**Optimizing Spam Filtering with Machine Learning**

ABSTRACT:

The upsurge in the volume of unwanted emails called spam has created an intense need for the development of more dependable and robust antispam filters. [Machine learning](https://www.sciencedirect.com/topics/computer-science/machine-learning) methods of recent are being used to successfully detect and filter spam emails. We present a [systematic review](https://www.sciencedirect.com/topics/psychology/systematic-literature-review) of some of the popular machine learning based email spam filtering approaches. Our review covers survey of the important concepts, attempts, efficiency, and the research trend in spam filtering. The preliminary discussion in the study background examines the applications of [machine learning techniques](https://www.sciencedirect.com/topics/computer-science/machine-learning-technique) to the email spam filtering process of the leading [internet service providers](https://www.sciencedirect.com/topics/computer-science/internet-service-provider) (ISPs) like Gmail, Yahoo and [Outlook](https://www.sciencedirect.com/topics/nursing-and-health-professions/angiographic-catheter) emails spam filters. Discussion on general email spam filtering process, and the various efforts by different researchers in combating spam through the use machine learning techniques was done. Our review compares the strengths and drawbacks of existing [machine learning approaches](https://www.sciencedirect.com/topics/computer-science/machine-learning-approach) and the open research problems in spam filtering. We recommended deep leaning and deep adversarial learning as the future techniques that can effectively handle the menace of spam emails.

1. INTRODUCTION

1.1 Overview:

Over recent years, as the popularity of mobile phone devices has increased, Short Message Service (SMS) has grown into a multi-billion dollar industry. At the same time, reduction in the cost of messaging services has resulted in growth in unsolicited commercial advertisements (spams) being sent to mobile phones. Due to Spam SMS, Mobile service providers suffer from some sort of financial problems as well as it reduces calling time for users. Unfortunately, if the user accesses such Spam SMS they may face the problem of virus or malware. When SMS arrives at mobile it will disturb mobile user privacy and concentration.

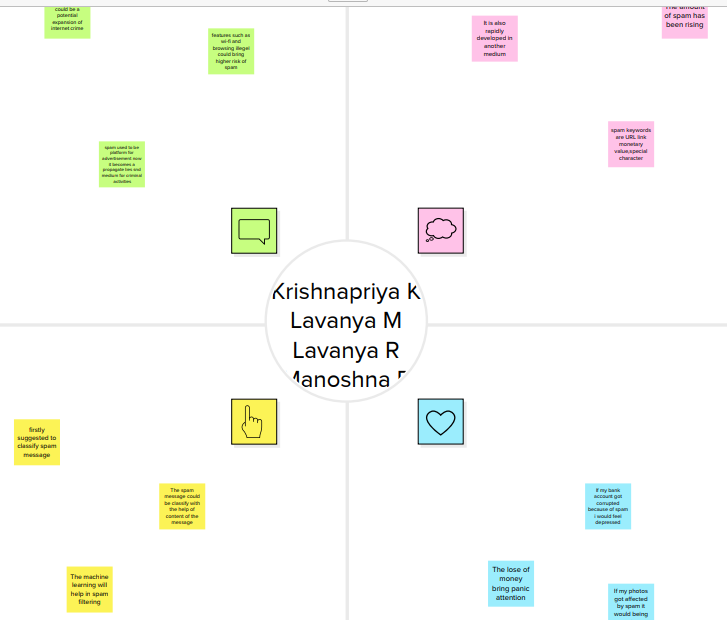
It may lead to frustration for the user. So Spam SMS is one of the major issues in the wireless communication world and it grows day by day. To avoid such Spam SMS people use white and black list of numbers.

1.2 Purpose:

This technique is not adequate to completely avoid Spam SMS. To tackle this problem it is needful to use a smarter technique which correctly identifies Spam SMS. Natural language processing technique is useful for Spam SMS identification. It analyses text content and finds patterns which are used to identify Spam and Non-Spam SMS.

2. PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map



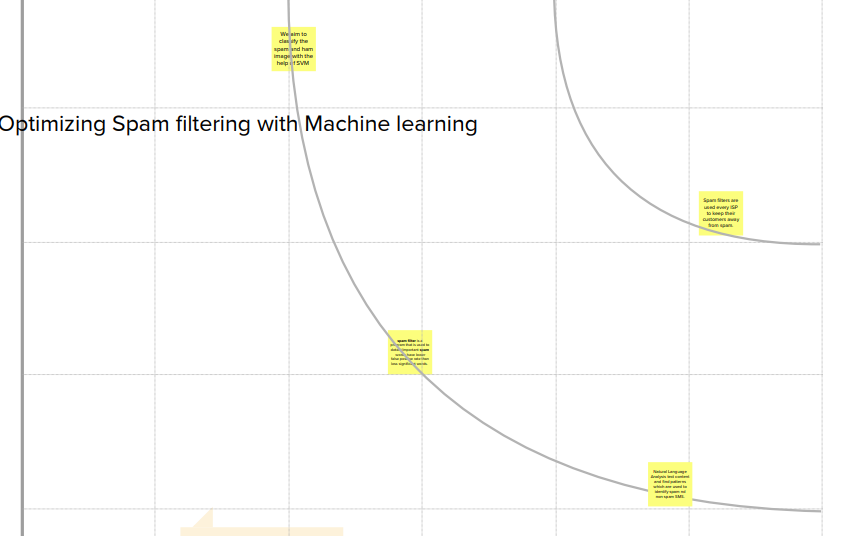
In the above Mural picture Our Group done the map with our ideas about Optimizing Spam Filtering with Machine Learning. We talk about the topic from user’s side what they Says, Thinks, Feels and Does. We added the screenshot of our Mural map.

2.2 Ideation & Brainstorming

In the below we added Screenshot of our Brainstorm and Ideation. In this template we give our brainstorm ideas about our project.







1. RESULT

The main page of Spam Detection, where you can know about the project and also from this page users can click onto the spam button and they will redirect onto the spam/ prediction page for providing the inputs

1. ADVANTAGES & DISADVANTAGES

Advantage: With the benefits of email spam filters, the security risk can be reduced since the user gets in hand the emails that have gone through various spam checks. Moreover, these email spam filters throw out malware, malicious, and virus-infested emails and protect user security.

Disadvantage: Thousands of spam emails may reach Inboxes before a spammer's email address, IP or domain is blacklisted. Spam filtering is machine-based so there is a room for mistakes called “false positives.” Bayesian filters may be fooled by spammers, e.g. in a case of using large blocks of legitimate text.

1. APPLICATIONS

An email message is made up of two major components which are the header and the body. The header is the area that have broad information about the content of the email. It includes the subject, sender and receiver. The body is the heart of the email. It can include information that does not have a pre-defined data. Examples include web page, audio, video, analog data, images, files, and [HTML](https://www.sciencedirect.com/topics/social-sciences/document-markup-languages) markup. The email header is comprised of fields such as sender's address, the recipient's address, or timestamp which indicate when the message was sent by intermediary servers to the Message Transport Agents (MTAs) that function as an office for organising mails. The header line usually starts with a “From” and it goes through some modification whenever it moves from one server to another through an in-between server. Headers allow the user to view the route the email passes through, and the time taken by each server to treat the mail. The available information have to pass through some processing before the classifier can make use of it for filtering.

7. FUTURE SCOPE

Efficient pattern detection in spam mail filtering plays crucial role. Using ANN model spam detection gives the spam patterns, non –spam patterns and general patterns which easily identify the whether the mail is spam or ham. The current method which uses the pattern detection method does not include the general patterns. ANN gives the general patterns of which user can decide to determine whether he wants to put the mail as spam or non-spam to avoid the loss of important mails. The images which are in forms of spams are also detected using File Properties, Histogram and Hough Transform. The current proposed system is for English language mails but as future scope we can design the system for multiple languages.

