# EMAIL SPAM DETECTION USING MACHINE LEARNING ALGORITHMS


## Abstract:

Machine learning and Feature Selection are playing a vital role in internet and cyber security also. 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.

Email Spam has become a major problem nowadays, with Rapid growth of internet users, Email spams is also increasing. People are using them for illegal and unethical conducts, phishing and fraud. Sending malicious link through spam emails which can harm our system and can also seek in into your system. Creating a fake profile and email account is much easy for the spammers, they pretend like a genuine person in their spam emails, these spammers target those peoples who are not aware about these frauds. So, it is needed to Identify those spam mails which are fraud, this project will identify those spam by using techniques of machine learning, this paper will discuss the machine learning algorithms and apply all these algorithm on our data sets and best algorithm is selected for the email spam detection having best precision and accuracy. The goal of this project is to develop an appropriate machine learning tool which can predict whether the Email is spam or not. The algorithm that can be used here are SVM, Logistic Regression and Random Forest

# TABLE OF CONTENTS

# CHAPTER I

# INTRODUCTION

# Jupiter

Jupyter, previously known as IPython Notebook, is a web-based, interactive development environment. Originally developed for Python, it has since expanded to support over 40 other programming languages including Julia and R.

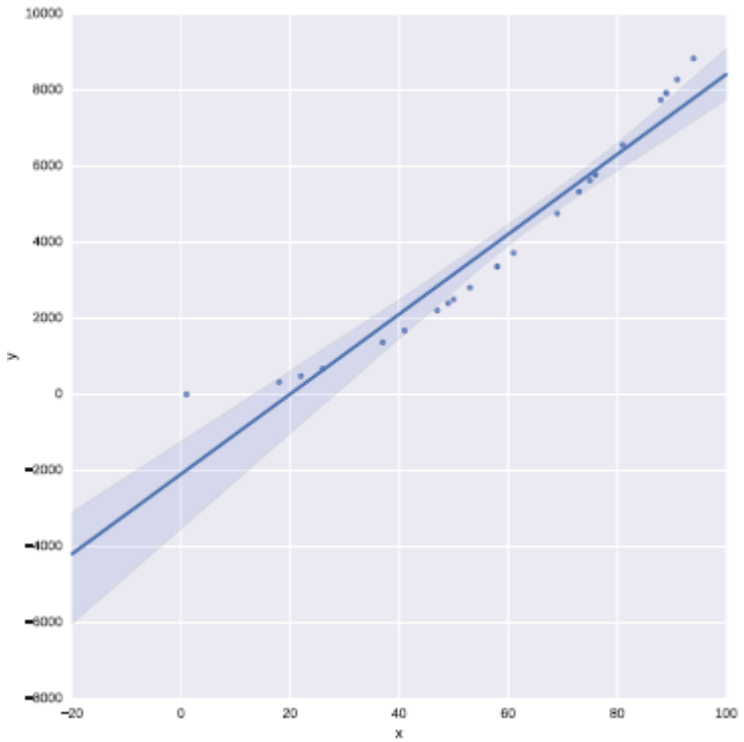
Jupyter allows for *notebooks*to be written that contain text, live code, images, and equations. These notebooks can be shared, and can even be hosted on GitHubfor free.

For each section of this tutorial, you can download a Juypter notebook that allows you to edit and experiment with the code and examples for each topic. Jupyter is part of the Anaconda distribution; it can be started from the command line using the jupyter command:

```
$ jupyter notebook
```

**Machine Learning**

We will now move on to the task of machine learning itself. In the following sections we will describe how to use some basic algorithms, and perform regression, classification, and clustering on some freely available medical datasets concerning breast cancer and diabetes, and we will also take a look at a DNA microarray dataset.



**SciKit-Learn**

SciKit-Learn provides a standardised interface to many of the most commonly used machine learning algorithms, and is the most popular and frequently used library for machine learning for Python. As well as providing many learning algorithms, SciKit-Learn has a large number of convenience functions for common preprocessing tasks (for example, normalisation or *k*-fold cross validation).

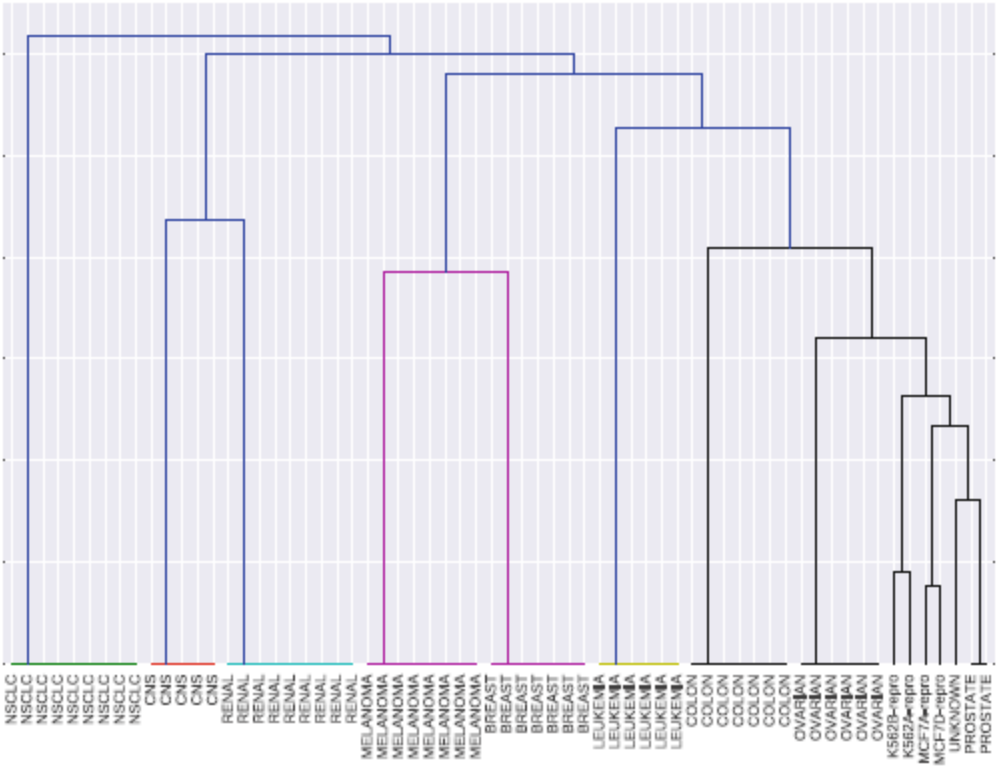
SciKit-Learn is a very large software library.

**Clustering**

Clustering algorithms focus on ordering data together into groups. In general clustering algorithms are unsupervised—they require no **y** response variable as input. That is to say, they attempt to find groups or clusters within data where you do not know the label for each sample. SciKit-Learn have many clusteringalgorithms, but in this section we will demonstrate hierarchical clustering on a DNA expression microarray dataset using an algorithm from the SciPy library.

We will plot a visualisation of the clustering using what is known as a dendrogram, also using the SciPy library.

The goal is to cluster the data properly in logical groups, in this case into the cancer types represented by each sample's expression data. We do this using agglomerative hierarchical clustering, using Ward's linkage method:




## Classification

weanalysed data that was **unlabelled**—we did not know to what class a sample belonged (known as unsupervised learning). In contrast to this, a supervised problem deals with **labelled** data where are aware of the discrete classes to which each sample belongs. When we wish to predict which class a sample belongs to, we call this a classification problem. SciKit-Learn has a number of algorithms for classification, in this section we will look at the Support Vector Machine.

We will work on the Wisconsin breast cancer dataset, split it into a training set and a test set, train a Support Vector Machine with a linear kernel, and test the trained model on an unseen dataset. The Support Vector Machine model should be able to predict if a new sample is malignant or benign based on the features of a new, unseen sample:

```
>>> from sklearn import cross_validation
>>> from sklearn import datasets
>>> from sklearn.svm import SVC
>>> from sklearn.metrics import classification_report
>>> X = datasets.load_breast_cancer().data
>>> y = datasets.load_breast_cancer().target
>>> X_train, X_test, y_train, y_test = cross_validation.
    train_test_split(X, y, test_size=0.2)
>>> svm = SVC(kernel="linear")
>>> svm.fit(X_train, y_train)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape=None, degree=3, gamma="auto",
    kernel="linear", max_iter=-1, probability=False,
    random_state=None, shrinking=True,  tol=0.001, verbose=
    False)
>>> svm.score(X_test, y_test)
0.95614035087719296
>>> y_pred = svm.predict(X_test)
>>> classification_report(y_test, y_pred)

             precision    recall  f1-score   support

  malignant       1.00      0.89      0.94        44
     benign       0.93      1.00      0.97        70

avg / total       0.96      0.96      0.96       114
```

You will notice that the SVM model performed very well at predicting the malignancy of new, unseen samples from the test set—this can be quantified nicely by printing a number of metrics using the classification report function. Here, the precision, recall, and *F*1 score (*F*1 = 2·precision·recall/precision+recall) for each class is shown. The support column is a count of the number of samples for each class.

Support Vector Machines are a very powerful tool for classification. They work well in high dimensional spaces, even when the number of features is higher than the number of samples. However, their running time is quadratic to the number of samples so large datasets can become difficult to train. Quadratic means that if you increase a dataset in size by 10 times, it will take 100 times longer to train.

Last, you will notice that the breast cancer dataset consisted of 30 features. This makes it difficult to visualize or plot the data. To aid in visualization of highly dimensional data, we can apply a technique called dimensionality reduction.

