NAAN MUDHALVAN PROJECT

OPTIMIZING SPAM FILTERING WITH MACHINE LEARNING

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

TO THE

THIRUVALLUVAR UNIVERSITY,SERKKADU,VELLLORE-632115

BY

M.NANDHINI ---35620U18025

D.SANGEETHA ---35620U18039

V.SATHYA ---35620U18041

P.VIGNESH ---35620U18049

APRIL-2023

GOVERNMENT ARTS AND SCIENCE COLLEGE,

ARAKKONAM-631051

(AFFILIATED TO THIRUVALLUVAR UNIVERSITY)

GUIDED AND HEAD OF DEPARTMENT

Dr. S. Selvakani

	* Scaling The Data
5.	MILESTONE 4: Model Building *Training the model in multiple Algorithms *Decision Tree Model *Random Forest Model * Naïve Bayes Model *ANN Model * Testing The Model
6.	MILESTONE 5: Performance Testing & Hyperparameter Tuning *Testing Model With Multiple Evaluation Metrics *Compare The Model *Comparing Model Accuracy Before & after applying hyperparameter Tuning
7.	MILESTONE 6: Model Deployment *Save the Best Model *Integrate with Web Framework * Building html Pages * Building Python code * Run the Web Application

INTRODUCTION

Optimizing Spam Filtering with Machine Learning Over recent years, as the popularity of mobile phone devices has increased, Short Message Service (SMS) has grown into a multi-billion dollar 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. Due to Spam SMS, Mobile service providers suffer from some sort of financial problems as well as it reduces calling time for users. Unfortunately, if the user accesses such Spam SMS they may face the problem of virus or malware. When SMS arrives at mobile it will disturb mobile user privacy and concentration. It may lead to frustration for the user. So Spam SMS is one of the major issues in the wireless communication world and it grows day by day. To avoid such Spam SMS people use white and black list of numbers. But this technique is not adequate to completely avoid Spam SMS. To tackle this problem it is needful to use a smarter technique which correctly identifies Spam SMS. Natural language processing technique is useful for Spam SMS identification. It analyses text content and finds patterns which are used to identify Spam and Non-Spam SMS.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

A business requirement for an SMS spam classification system would include the ability to accurately identify and flag spam messages, protect customers from unwanted or harmful messages, and comply with industry regulations and laws regarding spam messaging. Additionally, the system should be able to handle a high volume of messages, integrate with existing systems and databases, and provide reporting and analysis capabilities to track performance and improve the system over time. The system should also have an easy-to-use interface and be easy to maintain and update.

Activity 3: Literature Survey (Student Will Write)

project would involve researching and analysing existing studies, papers, and articles on the topic to gain a thorough understanding of the current state of SMS spam classification and to identify potential areas for improvement and future research. The survey would include looking at different methods and techniques used for identifying and flagging spam messages, such as machine learning algorithms, natural language processing, and rule-based systems. It would also involve evaluating the performance and effectiveness of these methods, as well as their limitations and challenges. Additionally, the literature survey would review the current state of SMS spam and trends in the industry, as well as any existing laws and regulations related to spam messaging. The survey would also investigate the datasets and feature representations used in previous studies, which would help to determine the best approach for the current project. Furthermore, It would be important to check the pre-processing techniques used in the research to understand how to properly clean and prepare the data for the classifier .

Activity 4: Social or Business Impact.

Social Impact:- it can help protect individuals from unwanted and potentially harmful messages. Spam messages can include phishing attempts, scams, and fraud, which can have serious financial and personal consequences for recipients. By accurately identifying and flagging spam messages, the system can help prevent these types of attacks and protect individuals from falling victim to them. Business Model/Impact:- it can help protect their customers and improve their reputation. Spam messages can harm a business's reputation and lead to customer complaints and lost business. By accurately identifying and flagging spam messages, the system can help protect businesses and improve their customer's trust.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Collect the dataset There are many popular open sources for collecting the data.

Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset As the dataset is downloaded.

Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_sp
lit
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
```

```
df = pd.read_csv("/content/spam.csv",encoding="la
tin")
```

```
df.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

● Handling missing values

● Handling categorical data

● Handling Imbalance Data Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps. Activity 2.1: Handling missing values

● Let’s find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   v1           5572 non-null   object
 1   v2           5572 non-null   object
 2   Unnamed: 2   50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

● For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step

```
df.isna().sum()
```

```
 v1  0
 v2  0
 Unnamed: 2  5522
 Unnamed: 3  5560
 Unnamed: 4  5566
 dtype: int64
```

● From the above code of analysis, we can infer that columns such as V1 and v2 are not having missing columns,unnamed columns are not required for analysis

● Renaming the columns according the requirement

```
df.rename({"v1":"label","v2":"text"},inplace=True
,axis=1)
```

```
df.tail()
```

	label	text	Unnamed: 2	Unnamed: 3	Unnamed: 4
5567	spam	This is the 2nd time we have tried 2 contact u...	NaN	NaN	NaN
5568	ham	Will Ì_ b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other s...	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd...	NaN	NaN	NaN
5571	ham	Rofl. Its true to its name	NaN	NaN	NaN

Activity 2.2: Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using manual encoding with the help of list comprehension.

● In our project,we have text column so we will be using natural language processing for processing the data. Output column is having classes we Converting into 0 and 1 by applying label encoding

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['label'] = le.fit_transform(df['label'])
```

Activity 2.3: Handling Imbalance Data

Data Balancing is one of the most important step, which need to be performed for classification models, because when we train our model on imbalanced dataset ,we will get biassed results, which means our model is able to predict only one class element

For Balancing the data we are using the SMOTE Method.

SMOTE: Synthetic minority over sampling technique, which will create new synthetic data points for under class as per the requirements given by us using KNN method.

```
print("Before OverSampling, counts of label '1': {}".format(sum(y_t
rain == 1)))
print("Before OverSampling, counts of label '0': {}  \n".format(sum
(y_train == 0)))

from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
x_train_res, y_train_res = sm.fit_resample(x_train, y_train.ravel()
)

print('After OverSampling, the shape of train_x: {}'.format(x_train
_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_tr
ain_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_tr
ain == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_tr
ain == 0)))
```

```
Before OverSampling, counts of label '1': 581
Before OverSampling, counts of label '0': 3876

After OverSampling, the shape of train_x: (7752,
8194)
After OverSampling, the shape of train_y: (7752,)

After OverSampling, counts of label '1': 581
After OverSampling, counts of label '0': 3876
```

From the above picture, we can infer that ,previously our dataset had 581 class 1, and 3876 class 0 items, after applying smote technique on the dataset the size has been changed for minority class.

Activity 2.3: Cleaning the text data

```
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

import re
corpus = []
length = len(df)

for i in range(0,length):
  text = re.sub("^a-Za-Z0-9]"," " ,df["text"][i])
  text = text.lower()
  text = text.split()
  pe = PorterStemmer()
  stopword = stopwords.words("english")
  text = [pe.stem(word) for word in text if not word
in set(stopword)]
  text = " ".join(text)
  corpus.append(text)
```

Text pre-processing includes

- Removing punctuation from the text using regular expression library
- Converting the sentence into lower case
- Tokenization – splitting the sentence into words
- Removing stop words from the data
- Stemming – stemming is the process of brining all the words into base form

```
from sklearn.feature_extraction.text import CountVect
orizer
cv = CountVectorizer(max_features=35000)
x =cv.fit_transform(corpus).toarray()
```

- After applying all the above functions, we will get corpus
- Converting the corpus into Document Term matrix using Count vectorizer

