Assignment 8

CS 532: Introduction to Web Science

Spring 2018

Hrishi Gadkari

1

Question

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

For the above question, I went through my gmail spam folder which had enough spam emails to download. I downloaded 20 emails in pdf file format. As the emails had to be in text format, I converted them using [1]. I made two folders test and train in which I pasted 10 spam emails individually as spam(count).txt. I then downloaded 20 emails from the inbox of my same gmail account which I used for downloading spam emails. They were also in pdf format which I later converted into [1] and pasted into train and test folders containing 10 emails each as nonspam(count).txt. After this I uploaded them on my Github account[2].

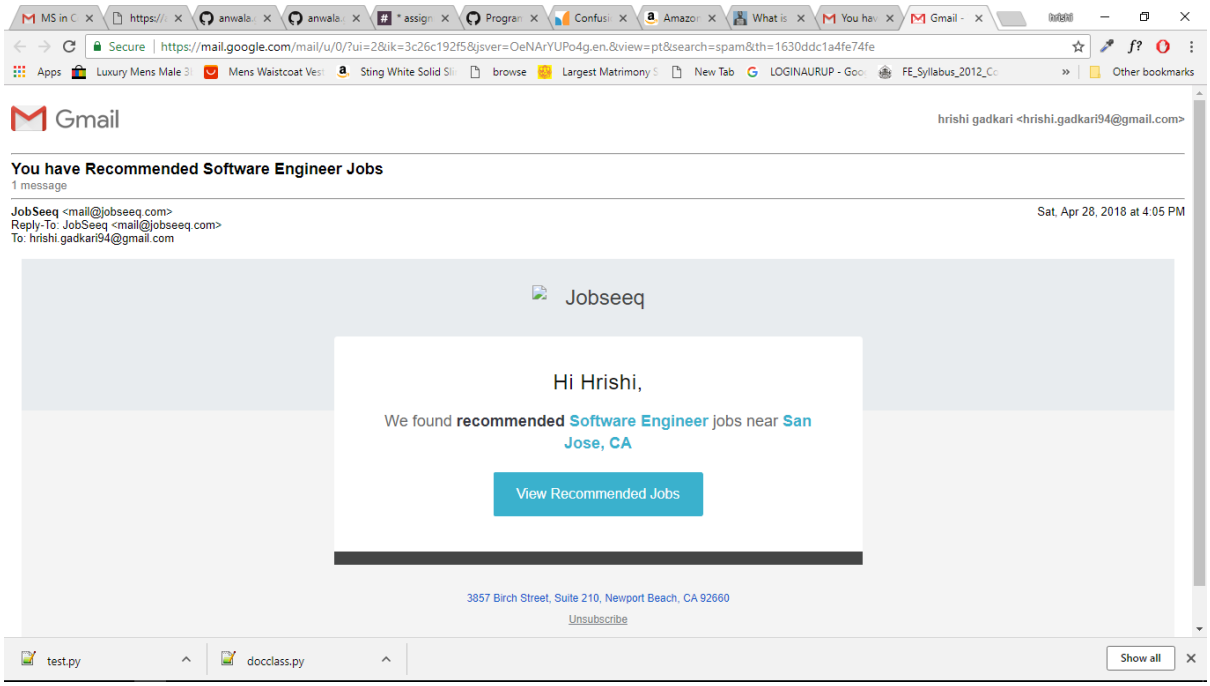

Figure 1: Emails downloaded in pdf format

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

In order to train the Training dataset, I modified the **docclass.py** code as discussed in Slack assignment8 channel. The code is as follows:

