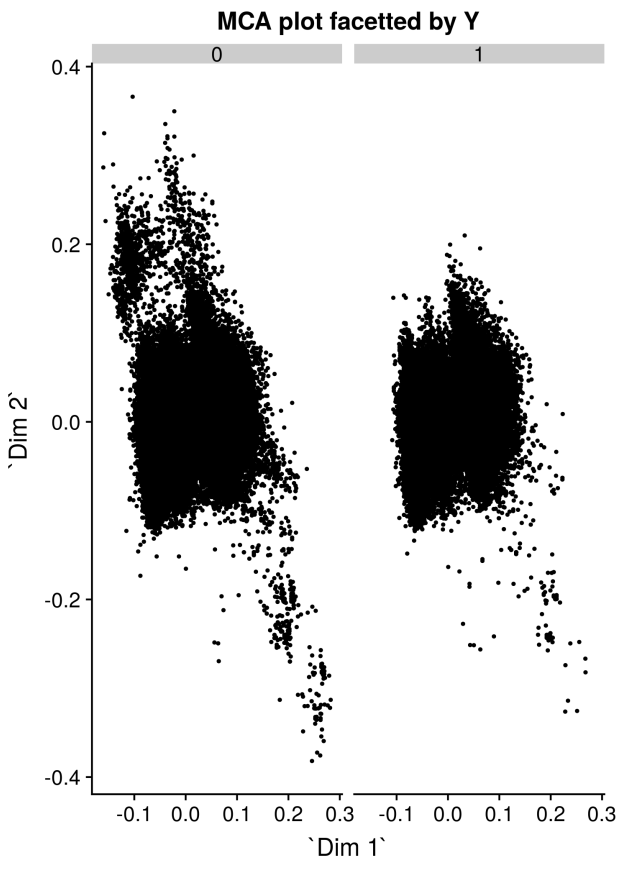
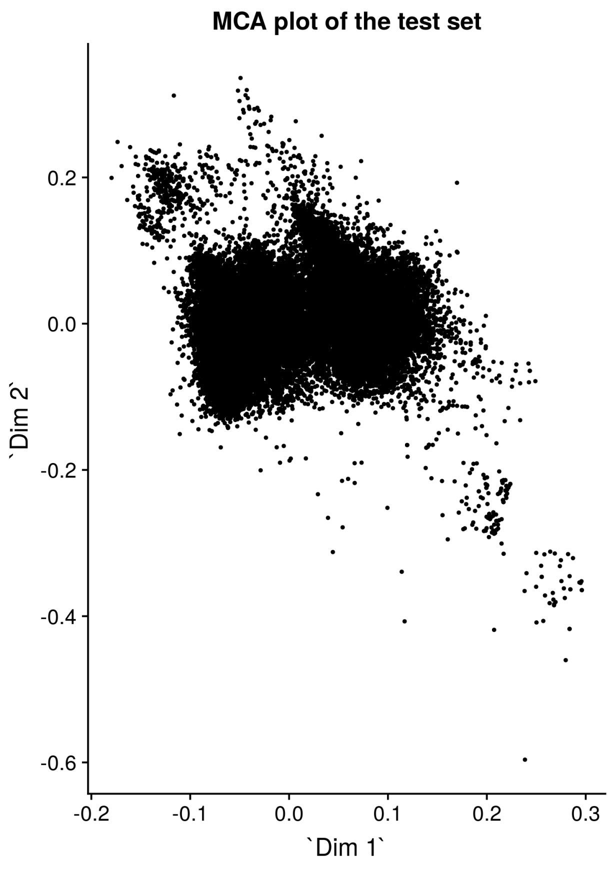
Technical test

Junior Data Scientist position at Datrik Intelligence

Antonio Ortega

October 28, 2018





Contents

1 Introduction

The dataset provided presents a classification problem, whereby the power of several features is to be harnessed to predict a binary label (1/0). A summary of the features available is presented in table 1.

binary	counter	edad	etiqueta	farmaco	identificador	nominal	ordinal	raza	sexo	Y
3	8	1	3	23	1	4	2	1	1	1

Table 1: 46 features are provided together with an identifier variable (*identificador*) and the label to be predicted Y.

Y, the target variable, is to be predicted using the information stored in the remaining features by means of (I) a linear model, and (II) a gradient boosting model. But first, the data needs to be preprocessed into a format that suits these algorithms. This is carried out in the R script `EDA.R`. A guide-through of the code is provided below.

2 EDA

1 Load libraries

```r
library(ggplot2)
library(viridis)
library(dplyr)
library(magrittr)
library(waffle)
library(tidyr)
library(tibble)
library(ade4)
library(data.table)
library(stringr)
library(FactoMineR)
library(cowplot)
library(pheatmap)
library(kableExtra)
# library(MASS) library(scales)
plot_dir <- "plots"
output_data <- "proc_data"
rdata_dir <- "RData"
tables_dir <- "tables"
```

2 Load data

Read the `datos.csv` file and split the training/validation set and the test set.

```r
datos <- read.table("datos.csv", sep = ",", header = T, stringsAsFactors = F)
datos <- datos[, c(colnames(datos)[1:4], colnames(datos)[5:(ncol(datos) - 1)] %>%
    sort, "Y")]
write(x = kable(x = datos %>% colnames %>% strsplit(., split = "_") %>% lapply(.,
    function(x) x[[1]]) %>% unlist %>% table %>% t, format = "latex", digits = 2),
    file = file.path(tables_dir, "data_summary.tex"))
```

```
datos$edad_integer <- str_match(string = datos$edad, pattern = "\\[\\d{2}-(\\d{2})\\)") %>%
    .[, 2] %>% as.integer

train_set <- datos[!is.na(datos["Y"]), ]
x_train <- train_set %>% select(-Y)
y_train <- select(train_set, Y) %>% mutate(Y = as.factor(Y))

test_set <- datos[is.na(datos["Y"]), ]
x_test <- test_set %>% select(-Y)
```

3 Visualization of race and age

A general picture of how the potentially relevant categories race and age dataset are distributed across the individuals is shown in figure 1.

