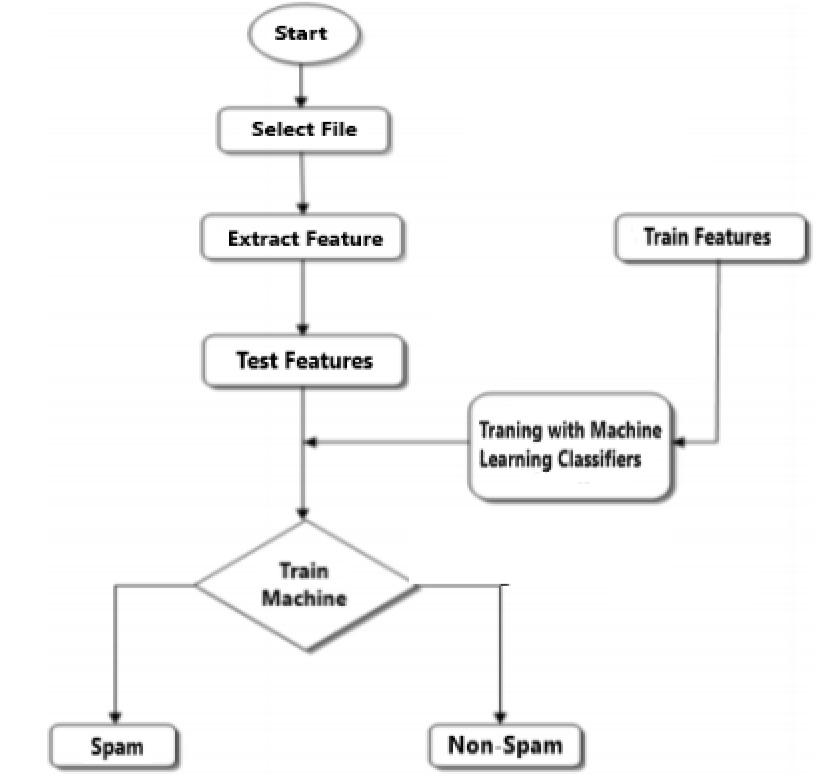
| **School of Electronics Engineering (SENSE)** | | | | |
| --- | --- | --- | --- | --- |
| **J COMPONENT – REPORT** | | | | |
| **COURSE CODE / TITLE** | CSE3501 – Information Security Analysis and Audit | | | |
| **PROGRAM / YEAR/ SEM** | B.Tech (ECE)/III Year/ FALL 2021-2022 | | | |
| **LAST DATE FOR REPORT SUBMISSION** | 04-12-2021 | | | |
| **DATE OF SUBMISSION** | 04-12-2021 | | | |
| **TEAM MEMBERS**  **DETAILS** | **REGISTER NO.** | | **NAME** | |
| 19BEC1017 | | G JAYASOORIYA | |
| 19BEC1128 | | NANDESH P G | |
| 19BEC1370 | | SANJAY S | |
| **PROJECT TITLE** | **Spam Email Detection using ML Classifiers** | | | |
| **COURSE HANDLER’S NAME** | **DR.N.SUBHASHINI** | **REMARKS** | |  |
| **COURSE HANDLER’S SIGN** |  |

**OBJECTIVE:**

We build a spam detector using 4 Machine Learning models and evaluate them with test data using different performance metrics used. The dataset we used was from Kaggle named “emails.csv” which had shuffled samples of email subjects and bodies containing both spam and ham emails in different proportions, which we converted into lemmas. As per our analysis, the Naive Bayes model and Random Forest models worked well for spam detection, whereas SVM performed the poorest among the 4 models.

**BLOCK DIAGRAM:**



**COMPONENTS/ SOFTWARE REQUIRED:**

* GOOGLE COLLABORATOR
* DATASETS
* “emails.csv” DATASET FROM KAGGLE

**PROJECT DESCRIPTION:**

Email has become one of the most important forms of communication. In 2014, there are estimated to be 4.1 billion email accounts worldwide, and about 196 billion emails are sent each day worldwide. Spam is one of the major threats posed to email users. In 2013, 69.6% of all email flows were spam. Therefore, an effective spam filtering technology is a significant contribution to the sustainability of the cyberspace and to our society.

