

(ISO 9001:2015 Certified), Accredited with ‘A’ Grade by NAAC

🕿: 08258 - 281039 – 281263, Fax: 08258 – 281265

**Department of Computer Science and Engineering**

B.E. CSE Program Accredited by NBA, New Delhi from 1-7-2018 to 30-6-20

Report on Mini Project

“SPAM EMAIL DETECTION”

**Course Code: 18CS601**

**Course Name: Machine Learning**

Semester: VI Section : B

**Submitted To,**

**Mrs. Shabari Shedthi B**

**Asst Prof Gd II,Dept of CSE,NMAMIT**

**Submitted By:**

Name: Ganesh Wagle USN: 4NM18CS058

Name: Lancy Joy Lobo USN: 4NM18CS079

**Date of submission:20-06-2021**

**Signature of Course Instructor**



**NMAM Institute of Technology**

**(An Autonomous Institute Affiliated to VTU, Belagavi) (A unit of NITTE Education Trust)**

**NITTE – 574110, UDUPI DIST., KARNATAKA**

### **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# CERTIFICATE

“Stroke Prediction using Logistic Regression”

is a bonafide work carried out by

Ganesh Wagle- 4NM18CS058

Lancy Joy Lobo - 4NM18CS079

in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in computer science and engineering prescribed by the Vishvesvaraya Technological University, Belagavi during the year 2019 - 2020

It is certified that all the corrections/suggestions indicated for internal assessment have been incorporated in the report.

The mini-project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the Bachelor of Engineering Degree.

Signature of Guide Signature of HOD

# ACKNOWLEDGEMENT

We believe that our project will be complete only after we thank the people who have contributed to making this project successful.

First and foremost, we express our deep sense of gratitude and indebtedness to our guide Mrs. Shabari Shedthi, Assistant Professor, Department of Computer Science and Engineering, for her guidance, constant encouragement, support, and suggestions for improvement during the course of our project.

We sincerely thank Dr. Jyothi Shetty, Head of Department of Computer Science and Engineering, Nitte Mahalinga Adyantaya Memorial Institute of Technology, Nitte.

Our sincere thanks to our beloved principal, Dr. Niranjan N. Chiplunkar for giving us an opportunity to carry out our project work at our college and providing us with all the needed facilities.

Finally, we thank the staff members of the Department of Computer Science and Engineering and all our friends for their honest opinions and suggestions throughout the course of our project.

Ganesh Wagle - 4NM18CS058

Lancy Joy Lobo - 4NM18CS079

**Abstract**

The following submission contains a detailed report on the Machine Learning Mini Project “Spam Email filtration” by Ganesh Wagle and Lancy Joy Lobo. The project has been implemented using Python and multiple data processing and data visualization libraries supported by Python. The purpose of the project is to classify whether a mail/sms is spam or not. To achieve this, A Machine Learning model has to be trained with balanced dataset consisting of finite correlated features .The report contains in-depth information of the process followed to develop the Machine Learning model.

**Table of contents**

(Page no)

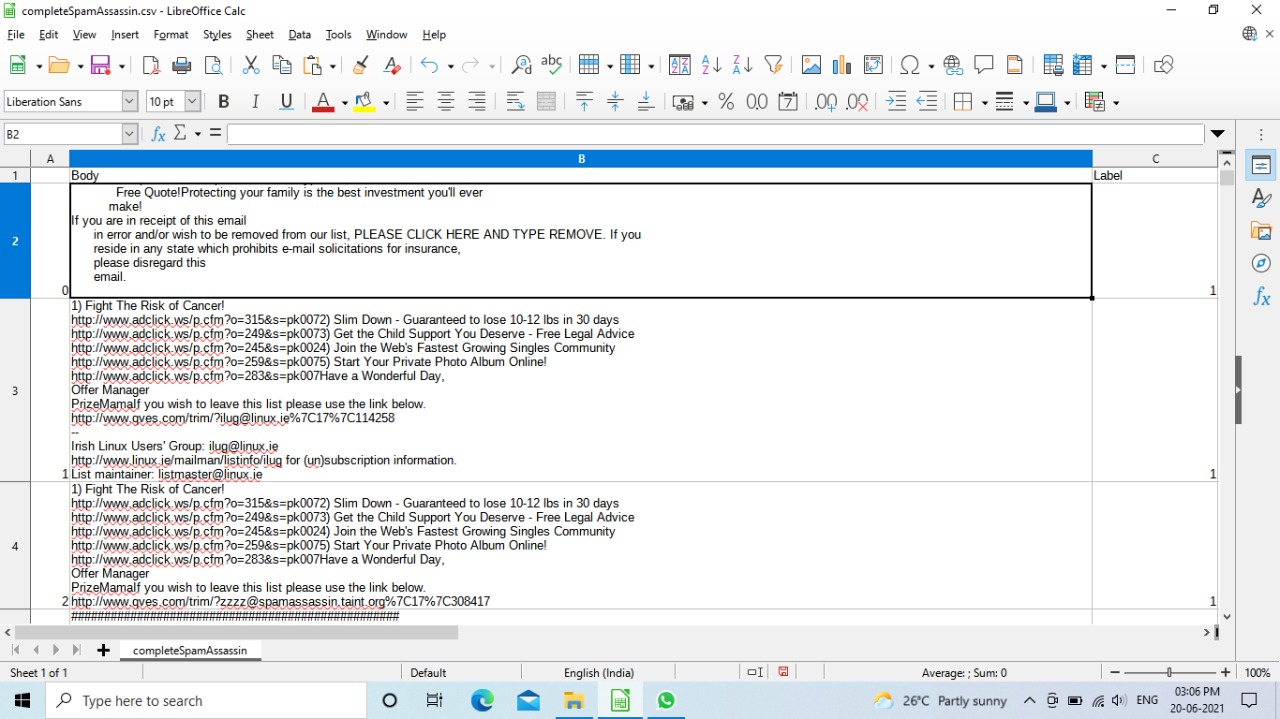
1. Introduction 6
2. Design 7
3. Exploratory Data Analysis 8
4. Feature Analysis 8
5. Feature Engineering 9
6. Accuracy 10
7. Model implementation 11
8. Conclusion 24
9. Reference 25

