**10/4/2019**

**ASSIGNMENT**

MACHINE LEARNING

**TOPIC:** EMAIL SPAM FILTER using MACHINE LEARNING

**SUBMITTED BY: AMAN RAJ**

**PROBLEM STATEMENT**

Being tired of getting so much SPAM in the email inbox of the unwanted email address catering to different businesses, So, we want to develop a SPAM filter for it that will detect SPAM emails but let the non- SPAM emails through.

Email Spam filter is a beginner’s example of document classification task which involves classifying an email as spam or non-spam (a.k.a. ham) mail. Spam box in your Gmail account is the best example of this. 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 done using machine learning models.

A spam filter is a program that is used to detect unsolicited and unwanted email and prevent those messages from getting to a user's inbox. Like other types of [filter](https://whatis.techtarget.com/definition/filter)ing programs, a spam filter looks for certain criteria on which it bases judgments. For example, the simplest and earliest versions (such as the one available with Microsoft's Hotmail) can be set to watch for particular words in the subject line of messages and to exclude these from the user's inbox.

This method is not especially effective, too often omitting perfectly legitimate messages (these are called false positives) and letting actual spam through. More sophisticated programs, such as [Bayesian filter](https://whatis.techtarget.com/definition/Bayesian-filter)s or other [heuristic](https://whatis.techtarget.com/definition/heuristic) filters, attempt to identify spam through suspicious word patterns or word frequency.

NEED:

When you have a business, getting rid of spam is all the more important due to the fact that these can eat up a lot of your inbox space, as well as a lot of your time when you start clearing these out. These emails can also carry malware and viruses that can compromise company security and data. What can you do to stop these from inundating your work email, and by extension, to stop these from compromising your company’s security? You can use spam filters. Spam filtering is an important tool that your company should use to help keep these unwanted messages from entering your inboxes, and to keep people from clicking on potentially harmful emails. According to studies, more than half of the emails that you get are actually classified as junk or spam. This fact alone shows you that there is a large potential for security issues due to these messages, not to mention the drop-in productivity because of the time people will spend on deleting such emails from their inbox.

**SOLUTION**

STEPS:

1. Preparing the text data.
2. Creating word dictionary.
3. Feature extraction process
4. Training the classifier

DATASET: **C:\Users\AMAN RAJ\Desktop\ML\train-mails, C:\Users\AMAN RAJ\Desktop\ML\test-mails**

**1. 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 pre-processed in the following ways:

a) 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.

b) Lemmatization – It is the process of grouping together the different inflected forms of a word so they can be analysed 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. So As of now you don’t need to do anything.

**2. Creating word dictionary.**

It can be seen that the first line of the mail is subject and the 3rd line contains the body of the email. We will only perform text analytics on the content to detect the spam mails. As a first step, we need to create a dictionary of words and their frequency. For this task, training set of 700 mails is utilized. This python function creates the dictionary for you.

Once the dictionary is created, we can add just a few lines of code written below to the above function to remove non-words about which we talked in step 1. I have also removed absurd single characters in the dictionary which are irrelevant here. Do not forget to insert the below code in the function def make\_Dictionary(train\_dir). 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 the scope of improving it later. If you are following this blog with provided data-set, make sure your dictionary has some of the entries given below as most frequent words. Here I have chosen 3000 most frequently used words in the dictionary.

**3. Feature extraction process.**

Once the dictionary is ready, we can extract word count vector (our feature here) of 3000 dimensions for each email of 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 text in training file was “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 296th, 359th, 415th, 495th index of 500 length word count vector and the rest are zero.

The below python code will generate a feature vector matrix whose rows denote 700 files of training set and columns denote 3000 words of dictionary. The value at index ‘*ij’* will be the number of occurrences of jth word of dictionary in ith file.

**4. Training the classifiers.**

For building email spam filter, we will train mathematical model that learns a decision boundary in features space between the two classes. Here, I will be using [scikit-learn ML library](http://scikit-learn.org/stable/) for training classifiers. It is an open source python ML library which comes bundled in 3rd party distribution [anaconda](https://www.continuum.io/downloads) or can be used by separate installation following [this](http://scikit-learn.org/stable/install.html). Once installed, we only need to import it in our program.

