Part 2 - IMDB data analysis essay

Student ID : C1981034

Module Title : Applied Machine Learning

Lecturer: Jose Camacho-Collados , Yuhua Li

Submission Date and Time: Tuesday, January 14th at 9:30am

Introduction

In the current era, due to the continuous development of artificial intelligence, machine learning has become increasingly popular, especially in this aspect of text processing. Because of the development of the Internet, more and more people use the Internet to comment on movie content. Positive comments and negative comments are mixed. If we can use machine learning sentiment analysis systems to judge negative comments, it will greatly improve our network environment.

In this essay, we will extract three features from a large amount of movie review data and use these three features to fit the sentiment analysis model. This essay will detail the process of feature extraction, the method of fitting the model, and verifying the accuracy of the model based on the test results

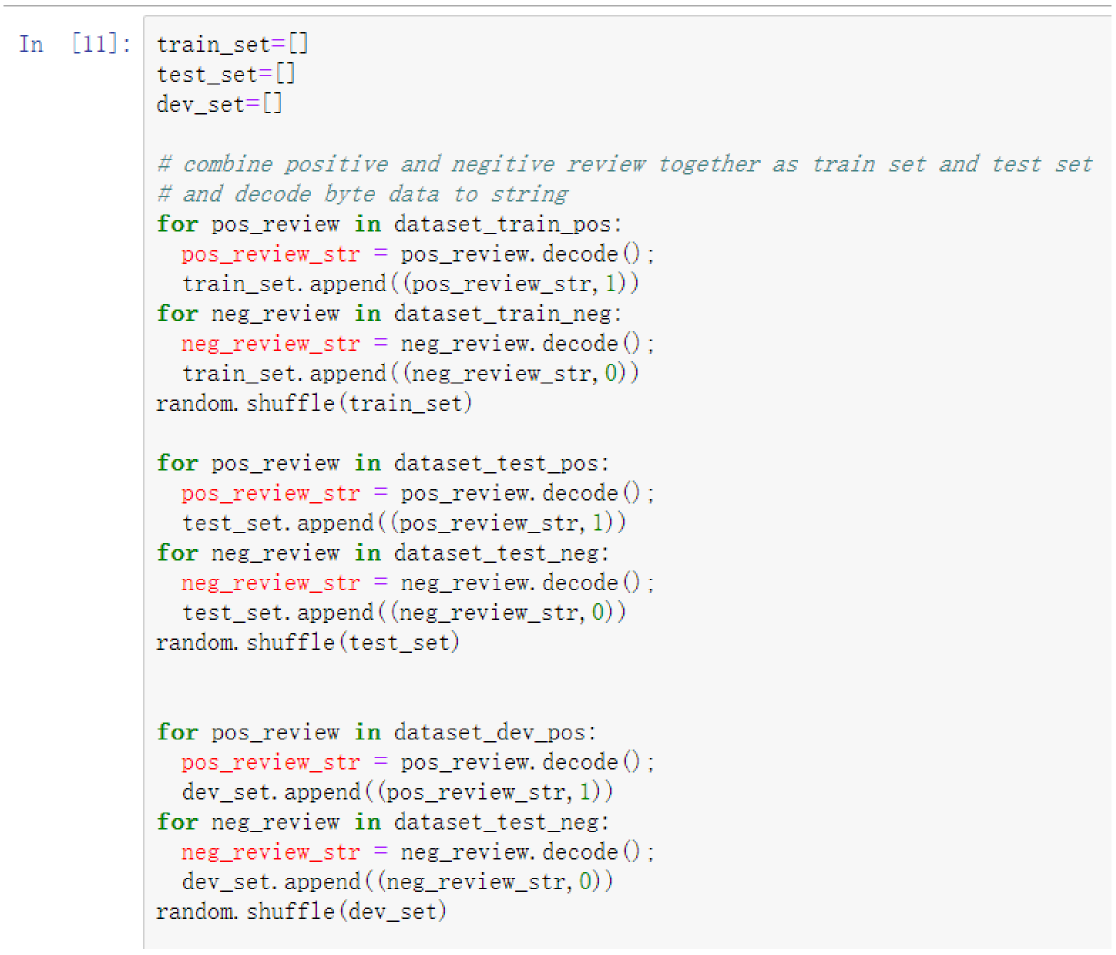
Data preprocessing

For doing a sentiment analysis from movie view, firstly it's necessary to preprocess the data which is from text file. To start with, some necessary modules has been imported, mainly Numpy for scientific computing, NLTK for text processing, and scikit-learn for fitting different models, doing feature extraction, and reduce the dimensions, which is most important module in this project.

