

(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 30-6-2021

Report on Mini Project

Spam Message Checker using Naive Bayes Classifier

#### Course Code : 18CS605 Course Name : Machine Learning Lab

##### Semester: VI SEM Section: A

**Submitted To:**

## Mrs. Divya Jennifer D’Souza

## Assistant Professor

## Department of Computer Science and Engineering

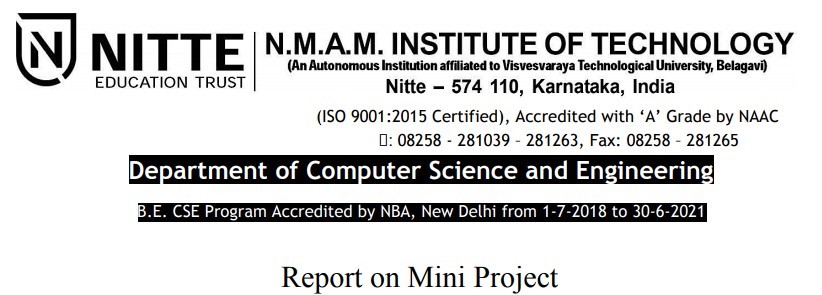
**Submitted By:**

Aditya Rao Chokkadi(4NM18CS007)

Aditya S Nambiar(4NM18CS008)

**Date of submission:** 31-5-2021

#### Signature of Course Instructor



**CERTIFICATE**

Certified that the mini project entitled

‘Spam Message Checker using Naive Bayes Classifier’

Is a bona fide work carried on by

**Aditya Rao Chokkadi (4NM18CS007)**

**Aditya S Nambiar (4NM18CS008)**

In partial fulfillment of requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering prescribed by Visvesvaraya Technological University, Belgaum during the year 2020-

2021

It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report. The mini project has been approved as it satisfies the academic requirements in respect of the project work prescribed for the Bachelor of Engineering Degree.

#### Name and Sign of Guide Name & Sign of HOD Mrs. Divya Jennifer D’Souza Dr. K R Udaya Kumar Reddy

#### Assistant Professor, CSE HOD, CSE

**ACKNOWLEDGEMENT**

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

We thank Dr. N Niranjan Chiplunkar, Principal of NMAMIT, Mrs. Divya Jennifer D’Souza, Assistant Professor, Department of CSE, Dr. K R Udaya Kumar Reddy for the support given through us throughout the entire duration of program.

Finally, we thank all our friends for all the honest opinions and feedback.

**Aditya Rao Chokkadi**

**Aditya S Nambiar**

# 

# ABSTRACT

To build a spam checker using Python and the multinomial Naive Bayes algorithm. Goal is to code a spam checker from scratch that classifies messages with accuracy greater than 80%. To build the spam checker, a dataset of 5,572 messages is used.

It will take a spam message as input and its corresponding output will tell us whether the message is a spam message or a ham (non-spam) message. The python libraries that are used to build the spam checker are as follows:-

1. Pandas (to work with dataframes)
2. Nltk (to tokenize, remove stopwords and stem the messages)
3. Tkinter (to create GUI (graphical user interface))

**Table of Contents**

**Contents Page No.**

Title Page 1

Certificate 2

Acknowledgement 3

Abstract 4

Table of Contents 5

Introduction 6

Literature Survey 7

Design 8

Implementation 9-19

Result 20-22

Conclusion 23

References 24

# INTRODUCTION

***Spam*** is digital junk mail: unsolicited communications sent in bulk over the internet or through any electronic messaging system. Spam is always unrequested. It’s annoying, it’s usually promotional, it’s sent to loads of people, and it’s coming whether we asked for it or not. If the definition of spam is unsolicited bulk messages, spamming is the act of sending these messages, and a person who engages the practice is a spammer.

