Web Science Assignment 8 Report

Dr. Nelson

James Pindell

4/5/2018

Content

General Info

(Spam classification using Naive Bayes classifier)

Support your answer: include all relevant discussion, assumptions, examples, etc.(10 points)

Problem 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

Solution

The solution for this problem is outlined by the following steps:

1. **Create Training Dataset:** In order to create the Training Dataset, 10 different emails from my inbox and 10 different emails from my spam folder were copied and pasted into different text files, called NotSpam1.txt, NotSpam2.txt, …, NotSpam10.txt and Spam1.txt, Spam2.txt, …, Spam10.txt, which are located in a folder called Training Dataset. You can find the folder and all of the text files from within the package.

2. **Create Testing Dataset**: In order to create the Testing Dataset, the exact same procedure was followed as described in Step 1. However, the only difference is that the emails that were copied and pasted into text files in the Testing Dataset are completely different from the emails that were copied and pasted into the Training Dataset. In addition, each email in the Training Dataset, called Email1.txt, Email2.txt, …, Email20.txt, is located in a folder called Testing Dataset. You can find the folder and all of the text files from within the package.

Problem 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.

Solution

The solution for this problem is outlined by the following steps:

1. **Modify Code**: In order to both train and test the Naive Bayes classifier, the code provided was modified and run. You can find both code files, called Classifier.py and ClassifierTest.py, from within the package, but snippets of both code files are shown below:

```
import sqlite3 as sqlite
import re
import math

'''
Original Code: https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter6/docclass.py
'''

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)')
```

```
import Classifier
from subprocess import check_output

'''
Original Code: https://cs532s18.slack.com/files/U8K4TSGJ1/F9Z33U1B6/test.py
'''

c = Classifier.naivebayes(Classifier.getwords)

#remove previous db file
check_output(['rm', 'jpindell.db'])

c.setdb('jpindell.db')

Classifier.spamTrain(c)

#classify files as Spam or Not Spam
file1 = open('Testing Dataset\\Email1.txt')
email1 = file1.read()
print( c.classify(email1) )

file2 = open('Testing Dataset\\Email2.txt')
email2 = file2.read()
print( c.classify(email2) )

file3 = open('Testing Dataset\\Email3.txt')
email3 = file3.read()
print( c.classify(email3) )

file4 = open('Testing Dataset\\Email4.txt')
```

2. **Classification Results:** The expected output was that the first half of the text files would return as “Not Spam”, and the second half would return as “Spam”. The actual results showed that the first half of the text files returned as “Not Spam”, and the second half returned as “Spam”. In other words, the classifier was 100% accurate. A snippet of the results are shown below:

```
C:\Users\James Pindell\Desktop\CS 432\Assignment 8>python ClassifierTest.py
C:\Users\James Pindell\Desktop\CS 432\Assignment 8\Classifier.py:13: FutureWarning: split() requires a non-empty pattern
 match.
  words=[s.lower() for s in splitter.split(doc)
Not Spam
Not Spam
Not Spam
Not Spam
Not Spam
Not Spam
Not Spam
Not Spam
Not Spam
Not Spam
Spam
Spam
Spam
Spam
Spam
Spam
Spam
Spam
Spam
Spam
```