Homework 8

Sentiment Analysis on IMDB movie reviews

Reference: https://github.com/ikhlaqsidhu/data-x/blob/master/07-tools-webscraping-crawling-nlp-sentiment-sc1t/notebook-nlp-sentiment-analysis-imdb-afo_v1.ipynb (https://github.com/ikhlaqsidhu/data-x/blob/master/07-tools-webscraping-crawling-nlp-sentiment-sc1t/notebook-nlp-sentiment-analysis-imdb-afo_v1.ipynb)

https://github.com/ikhlaqsidhu/data-x/blob/master/07a-tools-nlp-sentiment_add_missing_si/NLP1-slides_v2_afo.pdf (https://github.com/ikhlaqsidhu/data-x/blob/master/07a-tools-nlp-sentiment_add_missing_si/NLP1-slides_v2_afo.pdf)

REPO: https://github.com/anish-saha/IEOR135-SP19/tree/master/HW8 (https://github.com/anish-saha/IEOR135-SP19/tree/master/HW8)

Name:

Anish Saha

SID:

26071616

As you go through the notebook, you will encounter these main steps in the code:

1. Reading of file labeledTrainData.tsv from data folder in a dataframe `train`.
2. A function review_cleaner(train['review'],lemmatize,stem) which cleans the reviews in the input file.
3. A function train_predict_sentiment(cleaned_reviews, y=train["sentiment"],ngram=1,max_features=1000
4. You will see a model has been trained on unigrams of the reviews without lemmatizing and stemming.
5. Your task is in 5.TODO section.

Run the cells below-

In [1]:
```
# Remove warnings
import warnings
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
%matplotlib inline

#make compatible with Python 2 and Python 3
from __future__ import print_function, division, absolute_import
```

Data set

The labeled training data set consists of 25,000 IMDB movie reviews. There is also an unlabeled test set with 25,000 IMDB movie reviews. The sentiment of the reviews are binary, meaning an IMDB rating < 5 results in a sentiment score of 0, and a rating >=7 have a sentiment score of 1 (no reviews with score 5 or 6 are included in the analysis). No individual movie has more than 30 reviews.

File description

- **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.
- **testData** - The unlabeled test set. 25,000 rows containing an id, and text for each review.

Data columns

- **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

1. Data set statistics

In [2]:
```
import numpy as np
import pandas as pd
train = pd.read_csv("data/labeledTrainData.tsv", header=0, \
                    delimiter="\t", quoting=3)
# train.shape should be (25000,3)
```

In [3]:
```
train.head()
```

Out[3]:

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

In [4]:
```
# import packages

import bs4 as bs
import nltk

# nltk.download('all')
from nltk.tokenize import sent_tokenize # tokenizes sentences
import re

from nltk.stem import PorterStemmer
from nltk.tag import pos_tag
from nltk.corpus import stopwords
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

eng_stopwords = stopwords.words('english')
```

2.Preparing the data set for classification

We'll create a function called `review_cleaner` that reads in a review and:

- Removes HTML tags (using beautifulsoup)
- **Extract emoticons (emotion symbols, aka smileys :D)**
- Removes non-letters (using regular expression)
- Converts all words to lowercase letters and tokenizes them (using .split() method on the review strings, so that every word in the review is an element in a list)
- Removes all the English stopwords from the list of movie review words
- Join the words back into one string seperated by space, append the emoticons to the end

NOTE: Transform the list of stopwords to a set before removing the stopwords. I.e. assign `eng_stopwords = set(stopwords.words("english"))`. Use the set to look up stopwords. This will speed up the computations A LOT (Python is much quicker when searching a set than a list).

In [5]:
```
# 1.
from nltk.corpus import stopwords
from nltk.util import ngrams


ps = PorterStemmer()
wnl = WordNetLemmatizer()


def review_cleaner(reviews,lemmatize=True,stem=False):
    '''
    Clean and preprocess a review.

    1. Remove HTML tags
    2. Use regex to remove all special characters (only keep letters)
    3. Make strings to lower case and tokenize / word split reviews
    4. Remove English stopwords
    5. Rejoin to one string
    '''
    ps = PorterStemmer()
    wnl = WordNetLemmatizer()
        #1. Remove HTML tags

    cleaned_reviews=[]
    for i,review in enumerate(train['review']):
    # print progress
        if( (i+1)%500 == 0 ):
            print("Done with %d reviews" %(i+1))
        review = bs.BeautifulSoup(review).text

        #2. Use regex to find emoticons
        emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', review)

        #3. Remove punctuation
        review = re.sub("[^a-zA-Z]", " ",review)

        #4. Tokenize into words (all lower case)
        review = review.lower().split()

        #5. Remove stopwords
        eng_stopwords = set(stopwords.words("english"))

        clean_review=[]
        for word in review:
            if word not in eng_stopwords:
                if lemmatize is True:
                    word=wnl.lemmatize(word)
                elif stem is True:
                    if word == 'oed':
                        continue
                    word=ps.stem(word)
                clean_review.append(word)

        #6. Join the review to one sentence

        review_processed = ' '.join(clean_review+emoticons)
        cleaned_reviews.append(review_processed)
```

```
    return(cleaned_reviews)
```

