Decision Trees and Random Forests to Predict the Price Range of Mobile Phones

Alejandro Zarazo

30/5/2021

1. General Description

The dataset that will be used is available in any of the following addresses:

https://www.kaggle.com/iabhishekofficial/mobile-price-classification/download

https://github.com/azarazo/CYO_Project_Phone_Price/blob/main/train.csv

The data about mobile phones, registered in the mentioned dataset, will be used to estimate their price based on their characteristics. The fundamental idea is to know the relationship between the phones' characteristics (e.g., internal memory, number of nuclei, size of the LCD display, etc.) to determine the selling price of the product, based on a segmentation or stratification of the potential clients, so each phone can be offered to each segment in a more efficient way. We have estimated a referential segmentation that includes prices: * Low * Medium * High * Very High

For this reason, the challenge will be to predict the price class as a function of the characteristics of each device. Therefore, this investigation is structured in the following way:

First, the main idea of this work is presented. Second, a data preparation, conditioning and tidying process is carried out on the studied dataset. Third, an exploratory data analysis is performed to posteriorly propose and automatic learning algorithm that allows to make predictions that place a determined phone in its corresponding price segment, based on the available historical data. Finally, a discussion about the results is done and final observations and conclusions are presented.

1.1. Introduction

Segmenting the prices of specific products allows companies to categorize the type of client or user to whom the products can be offered, depending on the specifications and characteristics of those goods. Additionally, marketing and publicity campaigns can be carried out according to the objective client segment, what allows organizations to be more competitive and to better target the users bases depending on their specific needs, habits and expectations.

1.2. Objective of the Project

As it has been previously stated, the objective of this data science project is to develop a Machine Learning algorithm capable of training, testing and applying the selected technique to predict the price range that will correspond to a specific mobile phone based on the device's characteristics. For this purpose, we will use the provided data to assign the price segment using a validation dataset in the algorithm. Specific metrics will be used to evaluate the performance of the proposed algorithm, such as Root Mean Square Error (RMSE) and Precision.

2. Dataset

The dataset to be used will be downloaded from the following link:

https://github.com/drrueda/DataSets/archive/refs/heads/main.zip

```
# Downloading and charging the dataset
url <- "https://github.com/drrueda/DataSets/archive/refs/heads/main.zip"
download.file(url,"temp.zip", mode="wb")
unzip_result <- unzip("temp.zip", exdir = "data", overwrite = TRUE)
mobile <- read.csv(unzip_result)
```

Description of the data

The following is a description of each variable composing the dataset:

- battery_power: Total energy a battery can store in one time measured in mAh
- blue: Has bluetooth or not
- clock_speed: speed at which microprocessor executes instructions
- dual_sim: Has dual sim support or not
- fc: Front camera mega pixels
- four_g: Has 4G or not
- int_memory: Internal Memory in Gigabytes
- m_dep: Mobile Depth in cm
- mobile_wt: Weight of mobile phone
- n_cores: Number of cores of processor
- pc: Primary Camera mega pixels
- px_height: Pixel Resolution Height
- px_width: Pixel Resolution Width
- ram: Random Access Memory in Megabytes
- sc_h: Screen Height of mobile in cm
- sc_w: Screen Width of mobile in cm
- talk_time: longest time that a single battery charge will last when you are calling
- three_g: Has 3G or not
- touch_screen: Has touch screen or not
- wifi: Has wifi or not
- price_range: Range of price. This is the objective variable.

The following is a preview of the dataset to better understand it:

```
#Display the head of the mobile dataset
head(mobile,6)
```

```
##   battery_power blue clock_speed dual_sim fc four_g int_memory m_dep mobile_wt
## 1           842    0         2.2        0  1      0          7   0.6       188
## 2          1021    1         0.5        1  0      1         53   0.7       136
## 3           563    1         0.5        1  2      1         41   0.9       145
## 4           615    1         2.5        0  0      0         10   0.8       131
```

```
## 5          1821    1         1.2        0 13       1          44   0.6        141
## 6          1859    0         0.5        1  3       0          22   0.7        164
##   n_cores pc px_height px_width  ram sc_h sc_w talk_time three_g touch_screen
## 1       2  2        20      756 2549    9    7        19       0            0
## 2       3  6       905     1988 2631   17    3         7       1            1
## 3       5  6      1263     1716 2603   11    2         9       1            1
## 4       6  9      1216     1786 2769   16    8        11       1            0
## 5       2 14      1208     1212 1411    8    2        15       1            1
## 6       1  7      1004     1654 1067   17    1        10       1            0
##   wifi price_range
## 1    1           1
## 2    0           2
## 3    0           2
## 4    0           2
## 5    0           1
## 6    0           1
```

3. Methods and Analysis

3.1. Data Exploration, Cleaning and Visualization

3.1.1. Exploration

The structure of the data can be viewed here:

```r
#  To explore the variables
glimpse(mobile)
```

```
## Rows: 2,000
## Columns: 21
## $ battery_power <int> 842, 1021, 563, 615, 1821, 1859, 1821, 1954, 1445, 509, ~
## $ blue          <int> 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1,~
## $ clock_speed   <dbl> 2.2, 0.5, 0.5, 2.5, 1.2, 0.5, 1.7, 0.5, 0.5, 0.6, 2.9, 2~
## $ dual_sim      <int> 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1,~
## $ fc            <int> 1, 0, 2, 0, 13, 3, 4, 0, 0, 2, 0, 5, 2, 7, 13, 3, 1, 7, ~
## $ four_g        <int> 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,~
## $ int_memory    <int> 7, 53, 41, 10, 44, 22, 10, 24, 53, 9, 9, 33, 33, 17, 52,~
## $ m_dep         <dbl> 0.6, 0.7, 0.9, 0.8, 0.6, 0.7, 0.8, 0.8, 0.7, 0.1, 0.1, 0~
## $ mobile_wt     <int> 188, 136, 145, 131, 141, 164, 139, 187, 174, 93, 182, 17~
## $ n_cores       <int> 2, 3, 5, 6, 2, 1, 8, 4, 7, 5, 5, 8, 4, 4, 1, 2, 8, 3, 5,~
## $ pc            <int> 2, 6, 6, 9, 14, 7, 10, 0, 14, 15, 1, 18, 17, 11, 17, 16,~
## $ px_height     <int> 20, 905, 1263, 1216, 1208, 1004, 381, 512, 386, 1137, 24~
## $ px_width      <int> 756, 1988, 1716, 1786, 1212, 1654, 1018, 1149, 836, 1224~
## $ ram           <int> 2549, 2631, 2603, 2769, 1411, 1067, 3220, 700, 1099, 513~
## $ sc_h          <int> 9, 17, 11, 16, 8, 17, 13, 16, 17, 19, 5, 14, 18, 7, 14, ~
## $ sc_w          <int> 7, 3, 2, 8, 2, 1, 8, 3, 1, 10, 2, 9, 0, 1, 9, 15, 9, 2, ~
## $ talk_time     <int> 19, 7, 9, 11, 15, 10, 18, 5, 20, 12, 7, 13, 2, 4, 3, 11,~
## $ three_g       <int> 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,~
## $ touch_screen  <int> 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,~
## $ wifi          <int> 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,~
## $ price_range   <int> 1, 2, 2, 2, 1, 1, 3, 0, 0, 0, 3, 3, 1, 2, 0, 0, 3, 3, 1,~
```

There are variables that need to be adapted so that they can be used as input for the model. These variable are the following:

1. *blue* : is an integer and must be converted to factor

2. *dual_sim* : is an integer and must be converted to factor
3. *four_g* : is an integer and must be converted to factor
4. *three_g* : is an integer and must be converted to factor
5. *touch_screen* : is an integer and must be converted to factor
6. *wifi* : is an integer and must be converted to factor
7. *price_range* : is an integer and must be converted to factor, with leves **low cost**, **medium cost**, **high cost**, **very high cost**

Let's apply these transformations:

```r
# Transform the variables that are categorical
mobile <- mobile %>%
  mutate(
    blue = as.factor(blue),
    dual_sim  = as.factor(dual_sim),
    four_g = as.factor(four_g),
    three_g = as.factor(three_g),
    touch_screen = as.factor(touch_screen),
    wifi = as.factor(wifi),
    price_range = as.factor(price_range),
    # Categorical values that the price variable can assume
    price_range = sapply(price_range,
                         switch,"low cost","medium cost",
                         "high cost","very high cost"),
    # Order in which the price range must appear
    price_range = ordered(price_range,
                          levels=c("low cost","medium cost", "high cost","very high cost"))
  )
# We see how the variables are after transforming to factor
str(mobile)
```

```
## 'data.frame':    2000 obs. of  21 variables:
##  $ battery_power: int  842 1021 563 615 1821 1859 1821 1954 1445 509 ...
##  $ blue         : Factor w/ 2 levels "0","1": 1 2 2 2 2 1 1 1 2 2 ...
##  $ clock_speed  : num  2.2 0.5 0.5 2.5 1.2 0.5 1.7 0.5 0.5 0.6 ...
##  $ dual_sim     : Factor w/ 2 levels "0","1": 1 2 2 1 1 2 1 2 1 2 ...
##  $ fc           : int  1 0 2 0 13 3 4 0 0 2 ...
##  $ four_g       : Factor w/ 2 levels "0","1": 1 2 2 1 2 1 2 1 1 2 ...
##  $ int_memory   : int  7 53 41 10 44 22 10 24 53 9 ...
##  $ m_dep        : num  0.6 0.7 0.9 0.8 0.6 0.7 0.8 0.8 0.7 0.1 ...
##  $ mobile_wt    : int  188 136 145 131 141 164 139 187 174 93 ...
##  $ n_cores      : int  2 3 5 6 2 1 8 4 7 5 ...
##  $ pc           : int  2 6 6 9 14 7 10 0 14 15 ...
##  $ px_height    : int  20 905 1263 1216 1208 1004 381 512 386 1137 ...
##  $ px_width     : int  756 1988 1716 1786 1212 1654 1018 1149 836 1224 ...
##  $ ram          : int  2549 2631 2603 2769 1411 1067 3220 700 1099 513 ...
##  $ sc_h         : int  9 17 11 16 8 17 13 16 17 19 ...
##  $ sc_w         : int  7 3 2 8 2 1 8 3 1 10 ...
##  $ talk_time    : int  19 7 9 11 15 10 18 5 20 12 ...
##  $ three_g      : Factor w/ 2 levels "0","1": 1 2 2 2 2 2 2 2 2 2 ...
##  $ touch_screen : Factor w/ 2 levels "0","1": 1 2 2 1 2 1 1 2 1 1 ...
##  $ wifi         : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 2 2 1 1 ...
##  $ price_range  : Ord.factor w/ 4 levels "low cost"<"medium cost"<..: 2 3 3 3 2 2 4 1 1 1 ...
```

3.1.2. Data Tidying

The presence of missing values will be determined.

```r
# Missing values by column
colSums(is.na(mobile))
```

battery_power blue clock_speed dual_sim fc 0 0 0 0 0 four_g int_memory m_dep mobile_wt n_cores 0 0 0 0 0 pc px_height px_width ram sc_h 0 0 0 0 0 sc_w talk_time three_g touch_screen wifi 0 0 0 0 0 price_range 0

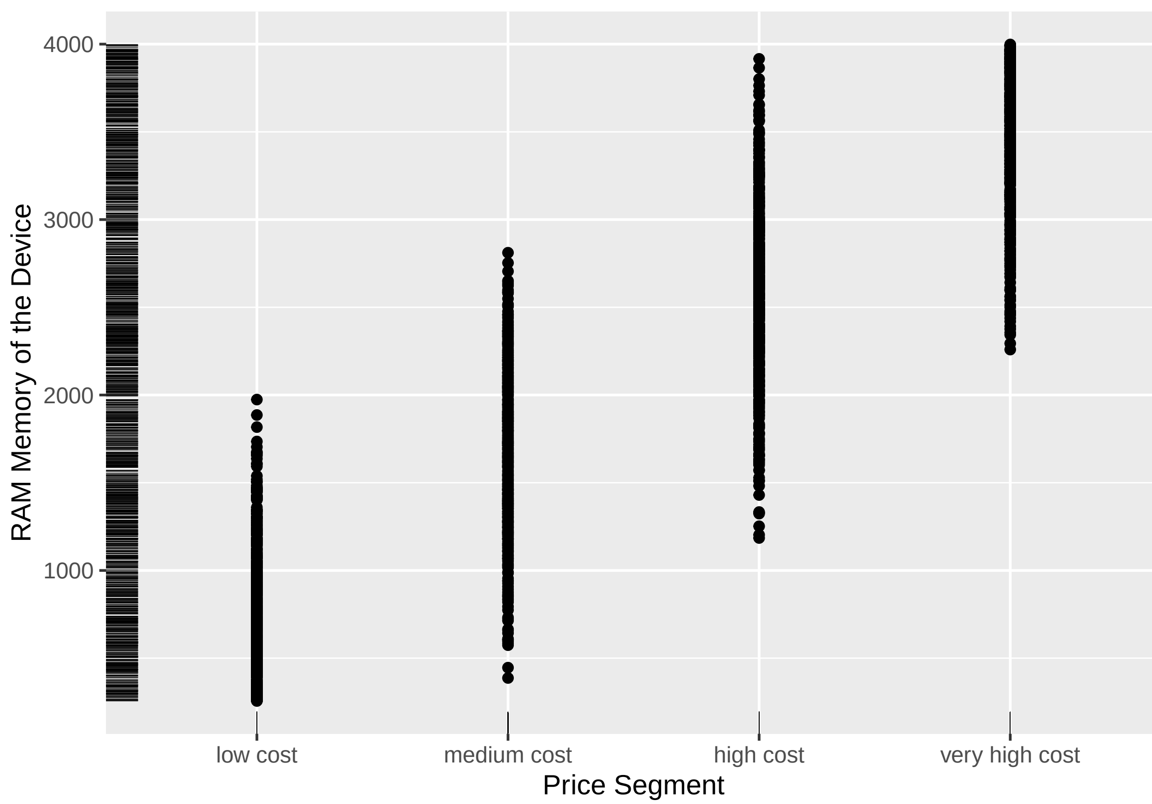
Accoring to these results, there are no missing values.

3.1.3. Visualization

Let's review some aspects related to the characteristics of mobile phones.

