly, using regular expression can finish most pre-process operation.

a. *re.sub(r'[A-Za-z]+://[^\s]*', 'URLLINK',text)* can turn all URLs into *'URLLINK'*

b. *re.sub(r'(@[A-Za-z0-9_]+)','USERMENTION',text)* can replace all usermentions

c. *re.sub(r'(#[A-Za-z0-9_]+)','HASHTAG',text)* can replace all hashtags

d. *re.sub(r"n\'t", " not", text) and re.sub(r"won't", "will not", text)* can restore all negative abbreviations

e. *re.sub(r'(.)\1+',r'\1\1',text)* can remove elongated words

f. *re.sub(r"[:|;][\-|\^|o|\']?[d|\)|\>|\]|*]+|=d|lol|:\'\-\)","positive",text)* can turn all smileys(like *:)))))))) :) :-) :-)) :^) LOL :o) :'-) =D :-D :] ;> :-**) into "positive"

g. *re.sub(r"[:|;][\'|\-]?[\(]+|:\'\-\(","negative",text)* can turn all frowns(like *:(:'(:-(:'-(*) into "negative"

h. *re.sub("[^a-zA-Z]", " ",text) and re.sub(" \w{1} ", " ",text)* can remove non-alphanumeric characters and one character

Thirdly, apply lemmatize to every word in twitter

Finally, remove stop words, because they are meaningless, like will, should, I.

In this part, I did not extract all uppercase and punctuations like !!!, because I think they will not affect the sentiment of twitter. They only enhance the emotions.

2. **Feature extract**

In this program, I successfully used two methods to extract features. One is ngram+tf-idf, the other is lexicon. Actually, I tried word2vec, but unfortunately, the classifier classified all twitters into neutral. I have to give up this method and you can see this method in my code.

For ngram+tf-idf, firstly we need produce unigrams and bigrams. On account of the operation speed, I used CountVectorizer() to extract unigrams and bigrams. After getting unigrams and bigrams, I calculated tf-idf value on training set. The feature of this method is based on tf-idf on unigrams and bigrams, the feature number is 10000. CountVectorizer() is used to extract features, and then TfidfTransformer() is used to calculate the weight of features.

For lexicon, firstly I download opinion-lexicon-English. This file contains positive words list and negative words list. By using these two lists, I calculated the number of positive words and negative words. The remaining words are neutral words. So I got a list which records the number of positive words, neutral words and negative words in the corpus. This list is the feature I need.

For the failed word2vec, I defined a model which contains 300 features for each twitter and then calculate the average word vector which is the feature for training process. However, this method did not work.

3. Train model

In this part, I will introduce the models I used for training. One is SVM, the other is Multinomial.

SVM is a supervised learning model and related learning algorithm for data analysis in classification and regression analysis. It has been widely used in text categorization. It usually achieves higher accuracy, although sometimes takes more time to run.

Bayesian classifier has three types, respectively is Multinomial Naive Bayes, the Binarized Multinomial Naive Bayes and Bernoulli Naive Bayes. The Multinomial Naive Bayes classifier is mainly used for the theme of the text classification.

4. Prediction

After getting model, I write code to measure the accuracy of prediction and use evaluation.py to calculate the f1 score and confuse metrixs.

We can see, the accuracy of model using Lexicon and SVM in testsets is 59%, 57% and 57%. Its f1 score is 0.234, 0.25 and 0.26.

```
Training Lexicon_SVM
0.5943342776203966
twitter-test1.txt (Lexicon_SVM): 0.234
            positive  negative  neutral
positive    0.426     0.161     0.413
negative    0.406     0.134     0.460
neutral     0.411     0.158     0.430

['neutral' 'neutral' 'neutral' ... 'neutral' 'neutral' 'neutral']
0.576133909287257
twitter-test2.txt (Lexicon_SVM): 0.250
            positive  negative  neutral
positive    0.516     0.119     0.365
negative    0.460     0.080     0.460
neutral     0.548     0.105     0.347

['positive' 'positive' 'positive' ... 'neutral' 'neutral' 'positive']
0.5769554247266611
twitter-test3.txt (Lexicon_SVM): 0.260
            positive  negative  neutral
positive    0.446     0.151     0.403
negative    0.392     0.190     0.418
neutral     0.433     0.148     0.418

['neutral' 'neutral' 'neutral' ... 'neutral' 'neutral' 'neutral']
```

