

Indian Institute of

Information Technology,

Nagpur

Natural Languag

Processing

Project Report

*Submitted By :*

Anuranjan Sarsam (BT19ECE003) Semester 6

Electronics and Communication Engineering Dept.

*Submitted To :*

## Dr. Pooja Jain

## Course Instructor

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **1.** | **INTRODUCTION** | **3** |
| **2.** | **PROBLEM STATEMENT** | **5** |
| **3.** | **SOURCE CODE** | **7** |
| **4.** | **TESTING, RESULTS AND DISCUSSION** | **22** |
| **5.** | **CONCLUSION** | **26** |
| **6.** | **REFERENCES** | **27** |
|  |  |  |

INTRODUCTION

Electronic spamming is the use of electronic messaging systems to send an unsolicited message (spam), especially advertising, as well as sending messages repeatedly on the same site. While the most widely recognized form of spam is email spam, the term is applied to similar abuses in other media: instant messaging spam, Usenet newsgroup spam, Web search engine spam, spam in blogs, wiki spam, online classified ads spam, mobile phone messaging spam, Internet forum spam, junk fax transmissions, social spam, spam mobile apps, television advertising and file sharing spam.The source and identity of the sender is anonymous and there is no option to cease receiving future e-mails. Span e-mail is usually sent by spam bot, which is program that continually sends out email. Often spammers will create a virus that install a spam bot into unsuspecting users computers and will use there internet connection and computer to send spam.

Spam e-mail are message randomly sent to multiple addressees by all sorts of groups, but mostly lazy advertisers and criminals who wish to lead you to phishing sites. The sites attempt to steal your personal, electronic, and financial information.

Bayesian Spam Detection/ Filtering is used to detect spam in an email. A Bayesian network is a representation of probabilistic relationships. This paper will show that Bayesian filtering can be simply implemented for a reasonably accurate text classifier and that it can be modified to make a significant impact on the accuracy of the filter.

Major approaches adopted towards spam filtering include text analysis, white and black lists of domain names and community based approaches. Text analysis of contents of mails is a widely used approach towards the spams. Many solutions deployable on server and client sides are available. Naive Bayes is one of the most popular algorithms used in these approaches. Spam Bayes and Mozilla Mail spam filter are examples of such solutions. But rejecting mails based on text analysis can be serious problem in case of false positives. Normally users and organizations would not want any genuine e-mails to be lost. Black list approach has been one of the earliest approaches tried for the filtering of spams. The strategy is to accept all the mails except the ones from the domain/e-mail ids. Explicitly blacklisted. With newer domains entering the category of spamming domains this strategy tends to not work so well. White list approach is the strategy of accepting the mails from the domains/addresses explicitly white listed and put others in a less priority queue, which is delivered only after sender responds to a confirmation request sent by the spam filtering system.

PROBLEM STATEMENT

Spamming is one of the major attacks that accumulate the large number of compromised machines by sending unwanted messages, viruses and phishing through emails. We have chosen this project because now days there are lot of people trying to fool you just by sending you fake e-mails like you have won 1000 dollars, this much amount is deposited in your account once you open this link then they will track you and try to hack your information. Sometimes relevant e-mails are considered as spam emails.

• Unwanted email irritating Internet consumers.

• Critical email messages are missed and/or delayed.

• Consumers change ISP's all the time looking for consistent email delivery.

• Loss of Internet performance and bandwidth.

• Millions of compromised computers.

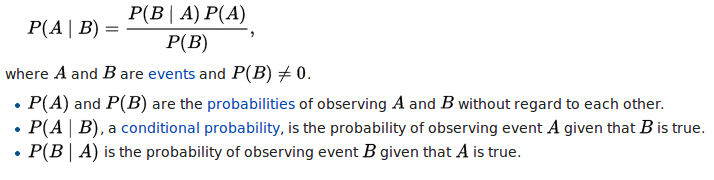
• Billions of dollars lost worldwide.

• Identity Theft.

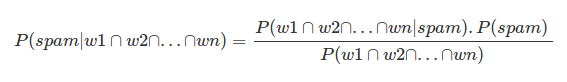
• Increase in Worms and Trojan Horses.

• Spam can crash mail servers and fill up hard drive

BAYE’S THEOREM



We have a message m = (w*1*, w*2*, . . . . , w*n*), where (w*1*, w*2*, . . . . , w*n*) is a set of unique words contained in the message. We need to find



If we assume that occurrence of a word are independent of all other words, we can simplify the above expression to

Image for post

In order to classify we have to determine which is greater

Image for post

SOURCE CODE

1.PRE-PROCESSING

#PRE PROCESSING

from os import walk

from os.path import join

import pandas as pd

import time

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import nltk

from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

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

from bs4 import BeautifulSoup

from wordcloud import WordCloud

from PIL import Image

EXAMPLE\_FILE = 'SpamData/01\_Processing/practice\_email.txt'

SPAM\_1\_PATH = 'SpamData/01\_Processing/spam\_assassin\_corpus/spam\_1'

SPAM\_2\_PATH = 'SpamData/01\_Processing/spam\_assassin\_corpus/spam\_2'

NONSPAM\_1\_PATH = 'SpamData/01\_Processing/spam\_assassin\_corpus/easy\_ham\_1'

NONSPAM\_2\_PATH = 'SpamData/01\_Processing/spam\_assassin\_corpus/easy\_ham\_2'

EMAIL\_TEXT\_JSON\_PATH = 'SpamData/01\_Processing/email-text.json'

WORD\_ID\_FILE = 'SpamData/01\_Processing/word-by-id.csv'

SPAM\_CAT = 1

NONSPAM\_CAT = 0

VOCABULARY\_SIZE = 2500

TRAINING\_DATA = 'SpamData/02\_Training/train-data.txt'

TESTING\_DATA = 'SpamData/02\_Training/test-data.txt'

WHALE\_FILE = 'SpamData/01\_Processing/wordcloud\_resources/whale-icon.png'

SKULL\_FILE = 'SpamData/01\_Processing/wordcloud\_resources/skull-icon.png'

THUMBS\_UP\_FILE = 'SpamData/01\_Processing/wordcloud\_resources/thumbs-up.png'

