Machine Learning Laboratory

Half-term Report

Miguel Taibo Martínez

September 2021

Contents

1 Problem Description

Quiron hospital in Malaga has gathered a dataset with relevant information to predict Obstructive Sleep Apnoea (OSA) in patients. This dataset has information about patients as Age, Gender, Weight, Height or whether they are snorers, as well as the target attribute Apnoea Hypopnea Index (AHI). Therefore, our main objective is to develop a machine learning procedure and obtain models that can predict OSA and its severity on new patients.

The problem will be divided into a regression and a classification problem. In the regression problem, AHI, which is directly related with OSA, will be predicted. Meanwhile, for classification a transformation based on medical research will be developed to get OSA severity, which will be predicted.

In this report we are going to focus on resources, libraries and code used in order to develop the machine learning procedure, which theoretical explanation is described in the PRDL report.

1.1 Machine Learning approach

In this project, we will start from a dataset ($Info\_BDApnea\_QuironMalaga$) and obtain an effective model to predict if a patient suffers from OSA. As we have been taught in lectures[1] and as we can see in books such as Aurélien Géron(2017)[2], it is necessary to split this task to perform it efficiently. Therefore, we are going to divide our project, and the report as well, into 6 stages.

1. **Problem description**: The study and definition of the problem is essentially both to have a clear idea of what is to be achieved and the steps to be followed to achieve it. Therefore, it is the first thing to do when looking for any solution.

2. **Data description**: To train any machine learning model, a dataset is needed. Thus, the next step will be to obtain this data. In our case, the data has already been provided to us, but we will have to carry out a preparation process. Additionally, a description of the data is necessary for a correct preparation.

3. **Exploratory Data Analysis**: A deep statistical analysis of the data will be developed, in comparison with the description of the previous stage. The complete understanding of the data is needed in order to efficiently experiment and design ML models. The visualization of correlations, distributions and data features will help us to obtain knowledge of the data.

4. **Feature Engineering**: In this stage, we will be using domain knowledge to extract features from the raw dataset. This will make the training a lot more efficient. This can consist in many different transformations of the data, such as attribute selection, reducing dimensionality (PCA) or scaling one attribute.

5. **Machine Learning models**: Several ML models will be trained. In this section different algorithms, hyperparameter selection and model validation methods (cross-validation) will be studied and evaluated.

6. **Evaluation**: The different Machine Learning models obtained are evaluated to select which ones perform better. This is not as simple as it looks, different performance metrics can be and will be taken into account. Therefore, visualization of this metric will be essential.x

7. **Conclusions**: This stage is not a part of the machine learning approach, although it is essential to synthesize what has been learned and the conclusions that have been reached during the project.

1.2 Set up

The project will be developed in R, using RStudio. Many resources and libraries will be used, although they will be explained as they are used in this report. RStudio also provides us with an environment to easily install and import the different R packages. The hardware to develop and run the project's code consists of a laptop with an Intel Core i7 of 8th generation and 16GB RAM memory with Ubuntu 19.04 as operating system.

GitHub will be used for version control and project storage. A personal GitHub repository has been created, https://github.com/MiguelTaibo/OSA-CODE-R. As we have just developed scripts, code structure is not complex at all. All the files referenced in this report can be found on that repository, although, most relevant parts of the code have been presented in the following sections.

Among these libraries, we can highlight **ggplot2**, **dplyr** and **caret**. **ggplot2** is an efficient and flexible framework for visualization of data in R. **dplyr** is a set of R functions designed for data manipulation. Both *ggplot2* and *dplyr* are part of *tydiverse*, the most popular and used collection of R packages specially designed for data science. Therefore, they have a large community and a lot of resources available. **caret** is a framework with machine learning tools. It has tools to develop different stages of a machine learning procedure, but we are going to use it to train all our machine learning models.

Instead of using my own laptop, I could be using a cloud computing platform, such as Rstudio cloud (https://rstudio.cloud/) for R or google Colab for python. These platforms provide a fast and easy-to-use service, as you do not have to worry about packages and versions. This also makes easier interoperability, as every developer works with the same versions and packages. Additionally, they securely store your files without threat of losing them, and you can even store past versions. Finally, using these platforms, you are not using your memory and computational resources. Nevertheless, you are also provided with a part of a server which, many times, can be very slow. Additionally, you are unable to work on the project when you do not have connectivity. These are the main reasons to use my own machine.