We are collecting the dataset of the spam mails from “emails.csv” file from Kaggle. Then we wrote a python code using 4 types of machine learning algorithms namely Decision Trees, SVM (support vector machine), Random Forest, and Naive Bayes. We wrote a python code for these 4 algorithms separately and checked the precision of each of the algorithm to check whether which algorithm is best for finding whether the mail which we receive is spam or not.

**CONCEPT LEARNED:**

* 4 Different Machine Learning algorithms
* Implementation of those algorithms in real-world problems

**ML model employed:**

**A. Naive Bayes with bag of words approach using TF-IDF**

Naive Bayes is the easiest classification algorithm (fast to build, regularly used for spam detection). It is a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features.

Feature extraction using BOW: TF-IDF

Term frequency-Inverse document frequency uses all the tokens in the dataset as vocabulary. The frequency of occurrence of a token from vocabulary in each document consists of the term frequency and the number of documents in which token occurs determines the Inverse document frequency. What this ensures is that, if a token occurs frequently in a document that token will have high TF but if that token occurs frequently in the majority of documents then it reduces the IDF, so stop words like an the, I which occur frequently are penalized and important words which contain the essence of document get a boost. Both these TF and IDF matrices for a particular document are multiplied and normalized to form the TF-IDF of a document

**B. Decision Trees**

Decision trees are used for classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. The decision of making strategic splits heavily affects a tree’s accuracy. The decision criteria are different for classification and regression trees. Information theory is a measure to define this degree of disorganization in a system known as Entropy. If the sample is completely homogeneous, then the entropy is zero and if the sample is an equally divided (50% – 50%), it has entropy of one. It chooses the split which has the lowest entropy compared to parent node and other splits. The lesser the entropy, the better it is.

**C. Support Vector Machine (SVM)**

SVM is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyperplane that differentiates the two classes very well. Support Vectors are simply the coordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line). If the data requires non-linear classification, SVM can employ Kernels, which are functions that takes low dimensional input space and transform it to a higher dimensional space i.e. they convert non separable problem to separable problem.

**D. Random Forest**

Random forest is like bootstrapping algorithm with Decision tree (CART) model. Random forest tries to build multiple CART model with different sample and different initial variables. For instance, it will take a random sample of 100 observation and 5 randomly chosen initial variables to build a CART model. It will repeat the process (say) 10 times and then make a final prediction on each observation. The final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction. Random forest gives much more accurate predictions when compared to simple CART/CHAID or regression models in many scenarios. These cases generally have high number of predictive variables and huge sample size. This is because it captures the variance of several input variables at the same time and enables high number of observations to participate in the prediction. Metrics Used

Following are the metrics we used to evaluate the performance of ML techniques:

1. Precision

Precision refers to the closeness of two or more measurements to each other. In Machine Learning, precision is the fraction of relevant instances among the retrieved instances. Precision = TP / (TP + FP) (Where TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative).

2. Accuracy

Accuracy refers to the closeness of a measured value to a standard or known value. Accuracy = (TP+TN) / ALL

3. Recall

Recall is how many of the true positives were recalled (found), i.e. how many of the correct hits were also found. Recall = TP / (TP + FN)

4. F-Score

F-scores are a statistical method for determining accuracy accounting for both precision and recall. It is essentially the harmonic mean of precision and recall.

5. AUC

AUC is the area under the ROC curve. The closer the AUC value is to 1, the better the model.

**IMPLEMENTATION:**

**CODE:**

from google.colab import files

uploaded = files.upload()

import numpy as np

import pandas as pd

import nltk

from nltk.corpus import stopwords

import string

df = pd.read\_csv("emails.csv")

df.head()

|  | **text** | **spam** |
| --- | --- | --- |
| **0** | Subject: naturally irresistible your corporate... | 1 |
| **1** | Subject: the stock trading gunslinger fanny i... | 1 |
| **2** | Subject: unbelievable new homes made easy im ... | 1 |
| **3** | Subject: 4 color printing special request add... | 1 |
| **4** | Subject: do not have money , get software cds ... | 1 |

df.shape

(5728, 2)

df.columns

Index(['text', 'spam'], dtype='object')

df.drop\_duplicates(inplace=True)

print(df.shape)

(5695, 2)

# to show the number of missing data

print(df.isnull().sum())

text 0

spam 0

dtype: int64

# download the stopwords package

nltk.download("stopwords")

[nltk\_data] Downloading package stopwords to /root/nltk\_data...