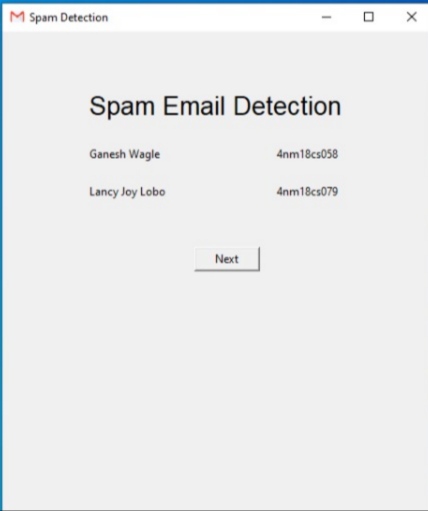
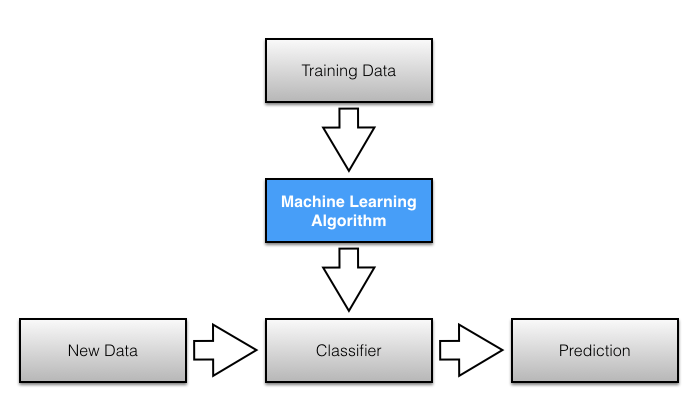
**Introduction**

The outcome of the mini project is classify a mail as Spam or not . We used Kaggle’s dataset for this purpose.Also here we have used the concept of count vectorizer to count the words in dataset which falls under spam and ham and using that the model predicts whether a mail is spam or not using Multinomial Naïve Bayes.The model The Project is divided into 3 different divisions as follows.

1. Exploratory Data Analysis (EDA): The dataset is explored completely using graphs, plots and made ready to be fit into a ML model. EDA is performed in by conducting Feature Analysis and Feature Engineering.
2. Model Implementation: Building a Count Vectorizer model to count the words and Multinomial NB to predict whether a mail is spam or not.
3. User Interface: Python’s built in library Tkinter has been implemented as UI.

**Dataset:**

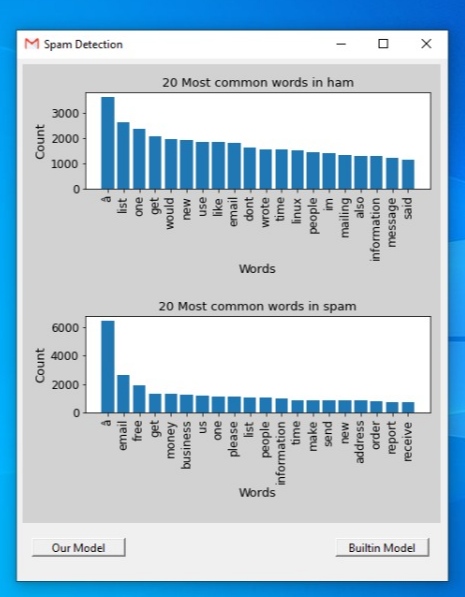
****

**DESIGN**

**Exploratory Data analysis**

**Feature Analysis:**

Since it’s a spam mail classification the dataset has only 3 colums.the index,body and label.if label=1 then the mail is spam or else it’s a ham.In this model we use count vectorizer to count the number of times the words have repeated in spam and ham.



