Assignment: 3.2 Exercise: Sentiment Analysis and Preprocessing Text

Download the labeled training dataset from this link: Bag of Words Meets Bags of Popcorn.

Part 1: Using the TextBlob Sentiment Analyzer

1. Import the movie review data as a data frame and ensure that the data is loaded properly.
2. How many of each positive and negative reviews are there?
3. Use TextBlob to classify each movie review as positive or negative. Assume that a polarity score greater than or equal to zero is a positive sentiment and less than 0 is a negative sentiment.
4. Check the accuracy of this model. Is this model better than random guessing?
5. For up to five points extra credit, use another prebuilt text sentiment analyzer, e.g., VADER, and repeat steps (3) and (4).

Part 2: Prepping Text for a Custom Model

If you want to run your own model to classify text, it needs to be in proper form to do so. The following steps will outline a procedure to do this on the movie reviews text.

1. Convert all text to lowercase letters.
2. Remove punctuation and special characters from the text.
3. Remove stop words.
4. Apply NLTK's PorterStemmer.
5. Create a bag-of-words matrix from your stemmed text (output from (4)), where each row is a word-count vector for a single movie review (see sections 5.3 & 6.8 in the Machine Learning with Python Cookbook). Display the dimensions of your bag-of-words matrix. The number of rows in this matrix should be the same as the number of rows in your original data frame.
6. Create a term frequency-inverse document frequency (tf-idf) matrix from your stemmed text, for your movie reviews (see section 6.9 in the Machine Learning with Python Cookbook). Display the dimensions of your tf-idf matrix. These dimensions should be the same as your bag-of-words matrix.

Part 1: Using the TextBlob Sentiment Analyzer

In [1]:

```
#preloading necessary packages
import pandas as pd
import numpy as np
import re as re
from textblob import TextBlob
from textblob.classifiers import NaiveBayesClassifier
import nltk
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
import html
```

1. Import the movie review data as a data frame and ensure that the data is loaded properly.

In [2]:

```
#Loading data into data frame
```

```
train = pd.read_csv('labeledTrainData.tsv.zip', delimiter="\t")
```

In [3]:

```
#checking train data
train.head()
```

Out[3]:

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...

Data frame contents

File descriptions: labeledTrainData - The labeled training set. The file is tab-delimited and has a header row followed by 25,000 rows containing an id, sentiment, and text for each review.

Data fields:
id - Unique ID of each review
sentiment - Sentiment of the review; 1 for positive reviews and 0 for negative reviews
review - Text of the review

2. How many of each positive and negative reviews are there?

In [4]:

```
#Checking coungs of positive and negative reviews
print("Number of rows in the data set with positive reviews in dataset :", sum(train['sent
print("Number of rows in the data set with Negative reviews in dataset :", sum(train['sent
```

```
Number of rows in the data set with positive reviews in dataset : 12500
Number of rows in the data set with Negative reviews in dataset : 12500
```

3. Use TextBlob to classify each movie review as positive or negative. Assume that a polarity score greater than or equal to zero is a positive sentiment and less than 0 is a negative sentiment.

In [5]:

```
#Creating a column in data frame with TextBlob Sentiment analysis
train['sentimentTB'] = train['review'].apply(lambda review: TextBlob(review).sentiment.po
```

In [6]:

```
#Calculating positive and negative review sentiment analysis count by TextBlob
print("Number of rows in the data set with positive reviews in dataset per textBlob Analys
print("Number of rows in the data set with Negative reviews in dataset per textBlob Analys
```

```
Number of rows in the data set with positive reviews in dataset per textBlob Analysis : 19
017
Number of rows in the data set with Negative reviews in dataset per textBlob Analysis : 59
83
```

4. Check the accuracy of this model. Is this model better than random guessing?

In [7]:

```
#Calculating Accuracy of textBlob where labelled test data and textBlob preduction for se
print("Accurate positive sentiment prediction by textBlob :", sum((train['sentiment'] > 0)
print("Accurate negative sentiment prediction by textBlob :", sum((train['sentiment'] <= (
```

```
Accurate positive sentiment prediction by textBlob : 11824
Accurate negative sentiment prediction by textBlob : 5307
```

Total number of agreements by textBlob: 11824+5307 = 18483

Total number of samples: 25000

Accuracy of textBob = (18483/25000)*100 = 73.932%

Accuracy of textBob model is about 73.932%. This is definitely better than random guessing as it would be a 50% accurate model with either yes or no.

