TECHNO MAIN SALT LAKE, KOLKATA

MACHINE LEARNING WITH PYTHON

Email Spam Filtering

TRAINING PROJECT REPORT

Submitted by:

BHUMIKA JINDAL

13000117112

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE & ENGINEERING

JUNE-JULY 2020

A REPORT OF THREE WEEKS INDUSTRIAL TRAINING AT WebTek Labs Pvt. Ltd.

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT
FOR THE AWARD OF THE DEGREE OF
BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE & ENGINEERING

CANDIDATE'S DECLARATION

I hereby declare that I have undertaken industrial training at "**WEBTEK LABS PVT. LTD.**" during a period from **22nd June to 14th July** in partial fulfillment of requirements for the award of degree of **Bachelor of Technology in Computer Science and Engineering at Techno Main Salt Lake**. The work which is being presented in the training report submitted to Department of COMPUTER SCIENCE & ENGINEERING at TECHNO MAIN SALT LAKE, KOLKATA is an authentic record of training work.

BHUMIKA JINDAL _____

Student Name *Signature of the Student*

ACKNOWLEDGEMENT

It gives me great pleasure to acknowledge the guidance, assistance and support of Ms. Mousita Dhar in making the Project and this Project report successful, which has been structured under her valued suggestion. She not only taught me about machine learning but also gave idea about its practical applications.

She gave me the golden opportunity to do this wonderful project on the topic ***Email Spam Filtering*** which also helped me in doing a lot of Research and to know about so many new things. She has helped me to accomplish the challenging task in a very short period of time.

I would also like to thank my college teachers who encouraged me to do the training. Finally, I express the constant support of my friends, family and professors for inspiring me throughout and encouraging me.

BHUMIKA JINDAL
SEMESTER: VII
C.S.E

CERTIFICATE OF APPROVAL

The project **"EMAIL SPAM FILTERING"** made by the efforts of **BHUMIKA JINDAL** is hereby approved as a creditable study for the **Bachelor of Technology** in **Computer Science & Engineering** and presented in a manner of satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned this project only for the purpose for which it is submitted.

Ms. Mousita Dhar

(Project in charge)

ABSTRACT

The upsurge in the volume of unwanted emails called spam has created an intense need for the development of more dependable and robust antispam filters. Machine learning methods of recent are being used to successfully detect and filter spam emails. We present a systematic review of some of the popular machine learning based email spam filtering approaches. Our review covers survey of the important concepts, attempts, efficiency, and the research trend in spam filtering. The preliminary discussion in the study background examines the applications of machine learning techniques to the email spam filtering process of the leading internet service providers (ISPs) like Gmail, Yahoo and Outlook emails spam filters. Discussion on general email spam filtering process, and the various efforts by different researchers in combating spam through the use machine learning techniques was done. Our review compares the strengths and drawbacks of existing machine learning approaches and the open research problems in spam filtering. We recommended deep leaning and deep adversarial learning as the future techniques that can effectively handle the menace of spam emails.

INTRODUCTION

Recently unsolicited commercial / bulk e-mail also known as spam, become a big trouble over the internet. Spam is waste of time, storage space and communication bandwidth. The problem of spam e-mail has been increasing for years. In recent statistics, 40% of all emails are spam which about 15.4 billion email per day and that cost internet users about $355 million per year. Automatic e-mail filtering seems to be the most effective method for countering spam at the moment and a tight competition between spammers and spam-filtering methods is going on.

Only several years ago most of the spam could be reliably dealt with by blocking e-mails coming from certain addresses or filtering out messages with certain subject lines. Spammers began to use several tricky methods to overcome the filtering methods like using random sender addresses and/or append random characters to the beginning or the end of the message subject line. Knowledge engineering and machine learning are the two general approaches used in e-mail filtering. In knowledge engineering approach a set of rules has to be specified according to which emails are categorized as spam or ham. A set of such rules should be created either by the user of the filter, or by some other authority (e.g. the software company that provides a particular rule-based spam-filtering tool). By applying this method, no promising results shows because the rules must be constantly updated and maintained, which is a waste of time and it is not convenient for most users.

Machine learning approach is more efficient than knowledge engineering approach; it does not require specifying any rules. Instead, a set of training samples, these samples is a set of pre classified e-mail messages. A specific algorithm is then used to learn the classification rules from these e-mail messages. Machine learning approach has been widely studied and there are lots of algorithms can be used in e-mail filtering. They include Naïve Bayes, support vector machines, Neural Networks, K-nearest neighbor, Rough sets and the artificial immune system.

INTRODUCTION TO PYTHON

PYTHON

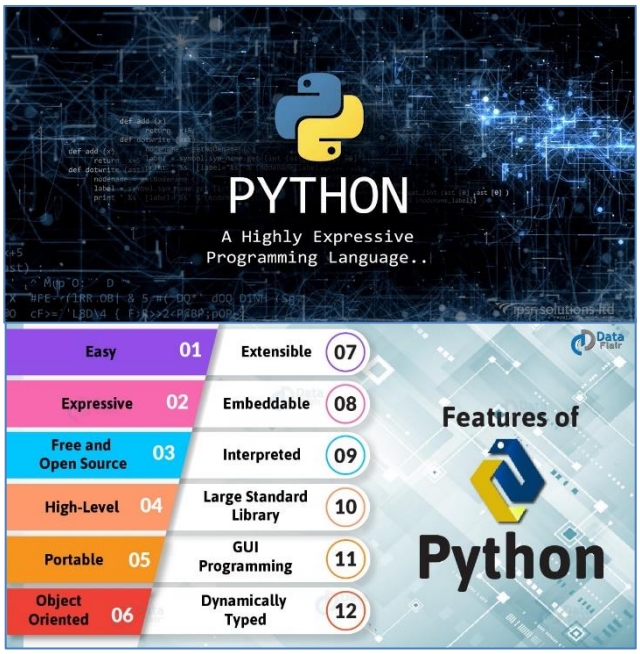
- Python is a high-level, general-purpose, open source, strictly typed programming language. The language provides constructs intended to enable clear programs on both a small and large scale.
- Python was created By Guido van Rossum.
- It lets us work more quickly and integrate your systems more effectively. We can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.
- The Python Software Foundation (PSF) is the organization behind Python.
- Python is used in many domains like Web Development, Data Analysis, Machine Learning, Internet of Things, GUI Development, Image processing, Data visualization, Game Development and so on.

Python Versions:

- It was first released in 1991
- Python 2.0 was released on 16th October 2000.
- Python 3.0 was released on 3rd December 2008.
- Python 3.6 is the current Version.