## Dimensionality Reduction

Another important method in machine learning, and data science in general, is dimensionality reduction. For this example, we will look at the Wisconsin breast cancer dataset once again. The dataset consists of over 500 samples, where each sample has 30 features. The features relate to

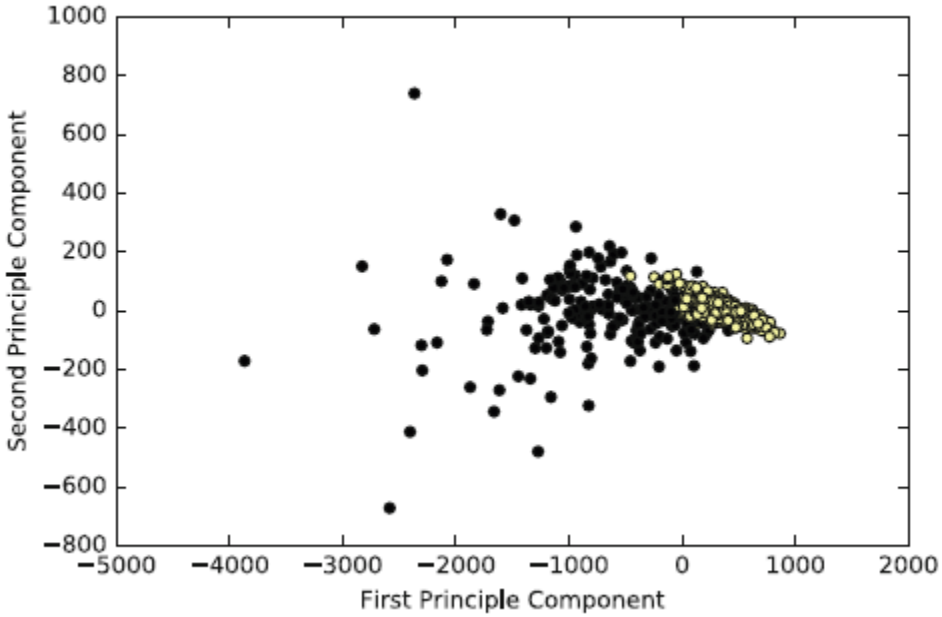
images of a fine needle aspirate of breast tissue, and the features describe the characteristics of the cells present in the images. All features are real values. The target variable is a discrete value (either malignant or benign) and is therefore a classification dataset.

You will recall from the Iris example in Sect. 7.3 that we plotted a scatter matrix of the data, where each feature was plotted against every other feature in the dataset to look for potential correlations (Fig. 3). By examining this plot you could probably find features which would separate the dataset into groups. Because the dataset only had 4 features we were able to plot each feature against each other relatively easily. However, as the numbers of features grow, this becomes less and less feasible, especially if you consider the gene expression example in Sect. 9.4 which had over 6000 features.

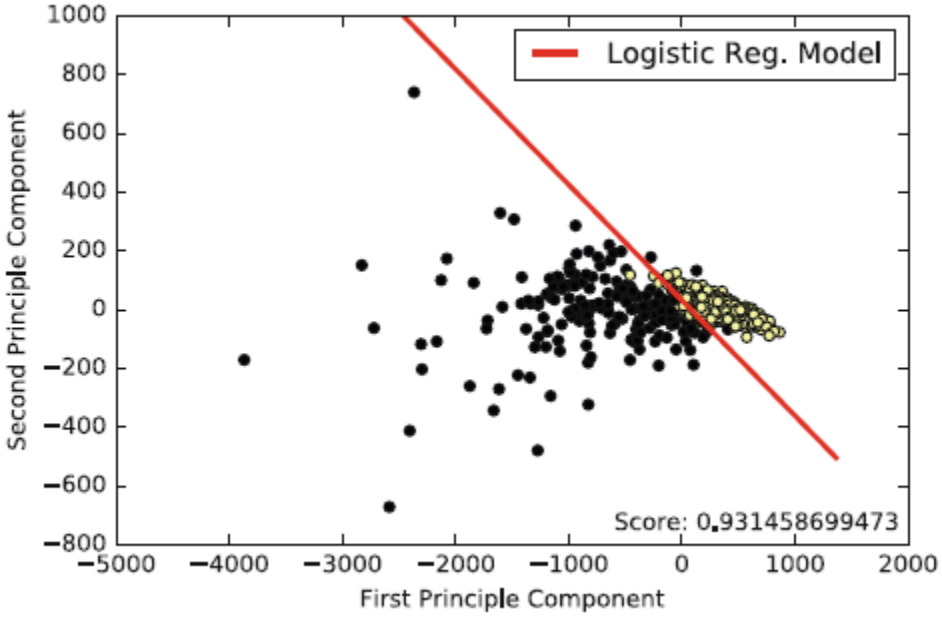
One method that is used to handle data that is highly dimensional is Principle Component Analysis, or PCA. PCA is an unsupervised algorithm for reducing the number of dimensions of a dataset. For example, for plotting purposes you might want to reduce your data down to 2 or 3 dimensions, and PCA allows

you to do this by generating components, which are combinations of the original features, that you can then use to plot your data.

PCA is an unsupervised algorithm. You supply it with your data, **X**, and you specify the number of components you wish to reduce its dimensionality to. This is known as transforming the data:



Again, you would not use this model for new data—in a real world scenario, you would, for example, perform a 10-fold cross validation on the dataset, choosing the model parameters that perform best on the cross validation. This model would be much more likely to perform well on new data. At the very least, you would randomly select a subset, say 30% of the data, as a test set and train the model on the remaining 70% of the dataset. You would evaluate the model based on the score on the test set and not on the training set




## NEURAL NETWORKS AND DEEP LEARNING

While a proper description of neural networks and deep learning is far beyond the scope of this chapter, we will however discuss an example use case of one of the most popular frameworks for deep learning: Keras4.

In this section we will use Keras to build a simple neural network to classify theWisconsin breast cancer dataset that was described earlier. Often, deep learning algorithms and neural networks are used to classify images—convolutional neural networks are especially used for image related classification. However,

they can of course be used for text or tabular-based data as well. In this we will build a standard feed-forward, densely connected neural network and classify a text-based cancer dataset in order to demonstrate the framework'susage.

In this example we are once again using the Wisconsin breast cancer dataset, which consists of 30 features and 569 individual samples. To make it more challenging for the neural network, we will

use a training set consisting of only 50% of the entire dataset, and test our neural network on the remaining 50% of the data.

Note,Keras is not installed as part of the Anaconda distribution, to install it use pip:

```
$sudo pip install keras
```

Keras additionally requires either Theano or TensorFlow to be installed. In the examples in this chapter we are using Theano as a backend, however the code will work identically for either backend. You can install Theano using pip, but it has a number of dependencies that must be installed first. Refer to the Theano and TensorFlow documentation for more information [12].

Keras is a modular API. It allows you to create neural networks by building a stack of modules, from the input of the neural network, to the output of the neural network, piece by piece until you have a complete network. Also, Keras can be configured to use your Graphics Processing Unit, or GPU. This makes training neural networks far faster than if we were to use a CPU. We begin by importing Keras:

```
>>> import keras
Using Theano backend.
Using gpu device 0: GeForce GTX TITAN X (CNMeM is enabled
    with initial size: 90.0 % of memory, cuDNN 4007)
```

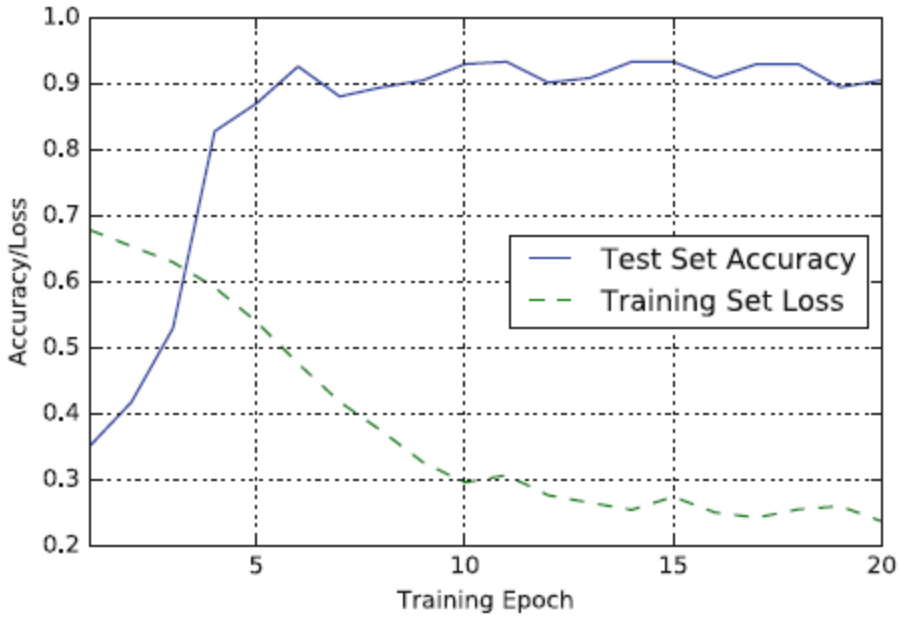
We may want to view the network's accuracy on the test (or its loss on the training set) over time (measured at each epoch), to get a better idea how well it is learning. An epoch is one complete cycle through the training data.

Fortunately, this is quite easy to plot as Keras' fit function returns a history object which we can use to do exactly this:

```
>>> plt.plot(h.history["val_acc"])
>>> plt.plot(h.history["loss"])
>>> plt.show()
```

This will result in a plot similar to that shown. Often you will also want to plot the loss on the test set and training set, and the accuracy on the test set and training set.

