Car Evaluation Machine learning Project

Mohammed Dahesh Ali

7/10/2021

1 Introduction

In this project, several machine learning techniques were applied for a multiclass classification problem. Particularly, the used dataset was for car evaluation purposes which is available from UCI Machine Learning Repository.

The Machine learning problem in this project is to evaluate a given car in terms of Not-accepted, Accepted, Good, and Very good according to six attributes. This machine learning project followed a template consisting several steps that are commonly applied in any ML project. Those steps are as follows:

1. Problem preparation includes the installation and loading the required r language packages; importing the dataset; and partitioning of the dataset into train and validation datasets.
2. Understanding the dataset using both statistical and visualization techniques.
3. According to the previous step and satisfying the assumptions of the machine learning algorithm that are candidates for the given problem, there will be some data preprocessing steps such as feature engineering, data cleaning, and others.
4. Evaluate several machine learning algorithms using a resampling technique and an evaluation metric.
5. Then, selection of the best performant algorithms and trying to improve their accuracy.
6. Finally, model finalizing and make predictions using the validation dataset.

The goals of this project are: first, to apply machine learning techniques that go beyond linear regression. Second, to apply the steps for any machine learning project. Third, to achieve the best predictive modeling performance in the chosen ML problem.

2 Data Collection

This project used a dataset called Car Evaluation. It can be downloaded from the link: (https://archive.ics.uci.edu/ml/datasets/Car+Evaluation). This dataset was generated from a hierarchical decision model besides it is commonly used to validate constructive induction. This dataset is dedicated to classification tasks especially multi classification. In addition, it consists of 6 categorical attributes besides the class attribute. On the other hand, it contains 1728 data instances, and there are no missing values in this dataset. Importantly, this dataset was used in many previous research projects. The following bullets show attribute name, and attribute values respectively.

1. Buying_price (input): v-high, high, med, low
2. Maintenance_price (input): v-high, high, med, low

3. Number_of_doors (input): 2, 3, 4, 5-more
4. Persons_capacity (input): 2, 4, more
5. Luggage_boot_size (input): small, med, big
6. Safety (input): low, med, high
7. Class (output): unacceptable, acceptable, good, very good.

3 Methodology

This section presents the general structure used through any predictive ML projects in R. It illustrates the implementation details for each step in this structure.

3.1 ML Problem Preparation

First, this project needed several R libraries that were required to accomplish some tasks through the project. For this reason, the code started by installing those packages using the following code:

```
#install.packages("glmnet")
#install.packages("e1071")
#install.packages("klaR")
#install.packages("kernlab")
#install.packages("ellipse")
#install.packages("randomForest")
```

Also in this step, the required R packages were loaded. In addition, the car evaluation dataset was loaded and attribute names were attached using the following R code:

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unable to identify current timezone 'W':
## please set environment variable 'TZ'
```

```
# Loading the dataset, Please draw attention to the given path for the dataset.
dataset_filename <- "C:\\edx\\Car_dataset.csv"
# load the dataset file in csv format.

car_dataset <- read.csv(dataset_filename, header = FALSE)


car_dataset[, 7] <-  as.factor(car_dataset[, 7])


# name the columns of the dataset.
colnames(car_dataset) <-
  c(
```

```
    "buying_price",
    "maintenance_price",
    "Number_of_doors",
    "persons_capacity",
    "lug_boot_size",
    "safety",
    "class"
  )
```

Note: The dataset file should be stored in the path shown in the above code or the code should be updated to reflect the dataset location.

After that, the predictive models that were created in this project needed to estimate their accuracy using some statistical methods. For this end, to achieve confidence those models were evaluated using unseen data. Thus, the entire car evaluation dataset splitted into two parts : 80% to be used for model training and evaluating (called edx in this project) and 20% as unseen data for validation (called validation in this project). Actually, this was done using the following R code:

```
# Divide the dataset into 80% for modelling (training and testing), and 20% for validation.
set.seed(5)
holdoutIndex <-
  createDataPartition(car_dataset$class, p = 0.80, list = FALSE)
validation <- car_dataset[-holdoutIndex, ]
edx <- car_dataset[holdoutIndex, ]
```

More importantly, data encoding processes were applied to the original dataset before it was loaded. The goal of this step was to convert the dataset into numeric attributes so that it can be modelled using machine learning techniques. Data encoding is as followed:

1. Buying_price (input): v-high, high, med, and low were changed to 4,3,2, and 1 respectively.
2. Maintenance_price (input): v-high, high, med, and low were changed to 4,3,2, and 1 respectively.
3. Number_of_doors (input): 2, 3, 4, and 5-more were changed to 2,3,4, and 5 respectively.
4. Persons_capacity (input): 2, 4, and more were changed to 2,4, and 5 respectively.
5. Luggage_boot_size (input): small, med, and big were changed to 1,2, and 3 respectively.
6. Safety (input): low, med, and high were changed to 1,2, and 3 respectively.
7. Class (output): unacceptable, acceptable, good, and very good. This attribute was converted into a factor datatype

3.2 Data Understanding

This step is considered very important because selection of preprocessing techniques or the training algorithms depends on how much the dataset is understood. For example, a good understanding of the data leads to correctly deciding which data cleaning, or data transformation techniques are needed. In addition, ML learning algorithms in general have some data assumptions and conditions to be met when they are applied. So understanding the given data helps in selecting the suitable algorithms. Finally, this leads to better predictive results.

3.2.1 Statistical Analysis

This first insight about the dataset was to know how many data instances did the dataset contain and how many features it contained using the following code:

```
 # Get the dimensions of the dataset.
dim(edx)
```

```
## [1] 1384    7
```

Second, it is good to examine some actual records from the dataset. This was done using the following code to see the first 20 instances:

```
# Have a look to the dataset
head(edx, n = 20)
```

```
##    buying_price maintenance_price Number_of_doors persons_capacity
## 1             4                 4               2                 2
## 5             4                 4               2                 2
## 6             4                 4               2                 2
## 7             4                 4               2                 2
## 8             4                 4               2                 2
## 9             4                 4               2                 2
## 10            4                 4               2                 4
## 11            4                 4               2                 4
## 12            4                 4               2                 4
## 13            4                 4               2                 4
## 14            4                 4               2                 4
## 15            4                 4               2                 4
## 16            4                 4               2                 4
## 17            4                 4               2                 4
## 20            4                 4               2                 5
## 21            4                 4               2                 5
## 23            4                 4               2                 5
## 24            4                 4               2                 5
## 25            4                 4               2                 5
## 26            4                 4               2                 5
##    lug_boot_size safety class
## 1              1      1 unacc
## 5              2      2 unacc
## 6              2      3 unacc
## 7              3      1 unacc
## 8              3      2 unacc
## 9              3      3 unacc
## 10             1      1 unacc
## 11             1      2 unacc
## 12             1      3 unacc
## 13             2      1 unacc
## 14             2      2 unacc
## 15             2      3 unacc
## 16             3      1 unacc
## 17             3      2 unacc
## 20             1      2 unacc
```

```
## 21              1      3 unacc
## 23              2      2 unacc
## 24              2      3 unacc
## 25              3      1 unacc
## 26              3      2 unacc
```

Third, it is important to know the data types of the dataset attributes such as numeric, characters, factors, and others as it helps in doing the corrected data preparation prior to modelling. Actually, this was accomplished using the following R code:

```
# Check data types of the dataset attributes.
sapply(edx, class)
```

```
##      buying_price maintenance_price   Number_of_doors  persons_capacity
##         "integer"         "integer"         "integer"         "integer"
##     lug_boot_size            safety             class
##         "integer"         "integer"          "factor"
```

Fourth, it could be a good idea to have a statistical summary for each attribute in terms of minimum , 25% quartile, median, mean, 75% quartile, and maximum value. Simply, this was achieved using the following R code:

```
# Summarize the dataset
summary(edx)
```

```
##   buying_price  maintenance_price Number_of_doors persons_capacity
##  Min.   :1.00   Min.   :1.000     Min.   :2.000   Min.   :2.000
##  1st Qu.:1.00   1st Qu.:1.000     1st Qu.:3.000   1st Qu.:2.000
##  Median :2.00   Median :3.000     Median :3.000   Median :4.000
##  Mean   :2.49   Mean   :2.501     Mean   :3.498   Mean   :3.668
##  3rd Qu.:3.00   3rd Qu.:4.000     3rd Qu.:4.000   3rd Qu.:5.000
##  Max.   :4.00   Max.   :4.000     Max.   :5.000   Max.   :5.000
##  lug_boot_size       safety        class
##  Min.   :1.000   Min.   :1.000   acc  :308
##  1st Qu.:1.000   1st Qu.:1.000   good : 56
##  Median :2.000   Median :2.000   unacc:968
##  Mean   :1.991   Mean   :2.014   vgood: 52
##  3rd Qu.:3.000   3rd Qu.:3.000
##  Max.   :3.000   Max.   :3.000
```

Fifth, As the class attribute has a factor datatype, it's levels can be shown using the following code:

```
 # get multiple class labels
levels(edx$class)
```

```
## [1] "acc"   "good"  "unacc" "vgood"
```

Sixth, then knowing how many data instances do they belong to each level of the class attribute would be important. Actually, this was done using the following code:

