INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

ROBERT GORDON UNIVERSITY ABERDEEN

Level: Level 05
Module Code: CM2604
Module: Machine Learning
Module Leader: Mr. Prasan Yapa

Due Data: 26th of March 2023

Machine Learning Coursework Report

Vinsuka Jeewandara
IIT ID – 20200471
RGU ID- 2118952

Contents

Git repository

https://github.com/Vinsuka/Machine-learning-CW.git

Data frame Preparation

I used following techniques to prepare the data frame by using the “spambase.data” and "spambase.names" files, which contains a dataset of emails labeled as spam or not spam, along with a list of features. The function reads the data file into a Pandas DataFrame after first extracting the feature names from the "spambase.names" file. This adds column names to the feature names with "spam" label.

```
# Extract feature names from spambase.names file
with open("spambase.names", "r") as f:
    feature_names = []
    for line in f:
        match = re.search(r"^(\w.*):\s*(.*)", line)
        if match:
            feature_names.append(match.group(1))

# Read spambase.data file into DataFrame
df = pd.read_csv("spambase.data", header=None, names=feature_names+["spam"])

df.to_csv("spamDataset.csv", index=False)
```

Pre – processing Methods

- **Data Cleaning Methods**

  - Check for null values and remove them.

Code Snippet

```
# Check for null values
print(df.isnull().sum())
# Remove null values
df = df.dropna()
index=False)
```

OutPut

```
word_freq_make        0

word_freq_address     0
```

word_freq_all	0
word_freq_3d	0
word_freq_our	0
word_freq_over	0
word_freq_remove	0
word_freq_internet	0
word_freq_order	0
word_freq_mail	0
word_freq_receive	0
word_freq_will	0
word_freq_people	0
word_freq_report	0
word_freq_addresses	0
word_freq_free	0
word_freq_business	0
word_freq_email	0
word_freq_you	0
word_freq_credit	0
word_freq_your	0
word_freq_font	0
word_freq_000	0
word_freq_money	0
word_freq_hp	0
word_freq_hpl	0
word_freq_george	0

word_freq_650	0
word_freq_lab	0
word_freq_labs	0
word_freq_telnet	0
word_freq_857	0
word_freq_data	0
word_freq_415	0
word_freq_85	0
word_freq_technology	0
word_freq_1999	0
word_freq_parts	0
word_freq_pm	0
word_freq_direct	0
word_freq_cs	0
word_freq_meeting	0
word_freq_original	0
word_freq_project	0
word_freq_re	0
word_freq_edu	0
word_freq_table	0
word_freq_conference	0
char_freq_;	0
char_freq_(	0
char_freq_[	0
char_freq_!	0

```
char_freq_$                0
char_freq_#                0
capital_run_length_average    0
capital_run_length_longest    0
capital_run_length_total      0
spam                       0
dtype: int64
```

- Check for duplicates and remove them.

Code Snippet

```
# Check for null values
print(df.isnull().sum())
# Remove null values
df = df.dropna()
index=False)
```

Output

```
391
```

- **Remove Outliers.**

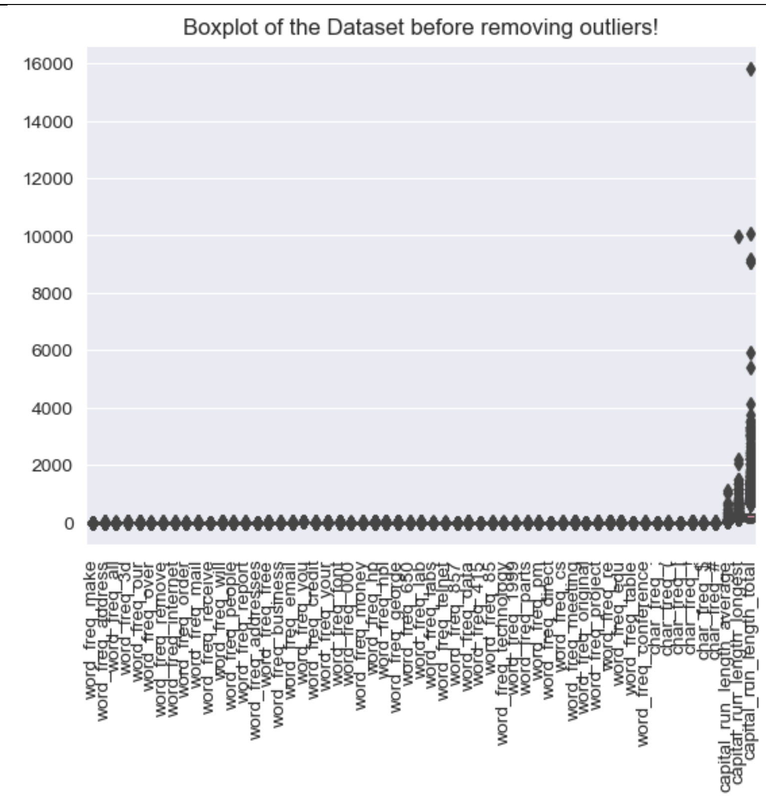
The analysis of the model can be significantly impacted by outliers, which could result in incorrect conclusions. To guarantee the data is reliable and impartial, it is crucial to find and delete outliers during the preprocessing stage.

