UNIVERSITY OF NORTH TEXAS

CSCE 5380 - DATA MINING

Spring 2024

Team - 26

Personal Email Categorizer

Submitted to
Prof.
Dr. Moawia Eldow

Submitted by

Karthik Rachamalli - 11149091
Naga Venkata Kanakalakshmi - 11725119
Vishnu Sudireddy - 11642773

1. Abstract:

In this rapid digitalizing world, the most important communication medium is “electronic mail”, which is famously known as “email”. The formal definition of an email is “The transfer of messages electronically from one electronic device to another across networks [1]”. As digitalization is very high in these days, the number of digital services are also skyrocketing, which in turn surged the number of emails hitting our inboxes at double the pace, which is a directly proportional relation. The main problem in increase in number of emails is managing them and not missing the important one among the unnecessary emails. And also, general approach is categorizing them manually by an individual will be the biggest challenge ever and near to impossible task. According to a study [2] nearly 64 million unnecessary emails are sent in a country like Great Britain per day, which is a very humongous in number. The biggest challenge is organizing those emails in individual’s inboxes and following up significant emails among such as huge amount of unnecessary emails is really a night mare. The project “**Personal Email Categorizer**” is saviour and single source of solution for the people who wants to maintain their inboxes healthy by organizing the emails by classifying them automatically based on the content of the email.

We have used the supervised machine learning algorithms such as Random Forest GINI, Random Forest Entropy, Naïve Bayes Multinomial, Naïve Bayes Gaussian, SVM [7] (5 types of models), and Gradient Boosting to classify or categorize the emails. Here for this project we will be using a sample personal dataset (5000+ Rows and 3000+ columns) from Kaggle which is extracted from an individual inbox particularly GMAIL, over the period of time to train the machine learning models. We have achieved an highest accuracy of **94.46% with SVM linear** supervised model and lowest of **87% with Naïve Bayes Gaussian** supervised machine learning model. We have built, trained, and tested nearly 10 different models and compared their results which can be found in the “Results” section below.

As part of future scope to the project we can extract the content from each individual who wants to use our “Personal Email Categorizer” project particularly and also can extend to other email clients as well.

2. Introduction:

Email has become a crucial medium for any official or unofficial activities, according to an article published by “Search Engine Land [3]” on “Why email is more important than ever”. In this busy world, organizing the emails and following up an important necessary email among the lots of unnecessary emails is very hard and quiet challenging. But the heavy flow of the emails into our inboxes in our day-to-day life has introduced many complexities and problems, please find some of them in “Background” section.

The project “**Personal Email Categorizer**” we are targeting is a one stop solution for the individuals, which will classify the emails using Machine learning’s some of best supervised algorithms such as “Random Forest GINI”, “Random Forest Entropy”, “Naïve Bayes Multinomial”, “Naïve Bayes Gaussian”, “SVM” [7] (5 types of models), and “Gradient Boosting” to classify or categorize the emails.

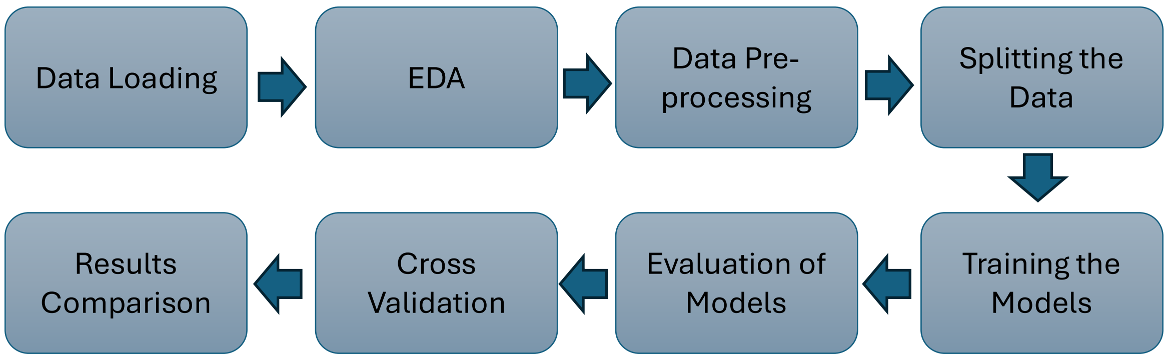
We have followed many steps as part of the project, on a high level please find them below.

- Data Loading
- Exploratory Data Analysis
- Data Pre-Processing
    - Handling NULL Values
    - Identify the Outliers
    - Normalization
- Split the data for Training and Testing
- Training the Models
- Testing
- Cross Validation
- Results Comparison

We will discuss in detail about what we have done in each of the step in the "Experiment Methodology" section below. And the results of the processes will be discussed in "Results" section below.

Please find the Flow Chart below to get the high-level approach we have taken which includes all the steps we have mentioned above.

2.1 Flow Chart:



2.2 Contributions:

We as a team divided the work into three equal parts, please find the detailed work contribution below.

Member	Percentage	Contributions
Karthik Rachamalli	33.33%	Exploratory Data Analysis, SVM, Gradient Boosting
Naga Venkata Kanakalakshmi	33.33%	Data Pre-Processing, Random Forest (GINI & Entropy), Cross Validation
Vishnu Sudireddy	33.33%	Data Loading, Naïve Bayes Gaussian, Naïve Bayes Multinomial

3. Background:

The main background of the project revolves around the email management. This solution handles the problem of overloaded email inboxes and provides the categorizing the incoming emails to the individuals. This domain majorly involves training the machine learning models and using those trained models to categorize the email on the personal customizations of the users. This acts as important solver ensuring vital communications through email.

To understand the background very keenly, Please find some of the challenges or problems that an individual is facing below.

- **Missing an important email [4]:** As we discussed, as the digital services are increasing in number will make the number of emails hitting our inboxes are also doubling day by day, which in turn causes a major issue of missing an important email by the individual in these overloaded emails.
- **Reduced Productivity [4]:** Due to this humongous number of emails, a par individual is losing their valuable time in organizing the email which in turn causing a noticeable reduction of their productivity. According to a survey, 33% USA adults are spending 1 plus hours of their time organizing their emails [4].
- **Spamming [5]:** Due to the surge of digitalization, spamming is another thing that is getting increased day by day. Spamming has a directly proportional relationship with digitalization. There are very high chances where an individual can be a victim of a spam email and will be in 100% dilemma every day in finding the genuine email.
- **Customization:** Every individual in this world has a unique way of doing the things. Same will also applicable, when it comes to organizing their emails. But, due to large number of emails, It will become very hard for them to customize.

To address the above issues, the individual needs to organize and categorize the emails manually which will raise many concerns such as

- Need to spend a lot of time every day monitoring their inboxes.
- Manually categorize the emails into their desired boxes.
- As the process is manual, there are high chances of wrong classification that happen on the emails.

We want to automate the categorization of emails using “**Personal Email Categorizer**” project, which will increase the productivity of the people, reduced spamming, decreased misclassification etc..

4. Experiment Methodology:

This section discusses detailed about the experiments performed as part the project “**Personal Email Categorizer**”. A well systematic approach and methodology which comprises of “Data Preparation”, “Training the ML Models”, “Evaluation”, “Validation” and “Comparison”. Please find the detailed overview of each step of the whole experiment.

4.1 Dataset:

First of all, we will discuss and understand the dataset that we are using to train the models and later evaluating the models.

- We have used a dataset with 5000 plus records of email samples of the data and 3000 plus features.
- Features are common individual words used in emails and class label is the target which will identify in which category the email belongs to, based on the individual preference.
- We have three classification target labels "Spam", "Not Spam", and "Suspicious".
- Records are the term frequency of these common words per email. Likewise we are planning to have 5000 plus email records.

We are planning to download the dataset from Kaggle and customize it with some of the email samples that we have out of our inboxes. We have pre-processed the data using various steps before model training such as handle missing values, identify outliers, and normalize features.

4.2 Data Loading:

By now, we have understood how the dataset looks like and what are the contents of it. Now, we will discuss how data is loaded to start the initial steps such as data pre-processing.

To load the data and understand the dataset, we have followed the below steps.

- We have downloaded the above discussed dataset in the "CSV" format and named it as "personal_emails.csv".
- We have used the "Google Colab" – a virtual python environment to perform all the steps in the experiment as part of the project.
- After uploading the dataset into "Google Colab", we have loaded the CSV dataset file using a "Pandas" Data Frame "read_csv()" function.
- To get the initial understanding of the dataset, we have displayed the records in the dataset. We have printed the top 15 records using head() method.
- To find the shape of the email, we have used the shape attribute which prints the rows number and columns number in the dataset.
- And we have printed the unique values of the target variable "predictionclass" and found three classification categories.

4.3 Exploratory Data Analysis:

After loading the data, next important step is to analyse the data as part of the dataset. As part of the EDA, we have performed majorly three steps. Please find them below.

Step 1: Target Data Values: We have first tried to find the unique values of the target variable "predictionclass". We have used the "value_counts()" on the dataframe and found "Spam", "Non-Spam", and "Suspicious" categories. This particular step provided us the insights of the categories present in the dataset.

Step 2: Mapping Target Data Values: We have used the data visualization techniques and plots to understand the categories and getting clarity and interpretability. We have used “Pie” chart to understand the distribution and highlighting the proportion of each category in the dataset.

Step 3: Data Distribution using Bar plot: We have used “Bar Plot” data visualization for detailed understanding of the distribution of email categories. Category count is represented on Y-axis and categories in X-axis.

4.4 Data Pre-Processing:

Next, we need to pre-process the data to cleanly training the ML models. There are many pre-processing techniques available. We have used below 3 techniques.

Technique 1: Handling NULL Values: We have handled the null values in the dataset. Firstly we calculated the null values of each feature using “isnull().sum()” method. This will help us to find the presence of the missing data in the dataset which is crucial for handling the strategies.

Technique 2: Identifying Outliers: Next, we also tried to find the outliers of the dataset to normalize it. We have used the dimensionality reduction using Principal Component Analysis (PCA) to decrease the dimensionality without losing the important information.

We have used the “Isolation Forest” for Anomaly detection to find the outliers in the reduced dataset. And we have used Scatter plots and box plots to clearly visualize the distribution of outliers in the dataset.

Technique 3: Normalization: To make sure all the features are on the similar scale, we have performed normalization on the dataset. “SimpleImputer” is used to insert the missing values by mean strategy. “MinMax” scaling is used to rescale feature values between 0 and 1. Data visualization technique “Box Plot” are used to view the distribution before the normalization and after normalization.

4.5 Split the data for Training and Testing:

Now we have done the pre-processing of the dataset, we will now split the data for training and testing. As part of this step,

- We have divided the dataset into features (X) and the target variable (y), where ‘X’ will have all the columns expect the first one column.
- ‘y’ contains the 'predictionclass' column, which is a target variable.
- Used “train_test_split” function from “sklearn.model_selection” module to split the data into training and testing sets.
- We have divided “70%” of the data as training data and “30%” of the data as testing data. To ensure the reproducibility of the split, we have set random_state parameter to 42.

4.6 <u>Training and Testing the Models</u>:

This section is the main step of the project, where we build, train and test the models. As part of the this project we have built the below ML models.

- Random Forest (GINI)
- Random Forest (Entropy)
- Naïve Bayes Gaussian
- Naïve Bayes Multinomial
- SVM
- Gradient Boosting

4.6.1 <u>Random Forest (GINI)</u>:

Random Forest is one of the ensemble learning algorithm that will build many decision trees during the training out outputs the classes modes using GINI impurity. It uses bagging, random sampling data, and random subsets of features. This will reduce the overfitting and decorrelates trees. During the prediction, it will join the tree outputs using the mode of class predictions based on GINI criteria. It offers the high performance, robustness and very ease of use with hyperparameters like n_estimators, max_depth, min_samples_split, and min_samples_leaf for tuning.