We can see, the accuracy of model using Ngram+tf-idf and MultinomialNB in testsets is 40%, 37% and 39%. Its f1 score is 0.204, 0.242 and 0.227.

```
Training Ngram_tfidf_MultinomialNB
0.4011331444759207
twitter-test1.txt (Ngram_tfidf_MultinomialNB): 0.204
            positive  negative  neutral
positive    0.402     0.168     0.430
negative    0.403     0.127     0.470
neutral     0.424     0.159     0.417

['neutral' 'positive' 'neutral' ... 'neutral' 'neutral' 'neutral']
0.3720302375809935
twitter-test2.txt (Ngram_tfidf_MultinomialNB): 0.242
            positive  negative  neutral
positive    0.531     0.087     0.382
negative    0.524     0.135     0.341
neutral     0.531     0.113     0.357

['neutral' 'positive' 'neutral' ... 'positive' 'neutral' 'neutral']
0.39318755256518084
twitter-test3.txt (Ngram_tfidf_MultinomialNB): 0.227
            positive  negative  neutral
positive    0.434     0.157     0.409
negative    0.462     0.153     0.385
neutral     0.429     0.151     0.421

['negative' 'negative' 'neutral' ... 'neutral' 'neutral' 'neutral']
```

We can see, the accuracy of model using Lexicon and MultinomialNB in testsets

is 58%, 54% and 55%. Its f1 score is 0.197, 0.213 and 0.222.

```
Training Lexicon_MultinomialNB
0.58328611898017
twitter-test1.txt (Lexicon_MultinomialNB): 0.197
           positive  negative  neutral
positive    0.424     0.165     0.412
negative    0.400     0.133     0.467
neutral     0.415     0.157     0.428

['neutral' 'neutral' 'neutral' ... 'neutral' 'neutral' 'neutral']
0.5421166306695464
twitter-test2.txt (Lexicon_MultinomialNB): 0.213
           positive  negative  neutral
positive    0.524     0.117     0.359
negative    0.451     0.080     0.469
neutral     0.540     0.108     0.352

['neutral' 'positive' 'positive' ... 'neutral' 'neutral' 'neutral']
0.5588730025231287
twitter-test3.txt (Lexicon_MultinomialNB): 0.222
           positive  negative  neutral
positive    0.447     0.143     0.409
negative    0.392     0.183     0.425
neutral     0.434     0.152     0.413

['neutral' 'neutral' 'neutral' ... 'neutral' 'neutral' 'neutral']
running time is 529.6075518131256 second
```

We can see, the accuracy of model using word2vec and MultinomialNB in testsets is 42%, 36% and 41%. Its f1 score is 0, 0 and 0. So I gave up this method.

```
0.4257790368271955
twitter-test1.txt (word2vec): 0.000
           positive  negative  neutral
positive    0.000     0.000     0.000
negative    0.000     0.000     0.000
neutral     0.416     0.158     0.426

['neutral' 'neutral' 'neutral' ... 'neutral' 'neutral' 'neutral']
0.36069114470842334
twitter-test2.txt (word2vec): 0.000
           positive  negative  neutral
positive    0.000     0.000     0.000
negative    0.000     0.000     0.000
neutral     0.530     0.109     0.361

['neutral' 'neutral' 'neutral' ... 'neutral' 'neutral' 'neutral']
0.4129520605550883
twitter-test3.txt (word2vec): 0.000
           positive  negative  neutral
positive    0.000     0.000     0.000
negative    0.000     0.000     0.000
neutral     0.434     0.153     0.413
```

And it took 502s to run three models.

5. Conclusion

From the result we can find that lexicon performs better than ngram+tf-idf and SVM achieves higher accuracy than MultinomialNB.