[nltk\_data] Unzipping corpora/stopwords.zip.

True

def process(text):

    nopunc = [char for char in text if char not in string.punctuation]

    nopunc = ''.join(nopunc)

    clean = [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]

    return clean

# to show the tokenization

df['text'].head().apply(process)

0 [Subject, naturally, irresistible, corporate, ...

1 [Subject, stock, trading, gunslinger, fanny, m...

2 [Subject, unbelievable, new, homes, made, easy...

3 [Subject, 4, color, printing, special, request...

4 [Subject, money, get, software, cds, software,...

Name: text, dtype: object

from sklearn.feature\_extraction.text import CountVectorizer

message = CountVectorizer(analyzer=process).fit\_transform(df['text'])

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

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

xtrain, xtest, ytrain, ytest = train\_test\_split(message, df['spam'], test\_size=0.20, random\_state=0)

print(message.shape)

**\*\*Naive Bayes\*\***

# create and train the Naive Bayes Classifier

from sklearn.naive\_bayes import MultinomialNB

classifier = MultinomialNB().fit(xtrain, ytrain)

print(classifier.predict(xtrain))

[0 0 0 ... 0 0 0]

print(ytrain.values)

[0 0 0 ... 0 0 0]

#print the predictions

print(classifier.predict(xtest))

#print the actual values

print(ytest.values)

[1 0 0 ... 0 0 0]

[1 0 0 ... 0 0 0]

# Evaluating the model on the training data set

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

pred = classifier.predict(xtest)

print(classification\_report(ytest, pred))

print()

print("Confusion Matrix: \n", confusion\_matrix(ytest, pred))

print("Accuracy: \n", accuracy\_score(ytest, pred))

precision recall f1-score support

0 1.00 0.99 0.99 870

1 0.97 1.00 0.98 269

accuracy 0.99 1139

macro avg 0.98 0.99 0.99 1139

weighted avg 0.99 0.99 0.99 1139

Confusion Matrix:

[[862 8]

[ 1 268]]

Accuracy:

0.9920983318700615

print("Accuracy:",metrics.accuracy\_score(ytest, pred))

Accuracy: 0.9920983318700615

print("f1\_score:",metrics.f1\_score(ytest, pred))

f1\_score: 0.98348623853211

print("confusion\_matrix:",metrics.confusion\_matrix(ytest, pred))

confusion\_matrix: [[862 8]

[ 1 268]]

print("precision\_score:",metrics.precision\_score(ytest, pred))

precision\_score: 0.9710144927536232

print("recall\_score:",metrics.recall\_score(ytest, pred))

recall\_score: 0.9962825278810409

print("roc\_auc\_score:",metrics.roc\_auc\_score(ytest, pred))

roc\_auc\_score: 0.9935435627910952

#import all the needed libraries

import mailbox

%matplotlib inline

import matplotlib.pyplot as plt

import csv

from textblob import TextBlob

import pandas

import sklearn

#import cPickle

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.svm import SVC, LinearSVC

from sklearn.metrics import classification\_report, f1\_score, confusion\_matrix

import sklearn.metrics as metrics

from sklearn.pipeline import Pipeline

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import StratifiedKFold, cross\_val\_score, train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import learning\_curve

#import metrics libraries

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import roc\_auc\_score

**\*\*DECISION TREE\*\***

#create and fit tree model

model\_tree=DecisionTreeClassifier()

model\_tree.fit(xtrain,ytrain)

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',

max\_depth=None, max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort='deprecated',

random\_state=None, splitter='best')

y\_pred = model\_tree.predict(xtest)

import sklearn.metrics as metrics

print(classification\_report(ytest, y\_pred))

print()

print("Confusion Matrix: \n", confusion\_matrix(ytest, y\_pred))

print("Accuracy: \n", accuracy\_score(ytest, y\_pred))

precision recall f1-score support

0 0.96 0.98 0.97 870

1 0.94 0.88 0.91 269

accuracy 0.96 1139

macro avg 0.95 0.93 0.94 1139

weighted avg 0.96 0.96 0.96 1139

Confusion Matrix:

[[855 15]

[ 33 236]]

Accuracy:

0.9578577699736611

print("Accuracy:",metrics.accuracy\_score(ytest, y\_pred))

