**INOFRMATICS INSTITUTE OF TECHNOLOGY**

In collaboration with

ROBERT GORDEN UNIVERSITY ABERDEEN

BSc. Artificial Intelligence & Data Science

Level 05

CM 2604

Machine Learning

**Coursework Report**

NADUN SHAMIKA SENARATHNE

IIT ID: 20210488

RGU ID: 2117538

Table of Contents

[Corpus Preparation 4](#_Toc130587600)

[Solution Methodology 4](#_Toc130587601)

[Evaluation Criteria 5](#_Toc130587602)

[Model Evaluation 6](#_Toc130587603)

[Confusion matrix for KNN (with PCA): 7](#_Toc130587604)

[Classification Report for KNN (PCA): 7](#_Toc130587605)

[Confusion matrix for Decision Tree (with PCA): 8](#_Toc130587607)

[Classification Report for Decision Tree (PCA): 8](#_Toc130587608)

[Confusion matrix for KNN (without PCA): 9](#_Toc130587610)

[Classification Report for KNN (without PCA): 9](#_Toc130587611)

[Confusion matrix for Decision Tree (without PCA): 10](#_Toc130587612)

[Classification Report for Decision Tree (without PCA): 10](#_Toc130587613)

[Experimental Results 11](#_Toc130587614)

[Limitations & Further Enhancements 12](#_Toc130587615)

[Limitations: 12](#_Toc130587616)

[Ways to overcome the limitations: 12](#_Toc130587617)

[Future enhancements: 12](#_Toc130587618)

[GitHub project URL: 12](#_Toc130587619)

[Appendix: 13](#_Toc130587620)

[Import dependencies 13](#_Toc130587621)

[Reading the dataset 13](#_Toc130587622)

[Adding headers to the dataset 14](#_Toc130587623)

[Checking for duplicate values 14](#_Toc130587624)

[Dropping duplicate data 15](#_Toc130587625)

[Checking for any Null values 15](#_Toc130587626)

[Getting a feature correlation plot 16](#_Toc130587627)

[Checking for outliers 17](#_Toc130587628)

[Boxplot of capital\_run\_length\_total 18](#_Toc130587629)

[Boxplot of capital\_run\_length\_longest 18](#_Toc130587630)

[Boxplot of capital\_run\_length\_average 19](#_Toc130587632)

[Making all the outliers as Null values from IQR technique 19](#_Toc130587634)

[Boxplot of capital\_run\_length\_total without outliers 20](#_Toc130587635)

[Boxplot of capital\_run\_length\_longest without outliers 20](#_Toc130587636)

[Boxplot of capital\_run\_length\_average without outliers 21](#_Toc130587637)

[Checking for Null values after turning all the outliers as Null values 21](#_Toc130587639)

[Removing all the Null values 22](#_Toc130587642)

[Converting the dataframe into a numpy array 22](#_Toc130587644)

[Separating the features of the dataset as the X variable 22](#_Toc130587646)

[Separating the labels of the dataset as the y variable 23](#_Toc130587648)

[Normalizing Dataset 23](#_Toc130587650)

[Before normalizing 23](#_Toc130587651)

[Normalizing 23](#_Toc130587653)

[After normalizing 23](#_Toc130587654)

[Removing the label colum from the dataframe 24](#_Toc130587656)

[Converting the above dataframe to X variable as features 24](#_Toc130587658)

[Create a PCA instance 24](#_Toc130587660)

[Plot the explained variances 24](#_Toc130587661)

[Save components to a DataFrame 26](#_Toc130587664)

[Performing PCA to the features dataset "X" 26](#_Toc130587666)

[Splitting the dataset 26](#_Toc130587667)

[Decision Tree Classifier Model 27](#_Toc130587669)

[Performing Hyperparameter Tuning 27](#_Toc130587670)

[Passing the above mentioned parameters to model 27](#_Toc130587671)

[Training the model 28](#_Toc130587672)

[Predicting 28](#_Toc130587674)

[Model accuracy 28](#_Toc130587675)

[K Nearest Neighbour Classifier Model 30](#_Toc130587678)

[Performing hyperparameter tuning 30](#_Toc130587679)

[Passing the above mentioned parameters to model 30](#_Toc130587680)

[Training the Model 31](#_Toc130587681)

[Predicting 31](#_Toc130587682)

[Model accuracy 31](#_Toc130587683)

# Corpus Preparation

The UCI website, which contains a huge selection of datasets that may be utilized for any machine learning task, is where the Spambase Data Set was collected from. The last column of the dataset, which has around 4600 data and 58 attributes/features, indicates if the provided email is spam or not. A spam email is indicated by the number 1, and a non-spam email by the number 0. By examining the qualities that show if a certain character or given word is commonly occurring in a letter, the emails were divided into spam and no spam groups.

In terms of the data preprocessing, actions like data cleansing were taken. The processes included examining and eliminating null values, eliminating duplicate values, and eliminating outliers.

K-nearest neighbors and Decision Tree, two machine learning techniques, were used to classify emails as spam or not spam. Following careful management of the data cleaning phase, the dataset was split into train and test halves, with 70% of the data being designated as train data and the remaining portion as test data. It is considerably better to allocate between 20 and 30% because the data can then have a larger percentage of training. Yet it varies according on the circumstances. There is no such thing as an ideal split percentage for data training.

# Solution Methodology

KNN and Decision Tree algorithms were used to classify spam and non-spam emails. An overview of the two algorithms that were employed may be found below.

**KNN**, or the k-nearest neighbor algorithm, is a non-parametric model that locates the k data points that are closest to a test point given a scenario, and then classifies the test point based on the majority class of the k-neighbors.

**Decision Tree**,The decision tree method divides the dataset into groups that resemble tree topologies. According to the values of the inputs, the trees will have a stopping criterion.

Using the Spambase dataset, which initially divides the data into train and test sets, the KNN and Decision Tree models were employed. Then, features were employed as inputs, and labels were used to produce results. The PCA technique was used to complete the classification challenge since it enhances the models' effectiveness and performance. To select the ideal parameters and enhance the models' output, methods like grid search were used. Moreover, feature engineering was carried out since it finds the dataset's most instructive characteristics, develops new features, and captures the intricate interactions between labels.

