Homework 3 (Due: 10/31/2018) Yifei Pei W1468299

COEN 281, Fall 2018
Professor Marwah

The objective of this HW is to implement a Naive Bayes classifier to predict whether a tweet was posted by a Republican or Democrat politician. The training data consist of about 13K tweets collected before the 2006 US presidential elections, There are about an equal number of Republican and Democrat tweets, and the tweets belong to three republican and three democrat twitter accounts.

To represent each tweet, we will use a commonly used model in natural language processing called 'bag of words' model. A bag of words representation of a document (tweet here) consists of words and their frequencies in the document. The order of words is ignored.

There four main tasks.

1. Tokenization: Parsing and converting the tweets to tokens. [**This is already done for you**]
2. Feature matrix construction from the training data set
3. Learning Naive Bayes parameters, priors and likelihoods, from the feature matrix.
4. Using the learned NB model to predict the labels of the test data set (about 4K tweets).

Tokenization

This task consists of converting each tweet into a sequence of "tokens" that can be used as features. Tokens are essentially characters and character sequences obtained after using white space as a separator. A lot these are noise that we want to remove; some are words or other character sequences that are useful features. A python package called *NLTK* (natural language toolkit) contains several tokenizers, including one for tweets. We use that tokenizer; in addition we do the following:

- remove stopwords. These are words that are frequently used in a language but do not carry any semantic information, e.g., the, an , a, this, is, was, etc.
- make all tokens lower case (this is done by the tweet tokenizer)
- removing twitter handles (again, done by the tweet tokenizer)
- remove punctuations, http links

Finally, we "lemmatize" the tokens. That means we convert different forms of a word to a common basic form, so that they can be recognized as the same work. E.g., vote, votes, voted would all be converted to vote; geese would be converted to goose,e tc. (There is a less sophisticated version of lemmatizer called a stemmer which just chops words to convert to the same base work; it doesn't work as well as a lemmatizer and we dont use it here.) There is a good description of the NLTK tokenizer here.

The output of this part is a cleaned up list of tokens for each tweet.

In [1]:
```
import pandas as pd
import string
import numpy as np

import nltk
#
# you may need to run the following
nltk.download('stopwords')
nltk.download('wordnet')
```

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

Out[1]:
```
True
```

In [2]:
```
# The data set has two columns - screen_name and text (which is the raw tweet)

## load tweets
tweets = pd.read_csv("tweets_train.csv", na_filter=False)

## screen_namee (accounts)
#  democrat - hillary, time kaine, TheDemocrats
# republicans - trunp, pence, GOP
```

In [3]:
```
tweets['screen_name'].unique()
```

Out[3]:
```
array(['GOP', 'TheDemocrats', 'HillaryClinton', 'timkaine', 'mike_pence',
       'realDonaldTrump'], dtype=object)
```

In [4]:
```
tweets.head()
```

Out[4]:

	screen_name	text
0	GOP	RT @GOPconvention: #Oregon votes today. That m...
1	TheDemocrats	RT @DWStweets: The choice for 2016 is clear: W...
2	HillaryClinton	Trump's calling for trillion dollar tax cuts f...
3	HillaryClinton	.@TimKaine's guiding principle: the belief tha...
4	timkaine	Glad the Senate could pass a #THUD / MilCon / ...

In [5]:
```
tweets.describe()
```

Out[5]:

	screen_name	text
count	13000	13000
unique	6	12982
top	realDonaldTrump	MAKE AMERICA GREAT AGAIN!
freq	2217	4

In [6]:
```
# add labels
# 1 for D's
# 0 for R's
tweets['label'] = tweets['screen_name'].str.contains('TheDemocrats|HillaryClinton|timkaine', regex=True)
tweets.describe()
```

Out[6]:

	screen_name	text	label
count	13000	13000	13000
unique	6	12982	2
top	realDonaldTrump	MAKE AMERICA GREAT AGAIN!	False
freq	2217	4	6554

The training data has 13K tweets, and each of the two classes have about an equal number of tweets.