Usage of python was also considered in order to develop the project. Python is one of the most popular programming languages with a huge community. This context provides Python with great data science frameworks, such as *scikit-learn* or *pandas*. Additionally, it is a language that grows year after year due to its simplicity and tools. There are two main tools for package and version management with python:

- **virtualenv** and **pip**: *virtualenv* creates a virtual environment with a python interpreter already installed on the machine. Afterwards, we can easily activate the virtual environment and install python packages using *pip*, which just apply to the virtual environment. It allows creating a list of installed packages which can be installed automatically on a project to improve interoperability.
- **Conda** is equivalent to *virtualenv* and *pip*. Additionally, it provides a graphical interface which makes it extremely easy. Moreover, it also works for package management in R. Nevertheless, in my experience, Conda uses a large amount of space for the virtual environments. Therefore, this tool can be very useful but simultaneously unscalable.

The main reason we have selected to use R is to learn about it, as I have already work with Python and I do not have experience with R. R is a language designed by statisticians, therefore, there can be fundamental differences among the two languages, so learning this language can apport to me more than just knowing another programming language.

2 Data Wrangling

In this project, we will start from the $Info\_BDApnea\_QuironMalaga$ dataset in xlsl format. In this section, which corresponding code is $ETL\_OSAData.R$, we are going to read $Info\_BDApnea\_QuironMalaga.xlxs$ file, and we will write $OSA\_DB\_UPM.xlsx$ file, between data will be cleaned and different computations and plot will be done to perform the cleanse. This section describes the procedure to carry on the data description section in the PRDL corresponding report.

At first, we read the Excel file with the data, obtaining a data frame. Then, with the R command $sapply(df\_tmp, class)$, we check the class of the different attributes of the data frame. Comparing the result of this command with the knowledge of the data frame, we know that some are read wrong. Therefore, the first thing we do is to change these classes to numerical and categorical.

Then we visualize the not available values in our data frame to evaluate how we should proceed with cleaning the data. After our analysis, we remove the unwanted attributes, by selecting the useful ones.

```
library(visdat)      # Import visdat library
vis_dat(df_tmp1)     # Visualiza NA values in dataset

library(dplyr)       # Import dplyr library
df_tmp2 <- df_tmp1 %>%
  select(Patient, Gender, Fumador, Roncador, IAH, Peso, Talla, Edad,
      PerCervical)       # Select the wanted attributes
vis_dat(df_tmp2)     # Visualiza NA values in dataset
```

After, as explained in the analysis, we must set all -1 values to not available, making use of the *naniar* library. Then we visualize the not available values in the resulting data frame, and as they are not a significant amount, we decide to drop them all.

```
library(naniar)      # Import naniar library
df_tmp3 <- df_tmp2 %>% replace_with_na(replace = list(
    Peso=-1, Talla=-1, Edad=-1, PerCervical=-1
  ) )                # Replace all -1 values with NAs

vis_dat(df_tmp3)     # Visualiza NA values in dataset
library(tidyr)       # Import tidyr library
df_final <- df_tmp3 %>%
    drop_na()        # Remove all na values
```

After, we have translated the attribute names and the factor levels using *remane* and *recode_factor* respectively. Below, an example is shown. Additionally, at the end of this stage we recompute the body mass index, as in the dataset it was wrong. We computed $BMI = Weight/Height^2$

```
# Rename of the attribute
df_final <- df_final %>%
    rename(Weight = Peso,      Height = Talla,
```

```
        Smoker = Fumador,   Snorer = Roncador,
        Age = Edad,         Cervical = PerCervical,
        AHI = IAH)

# Example of renaming the factors
df_final$Gender <- df_final$Gender %>%
    recode_factor(hombre = "male", mujer = "female")

# Computation of the BMI
df_OSA <- df_OSA %>%
  mutate(BMI = Weight/(Height/100)^2, .after = Height)
```

To finish the data wrangling, we prepared the data for classification, as it was only suitable for regression. In order to do this, we created a new column (**OSA**) which contained the same information as the target attribute (*AHI*) but in categorical format, not in numerical one. Thresholds from medical research information were found to categorize this information.

```
df_final <- df_final %>%
  mutate(OSA= ifelse(AHI <5, "Healthy", ifelse(AHI<15, "Mild", ifelse(AHI<30,"
      Moderate", "Severe")))) %>%
  transform(OSA = as.factor(OSA))
```

Additionally, different plots to carry on the cleaning procedure and show ideas on the PRDL report have been programmed.

