Mobile_Capston_IDV_report

David Brandtner

2022-09-13

Capstone IDV: Mobile Price Classification model

1. Introduction

The aim of this analysis is to implement a multiclass classification model that could separate mobile phones models in four price classes in order to guide a successful marketing price strategy.

The *Mobile Price Classification* dataset in total counts 2000 rows for 21 columns. Rows correspond to observations, that is classified mobiles, and columns correspond to 20 predictors, that is the mobile tech specs (see *Table 1*), plus the outcome class to predict: this is defined as *price_range* and it is composed by "*0*", "*1*", "*2*", "*3*" classes, here listed from the cheapest to the more expensive.

Table 1: Predictors reference table

Predictor name	Explanation and unit of measure
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 talk time with a full battery charge in hours
three_g	Has 3G or not
touch_screen	Has touch screen or not
wifi	Has wifi or not

2. Analysys

2.1 Datasets and cross validation technique

The *Mobile Price Classification* dataset comes in one file called *train*. This one is splitted into the ultimate test set - the independent set - called *test* (counting 1/3 of the whole dataset) and the remaining part still called *train* (counting 2/3 of the whole dataset).

After the preprocessing step a "*_pp*" suffix has been added to the original name of the *train* dataset. Coherently the same naming and predictors modifications has been applied to *test* dataset. Thus their updated names: *train_pp* and *test_pp*.

test_pp dataset has only a final overall Accuracy estimation purpose, while model training and trained model performance estimation has been operated on *train_pp*.

For trained model error estimation, Accuracy is calculated applying Repeated K-Fold Cross Validation technique. It means that dataset is divided into k subsets, where each time one of them is used as a test set and the other *k-1* subsets are put together to form a training set. This procedure is then repeated more times and finally their results are averaged. No further direct splitting of *train_pp* is here explicitly remarked and operated, because of the use of a dedicated model training function provided with automatic cross validation tuning.

2.2 Data Exploration and predictors hypothesis

As starting point is a key step to asses the outcome classes frequency in dataset, *price_range*, and check for the presence of *NA* values.

```
# check for outcome classes balance:
table(train$price_range)
```

```
##
##   0   1   2   3
## 400 400 400 400
```

The four classes occurrence is perfectly even, thus from this point of view model training is totally free from getting bias given by unbalanced frequencies.

```
# NA check:
sum(is.na(train))
```

```
## [1] 0
```

Since the sum of TRUE values in output is 0 there are not *NA* values.

Integrating Predictors reference table (*Table 1*) with a compact quick display of *train* dataset it has to be noticed that predictors are present in categorical and continuous types. The 6 categorical binary predictors are in the form of integer variable with value 0/1 representing state *not present/present.* Also the categorical outcome *price_range* is in the form of integer variable and it is useful not transforming it yet in factors for a profitable application of predictor weighting via correlation.

```
# structure compact display for train dataset:
str(train)
```

```
## 'data.frame':    1600 obs. of  21 variables:
##  $ 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 ...
##  $ 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     : int  0 1 1 0 0 1 0 1 0 1 ...
##  $ fc           : int  1 0 2 0 13 3 4 0 0 2 ...
##  $ four_g       : int  0 1 1 0 1 0 1 0 0 1 ...
##  $ 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      : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ touch_screen : int  0 1 1 0 1 0 0 1 0 0 ...
##  $ wifi         : int  1 0 0 0 0 0 1 1 0 0 ...
##  $ price_range  : int  1 2 2 2 1 1 3 0 0 0 ...
```

Categorical predictors belong in number of 5 to communication standards and 1 is about phone I/O hardware. In order to make consideration on how their frequencies vary through *price_range* classes a collective histogram plot is produced (*Fig. 1*).

```
# plotting categorical predictors frequencies:
cat <- c("blue", "dual_sim", "four_g", "three_g", "touch_screen",
         "wifi")

plot_categorical <- train %>% select(all_of(cat), "price_range") %>%
  dplyr::mutate(across(everything(), ~ as.factor(.x))) %>%
  reshape2::melt(measure.vars = cat) %>%
  ggplot(aes(x=value, fill=value)) + geom_bar(alpha = 0.9) +
  labs(x = "Category", y = "Count", title = "Categorical predictors frequencies",
       subtitle = "1 = present / 0 = not present", caption = "Fig.1") +
  scale_fill_manual(values=c("#0C6291", "#A63446")) +
  theme(plot.title = element_text(size = 15, face = "bold"),
        plot.subtitle = element_text(size = 11.5, face = "italic"),
        plot.caption = element_text(size = 15, hjust = 0),
        legend.position = "none") +
  facet_grid(rows = vars(price_range),
             cols = vars(variable),
             scales = "free_x")

plot_categorical
```

