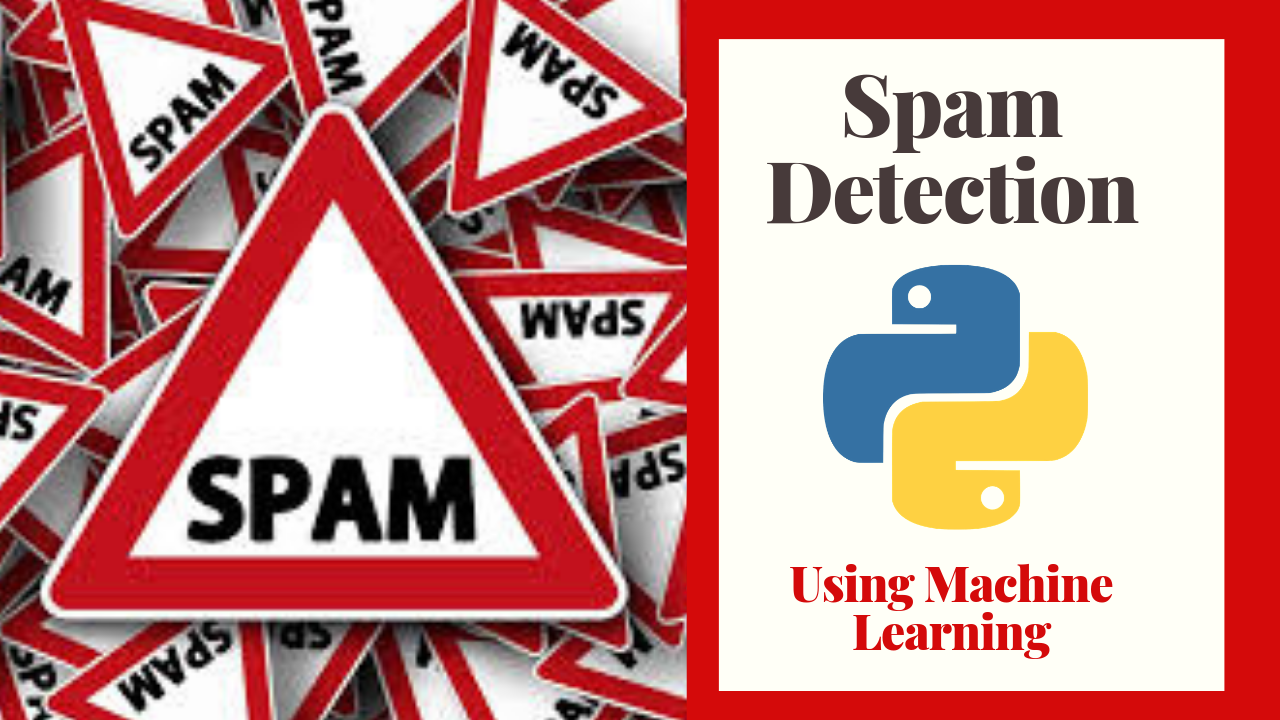
**PROJECT : BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER**

**au620121243002 : ABIRAMI MURUGAN**

**PHASE-5 SUBMISSION**

In this section we will document the complete project and prepare it for submission



**INTRODUCTION:**

In today's digital world, where spam messages inundate our inboxes and pose cybersecurity threats, the development of a Smarter AI-Powered spam classifier is paramount. Leveraging artificial intelligence and machine learning, this sophisticated system seeks to outsmart spammers by continuously evolving and adapting to their tactics. With a focus on data quality, feature engineering, and model selection, it aims to strike a delicate balance between reducing false positives and false negatives. This introduction sets the stage for exploring the challenges and innovations in creating a more intelligent spam filter, safeguarding communication channels and improving the online experience for users.

**OVERVIEW**

|  |  |
| --- | --- |
| Group | Artificial Intelligence - 2 |
| Project Title | Building a Smarter AI-Powered Spam Classifier |
| Language Used | Python |
| IDE | Google Collab / Jupyter Notebook |
| Python Version | Python 3.7 |
| Database | Required |
| Abstract | It is an ML – based project which involves ML steps like Data Collection, Exploratory Data Analysis, Model Training and Evaluation and also Performance Evaluation. |

**SIGNIFICANCE OF SPAM CLASSIFIER:**

A spam classifier plays a crucial role in filtering unwanted or unsolicited messages, particularly in emails. Its significance lies in:

1. **Efficient Filtering**: It helps in segregating legitimate messages from unsolicited ones, saving time and reducing the clutter in inboxes.

