SAINT MARY'S UNIVERSITY SINCE 1802

One University. One World. Yours.

MCDA 5580 DATA & TEXT MINING

Assignment 2

Classification with Car Data

Team Members

Names	A#
Chacko, Jelson	A00446838
Kuzhippallil, Maria A.	A00442283
Muotoe, Somto J.	A00442756
Reginold, Fabian Vaniyamveetil	A00447210

Table of Contents

List of Figures

List of Tables

Executive Summary

Data is growing exponentially every day and comprehending all the data with higher accuracy and a faster pace is practically unfeasible. Supervised machine learning algorithms have indeed proved to have superhuman capabilities in classifying high dimensional input data with great precision and accuracy. With the help of decision trees and random forest algorithms that runs through the dataset comprising of car details, these cars were effectively and accurately classified into 4 different categories. Cars with similar patterns and characteristics were categorized to glean business insights and drive business decisions. Furthermore, the rule engine generated by the machine learning model, help business decision-makers make data-driven decisions, thereby boosting the company's revenue.

Objective

The goal of classification algorithms is to recognize patterns in existing data and find similar patterns in future sets of data. In other words, the primary goal of this classification algorithm is to place cars into specific categories and answer questions like: Would the customer buy this car? Which category does this car fall into? In short, the objective is to utilize pre-categorized car datasets and classify future cars into respective categories of being bought or not.

About the Data

The data provided comprises the acceptability of a car based on various attributes such as the price of the car, its maintenance, the number of doors present, the number of seats in the car, the storage in the car and its safety.

A summary of the data that have been taken into consideration for this analysis is as follows:

Number of Records	Unique Records	Number of Attributes
1728	1728	7

Table 1. About the data

The description of all the features included in the data is given as follows:

Attribute	Feature Description
Price	The total price of the car categorized in bins such as very high, high, medium, low.
Maintenance	Overall maintenance price of the car categorized in bins such very high, high, medium, low
Doors	The total number of doors present in each car such as. 2,3,4,5 and more
Seats	The number of seats available for a car ranging from 2, 4 and so on.
Storage	Miscellaneous storage compartment present in the car ranging between small medium and big compartments.
Safety	The overall safety of the car ranked between low, medium, and high.

Table 2. Feature description (Attributes)

Target	Description
ShouldBuy	The variable whose values are to be modelled and predicted by other variables indicating whether a given car is accepted, unaccepted, good, or very high

Table 3. Feature description (Target)

Design/Methodology/Approach

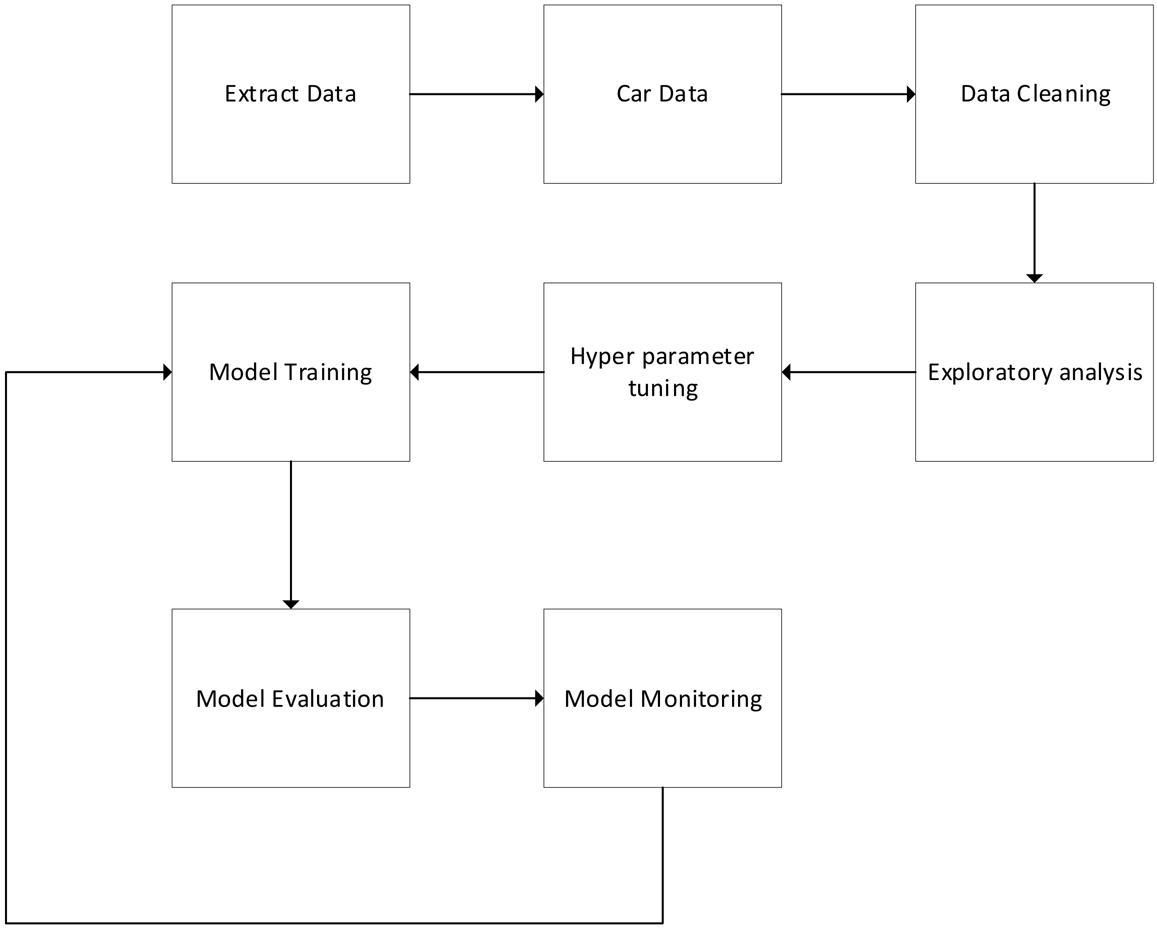

Figure 1. Design methodology

Data Extraction

Data Extraction is perhaps the most important part of this process because it includes the decision making on which data is most valuable in our analysis. For this analysis, we made use of the car dataset collected from the dev.cs.smu.ca server. We exported the entire table consisting of 1728 rows and 7 columns.