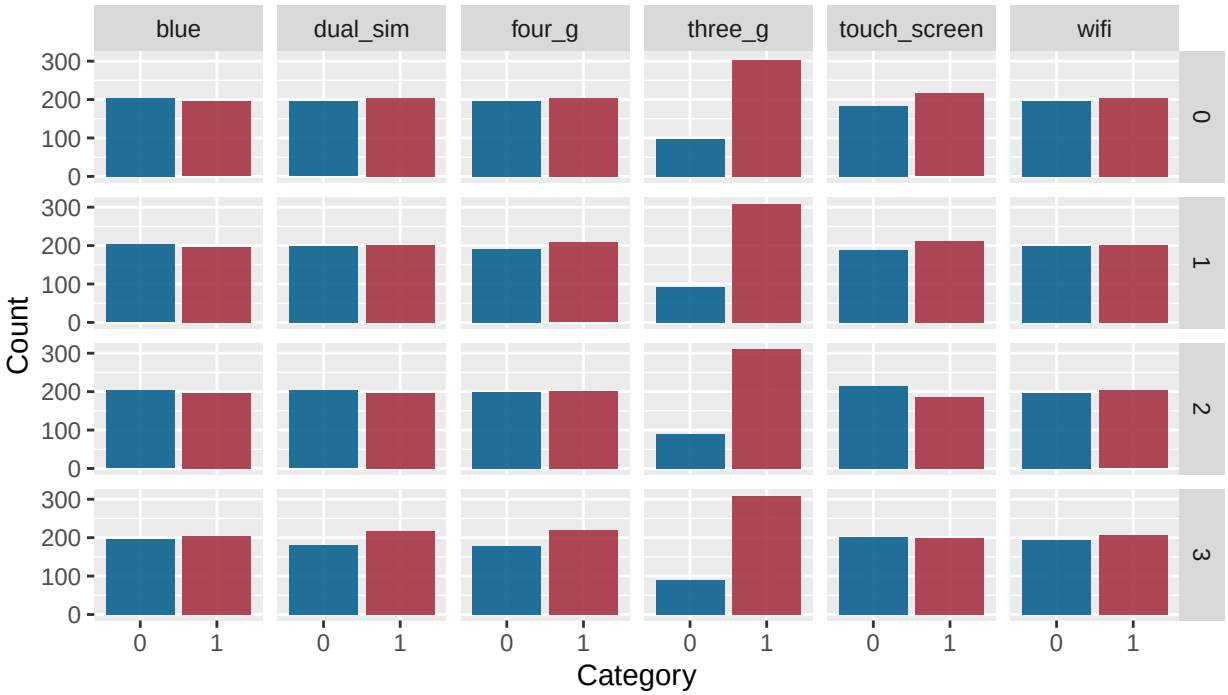

Fig.1

In general there is very little or none difference between *present/not present* frequencies so that there isn't a real discriminant predictor that realizes a separation between outcome categories, nor there isn't a combination of them that achieve this task. In particular Wi-Fi, Blue tooth and 3G are the most even represented. 3G is the basic mobile communication standard adopted since its large and even predominance in the *train* set of phones. For I/O hardware touch screen is little preferred to physical key in cheapest ("*0*" and"*1*") *price_range* mobiles. Only combining Dual SIM option along with 4G mobile communication standard gives a weak *"3"* class characterization over the others, but it is not something to build over, rather it is a mean to try an improvement in an already established model. Due this lack of information an elimination of these predictors is performed.

```
# removing categorical predictors
train_pp <- train[, !names(train) %in% cat]
```

The 14 continuous predictors are mobile tech specs described by very diverse unit of measure types: from spatial dimensions, to information unit of memory, to display characteristics. It is of interest therefore to check their numeric range for further reasoning.

```
# quick display of continuous predictors:
summary(train_pp)
```

```
##  battery_power     clock_speed          fc          int_memory
##  Min.   : 501.0   Min.   :0.500   Min.   : 0.000   Min.   : 2.00
##  1st Qu.: 854.8   1st Qu.:0.700   1st Qu.: 1.000   1st Qu.:16.00
##  Median :1222.5   Median :1.500   Median : 3.000   Median :31.00
##  Mean   :1237.1   Mean   :1.517   Mean   : 4.344   Mean   :31.82
```

```
##  3rd Qu.:1614.2   3rd Qu.:2.200   3rd Qu.: 7.000   3rd Qu.:47.00
##  Max.   :1996.0   Max.   :3.000   Max.   :19.000   Max.   :64.00
##      m_dep          mobile_wt        n_cores            pc
##  Min.   :0.1000   Min.   : 80.0   Min.   :1.000   Min.   : 0.000
##  1st Qu.:0.2000   1st Qu.:108.0   1st Qu.:3.000   1st Qu.: 5.000
##  Median :0.5000   Median :141.0   Median :4.000   Median :10.000
##  Mean   :0.5008   Mean   :139.7   Mean   :4.513   Mean   : 9.918
##  3rd Qu.:0.8000   3rd Qu.:169.2   3rd Qu.:7.000   3rd Qu.:15.000
##  Max.   :1.0000   Max.   :200.0   Max.   :8.000   Max.   :20.000
##    px_height         px_width         ram            sc_h           sc_w
##  Min.   :   0.0   Min.   : 500   Min.   : 256   Min.   : 5.0   Min.   : 0.00
##  1st Qu.: 273.8   1st Qu.: 874   1st Qu.:1205   1st Qu.: 9.0   1st Qu.: 2.00
##  Median : 544.0   Median :1244   Median :2174   Median :12.0   Median : 5.00
##  Mean   : 634.0   Mean   :1251   Mean   :2127   Mean   :12.3   Mean   : 5.82
##  3rd Qu.: 923.5   3rd Qu.:1636   3rd Qu.:3064   3rd Qu.:16.0   3rd Qu.: 9.00
##  Max.   :1960.0   Max.   :1997   Max.   :3998   Max.   :19.0   Max.   :18.00
##    talk_time      price_range
##  Min.   : 2.00   Min.   :0.00
##  1st Qu.: 6.00   1st Qu.:0.75
##  Median :11.00   Median :1.50
##  Mean   :10.94   Mean   :1.50
##  3rd Qu.:16.00   3rd Qu.:2.25
##  Max.   :20.00   Max.   :3.00
```

The whole numeric range spans through 5 orders of magnitude (10^{-1} -> 10^{4}) and this is an important issue to take in consideration during classification model adoption and tuning.

