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.

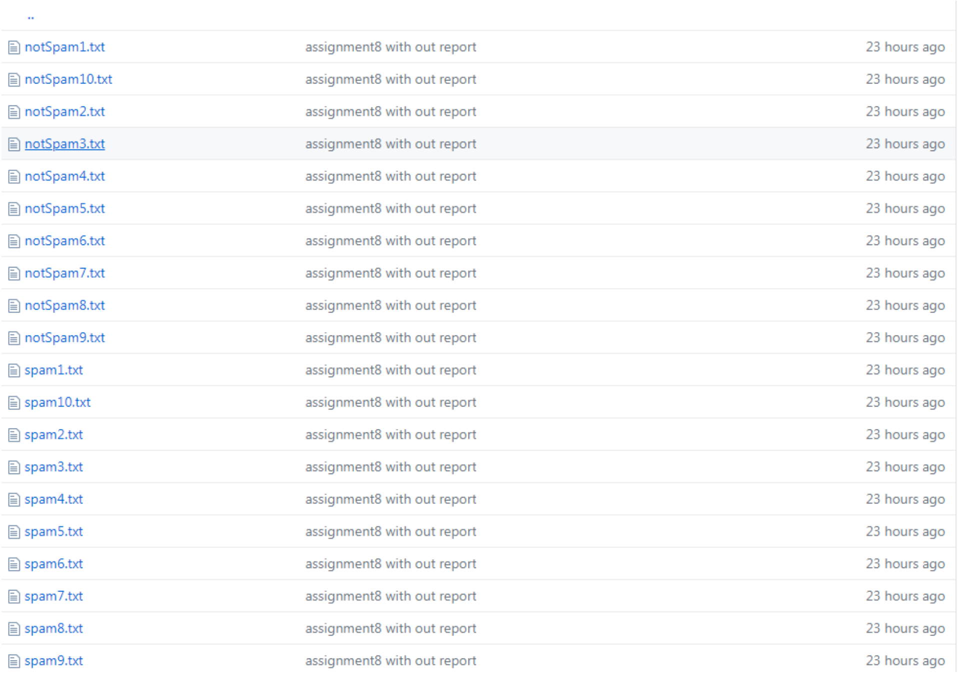
The Training dataset should:
a. consist of 10 text documents for email messages you consider spam
(from your spam folder)
b. consist of 10 text documents for email messages you consider not spam
(from your inbox)

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

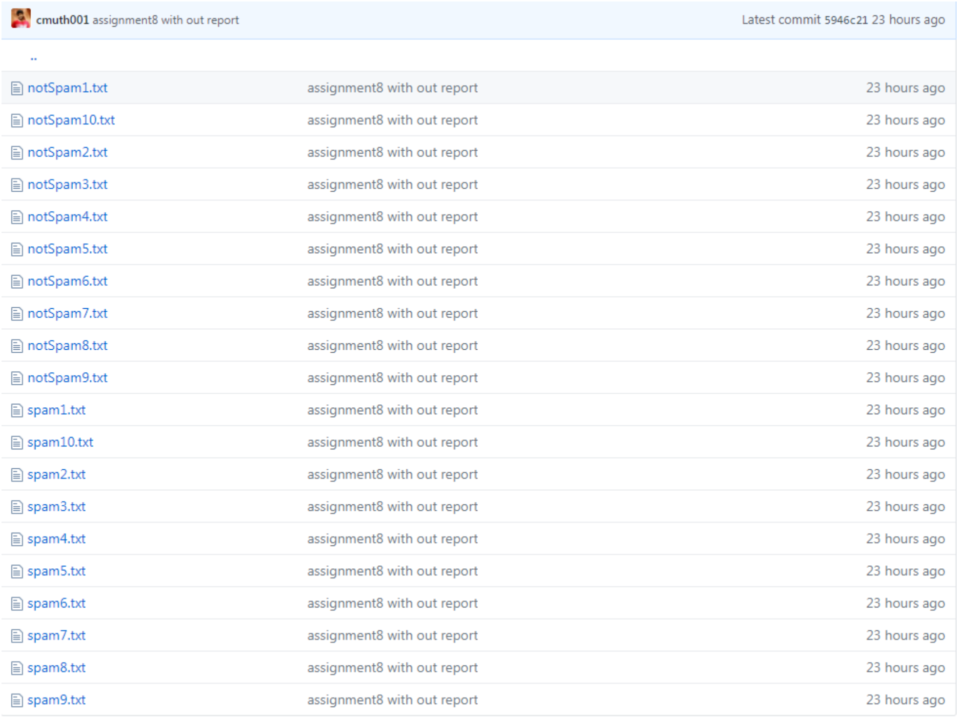
Answer

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.