3 Exploratory Data Analysis

In this section, the explanatory data analyse (EDA) will be developed, i.e., investigate and analyse the dataset in order to find and synthesize the main characteristics of the data. Visualization will be a key to achieve conclusions. We have divided this section into EDA for regression, EDA for classification and EDA for unsupervised learning, each one with its own R file: *EDA_Regression.R*, *EDA_Classification* and *EDA_Unsupervised*.

Nevertheless, at the start of each file we will be reading the Excel file produced in the previous section, and we will be working with two data frames. One with all the samples and one taking just healthy and severe patients. Below we can observe the code to develop this stage, which will be done in the three files.

```
rm(list=ls())

Input_file <- "OSA_DB_UPM.xlsx"
Data_Directory <- "~/MLLB/DATA/"
df_OSA <- read_excel(paste(Data_Directory, Input_file, sep = ""))

summary(df_OSA)
df_OSA <- df_OSA %>%
    transform(Patient = as.factor(Patient)) %>%
    transform(Gender = as.factor(Gender)) %>%
    transform(Smoker = as.factor(Smoker)) %>%
    transform(Snorer = as.factor(Snorer)) %>%
    transform(OSA = as.factor(OSA))
summary(df_OSA)

df_OSA_rm <- df_OSA %>% filter(OSA %in% c("Healthy", "Severe"))
df_OSA_rm$OSA <- droplevels(df_OSA_rm$OSA)
summary(df_OSA_rm)
```

3.1 EDA for Regression

In EDA for regression, we start studying categorical features: *Gender*, *Smoker* and *Snorer*. First, we combine box plots and bar plots to get more insights on the relationship between these features and the target one, *AHI*. Afterwards, in order to get more quantitative metrics, we perform a non-parametric kruskal-wallis test. The code below presents an example of each plot for Gender feature and the kruskal-wallis test development.

```
p_Gender <- df_OSA_rm %>%
    ggplot(aes(x=reorder(Gender,AHI,na.rm = TRUE), y=AHI)) +
    geom_boxplot() + labs(x="Gender", y="") +
    stat_summary(fun.data = stat_box_data, geom = "text")

p_Gender <- df_OSA %>%
    ggplot(aes(x=reorder(Gender,AHI,na.rm = TRUE))) +
    geom_bar() + labs(x="Gender")

```

```r
t_Gender_AHI <- kruskal.test(Gender ~ AHI, data = df)
t_Smoker_AHI <- kruskal.test(Smoker ~ AHI, data = df)
t_Snorer_AHI <- kruskal.test(Snorer ~ AHI, data = df)
```

After we start studying the numerical features: *BMI*, *Weight*, *Height*, *Age* and *Cervical*. First, we computed the correlations between the different features (input and target) and constructed a plot in order to visualize them with colours as well as their exact values. This is a quantitative and non-parametric method to study the relationships among the features. After, we studied scatter plots and histograms. *pairs* function allowed us to make a scatter plot for each pair of features and a histogram for each feature and combine them in the same figure.

```r
cor(df_OSA[,5:10])
corrplot(cor(df_OSA[,5:10]), method="color", addCoef.col = "black")

blue_pallete <- c("#d0efff", "#2a9df4", "#187bcd", "#1167b1")

pairs(df_OSA %>%
        select(AHI, Weight, Height, BMI, Age, Cervical) %>%
        filter(AHI<70),
      bg =blue_pallete[df_OSA$OSA],
      pch=21,
      col = blue_pallete[df_OSA$OSA],
      diag.panel = panel.hist ,oma=c(3,3,3,15))
par(xpd = TRUE)
legend("bottomright", fill = blue_pallete[df_OSA$OSA], legend = c( levels(df_
    OSA$OSA)))
```