The outliers in "capital run length total" that have values higher than 3900 have been removed in the code below, and the dataset has then been shown using a boxplot. This gives a summary of the distribution of the remaining data and helps in visually verifying that the outliers have been properly excluded.

Check for outliers

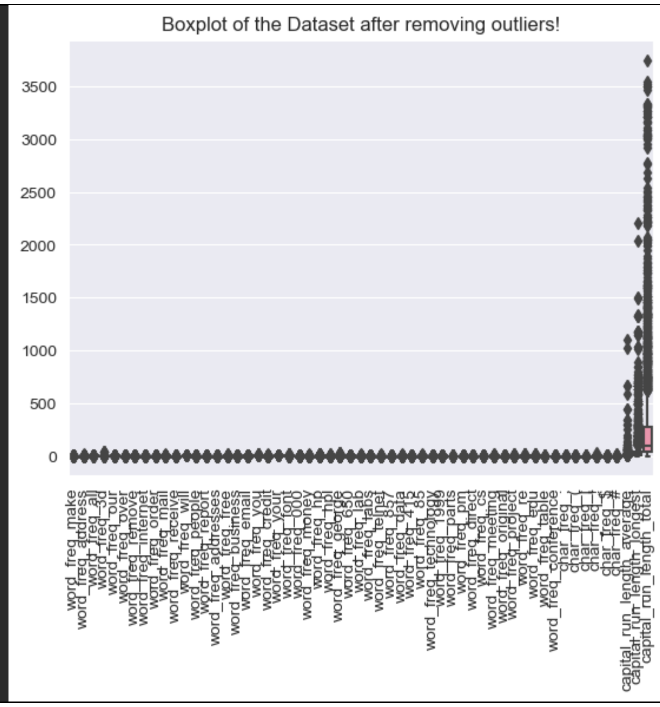
```
sns.boxplot(data=df.iloc[:, :-1])
plt.xticks(rotation=90)
plt.title("Boxplot of the Dataset before removing outliers!")
plt.show()
```

Visualization



Remove Outliers

```
df = df.drop(df[df["capital_run_length_total"] >3900].index)

sns.boxplot(data=df.iloc[:, :-1])
plt.xticks(rotation=90)
plt.title("Boxplot of the Dataset after removing outliers!")
plt.show()
```



- **Feature Scaling.**

To reduce any bias toward a certain feature, feature scaling includes changing the scales or ranges of the features in a dataset to be similar. StandardScaler, which scales the data to have a mean of 0 and a standard deviation of 1.

I've scaled the data using the StandardScaler method from the scikit-learn library.

Code Snippet

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X = scaler.fit_transform(X)
X[0:5]
```

Output

```
array([[-3.47583077e-01,  1.15967179e+00,  6.74942169e-01,
        -4.66817919e-02, -8.42620199e-03, -3.49870479e-01,
        -2.96023880e-01, -2.63168004e-01, -3.25435783e-01,
        -3.78192844e-01, -3.07377727e-01,  8.38420890e-02,
        -3.15477414e-01, -1.76533973e-01, -1.84998051e-01,
         8.25304648e-02, -3.25871334e-01,  2.03307367e+00,
         1.18593781e-01, -1.68212826e-01,  1.29616193e-01,
        -1.21972061e-01, -2.85908569e-01, -2.10423943e-01,
        -3.42773936e-01, -3.08821921e-01, -2.08552542e-01,
        -2.40404288e-01, -1.70754281e-01, -2.36358394e-01,
        -1.64262705e-01, -1.49513446e-01, -1.79968386e-01,
```

