OPTIMIZING SPAM FILTERING WITH MACHINE LEARNING

# 1. INTRODUCTION

1.1 OVERVIEW

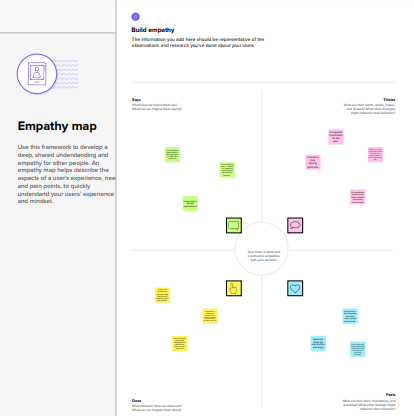
Over recent years, as the popularity of mobile phone devices has increased, Short Message Service (SMS) has grown into a multi-billion dollar industry. At the same time, reduction in the cost of messaging services has resulted in growth in unsolicited commercial advertisements (spams) being sent to mobile phones. Due to Spam, Mobile services 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.

1.2 PURPOSE

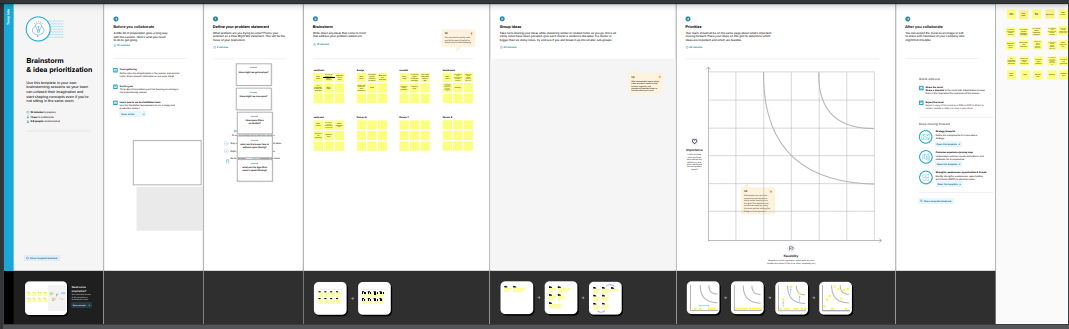
To avoid such Spam SMS people use white and black list of numbers. But this technique is not adequate to completely avoid Spam SMS. To tackle this problem it is needful to use a smarter technique which correctly identifies Spam SMS. Natural language processing technique is useful for Spam SMS identification.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