Plotting the loss and accuracy can be used to see if you are over fitting (you experience tiny loss on the training set, but large loss on the test set) and to see when your training has plateaued.




## EXISTING SYSTEM

In the current work exploratory data analysis steps like load the data, check the class labels,plot the count plot for each class and Featuraization convert the text into numeric form using bag of words and tfidf vectorizer are used before applying to SVM algorithm .Here , Logistic Regression is used as base model.

## DISADVANTAGES OF EXISTING SYSTEM

- Methods have performance limitations because of wide range of variations in data and amount of data is limited.
- Issue involved in rainfall classification is choosing the required sampling recess of Observation-Forecasting of rainfall, which is dependent upon the sampling interval of input data.
- Less accuracy

## LITERATURE SURVEY:

## 1. TITLE: MACHINE LEARNING METHODS FOR SPAM E-MAIL CLASSIFICATION

**Author**: Awad W.A, ELseuofi S.M

**YEAR**: - 2018

**Abstract:**

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 avery successful rate. In this paper we review some of the most popular machine learning methods (Bayesian classification, k-NN, ANNs, SVMs, Artificial immune system and Rough sets) and of their applicability to the problem of spam Email classification. Descriptions of the algorithms are presented,and the comparison of their performance on the SpamAssassin spam corpus is presented.

## 2. TITLE: MACHINE LEARNING BASED SPAM E-MAIL DETECTION

**Author**: Priti Sharma, Uma Bhardwaj

**YEAR**: - 2017

**Abstract:**

Spam email is one of the biggest issues in the world of internet. Spam emails not only influence the organisations financially but also exasperate the individual email user. This paper aims to propose a machine learning based hybrid bagging approach by implementing the two machine learning algorithms: Naïve Bayes and J48 (decision tree) for the spam email detection. In this process, dataset is divided into different sets and given as input to each algorithm. Total three experiments are performed and the results obtained are compared in terms of precision, recall, accuracy, f-measure, true negative rate, false positive rate and false negative rate. The two experiments are performed using individual Naïve Bayes & J48 algorithms. Third experiment is the proposed SMD system implemented using hybrid bagged approach. The overall accuracy of 87.5% achieved by the hybrid bagged approach based SMD system.

## 3. TITLE: IMPROVED EMAIL SPAM DETECTION MODEL WITH NEGATIVE SELECTION ALGORITHM AND PARTICLE SWARM OPTIMIZATION

**Author**:IsmailaIdris, Ali Selamat1

**YEAR**: - 2014

**Abstract:**

The adaptive nature of unsolicited email by the use of huge mailing tools prompts the need for spam detection. Implementation of different spam detection methods based on machine learning techniques was proposed to solve the problem of numerous email spam ravaging the system. Previous algorithm used in email spam detection compares each email message with spam and non-spam data before generating detectors while our proposed system inspired by the artificial immune system model with the adaptive nature of negative selection algorithm uses special features to generate detectors to cover the spam space. To cope with the trend of email spam, a novel model that improves the random generation of a detector in negative selection algorithm (NSA) with the use of stochastic distribution to model the data point using particle swarm optimization (PSO) was implemented. Local outlier factor is introduced as the fitness function to determine the local best (Pbest) of the candidate detector that gives the optimum solution. Distance measure is employed to enhance the distinctiveness between the non-spam and spam candidate detector. The detector generation process was terminated when the expected spam coverage is reached. The theoretical analysis and the experimental result show that the detection rate of NSA–PSO is higher than the standard negative selection algorithm. Accuracy for 2000 generated detectors with threshold value of 0.4 was compared. Negative selection algorithm is 68.86% and the proposed hybrid negative selection algorithm with particle swarm optimization is 91.22%.

## 4. TITLE: DETECTING SPAM BOTS IN ONLINE SOCIAL NETWORKING SITES

**Author**: Alex Hai Wang

**YEAR**: - 2018

**Abstract:**

As online social networking sites become more and more popular, they have also attracted the attentions of the spammers. In this paper, Twitter, a popular micro-blogging service, is studied as an example of spam bots detection in online social networking sites. A machine learning approach is proposed to distinguish the spam bots from normal ones. To facilitate the spam bots detection, three graph-based features, such as the number of friends and the number of followers, are extracted to explore the unique follower and friend relationships among users on Twitter. Three content-based features are also extracted from user's most recent 20 tweets. A real data set is collected from Twitter's public available information using two different methods. Evaluation experiments show that the detection system is efficient and accurate to identify spam bots in Twitter.

## 5. TITLE: SPAM DETECTION USING TEXT CLUSTERING

**Author**: M.Sasaki,H.Shinnou

**YEAR**: - 2005

**Abstract:**

We propose a new spam detection technique using the text clustering based on vector space model. Our method computes disjoint clusters automatically using a spherical k-means algorithm for all spam/non-spam mails and obtains centroid vectors of the clusters for extracting the cluster description. For each centroid vectors, the label (`spam' or `non-spam') is assigned by calculating the number of spam email in the cluster. When new mail arrives, the cosine similarity between the new mail vector and centroid vector is calculated. Finally, the label of the most relevant cluster is assigned to the new mail. By using our method, we can extract many kinds of topics in spam/non-spam email and detect the spam email efficiently. In this paper, we describe the our spam detection system and show the result of our experiments using the Ling-Spam test collection

## 6. TITLE: SMS SPAM DETECTION USING MACHINE LEARNING APPROACH

**Author**: HoushmandShirani-mehr

**YEAR**: - 2013

**Abstract:**

Over recent years, as the popularity of mobile phone devices has increased, Short Message Service (SMS) has grown into a multi-billion dollars industry. At the same time, reduction in the cost of messaging services has resulted in growth in unsolicited commercial advertisements (spams) being sent to mobile phones. In parts of Asia, up to 30% of text messages were spam in 2012. Lack of real databases for SMS spams, short length of messages and limited features, and their informal language are the factors that may cause the established email filtering algorithms to underperform in their classification. In this project, a database of real SMS Spams from UCI Machine Learning repository is used, and after preprocessing and feature extraction, different machine learning techniques are applied to the database. Finally, the results are compared and the best algorithm for spam filtering for text messaging is introduced. Final simulation results using 10-fold cross validation shows the best classifier in this work reduces the overall error rate of best model in original paper citing this dataset by more than half.

### 1.1 PROPOSED SYSTEM

In the proposed model Random forest is used as model for classification. Random forest gives best performance compared to another models

## Algorithms:

## SVM Algorithm:

**What are Support Vector Machines?**

Support Vector Machine (SVM) is a relatively simple Supervised Machine Learning

Algorithm used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2-dimensional space, this hyper-plane is nothing but a line. In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyperplane to separate the data. So by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes). However, there are various techniques to use for multi-class problems.

**Support Vector Machine for Multi-Class Problems**

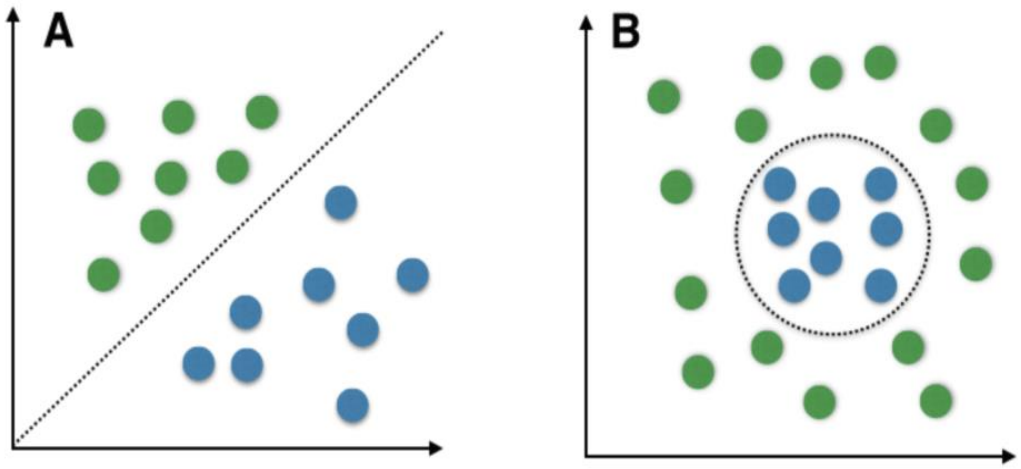
To perform SVM on multi-class problems, we can create a binary classifier for each class of the data. The two results of each classifier will be :

- The data point belongs to that class OR
- The data point does not belong to that class.

For example, in a class of fruits, to perform multi-class classification, we can create a binary classifier for each fruit. For say, the 'mango' class, there will be a binary classifier to predict if it IS a mango OR it is NOT a mango. The classifier with the highest score is chosen as the output of the SVM.

**SVM for complex (Non Linearly Separable)**

SVM works very well without any modifications for linearly separable data. Linearly Separable Data is any data that can be plotted in a graph and can be separated into classes using a straight line.




*A: Linearly Separable Data B: Non-Linearly Separable Data*

We use Kernelized SVM for non-linearly separable data. Say, we have some non-linearly separable data in one dimension. We can transform this data into two-dimensions and the data will become linearly separable in two dimensions. This is done by mapping each 1-D data point to a corresponding 2-D ordered pair.
So for any non-linearly separable data in any dimension, we can just map the data to a higher dimension and then make it linearly separable. This is a very powerful and general transformation.
A kernel is nothing a measure of similarity between data points. The kernel function in a kernelized SVM tell you, that given two data points in the original feature space, what the similarity is between the points in the newly transformed feature space.
There are various kernel functions available, but two of are very popular

- **Radial Basis Function Kernel (RBF):** The similarity between two points in the transformed feature space is an exponentially decaying function of the distance between the vectors and the original input space as shown below. RBF is the default kernel used in SVM.

