Applied Machine Learning Homework 4

Due 12/15/21 11:59PM EST

Q1: 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_data] Downloading package twitter_samples to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
```

1.2) Load the positive & negative tweets

In [2]:

```
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):

In [3]:

```
#code here
label = [1]*len(all_positive_tweets)+[0]*len(all_negative_tweets)
from sklearn.model_selection import train_test_split
X_dev,X_test,y_dev,y_test = train_test_split(all_positive_tweets+all_negative_tweets,
                                             label,test_size=0.2,stratify=label,random_s
tate=123)
```

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

In [4]:

```
#code here
X_dev = [tweet.replace('#','') for tweet in X_dev]
X_test = [tweet.replace('#','') for tweet in X_test]
```

1.5) Replace hyperlinks with '' in every tweet

In [5]:

```
#code here
```

```
import re
X_dev = [re.sub(r"http\S+", "", tweet) for tweet in X_dev]
X_test = [re.sub(r"http\S+", "", tweet) for tweet in X_test]
```

1.6) Remove all stop words

In [6]:

```
#code here
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
def remove_stopwords(sentence):
    words = word_tokenize(sentence)
    words = [w for w in words if w not in stopwords.words('english')]
    return ' '.join(words)
X_dev = [remove_stopwords(x) for x in X_dev]
X_test = [remove_stopwords(x) for x in X_test]
```

1.7) Remove all punctuations

In [7]:

```
#code here
import string
X_dev = [s.translate(str.maketrans('', '', string.punctuation)) for s in X_dev]
X_test = [s.translate(str.maketrans('', '', string.punctuation)) for s in X_test]
```

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

In [8]:

```
#code here
from nltk.stem import PorterStemmer
porter = PorterStemmer()
def stemmer(sent):
    words = word_tokenize(sent)
    words = [porter.stem(w) for w in words]
    return ' '.join(words)
X_dev = [stemmer(x) for x in X_dev]
X_test = [stemmer(x) for x in X_test]
```

Model training

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

In [9]:

```
#code here
from sklearn.feature_extraction.text import CountVectorizer
vector = CountVectorizer()
X_dev_bag = vector.fit_transform(X_dev)
X_test_bag = vector.transform(X_test)
```

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

In [10]:

```
#code here
from sklearn.linear_model import LogisticRegression
lr_bag = LogisticRegression()
lr_bag.fit(X_dev_bag,y_dev)
```

Out[10]:

```
LogisticRegression()
```

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

In [11]:

```
#code here
from sklearn.feature_extraction.text import TfidfVectorizer
tf_vector = TfidfVectorizer()
X_dev_tf = tf_vector.fit_transform(X_dev)
X_test_tf = tf_vector.transform(X_test)
```

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

In [12]:

```
#code here
lr_tf = LogisticRegression()
lr_tf.fit(X_dev_tf,y_dev)
```

Out[12]:

```
LogisticRegression()
```

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

In [13]:

```
#code here
print(f"Score of model with bag-of-words features: {lr_bag.score(X_test_bag,y_test)}")
print(f"Score of model with TF-IDF features: {lr_tf.score(X_test_tf,y_test)}")
```

```
Score of model with bag-of-words features: 0.7725
Score of model with TF-IDF features: 0.7705
```

The performance of the model with bag-of-words features is slightly better than model with TF-IDF features.