AML_HW5_Questions_NLP

April 28, 2024

1 Applied Machine Learning Homework 5: NLP

Due April 29,2024 (Monday) 11:59PM EST

1.0.1 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: Apurva Patel ##UNI: amp2365

1.0.2 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

[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 /root/nltk_data…
[nltk_data]   Unzipping corpora/twitter_samples.zip.
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

1.2) Load the positive & negative tweets

[2]:
```
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

[3]:
```
# Your Code Here

positive_tweets_df = pd.DataFrame({'tweet': all_positive_tweets, 'label': 1})
negative_tweets_df = pd.DataFrame({'tweet': all_negative_tweets, 'label': 0})

# combine the 2 DF
all_tweets_df = pd.concat([positive_tweets_df, negative_tweets_df],␣
 ↪ignore_index=True)
print(all_tweets_df.head())
```

```
                                               tweet  label
0  #FollowFriday @France_Inte @PKuchly57 @Milipol…      1
1  @Lamb2ja Hey James! How odd :/ Please call our…      1
2  @DespiteOfficial we had a listen last night :)…      1
3                               @97sides CONGRATS :)      1
4  yeaaaah yippppy!!!  my accnt verified rqst has…      1
```

1.4) Look at the class distribution of the tweets

[4]:
```
# Your Code Here

class_distribution = all_tweets_df['label'].value_counts()
print(class_distribution)
```

```
label
1    5000
0    5000
Name: count, dtype: int64
```

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

[5]:
```
# Your Code Here
```

```
dev_set, test_set = train_test_split(all_tweets_df, test_size=0.2,␣
 ↪random_state=42)
print("Development set shape:", dev_set.shape)
print("Test set shape:", test_set.shape)
```

```
Development set shape: (8000, 2)
Test set shape: (2000, 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

[6]:
```
# Your Code Here
dev_set['tweet'] = dev_set['tweet'].apply(lambda x: x.replace('#', ''))
test_set['tweet'] = test_set['tweet'].apply(lambda x: x.replace('#', ''))
```

1.7) Replace hyperlinks with '' in every tweet

[7]:
```
# Your Code Here
dev_set['tweet'] = dev_set['tweet'].apply(lambda x: re.sub(r'http\S+', '', x))
test_set['tweet'] = test_set['tweet'].apply(lambda x: re.sub(r'http\S+', '', x))
```

1.8) Remove all stop words

[8]:
```
# Your Code Here

stop_words = set(stopwords.words('english'))

dev_set['tweet'] = dev_set['tweet'].apply(lambda x: ' '.join([word for word in␣
 ↪x.split() if word.lower() not in stop_words]))
test_set['tweet'] = test_set['tweet'].apply(lambda x: ' '.join([word for word␣
 ↪in x.split() if word.lower() not in stop_words]))
```

1.9) Remove all punctuations

[9]:
```
# Your Code Here

dev_set['tweet'] = dev_set['tweet'].apply(lambda x: x.translate(str.
 ↪maketrans('', '', string.punctuation)))
test_set['tweet'] = test_set['tweet'].apply(lambda x: x.translate(str.
 ↪maketrans('', '', string.punctuation)))
```

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

[10]:
```
# Your Code Here
stemmer = PorterStemmer()
```

```
dev_set['tweet'] = dev_set['tweet'].apply(lambda x: ' '.join([stemmer.
 ↪stem(word) for word in x.split()]))
test_set['tweet'] = test_set['tweet'].apply(lambda x: ' '.join([stemmer.
 ↪stem(word) for word in x.split()]))
```

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

[11]:
```
# Your Code Here

vectorizer = CountVectorizer()
X_train_bow = vectorizer.fit_transform(dev_set['tweet'])
```

1.12) Train a Logistic Regression model on the development dataset

[12]:
```
# Your Code Here

lr_model = LogisticRegression()
lr_model.fit(X_train_bow, dev_set['label'])
```

[12]:
```
LogisticRegression()
```

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

[13]:
```
# Your Code Here

tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(dev_set['tweet'])
```

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

[14]:
```
# Your Code Here

lr_model_tfidf = LogisticRegression()
lr_model_tfidf.fit(X_train_tfidf, dev_set['label'])
```

[14]:
```
LogisticRegression()
```

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.

[15]:
```
# Your Code Here
from sklearn.metrics import classification_report

# prediction using BoW features
X_test_bow = vectorizer.transform(test_set['tweet'])
bow_predictions = lr_model.predict(X_test_bow)
```

```
# prediction using TF-IDF features
X_test_tfidf = tfidf_vectorizer.transform(test_set['tweet'])
tfidf_predictions = lr_model_tfidf.predict(X_test_tfidf)

# classification report for BoW model
print("Classification Report for Bag of Words model:")
print(classification_report(test_set['label'], bow_predictions))

# classification report for TF-IDF model
print("\nClassification Report for TF-IDF model:")
print(classification_report(test_set['label'], tfidf_predictions))
```

```
Classification Report for Bag of Words model:
              precision    recall  f1-score   support

           0       0.73      0.80      0.76       988
           1       0.78      0.71      0.74      1012

    accuracy                           0.75      2000
   macro avg       0.76      0.75      0.75      2000
weighted avg       0.76      0.75      0.75      2000


Classification Report for TF-IDF model:
              precision    recall  f1-score   support

           0       0.74      0.78      0.76       988
           1       0.78      0.73      0.76      1012

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

*Explanation here

Bag of Words (BoW) and TF-IDF models perform well, with similar overall performance metrics.

However, the TF-IDF model, which considers the importance of words in the corpus, shows slightly higher accuracy and slightly improved precision, recall, and F1-score compared to the BoW model.

This difference is due to TF-IDF's ability to capture more meaningful and discriminative features, resulting in slightly better classification performance on the test dataset.

The difference in results between the Bag of Words (BoW) model and the TF-IDF model lies in the way they represent the features of the text data.

- BoW simply counts the occurrence of each word in the document, whereas TF-IDF considers the importance of each word in the document relative to the entire corpus.

- TF-IDF tends to give higher weights to words that are more unique to the document but less frequent in the corpus, which can result in better performance, especially when dealing with large and diverse datasets.

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc

!pip install pypandoc
```

[22]:
```
!jupyter nbconvert --to html /content/AML_HW5_Questions_NLP.ipynb
```

```
[NbConvertApp] Converting notebook /content/AML_HW5_Questions_NLP.ipynb to html
[NbConvertApp] Writing 629162 bytes to /content/AML_HW5_Questions_NLP.html
```

[23]:
```
!jupyter nbconvert AML_HW5_Questions_NLP.ipynb --to latex
```

```
[NbConvertApp] Converting notebook AML_HW5_Questions_NLP.ipynb to latex
[NbConvertApp] Writing 43472 bytes to AML_HW5_Questions_NLP.tex
```