```
# check class distribution
cbind(freq = table(edx$class),
      percentage = prop.table(table(edx$class)) * 100)
```

```
##       freq percentage
## acc    308  22.254335
## good    56   4.046243
## unacc  968  69.942197
## vgood   52   3.757225
```

Seventh, the last but not the least, we looked to the interactions between the six attributes using the following code:

```
 # check correlations between input attributes
cases <- complete.cases(edx)
cor(edx[cases, 1:6])
```

```
##                   buying_price maintenance_price Number_of_doors
## buying_price       1.000000000     -0.0034447201     0.007258790
## maintenance_price -0.003444720      1.0000000000     0.002615908
## Number_of_doors    0.007258790      0.0026159081     1.000000000
## persons_capacity   0.009295700     -0.0056280030    -0.009181420
## lug_boot_size     -0.004053129      0.0071197260    -0.008797181
## safety             0.011228838     -0.0008115805    -0.009530318
##                   persons_capacity lug_boot_size        safety
## buying_price           0.009295700  -0.004053129  0.0112288379
## maintenance_price     -0.005628003   0.007119726 -0.0008115805
## Number_of_doors       -0.009181420  -0.008797181 -0.0095303182
## persons_capacity       1.000000000   0.012857328 -0.0031128030
## lug_boot_size          0.012857328   1.000000000  0.0131917756
## safety                -0.003112803   0.013191776  1.0000000000
```