Range inspection reveals a form of *NA* value missed in the previous check: 0 as minimum in *fc*, *pc*, *px_height* and, *sc_w*. It is not possible to have a photo camera that records at 0 megapixel and a display dimension that measures 0. Then 0 is here considered equal to *NA* and it has to chosen a strategy to deal with it.

At first it is advisable to get size of missing data.

```
# 0 as NA values removal in fc, pc, px_height and sc_w predictors:
train_pp %>%
  filter(fc == 0 | pc == 0 | px_height == 0 | sc_w == 0) %>%
  select(fc, pc, px_height, sc_w) %>% nrow()
```

```
## [1] 473
```

```
# table with 0 values frequency in target predictor:
zero_tab <- train_pp %>% select (fc, pc, px_height, sc_w) %>%
  plyr::ldply(function(c) sum(c == 0))
colnames(zero_tab) <- c("predictor", "0_freq")
zero_tab %>%
  kableExtra::kbl(caption = "0 occurencies in selected predictors") %>%
  kable_paper("hover", full_width = F)
```

A total of 473 over 1600 rows (29.6%) are affected by null values in one or more of the four predictors. *fc* is the heavily affected one (23%), followed by *sc_w* (8.4%), *pc* (4.9%) and *px_height* with only 2 records (practically unaffected) - see *Table 2*.

A method based on rows deletion cuts away too much information (29.6%), thus a preserving one is here preferred: null value substitution with the mean differentiated by *prince_range* class.

Table 2: 0 occurencies in selected predictors

predictor	0_freq
fc	368
pc	79
px_height	2
sc_w	135

```
# target predictors:
fill0_t <- c("fc", "pc", "px_height", "sc_w")

# function to substitute 0 values in target predictors with the mean
# of its belonging price class:
fill0_fun <- function(t, df){
  for (j in 1:length(t)){
    for (i in 1:4){
      # first it calculates predictor mean value (excluding the 0s)
      fill <- df %>% filter(price_range == i-1 & .data[[t[[j]]]] != 0) %>%
        summarise(mean = round(mean(.data[[t[[j]]]]), 0))
      # and then substitute it to the 0 positions
      df <- df %>% mutate("{t[j]}" :=
                    ifelse(price_range == i-1 & .data[[t[[j]]]] == 0, fill$mean,
                           .data[[t[[j]]]]))
    }
  }
  return(df)
}

# dataset modification applying fill0_fun():
train_pp <- fill0_fun(fill0_t, train_pp)
rm(fill0_t)
```

With this method each of treated predictor for its extent is affected by an approximation and this should be taken in consideration if chosen as candidate for classification model input.

A focus on predictors regarding display tech specs lets to take note that they are released as raw measures and in separate way. To technical describe a display efficiently, from market standards and commercial point of views, they have to be expressed jointly in these two new predictors:

- mega pixels (*mega_px*) = the total number of pixels, expressed as million of pixels, given by pixels height times pixel width;
- PPI (*sc_ppi*) or Pixels Per Inch = the pixels density of a screen measured as the number of pixels that fit on a 1-inch line. It is given by calculating the number of pixel that fits on the screen diagonal divided the length of screen diagonal in inches.

```
# mega_px and sc_ppi setting up:
train_pp <- train_pp %>% mutate(mega_px = (px_width*px_height)/10^6,
                                sc_ppi = sqrt(px_width^2+px_height^2)/
                                  (sqrt(sc_w^2+sc_h^2)/2.54))
```

For a more detailed predictors analysis it is wise to concentrate only on those predictors for which *price_range* shows a connection.

To proceed through this filtering step Pearson's correlation (linear correlation) is calculated between all continuous predictors. Only significant ($p < 0.05$) correlation coefficients r is taken in consideration and results are plotted with a heatmap in *Fig. 2*.

```
# Pearson's correlation of continuous predictors -
# calculation and plotting:

# function for correlation calculation and edit output:
cors <- function(df) {
  # turn all three matrices (r, n, and P into a data frame)
  M <- Hmisc::rcorr(as.matrix(df))
  # return the three data frames in a list return(Mdf)
  Mdf <- map(M, ~data.frame(.x))
}

# function for edit results of cors() as heatmap input:
formatted_cors <- function(df){
  cors(df) %>%
    map(~rownames_to_column(.x, var="measure1")) %>%
    map(~pivot_longer(.x, -measure1, "measure2")) %>%
    bind_rows(.id = "id") %>%
    pivot_wider(names_from = id, values_from = value) %>%
    mutate(sig_p = ifelse(P < .05, T, F), p_if_sig = ifelse(P <.05, P, NA),
           r_if_sig = ifelse(P <.05, r, NA))
}

# plotting correlation heatmap using formatted_cors() as input:
formatted_cors(train_pp) %>%
  ggplot(aes(measure1, measure2, fill = r, label = round(r_if_sig,2))) +
  geom_tile(col = "black") +
  labs(x = NULL, y = NULL, fill = "Pearson's\nCorrelation",
       title="Correlations in train_pp only for continuous predictors",
       subtitle="Only significant (p<0.05) Pearson's correlation coefficients shown",
       caption = "Fig.2") +
  scale_fill_gradient2(mid="#FBFEF9",low="#0C6291",
                       high="#A63446", limits=c(-1,1)) +
  geom_text(size=2.5) +
  theme_classic() +
  theme(plot.title = element_text(size = 15, face = "bold"),
        axis.text.x = element_text(angle = 45, hjust = 1 ,vjust = 1),
        legend.title = element_text(size = 9),
        legend.text = element_text(size = 7),
        plot.caption = element_text(size = 15, hjust = 0))
```