As part of this project –

- Declared a parameter grid which contains various hyper-parameter combinations.
- hyper-parameter combinations were iterated and trained the classifier using scikit-learn's RandomForestClassifier.
- GINI criterion is used for the calculation of impurity.
- Trained the model with training data and evaluated with testing data.
- Computed and printed performance metrics such as accuracy, precision, recall, and F1-score.

4.6.2 <u>Random Forest (Entropy)</u>:

This Random Forest builds decision trees using training and outputs the class ode using the entropy impurity measure. While prediction, this algorithm joins the outputs of various individual trees using the mode of class predictions based on “Entropy”. This increases the model’s predictive accuracy and generalization.

As part of this project –

- Declared a parameter grid which contains various hyper-parameter combinations.
- Selected the Entropy criterion for impurity calculation.
- Hyper-parameter combinations were iterated and trained the classifier using scikit-learn's Random Forest Classifier.
- Trained the model with training data and evaluated with testing data.
- Computed and printed performance metrics such as accuracy, precision, recall, and F1-score.

4.6.3 <u>**Naïve Bayes Gaussian**</u>:

It is a probabilistic classification algorithm which is based out of Bayes' theorem which assumes the features are independent and follows a continuous flow and follow a Gaussian distribution.

As part of this project –

- We have created a Gaussian Naive Bayes classifier and trained it on the training data.
- The predictions of the model were made with the testing data and calculated the performance metrics relied on the predicted and actual labels.
- Computed and printed performance metrics such as accuracy, precision, recall, and F1-score.
- Generated Confusion matrix to provide a breakdown of the predictions generated by model.

4.6.4 <u>**Naïve Bayes Multinomial**</u>:

The Multinomial Naive Bayes model is a type of the Naive Bayes algorithm which is created for the tasks that needs classification where the features are the frequency distributions. It is a probabilistic classification algorithm which is based out of Bayes' theorem which assumes the features are independent.

As part of this project –

- We have created a Multinomial Naive Bayes classifier and trained it on the training data.
- The predictions of the model were made with the testing data and calculated the performance metrics relied on the predicted and actual labels.
- Computed and printed performance metrics such as accuracy, precision, recall, and F1-score.
- Generated Confusion matrix to provide a breakdown of the predictions generated by model.

4.6.5 <u>**SVM**</u>:

Support Vector Machine (SVM) is one of the fantastic supervised learning algorithm, which is used for tasks that needs classification. Optimal hyperplane is the main concept of SVM, where it separates data points into different classes. It maximizes the margin between the targets. It is capable of handling of both linear and non-linear tasks which involves classification.

As part of this project –

- SVM models with various kernel functions such as linear, polynomial, and radial basis function (RBF) is implemented.
- Various hyperparameter configurations are implemented such as “Kernel”, “Regularization parameter”, and specific kernel params like degree and gamma.
- SVM models are trained with training dataset and evaluated with testing dataset.

- Computed and printed performance metrics such as accuracy, precision, recall, and F1-score.

4.6.6 Gradient Boosting:

Gradient Boosting is one of the ensemble learning technique used for classification tasks. It actually adds weak learners to an ensemble, which focuses on the mistakes made by previous ones. It minimizes a loss function iteratively, which helps the algorithm that combines the strengths of multiple weak learners. It is capable of high predictive accuracy and robustness against overfitting.

As part of this project –

- Gradient Boosting Classifier is trained with various hyperparams.
- We have used "number of estimators", "the learning rate", and "maximum depth of each tree" hyperparms.
- The parameter grid gives the different values for n_estimators, learning_rate, and max_depth to find different configurations.
- The model is trained with training data and evaluated with testing data.
- Computed and printed performance metrics such as accuracy, precision, recall, and F1-score.

4.7 Evaluation and Confirm Results:

The evaluation is done with using famous performance metrics – Accuracy, Precision, Recall, and F1-score of a machine learning model.

Accuracy – Overall correctness of the model. It is the ratio of correct predictions to all predications.
Precision – It is the ratio of correct predicted positives to total predicted positives.
Recall – It is the ratio of correct predicted positives to all predictions.
F1-score – It is the HM (harmonic mean) of precision and recall.

We will calculate Accuracy, Precision, Recall, and F1-score attributes parameters for every model we build as part of the project with different hyperparameters. If we get higher percentage values for these performance metrics we can confirm that model is trained well and giving good results. These results will be based on model performance that is trained on 70% of the dataset and tested on remaining 30% of the dataset.

We can compare various trained models performance metrics on the same dataset and can find the best model that has best performance metrics. Here in this project we will implement various supervised machine learning models and will train the models on the dataset. We will get the best model with highest percentages of performance metrics by comparing with other models.

4.7 Cross Validation:

To evaluate the performance of the machine learning model, the cross validation is fantastic technique that can be used to validate. It partitions the dataset into smaller subsets, where the model is trained on each subset and validated on the remaining subset iteratively, which assesses of how good the model will generalize to new not seen data.

As part of this project we manually found SVM with highest 94% of accuracy, so we performed cross validation to confirm it –

- We have defined the SVC() class from scikit-learn.
- Hyper params are defined.
- Different combinations of kernel type (linear, poly, rbf, sigmoid), regularization parameter C, polynomial degree (degree), and kernel coefficient (gamma) are configured.
- Used “GridSearchCV” to perform search over the hyper params
- We have “accuracy” evaluation metric for scoring attribute.

4.8 <u>Results Comparison</u>:

This is the final step of the project, where we have compared all the evaluation metrics of the models that we have used in the project and want to clear about which model is best model for our use case.

- Comparison shows the performance metrics (Accuracy, Precision, Recall, F1-score).
- Comparison done on Random Forest (Gini), Random Forest (Entropy), Naive Bayes (Gaussian), Naive Bayes (Multinomial), SVM (Linear), and Gradient Boosting
- Used Bar chart to achieve this.
- Each bar represents a specific metric.

Some insights-

- Random Forest (Gini) and Random Forest (Entropy) show almost same performance across on all metrics, showing high accuracy, precision, recall, and F1-score.
- SVM (Linear) is highest performance across the metrics.
- Naive Bayes (Gaussian) and Naive Bayes (Multinomial) are at lower scores,

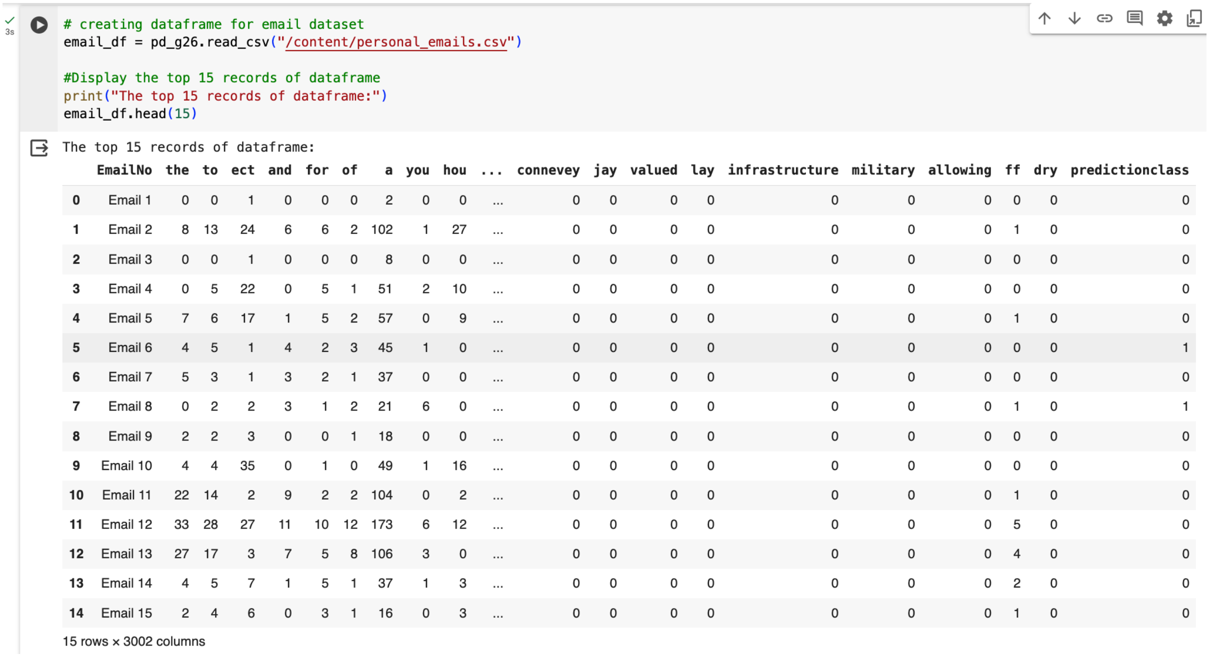
4.9 <u>Tools</u>:

We will be using Python for developing the Machine Learning Models Naïve Bayes [6], SVM [7], Random Forest, and Gradient boosting. These algorithms perfect for classification problem that we are trying to solve. We have used the “SkLearn” library for building and testing models.

5. <u>Results</u>:

Results is the most important section of any report. As part of this section, we will discuss the results of the steps that we have performed in the project. As part of the process/steps we performed in the project that are discussed on the top, we have result we need to discuss. This section discusses various types results from tables that shows the dataset preview to comparing the results of various supervised models evaluation results. Please find the results and their respective analysis for each step we performed in the project.

5.1 <u>Data Loading</u>:



```
# creating dataframe for email dataset
email_df = pd_g26.read_csv("/content/personal_emails.csv")

#Display the top 15 records of dataframe
print("The top 15 records of dataframe:")
email_df.head(15)
```

```
The top 15 records of dataframe:
```

	EmailNo	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastructure	military	allowing	ff	dry	predictionclass
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	0
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	0	0	0	1	0	0
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	0	0	0	0	0	0
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	0	0	0	0	0	0
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0	0	0	1	0	0
5	Email 6	4	5	1	4	2	3	45	1	0	...	0	0	0	0	0	0	0	0	0	1
6	Email 7	5	3	1	3	2	1	37	0	0	...	0	0	0	0	0	0	0	0	0	0
7	Email 8	0	2	2	3	1	2	21	6	0	...	0	0	0	0	0	0	0	1	0	1
8	Email 9	2	2	3	0	0	1	18	0	0	...	0	0	0	0	0	0	0	0	0	0
9	Email 10	4	4	35	0	1	0	49	1	16	...	0	0	0	0	0	0	0	0	0	0
10	Email 11	22	14	2	9	2	2	104	0	2	...	0	0	0	0	0	0	0	1	0	0
11	Email 12	33	28	27	11	10	12	173	6	12	...	0	0	0	0	0	0	0	5	0	0
12	Email 13	27	17	3	7	5	8	106	3	0	...	0	0	0	0	0	0	0	4	0	0
13	Email 14	4	5	7	1	5	1	37	1	3	...	0	0	0	0	0	0	0	2	0	0
14	Email 15	2	4	6	0	3	1	16	0	3	...	0	0	0	0	0	0	0	1	0	0

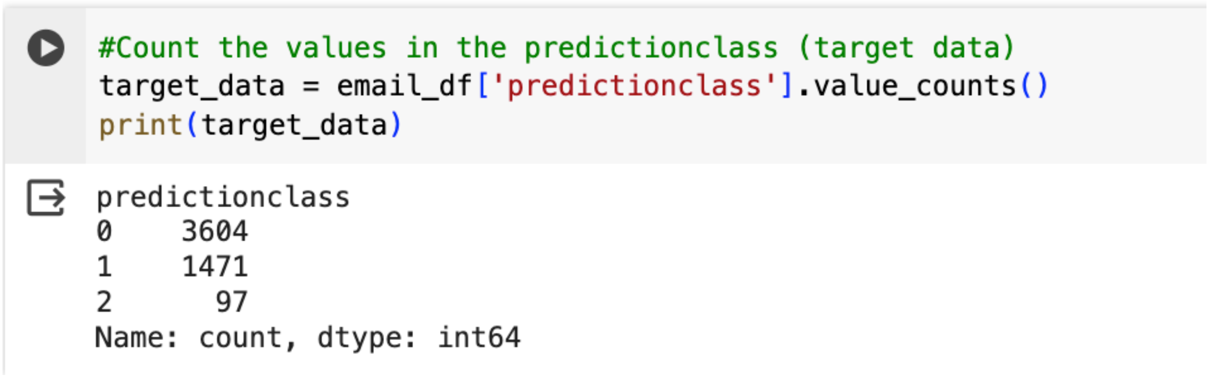
15 rows × 3002 columns

Figure 1: Dataset Loading

Figure 1 shows the top 15 rows of the dataset, where we can find and understand the initial dataset values. We can find the columns starting with "EmailNo" and ends with "Predictionclass".