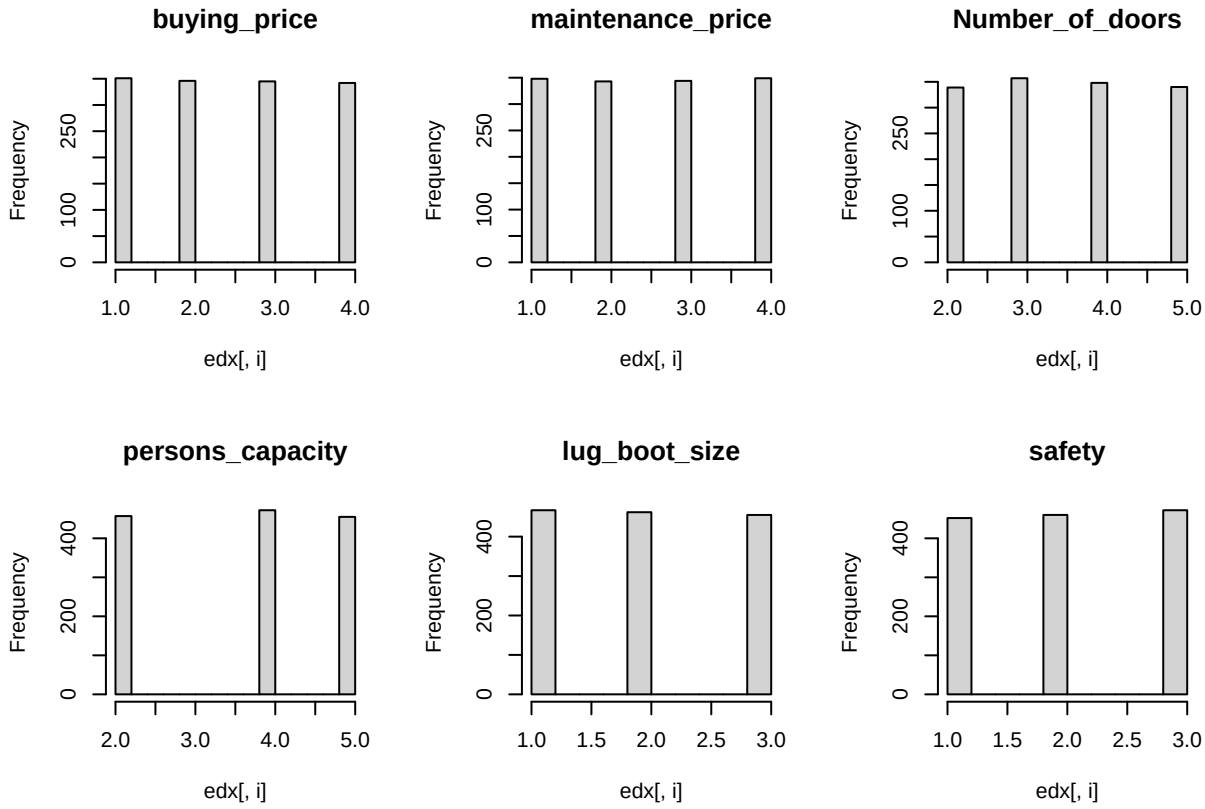
3.2.2 Data Visualization

Charts and plots are one of the most important methods to understand the data before machine learning modelling. Also, it is the fastest and the most effective way to get better understanding. For example, it shows data distribution, outliers, and the interactions between attributes. In particular, there are two types of data visualization techniques that could be utilized in any machine learning project. There are univariate plots that are used to show each attribute separately; and Multivariate plots that are used to show the correlations between attributes.

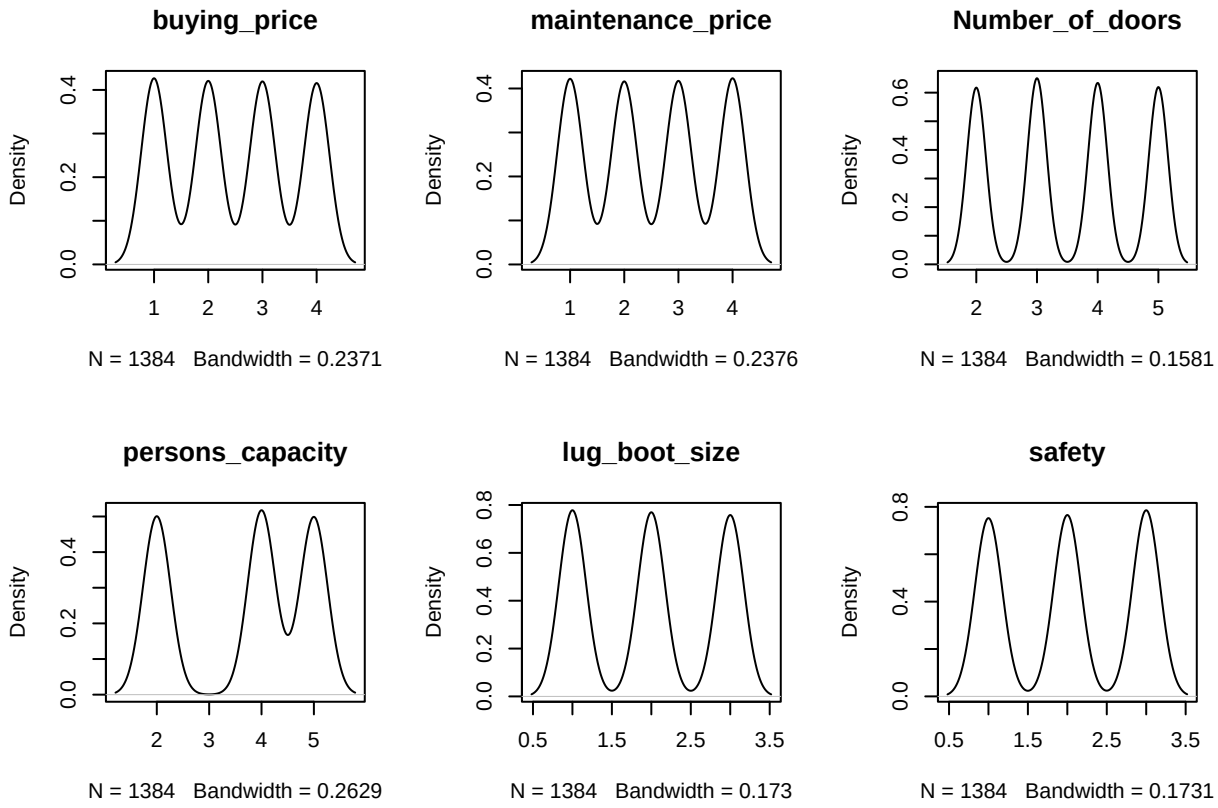
The first R code snippet calculated histogram plots for each attribute. It is observed from the model that all attribute distributions are similar but with different ranges. The second R code snippet showed density plots for each attribute. Plots proved that the data distribution for each attribute is not gaussian. So it could mean to normalize them. The third R code snippet was used to show box plots. The results stated that the data range is different for each attribute besides the data is not standardized. The fourth R code yielded bar plots to show the interactions between attributes and how they are grouped to each class. This type was used because the attributes are discrete.