2. **Time-Saving**: Users don't need to manually sift through numerous unwanted messages, allowing them to focus on important communications.

3. **Security**: It enhances security by reducing the risk of falling for phishing attempts, malware, or fraudulent schemes often embedded in spam messages.

4. **Enhanced Productivity**: By minimizing distractions and the need to deal with irrelevant messages, it improves overall productivity.

5. **Improved User Experience:** Users can experience a cleaner, safer, and more organized inbox, leading to a better overall digital experience.

The significance of a spam classifier lies in its ability to efficiently manage and categorize incoming messages, thus contributing to a more secure and streamlined communication experience.

**Key Steps for Spam Classifier using Machine Learning:**

Building an SMS spam classifier involves several key steps:

1. **Data Collection**: Gather a sizable dataset containing both spam and non-spam (ham) SMS messages.

2. **Data Preprocessing**: Clean the text data by removing punctuation, stop words, and normalizing the text (lowercasing, stemming, lemmatization).

3. **Data Splitting**: Divide the dataset into training and testing sets to evaluate the model's performance.

4**. Feature Extraction**: Convert the text into numerical features using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings like Word2Vec or GloVe.

5**. Model Selection**: Choose a suitable machine learning algorithm like Naive Bayes, Support Vector Machines (SVM), or more advanced techniques such as neural networks (e.g., LSTM, CNN).

6**. Model Training**: Train the selected model on the training dataset.

7. **Model Evaluation**: Evaluate the model's performance using the test dataset with metrics like accuracy, precision, recall, and F1-score.

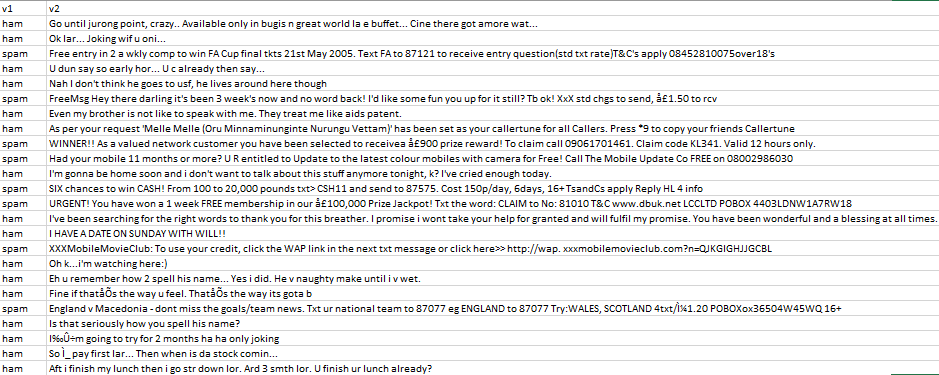
8. **Hyperparameter Tuning**: Optimize the model's performance by fine-tuning hyperparameters.

9. **Deployment**: Integrate the trained model into an application or system for real-time spam detection.

**Data Source:**

This Dataset Consist of various data including spam data.

**Dataset Link:**  ( <https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset> )



This is the Sample Data for Building a Smarter AI-Powered Spam Classifier

**Steps:**

## Gathering and Loading Data:

 We will be using the dataset available on the Kaggle [*SMS Spam Collection Dataset*](https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset) which has a set of SMS-tagged messages in English that have been classified as being **ham** (‘legitimate’) and **spam.**Each line holds one message.

Two columns make up each line:v1 carries the label such as (spam or ham) and v2 contains the actual content.