5.2 <u>Exploratory Data Analysis</u>:

As part of this section we will discuss the results of the EDA.



```
#Count the values in the predictionclass (target data)
target_data = email_df['predictionclass'].value_counts()
print(target_data)
```

```
predictionclass
0    3604
1    1471
2      97
Name: count, dtype: int64
```

Figure 2: EDA 1

Figure 2 shows the unique target values and their respective row count, where we can clearly see that '0' is highest and '2' is lowest.

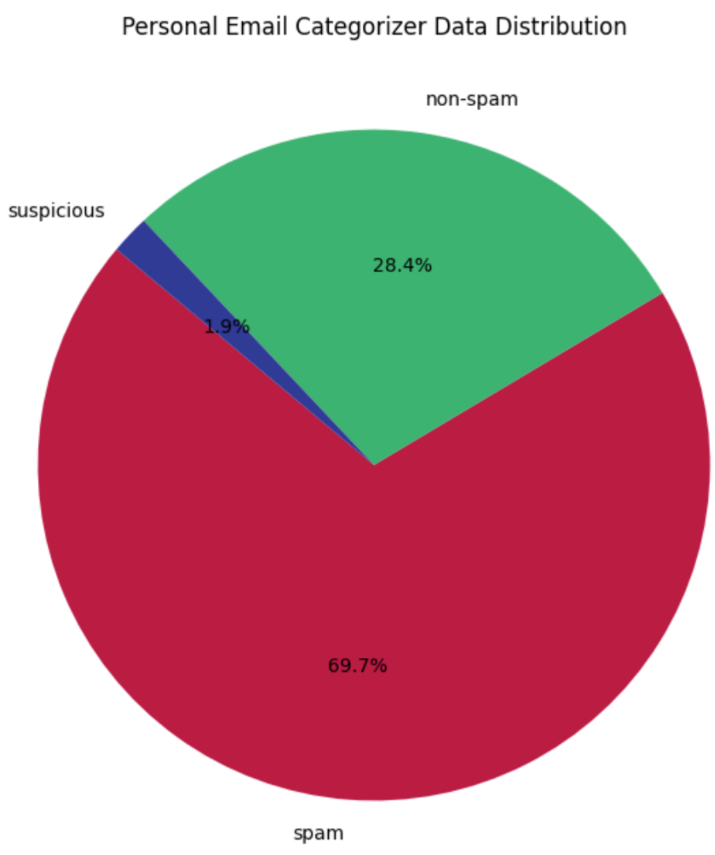


Figure 3: EDA – Pie Chart

Figure 3 represents a pie chart that shows all the target values of the dataset. Overall, we have 3 categories "Spam", "Non-spam", and "Suspicious". Out of the three categories, Spam has highest percentage in the dataset with the value of "69.7%" followed by "non-spam" with "28.4%". Suspicious category is least with "1.9%". All the categories are coloured with different colours for easy understanding.

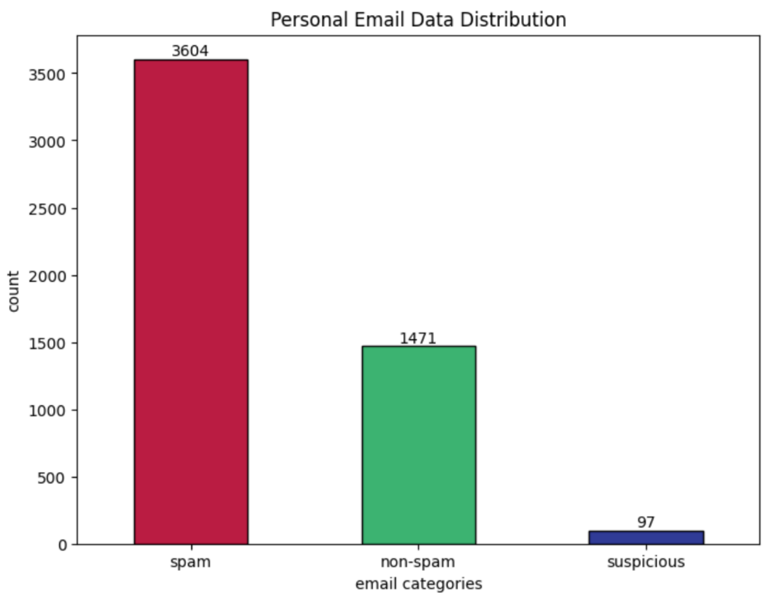
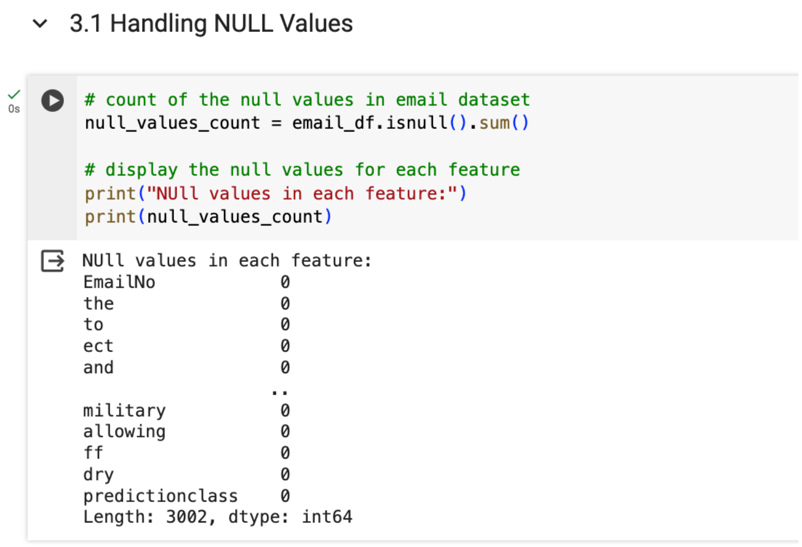


Figure 4: EDA – Bar Plot

Figure 4 represents a Bar plot that shows, how the target values are distributed. Overall as discussed three categories were found and a bar plot is plotted to see clearly how many messages are present in the dataset. We can see the spam has 3604 emails records and non-spam with 1471 email records and least is suspicious with 97 records.

5.3 Data Pre-processing:

As part of this section we will discuss the results of the data pre-processing.

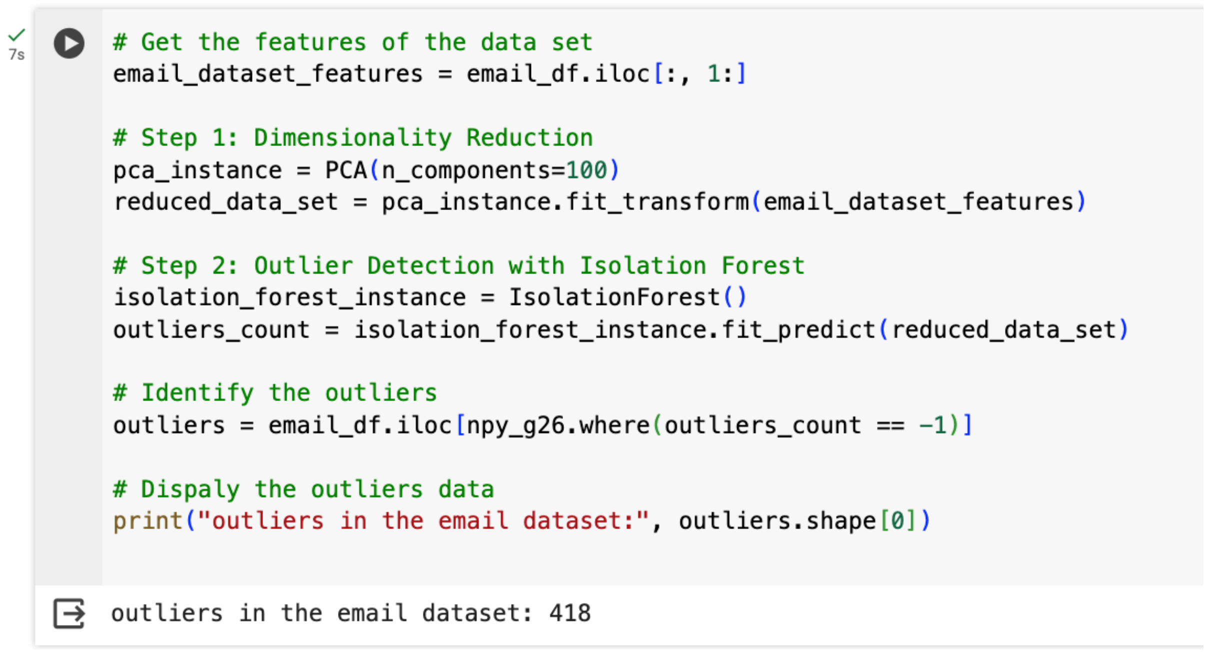

3.1 Handling NULL Values

```
# count of the null values in email dataset
null_values_count = email_df.isnull().sum()

# display the null values for each feature
print("NUll values in each feature:")
print(null_values_count)
```

```
NUll values in each feature:
EmailNo           0
the               0
to                0
ect               0
and               0
                 ..
military          0
allowing          0
ff                0
dry               0
predictionclass   0
Length: 3002, dtype: int64
```

Figure 5: Handling the Null Values

Figure 5 represents the result of finding the NULL values as part of the dataset. Results shows there are no null values as part of features.



```
# Get the features of the data set
email_dataset_features = email_df.iloc[:, 1:]

# Step 1: Dimensionality Reduction
pca_instance = PCA(n_components=100)
reduced_data_set = pca_instance.fit_transform(email_dataset_features)

# Step 2: Outlier Detection with Isolation Forest
isolation_forest_instance = IsolationForest()
outliers_count = isolation_forest_instance.fit_predict(reduced_data_set)

# Identify the outliers
outliers = email_df.iloc[npy_g26.where(outliers_count == -1)]

# Dispaly the outliers data
print("outliers in the email dataset:", outliers.shape[0])
```

```
outliers in the email dataset: 418
```

Figure 6: Finding the outliers.

Figure 6 shows the finding the outliers in the dataset. After running the code, we found 418 outliers, which initiates us to normalize the data.