In today’s world, spam filtering is a must to protect your business. Spam is not going away. It is estimated that 70 percent of all email sent globally is spam, and the volume of spam continues to grow because spam remains a lucrative business. Spammers get ever more sophisticated and creative in their tactics to get their messages into your inboxes and wreak their havoc. Spam filtering solutions must continually be updated to address this evolving threat.

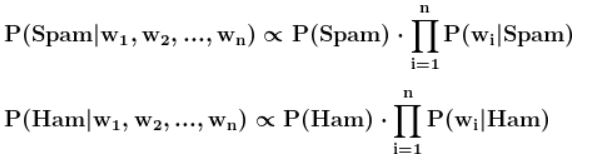
The various **types of spam** are:-

1. ***Email Spam***-  It clogs up your inbox and distracts you from the emails you actually want to read.
2. ***SEO Spam***- Also known as spamdexing, this is the abuse of search engine optimization (SEO) methods to improve search rankings for the spammer’s website.
3. ***Social networking Spam*- Spammers** spread their spam via fake “throwaway” accounts on popular social networking platforms.
4. ***Mobile Spam*-** It’s spam in SMS form. In addition to spammy text messages, some spammers also utilize push notifications to draw your attention to their offers.
5. ***Messaging Spam*-** Like email spam, but quicker. Spammers blast their messages out on instant messaging platforms including WhatsApp, Skype, and Snapchat.

**LITERATURE SURVEY**

In order to build the spam filter, multinomial naïve bayes algorithm is used. Multinomial Naive Bayes algorithm is a probabilistic learning method that is mostly used in Natural Language Processing (NLP). The algorithm is based on the Bayes theorem and predicts the tag of a text such as a piece of email or newspaper article. It calculates the probability of each tag for a given sample and then gives the tag with the highest probability as output.

In order to build a spam checker, multinomial naïve bayes is used. Its formula is as follows:-

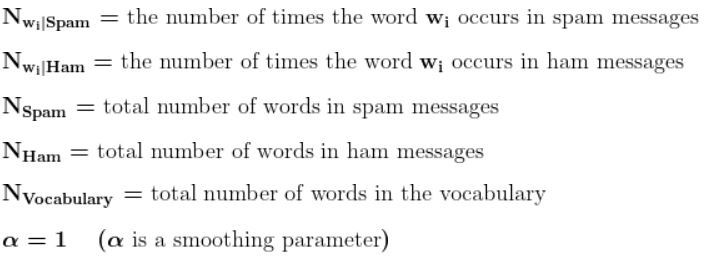


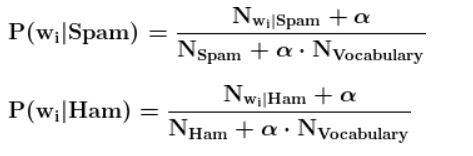
where "w1" is the first word, and w1,w2, **....** , wn is the entire message.

If P(Spam | w1,w2, ..., wn) is greater than P(Ham | w1,w2, .... , wn), then the message is spam. Otherwise, the message is Ham (non-Spam).