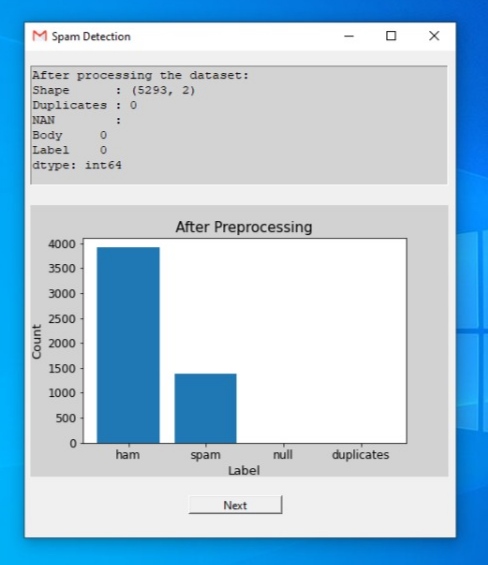
**Feature Engineering**

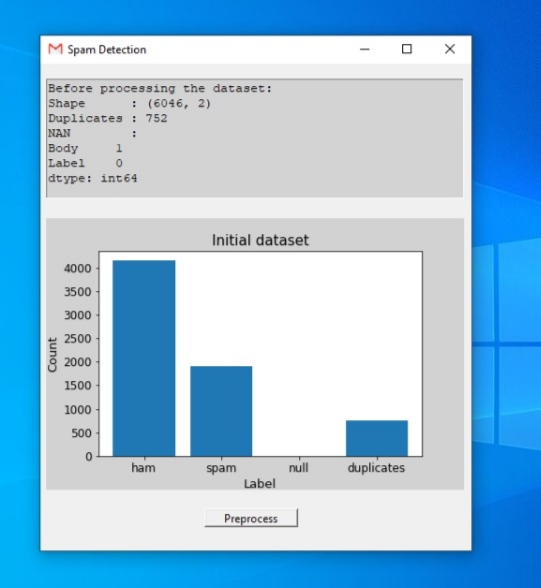
Inconsistencies in the dataset are handled and the dataset is converted into machine understandable format

1.Handling NULL values

No null values were present in the dataset.

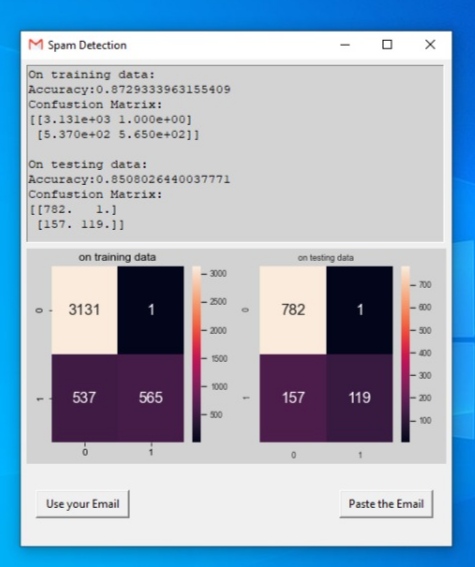
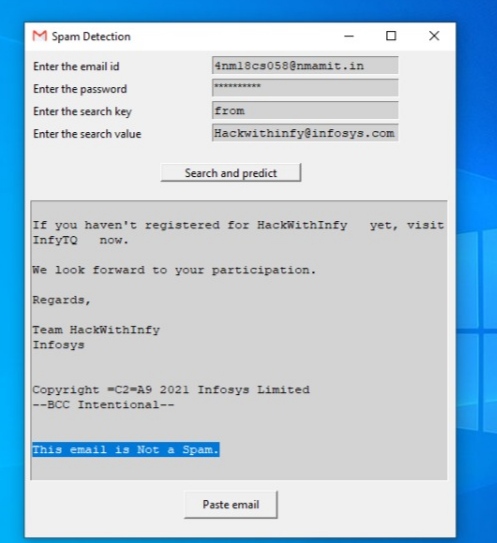
2.Handling duplicates

 There were around 76 duplicates in the dataset.successfully handled them using buit in functions provide by the Python.



**Accuracy**

Our model was able to predict whether a mail is spam or not with an accuracy of 98%.However when we used hardcoded them i.e. when we used our own code it gave an accuracy of 87%.



**Model Implementation:**

'''

To fecth email from your email address first you need to goto your email and enable imap

access and allow access through less secure source then only you will be able to access

your email.