Further, I have trained two models here namely **Naive Bayes** classifier and **Support Vector Machines** (SVM). Naive Bayes classifier is a conventional and very popular method for document classification problem. It is a supervised probabilistic classifier based on Bayes theorem assuming independence between every pair of features. SVMs are supervised binary classifiers which are very effective when you have higher number of features. The goal of SVM is to separate some subset of training data from rest called the support vectors (boundary of separating hyper-plane). The decision function of SVM model that predicts the class of the test data is based on support vectors and makes use of a kernel trick.

Once the classifiers are trained, we can check the performance of the models on test-set. We extract word count vector for each mail in test-set and predict its class (ham or spam) with the trained NB classifier and SVM model. Below is the full code for spam filtering application. You have to include the two functions we have defined before in step 2 and step

**Checking Performance: Email spam filter**

Let us check the performance of built email spam filter. Test-set contains 130 spam emails and 130 non-spam emails. If you have come so far, you will find below results. I have shown the confusion matrix of the test-set for both the models. The diagonal elements represent the correctly identified (a.k.a. true identification) mails where as non-diagonal elements represent wrong classification (false identification) of mails.

Both the models had similar performance on the test-set except that the SVM has slightly balanced false identifications. I must remind you that the test data was neither used in creating dictionary nor in the training set.

**PROGRAM**

import os

import numpy as np

from collections import Counter

from sklearn.naive\_bayes import MultinomialNB

from sklearn.svm import LinearSVC

from sklearn.metrics import confusion\_matrix

print("TRAINING DATA")

print("702 .txt Files->mail body data")

print("TESTING DATA")

print("260 .txt Files")

print("Please Wait...")

print("Feature Matrix: 760 x 3000 ")

print("5 X 5 Example")

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)

return dictionary

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;

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)

docID = docID + 1

for i in range(5):

for j in range(5):

print(features\_matrix[i][j],end=' ')

print()

return features\_matrix

train\_dir = 'C:\\Users\\AMAN RAJ\\Desktop\\ML\\train-mails'

dictionary = make\_Dictionary(train\_dir)

train\_labels = np.zeros(702)

train\_labels[351:701] = 1

train\_matrix = extract\_features(train\_dir)

model1 = LinearSVC()

model2 = MultinomialNB()

clf=model1.fit(train\_matrix,train\_labels)

clf2=model2.fit(train\_matrix,train\_labels)

test\_dir ='C:\\Users\\AMAN RAJ\\Desktop\\ML\\test-mails'

test\_matrix = extract\_features(test\_dir)

test\_labels = np.zeros(260)

test\_labels[130:260] = 1

result1 = model1.predict(test\_matrix)

result2 = model2.predict(test\_matrix)

print("\nLinearSVC Model")

print("Confusion Matrix:")

print(confusion\_matrix(test\_labels,result1))

print(clf.score(test\_matrix,test\_labels)\*100)