```
# show histograms for each input variable
par(mfrow = c(2, 3))
for (i in 1:6) {
```

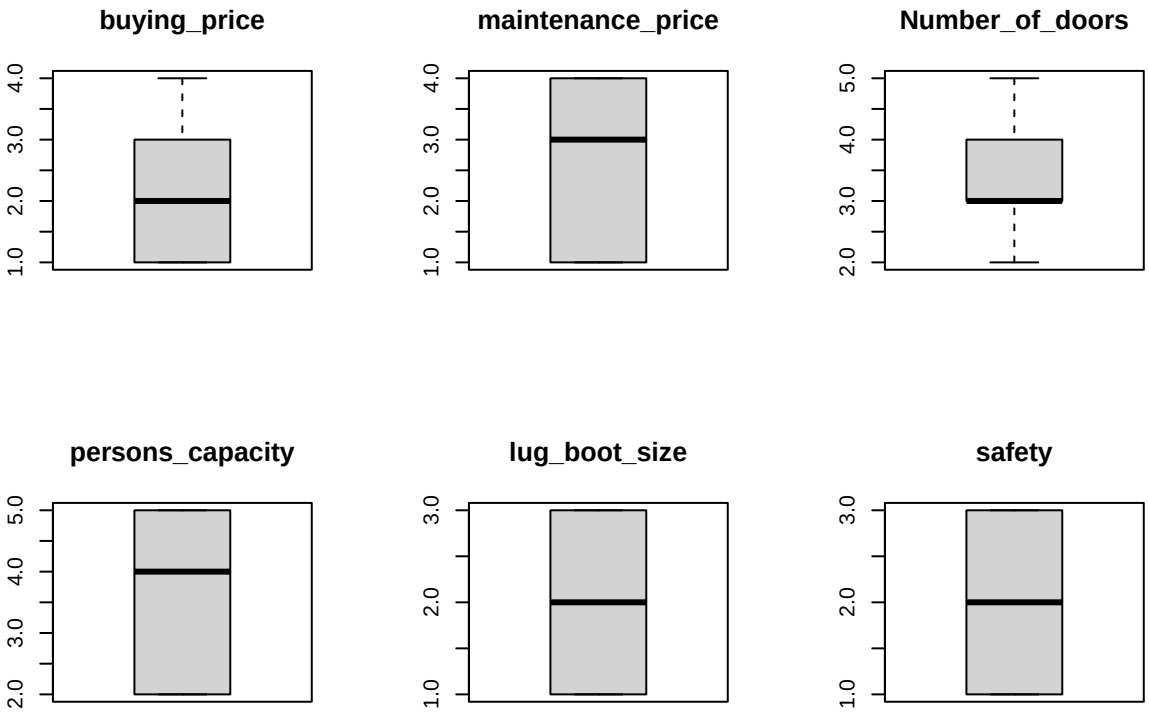
```
  hist(edx[, i], main = names(edx)[i])
}
```



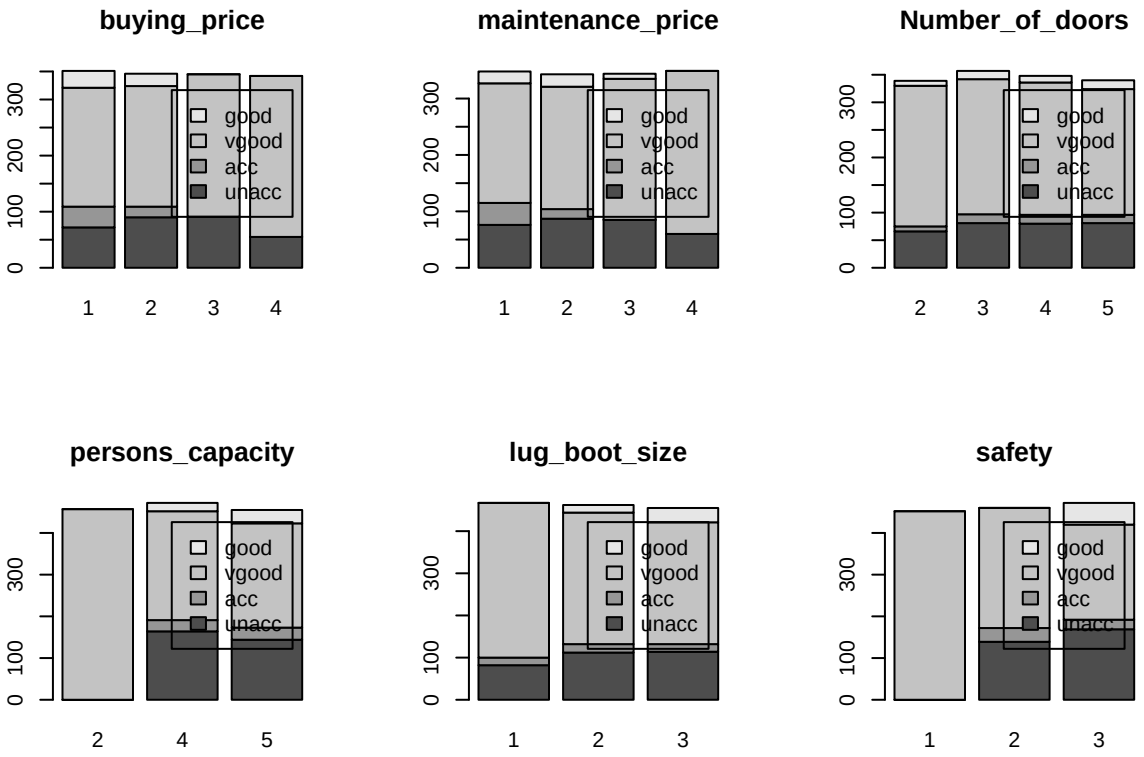