5. For up to five points extra credit, use another prebuilt text sentiment analyzer, e.g., VADER, and repeat steps (3) and (4).

In [8]:
```
#Importing vader module
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

In [9]:
```
#Creating a column with Vader sentiment analysis
pdanalyzer = SentimentIntensityAnalyzer()
train['sentimentV'] = train['review'].apply(lambda review: pdanalyzer.polarity_scores(revi
```

In [10]:
```
#Calculating positive and negative review sentiment analysis count by Vader
print("Number of rows in the data set with positive reviews in dataset per textBlob Analys
print("Number of rows in the data set with Negative reviews in dataset per textBlob Analys
```

```
Number of rows in the data set with positive reviews in dataset per textBlob Analysis : 16
611
Number of rows in the data set with Negative reviews in dataset per textBlob Analysis : 83
89
```

In [11]:
```
#Calculating Accuracy of textBlob where labelled test data and VADER preduction for sentin
print("Accurate positive sentiment prediction by textBlob :", sum((train['sentiment'] > 0)
print("Accurate negative sentiment prediction by textBlob :", sum((train['sentiment'] <= (
```

```
Accurate positive sentiment prediction by textBlob : 10731
Accurate negative sentiment prediction by textBlob : 6620
```

Total number of agreements by VADER: 10731+6620 = 17351

Total number of samples: 25000

Accuracy of VADER = (17351/25000)*100 = 69.404%

Accuracy of VADER model is about 68.524%. This is definitely better than random guessing as it would be a 50% accurate model with either yes or no.

Part 2: Prepping Text for a Custom Model

Clean Data

1. Convert all text to lowercase letters.
2. Remove punctuation and special characters from the text.
3. Remove stop words.

In [12]:
```
#creating function to clean text
def clean_text(text):
    text = BeautifulSoup(text).get_text() #beautifying text
    letters_only = re.sub("[^a-zA-Z]", " ", text) # clean the html charecters (non text)
    words = letters_only.lower().split()    # convert to lower text
    stops = set(stopwords.words("english")) # setting stop words to remove
    main_words = [w for w in words if not w in stops]
    return( " ".join( main_words ))
```

In [13]:
```
#applying clean function on the data frame and creating a new column with clean text
train['clean_review'] = train['review'].apply(clean_text)
```

4. Apply NLTK's PorterStemmer.

In [14]:
```
# import these modules
from nltk.stem import PorterStemmer
```

In [15]:
```
#Applying porterstemmer on clean_review
ps = PorterStemmer()
train['clean_review'] = train['clean_review'].apply(lambda review: ps.stem(review))
```

In [17]:
```
from nltk import word_tokenize # importing word_tokenize
#extracting and prinitng tokenized values sample
corpora = train['clean_review'].values
tokenized = [word_tokenize(corpus) for corpus in corpora]

print(tokenized[2222])
```

```
['go', 'immediately', 'rent', 'movie', 'bottom', 'shelf', 'local', 'video', 'store', 'cove
red', 'dust', 'one', 'touched', 'years', 'may', 'even', 'special', 'worth', 'ten', 'buck
s', 'swear', 'buy', 'many', 'films', 'compare', 'celluloid', 'version', 'goo', 'forms', 'b
ottom', 'trash', 'years', 'yes', 'gave', 'really', 'deserves', 'much', 'lower', 'scales',
'designed', 'stuff', 'like', 'mind']
```

In [18]:
```
#shape of train data frame
train.shape
```

Out[18]:
```
(25000, 6)
```

5. Create a bag-of-words matrix from your stemmed text (output from (4)), where each row is a word-count vector for a single movie review (see sections 5.3 & 6.8 in the Machine Learning with Python Cookbook). Display the dimensions of your bag-of-words matrix. The number of rows in this matrix should be the same as the number of rows in your original data frame.

In [19]:
```
#Creating bag_of_words matrix from clean review
count = CountVectorizer()
bag_of_words = count.fit_transform(train['clean_review'])
```

In [20]:
```
bag_of_words #Size of bag_of_words
```

Out[20]:
```
<25000x75529 sparse matrix of type '<class 'numpy.int64'>'
	with 2446144 stored elements in Compressed Sparse Row format>
```

Above shows that the number of rows is still 25000 (same as original dataframe train)

6. Create a term frequency-inverse document frequency (tf-idf) matrix from your stemmed text, for your movie reviews (see section 6.9 in the Machine Learning with Python Cookbook). Display the dimensions of your tf-idf matrix. These dimensions should be the same as your bag-of-words matrix.

In [21]:
```
# Import tf-idf encoding from sklearn library
from sklearn.feature_extraction.text import TfidfVectorizer