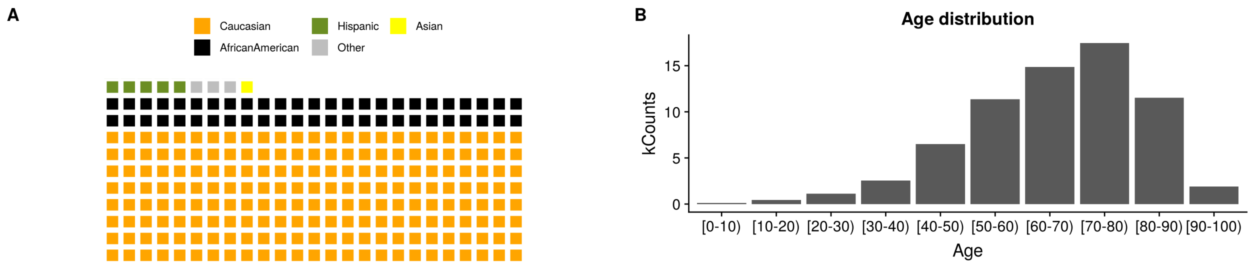


Figure 1: **A** Waffle plot showing the race distribution. A majority of the individuals are defined as Caucasian, with Afro Americans making up a significant though minor proportion. The remanining individuals are Hispanic, Asian and from other groups. **B** A histogram over the age groups reveals a trend for individuals to be aged 50 and 90 years old.

The non uniform distribution of individuals over these categories and eventually many others showcases the existence of some bias in the data.

4 Preprocess the training and test sets

Define a function to preprocess the train dataset and prepare it for the machine learning algorithms. The test set will follow a preprocessing parallel to the train

- Numerical variabels stay the same (quantitative)
- Ordinal variables (categories with order) stay the same (quantitative)
- Farmacos are considered ordinals (quantitative)
- Nominal variables (categories with no defined order) are reformatted to one-hot (qualitiative)
- Binary stays the same (qualitiative)
- Race is reformatted to one-hot (qualitiative)
- Sex stays the same (only made into 1/0) (qualitiative)
- Etiquetas. There is room for 3 etiquetas, but it is only their presence that matters. Therefore they are passed to one-hot encoding. The i,j cell will store how many times the jth etiqueta is present in the ith sample i.e. 0 to 3 times if it appears in none or 3 slots, respectively (qualitiative)

Once the function is defined, it was applied to preprocess the training and test datasets in parallel. See appendix B for source code.

5 Principal Component Analysis (PCA)

The distribution of edad (age), raza (race), gender (sexo) and Y (label) across individuals on the 2D plane capturing the most variance can be visualized by means of a PCA. The PCA takes numerical features and rotates them into a new space where variance (information) is maximised on each new feature (principal component). The first two can be used to generate the mentioned 2D plane. This is shown in figure 2

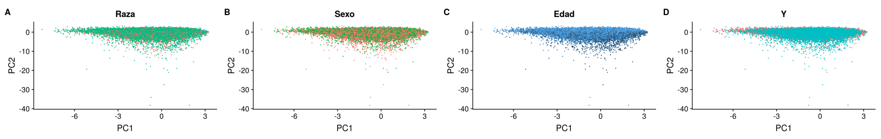


Figure 2: PCA plots for individuals colored by race, gender, age and label. No category seems to be clearly clustering on the 2D plane in any of the 4 features probed.

The PCA did not successfully evidence any clear pattern.

6 Multiple Correspondence Analysis (MCA)

An analogous analysis can be performed using categorical variables with the MCA algorithm from the R package FactoMineR. The algorithm was run with and without etiquetas (tags) to explore if saving these ensemble of one-hot features could be spared for computational efficiency. A plot of the contribution of the explored variables to the new 2D plane is shown in figure 3, whereas a plot of the individuals in this plane is shown on figure 4.

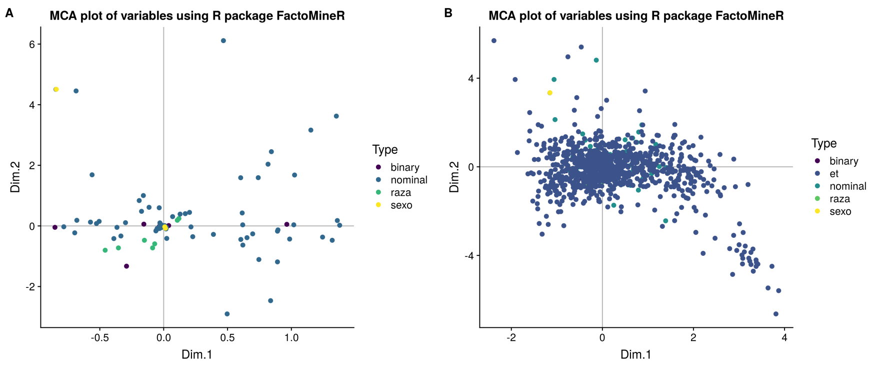


Figure 3: Visualization of the contribution of each categorical variable to the most informative MCA 2D plane. Etiquetas and nominal variables seemed to bring the most information. **A** Without etiquetas. **B** With etiquetas.

The analysis, whose results are shown in figures the mentioned figures, is able to successfuly find a pattern whereby individuals clustering in the top left corner are highly likely to be of class 0 (Y=0). The features extracted in this analysis will be added to the processed data in order to make use of their predictive power in the model training. Remarkably, if MCA is conducted without etiquetas, many individuals are projected on the same spot of the 2D plane. This effect is canceled when etiquetas are included, indicating that indeed etiquetas contribute to each of the individuals' diversity. For this reason, the top 5 MCA features produced when including etiquetas will be used to power the models. Figure 5 shows the single greatest contributions come from nominal variables, even though the whole of the etiquetas might contribute more in the end, given the sheer amount of etiquetas.