Relation between price and RAM memory:

```r
# Graph to see the relation of RAM memory vs Price
ggplot(mobile, aes(price_range, ram)) +
  geom_point() +
  geom_rug(size=0.1) +
  theme_set(theme_minimal(base_size = 18))+
  ylab('RAM Memory of the Device')+
  xlab('Price Segment')
```

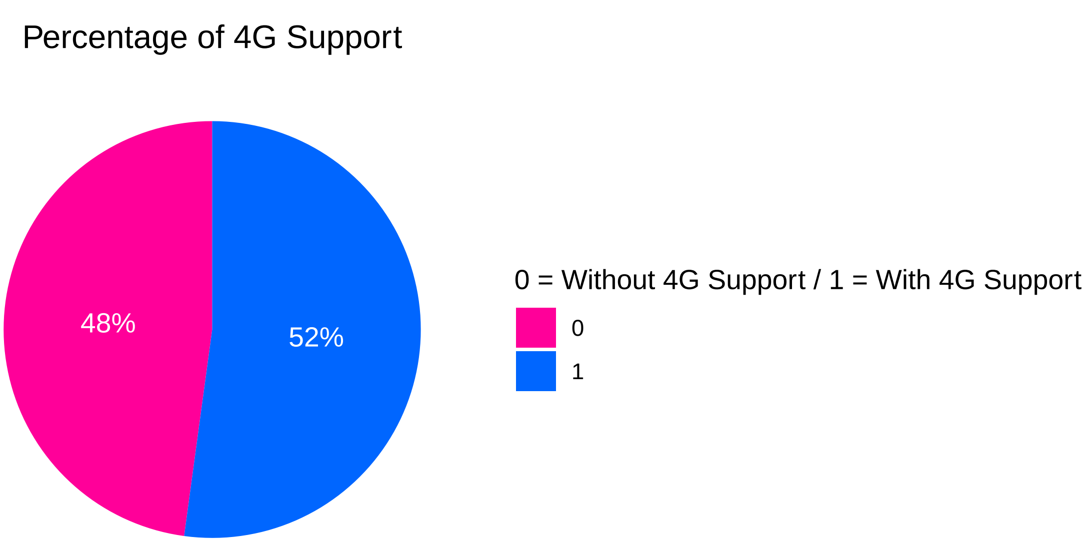


It can be seen that the amount of memory influences the price of phones.

Percentage of phones with 4G support:

```r
# We group and count if 4G support is included or not
mobile %>% group_by(four_g) %>% summarise(freq=n()) %>%
  # Pie chart to determine the percentage of phones with 4G support
ggplot( aes(x="", y=freq, fill=four_g)) +
  geom_bar(stat="identity", width=1)+
```

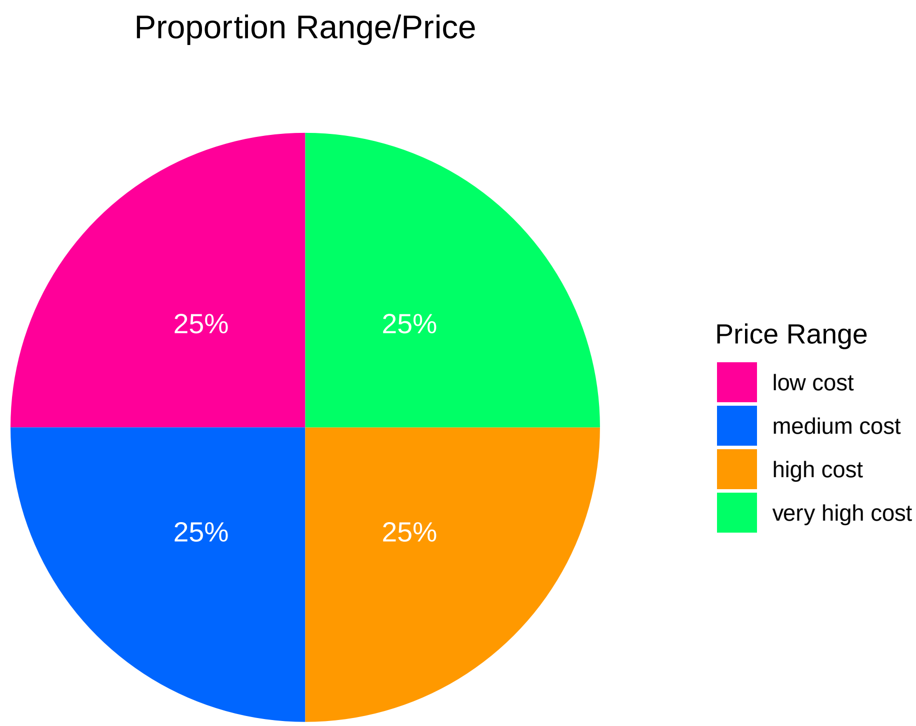
```r
  coord_polar("y", start=0) +
   # We obtain the percentage
  geom_text(aes(label = paste0(round((freq/sum(freq))*100), "%")),
            position = position_stack(vjust = 0.5),color="white")+
  # We apply color
  scale_fill_manual(values=c("#FF0099","#0066FF"))+
  # We include the legend
  labs(x = NULL, y = NULL,
       fill = "0 = Without 4G Support / 1 = With 4G Support", title = "Percentage of 4G Support")+
  theme_classic() +
  theme(axis.line = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        plot.title = element_text(hjust = 0.5),
        axis.title=element_text(size=9,face="bold"),
        legend.position = "right"
        )
```



Let's see the proportion of the price segments.

```r
# We group by price range
mobile %>% group_by(price_range) %>% summarise(freq=n()) %>%
  # We obtain the frequency by price
ggplot( aes(x="", y=freq, fill=price_range)) +
  geom_bar(stat="identity", width=1)+
  coord_polar("y", start=0) +
  # We obtain the percentage by price
  geom_text(aes(label = paste0(round((freq/sum(freq))*100), "%")),
            position = position_stack(vjust = 0.5),color="white")+
  scale_fill_manual(values=c("#FF0099", # We apply some color
                             "#0066FF","#FF9900","#00FF66")) +
```

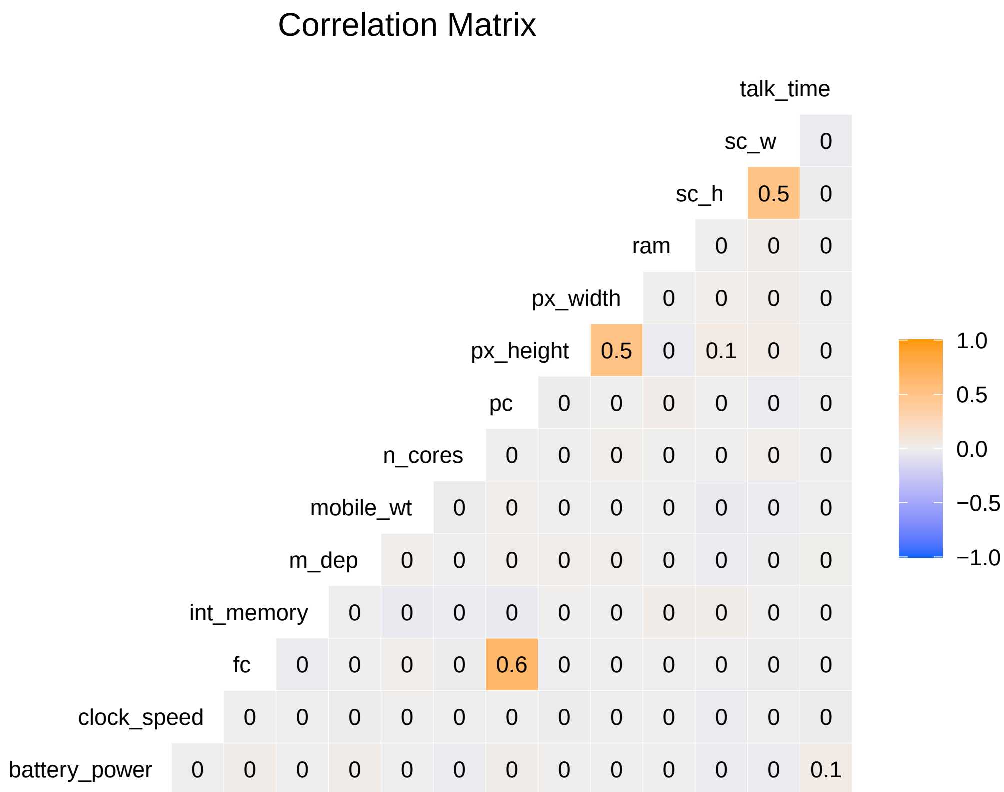
```
  labs(x = NULL, y = NULL, fill = "Price Range",
       title = "Proportion Range/Price")+
  theme_classic() +
  theme(axis.line = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        plot.title = element_text(hjust = 0.5),
        axis.title=element_text(size=9,face="bold"),
        legend.position = "right"
        )
```



The proportion is balanced among the four different price segments.