3. Function to train and validate a sentiment analysis model using Random Forest Classifier

In [6]:
```
from sklearn.ensemble import RandomForestClassifier
# # CountVectorizer can actucally handle a lot of the preprocessing for
 us
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics # for confusion matrix, accuracy score etc
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix


np.random.seed(0)


def train_predict_sentiment(cleaned_reviews, y=train["sentiment"],ngram=
1,max_features=1000):
    '''This function will:
    1. split data into train and test set.
    2. get n-gram counts from cleaned reviews
    3. train a random forest model using train n-gram counts and y (labe
ls)
    4. test the model on your test split
    5. print accuracy of sentiment prediction on test and training data
    6. print confusion matrix on test data results

    To change n-gram type, set value of ngram argument
    To change the number of features you want the countvectorizer to gen
erate, set the value of max_features argument'''

    print("Creating the bag of words model!\n")
    # CountVectorizer" is scikit-learn's bag of words tool, here we show
more keywords
    vectorizer = CountVectorizer(ngram_range=(1, ngram),analyzer = "wor
d",   \
                                 tokenizer = None,    \
                                 preprocessor = None, \
                                 stop_words = None,   \
                                 max_features = max_features)

    X_train, X_test, y_train, y_test = train_test_split(\
    cleaned_reviews, y, random_state=0, test_size=.2)

    # Then we use fit_transform() to fit the model / learn the vocabular
y,
    # then transform the data into feature vectors.
    # The input should be a list of strings. .toarraty() converts to a n
umpy array

    train_bag = vectorizer.fit_transform(X_train).toarray()
    test_bag = vectorizer.transform(X_test).toarray()
#     print('TOP 20 FEATURES ARE: ',(vectorizer.get_feature_names()[:2
0]))


    print("Training the random forest classifier!\n")
    # Initialize a Random Forest classifier with 75 trees
    forest = RandomForestClassifier(n_estimators = 50)
```

```
    # Fit the forest to the training set, using the bag of words as
    # features and the sentiment labels as the target variable
    forest = forest.fit(train_bag, y_train)


    train_predictions = forest.predict(train_bag)
    test_predictions = forest.predict(test_bag)

    train_acc = metrics.accuracy_score(y_train, train_predictions)
    valid_acc = metrics.accuracy_score(y_test, test_predictions)
    print(" The training accuracy is: ", train_acc, "\n", "The validatio
n accuracy is: ", valid_acc)
    print()
    print('CONFUSION MATRIX:')
    print('         Predicted')
    print('          neg pos')
    print(' Actual')
    c=confusion_matrix(y_test, test_predictions)
    print('     neg  ',c[0])
    print('     pos  ',c[1])

    #Extract feature importnace
    print('\nTOP TEN IMPORTANT FEATURES:')
    importances = forest.feature_importances_
    indices = np.argsort(importances)[::-1]
    top_10 = indices[:10]
    print([vectorizer.get_feature_names()[ind] for ind in top_10])
```

