CS532S19: Assignment #8

Due on Tuesday, April 30, 2019

Nwala, Alexander C

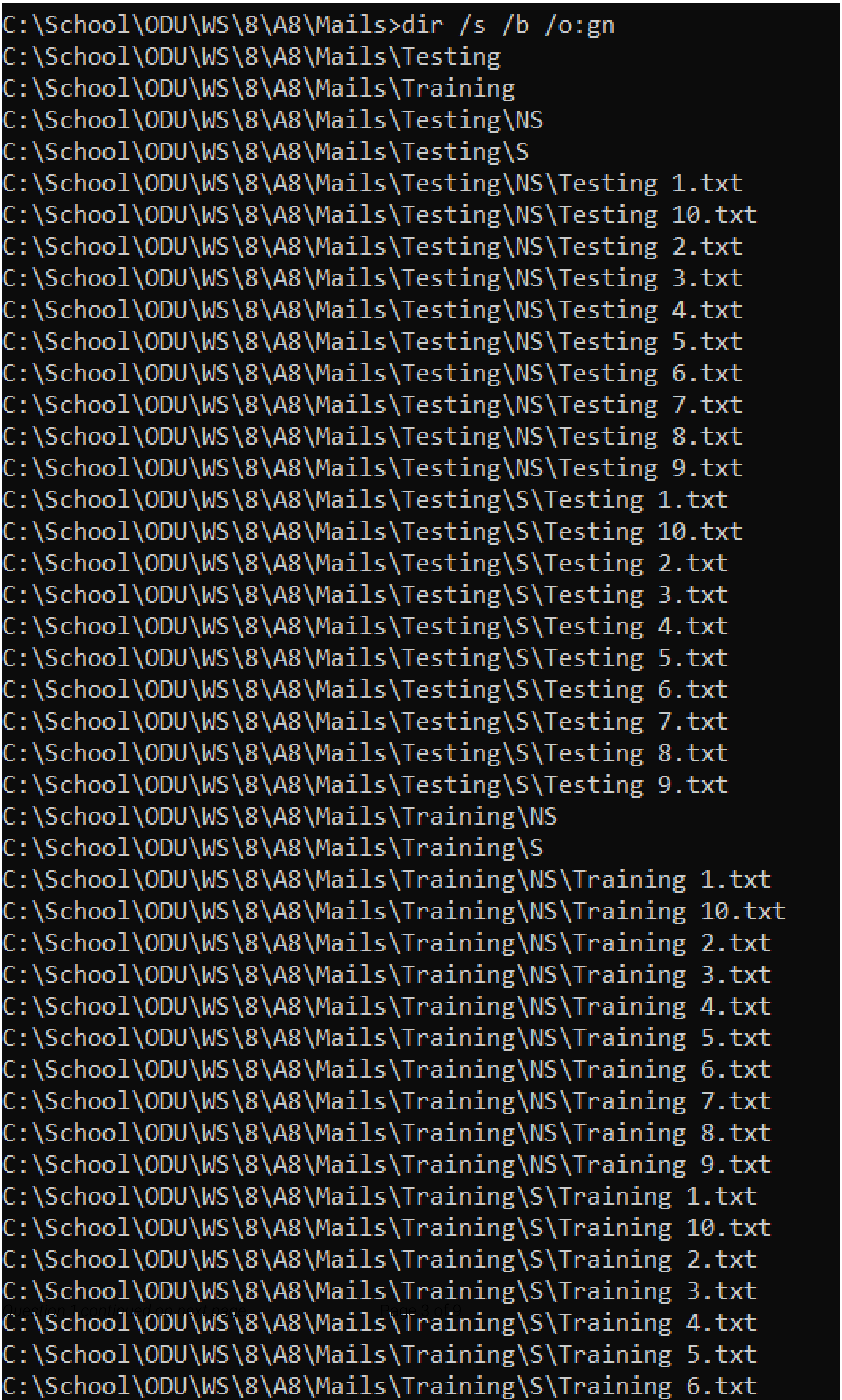
Giridharan Ganeshkumar

Question 1

1. Create two datasets the first called Testing, the second called Training. The Training dataset should - consist of 10 text documents for email messages you consider spam, 10 text documents for email messages you consider not spam.Then the Testing dataset should - consist of 10 text documents for email messages you consider spam, consist of 10 text documents for email messages you consider not spam.

1. This was a manual excercise where we accessed our mail accounts and created 20 mails from the spam folder and 20 mails from the inbox folder.
2. These are created so that we could train and test with naivebayes methods defined in the doc class
3. The below image list the text files or dataset used for this purpose and the same is also uploaded to the git hub.