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

```
          "'
      %(f,cat)).fetchone()
    if res==None: return 0
    else: return float(res[0])

  def incc(self,cat):
    count=self.catcount(cat)
    if count==0:
      self.con.execute("insert into cc values ('%s',1)" % (cat))
    else:
      self.con.execute("update cc set count=%d where category='%
          s'"
                       % (count+1,cat))

  def catcount(self,cat):
    res=self.con.execute('select count from cc where category="%
        s"'
                         %(cat)).fetchone()
    if res==None: return 0
    else: return float(res[0])

  def categories(self):
    cur=self.con.execute('select category from cc');
    return [d[0] for d in cur]

  def totalcount(self):
    res=self.con.execute('select sum(count) from cc').fetchone()
        ;
    if res==None: return 0
    return res[0]


  def train(self,item,cat):
    features=self.getfeatures(item)
    # Increment the count for every feature with this category
    for f in features:
      self.incf(f,cat)

    # Increment the count for this category
    self.incc(cat)
    self.con.commit()

  def fprob(self,f,cat):
    if self.catcount(cat)==0: return 0

    # The total number of times this feature appeared in this
    # category divided by the total number of items in this
        category
    return self.fcount(f,cat)/self.catcount(cat)
```

```
86
87    def weightedprob(self,f,cat,prf,weight=1.0,ap=0.5):
88      # Calculate current probability
89      basicprob=prf(f,cat)
90
91      # Count the number of times this feature has appeared in
92      # all categories
93      totals=sum([self.fcount(f,c) for c in self.categories()])
94
95      # Calculate the weighted average
96      bp=((weight*ap)+(totals*basicprob))/(weight+totals)
97      return bp
98
99
100
101
102 class naivebayes(classifier):
103
104   def __init__(self,getfeatures):
105     classifier.__init__(self,getfeatures)
106     self.thresholds={}
107
108   def docprob(self,item,cat):
109     features=self.getfeatures(item)
110
111     # Multiply the probabilities of all the features together
112     p=1
113     for f in features: p*=self.weightedprob(f,cat,self.fprob)
114     return p
115
116   def prob(self,item,cat):
117     catprob=self.catcount(cat)/self.totalcount()
118     docprob=self.docprob(item,cat)
119     return docprob*catprob
120
121   def setthreshold(self,cat,t):
122     self.thresholds[cat]=t
123
124   def getthreshold(self,cat):
125     if cat not in self.thresholds: return 1.0
126     return self.thresholds[cat]
127
128   def classify(self,item,default=None):
129     probs={}
130     # Find the category with the highest probability
131     max=0.0
132     for cat in self.categories():
133       probs[cat]=self.prob(item,cat)
134       if probs[cat]>max:
```

```
135         max=probs[cat]
136         best=cat
137
138     # Make sure the probability exceeds threshold*next best
139     for cat in probs:
140       if cat==best: continue
141       if probs[cat]*self.getthreshold(best)>probs[best]: return
              default
142     return best
143
144 class fisherclassifier(classifier):
145   def cprob(self,f,cat):
146     # The frequency of this feature in this category
147     clf=self.fprob(f,cat)
148     if clf==0: return 0
149
150     # The frequency of this feature in all the categories
151     freqsum=sum([self.fprob(f,c) for c in self.categories()])
152
153     # The probability is the frequency in this category divided
            by
154     # the overall frequency
155     p=clf/(freqsum)
156
157     return p
158   def fisherprob(self,item,cat):
159     # Multiply all the probabilities together
160     p=1
161     features=self.getfeatures(item)
162     for f in features:
163       p*=(self.weightedprob(f,cat,self.cprob))
164
165     # Take the natural log and multiply by -2
166     fscore=-2*math.log(p)
167
168     # Use the inverse chi2 function to get a probability
169     return self.invchi2(fscore,len(features)*2)
170   def invchi2(self,chi, df):
171     m = chi / 2.0
172     sum = term = math.exp(-m)
173     for i in range(1, df//2):
174         term *= m / i
175         sum += term
176     return min(sum, 1.0)
177   def __init__(self,getfeatures):
178     classifier.__init__(self,getfeatures)
179     self.minimums={}
180
181   def setminimum(self,cat,min):
```

```
182     self.minimums[cat]=min
183
184   def getminimum(self,cat):
185     if cat not in self.minimums: return 0
186     return self.minimums[cat]
187   def classify(self,item,default=None):
188     # Loop through looking for the best result
189     best=default
190     max=0.0
191     for c in self.categories():
192       p=self.fisherprob(item,c)
193       # Make sure it exceeds its minimum
194       if p>self.getminimum(c) and p>max:
195         best=c
196         max=p
197     return best
198
199 def eTrain(cl):
200
201     # train on spam
202         for i in range(1,11):
203                 filename = 'train/spam' + str(i) +'.txt'
204
205                 with open(filename, 'r', encoding='utf-8') as
                        trainFile:
206                     cl.train(trainFile.read(), 'spam')
207
208
209     # train on non spam
210         for i in range(1,11):
211                 filename = 'train/nonspam' + str(i) +'.txt'
212
213                 with open(filename, 'r', encoding='utf-8') as
                        trainFile1:
214                     cl.train(trainFile1.read(), 'not spam')
```

Listing 1: Python program to train the dataset with Naive Bayes Classifier

The files read as spam for training, I passed **spam** as a classifier in the second argument of the **c1.train** function whereas file read as nonspam , I passed **not spam**. The code is written in a function called **eTrain()** I then modified the **test.py** to test the Nave Bayes classifier on the Testing dataset as follows.

```
1 import docclass
2 from subprocess import check_output
3
4
5 cl = docclass.naivebayes(docclass.getwords)
```

```
6  #remove previous db file
7
8
9  cl.setdb('hrishi.db')
10 docclass.eTrain(cl)
11
12 for i in range(11,21):
13                 filename = 'test/spam' + str(i) +'.txt'
14
15                 with open(filename, 'r', encoding='utf-8') as
                       testFile:
16                     print(filename, cl.classify(testFile.
                           read()))
17
18
19 for i in range(11,21):
20                 filename = 'test/nonspam' + str(i) +'.txt'
21
22                 with open(filename, 'r', encoding='utf-8') as
                       testFile1:
23                     print(filename, cl.classify(testFile1.
                           read()))
24 #classify text: "the banking dinner" as spam or not spam
25 #print( cl.classify('the banking dinner') )
```

Listing 2: Python program to test the dataset with Naive Bayes Classifier

From the result we could see it identified 10/10 spam files as spam and 7/10 as non-spam. I think it did a good job.

