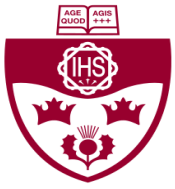


SAINT MARY'S UNIVERSITY SINCE 1802

One University. One World. Yours.

Master of Science in Computing and Data Analytics

Data and Text Mining

MCDA 5580

Assignment 2

Submitted To:

Ms. Trishla Shah

Submitted By:

Hemanchal Joshi (A00433394)

Rishi Karki (A00432524)

Haozhou Wang (A00431268)

TABLE OF CONTENTS

1. Executive Summary

We were given a set of car data from which we were supposed to take a specific attribute dependent variable which we have to use to further analyze the data. For the analysis, we used various classifying techniques in order to achieve maximum accuracy when it comes to our prediction in comparison with the test data. For this, we divided the data into test and train data and trained the train data into a model. We took the trained model and randomly choose various parameters like min-split and max-depth in order to classify the attributes into various categories and draw a conclusion. We divided them into **(70/30) (train/test) for classifying using python** and **(75/25) (train/test) as well as (80/20) (train/test) while we explored the classification techniques in R**. Overall, our target was to build a decision tree / random forest classifier to categorize the data and analyze the accuracy of the proposed training model.

Finally, after a lot of discussions we decided to use two programming languages to solve the issue: the R already has provided us with some powerful tools like rpart, pROC, caret and in Python, we can use the powerful scikit-learn pipeline tools to do it. What really matters is the principle but not the tools, but we also believe that the usage of different tools will help us have a better understanding of the algorithm.

2. Data Summary

The given car dataset can be summarized as below:

Total Records: 1728 rows
Dependent Variable: **shouldBuy where class levels are unacc, acc, good, vgood**
Independent Variables:

Price (high,low,med,vhigh)

Maintenance (high,low,med,vhigh)

Doors (2,3,4,5 more)

Seats (2,4,more)

Storage (big,med,small)

Safety (high,low,med)

From the tutorial , we know that the data is already clean(i.e. No empty or null values) so we can wither transform the string data into numeric or else use the existing dataset in order to further analyze the problem. For Python we transformed the string into numeric data as well as for R. We used OpenRefine to transform the data into numeric data. Even **when we transformed the data using Openrefine there was no change in the acquired result when compared to untransformed data.**

3. Approaches

There were various approaches and methodologies we could follow for classification techniques but according to the guidelines provided we follow the following pipeline :

- Data cleaning(transformation for multiple-class model)
- Build Decision Tree Classifier using R-part
- Calculate the accuracy confusion matrix/recall/RUC to get an intuitive understanding
- Using K-Fold to tune the parameter in the multiple reiterations to get the best parameters
- Using randomForest classifier to build the model (changing different number of trees varying from 400 to 700) , and check which factor influences the result most.

Mainly for each part our team will provide two results implemented by Python and R respectively.

4. Methodology

4.1. Decision Tree

Decision tree is a simple and clear algorithm. Decision tree diagrammatically infers and represents all the possible results and pathways that lead to various decision in

any scenario. Decision trees use tree like graph in order to model decision and their consequences [1].

Here, we can change every 1 variable at one time to get the optimal accuracy .but if we have multiple parameters , it will be very difficult to tell their relative influences(like a blackbox ,and this is why sometimes machine learning and deep learning lacks the explanations although it actually works). A better way to consider all the parameters into consideration is to use the grid search method. But it is beyond the topic of this assignment. We hope we can implement it with later tutorials.

4.2. Random forest and K folds

Random forest algorithm is a machine learning algorithm which works even without the tuning of the parameters as it builds a number of decision trees and thoroughly merge them together to get better results [2].

Similarly K-folds is cross-validation technique which uses AUC by visualizing ROC to determine which model is the best in order to predict the classes. We can use K-folds in Random forest algorithm to analyze the significant behavior of the individual dependent variable.

5. Classification Techniques

5.1. Using Python

From the tutorial , we already know that the data is clean. But we still need to do something to transform our data into the proper format.Our data consists of 7 columns. For the first six columns are regarded as independent variables and for the last variable “shouldBuy”,it is our dependent variable ,and they will be our prediction target.

Our data is not a two-class dataset. For example , for the column price ,there are 4 labels : vhigh, high, med and low. During the process of calculation, the non-

numerical words can’t be recognised by the model , so we need to convert string to an integer. This is a very common request so the scikit-learn has provided a function called LabelEncoder() to help us solve the problem.

```
le = LabelEncoder()
categories_non_numerical = [
    'price', 'maintenance', 'storage', 'safety', 'shouldBuy']
pima[categories_non_numerical] = pima[categories_non_numerical].apply(
    lambda col: le.fit_transform(col))
```

Next we need to split the dataset into a training dataset and test dataset. Sklearn also allows us to do it in one line :

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

The 0.3 means there will be 30% data used as test data and 70% data as training data.

5.1.1 Build model and predict

Now we have the available data , then what we need to do is to build the model and use it to make the prediction. So we import the model from sklearn :

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
```

And call it :

```
clf = clf.fit(X_train, y_train)
y_predict = clf.predict(X_test)
```

So now we have got the **y_predict** as the predicted result. And the **y_test** is used as test data.

The generated decision tree will be like :

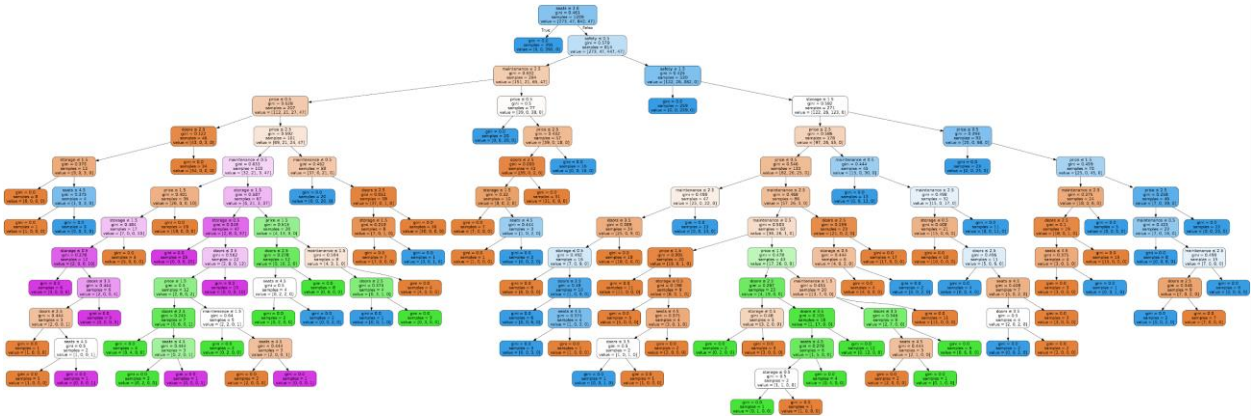


Fig: decision tree

- **Accuracy**

  Accuracy typically means how much the trained model specifies or comes close in accordance with the test or the accurate data. It is typically the total correct predictions with respect to the total number of predictions [3].