Data Pre-processing

Data pre-processing is a crucial first step for anyone dealing with data sets and one of the main reasons as to why this is true is because it leads to a cleaner and more manageable data set. In our analysis, we checked if the dataset consisted of null and redundant data to make sure we have clean data.

Exploratory Data Analysis

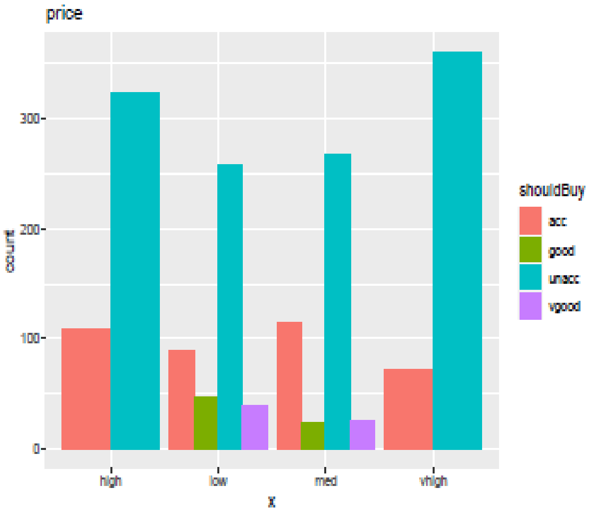

Figure 2. Price

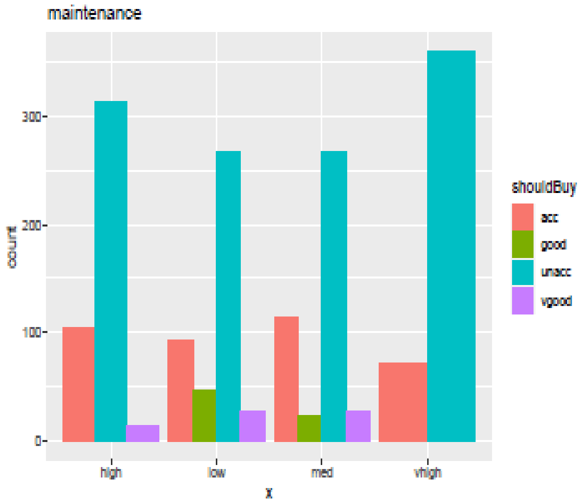

Figure 3. Maintenance

In the above two figures, cars with high overall price and high maintenance price respectively are generally unaccepted to be purchased by customers while fairly average priced cars are accepted the most.

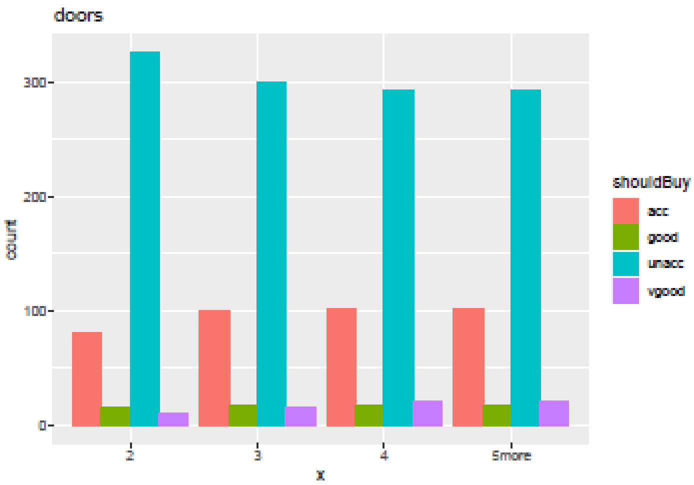

Figure 4. Doors

In this visualization, cars with doors equal to or more than 3 are accepted than the others. On the other hand, doors with 2 doors are highly unaccepted.

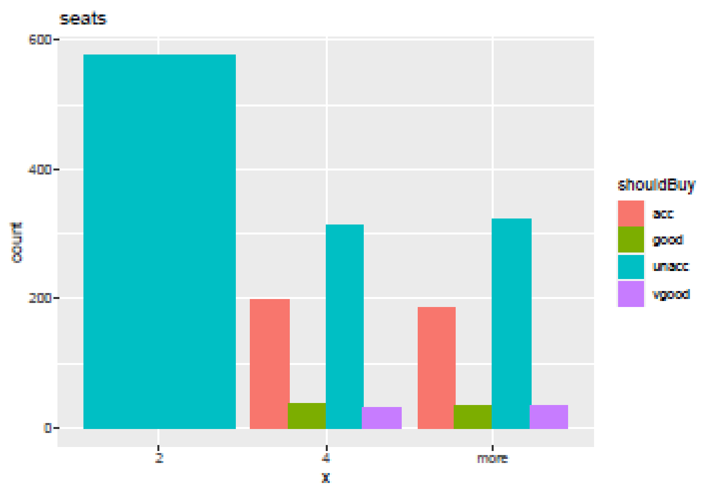

Figure 5. Seats

This figure tells us that 2-seater cars are unaccepted more than the other seaters by a great margin.

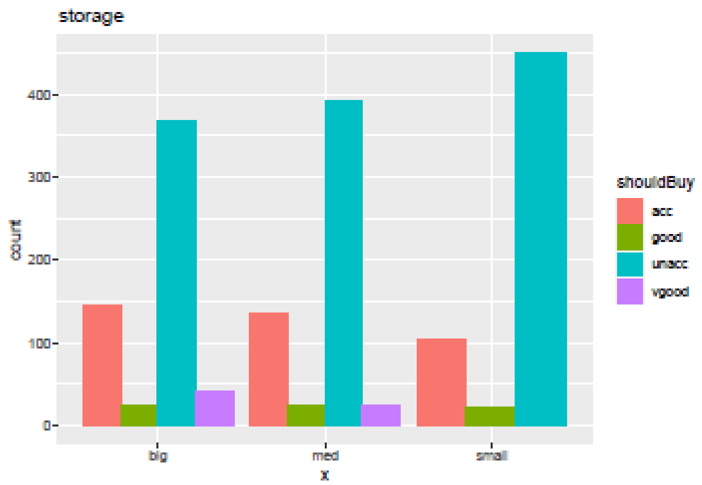

Figure 6. Storage

This analysis tells us that cars with bigger trunks are preferred more than the ones with smaller ones.

