SCS3201 – Machine Learning and Neural Computing

Assignment 2

B.A.A.D. Bandara

Index Number – 19000091

Table of Contents

1. Random Forrest

Popular machine learning algorithm Random Forest is a part of the supervised learning methodology. It can be applied to Machine Learning issues involving both classification and regression. It is built on the idea of ensemble learning, which is a method of integrating various classifiers to address difficult issues and enhance model performance.

As the name implies, this approach creates a tree-like model for its output with decision nodes and leaf nodes. Here, a choice is represented by a leaf node, and the decision nodes are organized in the order of two or more branches. A decision tree is used to manage categorical and continuous data. It is a simple and effective decision-making diagram. As one can see, utilizing trees is a straightforward and useful way to understand how decisions are made and to represent the results of algorithms. The main advantage of a decision tree is that it can quickly adjust to the dataset. Viewing and analyzing the finished model can be done using a well-organized "tree" diagram. However, because the random forest approach creates a large number of different decision trees and then averages these forecasts, it is significantly less likely to be affected by outliers.

The forest has nearly identical hyper parameters to those of a decision tree. Its ensemble approach of decision trees is constructed from data that has been randomly partitioned. This complete group can be compared to a forest with a different independent random sample for each tree. When there are enough trees present, the random forest method can become too slow and useless for real-time prediction. The random forest approach, in contrast, bases its conclusions on traits and observations that are randomly selected and built on numerous decision trees. However, because each decision tree in random forests is only built using a small number of predictors, the resulting decision trees are frequently not correlated. As a result, the random forest algorithm model is unlikely to outperform the dataset. Decision trees, as previously said, generally overwrite the training data, which makes it more probable that they will match the dataset's "noise" than the model that generated it.

The main difference between the random forest algorithm and decision trees is that decision trees are graphs that display every potential outcome of a choice using a branching technique. Comparatively, the random forest method yields a set of decision trees that work in line with the outcome. Decision trees frequently encounter the problem of over fitting if allowed to expand

unrestrictedly. The over fitting problem is solved when utilizing random forests because the final result is based on average or majority rating and is built using independent random samples of data. A single decision tree processes information more quickly than a random forest. A decision tree will generate a set of rules for performing prediction when given a data set with features as input. From data it randomly selects, random forest creates a decision tree and averages the outcomes. It does not use formulae.

The random forest algorithm's steps are as follows:

Step-1: Select random K data points from the training set.

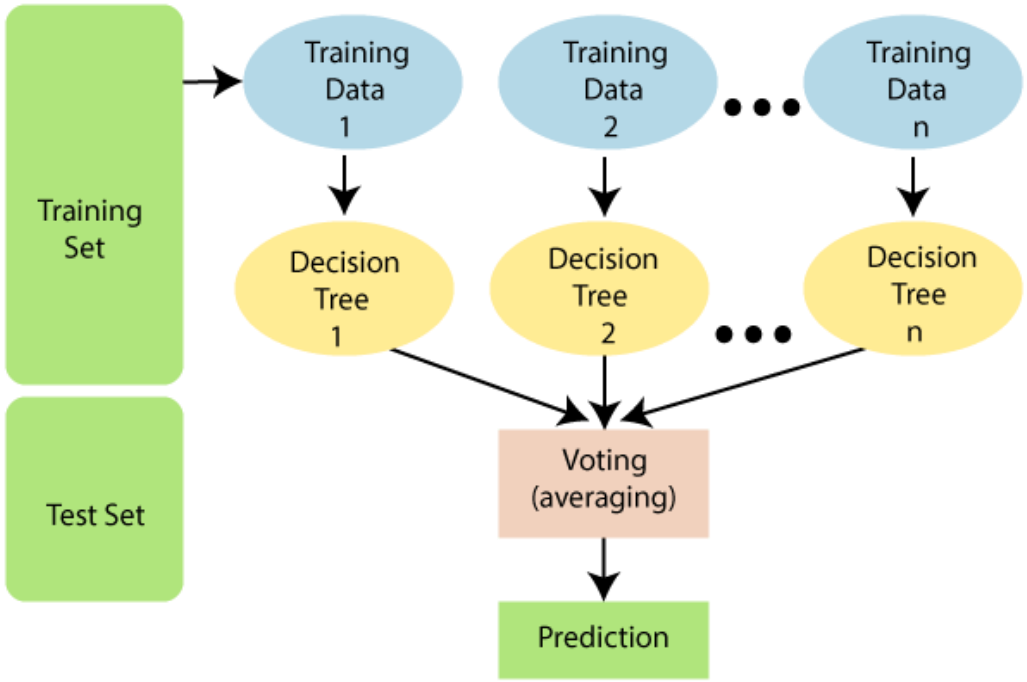
Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