# Evaluation Criteria

The metrics employed for the spam, no spam email categorization scenario, including as accuracy, precision, recall, and F1-scores, are described below along with the justifications for their use.

* **Accuracy**: In the case described, it is essential to have a high accuracy rate because it is crucial to properly differentiate and identify spam and non-spam emails in order to prevent missing crucial communications.
* **Precision & Recall**: For the classification instance, precision and recall metrics were utilized because they both represent the model's capacity to properly identify the number of true positives and false positives separately.
* **F1-Score**: This score was utilized to assess the model's performance since it balances precision and recall and provides a thorough picture of the models' overall effectiveness.

**Accuracy**

Shape

Description automatically generated with low confidence

**Precision**

Text

Description automatically generated with low confidence

**Recall**

**A picture containing text

Description automatically generated**

**F1-Score**

Table

Description automatically generated

# Model Evaluation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | KNN with PCA | Decision Tree with PCA | KNN without PCA | Decision Tree without PCA |
| Train and Test split proportion | Train – 75%,  Test – 25% | Train – 75%,  Test – 25% | Train – 70%,  Test – 30% | Train – 70%,  Test – 30% |
| Test Accuracy | 90.71% | 86.31 % | 90.13 % | 91.58 % |
| Precision (weighted) | 91% | 86% | 90% | 92% |
| Recall (weighted) | 91% | 86% | 90% | 92% |
| F1-Score (weighted) | 91% | 86% | 90% | 92% |

## Confusion matrix for KNN (with PCA):

Chart, treemap chart

Description automatically generated

## Classification Report for KNN (PCA):

## Calendar Description automatically generated with medium confidence

## 

## Confusion matrix for Decision Tree (with PCA):

Chart, treemap chart

Description automatically generated

## Classification Report for Decision Tree (PCA):

## 

## Confusion matrix for KNN (without PCA):

Chart, treemap chart

Description automatically generated

## Classification Report for KNN (without PCA):

## 

Calendar

Description automatically generated

## Confusion matrix for Decision Tree (without PCA):

Chart, treemap chart

Description automatically generated

## 

## Classification Report for Decision Tree (without PCA):

## 

Calendar

Description automatically generated

Based on the metrices that were utilized, both models gave good scores of accuracies. When PCA was applied, KNN, however, outperformed Decision Tree. By having a walkthrough classification report it can be argued that a high precision can be obtained if the amount of false positives are taken into consideration and high recall can be obtained if the false negatives were taken into consideration.

# Experimental Results

Using the K-Nearest and Decision tree algorithms, an experiment was done to evaluate and categorize the results of spam and no spam emails. More than 4800 emails made up the Spambase dataset, which was split into 25% of the data for testing and 75% of the data for training.

In K-Nearest methods, the number of k components was set automatically to determine the Euclidian distance between each data point from the hyperparameter tuning by passing its value. To reduce the problems associated with overfitting, the decision tree's tree depth was also passed the parameters associated with it.

To improve the performance of the two models, various metrics were applied to the training data. According to the findings, KNN had an accuracy rate of 90.71% after PCA was applied, whereas decision tree had an accuracy rate of 86.31 %.

Overall, it was found that KNN was more effective than decision tree when PCA was used, but decision tree was more effective when PCA was not used.

# Limitations & Further Enhancements

## Limitations:

* As the dataset used to train the model does not contain any real-world data, data bias may have an impact on the model's performance.
* A decision tree has a potential of being overfitted, especially if it is too complicated, the data is noisy, or the training process was improper.

## Ways to overcome the limitations:

* Utilizing a more representative dataset to provide results that are more exact and accurate.
* Techniques like pruning can be used for decision tree classification to address the overfitting issue.

## Future enhancements:

* Testing out various methods, such as Naive Bayes and Support Vector Machines (SVM), and comparing their performance to that of Decision Tree and K-Nearest Neighbors algorithms.
* Using the same models to various tasks, like sentiment analysis and topic classification, and evaluating the outcomes.

# GitHub project URL:

The whole code that was used to categorize spam and non-spam email using KNN and Decision Tree classifiers is available at the GitHub link below.

GitHub Logos and Usage · GitHub

[github.com/Nadun999](https://github.com/Nadun999/spam-email-classification-using-KNN-DT)

