Assignment 8

CS 532: Introduction to Web Science

Spring 2018

Chandrasekhar Reddy Muthyala

1

Question

(Spam classification using Naive Bayes classifier)
1. Create two datasets; the first called Testing, the second called Training.

The Training dataset should:
a. consist of 10 text documents for email messages you consider spam
 (from your spam folder)
b. consist of 10 text documents for email messages you consider not spam
 (from your inbox)

The Testing dataset should:
a. consist of 10 text documents for email messages you consider spam
 (from your spam folder)
b. consist of 10 text documents for email messages you consider not spam
(from your inbox)

Upload your datasets on github

Answer

To solve this problem I have created two data sets as per the requirement. One is **Training dataset** and other is **Testing dataset**. In both Training and Testing dataset, I added 10 spam and 10 non spam emails from spam folder and inbox respectively.

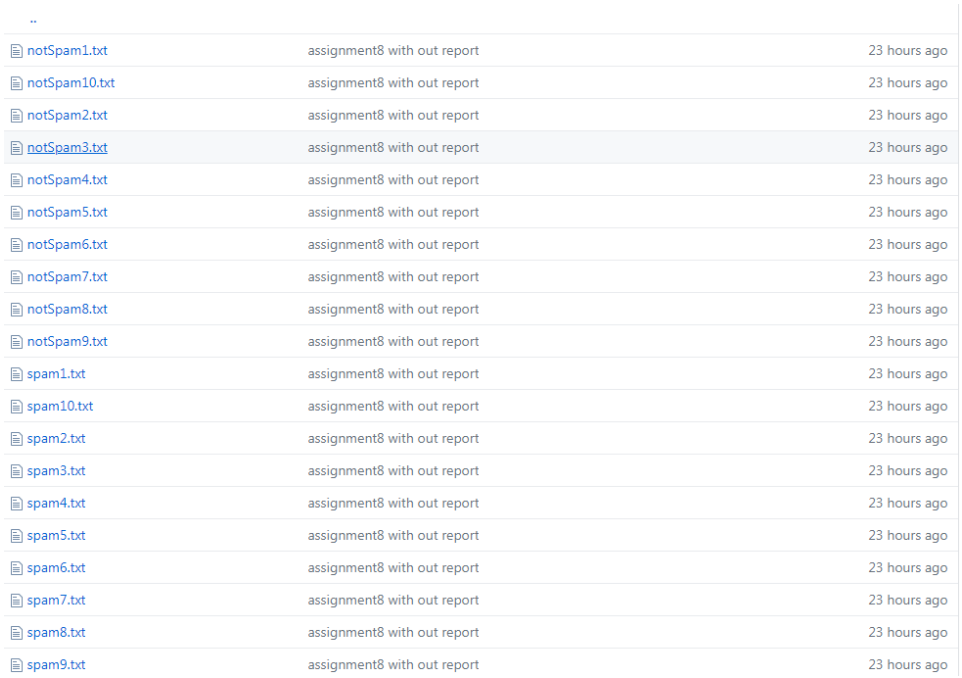

Figure 1: Training dataset image

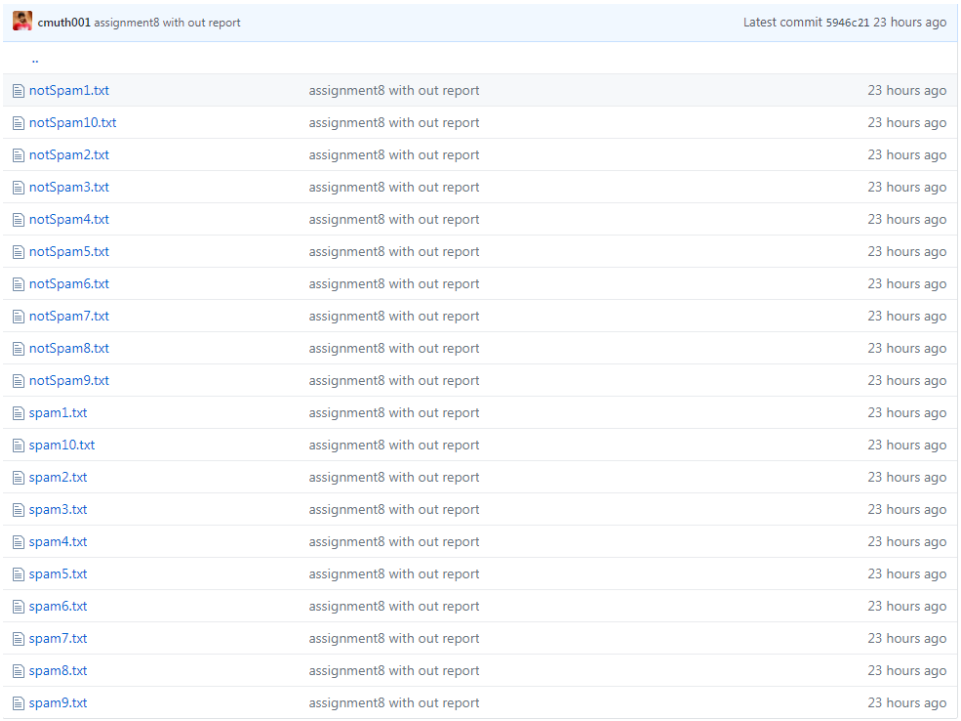

Figure 2: Testing dataset image

2

Question

```
2. Using the PCI book modified docclass.py code and test.py
(see Slack assignment-8 channel)
Use your Training dataset to train the Naive Bayes classifier
 ( e.g., docclass.spamTrain() )
Use your Testing dataset to test (test.py)
 the Naive Bayes classifier and report the classification results.
```

Answer

To solve this Spam classification using Naive Bayes classifier I have gone through class note and few other articles to understand how it will classify email whether spam or not. This classifier aggregates information using conditional probability. I have modified **docclass.py** code and **test.py** to get the result.

In **docclass.py** I added one more function checkSpamOrNot to train my dataset. In this function I am looping all the 10 spam and 10 non spam emails from train dataset. Read each text file and pass that text to train function for training.

To find spam or not spam email I have used **test.py** and modified as per the requirement. In order to calculate confusion matrix I used one variable output which will store all the True positives, True negatives, False positives and False negatives. Looping all the 10 spam and 10 non spam emails from test dataset to classify each email.

True positives: not spam email classifies as not spam email
True negatives: spam email classifies as spam email
False positives: spam email classifies as not spam email
False negatives: not spam email classifies as spam email
The test dataset results as given below:
True positives: 7
True negatives: 10
False positives: 0
False negatives: 3