To calculate P(wi|Spam) and P(wi|Ham), we need to use separate equations:-

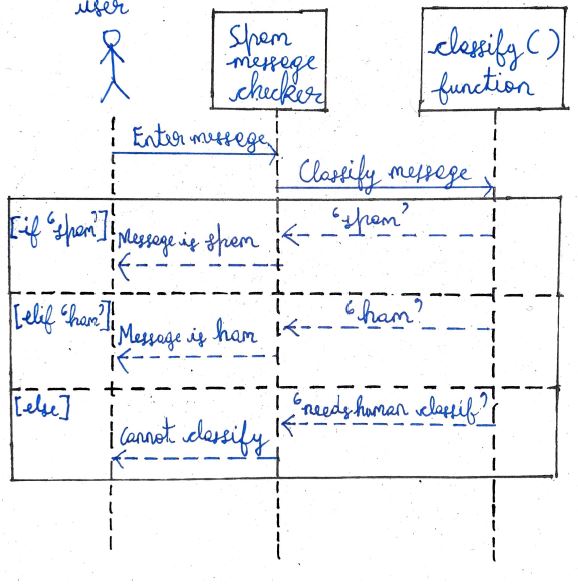
Where,





# DESIGN

# *Sequence Diagram:-*



**IMPLEMENTATION**

1. First the required libraries: tkinter, pandas and nltk are imported:-

from tkinter import \*

from tkinter import messagebox

import pandas as pd

from nltk.tokenize import RegexpTokenizer

from nltk.stem import PorterStemmer

from nltk.corpus import stopwords

1. The dataframe is created by making use of read\_csv() function present in pandas library:-

df=pd.read\_csv("spam.csv",encoding='ISO-8859-1');

Any unwanted(extra) columns are removed by making of **pd.drop().**

1. The dataframe is split into training\_set (80%) and testing\_set (20%) :-

data\_randomized = df.sample(frac=1, random\_state=1);

training\_test\_index = round(len(data\_randomized) \* 0.8);

training\_set=data\_randomized[:training\_test\_index].reset\_index(drop=True);

test\_set = data\_randomized[training\_test\_index:].reset\_index(drop=True);

After this, the training\_set dataframe is converted to an array called data which is further split into two arrays: spam\_messages and classification:-

data=training\_set.to\_numpy();

spam\_messages=data[:, 1]; # array where each element is a spam message

classification= data[:, 0]; # array where each element is either 'ham' or 'spam'

1. An instance of RegexpTokenizer (tokenizer) , a set of stopwords (sw) and a PorterStemmer() instance (ps) are created:-

tokenizer=RegexpTokenizer('\w+'); # used to break sentences into words

sw=set(stopwords.words('english')); # set containing stopwords of english language is created

ps=PorterStemmer(); # used to replace certain words with their simpler forms eg: 'running' becomes 'run'(meaning is not changed)

**Stopwords** are words such as ‘an’, ‘the’, ‘as’ etc that are used for grammatical purposes and do not describe the actual content of a message.

**Stemming** is the process of converting a word to its simplest form. For exampler the word ‘dancing’ would be stemmed to ‘dance’ always. This wont be a problem as both of them convey the same meaning.

1. A function called operation() is defined which takes as input an array or list of messages and performs tokenization, removal of stopwords and stemming operations on each of the messages present in array or list and at the end it returns a list called operated\_messages which contains the cleaned (operated) messages. This cleaning is required in order to increase the accuracy of Naïve Bayes Classifier:-

def operation(spam\_messages):

operated\_messages=[ ];

for spam in spam\_messages: # for each spam message

spam=spam.lower(); # convert letters to lowercase

tokenized\_spam=tokenizer.tokenize(spam); #break sentence to words

sw\_removed\_spam=[word for word in tokenized\_spam if word not in sw];

stemmed\_spam=[ps.stem(word) for word in sw\_removed\_spam];

#removing stopwords and performing stemming on each word

clean\_message=' '.join(stemmed\_spam); #joining to sentence

operated\_messages.append(clean\_message); #adding to list

return operated\_messages;

1. The spam\_messages array which we created earlier is passed as input to the operation() function and the returned list is stored in operated\_messages variable:-

operated\_messages=operation(spam\_messages); #list containing clean messages is obtained

The training\_set dataframe’s second column (which contains the messages) is replaced with the cleaned messages present in operated\_messages list. After that each message is split into a list of words in the training\_set dataframe:-

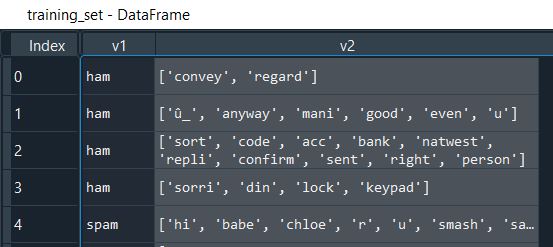
operated\_messages=operation(spam\_messages); #cleaned messages

training\_set['v2']=operated\_messages; #putting the cleaned message in 'v2' col.

training\_set['v2']=training\_set['v2'].str.split(); #creating word lists in each row of 'v2' column

After putting cleaned messages in ‘v2’ column:-



After str.split():-

.