'''

import numpy as np

import pandas as pd

import string

import matplotlib.pyplot as plt

from tkinter import \*

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from matplotlib.figure import Figure

import tkinter.font as tkFont

import imaplib

from tkinter import messagebox

from collections import Counter

from scipy.sparse import csr\_matrix

import math

import functools

import operator

import seaborn as sns

import email

vocab = {}

model = 0

#to find the vocabulary

def custum\_fit(data):

global vocab

unique\_words = set()

for each\_sentence in data:

for each\_word in each\_sentence:

unique\_words.add(each\_word)

for index, word in enumerate(sorted(list(unique\_words))):

vocab[word]=index

vocabflag = 0

'''

This function is used to find the frequency of each words in the email

Initially we use custum\_fit function for training data to find all possible words(vocabulary)

Then for testing data we check for each word in vocab how many times that word has

occured in the testing email

Then we return a csr matrix of that data

'''

def custum\_transform(data):

global vocab

if vocabflag==0:

custum\_fit(data)

row, col, val = [],[],[]

for idx, sentence in enumerate(data):

count\_word = dict(Counter(sentence))

for word, count in count\_word.items():

col\_index = vocab.get(word)

if col\_index is not None:

if col\_index>=0:

row.append(idx)

col.append(col\_index)

val.append(count)

return csr\_matrix((val,(row,col)),shape=(len(data),len(vocab)))

#Function used to find the accuracy of our model

def accuracy(actual,prediction):

actual = list(actual)

if len(actual) != len(prediction):

print('invalid input')

return

total = len(actual)

correct = 0

for i in range(total):

if actual[i]==prediction[i]:

correct = correct + 1

return correct/total

#It is used to find the confusion matrix

def confusion(actual,prediction):

actual = list(actual)

if len(actual) != len(prediction):

print('invalid input')

return

mat = np.zeros((2,2))

for i in range(len(actual)):

mat[actual[i],prediction[i]] = mat[actual[i],prediction[i]] + 1

return mat

class MultiNB:

def \_\_init\_\_(self,alpha=1):

#alpha is smoothing factor

self.alpha = alpha

def \_prior(self):

"""

Calculates prior for each unique class in y. P(y)

"""

P = np.zeros((self.n\_classes\_))

\_, self.dist = np.unique(self.y,return\_counts=True)

for i in range(self.classes\_.shape[0]):

P[i] = self.dist[i] / self.n\_samples

return P

def fit(self, X, y):

'''

P(xi∣y)=Nyi+α/Ny+αn this is the formula we used

It will basically find the probability of each word in the email being in

each of the class.

'''

self.y = y

self.n\_samples, self.n\_features = X.shape

self.classes\_ = np.unique(y)

self.n\_classes\_ = self.classes\_.shape[0]

self.class\_priors\_ = self.\_prior()

self.docidx, self.wordidx = X.nonzero()

count = X.data

classidx = []

for idx in self.docidx:

classidx.append(self.y.iloc[idx])

df = pd.DataFrame()

df['docidx'] = np.array(self.docidx)

df['wordidx'] = np.array(self.wordidx)

df['count'] = np.array(count)

df['classidx'] = np.array(classidx)

#print(df.info)

global vocab

self.N\_yi = df.groupby(['classidx','wordidx'])

#for key,item in self.N\_yi:

# print(self.N\_yi.get\_group(key))

self.N\_y = df.groupby('classidx')

#self.N\_yi = (self.N\_yi['count'].sum()

self.Pr = (self.N\_yi['count'].sum() + self.alpha) / (self.N\_y['count'].sum() +(self.alpha\*len(vocab)))

#Unstack series

self.Pr = self.Pr.unstack()

#Replace NaN or columns with 0 as word count with a/(count+|V|+1)

for c in range(0,2):

self.Pr.loc[c,:] = self.Pr.loc[c,:].fillna(self.alpha/(self.N\_y['count'].sum()[c] + len(vocab) ))

self.Pr\_dict = self.Pr.to\_dict()

#for key, item in self.pb\_ij:

#print(pb\_ij.get\_group(key), "\n\n")

#print(key,item)

'''

For a give email x it will find the probability of it belonging to the class h.

It will basically return the product of prob. of each word in x belonging to class h

which was previously calculated in fit function.

'''

def \_likelyhood(self, x, h):

tmp = []

X = x.toarray()[0]

indices = X.nonzero()[0]

for i in indices:

if i in self.Pr\_dict.keys():

tmp.append(float(math.pow(self.Pr\_dict[i][h],X[i])))

return np.exp(np.log(tmp).sum())

'''

It will find to which class the given email belongs to.

'''

def predict(self, X):

samples, features = X.shape

self.predict\_proba = np.zeros((samples,self.n\_classes\_))

for i in range(X.shape[0]):

# joint\_likelyhood = np.zeros((self.n\_classes\_))

for h in range(self.n\_classes\_):

self.predict\_proba[i,h] = self.class\_priors\_[h] \* self.\_likelyhood(X[i],h)

#print(self.predict\_proba[i,h])

# P(y) P(X|y)

#print(self.predict\_proba)

indices = np.argmax(self.predict\_proba,axis=1)

return self.classes\_[indices]

allemails = []

'''

It will iterate through each email in the dataframe which was given as input

then apply clean text function to each email and append the result to a list.