Finally, we studied feature combination for regression. Several linear regression models were studied, we have one example with all the features in code below. These models were different in the input features. Finally, we perform a different linear model for male patients and female patients. The results (metrics and residuals) of the different models were put together in a data frame and plots were built. We can highlight the violin plot of residuals by gender, which code appears below.

```r
df_OSA_lm <- df_OSA %>%
  mutate(Gender = as.numeric(Gender), Snorer=as.numeric(Snorer), Smoker=as.
      numeric(Smoker))

lm.fit=lm(AHI~Weight+BMI+Height+Cervical+Age+Gender+Snorer,
            data=subset(df_OSA_lm, Snorer %in% c(2,4,5)))
summary(lm.fit)

lm_males.fit=lm(AHI~Weight+BMI+Height+Cervical+Age, data=subset(df_OSA_lm,
    Gender=1))
lm_females.fit=lm(AHI~Weight+BMI+Height+Cervical+Age+Snorer, data=subset(df_
    OSA_lm, Gender=2))
summary(lm_males.fit)
summary(lm_females.fit)

df %>% ggplot(aes(x=Gender, y=residual)) +
        geom_violin() +
```

```
        geom_boxplot(width=.2)
```

3.2 EDA for Classification

As for regression problem, in EDA for classification, we started by studying categorical features. We mainly studied two bar plots for each categorical feature. The first one represented the number of patients with an OSA severity for the levels of the feature. Meanwhile, the second one is equivalent but representing the rate of the corresponding feature level. Therefore, each bar has height 1 and the information is in the filling by OSA severity. Below, we observe an example for each of these plots. Finally, we used chi-squared test to obtain a quantitative result of the EDA. Additionally, this has been done for both data frames with all the patients and filtering healthy and severe patients.

```
p_Gender <- df_OSA %>%
    group_by(Gender,OSA) %>%
    summarise(count=n()) %>%
    ggplot(aes(x=Gender, y=count, fill=OSA)) +
    geom_bar(stat="identity") + theme(legend.position="none")

p_Gender <- df_OSA %>% group_by(Gender) %>%
    summarise(
        Healthy=sum(OSA=="Healthy")/n(),
        Mild=sum(OSA=="Mild")/n(),
        Moderate=sum(OSA=="Moderate")/n(),
        Severe=sum(OSA=="Severe")/n()) %>%
    gather("OSA","rate",-Gender ) %>%
    ggplot(aes(x=Gender, y=rate, fill=OSA)) +
    geom_bar(stat="identity") + theme(legend.position="none")

chisq.test(df_OSA$OSA, df_OSA$Gender)
```

After we studied the numerical features. As this study was already done for the regression problem, it was decided to select the three features (*BMI*, *Cervical* and *Age*) and the dataset with healthy and severe patients due to reasons explained in the PRDL report. We will start by making a scatter plot for each pair of features with the OSA information as colour. After, for each feature, we will create a plot with a histogram for each OSA level. In order to get our quantitative results, we will be using a non-parametric kruskal-wallis test.

```
p1 <- xyplot(BMI ~ Cervical,
        groups = OSA, data = df_OSA_rm,
        auto.key = list(corner = c(1, 1), cex = 0.7), pch=4)

p1 <- ggplot(df_OSA_rm, aes(x = BMI)) +
    geom_histogram(aes(color = OSA), fill = "transparent",
            position = "identity", bins = 30, alpha = 0.1) +
    scale_color_manual(values = c("#00AF00", "#E7B800")) +
    scale_fill_manual(values = c("#00AF00", "#E7B800"))

kruskal.test(Weight ~ OSA, data = df_OSA)$p.value
```

```r
kruskal.test(Height ~ OSA, data = df_OSA)$p.value
kruskal.test(BMI ~ OSA, data = df_OSA)$p.value
kruskal.test(Age ~ OSA, data = df_OSA)$p.value
kruskal.test(Cervical ~ OSA, data = df_OSA)$p.value
```