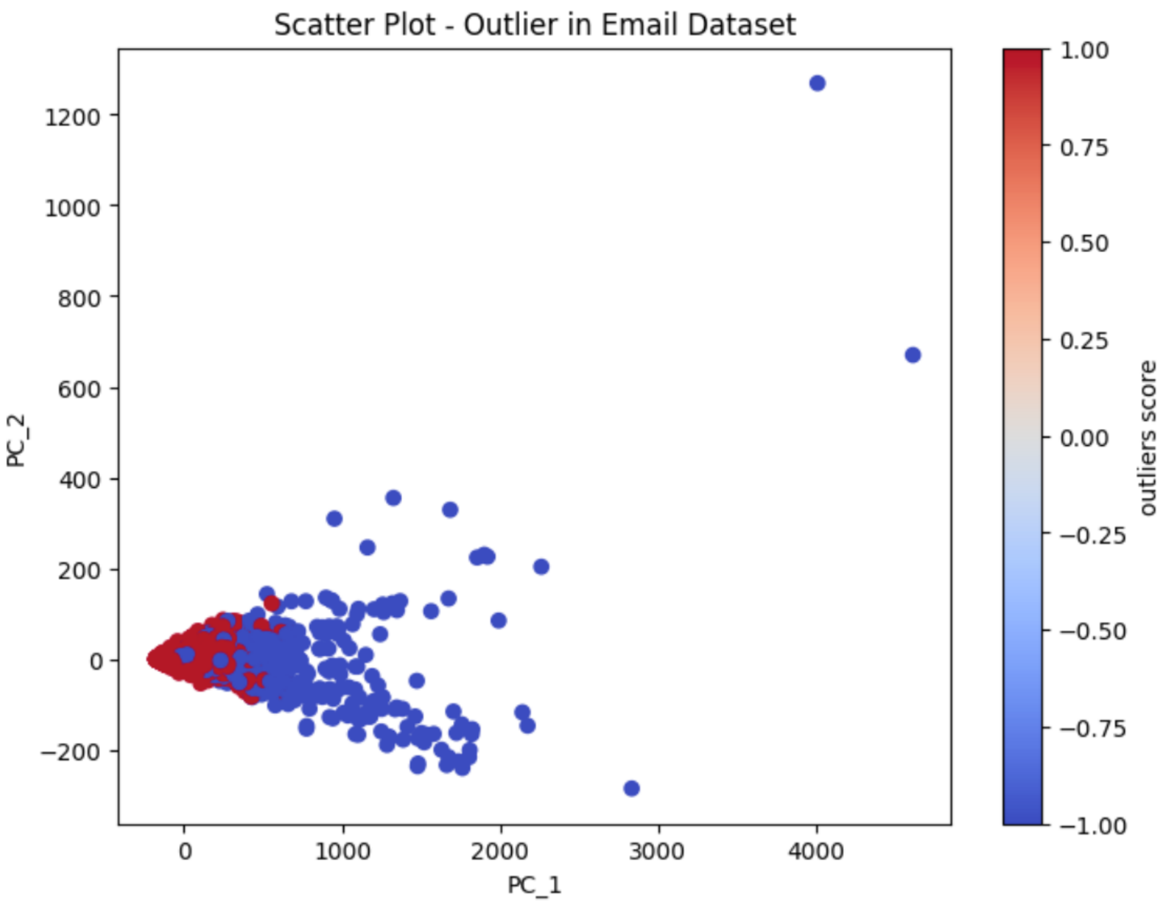
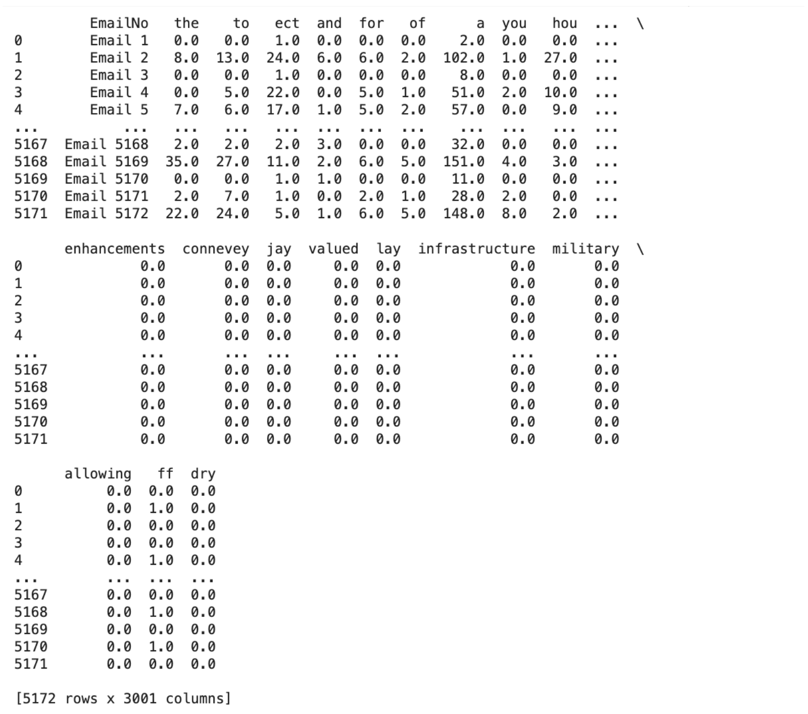


Figure 7: Scatter plot for Outliers

Figure 7 shows the outliers represented in scatter plot. As part of the plot the x-axis is PC-1 (principle component - 1) and y-axis is PC-2 (principle component - 2), we have a found on the right which represents the outliers scope. We can clearly see that blue dots out of the cluster, which clearly shows the outliers.

As we have outliers and to make sure all the features are on the similar scale, we have performed normalization on the dataset. “SimpleImputer” is used to insert the missing values by mean strategy. “MinMax” scaling is used to rescale feature values between 0 and 1. Data visualization technique “Box Plot” are used to view the distribution before the normalization and after normalization.

Figure 8 represents the output of the imputed dataset after the normalization. We have imputed with mean value.


```
        EmailNo   the    to   ect  and  for   of      a  you   hou  ...  \
0       Email 1   0.0   0.0   1.0  0.0  0.0  0.0    2.0  0.0   0.0  ...
1       Email 2   8.0  13.0  24.0  6.0  6.0  2.0  102.0  1.0  27.0  ...
2       Email 3   0.0   0.0   1.0  0.0  0.0  0.0    8.0  0.0   0.0  ...
3       Email 4   0.0   5.0  22.0  0.0  5.0  1.0   51.0  2.0  10.0  ...
4       Email 5   7.0   6.0  17.0  1.0  5.0  2.0   57.0  0.0   9.0  ...
...         ...   ...   ...   ...  ...  ...  ...    ...  ...   ...  ...
5167 Email 5168   2.0   2.0   2.0  3.0  0.0  0.0   32.0  0.0   0.0  ...
5168 Email 5169  35.0  27.0  11.0  2.0  6.0  5.0  151.0  4.0   3.0  ...
5169 Email 5170   0.0   0.0   1.0  1.0  0.0  0.0   11.0  0.0   0.0  ...
5170 Email 5171   2.0   7.0   1.0  0.0  2.0  1.0   28.0  2.0   0.0  ...
5171 Email 5172  22.0  24.0   5.0  1.0  6.0  5.0  148.0  8.0   2.0  ...

     enhancements  connevey  jay  valued  lay  infrastructure  military  \
0             0.0       0.0  0.0     0.0  0.0             0.0       0.0
1             0.0       0.0  0.0     0.0  0.0             0.0       0.0
2             0.0       0.0  0.0     0.0  0.0             0.0       0.0
3             0.0       0.0  0.0     0.0  0.0             0.0       0.0
4             0.0       0.0  0.0     0.0  0.0             0.0       0.0
...           ...       ...  ...     ...  ...             ...       ...
5167          0.0       0.0  0.0     0.0  0.0             0.0       0.0
5168          0.0       0.0  0.0     0.0  0.0             0.0       0.0
5169          0.0       0.0  0.0     0.0  0.0             0.0       0.0
5170          0.0       0.0  0.0     0.0  0.0             0.0       0.0
5171          0.0       0.0  0.0     0.0  0.0             0.0       0.0

     allowing   ff  dry
0         0.0  0.0  0.0
1         0.0  1.0  0.0
2         0.0  0.0  0.0
3         0.0  0.0  0.0
4         0.0  1.0  0.0
...       ...  ...  ...
5167      0.0  0.0  0.0
5168      0.0  1.0  0.0
5169      0.0  0.0  0.0
5170      0.0  1.0  0.0
5171      0.0  0.0  0.0

[5172 rows x 3001 columns]
```
Figure 8: Normalization - 1.

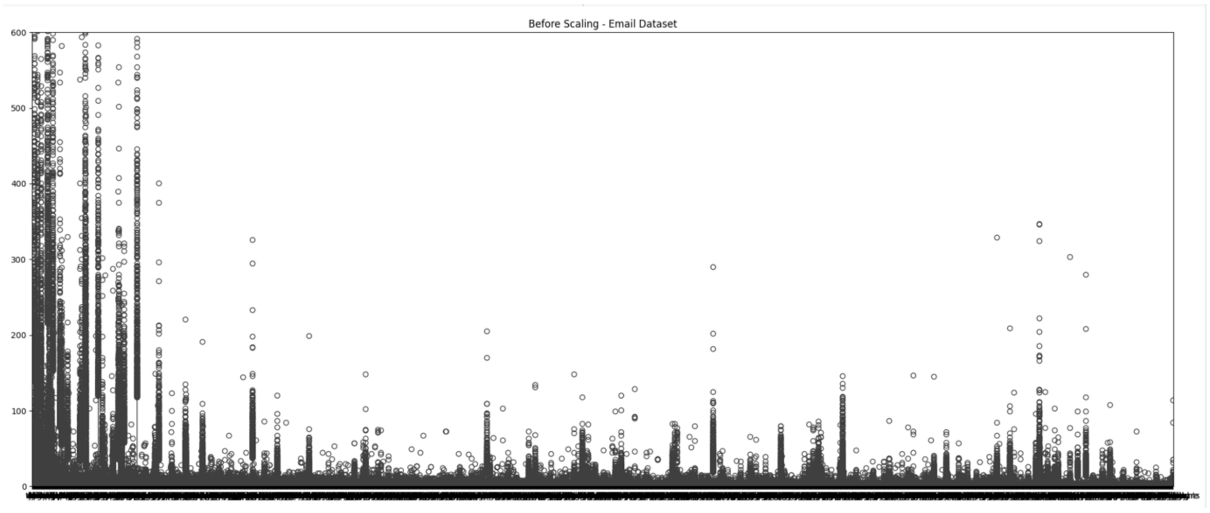


Figure 9: Normalization – Before Scaling.

Figure 9 represents before scaling the dataset, on the x-axis we have all the features, and on y-axis we have the values of respective features. We can clearly see that values of some features reach 600+ value. So next we will see the data how the dataset looks like after scaling. After the normalization, the data will be scaled within from 0 to 1. Figure 10 represents after scaling the dataset, on the x-axis we have all the features, and on y-axis we have the values of respective features. We can clearly see that all values resides between 0 and 1.