'''

def find\_mails(data):

global allemails

data = clean\_text(data)

allemails.append(data)

#Used to get the email from our gmail account

imap\_url = 'imap.gmail.com'

con = None

#Load the data

df = pd.read\_csv("completeSpamAssassin.csv");

print(df.head)

#drop the unwanted column

df.drop(['Unnamed: 0'],axis=1,inplace=True)

target = 'Label'

feature = 'Body'

#Loading the stopwords in english

stopwordslist = []

with open('stopwords.txt','r') as file:

for row in file:

stopwordslist.append(row.split('\n')[0])

punctuation = string.punctuation

punctuations = []

for char in punctuation:

punctuations.append(char)

'''

This function is used to remove all the \n, numbers, punctuations and stopwords.

'''

def clean\_text(text):

text = str(text)

#Remove backslash n

#text = [text.split('\\n')]

#text = ''.join(text[0])

text.replace('\n','')

#print('After joining;\n',text)

#Remove punctuations

newtext = ''

for char in text:

if char not in punctuations:

newtext += char

#print("After removing punctuations:\n",newtext)

#Removing stopwords

words = newtext.split()

corpus = []

for word in words:

if word.lower() not in stopwordslist:

corpus.append(word.lower())

#print('After removing the stopwords:\n',corpus)

#Remove numbers

for words in corpus:

word = ''

for char in words:

if char.isalpha():

word +=char

corpus[corpus.index(words)] = word

#Remove null from corpus

while("" in corpus):

corpus.remove("")

#print(corpus)

return corpus

#print(df[feature],type(df[feature]))

#Importing CountVectorizer

from sklearn.feature\_extraction.text import CountVectorizer

vectorizer = CountVectorizer(analyzer=clean\_text)

#Importing Multinomial Byes classifier

from sklearn.naive\_bayes import MultinomialNB

classifier = MultinomialNB()

me = MultiNB()

#function to get the body of the email

def get\_body(msg):

if msg.is\_multipart():

return get\_body(msg.get\_payload(0))

else:

return msg.get\_payload()

'''

It will search your email account to find the email that matches the requirement and

return their id.

'''

def search(key,value,con):

result, data = con.search(None,key,value)

return data

#To fetch the email from its id

def get\_email(result\_bytes):

msgs = []

for num in result\_bytes[0].split():

typ, data = con.fetch(num,'(RFC822)')

msgs.append(data)

return msgs

#Used to remove html tags from the email

def cleanemail(email):

i = 0

length = len(email)

cleantext1 = ''

while i<length:

if email[i]=='<':

while email[i]!='>':

i+=1

else:

cleantext1+=email[i]

i+=1

return cleantext1

#gui starts here

#Creating the parent window

root = Tk()

f = Frame(root,height=520,width=460)

f.pack()

#giving title

root.title('Spam Detection')

#specifying geometry

root.geometry('460x520')

root.iconbitmap('icon.ico')

label1 = Label(f,text='Spam Email Detection',font=tkFont.Font(size=20))

label1.place(x=90,y=60)

name1 = Label(f,text='Ganesh Wagle')

name2 = Label(f,text='Lancy Joy Lobo')

usn1 = Label(f,text='4nm18cs058')

usn2 = Label(f,text='4nm18cs079')

name1.place(x=90,y=120)

name2.place(x=90,y=160)

usn1.place(x=290,y=120)

usn2.place(x=290,y=160)

def email\_entry():

global f

f.destroy()

f = Frame(root,height=520,width=460)

f.pack()

usr\_label = Label(f,text='Enter the email id')

usr\_label.place(x=6,y=6)

usr = Text(f,bg='light gray')

usr.place(x=200,y=6,height=20,width=200)

pas\_label = Label(f,text = 'Enter the password')

pas\_label.place(x=6,y=30)

pas = Entry(f,show='\*',bg='light gray')

pas.place(x=200,y=30,height=20,width=200)

key\_label = Label(f,text='Enter the search key')

key\_label.place(x=6,y=54)

key = Text(f,bg='light gray')

key.place(x=200,y=54,height=20,width=200)

search\_label = Label(f,text='Enter the search value')

search\_label.place(x=6,y=78)

search\_value = Text(f,bg='light gray')

search\_value.place(x=200,y=78,height=20,width=200)

res = Text(f,bg='light gray')

res.place(x=6,y=160,height=300,width=446)

def start():

usrname = usr.get('1.0','end')

password = pas.get()

global con

con = imaplib.IMAP4\_SSL(imap\_url)

