CA02: Spam eMail Detection using Naive Bayes Classification Algorithm

Assignment Description

In this exercise we shall train the model with set of emails labelled as either from Spam or Not Spam. There are 702 emails equally divided into spam and non spam category. Next, we shall test the model on 260 emails. We shall ask model to predict the category of this emails and compare the accuracy with correct classification that we already know.

Instructions

I have created a git repository for the sample code (solution). You can fork or download it from: https://github.com/ArinB/MSBA-CA01-Spam-Mail-Naibe-Bayes. The data folders are in a Zip file at BrightSpace CA02 assignment folder.

Use the sample code to understand how the problem is solved. Then construct your own Colab / Jupyter Notebook using your own variables and showing step by step results with intermittent "print" statements in as much detail as you can. It's your job to explain clearly your understanding of the process at every step.

IMPORTANT:

- Keep all your Data folders/files under the above path
- Based on your Algorithm Assignment, CA02 will change (to CA03, CA04 … etc.)
- Your data folders "test-mails" and "train-mails" must be under your current folder where you will create your notebook.
- In your code you must use the relative path name of the data folder locations ("./test-mails", "./train-mails".
- **DO NOT CHANGE THE DATA FOLDER and DATA FILE NAMES** or tamper with the ORIGINAL DATA, as your program will be evaluated by running against the original data.

Code Explanation

Cleaning and Preparing the data

We have two folders ***test-mails*** and ***train-mails***. We will use train-mails to train the model. The sample email data set looks like:

```
Subject: re : 2 . 882 s - > np np> deat : sun , 15 dec 91 2 : 25 : 2 est > : michael <
mmorse @ vm1 . yorku . ca > > subject : re : 2 . 864 query
> > wlodek zadrozny ask " anything interest " > construction " s > np np " . . . second ,
> much relate : consider construction form > discuss list late reduplication ? > logical
sense " john mcnamara name " tautologous thus , > level , indistinguishable " , , here ? "
. ' john mcnamara name ' tautologous support those logic-base semantics irrelevant natural
language . sense tautologous ? supplies value attribute follow attribute value . fact
value name-attribute relevant entity ' chaim shmendrik ' , ' john mcnamara name ' false .
tautology , . ( reduplication , either . )
```

First line is subject and the content starts from the third line.
If you navigate to any of the train-mails or test-mails, you shall see file names in two patterns:

number-numbermsg[number].txt : example **3-1msg1.txt** (this are non spam emails)ORspmsg[Number].txt : example **spmsga162.txt (**these files are of spam emails)**.**

The very first step in text data mining task is to clean and prepare the data for a model. In **cleaning** we remove the non required words, expressions and symbols from text.

Consider a text: *"Hi, this is Alice. Hope you are doing well and enjoying your vacation."*

Here the words like is, this, are, and etc. don't really contribute to the analysis. Such words are also called **stop words**. Hence in this exercise, we consider only most frequent 3000 words of dictionary from email. Following is a code snippet.

After cleaning what we need from every email document, we should have some matrix representation of the word frequency.

For example if document contains the text: "Hi, this is Alice. Happy Birthday Alice" after cleaning, we want something in line:

```
word      :   Hi this is Alice Happy Birthday
frequency :   1   1    1  2      1      1
```

And we need this for every document. The **extract_features** function below does this and then removes less common words for every document.

```
def make_Dictionary(root_dir):
   all_words = []
   emails = [os.path.join(root_dir,f) for f in os.listdir(root_dir)]
   for mail in emails:
        with open(mail) as m:
            for line in m:
                words = line.split()
                all_words += words
   dictionary = Counter(all_words)
# if you have python version 3.x use commented version.
   # list_to_remove = list(dictionary)
   list_to_remove = dictionary.keys()   for item in list_to_remove:
       # remove if numerical.
       if item.isalpha() == False:
            del dictionary[item]
        elif len(item) == 1:
            del dictionary[item]
    # consider only most 3000 common words in dictionary.
dictionary = dictionary.most_common(3000)
return dictionary
```

make_Dictionary reads the email files from a folder, constructs a dictionary for all words.
Next, we delete words of length 1 and that are not purely alphabetical.
At last we only extract out 3000 most common words.

Extracting features and corresponding label matrix.

Next with the help of dictionary, we generate a label and word frequency matrix

```
word      :   Hi this is Alice Happy Birthday
frequency :   1   1    1  2      1      1
```

```
def extract_features(mail_dir):
  files = [os.path.join(mail_dir,fi) for fi in os.listdir(mail_dir)]
  features_matrix = np.zeros((len(files),3000))
  train_labels = np.zeros(len(files))
  count = 0;
  docID = 0;
  for fil in files:
    with open(fil) as fi:
      for i,line in enumerate(fi):
        if i == 2:
          words = line.split()
          for word in words:
            wordID = 0
            for i,d in enumerate(dictionary):
              if d[0] == word:
                wordID = i
                features_matrix[docID,wordID] = words.count(word)
      train_labels[docID] = 0;
      filepathTokens = fil.split('/')
      lastToken = filepathTokens[len(filepathTokens) - 1]
      if lastToken.startswith("spmsg"):
          train_labels[docID] = 1;
```

```
            count = count + 1
        docID = docID + 1
    return features_matrix, train_labels
```

Training and predicting with sklearn Naive Bayes

Basically, sklearn Naive Bayes provides three alternatives for model training:

- **Gaussian:** It is used in classification and it assumes that features follow a normal distribution.
- **Multinomial:** It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number x_i is observed over the n trials".
- **Bernoulli:** The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification with 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

In this exercise we shall use Gaussian. The sample code snippet looks like

```
TRAIN_DIR = "<use FULL path>/train-mails"
TEST_DIR = "<use FULL path>//test-mails"

dictionary = make_Dictionary(TRAIN_DIR) # using functions mentioned above.
features_matrix, labels = extract_features(TRAIN_DIR)
test_feature_matrix, test_labels = extract_features(TEST_DIR)

# from sklearn.naive_bayes import GaussianNB

model = GaussianNB()

#train model
model.fit(features_matrix, labels)

#predict
predicted_labels = model.predict(test_feature_matrix)
```

Accuracy Score

Next we compare the accuracy score for predicted labels. Accuracy score is just percentage of correct predictions. Again here, sklearn provides neat implementation for accuracy score calculation.

```
# from sklearn.metrics import accuracy_score

accuracy = accuracy_score(test_labels, predicted_labels)
```

Conclusion

The Naive Bayes considers the independence in features. For example it assumes the occurrence of one word/ feature is independent of other. But in real life it may not be so (occurrence of *morning* is high after *Good*).

Submitting Your Work

1. Complete your work in the Colab / Jupyter NB environment
2. Create a GitHub Repository for CA02
3. Upload the Notebook to this GitHub repository and Commit / Push. Do not forget to add a Readme.md file in your GitHub repository folder.
4. Upload your Notebook file (.ipynb) and Readme.md file at the BrightSpace Assignment Folder