8. APPENDIX

# Milestone 2: Data Collection & Preparation

Importing the libraries

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import nltk

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer from sklearn import datasets

from sklearn.model\_selection import train\_test\_split from imblearn.over\_sampling import SMOTE

# Read the Dataset

df = pd.read\_csv("spam.csv",encoding="latin") df.head()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **v1** | | **v2** | **Unnamed:** | **Unnamed:** | **Unnamed:** |
|  |  |  | **2** | **3** | **4** |
| **0** | ham | Go until jurong point, crazy.. Available only ... | NaN | NaN | NaN |
| **1** | ham | Ok lar... Joking wif u oni... | NaN | NaN | NaN |

1. spam Free entry in 2 a wkly comp to win FA Cup

fina...

1. ham U dun say so early hor... U c already then

say...

NaN NaN NaN

NaN NaN NaN

1. ham Nah I don't think he goes to usf, he lives aro... NaN NaN NaN



# Data preparation: 1.Handling missing values

df.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 5572 entries, 0 to 5571 Data columns (total 5 columns):

# Column Non-Null Count Dtype

1. v1 5572 non-null object
2. v2 5572 non-null object
3. Unnamed: 2 50 non-null object
4. Unnamed: 3 12 non-null object
5. Unnamed: 4 6 non-null object dtypes: object(5)

memory usage: 217.8+ KB

df.isna().sum()

|  |  |  |
| --- | --- | --- |
| v1 |  | 0 |
| v2 |  | 0 |
| Unnamed: | 2 | 5522 |
| Unnamed: | 3 | 5560 |
| Unnamed: | 4 | 5566 |

dtype: int64

df.rename({"v1":"label","v2":"text"},inplace=True,axis=1)

df.tail()

**text**

|  |  |  |
| --- | --- | --- |
| **Unnamed:** | **Unnamed:** | **Unnamed:** |
| **2** | **3** | **4** |
| NaN | NaN | NaN |
| NaN | NaN | NaN |
| NaN | NaN | NaN |
| NaN | NaN | NaN |
| NaN | NaN | NaN |

|  |  |
| --- | --- |
|  | **label** |
| **5567** | spam This is the 2nd ti |
| **5568** | ham Will Ì\_ b going to |
| **5569** | ham Pity, \* was in mood for |
| **5570** | ham The guy did some bitc |
| **5571** | ham Rofl. |

me we have tried 2

contact u... esplanade fr home?

that. So...any other

s...

hing but I acted like

i'd... Its true to its name



# Handling Categorical Values

from sklearn.preprocessing import LabelEncoder le = LabelEncoder()

df['label'] = le.fit\_transform(df['label'])

digits = datasets.load\_digits() x = digits.data

y = digits.target

# Handling Imbalance Data

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.20, random\_state =

### Given data is imbalanced one, we are balancing the data

print("Before OverSampling, counts of label '1':{}".format(sum(y\_train == 1))) print("Before OverSampling, counts of label '0': {} \n".format(sum(y\_train == 0)))

from imblearn.over\_sampling import SMOTE sm = SMOTE(random\_state =2)

x\_train\_res, y\_train\_res = sm.fit\_resample(x\_train, y\_train.ravel())

print('After OverSampling, the shape of train\_x: {}'.format(x\_train\_res.shape)) print('After OverSampling, the shape of train\_y:{} \n'.format(y\_train\_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y\_train\_res == 1))) print("After OverSampling, counts of label '0': {}".format(sum(y\_train\_res == 0)))

Before OverSampling, counts of label '1':147 Before OverSampling, counts of label '0': 151

After OverSampling, the shape of train\_x: (1540, 64) After OverSampling, the shape of train\_y:(1540,)

After OverSampling, counts of label '1': 154 After OverSampling, counts of label '0': 154

# Cleaning the text data

nltk.download("stopwords")

[nltk\_data] Downloading package stopwords to /root/nltk\_data... [nltk\_data] Package stopwords is already up-to-date!