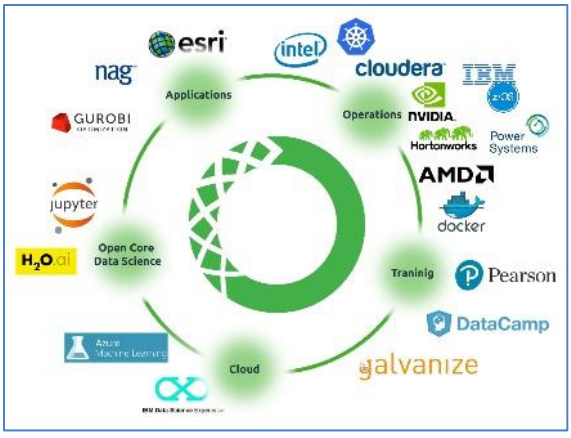
Python features:

- It is a dynamic, high level, free open source language.
- It is an Object-Oriented Language.
- It has GUI Programming support.
- It is a portable language.
- It is a Portable language.
- It is a Dynamically Typed Language.



ANNACONDA

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition.



PYTHON PACKAGES USED

NumPy:

NumPy is a fundamental package for scientific computing in python. Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

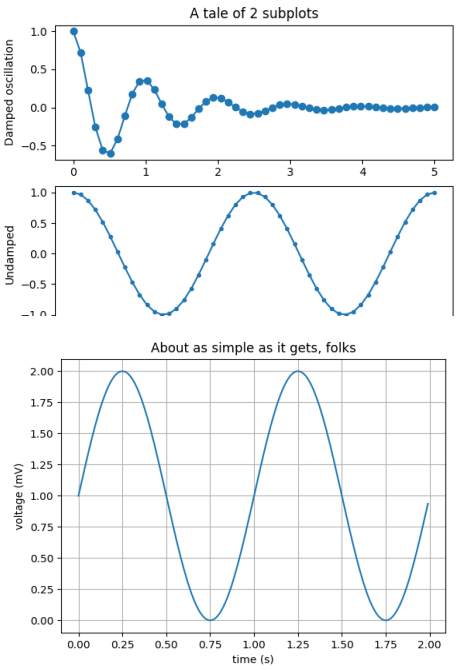
Pandas:

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Pandas library is well suited for data manipulation and analysis using python. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Matplotlib:

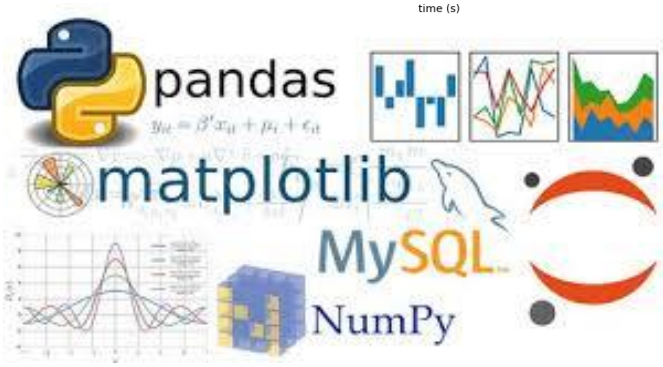
Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc. via an object-oriented interface or via a set of functions familiar to MATLAB users.



Seaborn:

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures.



TRAINING WORK UNDERTAKEN

Problem Description

The problem is to build an **Email Spam Filtering** model, which can be used to reduce the unnecessary emails in the inbox. Given a set of randomly picked email files and their respective labels for spam or not-spam classification. The objective is to build a model that is able to classify the email as spam or not spam based on the words used in the email.

Different Techniques of Spam Filtering

The increasing volume of unsolicited bulk e-mail (also known as spam) has generated a need for reliable anti-spam filters. Machine learning techniques now days used to automatically filter the spam e-mail in a very successful rate. In this project we have used some of the most popular machine learning methods for spam Email classification. Descriptions of the algorithms are presented, and the comparison of their performance on the Spam filtration is presented. There are different categories of spam filtering techniques that have been widely applied to overcome the problem of email spam which includes **Content Based Filtering Technique, Case Base Spam Filtering Method, Heuristic or Rule Based Spam Filtering Technique, Previous Likeness Based Spam Filtering Technique, Adaptive Spam Filtering Technique**. Content based filtering is usually used to create automatic filtering rules and to classify emails using machine learning approaches, such as Naïve Bayesian classification, Support Vector Machine, K Nearest Neighbor, Neural Networks.

Collecting Data from Kaggle:

Kaggle is a platform for predictive modelling and analytics competitions in which statisticians and data miners compete to produce the best models for predicting and describing the datasets uploaded by companies and users. This crowd sourcing approach relies on the fact that there are countless strategies that can be applied to any predictive modelling task and it is impossible to know beforehand which technique or analyst will be most effective. On 8 March 2017, Google announced that they were acquiring Kaggle. They will join the Google Cloud team and continue to be a distinct brand. In January 2018, Booz Allen and Kaggle launched Data Science Bowl, a machine learning competition to analyze cell images and identify nuclei.

About the Dataset:

This is a csv file containing related information of 5172 randomly picked email files and their respective labels for spam or not-spam classification. The csv file contains 5172 rows, each row for each email. There are 3002 columns. The first column indicates Email name. The name has been set with numbers and not recipients' name to protect privacy. The last column has the labels for prediction: 1 for spam, 0 for not spam. The remaining 3000 columns are the 3000 most common words in all the emails, after excluding the non-alphabetical characters/words. For each row, the count of each word(column) in that email(row) is stored in the respective cells. Thus, information regarding all 5172 emails are stored in a compact data frame rather than as separate text files.

SOURCE CODE AND OUTPUT

Importing the required Packages:

```
#Importing all the required packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

Loading the Dataset:

```
#Reading the dataset containing the frequency of words in each mail
email = pd.read_csv("C:\\Users\\Bhumika Jindal\\FSP_ML_7th Sem\\emails.csv")
email.head(20)
```

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastructure	military	allowing	ff	dry	Prediction
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	0
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	0	0	0	1	0	0
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	0	0	0	0	0	0
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	0	0	0	0	0	0
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0	0	0	1	0	0
5	Email 6	4	5	1	4	2	3	45	1	0	...	0	0	0	0	0	0	0	0	0	1
6	Email 7	5	3	1	3	2	1	37	0	0	...	0	0	0	0	0	0	0	0	0	0
7	Email 8	0	2	2	3	1	2	21	6	0	...	0	0	0	0	0	0	0	1	0	1
8	Email 9	2	2	3	0	0	1	18	0	0	...	0	0	0	0	0	0	0	0	0	0
9	Email 10	4	4	35	0	1	0	49	1	16	...	0	0	0	0	0	0	0	0	0	0
10	Email 11	22	14	2	9	2	2	104	0	2	...	0	0	0	0	0	0	0	1	0	0
11	Email 12	33	28	27	11	10	12	173	6	12	...	0	0	0	0	0	0	0	5	0	0
12	Email 13	27	17	3	7	5	8	106	3	0	...	0	0	0	0	0	0	0	4	0	0
13	Email 14	4	5	7	1	5	1	37	1	3	...	0	0	0	0	0	0	0	2	0	0
14	Email 15	2	4	6	0	3	1	16	0	3	...	0	0	0	0	0	0	0	1	0	0
15	Email 16	6	2	1	0	2	0	36	3	1	...	0	0	0	0	0	0	0	0	0	0
16	Email 17	3	1	2	2	0	1	17	0	0	...	0	0	0	0	0	0	0	1	0	1
17	Email 18	36	21	6	14	7	17	194	25	5	...	0	0	0	0	0	0	0	3	0	1
18	Email 19	1	3	1	0	2	0	14	0	0	...	0	0	0	0	0	0	0	0	0	0
19	Email 20	3	4	11	0	4	2	32	1	5	...	0	0	0	0	0	0	0	1	0	0

20 rows × 3002 columns

Data Preprocessing:

➢ Checking for null values in the data

```
#Checking for null values
email.isnull().sum()
```

```
Email No.     0
the           0
to            0
ect           0
and           0
             ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3002, dtype: int64
```

➢ Observing the description of every column

```
#Observing the description of every column
email.describe()
```

	the	to	ect	and	for	of	a	you	hou	in	...	connevey	51
count	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	...	5172.000000	
mean	6.640565	6.188128	5.143852	3.075599	3.124710	2.627030	55.517401	2.466551	2.024362	10.600155	...	0.005027	
std	11.745009	9.534576	14.101142	6.045970	4.680522	6.229845	87.574172	4.314444	6.967878	19.281892	...	0.105788	
min	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
25%	0.000000	1.000000	1.000000	0.000000	1.000000	0.000000	12.000000	0.000000	0.000000	1.000000	...	0.000000	
50%	3.000000	3.000000	1.000000	1.000000	2.000000	1.000000	28.000000	1.000000	0.000000	5.000000	...	0.000000	
75%	8.000000	7.000000	4.000000	3.000000	4.000000	2.000000	62.250000	3.000000	1.000000	12.000000	...	0.000000	
max	210.000000	132.000000	344.000000	89.000000	47.000000	77.000000	1898.000000	70.000000	167.000000	223.000000	...	4.000000	

8 rows × 3001 columns

➢ Checking the correlation among various features

```
#Checking the correlation among various features
email.corr()
```

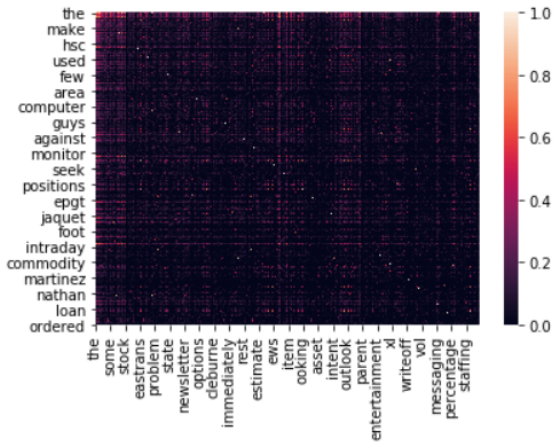
	the	to	ect	and	for	of	a	you	hou	in	...	connevey	jay	valued	lay
the	1.000000	0.852715	0.337249	0.841200	0.784112	0.796397	0.784451	0.471392	0.303621	0.845670	...	0.008926	0.075479	0.225586	0.223426
to	0.852715	1.000000	0.375480	0.825474	0.781971	0.752722	0.896466	0.508513	0.347993	0.881759	...	0.013250	0.101247	0.232847	0.255793
ect	0.337249	0.375480	1.000000	0.272863	0.369777	0.178028	0.400009	0.155783	0.974152	0.298387	...	0.134339	0.031431	0.046080	0.061550
and	0.841200	0.825474	0.272863	1.000000	0.751287	0.809665	0.815196	0.476764	0.235953	0.874276	...	0.005151	0.104454	0.272963	0.253440
for	0.784112	0.781971	0.369777	0.751287	1.000000	0.681457	0.744098	0.495852	0.329051	0.762659	...	0.022168	0.041775	0.236213	0.213631
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
military	0.129466	0.091639	-0.007690	0.084147	0.067151	0.073004	0.111685	0.006498	0.005429	0.120620	...	-0.002249	-0.002979	0.043408	0.104297
allowing	0.127019	0.120059	0.004368	0.124766	0.121057	0.108786	0.105358	0.082757	-0.000966	0.138099	...	-0.002675	-0.003543	-0.005130	0.018550
ff	0.341878	0.406666	0.141460	0.400225	0.301074	0.444252	0.464473	0.195058	0.114210	0.448303	...	0.005403	0.073690	0.130356	0.164296
dry	0.051021	0.071388	0.002492	0.042484	0.038126	0.026403	0.093822	0.028883	0.000601	0.077751	...	-0.003373	0.035028	-0.006468	0.018939
Prediction	-0.004421	0.055277	-0.120782	0.114364	-0.003101	0.197234	0.107776	0.130293	-0.128340	0.154055	...	-0.030375	-0.031694	0.098775	0.064315

3001 rows × 3001 columns

- **Visual representation of correlation matrix using a heat map**

```
#Visual Representation of correlation matrix of all columns
#plt.figure(figsize=(20,20))  #size used to increase the size
ax=sns.heatmap(email.corr().abs())
bottom,top=ax.get_ylim()
ax.set_ylim(bottom+0.5,top-0.5)
```

```
(2861.0, 0.0)
```



Selecting the features and label through data slicing

```
#Finding the features and label for the model
x = email.iloc[:,1:3001]
y = email.iloc[:,-1].values
```

Splitting of x and y into train and test data