```
-1.51727686e-01, -2.03674412e-01, -2.57300927e-01,
-3.38394947e-01, -6.16248362e-02, -1.87794353e-01,
-1.91970202e-01, -1.30041904e-01, -1.78504327e-01,
-2.10574708e-01, -1.32103408e-01, -3.06374403e-01,
-2.04031550e-01, -7.30743472e-02, -1.16463593e-01,
-1.58701502e-01, -6.14076182e-01, -1.64106436e-01,
 5.88894355e-01, -3.17204491e-01, -1.04242569e-01,
-4.75090797e-02,  9.17262377e-02,  5.43208973e-03],
[ 3.52054823e-01,  3.67788654e-01,  4.03664897e-01,
-4.66817919e-02, -2.69946191e-01,  6.63780372e-01,
 2.32168198e-01, -9.26765459e-02, -3.25435783e-01,
 1.05254239e+00,  8.32174639e-01,  2.53720155e-01,
 1.78475770e+00,  4.30240104e-01,  3.92628686e-01,
-1.42993371e-01, -1.73026911e-01,  1.68033041e-01,
 9.89060806e-01, -1.68212826e-01,  6.77481056e-01,
-1.21972061e-01,  9.38283373e-01,  7.84959480e-01,
-3.42773936e-01, -3.08821921e-01, -2.08552542e-01,
-2.40404288e-01, -1.70754281e-01, -2.36358394e-01,
-1.64262705e-01, -1.49513446e-01, -1.79968386e-01,
-1.51727686e-01, -2.03674412e-01, -2.57300927e-01,
-1.73617458e-01, -6.16248362e-02, -1.87794353e-01,
-1.91970202e-01, -1.30041904e-01, -1.78504327e-01,
-2.10574708e-01, -1.32103408e-01, -3.06374403e-01,
-2.04031550e-01, -7.30743472e-02, -1.16463593e-01,
-1.58701502e-01, -4.25935218e-02, -1.64106436e-01,
 1.07708679e-01,  4.38004358e-01,  6.04082864e-03,
-6.32815956e-03,  4.07435499e-01,  1.56897495e+00],
[-1.47686534e-01, -2.48120451e-01,  8.10580805e-01,
-4.66817919e-02,  1.31370263e+00,  3.37964027e-01,
 1.81864190e-01,  2.91030669e-02,  1.94153811e+00,
 2.32184594e-03,  1.75466941e+00, -1.31336794e-01,
 7.22582996e-02, -1.76533973e-01,  7.03533617e+00,
-2.43226186e-01, -1.94861829e-01,  1.55296420e+00,
-2.03592066e-01,  4.64209080e-01, -2.61715852e-01,
-1.21972061e-01,  3.01656272e+00, -7.15332331e-02,
-3.42773936e-01, -3.08821921e-01, -2.08552542e-01,
-2.40404288e-01, -1.70754281e-01, -2.36358394e-01,
-1.64262705e-01, -1.49513446e-01, -1.79968386e-01,
-1.51727686e-01, -2.03674412e-01, -2.57300927e-01,
-3.38394947e-01, -6.16248362e-02, -1.87794353e-01,
-4.33877215e-03, -1.30041904e-01, -1.78504327e-01,
 3.06545005e-01, -1.32103408e-01, -2.49051552e-01,
-1.39446023e-01, -7.30743472e-02, -1.16463593e-01,
-1.18977440e-01,  5.03003323e-03, -1.64106436e-01,
-6.06921485e-03,  4.54786777e-01, -8.12668609e-02,
 1.36410125e-01,  3.43824440e+00,  4.13526996e+00],
[-3.47583077e-01, -2.48120451e-01, -5.65182503e-01,
-4.66817919e-02,  4.41969335e-01, -3.49870479e-01,
 4.83688234e-01,  1.27125512e+00,  7.72629698e-01,
 5.80704175e-01,  1.37481862e+00, -2.89889656e-01,
 6.86173179e-01, -1.76533973e-01, -1.84998051e-01,
 7.00013629e-02, -3.25871334e-01, -3.49007926e-01,
 8.25141691e-01, -1.68212826e-01, -4.35641205e-01,
-1.21972061e-01, -2.85908569e-01, -2.10423943e-01,
-3.42773936e-01, -3.08821921e-01, -2.08552542e-01,
-2.40404288e-01, -1.70754281e-01, -2.36358394e-01,
```