True

True

True

import nltk

from nltk.corpus import stopwords from nltk.stem import PorterStemmer

import re corpus = []

length = len(df)

for i in range(0,length):

text = re.sub("[^a-zA-Z0-9]"," ",df["text"][i]) text = text.lower()

text = text.split() pe = PorterStemmer()

stopword = stopwords.words("english")

text = [pe.stem(word) for word in text if not word in set(stopword)] text = " ".join(text)

corpus.append(text)



corpus

number ,



'chang e one next escal',

'yetund class run water make ok pl',

'lot happen feel quiet beth aunt charli work lot helen mo',

'wait 4 bu stop aft ur lect lar dun c go get car come back n pick', 'aight thank comin',

...]

from sklearn.feature\_extraction.text import CountVectorizer cv = CountVectorizer(max\_features=35000)

X = cv.fit\_transform(corpus).toarray()

import pickle

pickle.dump(cv, open('cv1.pkl', 'wb'))

# Milestone 3: Exploratory Data Analysis

Descriptive Statistical

df.describe()



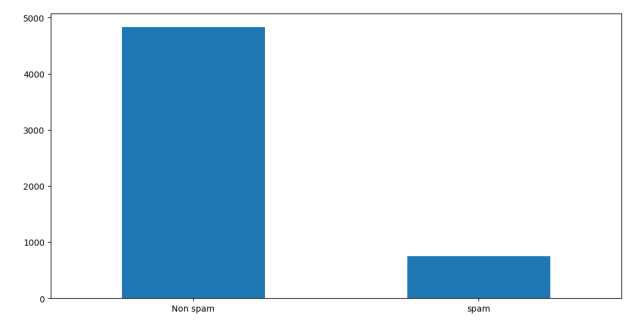
|  |  |
| --- | --- |
|  | **label** |
| **count** | 5572.000000 |
| **mean** | 0.134063 |
| **std** | 0.340751 |
| **min** | 0.000000 |
| **25%** | 0.000000 |
| **50%** | 0.000000 |
| **75%** | 0.000000 |
| **max** | 1.000000 |

df.shape

(5572, 5)

# Visual Analysis: 1.Univariate analysis

df["label"].value\_counts().plot(kind="bar",figsize=(12,6)) plt.xticks(np.arange(2), ('Non Spam','Spam'),rotation=0);



from sklearn.preprocessing import StandardScaler

x = StandardScaler().fit\_transform(x)

x\_bal = digits.data sc=StandardScaler() x\_bal=sc.fit\_transform(x\_bal)

# Splitting data into train and test

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.20, random\_state =

# Milestone 4: Model Building

Decision tree model

from sklearn.tree import DecisionTreeClassifier model = DecisionTreeClassifier() model.fit(x\_train\_res, y\_train\_res)



▾ DecisionTreeClassifier

DecisionTreeClassifier()

# Random forest model

from sklearn.ensemble import RandomForestClassifier model1 =RandomForestClassifier() model1.fit(x\_train\_res, y\_train\_res)

RandomForestClassifier()



▾ RandomForestClassifier

Naive Bayes model

from sklearn.naive\_bayes import MultinomialNB model = MultinomialNB()

model.fit(x\_train\_res, y\_train\_res)



▾ MultinomialNB

MultinomialNB()

# ANN model

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense

model = Sequential()

x\_train.shape

(1437, 64)

model.add(Dense(units = x\_train\_res.shape[1],activation="relu",kernel\_initializer="random\_

model.add(Dense(units=100,activation="relu",kernel\_initializer="random\_uniform"))

model.add(Dense(units=100,activation="relu",kernel\_initializer="random\_uniform"))

model.add(Dense(units=1,activation="sigmoid"))

model.compile(optimizer="adam",loss="binary\_crossentropy",metrics=['accuracy'])

generator =model.fit(x\_train\_res,y\_train\_res,epochs=10,steps\_per\_epoch=len(x\_train

