**ML ALGORITHMS IN PYTHON FOR**

**HAM AND SPAM MESSAGE CLASSIFICATION**

**PROBLEM STATEMENT:**

The battle against spam messages in electronic communication has intensified in the digital age, necessitating robust solutions for accurate message classification. This research presents a comprehensive study on the application of Machine Learning (ML) algorithms in Python for the classification of messages into two categories: "ham" (legitimate) and "spam" (unsolicited or malicious).

The study utilizes a diverse dataset of text messages, incorporating various linguistic features and message characteristics. Several ML algorithms, including Naïve Bayes, Support Vector Machines (SVM), and Decision Trees, are implemented to build classification models. Python's libraries, such as scikit-learn and NLTK, are employed for data preprocessing, feature extraction, and model development.

The performance of these models is rigorously evaluated using common metrics like precision, recall, F1-score, and accuracy. The research highlights the importance of feature selection, hyper parameter tuning, and cross-validation techniques in optimizing classification accuracy.

The results of this study demonstrate the effectiveness of ML algorithms in Python for the accurate classification of ham and spam messages. The developed models offer practical implications for email services, messaging apps, and cybersecurity to protect users from unwanted and potentially harmful messages. This research contributes to the ongoing efforts to enhance the efficiency of spam detection systems and improve the overall user experience in digital communication.

**DESIGN THINKING PROCESS:**

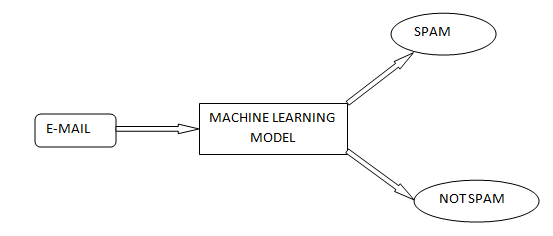
****

Figure: EMAIL SPAM CLASSIFIER USING ML MODELS

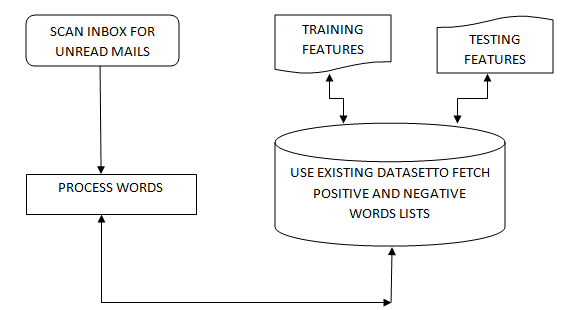
****

Figure: WORD PROCESSING AND CLASSIFICATION FOR

TRAINING USING EXISTING DATASET

**PROJECT PHASE DEVELOPMENT:**

* Preprocessing of Dataset
* Splitting the Datasets
* Building ML models such as Naïve Baised, Random Forest
* Visualization with Matplotlib and Seaborn
* Prediction

**DATASET:**

We will treat ham and spam message classification as a supervised machine learning problem. In a supervised machine learning problem, the inputs and the corresponding outputs are available during the algorithm training phase. During the training phase, the machine learning algorithm statistically learns to find the relationship between input texts and output labels. While testing, inputs are fed to the trained machine learning algorithm which then predicts the expected outputs without knowing the actual outputs.

For supervised ham and spam message classification, we need a dataset that contains both ham and spam messages along with labels that specify whether a message is a ham or spam.

All the Machine Learning Algorithms works on two stages:-

* Training Stage.
* Testing Stage.

So In the Training Stage Naive Bayes creates a Lookup table in which they store all the possibility of probability which we are going to use in the Algorithm for predicting the result.

And In the testing phase let suppose you have given a test point to the algorithm to predict the result, they fetch the values from the lookup table in which they store all the possibility of probability and use that value to predict the result.