# Appendix:

## Import dependencies

import pandas as pd

import numpy as np

import re

from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

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

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

import seaborn as sn

import matplotlib.pyplot as plt

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

from sklearn.model\_selection import GridSearchCV

from sklearn.decomposition import PCA

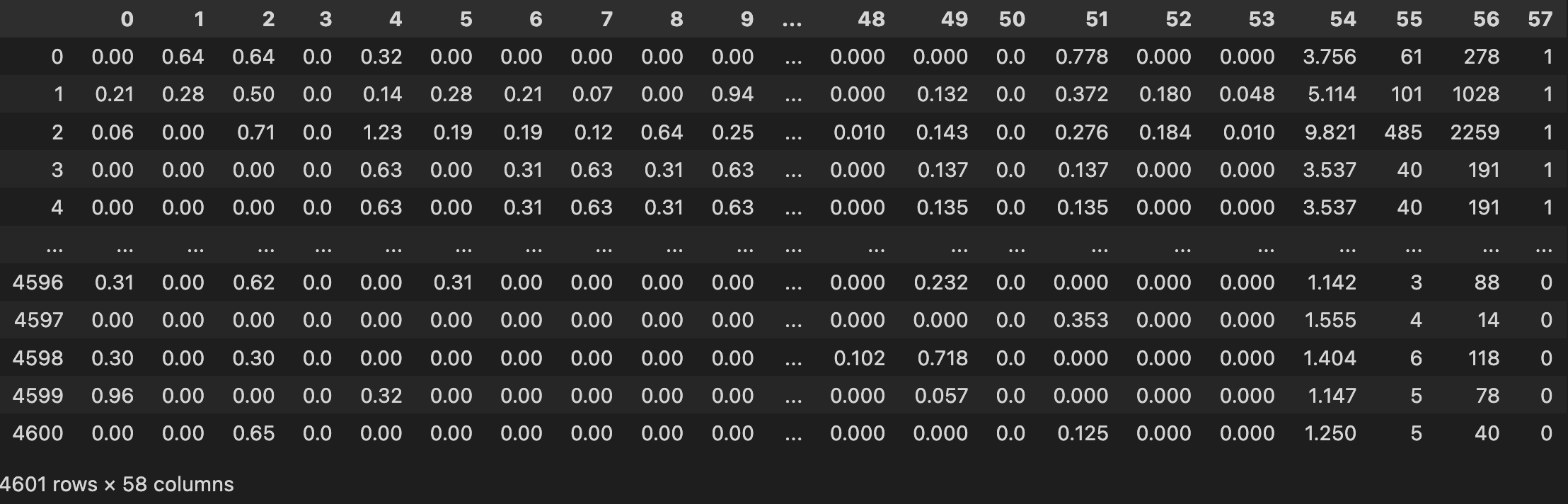
## Reading the dataset

# Reading in the spambase data from a CSV file and storing it in a pandas dataframe

# The header parameter is set to None since the data does not contain column headers

data = pd.read\_csv('spambase.data', header=None)

data



## Adding headers to the dataset

# Open and read the file containing the column names for the spam dataset

with open('./spambase.names') as spam:

text = spam.read()

# Use regular expression to find the column names from the text

# The column names are enclosed in a newline character followed by one or more alphanumeric characters or underscores,

# then optionally followed by non-alphanumeric characters and a colon

labels = re.findall(r'\n(\w\*\_?\W?):', text)

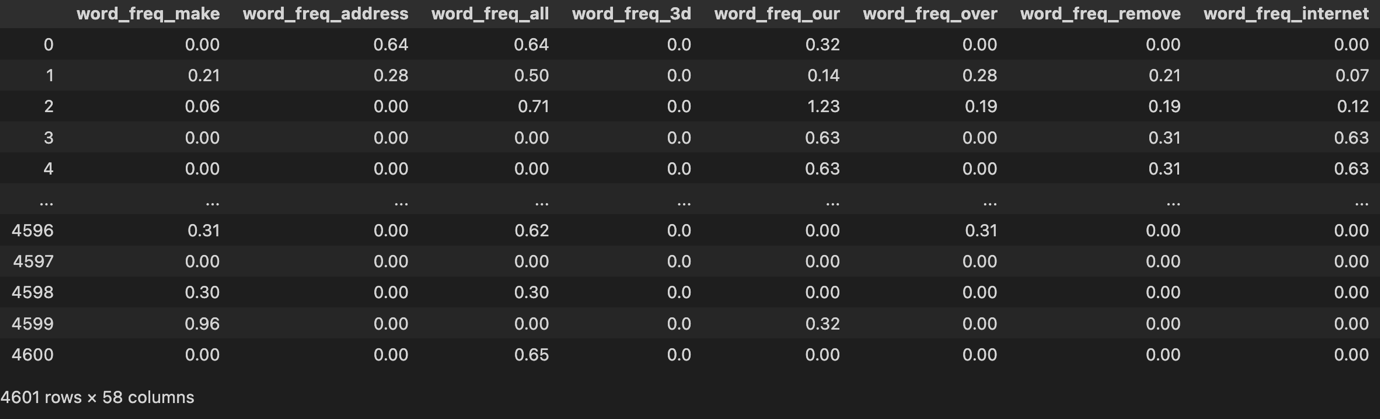
# Read the spam dataset file into a pandas dataframe

# Specify the header as None because the column names are included in the data file

# Specify the names of the columns as the labels found earlier plus an additional 'spam' column

df = pd.read\_csv('./spambase.data', header=None, names=labels + ['spam'])

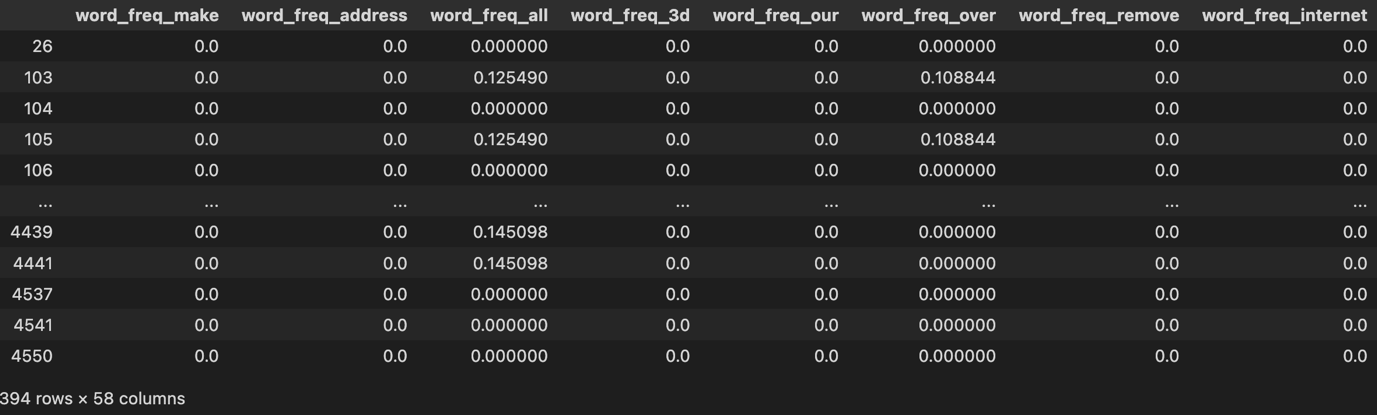
df



## Checking for duplicate values

duplicate = df[df.duplicated()]