```
# show density plot for each input variable
par(mfrow = c(2, 3))
cases <- complete.cases(edx)
for (i in 1:6) {
  plot(density(edx[cases, i]), main = names(edx)[i])
}
```




```
# show box plots for each input variable
par(mfrow = c(2, 3))
for (i in 1:6) {
  boxplot(edx[, i], main = names(edx)[i])
}
```




```
# show bar plots of each attribute by class
par(mfrow = c(2, 3))
for (i in 1:6) {
  barplot(
    table(edx$class, edx[, i]),
    main = names(edx)[i],
    legend.text = unique(edx$class)
  )
}
```



3.3 Data Pre-processing

According to the previous step, it was seen that the data distributions for each attribute is not normal, So that this could influence the predictive power for some training algorithms. Consequently, This project used a data transformation method called Box-Cox that is used to transform data to a nearly normal distribution. Accordingly, all of the training algorithms used in the next step employed this method during training. Importantly, this method was also applied to the validation dataset.

3.4 Algorithm Selection and Evaluation

3.4.1 k-fold cross validation

This project used a technique called k-fold cross validation as a resampling technique. This technique splits the data (edx in our case) into k sub datasets. Each sub dataset is used for testing one time and used for training k-1 times. It is an effective method that is commonly used when the overall dataset is relatively small. This technique ensures that each data sample appears one time in testing and k-1 times in training. In this project, k value was10 and the technique itself was repeated 5 times and the average performance was reported.

3.4.2 Accuracy and Kappa

In the R caret package, accuracy and kappa are used as evaluation metrics to assess how good the ML model is. Accuracy is the correctly classified data instances over all instances. It may be less useful with

multiclass classification and with imbalance data. Kappa is useful as a metric when the data is imbalanced as it depends on randomness.

The following R code shows the implementation of k-fold cross validation in this project:

```
## Use 10-fold cross validation with 5 repeats as resampling technique
train_resampling <-
  trainControl(method = "repeatedcv",
               number = 10,
               repeats = 5)
evaluation_metric <- "Accuracy"
```

Machine learning is an empirical science which means it totally depends on data. As we have no idea about which algorithm will achieve the best performance, this project tested several algorithms to finally select the best one. We used the following machine learning algorithms with the default configurations and hyperparameters:

1. Linear Discriminant Analysis (LDA): linear algorithm.
2. Regularized Logistic Regression (GLMNE): linear algorithm.
3. k-Nearest Neighbors (KNN): non-linear algorithm.
4. Classification and Regression Trees (CART): non-linear algorithm.
5. Support Vector Machine (SVM) : non-linear algorithm.
6. Random Forest (RF): non-linear algorithm.

Note: the random seed number was reset to 5 before each algorithm to train each algorithm using the same data split options.

```
## Use 10-fold cross validation with 5 repeats as resampling technique
train_resampling <-
  trainControl(method = "repeatedcv",
               number = 10,
               repeats = 5)
evaluation_metric <- "Accuracy"

# Linear Discriminant Analysis
set.seed(5)
fit.lda <-
  train(
    class ~ .,
    data = edx,
    method = "lda",
    metric = evaluation_metric,
    preProc = c("BoxCox"),
    trControl = train_resampling
  )

# Regularized Logistic Regression
set.seed(5)
fit.glmnet <-
  train(
```

```
    class ~ .,
    data = edx,
    method = "glmnet",
    metric = evaluation_metric,
    preProc = c("BoxCox"),
    trControl = train_resampling
  )


# k-Nearest Neighbors
set.seed(5)
fit.knn <-
  train(
    class ~ .,
    data = edx,
    method = "knn",
    metric = evaluation_metric,
    preProc = c("BoxCox"),
    trControl = train_resampling
  )


# Classification and Regression Trees
set.seed(5)
fit.cart <-
  train(
    class ~ .,
    data = edx,
    method = "rpart",
    metric = evaluation_metric,
    preProc = c("BoxCox"),
    trControl = train_resampling
  )


# Support Vector Machine
set.seed(5)
fit.svm <-
  train(
    class ~ .,
    data = edx,
    method = "svmRadial",
    metric = evaluation_metric,
    preProc = c("BoxCox"),
    trControl = train_resampling
  )