# Define some hiperparameters of encoded
vectorizer = TfidfVectorizer()

# Create the training set with the words encoded as features of the reviews
train_data_features = vectorizer.fit_transform(train['clean_review'])

print(train_data_features.shape)
```

```
(25000, 75529)
```

As above shows the shape (25000, 75529) matches the bag_of_words shape from above

Applying logistics regression model

In [22]:
```
# Import the logistic regression model from sklearn
from sklearn.linear_model import LogisticRegression

# Define the model
model = LogisticRegression(random_state=0, solver='lbfgs',
                            multi_class='multinomial')
# Train model
model.fit(train_data_features, train['sentiment'])
```

Out[22]:
```
LogisticRegression(multi_class='multinomial', random_state=0)
```

In [42]:
```
###### Testing the model against entire train data from origianl train data
```

In [33]:
```
# Read the test data
test = pd.read_csv("labeledTrainData.tsv.zip", header=0, delimiter="\t", \
                   quoting=3 )
print(test.shape)

# Clean the text of all reviews in the training set
print("Cleaning and parsing the test set movie reviews...\n")
test['clean_review'] = test['review'].apply(clean_text)

# Create the test set with the words encoded as features of the reviews
test_data_features = vectorizer.transform(test['clean_review'])


# Use the logistic regression model to make sentiment label predictions
result = model.predict(test_data_features)

# Copy the results to a pandas dataframe with an "id" column and a "sentiment" column
output = pd.DataFrame( data={"id":test["id"],"original_sentiment":test["sentiment"] ,"sent
output.head()
```

```
(25000, 3)
Cleaning and parsing the test set movie reviews...
```

Out[33]:

	id	original_sentiment	sentiment_custom
0	"5814_8"	1	1
1	"2381_9"	1	1
2	"7759_3"	0	0
3	"3630_4"	0	0
4	"9495_8"	1	0

In [36]:
```
#Calculating positive and negative review sentiment analysis count by my custom model
print("Number of rows in the data set with positive reviews in dataset per custom model :'
print("Number of rows in the data set with negative reviews in dataset per custom model :'
```

```
Number of rows in the data set with positive reviews in dataset per custom model : 12611
Number of rows in the data set with negative reviews in dataset per custom model : 12389
```

In [40]:
```
#Calculating Accuracy of custom model where labelled test data and VADER preduction for se
print("Accurate positive sentiment prediction by custom model :", sum((output['original_se
print("Accurate negative sentiment prediction by custom model :", sum((output['original_se
```

```
Accurate positive sentiment prediction by custom model : 11997
Accurate negative sentiment prediction by custom model : 11886
```

Total number of agreements by custom model: 11997+11886 = 23883

Total number of samples: 25000

Accuracy of VADER = (17351/25000)*100 = 95.532%

Accuracy of VADER model is about 95.532%. This is definitely better than random guessing as it would be a 50% accurate model with either yes or no.

Although the accuracy of the model is very high, it could be a resultant of train and test set being exactly same. This could be a situation of overfitting of the custom model.

performing prediction on test data in Kaggle from the challenge

In [43]:

```
# Read the test data
test2 = pd.read_csv("testData.tsv.zip", header=0, delimiter="\t", \
                    quoting=3 )
print(test2.shape)


# Clean the text of all reviews in the training set
print("Cleaning and parsing the test set movie reviews...\n")
test2['clean_review'] = test2['review'].apply(clean_text)

# Create the test set with the words encoded as features of the reviews
test_data_features = vectorizer.transform(test2['clean_review'])


# Use the logistic regression model to make sentiment label predictions
result = model.predict(test_data_features)

# Copy the results to a pandas dataframe with an "id" column and a "sentiment" column
output_test = pd.DataFrame( data={"id":test2["id"],"sentiment":result})
output_test.head()
```

```
(25000, 2)
Cleaning and parsing the test set movie reviews...
```

Out[43]:

	id	sentiment
0	"12311_10"	1
1	"8348_2"	0
2	"5828_4"	1
3	"7186_2"	1
4	"12128_7"	1

In [46]:

```
output_test.to_csv("test_result.csv", index=False, quoting=3 )
```

In [ ]: