Mobile Phones Price: a classification problem

Angelica Giangiacomi (2055739) Anthony Palmieri (2038503)
Pietro Maria Sanguin (2063170)

Statistical Learning course - 20/07/2022

Contents

Introduction

The arrival of summer usually leads to the purchase of new electronics. It is well known, in fact, that electronic devices, and in particular cell phones, become damaged especially when exposed to excessive heat, temperature changes, sand and salt, in short, all typical characteristics of the summer period. Given the pressing need to buy a new cell phone, we asked ourselves: what are the elements that most influence the price of a cell phone? A bigger screen, a larger battery capacity, a better performing ram, a better camera...? To give an answer as objective as possible, we looked at the "Mobile Price Classification dataset" in which the price ranges of some cell phones are listed. The name of the phone models considered are not specified, just as the year of production is not specified, however, other features useful for classification are present. Getting more technical, we are going to deal with a multiclass classification problem. In particular, we will analyze and study some attributes belonging to different cell phones with the purpose of determining the price range of each device, given its main qualities. The attributes at our disposal are many and, as we will see during the analysis, some of them will prove to be unnecessary for our purpose. First we are going to describe the

dataset we used and then we will move on to the preprocessing and data cleaning parts in which we will prepare the data for the training phase. After that, before starting to train the actual multiclass classification models, we are going to study in more details what the data looks like, trying to identify some patterns and distributions that might be present within the dataset and trying to identify a priori which might be the most important features for determining the price range of a cell phone. After this first part of analysis, we finally start training some *Supervised Learning* models, specifically: *Multinomial Logistic Regression*, *Linear Discriminant Analysis*, *Quadratic Discriminant Analysis* and *K-Nearest Neighbours*. Each of these will be trained several times under different conditions, for example when applying feature selection techniques.

1. The Dataset

As stated before, the dataset used in this project is the "Mobile Price Classification" taken from the Kaggle website[1]. Specifically, it consists of 2000 examples which we will split into Training and Test sets.

```
# import the dataset
phones_train <- read.csv('MobilePricesTrain.csv')
attach(phones_train)
```

Each observation referers to a mobile phone and is characterized by means of 20 features, some of them numerical (i.e. non categorical) [N] others categorical [C] as specified below:

1. ***Battery_power:*** Total energy a battery can store in mAh [N]
2. ***Blue:*** Has bluetooth or not [C]
3. ***Clock_speed:*** Speed at which microprocessor executes instructions [N]
4. ***Dual_sim:*** Has dual sim support or not [C]
5. ***Fc:*** Front Camera mega pixels [N]
6. ***Four_g:*** Has 4G or not [C]
7. ***int_memory:*** Internal Memory in Gigabytes [N]
8. ***m_dep:*** Mobile Depth in cm [N]
9. ***mobile_wt:*** Weight of mobile phone [N]
10. ***n_cores:*** Number of cores of processor [N]
11. ***pc:*** Primary Camera mega pixels [N]
12. ***px_height:*** Pixel Resolution Height [N]
13. ***px_width:*** Pixel Resolution Width [N]
14. ***ram:*** Random Access Memory in Megabytes [N]
15. ***sc_h:*** Screen Height of mobile in cm [N]
16. ***sc_w:*** Screen Width of mobile in cm [N]
17. ***talk_time:*** Longest time that battery will last by a call [N]
18. ***three_g:*** Has 3G or not [C]
19. ***touch_screen:*** Has touch screen or not [C]
20. ***wifi:*** Has wifi or not [C]

The main objective is to predict the price range of a given cell phone; the price ranges considered are 0, 1, 2, 3, which correspond to: Low cost, Medium cost, High cost, Very high cost. These will constitute the labels of our examples. In the following section we will discuss preprocessing and data cleaning.

2. Preprocessing and data cleaning

First of all, we check for missing data:

```r
# check for not available values
cat("Is there any Not Available value?", any(is.na(phones_train)),"\n")
```

```
## Is there any Not Available value? FALSE
```

```r
# check for NaN values
cat("Is there any Not a Number value?", any(is.nan(as.matrix(phones_train))))
```

```
## Is there any Not a Number value? FALSE
```

There are no missing values. In order to implement model selection and evaluation strategies we randomly divide the original dataset into two disjoint subsets:

- a training subset which we will use to train the parameters of our models (80% of original size);
- a test set wich we will use to test the various algorithms (20% of original size).

Eventually, on the training subset we will follow a cross validation approach to determine the optimal values of the hyperparameters (like regularization hyperparameters, for instance). We therefore proceed with the splitting:

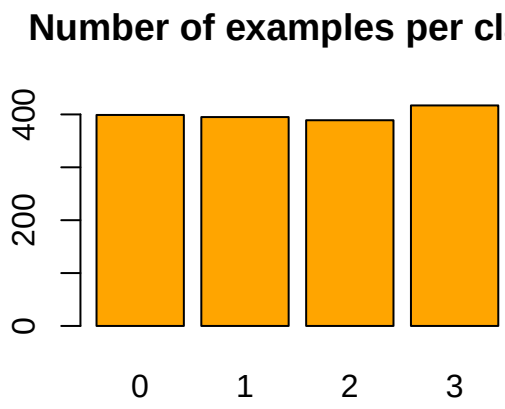
```r
# percentage of samples instantiated for training
p = 80
# set random seed
set.seed(1)
# vector of indices to get examples for Training
my_smp = sample.int(nrow(phones_train), nrow(phones_train)*p/100, replace=FALSE)
# 80% for Training set
Training_set = phones_train[my_smp, ]
# 20% for Test set
Test_set = phones_train[-my_smp, ]
```

Now we separate the label column (i.e., the one for the ***price_range***) from the features.

```r
# get Training set labels
y_train <- Training_set$price_range
# remove labels column from Training set
X_train <- Training_set[, -length(Training_set)]
# get Test set labels
y_test <- Test_set$price_range
# remove labels column from Test set
X_test <- Test_set[, -length(Test_set)]
```

As a next step, we check whether the data in the Training set is balanced across the different classes or not.

```r
# check if data is balanced
barplot(table(Training_set$price_range), col = "orange", main = "Number of examples per class")
```



The data within the Training set is almost perfectly balanced, so there is no need to employ any oversampling (such as SMOTE) or undersampling method. Before performing any analysis on our dataset, it is necessary to transform the categorical variables into factors.

```
# convert categorical variables to factors
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)
```

As a final step we also perform a standardization of the values of the non-categorical features to force them to be in the range [0,1].

```
library(caret)
```

```
## Caricamento del pacchetto richiesto: ggplot2
```

```
## Caricamento del pacchetto richiesto: lattice
```

```
# train MinMax Scaler on Training set
scaling <- preProcess(X_train, method="range")
# apply MinMax Scaler on Training set
X_train_scaled <- predict(scaling, X_train)
# apply MinMax Scaler on Test set
X_test_scaled <- predict(scaling, X_test)
```

Before training any algorithm, we begin to explore our data, searching for relationships among the various predictors, in order to possibly apply variable selection techniques.

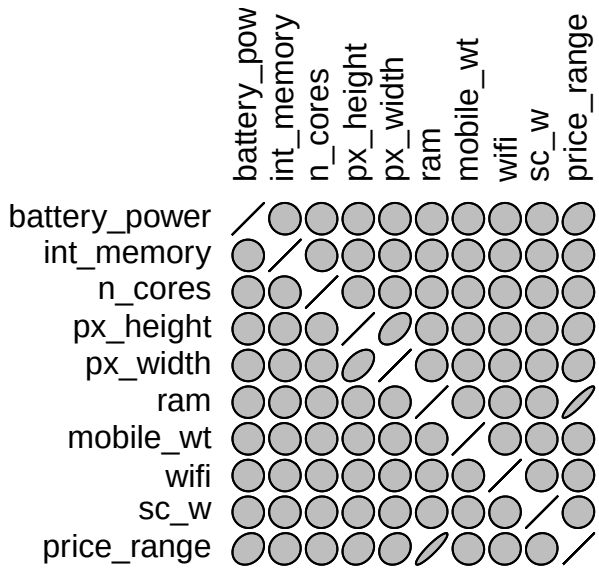
3. Exploratory data analysis

Since each observation consists of 20 features, in this section we plot some graphs to see if and what patterns and regularities exist among the various attributes in order to possibly identify those that are less useful for the analysis. To make the following plots more readable, some variables that had no correlation with any other were removed a priori.

```
library(ellipse)
```

```
##
## Caricamento pacchetto: 'ellipse'

## Il seguente oggetto è mascherato da 'package:graphics':
##
##     pairs
```

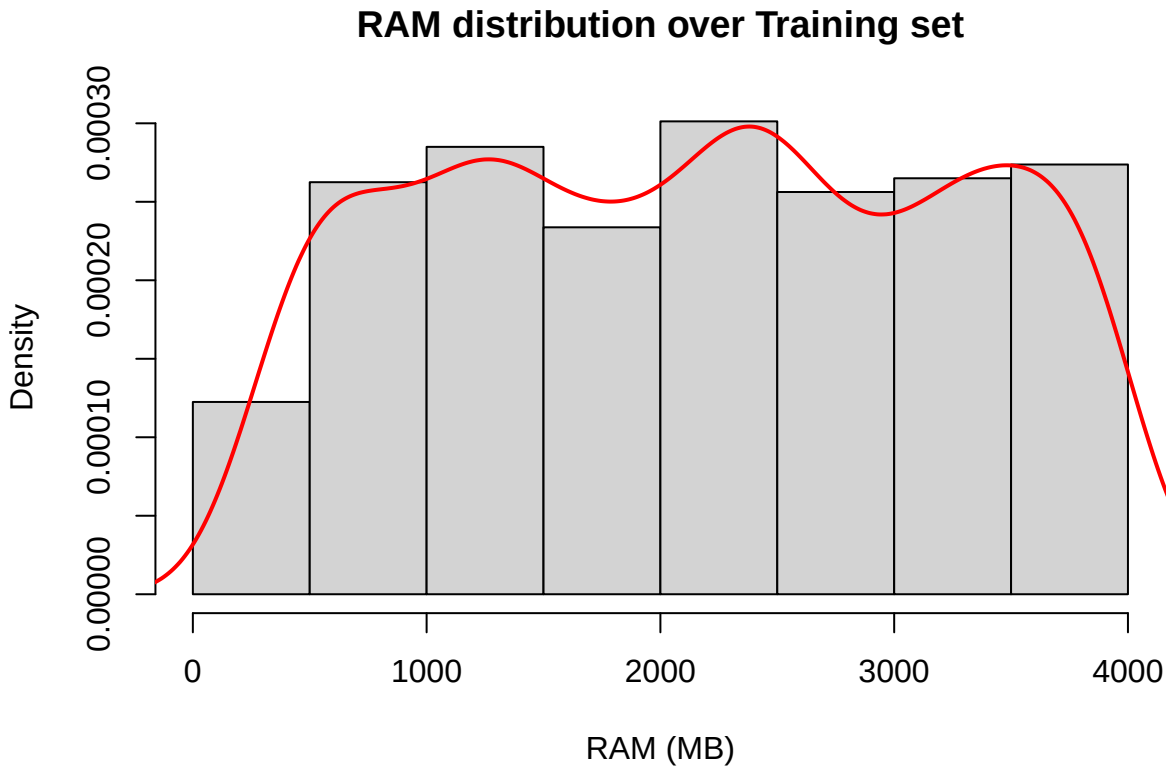
```
plotcorr(cor(phones_train[my_smp, c(1, 7, 10, 12, 13, 14, 9, 20, 16, 21)]))
```



Surprisingly, there seem to be just a few variables correlated with each other, especially in the case of *price_range*, for which we find only one strong correlation with the feature *ram*. Probably this is strongly influenced by the fact that the price range is a discrete variable, thus observations referring to different cell phones and having even quite different features, yet belong to the same price range.

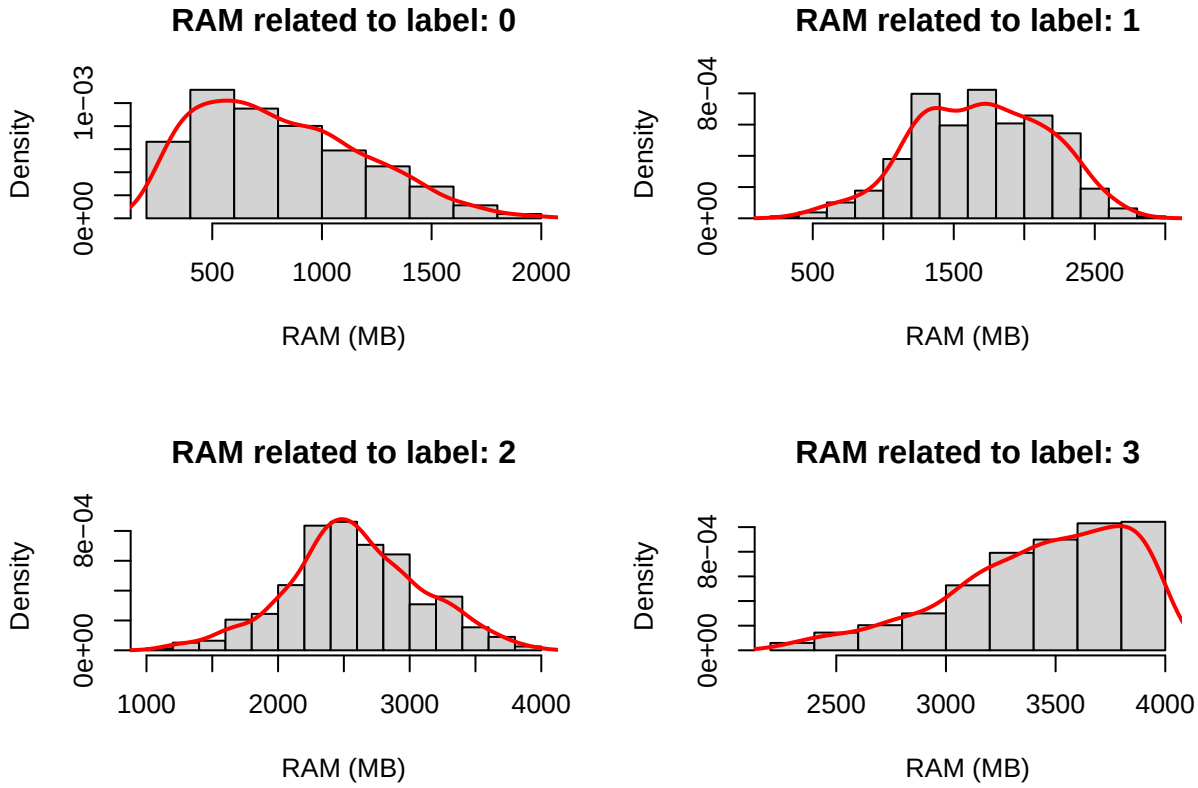
To better highlight and observe the strong correlation between *ram* and *price_range*, let us first graph the distribution of *ram* with respect to the entire training set, and then for each of the four price ranges individually.

```r
# plot RAM distribution
hist(X_train$ram, main="RAM distribution over Training set", freq=FALSE,  xlab="RAM (MB)")
lines(density(X_train$ram), col = "Red", lwd = 2)
```



The diagram above shows a roughly uniform distribution except for the first bin, which has a lower density than the others.

```r
# set four subplots
par(mfrow=c(2,2))

# histogram related to label 0
hist(X_train[y_train == 0, 14], main="RAM related to label: 0", xlab = "RAM (MB)", freq=FALSE)
lines(density(X_train[y_train == 0, 14]), col = "Red", lwd = 2)

# histogram related to label 1
hist(X_train[y_train == 1, 14], main="RAM related to label: 1", xlab = "RAM (MB)", freq=FALSE)
lines(density(X_train[y_train == 1, 14]), col = "Red", lwd = 2)

# histogram related to label 2
hist(X_train[y_train == 2, 14], main="RAM related to label: 2", xlab = "RAM (MB)", freq=FALSE)
lines(density(X_train[y_train == 2, 14]), col = "Red", lwd = 2)

# histogram related to label 3
hist(X_train[y_train == 3, 14], main="RAM related to label: 3", xlab = "RAM (MB)", freq=FALSE)
lines(density(X_train[y_train == 3, 14]), col = "Red", lwd = 2)
```




```
par(mfrow=c(1,1))
```

Unlike the full distribution, in the plots above, the two middle classes (i.e., 1 and 2) have a bell-shaped distribution, while the lowest and highest bands (i.e., 0 and 3) have a right-skewed and a left-skewed distribution, respectively. If we wanted to treat the classes individually, for the latter two it would be better to proceed with a *log-transformation* to prevent the tails from affecting the training steps too much; however, during the analysis we will use the dataset as a whole, so it will not be necessary to apply such a method.

Below, in an attempt to lighten the analysis, we study the Variance Inflation Factor (VIF) of the numerical attributes, to see if some features can be seen as linear combinations of others.

```
library(fmsb)
```

```
## Registered S3 methods overwritten by 'fmsb':
##   method    from
##   print.roc pROC
##   plot.roc  pROC
```

```
# get only numerical features on Training set
num_train = X_train[, -c(2,4,6,18,19,20)]
num_train <- as.data.frame(num_train)

# apply VIF on all features
vifs = c()
```

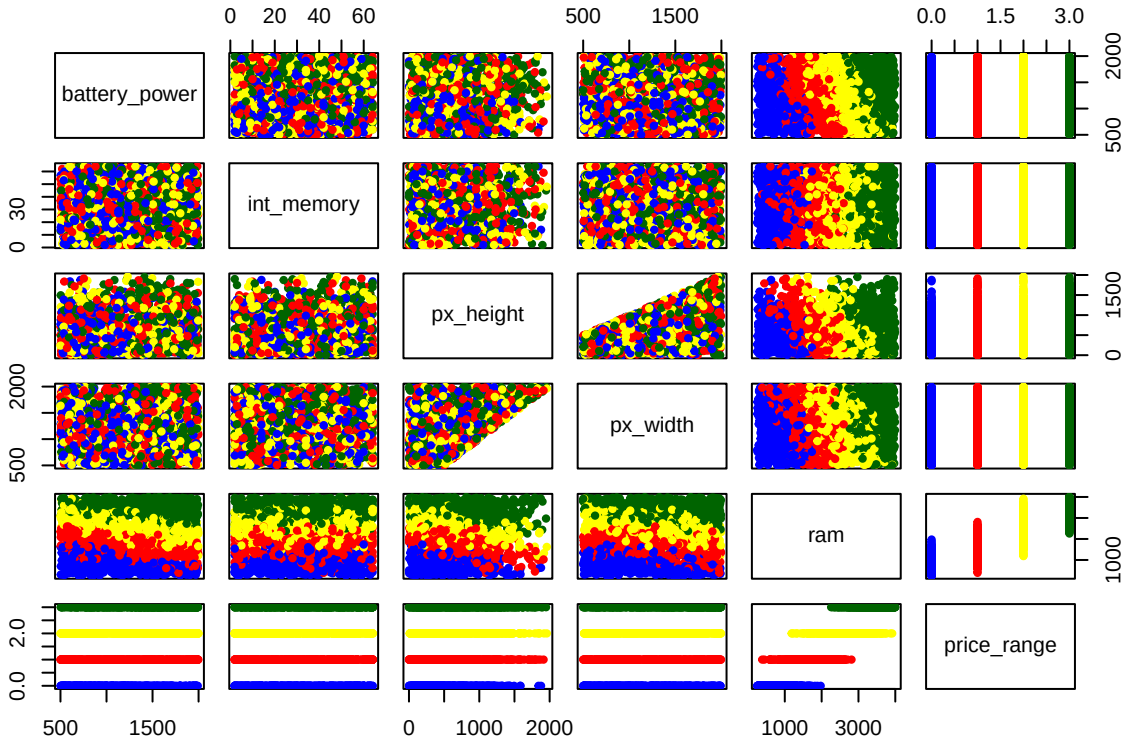
```
# loop over all numerical columns
for( i in 1:length(num_train) )
  {
    # First of all we want to create the formula x1~x2+x3+x4+... for each one of the columns
    # We use this formula in the Linear Model function

    # define a temporary dataframe having all columns but i-th one
    tmp <- num_train[,-c(i)]
    # define the string with the sum of all tmp's columns' names
    j = paste(names(tmp), collapse="+")
    lst = c(names(num_train)[i],j)
    # create the final formula
    formula = paste(lst, collapse='~')
    # call Linear Model function
    model = lm(formula, data = num_train)
    # add to the list the current VIF value
    vifs <- c(vifs, VIF(model))
}

# print all VIFs values
vifs
```

```
##  [1] 1.011139 1.002891 1.730141 1.010976 1.007034 1.003561 1.006990 1.736878
##  [9] 1.353688 1.352578 1.006079 1.399044 1.396769 1.006836
```

Since the values of the VIFs are all very close to 1, no important multicollinear relationships emerge among the numerical predictors. For this reason each model will be trained at first considering the whole set of features, and after that we will move on to apply also feature selection techniques.

As a last thing, we check whether possible nonlinear relationships exist among the predictors. To make the plots more readable, we keep only those that are then found to be most important for the analysis. In the following plot, we use blue for class 0, red for class 1, yellow for class 2 and darkgreen for class 3.

```
# check for non linear relations
subset_for_plot <- phones_train[my_smp, c(1, 7, 12, 13, 14, 21)]
cols <- character(nrow(subset_for_plot))
cols[] <- "black"

cols[subset_for_plot$price_range == 0] <- "blue"
cols[subset_for_plot$price_range == 1] <- "red"
cols[subset_for_plot$price_range == 2] <- "yellow"
cols[subset_for_plot$price_range == 3] <- "darkgreen"

pairs(subset_for_plot,col = cols, pch = 20)
```



As we can see, there seem to be no important nonlinear relationships among the features. In addition, the ram plots clearly show a strong correlation with the *price_range*, confirming what we already discussed above. In the following section we begin the actual analysis by training several algorithms.

4. Training phase

In this section we implement and discuss some different classification models: *Multinomial Logistic Regression*, *Linear Discriminant Analysis*, *Quadratic Discriminant Analysis* and *K-Nearest Neighbours*. For each of these models we will initially use all features, then we will remove the seemingly superfluous ones through *Feature Selection* strategies. In addition, for the *Multinomial Logistic Regression*, *Linear Discriminant Analysis* and *Quadratic Discriminant Analysis* it will be necessary to select a baseline; all of these models automatically set it by default considering the first or last class.

4.1 Multinomial Logistic Regression

The first classification model we are going to test is *Multinomial Logistic Regression*, which is a multiclass generalization of *Logistic Regression*. We start by fitting the complete model, considering all features, and then move on to discard seemingly superfluous ones through *feature selection* strategies. Specifically, we will first implement Backward Selection strategies, and later Forward Selection strategies as well. Although high test accuracies were achieved by all these models, as we will see Backward Selection is still able to gain a few percentage points in accuracy. To avoid making the report too heavy, in the following we will show only the final models achieved. Following feature selection, we will also take a look at the behavior of the regularized models by considering L_1 (Lasso) and L_2 (Ridge) penalty terms.