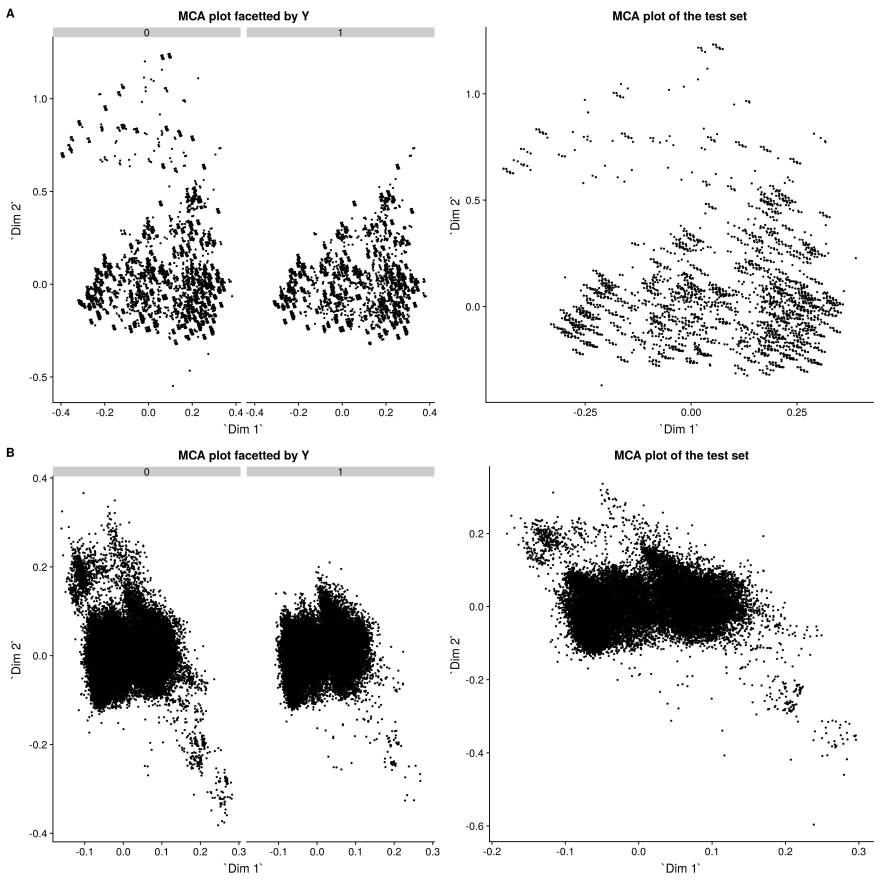


Figure 4: A visualization of the individuals on the MCA 2D plane facetted by label. A clear separation is visible for some individuals, clustering around the top left corner of the plot. A very similar pattern is observed in the test set. **A** Without etiquetas. **B** With etiquetas.

7 Heatmap

Finally, a heatmap (figure 6) is an alternative way to visualize the presence of any patterns/clustering in the processed dataset.

3 Discussion

More insights could be extracted by performing supervised projections of the dataset, where new highly informative planes are found taking into account the label. Moreover, more intensive processing, like the usage of an autoencoder, could be used to perform more powerful dimensionality reduction on the categorical variables, specially the etiquetas. The MCA analysis also indicates which could be the most important features. Further work engineering them is expected to improve model performance.

4 Session info

- 12 GB RAM
- 8 processors Intel(R) Core(TM) i7-6700HQ CPU at 2.60GHz

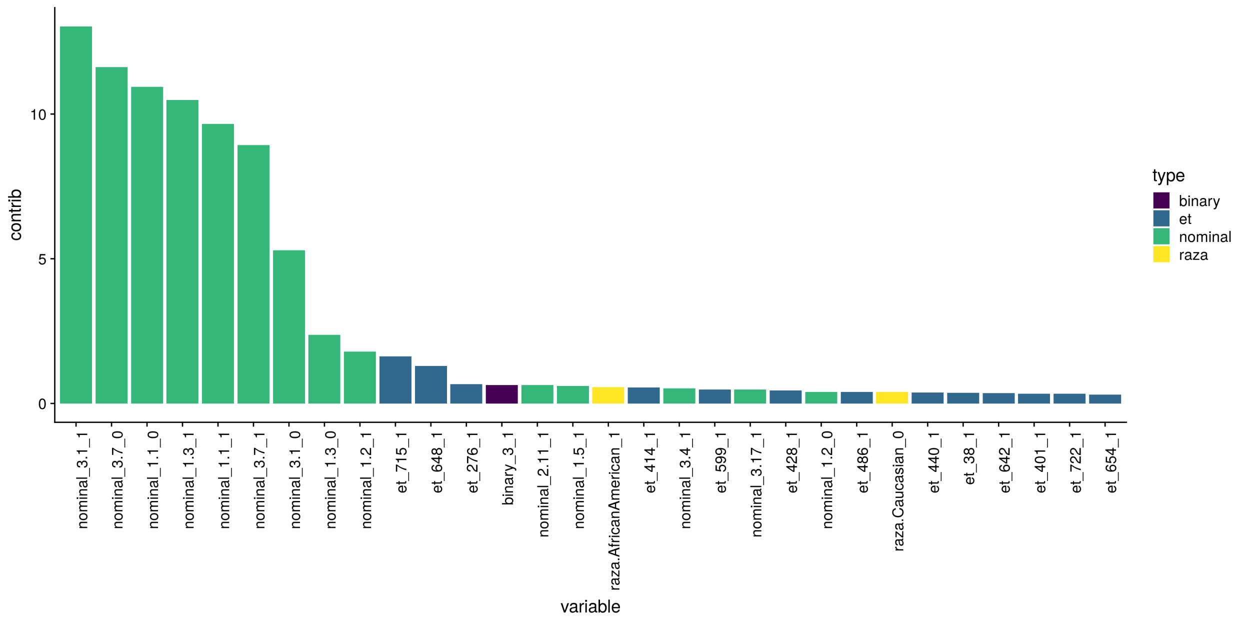


Figure 5: Top 30 variables contributing to the overall variance found by the MCA ordered by its size and coloured by type.