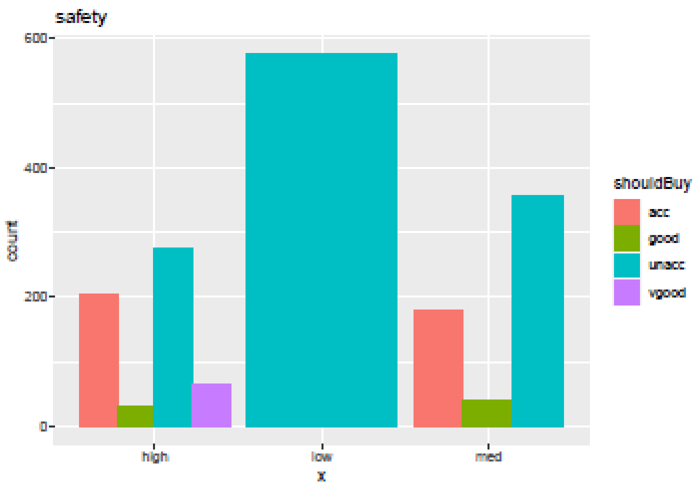

Figure 7. Safety

In this analysis, cars with low safety are highly unaccepted by customers so much so they are never considered as an option to be purchased.

Hyperparameter Tuning

Hyperparameter tuning is the process to determine the right combination of hyperparameter that increases the model performance.

In every machine learning analysis, it is essential to find the best version of the model. This is achieved by running many jobs by testing a range of hyperparameters on the dataset to get the best fit model.

In our analysis done for this data, we implemented the Random Search approach which selects random combinations to train the model and get the approximate optimized parameters. Now that we have a smaller number of parameters, Grid Search Cross-validation is then implemented which sets up a grid of hyper-parameters to select the best fit model parameters, thereby improving the performance of the model significantly.

The optimization algorithms used are -

- **Random Search** – sets grid of hyperparameter and randomly selects combinations for training model.
- **Grid Search** - sets grid of hyperparameter and every single combination of hyperparameter is checked. The model with the set of parameters with top accuracy is selected.

Hyperparameter tuning for Rpart:

- `minsplit`: the minimum number of observations that must exist in a node in order for a split to be attempted.
- `minbucket`: the minimum number of observations in any terminal (leaf) node
- `maxdepth`: maximum depth of any node of the final tree
- `cp`: Complexity Parameter

Hyperparameter tuning for Random Forest:

- `ntree`: Number of trees
- `mtry`*:* Number of variables randomly sampled.

Model Building and Prediction

After obtaining the parameters for the best fit model, we fit the final model to learn the relationship between the predictors and the target variable. As a result, a rule-based engine is generated after training the decision tree model which is comparatively more accurate and precise as compared to the traditional rule-based engine.

- **Rpart (Recursive Partitioning) –** is a supervised machine learning algorithm, used for building decision tree for classification and regression. The algorithm works by splitting independent variables into purest groups based on the levels of the target variable. Assessment of purity is decided based on the Gini index.
- **Random Forest** – is a supervised machine learning algorithm consisting of an ensemble of decision trees. Random forest builds multiple decision trees to get a more accurate prediction.

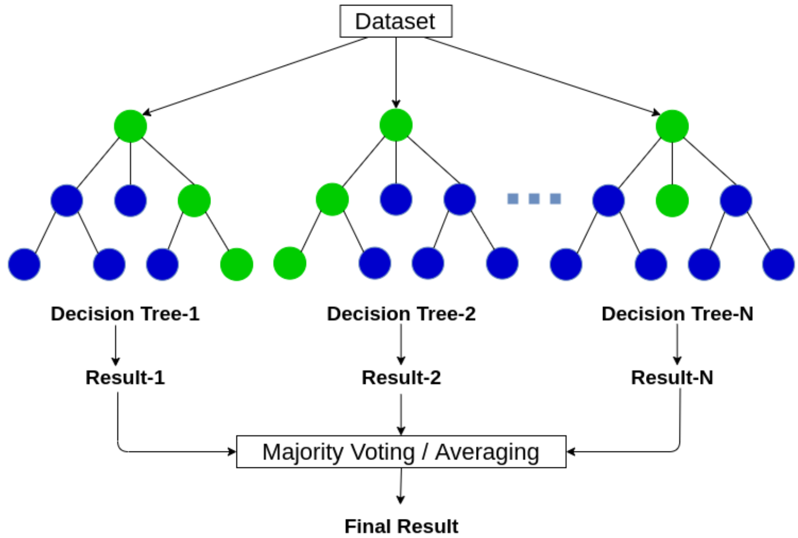

Figure 8. Random Forest Tree

Performance Assessment

- Confusion Matrix

		Actual	
		Positive	**Negative**
Predicted	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Table 4. Confusion matrix

- Accuracy: evaluates correctly predicted data points (TP+TN) to total data (TP+TN+FP+FN).

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Recall (Sensitivity): evaluates the proportion of actual positive (TP+FN) that are correctly identified (TP)

$$\text{Recall} = \frac{TP}{TP + FN}$$

- AUC (Area Under Curve): is a two-dimensional area under ROC Curve. It measures the ability of a classifier to distinguish between the classes. The greater the AUC, the greater the ability of the classifier to distinguish a class from the other different classes.

- F Score: is a single score that balances both the concerns of precision and recall in one number which provides a great measure.

Feature Selection

Non-informative variables can certainly add uncertainty and variability to the prediction, thereby shrinking the accuracy and precision of the model. Thus, we implemented feature selection using recursive partitioning and random forest methodologies to remove redundant, non-informative and spurious variables.