On the other hand, the correlation matrix is shown as follows:

```
ggcorr(mobile,label = T, size=3,
       label_size = 3, hjust=0.95,
       # Color for the variables with the highest positive correlation
       layout.exp = 3,low = "#0066FF",
       high = "#FF9900")+ # Variables with lower correlation
  labs(
    title="Correlation Matrix"
  )+
  theme_minimal()+
  theme(
    plot.title = element_text(hjust = 0.5),
    axis.title=element_text(size=8,face="bold"),
    axis.text.y=element_blank()
  )
```



According to the correlation matrix, most variables are independent (not correlated) from each other. However, there are indeed some variables that are correlated, as it is the case of:

- **sc_w** with respect to **sc_h** with a positive correlation of 0.5
- **px_width** with respect to **px_height** with a positive correlation of 0.5
- **fc** with respect to **pc** with a positive correlation of 0.6

3.2. Modeling Approaches

Since this problem falls within the classification category, a model based on decision trees and random forests will be used.

3.2.1. Decision Trees

This is a quite successful model for this type of situations, since it allows predictors that are correlated to be numeric or categorical values.

An object for the model will be constructed based on decision trees. But first the dataset will be divided into a training set and a test (validation) set.

The dplyr sample_frac() function will be used to obtain a subset of the original data, consisting of 70% of the total data. The function set.seed() will also be used to make this example reproducible. This is a common practice used to avoid overtraining Machine Learning models. Additionally, another function will be used, which permits to obtain samples from the dataset in a random, not continuous, way to avoid over adjusting or sub adjusting.

Why was the 70%-30% split selected? Tests with 60%-40%, 80%-20%, 90%-10% splits have been carried out, but the distribution 70%-30% has proven to be the most efficient. Additionally, 70%-30% training-test splits are recommended as a good Machine Learning practice.

```r
set.seed(1600) # To permit the reproduction of the results
# 70% of the dataset for training (random sample).
# We avoid overtraining.
x_train <-sample_frac(mobile,.7)
# 30% of the dataset for validation purposes
x_test <- setdiff(mobile, x_train)
```

With setdiff() of dplyr, a complementary data subset to the training one has been obtained, this one for the test set (the remaining 30%).

```r
# We create the model based on classification  trees
arbol <- rpart(formula = price_range ~ ., data = x_train)
```