  ```
  Accuracy: 0.9691714836223507
  ```

  Finally our accuracy is **96.92%(**with all default parameters**)**

- Recall

  Recall is also known as sensitivity is the term which is calculated over a period of time and is regarded as the total number of instances collected within that time frame.

  ```
  recall_score:  0.9344306015501669
  ```

  Finally our recall value is **93.44%(**with all default parameters**)**

- **Confusion Matrix**

  We compare the test dataset and the prediction dataset. Then get the final confusion matrix.

	acc	vgood	unacc	good
acc	105	1	4	1
vgood	2	19	1	0
unacc	6	0	362	0
good	0	1	0	17

Finally, our confusion matrix is below:

	acc	**vgood**	**unacc**	**good**
acc	**105**	1	4	1
vgood	2	**19**	1	0
unacc	6	0	**362**	0
good	0	1	0	**17**

- **AUC**

  The dataset provided is a multiple-class dataset ,so we need to bianrize the data to calculate the AUC score . Luckily the sklearn also provides the LabelBinarizer to help us do it :

  lb = LabelBinarizer()

  lb.fit(y_test)

  y_test1 = lb.transform(y_test)

  y_pred = lb.transform(y_predict)

  print("AUC score is :",roc_auc_score(y_test1, y_pred, average='macro'))

  ```
  AUC score is : 0.9598727284285885
  ```

  Finally, our AUC score is **95.99%(**with all default parameter**)**

- **ROC**

  ROC here is not designed for multiple-class. So we have two solutions to solve it:

  1. Generate the plot for different labels with the final result. This can be done by R because there is already a specific package to do it.
  2. Turn the final result to two-class to get the whole picture. This can be done by Python.

  So we follow the procure to deal with the data , and successfully convert our dataset into a proper format suitable for ROC curve plot.:

  y_predict[y_predict <= 1]=0

  y_predict[y_predict >= 2]=1

  y_test.replace(1,0,inplace=True)

  y_test.replace(2,1,inplace=True)

y_test.replace(3,1,inplace=True)

And use matplotlib to draw the plots :

fpr, tpr, thresholds = roc_curve(y_test, y_predict)

plt.plot([0, 1], [0, 1], linestyle='--')

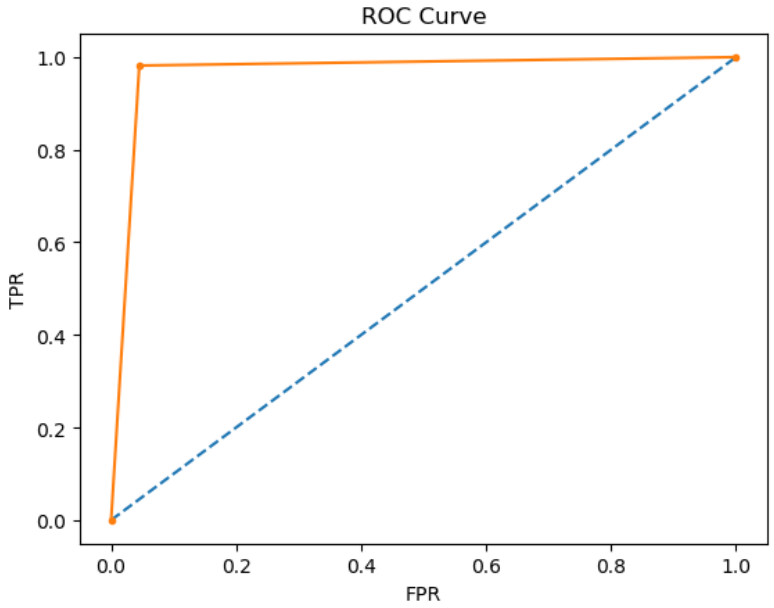


Fig: ROC curve

5.1.2 Tune parameters and K-fold

From the definition below , we know that K-fold can be used to get the best paramater during the iteration.

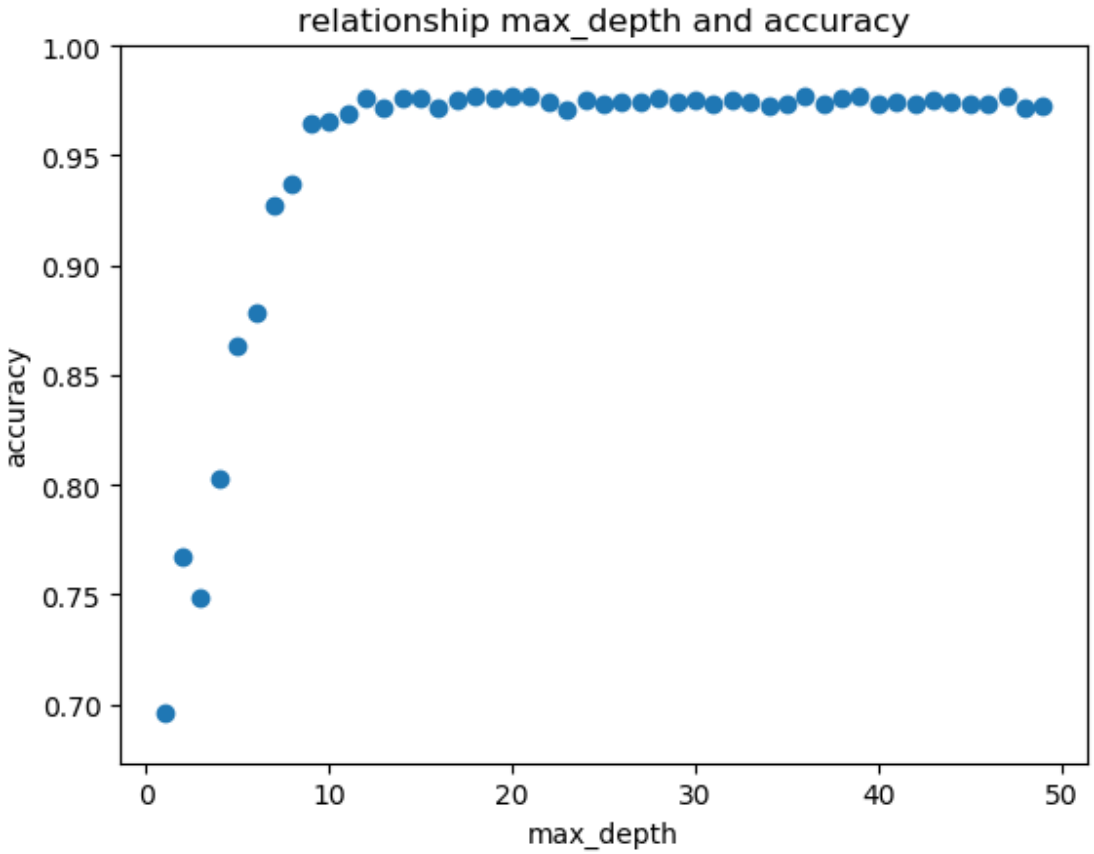
In the following code, we iterate 98 times , and perform the 7-fold cross validation each time with the change of ***min_samples_split*** paramater.