```
       -1.64262705e-01, -1.49513446e-01, -1.79968386e-01,
       -1.51727686e-01, -2.03674412e-01, -2.57300927e-01,
       -3.38394947e-01, -6.16248362e-02, -1.87794353e-01,
       -1.91970202e-01, -1.30041904e-01, -1.78504327e-01,
       -2.10574708e-01, -1.32103408e-01, -3.06374403e-01,
       -2.04031550e-01, -7.30743472e-02, -1.16463593e-01,
       -1.58701502e-01, -2.09464513e-02, -1.64106436e-01,
       -1.70810124e-01, -3.17204491e-01, -1.04242569e-01,
       -5.41501854e-02, -7.40211242e-02, -1.75938882e-01],
      [-3.47583077e-01, -2.48120451e-01, -5.65182503e-01,
       -4.66817919e-02,  4.41969335e-01, -3.49870479e-01,
        4.83688234e-01,  1.27125512e+00,  7.72629698e-01,
        5.80704175e-01,  1.37481862e+00, -2.89889656e-01,
        6.86173179e-01, -1.76533973e-01, -1.84998051e-01,
        7.00013629e-02, -3.25871334e-01, -3.49007926e-01,
        8.25141691e-01, -1.68212826e-01, -4.35641205e-01,
       -1.21972061e-01, -2.85908569e-01, -2.10423943e-01,
       -3.42773936e-01, -3.08821921e-01, -2.08552542e-01,
       -2.40404288e-01, -1.70754281e-01, -2.36358394e-01,
       -1.64262705e-01, -1.49513446e-01, -1.79968386e-01,
       -1.51727686e-01, -2.03674412e-01, -2.57300927e-01,
       -3.38394947e-01, -6.16248362e-02, -1.87794353e-01,
       -1.91970202e-01, -1.30041904e-01, -1.78504327e-01,
       -2.10574708e-01, -1.32103408e-01, -3.06374403e-01,
       -2.04031550e-01, -7.30743472e-02, -1.16463593e-01,
       -1.58701502e-01, -2.96052795e-02, -1.64106436e-01,
       -1.73180496e-01, -3.17204491e-01, -1.04242569e-01,
       -5.41501854e-02, -7.40211242e-02, -1.75938882e-01]])
```

- **Dimensionality reduction technique.**

For reducing the amount of features in a dataset while keeping the most crucial data, dimensionality reduction is applied. Principal Component Analysis (PCA), which converts the original features into a set of new features known as principal components that represent the most significant variance in the data.

I've used PCA in the given code to make the dataset's dimensions smaller.

I'm using the PCA transformation on the scaled dataset and setting the number of principal components to 44. A new variable called ScaledDataSet_PCA is subsequently created and contains the modified dataset. Finally, the explained variance ratio is visualized.

Code

```
from sklearn.decomposition import PCA
# Perform PCA with specified number of components
pca = PCA(n_components=44)
new_dataset = pca.fit_transform(ScaledDataSet)

from sklearn.decomposition import PCA

# Perform PCA with specified number of components
pca = PCA(n_components=44)
new_dataset = pca.fit_transform(ScaledDataSet)
```

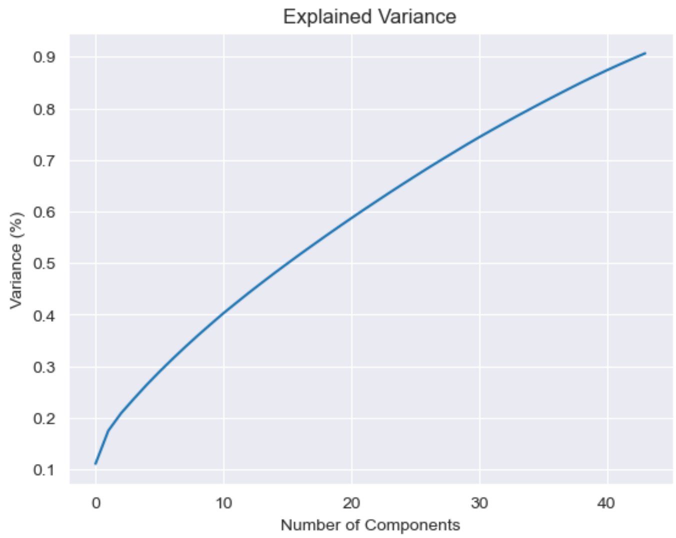
```
# Create a new DataFrame with the transformed data
ScaledDataSet_PCA = pd.DataFrame(data = new_dataset
                              , columns = ['PC1',
'PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10','PC11','PC12','PC13','PC14','PC15','PC16','PC17','PC18','PC
19','PC20',
                                          'PC21',
'PC22','PC23','PC24','PC25','PC26','PC27','PC28','PC29','PC30','PC31','PC32','PC33','PC34','PC35','PC36','PC37','
PC38','PC39','PC40',
                                          'PC41','PC42','PC43','PC44'])