# Random Forest
set.seed(5)
fit.rf <-
  train(
    class ~ .,
    data = edx,
```

```
    method = "rf",
    metric = evaluation_metric,
    preProc = c("BoxCox"),
    trControl = train_resampling
  )
```

3.4.3 Algorithm Comparison

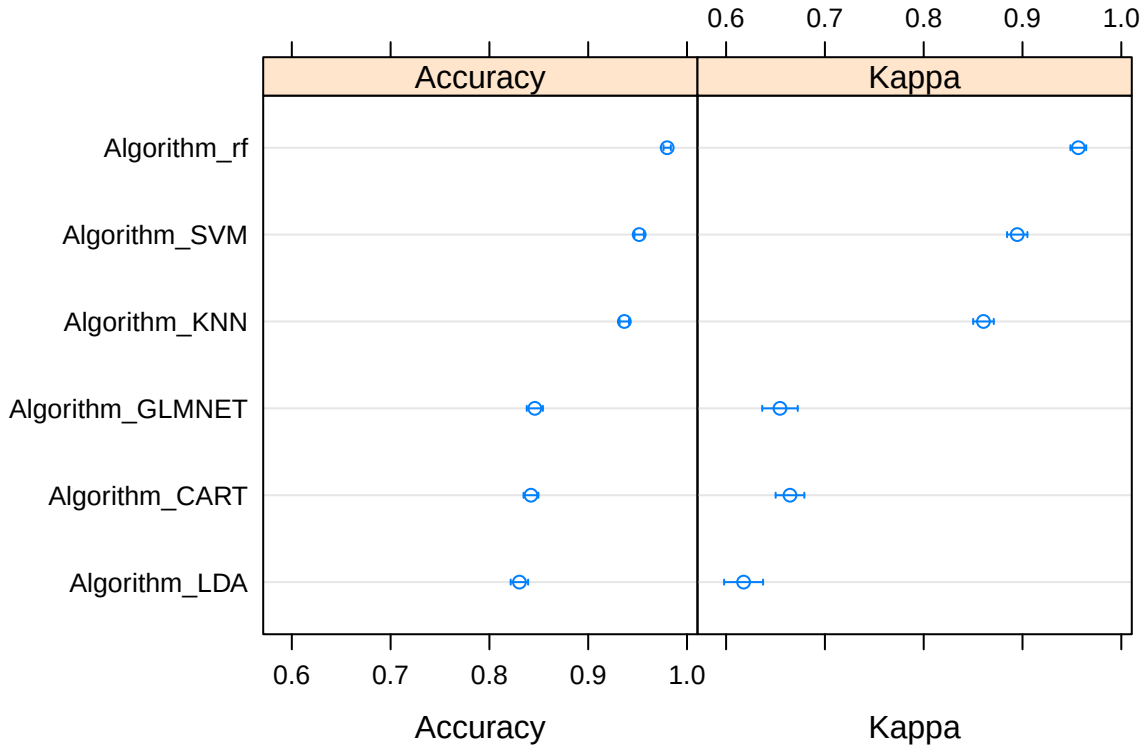
The R code below shows comparison results among the six machine learning algorithms. As the data is considered imbalanced according to statistics and visualization methods, we will depend on Kappa metric instead of accuracy to evaluate the above training algorithms:

```
#   algorithm Comparison
training_results <-
  resamples(
    list(
      Algorithm_LDA = fit.lda,
      Algorithm_GLMNET = fit.glmnet,
      Algorithm_KNN = fit.knn,
      Algorithm_CART = fit.cart,
      Algorithm_SVM = fit.svm,
      Algorithm_rf = fit.rf
    )
  )
summary(training_results)
```

```
##
## Call:
## summary.resamples(object = training_results)
##
## Models: Algorithm_LDA, Algorithm_GLMNET, Algorithm_KNN, Algorithm_CART, Algorithm_SVM, Algorithm_rf
## Number of resamples: 50
##
## Accuracy
##                       Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## Algorithm_LDA    0.7753623 0.8115942 0.8260870 0.8303524 0.8454682 0.9214286
## Algorithm_GLMNET 0.7841727 0.8324209 0.8483735 0.8459623 0.8558544 0.9428571
## Algorithm_KNN    0.8992806 0.9208633 0.9352518 0.9365648 0.9496403 0.9640288
## Algorithm_CART   0.7625899 0.8273381 0.8381470 0.8420634 0.8561151 0.8992806
## Algorithm_SVM    0.9136691 0.9420290 0.9496403 0.9515930 0.9637681 0.9927007
## Algorithm_rf     0.9424460 0.9730567 0.9854015 0.9799087 0.9856115 1.0000000
##                  NA's
## Algorithm_LDA       0
## Algorithm_GLMNET    0
## Algorithm_KNN       0
## Algorithm_CART      0
## Algorithm_SVM       0
## Algorithm_rf        0
##
## Kappa
##                       Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## Algorithm_LDA    0.4784199 0.5780031 0.6007638 0.6177243 0.6433613 0.8287367
```