Now we will define our tokenizer.

In [7]:
```
from nltk.stem import WordNetLemmatizer
#
#  Input : dataframe with a column names 'text' which contains raw tweets (one per row)
#  Output: A list of lists of tokens corrsponding to the 'text' column
#
def tokenize_tweets2(tweets):
    """Given a df with tweets in 'text' col, this function return tokens as a list of lists"""

    # apply tokenize to the 'text' coolumn in the tweets df
    tweet_tokenizer = nltk.tokenize.TweetTokenizer(preserve_case=False, reduce_len=True, strip_handles=True)
    tokens = tweets['text'].apply(tweet_tokenizer.tokenize)

    # filter
    misc = ['rt', '’', '…', '—', 'u', '”', 'w', '“', '...', '', 'http', 'https']
    to_remove = nltk.corpus.stopwords.words('English') + list(string.punctuation) + misc

    lemmatizer = WordNetLemmatizer()

    tokens = [[lemmatizer.lemmatize(token) for token in tw if token not in to_remove] for tw in tokens]
    return(tokens)
```

In [8]:
```
all_tokens = tokenize_tweets2(tweets)
print(len(all_tokens))
all_tokens[:10]
```

```
13000
```

Out[8]:
```
[['#oregon', 'vote', 'today', 'mean', '62', 'day', 'https://t.co/OoH9FVb7QS'],
 ['choice',
  '2016',
  'clear',
  'need',
  'another',
  'democrat',
  'white',
  'house',
  '#demdebate',
```

```
 '#wearedemocrats',
 'http://t.co/0n5g0YN46f'],
["trump's",
 'calling',
 'trillion',
 'dollar',
 'tax',
 'cut',
 'wall',
 'street',
 'time',
 'pay',
 'fair',
 'share',
 'https://t.co/y8vyESIOES'],
['guiding',
 'principle',
 'belief',
 'make',
 'difference',
 'public',
 'service',
 'https://t.co/YopSUeMqOX'],
['glad',
 'senate',
 'could',
 'pas',
 '#thud',
 'milcon',
 'vetaffairs',
 'approps',
 'bill',
 'solid',
 'provision',
 'virginia',
 'https://t.co/NxIgRC3hDi'],
['exclusive',
 'sits',
 'see',
 'sunday',
 'morning',
 '8:',
 '30a',
 'rtv',
 '6',
 'rtv',
 '6',
 'app'],
['chatham',
 'town',
 'council',
 'congress',
 'made',
 'strong',
 'mark',
 'community',
 'proud',
 'work',
 'together',
 'behalf',
 'va'],
['thank',
 'new',
 'orleans',
 'louisiana',
 '#makeamericagreatagain',
 '#votetrump',
 'https://t.co/tI1h9xT9GX',
 'https://t.co/0bf7BOlWEj'],
['happy', '241st', 'birthday', 'thank', 'https://t.co/mXsxkfcstC'],
['excited',
 'announce',
 'today',
 'plan',
 'build',
 'indiana',
 'neuro-diagnostic',
 'institute',
 'partnership',
 'https://t.co/hy4yRC…']]
```

The tokenizer can still be improved, but we will go with this.

Let's find the most common tokens, and we will use all tokens that at least occur 25 times as features.

In [9]:
```
from collections import Counter

counts = Counter([token for tokens in all_tokens for token in tokens])
print(len(counts))
counts.most_common(20)
```

```
23459
```

Out[9]:
```
[('hillary', 1159),
 ('trump', 1144),
 ('great', 749),
 ('clinton', 720),
 ('today', 709),
 ('make', 581),
 ('donald', 576),
 ('president', 564),
 ('day', 552),
 ('thank', 539),
 ('american', 512),
 ('new', 503),
 ('job', 503),
 ('u', 485),
 ('america', 480),
 ('people', 469),
 ('vote', 451),
 ('state', 442),
 ('get', 420),
 ('year', 415)]
```