We therefore begin by training the model considering the whole set of features.

```r
# original model with all predictors

# create formula to fit the model
features = paste(names(X_train), collapse='+')
formula = paste(c('y_train', features), collapse='~')

# Fit the Multinomial Logistic Regression model
mlr_full <- nnet::multinom(formula, data = X_train_scaled)
```

```
## # weights:  88 (63 variable)
## initial  value 2218.070978
## iter  10 value 1277.691086
## iter  20 value 887.289820
## iter  30 value 477.989101
## iter  40 value 201.456862
## iter  50 value 48.677998
## iter  60 value 28.115419
## iter  70 value 23.934800
## iter  80 value 22.031331
## iter  90 value 20.474564
## iter 100 value 19.877086
## final  value 19.877086
## stopped after 100 iterations
```

```r
# Summarize the model
summary(mlr_full)
```

```
## Call:
## nnet::multinom(formula = formula, data = X_train_scaled)
##
## Coefficients:
##   (Intercept) battery_power       blue clock_speed  dual_sim        fc
## 1   -839.4675      439.8764  0.4048624   -8.443251 -2.055487 -18.56521
## 2  -1610.9195      698.9925  8.5720110  -12.781196 -9.140219 -21.42978
## 3  -3135.7108     1056.6761 -4.3864897   -7.349001 -5.623333 -25.39536
##      four_g int_memory      m_dep  mobile_wt  n_cores       pc px_height
## 1  0.228228   6.360582  -5.331248  -75.93596 25.04467 13.14716  332.5864
## 2 -2.927443  25.571703 -11.733420 -111.41123 27.89814 26.18338  517.8058
## 3 -6.218032  60.669503   9.150073 -196.43517 30.69683 51.05919  816.4083
##   px_width      ram      sc_h      sc_w talk_time   three_g touch_screen
## 1 247.3235 1777.270 -12.48798 -27.10864 -8.398309  1.303584   -0.1540221
## 2 400.6393 2797.539 -19.39316 -19.16344  2.548584  1.755209    1.3617225
## 3 605.9899 4338.893 -10.98070  15.54196 -2.296476 -6.668005   -0.1422304
##         wifi
## 1  -4.616036
## 2  -9.148559
## 3 -11.265564
##
## Std. Errors:
##   (Intercept) battery_power      blue clock_speed dual_sim       fc    four_g
## 1    279.2618     148.24029  7.254372    16.42836 11.26550 23.48453  9.591306
## 2    328.7353     126.78686  8.339997    16.83624 11.73956 23.74706 10.043473
## 3    565.5077      70.20865 14.428993    24.85522 16.47316 31.51255 11.927652
```

```
##   int_memory    m_dep mobile_wt  n_cores       pc px_height px_width      ram
## 1   22.37570 16.35210  31.40027 16.20565 25.14854 113.25961 81.17585 581.7270
## 2   23.20793 16.36193  26.30871 16.14059 24.41326  95.53626 72.97222 497.0152
## 3   39.51310 30.01583  30.11588 27.55906 26.69748  72.65647 43.51209 329.6423
##       sc_h     sc_w talk_time   three_g touch_screen     wifi
## 1 13.03201 32.99152  9.496665  8.923353     5.023077 6.833652
## 2 12.98140 33.45506 13.496207  9.221318     5.782042 7.489794
## 3 24.59456 46.79688 26.880899 12.681946    14.273419 8.448962
##
## Residual Deviance: 39.75417
## AIC: 165.7542
```

```
# Make predictions on Training set
y_train_pred <- predict(mlr_full, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred == y_train),"\n")
```

```
## Training set accuracy:  0.995
```

```
# Make predictions on Test set
y_test_pred <- predict(mlr_full, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred == y_test))
```

```
## Test set accuracy:  0.975
```

Unfortunately, the *multinom()* function of the *nnet* package does not automatically provide the p-values. Let us compute them manually:

```
# let's compute the p-values
z <- summary(mlr_full)$coefficients/summary(mlr_full)$standard.errors
p <- (1 - pnorm(abs(z), 0, 1)) * 2
p
```

```
##    (Intercept) battery_power      blue clock_speed  dual_sim        fc
## 1 2.646884e-03  3.004078e-03 0.9554936   0.6072915 0.8552228 0.4292186
## 2 9.566388e-07  3.525070e-08 0.3040347   0.4477640 0.4362254 0.3668347
## 3 2.940423e-08  0.000000e+00 0.7611239   0.7674803 0.7328300 0.4203115
##      four_g int_memory     m_dep    mobile_wt    n_cores         pc
## 1 0.9810159  0.7762089 0.7444029 1.559226e-02 0.12224268 0.60112716
## 2 0.7706864  0.2705259 0.4733018 2.287898e-05 0.08390818 0.28349265
## 3 0.6021492  0.1246788 0.7604868 6.907896e-11 0.26534085 0.05581082
##      px_height     px_width          ram      sc_h      sc_w talk_time
## 1 3.319432e-03 2.313206e-03 2.249397e-03 0.3379346 0.4112562 0.3765111
## 2 5.960172e-08 4.012543e-08 1.815950e-08 0.1351967 0.5667725 0.8502205
## 3 0.000000e+00 0.000000e+00 0.000000e+00 0.6552588 0.7398023 0.9319183
##     three_g touch_screen      wifi
## 1 0.8838529    0.9755384 0.4993670
## 2 0.8490407    0.8138137 0.2219081
## 3 0.5990361    0.9920494 0.1824115
```

Now we use these p-values to put Backward Selection into practice: we proceed recursively, discarding the least significant predictor at each iteration. We obtain the final model, consisting only of the features: *battery_power*, *int_memory*, *mobile_wt*, *n_cores*, *px_height*, *px_width* and *ram*.

```
# Final model after BACKWARD SELECTION

# create formula to fit the model
features = paste(names(X_train[,-c(19, 18, 2, 8, 6, 15, 17, 4, 16, 5, 3, 11, 20)]), collapse='+')
formula = paste(c('y_train', features), collapse='~')

# Fit the Multinomial Logistic Regression model
mlr_BS <- nnet::multinom(formula, data = X_train_scaled)
```

```
## # weights:  36 (24 variable)
## initial  value 2218.070978
## iter  10 value 1188.324773
## iter  20 value 461.451714
## iter  30 value 66.418975
## iter  40 value 57.846941
## iter  50 value 57.110731
## iter  60 value 57.061265
## iter  70 value 56.990518
## iter  80 value 56.666449
## iter  90 value 56.639223
## iter 100 value 56.597053
## final  value 56.597053
## stopped after 100 iterations
```

```
# Summarize the model
summary(mlr_BS)
```

```
## Call:
## nnet::multinom(formula = formula, data = X_train_scaled)
##
## Coefficients:
##   (Intercept) battery_power int_memory mobile_wt  n_cores px_height  px_width
## 1   -187.8914      93.46961   2.177938 -17.48217 4.937836  72.61047  56.44485
## 2   -353.7594     149.15827   5.515712 -25.20377 6.866416 114.03043  89.15818
## 3   -621.7641     215.61717  10.865856 -40.63061 6.801808 168.90756 126.71127
##        ram
## 1 386.9263
## 2 607.7199
## 3 880.1623
##
## Std. Errors:
##   (Intercept) battery_power int_memory mobile_wt  n_cores px_height px_width
## 1    51.47416      25.60513   1.879752  5.632043 2.211041  19.43307 16.18974
## 2    60.60354      27.77546   2.438682  5.944947 2.487239  21.09749 17.44186
## 3    86.34515      31.70580   3.235302  7.110513 3.098373  24.78438 19.42132
##        ram
## 1 106.3574
## 2 114.5363
## 3 130.6586
```

```
##
## Residual Deviance: 113.1941
## AIC: 161.1941
```

```
# Make predictions on Training set
y_train_pred <- predict(mlr_BS, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred == y_train),"\n")
```

```
## Training set accuracy:  0.985
```

```
# Make predictions on Test set
y_test_pred <- predict(mlr_BS, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred == y_test))
```

```
## Test set accuracy:  0.9775
```

As can be seen below, the remaining features are all significant:

```
# let's compute the p-values
z <- summary(mlr_BS)$coefficients/summary(mlr_BS)$standard.errors
p <- (1 - pnorm(abs(z), 0, 1)) * 2
p
```

```
##    (Intercept) battery_power   int_memory    mobile_wt     n_cores    px_height
## 1 2.620276e-04  2.618070e-04 0.2466069456 1.908886e-03 0.025531606 1.866447e-04
## 2 5.306226e-09  7.867339e-08 0.0237122909 2.239898e-05 0.005768513 6.483453e-08
## 3 5.981882e-13  1.042144e-11 0.0007835853 1.102469e-08 0.028143255 9.421131e-12
##       px_width          ram
## 1 4.894601e-04 2.747841e-04
## 2 3.192139e-07 1.121086e-07
## 3 6.830203e-11 1.624145e-11
```

In particular, with Backward Selection we were able to greatly simplify the model and slightly increase the accuracy on test set, moving from 97.5% to 97.75%. We now proceed with Forward Selection: starting from the model made only of the intercept, we use the cross validation error to iteratively add the most relevant features.

```
# define function 'not in'. It will be useful for Forward Selection
 `%nin%` = Negate(`%in%`)
```

```
# FORWARD SELECTION

# From X_train take 25% of samples to use as validation set to monitor cross validation error
set.seed(1)
training_smp = sample.int(nrow(X_train), nrow(X_train)*75/100, replace=FALSE)
#validation_smp = setdiff(1:nrow(X_train), training_smp)
```

```
for( i in 1:length(X_train) ){

  if (i %nin% c(14, 12, 1, 13, 9, 2)){

    # create formula to fit the model
    features = paste(names(X_train[, c(14, 12, 1, 13, 9, 2, i)]), collapse='+')
    formula = paste(c('y_train', features), collapse='~')

    cat('\n',i, '\n')
    # Fit the Multinomial Logistic Regression model
    mlr <- nnet::multinom(formula, data = X_train_scaled, subset = training_smp)
    # Summarize the model
    summary(mlr)


    # Make predictions on Training set
    y_train_pred <- predict(mlr, X_train_scaled[training_smp, ])

    # Model accuracy on Training set
    cat("Training set accuracy: ", mean(y_train_pred == y_train[training_smp]),"\n")

    # Make predictions on Validation set
    y_val_pred <- predict(mlr, X_train_scaled[-training_smp, ])

    # Model accuracy on Validation set
    cat("Validation set accuracy: ", mean(y_val_pred == y_train[-training_smp]))

  }
}
```

We obtain the final model, consisting only of the following features: *battery_power*, *mobile_wt*, *px_height*, *px_width* and *ram*.

```
# final model after FORWARD SELECTION

# create formula to fit the model
features = paste(names(X_train[, c(14, 12, 1, 13, 9)]), collapse='+')
formula = paste(c('y_train', features), collapse='~')

# Fit the Multinomial Logistic Regression model
mlr_FS <- nnet::multinom(formula, data = X_train_scaled)
```

```
## # weights:  28 (18 variable)
## initial  value 2218.070978
## iter  10 value 1167.539634
## iter  20 value 222.718813
## iter  30 value 75.344704
## iter  40 value 71.518712
## iter  50 value 71.428100
## iter  60 value 71.166474
## iter  70 value 71.024605
## iter  80 value 71.011600
## iter  90 value 70.992078
```

```
## iter 100 value 70.896104
## final  value 70.896104
## stopped after 100 iterations
```

```
# Summarize the model
summary(mlr_FS)
```

```
## Call:
## nnet::multinom(formula = formula, data = X_train_scaled)
##
## Coefficients:
##   (Intercept)      ram px_height battery_power  px_width mobile_wt
## 1   -140.7067 294.7713  55.97313      71.01221  42.90348 -12.73728
## 2   -287.9978 494.2833  92.81187     121.21830  72.27432 -19.47551
## 3   -494.3763 705.9263 133.68357     172.90275 102.78430 -31.04239
##
## Std. Errors:
##   (Intercept)      ram px_height battery_power px_width mobile_wt
## 1    34.06443 71.69367  13.74527      17.21519 10.70346  3.641781
## 2    43.60590 80.68934  15.38656      19.54904 12.10970  4.026600
## 3    58.47986 90.04983  17.30622      21.84864 13.48824  4.763260
##
## Residual Deviance: 141.7922
## AIC: 177.7922
```

```
# Make predictions on Training set
y_train_pred <- predict(mlr_FS, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred == y_train),"\n")
```

```
## Training set accuracy:  0.983125
```

```
# Make predictions on Test set
y_test_pred <- predict(mlr_FS, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred == y_test))
```

```
## Test set accuracy:  0.9675
```

