Applied Machine Learning Homework 5

Due 2 May,2022 (Monday) 11:59PM EST

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

```
import nltk
nltk.download('twitter_samples')
from nltk.corpus import twitter_samples
```

```
[nltk_data] Downloading package twitter_samples to
[nltk_data]     C:\Users\cwp94\AppData\Roaming\nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
```

1.2) Load the positive & negative tweets

```
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

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

```
#code here
from sklearn.model_selection import train_test_split
print("size of positive tweets:", len(all_positive_tweets))
print("size of negative tweets:", len(all_negative_tweets))

pos_dev, pos_test, neg_dev, neg_test =
train_test_split(all_positive_tweets, all_negative_tweets,
test_size=0.2, random_state=42)
```

```
size of positive tweets: 5000
size of negative tweets: 5000
```

```
import numpy as np


## dev
pos_dev = np.array(pos_dev)
neg_dev = np.array(neg_dev)
X_dev = np.concatenate((pos_dev, neg_dev), axis=0)

pos_label_dev = np.ones((pos_dev.shape[0],1))
neg_label_dev = np.zeros((neg_dev.shape[0],1))
```

```
y_dev = np.concatenate((pos_label_dev, neg_label_dev), axis=0)

## test
pos_test = np.array(pos_test)
neg_test = np.array(neg_test)
X_test = np.concatenate((pos_test, neg_test), axis=0)

pos_label_test = np.ones((pos_test.shape[0],1))
neg_label_test = np.zeros((neg_test.shape[0],1))
y_test = np.concatenate((pos_label_test, neg_label_test), axis=0)
```

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.

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

```
#code here
import re

## dev
for i in range(X_dev.shape[0]):
    sentence = X_dev[i][:]
    X_dev[i] =re.sub('#', '', sentence)

## test
for i in range(X_test.shape[0]):
    sentence = X_test[i][:]
    X_test[i] = re.sub('#', '', sentence)
```

1.5) Replace hyperlinks with '' in every tweet

```
#code here
regex = r"(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]
{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\
s()<>]+|(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:'\".,<>?«»“”‘’]))"

## dev
for i in range(X_dev.shape[0]):
    sentence = X_dev[i][:]
    X_dev[i] = re.sub(regex, '', sentence)

## test
for i in range(X_test.shape[0]):
    sentence = X_test[i][:]
    X_test[i] = re.sub(regex, '', sentence)
```

1.6) Remove all stop words

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\cwp94\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

True
```

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

## dev
for i in range(X_dev.shape[0]):
    sentence = X_dev[i][:]

    word_tokens = sentence.split()
    filtered_sentence = [w for w in word_tokens if not w.lower() in
stop_words]
    X_dev[i] = ' '.join(filtered_sentence)

## test
for i in range(X_test.shape[0]):
    sentence = X_test[i][:]
    word_tokens = sentence.split()
    filtered_sentence = [w for w in word_tokens if not w.lower() in
stop_words]
    X_test[i] = ' '.join(filtered_sentence)
```

1.7) Remove all punctuations

```
## dev
for i in range(X_dev.shape[0]):
    sentence = X_dev[i][:]
    X_dev[i] = re.sub(r'[^\w\s]', '', sentence)

## test
for i in range(X_test.shape[0]):
    sentence = X_test[i][:]
    X_test[i] = re.sub(r'[^\w\s]', '', sentence)
```

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

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\cwp94\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!

True
```

```
#code here
from nltk.tokenize import word_tokenize
```

```
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()

def stemSentence(text):
    token_words = word_tokenize(text)
    stem_sentence = [porter.stem(word) for word in token_words]
    return " ".join(stem_sentence)

X_dev_clean = [stemSentence(sen) for sen in X_dev]
X_test_clean = [stemSentence(sen) for sen in X_test]
```

Model training

```
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
```

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

```
#code here
vector_bow = CountVectorizer()
X_dev_bow = vector_bow.fit_transform(X_dev_clean)
X_test_bow = vector_bow.transform(X_test_clean)
```

1.10) Train a supervised learning model of choice on the development dataset

```
#code here
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

lr_bow = LogisticRegression()
lr_bow.fit(X_dev_bow, y_dev.flatten())
```

```
LogisticRegression()
```

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

```
#code here

vector_tfidf = TfidfVectorizer()
X_dev_tfidf = vector_tfidf.fit_transform(X_dev_clean)
X_test_tfidf = vector_tfidf.transform(X_test_clean)
```

1.12) Train the same supervised learning algorithm on the development dataset with TF-IDF features

```
#code here
lr_tfidf = LogisticRegression()
lr_tfidf.fit(X_dev_tfidf, y_dev.flatten())
```

```
LogisticRegression()
```

1.13) Compare the performance of the two models on the test dataset

```
#code here
print("*Result from BoW Logistic Regression")
predictions_bow = lr_bow.predict(X_test_bow)
print(classification_report(y_test.flatten(), predictions_bow))
print()
print("*Result from TF-IDF Logistic Regression")
predictions_tfidf = lr_tfidf.predict(X_test_tfidf)
print(classification_report(y_test.flatten(), predictions_tfidf))
```

```
*Result from BoW Logistic Regression
              precision    recall  f1-score   support

         0.0       0.74      0.77      0.76      1000
         1.0       0.76      0.73      0.75      1000

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


*Result from TF-IDF Logistic Regression
              precision    recall  f1-score   support

         0.0       0.76      0.76      0.76      1000
         1.0       0.76      0.76      0.76      1000

    accuracy                           0.76      2000
   macro avg       0.76      0.76      0.76      2000
weighted avg       0.76      0.76      0.76      2000
```

The overall performance between the two models are very similar. Keeping in mind that the dataset was balanced between the positive and negative tweets, our most important metric will be accuracy. We can see that the accuracy for TF-IDF is slightly higher by 0.01, so we can conclude that TF-IDF Logistic Regression is a better model than BoW Logistic Regression.