```
#Splitting the training set and the test set
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.15,random_state=0)
```

Various Algorithms Used for Classification

We have used different machine learning algorithms for building the email spam filtering model. The description about the algorithm and the accuracy score of them are shown in the following pages. We have used the three algorithms as follows:

- **Naïve Bayes Classifier**
- **Support Vector Machine (SVM)**
- **Random Forest Classifier**

Naïve Bayes Classifier

A Naive Bayes classifier is a probabilistic machine learning model that is used for classification task. The crux of the classifier is based on the Bayes theorem.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayes Theorem:

Using Bayes theorem, we can find the probability of A happening, given that B has occurred. Here, B is the evidence and A is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

In this project we are **classifying mails** typed in by the user as either **'Spam' or 'Not Spam'**. Our original dataset was a folder of 5172 text files containing the emails. Now let us understand why we have separated the words from the mails. This is because, this is a **text-classification problem**. When a spam classifier looks at a mail, it searches for potential words that it has seen in the previous spam emails. If it finds a majority of those words, then it labels it as 'Spam'. Why did I say majority?

CASE 1: Suppose let's take a word 'Greetings'. Say, it is present in both 'Spam' and 'Not Spam' mails.
CASE 2: Let's consider a word 'lottery'. Say, it is present in only 'Spam' mails.
CASE 3: Let's consider a word 'cheap'. Say, it is present only in Spam.
If now we get a test email, and it contains all the three words mentioned above, there's high probability that it is a 'Spam' mail. The most effective algorithm for text-classification problems is the Naive Bayes algorithm, that works on the classic Bayes' theorem. This theorem works on every individual word in the test data to make predictions (the conditional probability with higher probability is the predicted result).

Say, our test email(S)is: "You have won a lottery"
How Naïve Bayes works on this Data:
P(S) = P('You') P('have') P('won') P('a') P('lottery') __ 1
Thus, P(S|Spam)= P('You'|Spam) P('have'|Spam) P('won'|Spam) P('a'|Spam) P('lottery'|Spam) …. (2)
Same calculation for P(S|Not_Spam) …. (3)
If (2) > (3), then 'Spam' Else, 'Not_Spam'.

WHAT IF THE PROBABILITY IS ZERO? Here comes the concept of Laplace Smoothing, where
P(words) = (word_count + 1)/(total_no_of_words + no_of_unique_words)

Here, we'll work on the existing **Gaussian Naive Bayes classifier** (under scikit-learn). To further understand how well Naive Bayes works for text-classification, we'll use another standard classifier, SVC, to see how the two models perform.

➢ Applying Naive Bayes algorithm for prediction

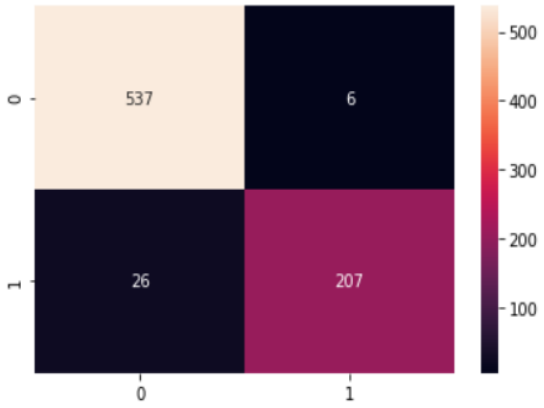
```
#Applying Gausian Naive Bayes Algorithm
gnb = GaussianNB()
gnb.fit(x_train,y_train)
y_pred_gnb = gnb.predict(x_test)

#Checking the accuracy score of train and test dataset
as_gnb = accuracy_score(y_pred_gnb,y_test)
ts_gnb = gnb.score(x_train,y_train)
print("\nNAIVE BAYES ALGORITHM\n----------------------")
print("Prediction Accuracy Score:",as_gnb)
print("Training Accuracy Score:",ts_gnb)
```

```
NAIVE BAYES ALGORITHM
---------------------
Prediction Accuracy Score: 0.9587628865979382
Training Accuracy Score: 0.9663330300272975
```

➢ Confusion matrix for Naive Bayes Algorithm

```
# Confusion Matrix for Gausian Naive Bayes Algorithm
cf_matrix_gnb=confusion_matrix(y_pred_gnb,y_test)
cm=pd.DataFrame(cf_matrix_gnb)
ax=sns.heatmap(cm,fmt="d",annot=True)
bottom,top=ax.get_ylim()
ax.set_ylim(bottom+0.5,top-0.5)
```

```
(2.0, 0.0)
```



➢ Splitting the training and test set with different size and Standardizing the features

```
#Splitting the training set and the test set with different test size
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.1,random_state=0)

#Scaling the values in the features columns within a range
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

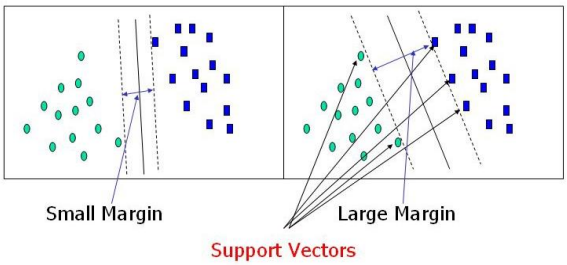
Support Vector Machine (SVM)

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text. Classifying data is a common task in machine learning. Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support-vector machines, a data point is viewed as a p-dimensional vector (a list of p numbers), and we want to know whether we can separate such points with a p-dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So, we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines is known as a maximum-margin classifier; or equivalently, the perceptron of optimal stability.

Tuning Parameters - Parameters are arguments that you pass when you create your classifier. Following are the important parameters for SVM:

1) **C -** It controls the tradeoff between smooth decision boundary and classifying training points correctly. A large value of c means you will get more training points correctly. Large value of c means you will get more intricate decision curves trying to fit in all the points. So, we try different values of c for our dataset to get the perfectly balanced curve and avoid over fitting.
2) **Gamma -** It defines how far the influence of a single training example reaches. If it has a low value it means that every point has a far reach and conversely high value of gamma means that every point has close reach. If gamma has a very high value, then the decision boundary is just going to be dependent upon the points that are very close to the line which effectively results in ignoring some of the points that are very far from the decision boundary.

Support Vector Machine is the most sought-after algorithm for classic classification problems. SVMs work on the algorithm of Maximal Margin, i.e., to find the maximum margin or threshold between the support vectors of the two classes (in binary classification).