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epoch  24/24  Epoch | 1/10  [==============================] 2/10 | - 1s | 3ms/step | - loss: | -65.2602 - accuracy: 0. |
| 24/24  Epoch 24/24 | [==============================] 3/10 [==============================] | * 0s * 0s | 3ms/step  3ms/step | * loss: * loss: | -1968.6589 - accuracy:  -16956.0859 - accuracy: |
| Epoch 24/24 | 4/10 [==============================] | - 0s | 3ms/step | - loss: | -78445.0156 - accuracy: |
| Epoch  24/24 | 5/10  [==============================] | - 0s | 3ms/step | - loss: | -261719.6250 - accuracy |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Epoch  24/24 | 6/10  [==============================] | - | 0s | 3ms/step | - | loss: | -680814.3750 - accuracy | |
| Epoch | 7/10 |  |  |  |  |  |  | |
| 24/24  Epoch | [==============================] 8/10 | - 0s | | 3ms/step | - loss: | | -1510385.0000 | - accurac |
| 24/24  Epoch | [==============================] 9/10 | - 0s | | 3ms/step | - loss: | | -2992491.0000 | - accurac |
| 24/24  Epoch 24/24 | [==============================] 10/10  [==============================] | * 0s * 0s | | 3ms/step  3ms/step | * loss: * loss: | | -5371300.0000  -9039938.0000 | * accurac * accurac |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epoch  24/24 | 1/10  [==============================] | - 0s | 3ms/step | - loss: | -14335302.0000 | - accura |
| Epoch | 2/10 |  |  |  |  |  |
| 24/24  Epoch 24/24 | [==============================] 3/10 [==============================] | * 0s * 0s | 3ms/step  3ms/step | * loss: * loss: | -21755360.0000  -31776266.0000 | * accura * accura |
| Epoch  24/24 | 4/10  [==============================] | - 0s | 3ms/step | - loss: | -44803328.0000 | - accura |
| Epoch | 5/10 |  |  |  |  |  |
| 24/24  Epoch | [==============================] 6/10 | - 0s | 3ms/step | - loss: | -61536672.0000 | - accura |
| 24/24  Epoch 24/24 | [==============================] 7/10 [==============================] | * 0s * 0s | 3ms/step  3ms/step | * loss: * loss: | -82709320.0000  -108614544.0000 | - accura  - accur |
| Epoch  24/24 | 8/10  [==============================] | - 0s | 2ms/step | - loss: | -140230864.0000 | - accur |
| Epoch | 9/10 |  |  |  |  |  |
| 24/24  Epoch 24/24 | [==============================] 10/10  [==============================] | * 0s * 0s | 3ms/step  4ms/step | * loss: * loss: | -177975296.0000  -223046240.0000 | * accur * accur |

# Testing the model

y\_pred=model.predict(x\_test) y\_pred



[1.],



[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.]], dtype=float32)

y\_pr = np.where(y\_pred>0.5,1,0)