```
## Algorithm_GLMNET 0.5131349 0.6200673 0.6563221 0.6545329 0.6760798 0.8739449
## Algorithm_KNN    0.7786121 0.8274235 0.8595684 0.8605044 0.8907340 0.9215930
## Algorithm_CART   0.5441264 0.6427987 0.6589539 0.6646801 0.6951350 0.7819364
## Algorithm_SVM    0.8135688 0.8729142 0.8917246 0.8946618 0.9203928 0.9840827
## Algorithm_rf     0.8785363 0.9422388 0.9681527 0.9564997 0.9692503 1.0000000
##                  NA's
## Algorithm_LDA       0
## Algorithm_GLMNET    0
## Algorithm_KNN       0
## Algorithm_CART      0
## Algorithm_SVM       0
## Algorithm_rf        0
```

```
dotplot(training_results)
```



Confidence Level: 0.95

```
# print the Best Model
print(fit.rf)
```

```
## Random Forest
##
## 1384 samples
##    6 predictor
##    4 classes: 'acc', 'good', 'unacc', 'vgood'
##
## Pre-processing: Box-Cox transformation (6)
```

```
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1247, 1245, 1246, 1247, 1245, 1244, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9722682  0.9403171
##   4     0.9765795  0.9495803
##   6     0.9799087  0.9564997
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

We can see that the best performing algorithm is Random Forest. It archives (96%) on Kappa metric and 98% on accuracy metric. Then, the Support Vector Machine algorithm is coming next as it achieves 89% and 95% respectively.

Results and Model Finalizing

Finalizing the most accurate model means to test it and make predictions on unseen data particularly using the validation dataset that was firstly splitted out. This method ensures avoiding many problems such as overfitting As it was seen in the previous section, the winner model was the random forest with its default hyperparameters. As it was seen in previous sections, there were some data preparation or transformations applied on training data (edx) such as data encoding, normalization, and standardization. So, the first important step in this section is to make those data preprocessing operations on the hold-out validation dataset (validation). Actually, this was performed using the following R code:

```
# doing data transform parameters
set.seed(5)

edxx <- edx[, 1:6]
Params <- preProcess(edxx, method = c("BoxCox"))
x <- predict(Params, edxx)

validationv <- predict(Params, validation[, 1:6])
```

Using the following R code, we can know how accurate the Random Forest model is on unseen data during training by generating what is called a confusion matrix. Actually, the results were 97% and 94% using accuracy and Kappa metrics respectively.

```
# make predictions On unseen data
set.seed(5)
preds <- predict(fit.rf, validation)
confusionMatrix(preds, validation$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction acc good unacc vgood
##      acc    71    2     1     2
##      good    5   11     0     0
##      unacc   0    0   241     0
##      vgood   0    0     0    11
```

```
##
## Overall Statistics
##
##                Accuracy : 0.9709
##                  95% CI : (0.9472, 0.986)
##     No Information Rate : 0.7035
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9362
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: acc Class: good Class: unacc Class: vgood
## Sensitivity              0.9342     0.84615       0.9959      0.84615
## Specificity              0.9813     0.98489       1.0000      1.00000
## Pos Pred Value           0.9342     0.68750       1.0000      1.00000
## Neg Pred Value           0.9813     0.99390       0.9903      0.99399
## Prevalence               0.2209     0.03779       0.7035      0.03779
## Detection Rate           0.2064     0.03198       0.7006      0.03198
## Detection Prevalence     0.2209     0.04651       0.7006      0.03198
## Balanced Accuracy        0.9578     0.91552       0.9979      0.92308
```

Finally, for production use and model deployment purposes, this model (Random Forest) may be retrained using the same configuration on all the available data. Then, saving the model to make predictions on new production data.

Conclusion

In this project, we have developed a machine learning model to make predictions on car evaluation dataset in the aim of correctly and accurately predict a decision of how acceptable to buy a car according to six data attributes. This project compared among six developed models using accuracy and Kappa evaluation metrics. The results were toward the Random forest model which achieved a competitive performance. Generally, non-linear models outperformed linear models in the current problem.

The used dataset was relatively small and with good quality. So, we recommend future works to apply our approach using new datasets which are larger in size and related to new data projects. Moreover, we recommend tuning the hyperparameters of our models, especially Random Forest and KNN. Also, using new algorithms for training such as neural networks would be preferable and worth experimenting.