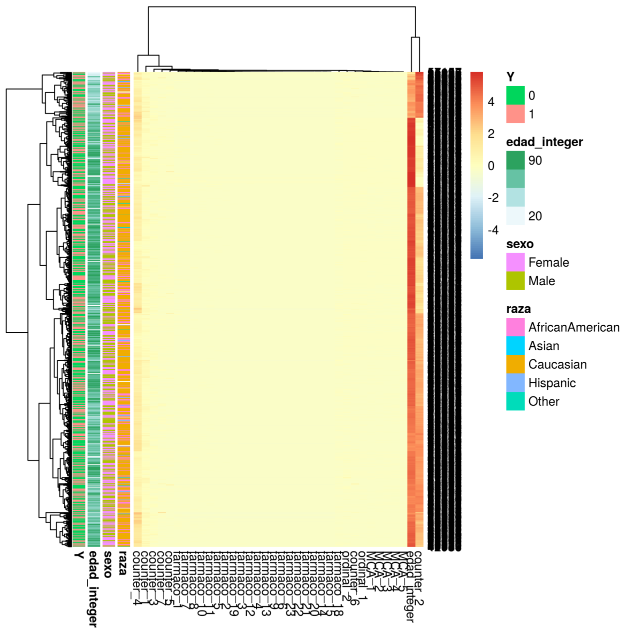
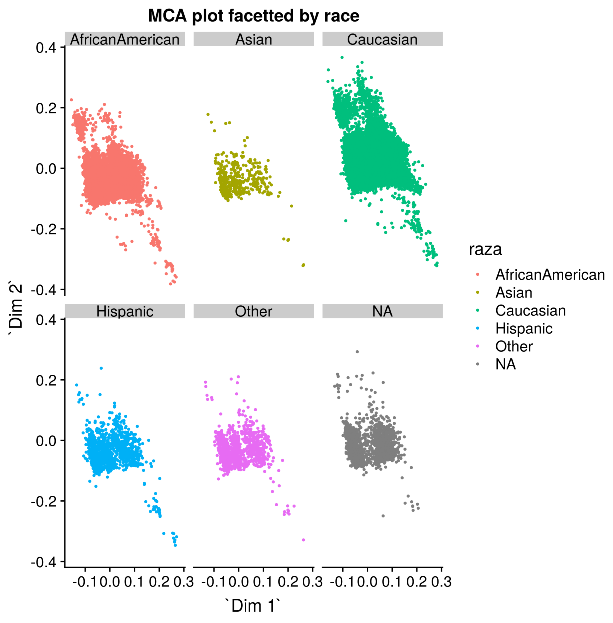


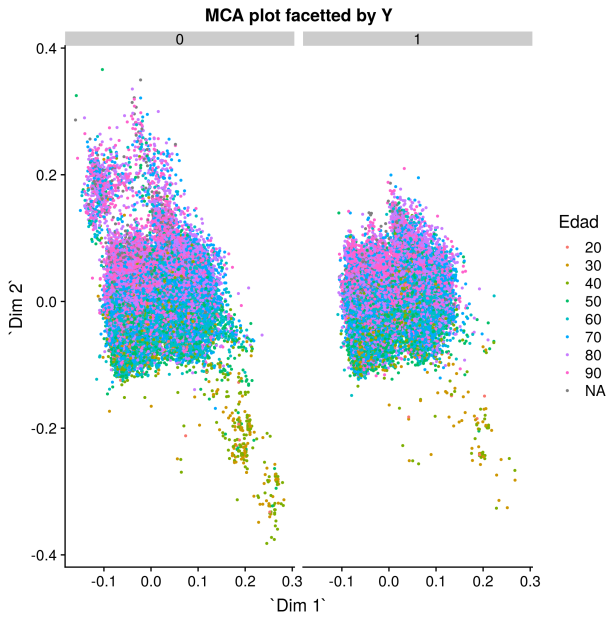
Figure 6: Heatmap of the training set including the top 5 MCA features. As expected, age and the counters with the most levels/categories are the first splitting features. However, the MCA features don't seem to contribute to the clustering.

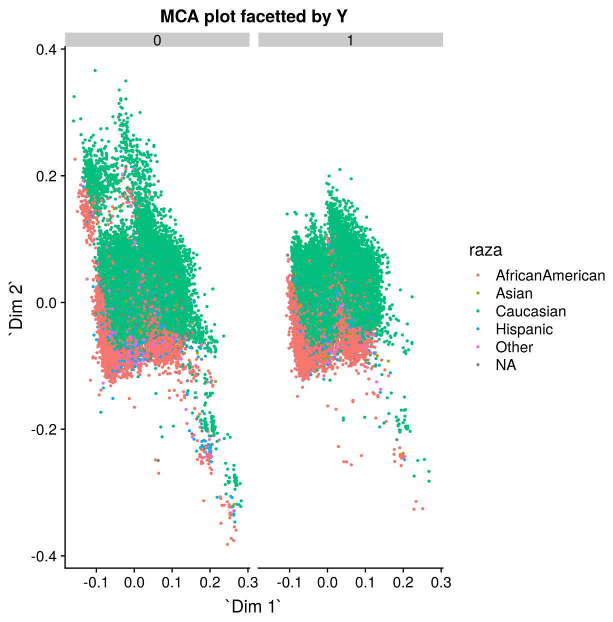
```
sessionInfo()

## R version 3.4.4 (2018-03-15)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.5 LTS
```

```
##
## Matrix products: default
## BLAS: /usr/lib/openblas-base/libblas.so.3
## LAPACK: /usr/lib/libopenblasp-r0.2.18.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=es_ES.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=es_ES.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=es_ES.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=es_ES.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] kableExtra_0.9.0  pheatmap_1.0.10   cowplot_0.9.2     FactoMineR_1.41
##  [5] stringr_1.3.1     data.table_1.11.8 ade4_1.7-11       tibble_1.4.2
##  [9] tidyr_0.8.1       waffle_0.8.0      magrittr_1.5      dplyr_0.7.7
## [13] viridis_0.5.1     viridisLite_0.3.0 ggplot2_3.1.0     knitr_1.20
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.5     purrr_0.2.5          lattice_0.20-35
##  [4] colorspace_1.3-2     htmltools_0.3.6      rlang_0.3.0.1
##  [7] pillar_1.3.0         glue_1.3.0           withr_2.1.2
## [10] RColorBrewer_1.1-2   bindrcpp_0.2.2       bindr_0.1.1
## [13] plyr_1.8.4           munsell_0.5.0        gtable_0.2.0
## [16] rvest_0.3.2          codetools_0.2-15     leaps_3.0
## [19] evaluate_0.10.1      extrafont_0.17       curl_3.2
## [22] Rttf2pt1_1.3.6       highr_0.7            Rcpp_0.12.19
## [25] readr_1.1.1          scales_1.0.0.9000    backports_1.1.2
## [28] flashClust_1.01-2    formatR_1.5          scatterplot3d_0.3-41
## [31] gridExtra_2.3        hms_0.4.2            digest_0.6.18
## [34] stringi_1.2.4        grid_3.4.4           rprojroot_1.3-2
## [37] tools_3.4.4          lazyeval_0.2.1       cluster_2.0.7-1
## [40] crayon_1.3.4         extrafontdb_1.0      pkgconfig_2.0.2
## [43] MASS_7.3-48          xml2_1.2.0           httr_1.3.1
## [46] rstudioapi_0.8       assertthat_0.2.0     rmarkdown_1.10
## [49] R6_2.3.0             compiler_3.4.4
```