THUMBS\_DOWN\_FILE = 'SpamData/01\_Processing/wordcloud\_resources/thumbs-down.png'

CUSTOM\_FONT\_FILE = 'SpamData/01\_Processing/wordcloud\_resources/OpenSansCondensed-Bold.ttf'

def get\_email\_body(path):

for root, dirname, filenames in walk(path):

for file\_name in filenames:

filepath = join(root,file\_name)

stream = open(filepath, encoding='latin\_1')

is\_body = False

lines = []

for line in stream:

if is\_body:

lines.append(line)

elif line == '\n':

is\_body=True

stream.close()

email\_body = '\n'.join(lines)

yield file\_name, email\_body

def dataframe\_from\_directory(path,classification):

rows = []

rows\_name = []

for file\_name, email\_body in get\_email\_body(path):

rows.append({'MESSAGE' : email\_body, 'CATEGORY' : classification})

rows\_name.append(file\_name)

return pd.DataFrame(rows,index=rows\_name)

spam\_emails.shape

non\_spam\_emails = dataframe\_from\_directory(NONSPAM\_1\_PATH,NONSPAM\_CAT)

non\_spam\_emails = non\_spam\_emails.append(dataframe\_from\_directory(NONSPAM\_2\_PATH,NONSPAM\_CAT))

print(non\_spam\_emails.shape)

non\_spam\_emails.head()

data = pd.concat([spam\_emails,non\_spam\_emails])

print(f' shape of entire dataframe is {data.shape}')

document\_ids = range(0,len(data.index))

data['DOC\_ID'] = document\_ids # column to hold index 0-5796

data['FILE\_NAME'] = data.index #creating separate column to hold original file names

data.set\_index('DOC\_ID',inplace=True) #changing index from filenales to DOC\_ID

data.head()

data.to\_json(EMAIL\_TEXT\_JSON\_PATH)

data['CATEGORY'].value\_counts()

amount\_of\_spam = data['CATEGORY'].value\_counts()[1]

amount\_of\_ham = data['CATEGORY'].value\_counts()[0]

category\_names = ['spam' , 'legit emails']

sizes = [amount\_of\_spam ,amount\_of\_ham]

chart\_colors = ['#b2ebf2','#e97171']

plt.figure(figsize=(6,6),dpi = 100) # dpi: density of pixels per inch

plt.pie(sizes ,labels=category\_names,textprops={'fontsize' : 14} ,startangle=90,colors=chart\_colors,explode=[0,0.1],

autopct='%1.0f%%')

plt.show()

#NLP

nltk.download('punkt')

stop\_words = set(stopwords.words('english')) #removing stopwords

stopwords

def clean\_messages\_no\_html(message , stemmer = PorterStemmer()

,stop\_words = set(stopwords.words('english'))):

filtered\_words = []

soup = BeautifulSoup(message , 'html.parser')

cleaned\_text = soup.get\_text()

words = word\_tokenize(cleaned\_text.lower())

for word in words:

if word not in stop\_words and word.isalpha(): #checking word in stopwords ? and removing punctuation

filtered\_words.append(stemmer.stem(word)) #stemming word

return filtered\_words

data[data['CATEGORY'] == 1].shape

data[data['CATEGORY'] == 0].shape

doc\_ids\_ham = data[data['CATEGORY'] == 0].index

doc\_ids\_spam = data[data['CATEGORY'] == 1].index

%%time

nested\_list = data['MESSAGE'].apply(clean\_messages\_no\_html) #List of lists

nested\_list.head()

nested\_list\_spam = nested\_list.loc[doc\_ids\_spam]

nested\_list\_ham = nested\_list.loc[doc\_ids\_ham]

print(type(nested\_list\_ham))

flat\_list\_spam = []

flat\_list\_ham = []

for sublist in nested\_list\_spam:

for word in sublist:

flat\_list\_spam.append(word)

for sublist in nested\_list\_ham:

for word in sublist:

flat\_list\_ham.append(word)

print(type(flat\_list\_ham))

# Converting list to pandas series

normal\_words = pd.Series(flat\_list\_ham).value\_counts()

# print(f'total number of normal words : {normal\_words.shape[0]}')

print(f'total number of unique normal words : {normal\_words.shape[0]}')

spammy\_words = pd.Series(flat\_list\_spam).value\_counts() # value\_counts() holds only unique words

# print(f'total number of spammy words : {spammy\_words.shape[0]}')

print(f'total number of unique spammy words : {spammy\_words.shape[0]}')

print(f'Most used normal words: \n\n{normal\_words[:10]}\n')

print(f'\nMost used spammy words: \n\n{spammy\_words[:10]}\n')

# Generating text as string for spam messages

spam\_str = ' '.join(flat\_list\_spam)

icon = Image.open(THUMBS\_UP\_FILE)

image\_mask = Image.new(mode='RGB',size=icon.size,color=(255,255,255))

image\_mask.paste(icon,box=icon)

rgb\_array = np.array(image\_mask)

word\_cloud = WordCloud(mask=rgb\_array,max\_words=2000,background\_color='white',max\_font\_size=300,

colormap='gist\_heat',font\_path=CUSTOM\_FONT\_FILE)

word\_cloud.generate(spam\_str.upper())

plt.figure(figsize=(16,8))

plt.imshow(word\_cloud,interpolation='bilinear')

plt.axis('off')

plt.show()