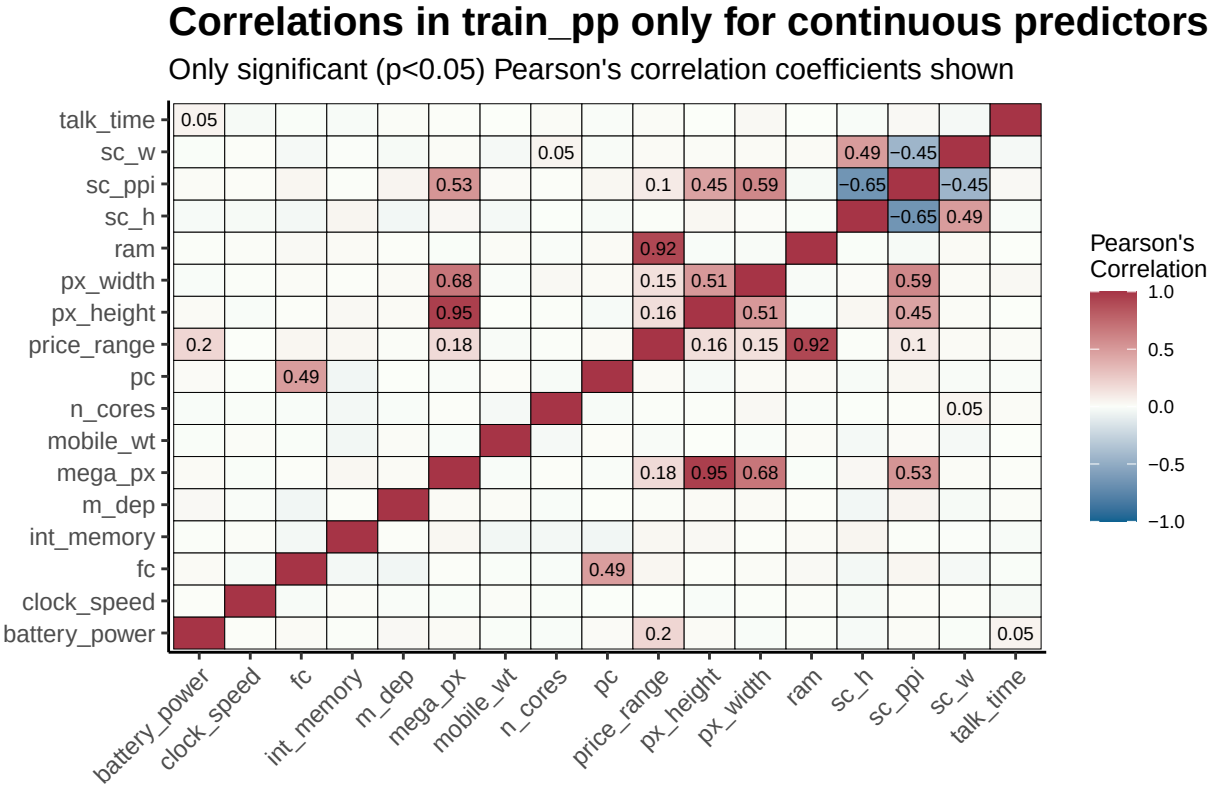

Fig.2

The bare results reading is that few tech specs drive the price stratification from the sight of linear correlation: *price_range* correlates very highly with *ram* ($r = 0.92$) and little with *battery_power* ($r = 0.2$), *mega_px* ($r = 0.18$), *px_height* ($r = 0.16$), *px_width* ($r = 0.15$) and *sc_ppi* ($r = 0.1$).

Going into details the newly created predictors register an r improvement in respect to correlation values of the tech specs combined to obtain them: this is true for *mega_px* but surprisingly true for *sc_ppi* that carries inside also *sc_w* and *sc_h* that does not show significant connection, when alone, with *price_range*.

Checking the behavior of linear correlation between this selection of predictors it has to be noticed that *mega_px* have a very high overlap with *px_height* ($r = 0.95$) along with a better correlation with *px_width* than the same *px_height* ($r = 0.68$ instead of $r = 0.51$). This redundancy could be eliminated selecting *mega_px* as the useful predictor between the two since its better correlation with the outcome variable *price_range*.

Proceeding in filtered predictors analysis, a better investigation regarding the characteristics of their distribution is performed: for this scope a collective panel of box plots overlapped to violin plots is plotted adding a separation by price class in *Fig. 3* (from *A* to *E and G)*.

ram proves its high r with *price_range* achieving a very good separation through price classes, but some overlaps between them still remain. With much more distribution overlapping *battery_power*, *mega_px*, *px_height* and *px_width* separates well only class "*3*" from the others. Mostly judging on medians confrontations in an high overlapping distributions outline *sc_ppi* realizes only mild separation between class "*0*" and the other three.

Many outliers are present in *mega_px* and *sc_ppi* distributions that are highly right skewed at the same time, as stressed by their plots. So that in case of the choice of a classification model sensitive to outliers here it is proposed the adoption of normalizing transformation: square root transformation for *mega_px* (new predictor *mega_px_t*) and log transformation for *sc_ppi* (new predictor *sc_ppi_t*), chosen along