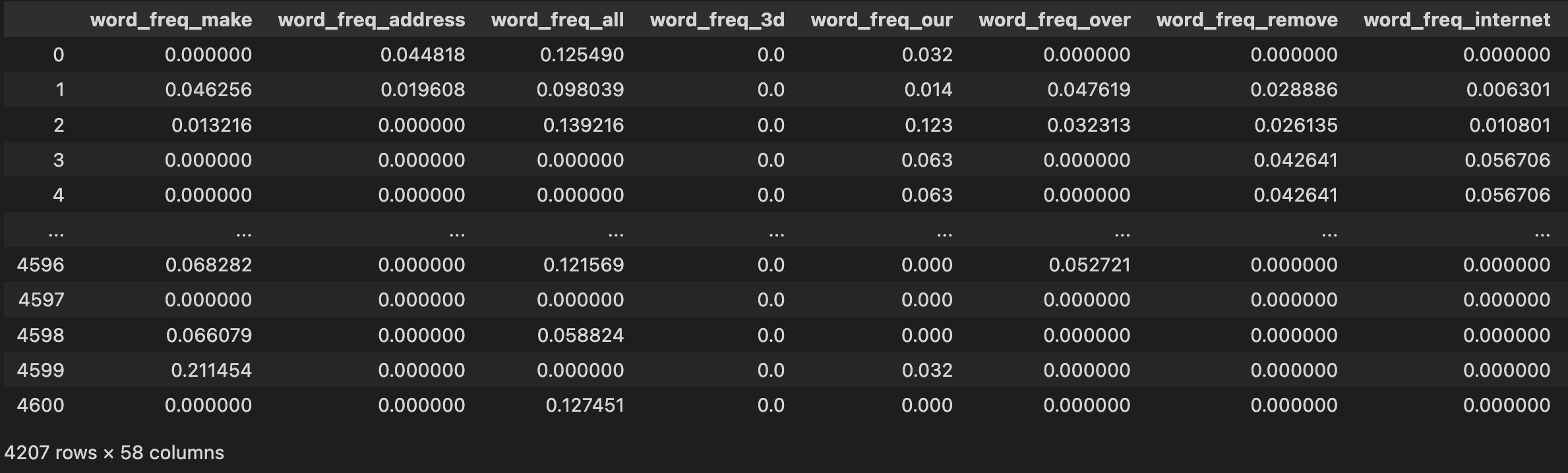
duplicate



## Dropping duplicate data

df = df.drop\_duplicates()

df



## Checking for any Null values

df.isnull().values.any()

Graphical user interface, text

Description automatically generated

df.isnull().sum()

A picture containing text

Description automatically generated

## Getting a feature correlation plot

# Calculate the correlation matrix of the features in the dataframe

correlation = df.corr()

# The correlation matrix shows the pairwise correlations between all the features in the dataframe

# A correlation of 1 indicates a perfect positive correlation (when one feature increases, so does the other)

# A correlation of -1 indicates a perfect negative correlation (when one feature increases, the other decreases)

# A correlation of 0 indicates no correlation between the features

# Values between -1 and 1 indicate varying degrees of correlation

# Visualize the correlation matrix using a heatmap

# The 'cmap' argument specifies the color map to use for the heatmap

sn.heatmap(correlation, cmap="BuPu")

# Add a title to the heatmap

plt.title("Feature Correlation")

# Ensure tight layout of the heatmap in the figure

plt.tight\_layout()

# Show the heatmap

plt.show()

Graphical user interface

Description automatically generated with medium confidence

## Checking for outliers

# Create a boxplot of the dataframe

# The 'title' argument specifies the title of the plot

# The 'rot' argument specifies the rotation angle of the x-axis labels

df.plot.box(title='Boxplot of Spam vs Non-Spam Email', rot=90)

# Show the boxplot

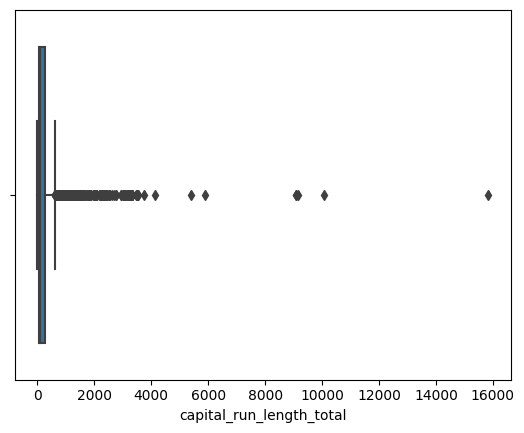
plt.show()

A picture containing graphical user interface

Description automatically generated

### Boxplot of capital\_run\_length\_total

sn.boxplot(x = df['capital\_run\_length\_total'])



### Boxplot of capital\_run\_length\_longest

sn.boxplot(x = df['capital\_run\_length\_longest'])

**Chart

Description automatically generated**

### Boxplot of capital\_run\_length\_average

sn.boxplot(x = df['capital\_run\_length\_average'])

**Chart, scatter chart

Description automatically generated**

### Making all the outliers as Null values from IQR technique

# For each of the three specified features, calculate the 75th and 25th percentiles

# using the numpy percentile function

for x in ['capital\_run\_length\_total','capital\_run\_length\_longest','capital\_run\_length\_average']:

q75, q25 = np.percentile(df.loc[:, x], [75, 25])

# Calculate the interquartile range (IQR) of the feature

intr\_qr = q75 - q25

# Calculate the maximum and minimum values allowed for the feature

max\_val = q75 + (1.5 \* intr\_qr)

min\_val = q25 - (1.5 \* intr\_qr)

# Replace any values below the minimum or above the maximum with NaN

df.loc[df[x] < min\_val, x] = np.nan

df.loc[df[x] > max\_val, x] = np.nan

### Boxplot of capital\_run\_length\_total without outliers

sn.boxplot(x = df['capital\_run\_length\_total'])

Chart, box and whisker chart

Description automatically generated

### Boxplot of capital\_run\_length\_longest without outliers

sn.boxplot(x = df['capital\_run\_length\_longest'])

Chart, box and whisker chart

Description automatically generated

### Boxplot of capital\_run\_length\_average without outliers

sn.boxplot(x = df['capital\_run\_length\_average'])

**Chart, box and whisker chart

Description automatically generated**

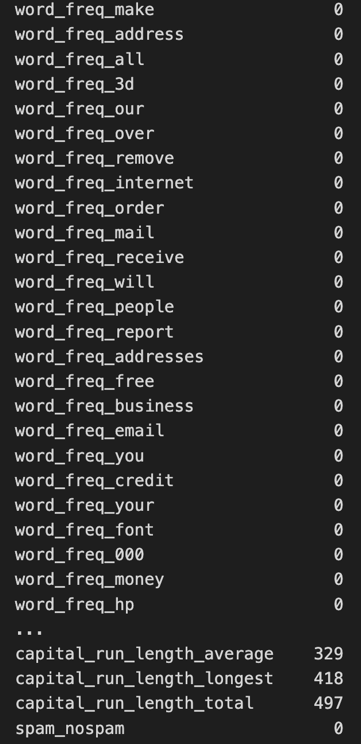
### Checking for Null values after turning all the outliers as Null values

df.isnull().values.any()