5 Appendix A


MCA plot facetted by race
AfricanAmerican
Asian
Caucasian
Hispanic
Other
NA
`Dim 2`
`Dim 1`
0.4
0.2
0.0
-0.2
-0.4
-0.1 0.0 0.1 0.2 0.3
raza
AfricanAmerican
Asian
Caucasian
Hispanic
Other
NA


MCA plot facetted by Y
0
1
`Dim 2`
`Dim 1`
0.4
0.2
0.0
-0.2
-0.4
-0.1 0.0 0.1 0.2 0.3
Edad
20
30
40
50
60
70
80
90
NA



6 Appendix B

```r
preprocess_data <- function(x_train, x_test, etiquettes = FALSE) {

    # Every feature is preprocessed in groups according to their type (nominal,
    # ordinal, etc) Train and test sets have separate pieces of code for 'hygiene',
    # even if it implies code cluttering The result for each group is a list with the
    # processed data for the training and test set respectively, in data.frame format

    if (etiquettes) {
        ## Preprocess etiquettes
        print("Processing etiquettes")
        train_etiquettes <- x_train[, c("etiqueta_1", "etiqueta_2", "etiqueta_3")]
        test_etiquettes <- x_test[, c("etiqueta_1", "etiqueta_2", "etiqueta_3")]
        rownames(train_etiquettes) <- x_train$identificador
        rownames(test_etiquettes) <- x_test$identificador

        # Extract a vector storing all the etiquetas occuring in the TRAINING SET ONLY
        unique_etiquettes <- train_etiquettes %>% unlist %>% unique %>% sort

        # Encode the etiquetas in format one-hot Initialize of vector of counts of
        # etiquetas to 0 to all of them
        etiquettes_template <- rep(0, length(unique_etiquettes))
        names(etiquettes_template) <- unique_etiquettes

        # For both train and test set
        etiquettes_proc <- lapply(list(train = train_etiquettes, test = test_etiquettes),
            function(y) {
                # For every row, apply the same function
                res <- y %>% as.matrix %>% apply(., 1, function(x) {
                  # Count how many times each etiqueta appears This returns a table with the counts
                  # of the etiquetas appearing in this individual but not the ones not appearing
```

```
              # (which obv should be 0)
              local_count <- table(x)
              # Set the counts of the template to 0
              et <- etiquettes_template
              # Drop any etiquette that's not in the training set. Makes sense when analyzing
              # test set
              local_count <- local_count[names(local_count) %in% names(et)]
              # Set the counts of the found etiquetas to the right count
              et[names(local_count)] <- local_count
              return(et)
            }) %>% t %>% unlist

            # Format the colnames of the etiquetas so they start with the et_ prefix
            colnames(res) <- paste0("et_", colnames(res))
            # Make the data structure a data.frame of factors
            res <- res %>% apply(., 2, as.factor) %>% as.data.frame
            return(res)
        })
}

## Preprocess nominals
print("Processing nominals")
# Drop nominal 4 Make one-hot encoding using acm.disjonctif
train_nominals_one_hot <- x_train %>% select(nominal_1:nominal_3) %>% acm.disjonctif() %>%
    apply(., 2, as.factor) %>% as.data.frame
test_nominals_one_hot <- x_test %>% select(nominal_1:nominal_3) %>% acm.disjonctif() %>%
    apply(., 2, as.factor) %>% as.data.frame

# Drop the nominal categories not present in the training set
missing_nominal_test <- colnames(train_nominals_one_hot)[!colnames(train_nominals_one_hot) %in%
    colnames(test_nominals_one_hot)]
missing_test <- as.data.frame(matrix(0, nrow = nrow(x_test), ncol = length(missing_nominal_test)))
colnames(missing_test) <- missing_nominal_test
test_nominals_one_hot <- cbind(test_nominals_one_hot, missing_test)[colnames(train_nominals_one_hot

# Make the list
nominals_proc <- list(train = train_nominals_one_hot, test = test_nominals_one_hot)

## Preprocess drugs (farmacos)
print("Processing drugs")

# Select the farmaco features
train_drugs <- x_train[, grep(pattern = "farmaco", x = colnames(x_train))]
test_drugs <- x_test[, grep(pattern = "farmaco", x = colnames(x_test))]

# Drop drugs with no variability (non-informative)
train_drugs <- train_drugs[, train_drugs %>% apply(., 2, function(x) length(unique(x))) >
    1]
test_drugs <- test_drugs[, colnames(train_drugs)]

# Replace strings with integer and make list
drugs_proc <- list(train = train_drugs, test = test_drugs) %>% lapply(function(x) {
    x[x == "No"] <- "-1"
    x[x == "Down"] <- "0"
    x[x == "Steady"] <- "1"
    x[x == "Up"] <- "2"
    x <- x %>% apply(., 2, as.integer) %>% as.data.frame
})


## Preprocess ordinal
print("Processing ordinals")

# Select the features
train_ordinal <- x_train %>% select(ordinal_1:ordinal_2)
test_ordinal <- x_test %>% select(ordinal_1:ordinal_2)

# Replace strings with integer and make list
ordinals_proc <- lapply(list(train = train_ordinal, test = test_ordinal), function(x) {
    x[x == "None"] <- "0"
    x[x == "Norm"] <- "1"
    x[x == ">7" | x == ">200"] <- "2"
    x[x == ">8" | x == ">300"] <- "3"
    x <- x %>% apply(., 2, as.integer) %>% as.data.frame
    x
})

## Preprocess binary
print("Processing binaries")

# Select the binary features and make them factors of 0 and 1.
train_binary <- x_train %>% select(binary_1:binary_3) %>% apply(., 2, function(x) as.factor(as.integ
    1)) %>% as.data.frame
test_binary <- x_test %>% select(binary_1:binary_3) %>% apply(., 2, function(x) as.factor(as.integer
    1)) %>% as.data.frame

# Make the list
binary_proc <- list(train = train_binary, test = test_binary)


## Preprocess counter
print("Processing counters")

# Just make the list (no need to modify them :))
counter_proc <- list(train = x_train %>% select(counter_1:counter_7), test = x_test %>%
    select(counter_1:counter_7))

## Preprocess race
print("Processing race")

# Make the list while one-hot encoding the same way as nominals
race_proc <- list(train = x_train %>% select(raza) %>% acm.disjonctif() %>% apply(.,
    2, as.factor) %>% as.data.frame, test = x_test %>% select(raza) %>% acm.disjonctif() %>%
    apply(., 2, as.factor) %>% as.data.frame)

## Preprocess sex
```

```r
    print("Processing sex")

    # Select the sex feature and process them like binaries
    sex_proc <- list(train = x_train %>% select(sexo) %>% apply(., 2, function(x) as.factor(as.integer(a
        1)) %>% as.data.frame, test = x_test %>% select(sexo) %>% apply(., 2, function(x) as.factor(as.i
        1)) %>% as.data.frame)

    ## Preprocess age
    print("Processing age")

    # Select the age feature in integer format
    train_age <- x_train$edad_integer
    test_age <- x_test$edad_integer

    # Compute the mean age in the TRAINING SET ONLY
    train_age_mean <- mean(train_age, na.rm = T)

    # Impute missing ages using this mean on both sets
    train_age[is.na(train_age)] <- train_age_mean
    test_age[is.na(test_age)] <- train_age_mean

    # Make the list
    age_proc <- list(train = data.frame(edad_integer = train_age), test = data.frame(edad_integer = test

    ## CBIND all the features for each dataset returning a single data.frame
    ## Optionally include etiquetas (huge amount of one-hot columns) Make a list for
    ## each dataset
    datasets <- c("train", "test")
    if (etiquettes) {
        processed_datasets <- lapply(datasets, function(x) {
            cbind(race_proc[[x]], sex_proc[[x]], age_proc[[x]], nominals_proc[[x]],
                counter_proc[[x]], drugs_proc[[x]], ordinals_proc[[x]], binary_proc[[x]],
                etiquettes_proc[[x]])
        })
    } else {
        processed_datasets <- lapply(datasets, function(x) {
            cbind(race_proc[[x]], sex_proc[[x]], age_proc[[x]], nominals_proc[[x]],
                counter_proc[[x]], drugs_proc[[x]], ordinals_proc[[x]], binary_proc[[x]])
        })
    }

    # Name the list and return it
    names(processed_datasets) <- datasets
    return(processed_datasets)
}

## Preprocess using the function above without and with etiquetas Store the indes
## of quantitative and qualitative features
```