compression magnitude needed for their high values respectively. Transformed distributions results are plotted in *Fig. 3-F* and *Fig. 3-H* and it could be registered an almost complete outliers reduction.

```
# predictors selection after correlation results and
# distribution transformation for mega_px and sc_ppi
train_pp <- train_pp %>%
  select(price_range, ram, battery_power, mega_px, px_height, px_width,
         sc_ppi) %>% mutate(price_range = as.factor(price_range),
                            mega_px_t = sqrt(mega_px), sc_ppi_t = log(sc_ppi))
```

```
# Predictors distribution analysis plotting a box plot
# overlapped with a violin plot:

# subtitles plot:
f_subt <- c("ram", "battery_power", "mega_px", "px_height", "px_width", "sc_ppi",
                "mega_px - normalized", "sc_ppi - normalized")

# y labels plot:
f_ylab <- c("bytes", "mAh", "MPixel", "pixels","pixels", "PPI", "sqrt of Mpixel",
            "log10 of PPI")

# plot template function:
plots_function <- function(data, column, subt, ylab){
      ggplot(data, aes(price_range, column,
                  group = price_range, fill = price_range)) +
      geom_boxplot(alpha = 0.6) + geom_violin(alpha = 0.2) +
      labs(subtitle = subt, fill = "Price Class") +
      scale_fill_manual(values=c("#0C6291","#969696","#884791", "#A63446")) +
      xlab("Price Class") + ylab(ylab) +
      theme_classic() +
      theme(plot.subtitle = element_text(size = 17, face = "italic"),
            axis.title = element_text(size = 11.5),
            legend.title = element_text(size = 11.5))
   }

# loop to produce actual plots, saving them in a list:
plot_list <- list()
for (n in 2:9){
plot <- eval(substitute(
  plots_function(train_pp, train_pp[,n], f_subt[n-1], f_ylab[n-1]), list(n = n)))
plot_list[[n-1]] <- plot
}

# final panel plot composition with patchwork library:
plot_list[[1]] + plot_list[[2]] + plot_list[[3]] + plot_list[[4]] + plot_list[[6]] +
  plot_list[[7]] + plot_list[[5]] + plot_list[[8]] +
  guide_area() + plot_layout(ncol = 3, guides = "collect") +
  plot_annotation(title = "Predictors distribution and outliers", caption = "Fig.3",
                  tag_levels = "A") &
  theme(plot.title = element_text(size = 20, face = "bold"),
        plot.caption = element_text(size = 23, hjust = 0.1))
```

Predictors distribution and outliers

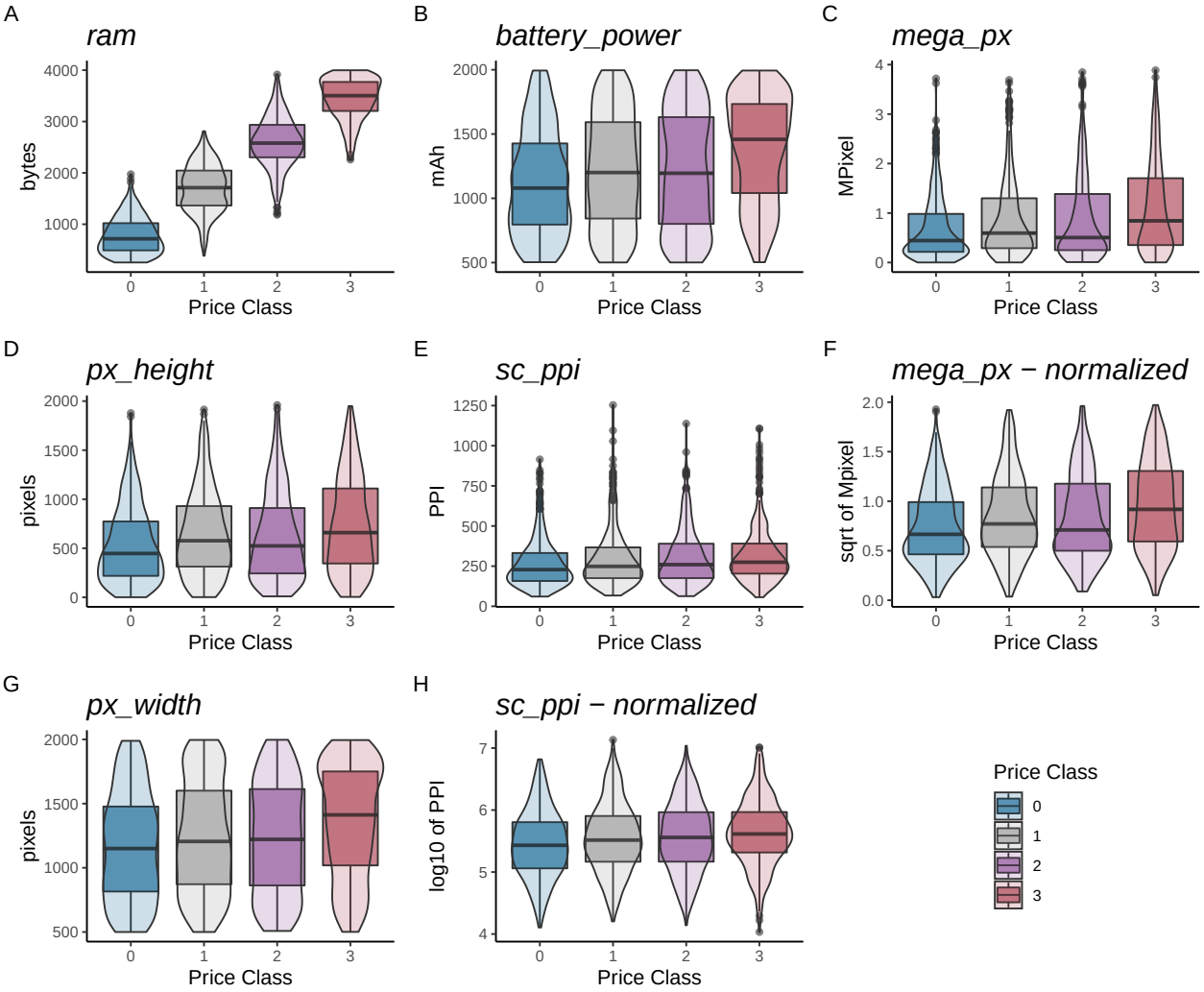

Fig.3

2.3 Modeling approach

Data Exploration has resulted in a maximum of 6 candidate predictors so that not forming a high dimensional space (e.g. < 10 predictors). Moreover all predictors are continuous variables.