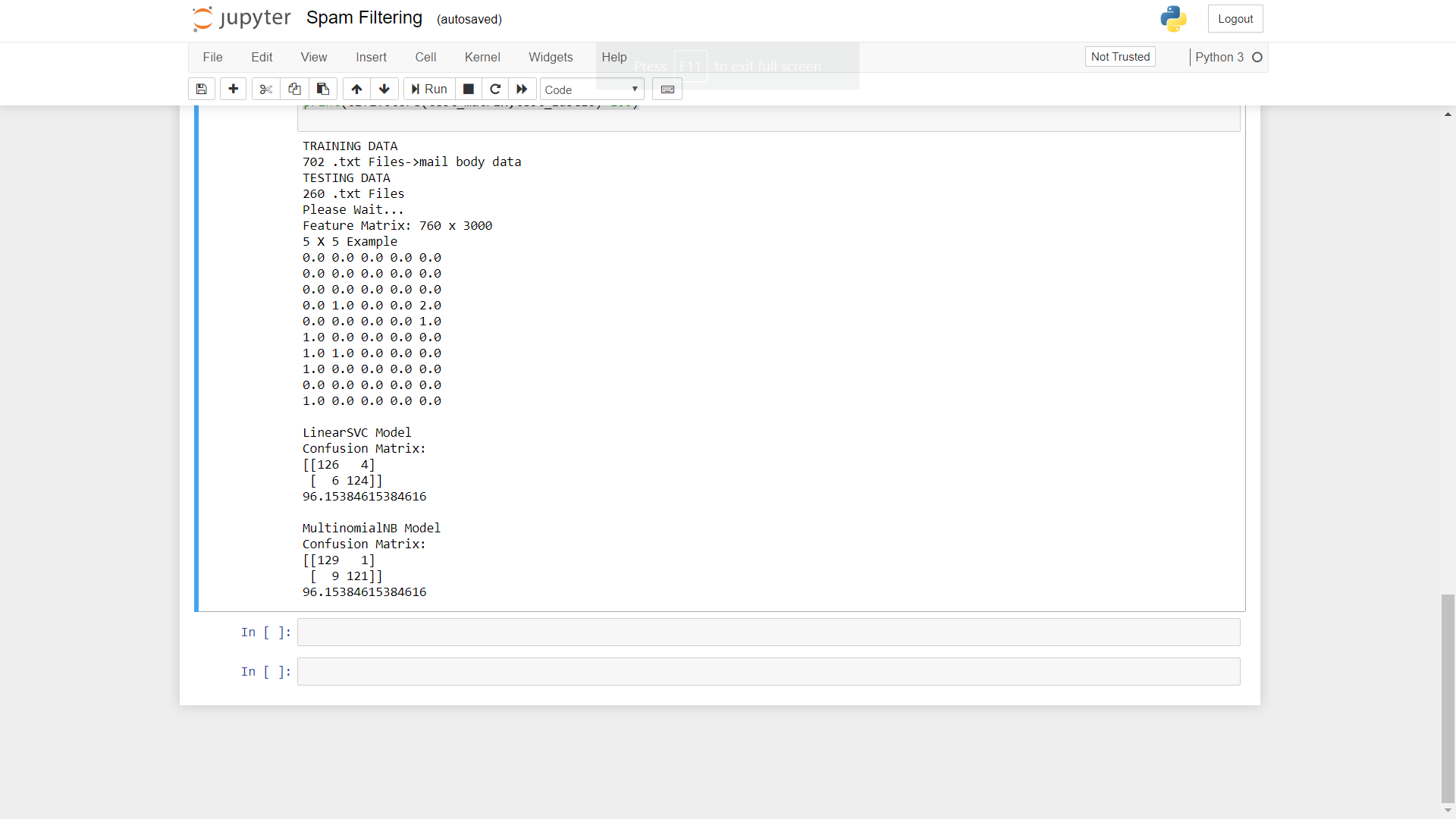
print("\nMultinomialNB Model")

print("Confusion Matrix:")

print(confusion\_matrix(test\_labels,result2))

print(clf2.score(test\_matrix,test\_labels)\*100)

**OUTPUT**



**RESULT**

**MODEL ACCURACY: 96% approximately**

**CONFUSION MATRIX:**

|  |  |  |
| --- | --- | --- |
| Multinomial NB | Ham | Spam |
| Ham | 129 | 1 |
| Spam | 9 | 121 |

|  |  |  |
| --- | --- | --- |
| SVM(Linear) | Ham | Spam |
| Ham | 126 | 4 |
| Spam | 6 | 124 |

SVM is mathematically complex model whereas Naive Bayes is relatively easy to understand. We are encouraged to study about these models from online sources. Apart from that, there can be a lot of experiments that can be done in order to find the effect of various parameters like

a) Amount of training data  
b) Dictionary size  
c) Variants of the ML techniques used (Gaussian NB, Bernoulli NB, SVC)  
d) Fine tuning of parameters of SVM models  
e) Improving the dictionary by eliminating insignificant words (may be manually)