The classification tree shows, in each node, the classification rule that is applied. The leaves of the tree correspond to the classification of the data. Additionally, interesting information can be displayed, such as the degree of importance of each variable, as it is presented as follows.

```r
# To show the importance of the variables used in the model
sprintf("variable.importance = %s ",c(summary(arbol)$variable.importance));
```

```
## Call:
## rpart(formula = price_range ~ ., data = x_train)
##   n= 1400
##
##           CP nsplit rel error    xerror       xstd
## 1 0.32313576      0 1.0000000 1.0535373 0.01464210
## 2 0.18738050      1 0.6768642 0.6768642 0.01788438
## 3 0.17782027      2 0.4894837 0.5086042 0.01736281
## 4 0.02007648      3 0.3116635 0.3508604 0.01573213
## 5 0.01625239      5 0.2715105 0.3126195 0.01513485
## 6 0.01051625      6 0.2552581 0.2925430 0.01478338
## 7 0.01000000     10 0.2131931 0.2762906 0.01447806
##
## Variable importance
##           ram battery_power      px_width     px_height            pc
##            74             6             6             5             2
##          sc_w     mobile_wt   clock_speed    int_memory         m_dep
##             2             1             1             1             1
##
## Node number 1: 1400 observations,    complexity param=0.3231358
##   predicted class=low cost        expected loss=0.7471429  P(node) =1
##     class counts:   354   354   354   338
##    probabilities: 0.253 0.253 0.253 0.241
##   left son=2 (719 obs) right son=3 (681 obs)
##   Primary splits:
##       ram           < 2170.5 to the left,  improve=248.552600, (0 missing)
##       battery_power < 1301   to the left,  improve= 17.178920, (0 missing)
##       px_height     < 1284.5 to the left,  improve= 11.312780, (0 missing)
##       px_width      < 1645.5 to the left,  improve= 10.123500, (0 missing)
##       int_memory    < 60.5   to the left,  improve=  3.326494, (0 missing)
##   Surrogate splits:
##       pc            < 14.5   to the left,  agree=0.535, adj=0.044, (0 split)
##       px_height     < 276    to the right, agree=0.535, adj=0.044, (0 split)
##       battery_power < 651.5  to the right, agree=0.529, adj=0.031, (0 split)
##       px_width      < 791.5  to the right, agree=0.529, adj=0.031, (0 split)
##       sc_w          < 10.5   to the left,  agree=0.528, adj=0.029, (0 split)
##
## Node number 2: 719 observations,    complexity param=0.1873805
```

```
##   predicted class=low cost        expected loss=0.5076495  P(node) =0.5135714
##     class counts:   354   299    66     0
##    probabilities: 0.492 0.416 0.092 0.000 
##   left son=4 (315 obs) right son=5 (404 obs)
##   Primary splits:
##       ram           < 1106   to the left,  improve=152.528200, (0 missing)
##       battery_power < 1463   to the left,  improve= 17.957900, (0 missing)
##       px_height     < 805.5  to the left,  improve= 16.674400, (0 missing)
##       px_width      < 1010.5 to the left,  improve= 12.341910, (0 missing)
##       n_cores       < 4.5    to the right, improve=  4.484993, (0 missing)
##   Surrogate splits:
##       px_width      < 682.5  to the left,  agree=0.576, adj=0.032, (0 split)
##       pc            < 1.5    to the left,  agree=0.569, adj=0.016, (0 split)
##       px_height     < 39.5   to the left,  agree=0.569, adj=0.016, (0 split)
##       battery_power < 551.5  to the left,  agree=0.567, adj=0.013, (0 split)
##       mobile_wt     < 199.5  to the right, agree=0.567, adj=0.013, (0 split)
## 
## Node number 3: 681 observations,    complexity param=0.1778203
##   predicted class=very high cost  expected loss=0.5036711  P(node) =0.4864286
##     class counts:     0    55   288   338
##    probabilities: 0.000 0.081 0.423 0.496 
##   left son=6 (349 obs) right son=7 (332 obs)
##   Primary splits:
##       ram           < 3108.5 to the left,  improve=137.946900, (0 missing)
##       battery_power < 1353   to the left,  improve= 23.469080, (0 missing)
##       px_width      < 1314.5 to the left,  improve= 16.445100, (0 missing)
##       px_height     < 1281   to the left,  improve= 15.051990, (0 missing)
##       mobile_wt     < 104.5  to the right, improve=  4.995099, (0 missing)
##   Surrogate splits:
##       px_width   < 1740.5 to the left,  agree=0.542, adj=0.060, (0 split)
##       m_dep      < 0.15   to the right, agree=0.539, adj=0.054, (0 split)
##       int_memory < 31.5   to the left,  agree=0.537, adj=0.051, (0 split)
##       sc_w       < 8.5    to the left,  agree=0.536, adj=0.048, (0 split)
##       px_height  < 180.5  to the right, agree=0.535, adj=0.045, (0 split)
## 
## Node number 4: 315 observations
##   predicted class=low cost        expected loss=0.1015873  P(node) =0.225
##     class counts:   283    32     0     0
##    probabilities: 0.898 0.102 0.000 0.000 
## 
## Node number 5: 404 observations,    complexity param=0.02007648
##   predicted class=medium cost     expected loss=0.3391089  P(node) =0.2885714
##     class counts:    71   267    66     0
##    probabilities: 0.176 0.661 0.163 0.000 
##   left son=10 (171 obs) right son=11 (233 obs)
##   Primary splits:
##       battery_power < 1108.5 to the left,  improve=18.511530, (0 missing)
##       ram           < 1508   to the left,  improve=16.292080, (0 missing)
##       px_height     < 709    to the left,  improve=11.408070, (0 missing)
##       px_width      < 1010.5 to the left,  improve=11.005670, (0 missing)
##       n_cores       < 4.5    to the left,  improve= 3.528947, (0 missing)
##   Surrogate splits:
##       px_height  < 126    to the left,  agree=0.606, adj=0.070, (0 split)
##       ram        < 1973.5 to the right, agree=0.606, adj=0.070, (0 split)
##       mobile_wt  < 88.5   to the left,  agree=0.589, adj=0.029, (0 split)
##       int_memory < 2.5    to the left,  agree=0.587, adj=0.023, (0 split)
```

```
##       px_width   < 760.5  to the left,  agree=0.587, adj=0.023, (0 split)
##
## Node number 6: 349 observations,    complexity param=0.01051625
##   predicted class=high cost       expected loss=0.3123209  P(node) =0.2492857
##     class counts:     0    55   240    54
##    probabilities: 0.000 0.158 0.688 0.155
##   left son=12 (315 obs) right son=13 (34 obs)
##   Primary splits:
##       px_height     < 1295   to the left,  improve=12.519720, (0 missing)
##       battery_power < 1354   to the left,  improve=11.213430, (0 missing)
##       ram           < 2670   to the left,  improve=10.106580, (0 missing)
##       px_width      < 1440   to the left,  improve= 9.087715, (0 missing)
##       int_memory    < 16.5   to the left,  improve= 1.901419, (0 missing)
##
## Node number 7: 332 observations
##   predicted class=very high cost  expected loss=0.1445783  P(node) =0.2371429
##     class counts:     0     0    48   284
##    probabilities: 0.000 0.000 0.145 0.855
##
## Node number 10: 171 observations,    complexity param=0.02007648
##   predicted class=medium cost     expected loss=0.3976608  P(node) =0.1221429
##     class counts:    64   103     4     0
##    probabilities: 0.374 0.602 0.023 0.000
##   left son=20 (68 obs) right son=21 (103 obs)
##   Primary splits:
##       ram       < 1541   to the left,  improve=40.465390, (0 missing)
##       px_width  < 1019.5 to the left,  improve=12.654230, (0 missing)
##       px_height < 892    to the left,  improve= 8.467650, (0 missing)
##       sc_h      < 18.5   to the right, improve= 5.176432, (0 missing)
##       sc_w      < 0.5    to the left,  improve= 4.885893, (0 missing)
##   Surrogate splits:
##       clock_speed < 2.55   to the right, agree=0.643, adj=0.103, (0 split)
##       px_width    < 580    to the left,  agree=0.626, adj=0.059, (0 split)
##       sc_h        < 18.5   to the right, agree=0.626, adj=0.059, (0 split)
##       sc_w        < 1.5    to the left,  agree=0.626, adj=0.059, (0 split)
##       mobile_wt   < 179    to the right, agree=0.620, adj=0.044, (0 split)
##
## Node number 11: 233 observations,    complexity param=0.01625239
##   predicted class=medium cost     expected loss=0.2961373  P(node) =0.1664286
##     class counts:     7   164    62     0
##    probabilities: 0.030 0.704 0.266 0.000
##   left son=22 (186 obs) right son=23 (47 obs)
##   Primary splits:
##       ram           < 1896.5 to the left,  improve=18.895200, (0 missing)
##       px_height     < 708    to the left,  improve=12.816980, (0 missing)
##       px_width      < 1112.5 to the left,  improve=12.667620, (0 missing)
##       battery_power < 1441.5 to the left,  improve= 4.949631, (0 missing)
##       n_cores       < 5.5    to the left,  improve= 4.366491, (0 missing)
##   Surrogate splits:
##       px_width < 506.5  to the right, agree=0.807, adj=0.043, (0 split)
##
## Node number 12: 315 observations,    complexity param=0.01051625
##   predicted class=high cost       expected loss=0.2793651  P(node) =0.225
##     class counts:     0    55   227    33
##    probabilities: 0.000 0.175 0.721 0.105
##   left son=24 (113 obs) right son=25 (202 obs)
```

```
##   Primary splits:
##       ram           < 2459.5 to the left,  improve=10.268550, (0 missing)
##       battery_power < 1423.5 to the left,  improve= 9.874939, (0 missing)
##       px_width      < 1197   to the left,  improve= 5.068894, (0 missing)
##       px_height     < 502    to the left,  improve= 4.039209, (0 missing)
##       int_memory    < 56.5   to the left,  improve= 1.474189, (0 missing)
##   Surrogate splits:
##       clock_speed   < 2.85   to the right, agree=0.654, adj=0.035, (0 split)
##       battery_power < 1938   to the right, agree=0.651, adj=0.027, (0 split)
##       n_cores       < 7.5    to the right, agree=0.651, adj=0.027, (0 split)
##       px_height     < 1267.5 to the right, agree=0.648, adj=0.018, (0 split)
##       mobile_wt     < 81     to the left,  agree=0.644, adj=0.009, (0 split)
##
## Node number 13: 34 observations
##   predicted class=very high cost  expected loss=0.3823529  P(node) =0.02428571
##     class counts:     0     0    13    21
##    probabilities: 0.000 0.000 0.382 0.618
##
## Node number 20: 68 observations
##   predicted class=low cost        expected loss=0.1911765  P(node) =0.04857143
##     class counts:    55    13     0     0
##    probabilities: 0.809 0.191 0.000 0.000
##
## Node number 21: 103 observations
##   predicted class=medium cost     expected loss=0.1262136  P(node) =0.07357143
##     class counts:     9    90     4     0
##    probabilities: 0.087 0.874 0.039 0.000
##
## Node number 22: 186 observations
##   predicted class=medium cost     expected loss=0.1989247  P(node) =0.1328571
##     class counts:     7   149    30     0
##    probabilities: 0.038 0.801 0.161 0.000
##
## Node number 23: 47 observations
##   predicted class=high cost       expected loss=0.3191489  P(node) =0.03357143
##     class counts:     0    15    32     0
##    probabilities: 0.000 0.319 0.681 0.000
##
## Node number 24: 113 observations,    complexity param=0.01051625
##   predicted class=high cost       expected loss=0.380531  P(node) =0.08071429
##     class counts:     0    42    70     1
##    probabilities: 0.000 0.372 0.619 0.009
##   left son=48 (68 obs) right son=49 (45 obs)
##   Primary splits:
##       battery_power < 1430.5 to the left,  improve=19.944500, (0 missing)
##       px_width      < 1206   to the left,  improve=12.840340, (0 missing)
##       px_height     < 504    to the left,  improve= 7.911809, (0 missing)
##       fc            < 11.5   to the right, improve= 4.286747, (0 missing)
##       m_dep         < 0.15   to the right, improve= 2.968637, (0 missing)
##   Surrogate splits:
##       talk_time   < 19.5   to the left,  agree=0.646, adj=0.111, (0 split)
##       clock_speed < 2.45   to the left,  agree=0.619, adj=0.044, (0 split)
##       int_memory  < 7.5    to the right, agree=0.619, adj=0.044, (0 split)
##       mobile_wt   < 81     to the right, agree=0.619, adj=0.044, (0 split)
##       ram         < 2174   to the right, agree=0.619, adj=0.044, (0 split)
##
```