Finally, we studied feature combination for the classification problem. In order to do this, we developed different logistic regression models. We started with a model with all the features, which gave us a good idea of which features were more and less relevant on these models. After, we perform a model with just one input for each feature. Finally, after some testing we achieved the best model by AIC metric.

```r
df_OSA_lm <- df_OSA_rm %>%
    select(Patient, Gender, Smoker, Snorer, OSA, Weight, Height, BMI, Age,
        Cervical) %>%
    mutate(Gender = as.numeric(Gender), Snorer=as.numeric(Snorer), Smoker=as.
        numeric(Smoker))

glm.fit <- glm(OSA ~ Weight+BMI+Gender+Smoker+Snorer+Height+Age+Cervical,
        data = df_OSA_lm, family = binomial)
summary(glm.fit)

glm_Cervical.fit <- glm(OSA ~ Cervical, data = df_OSA_lm, family=binomial)
summary(glm_Cervical.fit)

### Best model
glm_best.fit <- glm(OSA ~ Weight+BMI+Gender+Snorer+Height+Age, data = df_OSA_
    lm, family = binomial)
glm_best.fit$aic
summary(glm_best.fit)
```

3.3 EDA for Unsupervised Learning

In this section, we focused on principal component analysis (PCA) and t-distributed stochastic neighbour embedding (tSNE). In order to do this, we used two libraries that provide development and visualization tools, *factorextra* for PCA and *Rtsne* for tSNE.

```r
library(factoextra)
library(Rtsne)
```

For the PCA, we start by computing the 8 PCs, as we have 8 features, for our dataset. First we focus on importances of the PCs, after we check the importances of each feature on each PC. We also visualize the projection of all features on 2D space formed by the two first PCs, line 7 on code below. To finish, we plot all the samples on the 2D space formed by the two first PCs. In the code below we present all these steps for the entire dataset, although it was also performed for healthy and severe patients, and results were actually more interesting.

```r
OSA.pca <- df_OSA %>%
    select(Gender, Smoker, Snorer, Weight, Height, BMI, Age, Cervical) %>%
```

```
    prcomp(scale=TRUE)

fviz_eig(OSA.pca, addlabels = TRUE, ylim = c(0, 40))+labs(title="", x="PC", y="
    Eigenvalue")

fviz_pca_var(OSA.pca,
             col.var = "contrib",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE) +
    labs(title="", x="PC1", y="PC2")

p1 <- fviz_contrib(OSA.pca, choice = "var", axes = 1)+labs(title="PC1")
p2 <- fviz_contrib(OSA.pca, choice = "var", axes = 2)+labs(title="PC2")
p3 <- fviz_contrib(OSA.pca, choice = "var", axes = 3)+labs(title="PC3")
p4 <- fviz_contrib(OSA.pca, choice = "var", axes = 4)+labs(title="PC4")
p5 <- fviz_contrib(OSA.pca, choice = "var", axes = 5)+labs(title="PC5")
p6 <- fviz_contrib(OSA.pca, choice = "var", axes = 6)+labs(title="PC6")

grid.arrange(p1, p2, p3, p4, p5, p6, ncol=3, nrow=2)

fviz_pca_ind( OSA.pca, geom="point",
              habillage = df_OSA$OSA,
              addEllipses=TRUE,
              ellipse.level=0.7) +
  labs(title="", x="PC1", y="PC2")
```

tSNE is an algorithm specially developed for visualization. In order to make this visualization with OSA information possible we must first select just the input features of the data frame and OSA feature then create a new column to identify each sample, ID. After, we perform the tSNE algorithm and when plotting we join the previous dataframe using IDs. This was performed for both dataset. Finally, a data frame which features are the PCs instead of the original features was constructed. The tSNE procedure was applied to this dataframe, but the results were not good at all.

```
df_OSA_temp <- df_OSA %>%
    select(-Patient) %>% select(-AHI) %>%
    mutate(ID=row_number())

tSNE_fit <- df_OSA_temp %>%
    select(where(is.numeric)) %>%
    column_to_rownames("ID") %>%
    scale() %>%
    Rtsne(check_duplicates=FALSE)

tSNE_df <- tSNE_fit$Y %>%
    as.data.frame() %>%
    rename(tSNE1="V1",
           tSNE2="V2") %>%
    mutate(ID=row_number()) %>%
    inner_join(df_OSA_temp %>% select(OSA, ID), by="ID") %>%
    select(tSNE1, tSNE2, OSA)
```

