Rigved Sah, CS20M053
Sumit Negi, CS20M067

1. 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). Two sample emails your classifier will be tested on can be found in the folder test. 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 hyper-parameter tuning/kernel selection if any. The performance of the algorithm will be based on the accuracy on the test set.

Solution: Spam messages are messages sent to a large group of recipients without their prior consent, typically advertising for goods and services and often business opportunities.

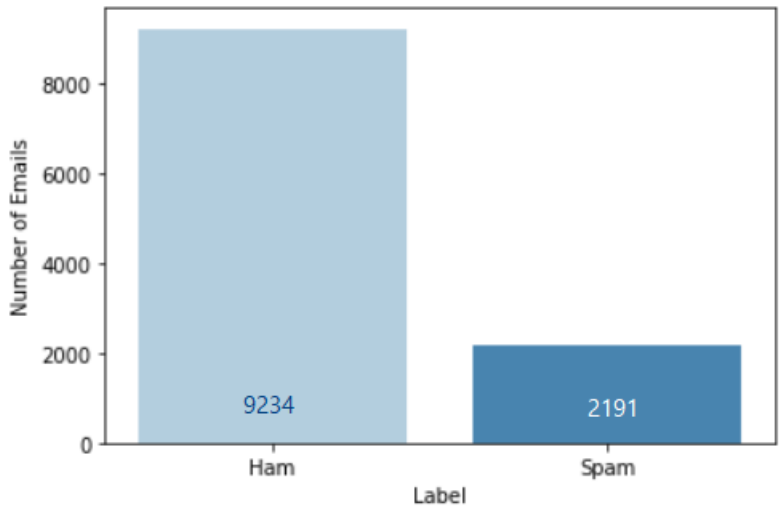
In the recent period, the percentage of scam messages amongst spam have increased sharply. Scam messages typically trick people into giving away money or personal details by offering an attractive or false deal. There are various traditional methods to prevent spam messages such as blacklisting certain senders or IP addresses but none of them are as impactful as recent techniques which uses aspect of machine learning and artificial intelligence.

In this assignment we created a spam classifier that reads the mails and classify them as Ham (Label 0) or Spam (Label 1). This classifier is an example of binary classifier and is implemented using Naive Bayes algorithm.

Dataset:
We downloaded the dataset from Kaggle. The dataset has a collection of 11425 mails labelled as either 0 (ham) or 1 (spam). The dataset contains around 81.82 % ham and around 19.18 % spam mails . The dataset is present in the assignment folder by the name "*train_dataset.csv*". The dataset looks like as,

1	Email	Label
2	Subject: re : summer work . . jinbaek , this is a project related to automated trading platform	0
3	Are you this much buzy	0
4	Headin towards busetop	0
5	Subject: freese notis mike : we are currently being billed for freese notis weather . i will nee	0
6	Subject: re : tanya ' s vacation shirley , i am going to take 5 vacation days in march , 3 / 13 / (	0
7	Subject: re : a personal favor thanks very much . i am attaching his resume for your review .	0
8	Subject: re : action learning project information vince , thanks for the information . kathy a	0
9	Jolly good! By the way, will give u tickets for sat eve 7.30. Speak before then x	0
10	Hey i booked the kb on sat already... what other lessons are we going for ah? Keep your sat r	0
11	Pls give her the food preferably pap very slowly with loads of sugar. You can take up to an ho	0
12	Ok...	0
13	Subject: component var tanya , some stability tests were performed on the simulation . (1)	0
14	Subject: forecasting project per our monday meeting , the time line for completion of the " p	0
15	Subject: re : informal interview with the enron research group fyi : i have arranged the follov	0
16	Hello beautiful r u ok? I've kinda ad a row wiv and he walked out the pub?? I wanted a night v	0
17	Sir, I have been late in paying rent for the past few months and had to pay a $ <#> char	0
18	Yo im right by yo work	0
19	Subject: zingales seminar fyi ! ---------- forwarded by shirley crenshaw ,	0
20	Sorry, I'll call later	0

The distribution of spam and ham mails in above dataset is as,



Methodology:
Our methodology for building classifier is as,

1. Data Pre-processing
2. Model Training
3. Model Evaluation

Data Pre-processing:
To prepare dataset for training we need to preprocess the raw dataset and convert it to a format that is suitable for training model. For e.g. let us consider a raw test mail as,

```
mail = "\\textbf{ From: raman@gmail.com } Subject: Re PRML Assigment 3 Can you please check out the naive bayes classifier sir taught in lectures 34 - 35. Also if you need slides of those lectures it is uploaded on Google Drive !"
```

1.) **Tokenization:**
Here for every mail in dataset we will convert the whole paragraphic content of that mail into a list of words (token). It is done because these tokens are the features on which we will train our model.