Accuracy: 0.9578577699736611

print("f1\_score:",metrics.f1\_score(ytest, y\_pred))

f1\_score: 0.9076923076923077

print("confusion\_matrix:",metrics.confusion\_matrix(ytest, y\_pred))

confusion\_matrix: [[855 15]

[ 33 236]]

print("precision\_score:",metrics.precision\_score(ytest, y\_pred))

precision\_score: 0.9402390438247012

print("recall\_score:",metrics.recall\_score(ytest, y\_pred))

recall\_score: 0.8773234200743495

print("roc\_auc\_score:",metrics.roc\_auc\_score(ytest, y\_pred))

roc\_auc\_score: 0.9300410203820023

**SVM**

#create and fit SVM model

model\_svm=SVC()

model\_svm.fit(xtrain,ytrain)

SVC(C=1.0, break\_ties=False, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='scale', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

#run model on test and print metrics

predicted\_class\_tree=model\_tree.predict(xtest)

print(classification\_report(ytest, predicted\_class\_tree))

print()

print("Confusion Matrix: \n", confusion\_matrix(ytest, predicted\_class\_tree))

print("Accuracy: \n", accuracy\_score(ytest, predicted\_class\_tree))

precision recall f1-score support

0 0.96 0.98 0.97 870

1 0.94 0.88 0.91 269

accuracy 0.96 1139

macro avg 0.95 0.93 0.94 1139

weighted avg 0.96 0.96 0.96 1139

Confusion Matrix:

[[855 15]

[ 33 236]]

Accuracy:

0.9578577699736611

print("Accuracy:",metrics.accuracy\_score(ytest,predicted\_class\_tree ))

Accuracy: 0.9578577699736611

print("f1\_score:",metrics.f1\_score(ytest,predicted\_class\_tree ))

f1\_score: 0.9076923076923077

print("roc\_auc\_score:",metrics.roc\_auc\_score(ytest,predicted\_class\_tree ))

roc\_auc\_score: 0.9300410203820023

print("recall\_score:",metrics.recall\_score(ytest, predicted\_class\_tree))

recall\_score: 0.8773234200743495

print("precision\_score:",metrics.precision\_score(ytest,predicted\_class\_tree ))

precision\_score: 0.9402390438247012

print("confusion\_matrix:",metrics.confusion\_matrix(ytest,predicted\_class\_tree ))

confusion\_matrix: [[855 15]

[ 33 236]]

**RANDOM FOREST**

from sklearn.ensemble import RandomForestClassifier

#create and fit model

model\_rf=RandomForestClassifier(n\_estimators=20,criterion='entropy')

model\_rf.fit(xtrain,ytrain)

RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None,

criterion='entropy', max\_depth=None, max\_features='auto',

max\_leaf\_nodes=None, max\_samples=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=20,

n\_jobs=None, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False)

#run model on test and print metrics

predicted\_class\_rf=model\_rf.predict(xtest)

print(classification\_report(ytest, predicted\_class\_rf))

print()

print("Confusion Matrix: \n", confusion\_matrix(ytest, predicted\_class\_rf))

print("Accuracy: \n", accuracy\_score(ytest, predicted\_class\_rf))

precision recall f1-score support

0 0.96 1.00 0.98 870

1 1.00 0.87 0.93 269

accuracy 0.97 1139

macro avg 0.98 0.93 0.95 1139

weighted avg 0.97 0.97 0.97 1139

Confusion Matrix:

[[869 1]

[ 36 233]]

Accuracy:

0.9675153643546971

print("Accuracy:",metrics.accuracy\_score(ytest,predicted\_class\_rf ))

Accuracy: 0.9675153643546971

print("f1\_score:",metrics.f1\_score(ytest, predicted\_class\_rf))

f1\_score: 0.9264413518886679

print("roc\_auc\_score:",metrics.roc\_auc\_score(ytest,predicted\_class\_rf ))

roc\_auc\_score: 0.9325107892150579

print("recall\_score:",metrics.recall\_score(ytest,predicted\_class\_rf ))

recall\_score: 0.8661710037174721

print("precision\_score:",metrics.precision\_score(ytest,predicted\_class\_rf ))

precision\_score: 0.9957264957264957

print("confusion\_matrix:",metrics.confusion\_matrix(ytest, predicted\_class\_rf))

confusion\_matrix: [[869 1]

[ 36 233]]

VIDEO DEMO LINK HERE:

**CHALLENGES FACED OR TO BE FACED:**

We were not able to get the Dataset we required so have to change the program as one of the significant issues that machine learning professionals face is the **Absence of good quality data**

Similarly, some of the challenges are

Data plays a significant role in the machine learning process.