# Generating text as string for ham messages

ham\_str = ' '.join(flat\_list\_ham)

icon = Image.open(THUMBS\_DOWN\_FILE)

image\_mask = Image.new(mode='RGB',size=icon.size,color=(255,255,255))

image\_mask.paste(icon,box=icon)

rgb\_array = np.array(image\_mask)

word\_cloud = WordCloud(mask=rgb\_array,max\_words=2000,background\_color='white',max\_font\_size=300,

colormap='gist\_heat',font\_path=CUSTOM\_FONT\_FILE)

word\_cloud.generate(ham\_str.upper())

plt.figure(figsize=(16,8))

plt.imshow(word\_cloud,interpolation='bilinear')

plt.axis('off')

plt.show()

stemmed\_nested\_list = data['MESSAGE'].apply(clean\_messages\_no\_html) # nested list of words in each emails bodies

flat\_stemmed\_list = [item for sublist in stemmed\_nested\_list for item in sublist]

unique\_words = pd.Series(flat\_stemmed\_list).value\_counts()

print(f' Total number of unique words : {unique\_words.shape[0]}')

unique\_words.head()

frequent\_words = unique\_words[0:VOCABULARY\_SIZE]

print( f'Most 10 frequently used words : \n {frequent\_words[0:10]}')

word\_ids = list(range(0,VOCABULARY\_SIZE))

vocab = pd.DataFrame({'VOCAB\_WORD':frequent\_words.index.values})

print(type(vocab))

vocab.index.name= 'WORD\_ID'

vocab.head()

vocab.to\_csv(WORD\_ID\_FILE,index\_label=vocab.index.name,header=vocab['VOCAB\_WORD'].name)

print(stemmed\_nested\_list)

word\_column\_df = pd.DataFrame.from\_records(stemmed\_nested\_list.to\_list())

word\_column\_df.head()# converting pandas series to list of list

word\_column\_df.shape

X\_train,X\_test,y\_train,y\_test = train\_test\_split(word\_column\_df, data['CATEGORY'], test\_size = 0.3, random\_state = 42)

X\_train.index.name = X\_test.index.name = 'DOC\_ID'

X\_train.head()

y\_train.head()

word\_index = pd.Index(vocab['VOCAB\_WORD'])

print(type(word\_index))

print(len(word\_index))

word\_index

def make\_sparse\_matrix(df, indexed\_words, labels):

'''

returns sparse matrix as dataframe

df : A dataframe with words in the column and document\_id as index (X\_train , X\_test)

indexed\_words : index of words order by word\_id

labels : category as pd series (y\_train,y\_test)

'''

nr\_rows = df.shape[0]

nr\_cols = df.shape[1]

dict\_list = [] # list of dictionaries

word\_set = set(indexed\_words)

for i in range(nr\_rows):

for j in range(nr\_cols):

word = df.iat[i,j]

if word in word\_set:

doc\_id = df.index[i]

word\_id = indexed\_words.get\_loc(word)

category = labels.at[doc\_id]

item = {'LABEL' : category , 'DOC\_ID' : doc\_id,'WORD\_ID' : word\_id,'OCCURENCE' : 1}

dict\_list.append(item)

return pd.DataFrame(dict\_list)

%%time

sparse\_train\_df= make\_sparse\_matrix(X\_train,word\_index,y\_train)

sparse\_train\_df.head()

sparse\_train\_df['OCCURENCE'].isnull().any()

sparse\_train\_df.shape

training\_grouped\_data = sparse\_train\_df.groupby(['DOC\_ID','WORD\_ID','LABEL']).sum()

training\_grouped\_data.head()

training\_grouped\_data = training\_grouped\_data.reset\_index()

training\_grouped\_data.head()

training\_grouped\_data.shape

np.savetxt(TRAINING\_DATA,training\_grouped\_data,fmt='%d')

%%time

sparse\_test\_df= make\_sparse\_matrix(X\_test,word\_index,y\_test)

sparse\_test\_df.head()

testing\_grouped\_data = sparse\_test\_df.groupby(['DOC\_ID','WORD\_ID','LABEL']).sum().reset\_index()

np.savetxt(TESTING\_DATA,testing\_grouped\_data,fmt='%d')

testing\_grouped\_data.columns

sparse\_test\_df.shape

2.TRAINING

import numpy as np

import pandas as pd

VOCABULARY\_SIZE = 2500

TRAINING\_DATA = 'SpamData/02\_Training/train-data.txt'

TESTING\_DATA = 'SpamData/02\_Training/test-data.txt'

TEST\_FEATURE\_MATRIX = 'SpamData/03\_Testing/test-features.txt'

TEST\_TARGET\_FILE = 'SpamData/03\_Testing/test-target.txt'

TOKEN\_SPAM\_PROB\_FILE = 'SpamData/03\_Testing/prob-spam.txt'

TOKEN\_HAM\_PROB\_FILE = 'SpamData/03\_Testing/prob-ham.txt'

TOKEN\_ALL\_PROB\_FILE = 'SpamData/03\_Testing/prob-all-tokens.txt'

sparse\_train\_data = np.loadtxt(TRAINING\_DATA, delimiter=' ',dtype='int')

sparse\_test\_data = np.loadtxt(TESTING\_DATA, delimiter=' ',dtype='int')

sparse\_train\_data.shape

sparse\_test\_data.shape

sparse\_train\_data[:5] # columns = doc\_id, word\_id, category, occurence

print(f'Number of rows in training file : {sparse\_train\_data.shape[0]}')

print(f'Number of rows in testing file : {sparse\_test\_data.shape[0]}')

print(f' Number of emails in training file : {np.unique(sparse\_train\_data[: , 0]).size}')

print(f' Number of emails in testing file : {np.unique(sparse\_test\_data[: , 0]).size}')

#Creating empty dataframe

column\_names = ['DOC\_ID'] + ['CATEGORY'] + list(range(0 , VOCABULARY\_SIZE))

column\_names[:5]

len(column\_names)

index\_names = np.unique(sparse\_train\_data[:,0])

index\_names

full\_train\_data = pd.DataFrame(index=index\_names, columns=column\_names)

full\_train\_data.head()

full\_train\_data.fillna(value=0,inplace=True)

#Creating Full Matrix from sparse matrix

In [16]:

def make\_full\_matrix(sparse\_matrix, nr\_words, doc\_idx=0,word\_idx=1,cat\_idx=2,freq\_idx=3):

'''

returns full matrix which contains occurences of each word in vocab in single row

sparse\_matrix -- input matrix (np array)

nr\_words -- size of vocabulary(2500)

doc\_idx -- DOC\_ID index in sparse matrix(0)

word\_idx -- 'WORD\_ID' index in sparse\_matrix(1)

cat\_idx -- 'CATEGORY' index (2)

freq\_idx -- 'OCCURENCES' index (3)