To do this we used *RegexpTokenizer* from NLTK. After applying tokenization on above example we will get output as,

```
[ '\\textbf{', 'From', ':', 'raman', '@gmail.com', '}', 'Subject', ':', 'Re', 'PRML', 'Assigment', '3', 'Can', 'you', 'please', 'check', 'out', 'the', 'naive', 'bayes', 'classifier', 'sir', 'taught', 'in', 'lectures', '34', '-', '35', '.', 'Also', 'if', 'you', 'need', 'slides', 'of', 'those', 'lectures', 'it', 'is', 'uploaded', 'on', 'Google', 'Drive', '!' ]
```

2.) **Removing Mail IDs:**
In this step for every token in list of tokens generated in above step we will remove that token if it is an email ID. It is done because email IDs does not given any indication whether a mail is spam or not, therefore email IDs are redundant and only add increases the size of token list.

This can be achieved by removing the token that has both @ and . characters present. After removing mail IDs from above example we will get output as,

```
[ '\\textbf{', 'From', ':', 'raman', '}', 'Subject', ':', 'Re', 'PRML', 'Assigment', '3', 'Can', 'you', 'please', 'check', 'out', 'the', 'naive', 'bayes', 'classifier', 'sir', 'taught', 'in', 'lectures', '34', '-', '35', '.', 'Also', 'if', 'you', 'need', 'slides', 'of', 'those', 'lectures', 'it', 'is', 'uploaded', 'on', 'Google', 'Drive', '!' ]
```

3.) **Removing Format Words:**
In this step for every token in list of tokens generated in tokenization step we will remove that token if it is some sort of a format word such as $\backslash pard$, $\backslash red255$ etc. It

is done because these format words does not add any meaning to the content and are just for beautification purposes.

For this purpose we have manually created a list of characters that marks the beginning of format words such as $\backslash, \{, \}$ etc. and removed those words that starts with such characters. After removing format words from above example we will get output as,

```
[ 'From', 'raman', 'Subject', 'Re', 'PRML', 'Assigment', '3', 'Can', 'you', 'please', 'check', 'out', 'the',
'naive', 'bayes', 'classifier', 'sir', 'taught', 'in', 'lectures', '34', '-', '35', 'Also', 'if', 'you', 'need', 'slides',
'of', 'those', 'lectures', 'it', 'is', 'uploaded', 'on', 'Google', 'Drive', '!' ]
```

4.) **Removing Numbers and Punctutations:**
In this step for every token in list of tokens generated in tokenization step we will remove that token if it is some sort of a number or a punctuation word such as $\#, +, \$$ etc.

It is done because numbers and punctuations alone does not give any hint about the mail being spam or not. Numbers along with some text may give hint e.g. Lucky draw of 5 million dollars might suggest email is a spam but since we tokenized the content and Naive Bayes works on word by words basis, the single token 5 is redundant and does not helps us in classification, therefore we removed numbers and punctuations.

For this purpose we simply checked each word that if it is either a number or punctuation or not and removed those that fell into any one of these two categories. After removing numbers and punctuations from above example we will get output as,

```
[ 'From', 'raman', 'Subject', 'Re', 'PRML', 'Assigment', '', 'Can', 'you', 'please', 'check', 'out', 'the', 'naive',
'bayes', 'classifier', 'sir', 'taught', 'in', 'lectures', '', '', '', 'Also', 'if', 'you', 'need', 'slides', 'of', 'those', 'lectures',
 'it', 'is', 'uploaded', 'on', 'Google', 'Drive', '' ]
```

5.) **Removing Stop Words:**
Stop words refers to the most common words in a language. For e.g. In English language stop words are $a, an, the, them$ etc. In this step for every token in list of tokens generated in tokenization step we will remove that token if it is a stop word.

