CMPE 255-02 Data Mining Program 1

Submitted By: Kashika Jain

SID: 012505649

Email: kashika.jain@sjsu.edu

F1 Score – 0.5209

PR1: Text Sentiment Analysis

Objective

The objective of the project is to perform text sentiment analysis for multiclass data by building a predictive model for classifying the given data into appropriate classes. We are provided with a training dataset containing 31000 records which have text message and classifications of those messages into 11 classes. The idea is to classify the training dataset into appropriate classes based on the predictive model built. The data provided is imbalanced, so I have experimented with many classification techniques and compared their efficiency using the F1- score metric.

Methodology -

Import data

1. Import the tab separated training data using pandas function.

```
train= pd.read_csv(filepath_or_buffer='train.dat', header=0, sep='\t')
```

Exploring Data

2. Perform exploratory data analysis on the training data to explore the data and see how many objects of each classes are present using describe and group by functions.

```
train.describe()
train.groupby('sentiment').describe()
```

Data Cleaning and Preprocessing

3. Convert the text part of the training data to string.

```
train['Text']=train['Text'].astype(str)
```

4. Import the stopwords from the NLTK library and remove the stopwords from the training data.
5. Clean the data using PorterStemmer from nltk.stem library to normalize the data for similar words with different spellings to words that mean the same

```
from nltk.stem import PorterStemmer
porter_stemmer = PorterStemmer()
def stem_sentences(sentence):
   tokens = sentence.split(" ")
   stemmed_tokens = [porter_stemmer.stem(token) for token in
tokens]
   return ' '.join(stemmed_tokens)

testfile[0]=testfile[0].apply(stem_sentences)
```

6. Remove the numerical data from the training dataset

```
import re
for i in range(len(testfile)):
   testfile[0][i] = re.sub('\d', '', testfile[0][i])
```

7. Remove the punctuations from the text and convert the words into lower case and split the sentences into words.
8. Use bag of words method to vectorize the words in the text using CountVectorization method from the sklearn.feature_extraction.text library for feature extraction to perform these text cleaning steps
9. Use TfidfTransformer (Term Frequency Inverse Document Frequency) method for feature extraction to calculate the term frequency of the terms in the dataset and accordingly calculate the similarity between the words.

Prediction Model

10. Split the preprocessed and cleaned data into training and testing dataset using train_test_split function imported from sklearn.model_selection. With the train test split as 80-20

```
msg_train, msg_text, label_train, label_text
=train_test_split(train['Text'],train['sentiment'], test_size=0.2)
```

11. Use Pipeline method to build the predictive model.

12. I used many different classification models to build the model –
    a) RandomForestClassifier with number of estimators 10, 100, 1000 and 3000
    b) RandomForestClassifier with balanced class
    c) SVC(Support Vector machines) using linear classification and varying values of C = 1,5,10 and 100 and gamma values ranging from 10,1, 0.1, 0.01, 0.001, 0.0001.
    d) GridSearchCV with RandomForestClassifier as estimator, class_weight balanced and cv=5
    e) GradientBoostingClassifier
13. The algorithm that performed the best was using SVC with C =1, gamma =1

```
pipeline =Pipeline([
   ('bow',CountVectorizer(analyzer=text_process)),
   ('tfidf', TfidfTransformer()),
   ('Classifier',SVC(C =1,kernel='linear', gamma = 1))
])
```

The CountVectorization and Tfidf functions are performed in this step. The text process function here calls the preprocessing function as shown below:

```
def text_process(mess):
"""
1. Remove Punctuation
2. remove stopwords
3. return list of clean words
"""
nopunc = [char for char in mess if char not in string.punctuation]
nopunc = ''.join(nopunc)
return [word for word in nopunc.split() if word.lower() not in
stopwords.words('english') ]
```

14. The model is fit using pipeline.fit method and the test and class labels of text data are passed as parameters to fit the model

```
pipeline.fit(msg_train,label_train)
```

Model Prediction and Evaluation

15. Perform prediction on the training data which was split as 20 percent to classify the texts into classification sentiments.

```
predictions = pipeline.predict(msg_text)
```

16. The model performed fairly well with an F1 score of 33% for the training split. This can be checked using the Classification Reprot metric of sklearn

```
print(classification_report(label_text, predictions))
```

17. Read the test file provided with 6575 texts using pandas function

```
 pd.read_csv(filepath_or_buffer='test.dat', header=None,  sep=
'\t',skip_blank_lines=False)
```

18. The blank lines in the test file are also included.
19. The testfile is converted into strings
20. PorterStemmer is performed on the testfile to normalize the data and numericals are removed from the testfile.
21. The model build using the SVC method is now applied on the test data to predict the multilabel classes.

```
predictions = pipeline.predict(testfile[0])
```

22. The file is saved using Numpy function

```
np.savetxt ('pred23.dat',predictions,fmt = '%d')
```

23. This file is uploaded in the portal to check for the F1 score.

Conclusion

The predictive model built for text sentiment analysis gives a F1 score of 52.1 %. I checked the score with different models and the models that performed well included Random Forest Classifier with number of estimators as 3000 and Gradient Boosting. However, I achieved the highest accuracy with Linear Support vector machines with C=1 and gamma =1.