4. Train and test Model on the IMDB data

In [7]:
```
# Clean the reviews in the training set 'train' using review_cleaner fun
ction defined above
# Here we use the original reviews without lemmatizing and stemming

original_clean_reviews=review_cleaner(train['review'],lemmatize=False,st
em=False)
train_predict_sentiment(cleaned_reviews=original_clean_reviews, y=train[
"sentiment"],ngram=1,max_features=1000)
```

```
Done with 500 reviews
Done with 1000 reviews
Done with 1500 reviews
Done with 2000 reviews
Done with 2500 reviews
Done with 3000 reviews
Done with 3500 reviews
Done with 4000 reviews
Done with 4500 reviews
Done with 5000 reviews
Done with 5500 reviews
Done with 6000 reviews
Done with 6500 reviews
Done with 7000 reviews
Done with 7500 reviews
Done with 8000 reviews
Done with 8500 reviews
Done with 9000 reviews
Done with 9500 reviews
Done with 10000 reviews
Done with 10500 reviews
Done with 11000 reviews
Done with 11500 reviews
Done with 12000 reviews
Done with 12500 reviews
Done with 13000 reviews
Done with 13500 reviews
Done with 14000 reviews
Done with 14500 reviews
Done with 15000 reviews
Done with 15500 reviews
Done with 16000 reviews
Done with 16500 reviews
Done with 17000 reviews
Done with 17500 reviews
Done with 18000 reviews
Done with 18500 reviews
Done with 19000 reviews
Done with 19500 reviews
Done with 20000 reviews
Done with 20500 reviews
Done with 21000 reviews
Done with 21500 reviews
Done with 22000 reviews
Done with 22500 reviews
Done with 23000 reviews
Done with 23500 reviews
Done with 24000 reviews
Done with 24500 reviews
Done with 25000 reviews
Creating the bag of words model!

Training the random forest classifier!

 The training accuracy is:  0.9999
 The validation accuracy is:  0.8216
```

```
CONFUSION MATRIX:
         Predicted
          neg pos
 Actual
     neg   [2102  446]
     pos   [ 446 2006]

TOP TEN IMPORTANT FEATURES:
['bad', 'worst', 'great', 'waste', 'awful', 'excellent', 'terrible', 'b
est', 'boring', 'worse']
```

5. TODO:

To do this exercise you only need to change argument values in the functions review_cleaner() and train_predict_semtiment(). Go through the functions to understand what they do. Perform the following -

1. For **UNIGRAM setting** ie. when ngram=1 in the function `train_predict_sentiment()`, compare the performance of original cleaned reviews in Sentiment anlysis to -
   A. lemmatized reviews
   B. stemmed reviews
2. For **BIGRAM setting** ie. when ngram=2 in the function `train_predict_sentiment()`, compare the performance of original cleaned reviews in sentiment analysis to:
   A. lemmatized reviews
   B. stemmed reviews
3. For **UNIGRAM setting** ie. ngram=1 and lemmatize = True , compare the performance of Sentiment analysis for these different values of maximum features = [10,100,1000,5000], you can change the value of argument max_features in `train_predict_sentiment()

SUBMISSION: For each question in 5. TODO report your results in a PDF.

Mention the review_cleaner( ) and train_predict_sentiment( ) argument setting that you used in each case. Do not submit any ipython notebook.

Example : For original review with unigram and 5000 max_features, I will report:

```
original_clean_reviews=review_cleaner(train['review'],lemmatize=False,stem=F
alse)
train_predict_sentiment(cleaned_reviews=original_clean_reviews, y=train["sen
timent"],ngram=1,max_features=5000)

The training accuracy is:  1.0
The validation accuracy is:  0.836
```

Also write a 100-200 word summary of your observations overall.