After import and download these modules, it's enough to load positive and negative reviews at Training dataset, Testing dataset and Development dataset:

```
# load IMDB data
path_train_pos = "IMDb/train/imdb_train_pos.txt"
path_train_neg = "IMDb/train/imdb_train_neg.txt"
path_test_pos = "IMDb/test/imdb_test_pos.txt"
path_test_neg = "IMDb/test/imdb_test_neg.txt"
path_dev_pos = "IMDb/dev/imdb_dev_pos.txt"
path_dev_neg = "IMDb/dev/imdb_dev_neg.txt"

dataset_train_pos =open(path_train_pos,'rb').readlines()
dataset_train_neg =open(path_train_neg,'rb').readlines()
dataset_test_pos =open(path_test_pos,'rb').readlines()
dataset_test_neg =open(path_test_neg,'rb').readlines()
dataset_dev_pos =open(path_dev_pos,'rb').readlines()
dataset_dev_neg =open(path_dev_neg,'rb').readlines()
```

Then I combine positive and negative review together as train set, test set and dev set, here I label positive reviews with 1 and negative reviews with 0 in a list of tuples as illustrated in the following:



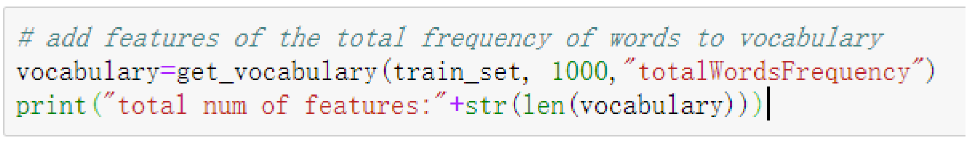
```
In  [11]: train_set=[]
test_set=[]
dev_set=[]

# combine positive and negitive review together as train set and test set
# and decode byte data to string
for pos_review in dataset_train_pos:
  pos_review_str = pos_review.decode();
  train_set.append((pos_review_str,1))
for neg_review in dataset_train_neg:
  neg_review_str = neg_review.decode();
  train_set.append((neg_review_str,0))
random.shuffle(train_set)

for pos_review in dataset_test_pos:
  pos_review_str = pos_review.decode();
  test_set.append((pos_review_str,1))
for neg_review in dataset_test_neg:
  neg_review_str = neg_review.decode();
  test_set.append((neg_review_str,0))
random.shuffle(test_set)


for pos_review in dataset_dev_pos:
  pos_review_str = pos_review.decode();
  dev_set.append((pos_review_str,1))
for neg_review in dataset_test_neg:
  neg_review_str = neg_review.decode();
  dev_set.append((neg_review_str,0))
random.shuffle(dev_set)
```

The train_set contains positive and negative reviews of the dataset we will use to extract features and fit the model. Test_set will be used to evaluate the fitted model. Dev_set is for fine tuning the model and avoid overfitting. And finally, I shuffle these sets to get a better accuracy.

Choice of features

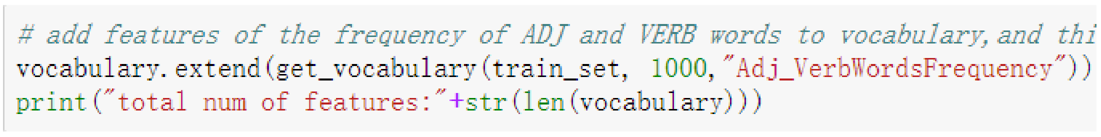
Here I choose 3 different type of features to fit model as following:

1. features of the total frequency of words (1000 features)