```
import pickle
pickle.dump(cv, open('cv1.pkl','wb'))
```

Saving the count vectorizer function for future use.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
df.describe()
```

```
             label
count  5572.000000
 mean     0.134063
  std     0.340751
  min     0.000000
  25%     0.000000
  50%     0.000000
  75%     0.000000
```

	label
max	1.000000

```
df.shape
```

```
(5572, 5)
```

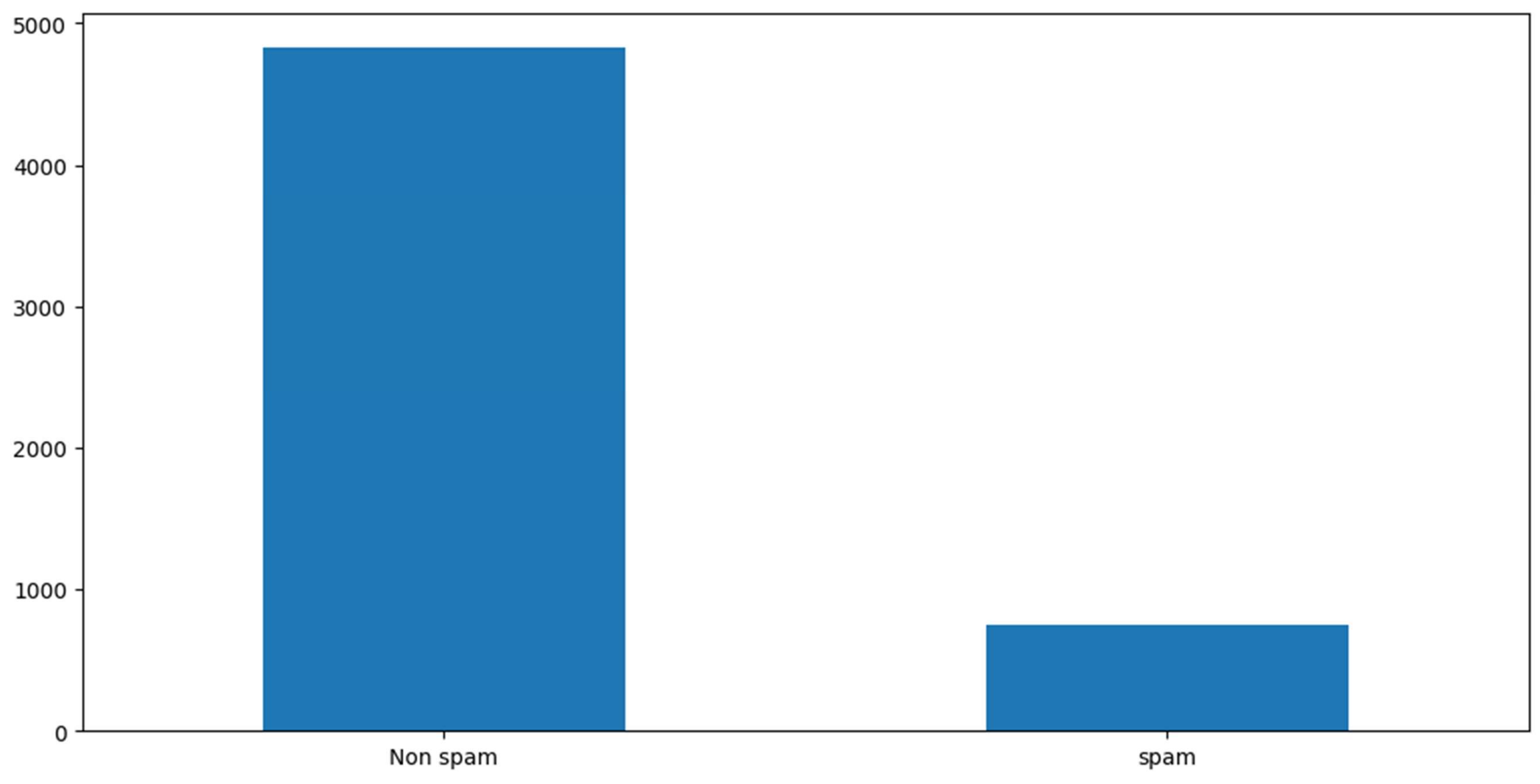
Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.