Specifically in this case we notice instead a decrease in accuracy, going from 97.5% to 96.75% on the test set. However, the obtained model is considerably simpler since only five features are considered. Before proceeding with the regularized models, we also try to fit a reduced model considering only the feature related to *ram*, since from the initial analysis it was found to be the variable most correlated with the *price_range*.

```
# Fit the model
mlr_ram <- nnet::multinom(y_train ~ ram, data = X_train_scaled)
```

```
## # weights:  12 (6 variable)
```

```
## initial  value 2218.070978
## iter  10 value 992.733794
## iter  20 value 915.005288
## final  value 914.948431
## converged
```

```
# Summarize the model
summary(mlr_ram)
```

```
## Call:
## nnet::multinom(formula = y_train ~ ram, data = X_train_scaled)
##
## Coefficients:
##   (Intercept)      ram
## 1   -4.375725 17.04859
## 2  -11.664750 31.51739
## 3  -22.137188 45.70063
##
## Std. Errors:
##   (Intercept)      ram
## 1   0.3133806 1.170349
## 2   0.6124186 1.574419
## 3   0.9389114 1.844236
##
## Residual Deviance: 1829.897
## AIC: 1841.897
```

```
# Make predictions on Training set
y_train_pred <- predict(mlr_ram, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred == y_train),"\n")
```

```
## Training set accuracy:  0.749375
```

```
# Make predictions on Test set
y_test_pred <- predict(mlr_ram, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred == y_test))
```

```
## Test set accuracy:  0.7875
```

Contrary to what one could expect from the preprocessing phase, *ram* is actually not the only relevant feature. Indeed, we can see that in this second case the accuracy obtained is significantly reduced, by about 20%.

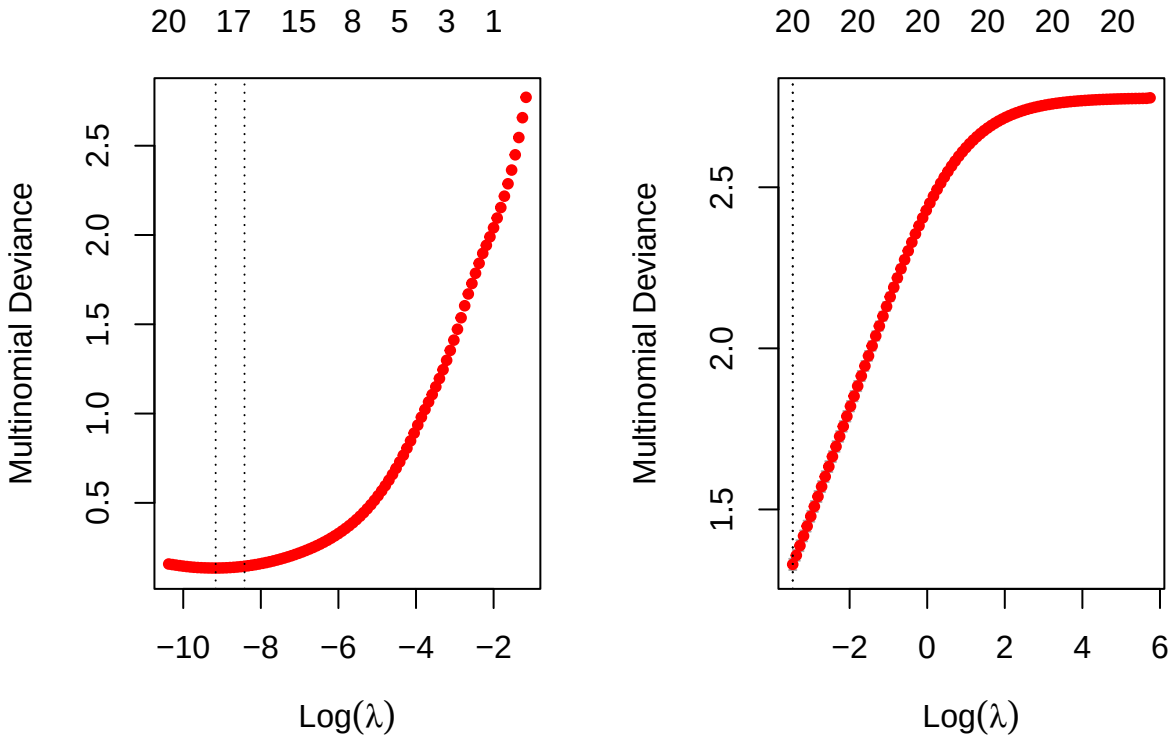
Finally, we also take a look at the regularized models (Lasso and Ridge Regression, respectively). To find the optimal hyperparameter λ we will use cross-validation strategies.

```
library(glmnet)
```

```
## Caricamento del pacchetto richiesto: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
# Lasso Regression
mlr_lasso = cv.glmnet(as.matrix(X_train_scaled), y_train, alpha=1, family="multinomial")

# Ridge Regression
mlr_ridge = cv.glmnet(as.matrix(X_train_scaled), y_train, alpha=0, family="multinomial")

par(mfrow = c(1, 2))

plot(mlr_lasso)#, main ='Lasso Regularization')
plot(mlr_ridge)#, main ='Ridge Regularization')
```




```
par(mfrow = c(1, 1))
```

```
cf_lasso = coef(mlr_lasso, s = mlr_lasso$lambda.min)
cf_ridge = coef(mlr_ridge, s = mlr_ridge$lambda.min)
```

Coefficients found by Lasso Regularization:

```
cf_lasso
```

```
## $`0`
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept)    140.74873105
## battery_power  -53.13879526
## blue             .
## clock_speed      0.37378980
## dual_sim         0.39819432
## fc               0.20247065
## four_g           0.42319542
## int_memory      -1.58523346
## m_dep           -0.00697961
## mobile_wt        7.53961439
## n_cores         -2.05583633
## pc               -0.68883352
## px_height      -41.58158044
## px_width       -30.34491360
## ram           -214.32810710
## sc_h             0.34760818
## sc_w             .
## talk_time        0.39037075
## three_g          0.64636863
## touch_screen     0.21321851
## wifi             1.33303013
##
## $`1`
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                             1
## (Intercept)    67.09398412
## battery_power -14.83144533
## blue            0.03327733
## clock_speed    -0.41528424
## dual_sim        .
## fc              0.22294283
## four_g          0.03591324
## int_memory     -0.73167558
## m_dep           0.00697961
## mobile_wt       1.75663155
## n_cores        -0.46955693
## pc             -0.70406639
## px_height     -10.66658104
## px_width       -8.99237931
## ram           -58.80612752
## sc_h           -0.34760818
## sc_w           -0.24716933
## talk_time      -0.57071339
## three_g         0.04324831
## touch_screen   -0.04494649
## wifi            0.41051238
##
## $`2`
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)   -21.10754579
## battery_power  14.83144533
## blue            .
## clock_speed    -0.37378980
## dual_sim       -0.06220973
## fc             -0.20247065
## four_g         -0.13981087
## int_memory      0.73167558
## m_dep          -1.12299503
## mobile_wt      -1.75663155
## n_cores         0.46955693
## pc              0.68883352
## px_height      10.66658104
## px_width        8.99237931
## ram            58.80612752
## sc_h           -0.52208335
## sc_w            .
## talk_time       .
## three_g        -0.04324831
## touch_screen   -0.27056344
## wifi           -0.41051238
##
## $`3`
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                             1
## (Intercept)   -186.73516937
## battery_power   54.52306355
## blue            -0.85455936
## clock_speed      0.43917777
## dual_sim         .
## fc              -0.27131655
## four_g          -0.03591324
## int_memory       4.31434971
## m_dep            0.26252784
## mobile_wt      -11.08503848
## n_cores          1.53942860
## pc               3.16564323
## px_height       43.53697676
## px_width        32.61688223
## ram            224.63513341
## sc_h             1.09771688
## sc_w             0.61443945
## talk_time        .
## three_g         -0.71922533
## touch_screen     0.04494649
## wifi            -1.36312072
```

Coefficients found by Ridge Regularization:

```
cf_ridge
```

```
## $`0`
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)    4.997911303
## battery_power -1.712773753
## blue           0.012962692
## clock_speed    0.134382145
## dual_sim       0.037551812
## fc            -0.172773539
## four_g        -0.007419102
## int_memory    -0.135028957
## m_dep         -0.018993511
## mobile_wt      0.155188663
## n_cores        0.073225560
## pc            -0.139439030
## px_height     -1.586662138
## px_width      -0.960734268
## ram           -7.236078084
## sc_h           0.056741156
## sc_w          -0.060898115
## talk_time     -0.159252074
## three_g       -0.083630005
## touch_screen   0.062654966
## wifi           0.001511575
##
## $`1`
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)    1.835043722
## battery_power -0.364944986
## blue           0.022411728
## clock_speed   -0.103791198
## dual_sim      -0.031792963
## fc             0.121564264
## four_g        -0.014862462
## int_memory     0.036275437
## m_dep          0.194394525
## mobile_wt     -0.033480553
## n_cores       -0.159071671
## pc             0.069898692
## px_height     -0.050329713
## px_width      -0.338573409
## ram           -1.989130286
## sc_h          -0.015728478
## sc_w          -0.049285367
## talk_time      0.104661970
## three_g        0.020731135
## touch_screen  -0.007330306
## wifi           0.022646470
##
## $`2`
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)   -0.687653529
## battery_power  0.252861688
```

```
## blue          -0.074470822
## clock_speed    0.007363024
## dual_sim      -0.035679208
## fc             0.151841575
## four_g        -0.059515333
## int_memory    -0.140193393
## m_dep         -0.158380540
## mobile_wt      0.278896998
## n_cores        0.067016301
## pc            -0.056580015
## px_height      0.349008908
## px_width       0.061597922
## ram            2.045883523
## sc_h          -0.197425704
## sc_w           0.036570575
## talk_time      0.026998997
## three_g        0.088923488
## touch_screen  -0.067621780
## wifi           0.006891778
##
## $`3`
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)   -6.14530150
## battery_power  1.82485705
## blue           0.03909640
## clock_speed   -0.03795397
## dual_sim       0.02992036
## fc            -0.10063230
## four_g         0.08179690
## int_memory     0.23894691
## m_dep         -0.01702047
## mobile_wt     -0.40060511
## n_cores        0.01882981
## pc             0.12612035
## px_height      1.28798294
## px_width       1.23770975
## ram            7.17932485
## sc_h           0.15641303
## sc_w           0.07361291
## talk_time      0.02759111
## three_g       -0.02602462
## touch_screen   0.01229712
## wifi          -0.03104982
```

Below we calculate accuracies on training and test sets, both for Lasso and Ridge:

```
# LASSO predictions

# Make predictions on Training set
y_train_pred <- predict(mlr_lasso, newx = as.matrix(X_train_scaled), s = "lambda.min",
                        type = "class")

# Model accuracy on Training set
```

```r
cat("Training set accuracy with LASSO: ", mean(y_train_pred == y_train),"\n")
```

```
## Training set accuracy with LASSO:  0.989375
```

```r
# Make predictions on Test set
y_test_pred <- predict(mlr_lasso, newx = as.matrix(X_test_scaled), s = "lambda.min",
                       type = "class")

# Model accuracy on Test set
cat("Test set accuracy with LASSO: ", mean(y_test_pred == y_test))
```

```
## Test set accuracy with LASSO:  0.975
```

```r
# RIDGE predictions

# Make predictions on Training set
y_train_pred <- predict(mlr_ridge , newx = as.matrix(X_train_scaled), s = "lambda.min",
                        type = "class")

# Model accuracy on Training set
cat("Training set accuracy with RIDGE: ", mean(y_train_pred == y_train),"\n")
```

```
## Training set accuracy with RIDGE:  0.875625
```

```r
# Make predictions on Test set
y_test_pred <- predict(mlr_ridge, newx = as.matrix(X_test_scaled), s = "lambda.min",
                       type = "class")

# Model accuracy on Test set
cat("Test set accuracy with RIDGE: ", mean(y_test_pred == y_test))
```

```
## Test set accuracy with RIDGE:  0.8525
```

As we can see, the accuracy achieved by Lasso is about the same as the one obtained with the model made of all features. Ridge on the other hand performed considerably worse, losing about 10% in accuracy.