'''

column\_names = ['DOC\_ID'] + ['CATEGORY'] + list(range(0 , VOCABULARY\_SIZE))

doc\_id\_names = np.unique(sparse\_matrix[:,0])

full\_matrix = pd.DataFrame(index=doc\_id\_names,columns=column\_names)

full\_matrix.fillna(value=0,inplace=True)

for i in range(sparse\_matrix.shape[0]): # iterating through all rows in sparse matrix and extracting word occurences

doc\_nr = sparse\_matrix[i][doc\_idx] # doc\_idx = 0 first column

word\_id = sparse\_matrix[i][word\_idx]

label = sparse\_matrix[i][cat\_idx]

occurence = sparse\_matrix[i][freq\_idx]

full\_matrix.at[doc\_nr,'DOC\_ID'] = doc\_nr

full\_matrix.at[doc\_nr,'CATEGORY'] = label

full\_matrix.at[doc\_nr,word\_id] = occurence

full\_matrix.set\_index('DOC\_ID',inplace=True)

return full\_matrix

%%time

full\_train\_data = make\_full\_matrix(sparse\_train\_data,VOCABULARY\_SIZE)

full\_train\_data.head() # most frequent words are at begining

#DOC\_ID : emails id of each email in dataset

#CATEGORY : wether email is spam or not 1 : spam 0 : legit

#columns 0 1 2 3 4 .......2499 indicates word\_id of top 2500 words in vocabulary

#each box inside table indicates number of occurance of word in particular email

#calculating probability of spam

spam\_prob = full\_train\_data['CATEGORY'].sum() / full\_train\_data['CATEGORY'].size

print(' Probability os email being spam ' ,round(spam\_prob,3))

full\_train\_features = full\_train\_data.loc[:,full\_train\_data.columns != 'CATEGORY']

full\_train\_features.head() #excluding category

email\_lengths = full\_train\_features.sum(axis=1)

email\_lengths.shape

total\_wordcounts = email\_lengths.sum()

total\_wordcounts

spam\_lengths = email\_lengths[full\_train\_data.CATEGORY == 1]

spam\_lengths.shape

spam\_wordcount = spam\_lengths.sum()

spam\_wordcount

ham\_lengths = email\_lengths[full\_train\_data.CATEGORY == 0]

ham\_lengths.shape

ham\_wordcount = ham\_lengths.sum()

ham\_wordcount

print(f'Avg number of words in spam emails : {spam\_wordcount / spam\_lengths.shape[0]}')

print(f'Avg number of words in ham emails : {ham\_wordcount / ham\_lengths.shape[0]}')

#Summing the Tokens Occuring in spam

full\_train\_features.shape

train\_spam\_tokens = full\_train\_features.loc[full\_train\_data.CATEGORY == 1]

train\_spam\_tokens.head()

# Calculating number of time each word appeared in all spam emails

summed\_spam\_tokens = train\_spam\_tokens.sum(axis=0) + 1 # +1 to avoid zero by error while calculating probability

summed\_spam\_tokens.shape # axis = 0 to sum columns

summed\_spam\_tokens.head()

#Summing tokens occuring in ham emails

train\_ham\_tokens = full\_train\_features.loc[full\_train\_data.CATEGORY == 0]

train\_ham\_tokens.head()

# Calculating number of time each word appeared in all ham emails

summed\_ham\_tokens = train\_ham\_tokens.sum(axis=0) + 1 # +1 to avoid zero by error while calculating probability

summed\_ham\_tokens.shape # axis = 0 to sum columns

#p( token | spam ) : Probabilty that token ocuurs given email is spam

prob\_tokens\_spam = summed\_spam\_tokens / (spam\_wordcount + VOCABULARY\_SIZE) #devide by VOCABULARY\_SIZE coz we added 1 to

#summed\_spam tokens

prob\_tokens\_spam[:5]

prob\_tokens\_spam.sum()

#p( token | nonspam ) : Probabilty that token ocuurs given email is non-spam

prob\_tokens\_ham = summed\_ham\_tokens / (ham\_wordcount + VOCABULARY\_SIZE)

prob\_tokens\_ham.sum()

prob\_token\_all = full\_train\_features.sum(axis=0) / total\_wordcounts

prob\_token\_all

#Saving trained model

np.savetxt(TOKEN\_SPAM\_PROB\_FILE,prob\_tokens\_spam)

np.savetxt(TOKEN\_HAM\_PROB\_FILE,prob\_tokens\_ham)

np.savetxt(TOKEN\_ALL\_PROB\_FILE,prob\_token\_all)

#Prepare test data

sparse\_test\_data.shape

%%time

full\_test\_data = make\_full\_matrix(sparse\_test\_data,nr\_words=VOCABULARY\_SIZE)

X\_test = full\_test\_data.loc[:, full\_test\_data.columns != 'CATEGORY']

y\_test = full\_test\_data.CATEGORY

X\_test.shape

np.savetxt(TEST\_TARGET\_FILE,y\_test)

np.savetxt(TEST\_FEATURE\_MATRIX,X\_test)

3.TESTING

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

VOCABULARY\_SIZE = 2500

TEST\_FEATURE\_MATRIX = 'SpamData/03\_Testing/test-features.txt'

TEST\_TARGET\_FILE = 'SpamData/03\_Testing/test-target.txt'

TOKEN\_SPAM\_PROB\_FILE = 'SpamData/03\_Testing/prob-spam.txt'

TOKEN\_HAM\_PROB\_FILE = 'SpamData/03\_Testing/prob-ham.txt'

TOKEN\_ALL\_PROB\_FILE = 'SpamData/03\_Testing/prob-all-tokens.txt'

#features

X\_test = np.loadtxt(TEST\_FEATURE\_MATRIX,delimiter=' ')

#target

y\_test = np.loadtxt(TEST\_TARGET\_FILE,delimiter=' ')

#token probabilities

prob\_token\_spam = np.loadtxt(TOKEN\_SPAM\_PROB\_FILE,delimiter=' ')

prob\_token\_ham = np.loadtxt(TOKEN\_HAM\_PROB\_FILE,delimiter=' ')

prob\_all\_tokens = np.loadtxt(TOKEN\_ALL\_PROB\_FILE,delimiter=' ')

X\_test.shape

y\_test.shape

print(f' shape of dot product is : {X\_test.dot(prob\_token\_spam).shape}')

shape of dot product is : (1724,)