1. Vocabulary list is created. It will contain all the unique words across the training set’s messages:-

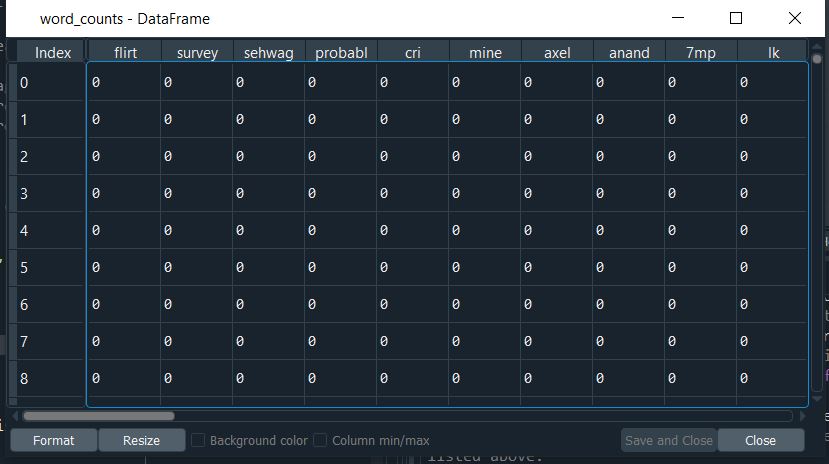
vocabulary=[ ];

for words\_list in training\_set['v2']:

for word in words\_list:

vocabulary.append(word);

vocabulary=list(set(vocabulary)); #6484 unique words

1. A dataframe word\_counts needs to be created where each word from the vocabulary has its own column. It is as shown below:-

Each cell contains the no of times that particular word appears in the corresponding message.

In order to this, a dictionary **word\_counts\_per\_sms** is created which is later converted to a dataframe by making use of pd.DataFrame() function:-

word\_counts\_per\_sms = {unique\_word: [0] \* len(training\_set['v2']) for unique\_word in vocabulary};

for index, words\_list in enumerate(training\_set['v2']):

for word in words\_list:

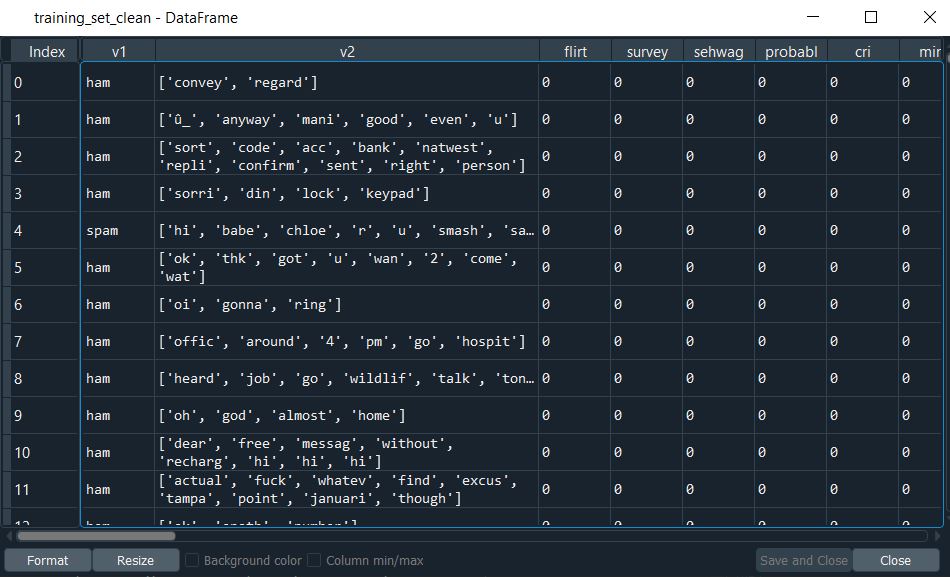
word\_counts\_per\_sms[word][index] += 1;

word\_counts = pd.DataFrame(word\_counts\_per\_sms);

training\_set and word\_counts dataframes are combined together (columnwise)

to get a new dataframe called training\_set\_clean by making use of pd.concat()

function:-

training\_set\_clean = pd.concat([training\_set, word\_counts], axis=1);

.