```
for i in range(2,100):
    clf = DecisionTreeClassifier(min_samples_split =i)
     scores = cross_val_score(estimator=clf, X=X_train, y=y_train, cv=7, n_jobs=4)
    depth.append((i,scores.mean()))

plt.scatter([f[0] for f in depth],[f[1] for f in depth])
```

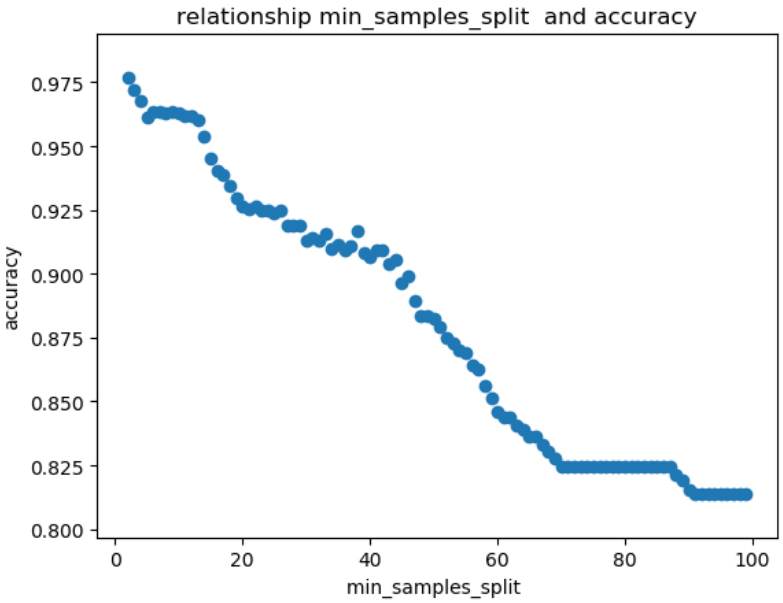
And, we will change ***max_depth/min_split/min_leaf*** parameters in the following paragraph.

- change max_depth
  Max_depth means the depth of the tree.



So it is clear to observe that the accuracy reaches the highest value after the max_depth of 13. And after that , the metrics will not change with the increase of max_dpeth.In other words, any depth greater 13 will make no impact towards the final result.

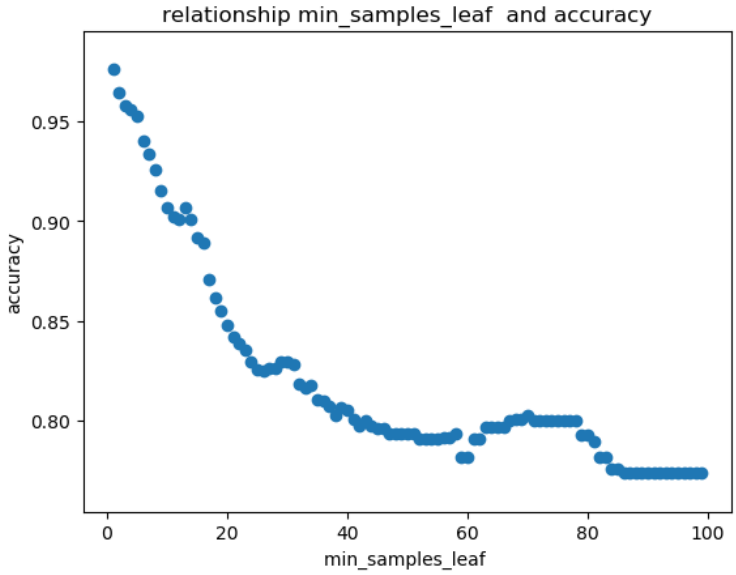
- **change min_split**
  Min_split is actually min_samples_split which is the minimum number of samples required to split an internal node:



We find that with the increase of ***min_samples_split*** , the accuracy goes down continuously. It will be best if we use the default parameter.

- **change min_leaf**

  Min_leaf is actually ***min_samples_leaf*** which means the minimum number of samples required to be at a leaf node.



We observe that the pattern of min_leaf nearly is identical with min_split. The increase of parameter doesn't lead to the positive change of accuracy. Still, the initial value is the optimal one.

- **Later improvement**

  In the examples above, we just change one parameter at one time. But in the real world, different factors will influence each other . So if we want to get a final bunch of optimal parameters, other methods, like gridsearch , will be more appropriate.

  With K-means, we can utilize our dataset(i.e. We can train our models multiple times get the best parameter without worrying about over-fitting problem).

5.1.3 Random Forest Classifier

- **Metrics**

  Random forest is still a classifier , so we will have the same metrics as the decision tree.

- **Iterations**

  In the random forest classifier , we will use ***entropy*** as the criterion. And one very influential factor is the number of trees. To make the best performance , we need to test multiple numbers to get the optimal one.