Using Random Forest Algorithm, the feature importance can be obtained as follows:

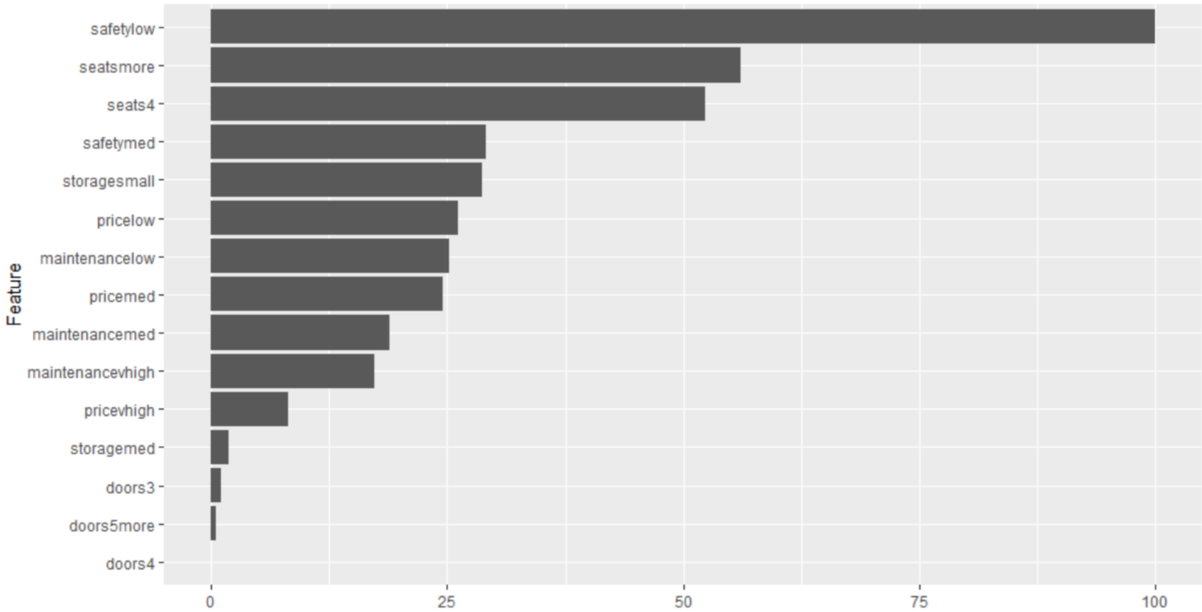

Figure 9. Feature Importance (Random Forest)

Using the Recursive Partitioning Algorithm, the feature importance can be obtained as follows:

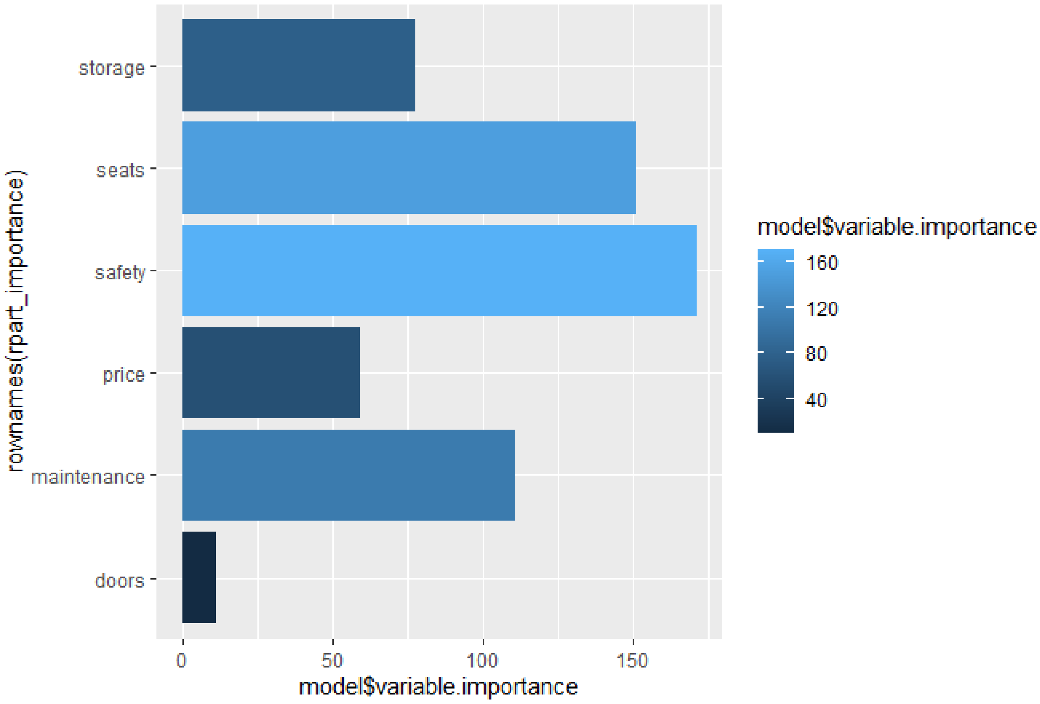

Figure 10. Feature Importance (Recursive Partitioning)

In our analysis, both the models gave us more or less the same variable importance ranging from safety being the highest to doors being the lowest. However, in our situation, all the variables are significantly important in improving the prediction performance of the predictors, hence, we retained all the features ranked by the score below.