It is done because stop words are common and present in both spam as well as non spam messages and thus does not helps in distinguishing them, therefore we

removed such words.

Additionally in this step we will also remove empty words (") and convert every word to lowercase.

For this purpose we have used the list of stop words in english language provided by $NLTK$ as well as added some by ourselves such as $could, might, would, may$ etc and for each word (token) in the list of tokens we removed those words that fell in the list of stop words. After removing words from above example we will get output as,

```
[ 'from', 'raman', 'subject', 'prml', 'assigment', 'can', 'please', 'check', 'naive', 'bayes', 'classifier', 'sir', 'taught',
'lectures', 'also', 'need', 'slides', 'lectures', 'uploaded', 'google', 'drive' ]
```

6.) **Lemmatizing:**
Lemmatizing is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form. For e.g. reader becomes read. In this step for every token in list of tokens generated in tokenization step we will lemmatize that word.

It is done because it might be the case that our training spam mail has word $jackpots$ and test mail might have word $jackpot$. Now despite same convenying same meaning our model will say that it has no idea about the test mail as our traning data does not contain the exact word $jackpots$ therefore it will reduce the accuracy, therefore we have used lemmatization. However lemmatization is harder to implement and slower compared to stemming but since our training dataset is not so large we implemented it.

For lemmatizing we used $WordNetLemmatizer$ from NLTK. After lemmatization we will get output as,

```
[ 'from', 'raman', 'subject', 'prml', 'assigment', 'can', 'please', 'check', 'naive', 'bayes', 'classifier', 'sir', 'taught',
'lecture', 'also', 'need', 'slide', 'lecture', 'uploaded', 'google', 'drive']
```

7.) **Removing Duplicate Words:**
In this step for every token in list of tokens generated in tokenization step if that token appears more than once then only one instance will be kept and remaining instances will be removed.

Removing duplicates is done because multiplying the probability of duplicate word

($P(DuplicateWord/Spam)$ and $P(DuplicateWord/Ham)$) with $P(Words/Spam)$ and $P(Words/Ham)$ respectively will only make the latter smaller which creates problems due to truncation. Removing duplicate words will also makes list of tokens smaller and thus faster to compute.

For this purpose we converted the list to tokens into a set (so all duplicates are removed) and then reconvert the set into the list. After removing duplicate words we will get output as,

```
[ 'lecture', 'can', 'raman', 'need', 'prml', 'check', 'subject', 'classifier', 'sir', 'from', 'bayes', 'taught', 'naive', 'assigment',
 'also', 'uploaded', 'please', 'google', 'slide', 'drive' ]
```

Here all the steps are carried out after one another and in a carefully selected sequence which complements the next step. After pre-processing the tokens in the list are the pre-processed features on which our model will be trained. After pre-processing the above raw mail is converted into datapoint with features as,

```
[ 'lecture', 'can', 'raman', 'need', 'prml', 'check', 'subject', 'classifier', 'sir', 'from', 'bayes', 'taught', 'naive', 'assigment',
 'also', 'uploaded', 'please', 'google', 'slide', 'drive' ]
```

Model Training:
We will use Naive Bayes algorithm to train our model. After pre-processing the dataset we now have a list of datapoints (mails) and each datapoint itself is a list of features (words). One thing to note is that each datapoint has different number of words therefore the number of features/dimensions for each datapoint is different.

Now we construct a dictionary data structure, named *dictionary* whose key is a word and value is a list containing two elements. The first element (index 0 element) indicates the frequency or the number of ham mails which contains the given word and the second element (index 1 element) indicates the frequency or the number of spam mails which containes the given word. For e.g.

$$dictionary['lottery'][0] = 10 \qquad dictionary['lottery'][1] = 35$$

tells there are 10 ham mails and 35 spam mails respectively which contains the word *lottery*. The dictionary generated is as,

necessary	132	50
whether	211	18
ieor	44	1
got	414	54
taking	215	31
fascinating	8	1
art	39	18
term	339	87
berkeley	75	2
project	543	50

After the frequency dictionary is generated we perform **Laplacian Smoothing** by incrementing the count of every word in dictionary for both ham and spam mails. This is analogous to adding one dummy ham and one dummy spam mail to the dataset. We also increase the count of total ham and spam mails by one. Smoothing guarantees that the there is at least one mail in ham and spam mails which contains the given word so that proabability for that word given ham or spam is non-zero which in turn guarantees probability of mail being spam or ham is non zero and thus decidable.