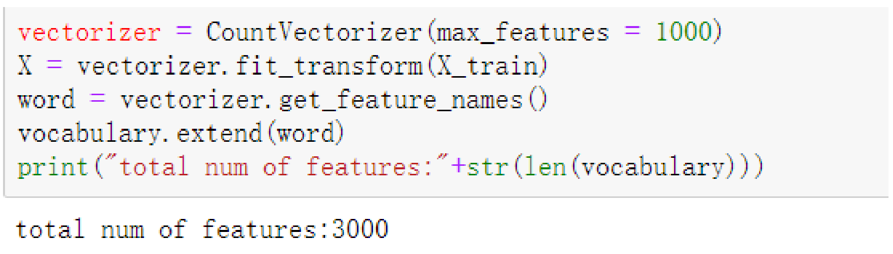
```
# add features of the total frequency of words to vocabulary
vocabulary=get_vocabulary(train_set, 1000,"totalWordsFrequency")
print("total num of features:"+str(len(vocabulary)))
```

2. features of the frequency of ADJ and VERB words (1000 features)



```
# add features of the frequency of ADJ and VERB words to vocabulary, and thi
vocabulary.extend(get_vocabulary(train_set, 1000,"Adj_VerbWordsFrequency"))
print("total num of features:"+str(len(vocabulary)))
```

3. features of key words from text by using CountVectorizer (1000 features)



```
vectorizer = CountVectorizer(max_features = 1000)
X = vectorizer.fit_transform(X_train)
word = vectorizer.get_feature_names()
vocabulary.extend(word)
print("total num of features:"+str(len(vocabulary)))
```

```
total num of features:3000
```

To implement the function of feature 1 and feature 2, I mostly used nltk module to get tokens from text. The feature 3 selection was implemented by using CountVectorizer from sklearn module. The details of these three features implement is in the code and well commented. After extracting a total of 3,000 features, it is necessary to start training the data.

Reduce the dimensionality and train data

In the choice of features section above, we totally selected 3000 features based on the 3 different kind of features. However, not all words may be equally relevant for our task. Therefore, in this project we are going to use the chi-squared test method from sklearn module. This method basically removes the features that appear to be irrelevant to a given class (in our case positive or negative).I apply this feature selection method to vocabulary dataset to keep only the 1000 most relevant features. And fit the model from these features as below:

```
fs_sentanalysis=SelectKBest(chi2, k=1000).fit(X_train, Y_train)
X_train_new = fs_sentanalysis.transform(X_train)

print ("Size original training matrix: "+str(X_train.shape))
print ("Size new training matrix: "+str(X_train_new.shape))

# Finally, we train the SVM classifier
svm_clf=svm.SVC(kernel="linear",gamma='auto')
svm_clf.fit(X_train_new,Y_train)
```

```
Size original training matrix: (15000, 3000)
Size new training matrix: (15000, 1000)
```

```
2]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
      decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
      max_iter=-1, probability=False, random_state=None, shrinking=True,
      tol=0.001, verbose=False)
```