The below diagram explains the working of the Random Forest algorithm:



Important Random Forrest Features

1. **Diversity**: Since each tree is unique, not all characteristics, variables, or features are taken into account when creating a particular tree.
2. **Immune to the dimensionality curse**: Because no tree takes into account every feature, the feature space is condensed.
3. **Parallelization**: Using various data and attributes, each tree is individually generated. This implies that we can create random forests by using the CPU to its fullest extent.
4. **Train-Test split**: In a random forest we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.
5. **Stability:** Because the outcome is based on majority vote or averaging, there is stability.

2. Introduction to Data Set

Data set: Car Evaluation Data Set

https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

Number of instances 1728

Number of attributes 6

This data set can be used to predict the Car Class (Evaluate) using following 6 input attributes.

Input attributes

- Buying price: vhigh, high, med, low.
- Maintenance price: vhigh, high, med, low.
- Number of doors: 2, 3, 4, 5more.
- Person capacity: 2, 4, more.
- The size of luggage boot: small, med, big.
- Estimated safety of the car: low, med, high.

Output variables

- Car class – qualitative: unacc, acc, good, vgood

3. Data Preprocessing

After analyzing the data set, I found that the data set did not have any missing values. For make sure there are no missing values I remove it also according to following

```
#19000091
#SCS3201 - Machine Learning and Neural Computing
#Assignment 2

import pandas as pd

#read the data file
car_evaluvation = pd.read_csv("car.csv")

#print the data head
print(car_evaluvation.head())

#if there are any missing values remove them
car_evaluvation = car_evaluvation.dropna()
```

But all the variables attributes were categorical. So, I had to convert them to numerical values by using dictionaries and encoding the values in all the variables as following

```
#data preprocessing
buyingAndmaintDataClassification ={
    'vhigh' : 3,
    'high' : 2,
    'med' : 1,
    'low' : 0
    }

car_evaluvation['buying'] = car_evaluvation['buying'].apply( lambda id : buyingAndmaintDataClassification [id])

car_evaluvation['maint'] = car_evaluvation['maint'].apply( lambda id : buyingAndmaintDataClassification [id])


doorsDataClassification ={
    '2' : 2,
    '3' : 3,
    '4' : 4,
    '5more' : 5
    }

car_evaluvation['doors'] = car_evaluvation['doors'].apply( lambda id : doorsDataClassification [id])


personsDataClassification ={
    '2' : 2,
    '4' : 4,
    'more' : 5
    }

car_evaluvation['persons'] = car_evaluvation['persons'].apply( lambda id : personsDataClassification [id])
```

```
lug_bootDataClassification ={
    'small' : 1,
    'med' : 2,
    'big' : 3
    }

car_evaluvation['lug_boot'] = car_evaluvation['lug_boot'].apply( lambda id : lug_bootDataClassification [id])


safetyDataClassification ={
    'low' : 1,
    'med' : 2,
    'high' : 3
    }

car_evaluvation['safety'] = car_evaluvation['safety'].apply( lambda id : safetyDataClassification [id])


car_classDataClassification ={
    'unacc' : 1,
    'acc' : 2,
    'good' : 3,
    'vgood' : 4
    }

car_evaluvation['car_class'] = car_evaluvation['car_class'].apply( lambda id : car_classDataClassification [id])
```