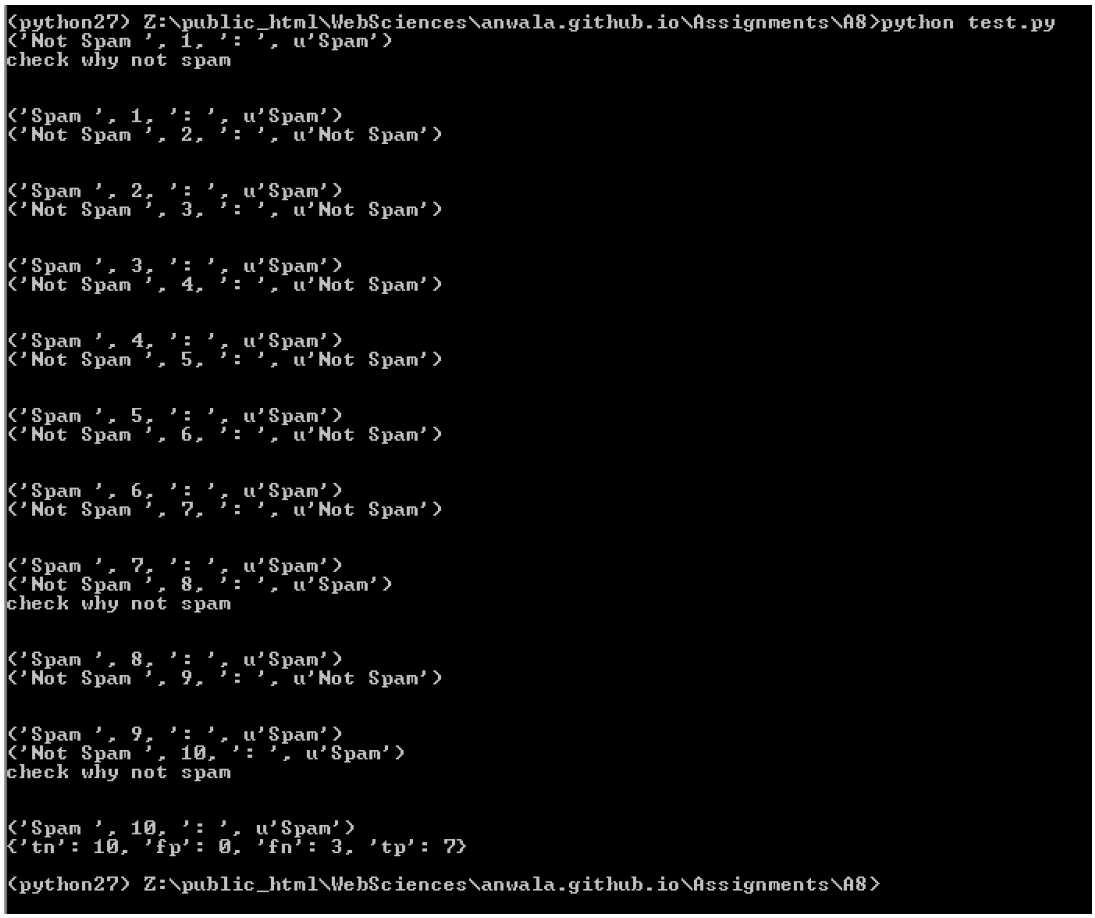

Figure 3: Out put of my test dataset

```
1  #from pysqlite2 import dbapi2 as sqlite
2  import sqlite3 as sqlite
3  import re
4  import math
5
6  def getwords(doc):
7    splitter=re.compile('\\W*')
8    #print(doc)
9    # Split the words by non-alpha characters
10   words=[s.lower() for s in splitter.split(doc)
11           if len(s)>2 and len(s)<20]
12
13   # Return the unique set of words only
14   toreturn = dict([(w,1) for w in words])
15   return toreturn
16
17 class classifier:
18   def __init__(self,getfeatures,filename=None):
19     # Counts of feature/category combinations
20     self.fc={}
21     # Counts of documents in each category
22     self.cc={}
23     self.getfeatures=getfeatures
24
25   def setdb(self,dbfile):
26     self.con=sqlite.connect(dbfile)
27     self.con.execute('create table if not exists fc(feature,
           category,count)')
28     self.con.execute('create table if not exists cc(category,
           count)')
29
30
31   def incf(self,f,cat):
32     count=self.fcount(f,cat)
33     if count==0:
34       self.con.execute("insert into fc values ('%s','%s',1)"
35                        % (f,cat))
36     else:
37       self.con.execute(
38         "update fc set count=%d where feature='%s' and category
               ='%s'"
39         % (count+1,f,cat))
40
41   def fcount(self,f,cat):
42     res=self.con.execute(
43       'select count from fc where feature="%s" and category="%s
           "'
44      %(f,cat)).fetchone()
```

```
45      if res==None: return 0
46      else: return float(res[0])
47
48    def incc(self,cat):
49      count=self.catcount(cat)
50      if count==0:
51        self.con.execute("insert into cc values ('%s',1)" % (cat))
52      else:
53        self.con.execute("update cc set count=%d where category='%
            s'"
54                         % (count+1,cat))
55
56    def catcount(self,cat):
57      res=self.con.execute('select count from cc where category="%
            s"'
58                           %(cat)).fetchone()
59      if res==None: return 0
60      else: return float(res[0])
61
62    def categories(self):
63      cur=self.con.execute('select category from cc');
64      return [d[0] for d in cur]
65
66    def totalcount(self):
67      res=self.con.execute('select sum(count) from cc').fetchone()
            ;
68      if res==None: return 0
69      return res[0]
70
71
72    def train(self,item,cat):
73      features=self.getfeatures(item)
74      # Increment the count for every feature with this category
75      for f in features:
76        self.incf(f,cat)