Now at the last stage of training we generate a probability table named *probability_table* which is similar to *dictionary* but unlike *dictionary* it tells the probability of a word given ham or spam. For e.g.

$$probability\_table['lottery'][0] = 0.0015 \quad probability\_table['lottery'][1] = 0.0459$$

tells the probability of *lottery* given ham, $P(\text{lottery}/Ham)$ is 0.0015 and probability of *lottery* given spam, $P(\text{lottery}/Spam)$ is 0.0459. The probability table is generated using dictionary as,

$$probability\_table[word][0] = \frac{dictionary[word][0]}{Number\ of\ Ham\ Mails}$$

$$probability\_table[word][1] = \frac{dictionary[word][1]}{Number\ of\ Spam\ Mails}$$

The probability table generated is as,

necessary	0.01429344883595019	0.02281021897810219
whether	0.02284786139685977	0.008211678832116789
ieor	0.00476448294531673	0.0004562043795620438
got	0.04482945316729832	0.024635036496350366
taking	0.023280996210070383	0.014142335766423358
fascinating	0.0008662696264212236	0.0004562043795620438
art	0.004223064428803465	0.008211678832116789
term	0.03670817541959935	0.03968978102189781
berkeley	0.008121277747698972	0.0009124087591240876
project	0.05879805089334055	0.02281021897810219

Model Evaluation:
Before discussing the evaluation of our model let's see how our model classifies the emails. It uses Naive Bayes algorithm to calculate probabilities of test mail being spam and ham and assigns the label that has higher probability. Here the testing mails are pre-processed in the same way as training mails.

The formula to calculate probability of spam and ham given mail (bunch of words) are as,

$$P(Spam/Words) = \frac{P(Spam) \cdot P(Words/Spam)}{P(Words)}$$

$$P(Ham/Words) = \frac{P(Ham) \cdot P(Words/Ham)}{P(Words)}$$

Since any mail is equally likely to be a spam or ham therefore we took,

$$P(Ham) = P(Spam) = 0.5$$

In Naive Bayes algorithm words ocurring in mails are assumed to be independent therefore $P(Words/Spam)$ can be written as,

$$P(Words/Spam) = P(Word_1/Spam) \dots P(Word_k/Spam)$$

and as we saw earlier that,

$$P(Word_k/Spam) = probability\_table[word_k][1]$$

I have shown above equations using spam but same also applies for ham mails also.

Note:
Because probabilities lie between 0 and 1 therefore multiplying probability of words will make $P(Words/Spam)$ lesser and lesser and due to truncation rules sometimes the last digits are truncated and we get $P(Words/Spam) = 0$. Same goes with $P(Words/Ham)$ too.

This affects the accuracy of the classifier when both becomes 0 and we have to break tie arbitrary. To deal with this we instead have $P(Words/Spam)$ and $P(Words/Ham)$ as,

$$P(Words/Ham) = 10^{175} \cdot P(Word_1/Spam) \dots P(Word_k/Spam)$$
$$P(Words/Spam) = 10^{175} \cdot P(Word_1/Spam) \dots P(Word_k/Spam)$$

After $P(Spam/Words)$ and $P(Ham/Words)$ for a mail is calculated we see which among these is greater and assigns label to the mail corresponding to the greater