# print(usrname[:-1],password)

try:

con.login(usrname[:-1],password)

con.select('"[Gmail]/All Mail"')

print('Connection established')

except:

messagebox.showerror("error",'Check your credentials again')

#con.close()

# con.logout()

print('Check your crentials again')

return

key\_value = key.get('1.0','end')

value\_search = search\_value.get('1.0','end')

try:

resul = search(key\_value[:-1],value\_search[:-1],con)

global classifier,me,model

final\_result = 'Emails that which satisfy your query are '

for num in resul[0].split():

final\_result += str(num)

print(final\_result)

msgs = get\_email(resul)

#print(msgs)

for msg in msgs:

mail = '\nSubject:'+email.message\_from\_bytes(msg[0][1])['Subject']

mail = mail + '\n' + cleanemail(get\_body(email.message\_from\_bytes(msg[0][1])))

#print(mail)

final\_result += mail+'\n'

#df1 = pd.DataFrame()

#df1['email'] = pd.Series(mail,dtype='object')

#mail = vectorizer.transform(df1['email'])

mail = clean\_text(mail)

mail = custum\_transform([mail])

check = ''

prediction = None

if model==1:

print(me.predict(mail))

prediction = me.predict(mail)[0]

else:

prediction = classifier.predict(mail)[0]

if prediction==1:

check = "a Spam!!!\n\n"

else:

check = "Not a Spam.\n\n"

final\_result += "\nThis email is "+check

res.delete('1.0','end')

res.insert(INSERT,final\_result)

except:

messagebox.showerror("error",'Check your Search Inputs')

con.close()

con.logout()

print('Check search value')

search\_but = Button(f,text='Search and predict',command=start)

search\_but.place(x=145,y=120,height=20,width=150)

paste\_email = Button(f,text='Paste email',command=predict\_email)

paste\_email.place(x=170,y=470,height=30,width=100)

def predict\_email():

global f,model

f.destroy()

f = Frame(root,height=520,width=460)

f.pack()

label1 = Label(f,text='Paste your email here:',font=tkFont.Font(size=10))

label1.place(x=6,y=6)

messageWindow = Text(f,bg='light gray')

messageWindow.place(x=6,y=40,height=380,width=446)

label2 = Label(f,text="",font=tkFont.Font(size=13))

def predict():

nonlocal label2

email1 = messageWindow.get('1.0','end')

#df1 = pd.DataFrame()

global classifier,me

email1 = clean\_text(email1)

email1 = custum\_transform([email1])

prediction = None

if model==1:

print(me.predict(email1))

prediction = me.predict(email1)[0]

else:

prediction = classifier.predict(email1)[0]

if prediction==1:

label2.config(text ="Entered email is a Spam!!!.",fg='red' )

else:

label2.config(text ="Entered email is Not a Spam.",fg='green' )

label2.place(x=120,y=430)

predict\_button = Button(f,text='Predict',command=predict)

predict\_button.place(x=6,y=430,height=30,width=100)

goto\_email = Button(f,text='Goto email',command=email\_entry)

goto\_email.place(x=6,y=470,height=30,width=100)

def train\_ours():

global f,root,model

model = 1

f.destroy()

f = Frame(root,height=520,width=460)

f.pack()

root.geometry('460x520')

#Convert a collection of tokens to a matrix of tokens

#global vectorizer

#message = vectorizer.fit\_transform(df[feature])

df['Body'].apply(find\_mails)

message = custum\_transform(allemails)

global vocabflag

vocabflag = 1

#Split the data into 80% training and 20% testing

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

x\_train,x\_test,y\_train,y\_test = train\_test\_split(message,df[target],test\_size=0.20,random\_state=0)

print(x\_train.shape,type(x\_train))

global me

me.fit(x\_train,y\_train)

#Evaluate the model on the training data set

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

pred = me.predict(x\_train)

#On testing data

pred1 = me.predict(x\_test)

con1 = confusion(y\_train,pred)

con2 = confusion(y\_test,pred1)

print('\nOn training data:\nAccuracy:'+str(accuracy(y\_train, pred)))

msg1 = 'On training data:\nAccuracy:'+str(accuracy(y\_train, pred))

msg2 = 'Confustion Matrix:\n'+str(con1)

print('On testing data:\nAccuracy:'+str(accuracy(y\_test, pred1)))

msg3 = '\n\nOn testing data:\nAccuracy:'+str(accuracy(y\_test, pred1))

msg4 = 'Confustion Matrix:\n'+str(con2)

final\_message=msg1+'\n'+msg2+msg3+'\n'+msg4

#create the message window

messageWindow = Text(f,bg='light gray')

messageWindow.place(x=6,y=6,height=190,width=446)

messageWindow.insert(INSERT,final\_message)

fig = Figure()

a = fig.add\_subplot(121)

a.set\_title('on training data')

canvas = FigureCanvasTkAgg(fig, master=f)

fig.patch.set\_facecolor((.8242,.8242,.8242))