  Below are the metrics we get when applying the different number of trees.

	400	450	500	***550***	600	650	700
accuracy	97.11%	96.92%	96.92%	***97.30%***	96.92%	96.92%	96.92%
recall	91.94%	91.87%	92.63%	***93.08%***	92.94%	92.78%	92.78%
auc	95.36%	95.30%	95.63%	***95.96%***	95.825	95.78%	95.70%

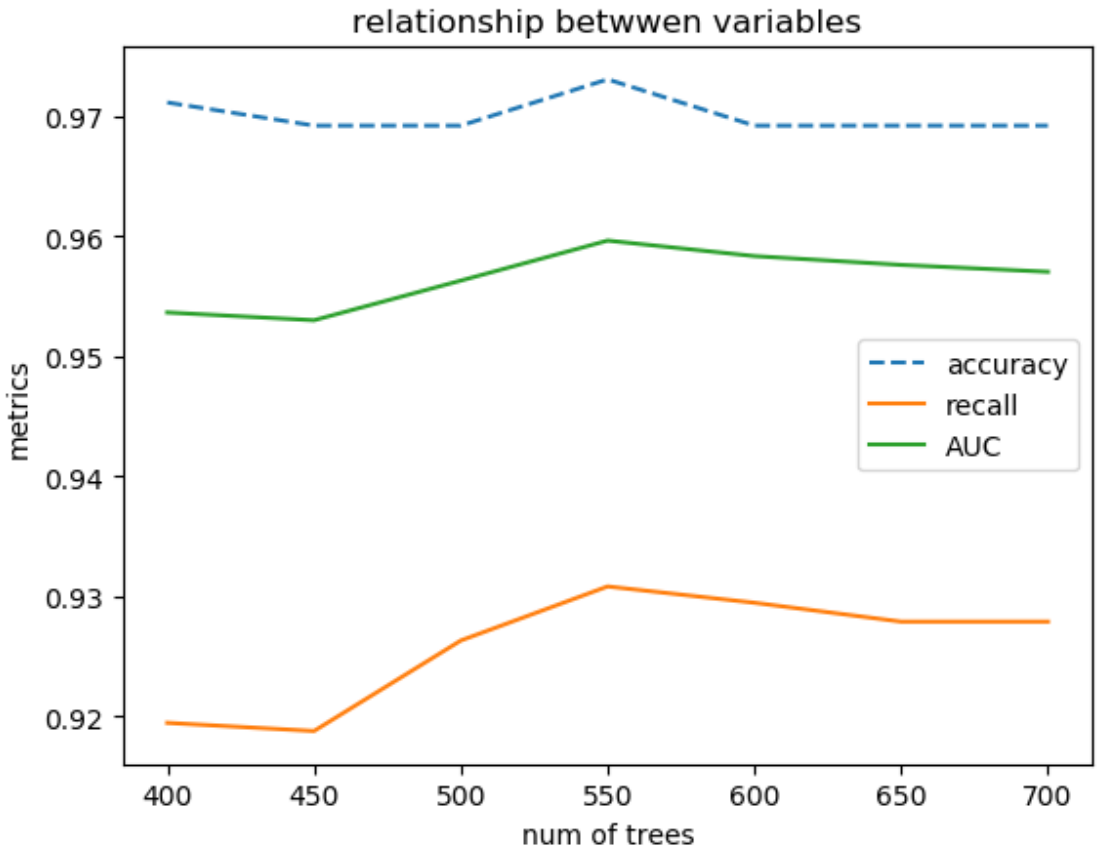


Fig: relationship between number of trees and accuracy

So, the graph above provides us a better vision. overall we think when the number of trees 550 , the random forest classifier has the best performance.

- **The most influential factor**

  There are 6 independent variables in the dataset. We will want to know which one affects the prediction result most. There is one attribute in the model called feature_importances that can help us decide

  Just apply the following code :

  feature_importances = pd.DataFrame(clf.feature_importances_,index=X_train.columns, columns=['importance']) .sort_values('importance', ascending=False)

  And then we can order the factors ordered by importance.

FF

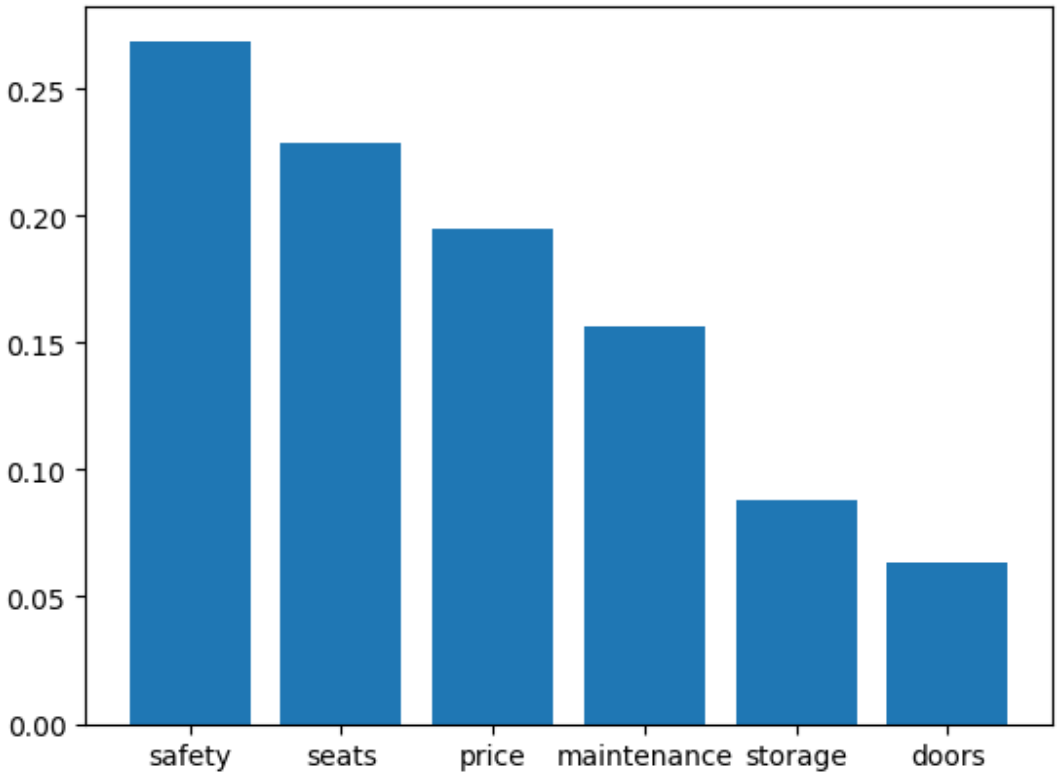


Fig: The independent variables which influence the prediction result most

The safety is the most important factor when the customer decided whether to buy a car or not.

5.2. Using R

The first step was confusing whether or not we had to clean or transform the given dataset. It was decided to do both ways as we split tasks and the end result did not change so we decided not to present it in the report.

After that, the division of data into two samples was done. Amongst all the available rows we had to divide them into train data which we will train using the prediction model and the test dataset which is the actual dataset on which we test the accuracy achieved by the

trained model. For this, it was very important for the data to be spread randomly so we shuffled the entire dataset in order to achieve uniformity throughout the process and in our end result. We shuffled the dataset and saved it in as the actual data.