77
78      # Increment the count for this category
79      self.incc(cat)
80      self.con.commit()
81
82    def fprob(self,f,cat):
83      if self.catcount(cat)==0: return 0
84
85      # The total number of times this feature appeared in this
86      # category divided by the total number of items in this
            category
87      return self.fcount(f,cat)/self.catcount(cat)
88
89    def weightedprob(self,f,cat,prf,weight=1.0,ap=0.5):
```

```
90      # Calculate current probability
91      basicprob=prf(f,cat)
92
93      # Count the number of times this feature has appeared in
94      # all categories
95      totals=sum([self.fcount(f,c) for c in self.categories()])
96
97      # Calculate the weighted average
98      bp=((weight*ap)+(totals*basicprob))/(weight+totals)
99      return bp
100
101
102
103
104 class naivebayes(classifier):
105
106   def __init__(self,getfeatures):
107     classifier.__init__(self,getfeatures)
108     self.thresholds={}
109
110   def docprob(self,item,cat):
111     features=self.getfeatures(item)
112
113     # Multiply the probabilities of all the features together
114     p=1
115     for f in features: p*=self.weightedprob(f,cat,self.fprob)
116     return p
117
118   def prob(self,item,cat):
119     catprob=self.catcount(cat)/self.totalcount()
120     docprob=self.docprob(item,cat)
121     return docprob*catprob
122
123   def setthreshold(self,cat,t):
124     self.thresholds[cat]=t
125
126   def getthreshold(self,cat):
127     if cat not in self.thresholds: return 1.0
128     return self.thresholds[cat]
129
130   def classify(self,item,default=None):
131     probs={}
132     # Find the category with the highest probability
133     max=0.0
134     for cat in self.categories():
135       probs[cat]=self.prob(item,cat)
136       if probs[cat]>max:
137         max=probs[cat]
138         best=cat
```

```
139
140     # Make sure the probability exceeds threshold*next best
141     for cat in probs:
142       if cat==best: continue
143       if probs[cat]*self.getthreshold(best)>probs[best]: return
              default
144     return best
145
146 class fisherclassifier(classifier):
147   def cprob(self,f,cat):
148     # The frequency of this feature in this category
149     clf=self.fprob(f,cat)
150     if clf==0: return 0
151
152     # The frequency of this feature in all the categories
153     freqsum=sum([self.fprob(f,c) for c in self.categories()])
154
155     # The probability is the frequency in this category divided
          by
156     # the overall frequency
157     p=clf/(freqsum)
158
159     return p
160   def fisherprob(self,item,cat):