**PROGRAM CODE FOR LOADING AND PREPROCESSING DATASET:**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

df = pd.read\_csv("/content/drive/MyDrive/Mail Spam-NLP/spam.csv", encoding="ISO-8859-1")

df.head(10)

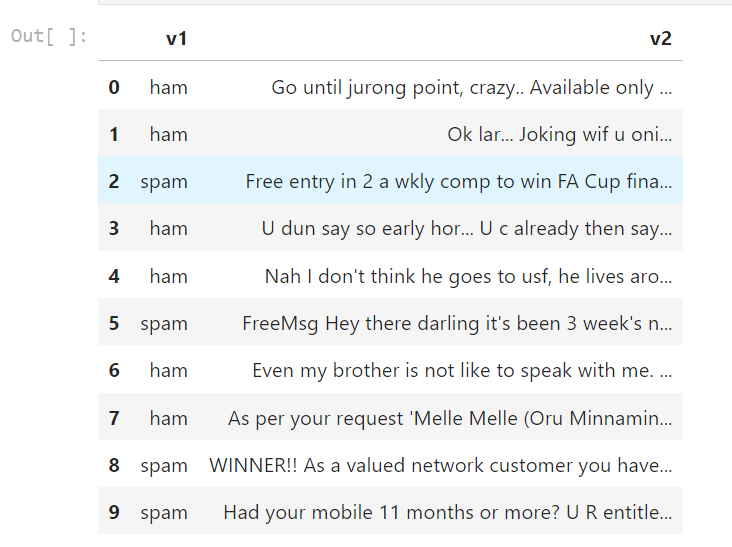


#Drop empty columns

cols = [2,3,4]

df.drop(df.columns[cols],axis=1,inplace=True)

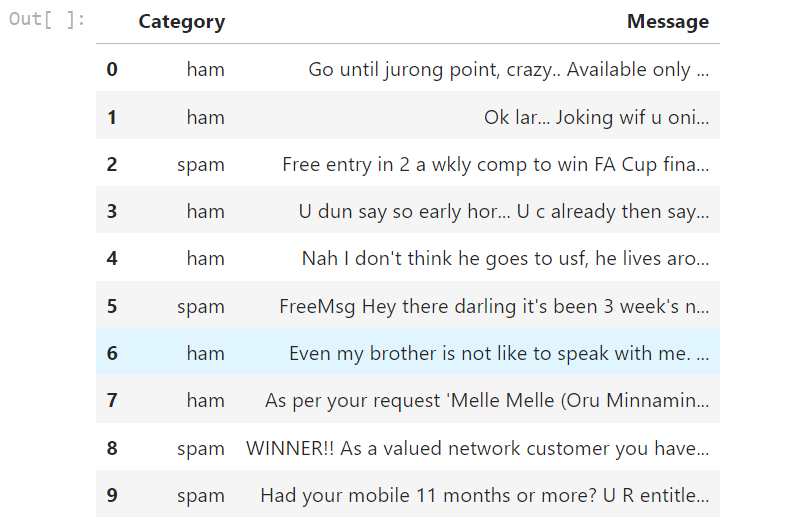
df.head(10)



#Rename columns as category and message

df.rename(columns = {'v1':'Category', 'v2':'Message'}, inplace = True)

df.head(10)



print(f'Dataset consist of {df.shape[0]} E-Mails.')

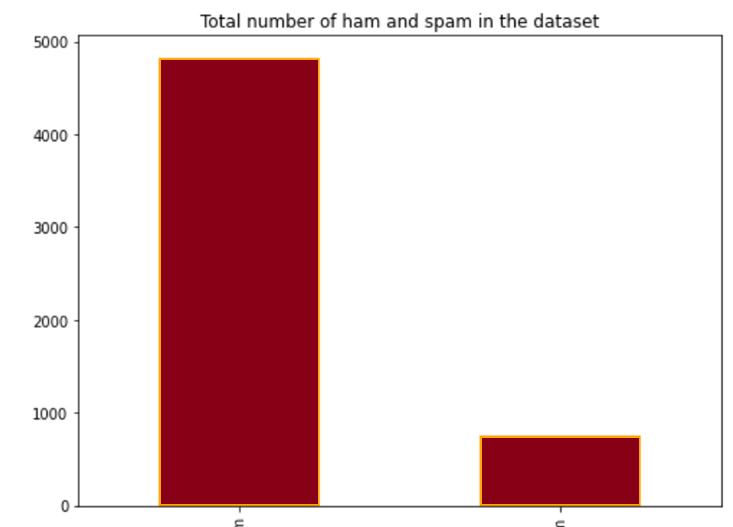
df['Category'].value\_counts()

plt.figure(figsize=(8,6))

df['Category'].value\_counts().plot.bar(color = ["orange","orange"])

plt.title('Total number of ham and spam in the dataset')

plt.show()