Two algorithm have been chosen to implement the multiclass classification model:

- KNN
- linear SVM

They have similar characteristics for outlier sensitivity and dataset scaling need, instead differences lies in classifier types and capacity to deal with dimensionality. KNN is a non linear classifier and prefer low number of predictors. Linear SVM is a linear classifier and deal very well also with high dimensionality dataset.

2.3.1 KNN

A point in predictors space is classified by a majority vote of its k-neighbors, with the point being assigned to the *class* most common amongst its K Nearest Neighbors measured by the Distance Function. For this KNN algorithm factor k is chosen empirically via a tuning grid and the Euclidean Distance Function is the chosen distance metric.

$$Euclidean\ Distance\ Function = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

2.3.2 linear SVM

Supportin Vector Machine - SVM - algorithm here is tuned to operate classification with more than 2 classes working alternatively with *one-vs-others* scheme. The following working mechanism is intended therefore valid also for the multiclass classification purpose.

Linear SVM is a linear classification algorithm that separate 2 classes of data with 2 parallel hyperplanes, the Hard Margins, so that the distance between them $\frac{2}{\|w\|}$ is as large as possible. The w and b that solve this problem determine the classifier:

$$argmax(w^*, b^*)\ \frac{2}{\|w\|}\ such\ that\ y_i(w^{\rightarrow}.X^{\rightarrow} + b) \geq 1$$

Since the border points in the predictors space that determines the Hard Margins could represent a training dataset unwanted variance, which could hinder a better class prediction in the unknown dataset for an *overfitting* phenomenon, a Cost parameter C is added to obtain Soft Margin.

With parameter C points inside $\frac{2}{\|w\|}$ is permitted but penalized to deal with variance. Rewriting the equation for Soft Margin:

$$argmin(w^*, b^*)\ \frac{\|w\|}{2}\ such\ that\ y_i(w^{\rightarrow}.X^{\rightarrow} + b) \geq 1$$

$$argmin(w^*, b^*)\ \frac{\|w\|}{2}\ +\ c\ \sum_{i=1}^{n}\zeta_1$$

Parameter C is chosen empirically via a tuning grid.

2.3.3 Predictors scaling

As predictors scales vary considerably a min-max normalization is applied to rescale their range in [0, 1].

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

2.3.4 Performance metric

Multiclass classification model performance is evaluated by the fraction of correct classifications, the Overall Accuracy metric:

$$Overall\ Accuracy = \frac{correct\ classifications}{all\ classifications}$$

3. Results

3.1 Models common set up

Since the adoption of 2 algorithms sensitive to outliers for classification scope, the choice of input predictors not hindered by outliers is the advisable choice.

The predictors selection is therefore: *ram*, *battery_power*, *px_height*, *px_width*, *mega_px_t* and *sc_ppi_t*.

```
# predictors selection with mega_px_t and sc_ppi_t:
ps <- subset(colnames(train_pp[-1]),
             str_ends(colnames(train_pp[-1]), "_p+.",negate = T))
ps
```

```
## [1] "ram"           "battery_power" "px_height"     "px_width"
## [5] "mega_px_t"     "sc_ppi_t"
```

A power set of this selection is produced to be used as input predictors subset in next models tuning.

```
# create power set of predictor selection (excluding first empty element)
power_list <- powerSet(ps)[-1]
length(power_list)
```

```
## [1] 63
```

As described in in *§2.3.3* min/max normalization in range of [0,1] is operated on candidate input predictors.

```
# predictors min/max normalization:
preprocessParams_train <- preProcess(train_pp[ps], method=c("range"))
train_pp[ps] <- predict(preprocessParams_train, train_pp[ps])
```

As described in *§2.1* the validation method chosen is Repeated K-Fold cross-validation and here the details of its parameters: 10 samples using 10% of the observations each, repeated for 3 times.

```
# trainControl() parameters for cross-validation setting up:
set.seed(1)
control <- trainControl(method = "repeatedcv", number = 10, p = 0.9,
                        repeats = 3)
```

3.1 KNN best model

The strategy to compose the model in terms of which predictors to use, removing the irrelevant ones (redundancy and low performing), taking the lower number of predictors to use (low dimensionality) and which k parameter to choose is totally empirical and data driven:

- every 63 predictors subset generated in the predictors selection power set *(§3)* is tested;

- for each subset k parameter is tested multiple time via a tuning grid that spans through an odd values series between 3 and 31.

```
# k tuning grid setting up
kgrid = seq(3, 31, 2)
```

Now the actual 63 models training, training results and best tune selection.

```
# summary results dataframe setting up:
rs <- data.frame(subset = character(), model_id = numeric(),
                 pred_nr = numeric(), k = integer(), accuracy = numeric())

# loop for try out all possible model from power set subsets:
top_acc = 0
best_train_knn <- list()

for (i in 1:length(power_list)){
  set.seed(1)
  train_knn <- train(x = train_pp[power_list[[i]]], y = train_pp$price_range,
                     method = "knn",
                     tuneGrid = data.frame(k = kgrid),
                     trControl = control)

    # saving best train_knn object
  if(max(train_knn$results$Accuracy) > top_acc){
        top_acc = max(train_knn$results$Accuracy)
        best_train_knn = train_knn}

    # updating results dataframe
  rs <- rs %>% add_row(
    subset = paste(power_list[[i]], collapse = " + "),
    model_id = i,
    pred_nr = length(power_list[[i]]),
    k = train_knn$finalModel$k,
    accuracy = max(train_knn$results$Accuracy)
    )
}
```