List of Rules – Decision Tree

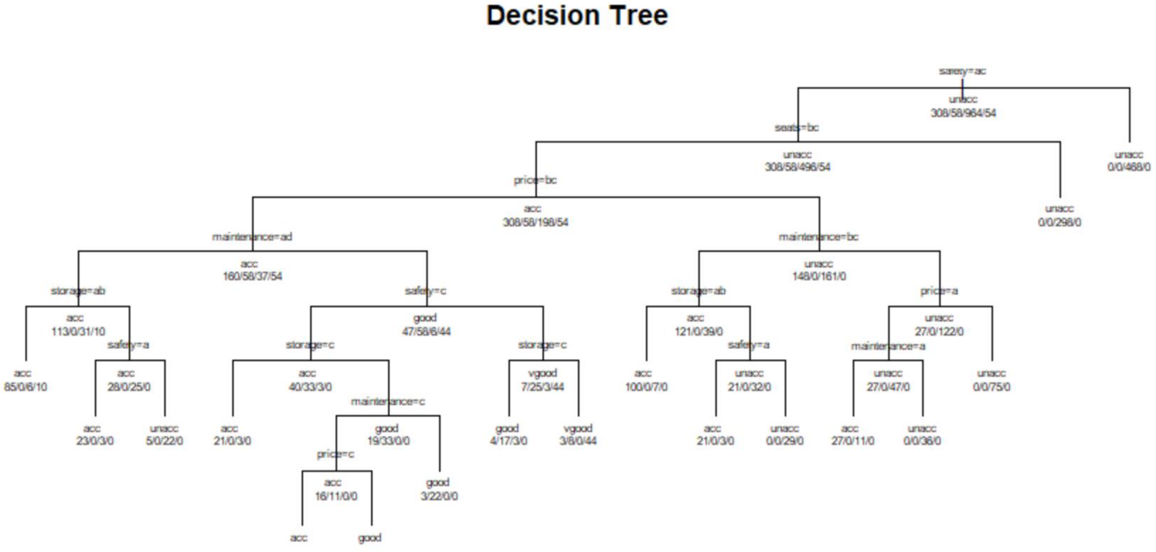

Figure 11. Decision Tree

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

  1) root 1384 420 unacc (0.22254335 0.04190751 0.69653179 0.03901734)
    2) safety=high,med 916 420 unacc (0.33624454 0.06331878 0.54148472 0.05895197)
      4) seats=4,more 618 310 acc (0.49838188 0.09385113 0.32038835 0.08737864)
        8) price=low,med 309 149 acc (0.51779935 0.18770227 0.11974110 0.17475728)
         16) maintenance=high,vhigh 154  41 acc (0.73376623 0.00000000 0.20129870 0.06493506)
           32) storage=big,med 101  16 acc (0.84158416 0.00000000 0.05940594 0.09900990) *
           33) storage=small 53  25 acc (0.52830189 0.00000000 0.47169811 0.00000000)
             66) safety=high 26   3 acc (0.88461538 0.00000000 0.11538462 0.00000000) *
             67) safety=med 27   5 unacc (0.18518519 0.00000000 0.81481481 0.00000000) *
         17) maintenance=low,med 155  97 good (0.30322581 0.37419355 0.03870968 0.28387097)
           34) safety=med 76  36 acc (0.52631579 0.43421053 0.03947368 0.00000000)
             68) storage=small 24   3 acc (0.87500000 0.00000000 0.12500000 0.00000000) *
             69) storage=big,med 52  19 good (0.36538462 0.63461538 0.00000000 0.00000000)
              138) maintenance=med 27  11 acc (0.59259259 0.40740741 0.00000000 0.00000000)
                276) price=med 13   0 acc (1.00000000 0.00000000 0.00000000 0.00000000) *
                277) price=low 14   3 good (0.21428571 0.78571429 0.00000000 0.00000000) *
              139) maintenance=low 25   3 good (0.12000000 0.88000000 0.00000000 0.00000000) *
           35) safety=high 79  35 vgood (0.08860759 0.31645570 0.03797468 0.55696203)
             70) storage=small 24   7 good (0.16666667 0.70833333 0.12500000 0.00000000) *
             71) storage=big,med 55  11 vgood (0.05454545 0.14545455 0.00000000 0.80000000) *
        9) price=high,vhigh 309 148 unacc (0.47896440 0.00000000 0.52103560 0.00000000)
         18) maintenance=low,med 160  39 acc (0.75625000 0.00000000 0.24375000 0.00000000)
           36) storage=big,med 107   7 acc (0.93457944 0.00000000 0.06542056 0.00000000) *
           37) storage=small 53  21 unacc (0.39622642 0.00000000 0.60377358 0.00000000)
             74) safety=high 24   3 acc (0.87500000 0.00000000 0.12500000 0.00000000) *
             75) safety=med 29   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
         19) maintenance=high,vhigh 149  27 unacc (0.18120805 0.00000000 0.81879195 0.00000000)
           38) price=high 74  27 unacc (0.36486486 0.00000000 0.63513514 0.00000000)
             76) maintenance=high 38  11 acc (0.71052632 0.00000000 0.28947368 0.00000000) *
             77) maintenance=vhigh 36   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
           39) price=vhigh 75   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
      5) seats=2 298   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
    3) safety=low 468   0 unacc (0.00000000 0.00000000 1.00000000 0.00000000) *
```

Rules

IF safety == low THEN *unacc*

ELSE IF safety == high OR safety == med
IF seats = 2 THEN *unacc*
ELSE IF seats == 4 OR seats == more
IF price == low OR price == med

