OLD DOMINION UNIVERSITY

Assignment 8

David Bayard

April 13, 2019

QUESTION 1.

Create two datasets; the first called Testing, the second called Training.

Solution:

Two datasets were created by copying and pasting the text from 20 emails from the Spam folder, and another 20 from the inbox. Specific emails that contained certian query terms such as "ODU", or "Monarchs" were selected to increase the odds of the classifier selecting the correct class per each document. Moreover, most of the emails shared a similar topic, being school related. This was purposefully done to test the classifier to a fuller extent.

The emails are split into two directories, being Spam and Not Spam. Within these directories, another two directories exist, called Test and Train. These folders represent the testing and training sets, containing 20 emails each. First, the training sets of Spam and Not Spam text files are run through the classifier, then the text files in the test directories are fed to the classifier.

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.

Solution:

As specified above, the PCI book implements the Naive Bayes classifier functionality. Changes were made to the docclass.py file in order to read from the.txt files, instead of the SQLite database. Other then these changes, the original functions supplied by the PCI book were used as supplied.

In order to train the Naive Bays classifier, three lines of code needed to be executed. First, an instance of the naivebays is instantiated and holds a reference to the getwords function. This is important becuase this function will later be used to extract all of the text from each of the .txt files, returning a lowercase count of each term.

```
1 cl=docclass.naivebayes(docclass.getwords)
2
3 docclass.spamTrain(cl)
4 docclass.testClassifier(cl)
```

Listing 1: test.py driver function

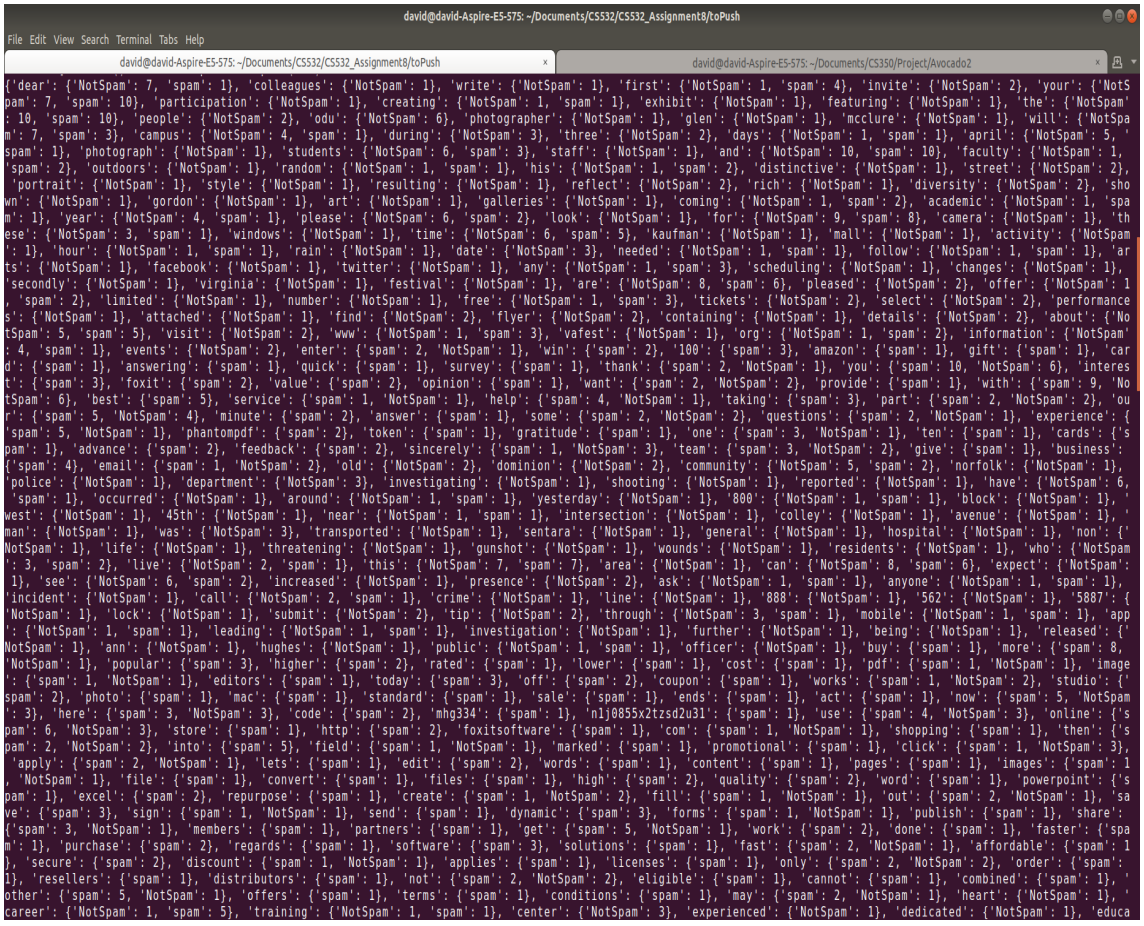
When the naivebayes object is instantiated, a classifier object is created, and passed a reference the getwords function. In the function below, the spam files numbered from 1-11 are opened, and read into the InputStream, and then into strings. Every iteration, these strings are passed to the classifiers train method, which converts the words to lowercase, and keeps count of how many documents contain each word, and in which class they fall into. Sample output of this classification is shown below.

```
for r in range(1,11):
    spamFileName = "spamFile" + str(r)
    notSpamFileName = "notSpam" + str(r)

    spamFile = open("./Spam/Train/spam" + str(r) + ".txt", "r")
    notSpamFile = open("./Not_Spam/Train/NotSpam" + str(r) + ".txt", "r")
    text = spamFile.read()
    text2 = notSpamFile.read()

    cl.train(text2, "NotSpam")
    cl.train(text, 'spam')
```

Listing 2: Pass file input stream as string to classifier



(a) Extracting Text

As seen in the image above, the words from all of the .txt files read in the function above are being extracted, transformed to lowercase, and counted. After the loop iterating over the files finishes running, the classifier object will contain each word in each document, a count of that word per document, and the class to which the word belongs. This data will be used when running the test files through the classifier to determine the correct class.