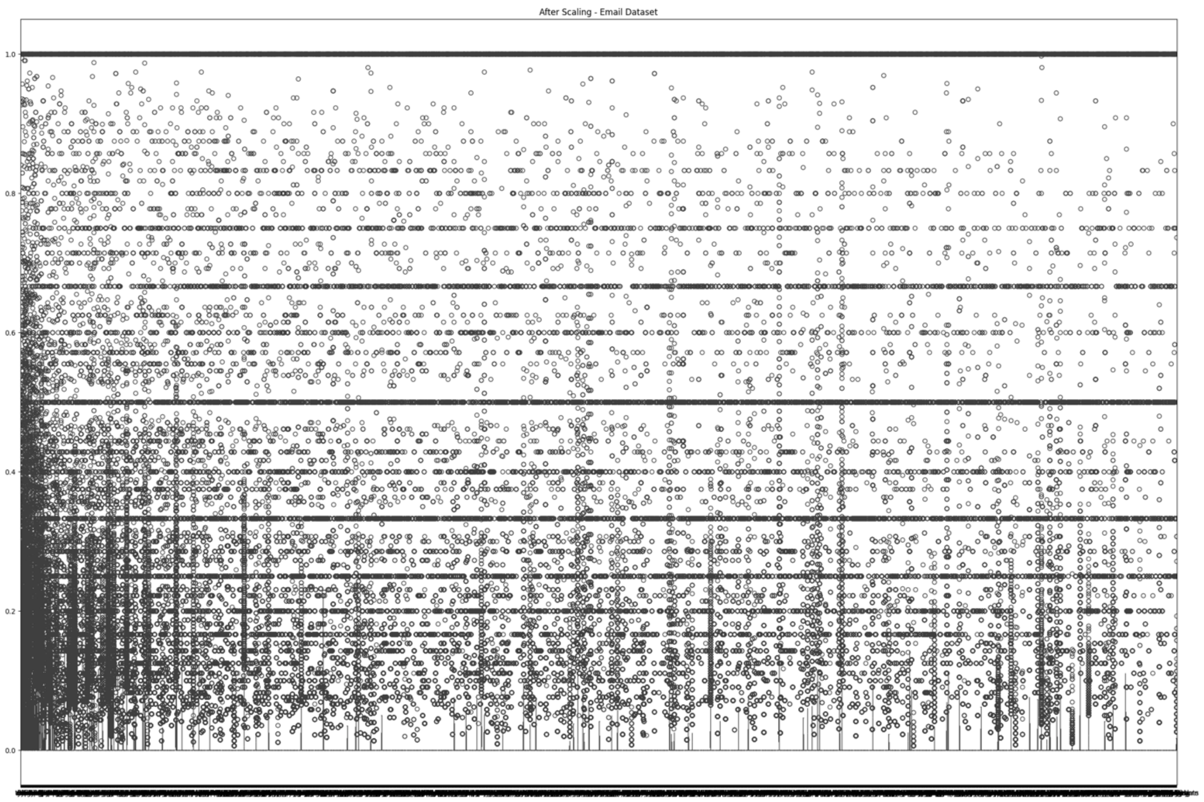


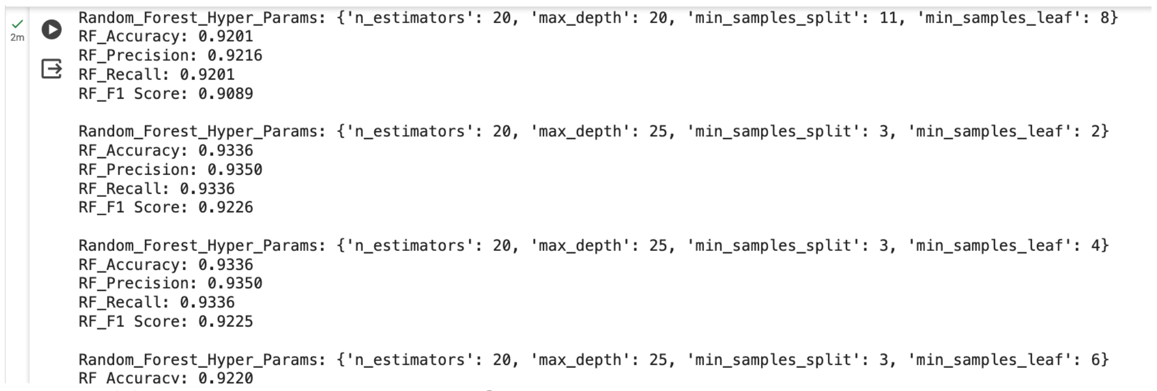
Figure 10: Normalization – Before Scaling.

5.4 <u>Training Models</u>:

After data pre-processing, we will split the data into 70% into training set and 30% testing set. Then we will subject the data to various ML models.

5.4.1 <u>Random Forest using GINI impurity</u>:

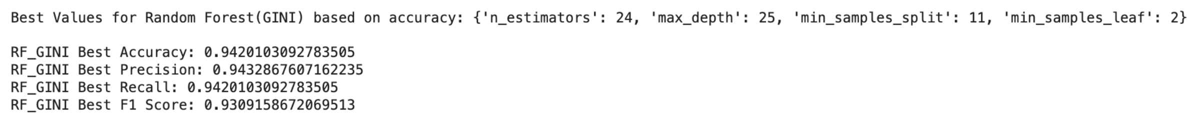
First supervised machine learning algorithm we have used to subject the train data and test data is "Random Forest" that uses GINI impurity.



```
Random_Forest_Hyper_Params: {'n_estimators': 20, 'max_depth': 20, 'min_samples_split': 11, 'min_samples_leaf': 8}
RF_Accuracy: 0.9201
RF_Precision: 0.9216
RF_Recall: 0.9201
RF_F1 Score: 0.9089

Random_Forest_Hyper_Params: {'n_estimators': 20, 'max_depth': 25, 'min_samples_split': 3, 'min_samples_leaf': 2}
RF_Accuracy: 0.9336
RF_Precision: 0.9350
RF_Recall: 0.9336
RF_F1 Score: 0.9226

Random_Forest_Hyper_Params: {'n_estimators': 20, 'max_depth': 25, 'min_samples_split': 3, 'min_samples_leaf': 4}
RF_Accuracy: 0.9336
RF_Precision: 0.9350
RF_Recall: 0.9336
RF_F1 Score: 0.9225

Random_Forest_Hyper_Params: {'n_estimators': 20, 'max_depth': 25, 'min_samples_split': 3, 'min_samples_leaf': 6}
RF_Accuracy: 0.9220
```

Figure 11: Random Forest using GINI impurity

Figure 11 represents the output while training the Random Forest ML models that is using GINI impurity. In the output we can see the ML model uses various hyperparams n_estimators, max_depth, min_samples_split, and min_samples_leaf for tuning the models. It has used many combinations of these tuning params to build a strong model.



```
Best Values for Random Forest(GINI) based on accuracy: {'n_estimators': 24, 'max_depth': 25, 'min_samples_split': 11, 'min_samples_leaf': 2}

RF_GINI Best Accuracy: 0.9420103092783505
RF_GINI Best Precision: 0.9432867607162235
RF_GINI Best Recall: 0.9420103092783505
RF_GINI Best F1 Score: 0.9309158672069513
```

Figure 12: Random Forest using GINI impurity - metrics

Metric	Value
Accuracy	0.94201030927
Precision	0.9432867
Recall	0.9420103092
F1 Score	0.930915867

Table 1: Random Forest using GINI impurity - metrics

Table 1 represents the evaluation metrics of the Random Forest using GINI impurity.

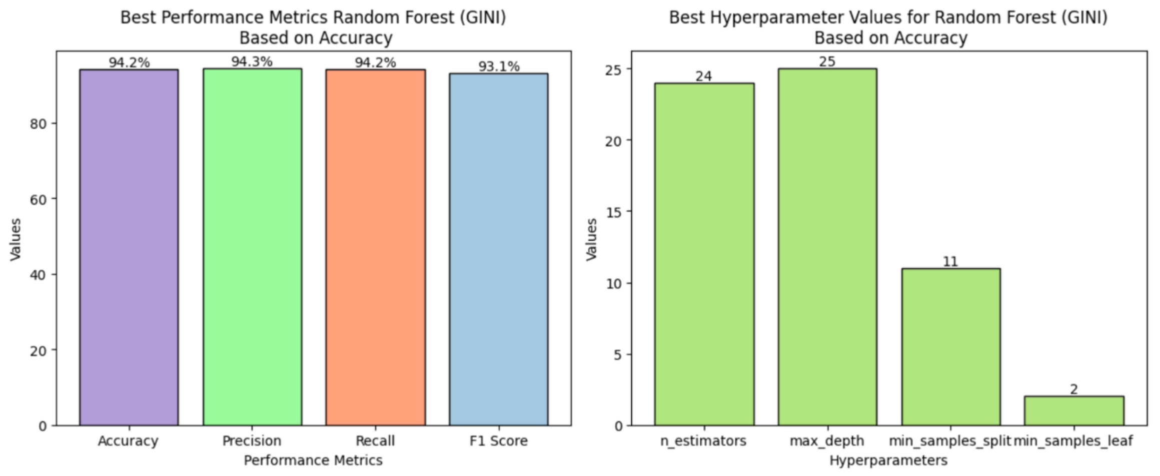


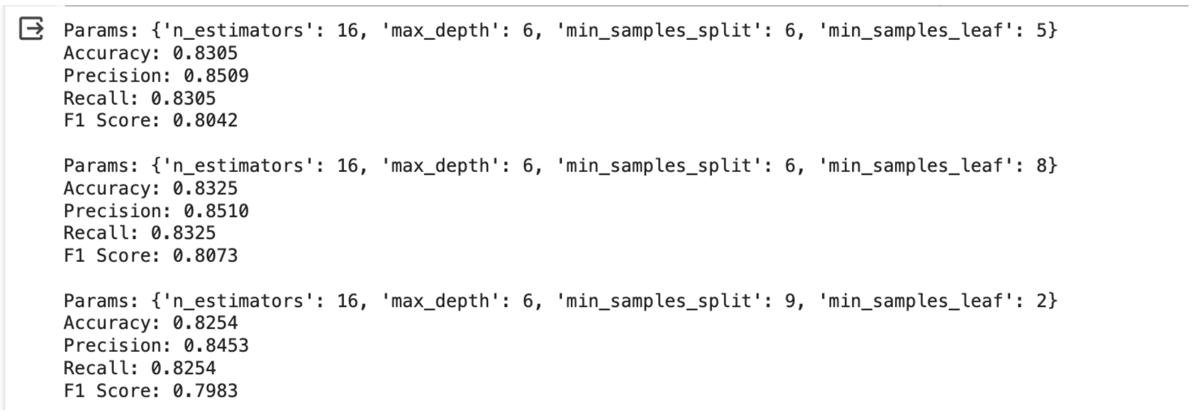
Figure 13: Random Forest using GINI impurity - plots

Figure 13 shows the bar plots for the evaluation metrics and best tuning params for the ML model. Left bar plot shows the evaluation metrics whereas right bar plot shows the bar plot of the best tuning params. Best Hyperparam are n_estimators = 24, max_depth = 25, min_samples_split = 11, and min_samples_leaf = 12.

5.4.2 <u>Random Forest using Entropy impurity</u>:

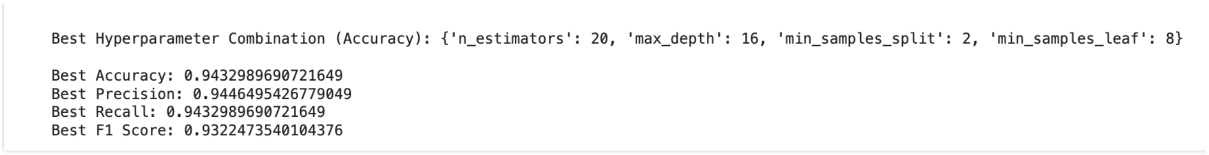
Second supervised machine learning algorithm we have used to subject the train data and test data is “Random Forest” that uses Entropy impurity.