```
        IF  maintenance == high OR maintenance == vhigh
                IF storage == small
                        IF safety == high THEN acc
                        ELSE IF safety == med THEN unacc
                ELSE IF storage == big OR storage == med THEN acc
        ELSE IF maintenance == low OR maintenance == med
                IF safety ==  med
                        IF storage == small THEN acc
                        ELSE IF storage == big OR storage == med
                                IF maintenance == med
                                        IF price == med THEN acc
                                        ELSE IF price == low THEN good
                                ELSE IF maintenance == low THEN good
                ELSE IF safety == high
                        IF storage == small THEN good
                        ELSE IF storage == big OR storage == med THEN vgood
ELSE IF price == high OR price == vhigh
        IF maintenance == low OR maintenance == med
                IF storage == big OR storage == med THEN acc
                ELSE IF storage == small
                        IF safety == high THEN acc
                        ELSE IF safety == med THEN unacc
        ELSE IF maintenance == high OR maintenance == vhigh
                IF price == vhigh THEN unacc
                ELSE IF price == high
                        IF maintenance == high THEN acc
                        ELSE IF maintenance == vhigh THEN unacc
```

Model Evaluation

Model evaluation is a substantial part of the machine learning model development process which evaluates how well the model has performed in classifying into respective categories. In our analysis, we evaluated the model using parameters such as accuracy, recall, specificity, and other necessary parameters.

Before Hyperparameter tuning:

pred_random_forest	acc	good	unacc	vgood
acc	72	1	7	2
good	2	10	0	0
unacc	2	0	239	0
vgood	0	0	0	9

Table 5. Confusion Matrix for random forest before tuning

Prediction	acc	good	unacc	vgood
acc	68	0	12	3
good	6	10	1	0
unacc	2	0	233	0
vgood	0	1	0	8

Table 6. Confusion Matrix for decision tree before tuning

After Hyperparameter tuning:

pred_random_forest	acc	good	unacc	vgood
acc	75	0	0	1
good	1	11	0	0
unacc	0	0	246	0
vgood	0	0	0	10

Table 7. Confusion Matrix for random forest after tuning

Prediction	acc	good	unacc	vgood
acc	63	0	8	0
good	6	11	1	0
unacc	6	0	237	0
vgood	1	0	0	11

Table 8. Confusion Matrix for decision tree after tuning

From the tables above and the comparison matrix below, we can see the increase in the accuracy of the models before we tuned the parameters and then after tuning. In Table 9, we see that the accuracy of the decision tree model increased a bit from 92% to 93%. Furthermore, we observe a surge in the accuracy in the random forest model going from an already high 95% to an almost perfect 99% accuracy (Table 10).

Comparison Matrix	
Algorithm	**Accuracy**
Decision Tree	92%
Random Forest	95%

Table 9. Accuracy before hyperparameter tuning

Comparison Matrix	
Algorithm	**Accuracy**
Decision Tree	93%
Random Forest	99%

Table 10. Accuracy after hyperparameter tuning

For more information on the performance analysis of the models, please see Appendix C.

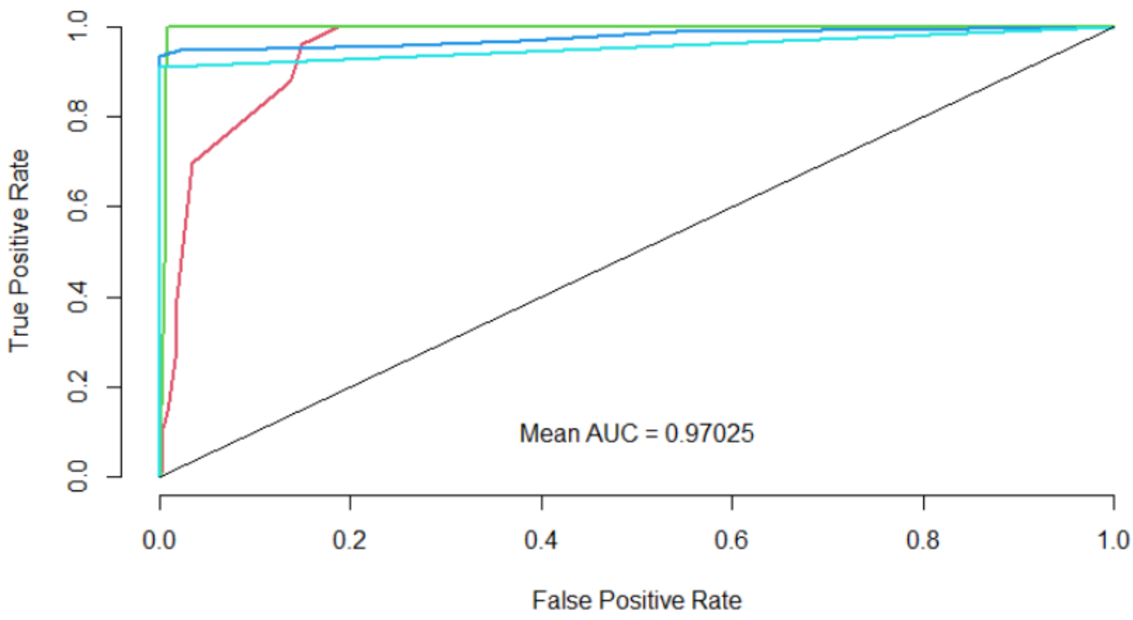

Figure 12. Multiclass ROC (Recursive Partitioning)

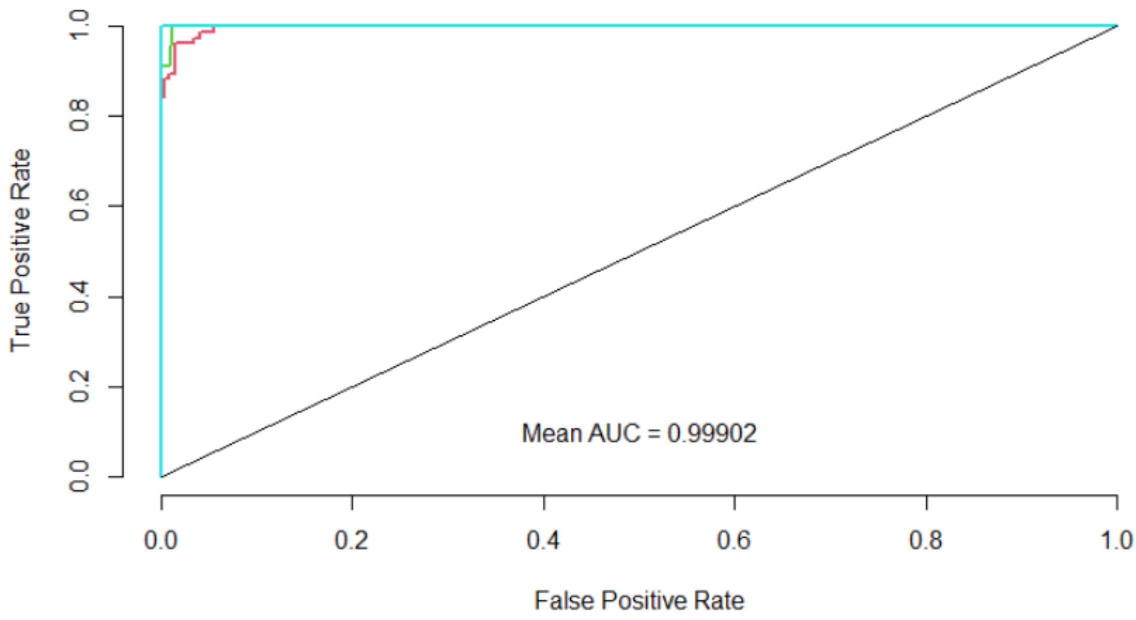

Figure 13. Multiclass ROC (Random Forest)

The ROC curve shows the trade-off between sensitivity and specificity. In our analysis, the first figure tells us that there is a 97% chance that the decision tree model will be able to distinguish a class from the other different classes while the second figure tells us that there is a 99.9% chance the random forest model will accurately distinguish a class from the other classes.

Conclusion and Next Steps

To wrap up, classification in supervised learning for both decision and random forest uses underlying statistical models to perform analytical tasks that would take human beings many hours to classify. After extracting the data and pre-processing it, and applying hyper-parameter tuning which helped in building the best-fit machine learning classification supervised model for further analysis and recommendations.

From the analysis, we observed that random forest consistently had a higher prediction accuracy than decision trees. However, from both models, we got a very similar hierarchy of importance in the variables.

This consequently leads us to offer meaningful recommendations such as increasing the safety of the car, decrease the manufacturing of two-seater cars, targeted marketing for potential customers who would buy the car, and so on, with great accuracy and precision. Thereby guiding the business decision-makers to make data-driven decisions and solve business challenges.

Our next steps would be to use wrapper classes around Random Forest for feature selection and use Bayes Search for hyperparameter tuning to converge at the local minima quickly thereby reducing the training time.

Appendix-A [References]

5 Types of Classification Algorithms in Machine Learning. (n.d.). Retrieved from https://monkeylearn.com/blog/classification-algorithms/

Brownlee, J. (2021, January 13). *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*. Retrieved from https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

Classification in R Programming: The all in one tutorial to master the concept! (n.d.). Retrieved from https://data-flair.training/blogs/classification-in-r/

Narkhede, S. (2018, June 26). *Understanding AUC - ROC Curve*. Retrieved from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Waseem, M. (2020, July 21). *How To Implement Classification In Machine Learning?* Retrieved from https://www.edureka.co/blog/classification-in-machine-learning/

Appendix-B [R Code]