The best performing model is *model 19* composed by 3 predictors - *ram* , *battery_power* and *mega_px_t* - with parameter $k = 13$. *Table 3* summarizes also the other valuable results representing, for each of the six groups defined by the number of predictors used as model input, their specific best model for accuracy. This grows till the use of 3 predictors, then starts to decrease. If *ram* is the solid block to build over accounting alone for the 75.3% of accuracy (*model 1*), *battery_power* is the second fixed choice, in all model, since its good accuracy gain, 82.5% (*model 3*). Among remaining predictors *mega_px_t* carries the most meaningful for data KNN also outperforming the predictors combined to create it: its contribution alone raises accuracy to the max and every other addition carries redundancy or not performing data. Dimensionality for KNN is an issue and dealing with a predictor, which condenses information from two separate sources, fits better its needs. Among the basics keep dimensionality low.

Therefore *model 19* is used to make class prediction on the *test_pp* dataset to asses its validity.

```
# class prediction with model19:
y_hat_knn <- predict(best_train_knn, test_pp[ps])
```

Table 3: KNN|Summary of best performing models. In red model19 the best performing amongst all.

subset	model_id	pred_nr	k	accuracy
ram	1	1	27	0.7537500
ram + battery_power	3	2	31	0.8252083
ram + battery_power + mega_px_t	**19**	**3**	**13**	**0.9183333**
ram + battery_power + mega_px_t + sc_ppi_t	51	4	13	0.9085417
ram + battery_power + px_height + px_width + mega_px_t	31	5	19	0.8975000
ram + battery_power + px_height + px_width + mega_px_t + sc_ppi_t	63	6	27	0.8877083

Table 4: KNN|Model19 performance: final overall Accuracy in red.

model_id	pred_nr	k	accuracy	dataset
19	3	13	0.9183333	train_pp
19	**3**	**13**	**0.9343750**	**test_pp**

```
# confusion matrix and statistics:
cm_knn <- confusionMatrix(y_hat_knn, test_pp$price_range, mode = "everything")

# final overall Accuracy:
cm_knn$overall["Accuracy"]
```

```
## Accuracy
## 0.934375
```

Final **KNN overall Accuracy is 0.9343750**.

It is higher than training results on *train_pp* (*see Table 4*) proving all the measure adopted to contain variance effect have worked in the right direction.

Breaking down into detail, looking at other evaluation metrics and confusion matrix, it is possible to register performance differences through the four price classes (*see Table5 and Table6*):

- the cheaper one, class 0, gives the largest contribution to whole accuracy having a very high sensitivity an specificity;
- middle classes, 1 and 2, performs better in true negative recognition than true positive assignation and with their mild metrics they hinder performance in its complex;
- for class 3, the expensive one, while performs mildly in true positive recognition it performs extremely well in true negative recognition, making this class the second best contributor to whole accuracy.

Table 5: KNN|Multiclass confusion matrix: col/reference, row/prediction.

	0	1	2	3
0	386	14	0	0
1	14	377	27	0
2	0	9	359	27
3	0	0	14	373

Table 6: KNN|Selected performance metrics by class.

	Sensitivity	Specificity	F1
Class: 0	0.9650	0.9883333	**0.9650000**
Class: 1	0.9425	0.9658333	**0.9217604**
Class: 2	0.8975	0.9700000	**0.9031447**
Class: 3	0.9325	0.9883333	**0.9479034**

3.2 Linear SVM best model

The strategy to compose the model in terms of which predictors to use (removing the irrelevant ones or taking the max number of useful ones) and which C parameter to choose is totally empirical and data driven:

- every 63 predictors subset generated in the predictors selection power set *(§3)* is tested;
- for each subset C parameter is tested multiple time via a tuning grid that spans through a 0.075 incremental step series between 2 and 2.3.

```
# C tuning grid setting up
Cgrid = expand.grid(C = seq(2, 2.3, length = 5))
```

Now the actual 63 models training, training results and best tune selection.

```
# summary results dataframe setting up:
rs2 <- data.frame(subset = character(), model_id = numeric(),
                  pred_nr = numeric(), C = numeric(), accuracy = numeric())

# loop for try out all possible model from power set subsets:
top_acc = 0
best_train_svm <- list()

for (i in 1:length(power_list)){
  set.seed(1)

  train_svm <- train(price_range ~ ., data = cbind(train_pp["price_range"],
                     train_pp[power_list[[i]]]), tuneGrid = Cgrid,
                     method = "svmLinear", trControl = control)

  # saving best train_knn object
  if(max(train_svm$results$Accuracy) > top_acc){
    top_acc = max(train_svm$results$Accuracy)
    best_train_svm = train_svm}

  # updating results dataframe
  rs2 <- rs2 %>% add_row(
    subset = paste(power_list[[i]], collapse = " + "),
    model_id = i,
    pred_nr = length(power_list[[i]]),
    C = train_svm$bestTune$C,
    accuracy = max(train_svm$results$Accuracy)
  )
}
```

