HW2_Q3_NaiveBayes

October 13, 2017

In [1]:
```
# total number of documents = 2144
# total number of words = 1448
import numpy as np
import pandas as pd
from collections import Counter
```

0.0.1 (a) Implement Naive Bayes Classifier

In [2]:
```
num_doc = 2144
vocab_size = 1448
```

In [3]:
```
train_label = np.zeros(2144)
df = pd.DataFrame(0,index = range(1,1449,1), columns = ['spam', 'non-spam'], dtype=int)

# used for storing the features of each document in nested dictionary
doc_feature_dict = {"spam":{}, "non-spam":{}}
doc_feature = {}
```

In [4]:
```
with open('SPARSE.TRAIN', 'r') as f:
    lines = f.readlines()
    row_index = 0
    for line in lines:
        features = line.split()
        train_label[row_index] = features[0]
        doc_feature_dict['spam'][row_index] = []
        doc_feature_dict['non-spam'][row_index] = []
        doc_feature[row_index] = []
```

```
            for feature in features[1:]:
                w_index, w_count = feature.split(":")
                w_index = int(w_index.strip())
                w_count = int(w_count.strip())
                doc_feature[row_index].append(w_index)
                # 1 means spam
                if features[0] == "1":
                    df.at[w_index,"spam"] += w_count
                    doc_feature_dict['spam'][row_index].append(w_index)
                if features[0] == "-1":
                    df.at[w_index,"non-spam"] += w_count
                    doc_feature_dict['non-spam'][row_index].append(w_index)
                    #print features[0]
            row_index += 1
        print row_index
```

```
2144
```

```
In [5]: def estimate_prior(train_label):
            # numpy convert str to number
            spam_count = np.count_nonzero(train_label == 1)
            print spam_count
            non_spam_count = np.count_nonzero(train_label == -1)
            print non_spam_count
            total_count = spam_count + non_spam_count
            spam_prior = float(spam_count)/float(total_count)
            non_spam_prior = float(non_spam_count)/float(total_count)
            return spam_prior, non_spam_prior

        spam_prior, non_spam_prior = estimate_prior(train_label)
```

```
1070
1074
```

```
In [6]: #1+ total count of word wj in all documents with label y
        #loop through entire vocab, sum all the total count of word wj in all documents with label y
```

```
# class_conditional_table

df['spam_prob'] = (df['spam'] + 1) / (df['spam'].sum()+1448)
df['non-spam_prob'] = (df['non-spam'] + 1) / (df['non-spam'].sum()+1448)
#df['non-spam_prob'] = df['non-spam']/float(1074)

df
```

Out[6]:

	spam	non-spam	spam_prob	non-spam_prob
1	28	16	0.000271	0.000227
2	87	69	0.000821	0.000935
3	12	11	0.000121	0.000160
4	98	53	0.000924	0.000721
5	13	26	0.000131	0.000361
6	25	57	0.000243	0.000775
7	147	47	0.001381	0.000641
8	11	20	0.000112	0.000280
9	57	69	0.000541	0.000935
10	53	71	0.000504	0.000962
11	80	169	0.000756	0.002270
12	112	61	0.001055	0.000828
13	104	56	0.000980	0.000761
14	685	82	0.006403	0.001108
15	29	24	0.000280	0.000334
16	1	34	0.000019	0.000467
17	60	17	0.000569	0.000240
18	63	50	0.000597	0.000681
19	95	20	0.000896	0.000280
20	72	22	0.000681	0.000307
21	42	3	0.000401	0.000053
22	5	42	0.000056	0.000574
23	84	1	0.000793	0.000027
24	30	20	0.000289	0.000280
25	53	3	0.000504	0.000053
26	50	21	0.000476	0.000294
27	5	7	0.000056	0.000107
28	151	27	0.001419	0.000374
29	66	102	0.000625	0.001376

```
30      18        23   0.000177       0.000321
...    ...       ...        ...            ...
1419    11        22   0.000112       0.000307
1420    27        40   0.000261       0.000548
1421   184        79   0.001727       0.001068
1422    95       329   0.000896       0.004407
1423     8        15   0.000084       0.000214
1424    76         3   0.000719       0.000053
1425    21        25   0.000205       0.000347
1426    45        25   0.000429       0.000347
1427    98        68   0.000924       0.000921
1428    55        80   0.000523       0.001082
1429    31        96   0.000299       0.001295
1430    95       127   0.000896       0.001709
1431   456       418   0.004266       0.005596
1432   242       138   0.002268       0.001856
1433    37        29   0.000355       0.000401
1434     0        28   0.000009       0.000387
1435    45        31   0.000429       0.000427
1436     8        65   0.000084       0.000881
1437   156      1034   0.001465       0.013822
1438     4       125   0.000047       0.001683
1439     7        94   0.000075       0.001269
1440   124         1   0.001167       0.000027
1441    76       101   0.000719       0.001362
1442     1        27   0.000019       0.000374
1443   440       289   0.004116       0.003873
1444    24        30   0.000233       0.000414
1445    18        34   0.000177       0.000467
1446    41         1   0.000392       0.000027
1447   309        42   0.002894       0.000574
1448    14        13   0.000140       0.000187

[1448 rows x 4 columns]
```