Figure 14 represents the output while training the Random Forest ML models that is using Entropy impurity. In the output we can see the ML model uses various hyperparams n_estimators, max_depth, min_samples_split, and min_samples_leaf for tuning the models. It has used many combinations of these tuning params to build a strong model.



```
Params: {'n_estimators': 16, 'max_depth': 6, 'min_samples_split': 6, 'min_samples_leaf': 5}
Accuracy: 0.8305
Precision: 0.8509
Recall: 0.8305
F1 Score: 0.8042

Params: {'n_estimators': 16, 'max_depth': 6, 'min_samples_split': 6, 'min_samples_leaf': 8}
Accuracy: 0.8325
Precision: 0.8510
Recall: 0.8325
F1 Score: 0.8073

Params: {'n_estimators': 16, 'max_depth': 6, 'min_samples_split': 9, 'min_samples_leaf': 2}
Accuracy: 0.8254
Precision: 0.8453
Recall: 0.8254
F1 Score: 0.7983
```

Figure 14: Random Forest using Entropy impurity



```
Best Hyperparameter Combination (Accuracy): {'n_estimators': 20, 'max_depth': 16, 'min_samples_split': 2, 'min_samples_leaf': 8}

Best Accuracy: 0.9432989690721649
Best Precision: 0.9446495426779049
Best Recall: 0.9432989690721649
Best F1 Score: 0.9322473540104376
```

Figure 15: Random Forest using GINI impurity - metrics

Metric	Value
Accuracy	0.9432989690721649
Precision	0.9446495426779049
Recall	0.9432989690721649
F1 Score	0.9322473540104376

Table 2: Random Forest using Entropy impurity - metrics

Table 2 represents the evaluation metrics of the Random Forest using Entropy impurity.

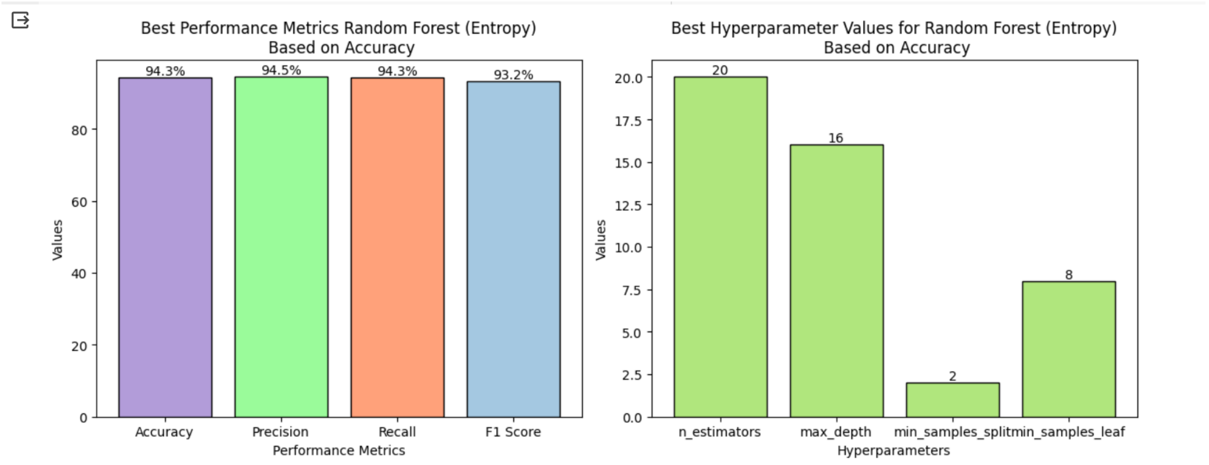
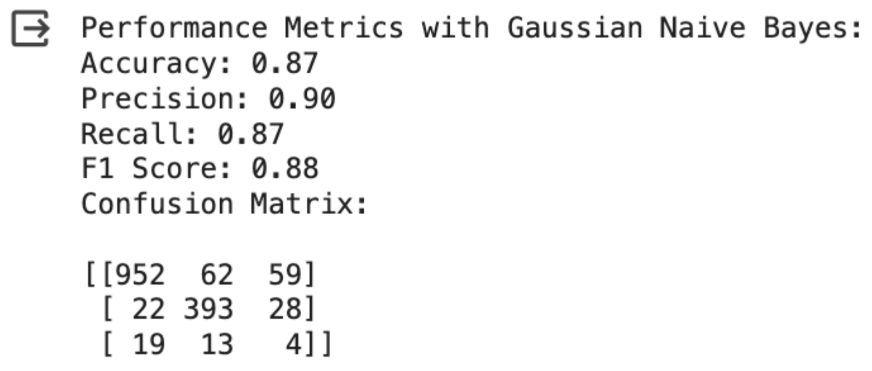


Figure 16: Random Forest using Entropy impurity - plots

Figure 16 shows the bar plots for the evaluation metrics and best tuning params for the ML model. Left bar plot shows the evaluation metrics whereas right bar plot shows the bar plot of the best tuning params. Best Hyperparam are n_estimators = 20, max_depth = 26, min_samples_split = 2, and min_samples_leaf = 8.

5.4.3 Naïve Bayes Gaussian:

Next, supervised machine learning algorithm we have used to subject the train data and test data is Naïve Bayes Gaussian.



```
Performance Metrics with Gaussian Naive Bayes:
Accuracy: 0.87
Precision: 0.90
Recall: 0.87
F1 Score: 0.88
Confusion Matrix:

[[952  62  59]
 [ 22 393  28]
 [ 19  13   4]]
```

Figure 17: Evaluation metrics and Confusion matrix.

Figure 17 represents the performance metrics with gaussian type of Naïve Bayes algorithm. We have also plotted confusion matrix that represents the models important information.

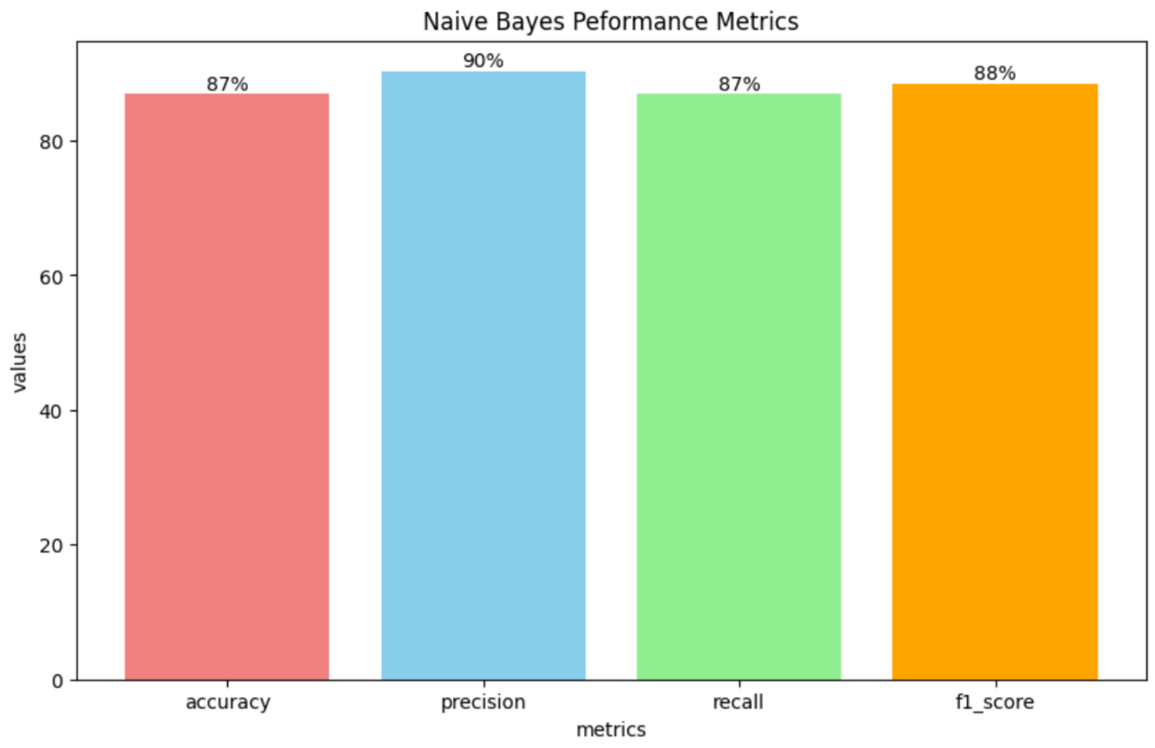


Figure 18: plot

Figure 18 represents the bar plot that plots the evaluation metrics of the Naïve Bayes Gaussian supervised machine learning algorithm.

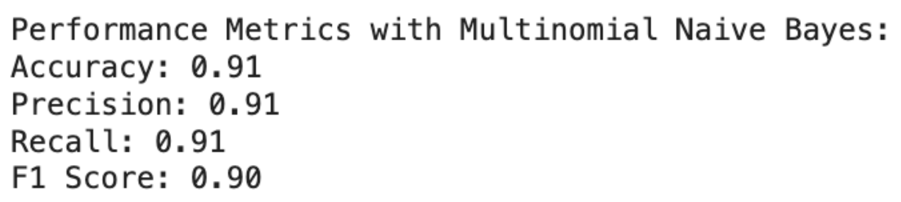
Metric	Value
Accuracy	0.87
Precision	0.90
Recall	0.87
F1 Score	0.88

Table 3: Naïve bayes – Gaussian

Table 3 represents the evaluation metrics of the Naïve bayes – Gaussian.

5.4.4 Naïve Bayes Multinomial:

Next, supervised machine learning algorithm we have used to subject the train data and test data is Naïve Bayes Multinomial.



```
Performance Metrics with Multinomial Naive Bayes:
Accuracy: 0.91
Precision: 0.91
Recall: 0.91
F1 Score: 0.90
```

Figure 19: Evaluation metrics.

Figure 19 represents the performance metrics with Multinomial type of Naïve Bayes algorithm. When compared with Gaussian type in figure 17, multinomial has done better with this dataset.

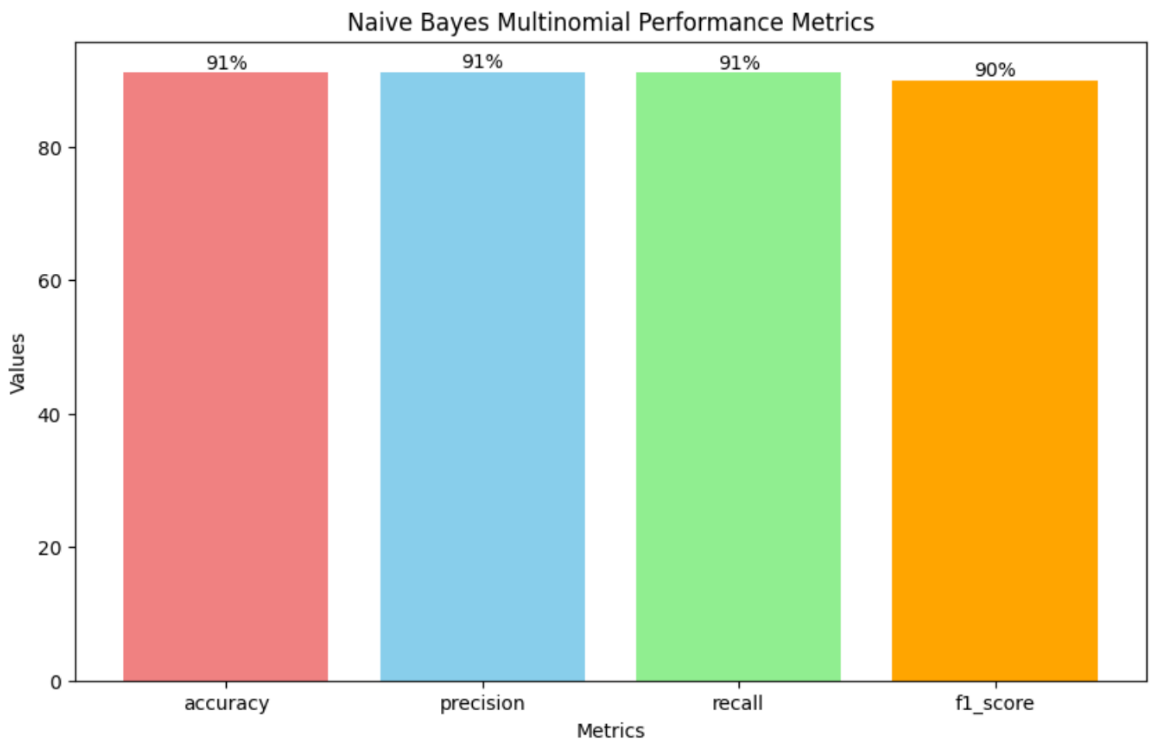


Figure 20: plot

Figure 20 represents the bar plot that plots the evaluation metrics of the Naïve Bayes Multinomial supervised machine learning algorithm.