- **Polynomial Kernel:** The Polynomial kernel takes an additional parameter, 'degree' that controls the model's complexity and computational cost of the transformation

A very interesting fact is that SVM does not actually have to perform this actual transformation on the data points to the new high dimensional feature space. This is called the kernel trick.

The Kernel Trick:

Internally, the kernelized SVM can compute these complex transformations just in terms of similarity calculations between pairs of points in the higher dimensional feature space where the transformed feature representation is implicit.

This similarity function, which is mathematically a kind of complex dot product is actually the kernel of a kernelized SVM. This makes it practical to apply SVM, when the underlying feature space is complex, or even infinite-dimensional. The kernel trick itself is quite complex and is beyond the scope of this article.

**Important Parameters in Kernelized SVC ( Support Vector Classifier)**

1. **The Kernel** :The kernel, is selected based on the type of data and also the type of transformation. By default, the kernel is Radial Basis Function Kernel (RBF).

2. **Gamma** :This parameter decides how far the influence of a single training example reaches during transformation, which in turn affects how tightly the decision boundaries end up surrounding points in the input space. If there is a small value of gamma, points farther apart are considered similar. So more points are grouped together and have smoother decision boundaries (may be less accurate). Larger values of gamma cause points to be closer together (may cause overfitting).

3. **The 'C' parameter** :This parameter controls the amount of regularization applied on the data. Large values of C mean low regularization which in turn causes the training data to fit very well (may cause overfitting). Lower values of C mean higher regularization which causes the model to be more tolerant of errors (may lead to lower accuracy).

**Pros of Kernelized SVM:**

1. They perform very well on a range of datasets.

2. They are versatile : different kernel functions can be specified, or custom kernels can also be defined for specific datatypes.
3. They work well for both high and low dimensional data.

**Cons of Kernelized SVM:**

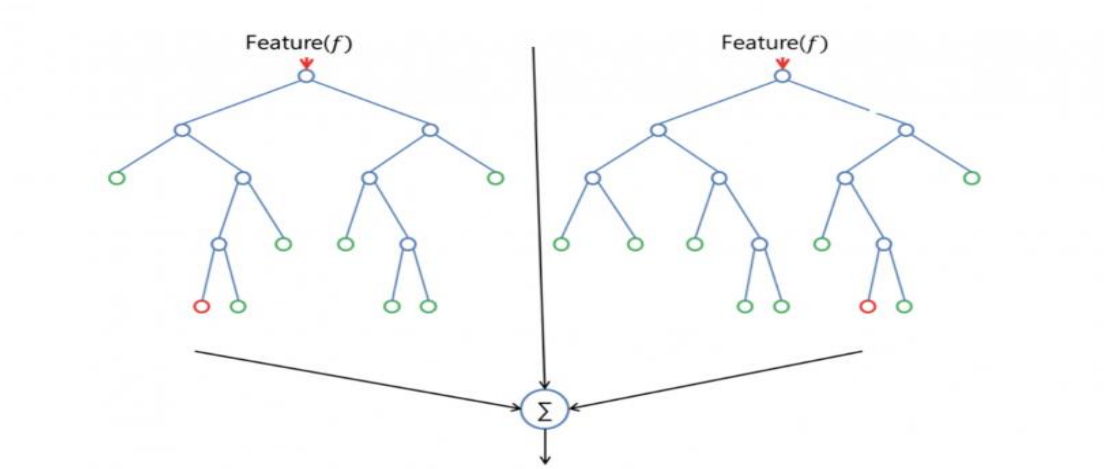
1. Efficiency (running time and memory usage) decreases as size of training set increases.
2. Needs careful normalization of input data and parameter tuning.
3. Does not provide direct probability estimator.
4. Difficult to interpret why a prediction was made.

## Random Forest Algorithm:-

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Put simply: random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. Let's look at random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:

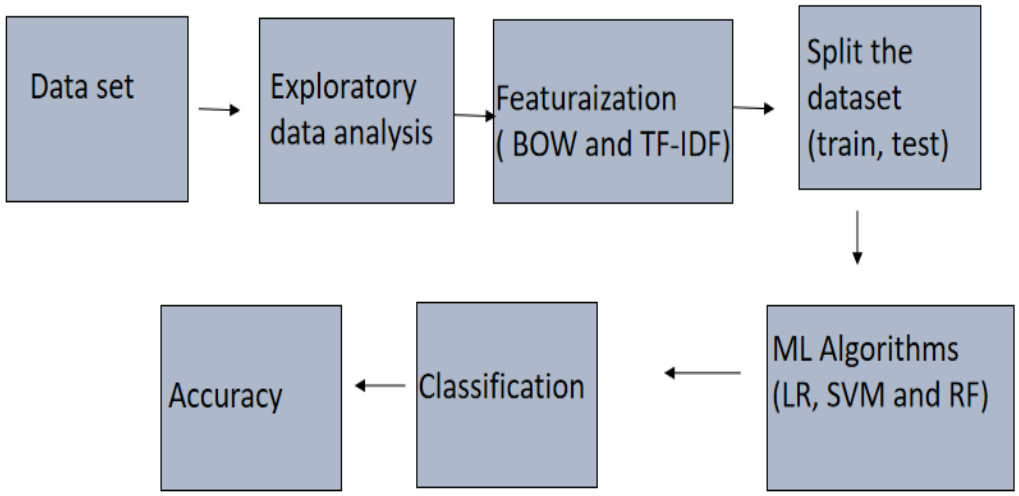



Random forest has nearly the same hyper parameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's repressor.

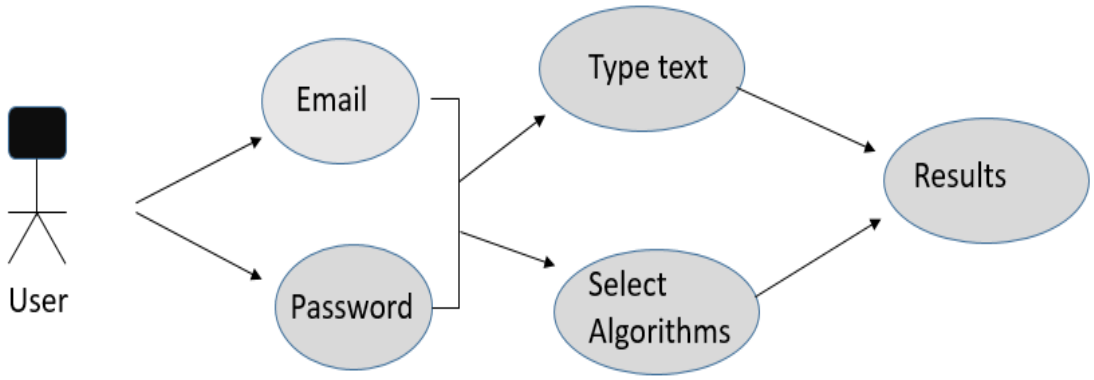
Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in random forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

**Back End Module Diagrams:**



## Front End Module Diagrams:




## SYSTEM SPECIFICATION:

## HARDWARE REQUIREMENTS:

| | | |
|---|---|---|
| PROCESSOR | : | Intel I5 |
| RAM | : | 4GB |
| HARD DISK | : | 40 GB |

**SOFTWARE REQUIREMENTS:**

PYTHON IDE : Anaconda Jupyter Notebook

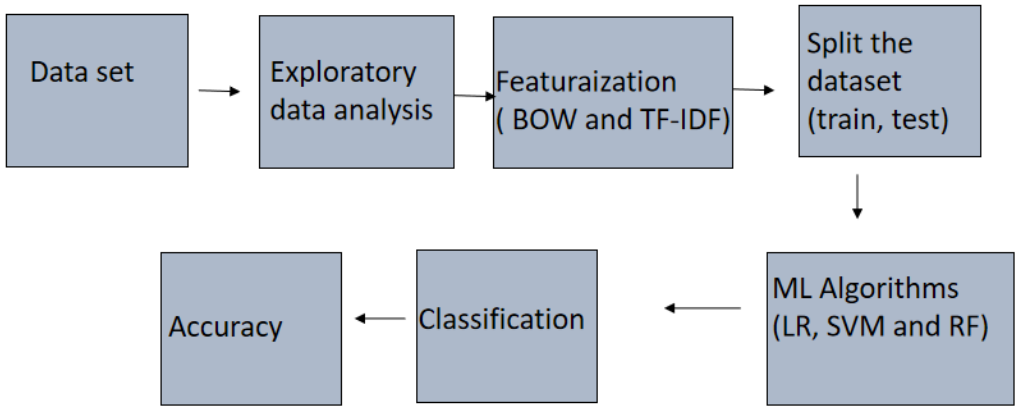
PROGRAMMING LANGUAGE : Python

# IMPLEMENTATION

## Back End Module:

In back end we have the dataset in .csv format. First load the data using pandas library. There are 5171 rows and 3 columns or features. There are some null values in the data. Detect the null values and replace with mean if the feature are numeric feature otherwise most frequently word replace. Here I have used Simple Imputer to replace the all null values. This is binary class classification and imbalanced dataset. I have used BOW to convert the text data into numeric data.

## Back EndModule Diagrams:



**DATA CLEANING:**

In this module the data is cleaned. After cleaning of the data, the data is grouped as per requirement. This grouping of data is known as data clustering. Then check if there is any missing value in the data set or not. It there is some missing value then change it by any default value. After that if any data need to change its format, it is done. That total process before the prediction is known is data pre-processing. After that the data is used for the prediction and forecasting step.