```
shuffle_index <- sample(1:nrow(carData))
carData <- carData[shuffle_index, ]
```

And, then we used the sample in order to split the dataset into training and test in the ratio of 75:25(train:test). We even used the dataset in ration 80:20 and got a little different result which will be mentioned below alongside the end result.

```
sample <- sample.int(n = nrow(carData), size = floor(.75*nrow(carData)), replace = F)

data_train <- carData[sample,]
data_test <- carData[-sample,]
```

Where, the data_train now has a total of 1296 rows and data_test has a total of 432 rows.

For the above classified train data we used the following methodologies in order to classify them and come into a conclusion.

5.2.1. Decision Tree using R-part

- Decision tree classifier

```
node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 1296 395 unacc (0.21990741 0.04320988 0.69521605 0.04166667)
    2) seats=4,more 868 395 unacc (0.32834101 0.06451613 0.54493088 0.06221198)
      4) safety=high,med 587 302 acc (0.48551959 0.09540034 0.32708688 0.09199319)
        8) price=low,med 292 144 acc (0.50684932 0.19178082 0.11643836 0.18493151)
         16) maintenance=high,vhigh 144  38 acc (0.73611111 0.00000000 0.19444444 0.06944444)
           32) storage=big,med 96  17 acc (0.82291667 0.00000000 0.07291667 0.10416667) *
           33) storage=small 48  21 acc (0.56250000 0.00000000 0.43750000 0.00000000)
             66) safety=high 27   4 acc (0.85185185 0.00000000 0.14814815 0.00000000) *
             67) safety=med 21   4 unacc (0.19047619 0.00000000 0.80952381 0.00000000) *
         17) maintenance=low,med 148  92 good (0.28378378 0.37837838 0.04054054 0.29729730)
           34) safety=med 71  33 acc (0.53521127 0.43661972 0.02816901 0.00000000)
             68) storage=small 22   2 acc (0.90909091 0.00000000 0.09090909 0.00000000) *
             69) storage=big,med 49  18 good (0.36734694 0.63265306 0.00000000 0.00000000)
              138) maintenance=med 23  10 acc (0.56521739 0.43478261 0.00000000 0.00000000)
                276) price=med 10   0 acc (1.00000000 0.00000000 0.00000000 0.00000000) *
                277) price=low 13   3 good (0.23076923 0.76923077 0.00000000 0.00000000) *
              139) maintenance=low 26   5 good (0.19230769 0.80769231 0.00000000 0.00000000) *
           35) safety=high 77  33 vgood (0.05194805 0.32467532 0.05194805 0.57142857)
             70) storage=small 26   6 good (0.07692308 0.76923077 0.15384615 0.00000000) *
             71) storage=big,med 51   7 vgood (0.03921569 0.09803922 0.00000000 0.86274510) *
        9) price=high,vhigh 295 137 unacc (0.46440678 0.00000000 0.53559322 0.00000000)
         18) maintenance=high,low,med 222  85 acc (0.61711712 0.00000000 0.38288288 0.00000000)
           36) storage=big,med 147  38 acc (0.74149660 0.00000000 0.25850340 0.00000000)
             72) maintenance=low,med 93   9 acc (0.90322581 0.00000000 0.09677419 0.00000000) *
             73) maintenance=high 54  25 unacc (0.46296296 0.00000000 0.53703704 0.00000000)
              146) price=high 28   3 acc (0.89285714 0.00000000 0.10714286 0.00000000) *
              147) price=vhigh 26   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
           37) storage=small 75  28 unacc (0.37333333 0.00000000 0.62666667 0.00000000)
             74) safety=high 37   9 acc (0.75675676 0.00000000 0.24324324 0.00000000) *
             75) safety=med 38   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
         19) maintenance=vhigh 73   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
      5) safety=low 281   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
    3) seats=2 428   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
```

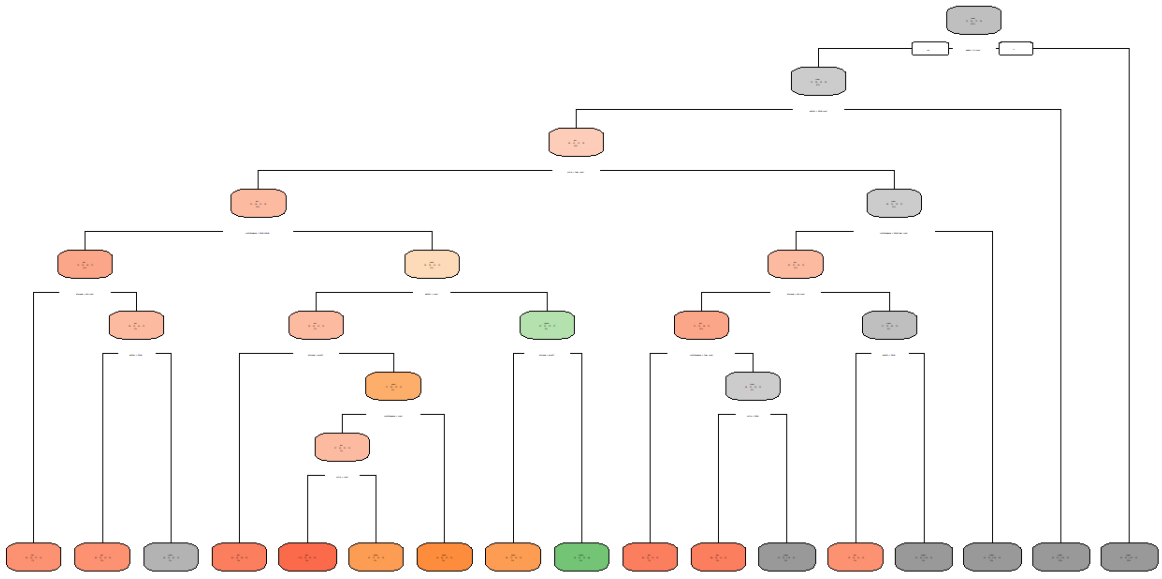


Fig: Decision Tree classifier for max-depth: 13

Since the diagram itself is not visible as the tree is overfitted so we decided to take the max depth as 5 in order to see the visible functionalities and it's flow

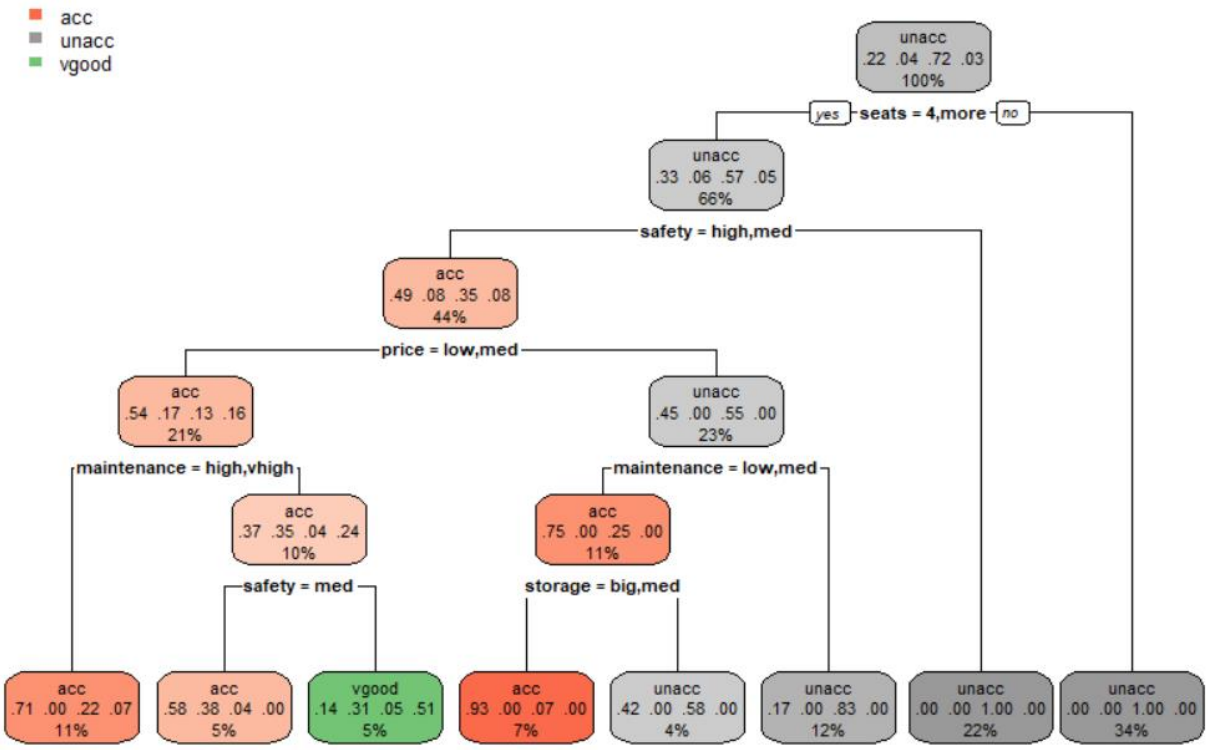


Fig: Decision Tree classifier for max-depth: 5

- Confusion matrix

```
#Confusion Matrix
treeCM = table(data_test[["shouldBuy"]],predictCar)
treeCM
      predictCar
       acc good unacc vgood
acc     89    6     3     1
good     0    9     0     4
unacc   11    0   298     0
vgood    3    0     0     8
```

- Accuracy