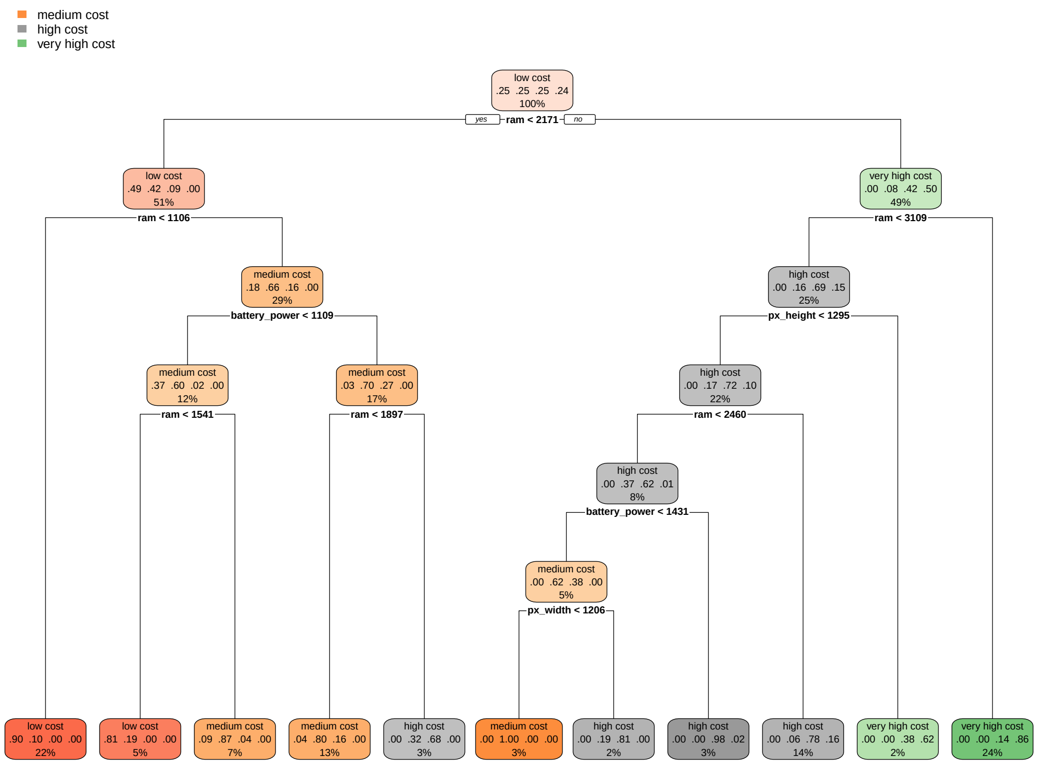
```
## Node number 25: 202 observations
##   predicted class=high cost       expected loss=0.2227723  P(node) =0.1442857
##     class counts:     0    13   157    32
##    probabilities: 0.000 0.064 0.777 0.158
##
## Node number 48: 68 observations,    complexity param=0.01051625
##   predicted class=medium cost     expected loss=0.3823529  P(node) =0.04857143
##     class counts:     0    42    26     0
##    probabilities: 0.000 0.618 0.382 0.000
##   left son=96 (36 obs) right son=97 (32 obs)
##   Primary splits:
##       px_width    < 1206   to the left,  improve=22.367650, (0 missing)
##       px_height   < 903.5  to the left,  improve=11.117650, (0 missing)
##       pc          < 1.5    to the right, improve= 3.939076, (0 missing)
##       m_dep       < 0.15   to the right, improve= 3.518115, (0 missing)
##       clock_speed < 1.95   to the left,  improve= 2.324410, (0 missing)
##   Surrogate splits:
##       px_height   < 504    to the left,  agree=0.750, adj=0.469, (0 split)
##       ram         < 2299   to the left,  agree=0.647, adj=0.250, (0 split)
##       mobile_wt   < 148.5  to the right, agree=0.618, adj=0.188, (0 split)
##       pc          < 1.5    to the right, agree=0.618, adj=0.188, (0 split)
##       clock_speed < 1.95   to the left,  agree=0.603, adj=0.156, (0 split)
##
## Node number 49: 45 observations
##   predicted class=high cost       expected loss=0.02222222  P(node) =0.03214286
##     class counts:     0     0    44     1
##    probabilities: 0.000 0.000 0.978 0.022
##
## Node number 96: 36 observations
##   predicted class=medium cost     expected loss=0  P(node) =0.02571429
##     class counts:     0    36     0     0
##    probabilities: 0.000 1.000 0.000 0.000
##
## Node number 97: 32 observations
##   predicted class=high cost       expected loss=0.1875  P(node) =0.02285714
##     class counts:     0     6    26     0
##    probabilities: 0.000 0.188 0.812 0.000
```

```
##  [1] "variable.importance = 616.434220828818 "
##  [2] "variable.importance = 48.3301253830933 "
##  [3] "variable.importance = 46.8018698624219 "
##  [4] "variable.importance = 44.0884289264599 "
##  [5] "variable.importance = 17.5644708207675 "
##  [6] "variable.importance = 16.3279967040378 "
##  [7] "variable.importance = 9.43460437291979 "
##  [8] "variable.importance = 8.91041058112946 "
##  [9] "variable.importance = 8.38298580046322 "
## [10] "variable.importance = 7.47904820312531 "
## [11] "variable.importance = 2.38031724758195 "
## [12] "variable.importance = 2.21605516674057 "
## [13] "variable.importance = 0.27261635866774 "
```

Note that the variable that indicates the RAM memory seems to be the one with the highest importance in this model.

The graphical representation of this tree is the following.

```r
# Displays the graph of the created tree
rpart.plot(arbol)
```



We know this must be finally done with the test set, but let's apply this prediction to the training set to see how the model behaves for this sub dataset.

```r
# Predict using the training set
# (IMPORTANT NOTE: these are predictions for the TRAINING SET)
predict_train <- predict(arbol,x_train, type = "class")
# We obtain the metrics
cfm_train <- confusionMatrix(data =predict_train,
                             reference =x_train$price_range)
```

```r
# Accuracy of the model (training set)
sprintf('Accuracy = %10.2f',cfm_train$overall[1]*100)
```

```
## [1] "Accuracy =      84.07"
```

A precision of about 84% is obtained for the training set.

