# HW9 - Email Classification

Adeniran Adeniyi

DUE Wednesday, April 28, 2021 by 11:59pm

# Q1

*You may choose a topic to classify your emails on (but choose only 1 topic). This can be spam, shopping emails, school emails, etc.*
*The Training dataset should consist of:*

- 20 text documents for email messages you consider on your chosen topic
- 20 text documents for email messages you consider not on your chosen topic

*The Testing data-set should consist of:*

- 5 text documents for email messages you consider on your chosen topic
- 5 text documents for email messages you consider not on your chosen topic

*Make sure that these are plain-text documents and that they do not include HTML tags. The documents in the Testing set should be different than the documents in the Training set.*
*Upload your datasets to your GitHub repo. Please do not include emails that contain sensitive information*

*What topic did you decide to classify on?*

## Answer

*The topic of my classification I did was shopping emails types.*

## Discussion

*I followed these instruction below:*

- For this I created a folder called Q1.
- In Q1 I had two folders called train and test.

- These folder contained my train data sets and test data set respectively.
- Under the two sub folders train and test, I had two folders - mytopic, notmytopic- that contained text files associated with my topic or without my topic in the respective folders as described.
- I used copied content of emails used to the text files manual. I used a second email when shopping emails where running out in my first email address

# Q2

*Use the example code in the class Colab notebook to train and test the Naive Bayes classifier.*

- *Use your Training dataset to train the Naive Bayes classifier.*
- *Use your Testing dataset to test the Naive Bayes classifier.*

*Create a table to report the classification results for each email message in the Testing dataset. The table should include what the classifier reported (on-topic or off-topic) and the actual classification.*

*For those emails that the classifier got wrong, look at the text and discuss what factors might have caused the classifier to be incorrect.*

## Answer

**Table 1:** Result of after classification

| Filename | actualTopic | classifiedAs |
|---|---|---|
| Q1/test/mytopic\1.txt | shopping | on-topic |
| Q1/test/mytopic\2.txt | shopping | on-topic |
| Q1/test/mytopic\3.txt | shopping | on-topic |
| Q1/test/mytopic\4.txt | shopping | on-topic |
| Q1/test/mytopic\5.txt | shopping | on-topic |
| Q1/test/notmytopic\1.txt | non-shopping | off-topic |
| Q1/test/notmytopic\2.txt | non-shopping | off-topic |
| Q1/test/notmytopic\3.txt | non-shopping | off-topic |
| Q1/test/notmytopic\4.txt | non-shopping | off-topic |
| Q1/test/notmytopic\5.txt | non-shopping | off-topic |

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 25 03:46:18 2021

@author: aadeniran
"""
import re
import math

"""
Splits the document into a list of words by non-alphs characters
"""
def getwords(doc):
  splitter=re.compile('\W+')  # different than book
  #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
  uniq_words = dict([(w,1) for w in words])

  return uniq_words

class basic_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

  # Increase the count of a feature/category pair
  def incf(self,f,cat):
    self.fc.setdefault(f, {})
    self.fc[f].setdefault(cat, 0)
    self.fc[f][cat]+=1