```
library(tidyverse)
library(ggplot2)
library(dplyr)
library(rpart)
library(pROC)
library(caret)
library(mlr)
library(ROCR)
library(klaR)

# RANDOM SEED
set.seed(100)

# DATA EXTRACTION
cardata <- read.csv("car.data.csv")

# DATA EXPLORATION

dim(cardata)

# transform all columns to factor variables
cardata <- map_df(cardata, as.factor)

i <- 0;
lapply(cardata %>% dplyr::select(price:safety), function(x, y = colnames(cardata)){
    i <<- i + 1;
    ggplot(cardata) +
        geom_bar(aes(x, fill = shouldBuy), position = "dodge") +
        ggtitle(y[i])
})

lapply(cardata, function(x) { table(x) }) #no typos error

sum(!complete.cases(cardata)) #rows with NA

cardata <- cardata[complete.cases(cardata),]
#data.samples <- sample(1:nrow(cardata), nrow(cardata) *0.7, replace = FALSE)

#PARTITIONING DATA
data.samples <- createDataPartition(cardata$price, p = 0.8, list = FALSE, times = 1)

# SPLIT DATA INTO TRAIN AND TEST
train.data <- cardata[data.samples, ]
test.data <- cardata[-data.samples, ] %>% select(-shouldBuy)

# MODEL BULDING - Random Forest Tree
#-----------------------------------

# HYPER TUNING OF PARAMETERS

# USE STRATIFIED K FOLD ?

prop.table(table(cardata[data.samples,]$shouldBuy))
prop.table(table(cardata[-data.samples,]$shouldBuy))

#K FOLD
cartask <- mlr::makeClassifTask(data = mutate_all(cardata,as.factor) , target =
"shouldBuy")
tree <- mlr::makeLearner("classif.randomForest")
treeParamSpace <- makeParamSet(
    makeIntegerParam("ntree", lower = 390, upper = 410),
    makeIntegerParam("mtry", lower = 5, upper = 5))

# randSearch <- makeTuneControlRandom(maxit = 150)

gridSearchCV <- makeTuneControlGrid()

# bayesSearch <- makeTuneControlMBO()

cvForTuning <- makeResampleDesc("CV", iters = 10, stratify = TRUE)

tunedTreePars <- tuneParams(tree, task = cartask,
                            resampling = cvForTuning,
```

```
                            par.set = treeParamSpace,
                            control = gridSearchCV)

rfmodel <- randomForest::randomForest(shouldBuy ~ ., data = train.data, importance = TRUE,
                                      ntree = 399,
                                      mtry = 5)

pred_random_forest <- predict(rfmodel, test.data)

table(pred_random_forest, cardata[-data.samples, ]$shouldBuy) #confusion matrix

#ACCURACY, SPECIFICITY, SENSITIVITY, ROC

confusionMatrix(pred_random_forest, factor(cardata[-data.samples, ]$shouldBuy))

# plot(roc(response = cardata[-data.samples, ]$shouldBuy,
#          predictor = factor(pred_random_forest, ordered = TRUE)))
#
# text(x = 0, y = 0.1, labels = paste("AUC =", auc(roc(response = cardata[-data.samples,
]$shouldBuy, predictor = factor(pred_random_forest,ordered = TRUE)))))

multiclass_roc_random_forest <- function(){
    lvls = levels(cardata$shouldBuy)

    aucs = c()
    plot(x=NA, y=NA, xlim=c(0,1), ylim=c(0,1),
         ylab='True Positive Rate',
         xlab='False Positive Rate',
         bty='n')

    for (type.id in 1:4) {
        type = as.factor(train.data$shouldBuy == lvls[type.id])

        #nbmodel = randomForest::randomForest(type ~ ., data=train.data[, -7],
importance=TRUE)

        rfmodel <- randomForest::randomForest(type ~ ., data = train.data[, -7], importance
= TRUE,
                                              ntree = 399,
                                              mtry = 5)
        rfprediction = predict(rfmodel, test.data, type = "prob")

        score = rfprediction[, 'TRUE']
        actual.class = cardata[-data.samples, ]$shouldBuy == lvls[type.id]

        pred = prediction(score, actual.class)
        nbperf = ROCR::performance(pred, "tpr", "fpr")

        roc.x = unlist(nbperf@x.values)
        roc.y = unlist(nbperf@y.values)
        lines(roc.y ~ roc.x, col=type.id+1, lwd=2)

        nbauc = ROCR::performance(pred, "auc")
        nbauc = unlist(slot(nbauc, "y.values"))
        aucs[type.id] = nbauc
    }

    lines(x=c(0,1), c(0,1))

    text(x = 0.5, y = 0.1, labels = paste("Mean AUC =", round(mean(aucs), 5)))

}

multiclass_roc_random_forest()