sns.set(font\_scale=.75) # for label size

sns.heatmap(con1, annot=True, annot\_kws={"size": 16},ax=a,fmt='g')

a = fig.add\_subplot(122)

a.set\_title('on testing data')

canvas = FigureCanvasTkAgg(fig, master=f)

sns.set(font\_scale=.75) # for label size

sns.heatmap(con2, annot=True, annot\_kws={"size": 16},ax=a,fmt='g')

fig.tight\_layout()

canvas.get\_tk\_widget().place(x=6,y=202,height=230,width=448)

canvas.draw()

but = Button(root,text='Use your Email',command=email\_entry)

but.place(x=16,y=460,height=30,width=100)

predict = Button(root,text='Paste the Email',command=predict\_email)

predict.place(x=340,y=460,height=30,width=100)

def train\_builtin():

global f,root

f.destroy()

f = Frame(root,height=520,width=460)

f.pack()

root.geometry('460x520')

#Convert a collection of tokens to a matrix of tokens

#global vectorizer

#message = vectorizer.fit\_transform(df[feature])

df['Body'].apply(find\_mails)

message = custum\_transform(allemails)

global vocabflag

vocabflag = 1

#Split the data into 80% training and 20% testing

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

x\_train,x\_test,y\_train,y\_test = train\_test\_split(message,df[target],test\_size=0.20,random\_state=0)

print(x\_train.shape,type(x\_train))

#Create and train the Naive Bayes Classifier

global classifier

classifier = classifier.fit(x\_train,y\_train)

#Evaluate the model on the training data set

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

pred = classifier.predict(x\_train)

#On testing data

pred1 = classifier.predict(x\_test)

con1 = confusion(y\_train,pred)

con2 = confusion(y\_test,pred1)

print('\nOn training data:\nAccuracy:'+str(accuracy(y\_train, pred)))

msg1 = 'On training data:\nAccuracy:'+str(accuracy(y\_train, pred))

msg2 = 'Confustion Matrix:\n'+str(con1)

print('On testing data:\nAccuracy:'+str(accuracy(y\_test, pred1)))

msg3 = '\n\nOn testing data:\nAccuracy:'+str(accuracy(y\_test, pred1))

msg4 = 'Confustion Matrix:\n'+str(con2)

final\_message=msg1+'\n'+msg2+msg3+'\n'+msg4

#create the message window

messageWindow = Text(f,bg='light gray')

messageWindow.place(x=6,y=6,height=190,width=446)

messageWindow.insert(INSERT,final\_message)

fig = Figure()

a = fig.add\_subplot(121)

a.set\_title('on training data')

canvas = FigureCanvasTkAgg(fig, master=f)

fig.patch.set\_facecolor((.8242,.8242,.8242))

sns.set(font\_scale=.75) # for label size

sns.heatmap(con1, annot=True, annot\_kws={"size": 16},ax=a,fmt='g')

a = fig.add\_subplot(122)

a.set\_title('on testing data')

canvas = FigureCanvasTkAgg(fig, master=f)

sns.set(font\_scale=.75) # for label size

sns.heatmap(con2, annot=True, annot\_kws={"size": 16},ax=a,fmt='g')

fig.tight\_layout()

canvas.get\_tk\_widget().place(x=6,y=202,height=230,width=448)

canvas.draw()

but = Button(root,text='Use your Email',command=email\_entry)

but.place(x=16,y=460,height=30,width=100)

predict = Button(root,text='Paste the Email',command=predict\_email)

predict.place(x=340,y=460,height=30,width=100)

def words():

global f

global root

global allemails

f.destroy()

f = Frame(root,height=558,width=460)

f.pack()

df.loc[df['Label']==0]['Body'].apply(find\_mails)

allemails = functools.reduce(operator.iconcat, allemails, [])

#print(allemails)

count = Counter(allemails)

words = [word[0] for word in count.most\_common(20)]

values = [value[1] for value in count.most\_common(20)]

fig = Figure(figsize=(6,6))

fig.patch.set\_facecolor((.8242,.8242,.8242))

a = fig.add\_subplot(211)

a.bar(words,values)

a.set\_title('20 Most common words in ham',fontsize=13)

a.set\_xlabel('Words',fontsize=13)

a.tick\_params(axis='x', rotation=90)

a.set\_ylabel('Count',fontsize=13)

a.tick\_params(axis='both', which='major', labelsize=12)

fig.tight\_layout()

#canvas = FigureCanvasTkAgg(fig, master=f)

#canvas.get\_tk\_widget().place(x=6,y=6,height=390,width=446)

# canvas.draw()

allemails = []

df.loc[df['Label']==1]['Body'].apply(find\_mails)

allemails = functools.reduce(operator.iconcat, allemails, [])

count = Counter(allemails)

words = [word[0] for word in count.most\_common(21)]

words.pop(3)

values = [value[1] for value in count.most\_common(21)]

values.pop(3)