7 Appendix C

```r
# Save processed datasets without one-hot encoding data but with mca features
write.table(x = train_sup_proj_data$proj, file = file.path(output_data, "x_train_mca.csv"),
    sep = ",", row.names = x_train$identificador, quote = F, col.names = T)
write.table(x = test_sup_proj_data$proj, file = file.path(output_data, "x_test_mca.csv"),
    sep = ",", row.names = x_test$identificador, quote = F, col.names = T)

# Save processed datasets with one-hot encoding data (full) and the mca features
# found with them too
write.table(x = train_sup_proj_data_full$proj, file = file.path(output_data, "x_train_mca_full.csv"),
    sep = ",", row.names = x_train$identificador, quote = F, col.names = T)
write.table(x = test_sup_proj_data_full$proj, file = file.path(output_data, "x_test_mca_full.csv"),
    sep = ",", row.names = x_test$identificador, quote = F, col.names = T)

# Save label to a separate file
write.table(x = y_train, file = file.path(output_data, "y_train.csv"), sep = ",",
    row.names = x_train$identificador, quote = F, col.names = T)
```

models

October 28, 2018

Technical test for Junior Data Scientist position at Datrik Intelligence
Antonio Ortega

1 Prepare environment

1.1 Load libraries

```
In [56]: import pandas as pd
         import matplotlib.pyplot as plt
         import numpy as np
         import os.path
         from sklearn.model_selection import GridSearchCV
         from sklearn.decomposition import PCA
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import RidgeClassifier
         from sklearn.ensemble import GradientBoostingClassifier
         from sklearn.ensemble import AdaBoostClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.metrics import auc
```

1.2 Utility functions

```
In [2]: def plot_roc(y_train, y_predict_train_bin, y_test, y_predict_test_bin):
          lw=2
          fpr = dict()
          tpr = dict()
          roc_auc = dict()
          fpr[0], tpr[0], _ = roc_curve(y_true=y_train, y_score = y_predict_train_bin)
          roc_auc[0] = auc(fpr[0], tpr[0])
          fpr[1], tpr[1], _ = roc_curve(y_true=y_test, y_score = y_predict_test_bin)
          roc_auc[1] = auc(fpr[1], tpr[1])
          cols = ['darkorange', "yellow"]
          datasets = ["Train", "Test"]
          plt.figure()
          for i in range(2):
            plt.plot(fpr[i], tpr[i], color=cols[i],
                     lw=lw, label='%s (area = %0.2f)' % (datasets[i], roc_auc[i]))

          plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
          plt.xlim([0.0, 1.0])
```

```
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic (ROC)')
    plt.legend(loc="lower right")
    plt.show()
```

1.3 Load datasets

```
In [36]: x_train_mca = pd.read_csv(os.path.join("proc_data", "x_train_mca_full.csv"), dtype=np.float64)
         x_train = pd.read_csv(os.path.join("proc_data", "x_train.csv"), dtype=np.float64)
         x_train = pd.concat([x_train, x_train_mca.filter(like='MCA_')], axis=1)

         x_pred_mca = pd.read_csv(os.path.join("proc_data", "x_test_mca_full.csv"))
         x_pred = pd.read_csv(os.path.join("proc_data", "x_test.csv"), dtype=np.float64)
         x_pred = pd.concat([x_pred, x_pred_mca.filter(like='MCA_')], axis=1)

         y_train = pd.read_csv(os.path.join("proc_data", "y_train.csv"))
         X_train, X_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.33, random_s
```