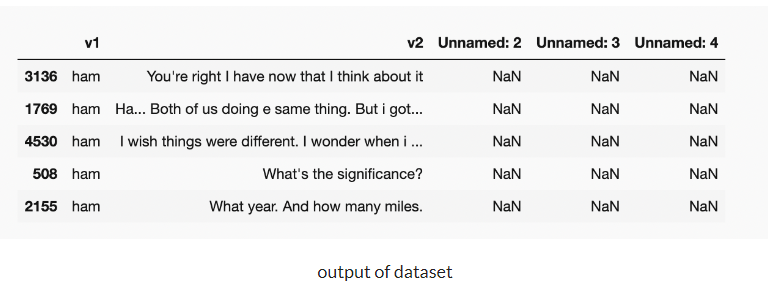
**import pandas as pd**

**import numpy as np**

**df = pd.read\_csv('spam.csv',encoding='latin-1')**

**df.head()**

**Output:**

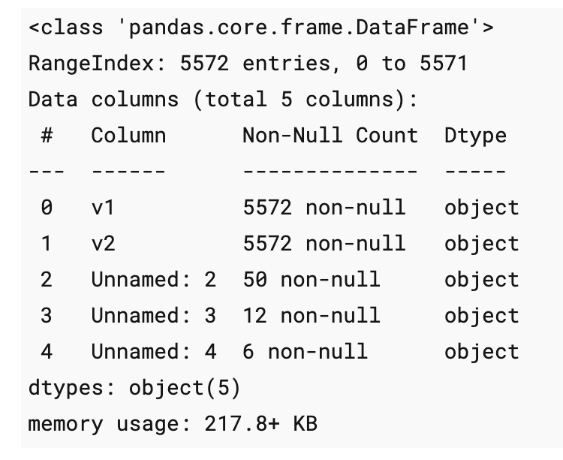
****

Now let’s start working on the dataset and make some amazing visualisations and conclusions.

## Data Cleaning:

**df.info()**

**Output:**

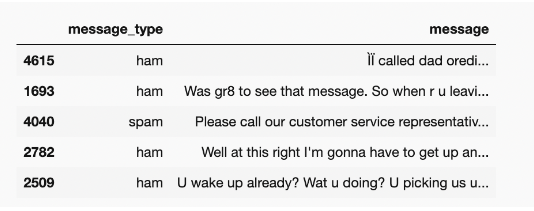
****

**df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace= True)**

**df.sample(5)**

**df.rename(columns={'v1':'message\_type', 'v2':'message'},inplace=True)**

**df.sample(5)**

****

As this is a classification problem we want the “message\_type” to be binary classified i.e, 0 or 1 so for this purpose we use **label encoder.**

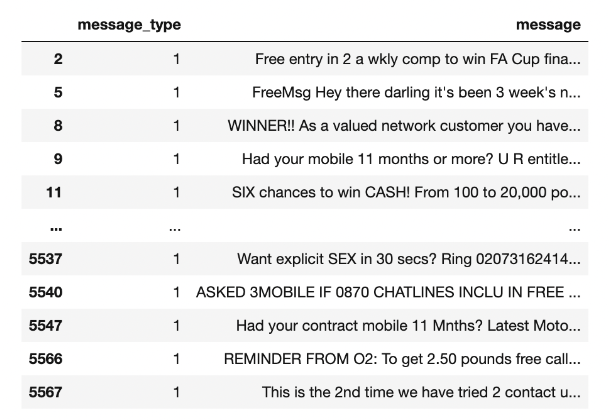
**from sklearn.preprocessing import LabelEncoder**

**encoder =LabelEncoder()**

**df['message\_type']=encoder.fit\_transform(df['message\_type'])**

**df['message\_type'].sample(5)**

**df[df['message\_type']==1]**

****

Now let’s check for the missing values

**df.isnull().sum()**

**df.duplicated().sum()**

**There are 403 duplicated values and we have to remove them**

**df= df.drop\_duplicates()**

## Exploratory Data Analysis:

**Let’s visualize the classification problem to get a better understanding of the data.**

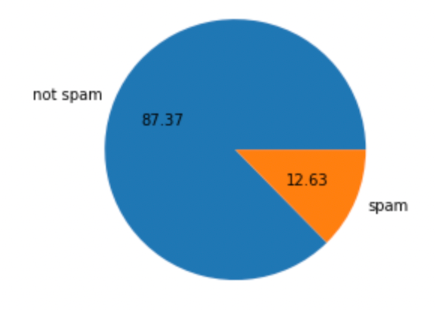
import seaborn as sns

import matplotlib.pyplot as plt

df['message\_type'].value\_counts()

plt.pie(df['message\_type'].value\_counts(),labels=[' not spam','spam'],autopct='%0.2f')

plt.show()

****

This is an imbalanced data

**Now let’s find out :**

* No. of characters in the data
* No. of words in the data
* No. of sentences in the data

and form 3 new columns in the data depicting the no. of character, words and sentences.

