Assignment 4_Part1

@authors: Priyanka Jha and SivaKumar AP

Part1 of assignment4 is a spam classification problem. ***spam.py*** is the program file submitted for this problem and contains the training and classification code for both Naïve Bayes and Decision Tree classifiers.

Caveat: The program uses the argument model name to build the Naive Bayes and the decision Tree classifier, so it is important that we pass the argument carefully for train and reuse it for test or else error will be thrown as classifier relevant objects will not be built.

Naïve Bayes classifier has a simple continuous feature implementation done in spam.py. Two separate counter objects are created to hold the counts of spam and non-spam words read from the email folders spam and notspam. As the requirement stated, a bag of words model is implemented where all the words in the document are read and number of occurrence are counted. The files in the train folders are read and split into words, then regular expression is used to create a bag of words. The model created during the training (i.e. the counter objects are stored in Pickle object). Please refer to the code to see the implementation.

The idea is to use Bayes law for classification –

P(spam|the words in the document to be classified) = P(Words in the document|spam email)*P(spam email)/ P(words in the document)

Or, P(spam|the words in the document to be classified) is proportional to P(Words in the document|spam email)*P(spam email)

For each document, posterior of all classes (binary in this case) is calculated using the above calculations, and class with maximum posterior is assigned to that document. For these calculation the pickle objects are used that were created during the training so that same calculations do not need to be redundantly done.

A separated function has been created to display the percent accuracy and the confusion matrix. All the true positives, true negatives, false positives, and false negatives are calculated. The percent accuracy is created using the formula (tp+tn)*100/total number of documents.

The confusion matrix is printed as –

Total Docs	predicted no	predicted yes
Actual Yes	TN	FP
Actual No	FN	TP

Classification results: As mentioned earlier, the model trained on the train folder is used to do classification on the test folder. The results are as below:

```
---------------------------------Percent Accuracy--------------------------
The accuracy of the model is:  97.37666405638214 %

------------------Confusion Matrix for Naive Bayes Classifier-----------
Total =2554 ---------------Predicted NO-----------------Predicted YES
```

```
Actual YES----------------- TN = 1355 -----------------FP = 14
Actual NO-------------------FN = 53 -------------------TP = 1132
```

Naïve Bayes implementation on binary feature is also done. However, the performance of NaiveBayes model with continuous data is better giving more accurate results.

Ten words associated with spam and notspam documents:

```
10 words most associated with Spam
ssz 0.9337520058244441
cypherpunks 0.9335503569308581
minder 0.9333871439696937
cpunks 0.9333679420288192
iiq 0.932922842969191
enenkio 0.9328434946296404
insuranceiq 0.9326747946342285
000080 0.932386130197635
kingdom 0.9320915746500903
zzzzason 0.9320476960305942
3cfont 0.9320025900371258
2ffont 0.9320025900371258

10 words most associated with NotSpam
cnet 1.06339054560425
exmh 1.0633130627257894
freshrpms 1.0633089995764204
clickthru 1.0631171016846233
zzzlist 1.0629956262863844
lockergnome 1.0629748932952432
redhat 1.062915945957062
zdnet 1.062899246590071
razor 1.062796186428717
listman 1.0626728147529734
rpm 1.0626373712487445
egwn 1.0626141640346674
```

Decision tree implementation using continuous feature is also done in spam.py file. A sparse vector is created with all the words present in the documents in train folder and labelled as spam and notspam. There is one row per document in the dataTable object which is a list of lists. The featureList list object is used to hold the list of features (words). The index of the word in featureList matches with the index of the count of that word in a document row in dataTable. If the word is not present, the count in the dataTable is zero. Data cleaning and preprocessing is done is the similar fashion as Naïve Bayes and same function is used to increase modularity.

The function buildtree is used to create the tree in the train mode. This function is called recursively to see which column gives better split of the dataTable and its subsets. For the dataTable created, a function called divideset is used to split the dataTable (or a subset of it) into two sets based on the value that is passed to the function. For example if set is to be divided for feature “sincere” for value 3, it will use the index of “sincere” from featureList and then divide the dataTable into 2 sets. The first set will

contain all rows where the count of this feature is greater than or equal to 3. All other rows are appended to the second set.

A function called entropy calculates the entropy of a dataset across all possible class values for the dataTable or its subset. Simply put, entropy is the sum of all p(x)*log(p(x)), where p(x) is the ratio of the class to total count of all classes. The higher the entropy, the less homogenous the set is. The idea of course is to achieve maximum homogeneity for a set. Entropy is opposite of the information gain, which is being used in this program to calculate the homogeneity. The column with maximum information gain is chosen to be the root node and the same process is used recursively to choose the next node and so on. The split of the entire dataTable that gives the maximum information gain is the root node of the decision tree. The split is done using divideset function on the mean value of the column values.

A class called decisionnode is used to store the split of the tree, i.e. set 1 from divideset is store in tb (true branch) and set 2 is stored in fb (false branch). The column and the mean value that gives the maximum information gain is also included in the decisionnode class. The results in this class is the dictionary of results for the branch that is split and hold the unique counts if the information gain becomes zero, else the value is None. The tree model and featureList is stored in the Pickle object for reuse during classification.

The recursive printtree is called to print a simple visual representation of the tree created. The tree is created as shown in the appendix for the documents in the train folder using index number of the column.