According to the below screenshot, we can see all the data are in numerical values after preprocessing the data.

```
  buying  maint doors persons lug_boot safety car_class
0  vhigh  vhigh     2       2    small    low     unacc
1  vhigh  vhigh     2       2    small    med     unacc
2  vhigh  vhigh     2       2    small   high     unacc
3  vhigh  vhigh     2       2      med    low     unacc
4  vhigh  vhigh     2       2      med    med     unacc
   buying  maint  doors  persons  lug_boot  safety  car_class
0       3      3      2        2         1       1          1
1       3      3      2        2         1       2          1
2       3      3      2        2         1       3          1
3       3      3      2        2         2       1          1
4       3      3      2        2         2       2          1
```

4. Model Fitting

First all the input attributes set into the X axis. Output (predicting) attribute set into the Y axis.

Then we split the data set random as 75% percent of data as training data and 25% percent of data as testing data

After that trained the model with Random Forrest Classifier with xTrain data and yTrain data

Codes are below about model fitting

```
#print the data head after data preprocessing
print(car_evaluvation.head())

x = car_evaluvation.iloc[:, :6]
y = car_evaluvation.iloc[:,6]

from sklearn.model_selection import train_test_split

xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = 0.25)

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators = 4, bootstrap = True, ccp_alpha = 0.00)

rfc.fit(xTrain , yTrain)
```

5. Evaluation

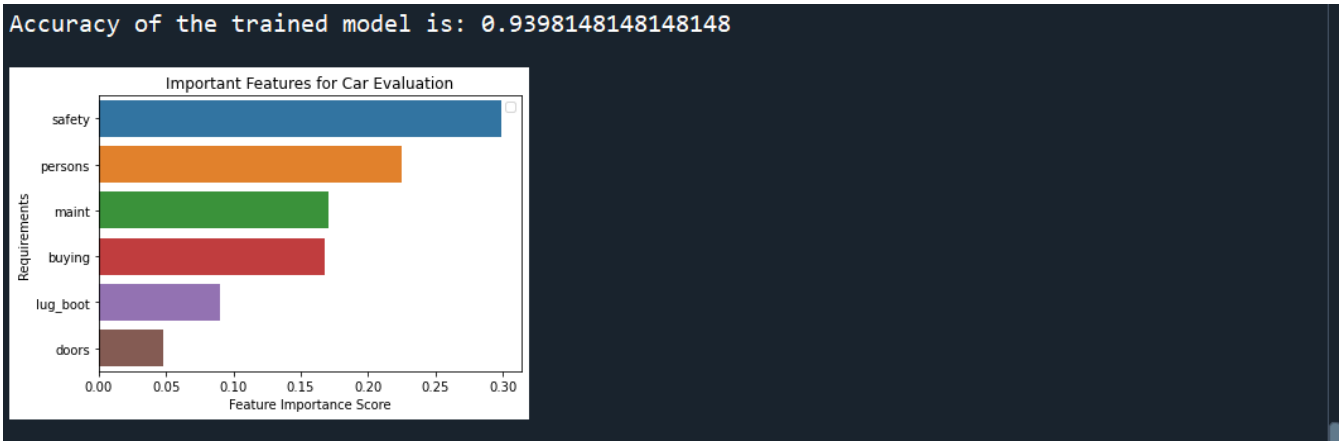
Then predict the results with using trained data model and xTest data.

For evaluate the accuracy of trained model calculate Mean Accuracy by using xTest, yTest and trained data model.

Also get an idea about the data model and the way of Random Forrest classification was done, I mapped the classification group into the bar plot.