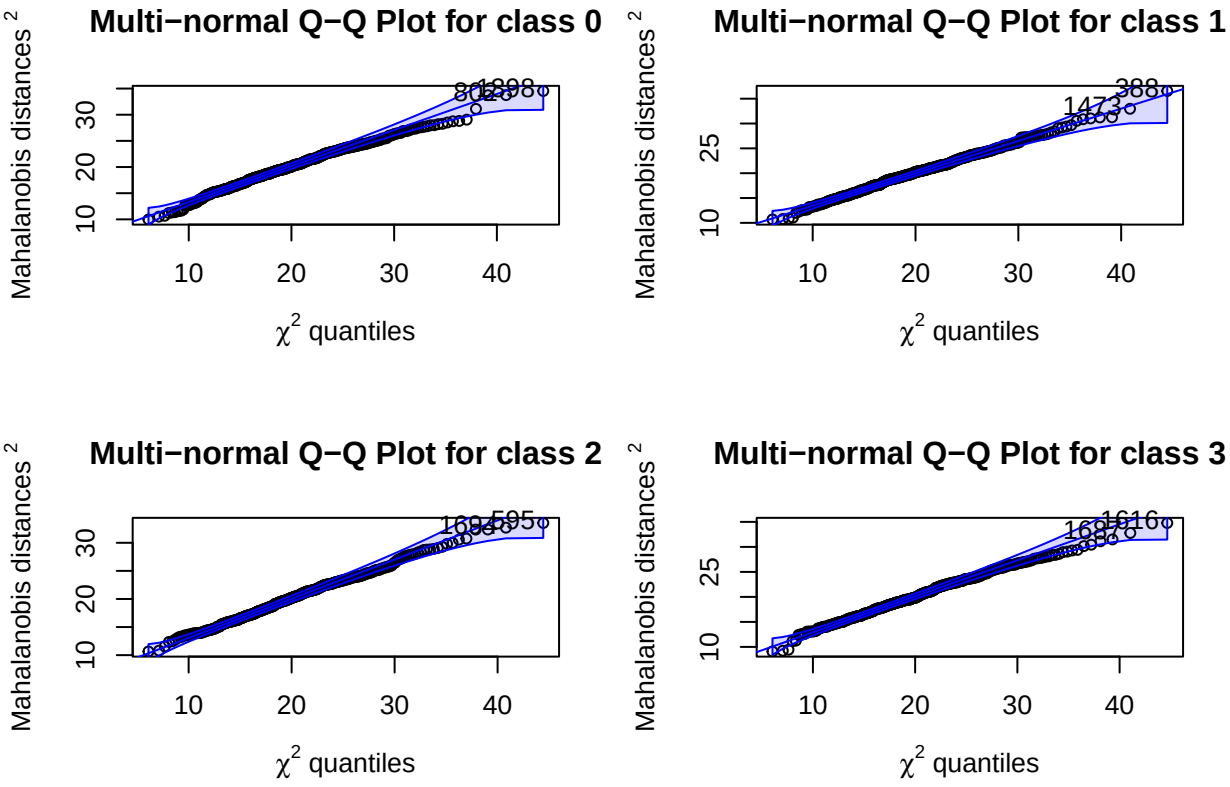
4.2 Linear Discriminant Analysis

In this section we implement *Linear Discriminant Analysis*. As in the previous case, we start by using all the provided features, and then apply Forward Selection to select the most useful ones. Before implementing this algorithm though, we need to check whether the data within each class approximately follows a multivariate normal distribution. To do this we use the multivariate *QQ-plots*, given below:

```r
library('RVAideMemoire')
```

```
## *** Package RVAideMemoire v 0.9-81-2 ***
```

```
par(mfrow = c(2,2))

mqqnorm(X_train[y_train == 0, ], main = "Multi-normal Q-Q Plot for class 0")
mqqnorm(X_train[y_train == 1, ], main = "Multi-normal Q-Q Plot for class 1")
mqqnorm(X_train[y_train == 2, ], main = "Multi-normal Q-Q Plot for class 2")
mqqnorm(X_train[y_train == 3, ], main = "Multi-normal Q-Q Plot for class 3")
```

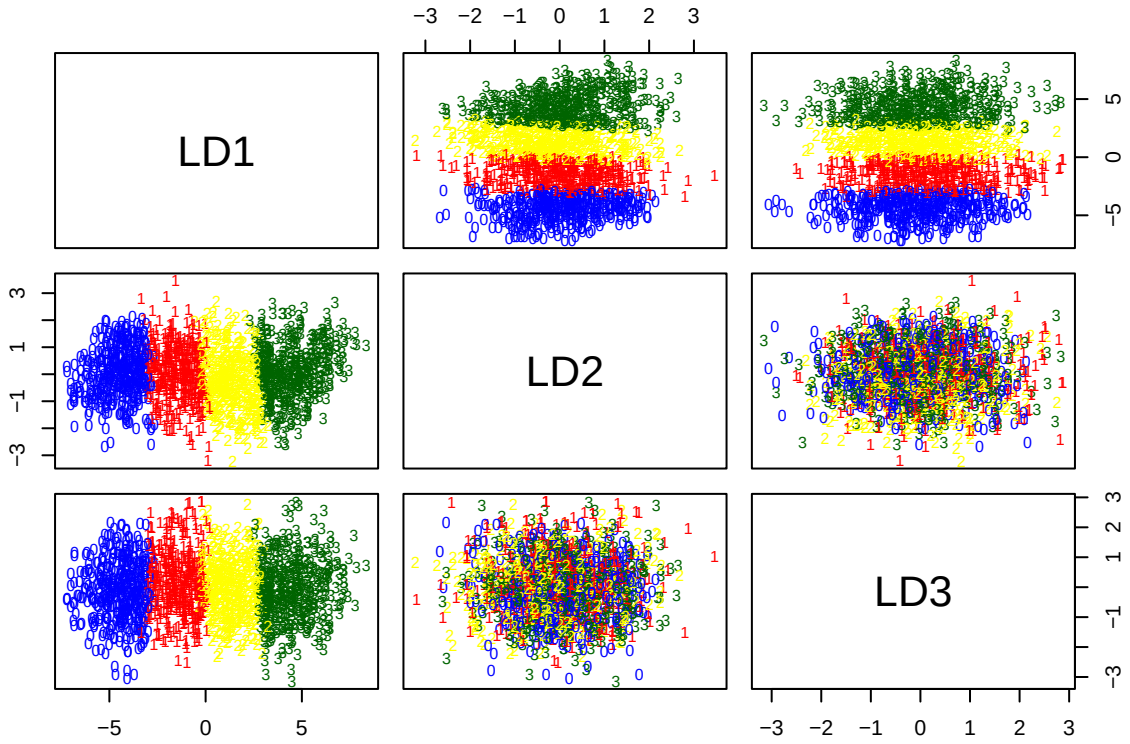



```
par(mfrow = c(1,1))
```

From the figures above we can see that the data appears to follow a multivariate normal distribution. We then proceed with the implementation of *Linear Discriminant Analysis*; initially we keep the entire feature set.

```
library(MASS)
```

```
# original model considering all features
LDA <- lda(y_train~. , data = X_train_scaled)

# plot LDA
plot(LDA, col = c('blue','red','yellow','darkgreen')[factor(y_train)])
```




```
# Make predictions on Training set
y_train_pred <- predict(LDA, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred$class == y_train),"\n")
```

```
## Training set accuracy:  0.9525
```

```
# plot confusion matrix for Training set
cat("Confusion Matrix for the Training set:", "\n")
```

```
## Confusion Matrix for the Training set:
```

```
CM_train <- table(y_train_pred$class, y_train)
CM_train
```

```
##    y_train
##       0   1   2   3
##   0 388   9   0   0
##   1  11 375  25   0
##   2   0  11 361  17
##   3   0   0   3 400
```

```
cat("\n\n")
```

```
# Make predictions on Test set
y_test_pred <- predict(LDA, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred$class == y_test),'\n')
```

```
## Test set accuracy:  0.9625
```

```
# plot confusion matrix for Test set
cat("Confusion Matrix for the Test set:", "\n")
```

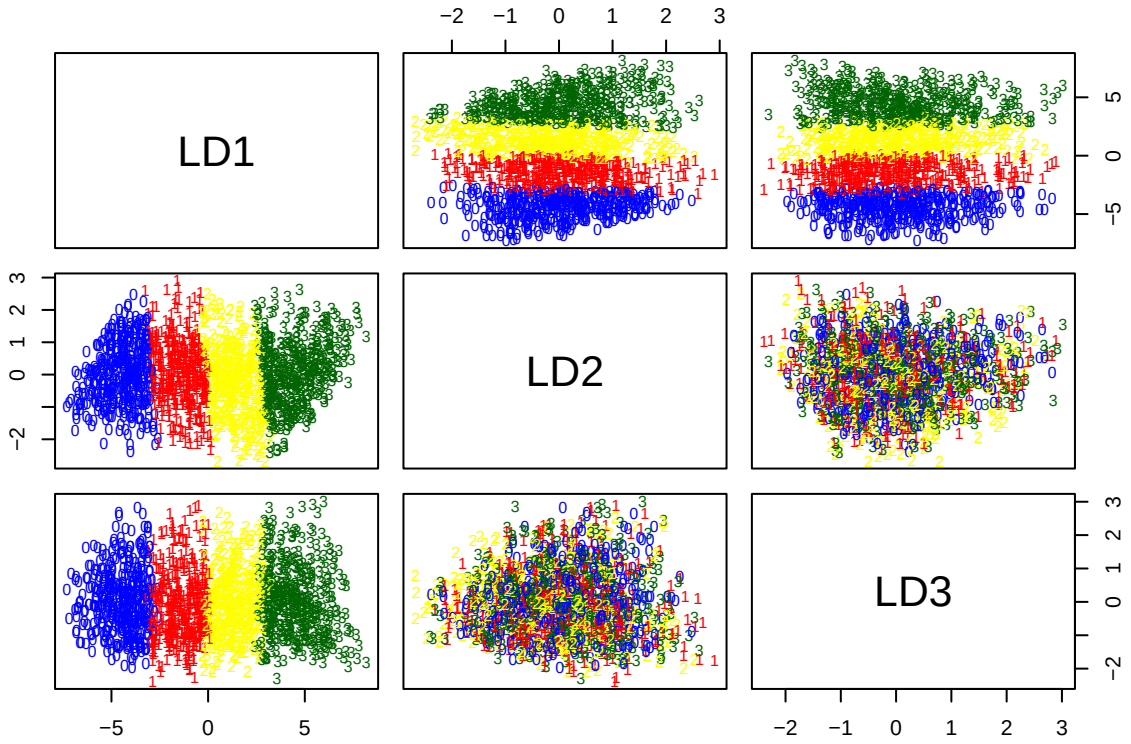
```
## Confusion Matrix for the Test set:
```

```
CM_test <- table(y_test_pred$class, y_test)
CM_test
```

```
##    y_test
##       0   1   2   3
##   0  99   1   0   0
##   1   2 103   6   0
##   2   0   1 104   4
##   3   0   0   1  79
```