161     # Multiply all the probabilities together
162     p=1
163     features=self.getfeatures(item)
164     for f in features:
165       p*=(self.weightedprob(f,cat,self.cprob))
166
167     # Take the natural log and multiply by -2
168     fscore=-2*math.log(p)
169
170     # Use the inverse chi2 function to get a probability
171     return self.invchi2(fscore,len(features)*2)
172   def invchi2(self,chi, df):
173     m = chi / 2.0
174     sum = term = math.exp(-m)
175     for i in range(1, df//2):
176         term *= m / i
177         sum += term
178     return min(sum, 1.0)
179   def __init__(self,getfeatures):
180     classifier.__init__(self,getfeatures)
181     self.minimums={}
182
183   def setminimum(self,cat,min):
184     self.minimums[cat]=min
185
```

```
186   def getminimum(self,cat):
187     if cat not in self.minimums: return 0
188     return self.minimums[cat]
189   def classify(self,item,default=None):
190     # Loop through looking for the best result
191     best=default
192     max=0.0
193     for c in self.categories():
194       p=self.fisherprob(item,c)
195       # Make sure it exceeds its minimum
196       if p>self.getminimum(c) and p>max:
197         best=c
198         max=p
199     return best
200
201
202 def sampletrain(cl):
203   cl.train('Nobody owns the water.','good')
204   cl.train('the quick rabbit jumps fences','good')
205   cl.train('buy pharmaceuticals now','bad')
206   cl.train('make quick money at the online casino','bad')
207   cl.train('the quick brown fox jumps','good')
208
209 def spamTrain(cl):
210   cl.train('the the', 'not spam')
211   cl.train('cheap cheap cheap banking the', 'spam')
212   cl.train('the', 'not spam')
213   cl.train('cheap cheap banking banking banking the the', 'spam
          ')
214   cl.train('cheap cheap cheap cheap cheap buy buy the', 'spam')
215   cl.train('banking the', 'not spam')
216   cl.train('buy banking the', 'not spam')
217   cl.train('the', 'not spam')
218   cl.train('the', 'not spam')
219   cl.train('cheap buy dinner the the', 'not spam')
220
221 def checkSpamOrNot(cl):
222   for i in range(1,11):
223     with open('./trainData/notSpam'+str(i)+'.txt') as f:
224           txt = f.read()
225           cl.train(txt, 'Not Spam')
226           # print("Not_Spam ",x,": ",cl.classify(email), file=
                open('output.txt', 'a+'))
227
228     with open('./trainData/spam'+str(i)+'.txt') as f:
229           txt = f.read()
230           cl.train(txt, 'Spam')
231           # print("Spam ",x,": ",cl.classify(email), file=open('
                output.txt', 'a+'))
```