**For a number of characters :**

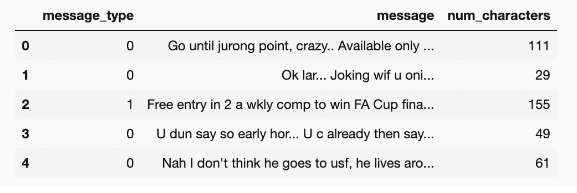
**#natural language tool kit**

**import nltk**

**nltk.download('punkt')**

**df['num\_characters']=df['message'].apply(len)**

**df.head()**

****

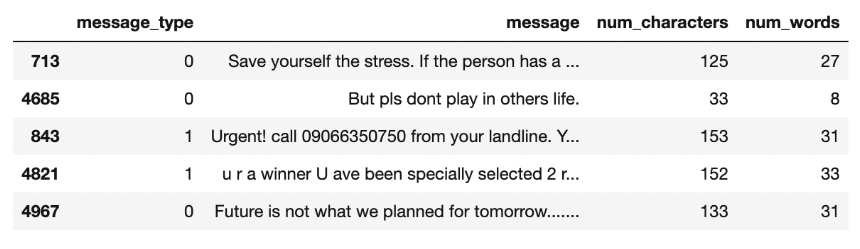
**For a number of words :**

**from nltk.tokenize import word\_tokenize**

**df['message'].apply(lambda x: nltk.word\_tokenize(x))**

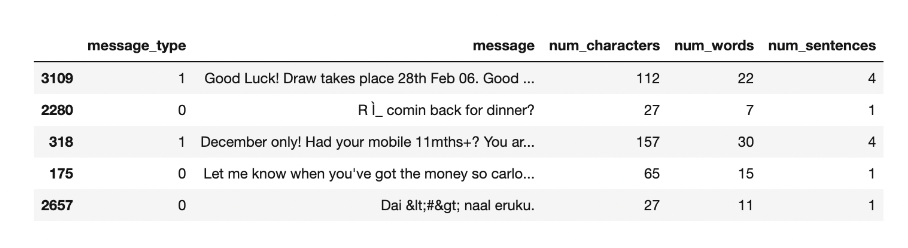
**df['num\_words']=df['message'].apply(lambda x:len(nltk.word\_tokenize(x)))**