Now we try to implement the model using the same features selected for Multinomial Logistic Regression through Forward Selection:

```
# Linear Discriminant Analysis with the same features
# found for the Multinomial Logistic Regression model, through Forward Selection

# Fit the Linear Discriminant Analysis model
LDA <- lda(y_train~ram+px_height+battery_power+px_width+mobile_wt, data = X_train_scaled)

# plot LDA
plot(LDA, col = c('blue','red','yellow','darkgreen')[factor(y_train)])
```




```
# Make predictions on Training set
y_train_pred <- predict(LDA, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred$class == y_train),"\n")
```

```
## Training set accuracy:  0.95375
```

```
# plot confusion matrix for Training set
cat("Confusion Matrix for the Training set:", "\n")
```

```
## Confusion Matrix for the Training set:
```

```
CM_train <- table(y_train_pred$class, y_train)
CM_train
```

```
##    y_train
##       0   1   2   3
##   0 391   7   0   0
##   1   8 378  26   0
##   2   0  10 359  19
##   3   0   0   4 398
```

```
cat("\n\n")
```

```
# Make predictions on Test set
y_test_pred <- predict(LDA, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred$class == y_test),'\n')
```

```
## Test set accuracy:  0.9625
```

```
# plot confusion matrix for Test set
cat("Confusion Matrix for the Test set:", "\n")
```

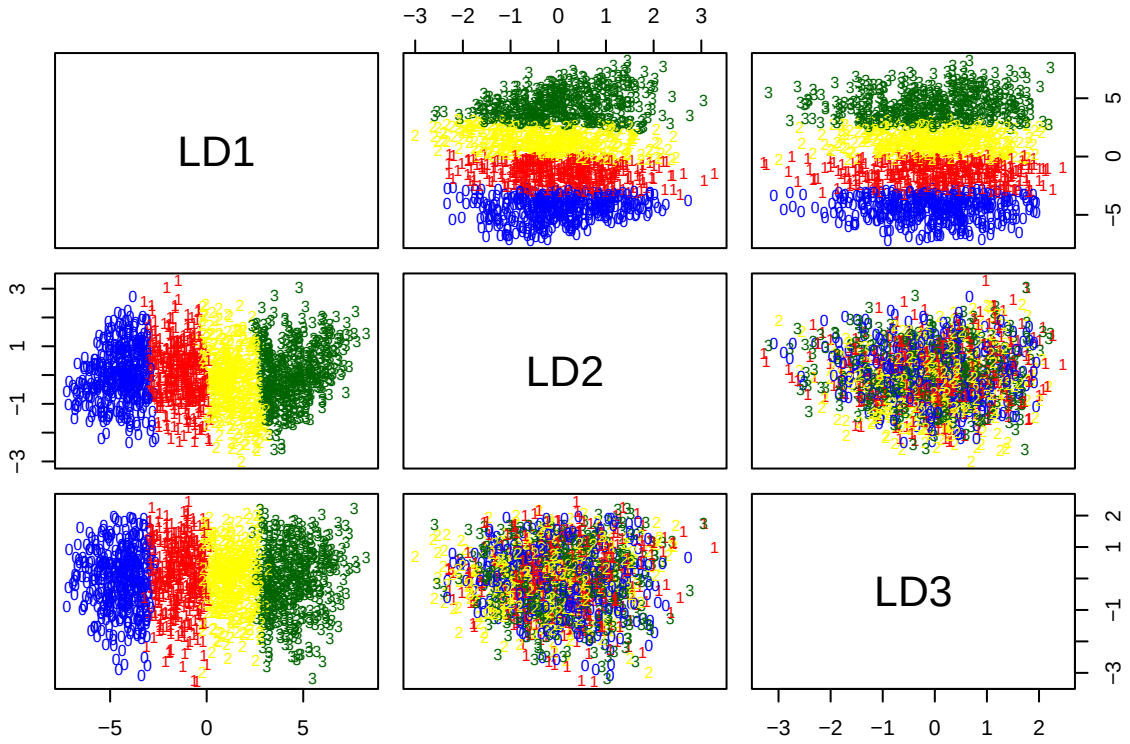
```
## Confusion Matrix for the Test set:
```

```
CM_test <- table(y_test_pred$class, y_test)
CM_test
```

```
##    y_test
##       0   1   2   3
##   0  98   0   0   0
##   1   3 105   7   0
##   2   0   0 104   5
##   3   0   0   0  78
```

We now implement *Linear Discriminant Analysis* using the same features selected for Multinomial Logistic Regression through backward selection:

```
# Linear Discriminant Analysis with the same features found
# for the Multinomial Logistic Regression model, through Backward Selection

# Fit the Linear Discriminant Analysis model
LDA <- lda(y_train~ram+battery_power+int_memory+mobile_wt+n_cores+px_height+px_width,
           data = X_train_scaled)

# plot LDA
plot(LDA, col = c('blue','red','yellow','darkgreen')[factor(y_train)])
```




```
# Make predictions on Training set
y_train_pred <- predict(LDA, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred$class == y_train),"\n")
```

```
## Training set accuracy:  0.95
```

```
# plot confusion matrix for Training set
cat("Confusion Matrix for the Training set:", "\n")
```

```
## Confusion Matrix for the Training set:
```

```
CM_train <- table(y_train_pred$class, y_train)
CM_train
```

```
##    y_train
##       0   1   2   3
##   0 391   7   0   0
##   1   8 377  25   0
##   2   0  11 359  24
##   3   0   0   5 393
```

```
cat("\n\n")
```

```
# Make predictions on Test set
y_test_pred <- predict(LDA, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred$class == y_test),'\n')
```

```
## Test set accuracy:  0.9625
```

```
# plot confusion matrix for Test set
cat("Confusion Matrix for the Test set:", "\n")
```

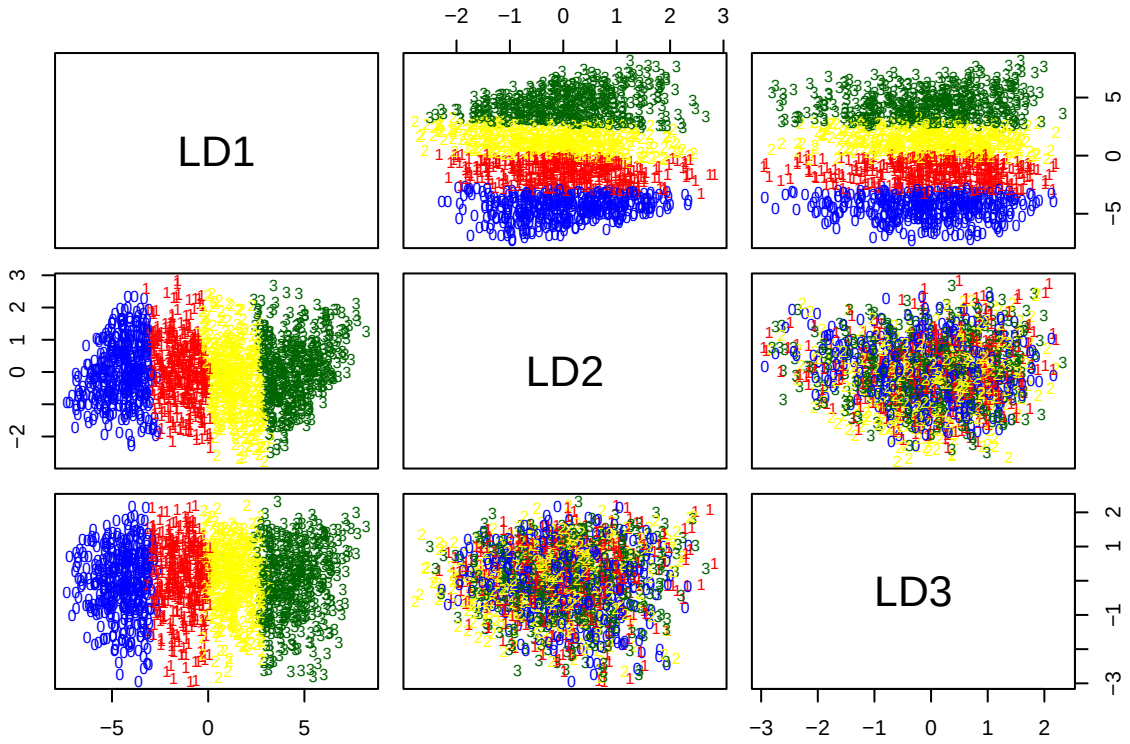
```
## Confusion Matrix for the Test set:
```

```
CM_test <- table(y_test_pred$class, y_test)
CM_test
```

```
##    y_test
##       0   1   2   3
##   0  99   0   0   0
##   1   2 104   7   0
##   2   0   1 104   5
##   3   0   0   0  78
```

Finally we proceed with Forward Selection. It will be interesting to check whether the most important features for Linear Discriminant Analysis coincide with those for Multinomial Logistic Regression, or whether there are variations. In order not to make the report too heavy to read, below the script with only the final features is given.

```
# final model after FORWARD SELECTION

# Fit the Linear Discriminant Analysis model
LDA <- lda(y_train~ram+battery_power+px_height+px_width+mobile_wt+wifi+sc_w, data = X_train_scaled)

# plot LDA
plot(LDA, col = c('blue','red','yellow','darkgreen')[factor(y_train)])
```




```
# Make predictions on Training set
y_train_pred <- predict(LDA, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred$class == y_train),"\n")
```

```
## Training set accuracy:  0.95875
```

```
# plot confusion matrix for Training set
cat("Confusion Matrix for the Training set:", "\n")
```

```
## Confusion Matrix for the Training set:
```

```
CM_train <- table(y_train_pred$class, y_train)
CM_train
```

```
##    y_train
##       0   1   2   3
##   0 392   8   0   0
##   1   7 377  21   0
##   2   0  10 367  19
##   3   0   0   1 398
```

```
cat("\n\n")
```

```
# Make predictions on Test set
y_test_pred <- predict(LDA, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred$class == y_test),'\n')
```

```
## Test set accuracy:  0.975
```

```
# plot confusion matrix for Test set
cat("Confusion Matrix for the Test set:", "\n")
```

```
## Confusion Matrix for the Test set:
```

```
CM_test <- table(y_test_pred$class, y_test)
CM_test
```

```
##    y_test
##       0   1   2   3
##   0 101   0   0   0
##   1   0 105   5   0
##   2   0   0 106   5
##   3   0   0   0  78
```

To conclude, we notice that all the different models we tested perform relatively well, achieving high accuracies both on training and test sets. The best model (in terms of test set accuracy) obtained is the one in which Forward Selection is applied. In this case the accuracy achieved on the test set is almost the same as the one we obtained with Multinomial Logistic Regression with Backward Selection. However, the feature sets retained in these two models turn out to be different. In the following section we will go on to implement a generalization of *Linear Discriminant Analysis*, namely *Quadratic Discriminant Analysis*.