**Text

Description automatically generated with low confidence**

df.isnull().sum()

****

### Removing all the Null values

# Drop all rows that contain NaN values from the dataframe

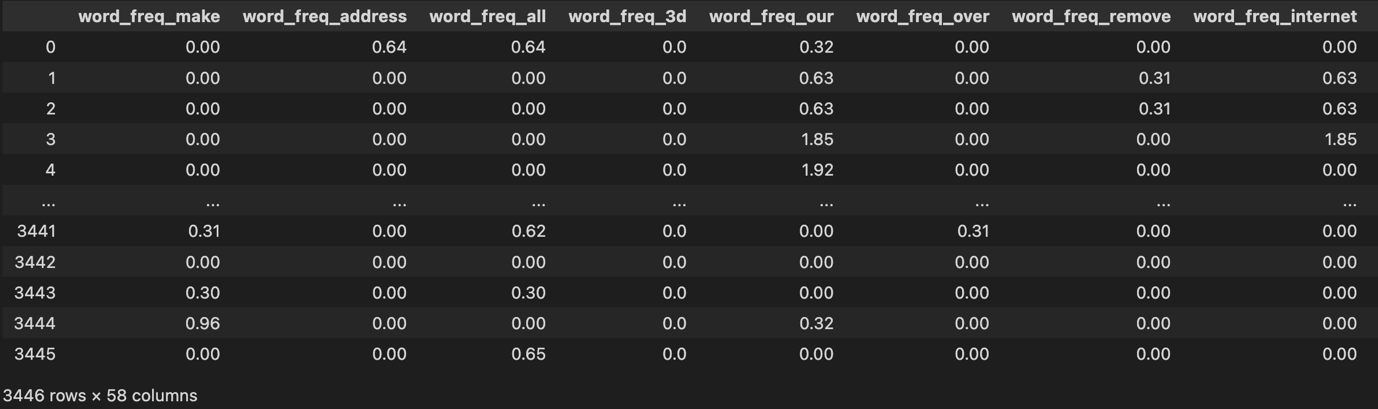
df2 = df.dropna()

# Reset the index of the dataframe after dropping NaN values

df2 = df.dropna().reset\_index(drop=True)

df = df2

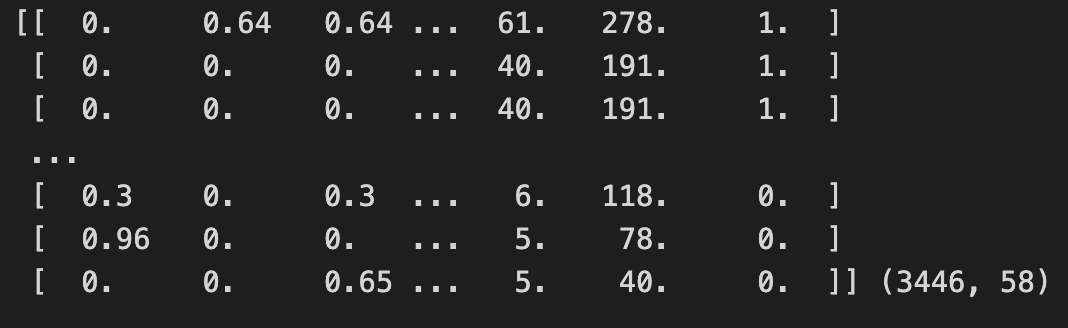
df

****

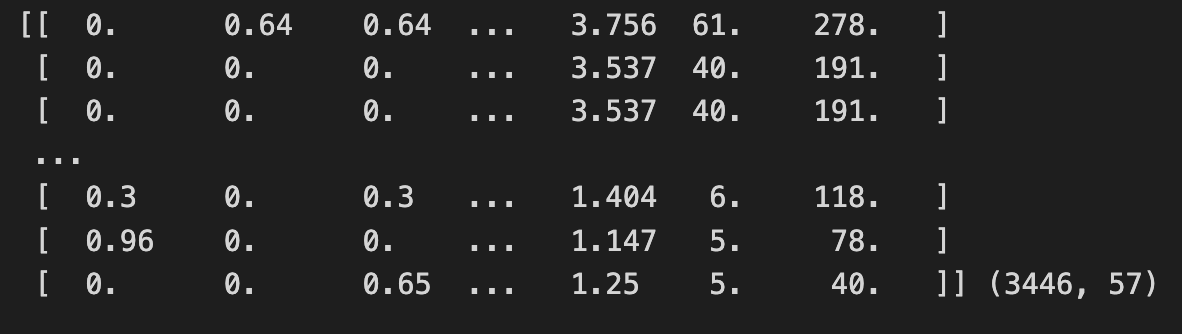
## Converting the dataframe into a numpy array

dataset = df.to\_numpy()

print(dataset,dataset.shape)

****

## Separating the features of the dataset as the X variable

****

## Separating the labels of the dataset as the y variable

y = dataset[:,57]

print(y,y.shape)

**Graphical user interface, text, application

Description automatically generated**

## Normalizing Dataset

### Before normalizing

df.describe()

**Text

Description automatically generated with low confidence**

### Normalizing

# Instantiate a MinMaxScaler object with a feature range of (0, 1)

scaler = preprocessing.MinMaxScaler(feature\_range=(0, 1))

# Use the fit\_transform method to scale the original dataframe and store the result in a new variable

normalized\_scale = scaler.fit\_transform(df)

# Create a new dataframe using the scaled data, with the same indices and column names as the original dataframe

df\_scale = pd.DataFrame(normalized\_scale, index=df.index, columns=df.columns)

# Overwrite the original dataframe with the scaled data

df = df\_scale

### After normalizing

df.describe()

**A picture containing text

Description automatically generated**

## Removing the label colum from the dataframe

df.drop('spam\_nospam', axis=1, inplace=True)

df

**A picture containing text

Description automatically generated**

## Converting the above dataframe to X variable as features

X = df.to\_numpy()

print(X,X.shape)