#Set the prior

P(spam \, | \, X ) = \frac{ \, P(X \, | \, spam \,) \, p(spam \,)} { P(X) }

PROB\_SPAM = 0.311

# Taking log to make it easier for calculations

# P(spam|X) = p(X|spam)\*p(spam)/p(X) can be written as P(spam|X) = log(p(X|spam)) + log(p(spam)) - log(p(X))

np.log(prob\_token\_spam)

# calculating probablity of emails being spam for all testing dataset based on values we got fron training model

joint\_log\_spam = X\_test.dot(np.log(prob\_token\_spam) - np.log(prob\_all\_tokens)) + np.log(PROB\_SPAM)

joint\_log\_spam[:5]

# calculating probablity of emails being non-spam for all testing dataset based on values we got fron training model

joint\_log\_ham = X\_test.dot(np.log(prob\_token\_ham) - np.log(prob\_all\_tokens)) + np.log(1-PROB\_SPAM)

joint\_log\_ham[:5]

joint\_log\_ham.size

joint\_log\_spam.size

checking for higher joint probability

$$ P(Spam \, | \, X \,) &gt; P(Ham \, | X ) $$

predictions = joint\_log\_spam > joint\_log\_ham

predictions[-8:] \* 1

y\_test[-8:]

predictions.shape

predictions[1333]

y\_test[1333]

predictions[980]

y\_test[980]

predictions[2]

y\_test[2]

predicted\_result = []

actual\_result = []

num = 100

for i in range(10):

predicted\_result.append(predictions[num])

actual\_result.append(y\_test[num])

num += 1

print(predicted\_result)

print(actual\_result)

[True, True, True, True, True, True, True, False, True, True]

[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Simplifying

Removing denominator P(X) as we are deviding that to both spam and ham emails so P(spam|X) > P(ham|X) remain uneffected

joint\_log\_spam = X\_test.dot(np.log(prob\_token\_spam)) + np.log(PROB\_SPAM)

joint\_log\_ham = X\_test.dot(np.log(prob\_token\_ham)) + np.log(1-PROB\_SPAM)

predictions = joint\_log\_spam > joint\_log\_ham

# output 1 : spam email 0 : legit email

def compare\_results(index):

predicted = predictions[index] \* 1

actual = y\_test[index]

print("predicted value = " ,predicted)

print("actual value = " ,actual)

if(predicted == actual):

print("model predicted correctly")

else:

print("model predicted wrongly")

compare\_results(1680);

compare\_results(790);

compare\_results(1719);

compare\_results(1500);

predicted value = 0

actual value = 0.0

model predicted correctly

predicted value = 0

actual value = 0.0

model predicted correctly

predicted value = 1

actual value = 0.0

model predicted wrongly

predicted value = 0

actual value = 0.0

model predicted correctly

# Getting results for consecutive 10 mails

def get\_results1(index):

predicted\_result = []

actual\_result = []

for i in range(10):

predicted\_result.append(predictions[index])

actual\_result.append(y\_test[index])

index += 1

print(predicted\_result)

print(actual\_result)

get\_results1(1714)

[False, False, False, False, False, True, False, False, False, False]

[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Metrics and Accuracy

#Accuracy

nr\_correct\_predictions = (y\_test == predictions).sum()

print(f'Number of emails classified correctly are : {nr\_correct\_predictions}')

nr\_wrong\_predictions = X\_test.shape[0] - nr\_correct\_predictions

print(f'Number of wrong predictions : {nr\_wrong\_predictions}')

#Number of emails classified correctly are : 1685

#Number of wrong predictions : 39

y\_test[(y\_test != predictions)]

predictions[(y\_test != predictions)]

accuracy = nr\_correct\_predictions/len(X\_test) \* 100

print(f' Accuracy of model is : {round(accuracy,2)}%')

Accuracy of model is : 97.74%

frac\_wrong = nr\_wrong\_predictions/len(X\_test) \* 100

print(f'Fraction of wrong predictions : {round(frac\_wrong,2)}%')

#Fraction of wrong predictions : 2.26%

Visualising the results

# Chart styling info

xaxis\_label = 'P(Non-Spam | X)'

yaxis\_label = 'P(Spam | X)'

plt.figure(figsize=(12,7))

plt.xlabel(xaxis\_label,fontsize = 14)

plt.ylabel(yaxis\_label,fontsize=14)

#Set limits

plt.xlim([-14000,1])

plt.ylim([-14000,1])

plt.scatter(joint\_log\_ham,joint\_log\_spam,color='navy')

plt.show()

# Chart styling info

xaxis\_label = 'P(Non-Spam | X)'

yaxis\_label = 'P(Spam | X)'

linedata = np.linspace(start=-14000,stop=1,num=1000)

plt.figure(figsize=(12,7))

plt.xlabel(xaxis\_label,fontsize = 14)

plt.ylabel(yaxis\_label,fontsize=14)

#Set limits

plt.xlim([-14000,1])

plt.ylim([-14000,1])

plt.scatter(joint\_log\_ham,joint\_log\_spam,color='navy',alpha=0.5,s=25)

plt.plot(linedata,linedata,color='orange')

plt.show()

# Chart styling info

xaxis\_label = 'P(Non-Spam | X)'

yaxis\_label = 'P(Spam | X)'

linedata = np.linspace(start=-14000,stop=1,num=1000) # for plotting diagonal line

plt.figure(figsize=(18,7))

#CREATING TWO SEPARATE CHARTS SIDE BY SIDE

#Chart number : 1

plt.subplot(1,2,1) # 1:rows 2:columns 1:index

plt.xlabel(xaxis\_label,fontsize = 14)

plt.ylabel(yaxis\_label,fontsize=14)

#Set limits

plt.xlim([-14000,1])

plt.ylim([-14000,1])

plt.scatter(joint\_log\_ham,joint\_log\_spam,color='navy',alpha=0.5,s=25)

plt.plot(linedata,linedata,color='orange')

#Chart number : 2

plt.subplot(1,2,2)

plt.xlabel(xaxis\_label,fontsize = 14)

plt.ylabel(yaxis\_label,fontsize=14)