  # Increase the count of a category
  def incc(self,cat):
    self.cc.setdefault(cat, 0)
    self.cc[cat]+=1

  # The number of times a feature has appeared in a category
  def fcount(self,f,cat):
```

```
47      if f in self.fc and cat in self.fc[f]:
48        return float(self.fc[f][cat])
49      return 0.0
50
51    # The number of items in a category
52    def catcount(self,cat):
53      if cat in self.cc:
54          return float(self.cc[cat])
55      return 0
56
57    # The total number of items
58    def totalcount(self):
59      return sum(self.cc.values())
60
61    # The list of all categories
62    def categories(self):
63      return self.cc.keys()
64
65    def train(self,item,cat):
66      features=self.getfeatures(item)
67      # Increment the count for every feature with this category
68      for f in features:
69        self.incf(f,cat)
70
71      # Increment the count for this category
72      self.incc(cat)
73
74    def fprob(self,f,cat):
75      if self.catcount(cat)==0: return 0
76
77      # The total number of times this feature appeared in this
78      # category divided by the total number of items in this category
79      return self.fcount(f,cat)/self.catcount(cat)
80
81      """
82      default weight=1.0,ap=0.5
83      weight >= 0.6 and ap >= 0.9 for the classifier to be wrong
84      weight >=0.7  and ap >= 0.8
85      weight >=0.8  and ap >= 0.7
86      weight >=0.9  and ap >= 0.6
87      weight =15  and ap = 0.5
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 #class naivebayes(classifier):   # change for basic_classifier
102 class naivebayes(basic_classifier):
103   def __init__(self,getfeatures):
104     #classifier.__init__(self,getfeatures)  # change for
      basic_classifier
105     basic_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
```

**Listing 1:** This contained all the functions meant for classifying classify.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Apr 25 04:38:14 2021
4
5  @author: aadeniran
6  """
7
8  from classify import *
9  import os
10 import glob
11 import pandas as pd
12
13 """
14 Create a list of that classified topic as on topic and a list that
       classifies topic as off-topic
15 """
16 goodList =[]
17 badList =[]
18
19 def sampleTrain(cl):
20  for a in goodList:
21      #for shopping
22      cl.train(a, "on-topic")
23
24  for b in badList:
25      #for non-shopping classification
26      cl.train(b, "off-topic")
27
28 """
29 Train files pertaining to my topic
30 """
31
32 #Enter all shopping training data
33 path = 'Q1/train/mytopic'
34 for filename in glob.glob(os.path.join(path, '*.txt')):
35     with open(os.path.join(os.getcwd(), filename), 'r') as f:
36         text = f.read()
37         encoded = text.encode("ascii", "ignore")
38         decode_string = encoded.decode()
39         goodList.append(decode_string)
40 """
41 Train files not pertaining to my topic
```

```
"""
#Enter all non shopping training data
path = 'Q1/train/notmytopic/'
for filename in glob.glob(os.path.join(path, '*.txt')):
     with open(os.path.join(os.getcwd(), filename), 'r') as f:
        text = f.read()
        encoded = text.encode("ascii", "ignore")
        decode_string = encoded.decode()
        badList.append(decode_string)


#call the classifier and store trained set using sampleTrain() function

cl = naivebayes(getwords)
sampleTrain(cl)

print("End of training \n")

#create a pandas data frame for table creation.
table = pd.DataFrame(columns=["Filename","actualTopic","classifiedAs"])

#Test shopping test data
print("Shopping test data: \n")
path = 'Q1/test/mytopic'
#Read text files from the directory above
for filename in glob.glob(os.path.join(path, '*.txt')):
    with open(os.path.join(os.getcwd(), filename), 'r') as f:
        text = f.read()
        #encode the text file as ascii then decode it back
        classValue = cl.classify(text.encode("ascii", "ignore").decode
    (), default='unknown')
        #create a row for the pandas dataframe
        new_row = {'Filename':filename, 'actualTopic':"shopping", '
    classifiedAs':classValue}
        #insert row into the data frame
        table = table.append(new_row,ignore_index=True)
        print("For ", filename, " classified as ", classValue)

print("\n\n")
#Test non shopping test data
print("Non-shopping test data:\n")
path = 'Q1/test/notmytopic/'
for filename in glob.glob(os.path.join(path, '*.txt')):
    with open(os.path.join(os.getcwd(), filename), 'r') as f:
        text = f.read()
        #encode the text file as ascii then decode it back
        classValue = cl.classify(text.encode("ascii", "ignore").decode
```

```
    (), default='unknown')
87          #create a row for the pandas dataframe
88          new_row = {'Filename':filename, 'actualTopic':"non-shopping", '
    classifiedAs':classValue}
89          #insert row into the data frame
90          table = table.append(new_row,ignore_index=True)
91          print("For ", filename, " classified as ", classValue)
92
93 table.to_csv("Q5/final.csv",index=False)
```