4.3 Quadratic Discriminant Analysis

An important assumption of the LDA is that all groups share the same variance matrix. In our case however, while similar on many of the entries, these matrices still have some significantly different values. This prompted us to also test the *Quadratic Discriminant Analysis*, which does not assume equal variance across groups instead. As in the previous case, we will start by testing the full model with all features, we will then go on to test the reduced models considering only the feature sets obtained by: Backward/Forward Selection on the Multinomial Logistic Regression, Forward Selection on the LDA, and then we will conclude by implementing the actual Forward Selection in this case as well.

```
# original model considering all features
QDA <- qda(y_train~. , data = X_train_scaled)

# Make predictions on Training set
y_train_pred <- predict(QDA, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred$class == y_train),"\n")
```

```
## Training set accuracy:  0.965
```

```
# plot confusion matrix for Training set
cat("Confusion Matrix for the Training set:", "\n")
```

```
## Confusion Matrix for the Training set:
```

```
CM_train <- table(y_train_pred$class, y_train)
CM_train
```

```
##    y_train
##       0   1   2   3
##   0 393  12   0   0
##   1   6 377  13   0
##   2   0   6 363   6
##   3   0   0  13 411
```

```
cat("\n\n")
```

```
# Make predictions on Test set
y_test_pred <- predict(QDA, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred$class == y_test),'\n')
```

```
## Test set accuracy:  0.94
```

```
# plot confusion matrix for Test set
cat("Confusion Matrix for the Test set:", "\n")
```

```
## Confusion Matrix for the Test set:
```

```
CM_test <- table(y_test_pred$class, y_test)
CM_test
```

```
##    y_test
##       0   1   2   3
##   0  98   3   0   0
##   1   3 100   5   0
##   2   0   2 100   5
##   3   0   0   6  78
```

Now let's consider a reduced model keeping just the features selected for Multinomial Logistic Regression through Forward Selection:

```
# Quadratic Discriminant Analysis with the same features found
# for the Multinomial Logistic Regression model, through Forward Selection

# Fit the Linear Discriminant Analysis model
QDA <- qda(y_train~ram+px_height+battery_power+px_width+mobile_wt, data = X_train_scaled)
```

```
# Make predictions on Training set
y_train_pred <- predict(QDA, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred$class == y_train),"\n")
```

```
## Training set accuracy:  0.97
```

```
# plot confusion matrix for Training set
cat("Confusion Matrix for the Training set:", "\n")
```

```
## Confusion Matrix for the Training set:
```

```
CM_train <- table(y_train_pred$class, y_train)
CM_train
```

```
##    y_train
##       0   1   2   3
##   0 395  12   0   0
##   1   4 374  10   0
##   2   0   9 370   4
##   3   0   0   9 413
```

```
cat("\n\n")
```

```
# Make predictions on Test set
y_test_pred <- predict(QDA, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred$class == y_test),'\n')
```

```
## Test set accuracy:  0.9625
```

```
# plot confusion matrix for Test set
cat("Confusion Matrix for the Test set:", "\n")
```

```
## Confusion Matrix for the Test set:
```

```
CM_test <- table(y_test_pred$class, y_test)
CM_test
```

```
##    y_test
##       0   1   2   3
##   0 100   2   0   0
##   1   1 101   4   0
##   2   0   2 105   4
##   3   0   0   2  79
```

We now implement *Quadratic Discriminant Analysis* using the same features selected for Multinomial Logistic Regression through Backward Selection:

```r
# Quadratic Discriminant Analysis with the same features found
# for the Multinomial Logistic Regression model, through Backward Selection

# Fit the Linear Discriminant Analysis model
QDA <- qda(y_train~ram+battery_power+int_memory+mobile_wt+n_cores+px_height+px_width,
           data = X_train_scaled)

# Make predictions on Training set
y_train_pred <- predict(QDA, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred$class == y_train),"\n")
```

```
## Training set accuracy:  0.965
```

```r
# plot confusion matrix for Training set
cat("Confusion Matrix for the Training set:", "\n")
```

```
## Confusion Matrix for the Training set:
```

```r
CM_train <- table(y_train_pred$class, y_train)
CM_train
```

```
##    y_train
##       0   1   2   3
##   0 393  15   0   0
##   1   6 373  11   0
##   2   0   7 366   5
##   3   0   0  12 412
```

```r
cat("\n\n")
```

```r
# Make predictions on Test set
y_test_pred <- predict(QDA, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred$class == y_test),'\n')
```

```
## Test set accuracy:  0.9625
```

```r
# plot confusion matrix for Test set
cat("Confusion Matrix for the Test set:", "\n")
```

```
## Confusion Matrix for the Test set:
```

```r
CM_test <- table(y_test_pred$class, y_test)
CM_test
```

```
##    y_test
##       0   1   2   3
##   0  99   1   0   0
##   1   2 102   4   0
##   2   0   2 103   2
##   3   0   0   4  81
```

Let's also test the model using the same features selected for LDA through Forward Selection.

```
# Quadratic Discriminant Analysis with the same features found
# for LDA model, through Forward Selection

# Fit the Linear Discriminant Analysis model
QDA <- qda(y_train~ram+battery_power+px_height+px_width+mobile_wt+wifi+sc_w, data = X_train_scaled)

# Make predictions on Training set
y_train_pred <- predict(QDA, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred$class == y_train),"\n")
```

```
## Training set accuracy:  0.96625
```

```
# plot confusion matrix for Training set
cat("Confusion Matrix for the Training set:", "\n")
```

```
## Confusion Matrix for the Training set:
```

```
CM_train <- table(y_train_pred$class, y_train)
CM_train
```

```
##    y_train
##       0   1   2   3
##   0 395  17   0   0
##   1   4 369   7   0
##   2   0   9 370   5
##   3   0   0  12 412
```

```
cat("\n\n")
```

```
# Make predictions on Test set
y_test_pred <- predict(QDA, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred$class == y_test),'\n')
```

```
## Test set accuracy:  0.96
```

```
# plot confusion matrix for Test set
cat("Confusion Matrix for the Test set:", "\n")
```

```
## Confusion Matrix for the Test set:
```

```
CM_test <- table(y_test_pred$class, y_test)
CM_test
```

```
##    y_test
##       0   1   2   3
##   0 100   3   0   0
##   1   1 101   4   0
##   2   0   1 103   3
##   3   0   0   4  80
```

To conclude let's now proceed with the actual Forward Selection. For simplicity we will just report the final model.

```
# final model after FORWARD SELECTION

# Fit the Linear Discriminant Analysis model
QDA <- qda(y_train ~ ram+battery_power+clock_speed+mobile_wt+px_height+px_width+fc,
           data = X_train_scaled)

# Make predictions on Training set
y_train_pred <- predict(QDA, X_train_scaled)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred$class == y_train),"\n")
```

```
## Training set accuracy:  0.966875
```

```
# plot confusion matrix for Training set
cat("Confusion Matrix for the Training set:", "\n")
```

```
## Confusion Matrix for the Training set:
```

```
CM_train <- table(y_train_pred$class, y_train)
CM_train
```

```
##    y_train
##       0   1   2   3
##   0 393  11   0   0
##   1   6 376  10   0
##   2   0   8 366   5
##   3   0   0  13 412
```

```
cat("\n\n")
```

```
# Make predictions on Test set
y_test_pred <- predict(QDA, X_test_scaled)

# Model accuracy on Test set
cat("Test set accuracy: ", mean(y_test_pred$class == y_test),'\n')
```

```
## Test set accuracy:  0.9575
```

```
# plot confusion matrix for Test set
cat("Confusion Matrix for the Test set:", "\n")
```

```
## Confusion Matrix for the Test set:
```

```
CM_test <- table(y_test_pred$class, y_test)
CM_test
```

```
##    y_test
##       0   1   2   3
##   0 100   4   0   0
##   1   1  99   4   0
##   2   0   2 105   4
##   3   0   0   2  79
```

Even in this case, the accuracies achieved both on the training and the test set are considerably high, even though *Quadratic Discriminant Analysis* seemed to perform slightly worse than the other models tested so far. The maximum achieved accuracy on the test set is about 96.25%. In the following section we will go on to implement a final multiclass classification model, the *K-NN*.

4.4 K-Nearest Neighbors

The last classification model we are going to train is the *K-Nearest Neighbors*. In particular we will use a *10-Fold Cross Validation* approach, with a grid-search over the parameter K, number of Neighbors. We will test the model first on the original dataset and then after applying a scaling to the data.

```
library(ISLR)
```

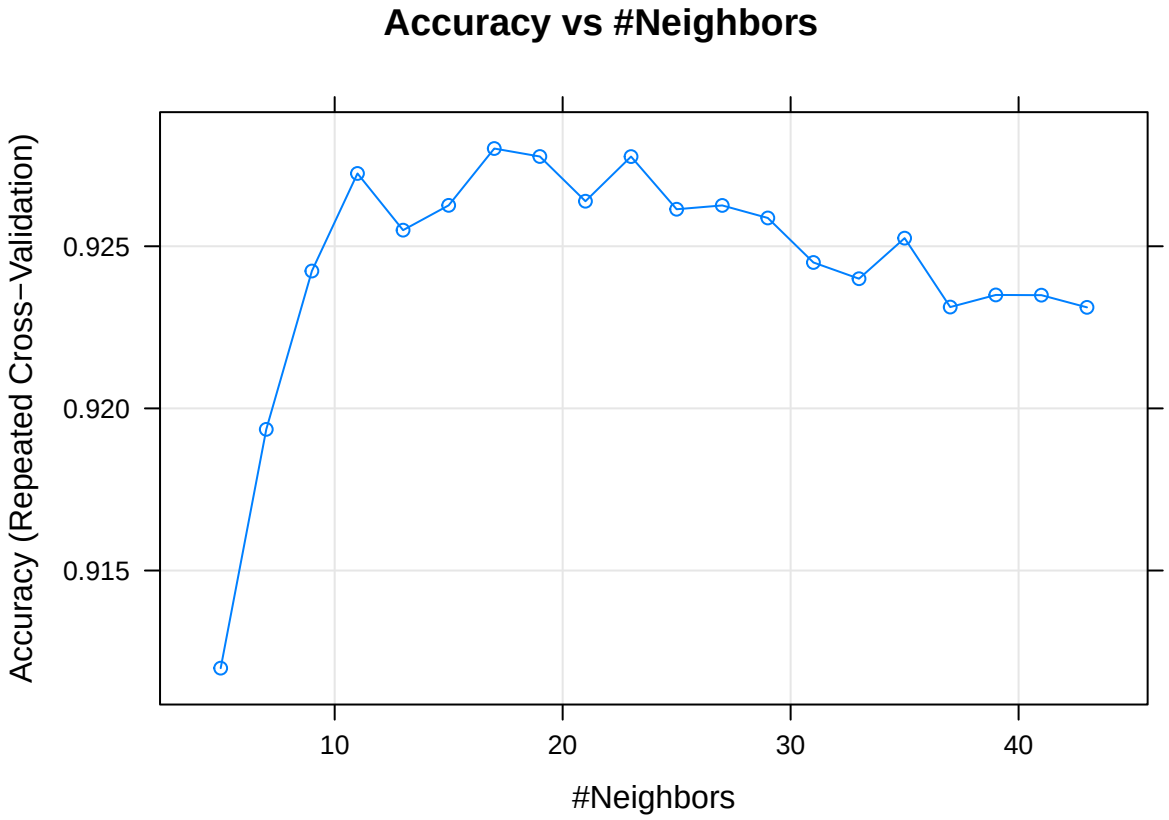
```
# train the K-Nearest Neighbors with the original Dataset

set.seed(1)
ctrl <- trainControl(method="repeatedcv", repeats = 5)
knn <- train(as.factor(price_range) ~ ., data = Training_set, method = "knn", trControl = ctrl,
             tuneLength = 20)

# Output of kNN fit
knn
```

```
## k-Nearest Neighbors
##
## 1600 samples
##   20 predictor
##    4 classes: '0', '1', '2', '3'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1441, 1440, 1440, 1440, 1440, 1439, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
```

```
##    5  0.9119899  0.8826263
##    7  0.9193533  0.8924486
##    9  0.9242356  0.8989594
##   11  0.9272451  0.9029715
##   13  0.9254966  0.9006394
##   15  0.9262614  0.9016568
##   17  0.9280138  0.9039952
##   19  0.9277678  0.9036644
##   21  0.9263888  0.9018217
##   23  0.9277654  0.9036556
##   25  0.9261411  0.9014856
##   27  0.9262590  0.9016451
##   29  0.9258723  0.9011308
##   31  0.9244996  0.8992971
##   33  0.9239988  0.8986307
##   35  0.9252497  0.9003003
##   37  0.9231246  0.8974696
##   39  0.9234980  0.8979639
##   41  0.9234918  0.8979525
##   43  0.9231160  0.8974544
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 17.
```

Below is the figure regarding the behavior of accuracy versus K values.

```
plot(knn, main="Accuracy vs #Neighbors")
```




```
y_train_pred <- predict(knn,newdata = Training_set)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred == as.factor(y_train)), '\n')
```

```
## Training set accuracy:  0.945625
```

```
# Get the confusion matrix to see accuracy value and other parameter values
table(y_train_pred, as.factor(y_train))
```

```
##
## y_train_pred   0   1   2   3
##            0 391  18   0   0
##            1   8 367  20   0
##            2   0  10 354  16
##            3   0   0  15 401
```

```
y_test_pred <- predict(knn,newdata = Test_set)

# Model accuracy on Test set
cat("\nTest set accuracy: ", mean(y_test_pred == as.factor(y_test)), '\n')
```

```
##
## Test set accuracy:  0.9475
```

```r
# Get the confusion matrix to see accuracy value and other parameter values
table(y_test_pred, as.factor(y_test))
```

```
##
## y_test_pred   0   1   2   3
##           0  98   2   0   0
##           1   3 102   4   0
##           2   0   1 101   5
##           3   0   0   6  78
```

We now implement again the same model, but this time after applying standard scaling to the data. That is, the data is centered (the mean is subtracted) and rescaled (i.e., divided by the standard deviation).