Table 7 summarizes the valuable results representing, for each of the six groups defined by the number of predictors used as model input, their specific best model for Accuracy. This has a sustained growth till the use of 3 predictors: if *ram* is the solid block to build over accounting alone for the 75.68% of accuracy (*model 1*), *battery_power* is the second fixed choice, in all model, since its good accuracy gain to 82.9% (*model 3*). Addition of *mega_px_t* carries the most meaningful step forward reaching accuracy of 94% (*model 19*). *model 15,* with 4 number of predictors where *px_height* and *px_width* take the place of *mega_px_t,* outperforms the previous model with 95.75%. It seems that SVM could manage well dimensionlity increase and the double source of *mega_px_t* has a better performance than this one alone. In *model 31*, at 5 number of predictors, *px_height*, *px_width and mega_px_t* works together but it is registered a little accuracy loss, maybe for variance effect not compensated by parameter C at his max limit. The best performing model is *model 63* composed by the whole 6 predictors selection - *ram*, *battery_power*, *px_height*, *px_width*, *mega_px_t* and *sc_ppi_t* - with parameter $C = 2.150$. Accuracy raises to 95,8% and the SVM vocation to deal with high dimensionality is clear. All the useful information is efficiently put to use.

Table 7: SVM|Summary of best performing models. In red model63 the best performing amongst all.

subset	model_id	pred_nr	C	accuracy
ram	1	1	2.000	0.7568750
ram + battery_power	3	2	2.150	0.8291667
ram + battery_power + mega_px_t	19	3	2.225	0.9414583
ram + battery_power + px_height + px_width	15	4	2.225	0.9575000
ram + battery_power + px_height + px_width + mega_px_t	31	5	2.225	0.9568750
ram + battery_power + px_height + px_width + mega_px_t + sc_ppi_t	**63**	**6**	**2.150**	**0.9581250**

Therefore *model 63* is used to make class prediction on the *test_pp* dataset to asses its validity.

```
# class prediction with model63:
y_hat_svm <- predict(best_train_svm, test_pp[ps])

# confusion matrix and statistics:
cm_svm <- confusionMatrix(y_hat_svm, test_pp$price_range, mode = "everything")

# final overall Accuracy:
cm_svm$overall["Accuracy"]
```

```
## Accuracy
##    0.965
```

Final **linear SVM overall Accuracy is 0.9650000**.

It is higher than training results on *train_pp* (*see Table 8*) proving all the measure adopted to contain variance effect have worked in the right direction.

Breaking down into detail, looking at other evaluation metrics and confusion matrix, it is possible to register performance differences through the four price classes (*see Table9 and Table10*):

- the cheaper one, class 0, gives the largest contribution to whole accuracy having a very high sensitivity and an almost perfect specificity;
- middle classes has a very high sensitivity and specificity, very similarly to class 3, the expensive one, that however performs a little better;
- middle class 2, in respect to the others, performs badly in true positive recognition with a marked lower sensitivity, while maintain a high specificity similar to class 1 and 3.

Table 8: SVM Model63| performance: final overall Accuracy in red.

model_id	pred_nr	C	accuracy	dataset
63	6	2.15	0.958125	train_pp
63	**6**	**2.15**	**0.965000**	**test_pp**

Table 9: SVM Model63|Multiclass confusion matrix: col/reference, row/prediction.

	0	1	2	3
0	390	4	0	0
1	10	389	11	0
2	0	7	376	11
3	0	0	13	389

Table 10: SVM Model63|Selected performance metrics by class.

	Sensitivity	Specificity	F1
Class: 0	0.9750	0.9966667	**0.9823678**
Class: 1	0.9725	0.9825000	**0.9604938**
Class: 2	0.9400	0.9850000	**0.9471033**
Class: 3	0.9725	0.9891667	**0.9700748**

4. Conclusions

The aim of the multiclass classification model implementation is the correct classification of a phone model into the already stratified market price classes. This will help to guide a successful marketing strategy.

The *Mobile Price Classification* dataset has restrained dimension (only 2000 rows), but a good array of possible predictors (20) that are phone tech specs.

Its splitting has been performed in order to obtain an ultimate *test* set (400 rows) and a *train* set (1600 rows).

After this an exploratory data analysis has been performed finding a mix of categorical predictors (6) and continuous predictors (14). Analysis has been focused on the latter since the former has been found not informative.

In continuous predictors the possibility to create new predictors rationalizing the existing ones has been followed and *mega_px* (display Megapixels) and *sc_ppi* (display Pixel Per Inch) has been implemented.

So a filtering step has been adopted to removing irrelevant predictors selecting only those having a significant ($p > 0.05$) linear correlation with the outcome class variable: six of them have been found positive including *mega_px* and *sc_ppi* (on which a replica - with suffix *_t* - with normalizing transformation has been performed to reduce their outliers excess).

KNN , a non linear multiclass classifier algorithm, and linear SVM, a linear multiclass classifier algorithm, have been adopted in competition to develop the best performing class prediction model.

After the essay of several prototypes **SVM model 63** has resulted in the best performing model reaching an **overall Accuracy of 0.965**.

Looking at performances separated by class, class 0 and 3 are the better right classified, while class 1 and 2 are leaved behind. Class 0 is the best absolute classified, while class 2 is the most problematic one.

This class performances is true in summary also for KNN results but with a worst disadvantage of class 1 and 2 into respect of the leading ones. Class 2 remains also in KNN model the most problematic one.

For this same trend by class an ensemble building with KNN and linear SVM models are here not suggested.

Newly created predictors has demonstrated to carry valid information, being part of the success of the best KNN model 19 (*mega_px_t*) and of the winning SVM model 63 (*mega_px_t*, *sc_ppi_t*).

Like future perspective should be taken in consideration:

1. an effort to collect data without so massive missing data in order to reach a complete and reliable investigation on all available predictors;
2. an expansion of dataset row dimension collecting more data if possible.