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].

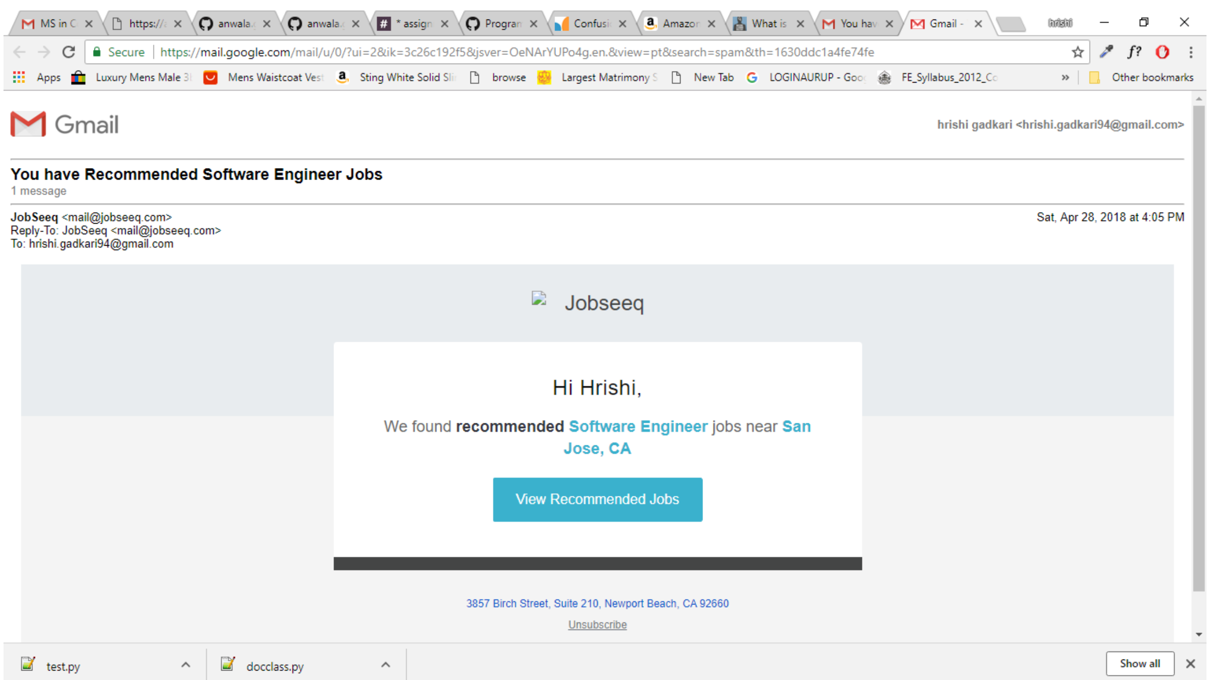

Figure 1: Emails downloaded in pdf format

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:

```
import sqlite3 as sqlite
import re
import math

def getwords(doc):
  splitter=re.compile('\\W*')

  # Split the words by non-alpha characters
  words=[s.lower() for s in splitter.split(doc)
          if len(s)>2 and len(s)<20]

  # Return the unique set of words only
  return dict([(w,1) for w in words])

class classifier:
  def __init__(self,getfeatures,filename=None):
    # Counts of feature/category combinations
    self.fc={}
    # Counts of documents in each category
    self.cc={}
    self.getfeatures=getfeatures

  def setdb(self,dbfile):
    self.con=sqlite.connect(dbfile)
    self.con.execute('create table if not exists fc(feature,category,count)')
    self.con.execute('create table if not exists cc(category,count)')


  def incf(self,f,cat):
    count=self.fcount(f,cat)
    if count==0:
      self.con.execute("insert into fc values ('%s','%s',1)"
                      % (f,cat))
    else:
      self.con.execute(
        "update fc set count=%d where feature='%s' and category='%s'"
        % (count+1,f,cat))

  def fcount(self,f,cat):
    res=self.con.execute(
      '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)
```

```
  def weightedprob(self,f,cat,prf,weight=1.0,ap=0.5):
    # Calculate current probability
    basicprob=prf(f,cat)

    # Count the number of times this feature has appeared in
    # all categories
    totals=sum([self.fcount(f,c) for c in self.categories()])

    # Calculate the weighted average
    bp=((weight*ap)+(totals*basicprob))/(weight+totals)
    return bp




class naivebayes(classifier):

  def __init__(self,getfeatures):
    classifier.__init__(self,getfeatures)
    self.thresholds={}

  def docprob(self,item,cat):
    features=self.getfeatures(item)

    # Multiply the probabilities of all the features together
    p=1
    for f in features: p*=self.weightedprob(f,cat,self.fprob)
    return p

  def prob(self,item,cat):
    catprob=self.catcount(cat)/self.totalcount()
    docprob=self.docprob(item,cat)
    return docprob*catprob

  def setthreshold(self,cat,t):
    self.thresholds[cat]=t

  def getthreshold(self,cat):
    if cat not in self.thresholds: return 1.0
    return self.thresholds[cat]

  def classify(self,item,default=None):
    probs={}
    # Find the category with the highest probability
    max=0.0
    for cat in self.categories():
      probs[cat]=self.prob(item,cat)
      if probs[cat]>max:
```

```
        max=probs[cat]
        best=cat

    # Make sure the probability exceeds threshold*next best
    for cat in probs:
      if cat==best: continue
      if probs[cat]*self.getthreshold(best)>probs[best]: return default
    return best

class fisherclassifier(classifier):
  def cprob(self,f,cat):
    # The frequency of this feature in this category
    clf=self.fprob(f,cat)
    if clf==0: return 0

    # The frequency of this feature in all the categories
    freqsum=sum([self.fprob(f,c) for c in self.categories()])

    # The probability is the frequency in this category divided by
    # the overall frequency
    p=clf/(freqsum)

    return p
  def fisherprob(self,item,cat):
    # Multiply all the probabilities together
    p=1
    features=self.getfeatures(item)
    for f in features:
      p*=(self.weightedprob(f,cat,self.cprob))

    # Take the natural log and multiply by -2
    fscore=-2*math.log(p)

    # Use the inverse chi2 function to get a probability
    return self.invchi2(fscore,len(features)*2)
  def invchi2(self,chi, df):
    m = chi / 2.0
    sum = term = math.exp(-m)
    for i in range(1, df//2):
        term *= m / i
        sum += term
    return min(sum, 1.0)
  def __init__(self,getfeatures):
    classifier.__init__(self,getfeatures)
    self.minimums={}

  def setminimum(self,cat,min):
```

```
    self.minimums[cat]=min

  def getminimum(self,cat):
    if cat not in self.minimums: return 0
    return self.minimums[cat]
  def classify(self,item,default=None):
    # Loop through looking for the best result
    best=default
    max=0.0
    for c in self.categories():
      p=self.fisherprob(item,c)
      # Make sure it exceeds its minimum
      if p>self.getminimum(c) and p>max:
        best=c
        max=p
    return best

def eTrain(cl):

    # train on spam
        for i in range(1,11):
                filename = 'train/spam' + str(i) +'.txt'

                with open(filename, 'r', encoding='utf-8') as trainFile:
                        cl.train(trainFile.read(), 'spam')


    # train on non spam
        for i in range(1,11):
                filename = 'train/nonspam' + str(i) +'.txt'

                with open(filename, 'r', encoding='utf-8') as trainFile1:
                        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.