Now, the definitive test will be performed. Let's see how this model does using the TEST SET that was previously created.

```r
prediccion <- predict(arbol,
                      newdata = x_test, type = "class")
```

We cross the predictions with the actual data of the test set to generate a confusion matrix.

```r
# We train our model
cfm_arbol <- confusionMatrix(prediccion,x_test[["price_range"]])
# Metrics of the model
cfm_arbol$overall
```

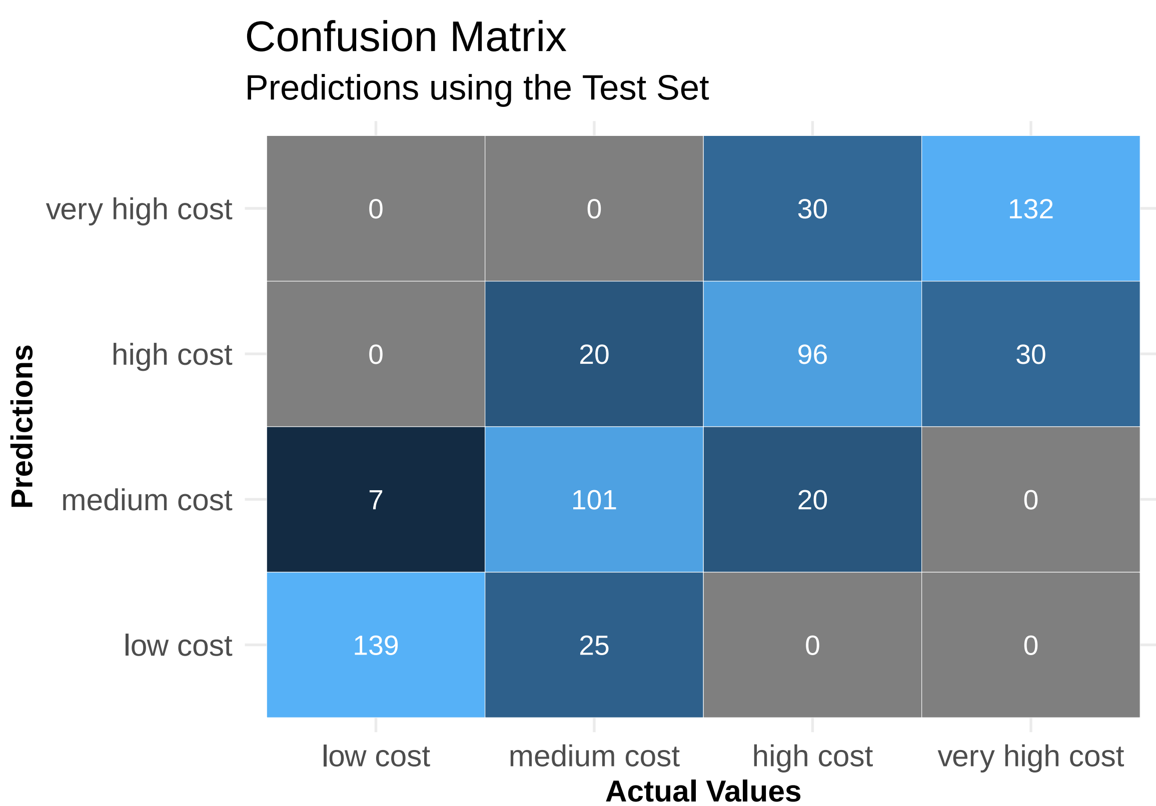
```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
```

```
##    7.800000e-01   7.064579e-01   7.446834e-01   8.125331e-01   2.700000e-01
## AccuracyPValue  McnemarPValue
##   6.008398e-149            NaN
```

Not bad. The precision (accuracy), Kappa and other statistics have quite acceptable values. According to these metrics, the model is capable of explaining 78% of the price ranges.

The confusion matrix for the test set is the following:

```r
# We graph the confusion matrix
ggplot(data = as.data.frame(cfm_arbol$table),
       # Prediction vs actual values
       aes(x = Reference, y = Prediction)) +
    geom_tile(aes(fill = log(Freq)),
              colour = "white") +
    geom_text(aes(x = Reference, y = Prediction,
                  label = Freq),color="white") +
    labs(
      title =  'Confusion Matrix',
      subtitle = 'Predictions using the Test Set',
      x = "Actual Values",
      y = "Predictions"
    )+
    theme_minimal()+
    theme(
          title = element_text(size=14),
          axis.title=element_text(size=12, face="bold"),
          axis.text.x=element_text(size=12),
           axis.text.y=element_text(size=12),
          legend.position = "none"
          ) +
     scale_colour_gradient2()
```



3.2.2. Random Forests

Since the decision trees approach may tend to suffer from overfitting, a second technique will be used to address the studied problem. Now, a single tree will not be used, but a group of trees that work together to improve the performance of the initially proposed model.

Similar as in the previous model, a *RandomForest* type object will be used. Again, a seed will be used so that the results can be reproduced. An important factor for this algorithm is to determine the number of trees to use because, the greater the number, the heavier the calculation process will be.

Having carried out preliminary tests with a number of trees $100 \leq N_{Trees} \leq 500$, we have opted for building a forest consisting of 300 trees.

```
# To create the predictive model
set.seed(2000)
# We create the random forest model
RF_model<-randomForest(price_range ~ ., data = x_train, importance=TRUE, ntree = 300)
RF_model
```

Call: randomForest(formula = price_range ~ ., data = x_train, importance = TRUE, ntree = 300) Type of random forest: classification Number of trees: 300 No. of variables tried at each split: 4

```
OOB estimate of  error rate: 12.86%
```

Confusion matrix: low cost medium cost high cost very high cost class.error low cost 333 21 0 0 0.05932203 medium cost 30 295 29 0 0.16666667 high cost 0 43 285 26 0.19491525 very high cost 0 0 31 307 0.09171598

According to the statistics of this model, the number of variables that are tested in each branch (or split) is 4, with an estimated rate of error close to 12.9% for the training set.

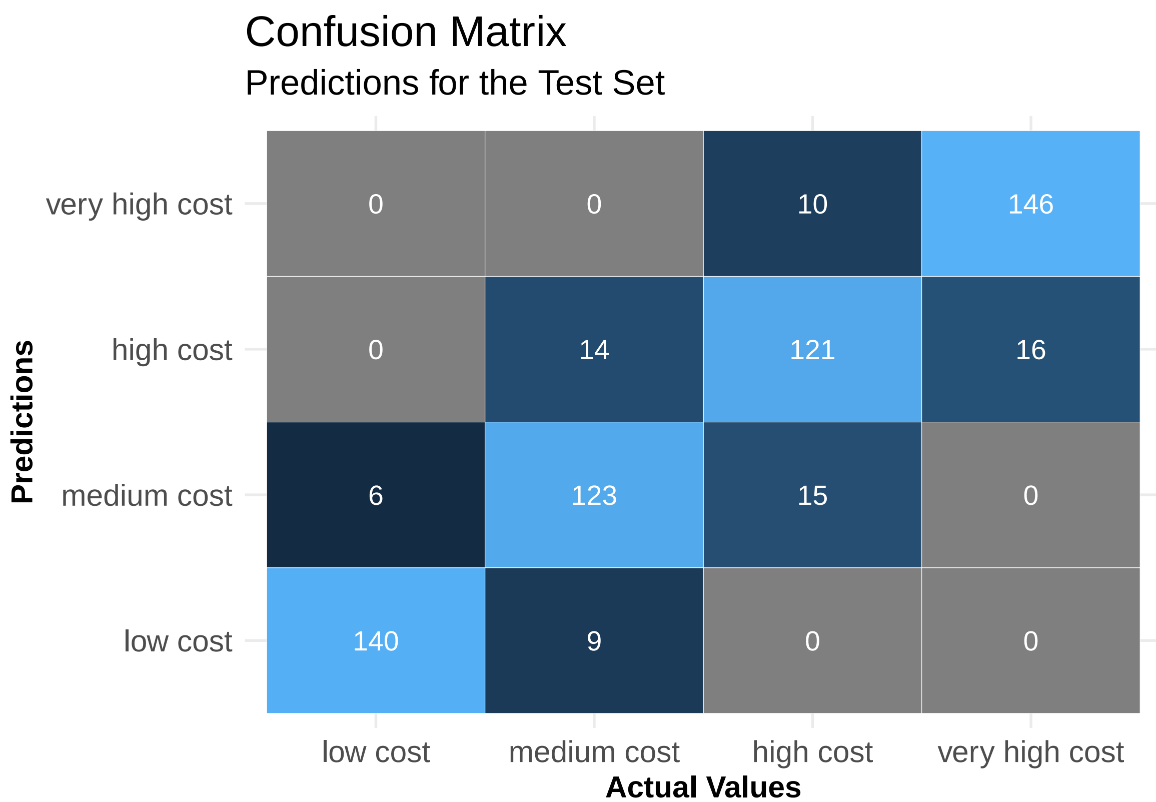
We will perform the test over the TEST SET.

```
# Prediction with the test set
prediccion_RF <- predict(RF_model, newdata = x_test, type = "class")
```