1.4 Overview of the input data

```
In [4]: print("{} features loaded".format(X_train.columns.values.shape[0]))
        feature_count = np.unique([e.replace(".", "_").split("_")[0] for e in X_train.columns.values.to
        pd.DataFrame({"feature":feature_count[0], "count": feature_count[1]}).T
```

```
815 features loaded
```

```
Out[4]:            0       1        2     3    4        5        6        7     8  \
        feature  MCA  binary  counter  edad   et  farmaco  nominal  ordinal  raza
        count      5       3        7     1  719       21       51        2     5

                   9
        feature  sexo
        count      1
```

1.5 Scale the dataset

This is carried out in order to give the same importance to all variables, as *a priori* we do not know anything about the label and thus cannot state any feature should be more important than any other.

```
In [5]: scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
```

2 Linear regression model

2.1 Fit (train) the best model to the training set

Use the implemented Ridge Classifier model from scikit-learn. Perform grid search 5-fold cross-validation to obtain a robust AUC estimate while selecting the best alpha (penalizing factor over linear weights). Try values [0,1,5,7,9,10]

```
In [6]: ridge = RidgeClassifier(max_iter=2000)
        parameters = {"alpha": [0, 1, 5, 7, 9, 10]}
        ridge_cv = GridSearchCV(estimator=ridge, param_grid=parameters, cv=5)
```

```
In [7]: ridge_cv.fit(X_train, y_train.Y)

Out[7]: GridSearchCV(cv=5, error_score='raise-deprecating',
               estimator=RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=2000, normalize=False, random_state=None, solver='auto',
                tol=0.001),
               fit_params=None, iid='warn', n_jobs=None,
               param_grid={'alpha': [0, 1, 5, 7, 9, 10]}, pre_dispatch='2*n_jobs',
               refit=True, return_train_score='warn', scoring=None, verbose=0)
```

2.2 Show the best performing model as resulting from CV

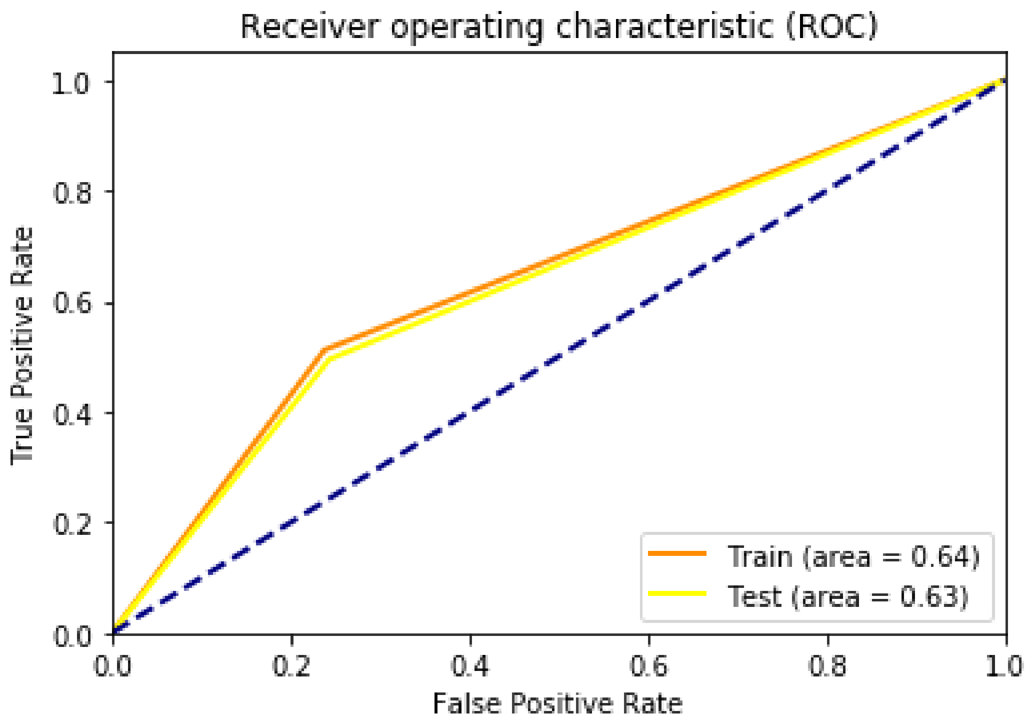
```
In [8]: ridge_cv.best_estimator_

Out[8]: RidgeClassifier(alpha=10, class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=2000, normalize=False, random_state=None, solver='auto',
                tol=0.001)
```

2.3 Exploit (predict) the trained model in the training and test sets

```
In [53]: y_predict_train = ridge_cv.predict(X_train)
         y_predict_train_bin = np.round(y_predict_train)
         print("Accuracy score (train): {0:.3f}".format(accuracy_score(y_train, y_predict_train_bin, no
         print("AUC score (train): {0:.3f}".format(roc_auc_score(y_train, y_predict_train_bin)))

         y_predict_test = ridge_cv.predict(X_test)
         y_predict_test_bin = np.round(y_predict_test)
         print("Accuracy score (test): {0:.3f}".format(accuracy_score(y_test, y_predict_test_bin, normal
         print("AUC score (test): {0:.3f}".format(roc_auc_score(y_test, y_predict_test_bin)))

Accuracy score (train): 0.647
AUC score (train): 0.637
Accuracy score (test): 0.635
AUC score (test): 0.626
```

2.4 Display ROC curve for both sets

```
In [54]: plot_roc(y_train, y_predict_train_bin, y_test, y_predict_test_bin)
```



2.5 Save predictions of true test (pred) to separate file

```
In [55]: prediction = pd.DataFrame(data = ridge_cv.predict(x_pred), index = x_pred.index, columns=["pre
         prediction.index.names = ['identificador']
         prediction.to_csv(os.path.join("predictions", "linear_classifier.csv"))
```