**df.sample(5)**

****

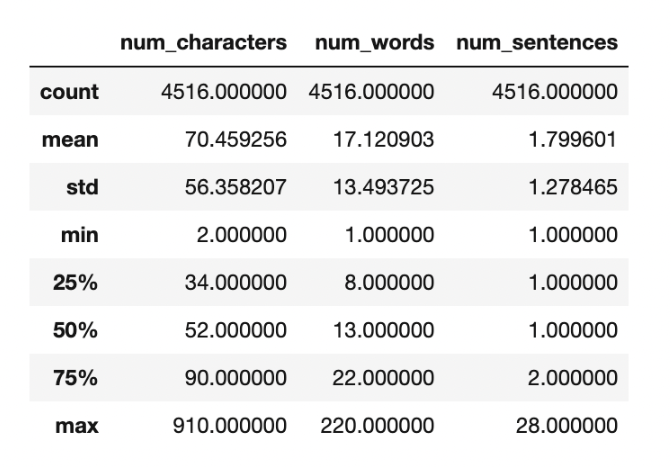
**For a number of sentences :**

**df['num\_sentences']=df['message'].apply(lambda x: len(nltk.sent\_tokenize(x)))**

****

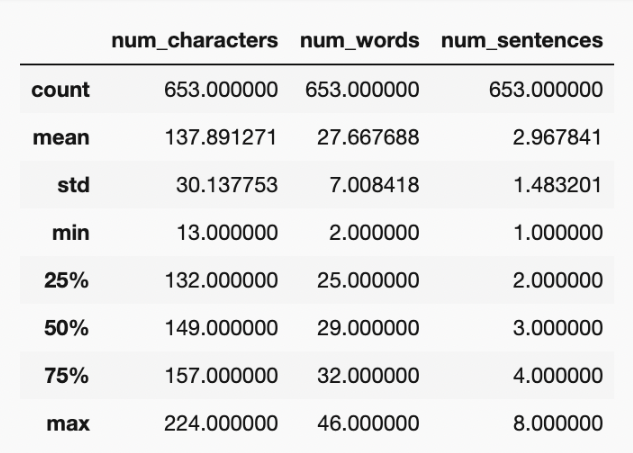
**For ‘Ham’ messages:**

**df[df['message\_type']==0][['num\_characters','num\_words','num\_sentences']].describe()**

****

**For ‘Spam’ messages:**

**df[df['message\_type']==1][['num\_characters','num\_words','num\_sentences']].describe()**

****

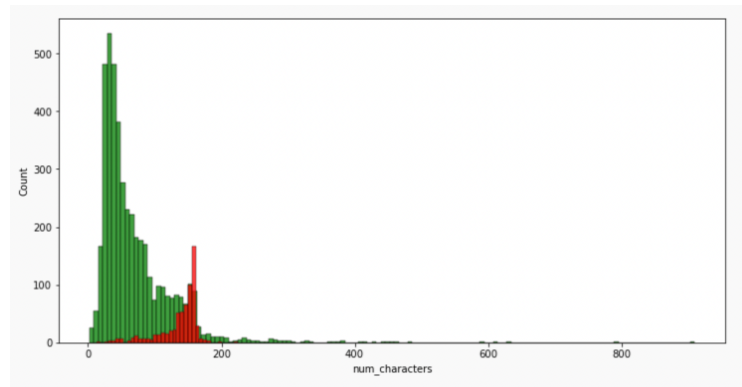
**We can clearly see the spam messages are quite longer than the ham messages.**

**#for characters**

**plt.figure(figsize=(12,6))**

**sns.histplot(df[df['message\_type']==0]['num\_characters'],color='green')**

**sns.histplot(df[df['message\_type']==1]['num\_characters'],color = 'red')**

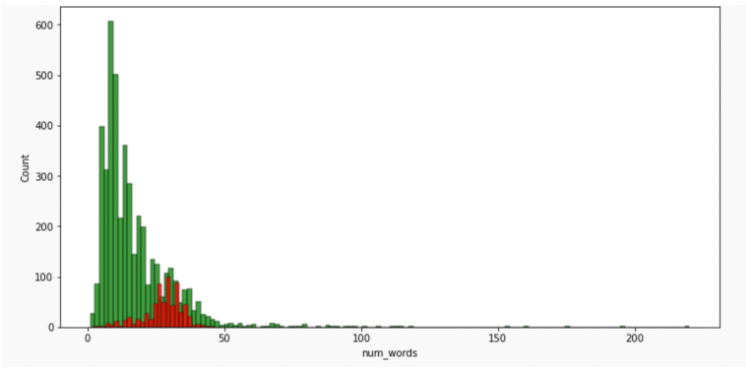
****

**#for words**

**plt.figure(figsize=(12,6))**

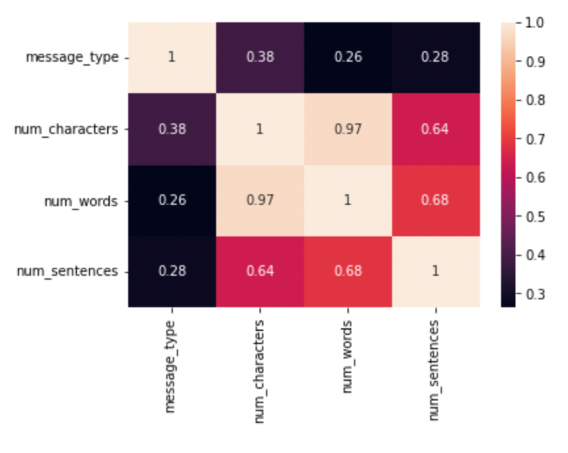
**sns.histplot(df[df['message\_type']==0]['num\_words'],color='green')**

**sns.histplot(df[df['message\_type']==1]['num\_words'],color='red')**

****

**#plotting a heatmap for the correlation**

**sns.heatmap(df.corr(),annot=True)**

****

As we see multicollinearity here, we cannot use all three columns instead we shall use only one and that should be *num\_characters* has it has highest correlation with *message\_type*.