**Irrelevant features.**

This process occurs when data is unable to establish an accurate relationship between input and output variables which affects the factors like Maximization the training time, Enhance the complexity of the model, Add more features to the data, Reduce regular parameters, Increasing the training time of model

**Overfitting and Underfitting.**

Overfitting refers to a machine learning model trained with a massive amount of data that negatively affect its performance.

**Imperfections in the Algorithm When Data Grows**

 So you need regular monitoring and maintenance to keep the algorithm working as the quality of spam also keeps evolving ,Thus is one of the most exhausting issues faced.

**APPLICATIONS:**

Email Spam Detector is Exclusively used in email platforms such as Google Gmail, Yahoo, Outlook. Now a days we need big servers to store mails. Due to these spam mails, there is a lot of useless data is stored leaving a carbon footprint in the process of storing those mails, thereby we use a lot of energy in this process, through spam mail filters we can filter out all the spam mails thereby leaving less carbon footprint behind as global warming being the major issue in the modern era.

There are many spam phishing attacks taking place which is a serious concern in cyber security. By stopping and filtering these spam mails we will be able to stop this kind of cyber-attack and protect the user information from stealing. These are some of the crucial applications where spam mail detections play an important role.

**CONCLUSION:**

It is clear from the comparison that the SVM model did not work out very well to solve our problem of spam detection. Naive Bayes and Random Forest both work pretty well. While the Naive Bayes algorithm is having high accuracy and good precision, the recall value is poorer compared to Decision Tree and Random Forest. Since the SVM model could not predict any positive values at all, its accuracy, recall, and F-score were 0. As far as the F-score is concerned, Decision Tree and Random Forest have a good score as a result of both good precision and recall. NB and Random Forest had the best AUC scores. Overall, we think that both Naive Bayes and Random Forest will be very good for spam detection. Of course, we can employ boosting or stacking ensemble methods to combine two or more of these models into a really good spam detector.

**REFERENCES:**

[1] Anti-Phishing Working Group. (2018). Phishing Activity Trends Report 1st Quarter 2018. [Online].Available: <http://docs.apwg.org/Preports/apwg_trends_report_q1_2018.pdf>

[2] PhishLabs. (2018). 2018 Phish Trends & Intelligence

Report.[Online]. Available:

https://info.phishlabs.com/hubfs/2018%20PTI%20Report/Phis

hLabs%20Trend %20Report\_2018-digital.pdf

[3] M. Nguyen, T. Nguyen, and T. H. Nguyen. (2018). ``A deep

learning model with hierarchical LSTMs and supervised

attention for anti-phishing.''[Online]. Available:

<https://arxiv.org/abs/1805.01554>

[4] Anti-Phishing Working Group. (2016). Phishing Activity

TrendsReport 4th Quarter 2016. [Online]. Available:

<http://docs.apwg.org/reports/apwg_trends_report_q4_2016.pdf>

[5] Anti-Phishing Working Group. (2015). Phishing Activity

Trends Report 1st-3rd Quarter 2015. [Online]. Available:

http://docs.apwg.org/ Preports/apwg\_trends\_report\_q1-

q3\_2015.pdf

[6] L. M. Form, K. L. Chiew, S. N. Sze, and W. K. Tiong,

``Spam e-mail detection technique by using hybrid features,'' in

Proc. 9th Int. Conf. IT Asia (CITA), Aug. 2015, pp. 1\_5.

[7] M. Hiransha, N. A. Unnithan, R. Vinayakumar, and K.

Soman, ``Deep learning-based Spam e-mail detection,'' in Proc.

1st AntiPhishing Shared Pilot 4th ACM Int. Workshop Secure.

Privacy Anal. (IWSPA)A. D. R. Verma, Ed. Tempe, AZ, USA,

Mar. 2018.

[8] L. Yu, W. Zhang, J. Wang, and Y. Yu, ``SeqGAN: Sequence

generative adversarial nets with policy gradient,'' in Proc.

AAAI, 2017, pp. 2852\_2858.

[9] vishal dineshkumar soni. (2018). Role of ai in industry in

emergency Services. International engineering journal for

research & development, 3(2), 6.

https://doi.org/10.17605/osf.io/c67bm