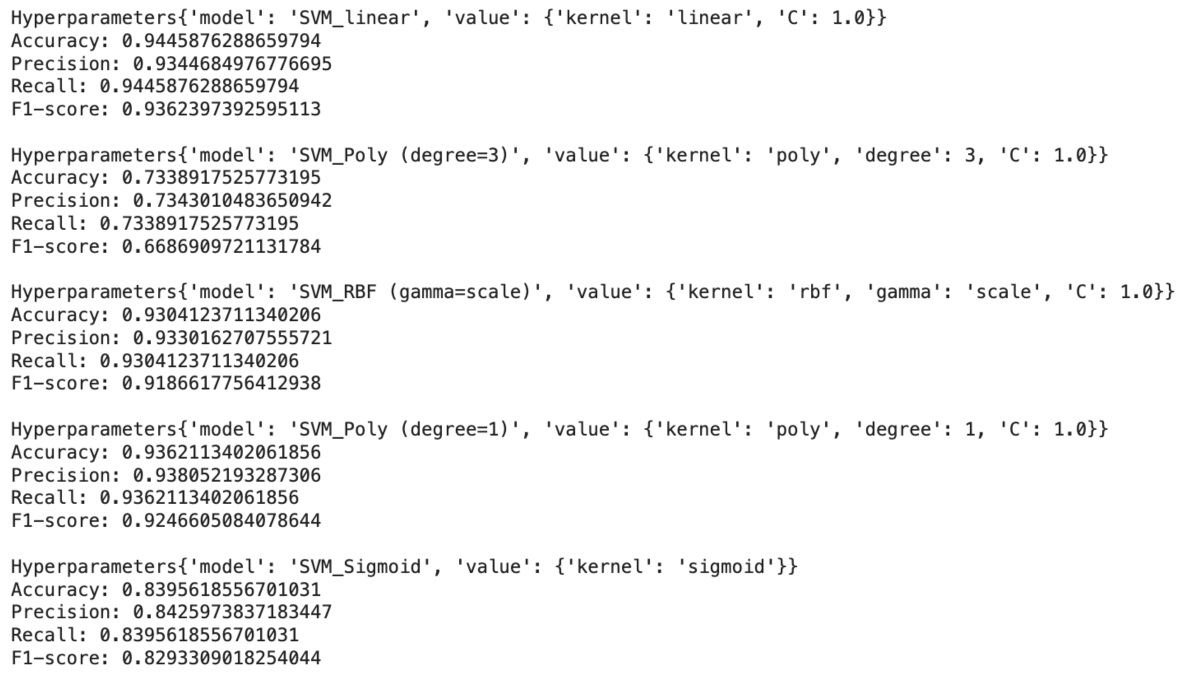
Metric	Value
Accuracy	0.91
Precision	0.91
Recall	0.91
F1 Score	0.90

Table 4: Naïve bayes – Multinomial

Table 4 represents the evaluation metrics of the Naïve bayes – Multinomial.

5.4.5 SVM:

Next, supervised machine learning algorithm we have used to subject the train data and test data is SVM with various hyper parameters.



```
Hyperparameters{'model': 'SVM_linear', 'value': {'kernel': 'linear', 'C': 1.0}}
Accuracy: 0.9445876288659794
Precision: 0.9344684976776695
Recall: 0.9445876288659794
F1-score: 0.9362397392595113

Hyperparameters{'model': 'SVM_Poly (degree=3)', 'value': {'kernel': 'poly', 'degree': 3, 'C': 1.0}}
Accuracy: 0.7338917525773195
Precision: 0.7343010483650942
Recall: 0.7338917525773195
F1-score: 0.6686909721131784

Hyperparameters{'model': 'SVM_RBF (gamma=scale)', 'value': {'kernel': 'rbf', 'gamma': 'scale', 'C': 1.0}}
Accuracy: 0.9304123711340206
Precision: 0.9330162707555721
Recall: 0.9304123711340206
F1-score: 0.9186617756412938

Hyperparameters{'model': 'SVM_Poly (degree=1)', 'value': {'kernel': 'poly', 'degree': 1, 'C': 1.0}}
Accuracy: 0.9362113402061856
Precision: 0.938052193287306
Recall: 0.9362113402061856
F1-score: 0.9246605084078644

Hyperparameters{'model': 'SVM_Sigmoid', 'value': {'kernel': 'sigmoid'}}
Accuracy: 0.8395618556701031
Precision: 0.8425973837183447
Recall: 0.8395618556701031
F1-score: 0.8293309018254044
```

Figure 21: Evaluation metrics.

Figure 21 represents the performance metrics with SVM algorithm with various hyper parameters. Out of all the hyper parameters, we can see that “SVM linear” has performed very good with highest accuracy.

Metric	SVM Linear	SVM poly (degree 3)	SVM RBF	SVM Poly (degree 1)	SVM Sigmoid
Accuracy	0.94458	0.733	0.93041	0.93621	0.83956
Precision	0.9344	0.73430	0.93301	0.93805	0.84259
Recall	0.94458	0.7338	0.93041	0.9362113	0.83956
F1 Score	0.9362	0.66869	0.9186	0.924660	0.8293

Table 5: SVM with different hyper params

Table 5 represents the evaluation metrics of the SVM with different hyper params.

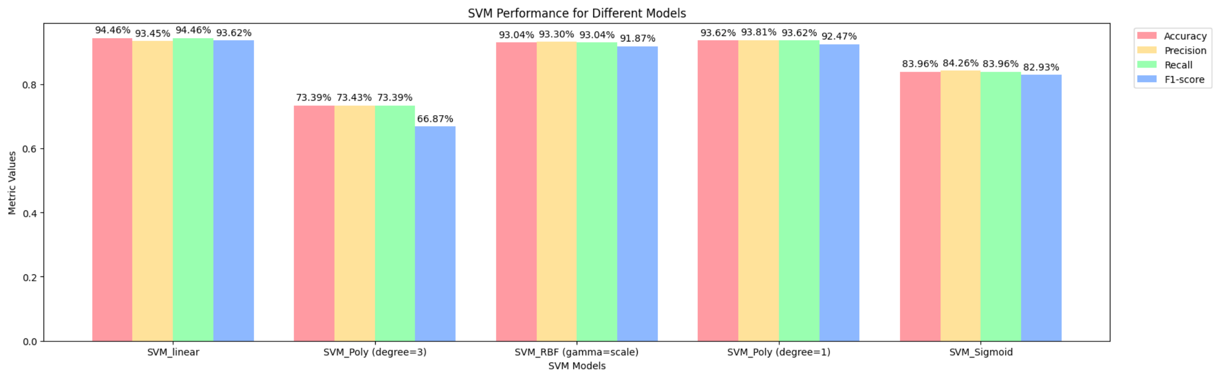
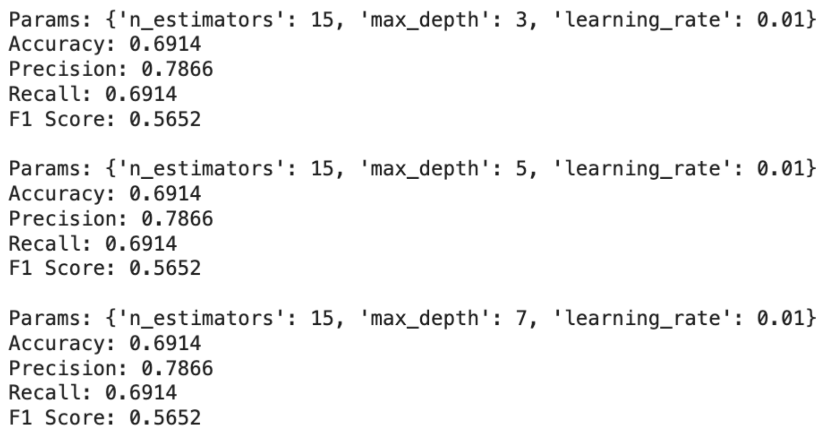


Figure 22: Plot

Figure 22 represents the bar plot that plots the evaluation metrics of the SVM supervised machine learning algorithm with different hyper params.

5.4.6 <u>Gradient Boosting</u>:

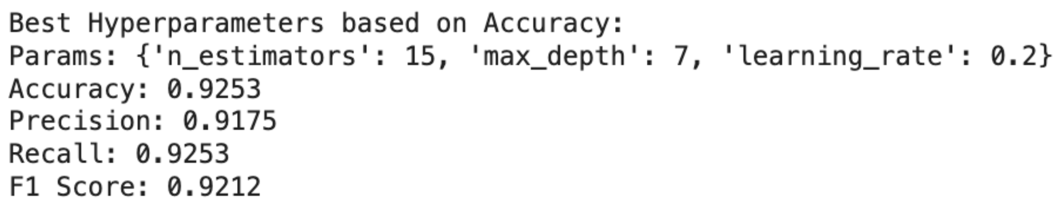
Next, supervised machine learning algorithm we have used to subject the train data and test data is Gradient Boosting with various tuning parameters.



```
Params: {'n_estimators': 15, 'max_depth': 3, 'learning_rate': 0.01}
Accuracy: 0.6914
Precision: 0.7866
Recall: 0.6914
F1 Score: 0.5652

Params: {'n_estimators': 15, 'max_depth': 5, 'learning_rate': 0.01}
Accuracy: 0.6914
Precision: 0.7866
Recall: 0.6914
F1 Score: 0.5652

Params: {'n_estimators': 15, 'max_depth': 7, 'learning_rate': 0.01}
Accuracy: 0.6914
Precision: 0.7866
Recall: 0.6914
F1 Score: 0.5652
```

Figure 23: Gradient Boosting

Figure 23 represents the output while training the Gradient boosting ML models that is using various tuning params. In the output we can see the ML model uses various hyperparams n_estimators, max_depth, and learning_rate for tuning the models. It has used many combinations of these tuning params to build a strong model.



```
Best Hyperparameters based on Accuracy:
Params: {'n_estimators': 15, 'max_depth': 7, 'learning_rate': 0.2}
Accuracy: 0.9253
Precision: 0.9175
Recall: 0.9253
F1 Score: 0.9212
```

Figure 24: Evaluation metrics and Best Hyper params.

Figure 24 represents the performance metrics with Gradient Boosting algorithm. We have also printed best hyper params that represents the models important information.