In [10]:
```
top_words = [k for k in counts.keys() if counts.get(k) > 25]
len(top_words)
```

Out[10]:
```
927
```

top_words are our features. Now let's construct a feature matrix from these top words

Feature Martix Construction

Problem 1 (15 points) Compute feature matrix

Now we will extract the features from the training data and construct a feature matrix. The bad news is this matrix can be very large. In our case it is about 13K X 1K, or about 13M x 4 bytes ~ 52M, which will easily fit in the RAM of your laptops, but the training set could have easily been 10x or 100x the current size, and the number of features 10x in which case you would be out of luck. The good news is this matrix is likely to be very sparse. In fact, each tweet is not likely to contain more than 10-20 tokens, so even if this matrix becomes very large, we would be okay if we use a sparse representation.

In a sparse representation, only the non-zero entities and their indices are saved. Scipy provides several formats for sparse matrices. In this assignment, it doesn't matter which one you use (in fact, we could have even used a dense matrix). However, since we have to sum along columns (or features), the most suitable one is csc (or compressed sparse column) format.

To make it easier to estimate priors and likelihoods, we will construct two feature matrices - one for each for the two classes. For this, first we need to figure out how many data points are in each class.

While setting elements of a csc_matrix you may get a 'SparseEfficiencyWarning'; you can ignore that.

In [11]:
```
num_feat = len(top_words)

# set this to the correct values
nTrainR = tweets['screen_name'].str.contains('GOP|mike_pence|realDonaldTrump', regex=True).sum()  # number of R (0) tra
nTrainD = tweets['screen_name'].str.contains('TheDemocrats|HillaryClinton|timkaine', regex=True).sum()  # number of D (

# create sparse feature matrix
from scipy.sparse import csc_matrix

rfmat = csc_matrix((nTrainR, num_feat), dtype=int)
dfmat = csc_matrix((nTrainD, num_feat), dtype=int)

#
# populate rfmat and dfmat with the counts of the features
# Remember: all tokens are not features
#
# a function that might be useful is <list>.index()
#
rfmatarray = rfmat.toarray()
dfmatarray = dfmat.toarray()
```

In [12]:
```
name = tweets['screen_name'].str.contains('GOP|mike_pence|realDonaldTrump', regex=True)
rnm = 0
dnm = 0
for i in range(len(tweets)):
    if name[i]:
        for j in range(len(top_words)):
            rfmatarray[rnm][j] = all_tokens[i].count(top_words[j])
        rnm = rnm + 1
    else:
        for j in range(len(top_words)):
            dfmatarray[dnm][j] = all_tokens[i].count(top_words[j])
        dnm = dnm +1
```

Learning Naive Bayes Model Parameters ¶

Problem 2 (5 points) compute log priors

Problem 3 (30 points) compute log likelihoods using Laplace smoothing

Now we can compute the model parameters, this is, the likelihoods and priors for the two classes. As we discussed in class, since the probabilities can be very small numbers, we will compute log likelihoods and log priors. Aslo use Laplace (aka add one) smoothing.

To sum a matrix column, you can use something like dfmat[:,i].sum()