## 2. PROBLEM DEFINITION & DESIGN THINKING

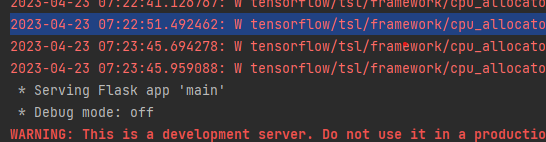
2.1 EMPATHY MAP

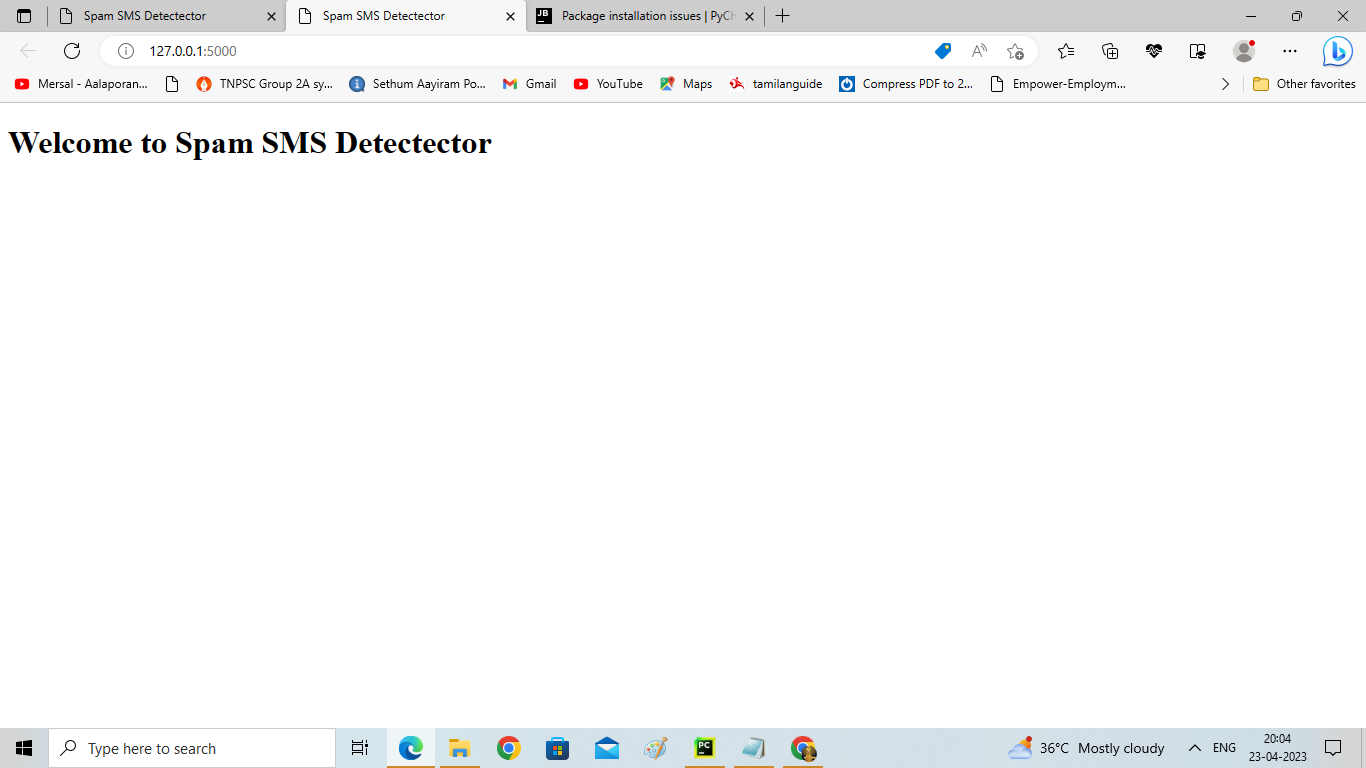


2.2 IDEATION & BRAINSTORMING MAP



# 3. RESULT







# C:\Users\soni\Documents\machinelearningproject\outputscreenshots\Screenshot_2023_0423_172156.png



4. ADVANTAGES & DISADVANTAGES

ADVANTAGES:

* The proposed model is trained well to identify the SMS category in terms of Ham or Spam with TF-IDF features and oversampling technique. The performance of the proposed approach was also evaluated on the spam email dataset with significant 99% accuracy.
* It is very effective and is also adaptive, so hard to fool.
* Based on text classification methods: Decision tree model, Random forest model, Naïve Bayes model, ANN model.
* Phenomenally accurate.
* Learns new spammer tactics automatically.
* Adapt to changing spam.

DISADVANTAGES:

* Need lots of training data.
* Spammers are learning too-Images, synonyms, misspellings,…
* Hard to get good email corpora.
* Need huge attributes.
* Functions best with individual user settings.
* Accuracy dramatically decreases when deployed as a generic gateway solution.
* Requires more processing power.

# 5. APPLICATIONS

Since having a good term representation is one of the most important parts for getting a good classifier, we have to face the fact that SMS messages have not the same structure and characteristics than email messages. We have described techniques used to filter spam email messages, but we cannot state they can be also effective filtering SMS.

SMS are usually shorter than email messages. Only 160 characters are allowed in a standard SMS text, and that could be a problem because using fewer words means less information to work with.

Also, due to the above constraint, people tend to use acronyms when writing SMS. Moreover, the abbreviations used by SMS users are not standard for a language, but they depend on the users communities. Such language variability provides more terms or features, and a more sparse representation. We have to test if the state of the art methods used to extract terms from email messages are also suitable for SMS texts.