#Set limits

plt.xlim([-2000,1]) # reducing limits to get clear scatters

plt.ylim([-2000,1])

plt.scatter(joint\_log\_ham,joint\_log\_spam,color='navy',alpha=0.5,s=5)

plt.plot(linedata,linedata,color='red')

plt.show()

sns.set\_style='whitegrid'

labels = 'Actual category'

#creating DF with xaxis\_label : column names for ham probs , yaxis\_label : column names for spam, labels :column name for labels

summary\_df = pd.DataFrame({xaxis\_label : joint\_log\_ham,yaxis\_label : joint\_log\_spam, labels : y\_test}) #concatinating all three

sns.lmplot(x=xaxis\_label,y=yaxis\_label,data=summary\_df, size=7,fit\_reg=False,

scatter\_kws={'alpha': 0.5 , 's' : 25},hue=labels)

plt.xlim([-2000,1]) # reducing limits to get clear scatters

plt.ylim([-2000,1])

plt.plot(linedata,linedata,color='black')

plt.legend(('Decision boundary','non-spam','spam'),loc='lower right',fontsize=14)

plt.show()

#False positives and False negatives

np.unique(predictions,return\_counts=True) # 1136 : ham 588 : spam ; according to model

true\_positives = (predictions == 1) & (y\_test == 1)

true\_positives.sum()

true\_negatives = (predictions == 0) & (y\_test == 0)

true\_negatives.sum()

false\_positives = (predictions == 1) & (y\_test == 0)

false\_positives.sum()

false\_negatives = (predictions == 0) & (y\_test == 1)

false\_negatives.sum()

e[0]

#Recall score

recall-score = true positives / (true positives + false negatives)

recall\_score = true\_positives.sum()/(true\_positives.sum() + false\_negatives.sum())

recall\_score

print(f'Recall score is : {round((recall\_score),4) \* 100}% ')

Recall score is : 96.6%

# Precision

precision = true\_positives / (true\_positives + false\_positives)

In [132]:

precision = true\_positives.sum()/(true\_positives.sum() + false\_positives.sum())

precision

print(f'Precision is : {round((precision),4) \* 100}% ')

#Precision is : 96.77%

F-score (combining precision and recall score to get max value)Â¶

f\_score = 2 \* (precision \* recall\_score)/(precision + recall\_score)

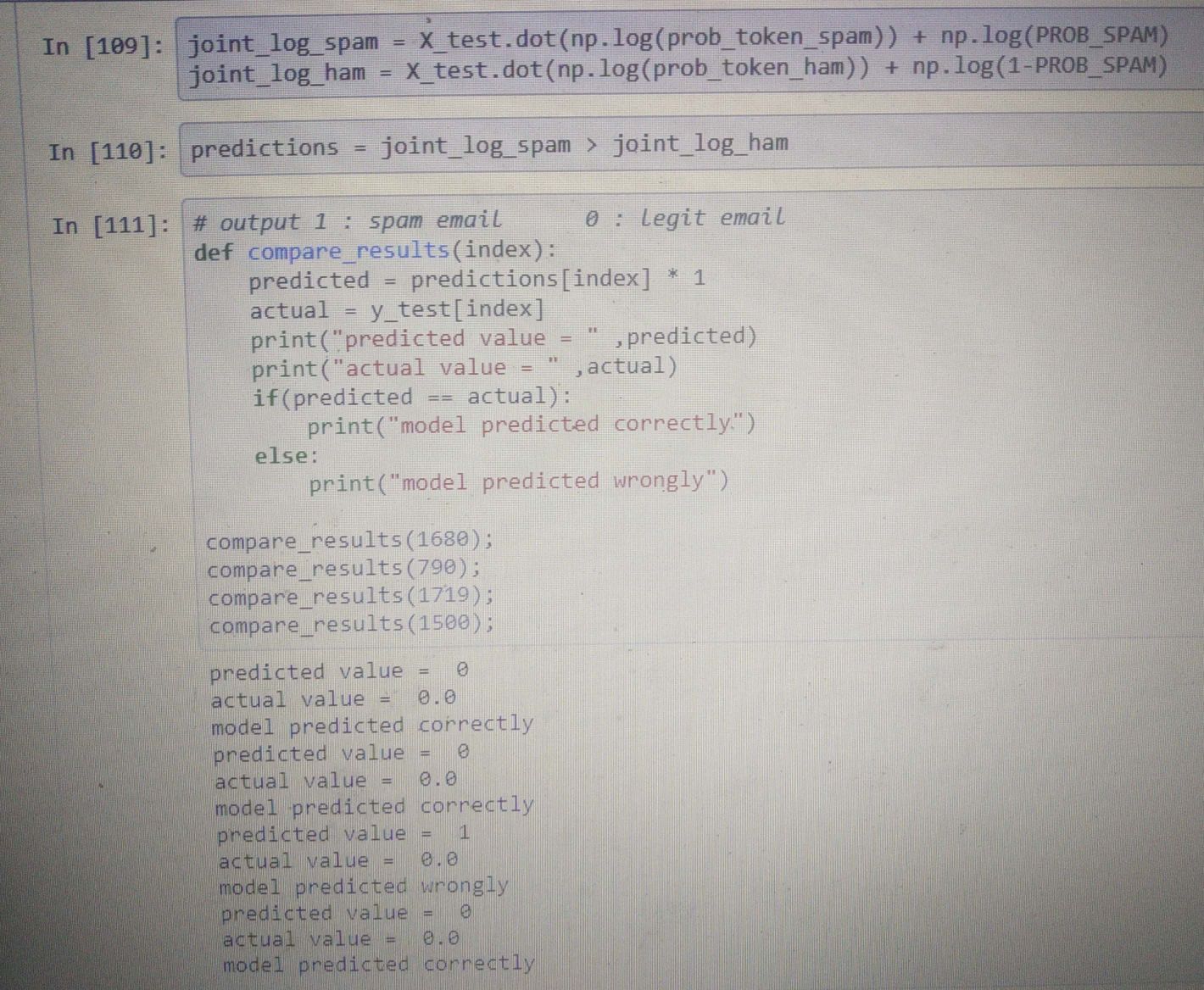
print(f'F score is : {round(f\_score,4) \* 100}')