In [13]:
```
# compute log priors
from math import log2
log_prior_republican = log2(nTrainR/len(tweets))
log_prior_democrat = log2(nTrainD/len(tweets))
# compute log likelihoods
log_likelihoods_R = np.zeros((927,2))
log_likelihoods_D = np.zeros((927,2))

for i in range(927):
    log_likelihoods_R[i,0] = log2((rfmatarray[:,i].sum() + 1)/(rfmatarray.shape[0] + 2))
    log_likelihoods_R[i,1] = log2(1- (rfmatarray[:,i].sum() + 1)/(rfmatarray.shape[0] + 2))
    log_likelihoods_D[i,0] = log2((dfmatarray[:,i].sum() + 1)/(dfmatarray.shape[0] + 2))
    log_likelihoods_D[i,1] = log2(1- (dfmatarray[:,i].sum() + 1)/(dfmatarray.shape[0] + 2))
```

```
print('The log prior for Republican:', log_prior_republican)
print('The log prior for Democrat:', log_prior_democrat)
print('The log likelihoods for Republican:', log_likelihoods_R[:,0])
print('The log likelihoods for Democrat:', log_likelihoods_D[:,0])
```

```
The log prior for Republican: -0.9880640452604194
The log prior for Democrat: -1.012035529743634
The log likelihoods for Republican: [ -5.40247573  -4.19478436  -8.43067263  -4.2777207   -8.03474395
  -6.04924352  -7.63420602  -6.10874453  -6.46914677  -6.54931712
  -7.50867514  -6.63420602  -5.73608563 -12.67860014  -8.5911373
  -8.77170954  -8.5911373   -6.33875014  -8.50867514  -7.54931712
  -8.43067263  -5.09363764  -7.92371264  -7.82061914  -8.03474395
  -9.35667204  -4.75378764  -8.43067263  -7.21916852  -8.77170954
  -7.54931712  -7.87124522  -9.67860014  -6.67860014  -7.82061914
  -5.85842118  -6.43067263  -8.67860014  -7.97816042  -8.5911373
  -8.97816042  -6.21916852  -6.61251095  -7.77170954  -7.39319792
  -6.06389029  -6.2692092   -6.48877558  -8.21916852  -3.87770024
  -4.2777207   -5.02754845  -7.03474395  -6.57007568  -7.63420602
  -7.50867514  -7.77170954  -6.02038866  -7.92371264  -4.30791273
  -8.87124522  -5.89724043  -6.06389029  -4.8268511   -6.82061914
  -8.03474395  -9.35667204  -4.77170954 -12.67860014  -8.50867514
  -8.03474395  -6.5911373   -8.03474395  -9.09363764  -7.43067263
  -8.03474395  -5.64517714  -5.35667204  -5.52885302  -6.67860014
 -12.67860014  -6.63420602  -7.5911373   -5.58056806  -6.13944133
```

Prediction on Test Set

Now we have a trained Naive Bayes classifier. We will load the test data set and make the predictions. Note: If a token is not a feature, ignore it.

Problem 4 (5 points) Load test data and tokenize

Problem 5 (30 points) Using the trained NB classifier predict the labels

Problem 6 (5 points) Calculate accuracy, recall, and precision of your predictions

In [14]:
```
# Load test data and tokenize.
tweets_test = pd.read_csv("tweets_test.csv", na_filter=False)
all_tokens_test = tokenize_tweets2(tweets_test)
print(len(all_tokens_test))
```

```
4298
```

In [15]:
```
#Initialize a 4298*927 numpy array
test = np.zeros((4298,927), dtype=int)
```

In [16]:
```
#Count the features appeared in every sentences.
for i in range(4298):
    for j in range(len(top_words)):
        for element in all_tokens_test[i]:
            if element == top_words[j]:
                test[i][j] = test[i][j]+1
```

In [17]:
```
# For test dataset, I only emphasize whether it's token or not token. So I initialize test_record array.
# Number of rows is 4298. Number of solumns is 930. The first 927 columns are recording the 927 features. The last
# three columns for posteriori probability of republican, posteriori probability of democrat, and result label.
test_record = np.zeros((4298,930), dtype=int)
for i in range(4298):
    for j in range(len(top_words)):
        if test[i][j] >=1:
            test_record[i][j] =1
```

In [18]:
```
#Initialize the posteriori probability of republican, posteriori probability of democrat to log_prior
test_record[:,927] = log_prior_republican
test_record[:,928] = log_prior_democrat

for i in range(4298):
    for j in range(927):
        if test_record[i][j] == 1:
            test_record[i][927] = test_record[i][927] + log_likelihoods_R[j][0]
            test_record[i][928] = test_record[i][928] + log_likelihoods_D[j][0]
        else:
            test_record[i][927] = test_record[i][927] + log_likelihoods_R[j][1]
            test_record[i][928] = test_record[i][928] + log_likelihoods_D[j][1]
```