Prediction using Support Vector Classifier Algorithm

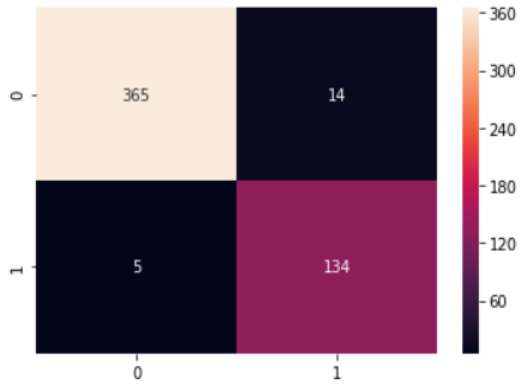
```
#Applying Support Vector Classification Algorithm
svc = SVC(C=0.001,kernel='linear') #C is the regularization parameter. As C increases, model overfits.
svc.fit(x_train,y_train)
y_pred_svc = svc.predict(x_test)

#Checking the accuracy score of train and test dataset
as_svc = accuracy_score(y_pred_svc,y_test)
ts_svc = svc.score(x_train,y_train)
print("\nSUPPORT VECTOR CLASSIFICATION ALGORITHM\n---------------------------------------")
print("Prediction Accuracy Score:",as_svc)
print("Training Accuracy Score:",ts_svc)
```

```
SUPPORT VECTOR CLASSIFICATION ALGORITHM
---------------------------------------
Prediction Accuracy Score: 0.9633204633204633
Training Accuracy Score: 0.9804469273743017
```

➢ Confusion Matrix for Support Vector Classifier

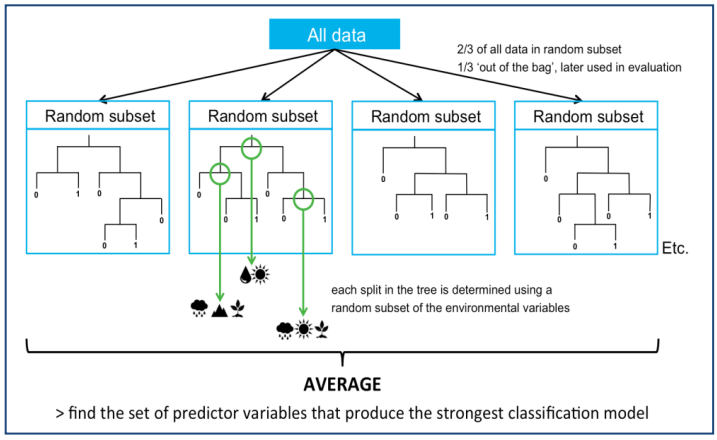
```
# Confusion Matrix for Support Vector Classifier
cf_matrix_svm=confusion_matrix(y_pred_svc,y_test)
cm=pd.DataFrame(cf_matrix_svm)
ax=sns.heatmap(cm,fmt="d",annot=True)
bottom,top=ax.get_ylim()
ax.set_ylim(bottom+0.5,top-0.5)
```

```
(2.0, 0.0)
```



➢ Random Forest Classifier

Random forests are a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests create decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.



Random forests have a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset. It technically is an ensemble method (based on the divide-and-conquer approach) of decision trees generated on a randomly split dataset. This collection of decision tree classifiers is also known as the forest. The individual decision trees are generated using an attribute selection indicator such as information gain, gain ratio, and Gini index for each attribute. Each tree depends on an independent random sample. In a classification problem, each tree votes and the most popular class is chosen as the final result. In the case of regression, the average of all the tree outputs is considered as the final result. It is simpler and more powerful compared to the other non-linear classification algorithms.

How does the algorithm work?

It works in four steps:
1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.

Advantages:

- Random forests are considered as a highly accurate and robust method because of the number of decision trees participating in the process.
- It does not suffer from the overfitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
- The algorithm can be used in both classification and regression problems.
- Random forests can also handle missing values. There are two ways to handle these: using median values to replace continuous variables, and computing the proximity-weighted average of missing values.
- We can get the relative feature importance, which helps in selecting the most contributing features for the classifier.

Disadvantages:

- Random forests are slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming.
- The model is difficult to interpret compared to a decision tree, where you can easily make a decision by following the path in the tree.

➢ Prediction using Random Forest Classifier Algorithm

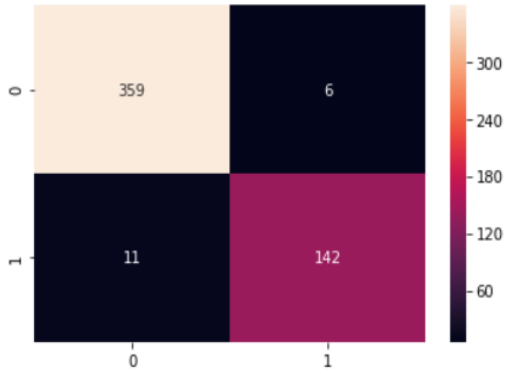
```
#Applying Random Forest Classification Algorithm
rfc = RandomForestClassifier(n_estimators=150,criterion='entropy',min_samples_split=.01,random_state=0)
# n_estimators = No. of trees in the forest
# criterion = basis of making the decision tree split, either on gini impurity('gini'), or on information gain('entropy')
# min_samples_split = Minimum number of samples required to split an internal node
rfc.fit(x_train,y_train)
y_pred_rfc = rfc.predict(x_test)

#Checking the accuracy score of train and test dataset
as_rfc = accuracy_score(y_pred_rfc,y_test)
ts_rfc = rfc.score(x_train,y_train)
print("\nRANDOM FOREST CLASSIFICATION ALGORITHM\n---------------------------------------")
print("Prediction Accuracy Score:",as_rfc)
print("Training Accuracy Score:",ts_rfc)
```

```
RANDOM FOREST CLASSIFICATION ALGORITHM
---------------------------------------
Prediction Accuracy Score: 0.9671814671814671
Training Accuracy Score: 0.9935539321014182
```

➢ Confusion Matrix for Random Forest Classifier

```
# Confusion Matrix for Random Forest Classifier
cf_matrix=confusion_matrix(y_pred_rfc,y_test)
cm=pd.DataFrame(cf_matrix)
ax=sns.heatmap(cm,fmt="d",annot=True)
bottom,top=ax.get_ylim()
ax.set_ylim(bottom+0.5,top-0.5)
```

```
(2.0, 0.0)
```



➢ Classification Report for Random Forest Classifier Algorithm

```
#Classification Report for Random Forest Classifier
print(classification_report(y_pred_rfc,y_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       365
           1       0.96      0.93      0.94       153

    accuracy                           0.97       518
   macro avg       0.96      0.96      0.96       518
weighted avg       0.97      0.97      0.97       518
```

Here we observe that the Random Forest Classifier algorithm gives us the best accuracy as compared to the Naïve Bayes And the Support Vector Classifier.