* Spam Titan
* Maliwasher
* SpamSieve
* Comodo Dome Antispam
* Spamfighter
* MX Guarddog

# 6. CONCLUSION

SMS Spam identification is one of the important task in present world, which is wasting user’s valuable time as well as money. Present algorithm tackles this issues.

Present Work is useful to identify Spam SMS from SMS dataset. Experimental work shows that 98.12% SMS are identified correctly as Spam SMS’s from the dataset.

It also checks algorithm errors by most important error checking technique MAE and RMSE. MAE of current algorithm is 0.091 and RMSE is 0.3 which is very less. Therefore present study correctly identifies Spam SMS’s as compared to other algorithms. There is more scope to increase accuracy in identifying Spam SMS. The merit of our approach which lies in the various machine recognizable statistics derived from the skeleton of the document (HTML tags).

# 7. FUTURE SCOPE

The proposed approach is helpful because it can automatically detect SMS categories. So, there is no need for human interaction for categorical purposes, and the proposed model will automatically detect the SMS category.

This research can be further explored by hybrid machine learning techniques to enhance the accuracy of results, which will be beneficial in categorizing SMS.

# 8. APPENDIX

# SOURCE CODE:

import numpy as np #scientific computation

import pandas as pd #loading dataset file

import matplotlib.pyplot as plt #visualization

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

import nltk #preprocessing

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

# read the dataset

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

df.head()

#Give consise  summary of the dataframe

df.info()

#return the sum of all no values

df.isna().sum()

#rename the dataset

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

df.tail()

#HANDILING CATEGROICAL VALUES

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

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

#CLEANING THE TEXT DATA

nltk.download("stopwords")

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

# splting datq into train and validatiob sets using train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(max\_features=35000)

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

y = pd.get\_dummies(df['label'])

y = y.iloc[:, 1].values

import pickle

pickle.dump(cv, open('cv1.pkl','wb'))

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=0)

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 == 1)))

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

df.describe()

df.shape

df["label"].value\_counts().plot(kind="bar",figsize=(12,6))

plt.xticks(np.arange(2),  ('Non spam', 'spam'),rot

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=0)

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()

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

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

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

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

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

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

model = Sequential()

x\_train.shape

model.add(Dense(units = x\_train\_res.shape[1],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\_res)//64)

generator = model.fit(x\_train\_res,y\_train\_res,epochs=10,steps\_per\_epoch=len(x\_train\_res)//64)

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

y\_pred

y\_pred1=model.predict(x\_train)

y\_pred1

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

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

y\_test

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

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

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)

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\_stopwords)]

new\_review = ' '.join(new\_review)

new\_corpus = [new\_review]

new\_X\_test = cv.transform(new\_corpus).toarray()

new\_y\_pred = model.predict(new\_X\_test)

print(new\_y\_pred)

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

return new\_review

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

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 Naive Bayes:- ' ,score\*100)

#COMPARE THE MODEL

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

cm=confusion\_matrix(y\_test,y\_pred)

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

print(cm)

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

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

cm1=confusion\_matrix(y\_test,y\_pred1)

score = accuracy\_score(y\_test,y\_pred1)

print(cm1)

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

model.save('spam.h5')

from sklearn.svm import SVC

svm1=SVC(kernel='rbf')

svm1.fit(x\_train\_res, y\_train\_res)

y\_pred4=svm1.predict(x\_test)

from sklearn.metrics import accuracy\_score

svm\_rbf=accuracy\_score(y\_test,y\_pred4)

svm\_rbf

svm2=SVC(kernel='sigmoid')

svm2.fit(x\_train, y\_train)

y\_pred5=svm2.predict(x\_test)

from sklearn.metrics import accuracy\_score

svm\_sig=accuracy\_score(y\_test,y\_pred5)

svm\_sig

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()

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

y\_pred6=model.predict(x\_test)

from sklearn.metrics import accuracy\_score

dec\_tree=accuracy\_score(y\_test,y\_pred6)

dec\_tree

!pip install nbconvert

from google.colab import drive

drive.mount('/content/drive')

! pwd