With the training data loaded into the classifier object, the testing data may now be loaded using the classify method in order to determine which class the test document belongs to. The code below loops over the remaining 20 documents, and predicts whether each documement is spam or not spam.

```
for r in range(11,21):

    spamFileName = "spamFile" + str(r)
    notSpamFileName = "notSpam" + str(r)

    spamFile = open("./Spam/Test/spam" + str(r) + ".txt", "r")
    notSpamFile = open("./Not_Spam/Test/NotSpam" + str(r) + ".txt", "r")
    text = spamFile.read()
    text2 = notSpamFile.read()

    predictNotSpam = cl.classify(text2, default='unknown')
    predictSpam = cl.classify(text,default='unknown')
```

Listing 3: Clustering into 5 groups

The results of the classify function, as well as a title line from each document are recorded in the documentation.txt file. From reading the documentation.txt file, it is visible that the classifier issued one false positive, classifying notSpam15.txt as spam.

Overall, the classifier performed well, labeling 19 out of 20 documents in the proper class, but ranking the notSpam document as spam is a big mistake. This would be fixed by increasing the threshhold for a document to be classified as spam. While this may increase the odds of labeling spam documents as not spam, it helps prevent documents that are not spam from being labeled as spam.

QUESTION 3

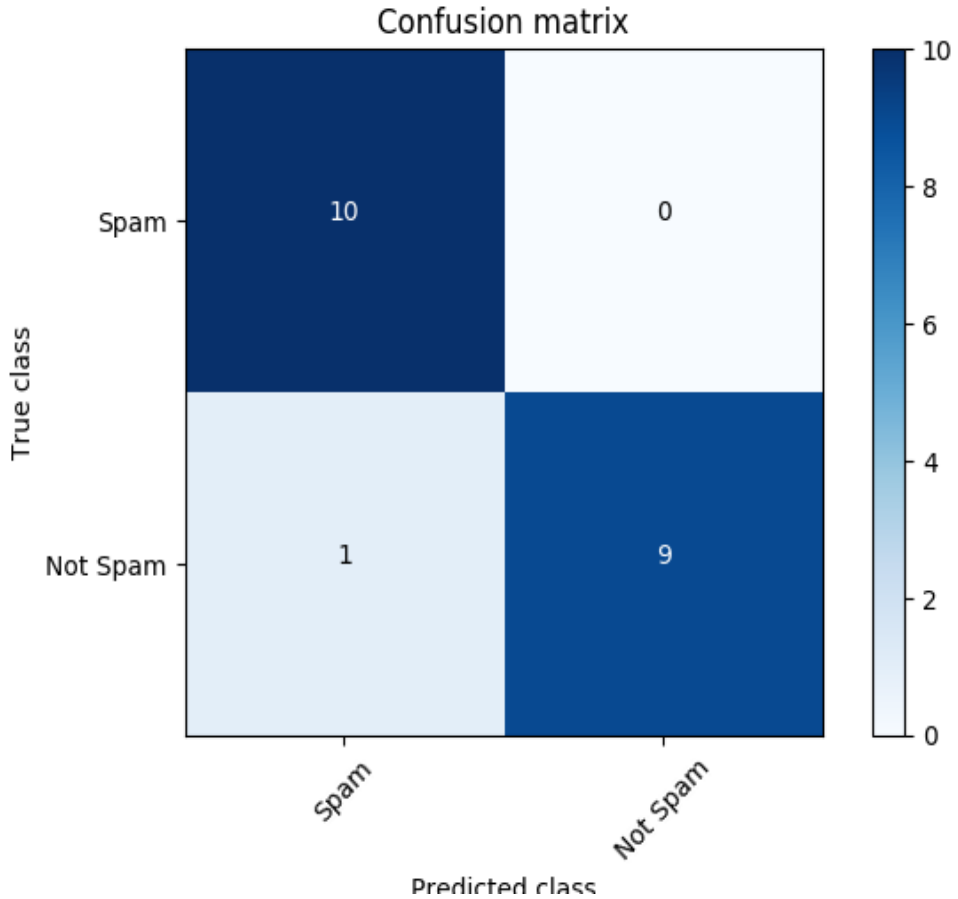
Draw a confusion matrix for your classification results

Solution:

The sklearn library is used here to draw the confusion matrix. It takes the true class values along with the predicted class values and draws the confusion matrix.

In the confusion matrix shown below, the top left corner represents the properly selected spam class, and the bottom left row represents false positives of spam. Thus out of the 10 spam documents, 10 were properly predicted as spam, but 1 document was classified as spam when it is not spam.

The bottom right corner represents the documents that are not spam. The classifier predicted only 9 documents as not spam, and the top right row displaying false negatives remains empty.



QUESTION 4

Report the precision and accuracy scores of your classification results

Solution:

The formulas necessary to calculate the accuracy and precision are displayed below.

$$Precision = \frac{tp}{tp + fp}$$
$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

The equations above represent tp as true positive, fp = false positive, tn as true negative, and fn as false negative. The equations are filled in with the values from the confusion matrix.

$$Precision = \frac{10}{10+1} = \frac{10}{11} = .91$$
$$Accuracy = \frac{10+9}{9+10+1+0} = \frac{19}{20} = .95$$

As seen above, the classifier has a fairly high precision and accuracy. These values would undoubtedly change given a larger sample size, but for a sample of only 20 files, the values are reasonable.