Performance

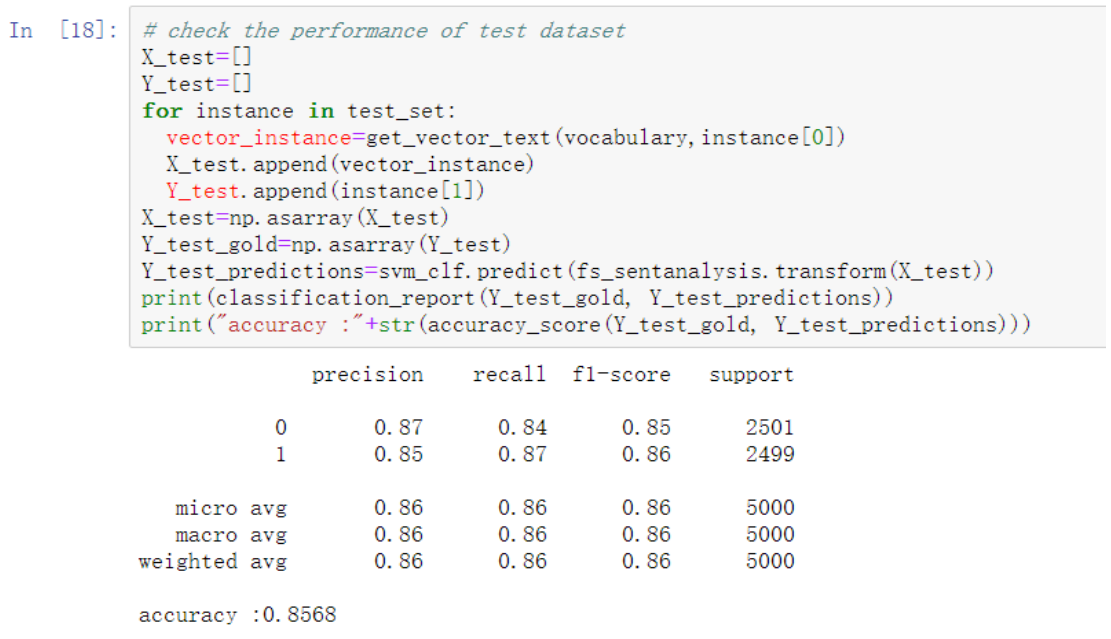
Base on the trained model, we can check its performance, firstly I need to check the performance of development dataset for fine tuning:

```
In [16]: # check the performance of development set firstly, this is for fine tuning

X_dev=[]
Y_dev=[]
for instance in dev_set:
  vector_instance=get_vector_text(vocabulary,instance[0])
  X_dev.append(vector_instance)
  Y_dev.append(instance[1])
X_dev=np.asarray(X_dev)
Y_dev_gold=np.asarray(Y_dev)
Y_dev_predictions=svm_clf.predict(fs_sentanalysis.transform(X_dev))
print(classification_report(Y_dev_gold, Y_dev_predictions))
print("accuracy :"+str(accuracy_score(Y_dev_gold, Y_dev_predictions)))
```

```
              precision    recall  f1-score   support

           0       0.86      0.84      0.85      2501
           1       0.85      0.87      0.86      2518

   micro avg       0.85      0.85      0.85      5019
   macro avg       0.85      0.85      0.85      5019
weighted avg       0.85      0.85      0.85      5019

accuracy :0.8537557282327157
```

	precision	recall	f-measure	accuracy
Dev-set	0.85	0.85	0.85	0.85

From the performance result of development set, we may see the performance is really good with the fitted model, therefore I keep these parameters and check the performance of test dataset



```
In  [18]: # check the performance of test dataset
          X_test=[]
          Y_test=[]
          for instance in test_set:
            vector_instance=get_vector_text(vocabulary, instance[0])
            X_test.append(vector_instance)
            Y_test.append(instance[1])
          X_test=np.asarray(X_test)
          Y_test_gold=np.asarray(Y_test)
          Y_test_predictions=svm_clf.predict(fs_sentanalysis.transform(X_test))
          print(classification_report(Y_test_gold, Y_test_predictions))
          print("accuracy :"+str(accuracy_score(Y_test_gold, Y_test_predictions)))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.85      2501
           1       0.85      0.87      0.86      2499

   micro avg       0.86      0.86      0.86      5000
   macro avg       0.86      0.86      0.86      5000
weighted avg       0.86      0.86      0.86      5000

accuracy :0.8568
```

	precision	recall	f-measure	accuracy
Test-set	0.86	0.86	0.86	0.86

From the data obtained from the above experiments, our fitted model works very well, which can explain that the correct features and parameters are selected.

Conclusion

From the above experimental results, our fitted model works very well. There is no doubt that this model will be perfectly applied to the classification of most movie reviews. However, there still be some things to improve: feature extraction is very slow, and it takes a lot of time to fit the data, these are very expensive. I think this problem can be solved by optimizing the algorithm of the feature extraction part, in this way the efficiency of the program may be got effectively improved.

Reference

Some code is referred from CMT-307 lab code.

Extra credit (optional - 15% extra marks in the second part):

d. Code release:

https://github.com/kagamikuro/CMT307-IMDB-project