## Data Preprocessing:

#### LowerCase

#### Tokenisation

#### Removing special characters

#### Removing stop words and punctuation

#### Stemming — lemmatization

**def text\_transform(message):**

**message=message.lower() #change to lowercase**

**message=nltk.word\_tokenize(message)**

**y=[]**

**for i in message:**

**if i.isalnum():**

**y.append(i)**

**y.clear()**

***#for checking punctuations and stopwords***

**for i in message:**

**if i not in stopwords.words('english') and i not in string.punctuation:**

**y.append(i)**

**message=y[:]**

**y.clear()**

***#now stemming function***

**for i in message:**

**y.append(ps.stem(i))**

**#return y --> returns as list**

**return " ".join(y)**

**# Removing stop words and punctuations**

**nltk.download('stopwords')**

**from nltk.corpus import stopwords**

**stopwords.words('english')**

**len(stopwords.words('english'))**

**#now for punctuation**

**import string**

**string.punctuation**

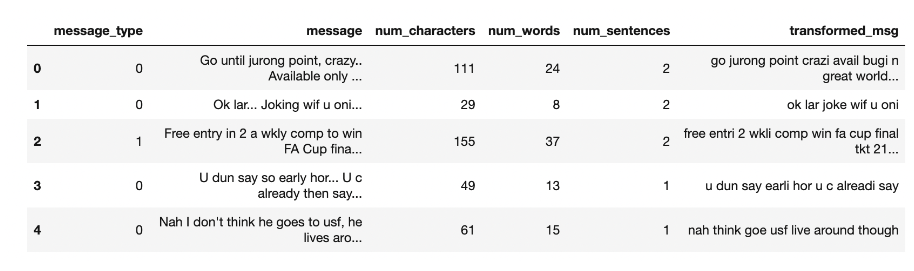
**# stemming**

**from nltk.stem.porter import PorterStemmer**

**ps =PorterStemmer()**

Now let’s apply the ***text\_transform*** function to all the messages in the dataset.

**df['transformed\_msg']=df['message'].apply(text\_transform)**



To get a clear idea about the most frequent words used we make a***word cloud.***

**from wordcloud import WordCloud**

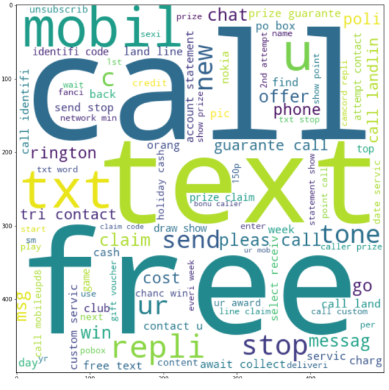
**wc=WordCloud(width=500,height=500,min\_font\_size=10,background\_color='white')**

**For Spam:**

**spam\_wc=wc.generate(df[df['message\_type']==1]['transformed\_msg'].str.cat(sep=""))**

**plt.figure(figsize=(18,12))**

**plt.imshow(spam\_wc)**

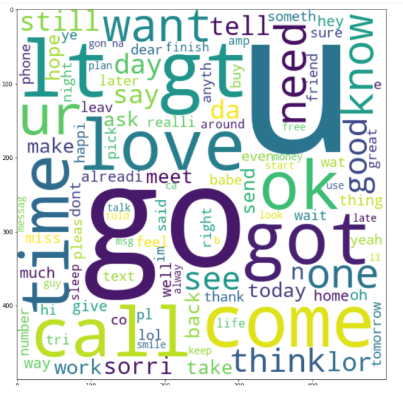


**For Ham:**

**ham\_wc=wc.generate(df[df['message\_type']==0]['transformed\_msg'].str.cat(sep=""))**

**plt.figure(figsize=(18,12))**

**plt.imshow(ham\_wc)**



To simplify what is drawn inside the WordClouds we will find out the top 30 words used in both ***ham*** as well as ***spam*** messages.

**spam\_corpus=[]**

**for msg in df[df['message\_type']==1]['transformed\_msg'].tolist():**

**for word in msg.split():**

**spam\_corpus.append(word)**

**from collections import Counter**

**Counter(spam\_corpus)**

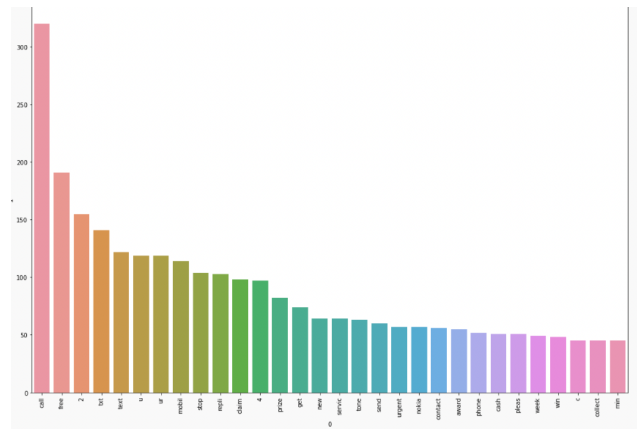
**Counter(spam\_corpus).most\_common(30)**