Listing 1: Python script for training and classify dataset

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 27 00:45:20 2018

@author: cmuthyal
"""

import docclass
from subprocess import check_output

cl = docclass.naivebayes(docclass.getwords)
#remove previous db file
# check_output(["rm", "chandu.db"])

cl.setdb("SpamOrNot.db")
# docclass.spamTrain(cl)
# docclass.sampletrain(cl)
docclass.checkSpamOrNot(cl)
#classify text: "the banking dinner" as spam or not spam
output = {'tp':0, 'tn':0, 'fp':0, 'fn':0}
for i in range(1,11):
    with open('./testData/notSpam'+str(i)+'.txt') as f:
            txt = f.read()
            notSpamStatus = cl.classify(txt)
            if notSpamStatus == 'Not Spam':
                    output['tp']+=1
                    print("Not Spam ",i,": ",notSpamStatus)
            else:
                    output['fn']+=1
                    print("Not Spam ",i,": ",notSpamStatus)
                    print('check why not spam')
            print('\n')

            # print("Not_Spam ",x,": ",cl.classify(email), file=
                open('output.txt', 'a+'))

    with open('./testData/spam'+str(i)+'.txt') as f:
            txt = f.read()
            spamStatus = cl.classify(txt)
            if spamStatus == 'Spam':
                    output['tn']+=1
            else:
                    output['fp']+=1
            print("Spam ",i,": ",spamStatus)
print(output)
```

Listing 2: Python script to classify test dataset

3

Question

3. Draw a confusion matrix for your classification results (see: https://en.wikipedia.org/wiki/Confusion_matrix)

Answer

The below confusion matrix is drawn using 2nd problem output.

	Not Spam	**spam**
Not Spam	True Positives - 7	False Negative - 3
spam	False Positives - 0	True Negatives -10

Table 1: confusion matrix

4

Question

`4. Report the precision and accuracy scores of your classification results (see: https://en.wikipedia.org/wiki/Precision_and_recall)`

Answer

By using True positives, True negatives, False positives and False negatives
I calculated precision and accuracy scores.

$$Precision = \frac{tp}{tp+fp} = \frac{7}{7+0} = 1$$

$$Recall = \frac{tp}{tp+fn} = \frac{7}{7+3} = 0.7$$

References

[1] https://en.wikipedia.org/wiki/Precision_and_recall.

[2] https://en.wikipedia.org/wiki/Confusion_matrix.

[3] http://scikit-learn.org/stable/modules/naive_bayes.html.

[4] https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

[5] https://github.com/arthur-e/Programming-Collective-Intelligence.