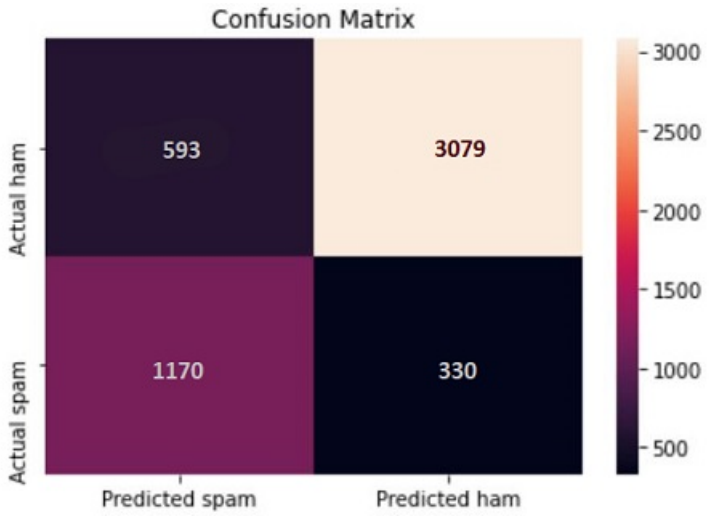
probability.

Now that we know how our model is assiging labels to mails we will see how our model is evaulated. We evaluated our model on several datasets. The analysis of the evaluations for each of the dataset is as,

1.) Test Dataset 1
This is a different dataset from ours that we downloaded from Kaggle. It has 5172 mails in total out of which 1500 mails are spam and remaining 3672 mails are ham. The dataset looks as,

1	Email No.	the	to	ect	and	for	of	a	you	hou	in	on	is
2	Email 1	0	0	1	0	0	0	2	0	0	0	0	1
3	Email 2	8	13	24	6	6	2	102	1	27	18	21	13
4	Email 3	0	0	1	0	0	0	8	0	0	4	2	0
5	Email 4	0	5	22	0	5	1	51	2	10	1	5	9
6	Email 5	7	6	17	1	5	2	57	0	9	3	12	2
7	Email 6	4	5	1	4	2	3	45	1	0	16	12	8
8	Email 7	5	3	1	3	2	1	37	0	0	9	4	6
9	Email 8	0	2	2	3	1	2	21	6	0	2	6	2
10	Email 9	2	2	3	0	0	1	18	0	0	3	3	2
11	Email 10	4	4	35	0	1	0	49	1	16	9	4	1
12	Email 11	22	14	2	9	2	2	104	0	2	35	13	21
13	Email 12	33	28	27	11	10	12	173	6	12	28	47	27
14	Email 13	27	17	3	7	5	8	106	3	0	22	33	16
15	Email 14	4	5	7	1	5	1	37	1	3	8	8	6
16	Email 15	2	4	6	0	3	1	16	0	3	6	4	1
17	Email 16	6	2	1	0	2	0	36	3	1	8	4	6
18	Email 17	3	1	2	2	0	1	17	0	0	1	1	0
19	Email 18	36	21	6	14	7	17	194	25	5	59	37	16
20	Email 19	1	3	1	0	2	0	14	0	0	1	1	5
21	Email 20	3	4	11	0	4	2	32	1	5	1	3	9

The confusion matrix for this test dataset is as,



Overall Accuracy : 82.15%

Accuracy of classifying Ham : 83.85%

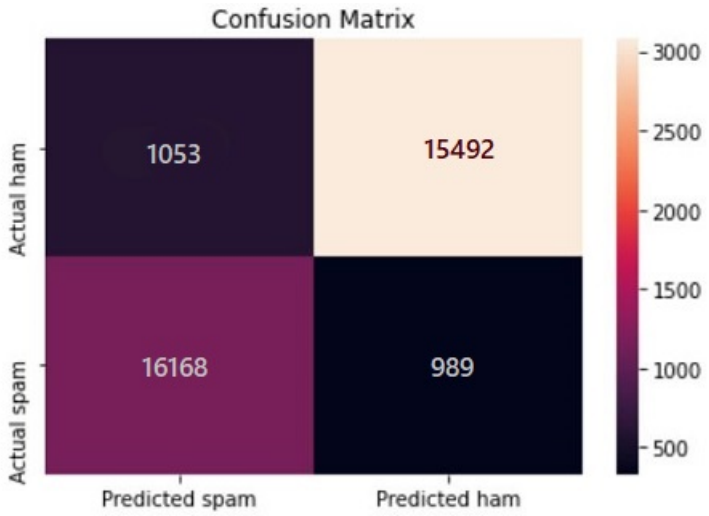
Accuracy of classifying Spam : 78.00%

2.) Test Dataset 2

This is a different dataset from ours that we downloaded from Kaggle. It has 33702 mails in total out of which 17157 mails are spam and remaining 16545 mails are ham. The dataset looks as,