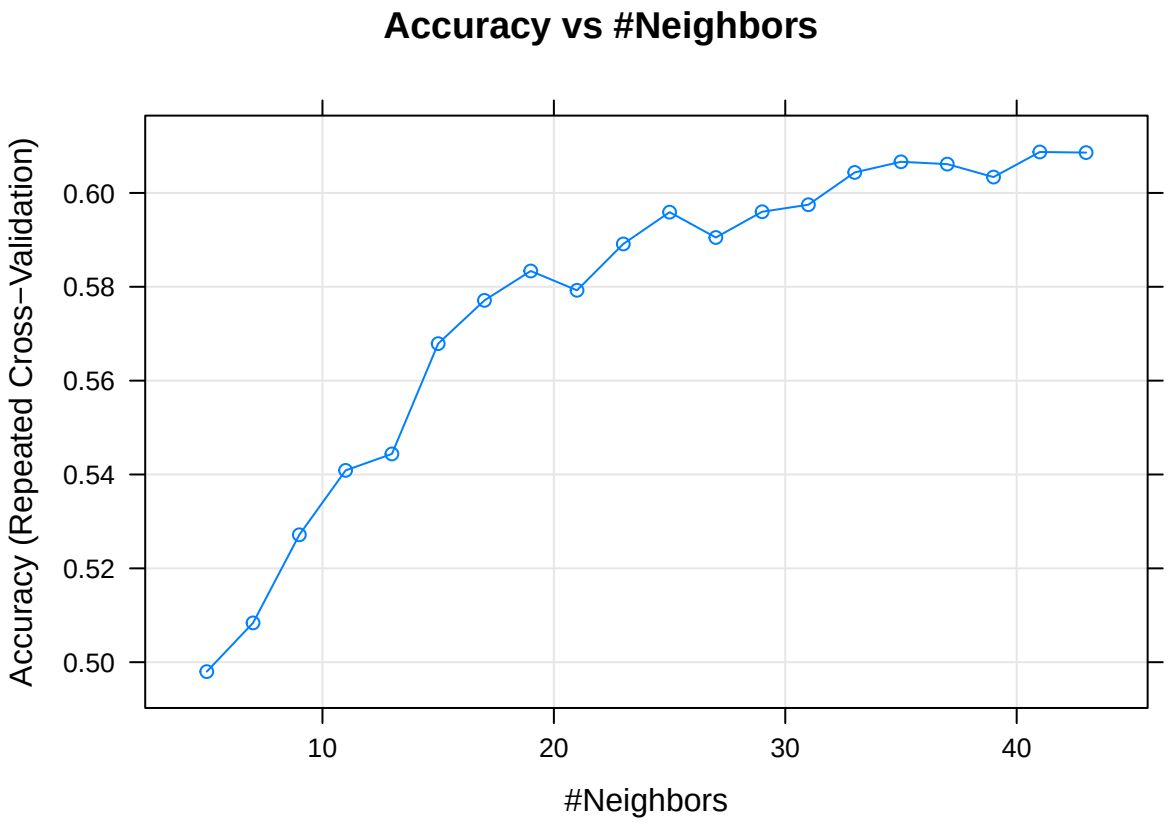
```r
# train the K-Nearest Neighbors with scaled data

set.seed(1)
ctrl <- trainControl(method="repeatedcv", repeats = 5)
knn <- train(as.factor(price_range) ~ ., data = Training_set, method = "knn", trControl = ctrl,
             tuneLength = 20, preProcess = c("center","scale"))

# Output of kNN fit
knn
```

```
## k-Nearest Neighbors
##
## 1600 samples
##   20 predictor
##    4 classes: '0', '1', '2', '3'
##
## Pre-processing: centered (20), scaled (20)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1441, 1440, 1440, 1440, 1440, 1439, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.4979975  0.3309120
##    7  0.5083804  0.3446191
##    9  0.5271560  0.3697499
##   11  0.5408753  0.3878833
##   13  0.5443848  0.3925215
##   15  0.5678862  0.4238228
##   17  0.5771035  0.4361361
##   19  0.5833693  0.4444934
##   21  0.5792544  0.4390962
##   23  0.5891264  0.4522269
##   25  0.5959026  0.4612638
##   27  0.5905023  0.4540584
##   29  0.5959953  0.4613935
##   31  0.5974860  0.4633495
##   33  0.6043689  0.4725345
##   35  0.6066355  0.4755752
##   37  0.6061456  0.4749369
##   39  0.6033789  0.4711960
```

```
##   41  0.6087290  0.4782860
##   43  0.6085962  0.4781156
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 41.
```

Below we visualize the behavior of accuracy with respect to K values.

```
plot(knn, main="Accuracy vs #Neighbors")
```




```
y_train_pred <- predict(knn,newdata = Training_set)

# Model accuracy on Training set
cat("Training set accuracy: ", mean(y_train_pred == as.factor(y_train)), '\n')
```

```
## Training set accuracy:  0.684375
```

```
# Get the confusion matrix to see accuracy value and other parameter values
table(y_train_pred, as.factor(y_train))
```

```
##
## y_train_pred   0   1   2   3
##            0 329  93   7   0
##            1  65 194  67   6
##            2   5  95 242  81
##            3   0  13  73 330
```

```r
y_test_pred <- predict(knn,newdata = Test_set)

# Model accuracy on Test set
cat("\nTest set accuracy: ", mean(y_test_pred == as.factor(y_test)), '\n')
```

```
##
## Test set accuracy:  0.6675
```

```r
# Get the confusion matrix to see accuracy value and other parameter values
table(y_test_pred, as.factor(y_test))
```

```
##
## y_test_pred  0  1  2  3
##           0 84 28  1  0
##           1 14 54 27  0
##           2  3 19 62 16
##           3  0  4 21 67
```

Surprisingly, the *K-NN* performs worse on the scaled data than it does on the original dataset. As we can see from the confusion matrices, the model struggles on multiple classes and in fact this is the worst among all the implemented models. Among the values we tested, the best K turns out to be 41. We notice though that the plot shows an increasing trend, so we may wonder if increasing the values of K may actually lead to better accuracy scores. Interestingly, from additional tests we will not report here, it turned out that such an increase becomes more and more insignificant as K grows larger, leading the accuracies to stabilize around the above obtained thresholds. In any case, both these *K-NN* models fail to achieve the same performance obtained with *Multinomial Logistic Regression*, *Linear Discriminant Analysis* and *Quadratic Discriminant Analysis*.

5. Conclusions

In this report we dealt with a mobile price classification problem, which consists of assigning to each observation one of the following four labels:

- 0 (low-cost);
- 1 (medium-cost);
- 2 (high-cost);
- 3 (very high-cost).

We began with the preprocessing phase in which after analyzing the structure of the dataset, we checked for the presence of missing values, class balancing, and we applied feature scaling. After that, we proceeded with an Exploratory Data Analysis in which we checked for possible relationships (linear, multicollinear and nonlinear) among the various predictors. From this first analysis we found that the variable *ram* appears to be the most correlated with the *price range*. However, in the training phase it later turned out not to be the only relevant feature. Specifically, in the training part we implemented four different supervised learning algorithms:

- Multinomial Logistic Regression;
- Linear Discriminant Analysis;
- Quadratic Discriminant Analysis;
- K-Nearest Neighbors.

For the Multinomial Logistic Regression, we trained the model first by considering the entire feature set; then we applied Backward Selection using *p-values* and Forward Selection by exploiting the *cross-validation error*. We then went on to compare the full model with the reduced model, in which we considered only the *ram* variable. This comparison, as mentioned above, showed that *ram* alone is not sufficient to correctly classify the observations. Finally, we also tested the regularized models by considering the L_1 and L_2 penalty terms (*Lasso* and *Ridge*).

We then implemented the Linear Discriminant Analysis, though only after checking that the data within each class roughly followed a multivariate normal distribution. We first considered the entire feature set, then we tested the model using the same predictors derived with Backward and Forward Selection as in the Multinomial case. Lastly, we also applied an actual Forward Selection.

We did the same for the Quadratic Discriminant Analysis as well. As before, we firstly tested the model keeping all the features, then we tested it considering just the feature sets obtained through Backward/Forward Selection on the previous models and then we implemented an actual Forward Selection.

At the end, we implemented the *K-Nearest Neighbors* using both the original dataset and a scaled version of it. In both cases we made use of a grid-search to find the best-fitting neighbors value and a 10-Fold Cross Validation.

Below we report the results obtained with all implemented models. In the following, (BWS) and (FWS) stand for Backward and Forward Selection, respectively.

Model	**Train set accuracy** (%)	**Test set accuracy** (%)
Multinomial Logistic Regression (Full Dataset)	99.50	97.50
Multinomial Logistic Regression (BWS)	98.50	97.75
Multinomial Logistic Regression (FWS)	98.31	96.75
Multinomial Logistic Regression (ram only)	74.94	78.75
Multinomial Logistic Regression (Lasso)	98.94	97.50
Multinomial Logistic Regression (Ridge)	87.56	85.25
Linear Discriminant Analysis (Full Dataset)	95.25	96.25
Linear Discriminant Analysis (FWS Multinom.)	95.38	96.25
Linear Discriminant Analysis (BWS Multinom.)	95.00	96.25
Linear Discriminant Analysis (FWS)	95.88	97.50
Quadratic Discriminant Analysis (Full Dataset)	96.50	94.00
Quadratic Discriminant Analysis (FWS Multinom.)	97.00	96.25
Quadratic Discriminant Analysis (BWS Multinom.)	96.50	96.25
Quadratic Discriminant Analysis (FWS LDA)	96.63	96.00
Quadratic Discriminant Analysis (FWS)	96.69	95.75
K-NN (not scaled)	94.56	94.75
K-NN (scaled)	68.44	66.75

In general, all implemented models but K-NN (scaled) and Multinomial Logistic Regression (ram only), perform very well; the highest accuracy value achieved on the test set is 97.75%, obtained by the Multinomial Logistic Regression (BWS). Anyway, there are also other models more or less satisfactory, like: *Multinomial Logistic Regression* (Full Dataset), *Multinomial Logistic Regression* (LASSO) and *Linear Discrimininat Analysis* (FWS), which are able to reach an accuracy value of about 97.50% on test set.

Bibliography

[1] https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification.

[2] *The Elements of Statistical Learning, Data Mining, Inference and Prediction*, Hastie T., Tibshirani R. and Friedman J. (2009).

[3] *An Introduction to Statistical Learning*, James G., Witten D., Hastie T. and Tibshirani R. (2021).