Appendix A

```
import collections
from collections import Counter
import math
from sklearn.metrics import confusion_matrix
import numpy as np
import pandas as pd

# set of all words in docs
vocab = set()

# list of all words
allWords = []

# count of words in vocab
countVocab = 0

# list of all document labels
allDocs = []

# set of all document labels
docTypes = set()

# dictionary of count of each doc type
docCou1nt = {}

# dictionary of probabilities of each doc type
docProbabilities = {}

# dictionary of concatenated docs of each type
docCombined = {}

# dictionary of word counts for each doc type plus count of words in vocab
docWordCount = {}

# dictionary of dictionaries of word probabilities for each doc type
docWordProb = collections.defaultdict(dict)

# dictionary of minimum word probabilities for each doc type
docMinProb = {}

# total number of documents
numDocs = 0

# label for training classes
label = ""

# list of correct classes for test set
testAnswers = []

# list of assigned classes for test set
testGuesses = []

# open and read training file
trainingFile = open(r"C:\Users\jeffp\OneDrive\Documents\GitHub\CIS_678_Project2\forumTraining.data", "r")

# read data from training file and close
for row in trainingFile:
    words = row.split()
    label = words[0]
    allDocs.append(label)
    docTypes.add(label)
    if label in docCombined.keys():
        docCombined[label] += words
    else:
        docCombined[label] = words
    del words[0]
    allWords += words
    for w in words:
```

```python
        vocab.add(w)
trainingFile.close()

# count total words in vocab
countVocab = len(vocab)

# count total number of docs and number of each type
numDocs = len(allDocs)
docCount = Counter(allDocs)

# calculate probabilities, word counts, and word probabilities for each doc type
for doc in docTypes:
    docProbabilities[doc] = docCount[doc] / numDocs
    docWordCount[doc] = len(docCombined[doc]) + countVocab
    wordCount = Counter(docCombined[doc])
    docMinProb[doc] = 1 / docWordCount[doc]
    for w in wordCount:
        docWordProb[doc][w] = (wordCount[w] + 1) / docWordCount[doc]


# classifies a new document given a list of its words
def classifyDoc(wordList):

    # bring in global variables used
    global docTypes
    global docWordProb
    global docMinProb

    # set starting maximum probability to negative infinity
    maxProb = float('-inf')

    # find the most likely class for the given word list
    guessClass = ""
    for d in docTypes:

        # add the logs of the probabilities instead of multiplying the base figures to avoid underflow
        prob = math.log(docProbabilities[d])
        for word in wordList:
            if word in docWordProb[d].keys():
                prob += math.log(docWordProb[d][word])
            else:
                prob += math.log(docMinProb[d])

        # no need to convert the log figures back since they are only used in comparisons amongst themselves
        if prob > maxProb:
            maxProb = prob
            guessClass = d

    return guessClass


# sets the guesses and answers for the test set, returns the % correct
def naiveBayes(exclusionRate):

    # bring in global variables needed
    global allWords
    global vocab
    global testGuesses
    global testAnswers

    # clear current guesses and answers
    testAnswers.clear()
    testGuesses.clear()

    # open test file and set correct count to 0
    testFile = open(r"C:\Users\jeffp\OneDrive\Documents\GitHub\CIS_678_Project2\forumTest.data", "r")
    correctCount = 0

    # create list of the top n% most common words from the training set
    mostCommon = [word for word, word_count in Counter(allWords).most_common(int(len(vocab) * exclusionRate))]

    # read in the test docs, remove most common words, and predict correct class
    for line in testFile:
        words1 = line.split()
        testAnswers.append(words1[0])
        del words1[0]
        wordsNoCommon = [word for word in words1 if word not in mostCommon]
        testGuesses.append(classifyDoc(wordsNoCommon))

    testFile.close()

    # check the correct guess rate and return it
    for i, val in enumerate(testAnswers):
        if testGuesses[i] == val:
            correctCount += 1
    return correctCount / len(testGuesses)


# finds the optimal n% (between .1% and 1%) most common words to exclude for highest correct rate
def findOptimalExclusion():
    exclusionRate = 0.000
    maxAccuracy = -1
    optimalExclusion = exclusionRate
    while exclusionRate < 0.011:
        accuracy = naiveBayes(exclusionRate)
        if accuracy > maxAccuracy:
            maxAccuracy = accuracy
            optimalExclusion = exclusionRate
        exclusionRate += 0.001
    print('optimal exclusion: %r' % "{:.2%}".format(optimalExclusion))
    return optimalExclusion


# find the optimal exclusion % and use that to create lists of answers and guesses
correctRate = naiveBayes(findOptimalExclusion())

# create labels for confusion matrix and set display settings
docTypesList = list(docTypes)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

# compare answers and guesses using precision, recall, CM, F1, and misclassification rate
# recall and precision method taken from https://stats.stackexchange.com/questions/51296
cm = confusion_matrix(testAnswers, testGuesses, labels=docTypesList)
recall = np.diag(cm) / np.sum(cm, axis=1)
precision = np.diag(cm) / np.sum(cm, axis=0)
meanRecall = np.mean(recall)
meanPrecision = np.mean(precision)
f1 = (meanPrecision * meanRecall) / (meanPrecision + meanRecall)
misclassificationRate = 1 - correctRate

# print results
print('Recall: %r' % "{:.2%}".format(meanRecall))
print('Precision: %r' % "{:.2%}".format(meanPrecision))
print('F1: %r' % "{:.2%}".format(f1))
print('Misclassification Rate: %r' % "{:.2%}".format(misclassificationRate))
cmDataFrame = pd.DataFrame(cm, index=docTypesList, columns=docTypesList)
print(cmDataFrame)

# write results to files
cmDataFrame.to_csv('confusionMatrix.csv')
with open("results.txt", 'w') as f:
    f.write('Recall: %r' % "{:.2%}".format(meanRecall) + '\n')
    f.write('Precision: %r' % "{:.2%}".format(meanPrecision) + '\n')
    f.write('F1: %r' % "{:.2%}".format(f1) + '\n')
    f.write('Misclassification Rate: %r' % "{:.2%}".format(misclassificationRate) + '\n')
    f.write('Classes Guesses \n')
    for i, val in enumerate(testAnswers):
        f.write(val + ' ' + testGuesses[i] + '\n')
    f.close()

```