In [8]:
```
lemmatized_clean_reviews=review_cleaner(train['review'],lemmatize=True,s
tem=False)
stemmed_clean_reviews=review_cleaner(train['review'],lemmatize=False,ste
m=True)
```

```
Done with 500 reviews
Done with 1000 reviews
Done with 1500 reviews
Done with 2000 reviews
Done with 2500 reviews
Done with 3000 reviews
Done with 3500 reviews
Done with 4000 reviews
Done with 4500 reviews
Done with 5000 reviews
Done with 5500 reviews
Done with 6000 reviews
Done with 6500 reviews
Done with 7000 reviews
Done with 7500 reviews
Done with 8000 reviews
Done with 8500 reviews
Done with 9000 reviews
Done with 9500 reviews
Done with 10000 reviews
Done with 10500 reviews
Done with 11000 reviews
Done with 11500 reviews
Done with 12000 reviews
Done with 12500 reviews
Done with 13000 reviews
Done with 13500 reviews
Done with 14000 reviews
Done with 14500 reviews
Done with 15000 reviews
Done with 15500 reviews
Done with 16000 reviews
Done with 16500 reviews
Done with 17000 reviews
Done with 17500 reviews
Done with 18000 reviews
Done with 18500 reviews
Done with 19000 reviews
Done with 19500 reviews
Done with 20000 reviews
Done with 20500 reviews
Done with 21000 reviews
Done with 21500 reviews
Done with 22000 reviews
Done with 22500 reviews
Done with 23000 reviews
Done with 23500 reviews
Done with 24000 reviews
Done with 24500 reviews
Done with 25000 reviews
Done with 500 reviews
Done with 1000 reviews
Done with 1500 reviews
Done with 2000 reviews
Done with 2500 reviews
Done with 3000 reviews
Done with 3500 reviews
```

```
Done with 4000 reviews
Done with 4500 reviews
Done with 5000 reviews
Done with 5500 reviews
Done with 6000 reviews
Done with 6500 reviews
Done with 7000 reviews
Done with 7500 reviews
Done with 8000 reviews
Done with 8500 reviews
Done with 9000 reviews
Done with 9500 reviews
Done with 10000 reviews
Done with 10500 reviews
Done with 11000 reviews
Done with 11500 reviews
Done with 12000 reviews
Done with 12500 reviews
Done with 13000 reviews
Done with 13500 reviews
Done with 14000 reviews
Done with 14500 reviews
Done with 15000 reviews
Done with 15500 reviews
Done with 16000 reviews
Done with 16500 reviews
Done with 17000 reviews
Done with 17500 reviews
Done with 18000 reviews
Done with 18500 reviews
Done with 19000 reviews
Done with 19500 reviews
Done with 20000 reviews
Done with 20500 reviews
Done with 21000 reviews
Done with 21500 reviews
Done with 22000 reviews
Done with 22500 reviews
Done with 23000 reviews
Done with 23500 reviews
Done with 24000 reviews
Done with 24500 reviews
Done with 25000 reviews
```

In [9]:
```
# Question 1a
train_predict_sentiment(cleaned_reviews=lemmatized_clean_reviews, y=trai
n["sentiment"], ngram=1)
```

```
Creating the bag of words model!

Training the random forest classifier!

 The training accuracy is:  0.99995
 The validation accuracy is:  0.8314

CONFUSION MATRIX:
         Predicted
          neg pos
 Actual
     neg   [2121  427]
     pos   [ 416 2036]

TOP TEN IMPORTANT FEATURES:
['bad', 'worst', 'great', 'waste', 'awful', 'excellent', 'best', 'terri
ble', 'boring', 'nothing']
```

In [10]:
```
# Question 1b
train_predict_sentiment(cleaned_reviews=stemmed_clean_reviews, y=train[
"sentiment"], ngram=1)
```

```
Creating the bag of words model!

Training the random forest classifier!

 The training accuracy is:  1.0
 The validation accuracy is:  0.819

CONFUSION MATRIX:
         Predicted
          neg pos
 Actual
     neg   [2100  448]
     pos   [ 457 1995]

TOP TEN IMPORTANT FEATURES:
['bad', 'worst', 'wast', 'great', 'aw', 'love', 'excel', 'bore', 'terri
bl', 'best']
```

In [11]:
```
# Question 2a
train_predict_sentiment(cleaned_reviews=lemmatized_clean_reviews, y=trai
n["sentiment"], ngram=2)
```

```
Creating the bag of words model!

Training the random forest classifier!

 The training accuracy is:  1.0
 The validation accuracy is:  0.82

CONFUSION MATRIX:
         Predicted
          neg pos
 Actual
     neg   [2118  430]
     pos   [ 470 1982]

TOP TEN IMPORTANT FEATURES:
['bad', 'worst', 'great', 'awful', 'waste', 'boring', 'terrible', 'exce
llent', 'worse', 'nothing']
```

In [12]:
```
# Question 2b
train_predict_sentiment(cleaned_reviews=stemmed_clean_reviews, y=train[
"sentiment"], ngram=2)
```

```
Creating the bag of words model!

Training the random forest classifier!

 The training accuracy is:  1.0
 The validation accuracy is:  0.825

CONFUSION MATRIX:
         Predicted
          neg pos
 Actual
     neg   [2111  437]
     pos   [ 438 2014]

TOP TEN IMPORTANT FEATURES:
['bad', 'wast', 'worst', 'great', 'aw', 'bore', 'excel', 'love', 'terri
bl', 'stupid']
```

In [13]:
```
# Question 3a
train_predict_sentiment(cleaned_reviews=lemmatized_clean_reviews, y=trai
n["sentiment"], ngram=1, max_features=10)
```

```
Creating the bag of words model!

Training the random forest classifier!

 The training accuracy is:  0.8714
 The validation accuracy is:  0.5638

CONFUSION MATRIX:
         Predicted
          neg pos
 Actual
     neg   [1433 1115]
     pos   [1066 1386]

TOP TEN IMPORTANT FEATURES:
['film', 'movie', 'one', 'good', 'character', 'time', 'like', 'get', 's
tory', 'even']
```