```
> #calculate accuracy for test_data
> sum(diag(treeCM)/sum(treeCM))
[1] 0.9351852
```

5.2.2. Random Forest and K-fold

Random forest helps to create a series of decision tree. The AUC for each parameter of the independent variable was calculated in

The AUC value for each of the class levels is illustrated below:

Class levels	AUC value
acc	0.877
good	0.6715
unacc	0.8973
vgood	0.7824

- **ROC-AUC**
  The ROC curve for each of the class levels is plotted below:

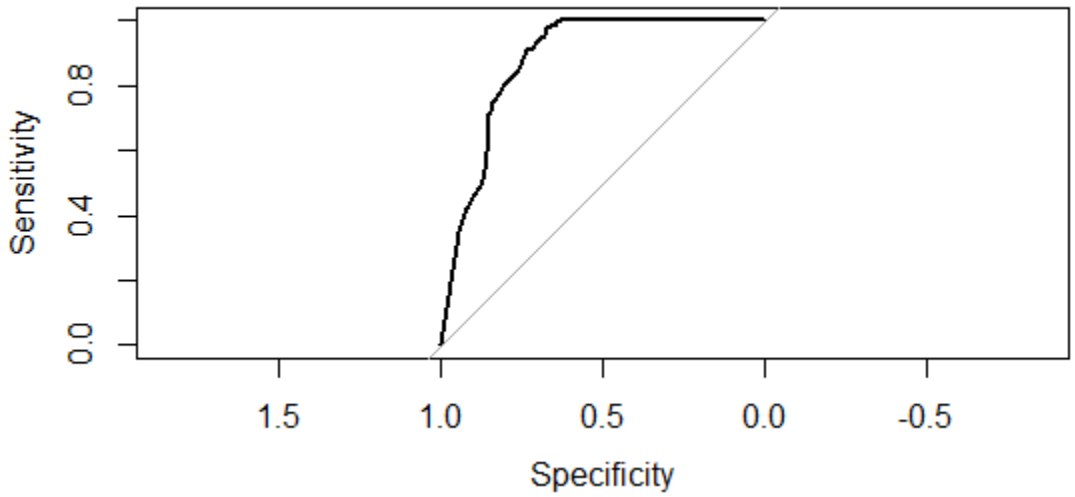


Fig: ROC curve for acc

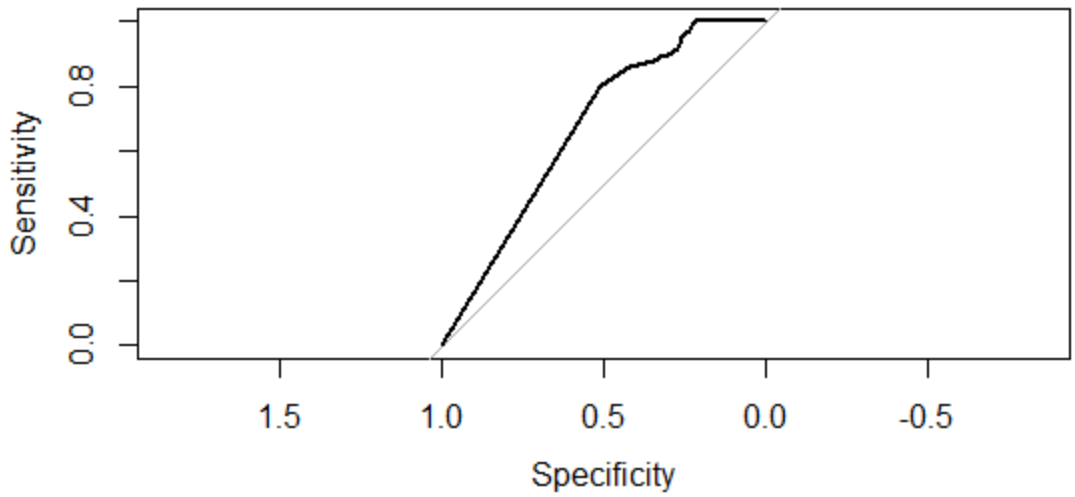


Fig: ROC curve for good

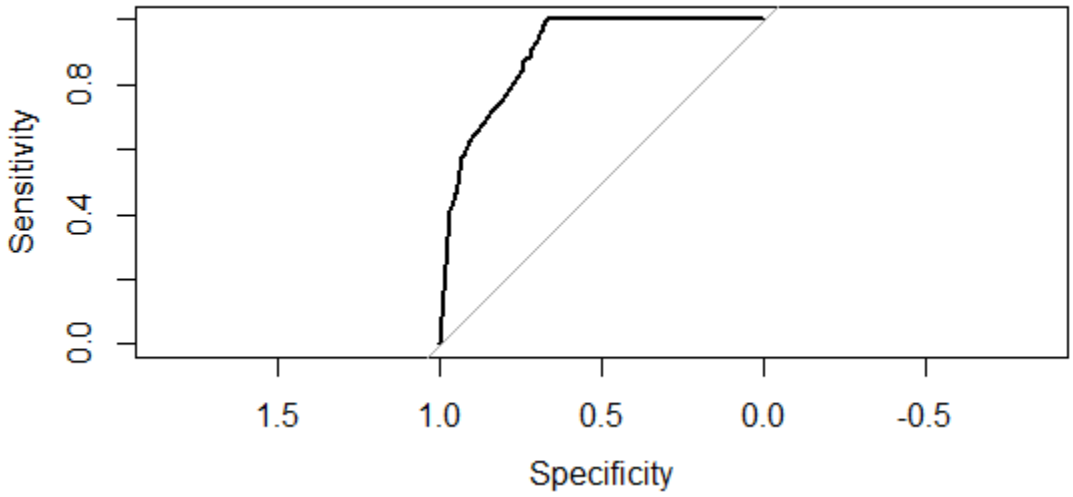


Fig: ROC curve for unacc

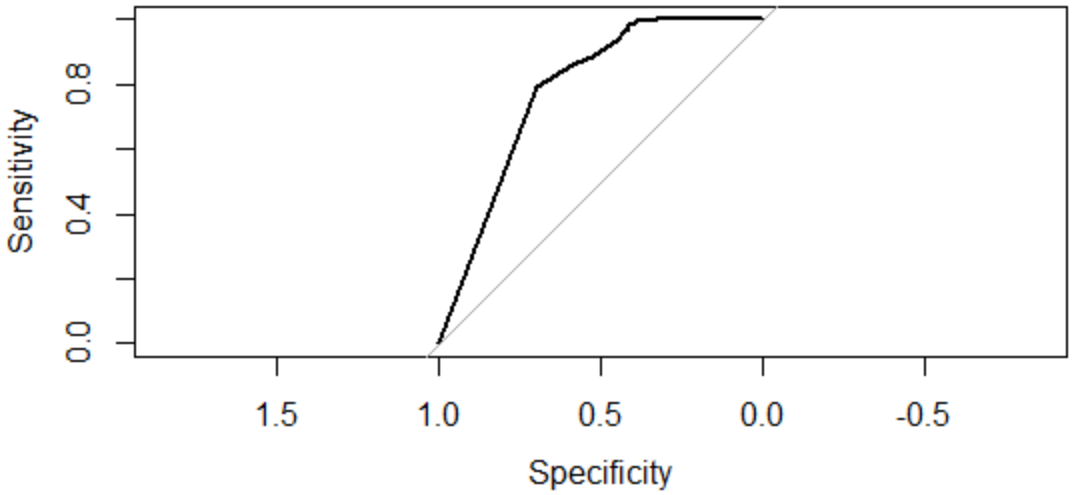


Fig: ROC curve for vgood

Here, we use cross-validation method for determining the significance of each variable.