In [19]:
```
# Compare the posteriori probabilities and get the labels. 0 for republican. 1 for democrat
for i in range(4298):
    if test_record[i][927] > test_record[i][928]:
        test_record[i][929] = 0
    else:
        test_record[i][929] = 1
```

In [20]:
```
# Real labels saved in test_actual_label list
label = tweets_test['screen_name']
test_actual_label = []

for i in range(4298):
    if label[i] == 'TheDemocrats' or label[i] == 'HillaryClinton' or label[i] == 'timkaine':
        test_actual_label.append(1)
    else:
        test_actual_label.append(0)
```

In [21]:
```
test_predict_label = test_record[:,-1]
TP = 0
FP = 0
FN = 0
TN = 0
for i in range(4298):
    if test_actual_label[i] == 1 and test_predict_label[i] == 1:
        TP = TP +1
    elif test_actual_label[i] == 1 and test_predict_label[i] == 0:
        FN = FN +1
    elif test_actual_label[i] == 0 and test_predict_label[i] == 1:
        FP = FP +1
    else:
        TN = TN +1
```

In [22]:
```
Accuracy = (TP + TN)/(TP + TN + FP + FN)
Recall = TP/(TP+FN)
Precision = TP/(TP+FP)
print("Accuracy: ",Accuracy)
print("Recall:", Recall)
print("Precision:", Precision)
```

```
Accuracy:  0.8024662633783155
Recall: 0.7896645512239348
Precision: 0.8189938881053126
```

Problem 7 (5 points) List the features with top ten likelihoods for each of the two classes. What is the likelihood for 'hillary', that is, P(hillary|class)? Is it in the top ten? How important is it in this classification problem?

Solution:

In [23]:
```
'''
Based on the results below, 'hillary' is in top ten likelihoods features. But it is not important in this
classification problem because the difference between P(hillary|Republican) and P(hillary|Democrat) is quite small.
'''
```

Out[23]:
```
"\nBased on the results below, 'hillary' is in top ten likelihoods features. But it is not important in this \nclassi
fication problem because the difference between P(hillary|Republican) and P(hillary|Democrat) is quite small.\n"
```

In [24]:
```
log_likelihoods_R_list = log_likelihoods_R[:,0].tolist()
log_likelihoods_D_list = log_likelihoods_D[:,0].tolist()
```

In [25]:
```
a = np.argsort(log_likelihoods_R_list)[::-1][0:10]
b = np.argsort(log_likelihoods_D_list)[::-1][0:10]
```

In [26]:
```
top_ten_likelihoods_R_feature = []
top_ten_likelihoods_D_feature = []
for i in range(10):
    top_ten_likelihoods_R_feature.append(top_words[a[i]])
    top_ten_likelihoods_D_feature.append(top_words[b[i]])
```

In [27]:
```
print('The features with top ten likelihoods for Republican:',top_ten_likelihoods_R_feature)
print('The features with top ten likelihoods for Democrats:',top_ten_likelihoods_D_feature)
```

```
The features with top ten likelihoods for Republican: ['clinton', 'hillary', 'great', 'thank', 'today', 'new', 'day',
'indiana', 'job', 'state']
The features with top ten likelihoods for Democrats: ['trump', 'hillary', 'donald', 'president', 'today', 'american',
'make', 'u', 'vote', 'one']
```

In [28]:
```
#Get the index of 'hillary' in the top_words
for i in range(927):
    if top_words[i] == 'hillary':
        break
```

In [29]:
```
print('The log2 possibility of hillary in Republican is',log_likelihoods_R_list[i])
print('The log2 possibility of hillary in Democrat is', log_likelihoods_D_list[i])
print('P(hillary|Republican)', 2**log_likelihoods_R_list[i])
print('P(hillary|Democrat)', 2**log_likelihoods_D_list[i])
```

```
The log2 possibility of hillary in Republican is -3.4789277942630945
The log2 possibility of hillary in Democrat is -3.4922446997710623
P(hillary|Republican) 0.0896888346552776
P(hillary|Democrat) 0.08886476426799009
```

Problem 8 (5 points) How important are the priors in this problem?