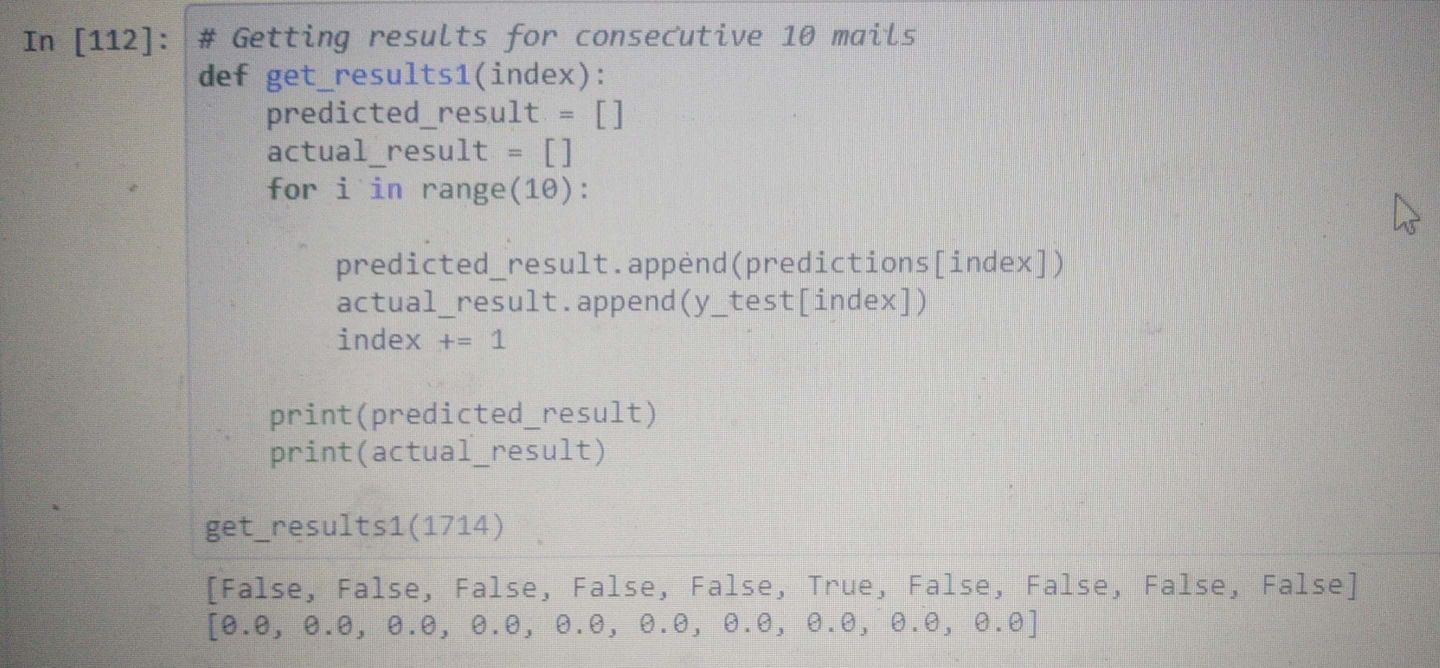
F score is : 96.69

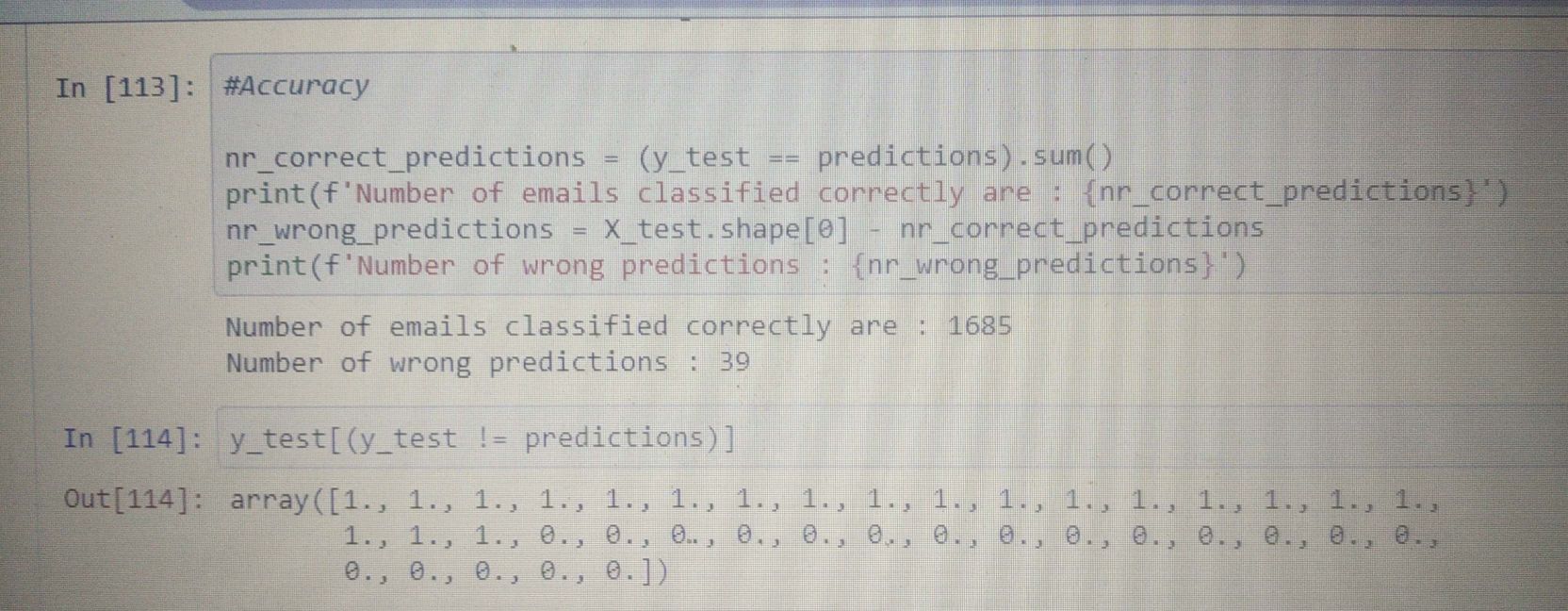
TESTING,RESULTS AND DISCUSSION

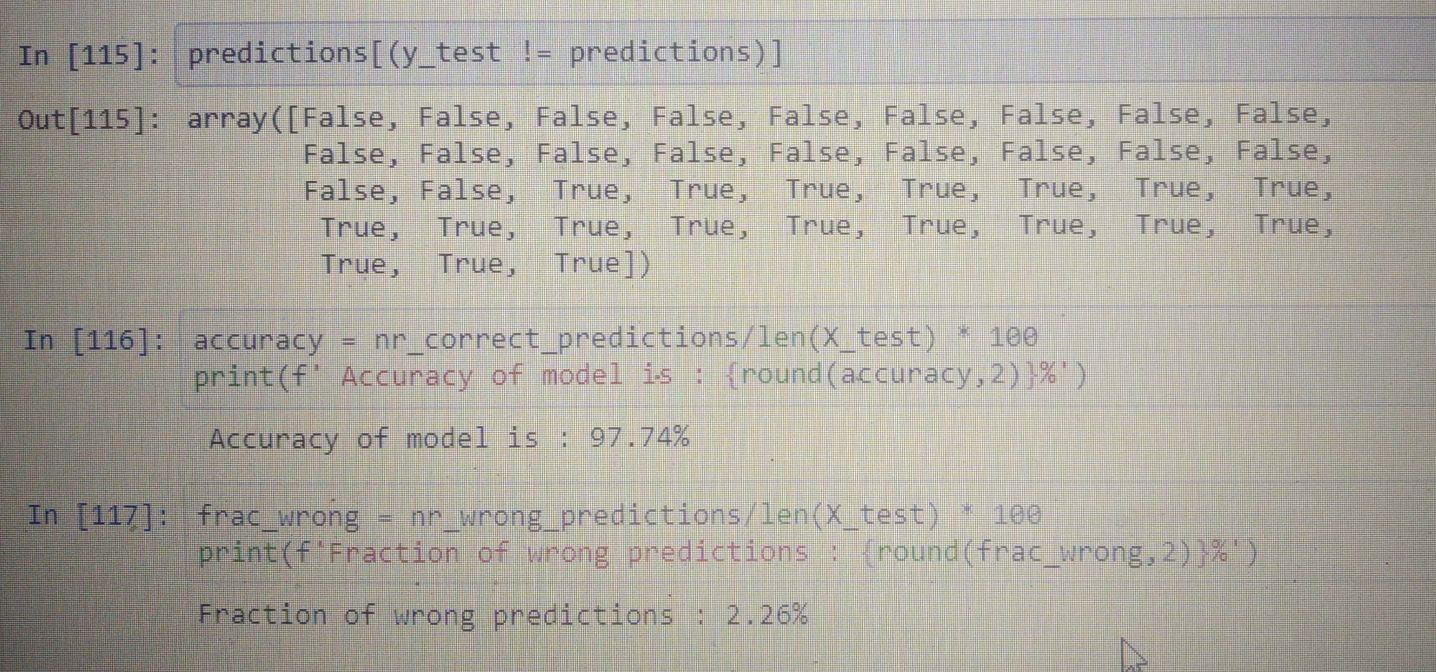
In order to create algorithm for this, we need to teach our program what a spam email looks like and what non-spam emails look like. . We also need a way to test the accuracy of our spam filter. One idea would be to test it on the same data that we used for training. However, this can lead to a major problem in ML called overfitting which means that our model is too biased towards the training data and may not work as well on elements outside of that training set. One common way to avoid this is to split our labeled data for training/testing. This ensures we test on different data than we trained on. It’s important to note that we need a mix of both spam and non-spam data in our data sets, not just the spam ones. We really want our training data to be as similar to real email data as possible.