y\_test

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| array([2, | 8, | 2, | 6, | 6, | 7, | 1, | 9, | 8, | 5, | 2, | 8, | 6, | 6, | 6, | 6, | 1, | 0, | 5, | 8, | 8, | 7, |
| 8, | 4, | 7, | 5, | 4, | 9, | 2, | 9, | 4, | 7, | 6, | 8, | 9, | 4, | 3, | 1, | 0, | 1, | 8, | 6, | 7, | 7, |
| 1, | 0, | 7, | 6, | 2, | 1, | 9, | 6, | 7, | 9, | 0, | 0, | 5, | 1, | 6, | 3, | 0, | 2, | 3, | 4, | 1, | 9, |
| 2, | 6, | 9, | 1, | 8, | 3, | 5, | 1, | 2, | 8, | 2, | 2, | 9, | 7, | 2, | 3, | 6, | 0, | 5, | 3, | 7, | 5, |
| 1, | 2, | 9, | 9, | 3, | 1, | 7, | 7, | 4, | 8, | 5, | 8, | 5, | 5, | 2, | 5, | 9, | 0, | 7, | 1, | 4, | 7, |
| 3, | 4, | 8, | 9, | 7, | 9, | 8, | 2, | 6, | 5, | 2, | 5, | 8, | 4, | 8, | 7, | 0, | 6, | 1, | 5, | 9, | 9, |
| 9, | 5, | 9, | 9, | 5, | 7, | 5, | 6, | 2, | 8, | 6, | 9, | 6, | 1, | 5, | 1, | 5, | 9, | 9, | 1, | 5, | 3, |
| 6, | 1, | 8, | 9, | 8, | 7, | 6, | 7, | 6, | 5, | 6, | 0, | 8, | 8, | 9, | 8, | 6, | 1, | 0, | 4, | 1, | 6, |
| 3, | 8, | 6, | 7, | 4, | 5, | 6, | 3, | 0, | 3, | 3, | 3, | 0, | 7, | 7, | 5, | 7, | 8, | 0, | 7, | 8, | 9, |
| 6, | 4, | 5, | 0, | 1, | 4, | 6, | 4, | 3, | 3, | 0, | 9, | 5, | 9, | 2, | 1, | 4, | 2, | 1, | 6, | 8, | 9, |
| 2, | 4, | 9, | 3, | 7, | 6, | 2, | 3, | 3, | 1, | 6, | 9, | 3, | 6, | 3, | 2, | 2, | 0, | 7, | 6, | 1, | 1, |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9, | 7, | 2, | 7, | 8, | 5, | 5, | 7, | 5, | 2, | 3, | 7, | 2, | 7, | 5, | 5, | 7, | 0, | 9, | 1, | 6, | 5, |
| 9, | 7, | 4, | 3, | 8, | 0, | 3, | 6, | 4, | 6, | 3, | 2, | 6, | 8, | 8, | 8, | 4, | 6, | 7, | 5, | 2, | 4, |
| 5, | 3, | 2, | 4, | 6, | 9, | 4, | 5, | 4, | 3, | 4, | 6, | 2, | 9, | 0, | 1, | 7, | 2, | 0, | 9, | 6, | 0, |
| 4, | 2, | 0, | 7, | 9, | 8, | 5, | 4, | 8, | 2, | 8, | 4, | 3, | 7, | 2, | 6, | 9, | 1, | 5, | 1, | 0, | 8, |
| 2, | 1, | 9, | 5, | 6, | 8, | 2, | 7, | 2, | 1, | 5, | 1, | 6, | 4, | 5, | 0, | 9, | 4, | 1, | 1, | 7, | 0, |
| 8, | 9, | 0, | 5, | 4, | 3, | 8, | 8]) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

from sklearn.metrics import confusion\_matrix,accuracy\_score cm = confusion\_matrix(y\_test, y\_pr)

score = accuracy\_score(y\_test,y\_pr) print(cm)

print('Accuracy Score Is:-' ,score\*100)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [[ | 0 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0]] |

Accuracy Score Is:- 9.722222222222223

# In ANN we first have to save the model to the test the inputs

from keras.models import load\_model from keras import models

from keras import models

def new\_review(new\_review): new\_review = new\_review

new\_review = re.sub('[^a-zA-Z]', ' ',new\_review) new\_review = new\_review.lower()

new\_review = new\_review.split() ps = PorterStemmer()

all\_stopwords = stopwords.words('english') all\_stopwords.remove('not')

new\_review = [ps.stem(word) for word in new\_review if not word in set(all\_stopw new\_review = ' '.join(new\_review)

new\_corpus = [new\_review]

new\_x\_test = cv.transform(new\_corpus).toarray() print(new\_x\_test)

new\_y\_pred = load\_model.predict(new\_x\_test) print(new\_y\_pred)

new\_x\_pred = np.where(new\_y\_pred>0.5,1,0) return new\_y\_pred

new\_review = new\_review(str(input("Enter new review...")))