3 Gradient Boosting model

3.1 Fit (train) the best model to the training set

Use the implemented Gradient Boosting Classifier model from scikit-learn. Perform grid search 5-fold cross-validation to obtain a robust AUC estimate while selecting the learning rate (the strength of the update), the maximum number of features (how many branches are possible at each split) and the maximum number of leafs (constraints the size of the tree and is traded off by the number of trees).

```
In [11]: gb = GradientBoostingClassifier(random_state = 0, subsample = 0.9, n_estimators=500)
         param_grid = dict(
             learning_rate = [0.05, 0.1, 0.3],
             max_features=[1,5,10],
             max_leaf_nodes = [2,3,4])
         gb_cv = GridSearchCV(estimator=gb, param_grid=param_grid, cv=5, verbose=True,n_jobs=2)
```

```
In [12]: gb_cv.fit(X_train_scaled, y_train.values.reshape((-1,)))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:  6.7min
[Parallel(n_jobs=2)]: Done 180 out of 180 | elapsed: 26.3min finished
```

```
Out[12]: GridSearchCV(cv=5, error_score='raise-deprecating',
                estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
                       learning_rate=0.1, loss='deviance', max_depth=3,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_sampl...      subsample=0.9, tol=0.0001, validation_fract
                       verbose=0, warm_start=False),
                fit_params=None, iid='warn', n_jobs=2,
                param_grid={'learning_rate': [0.05, 0.1, 0.3, 0.5], 'max_features': [1, 5, 10], 'max_lea
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=True)
```

3.2 Show the best performing model as resulting from CV

```
In [18]: gb_cv.best_estimator_
```

```
Out[18]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
                       learning_rate=0.1, loss='deviance', max_depth=3,
                       max_features=10, max_leaf_nodes=4, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=500, n_iter_no_change=None, presort='auto',
                       random_state=0, subsample=0.9, tol=0.0001,
                       validation_fraction=0.1, verbose=0, warm_start=False)
```

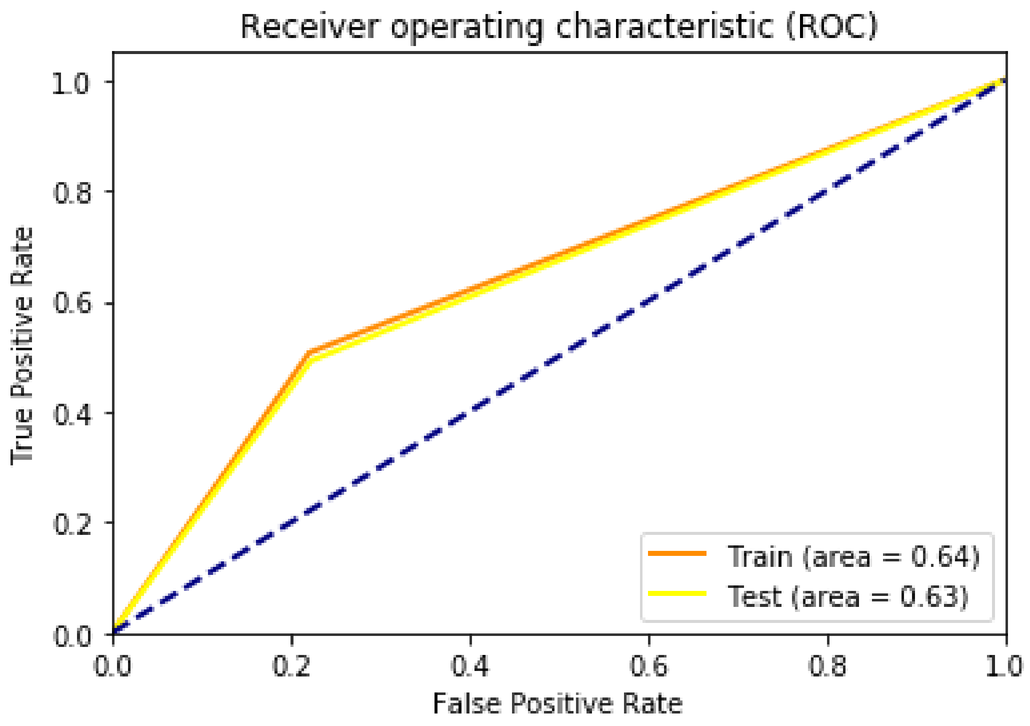
3.3 Exploit (predict) the trained model in the training and test sets

```
In [113]: y_predict_train = gb_cv.predict_proba(X_train_scaled)[:,1]
          y_predict_train_bin = np.round(y_predict_train)
          print("Accuracy score (training): {0:.3f}".format(
                  accuracy_score(y_train, y_predict_train_bin, normalize=True, sample_weight=None)
              ))
          print("AUC score (training): {0:.3f}".format(roc_auc_score(y_train, y_predict_train_bin)))

          y_predict_test = gb_cv.predict_proba(X_test_scaled)[:,1]
          y_predict_test_bin = np.round(y_predict_test)
          print("Accuracy score (test): {0:.3f}".format(
                  accuracy_score(y_test, y_predict_test_bin, normalize=True, sample_weight=None)
              ))
          print("AUC score (test): {0:.3f}".format(roc_auc_score(y_test, y_predict_test_bin)))
          # print("Learning rate: ", learning_rate)
          # print("Accuracy score (training): {0:.3f}".format(gb.score(X_train_scaled, y_train)))
          # print("Accuracy score (validation): {0:.3f}".format(gb.score(X_test_scaled, y_test)))
```

```
Accuracy score (training): 0.654
AUC score (training): 0.643
Accuracy score (test): 0.645
AUC score (test): 0.635
```

3.4 Display ROC curve for both sets

```
In [20]: plot_roc(y_train, y_predict_train_bin, y_test, y_predict_test_bin)
```



3.5 Save predictions of true test (pred) to separate file

```
In [41]: prediction = pd.DataFrame(data = gb_cv.predict_proba(x_pred)[:,1], index = x_pred.index, colum
         prediction.index.names = ['identificador']
         prediction.to_csv(os.path.join("predictions", "gradient_boosting.csv"))
```

4 Discussion

While both models achieve similar AUC scores, around 0.64, they are far from perfect and couldbe improved. As mentioned in the EDA, further data engineering prior to model training could make this process easier. Moreover, models with more capacity (i.e. more iterations, etc) or more complex (neural networks) could have been developed but were not due to computing power capacity, time and the scope of the project. Some of the frameworks offering such models are keras and PyMC3. PyMC3 is a probabilistic programming framework that could have been used here to return a bayesian estimate (true probability based on model and data) of the label being 1.

The models have nevertheless partially captured the function connecting the input features to the target label.