**PROGRAM CODE FOR FEATURE EXTRACTION AND CLASSIFICATION (NAÏVE BAISED & RANDOM FOREST):**

**(Choice of machine learning algorithm, model training, and evaluation metrics)**

# WordCloud

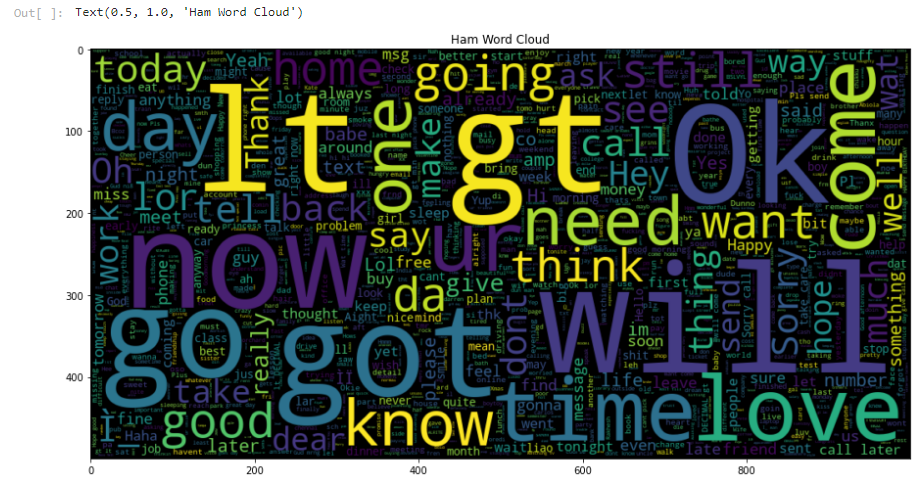
from wordcloud import WordCloud

plt.figure(figsize = (15,15))

wc = WordCloud(max\_words = 2000 , width = 1000 , height = 500).generate(" ".join(df[df.Category =="ham" ].Message))

plt.imshow(wc , interpolation = 'bilinear')

plt.title("Ham Word Cloud")



plt.figure(figsize = (15,15))

wc = WordCloud(max\_words = 2000 , width = 1000 , height = 500).generate(" ".join(df[df.Category =="spam" ].Message))

plt.imshow(wc , interpolation = 'bilinear')

plt.title("Spam Word Cloud")

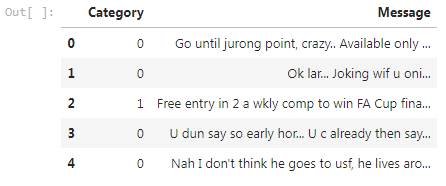


Test-Train Split

#0: Ham, 1: Spam

df['Category']=df['Category'].apply(lambda x: 1 if x=='spam' else 0)

df.head()



X=df['Message']

Y=df['Category']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,Y)

# Naive Baised Model

#Defineing Naive Baised

clf\_NaiveBaised= Pipeline([

('vectorizer', CountVectorizer()),

('nd', MultinomialNB())

])

#Fiting the algorithm

clf\_NaiveBaised.fit(X\_train,y\_train)



#Make prediction on X\_test

y\_pred\_NB=clf\_NaiveBaised.predict(X\_test)



# Random Forest Model

clf\_rf= Pipeline([

('vectorizer', CountVectorizer()),

('rf', RandomForestClassifier(n\_estimators=100))

])

clf\_rf.fit(X\_train,y\_train)



y\_pred\_RF=clf\_rf.predict(X\_test)

rf\_acc=accuracy\_score(y\_test,y\_pred\_RF)

rf\_acc

