CS5691

PATTERN RECOGNITION AND MACHINE LEARNING(PRML)

ASSIGNMENT 3

SUBMITTED BY: -

BISHAL KAHAR

MTECH(CSE)

CS22M032

SPAM or HAM?

Q) In this assignment, you will build a spam classifier from scratch. No training data will be provided. You are free to use whatever training data that is publicly available/does not have any copyright restrictions (You can build your own training data as well if you think that is useful). You are free to extract features as you think will be appropriate for this problem. The final code you submit should have a function/procedure which when invoked will be able to automatically read a set a emails from a folder titled test in the current directory. Each file in this folder will be a test email and will be named 'email#.txt' ('email1.txt', 'email2.txt', etc). For each of these emails, the classifier should predict +1 (spam) or 0 (non Spam). You are free to use whichever algorithm learnt in the course to build a classifier (or even use more than one). The algorithms (except SVM) need to be coded from scratch. Your report should clearly detail information relating to the data-set chosen, the features extracted and the exact algorithm/procedure used for training including hyperparameter tuning/kernel selection if any. The performance of the algorithm will be based on the accuracy on the test set.

Explanation:

Here is the Description of the data :

I'm using the Naive Bayes technique for classification in this assignment. Bernoulli distribution is used. I obtained the dataset I used for this project from the subsequent link:

https://www.kaggle.com/datasets/wanderfj/enron-spam

The description about the above-mentioned data can be found here:

http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/readme.txt

This data is divided into two distinct folders, one for non-spam/ham emails and the other for spam emails. Overall, it contains 5975 emails, of which 1500 are spam and 3672 are not.

Steps that the code took :

1) **In the first step we read the spam data and ham data in two separate lists as it is present in the text files.**
2) In the second step we have cleaned the data as part of our data-preprocessing. In this step we do three major operations
    I. **The punctuation is removed.**
    II. **The stopwords are eliminated.**
    III. **The non-alphanumeric words are eliminated.**
3) **The vectorized representation of the processed data is now being created. To begin, we first compile both spam and non-spam emails into a collection to send to the count-vectorizer. We can therefore obtain the frequency of each word in each mail from our corpus using the count vectorizer**

4) **I then calculate the likelihood that each phrase will appear in spam or not using the following formula:**

Probability that word I belongs to a category:

(the total number of mails in the category divided by the number of mails containing the term i)

It is determined as follows:

5) **separating the vectors into groups for spam and non-spam, then adding them all up. the corresponding total number of mails in that category, then dividing by that number. Now that I added the number of occurrences rather than the presence or absence indicator, the result does not give us a value between 0 and 1. As a result, I normalised the result to maintain the weight of each word. The more times it appears in a mail, the greater the contribution.**

6) **After that, I create the linear seperator and bias that are found in the decision function's logarithmic formula.**

Therefore, I run transpose(W) * the normalised frequency vector to obtain the anticipated labels of the train data. I can determine the train's correctness by comparing it to the actual labels as 98.76256767208044 %.

The final step involves the creation of a function named classifier(), which receives input from the test folder in the current directory and outputs either "email#.txt is predicted as spam" or "email#.txt is predicted as non-spam."

Note:- The console working directory should be used as the current directory for reading input for training and testing so that relative indirection for train and test data functions as intended.