Codes and results are below about evaluation.

```
yPrediction = rfc.predict(xTest)

from sklearn import metrics

print("Accuracy of the trained model is:",metrics.accuracy_score(yTest, yPrediction))

featureImportance = pd.Series(rfc.feature_importances_,index=['buying','maint','doors','persons','lug_boot','safety']).sort_va
featureImportance

import matplotlib.pyplot as plt
import seaborn as sns

# Creating a bar plot to see the what requirement are more important to decide the car class evaluation
sns.color_palette("Paired")
sns.barplot(x = featureImportance, y = featureImportance.index)

# Add labels to the graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Requirements')
plt.title("Important Features for Car Evaluation")
plt.legend()
plt.show()
```



According to those, predictions of these trained model which was trained using Random Forrest classification is almost correct. Trained data model's accuracy is more than 93 percent. It means more than lot of predictions can be correct. However we can't guarantee that every prediction is correct. There can be wrong predictions.

According to bar plot called "Important Features for Car Evaluation" we can identify what are the requirements which effect to evaluate the car class. According to the results, "Safety" of the car is most important requirement to evaluate the car class. We can identify other requirement according to their importance using the bar plot. So we can evaluate what is the most important requirement in prediction.

6. Appendix (Code)

```
#19000091
#SCS3201 - Machine Learning and Neural Computing
#Assignment 2
import pandas as pd
#read the data file
car_evaluvation = pd.read_csv("car.csv")
#print the data head
print(car_evaluvation.head())
```

```
#if there are any missing values remove them
car_evaluvation = car_evaluvation.dropna()
#data preprocessing
buyingAndmaintDataClassification = {
    'vhigh' : 3,
    'high' : 2,
    'med' : 1,
    'low' : 0
    }
car_evaluvation['buying'] = car_evaluvation['buying'].apply( lambda id :
buyingAndmaintDataClassification [id])
car_evaluvation['maint'] = car_evaluvation['maint'].apply( lambda id :
buyingAndmaintDataClassification [id])


doorsDataClassification = {
    '2' : 2,
    '3' : 3,
    '4' : 4,
    '5more' : 5
    }
car_evaluvation['doors'] = car_evaluvation['doors'].apply( lambda id : doorsDataClassification
[id])
personsDataClassification = {
    '2' : 2,
    '4' : 4,
    'more' : 5
    }
```

```
car_evaluvation['persons'] = car_evaluvation['persons'].apply( lambda id : personsDataClassification [id])

lug_bootDataClassification = {

    'small' : 1,

    'med' : 2,

    'big' : 3

    }

car_evaluvation['lug_boot'] = car_evaluvation['lug_boot'].apply( lambda id : lug_bootDataClassification [id])


safetyDataClassification = {

    'low' : 1,

    'med' : 2,

    'high' : 3

    }

car_evaluvation['safety'] = car_evaluvation['safety'].apply( lambda id : safetyDataClassification [id])

car_classDataClassification = {

    'unacc' : 1,

    'acc' : 2,

    'good' : 3,

    'vgood' : 4

    }

car_evaluvation['car_class'] = car_evaluvation['car_class'].apply( lambda id : car_classDataClassification [id])

#print the data head after data preprocessing

print(car_evaluvation.head())

x = car_evaluvation.iloc[:, :6]
```

```
y = car_evaluvation.iloc[:,6]

from sklearn.model_selection import train_test_split

xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = 0.25)

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators = 4, bootstrap = True, ccp_alpha = 0.00)

rfc.fit(xTrain , yTrain)

yPrediction = rfc.predict(xTest)

from sklearn import metrics

print("Accuracy of the trained model is:",metrics.accuracy_score(yTest, yPrediction))

featureImportance =
pd.Series(rfc.feature_importances_,index=['buying','maint','doors','persons','lug_boot','safety']).
sort_values(ascending=False)

featureImportance

import matplotlib.pyplot as plt

import seaborn as sns

# Creating a bar plot to see the what requirement are more important to decide the car class
evaluation

sns.color_palette("Paired")

sns.barplot(x = featureImportance, y = featureImportance.index)

# Add labels to the graph

plt.xlabel('Feature Importance Score')

plt.ylabel('Requirements')

plt.title("Important Features for Car Evaluation")

plt.legend()

plt.show()
```