Applied Machine Learning Homework 5: NLP

Due May 2,2023 (Tuesday) 11:59PM EST

Instructions

1) Please push the .ipynb and .pdf to Github Classroom prior to the deadline, .py file is optional (not needed).
2) Please include your Name and UNI below.

Name: Liang Hu

UNI: lh3057

Natural Language Processing

We will train a supervised training model to predict if a tweet has a positive or negative sentiment.

Dataset loading & dev/test splits

1.1) Load the twitter dataset from NLTK library

In [1]:
```
import nltk
nltk.download('twitter_samples')
from nltk.corpus import twitter_samples
nltk.download('punkt')
nltk.download('stopwords')

import warnings
warnings.filterwarnings("ignore")

from nltk.corpus import stopwords
stop = stopwords.words('english')
import pandas as pd
import string
import re
from sklearn.model_selection import train_test_split
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report
# Feel free to import any other packages you need
```

```
[nltk_data] Downloading package twitter_samples to
[nltk_data]     /Users/larry_1/nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
[nltk_data] Downloading package punkt to /Users/larry_1/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/larry_1/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

1.2) Load the positive & negative tweets

In [3]:
```
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

1.3) Make a data frame that has all tweets and their corresponding labels

In [4]:
```
# code here
df = pd.DataFrame({'tweet': all_positive_tweets + all_negative_tweets,
                   'label': ['positive'] * len(all_positive_tweets) + ['negative'] *
```

1.4) Look at the class distribution of the tweets

In [5]:
```
df["label"].value_counts()
```

Out[5]:
```
positive    5000
negative    5000
Name: label, dtype: int64
```

1.5) Create a development & test split (80/20 ratio):

In [6]:
```
X_dev,X_test,y_dev,y_test = train_test_split(df["tweet"],df["label"],test_size = 0.2
```

Data preprocessing

We will do some data preprocessing before we tokenize the data. We will remove # symbol, hyperlinks, stop words & punctuations from the data. You can use the re package in python to find and replace these strings

1.6) Replace the # symbol with '' in every tweet

In [7]:
```
X_dev = X_dev.str.replace('#','')
X_test = X_test.str.replace('#','')
```

1.7) Replace hyperlinks with '' in every tweet

In [8]:
```
X_dev = X_dev.str.replace(r'http\S+', '')
X_test = X_test.str.replace(r'http\S+', '')
```

1.8) Remove all stop words

In [9]:
```
for word in stop:
    X_dev = X_dev.str.replace(' '+word+' ',' ').replace(' '+word,' ').replace(word+'
    X_test = X_test.str.replace(' '+word+' ',' ').replace(' '+word,' ').replace(word
```

1.9) Remove all punctuations

In [10]:
```
for punctuation in string.punctuation:
    X_dev = X_dev.str.replace(punctuation,' ')
    X_test = X_test.str.replace(punctuation,' ')
```

1.10) Apply stemming on the development & test datasets using Porter algorithm

In [11]:
```
#code here
def stemSentence(sentece):
    porter = PorterStemmer()
    token_words = word_tokenize(sentece)
    stem_sentence = [porter.stem(word) for word in token_words]
    return " ".join(stem_sentence)

for index,sentence in X_dev.iteritems():
    X_dev[index] = stemSentence(sentence)
for index,sentence in X_test.iteritems():
    X_test[index] = stemSentence(sentence)
```

Model training

1.11) Create bag of words features for each tweet in the development dataset

In [23]:
```
#code here
vector = CountVectorizer(stop_words = 'english')
X_dev_bow = vector.fit_transform(X_dev)
feature_names = vector.get_feature_names()
```

In [25]:
```
print("Shape of development dataset bag of words features:", X_dev_bow.shape)
```

```
Shape of development dataset bag of words features: (8000, 14309)
```

1.12) Train a Logistic Regression model on the development dataset

In [26]:
```
#code here
lr = LogisticRegression(random_state=42).fit(X_dev_bow,y_dev)
```

1.13) Create TF-IDF features for each tweet in the development dataset

Approach of splitting data after data processing is efficient, however follow the notebook approach for now for the train test split first and then performing preprocessing steps.

Ensure you do the data preprocessing steps on test dataset too in 1.11 and 1.13

In [27]:
```
#code here
vector_tf = TfidfVectorizer()
X_dev_tf = vector_tf.fit_transform(X_dev)
feature_names_tf = vector_tf.get_feature_names()
# print(feature_names_tf.shape())
```

In [28]:
```
X_dev_tf_dense = X_dev_tf.toarray()
print("Shape of development dataset TF-IDF features:", X_dev_tf_dense.shape)
```

```
Shape of development dataset TF-IDF features: (8000, 14497)
```

1.14) Train the Logistic Regression model on the development dataset with TF-IDF features

In [30]:
```
#code here
lr_tf = LogisticRegression(random_state=42).fit(X_dev_tf,y_dev)
```

1.15) Compare the performance of the two models on the test dataset using a classification report and the scores obtained. Explain the difference in results obtained.

In [35]:
```
X_test_bow = vector.transform(X_test)
y_pred_bow = lr.predict(X_test_bow)
X_test_tf = vector_tf.transform(X_test)
y_pred_tf = lr_tf.predict(X_test_tf)
print(f"The performance of bag of words is {lr.score(X_test_lr,y_test)}")
print(f"The performance of tf-idf is {lr_tf.score(X_test_tf,y_test)}")
```

```
The performance of bag of words is 0.7525
The performance of tf-idf is 0.7515
```

In [36]:
```
# Generate classification reports
report_bow = classification_report(y_test, y_pred_bow)
report_tf = classification_report(y_test, y_pred_tf)
print("Classification report for the model trained on bag of words features:\n", repo
print("Classification report for the model trained on TF-IDF features:\n", report_tf
```

```
Classification report for the model trained on bag of words features:
              precision    recall  f1-score   support

    negative       0.73      0.80      0.77      1006
    positive       0.78      0.70      0.74       994

    accuracy                           0.75      2000
   macro avg       0.75      0.75      0.75      2000
weighted avg       0.75      0.75      0.75      2000

Classification report for the model trained on TF-IDF features:
              precision    recall  f1-score   support

    negative       0.74      0.77      0.76      1006
    positive       0.76      0.73      0.75       994

    accuracy                           0.75      2000
   macro avg       0.75      0.75      0.75      2000
weighted avg       0.75      0.75      0.75      2000
```

The performance of bag of words is 0.7405. The performance of tf-idf is 0.748. The bag of words weights more on the high frequency word. On the other hand, the TF-IDF model weights more on the unique word. Using the tf-idf method to represent text data has provided better results compared to the bag of words approach.