Question 1 continued on next page…



```
C:\School\ODU\WS\8\A8\Mails>dir /s /b /o:gn
C:\School\ODU\WS\8\A8\Mails\Testing
C:\School\ODU\WS\8\A8\Mails\Training
C:\School\ODU\WS\8\A8\Mails\Testing\NS
C:\School\ODU\WS\8\A8\Mails\Testing\S
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 1.txt
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 10.txt
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 2.txt
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 3.txt
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 4.txt
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 5.txt
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 6.txt
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 7.txt
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 8.txt
C:\School\ODU\WS\8\A8\Mails\Testing\NS\Testing 9.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 1.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 10.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 2.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 3.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 4.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 5.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 6.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 7.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 8.txt
C:\School\ODU\WS\8\A8\Mails\Testing\S\Testing 9.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS
C:\School\ODU\WS\8\A8\Mails\Training\S
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 1.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 10.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 2.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 3.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 4.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 5.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 6.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 7.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 8.txt
C:\School\ODU\WS\8\A8\Mails\Training\NS\Training 9.txt
C:\School\ODU\WS\8\A8\Mails\Training\S\Training 1.txt
C:\School\ODU\WS\8\A8\Mails\Training\S\Training 10.txt
C:\School\ODU\WS\8\A8\Mails\Training\S\Training 2.txt
C:\School\ODU\WS\8\A8\Mails\Training\S\Training 3.txt
C:\School\ODU\WS\8\A8\Mails\Training\S\Training 4.txt
C:\School\ODU\WS\8\A8\Mails\Training\S\Training 5.txt
C:\School\ODU\WS\8\A8\Mails\Training\S\Training 6.txt
```

Question 1 continued on next page

Question 2

2. Using the PCI book modified docclass.py code and test.py Use your Training dataset to train the Naive Bayes classifier. Use your Testing dataset to test the Naive Bayes classifier and report the classification results.

1. The naivebayes algorithm is used where we can train the dataset to be classified in categories.
2. The implementation is based on SQLite to store the words and the various categories and select statements mentioned in the classifier method returns the classifier object.
3. As listed in the below script we can call the classify method to check which category the current data belongs.
4. Below is the completed class implementation as per the PCI text book, the next script is the main method where call the spam train and not spam train methods.

Listing 1: Python Script

```
import sqlite3 as sqlite
import re
import math

class docclass:
    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
    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 spamTrain(cl):
        cl = docclass.naivebayes(docclass.getwords)
        for x in range(11):
```

Question 2 continued on next page...

```
            if(x != 0):
                fileTemp = "C:\Mails\Training\S\Training "+ str(x) +".txt"
                with open(fileTemp, "r") as file:
                    data = file.read()
                    cl.train(data,"Spam")

    def notSpamTrain(cl):
        cl = docclass.naivebayes(docclass.getwords)
        for x in range(11):
            if(x != 0):
                fileTemp = "C:\\Mails\\Training\\NS\\Training "+ str(x) +".txt"
                with open(fileTemp, "r") as file:
                    data = file.read()
                    cl.train(data,"Not Spam")

    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
            self.setdb('autocreated_db_file')

        def setdb(self,dbfile):
            self.con=sqlite.connect(dbfile)
            print('Connected')
            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):
        docclass.classifier.__init__(self,getfeatures)
        self.thresholds={}
```

Question 2 continued on next page...

```

        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
```

Question 2 continued on next page. . .

```
        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
```

1. Below is the main method where we create the database file
2. The first step it to use the classify, naivebayes classifier is identified for the purpose and we get hold of the classifier.
3. Next step is to call the spam train and not spam train methods
4. Once trained we can call the classify the method that gives results if the mail passed is an spam or not spam.

Listing 2: Python Script

```
import os
from subprocess import check_output

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

Question 2 continued on next page...

```python

dbfile = 'anwala.db'

if( os.path.exists(dbfile) ):
    check_output(['rm', dbfile])
    print('removed dbfile:', dbfile)

cl.setdb(dbfile)
docclass.spamTrain(cl)
docclass.notSpamTrain(cl)

for x in range(11):
    if(x != 0):
        fileTemp = "C:\\Mails\\Testing\\S\\Testing "+ str(x) +".txt"
        with open(fileTemp, "r") as file:
            data = file.read()
            print(cl.classify(data))

for x in range(11):
    if(x != 0):
        fileTemp = "C:\\Mails\\Testing\\NS\Testing "+ str(x) +".txt"
        with open(fileTemp, "r") as file:
            data = file.read()
            print(cl.classify(data))
```