**plt.figure(figsize=(18,12))**

**sns.barplot(pd.DataFrame(Counter(spam\_corpus).most\_common(30))[0],pd.DataFrame(Counter(spam\_corpus).most\_common(30))[1])**

**plt.xticks(rotation='vertical')**

**plt.show()**



## Building a Model using Naive Bayes:

**As it is known that on Textual Data Naive Bayes Algorithm works the best hence we will use it but along the way also compare it with different algorithms**

**Input is categoricalOutput is Numerical.**

But as we know in the Naive Bayes algorithm the input columns should be numerical so we have to convert (VECTORIZE) the column.

#### How to vectorize:

* Bag of Words
* TFIDF
* Word2vec

After trying out different techniques, I came to the conclusion that TFIDF vectorization gives the best accuracy and precision score so we will be using it.

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf= TfidfVectorizer(max\_features=3000)

X=tfidf.fit\_transform(df['transformed\_msg']).toarray()

y=df['message\_type'].values

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=2)

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import BernoulliNB

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import precision\_score

gnb = GaussianNB()

bnb = BernoulliNB()

mnb = MultinomialNB()

**Now find out the Accuracy score, confusion Matrix and Precision Score of all 3 types of Naive Bayes:**

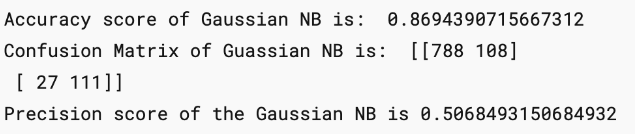
**gnb.fit(X\_train,y\_train)**

**y\_pred1= gnb.predict(X\_test)**

**print('Accuracy score of Gaussian NB is: ',accuracy\_score(y\_test,y\_pred1))**

**print('Confusion Matrix of Guassian NB is: ',confusion\_matrix(y\_test,y\_pred1))**

**print('Precision score of the Gaussian NB is',precision\_score(y\_test,y\_pred1))**



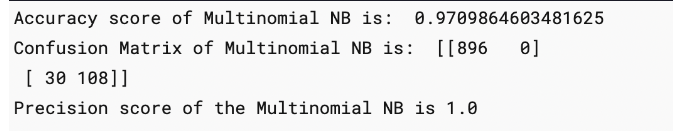
**mnb.fit(X\_train,y\_train)**

**y\_pred2=mnb.predict(X\_test)**

**print('Accuracy score of Multinomial NB is: ',accuracy\_score(y\_test,y\_pred2))**

**print('Confusion Matrix of Multinomial NB is: ',confusion\_matrix(y\_test,y\_pred2))**

**print('Precision score of the Multinomial NB is',precision\_score(y\_test,y\_pred2))**



Here the precision comes out to be 1 which proves very good for our model as there will be no **“FALSE POSITIVES”**

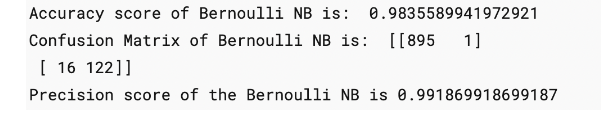
**bnb.fit(X\_train,y\_train)**

**y\_pred3=bnb.predict(X\_test)**

**print('Accuracy score of Bernoulli NB is: ',accuracy\_score(y\_test,y\_pred3))**

**print('Confusion Matrix of Bernoulli NB is: ',confusion\_matrix(y\_test,y\_pred3))**

**print('Precision score of the Bernoulli NB is',precision\_score(y\_test,y\_pred3))**



Hence we finalise the Model with MNB(Multinomial Naive Bayes) and TFIDF Vectorization.