DATA VISUALIZATION

➢ Finding the Mean of the 'Prediction' Label

```
email.groupby('Prediction').mean()
```

Prediction	the	to	ect	and	for	of	a	you	hou	in	...	enhancements	connevey	jay	va
0	6.673747	5.851307	6.232298	2.633715	3.133987	1.841776	49.485566	2.107298	2.595861	8.701797	...	0.007353	0.007081	0.016612	0.00
1	6.559333	7.012667	2.479333	4.157333	3.102000	4.549333	70.283333	3.346000	0.625333	15.247333	...	0.001333	0.000000	0.002667	0.02

2 rows × 3000 columns

➢ Finding the number of spam and non-spam email

```
email.groupby('Prediction').count()
```

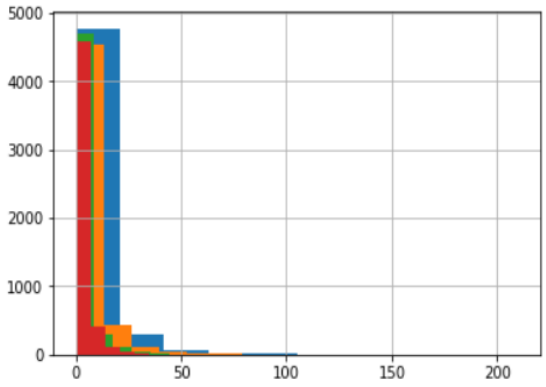
Prediction	Email No.	the	to	ect	and	for	of	a	you	hou	...	enhancements	connevey	jay	valued	lay	infrastructure	military	allowing
0	3672	3672	3672	3672	3672	3672	3672	3672	3672	3672	...	3672	3672	3672	3672	3672	3672	3672	3672
1	1500	1500	1500	1500	1500	1500	1500	1500	1500	1500	...	1500	1500	1500	1500	1500	1500	1500	1500

2 rows × 3001 columns

➢ Histogram of some features

```
email['the'].hist()
email['to'].hist()
email['and'].hist()
email['you'].hist()
```

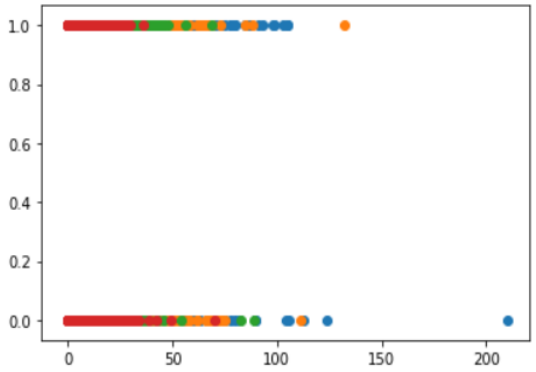
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f31c4e46a0>
```



➢ Scatter plot for some features

```
plt.scatter(email['the'],email['Prediction'])
plt.scatter(email['to'],email['Prediction'])
plt.scatter(email['and'],email['Prediction'])
plt.scatter(email['you'],email['Prediction'])
```

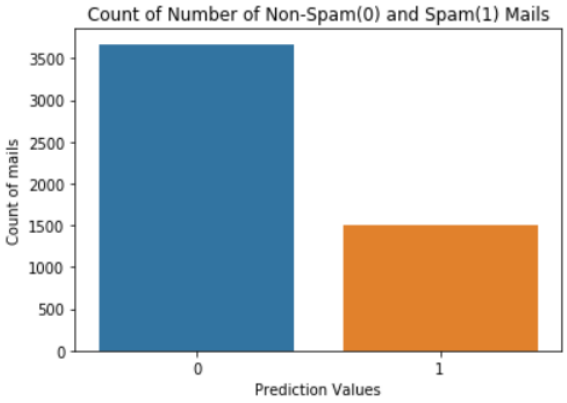
```
<matplotlib.collections.PathCollection at 0x1f31c59c080>
```



➢ Count plot for the prediction label

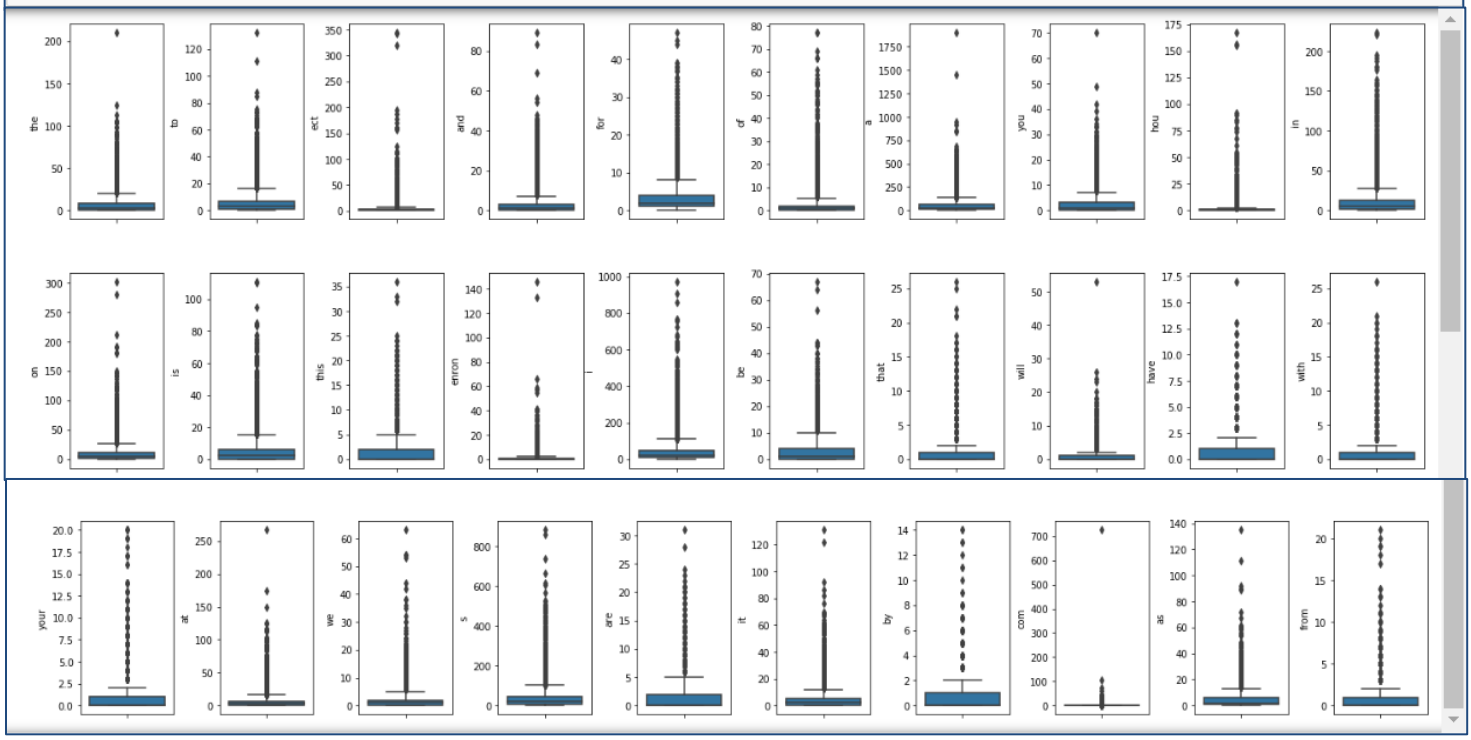
```
sns.countplot(x=email['Prediction'])
plt.xlabel("Prediction Values")
plt.ylabel("Count of mails")
plt.title("Count of Number of Non-Spam(0) and Spam(1) Mails")
```