**Data Prediction and forecasting:**

In this step, the pre-processed data is taken for the prediction. This prediction can be done in any process which are mentioned above. But the Linear Regression algorithm score more prediction accuracy than the other algorithm. So, in this project the linear regression method is used for the prediction. For that, the pre-processed data is splitted for the train and test purpose. Then a predictive object is created to predict the test value which is trained by the trained value. Then the object is used to forecast data for next few years.

**DATA SPLITTING:**

For each experiment, we split the entire dataset into 70% training set and 30% test set. We used the training set for resampling, hyper parameter tuning, and training the model and we used test set to test the performance of the trained model. While splitting the data, we specified a random seed (any random number), which ensured the same data split every time the program executed.

**TRAINING AND TESTING:**

Algorithms learn from data. They find relationships, develop understanding, make decisions, and evaluate their confidence from the training data they're given. And the better the training data is, the better the model performs.

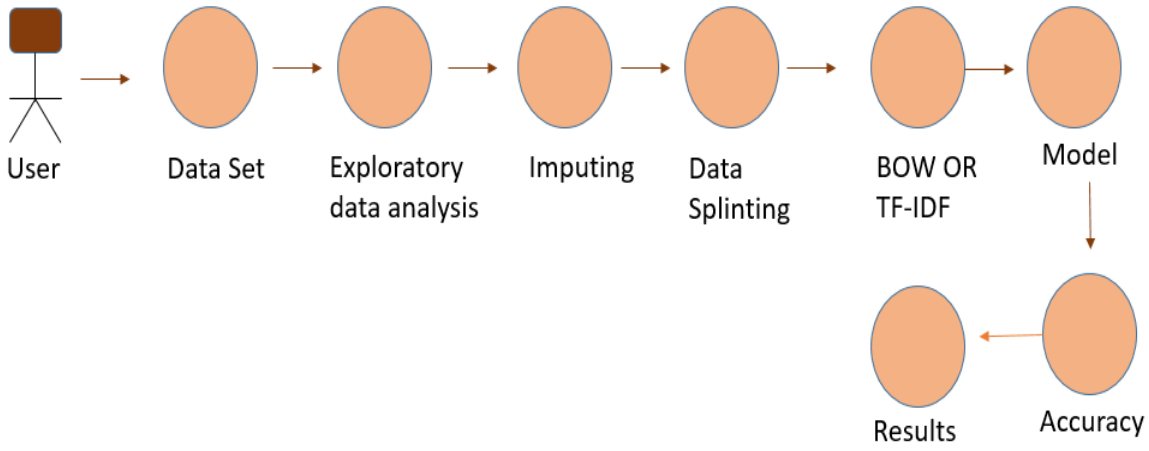
In fact, the quality and quantity of your training data has as much to do with the success of your data project as the algorithms themselves.

Now, even if you've stored a vast amount of well-structured data, it might not be labeled in a way that actually works for training your model. For example, autonomous vehicles don't just need pictures of the road, they need labeled images where each car, pedestrian, street sign and more are annotated; sentiment analysis projects require labels that help an algorithm understand when someone's using slang or sarcasm; chatbots need entity extraction and careful syntactic analysis, not just raw language.

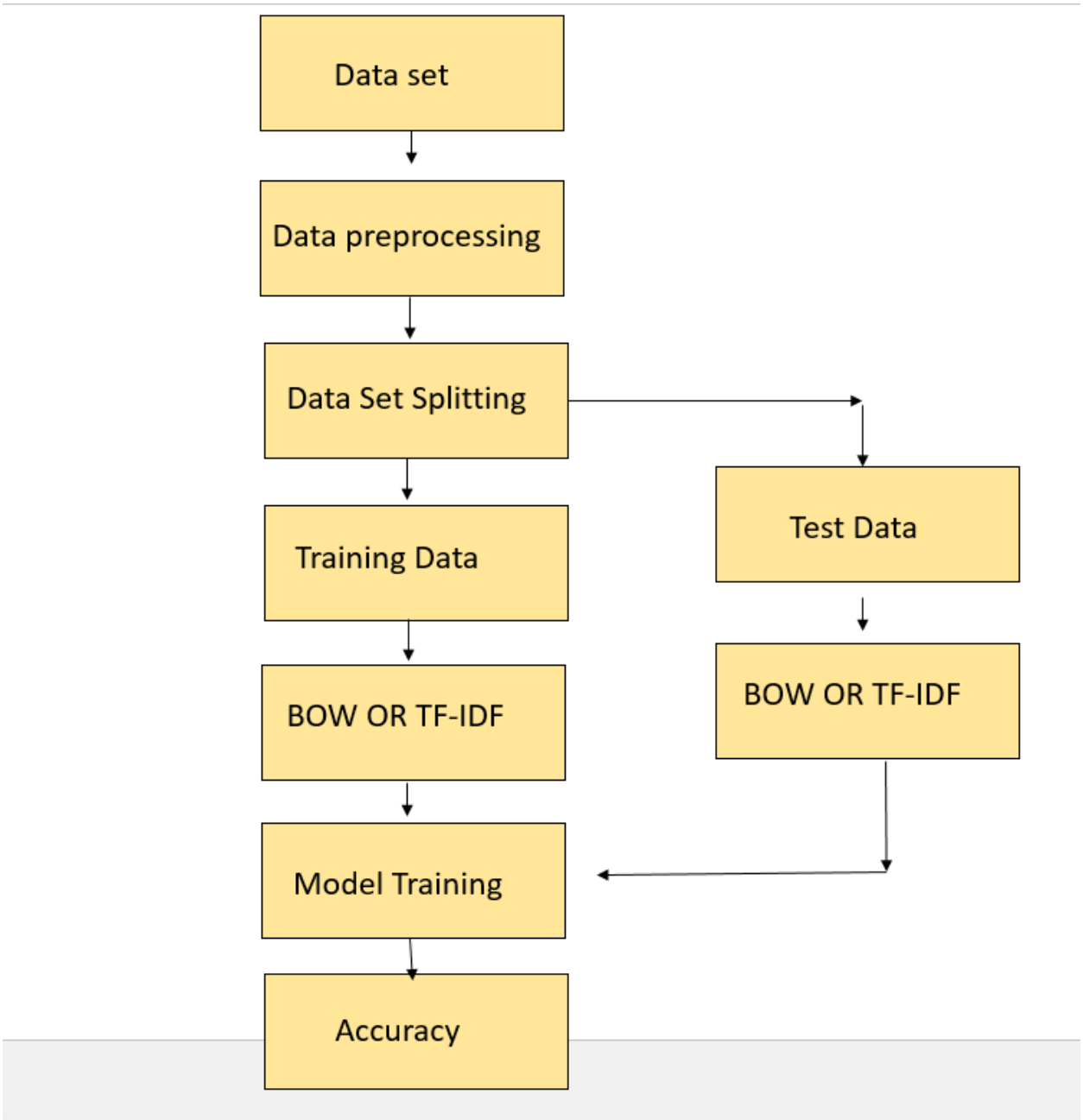
In other words, the data you want to use for training usually needs to be enriched or labeled. Or you might just need to collect more of it to power your algorithms. But chances are, the data you've stored isn't quite ready to be used to train your classifiers.

Because if you're trying to make a great model, you need great training data. And we know a thing or two about that. After all, we've labeled over 5 billion rows of data for some of the most innovative companies in the world. Whether it's images, text, audio, or, really, any other kind of data, we can help create the training set that makes your models successful.

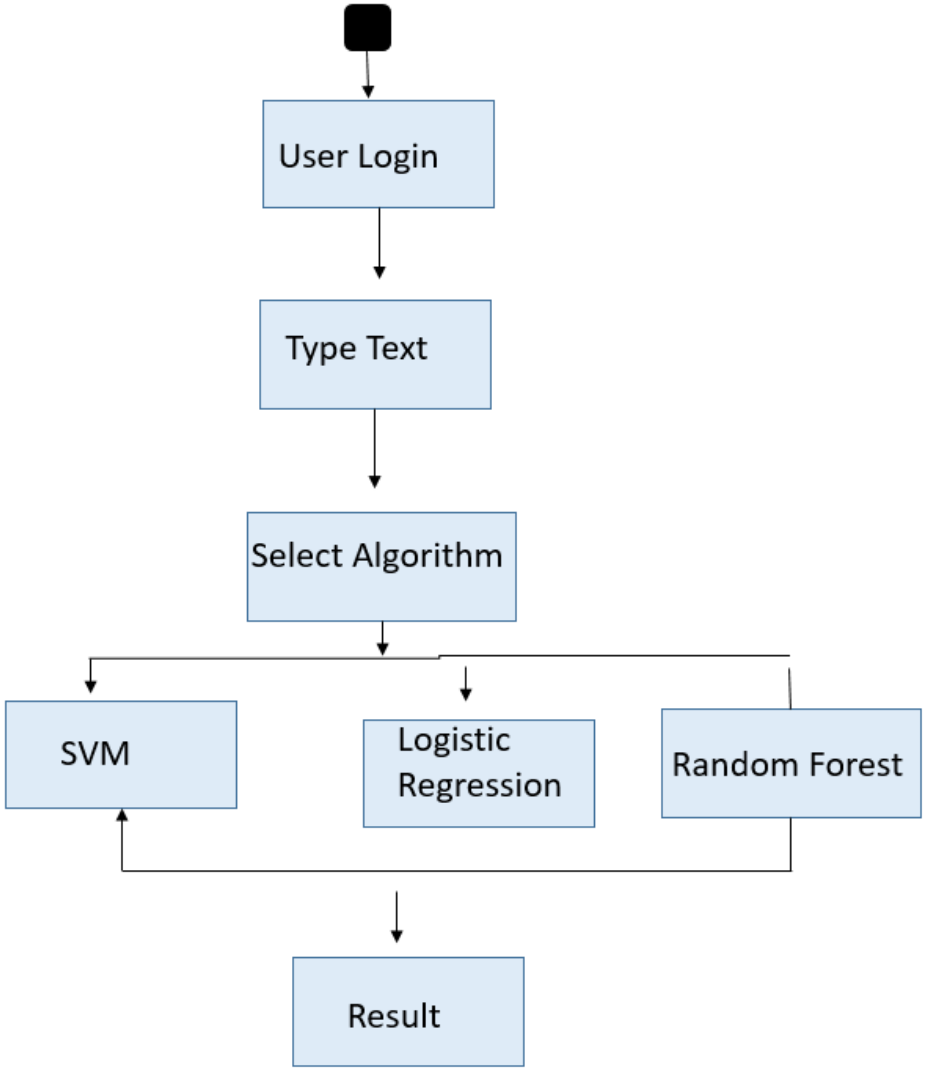
## Use Case Diagram:




**State Diagram:**




**Activity Diagram:**




# CHAPTER 3

## SOFTWARE SPECIFICATION

### 3.1 GENERAL

**ANACONDA**

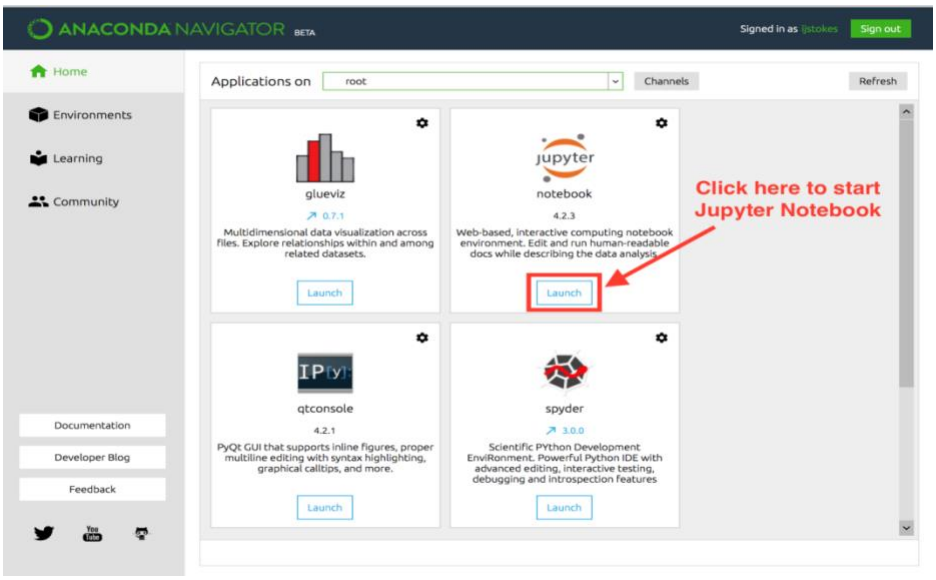
It is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

Anaconda distribution comes with more than 1,500 packages as well as the Conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the Command Line Interface (CLI).

The big difference between Conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason Conda exists. Pip installs all Python package dependencies required, whether or not those conflict with other packages you installed previously.

So your working installation of, for example, Google Tensorflow, can suddenly stop working when you pip install a different package that needs a different version of the Numpy library. More insidiously, everything might still appear to work but now you get different results from your data science, or you are unable to reproduce the same results elsewhere because you didn't pip install in the same order.

Conda analyzes your current environment, everything you have installed, any version limitations you specify (e.g. you only want tensorflow>= 2.0) and figures out how to install compatible dependencies. Or it will tell you that what you want can't be done. Pip, by contrast, will just install the thing you wanted and any dependencies, even if that breaks other things.Open source packages can be individually installed from the Anaconda repository, Anaconda Cloud (anaconda.org), or your own private repository or mirror, using the conda install command. Anaconda Inc compiles and builds all the packages in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. You can also install anything on PyPI into a Conda environment using pip, and Conda knows what it has installed and what pip has installed. Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories.The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.




Anaconda Navigator is a desktop Graphical User Interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

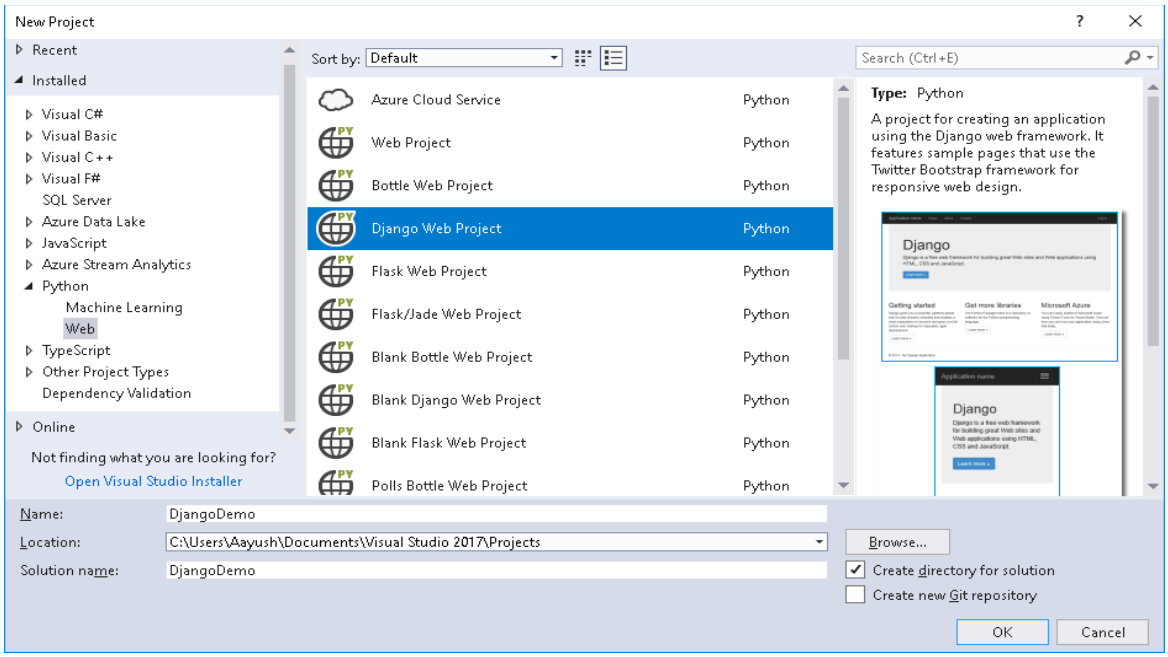
- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glueviz
- Orange
- Rstudio
- Visual Studio Code

Microsoft .NET is a set of Microsoft software technologies for rapidly building and integrating XML Web services, Microsoft Windows-based applications, and Web solutions. The .NET Framework is a language-neutral platform for writing programs that can easily and securely interoperate. There’s no language barrier with .NET: there are numerous languages available to

the developer including Managed C++, C#, Visual Basic and Java Script. The .NET framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate.

".NET" is also the collective name given to various software components built upon the .NET platform. These will be both products (Visual Studio.NET and Windows.NET Server, for instance) and services (like Passport, .NET My Services, and so on).

Microsoft VISUAL STUDIO is an Integrated Development Environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps.




Python is a powerful multi-purpose programming language created by Guido van Rossum. It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time. Python features are:

- Easy to code
- Free and Open Source
- Object-Oriented Language
- GUI Programming Support

- High-Level Language
- Extensible feature
- Python is Portable language
- Python is Integrated language
- Interpreted
- Large Standard Library
- Dynamically Typed Language

## PYTHON:

- Python is a powerful multi-purpose programming language created by Guido van Rossum.
- It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time.

Features Of Python :

1.Easy to code:
Python is high level programming language. Python is very easy to learn language as compared to other language like c, c#, java script, java etc. It is very easy to code in python language and anybody can learn python basic in few hours or days. It is also developer-friendly language.

2. Free and Open Source:
Python language is freely available at official website and you can download it from the given download link below click on the Download Python keyword.
Since, it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

3.Object-Oriented Language:
One of the key features of python is Object-Oriented programming. Python supports object oriented language and concepts of classes, objects encapsulation etc.

4. GUI Programming Support:
Graphical Users interfaces can be made using a module such as PyQt5, PyQt4, wxPython or Tk in python.
PyQt5 is the most popular option for creating graphical apps with Python.

5. High-Level Language:
Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

6.Extensible feature:
Python is a Extensible language. we can write our some python code into c or c++ language and also we can compile that code in c/c++ language.

7. Python is Portable language:
Python language is also a portable language. for example, if we have python code for windows and if we want to run this code on other platform such as Linux, Unix and Mac then we do not need to change it, we can run this code on any platform.

8. Python is Integrated language:
Python is also an Integrated language because we can easily integrated python with other language like c, c++ etc.

9. Interpreted Language:
Python is an Interpreted Language. because python code is executed line by line at a time. like other language c, c++, java etc there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called bytecode.

10. Large Standard Library
Python has a large standard library which provides rich set of module and functions so you do not have to write your own code for every single thing.There are many libraries present in python for such as regular expressions, unit-testing, web browsers etc.

11. Dynamically Typed Language:
Python is dynamically-typed language. That means the type (for example- int, double, long etc) for a variable is decided at run time not in advance.because of this feature we don't need to specify the type of variable.