# Transform the original data with PCA
ScaledDataSet_PCA.head()

# Print the original and transformed data shapes
data_pca = pca.transform(ScaledDataSet)
print("Original shape:", ScaledDataSet.shape)
print("Transformed shape:", ScaledDataSet_PCA.shape)

# Print the explained variance of each principal component
pca.explained_variance_

# Print the explained variance ratio of each principal component
pca.explained_variance_ratio_

# Plot the cumulative explained variance ratio
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Explained')
plt.title('Explained Variance Ratio')
plt.show()
```

pca.explained_variance_ **Output**

```
array([6.42730758, 3.7197125 , 1.96786478, 1.62375054, 1.5677352 ,
       1.47858515, 1.40549641, 1.35627027, 1.30909804, 1.26005669,
       1.24107183, 1.16349334, 1.15072934, 1.11349128, 1.09602481,
       1.06691036, 1.05397054, 1.02861619, 1.02105627, 1.00302291,
       0.99015325, 0.97354415, 0.95723365, 0.95228803, 0.93100655,
       0.91889095, 0.90962758, 0.88414311, 0.86965958, 0.86832542,
       0.84210998, 0.81752656, 0.80470911, 0.79169553, 0.77024081,
       0.7639432 , 0.7488211 , 0.73692167, 0.72805423, 0.69883864,
       0.67979339, 0.65503778, 0.63892205, 0.60852642])
```

pca.explained_variance_ratio_ **Output**

```
array([0.11078928, 0.06411771, 0.03392063, 0.02798904, 0.02702348,
       0.02548678, 0.02422693, 0.02337841, 0.02256528, 0.02171995,
       0.0213927 , 0.02005546, 0.01983544, 0.01919356, 0.01889248,
       0.01839063, 0.01816758, 0.01773054, 0.01760023, 0.01728938,
       0.01706754, 0.01678125, 0.0165001 , 0.01641485, 0.01604802,
       0.01583918, 0.0156795 , 0.01524022, 0.01499056, 0.01496757,
       0.01451568, 0.01409193, 0.01387099, 0.01364667, 0.01327685,
       0.0131683 , 0.01290764, 0.01270252, 0.01254967, 0.01204607,
```

0.01171779, 0.01129107, 0.01101328, 0.01048934])
Visualization


K Nearest Neighbors (KNN) Classification

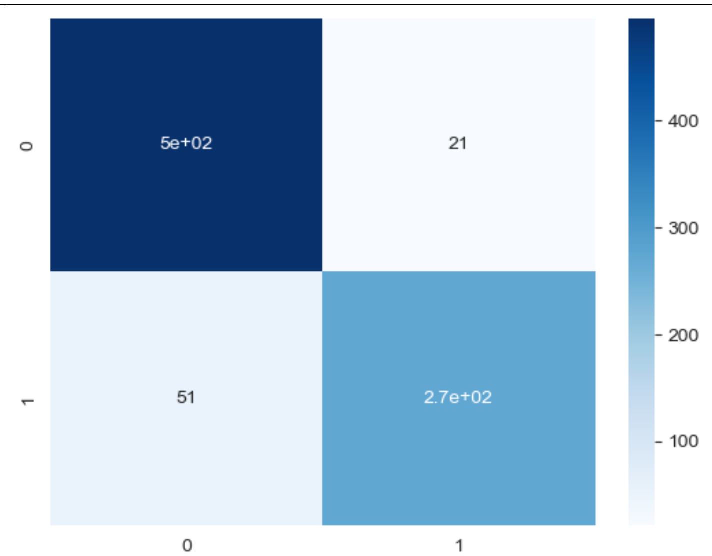
- **KNN Implementation**

Code

```
import pandas as pd
import seaborn as sns
%matplotlib inline

# Load dataset
dataSet = pd.read_csv('spamDataset.csv')
dataSet.describe()

# Split features and target variable
X = dataSet.iloc[:,0:57]
X.head()
y = dataSet.iloc[:, -1]
y.head()

# Apply StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
X[0:5]

# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Print the shapes of the train and test sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
```

