CS405 Machine Learning

Lab #3 Bayes

Text mining (deriving information from text) is a wide field which has gained popularity with the huge text data being generated. Automation of a number of applications like sentiment analysis, document classification, topic classification, text summarization, machine translation, etc., has been achieved using machine learning models. In this lab, you are required to write your spam filter by using the naïve Bayes method. This time you should not use 3 rd party libraries including scikit-learn.

Instruction

Spam filtering is a beginner's example of the document classification task which involves classifying an email as spam or non-spam (a.k.a. ham) mail. An email dataset will be provided. We will use the following steps to build this application:

1. Preparing the text data
2. Creating a word dictionary
3. Feature extraction
4. Training the classifier
5. Checking the results on the test set

Preparing the text data

The data-set used here, is split into a training set and a test set containing 702 mails and 260 mails respectively, divided equally between spam and ham mails. You will easily recognize spam mails as it contains `spmsg` in its filename.

In any text mining problem, text cleaning is the first step where we remove those words from the document which may not contribute to the information we want to extract. Emails may contain a lot of undesirable characters like punctuation marks, stop words, digits, etc which may not be helpful in detecting the spam email. The emails in Ling-spam corpus have been already preprocessed in the following ways:

1. **Removal of stop words –** Stop words like "and", "the", "of", etc are very common in all English sentences and are not very meaningful in deciding spam or legitimate status, so these words have been removed from the emails.
2. **Lemmatization –** It is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. For example, "include", "includes," and "included" would all be represented as "include". The context of the sentence is also preserved in lemmatization as opposed to stemming (another buzz word in text mining which does not consider meaning of the sentence)

We still need to remove the non-words like punctuation marks or special characters from the mail documents. There are several ways to do it. Here, we will remove such words after creating a dictionary, which is a very convenient method to do so since when you have a dictionary; you need to remove every such word only once.

Creating word dictionary

We will only perform text analytics on the content to detect the spam mails. As the first step, we need to create a dictionary of words and their frequency. For this task, a training set of 700 mails is utilized. This python function will create the dictionary for you.

```
def make_Dictionary(train_dir):
    emails = [os.path.join(train_dir,f) for f in os.listdir (train_dir)]
    all_words = []
    for mail in emails:
        with open (mail) as m:
            for i,line in enumerate (m) :
                if i == 2:
                    words = line.split()
                    all_words += words
    dictionary = Counter(all_words)
    # Paste code for non-word removal here

    return dictionary
```

Once the dictionary is created we can add just a few lines of code written below to the above function to remove non-words. Absurd single characters in the dictionary which are irrelevant here are also removed. Do not forget to insert the below code in the function of make_Dictionary:

```
list_to_remove = list(dictionary.keys())
for item in list_to_remove:
    if item.isalpha() == False:
        del dictionary[item]
    elif len(item) == 1:
        del dictionary[item]
dictionary = dictionary.most_common(3000)
```

```
import os
import numpy as np
from collections import Counter
def make_Dictionary(train_dir):
    emails = [os.path.join(train_dir,f) for f in os.listdir (train_dir)]
    all_words = []
    for mail in emails:
        with open (mail) as m:
            for i,line in enumerate (m) :
                if i == 2:
                    words = line.split()
                    all_words += words
    dictionary = Counter(all_words)
    list_to_remove = list(dictionary.keys())
    for item in list_to_remove:
        if item.isalpha() == False:
            del dictionary[item]
        elif len(item) == 1:
            del dictionary[item]
    dictionary = dictionary.most_common(3000)
    # Paste code for non-word removal here
    return dictionary
```

The dictionary can be seen by the command "print dictionary". You may find some absurd word counts to be high but don't worry, it's just a dictionary and you always have a chance to improve it later. If you use the provided dataset, make sure your dictionary has some of the entries given below as most frequent words. Here 3000 most frequently used words are chosen in the dictionary.

```
# To show the most frequent words in train-mails
dictionary = make_Dictionary('ling-spam/train-mails')
dictionary
```

Feature Extraction Process

Once the dictionary is ready, we can extract word count vector (our feature here) of 3000 dimensions for each email of the training set. Each **word count vector** contains the frequency of 3000 words in the training file. Of course you might have guessed by now that most of them will be zero. Let us take an example. Suppose we have 500 words in our dictionary. Each word count vector contains the frequency of 500 dictionary words in the training file. Suppose the text in the training file is "Get the work done, work done", then it will be encoded as $[0,0,0,0,0,.......0,0,2,0,0,0,......,0,0,1,0,0,...0,0,1,0,0,......2,0,0,0,0,0]$ Here, all the word counts are placed at the 296th, 359th, 415th, 495th elements of the word count vector in the length of 500 and the rest are zero.

The below python code will generate a feature vector matrix whose rows denote 700 files of the training set and columns denote 3000 words of the dictionary. The value at index ij will be the number of occurrences of the j^{th} word of the dictionary in the i^{th} file

```
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))
    docID = 0
    _i = 0
    print(len(files))
    for fil in files:
        _i+=1
        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]+=1
            docID = docID + 1
        print('\r','done {} files'.format(_i),flush=True,end='')
    return features_matrix
```

```
features_matrix = extract_features("ling-spam/train-mails")
features_matrix
```

```
702
 done 702 files
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 2., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Training the Classifiers

Here you should write your Naïve Bayes classifiers after fully understanding its principle.

```
########### Write Your Code Here ###########

############################################
```

Checking Performance

The test set contains 130 spam emails and 130 non-spam emails. Please compute accuracy, recall, F-1 score to evaluate the performance of your spam filter.

```
########### Write Your Code Here ###########

############################################
```

```
accuracy: 0.5269230769230769
recall: 0.2230769230769231
f1: 0.32044198895027626
```

Questions

1. Describe another real-world application where the naïve Bayes method can be applied
2. What are the strengths of the naïve Bayes method; when does it perform well?
3. What are the weaknesses of the naïve Bayes method; when does it perform poorly?
4. What makes the naïve Bayes method a good candidate for the classification problem, if you have enough knowledge about the data?