In [14]:
```
# Question 3b
train_predict_sentiment(cleaned_reviews=lemmatized_clean_reviews, y=trai
n["sentiment"], ngram=1, max_features=100)
```

```
Creating the bag of words model!

Training the random forest classifier!

 The training accuracy is:  0.9999
 The validation accuracy is:  0.7198

CONFUSION MATRIX:
         Predicted
          neg pos
 Actual
     neg   [1850  698]
     pos   [ 703 1749]

TOP TEN IMPORTANT FEATURES:
['bad', 'great', 'movie', 'film', 'one', 'best', 'even', 'like', 'nothi
ng', 'love']
```

In [15]:
```
# Question 3c
train_predict_sentiment(cleaned_reviews=lemmatized_clean_reviews, y=trai
n["sentiment"], ngram=1, max_features=1000)
```

```
Creating the bag of words model!

Training the random forest classifier!

 The training accuracy is:  0.99995
 The validation accuracy is:  0.8204

CONFUSION MATRIX:
         Predicted
          neg pos
 Actual
     neg   [2111  437]
     pos   [ 461 1991]

TOP TEN IMPORTANT FEATURES:
['bad', 'worst', 'great', 'waste', 'awful', 'excellent', 'boring', 'wor
se', 'best', 'terrible']
```

In [16]:
```
# Question 3d
train_predict_sentiment(cleaned_reviews=lemmatized_clean_reviews, y=trai
n["sentiment"], ngram=1, max_features=5000)
```

```
Creating the bag of words model!

Training the random forest classifier!

 The training accuracy is:  1.0
 The validation accuracy is:  0.8452

CONFUSION MATRIX:
         Predicted
          neg pos
 Actual
     neg   [2191  357]
     pos   [ 417 2035]

TOP TEN IMPORTANT FEATURES:
['worst', 'bad', 'great', 'waste', 'awful', 'excellent', 'wonderful',
'boring', 'terrible', 'best']
```

Observations

Part 1

a) The performance of original cleaned reviews in sentiment analysis using the unigram model for lemmatized reviews achieved a training accuracy of 99.995%, and a validation (test set) accuracy of 83.14%.

b) The performance of original cleaned reviews in sentiment analysis using the unigram model for stemmed reviews achieved a training accuracy of 100.0%, and a validation (test set) accuracy of 81.9%.

Overall, it seems that that the models using the unigram setting had relatively similar performance regardless of whether the original reviews were cleaned using lemmatization or stemming. The performance metrics indicate that the models performed relatively well using the unigram setting.

Part 2

a) The performance of original cleaned reviews in sentiment analysis using the bigram model for lemmatized reviews achieved a training accuracy of 100.0%, and a validation (test set) accuracy of 82.0%.

b) The performance of original cleaned reviews in sentiment analysis using the bigram model for stemmed reviews achieved a training accuracy of 100.0%, and a validation (test set) accuracy of 82.5%.

Overall, it seems that the models using the unigram setting had relatively similar performance regardless of whether the original reviews were cleaned using lemmatization or stemming, although stemming proved to have slightly better results. The performance metrics indicate that the models performed relatively well using the bigram setting.

Part 3

a) With max_features=10, the performance of original cleaned reviews in sentiment analysis using the unigram model for lemmatized reviews achieved a training accuracy of 87.14% and a validation (test set) accuracy of 56.38%

b) With max_features=100, the performance of original cleaned reviews in sentiment analysis using the unigram model for lemmatized reviews achieved a training accuracy of 99.99% and a validation (test set) accuracy of 71.98%.

c) With max_features=1000, the performance of original cleaned reviews in sentiment analysis using the unigram model for lemmatized reviews achieved a training accuracy of 99.995% and a validation (test set) accuracy of 82.04%.

d) With max_features=5000, the performance of original cleaned reviews in sentiment analysis using the unigram model for lemmatized reviews achieved a training accuracy of 100.0% and a validation (test set) accuracy of 84.52%.

Overall, it seems that as the value of the parameter `max_features` increases, the performance for the unigram model improves. With more features, the model's predictions are more accurate. This, however, may be prone to bias, as using more features with the random forest classifier may not increase accuracy for a different dataset. As such, bias-variance tradeoffs would need to be considered when tackling this problem.