#fig = Figure(figsize=(10,3))

b = fig.add\_subplot(212)

b.bar(words,values)

b.set\_title('20 Most common words in spam',fontsize=13)

b.set\_xlabel('Words',fontsize=13)

b.tick\_params(axis='x', rotation=90)

b.tick\_params(axis='both', which='major', labelsize=12)

b.set\_ylabel('Count',fontsize=13)

fig.tight\_layout()

canvas = FigureCanvasTkAgg(fig, master=f)

canvas.get\_tk\_widget().place(x=6,y=6,height=490,width=446)

canvas.draw()

allemails = []

but1 = Button(root,text='Our Model',command=train\_ours)

but1.place(x=16,y=512,height=20,width=100)

but2 = Button(root,text='Builtin Model',command=train\_builtin)

but2.place(x=340,y=512,height=20,width=100)

root.geometry('460x558')

def preprocess():

#destroy the old frame

global f

f.destroy()

f = Frame(root,height=520,width=460)

f.pack()

#Removing the duplicates

df.drop\_duplicates(inplace=True)

#Removing the null values

df.dropna(inplace=True)

msg = '''After processing the dataset:

Shape : '''+str(df.shape)+'''

Duplicates : '''+str(df.duplicated().sum())+'''

NAN :\n'''+str(df.isnull().sum())

#creating text

text1 = Text(f,bg='light gray')

text1.place(x = 6, y = 16,height=128,width=446)

text1.insert(INSERT,msg)

#Plotting graph

x\_axis = ['ham','spam','null','duplicates']

nonspam = df[target].value\_counts()[0]

spam = df[target].value\_counts()[1]

dup = df.duplicated().sum()

null = df.isnull().sum()[0]

y\_axis = [nonspam,spam,null,dup]

fig = Figure()

fig.patch.set\_facecolor((.8242,.8242,.8242))

a = fig.add\_subplot(111)

a.bar(x\_axis,y\_axis)

a.set\_title('After Preprocessing',fontsize=15)

a.set\_xlabel('Label',fontsize=13)

a.set\_ylabel('Count',fontsize=13)

a.tick\_params(axis='both', which='major', labelsize=12)

canvas = FigureCanvasTkAgg(fig, master=f)

canvas.get\_tk\_widget().place(x=6,y=165,height=290,width=446)

canvas.draw()

but = Button(root,text='Next',command=words)

but.place(x=175,y=475,height=20,width=100)

def initial\_screen():

#destroy the old frame

global f

f.destroy()

#Create a new frame

f = Frame(root,height=520,width=460)

f.pack()

pre = Button(f,text='Preprocess',command=preprocess)

pre.place(x=175,y=475,height=20,width=100)

#creating text

text1 = Text(f,bg='light gray')

msg = '''Before processing the dataset:

Shape : '''+str(df.shape)+'''

Duplicates : '''+str(df.duplicated().sum())+'''

NAN :\n'''+str(df.isnull().sum())

text1.insert(INSERT,msg)

text1.place(x = 6, y = 16,height=128,width=446)

#Plotting graph

x\_axis = ['ham','spam','null','duplicates']

nonspam = df[target].value\_counts()[0]

spam = df[target].value\_counts()[1]

dup = df.duplicated().sum()

null = df.isnull().sum()[0]

y\_axis = [nonspam,spam,null,dup]

fig = Figure()

fig.patch.set\_facecolor((.8242,.8242,.8242))

a = fig.add\_subplot(111)

a.bar(x\_axis,y\_axis)

a.set\_title('Initial dataset',fontsize=15)

a.set\_xlabel('Label',fontsize=13)

a.set\_ylabel('Count',fontsize=13)

a.tick\_params(axis='both', which='major', labelsize=12)

canvas = FigureCanvasTkAgg(fig, master=f)

canvas.get\_tk\_widget().place(x=6,y=165,height=290,width=446)

canvas.draw()

button = Button(f,text='Next',command=initial\_screen)

button.place(x=205,y=230,width=70)

root.mainloop()

**Conclusion:**

Thus we have been able to design and develop a ML Naïve Bayes model to make accurate predictions for classifying emails.we have suucessfully developed a model which has an accuracy of 98%(using built in models).

**References**

1. Kaggle dataset:

<https://www.kaggle.com/omkarpathak27/spam-filtering>

Numpy documentation:

[https://numpy.org/doc/stable/](%20https:/numpy.org/doc/stable/)

1. Pandas documentation:

<https://pandas.pydata.org/docs/>

1. Matplotlib documentation:

<https://matplotlib.org/stable/users/index.html>

1. Seaborn documentation:

[https://seaborn.pydata.org/](%20https:/seaborn.pydata.org/)

1. Sklearn pre-processing module documentation:

<https://scikit-learn.org/stable/modules/preprocessing.html>