```
Random Forest

1296 samples
   6 predictor
   4 classes: 'acc', 'good', 'unacc', 'vgood'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 1165, 1167, 1167, 1167, 1167, 1166, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  2     0.9639828  0.9236069
  4     0.9735238  0.9439422
  6     0.9766027  0.9502031

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 6.
> varImp(rf_default)
rf variable importance

            Overall
safety       100.00
seats         63.16
maintenance   51.72
storage       35.45
price         28.70
doors          0.00
```

5.2.3 Most Significant Variables

On the basis of k-fold cross-validation we derived the following chart which signifies the attributes the dependent variables are most influenced by in the dataset and shows which variable has the most effect on the decision making.

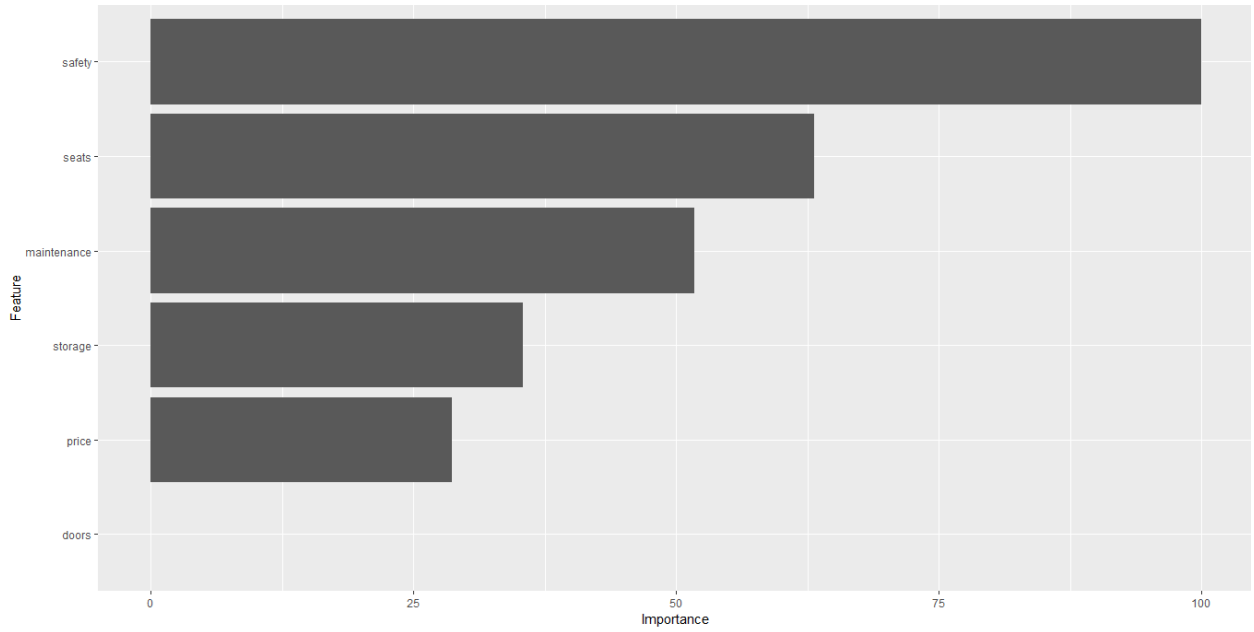


Fig: Most Significant Variables

Here, we can see that safety is the factor that affects the most in the decision for getting the car rather than seats and other factors number of doors being the least significant one.

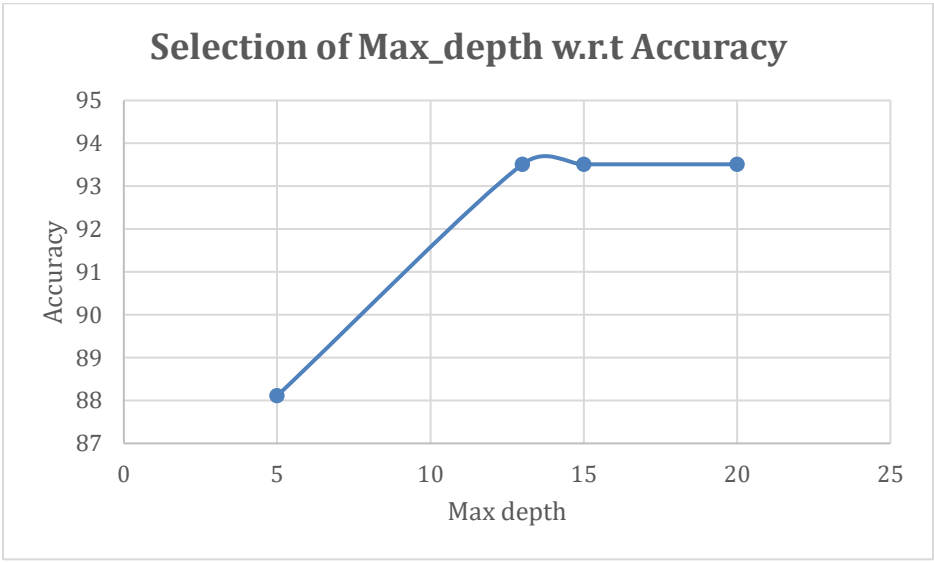


Fig: Max-depth with respect to Accuracy

6. Conclusion

Hence, we used python and R separately as per our preferences in order to classify the dataset. While using Python , we use scikit-learn(sklearn) to implement the whole pipeline.We split the 70% dataset to train and 30% to test.In decision tree model , the accuracy is 97%.Other metrics are also satisfying. And by random forest model ,we use entropy as criterion.After multiple iterations, we find when the number of trees is 550 ,the model has a best prediction result. While using R, mostly we used R-part to make the decision tree classifier and came to the conclusion that while maxdepth is 10 > it gives the significant accuracy i.e. 94.21% while applied on a 75:25 (train:test) model. Similarly, through random forest we found the ROC for each parameter of the independent variable and then finally used cross-validation to conclude that safety is the most prominent factor which is taken into consideration, whereas the number of doors is not considered as important as others.

7. References

[1] **Medium. (2019).** *Decision Trees—A simple way to visualize a decision*. [online] Available at: https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb [Accessed 5 Jun. 2019].

[2]**Towards Data Science.** (2019). *The Random Forest Algorithm*. [online] Available at: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd [Accessed 5 Jun. 2019].

[3] **Google Developers. (2019).** *Classification: Accuracy | Machine Learning Crash Course | Google Developers*. [online] Available at: https://developers.google.com/machine-learning/crash-course/classification/accuracy [Accessed 5 Jun. 2019].

8. Appendix A: Python Code