1	Email	Label
2	Subject: unfaithful wives 0	1
3	Subject: business joint venture	1
4	Subject: re :	0
5	Subject: bank inheritance	1
6	Subject: fw : real estate issues re : ubsw energy , llc in houston	0
7	Subject: urgent news alert ! (otcbb : gspm) gold is hot !	1
8	Subject: national lottery headquarters :	1
9	Subject: fw :	0
10	Subject: re : rahil	0
11		1
12	Subject:	1
13	Subject: top quality medicine here	1
14	Subject: off duty days	0
15	Subject: attn : security update from citibank msgid # 92379245	1
16	Subject: re : certified original birth certificate / uk birth registry &	0
17	Subject: re : tanya vacation	0
18	Subject: want your medication ? we have it	1
19	Subject: start date : 12 / 19 / 01 ; hourahead hour : 24 ;	0
20	Subject: excellent values on necessary software	1

The confusion matrix for this test dataset is as,



Overall Accuracy : 93.94%

Accuracy of classifying Ham : 93.64%

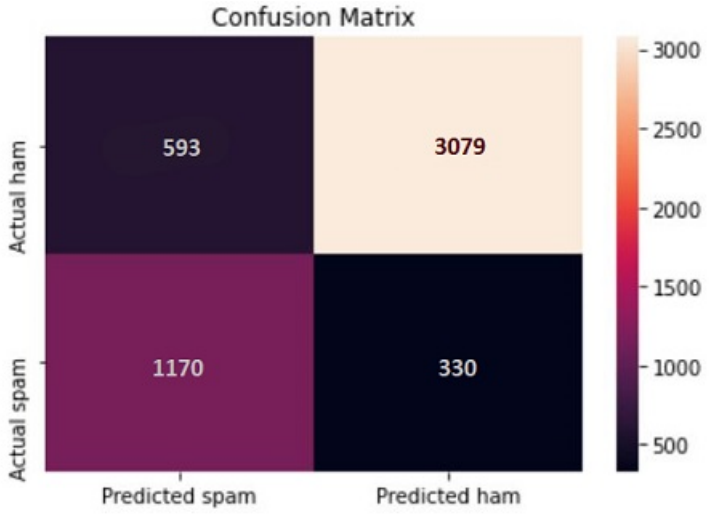
Accuracy of classifying Spam : 94.24%

3.) Test Dataset 3

We created this dataset from our personal email accounts. Here spam mails are picked from spam folder and ham mails are the mails we sent to friends or teachers.

This dataset also contains the two test mails that were provided to us with this assignment. It has 12 mails in total out of which 6 mails are spam and remaining 6 mails are ham. The dataset looks as,

The confusion matrix for this test dataset is as,



Overall Accuracy : 75.00%

Accuracy of classifying Ham : 83.33%

Accuracy of classifying Spam : 66.67%

Conclusion:

We tried different datasets for training the model but the highest average accuracy was achieved by this dataset. Pre-processing has noticeable effect on the model and significantly improved the speed as well as precision of the model. Along with training data we will pre-process test data so that it can be converted into a format suitable for assigning label. Overall Naive Bayes is a very simple and fundamental algorithm with very good practical accuracy.

Executing Code:
We have implemented a function in our code named $classify$ which when called will automatically read all the mails from the *test* folder present in the current directory. $classify$ does not accept any parameters and have to invoked as,

$$classify()$$

Please make sure to also put the file titled *training_dataset.csv* in the same directory where the source code is kept. We require this dataset to train our model. It will take around 10 mins to train the model.

After calling $classify$ it will automatically start labelling the mails and finally will output a *output.csv* file in current directory file which has two columns *Email* and *Label*. *Email* column will contain the name of email.txt file in the test folder and corresponding value in *Label* column will be the label assigned to that mail.txt file. The *output.csv* looks as,

Email	Label
email4	1
email9	1
email3	0
email11	0
email12	0
email5	0
email6	0
email2	1
email7	0
email10	1

We wrote this code in *Google Colab* and it needs to import certain libraries and download corpus words such as *stop words*. So please try to run this code in *Google Colab* so that it can automatically download and import libraries ensuring the smooth and expected functioning.