**Listing 2:** The driver for training testing the data files trainClassify.py

## Discussion

*From the table 1, I did not have any emails classified wrongly.*

*I followed these instruction below:*

- In Listing 1 classify.py has all the necessary function to run the classify.
  - From line 13 to line 23 the getwords(doc) function splits a document into a list of words by dividing on any character that isn't a letter. The words are all in lower letter. The retrieves only unique words from the list using a dictionary

    From line 25 to line 99, the basic_classifier class helps defined necessary function that are used in naivabayes() function in line 102
  - basic_classifier uses three instances variables fc(stores count for different features in the different classifications), cc(a dictionary type that has the number of times every classification has been used), getfeatures (extracts the features from the items being classified )
  - Plus other helper functions - incf(),incc(),fcount(),catcount(),totalcount(),categories()- to increment and access the counts to store training data in a file or db
  - train() function to help process training data by extracting words and updating counts
  - fprob() displays the probability that a word appears in a category by using the Multiple Bernouli method
  - weightedprob() This function returns the weighted probability of Pr(w—c) using assumed probablities
  - In lines 102 to line 142 is used naivebayes() function, I had to change the argument from classifier to basic_classifier and classifier.__init__(self,getfeatures) to basic_classifier.__init__(self,getfeatures) since I am not using the sql one.
  - the docprob function extracts the features of the word and multiply all of the weighted probabilities together which give the overall document probability. Pr(d—c)

- prob function uses Bayes's rule to calculate the probability (c—d)

- In Listing 5, trainClassify.py I used it to perform the following tasks as a driver
  - In lines 19 to 26 sampleTrain() function will be used to train a text file for on-topic and off-topic

    Lines 32 to 50 first trains textfiles that where on topic and appending it to the good list then trains the off-topic and appends it to the badList variable. In line 55 and 56 the neccessary call to run the functio is called.
  - Once classification is done, I created a dataFrame to keep track of the final result of the predicted values, the actual values and pathnames as columns.
  - Lines 63 to 75 handles running the test data files that on-topic and saves it in a pandas dataframe table.

```
63 #Test shopping test data
64 print("Shopping test data: \n")
65 path = 'Q1/test/mytopic'
66 #Read text files from the directory above
67 for filename in glob.glob(os.path.join(path, '*.txt')):
68     with open(os.path.join(os.getcwd(), filename), 'r') as f:
69         text = f.read()
70         #encode the text file as ascii then decode it back
71         classValue = cl.classify(text.encode("ascii", "ignore")
   .decode(), default='unknown')
72         #create a row for the pandas dataframe
73         new_row = {'Filename':filename, 'actualTopic':"shopping
   ", 'classifiedAs':classValue}
74         #insert row into the data frame
75         table = table.append(new_row,ignore_index=True)
```

**Listing 3:** Snapshot of trainClassify.py test data for on-topic

  - Lines 79 to 91 handles running the test data files that off-topic and saves it in the same pandas dataframe variable table.

```
79 #Test non shopping test data
80 print("Non-shopping test data:\n")
81 path = 'Q1/test/notmytopic/'
82 for filename in glob.glob(os.path.join(path, '*.txt')):
83     with open(os.path.join(os.getcwd(), filename), 'r') as f:
84         text = f.read()
85         #encode the text file as ascii then decode it back
86         classValue = cl.classify(text.encode("ascii", "ignore")
   .decode(), default='unknown')
87         #create a row for the pandas dataframe
88         new_row = {'Filename':filename, 'actualTopic':"non-
   shopping", 'classifiedAs':classValue}
89         #insert row into the data frame
```

```
90          table = table.append(new_row,ignore_index=True)
91          print("For ", filename, " classified as ", classValue)
```

**Listing 4:** Snapshot of trainClassify.py test data for off-topic

- I finally save the pandata dataframe as final.csv file in Q2 folder, in line 92

**Listing 5:** Snapshot of trainClassify.py saving final result as a csv file

- Table 1, shows the final result.