APPLICATIONS OF PYTHON :

WEB APPLICATIONS

- You can create scalable Web Apps using frameworks and CMS (Content Management System) that are built on Python. Some of the popular platforms for creating Web Apps are:Django, Flask, Pyramid, Plone, Django CMS.
- Sites like Mozilla, Reddit, Instagram and PBS are written in Python.

SCIENTIFIC AND NUMERIC COMPUTING

- There are numerous libraries available in Python for scientific and numeric computing. There are libraries like:SciPy and NumPy that are used in general purpose computing.

And, there are specific libraries like: EarthPy for earth science, AstroPy for Astronomy and so on.

- Also, the language is heavily used in machine learning, data mining and deep learning.

CREATING SOFTWARE PROTOTYPES

- Python is slow compared to compiled languages like C++ and Java. It might not be a good choice if resources are limited and efficiency is a must.
- However, Python is a great language for creating prototypes. For example: You can use Pygame (library for creating games) to create your game's prototype first. If you like the prototype, you can use language like C++ to create the actual game.
-

GOOD LANGUAGE TO TEACH PROGRAMMING

- Python is used by many companies to teach programming to kids
- It is a good language with a lot of features and capabilities. Yet, it's one of the easiest language to learn because of its simple easy-to-use sy

# CHAPTER 4

# IMPLEMENTATION

## 4.1 GENERAL

Python is a program that was originally designed to simplify the implementation of numerical linear algebra routines. It has since grown into something much bigger, and it is used to implement numerical algorithms for a wide range of applications. The basic language used is very similar to standard linear algebra notation, but there are a few extensions that will likely cause you some problems at first.

## 4.2 CODE IMPLEMENTATION

### Back End Code:-

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#list of useful imports that  I will use
%matplotlib inline
import os

import matplotlib.pyplot as plt
import pandas as pd

import numpy as np

import seaborn as sns
import random
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
```

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
# Imputer
#from sklearn.preprocessing import Imputer

from sklearn.impute import SimpleImputer
#imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
from sklearn_pandas import DataFrameMapper, CategoricalImputer



# Run this cell to mount your Google Drive.
from google.colab import drive
drive.mount('/content/drive')


data = pd.read_csv('/content/drive/My Drive/spam_ham_dataset.csv
')
data.head()

data = data.drop(columns=['Unnamed: 0'])
data.head()
data.shape

#Check the data
data.info()

#Check the missing values in the data
data.isnull().sum()

data['label'].value_counts()
```

```
# Prepare Data
df = data.groupby('label').size().reset_index(name='counts')
n = df['label'].unique().__len__()+1
all_colors = list(plt.cm.colors.cnames.keys())
random.seed(100)
c = random.choices(all_colors, k=n)

# Plot Bars
plt.figure(figsize=(6,6), dpi= 80)
plt.bar(df['label'], df['counts'], color=c, width=.5)
for i, val in enumerate(df['counts'].values):
    plt.text(i, val, float(val), horizontalalignment='center', v
erticalalignment='bottom', fontdict={'fontweight':500, 'size':12
})

# Decoration
plt.gca().set_xticklabels(df['label'], rotation=60, horizontalal
ignment= 'right')

plt.ylim(0, 4000)
plt.show()



from sklearn.utils import resample
# Separate majority and minority classes
df_majority = data[data['label']== 'ham']
df_minority = data[data['label']== 'spam']

# Downsample majority class and upsample the minority class
df_minority_upsampled = resample(df_minority, replace=True,n_sam
ples=2500,random_state=123)
```

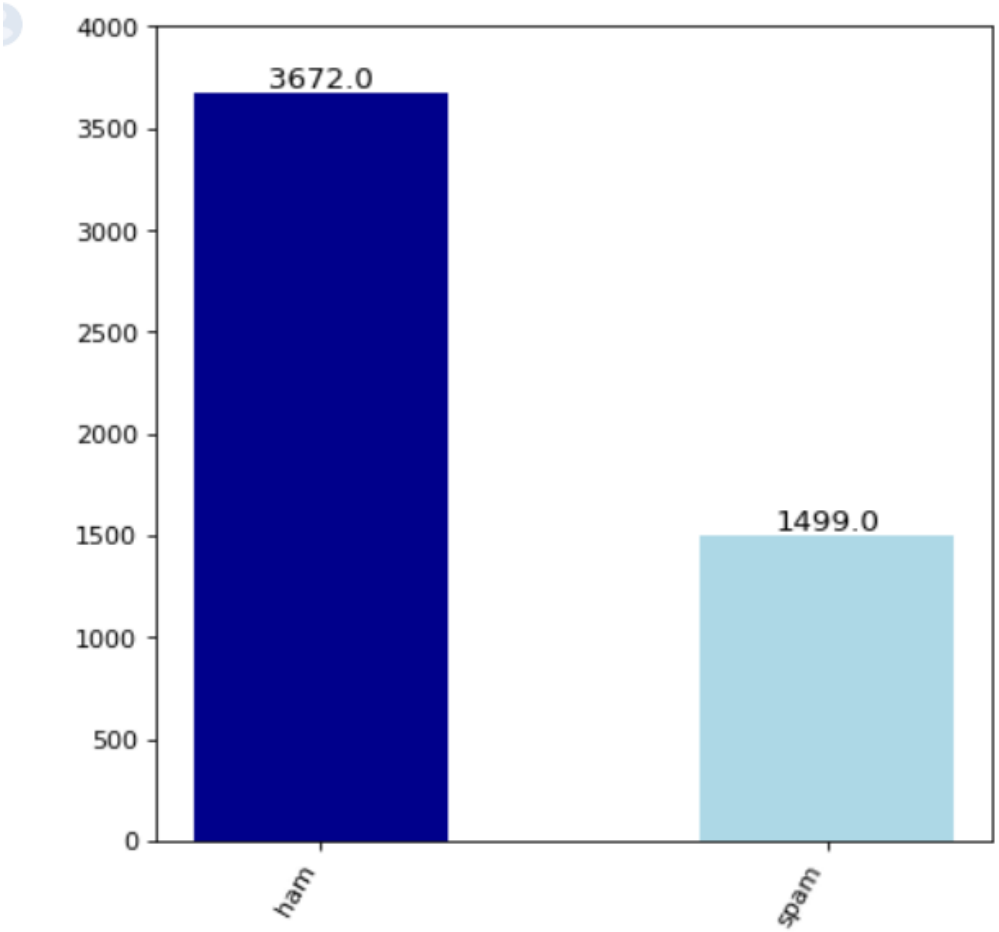
```
df_majority_downsampled = resample(df_majority, replace=False,n_
samples=2500,random_state=123)

# Combine minority class with downsampled majority class
df_upsampled = pd.concat([df_minority_upsampled, df_majority_dow
nsampled])

# Display new class counts
df_upsampled['label'].value_counts()


# shuffle the DataFrame rows
data= df_upsampled.sample(frac = 1)


# Prepare Data
df = data.groupby('label').size().reset_index(name='counts')
n = df['label'].unique().__len__()+1
all_colors = list(plt.cm.colors.cnames.keys())
random.seed(100)
c = random.choices(all_colors, k=n)

# Plot Bars
plt.figure(figsize=(6,6), dpi= 80)
plt.bar(df['label'], df['counts'], color=c, width=.5)
for i, val in enumerate(df['counts'].values):
    plt.text(i, val, float(val), horizontalalignment='center', v
erticalalignment='bottom', fontdict={'fontweight':500, 'size':12
})

# Decoration
```

```
plt.gca().set_xticklabels(df['label'], rotation=60, horizontalal
ignment= 'right')

plt.ylim(0, 3000)
plt.show()


# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase


# we are removing the words from the stop words list: 'no', 'nor
', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'our
selves', 'you', "you're", "you've",\
```

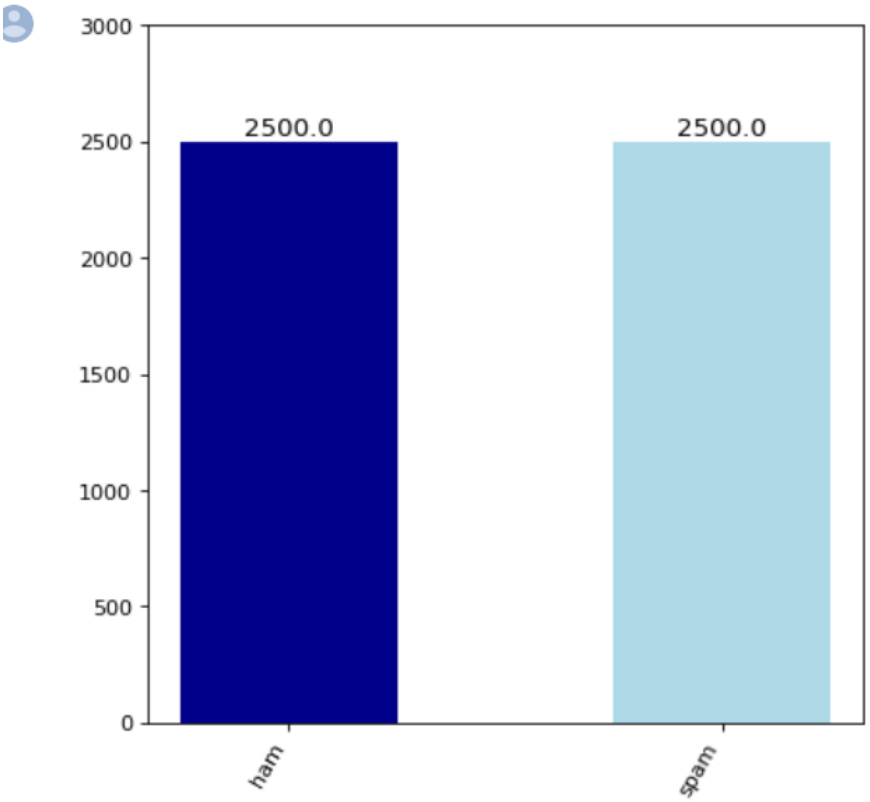
```
            "you'll", "you'd", 'your', 'yours', 'yourself', 'you
rselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it'
s", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'who
m', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'bei
ng', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if'
, 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'betw
een', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in',
'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'w
hy', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'sam
e', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'sh
ould', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "could
n't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn
', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'sh
ouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]


data['text'].head(5)
print("printing some random reviews")
print(9, data['text'].values[9])
```

```
print(34, data['text'].values[34])
print(147, data['text'].values[147])

# Combining all the above stundents
from tqdm import tqdm
def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not
 in stopwords)
        preprocessed_text.append(sent.lower().strip())
    return preprocessed_text


preprocessed_text = preprocess_text(data['text'].values)

print("printing some random reviews")
print(9, preprocessed_text[9])
print(34, preprocessed_text[34])
print(147, preprocessed_text[147])
```