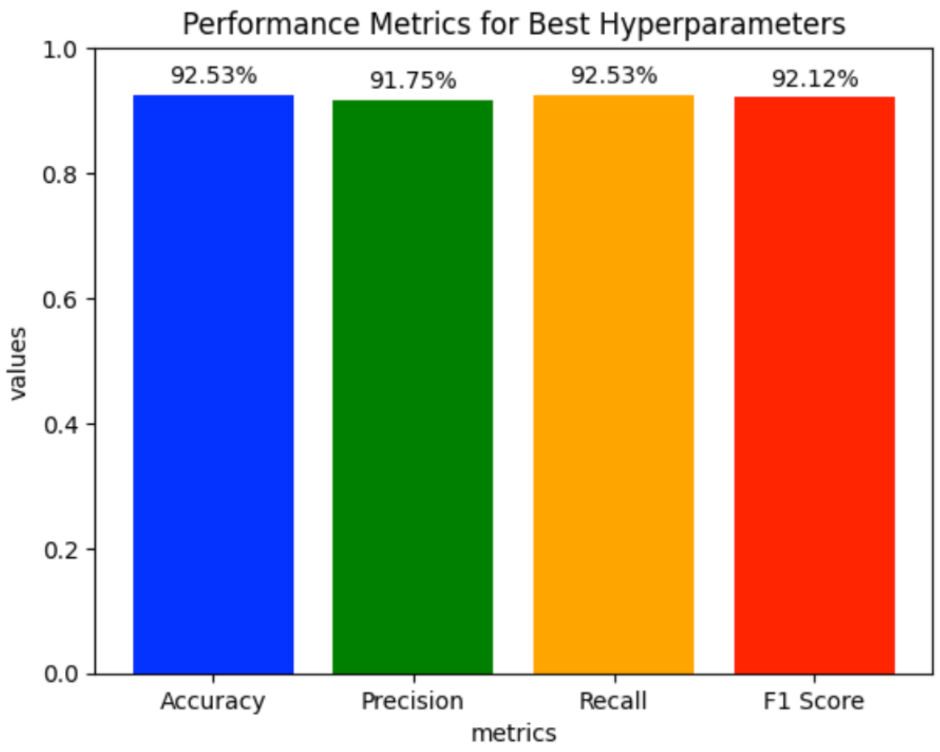


Figure 25: Plot

Figure 25 represents the bar plot that plots the evaluation metrics of the Gradient Boosting supervised machine learning algorithm with different hyper params.

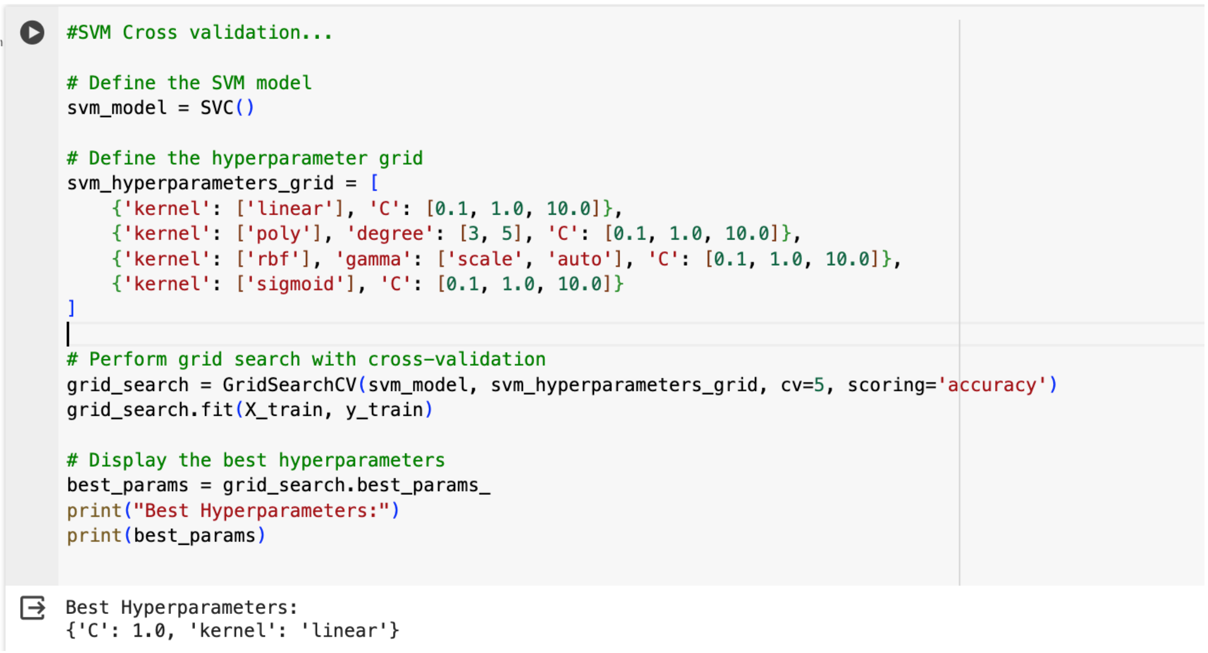
Metric	Value
Accuracy	0.9253
Precision	0.9175
Recall	0.9253
F1 Score	0.9212

Table 6: Naïve bayes – Multinomial

Table 6 represents the evaluation metrics of the Gradient Boosting.

5.4.7 Cross Validation:

Figure 27 represents the “Cross Validation” done on the SVM. As we manually found that SVM linear has generated highest accuracy, we also want to find that SVM linear is the best model using “GridSearchCV” cross validator. And we have successfully got the result as “SVM linear” by “GridSearchCV” cross validator API.



```
#SVM Cross validation...

# Define the SVM model
svm_model = SVC()

# Define the hyperparameter grid
svm_hyperparameters_grid = [
    {'kernel': ['linear'], 'C': [0.1, 1.0, 10.0]},
    {'kernel': ['poly'], 'degree': [3, 5], 'C': [0.1, 1.0, 10.0]},
    {'kernel': ['rbf'], 'gamma': ['scale', 'auto'], 'C': [0.1, 1.0, 10.0]},
    {'kernel': ['sigmoid'], 'C': [0.1, 1.0, 10.0]}
]

# Perform grid search with cross-validation
grid_search = GridSearchCV(svm_model, svm_hyperparameters_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Display the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:")
print(best_params)
```

```
Best Hyperparameters:
{'C': 1.0, 'kernel': 'linear'}
```

Figure 26: Cross Validation.

5.4.7 <u>Comparing Results</u>:

By now, we have trained all the models and evaluated them one by one, now we will compare all the model evaluations and plot on a bar plot. On X-axis we will consider taking the models and Y-axis we will consider the percentage. Out of all "SVM Linear" has best accuracy and other params and it is also highlighted. Lowest accuracy was recorded for Naïve Bayes Gaussian. Figure 27 shows the final comparison bar plot.

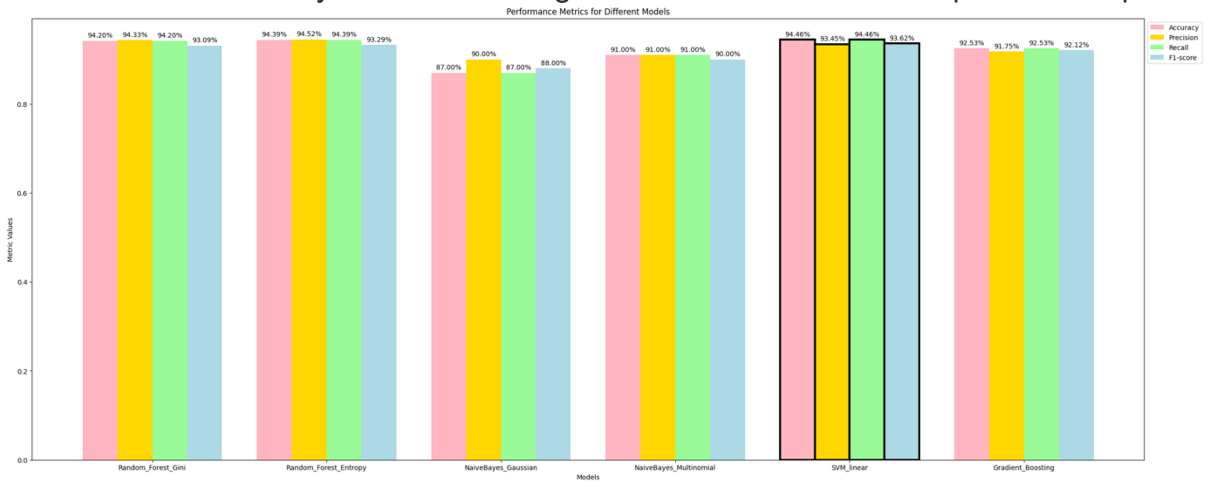


Figure 27: final Comparison Plot

6. <u>Related Work</u>:

As part of the Related work section, we will compare the existing solutions that are trying and solved partial problem. As technology is growing these days, there are numerous email clients such as Gmail [9], Outlook [10], Yahoo, etc.. developed standard categorizers which are capable of classifying the emails which are arriving their user’s inboxes with the following limitations.

- **Incapable of Custom Classification:** These Clients follow their own category strategizer to categorize the emails and provides no customization.
- **Classification Inconsistency:** While classifying the email there will be lot of inconsistency.
- **Wrong classification of Spam:** Misclassification is at its peaks.
- **Heavy Dependency on Metadata:** These clients over dependent on the metadata for the email classification which causes limited classification.
- **Manual rules:** Very limited customization based on the sender email address.
- **Limited Adaptability of user behaviour:** Adaptability is limited of the user patterns.

“Personal Email Categorizer” provides classification of emails in a very customized way to the users who wants to utilize it. This project is very capable of classifying the emails very consistently. We have used 6 types of major supervised Machine Learning models. We have “Random Forest (GINI & Entropy)”, “Naïve Bayes (Gaussian & Multinomial)”, SVM and Gradient Boosting with various types of hyper params. Spam/Non-spam/Suspicious classification will be performed correctly as we can detect the spam email patterns based on the user preferences. We have achieved a 94.5% accuracy in predicting the classification using SVM model. In future we can take this SVM model and build a live model which can take the input from the live GMAIL or any other inbox and classify them. The unique thing about this project is we have done “Cross Validation” to confirm the accuracy of the SVM linear is the best for the take dataset. This approach will give a customize way of categorization of emails to the users.

7. <u>Conclusion</u>:

In conclusion, Personal Email Categorizer can a comprehensive solution for categorizing the emails and reduce the burden on the individuals and increase their productivity. It also reduces the unnecessary spamming these days, the individuals are suffering with. This project drastically reduces the manual errors that will occur while organizing the emails. Finally, SVM (Linear) is highest performance across the metrics. Random Forest (Gini) and Random Forest (Entropy) show almost same performance across on all metrics, showing high accuracy, precision, recall, and F1-score. Naive Bayes (Gaussian) and Naive Bayes (Multinomial) are at lower scores.

7.1 <u>Future Scope</u>:

As part of future scope, we wanted to connect directly with GMAIL SMTP server and read the incoming emails. As part of this scope, the email is tokenized and will be performing all NLP techniques and will find the frequencies of the words that are present in the email and will format that in to a row like structure as part of dataset. Then we will subject that new row to the SVM linear algorithm to classify the target label. Additionally, we will add the new email to the dataset directly to increase the corpus base and make sure the models are trained continuously on the incoming data and perform better and accurate day by day.

References:

[1]https://www.cloudflare.com/learning/email-security/what-is email/#:~:text=Electronic%20mail%2C%20commonly%20shortened%20to,that%20are%20sent%20and%20received.
[2]https://www.theguardian.com/technology/shortcuts/2019/nov/26/pointless-emails-theyre-not-just-irritating-they-have-a-massive-carbon-footprint
[3]https://searchengineland.com/10-reasons-why-email-is-more-important-than-ever-376086
[4]https://tech.co/news/email-overload-and bankruptcy#:~:text=Email%20Overload%3A%20Every%20Employee's%20Nightmare&text=33%25%20of%20those%20surveyed%20said,not%20read%20and%20replied%20to
[5] https://www.secretservice.gov/investigation/Preparing-for-a-Cyber-Incident/phishing
[6]https://www.ibm.com/topics/naivebayes#:~:text=The%20Na%C3%AFve%20Bayes%20classifier%20is,a%20given%20class%20or%20category.
[7]https://en.wikipedia.org/wiki/Support_vector_machine
[8]https://en.wikipedia.org/wiki/Decision_tree#:~:text=A%20decision%20tree%20is%20a,only%20contains%20conditional%20control%20statements.
[9]https://support.google.com/mail/answer/3094499?hl=en&co=GENIE.Platform%3DDesktop

[10]https://support.microsoft.com/en-us/office/use-categories-in-outlook-87f27f03-4d9f-48dd-9623-2702692a4480