```
from sklearn.preprocessing import LabelEncoder,LabelBinarizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from sklearn.metrics import roc_curve, roc_auc_score,confusion_matrix,auc,recall_score
from sklearn.multiclass import OneVsRestClassifier

import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import Image
import pydotplus


def read_data():

    pima = pd.read_csv("car.data.txt", header=0)
    print(pima.head())

    le = LabelEncoder()
    categories_non_numerical = [
        'price', 'maintenance', 'storage', 'safety', 'shouldBuy']
    pima[categories_non_numerical] = pima[categories_non_numerical].apply(
        lambda col: le.fit_transform(col))
    print(set(pima['price']))

    pima['doors'].replace('5more', '5', inplace=True)
    pima['seats'].replace('more', '5', inplace=True)

    # split dataset in features and target variable
    feature_cols = ['price', 'maintenance',
                    'doors', 'seats', 'storage', 'safety']
    X = pima[feature_cols]  # Features

    target_cols = ['shouldBuy']
    y = pima[target_cols]  # Target variable
```

```
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=1)  # 70% training and 30% test

    return X_train, X_test, y_train, y_test


def generate_decision_tree_classifier():

    # Create Decision Tree classifer object
    # e.g.：max_depth, max_leaf_nodes, min_impurity_split, and min_samples_leaf
    clf = DecisionTreeClassifier()

    X_train, X_test, y_train, y_test = read_data()
    # Train Decision Tree Classifer
    clf = clf.fit(X_train, y_train)

    # Predict the response for test dataset
    y_predict = clf.predict(X_test)

    print("below is the predict result and test result")
    print(y_predict)
    print(y_test)

    # about K-fold test
    from sklearn.model_selection import cross_val_score
    # kf = KFold(25, n_folds=5, shuffle=False)
    depth = []
    for i in range(1,50):
        clf = DecisionTreeClassifier(max_depth=i)
        # Perform 7-fold cross validation
        scores = cross_val_score(estimator=clf, X=X_train, y=y_train, cv=7, n_jobs=4)
        depth.append((i,scores.mean()))
    print(depth)
    x=[f[0] for f in depth]
    y=[f[1] for f in depth]
    print("x is ")
    print(x)
    print(y)
    plt.scatter(x,y)
```

```
plt.title("relationship between")
plt.xlabel("max_depth")
plt.ylabel("accuracy")
plt.show()

# Model Accuracy, how often is the classifier correct?
print("Accuracy:", accuracy_score(y_test, y_predict))
print("recall_score: ",recall_score(y_test,y_predict, average='macro'))

# calculate confusion matrix
# Just For random test
cm = pd.DataFrame(
    confusion_matrix(y_test, y_predict),
    columns=['l', '2', '3', '4'],
    index=['1', '2', '3', '4']
)
print(cm)

# auc and roc
lb = LabelBinarizer()
lb.fit(y_test)
y_test1 = lb.transform(y_test)
y_pred = lb.transform(y_predict)
print(roc_auc_score(y_test1, y_pred, average='macro'))

n_classes = 4
"""
TODO: cannot find a better way to deal with multiple-class
roc and auc plot.
So convert it into two-classes problem
"""
#  predict
print(type(y_predict))
y_predict[y_predict <= 1]=0
y_predict[y_predict >= 2]=1
#  y_test
y_test.replace(1,0,inplace=True)
y_test.replace(2,1,inplace=True)
y_test.replace(3,1,inplace=True)
```

```
    fpr, tpr, thresholds = roc_curve(y_test,y_predict)
    plt.plot([0, 1], [0, 1], linestyle='--')
    plt.plot(fpr, tpr, marker='.')
    plt.show()


def generate_random_forest_classfier():
    # 500 means the number of ntrees
    from sklearn.ensemble import RandomForestClassifier
    clf = RandomForestClassifier(n_estimators=600, criterion="entropy")
    X_train, X_test, y_train, y_test = read_data()
    clf.fit(X_train, y_train)

    print(clf)
    predicitions = clf.predict(X_test)

    print("Accuracy:", accuracy_score(y_test, predicitions))
    feature_importances = pd.DataFrame(clf.feature_importances_,
                                       index=X_train.columns,
                                       columns=['importance']) .sort_values('importance',ascending =
False)
    importance = list(feature_importances.iloc[:,0])
    names = list(feature_importances.index)

    plt.bar(names,importance)
    plt.show()



if __name__ == '__main__':
    # uncomment the following comment to run 2 classifer trees both
    #generate_random_forest_classfier()
    generate_decision_tree_classifier()

    # graph.write_png('diabetes.png')
    # Image(graph.create_png())
```

9. Appendix B: R Script

- **R-Part**

```
carData <- read.csv("C:/Users/Tushar Mahat/Downloads/Rishi Downloads/MCDA 2nd Sem/Data Mining/Assignment 2/car_data.csv")

View(carData)

set.seed(50)

head(carData)

tail(carData)

#First we shuffle carData prior to giving ratio of train and test dataset
shuffle_index <- sample(1:nrow(carData))
carData <- carData[shuffle_index, ]
head(carData)

tail(carData)

#Deciding train and test ratio
sample <- sample.int(n = nrow(carData), size = floor(.75*nrow(carData)), replace = F)

data_train <- carData[sample,]
data_test <- carData[-sample,]

library(rpart)
library(rpart.plot)

#create model
treeCar =
rpart(shouldBuy~.,data=data_train,method="class",control=rpart.control(maxdepth = 13))

#list rules
treeCar

#plot tree
```

```
rpart.plot(treeCar)

#prediction
predictCar<-predict(treeCar, newdata=data_test, type ='class')
head(predictCar)

#Confusion Matrix
treeCM = table(data_test[["shouldBuy"]],predictCar)
treeCM

#calculate accuracy for test_data
sum(diag(treeCM)/sum(treeCM))
```

- **Random Forest**

```
#Try randomForest
x=data_train[,1:6]
y=data_train[,7]
library(randomForest)
library(pROC)

rf=randomForest(x,y,nodesize=10)
rfp=predict(rf,x)
rfCM=table(rfp,y)
rfCM
sum(diag(rfCM))/sum(rfCM)
rfProb=predict(rf,x,type="prob")


rocAcc = roc(data_train[,6],rfProb[,1])
plot(rocAcc)
rocAcc

rocGood = roc(data_train[,6],rfProb[,2])
plot(rocGood)
rocGood

rocUnacc = roc(data_train[,6],rfProb[,3])
plot(rocUnacc)
rocUnacc
```

```
rocVgood = roc(data_train[,6],rfProb[,4])
plot(rocVgood)
rocVgood
```

- **K fold**

```
# Use caret package for 10 fold cross validation
library(caret)
control <- trainControl(method="repeatedcv", number=10, repeats=3)
seed <- 7
metric <- "Accuracy"
set.seed(seed)
rf_default <- train(x,y, method="rf", metric="accuracy", trControl=control)
print(rf_default)
varImp(rf_default)
ggplot(varImp(rf_default))
```