```
import docclass
from subprocess import check_output


cl = docclass.naivebayes(docclass.getwords)
```

```
#remove previous db file


cl.setdb('hrishi.db')
docclass.eTrain(cl)

for i in range(11,21):
                filename = 'test/spam' + str(i) +'.txt'

                with open(filename, 'r', encoding='utf-8') as testFile:
                        print(filename, cl.classify(testFile.read()))


for i in range(11,21):
                filename = 'test/nonspam' + str(i) +'.txt'

                with open(filename, 'r', encoding='utf-8') as testFile1:
                        print(filename, cl.classify(testFile1.read()))
#classify text: "the banking dinner" as spam or not spam
#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.

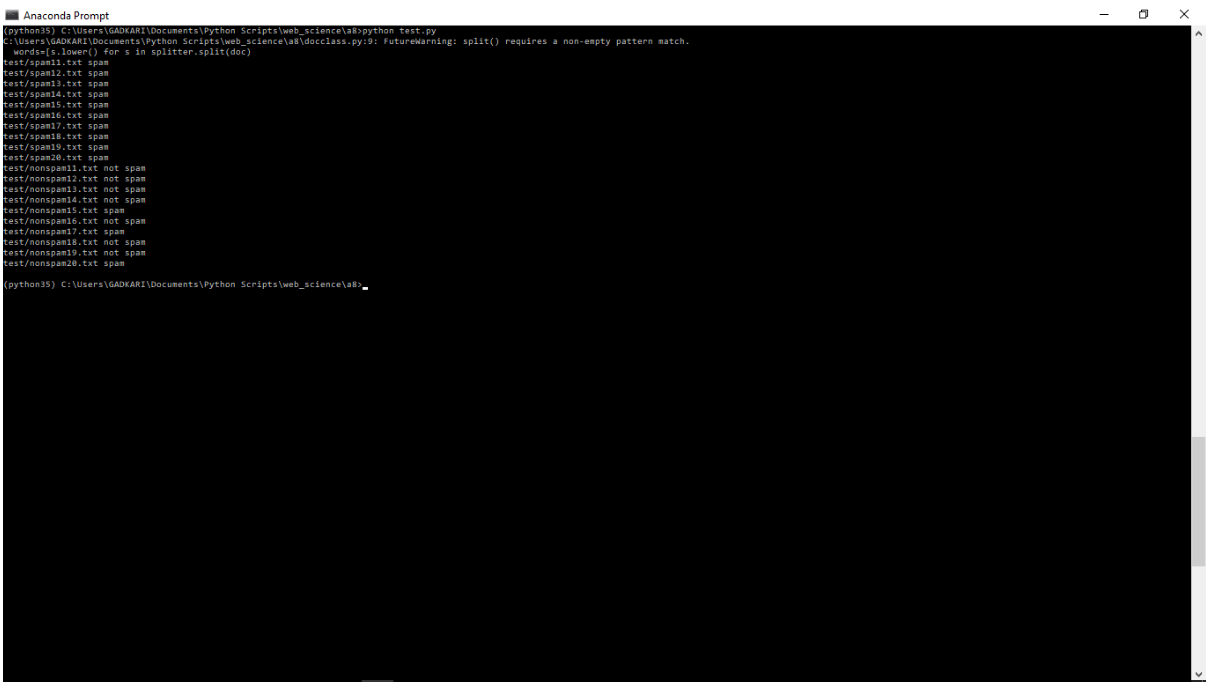

Figure 2: Classification Results

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 follows:

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:

```
precision =         true positives                        =       10        =  1
                  ---------------------------------------    -----------
                      true positives + false positives       10 + 0
```

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

```
accuracy =    correct predictions      =       10 + 7          =  0.85
              -------------------------      -------------------
                      total predicyions         10 + 0 + 3 + 7
```

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/