## Back End Screen Shots:

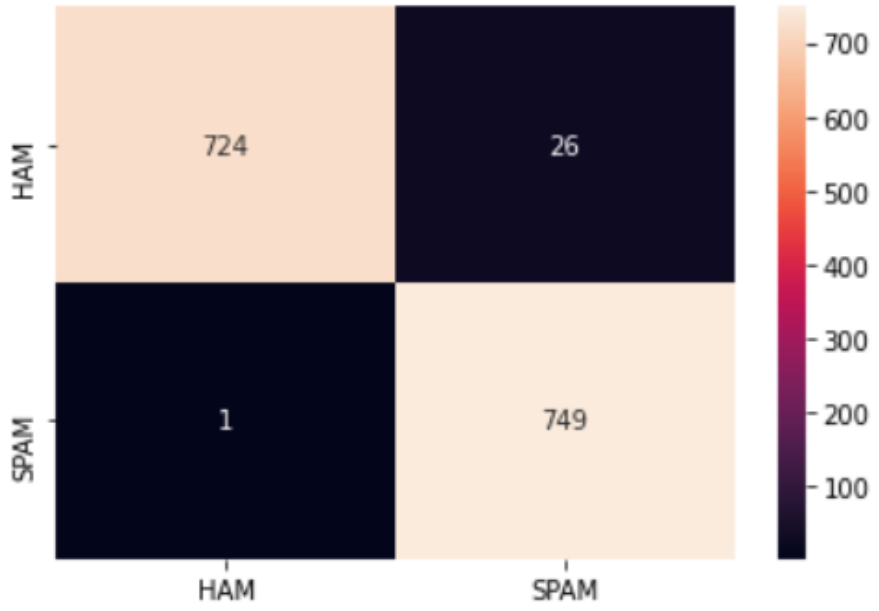



Data points based on class label before balancing



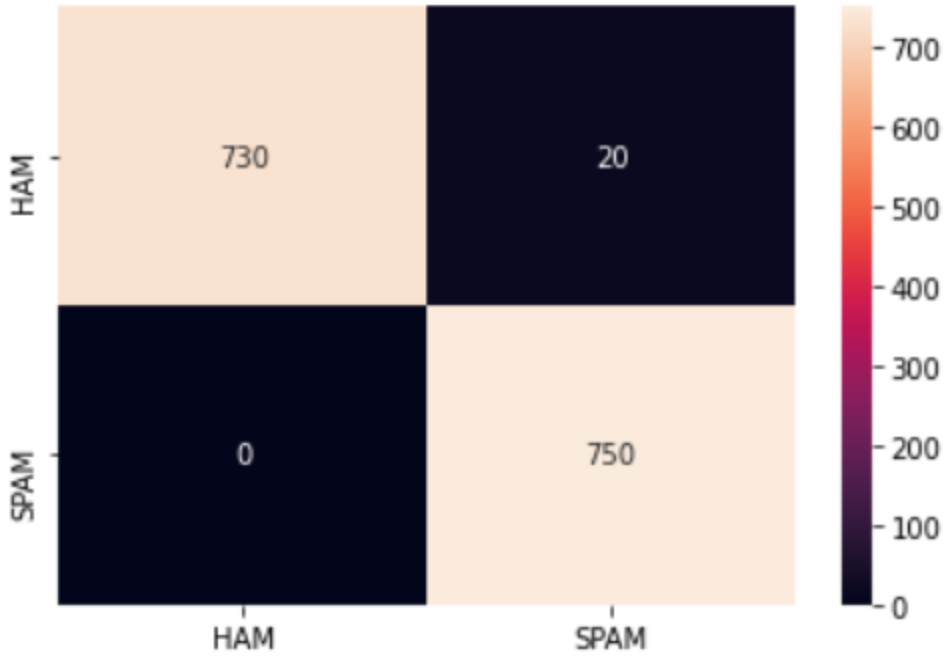

Data points based on class label after balancing

```
AUC on Test data is 0.982
AUC on Train data is 0.9988571428571429
----------------------------
```



| | Text | original_Classlabel | predicted_classlebel |
|---|---|---|---|
| 0 | subject neon discussion november 1 material up... | HAM | HAM |
| 1 | subject 30 seconds refinance hello sent email ... | SPAM | SPAM |
| 2 | subject vicodin no script neeeded place ggo ur... | SPAM | SPAM |
| 3 | subject fw duke energy trading marketing l l c... | HAM | HAM |
| 4 | subject special offers various today special o... | SPAM | SPAM |
| 5 | subject deal 34342 daren thu asked extend deal... | HAM | HAM |
| 6 | subject witch htmlbody sir brbr resently sent ... | SPAM | SPAM |
| 7 | subject 98 1601 please extend sale lockhart so... | HAM | HAM |
| 8 | subject april nominations shell deer park forw... | HAM | HAM |
| 9 | subject revised july 2000 availabilities revis... | HAM | HAM |
| 10 | subject sale shoreline fyi forwarded stella l ... | HAM | HAM |
| 11 | subject meter 980070 daren meter recorded flow... | HAM | HAM |
| 12 | subject new product cialis soft tabs hi new pr... | SPAM | SPAM |
| 13 | subject want rolex watch | SPAM | SPAM |
| 14 | subject unify unix team working unix server un... | HAM | HAM |
| 15 | subject italian rolex throw away prices derm s... | SPAM | SPAM |
| 16 | subject tenaska iv 11 01 darren time please re... | HAM | HAM |
| 17 | subject hr generalist group norma villarreal h... | HAM | SPAM |
| 18 | subject meet lonely fun girls want newest plac... | SPAM | SPAM |
| 19 | subject august 2000 iferc sds noms attached au... | HAM | HAM |

```
Accuracy on Test data is 0.9866666666666667
Accuracy on Train data is 1.0
----------------------------
```




## CHAPTER 5
## CONCLUSION AND REFERENCES

This survey paper elaborates different Existing Spam Filtering system through Machine learning techniques by exploring several methods, concluding the overview of several Spam Filtering techniques and summarizing the accuracy of different proposed approach regarding several parameters. Moreover, all the existing methods are effective for email spam filtering. Some have effective outcome and some are trying to implement another process for increasing their accuracy rate. Though all are effective but still now spam filtering system have some lacking which are the major concern for researchers and they are trying to generate next generation spam filtering process which have the ability to consider large number of multimedia data and filter the spam email more prominently.

## REFERENCES:


1. SuryawanshiShubhangi, GoswamiAnurag and PatilPramod, Email Spam Detection: An Empirical Comparative Study of Different ML and Ensemble Classifiers, pp. 69-74, 2019.

2. A. Karim, S. Azam, B. Shanmugam, K. Krishnan and M. Alazab, "A Comprehensive Survey for Intelligent Spam Email Detection", IEEE Access, vol. 7, pp. 168261-168295, 2019.

3. K. Agarwal and T. Kumar, "Email Spam Detection Using Integrated Approach of Naïve Bayes and Particle Swarm Optimization", 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), pp. 685-690, 2018.

4. HarisinghaneyAnirudh, Aman Dixit Saurabh Gupta and Anuja Arora, "Text and image-based spam email classification using KNN Naïve Bayes and Reverse DBSCAN algorithm", Optimization Reliabilty and Information Technology (ICROIT), pp. 153-155, 2014.

5. MohamadMasurah and Ali Selamat, "An evaluation on the efficiency of hybrid feature selection in spam email classification", Computer Communications and Control Technology (I4CT) 2015 International Conference on, pp. 227-231, 2015.

6. Shradhanjali Prof and ToranVerma, "E-Mail Spam Detection and Classification Using SVM and Feature Extraction", International Jouranl Of Advance Reasearch Ideas and Innovation In Technology, 2017, ISSN 2454-132X.

7. W.A Awad and S.M ELseuofi, Machine Learning Methods for Spam E-Mail Classification.International Journal of Computer Science & Information Technology, 2011.

8. A. K. Ameen and B. Kaya, "Spam detection in online social networks by deep learning", 2018 International Conference on Artificial Intelligence and Data Processing (IDAP), pp. 1-4, 2018.

9. D.D. Diren, S. Boran, I.H. Selvi and T. Hatipoglu, Root Cause Detection with an Ensemble Machine Learning Approach in the Multivariate Manufacturing Process, 2019.

10. TasnimKabir, AbidaSanjanaShemonti and AtifHasan Rahman, "Notice of Violation of IEEE Publication Principles: Species Identification Using Partial DNA Sequence: A Machine Learning Approach", 2018 IEEE 18th International Conference on Bioinformatics and Bioengineering (BIBE), 2018.