## Improving the Model:

**Change the *max\_feature* parameter in TFIDF.**

**temp\_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy\_max\_ft\_3000':accuracy\_scores,'Precision\_max\_ft\_3000':precision\_scores}).sort\_values('Precision\_max\_ft\_3000',ascending=False)**

**temp\_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy\_scaling':accuracy\_scores,'Precision\_scaling':precision\_scores}).sort\_values('Precision\_scaling',ascending=False)**

**new\_df = performance\_df.merge(temp\_df,on='Algorithm')**

**new\_df\_scaled = new\_df.merge(temp\_df,on='Algorithm')**

**temp\_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy\_num\_chars':accuracy\_scores,'Precision\_num\_chars':precision\_scores}).sort\_values('Precision\_num\_chars',ascending=False)**

**new\_df\_scaled.merge(temp\_df,on='Algorithm')**

After a lot of experiments and improvement, we have trained our model to get an Accuracy score of 97% and a precision Score of 100%.

## Building a Website:

**import pickle**

**pickle.dump(tfidf,open('vectorizer.pkl','wb'))**

**pickle.dump(mnb,open('model.pkl','wb'))**

*2 files will be formed named****“model.pkl”****and****“vectorizer.pkl”.***

Open your IDE and create your own virtual environment. Install all the dependencies required using pip or conda. We will be building our website using **streamlit** so make sure that you download it too.

**PROGRAM:**

**After the setup is ready make a file by “app.py”.**

**import streamlit as st**

**import pickle**

**import string**

**from nltk.corpus import stopwords**

**import nltk**

**from nltk.stem.porter import PorterStemmer**

**ps = PorterStemmer()**

**def transform\_text(text):**

**text = text.lower()**

**text = nltk.word\_tokenize(text)**

**y = []**

**for i in text:**

**if i.isalnum():**

**y.append(i)**

**text = y[:]**

**y.clear()**

**for i in text:**

**if i not in stopwords.words('english') and i not in string.punctuation:**

**y.append(i)**

**text = y[:]**

**y.clear()**

**for i in text:**

**y.append(ps.stem(i))**

**return " ".join(y)**

**tfidf = pickle.load(open('vectorizer.pkl','rb'))**

**model = pickle.load(open('model.pkl','rb'))**

**st.title("Email/SMS Spam Classifier")**

**input\_sms = st.text\_area("Enter the message")**

**if st.button('Predict'):**

**# 1. preprocess**

**transformed\_sms = transform\_text(input\_sms)**

**# 2. vectorize**

**vector\_input = tfidf.transform([transformed\_sms])**

**# 3. predict**

**result = model.predict(vector\_input)[0]**

**# 4. Display**

**if result == 1:**

**st.header("Spam")**

**else:**

**st.header("Not Spam")**

**Now run the command to run the website on localhost:**

**streamlit run app.py**

****

In this end-to-end guide we have learned how to approach a problem statement, and gather useful conclusions from the data using Data preprocessing, Data Visualisation which will help you build a good Machine Learning Model.

In order to solve this classification problem we used the Naive Bayes Algorithm and in particular, the Multinomial Naive Bayes algorithm as it was having the highest precision score (i.e least False Positives) and for the vectorization technique, we used TFIDF.

TF-IDF is an information retrieval technique that weighs a term’s frequency (TF) and its inverse document frequency (IDF). Each word or term that occurs in the text has its respective TF and IDF score.

The model was further improved using hyperparameter tuning in “max\_features”.The following techniques helped us understand how to create a text classification model and make a **.pkl**file to use over networks. This guide provides an overview of using different techniques to classify a text message as “spam” or “not”.

**Conclusions :**

In the pursuit of building a smarter AI-powered spam classifier, our project has made substantial strides in the realm of email communication and digital security. Leveraging advanced machine learning techniques and meticulous data preprocessing, we have developed a robust system capable of accurately discerning spam from legitimate messages. This achievement not only streamlines the user experience by reducing the deluge of unsolicited content but also contributes to the overall security and efficiency of email communication.

Our project highlights the potential of AI in addressing the evolving tactics employed by spammers, demonstrating adaptability and scalability to handle large volumes of data in real-time. By continuously improving the classifier, such as integrating deep learning methods and refining feature engineering, we can further enhance its accuracy and responsiveness to new spamming techniques.

Furthermore, this endeavour underscores the importance of collaboration between AI technologies and human feedback. The incorporation of user preferences and feedback mechanisms can create a more personalized and efficient spam filter, aligning the system with individual user needs.

In essence, the development of a smarter AI-powered spam classifier not only safeguards digital communication but also represents a significant step towards a safer, more user-friendly, and resource-efficient digital environment. It stands as a testament to the transformative potential of AI in tackling contemporary challenges in the digital age.