# Q3

*Draw a confusion matrix for your classification results (see Module 13, slides 40, 42, 43).*

- *This should be a table in Markdown or LaTeX, not a screenshot of output or image generated by another program. There's an example of a LaTeX confusion matrix in the Overleaf report template.*

*Based on the results in the confusion matrix, how well did the classifier perform?*

*Would you prefer an email classifier to have more false positives or more false negatives? Why?*

## Answer

**Table 2:** Confusion Matrix For result in Q2

| | | Actual | |
|---|---|---|---|
| | | Shopping | Non-shopping |
| Predicted | Shopping | 5 (TP) | 0 (FP) |
| | Non-shopping | 0 (FN) | 5 (TN) |

## Discussion

*Based on the result from the confusion matrix the classifier program did a perfect job, since there was not values for the False negatives and False Positives.*
*Personally I would want the email classifier to have less false positive and more false negative*

*because I want more email well classified in the right category - I do not mind if I loose documents that are meant for shopping categories as long as all documments in shopping category is shopping email I am good.*

*I followed these instruction below:*

-

# Q4. (Extra credit, 1 point)

*Report the precision and accuracy scores of your classification results (see Module 13, slide 43). Include the formulas you used to compute these values.*

## Answer

*Formular for precision*
$P = tp/(tp + fp) = 5/(5+5) = 5/10 = 0.5$

*Formular for Recall*

$R = tp/(tp+fn) = 5/(5+0) = 1$

# Q5. (Extra credit, 2 point)

*Tune your classifier by updating weights to obtain better classification results. You may want to change the default weights (weight, ap) given to weightedprob() or the threshold used for the Bayesian classifier or change how the words are extracted from the document (for this you will need to re-train the model). Report the changes you made, re-run your Testing dataset, and show that the performance improved (either by using the confusion matrix or by computing precision and accuracy).*
*If your classifier got all of the items correct in Q2, change the weights to make the classifier perform worse and discuss the results.*

## Answer

```
87      weight =15  and ap = 0.5
```

**Listing 6:** Weight and ap tuning to make the classify wrongly. snapshot classify.py

## Discussion

*I followed these instruction below:*

- When I used the weight value and ap value from Listing 6, I noticed that the classifier wrongly classified
  Q1/test/notmytopic\1.txt
  classified Q1/test/notmytopic\2.txt
  classified Q1/test/notmytopic\3.txt
  classified as on-topic instead of off-topic.

  Table 3 shows the resulting output

**Table 3:** Result of after Changing weight value from 1 to 15

| Filename | actualTopic | classifiedAs |
|---|---|---|
| Q1/test/mytopic\1.txt | shopping | on-topic |
| Q1/test/mytopic\2.txt | shopping | on-topic |
| Q1/test/mytopic\3.txt | shopping | on-topic |
| Q1/test/mytopic\4.txt | shopping | on-topic |
| Q1/test/mytopic\5.txt | shopping | on-topic |
| Q1/test/notmytopic\1.txt | non-shopping | on-topic |
| Q1/test/notmytopic\2.txt | non-shopping | on-topic |
| Q1/test/notmytopic\3.txt | non-shopping | on-topic |
| Q1/test/notmytopic\4.txt | non-shopping | off-topic |
| Q1/test/notmytopic\5.txt | non-shopping | off-topic |

- confusion metrics is below

**Table 4:** Confusion Matrix result for table 3

| | | Actual | |
|---|---|---|---|
| | | Shopping | Non-shopping |
| Predicted | Shopping | 5 (TP) | 3 (FP) |
| | Non-shopping | 0 (FN) | 2 (TN) |

# References


- https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter6/docclass.py
- https://docs.google.com/presentation/d/1OpfBDl2YEE7AONVeKUyHA-J7a1mRjncD7cen8F6BG1A/edit#slide=id.g7f83ebe645_0_0
- https://github.com/cs432-websci-master/public/blob/main/spr21/432_PCI_Ch06.ipynb