Solution:

In [30]:
```
'''
From the results below, the priors are not important in this problem because the difference is small.
'''
```

Out[30]:
```
'\nFrom the results below, the priors are not important in this problem because the difference is small. \n'
```

In [31]:
```
print("log_prior_republican is", log_prior_republican)
print("log_prior_democrat is", log_prior_democrat)
```

```
log_prior_republican is -0.9880640452604194
log_prior_democrat is -1.012035529743634
```

Extra credit (5 points): Compute the accuracy of the test set without Laplace smoothing and compare with the above.

In [32]:
```
'''
Based on the solution below, after comparing the two accuracies (below and above), I find the accuracies dropped a lot
from 0.8024662633783155 (with Laplace smoothing) to 0.5132619823173569 (without Laplace smoothing).
'''
```

Out[32]:
```
'\nBased on the solution below, after comparing the two accuracies (below and above), I find the accuracies dropped a
lot \nfrom 0.8024662633783155 (with Laplace smoothing) to 0.5132619823173569 (without Laplace smoothing).\n'
```

In [33]:
```
prior_republican = nTrainR/len(tweets)
prior_democrat = nTrainD/len(tweets)
# compute log likelihoods
likelihoods_R = np.zeros((927,2))
likelihoods_D = np.zeros((927,2))

for i in range(927):
    likelihoods_R[i,0] = (rfmatarray[:,i].sum() )/(rfmatarray.shape[0])
    likelihoods_R[i,1] = 1- (rfmatarray[:,i].sum())/(rfmatarray.shape[0])
    likelihoods_D[i,0] = (dfmatarray[:,i].sum())/(dfmatarray.shape[0])
    likelihoods_D[i,1] = 1- (dfmatarray[:,i].sum())/(dfmatarray.shape[0])
```

In [34]:
```
#Initialize the posteriori probability of republican, posteriori probability of democrat to priors
test_record[:,927] = prior_republican
test_record[:,928] = prior_democrat

for i in range(4298):
    for j in range(927):
        if test_record[i][j] == 1:
            test_record[i][927] = test_record[i][927] * likelihoods_R[j][0]
            test_record[i][928] = test_record[i][928] * likelihoods_D[j][0]
        else:
            test_record[i][927] = test_record[i][927] * likelihoods_R[j][1]
            test_record[i][928] = test_record[i][928] * likelihoods_D[j][1]
```

In [35]:
```
# Compare the posteriori probabilities and get the labels. 0 for republican. 1 for democrat
for i in range(4298):
    if test_record[i][927] > test_record[i][928]:
        test_record[i][929] = 0
    else:
        test_record[i][929] = 1
```

In [36]:
```
test_predict_label = test_record[:,-1]
TP = 0
FP = 0
FN = 0
TN = 0
for i in range(4298):
    if test_actual_label[i] == 1 and test_predict_label[i] == 1:
        TP = TP +1
    elif test_actual_label[i] == 1 and test_predict_label[i] == 0:
        FN = FN +1
    elif test_actual_label[i] == 0 and test_predict_label[i] == 1:
        FP = FP +1
    else:
        TN = TN +1
```

In [37]:
```
Accuracy2 = (TP + TN)/(TP + TN + FP + FN)
print("Accuracy with Laplace smoothing: ", Accuracy)
print("Accuracy without Laplace smoothing: ",Accuracy2)
```

```
Accuracy with Laplace smoothing:  0.8024662633783155
Accuracy without Laplace smoothing:  0.5132619823173569
```