```
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Train a K-Nearest Neighbors classifier with k=5
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train,y_train)

# Make predictions on the test set
pred = model.predict(X_test)
pred[0:5]

# Show the actual labels for the first 5 examples in the test set
y_test[0:5]

# Compute the accuracy of the classifier on the test set
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, pred)
accuracy
```

- **Confusion Matrix**



- **Classification Report**

```
              precision    recall  f1-score   support

           0       0.92      0.93      0.92       519
           1       0.89      0.86      0.88       322

    accuracy                           0.91       841
   macro avg       0.90      0.90      0.90       841
weighted avg       0.91      0.91      0.91       841
```

Decision Tree Classification

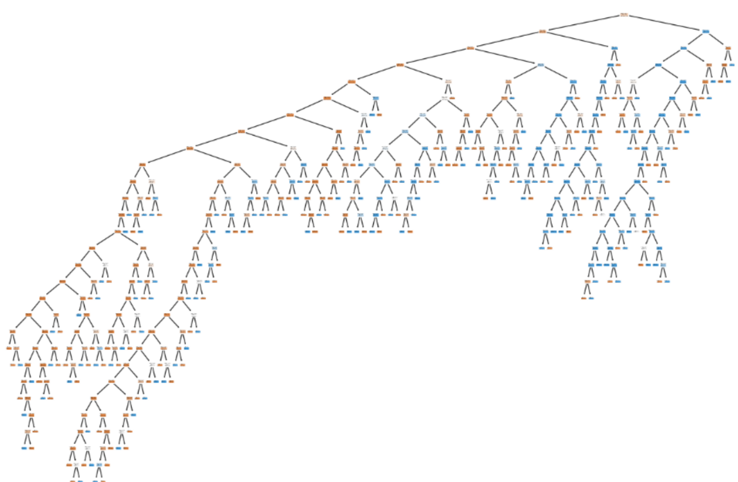
- **Decision Tree Implementation**

Code

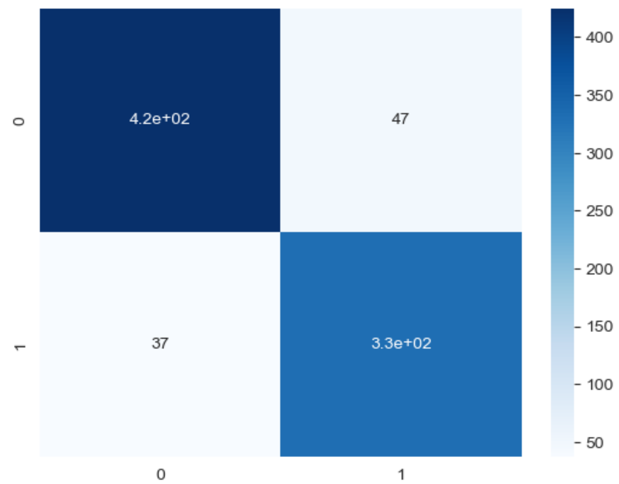
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Load dataset and show summary statistics
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

DTdata = pd.read_csv('spamDataset.csv')
DTdata.describe()

# Split features and target variable
X = DTdata.iloc[:,0:57]
X.head()
y = DTdata.iloc[:, -1]
y.head()

# Standardize features using StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
X[0:5]

# Split the dataset into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print the shapes of the train and test sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
```

```
print("y_test shape:", y_test.shape)

# Decision tree model and train it
dtModel = DecisionTreeClassifier(random_state=42)
dtModel.fit(X_train, y_train)

#Predict Values
pred = dtModel.predict(X_test)
pred

#Original Values
y_test

#Accuracy for testing data

from sklearn.metrics import accuracy_score
accuracy_score(y_test, pred)
```

Decision Tree Graph



- **Confusion Matrix**



- **Classification Report**

```
              precision    recall  f1-score   support

           0       0.92      0.90      0.91       472
           1       0.88      0.90      0.89       369

    accuracy                           0.90       841
   macro avg       0.90      0.90      0.90       841
weighted avg       0.90      0.90      0.90       841
```

Limitation and Future Enhancements

Limitation

- When the decision tree is deep its difficult to find the optimal solution.
- KNN algorithm result is very sensitive to outliers.
- The ML models were generated through a very old dataset.
- Reduction of accuracy with the implementation of PCA

Future Enhancements

- Create the models and compare the algorithm's optimization to other machine learning algorithms.