● The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

```
df["label"].value_counts().plot(kind="bar",figsiz
e=(12,6))
 plt.xticks(np.arange(2),  ('Non spam', 'spam'),r
otation=0);
```



- In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features.

Countplot:-

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for barplot() , so you can compare counts across nested variables.

From the graph we can infer that , more data belongs class 0 than class 1

Scaling the Data

Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction

Models such as KNN, Logistic regression need scaled data, as they follow distance based method and Gradient Descent concept.

We will perform scaling only on the input values.Once the dataset is scaled, it will be converted into an array and we need to convert it back to a dataframe.

Splitting data into train and test

Now let’s split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
 from sklearn.model_selection import train_test_split
 x_train, x_test, y_train, y_test = train_test_split(
x, y, test_size=0.20, random_state=0)
```

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

Activity 1.1: Decision tree model A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
 model.fit(x_train_res, y_train_res)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

Activity 1.2: Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
 from sklearn.ensemble import  RandomForestClassi
fier

model = RandomForestClassifier()
 model.fit(x_train_res, y_train_res)
```

```
RandomForestClassifier
RandomForestClassifier()
```

Activity 1.3: Naïve Bayes model

A function named MultinomialNB is created and train and test data are passed as the parameters. Inside the function, MultinomialNB algorithm is initialised and training data is passed to the model with .fit() function.

Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
from sklearn.naive_bayes import MultinomialNB
 model = MultinomialNB()


 model.fit(x_train_res, y_train_res)
```

```
▾ MultinomialNB
MultinomialNB()
```

Activity 1.5: ANN model

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs.

```
 generator = model.fit(x_train_res,y_train_res,ep
ochs=10,steps_per_epoch=len(x_train_res)//64)
```

```
Epoch 1/10
121/121 [==============================] - 162s 1s/step - loss: 0.1224 -
accuracy: 0.9628
Epoch 2/10
121/121 [==============================] - 161s 1s/step - loss: 0.0163 -
accuracy: 0.9962
Epoch 3/10
121/121 [==============================] - 170s 1s/step - loss: 0.0097 -
accuracy: 0.9981
```

```
Epoch 4/10
121/121 [==============================] - 162s 1s/step - loss: 0.0083 -
accuracy: 0.9982
Epoch 5/10
121/121 [==============================] - 163s 1s/step - loss: 0.0078 -
accuracy: 0.9983
Epoch 6/10
121/121 [==============================] - 162s 1s/step - loss: 0.0075 -
accuracy: 0.9985
Epoch 7/10
121/121 [==============================] - 175s 1s/step - loss: 0.0077 -
accuracy: 0.9983
Epoch 8/10
121/121 [==============================] - 164s 1s/step - loss: 0.0068 -
accuracy: 0.9986
Epoch 9/10
121/121 [==============================] - 163s 1s/step - loss: 0.0061 -
accuracy: 0.9988
Epoch 10/10
111/121 [==========================>...] - ETA: 13s - loss: 0.0066 - accuracy:
0.9986
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure
that your dataset or generator can generate at least `steps_per_epoch * epochs`
batches (in this case, 1210 batches). You may need to use the repeat() function
when building your dataset.
121/121 [==============================] - 147s 1s/step - loss: 0.0066 -
accuracy: 0.9986
```

Activity 2: Testing the model

```
y_pred=model.predict(x_test)
y_pred
```

```
35/35 [==============================] - 7s 198ms/step
array([[1.7400573e-09],
    [5.8196643e-03],
    [4.6225469e-13],
    ...,
    [9.0467489e-05],
    [4.7840415e-13],
    [1.5554736e-15]], dtype=float32)
```