Enter new review...

# Milestone 5: Performance Testing & Hyperparameter Tuning Compare the model

from sklearn.metrics import confusion\_matrix,accuracy\_score, classification\_report cm = confusion\_matrix(y\_test, y\_pred)

score = accuracy\_score(y\_test,y\_pred) print(cm)

print('Accuracy Score Is Naive Bayes:-' ,score\*100)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [[ | 0 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0]] |

Accuracy Score Is Naive Bayes:- 9.722222222222223

y\_pred1 = np.where(y\_pred>0.5,1,0) y\_pred



[1.],



[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.],

[1.]], dtype=float32)

cm = confusion\_matrix(y\_test, y\_pred) score = accuracy\_score(y\_test,y\_pred) print(cm)

print('Accuracy Score Is:-' ,score\*100)

cm1 = confusion\_matrix(y\_test, y\_pred1) score1 = accuracy\_score(y\_test,y\_pred1) print(cm1)

print('Accuracy Score Is:-' ,score1\*100)

|  |  |  |  |
| --- | --- | --- | --- |
| [[ 0 27 0 | 0 0 | 0 0 | 0 0 0] |
| [ 0 35 0 | 0 0 | 0 0 | 0 0 0] |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [ | 0 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0]] |

Accuracy Score Is:- 9.722222222222223

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [[ | 0 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0]] |

Accuracy Score Is:- 9.722222222222223

from sklearn.metrics import confusion\_matrix,accuracy\_score cm = confusion\_matrix(y\_test, y\_pr)

score = accuracy\_score(y\_test,y\_pr) print(cm)

print('Accuracy Score Is:-' ,score\*100)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [[ | 0 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0]] |

Accuracy Score Is:- 9.722222222222223

# Comparing model accuracy before & after applying hyperparameter tuning

from sklearn.metrics import confusion\_matrix,accuracy\_score cm = confusion\_matrix(y\_test, y\_pr)

score = accuracy\_score(y\_test,y\_pr) print(cm)

print('Accuracy Score Is:-' ,score\*100)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [[ | 0 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [ | 0 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |

[ 0 41 0 0 0 0 0 0 0 0]]

Accuracy Score Is:- 9.722222222222223

Milestone 6: Model Deployment Save the best model model.save('spam.h5')

# Integrate with Web Framework

Build Python code

from flask import Flask, render\_template, request import pickle

import numpy as np import re

import nltk

from nltk.corpus import stopwords from nltk.stem import PorterStemmer

from tensorflow.keras.models import load\_model

load\_model = load\_model('spam.h5')

cv = pickle.load(open('cv1.pkl','rb')) app = Flask( name )

@app.route('/') def home():

return render\_template('home.html')

@app.route('/Spam',methods=['Post','GET']) def prediction():

return render\_template('spam.html')

@app.route('/predict',methods=['POST',]) def predict():

if request.method == 'POST': message = request.form['message'] data = message

new\_review= str(data) print(new\_review)

new\_review = re.sub('[^a-zA-Z]', ' ', new\_review) new\_review = new\_review.lower()

new\_review = new\_review.split() ps = PorterStemmer()

all\_stopwords = stopwords.words('english') all\_stopwords.remove('not')

new\_review = [ps.stem(word) for word in new\_review if not word in set(all\_stopwords)] new\_review = ' '.join(new\_review)

new\_corpus = [new\_review]

new\_x\_test = cv.transform(new\_corpus).toarray() print(new\_x\_test)

new\_y\_pred = load\_model.predict(new\_x\_test) new\_x\_pred = np.where(new\_y\_pred>0.5,1,0) print(new\_x\_pred)

if new\_review[0][1]==1:

return render\_template('result.html', prediction="Spam") else :

return render\_template('result.html', prediction="Not a Spam")

if name =="main ":

port=int(os.environ.get('PORT',5000)) app.run(debug=False)