```
Text(0.5, 1.0, 'Count of Number of Non-Spam(0) and Spam(1) Mails')
```



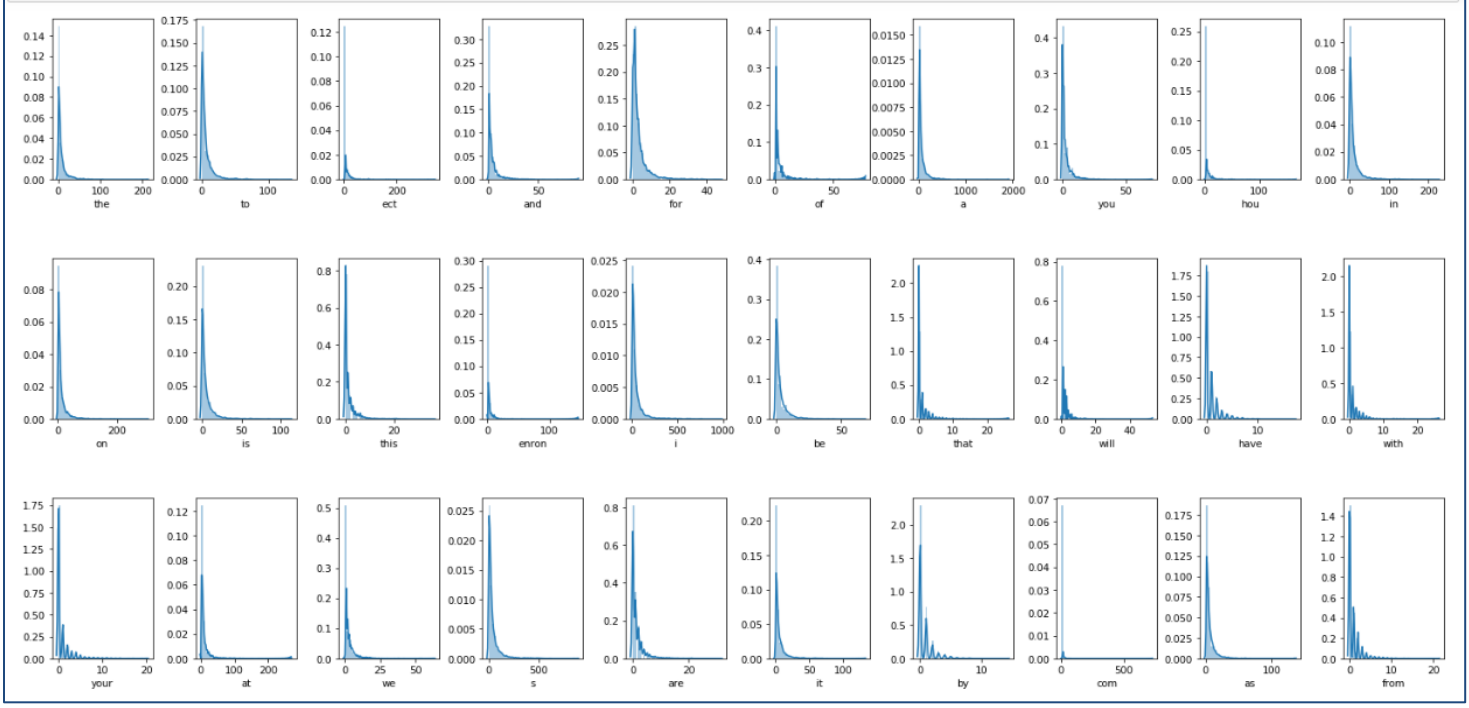
➢ Boxplot for the first 30 features

```
#Box plot for first 30 columns containing words
fig,axs=plt.subplots(ncols=10,nrows=3,figsize=(20,10))  #Gives width and height in figsize
index=0
axs=axs.flatten()  #To flatten to 1D
df=email.iloc[1:,1:]
for k,v in df.items(): #items is dataframe returning two values-index and names of allthe columns
    sns.boxplot(y=v,data=df,ax=axs[index],orient='v')
    if(index==29):
        break;
    index+=1
plt.tight_layout(pad=0.4, w_pad=0.1, h_pad=5.0)  #Used for styling
```



➢ Distplot for the first 30 features

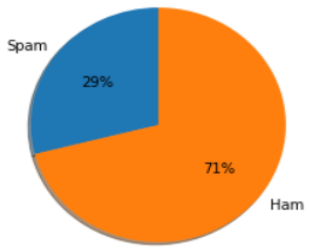
```
#Distplot for first 30 columns containing words
fig,axs=plt.subplots(ncols=10,nrows=3,figsize=(20,10))
index=0
axs=axs.flatten()
df=email.iloc[1:,1:]
for k,v in df.items():
    sns.distplot(v,ax=axs[index])
    if(index==29):
        break;
    index+=1
plt.tight_layout(pad=0.4, w_pad=0.1, h_pad=5.0)
```



➢ Pie Chart for the Prediction Label

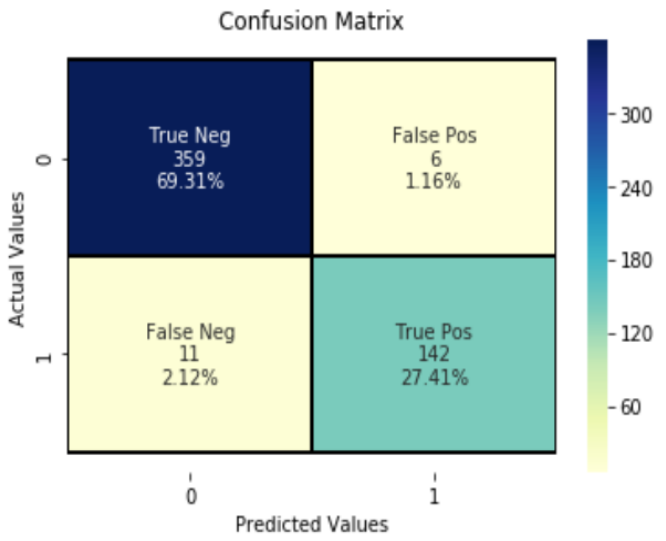
```
#Drawing the pie-chart for number of spam and non-spam emails
ham=(email.groupby('Prediction').count()).iloc[0][0]
spam=(email.groupby('Prediction').count()).iloc[1][0]
plt.pie([spam,ham], labels=['Spam','Ham'], autopct='%0.f%%', shadow=True, startangle=90)
```