9) Now we begin with the implementation of Naïve Bayes Classifier. The **training\_set\_clean** dataframe is divided into two dataframes : **spam\_messages** (where v1==’spam’ ) and **ham\_messages** (where v1==’ham’) :-

spam\_messages = training\_set\_clean[training\_set\_clean['v1'] == 'spam']; #spam dataframe

ham\_messages = training\_set\_clean[training\_set\_clean['v1'] == 'ham']; #ham dataframe.

10) In the naïve bayes formula, the constant terms are P(Spam), P(Ham),

NSpam, NHam , NVocabulary and α (alpha). These values can be calculated beforehand as their values don’t change later on.

P(Spam) and P(Ham) are calculated as follows:-

p\_spam = len(spam\_messages) / len(training\_set\_clean); #(no of msg)/total

p\_ham = len(ham\_messages) / len(training\_set\_clean);

Nspam and Nham are calculated as follows:-

n\_words\_per\_spam\_message = spam\_messages['v2'].apply(len); #gives a series

n\_spam = n\_words\_per\_spam\_message.sum();

n\_words\_per\_ham\_message = ham\_messages['v2'].apply(len);

n\_ham = n\_words\_per\_ham\_message.sum();

NVocabulary is calculated as follows:-

n\_vocabulary = len(vocabulary); #total no of words in vocabulary

α is a constant called smoothing parameter and is set to 1:

alpha = 1;

11) Two dictionaries: parameters\_spam and parameters\_ham are created in order to store the values of P(wi|Spam) and P(wi|Ham) respectively. Each key in the dictionary will be a word from the vocabulary and each value will be the respective probability value. At the start the values are assigned as 0 :-

parameters\_spam = {unique\_word:0 for unique\_word in vocabulary};

parameters\_ham = {unique\_word:0 for unique\_word in vocabulary};

For each word in the vocabulary, P(wi|Spam) and P(wi|Ham) is calculated using the formula mentioned in an earlier section and these values are stored in the dictionaries paramenters\_spam and parameters\_ham respectively:-

for word in vocabulary:

n\_word\_given\_spam = spam\_messages[word].sum() ;

p\_word\_given\_spam=(n\_word\_given\_spam + alpha) / (n\_spam+alpha\*n\_vocabulary)

parameters\_spam[word] = p\_word\_given\_spam;

n\_word\_given\_ham = ham\_messages[word].sum() ;

p\_word\_given\_ham = (n\_word\_given\_ham + alpha) / (n\_ham + alpha\*n\_vocabulary);

parameters\_ham[word] = p\_word\_given\_ham;

12) Since we have all the required values, we can finally calculate P(Spam|Message) and P(Ham|Message) where Message= w1w2w3-------wn,

each wi represents a word of the message.

If the value of P(Spam|Message) is greater than the value of P(Ham|Message) then the given message is classified as a Spam message.

If the value of P(Spam|Message) is less than the value of P(Ham|Message) then the given message is classified as a Ham message.

If the value of P(Spam|Message) is equal to the value of P(Ham|Message) then the given message cannot be classified by the classifier as it is equally likely to be ham or spam. So, it is upto a human to decide whether it is spam or not.

In order to implement this logic a fuction called classify() is defined which takes a message as input. It takes the message and passes it to the operation() function in order to clean the message.