Some of the testing cases and predictions are showed below

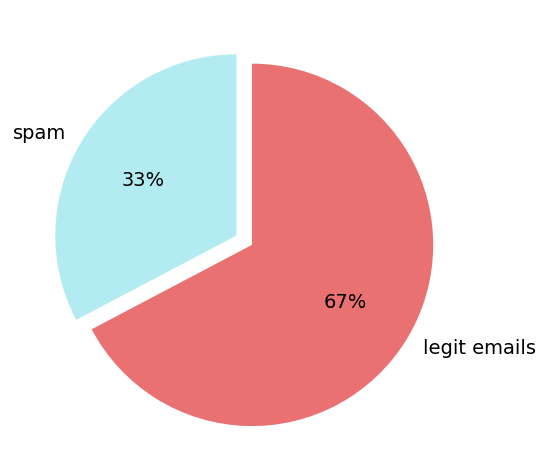


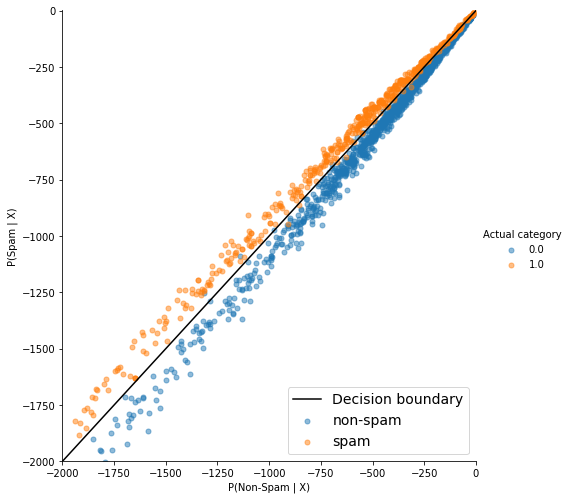






Some of the graphs and visualization areas from our projects are listed below.





CONCLUSION

We are able to classify the emails as spam or non-spam. With high number of emails lots if people using the system it will be difficult to handle all possible mails as our project deals with only limited amount of corpus.

There is a wide scope of enhancement in our project. Following enhancements can be done: Filtering of spams can be done on the basis of its contents. The spam email classification is very important in classifying e-mails and to separate e-mails that are spam or non-spam. This method can be used by big organization to distinguish good mails that is only the mails they wish to receive.

REFERENCES

<https://github.com/Anuranjansarsam/NLP-Mini-Project.git>

<https://towardsdatascience.com/spam-classifier-in-python-from-scratch-27a98ddd8e73>