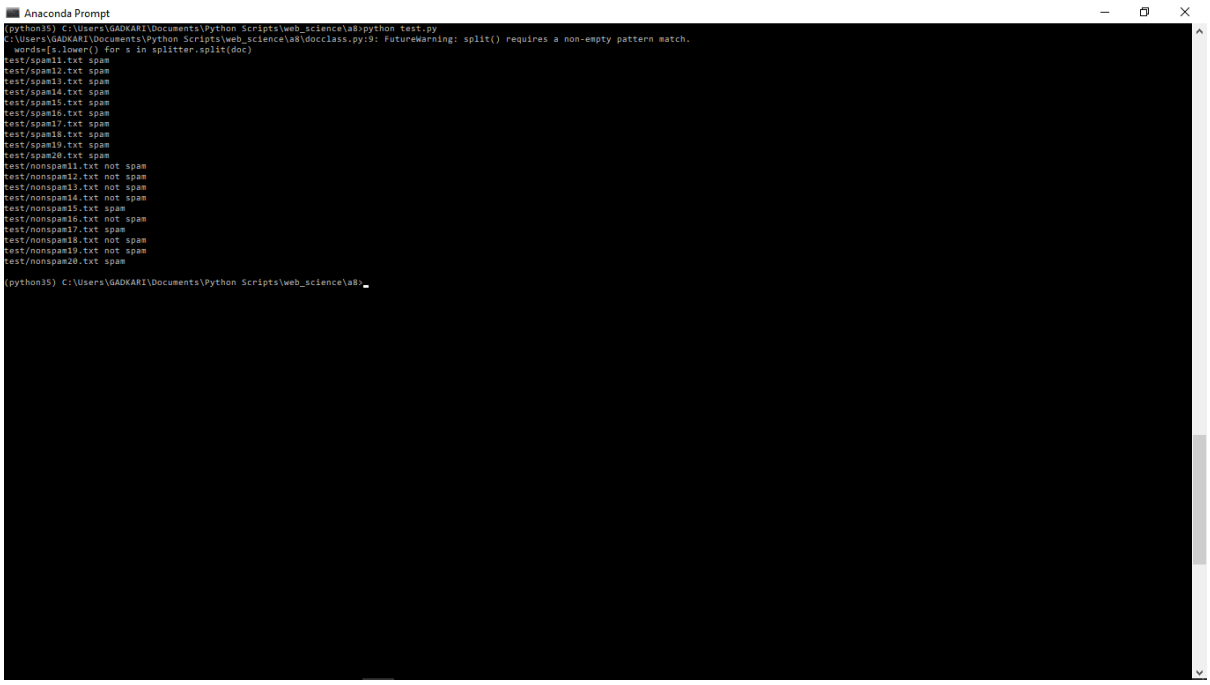

Figure 2: Classification Results

3

Question

```
==================================
===Each question below is for 3 points extra credit===
==================================

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

Confusion Matrix	**Predicted Spam**	**Predicted Non Spam**
True Spam	10	0
True Non Spam	3	7

Table 1: Confusion Matrix

Answer

In-order to understand a confusion matrix I went through [3] as mentioned in the question. As per the classification results and by the understanding of confusion matrix, the matrix would look as above:

4

Question

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

Answer

.For calculating precision I went through the [4] as mentioned in the question. Based on the classification results the precision is calculated as follows:

$$\text{precision} = \frac{\text{true positives}}{\text{true positives + false positives}} = \frac{10}{10 + 0} = 1$$

For calculating the accuracy score I went through [5] and calculated as follows:

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{total predicyions}} = \frac{10 + 7}{10 + 0 + 3 + 7} = 0.85$$

References

[1] "Convert PDF to Text Online" PDF to Text, n.d. Web. April 30, 2018. http://pdftotext.com/.

[2] "GitHub."Hrishi29/anwala.github.io.,n.d. Web. April 30, 2018. https://github.com/Hrishi29/anwala.github.io/tree/master/Assignments

[3] "Confusion matrix." Wikipedia, April 27, 2018. Web. April 30, 2018. https://en.wikipedia.org/wiki/Confusion_matrix

[4] "Precision and recall." Wikipedia, April 09, 2018. Web. April 30, 2018. https://en.wikipedia.org/wiki/Precision_and_recall

[5] What is a Confusion Matrix in Machine Learning "Machine Learning Mastery ." December 05, 2017. N.p., Web. April 30, 2018. https://machinelearningmastery.com/confusion-matrix-machine-learning/