The cleaned message is then split into a list of words. After that with the help of a for loop the values of P(Spam|Message) and P(Ham|Message) are calculated. After the end of the for loop, these two probability values are compared and the appropriate classification is returned by the classify() function:-

def classify(message): #Naive bayes classifier function

mylist=[message]; #converting to list as operation function requires list or array as input

mylist=operation(mylist); #cleaning the message

message=mylist[0]; #cleaned message is obtained

message=message.split(); #Converting to list where each element of the list is a word of the sentence

p\_spam\_given\_message = p\_spam;

p\_ham\_given\_message = p\_ham;

for word in message:

if word in parameters\_spam:

p\_spam\_given\_message \*= parameters\_spam[word];

if word in parameters\_ham:

p\_ham\_given\_message \*= parameters\_ham[word];

if p\_ham\_given\_message > p\_spam\_given\_message:

return 'ham';

elif p\_spam\_given\_message > p\_ham\_given\_message:

return 'spam';

else:

return 'needs human classification';

13) The naïve bayes classifier has been implemented. The only thing remaining is to create a GUI. This is created using tkinter python library. The GUI consists of a Label, TextBox and a Button:-

root=Tk();

root.title("Spam message checker");

root.iconbitmap('spam.ico');

myLabel=Label(root,text="Enter the message:");

myTextBox=Text(root,width=60,height=20);

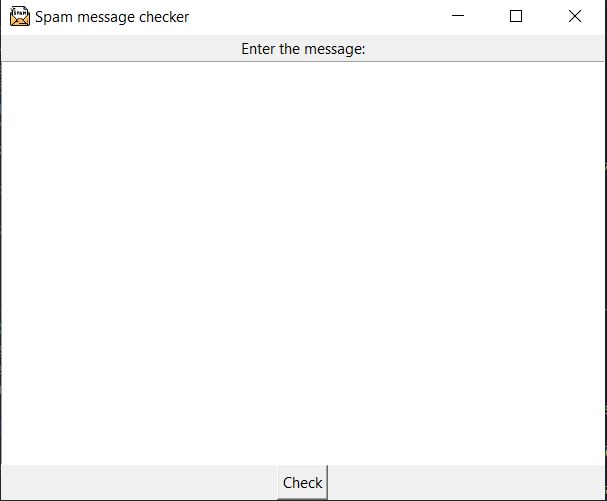
myButton=Button(root,text="Check",command=myClick);

myLabel.grid(row=0,column=0);

myTextBox.grid(row=1,column=0);

myButton.grid(row=2,column=0);

root.mainloop();

The message to be checked is typed or pasted within the textbox and then the button is pressed. On pressing the button myClick() function is called, which is defined as follows:-

def myClick():

message=myTextBox.get(1.0,END);

result=classify(message);

if(result=='ham'):

messagebox.showinfo("Result","The entered message is not a spam message");

elif(result=='spam'):

messagebox.showinfo("Result","The entered message is a spam message!");

else:

messagebox.showerror("ERROR!","The entered message cannot be classified to either group!");

myTextBox.delete(1.0,END);

The textbox’s contents are taken and put in the variable message. After that the function classify is called with message as input. The classify() function calculates the probabilities and returns the appropriate classification( ‘spam’ or ‘ham’ or ‘needs human classification’). This value returned by classify() is stored in a variable called result.

If the value of result is ‘ham’ then a messagebox is made to appear on the screen telling the user that the entered message is not spam.