# MODEL BULDING - Recursive Partitioning Tree
#-------------------------------------------

# HYPER TUNING OF PARAMETERS
cartask <- mlr::makeClassifTask(data = mutate_all(cardata,as.factor) , target =
"shouldBuy")
tree <- mlr::makeLearner("classif.rpart")
treeParamSpace <- makeParamSet(
    makeIntegerParam("minsplit", lower = 5, upper = 20),
    makeIntegerParam("minbucket", lower = 1, upper = 6),
```

```
    makeNumericParam("cp", lower = 0.01, upper = 0.1),
    makeIntegerParam("maxdepth", lower = 5, upper = 30))

# randSearch <- makeTuneControlRandom(maxit = 100)

gridSearchCV <- makeTuneControlGrid()

bayesCV <- makeTuneControlMBO()

cvForTuning <- makeResampleDesc("CV", iters = 10, stratify = TRUE)

tunedTreePars <- tuneParams(tree, task = cartask,
                            resampling = cvForTuning,
                            par.set = treeParamSpace,
                            control = gridSearchCV)

# bayes:  minsplit=18; minbucket=5; cp=0.0114; maxdepth=13
# minsplit = 17, minbucket = 4, cp = 0.01, maxdepth=16

model <- rpart(shouldBuy~.,method = "class",
             data = train.data,
             control = rpart.control(minsplit = 5, minbucket = 1, cp = 0.01, maxdepth =
27))

#PREDICT
pred <- predict(model, test.data, type = "class")

# plotcp(model)

plot(model, uniform = TRUE, margin = 0, main = "Original Tree")
text(model, use.n = TRUE, all = TRUE, cex = 0.5)

table(pred, cardata[-data.samples, ]$shouldBuy) #confusion matrix

#ACCURACY, SPECIFICITY, SENSITIVITY, ROC

confusionMatrix(pred, factor(cardata[-data.samples,]$shouldBuy))

# plot(roc(response = cardata[-data.samples,]$shouldBuy, predictor = factor(pred, ordered =
TRUE)))
# text(x = 0, y = 0.1, labels = paste("AUC =", auc(roc(response = cardata[-data.samples,
]$shouldBuy,predictor = factor(pred, ordered = TRUE)))))

multiclass_roc_rpart <- function(){
    lvls = levels(cardata$shouldBuy)

    aucs = c()
    plot(x=NA, y=NA, xlim=c(0,1), ylim=c(0,1),
         ylab='True Positive Rate',
         xlab='False Positive Rate',
         bty='n')

    for (type.id in 1:4) {
        type = as.factor(train.data$shouldBuy == lvls[type.id])

        rpart_model <- rpart(type~.,method = "class",
                             data = train.data[, -7],
                             control = rpart.control(minsplit = 5, minbucket = 1, cp =
0.01, maxdepth = 27))

        rpart_prediction = predict(rpart_model, test.data, type = "prob")

        score = rpart_prediction[, 'TRUE']
        actual.class = cardata[-data.samples, ]$shouldBuy == lvls[type.id]

        pred = prediction(score, actual.class)
        nbperf = ROCR::performance(pred, "tpr", "fpr")

        roc.x = unlist(nbperf@x.values)
        roc.y = unlist(nbperf@y.values)
        lines(roc.y ~ roc.x, col=type.id+1, lwd=2)

        nbauc = ROCR::performance(pred, "auc")
        nbauc = unlist(slot(nbauc, "y.values"))
        aucs[type.id] = nbauc
    }
```

```
    lines(x=c(0,1), c(0,1))

    text(x = 0.5, y = 0.1, labels = paste("Mean AUC =", round(mean(aucs), 5)))

}

multiclass_roc_rpart()
```

Appendix-C [Performance Analysis]

	Class: acc	Class: good	Class: unacc	Class: vgood
Sensitivity	0.9737	0.90909	0.9756	0.72727
Specificity	0.9627	0.99399	1.0000	1.00000
Pos Pred Value	0.8810	0.83333	1.0000	1.00000
Neg Pred Value	0.9923	0.99699	0.9423	0.99107
Prevalence	0.2209	0.03198	0.7151	0.03198
Detection Rate	0.2151	0.02907	0.6977	0.02326
Detection Prevalence	0.2442	0.03488	0.6977	0.02326
Balanced Accuracy	0.9682	0.95154	0.9878	0.86364

Table 11. Performance Assessment for random forest before hyperparameter tuning

	Class: acc	Class: good	Class: unacc	Class: vgood
Sensitivity	0.8947	0.90909	0.9472	0.72727
Specificity	0.9440	0.97898	0.9796	0.99700
Pos Pred Value	0.8193	0.58824	0.9915	0.88889
Neg Pred Value	0.9693	0.99694	0.8807	0.99104
Prevalence	0.2209	0.03198	0.7151	0.03198
Detection Rate	0.1977	0.02907	0.6773	0.02326
Detection Prevalence	0.2413	0.04942	0.6831	0.02616
Balanced Accuracy	0.9194	0.94403	0.9634	0.86213

Table 12. Performance Assessment for rpart before hyperparameter tuning

	Class: acc	Class: good	Class: unacc	Class: vgood
Sensitivity	0.8289	1.00000	0.9634	1.00000
Specificity	0.9701	0.97898	0.9388	0.99700
Pos Pred Value	0.8873	0.61111	0.9753	0.91667
Neg Pred Value	0.9524	1.00000	0.9109	1.00000
Prevalence	0.2209	0.03198	0.7151	0.03198
Detection Rate	0.1831	0.03198	0.6890	0.03198
Detection Prevalence	0.2064	0.05233	0.7064	0.03488
Balanced Accuracy	0.8995	0.98949	0.9511	0.99850

Table 13. Performance Assessment for rpart after hyperparameter tuning

	Class: acc	Class: good	Class: unacc	Class: vgood
Sensitivity	0.9868	1.00000	1.0000	0.90909
Specificity	0.9963	0.99700	1.0000	1.00000
Pos Pred Value	0.9868	0.91667	1.0000	1.00000
Neg Pred Value	0.9963	1.00000	1.0000	0.99701
Prevalence	0.2209	0.03198	0.7151	0.03198
Detection Rate	0.2180	0.03198	0.7151	0.02907
Detection Prevalence	0.2209	0.03488	0.7151	0.02907
Balanced Accuracy	0.9916	0.99850	1.0000	0.95455

Table 14. Performance Assessment for random forest after hyperparameter tuning