In [7]:
```
test_label = np.zeros(800)
test_matrix = np.zeros((800, 1448))
```

```
        with open('SPARSE.TEST', 'r') as f:
            lines = f.readlines()
            row_index = 0
            for line in lines:
                features = line.split()
                #print features
                test_label[row_index] = features[0]
                for feature in features[1:]:
                    w_index, w_count = feature.split(":")
                    w_index = int(w_index.strip())
                    w_count = int(w_count.strip())
                    #print type(w_index), w_index, type(w_count), w_count
                    test_matrix[row_index, w_index-1] = w_count
                row_index += 1
```

In [8]:
```
def predict(test_matrix):

    p_spam = np.log(spam_prior) + np.dot(test_matrix, np.log(df.as_matrix(['spam_prob'])[:,0]))
    p_non_spam = np.log(non_spam_prior) + np.dot(test_matrix, np.log(df.as_matrix(['non-spam_prob'])[:,0]))

    predictions = np.zeros(800)
    for i in range(800):
        if p_spam[i] > p_non_spam[i]:
            predictions[i] = 1
        else:
            predictions[i] = -1
    return predictions
```

In [31]:
```
pred = predict(test_matrix)
def score(predict_labels, test_labels):
    if len(predict_labels) != 800 or len(test_labels) != 800:
        print "error"
    diff = np.sum(predict_labels != test_labels)
    score = float(diff)/float(len(test_labels))
    print '{:.2%}'.format(score)
    return score
print "error rate: ", score(pred, test_label)
```

```
error rate:  1.62%
```

```
0.01625
```

np.dot(test_matrix, np.log(df.as_matrix(['spam_prob'])))).shape
df.as_matrix(['spam_prob'])[:,0]
p_spam = np.log(spam_prior) + np.dot(test_matrix, np.log(df.as_matrix(['spam_prob'])[:,0])) p_spam

0.0.2 Find the 5 tokens that are most indicative of the SPAM class

```
In [22]: df['Spammy_Score'] = np.log(df['spam_prob']/df['non-spam_prob'])

In [23]: df.nlargest(5, 'Spammy')
```

Out[23]:

	spam	non-spam	spam_prob	non-spam_prob	Spammy	Spammy_Score
616	1609	0	0.015028	0.000013	7.025792	7.025792
1210	1523	0	0.014225	0.000013	6.970896	6.970896
1357	323	0	0.003024	0.000013	5.422546	5.422546
394	258	0	0.002418	0.000013	5.198630	5.198630
1369	256	0	0.002399	0.000013	5.190879	5.190879

```
In [24]: top5_spammy_words = df.nlargest(5, 'Spammy').index.values

In [26]: tokens = {}
         with open('TOKENS_LIST', 'r') as f:
             lines = f.readlines()
             for idx in top5_spammy_words:
                 line_idx = idx-1
                 word = lines[line_idx].split()[1]
                 print idx, word, df.at[idx,"Spammy_Score"]

 616 httpaddr 7.02579188212
1210 spam 6.97089616038
1357 unsubscrib 5.42254593993
394 ebai 5.19863048584
1369 valet 5.19087850903
```