The confusion matrix for the test set is the following:

```
# We create the confusion matrix (estimated price range vs actual values)
# Test set
cfm_RF <- confusionMatrix(data = prediccion_RF,
                       reference =x_test$price_range)
ggplot(data = as.data.frame(cfm_RF$table),
       aes(x = Reference, y = Prediction)) +
    geom_tile(aes(fill = log(Freq)),
              colour = "white") +
    geom_text(aes(x = Reference, y = Prediction,
                  label = Freq),color="white") +
    labs(
      title =  'Confusion Matrix',
      subtitle = 'Predictions for the Test Set',
      x = "Actual Values",
      y = "Predictions"
    )+
    theme_minimal()+
    theme(
          title = element_text(size=14),
          axis.title=element_text(size=12, face="bold"),
          axis.text.x=element_text(size=12),
           axis.text.y=element_text(size=12),
          legend.position = "none"
```

```
        ) +
    scale_colour_gradient2()
```



The precision of the model is about 88%.

```
# Precision of the RandomForest model (with the TEST SET)
sprintf('Accuracy = %10.2f',cfm_RF$overall[1]*100)
```

```
## [1] "Accuracy =      88.33"
```

The model based on decision trees established a starting point for modeling the studied problem, which allows to confirm that, using random forests, the model's performance is outstandingly improved. In general, both models are appropriate to solve this challenge.

4. Results and Discussion

Two approaches have been proposed to solve the problem of determining the price range or segment for specific mobile phones, based on their characteristics and considering historical data.

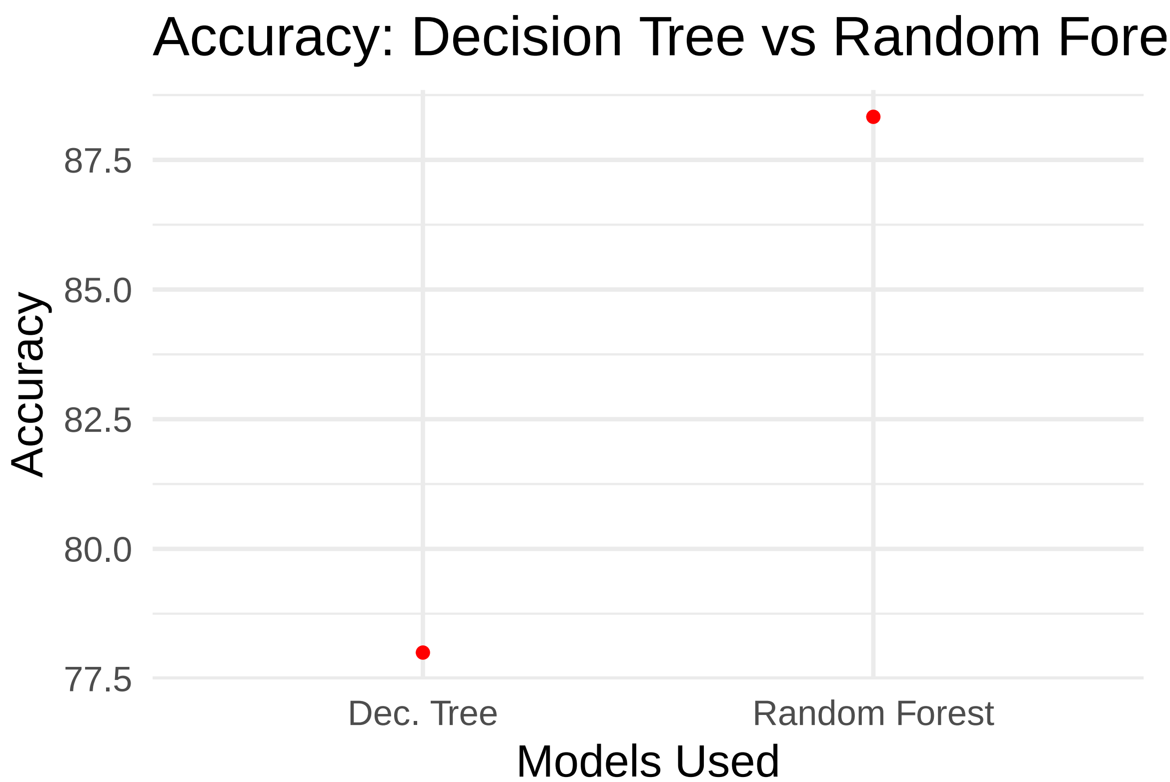
The model based on decision trees provides an accuracy of around 78%, which is an acceptable result, but which can also be substantially improved.

Looking to improve the results obtained by the decision trees approach, a model based on random forests was implemented. The random forest model outdid the decision trees model, by obtaining an accuracy of about 88%, almost 10 percentage points higher. These results are displayed in the graph that follows:

```
# We create a table with the statistics Accuracy/Kappa
acc<-data.frame(
    "Accuracy"= c(round((cfm_arbol$overall[1])*100,2),round((cfm_RF$overall[1])*100,2)),
```

```r
    "Kappa" = c(round((cfm_RF$overall[2])*100,2),round((cfm_RF$overall[2])*100,2))
    )
```

```r
# Graph of the precision percentage of the two used models
qplot(c('Dec. Tree','Random Forest'), acc$Accuracy,main = 'Accuracy: Decision Tree vs Random Forest',
      ylab = 'Accuracy',xlab = 'Models Used',color = I("red"),size= I(2))
```



5. Conclusions

Two classification models have been developed and applied to assign a price range to mobile phones based on their main technical and functional characteristics.

According to the analysis performed throughout this project, the model based on *Random Forests* has the best performance in predicting the price range of mobile phones.

The Random Forest model creates multiple trees on the data subset and combines the output of all the trees, reducing, in this way, the overfitting problem that decision trees have. Random Forest also reduces the variance, therefore improving accuracy.

The main limitation of the Random Forest approach is that the inclusion of a large number of trees can make the algorithm substantially slow and ineffective for real-time prediction purposes. This type of algorithms is generally fast to train but are slow to create predictions once they have been trained.

To speed computations, the number of estimators should be lowered. To increase the accuracy of the model, the number of trees should be increased. Specify the maximum number of features to be considered at each node/branch split; increasing tree size would increase the accuracy.

Finally, it must be noted that, even though seed values have been used for reproducibility purposes, it is probable that results vary when running the code, due to factors (random in nature) that are intrinsic to the methods and functions used. Therefore, new executions may display slightly different results.

6. Appendix - Operating System Used

```r
print("SO:")
```

```
## [1] "SO:"
```

```r
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          4
## minor          0.5
## year           2021
## month          03
## day            31
## svn rev        80133
## language       R
## version.string R version 4.0.5 (2021-03-31)
## nickname       Shake and Throw
```