```
([<matplotlib.patches.Wedge at 0x1f31bef84a8>,
  <matplotlib.patches.Wedge at 0x1f31bef8e48>],
 [Text(-0.8692196664065499, 0.6741343868488585, 'Spam'),
  Text(0.8692196664065501, -0.6741343868488582, 'Ham')],
 [Text(-0.47411981803993625, 0.36770966555392276, '29%'),
  Text(0.47411981803993636, -0.36770966555392265, '71%')])
```



➢ Better Visualization of Confusion Matrix of Random Forest Classifier

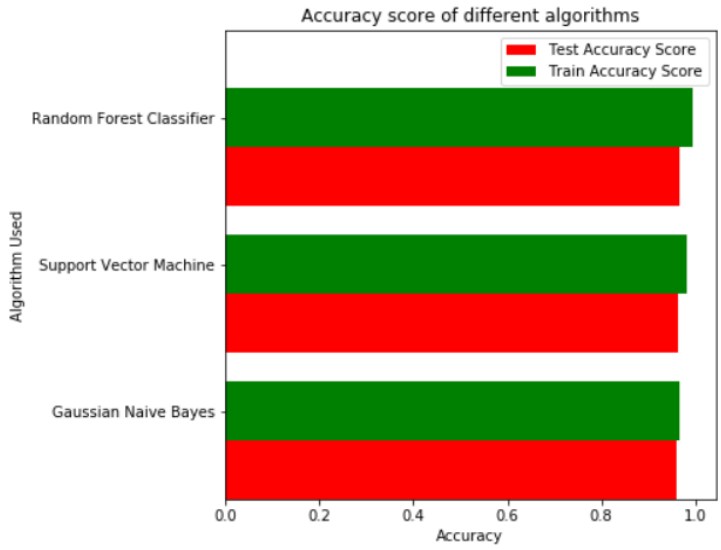
```
# Better Visualization of Confusion Matrix for Random Forest Classifier
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)

fig,corr=plt.subplots(figsize=(6,4))
ax=sns.heatmap(cf_matrix, linewidths=.5,linecolor='black',annot=labels,fmt='',cmap='YlGnBu')
bottom,top=ax.get_ylim()
ax.set_ylim(bottom+0.6,top-0.6)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.savefig('Confusion Matrix')
```



➢ Visual Comparison of the accuracy of the three Algorithms

```
a_y=[as_gnb,as_svc,as_rfc]  #Accuracy Score of various algorithms
t_y=[ts_gnb,ts_svc,ts_rfc]  #Training Score of various algorithms
x=['Gaussian Naive Bayes','Support Vector Machine','Random Forest Classifier']

df = pd.DataFrame(dict(graph=x, n=a_y, m=t_y))
ind = np.arange(len(df))
width = 0.4
fig, ax = plt.subplots(figsize=(6,6))

ax.barh(ind, df.n, width, color='red', label='Test Accuracy Score')
ax.barh(ind + width, df.m, width, color='green', label='Train Accuracy Score')
ax.set(yticks=ind + width, yticklabels=df.graph, ylim=[2*width - 1, len(df)])
ax.legend()

plt.xlabel('Accuracy')
plt.ylabel('Algorithm Used')
plt.title('Accuracy score of different algorithms')
```

```
Text(0.5, 1.0, 'Accuracy score of different algorithms')
```



RESULTS

➢ COMPARING LEARNING SYSTEMS

The accuracy score (prediction score) obtained after applying the different algorithms are:

ALGORITHM USED	*ACCURACY SCORE*
NAÏVE BAYES	**95.87%**
SUPPORT VECTOR MACHINE	**96.33%**
RANDOM FOREST CLASSIFIER	**96.71%**

All the three algorithms gave good accuracy of 96% approx. Thus, all three are good algorithms for Email Spam Filtering Model. It is seen from the above table that we get maximum accuracy in Random Forest Classifier Algorithm of approximately 97%.

DISSCUSSION

We applied different classification algorithms to find out best accuracy or prediction score on the given dataset. Before applying the algorithms, the dataset has been preprocessed or as we can say the data is cleaned. We checked if any null values or strings are present in the dataset. We also checked the correlation between different columns to find out X and Y and then the data is finally splitted to create test and train set and after that we apply different algorithms to check whether an E-mail is spam or not.

CONCLUSION

- The main goal of this project is to find whether an E-mail is spam or not.
- We do the prediction by taking common words present in spam mail as features and then based on the presence of those words in the received mail we classify the mail as spam or not spam.
- E-mail spam filtering is very useful as it will help the user to easily classify their mails and save their time.
- This project involved three supervised learning algorithms i.e. Naive Bayes, Support vector machine and random forest classifier. All of them gave different accuracy score. We got most score in Random Forest Classifier but it may vary depending upon datasets.
- We concluded that our prediction was most accurate using Random Forest Classifier.

FUTURE SCOPE

In this project, we have applied various machine learning algorithms for spam filtering. Substantial work has been done to improve the effectiveness of spam filters for classifying emails as either ham (valid messages) or spam (unwanted messages) by means of ML classifiers. Further research work needs to be conducted to tackle the fact that email spam filtering is a concept drift problem. The future of email spam filters lies in deep learning for content-based classification, neural networks and deep adversarial learning techniques. The traditional machine learning algorithms finds it very hard to mine adequately-represented features because to the limitations that characterized such algorithms. The shortcomings of the usual machine learning algorithms include: need for knowledge from expert in a particular field, curse of dimensionality, and high computational cost. Deep learning will be far more effective in solving the problem of spam email because as number of available training data is increasing, the effectiveness and efficiency of deep learning becomes more pronounced. For future work, we suggest that a combination of keywords and descriptive characteristics may provide more accurate classification, as well as the combination of spam classificatory techniques. We have interest in continuing professional development of this thesis work, incorporating new features that provide artificial neural networks, as well as a self-learning system that allows work in the real world on the basis of new features and classification techniques spam. The ultimate goal would be to obtain a 99.9% of maximum accuracy.

REFERENCES

For completing my project on email spam filtering using machine learning, I have collected and used information from various sources. Some of them are:

- https://www.kaggle.com/
- https://www.python.org/
- https://anaconda.org/anaconda/python/
- http://www.numpy.org/
- https://matplotlib.org/
- http://scikit-learn.org/
- https://pandas.pydata.org/