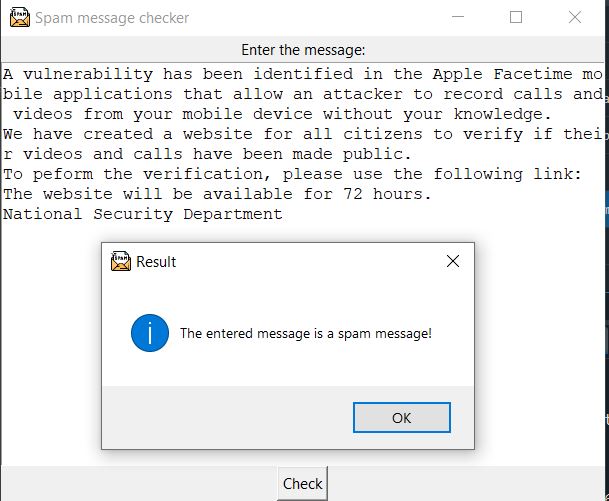
If the value of result is ‘spam’ then a messagebox is made to appear on the screen telling the user that the entered message is spam.

If the value of result is 'needs human classification' then a messagebox is made to appear on the screen telling the user that the entered message cannot be classified into either group.

The user can close the messagebox by pressing the OK button and on doing so, all the text within the textbox gets deleted.

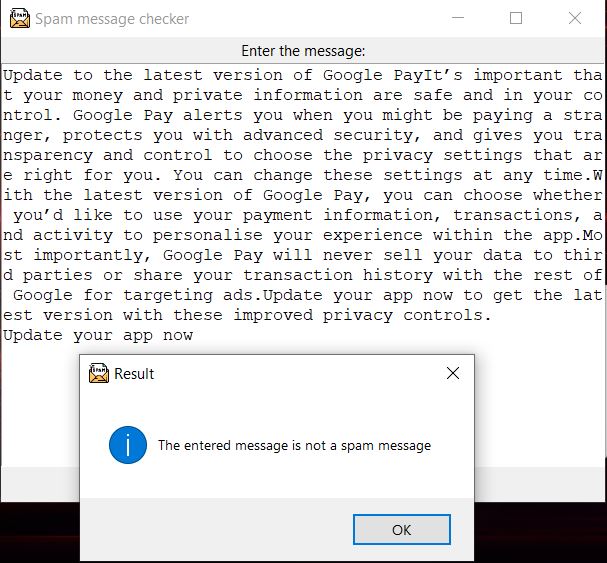
The user can either enter another message to be checked or close the GUI.

MessageBox when the message is classified as ‘spam’ :-



MessageBox when the message is classified as ‘ham’ :-

.



**RESULT**

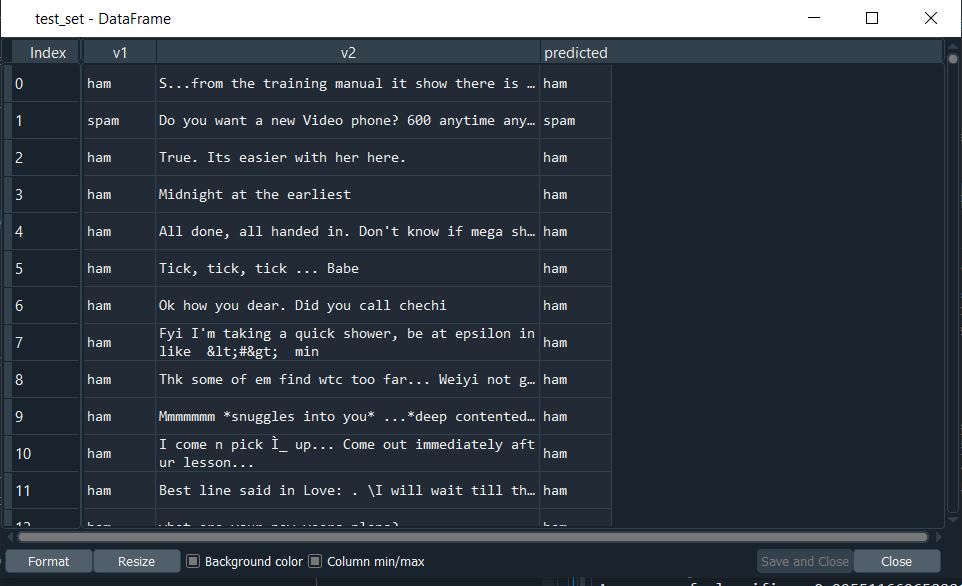
The only task remaining is to calculate the accuracy of the naïve bayes classifier.