**Text

Description automatically generated**

## Create a PCA instance

# Instantiate a PCA object with n\_components = 45 as when taking 90% varience, 45

components

# make most of the effect on the final result

pca = PCA(n\_components=45)

# Used to plot the explained\_variance\_ratio\_ histogram

# Use the fit\_transform method to fit the PCA model to the data and transform the data

pca\_Components = pca.fit\_transform(df)

# Print the shape of the transformed data

print(pca\_Components.shape)

## Plot the explained variances

# Calculate the cumulative sum of explained variance ratios using the cumsum

function from numpy

cumulative\_variances = np.cumsum(pca.explained\_variance\_ratio\_)

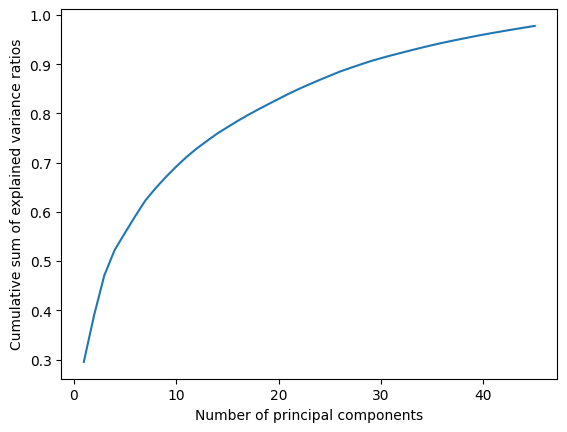
# Plot the cumulative sum of explained variance ratios using matplotlib

plt.plot(range(1, len(cumulative\_variances) + 1), cumulative\_variances)

plt.xlabel("Number of principal components")

plt.ylabel("Cumulative sum of explained variance ratios")

plt.show()

****

# Define a list of features to be used in the plot

features = range(pca.n\_components\_)

# Plot the explained variances for each principal component using the bar function from matplotlib

plt.bar(features, pca.explained\_variance\_ratio\_, color=('purple','red','yellow'))

# Add labels to the x-axis and y-axis using the xlabel and ylabel functions

plt.xlabel('PCA features')

plt.ylabel('variance %')

# Set the tick labels on the x-axis to the feature numbers using the xticks function

plt.xticks(features)

# Print the list of features

print(features)

**Chart

Description automatically generated**

## Save components to a DataFrame

PCA\_components = pd.DataFrame(pca\_Components)

PCA\_components

**Graphical user interface, text

Description automatically generated**

## Performing PCA to the features dataset "X"

X = pca.fit\_transform(X)

## Splitting the dataset

# Split the data and target into training and testing sets using train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30)

# Print the training data and its shape

print("\ntrain data :\n", X\_train, X\_train.shape)

# Print the testing data and its shape

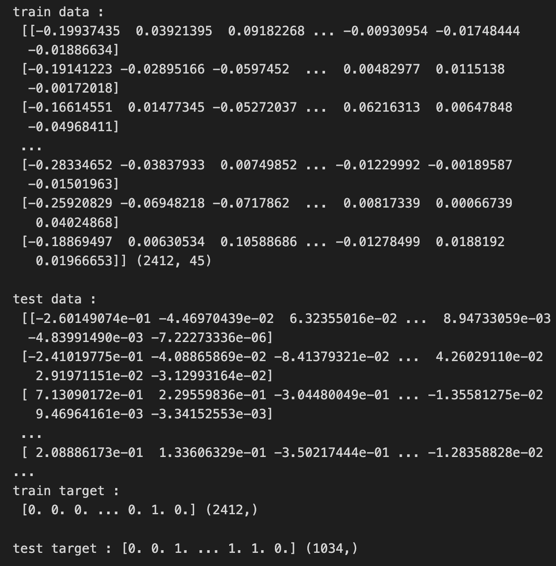
print("\ntest data :\n", X\_test, X\_test.shape)

# Print the training target and its shape

print("\ntrain target :\n", y\_train, y\_train.shape)

# Print the testing target and its shape

print("\ntest target :", y\_test, y\_test.shape)

****

## Decision Tree Classifier Model

# Create an instance of the DecisionTreeClassifier class

model\_dt = DecisionTreeClassifier()

## Performing Hyperparameter Tuning

# Define the hyperparameter grid to search over

param\_grid = {

"max\_depth": [2, 4, 6, 8, 10],

"min\_samples\_split": [2, 4, 6, 8, 10],

"min\_samples\_leaf": [1, 2, 3, 4, 5]

}

# Create an instance of the GridSearchCV class with the decision tree model and hyperparameter grid

grid\_search = GridSearchCV(model\_dt, param\_grid, cv=5)

# Fit the GridSearchCV instance to the training data

grid\_search.fit(X\_train, y\_train)

# Print the best hyperparameters and corresponding score found by GridSearchCV

print("Best hyperparameters: ", grid\_search.best\_params\_)

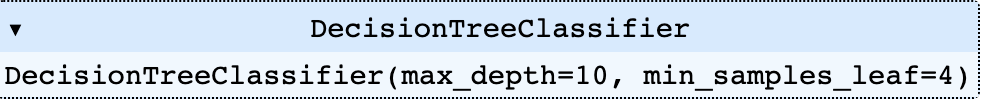
print("Best score: ", grid\_search.best\_score\_)

### Passing the above mentioned parameters to model

model\_dt = grid\_search.best\_estimator\_

## Training the model

model\_dt.fit(X\_train,y\_train)

****

## Predicting

y\_predict\_dt = model\_dt.predict(X\_test)

## Model accuracy

# Model Validation Accuracy

accuracy = accuracy\_score(y\_test,y\_predict\_dt)

print("accuracy : ",accuracy)

# Model Confusion Matrix

conf\_mat\_dt = confusion\_matrix(y\_test, y\_predict\_dt)

print("\nconfusion matrix : \n",conf\_mat\_dt)

# Model Classification Report

clf\_report = classification\_report(y\_test, y\_predict\_dt)