Classification results: As mentioned earlier, the model trained on the train folder is used to do classification on the test folder. The accuracy and the confusion matrix are created using the same function as for Bayes classifier to promote modularity. The results are as below:

```
--------------------------------Percent Accuracy-------------------------

The accuracy of the model is:  93.18715740015662 %

-----------------Confusion Matrix for Decision Tree Classifier---------

Total = 2554 ---------------Predicted NO-----------------Predicted YES
Actual YES-------------------TN = 1262 ---------------------FP = 107
Actual NO--------------------FN = 67 ----------------------TP = 1118
```

Classification result for decision tree on binary feature:

A simple code change was needed to model the decision tree on binary feature. Instead of splitting the dataTable on the mean of continuous column value, the dataset was split on hard-coded value 1. After training the decision tree model on binary feature, there was a huge increase in the classification accuracy. Here is the accuracy and confusion matrix observed:

```
--------------------------------Percent Accuracy-------------------------
```

```
The accuracy of the model is:  98.66875489428348 %

------------------Confusion Matrix for Decision Tree Classifier---------
Total = 2554 ---------------Predicted NO-----------------Predicted YES
Actual YES----------------------TN = 1346 ----------------FP = 23
Actual NO-----------------------FN = 11 ------------------TP = 1174
```

Thus, the model built on the binary feature is better for decision tree classification of "spam and not spam" documents.

Appendix –

1. Full decision tree created from training the continuous model -

```
59285 : 1 ?
T-> 59285 : 1 ?
  T-> 61976 : 1 ?
    T-> 79545 : 1 ?
      T-> {'notspam': 1119}
      F-> 58504 : 1 ?
        T-> {'notspam': 6}
        F-> {'spam': 2}
    F-> 93279 : 1 ?
      T-> 101722 : 1 ?
        T-> {'notspam': 4}
        F-> 23163 : 1 ?
          T-> {'spam': 16}
          F-> {'notspam': 2}
      F-> {'notspam': 27}
  F-> 60083 : 1 ?
    T-> 91872 : 1 ?
      T-> {'spam': 30}
      F-> 70257 : 1 ?
        T-> {'notspam': 2}
        F-> 88174 : 1 ?
          T-> {'notspam': 2}
          F-> {'spam': 14}
    F-> 11414 : 1 ?
      T-> {'notspam': 8}
      F-> 1453 : 1 ?
        T-> {'notspam': 3}
        F-> {'spam': 12}
F-> 24 : 1 ?
  T-> 35852 : 1 ?
    T-> 7283 : 1 ?
      T-> 21269 : 1 ?
        T-> 40661 : 1 ?
          T-> {'spam': 29}
          F-> 14150 : 1 ?
```

```
            T-> {'notspam': 6}
            F-> {'spam': 1}
        F-> 33423 : 1 ?
          T-> 23966 : 1 ?
            T-> 17443 : 1 ?
              T-> {'notspam': 32}
              F-> 33610 : 1 ?
                T-> {'notspam': 6}
                F-> {'spam': 1}
            F-> {'spam': 1}
          F-> {'spam': 9}
      F-> 69894 : 1 ?
        T-> {'notspam': 67}
        F-> 77101 : 1 ?
          T-> {'notspam': 39}
          F-> 20921 : 1 ?
            T-> {'spam': 7}
            F-> 18947 : 1 ?
              T-> {'notspam': 14}
              F-> {'spam': 1}
    F-> 16535 : 1 ?
      T-> 75381 : 1 ?
        T-> {'notspam': 27}
        F-> 51597 : 1 ?
          T-> {'notspam': 5}
          F-> {'spam': 7}
      F-> 51597 : 1 ?
        T-> {'spam': 79}
        F-> 97331 : 1 ?
          T-> 66038 : 1 ?
            T-> 5846 : 1 ?
              T-> {'notspam': 6}
              F-> {'spam': 1}
            F-> {'spam': 35}
          F-> 23163 : 1 ?
            T-> {'spam': 4}
            F-> 49770 : 1 ?
              T-> {'notspam': 15}
              F-> 10415 : 1 ?
                T-> {'spam': 3}
                F-> {'notspam': 1}
  F-> 61939 : 1 ?
    T-> 100108 : 1 ?
      T-> 79279 : 1 ?
        T-> {'spam': 127}
        F-> 37774 : 1 ?
          T-> {'notspam': 2}
          F-> 28485 : 1 ?
            T-> {'notspam': 1}
            F-> 4864 : 1 ?
              T-> {'spam': 34}
              F-> 82202 : 1 ?
                T-> {'notspam': 1}
                F-> {'spam': 11}
```

```
      F-> {'spam': 518}
    F-> 18947 : 1 ?
      T-> 47226 : 1 ?
        T-> {'notspam': 2}
        F-> 7464 : 1 ?
          T-> {'spam': 126}
          F-> 65525 : 1 ?
            T-> 47013 : 1 ?
              T-> {'spam': 16}
              F-> 97417 : 1 ?
                T-> 8282 : 1 ?
                  T-> {'spam': 1}
                  F-> {'notspam': 5}
                F-> 1452 : 1 ?
                  T-> {'notspam': 1}
                  F-> {'spam': 11}
            F-> 76566 : 1 ?
              T-> 91830 : 1 ?
                T-> 12626 : 1 ?
                  T-> {'spam': 4}
                  F-> {'notspam': 2}
                F-> {'spam': 16}
              F-> {'spam': 64}
      F-> 93279 : 1 ?
        T-> 5531 : 1 ?
          T-> 474 : 1 ?
            T-> {'notspam': 1}
            F-> {'spam': 6}
          F-> {'spam': 27}
        F-> 22376 : 1 ?
          T-> {'notspam': 26}
          F-> {'spam': 1}
```