..		
notSpam1.txt	assignment8 with out report	23 hours ago
notSpam10.txt	assignment8 with out report	23 hours ago
notSpam2.txt	assignment8 with out report	23 hours ago
notSpam3.txt	assignment8 with out report	23 hours ago
notSpam4.txt	assignment8 with out report	23 hours ago
notSpam5.txt	assignment8 with out report	23 hours ago
notSpam6.txt	assignment8 with out report	23 hours ago
notSpam7.txt	assignment8 with out report	23 hours ago
notSpam8.txt	assignment8 with out report	23 hours ago
notSpam9.txt	assignment8 with out report	23 hours ago
spam1.txt	assignment8 with out report	23 hours ago
spam10.txt	assignment8 with out report	23 hours ago
spam2.txt	assignment8 with out report	23 hours ago
spam3.txt	assignment8 with out report	23 hours ago
spam4.txt	assignment8 with out report	23 hours ago
spam5.txt	assignment8 with out report	23 hours ago
spam6.txt	assignment8 with out report	23 hours ago
spam7.txt	assignment8 with out report	23 hours ago
spam8.txt	assignment8 with out report	23 hours ago
spam9.txt	assignment8 with out report	23 hours ago

Figure 1: Training dataset image



cmuth001 assignment8 with out report Latest commit 5946c21 23 hours ago

..

notSpam1.txt	assignment8 with out report	23 hours ago
notSpam10.txt	assignment8 with out report	23 hours ago
notSpam2.txt	assignment8 with out report	23 hours ago
notSpam3.txt	assignment8 with out report	23 hours ago
notSpam4.txt	assignment8 with out report	23 hours ago
notSpam5.txt	assignment8 with out report	23 hours ago
notSpam6.txt	assignment8 with out report	23 hours ago
notSpam7.txt	assignment8 with out report	23 hours ago
notSpam8.txt	assignment8 with out report	23 hours ago
notSpam9.txt	assignment8 with out report	23 hours ago
spam1.txt	assignment8 with out report	23 hours ago
spam10.txt	assignment8 with out report	23 hours ago
spam2.txt	assignment8 with out report	23 hours ago
spam3.txt	assignment8 with out report	23 hours ago
spam4.txt	assignment8 with out report	23 hours ago
spam5.txt	assignment8 with out report	23 hours ago
spam6.txt	assignment8 with out report	23 hours ago
spam7.txt	assignment8 with out report	23 hours ago
spam8.txt	assignment8 with out report	23 hours ago
spam9.txt	assignment8 with out report	23 hours ago

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

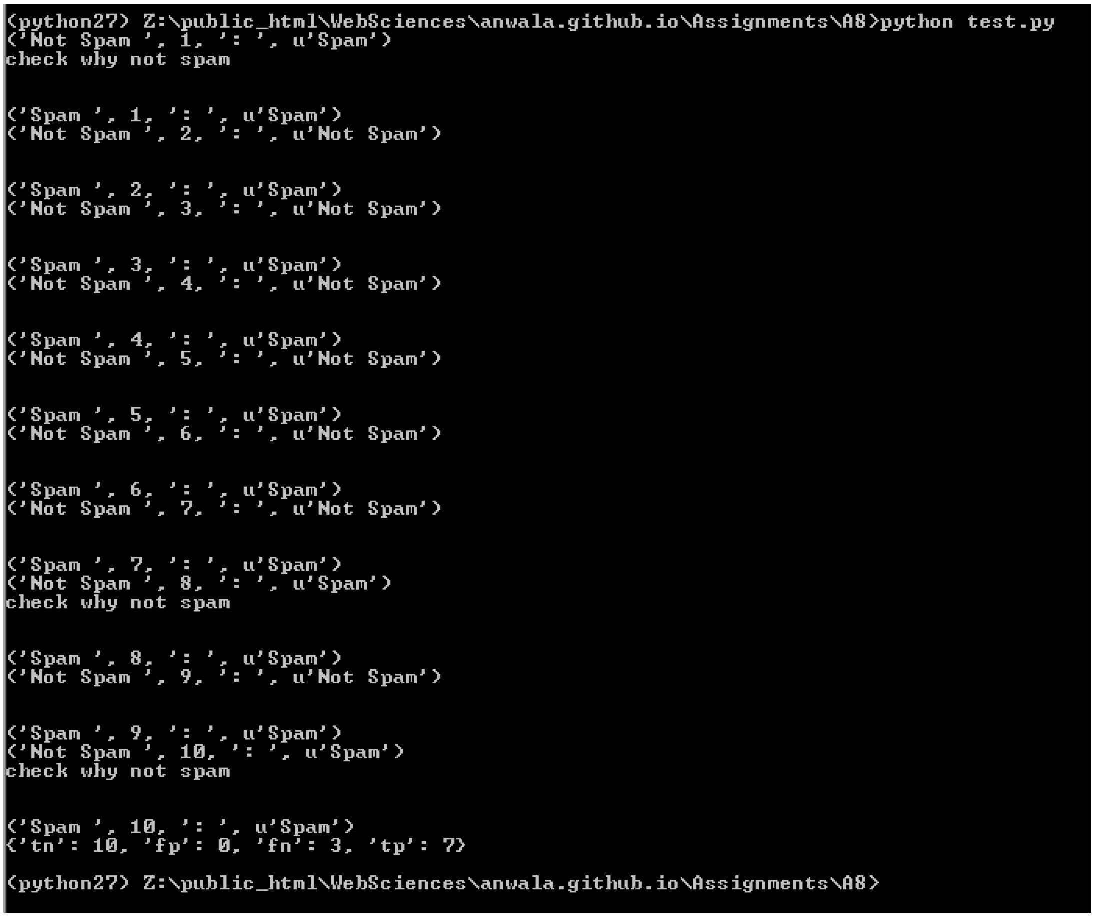
The test dataset results as given below:

True positives:: 7

True negatives: 10

False positives: 0

False negatives: 3



```
(python27) Z:\public_html\WebSciences\anwala.github.io\Assignments\A8>python test.py
('Not Spam ', 1, ': ', u'Spam')
check why not spam


('Spam ', 1, ': ', u'Spam')
('Not Spam ', 2, ': ', u'Not Spam')


('Spam ', 2, ': ', u'Spam')
('Not Spam ', 3, ': ', u'Not Spam')


('Spam ', 3, ': ', u'Spam')
('Not Spam ', 4, ': ', u'Not Spam')


('Spam ', 4, ': ', u'Spam')
('Not Spam ', 5, ': ', u'Not Spam')


('Spam ', 5, ': ', u'Spam')
('Not Spam ', 6, ': ', u'Not Spam')


('Spam ', 6, ': ', u'Spam')
('Not Spam ', 7, ': ', u'Not Spam')


('Spam ', 7, ': ', u'Spam')
('Not Spam ', 8, ': ', u'Spam')
check why not spam


('Spam ', 8, ': ', u'Spam')
('Not Spam ', 9, ': ', u'Not Spam')


('Spam ', 9, ': ', u'Spam')
('Not Spam ', 10, ': ', u'Spam')
check why not spam


('Spam ', 10, ': ', u'Spam')
{'tn': 10, 'fp': 0, 'fn': 3, 'tp': 7}

(python27) Z:\public_html\WebSciences\anwala.github.io\Assignments\A8>
```

Figure 3: Out put of my test dataset

```
#from pysqlite2 import dbapi2 as sqlite
import sqlite3 as sqlite
import re
import math

def getwords(doc):
  splitter=re.compile('\\W*')
  #print(doc)
  # 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
  toreturn = dict([(w,1) for w in words])
  return toreturn

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 sampletrain(cl):
  cl.train('Nobody owns the water.','good')
  cl.train('the quick rabbit jumps fences','good')
  cl.train('buy pharmaceuticals now','bad')
  cl.train('make quick money at the online casino','bad')
  cl.train('the quick brown fox jumps','good')

def spamTrain(cl):
  cl.train('the the', 'not spam')
  cl.train('cheap cheap cheap banking the', 'spam')
  cl.train('the', 'not spam')
  cl.train('cheap cheap banking banking banking the the', 'spam')
  cl.train('cheap cheap cheap cheap cheap buy buy the', 'spam')
  cl.train('banking the', 'not spam')
  cl.train('buy banking the', 'not spam')
  cl.train('the', 'not spam')
  cl.train('the', 'not spam')
  cl.train('cheap buy dinner the the', 'not spam')

def checkSpamOrNot(cl):
  for i in range(1,11):
    with open('./trainData/notSpam'+str(i)+'.txt') as f:
          txt = f.read()
          cl.train(txt, 'Not Spam')
          # print("Not_Spam ",x,": ",cl.classify(email), file=open('output.txt', 'a+'))

    with open('./trainData/spam'+str(i)+'.txt') as f:
          txt = f.read()
          cl.train(txt, 'Spam')
          # print("Spam ",x,": ",cl.classify(email), file=open('output.txt', 'a+'))
```

Listing 1: Python script for training and classify dataset

```python
# -*- 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('./trainData/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('./trainData/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.