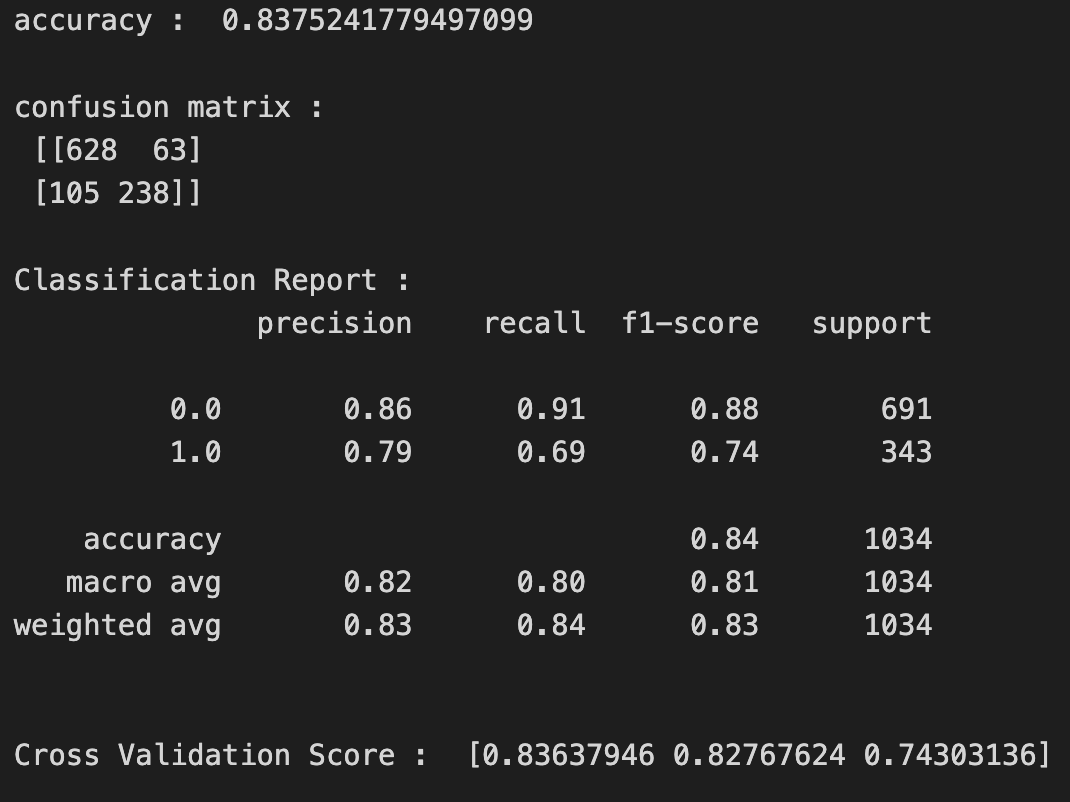
print("\nClassification Report : ")

print(clf\_report)

# Model Cross Validation Score

score = cross\_val\_score(model\_dt, X, y, cv=3)

print("\nCross Validation Score : ",score)

****

# Generate the confusion matrix for the model

# The confusion\_matrix function from scikit-learn is used to calculate the confusion matrix.

# The labels parameter is set to model\_dt.classes\_ to ensure that the labels in the confusion matrix

# match the classes in the model.

conf\_mat\_dt = confusion\_matrix(y\_test, y\_predict\_dt,labels=model\_dt.classes\_)

# calculates the group names, group counts, and group percentages for each cell in the confusion matrix.

# These values are used to create the annotations for each cell in the matrix.

group\_names = ['True No Spam','False Spam',

'False No Spam','True Spam']

group\_counts = ["{0:0.0f}".format(value) for value in

conf\_mat\_dt.flatten()]

group\_percentages = ["{0:.2%}".format(value) for value in

conf\_mat\_dt.flatten()/np.sum(conf\_mat\_dt)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in

zip(group\_names,group\_counts,group\_percentages)]

labels = np.asarray(labels).reshape(2,2)

# the sns.heatmap function from the seaborn library is used to plot the confusion matrix with the annotations.

# The yticklabels and xticklabels parameters are set to model\_dt.classes\_ to ensure that the labels on the y and x axes

# match the classes in the model. The title, xlabel, and ylabel parameters are used to set the title and labels for the

# plot.

ax = sn.heatmap(conf\_mat\_dt, annot=labels,yticklabels=model\_dt.classes\_,xticklabels=model\_dt.classes\_, fmt='', cmap='YlGn')

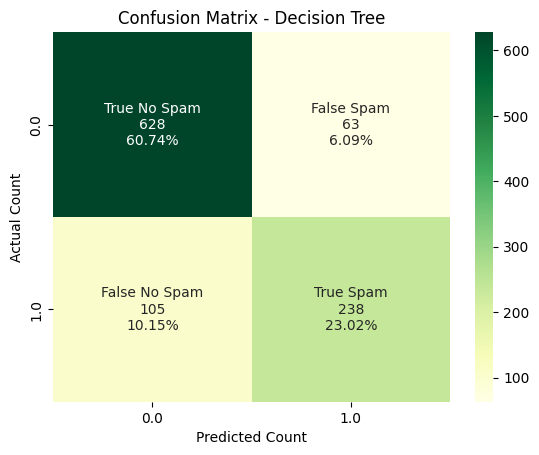
ax.set(

title='Confusion Matrix - Decision Tree',

xlabel='Predicted Count',

ylabel='Actual Count')

ax.plot

****

## K Nearest Neighbour Classifier Model

model\_Knn = KNeighborsClassifier()

## Performing hyperparameter tuning

# Define the parameter grid for tuning the hyperparameters

param\_grid\_K = {'n\_neighbors': [3, 5, 7, 9, 11], 'weights': ['uniform', 'distance']}

# Create a GridSearchCV object to tune the hyperparameters using cross-validation

grid\_search\_K = GridSearchCV(model\_Knn, param\_grid\_K, cv=5)

# Fit the GridSearchCV instance to the training data

grid\_search\_K.fit(X\_train, y\_train)

# Print the best parameters and the best score

print("Best parameters: ", grid\_search\_K.best\_params\_)

print("Best score: ", grid\_search\_K.best\_score\_)