A new column called ‘predicted’ is added to test\_set dataframe which is made to contain the values predicted ( or returned) by the classify() function.

Basically the classify() function is applied to each of the messages present in ‘v2’ column and the returned values are stored in the new column called ‘predicted’. These are the values predicted by the naïve bayes classifier:-

test\_set['predicted'] = test\_set['v2'].apply(classify);

test\_set after addition of new column (predicted):-

In order to calculate the no of correctly classified messages in test\_set, in each row of the test\_set, the values of ‘v1’ column and ‘predicted’ column should be compared. If both of them have the same value for a row then the value of variable correct should be incremented by 1:-

correct = 0;

for row in test\_set.iterrows(): #iterrows() returns (index,series) pairs

row=row[1]; #we want the second element of the tuple(series) not the index

if row['v1'] == row['predicted']:

correct += 1;

The total no of messages in test\_set is obtained as follows:-

total = test\_set.shape[0]; #test\_set.shape=(1114,3), shape[0]=1114

Since the no of correctly classified messages and the total no of messages

have been obtained, the accuracy can be calculated by using the following formula:-

Accuracy=no of correctly classified messages/ total no of messages.

The following strings are printed on console:-

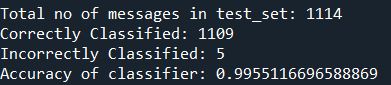
print("\n");

print('Correctly Classified:', correct);

print('Incorrectly Classified:', total - correct);

print('Accuracy of classifier:', correct/total);

The result is:-



The implemented Multinomial Naïve Bayes Classifier classified the messages of test\_set with an accuracy of 99.55%.

For better understanding a confusion matrix can be created as follows:-

from sklearn.metrics import confusion\_matrix

y\_actual=test\_set['v1'].to\_numpy();

y\_predicted=test\_set['predicted'].to\_numpy();

cm=confusion\_matrix(y\_actual,y\_predicted);

print("Confusion matrix:-");

print(cm); #1st row-ham,2nd row-spam,1st column-ham,2nd column-spam

# confusion\_matrix is imported from sklearn library. The first column(‘v1’) of test\_set (which contains ‘ham’ and ‘spam’ values in each row) is converted to a numpy array and stored in y\_actual variable.

# Similarly, ‘predicted’ column of test\_set (which contains predicted ‘ham’ and ‘spam’ values in each row) is converted to a numpy array and stored in y\_predicted variable.

# These two arrays are passed as input to the confusion\_matrix() function, This function creates and returns a confusion matrix which is stored in variable cm. After this the confusion matrix is displayed on the console:-

# C:\Users\Addo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\cm.jpg

It conveys the following information:-

1. out of 972+1=973 ham messages in test\_set, the classifier classified 972 messages correctly as ham and classifed 1 ham message incorrectly as spam.
2. out of 4+137=141 spam messages in test\_set, the classifier classified 137 messages correctly as spam and classified 4 spam messages incorrectly as ham.

**CONCLUSION**

Spam checker using multinomial naïve bayes classifier has been implemented which classified the messages present in the test\_set with an accuracy of 99.55%.

The GUI has also been implemented using tkinter which takes a single message as input and passes it as input to the naïve bayes classifier which returns the corresponding classification which in turn is displayed to the user in the form of a messagebox.

**REFERENCES**

* <https://scikitlearn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html>
* https://www.avast.com/c-spam
* https://www.geeksforgeeks.org/applying-multinomial-naive-bayes-to-nlp-problems/
* https://docs.python.org/3/library/tk.html
* <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.split.html>
* <https://www.kite.com/python/docs/nltk.RegexpTokenizer>
* <https://www.geeksforgeeks.org/python-stemming-words-with-nltk/>
* https://www.geeksforgeeks.org/removing-stop-words-nltk-python/