B455: Project 5

Serena Patel

(04/25/2020)

output: pdf

This project is about the classification of movie reviews into positive and negative reviews by training a logistic regression model. We will be using the IMDb dataset. "The movie review dataset consists of 50,000 popular movie reviews that are labeled as either positive or negative; here, a positive means that a movie was rated with more than six stars on IMDb, and negative means that a movie was rated with fewer than five stars on IMDb." The key steps for the training process include: clean text data, process documents into tokens, transform words into feature words, access word relevancy, and build the logistic model.

Since the the testing and training data was separated into positive and negative reviews where each folder had each review was in a separate text file. I combined these files into one csv file where it had the review from the data file followed by 1 or 0 depending on if it is positive or negative.

In [0]:
```
import pandas as pd

dataset = pd.read_csv('movie_data.csv')
```

Transforming Documents Into Feature Vectors

By transforming our document into feature vectors, we will create a vocabulary to help us separate, or parse, the data into feature vector. I used sklearn's CountVectorizer feature to implement thi. This simplifies the process.

In [20]:
```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer()
docs = (['The sun is shining',
         'The weather is sweet',
         'The sun is shining, the weather is sweet, and one and one is
two'])
bag = count.fit_transform(docs)

print(count.vocabulary_)
print(bag.toarray())
```

```
{'the': 6, 'sun': 4, 'is': 1, 'shining': 3, 'weather': 8, 'sweet': 5
, 'and': 0, 'one': 2, 'two': 7}
[[0 1 0 1 1 0 1 0 0]
 [0 1 0 0 0 1 1 0 1]
 [2 3 2 1 1 1 2 1 1]]
```

Word Relevancy

By using term frequency-inverse document frequency, we can calculate the relevancy of the words. We find this by looking at the number of times a word occurs in relation to the number of words we are analyzing. This allows us to see the words as numbers instead words.

In [21]:
```
from sklearn.feature_extraction.text import TfidfTransformer
np.set_printoptions(precision=2)

tfidf = TfidfTransformer(use_idf=True, norm='l2', smooth_idf=True)

print(tfidf.fit_transform(bag).toarray())
```

```
[[0.   0.43 0.   0.56 0.56 0.   0.43 0.   0.  ]
 [0.   0.43 0.   0.   0.   0.56 0.43 0.   0.56]
 [0.5  0.45 0.5  0.19 0.19 0.19 0.3  0.25 0.19]]
```

Data Cleaning

Cleaning the data is essential. The essence of the analysis is about words, whether they're negative or positive. So, the emoticons, such as punctuation or other grammatical markings, do not necessarily add much. So, we can consider it to be noise and take it out of the data so we are only looking at words.

In [0]:
```
import re
def preprocessor(text):
  text =re.sub('<[^>]*>', '', text)
  emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text)
  text = re.sub('[\W]+', ' ', text.lower()) + ' '.join(emoticons).repl
ace('-', '')

  return text

dataset['review'] = dataset['review'].apply(preprocessor)
```

Tokenization

"Tokenization is the process of splitting a string into a list of tokens." In particular, we are stemming our words. Stemming essentially takes variations of a word and changes it to its essence. For example, 'watching' and 'watched' turns into 'watch'. We do this because it simplifies our data by taking out the noise from the word. We do not care about if a word is present or past tense, we care about if it has a negative or positive connotation. We also utilize stop words to take out 'useless' words. These are words, such as prepositions or transition words, that add no real value to our analysis of the sentence.

We utilize the nltk library for this implementation.

In [25]:
```
from nltk.stem.porter import PorterStemmer

porter = PorterStemmer()

def tokenizer(text):
  return text.split()

def tokenizer_stemmer(text):
  return[porter.stem(word) for word in text.split()]

tokenizer('runners like running thus they run')
tokenizer_stemmer('runners like running thus they run')

import nltk
nltk.download('stopwords')

from nltk.corpus import stopwords

stop = stopwords.words('english')
[w for w in tokenizer_stemmer('runners like running thus they run') if
w not in stop]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[25]:
```
['runner', 'like', 'run', 'thu', 'run']
```

Transform Data Into Vectors

By utilizing the TFID vectorization, we get the frequency of words to understand the importance of the word. The TfidVectorizer "will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents." This is a way of understanding how important the words are. This is the final cleaning process before the model is implemented because after cleaning the data as a whole. This goes through and considers the value of the words itself now that we have not only eliminated any data that is of no value and have given the left over words a weight, we can understand the importance of the word as a whole and how it relates to the prediction model. We are left with two feature vectors, y and X which we will utilize in the Linear Regression model.

In [0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=True,
                        preprocessor=None,  # applied preprocessor in
Data Cleaning
                        tokenizer=tokenizer_stemmer,
                        use_idf=True,
                        norm='l2',
                        smooth_idf=True)

y = dataset.sentiment.values
X = tfidf.fit_transform(dataset.review)
```

Linear Regression

Really, this is the implementation of the Linear Regression model with the adjusted dataset. We implement our y and X feature vectors and our target vectors (training and testing data). We will utilize sklearn's LogisticRegressionCV which allows us to compute the linear regression model along with the cross-validation. In this case, we did a 5-fold cross validation.

In [27]:
```
from sklearn.model_selection import train_test_split
import pickle
from sklearn.linear_model import LogisticRegressionCV


X_train, X_test, y_train, y_test = train_test_split(X, y, random_state
=1, test_size=0.5, shuffle=False)

lrcv = LogisticRegressionCV(cv=5,
                            scoring='accuracy',
                            random_state=0,
                            n_jobs=-1,
                            verbose=3,
                            max_iter=300).fit(X_train, y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  2.5min finish
ed
```

In [0]:
```
saved_model = open('saved_model.sav', 'wb')
pickle.dump(lrcv, saved_model)
saved_model.close()
```

Model Evaluation

We find our accuracy score for our model to be 89.61%, which isn't too bad.

In [29]:
```
filename = 'saved_model.sav'
saved_lrcv = pickle.load(open(filename, 'rb'))
lrcv.score(X_test, y_test)
```

Out[29]:
```
0.89608
```

In [30]:
```
yhat = lrcv.predict(X_test)
yhat
```

Out[30]:
```
array([0, 1, 1, ..., 0, 1, 1])
```

Resources

- Sentiment Analysis With Python (https://towardsdatascience.com/sentiment-analysis-with-python-part-2-4f71e7bde59a)
- How to Clean Text for Machine Learning with Python (https://machinelearningmastery.com/clean-text-machine-learning-python/)
- How to Tokenize Tweets with Python (https://towardsdatascience.com/an-introduction-to-tweettokenizer-for-processing-tweets-9879389f8fe7)
- Building Blocks: Text Pre-Processing (https://towardsdatascience.com/building-blocks-text-pre-processing-641cae8ba3bf)
- Natural Language Processing: Text Data Vectorization (https://medium.com/@paritosh_30025/natural-language-processing-text-data-vectorization-af2520529cf7)
- How to Prepare Text Data for Machine Learning (https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/)

Dataset from Large Movie Review Dataset (http://ai.stanford.edu/~amaas/data/sentiment/)