```
y_test
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=uint8)
```

```
 from sklearn.metrics import confusion_matrix,acc
uracy_score
 cm = confusion_matrix(y_test, y_pr)
 score = accuracy_score(y_test,y_pr)
 print(cm)
 print('Accuracy Score Is:- ' ,score*100)
```

```
[[933  16]
 [ 15 151]]
Accuracy Score Is:-  97.21973094170404
```

In ANN we first have to save the model to the test the inputs

This code defines a function named "new_review" which takes in a new_review as an input. The function then converts the input new_review from a list to a numpy array. It reshapes the new_review array as it contains only one record. Then, it applies feature scaling to the reshaped new_review array using a scaler object 'sc' that should have been previously defined and fitted. Finally, the function returns the prediction of the classifier on the scaled new_review

```
 def new_review(new_review):
    new_review = new_review
    new_review = re.sub('[^a-zA-
Z]', ' ', new_review)
    new_review = new_review.lower()
    new_review = new_review.split()
    ps = PorterStemmer()
    all_stopwords = stopwords.words('english')
    all_stopwords.remove('not')
    new_review = [ps.stem(word) for word in new_review if not word in   set(all_stopwords)]
    new_review = ' '.join(new_review)
    new_corpus = [new_review]
    new_X_test = cv.transform(new_corpus).toarray()
    new_y_pred = model.predict(new_X_test)
    print(new_y_pred)
    new_X_pred = np.where(new_y_pred>0.5,1,0)
    return new_review
 new_review = new_review(str(input("Enter new review...")))
```

```
1/1 [==============================] - 0s
64ms/step
[[0.9919058]]
```

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

Activity 1.1: Compare the model

For comparing the above four models, the compareModel function is defined

```
 from sklearn.metrics import confusion_matrix,acc
uracy_score
 cm=confusion_matrix(y_test,y_pr)
 score = accuracy_score(y_test,y_pr)
 print(cm)
 print('Accuracy Score Is Naive Bayes:- ' ,score*
100)
```

```
[[933  16]
 [ 15 151]]
Accuracy Score Is Naive Bayes:-  97.21973094170404
```

Activity 2: Comparing model accuracy before & after applying hyperparameter tuning

Evaluating performance of the model From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 5 folds). Our model is performing well. So, we are saving the model by model.save("model.h5")

Milestone 6: Model Deployment

Activity 1: Save the best model Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

Activity 2: Integrate with Web Framework In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks ● Building HTML Pages

● Building server side script

● Run the web application

Activity 2.1: Building Html Pages: For this project create two HTML files namely

● index.html

● spam.html

● result.html

and save them in the templates folder.

Activity 2.2: Build Python code: Import the libraries

```
from flask import Flask, render_template, request
import pickle
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from tensorflow.keras.models import load_model


loaded_model = load_model('spam.h5')
cv = pickle.load(open('cv1.pkl','rb'))
app = Flask(__name__)
```

```
loaded_model = load_model('spam.h5')
cv = pickle.load(open('cv1.pkl','rb'))
app = Flask(__name__)
```

```
@app.route('/predict',methods=['POST'])
def predict():
    if request.method == 'POST':
       message = request.form['message']
       data = message

    new_review = str(data)
    print(new_review)
    new_review = re.sub('[^a-zA-Z]', ' ',new_review)
    new_review = new_review.lower()
    new_review = new_review.split()
    ps = PorterStemmer()
    all_stopwords = stopwords.words('english')
    all_stopwords.remove('not')
    new_review = [ps.stem(word) for word in new_review if not word in   set(all_stopwords)]
    new_review = ' '.join(new_review)
    new_corpus = [new_review]
    new_X_test = cv.transform(new_corpus).toarray()
    new_y_pred = model.predict(new_X_test)
    print(new_y_pred)
    new_X_pred = np.where(new_y_pred>0.5,1,0)
    print(new_X_pred)
    if new_review[0][0]==1:
       return render_template('result.html', prediction="Spam")
    else :
       return render_template('result.html', prediction="Not a Spam")
```