### Passing the above mentioned parameters to model

model\_Knn = grid\_search\_K.best\_estimator\_

## Training the Model

model\_Knn.fit(X\_train,y\_train)

## Predicting

y\_predict\_Knn = model\_Knn.predict(X\_test)

## Model accuracy

# Model Validation Accuracy

accuracy = accuracy\_score(y\_test,y\_predict\_Knn)

print("accuracy : ",accuracy)

# Model Confusion Matrix

conf\_mat\_knn = confusion\_matrix(y\_test, y\_predict\_Knn)

print("\nconfusion matrix : \n",conf\_mat\_knn)

# Model Classification Report

clf\_report = classification\_report(y\_test, y\_predict\_Knn)

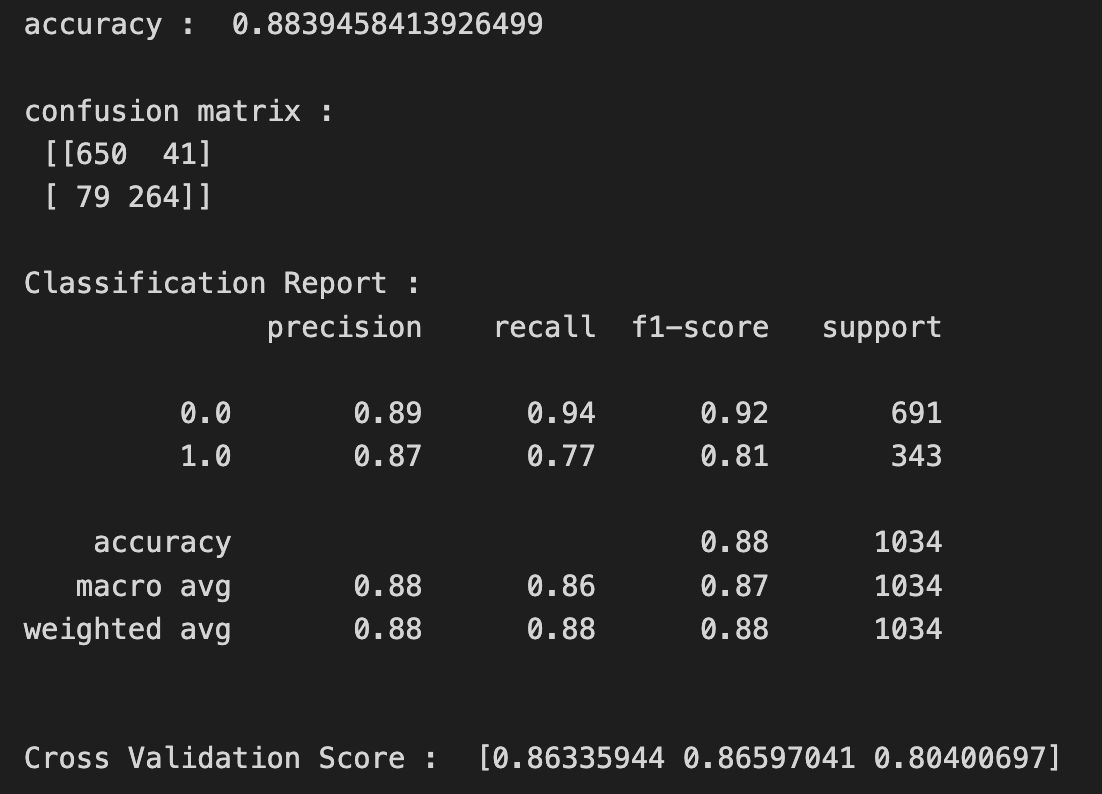
print("\nClassification Report : ")

print(clf\_report)

# Model Cross Validation Score

score = cross\_val\_score(model\_Knn, X, y, cv=3)

print("\nCross Validation Score : ",score)

****

# Generate the confusion matrix for the model

# The confusion\_matrix function from scikit-learn is used to calculate the confusion matrix.

# The labels parameter is set to model\_dt.classes\_ to ensure that the labels in the confusion matrix

# match the classes in the model.

conf\_mat\_knn = confusion\_matrix(y\_test, y\_predict\_Knn,labels=model\_Knn.classes\_)

# calculates the group names, group counts, and group percentages for each cell in the confusion matrix.

# These values are used to create the annotations for each cell in the matrix.

group\_names = ['True No Spam','False Spam',

'False No Spam','True Spam']

group\_counts = ["{0:0.0f}".format(value) for value in

conf\_mat\_knn.flatten()]

group\_percentages = ["{0:.2%}".format(value) for value in

conf\_mat\_knn.flatten()/np.sum(conf\_mat\_knn)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in

zip(group\_names,group\_counts,group\_percentages)]

labels = np.asarray(labels).reshape(2,2)

# the sns.heatmap function from the seaborn library is used to plot the confusion matrix with the annotations.

# The yticklabels and xticklabels parameters are set to model\_dt.classes\_ to ensure that the labels on the y and x axes

# match the classes in the model. The title, xlabel, and ylabel parameters are used to set the title and labels for the

# plot.

ax = sn.heatmap(conf\_mat\_knn, annot=labels,yticklabels=model\_Knn.classes\_,xticklabels=model\_Knn.classes\_, fmt='', cmap='YlGn')

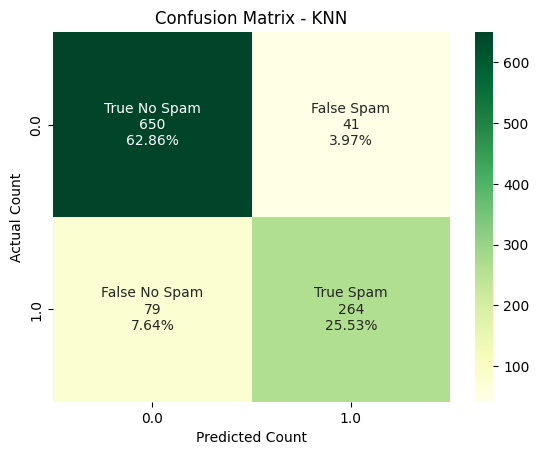
ax.set(

title='Confusion Matrix - KNN',

xlabel='Predicted Count',

ylabel='Actual Count')

ax.plot

****