4 Experimental Setup

In this section we studied feature scaling, dimensionality reduction and feature selection. All the code from this section can be found in file, *experimental_setup.R* Although, we did not develop any code for dimensionality reduction. Meanwhile, for feature scaling, we just applied a standardization process to all numerical features.

```
## Feature Scaling
df_OSA_scaled <- df_OSA %>%
  mutate(Weight=scale(Weight)) %>%
  mutate(Age=scale(Age)) %>%
  mutate(Height=scale(Height)) %>%
  mutate(BMI=scale(BMI)) %>%
  mutate(Cervical=scale(Cervical))
```

In feature selection, we developed the code for two procedures. Among wrapping techniques, we used RFE algorithm. As a pre-text task, we should separate data into train and test, using a 75%-25% split. After we define the parameters for the algorithm (ML model to use and cross-validation procedure) we can fit the algorithm, which is done in line 19. It is also important to say that this procedure has been done to both dataset with all the patients and with healthy and severe ones and both classification and regression problems.

```
## Feature Selection

# Set train and test data
set.seed(101)
smp_size <- floor(0.75 * nrow(mtcars))
train_ind <- sample(seq_len(nrow(mtcars)), size = smp_size)
df_OSA.train=df_OSA[train_ind,]
df_OSA.test=df_OSA[-train_ind,]

# Wrapping algorithm: RFE
control_rfe = rfeControl(functions = rfFuncs, # random forest
                         method = "repeatedcv", # repeated cv
                         repeats = 5, # number of repeats
                         number = 10) # number of folds
set.seed(50)
X_train <- df_OSA %>%
    select(Weight, Age, BMI, Height, Cervical, Gender, Smoker, Snorer)
Y_train <- df_OSA %>% select(AHI, OSA)

result_rfe_AHI <-
  rfe(x = X_train[,1:8], y = Y_train[,1],
      sizes = c(1:8),
      rfeControl = control_rfe)

result_rfe_AHI
predictors(result_rfe_AHI)
```

Finally, we performed an embedded feature selection technique, making use of random forest as well. As we can see below, by training a random for the data, we can get the

importances of each feature. A function to get an easy to visualize plot has been written. This decision has been done based on the fact that we needed to evaluate 4 models (for the 2 different datasets in the 2 different problems).

```
# Embedded algorithm: based on random forest
model_rf<-randomForest(AHI ~ Weight+BMI+Height+Cervical+Age+Snorer+Smoker+
    Gender, data = df_OSA)

model_rf_importance <- function(model_rf) {
  df <- data.frame(Feature=factor(), Importance=double())
  features <- c("Weight","BMI","Height","Cervical","Age","Snorer","Smoker","
      Gender")
  i <- 1;
  for (f in features) {
    new_row <- data.frame(Feature=f, Importance=model_rf$importance[i]);
    df <- rbind(df,new_row)
    i <- i+1
  }
  df <- df %>% arrange(desc(Importance))
  print(df)
  df %>%
       ggplot(aes(x=reorder(Feature,desc(Importance)), y=Importance))+
       geom_col() +
       labs(x="Importance")
}

model_rf_importance(model_rf)
```

5 Machine Learning Models

In this section, we are going to develop a set of machine learning models for both regression and classification problems. For both problems, we will start by separating the data into train and test (90%-10% split has been used for most experiments) and defining a 10-Fold cross-validation train control.

```
set.seed(101)
smp_size <- floor(0.9 * nrow(df_OSA))
train_ind <- sample(seq_len(nrow(df_OSA)), size = smp_size)
df_OSA.train=df_OSA[train_ind,]
df_OSA.test=df_OSA[-train_ind,]

smp_size_rm <- floor(0.9 * nrow(df_OSA_rm))
train_ind_rm <- sample(seq_len(nrow(df_OSA_rm)), size = smp_size_rm)
df_OSA_rm.train=df_OSA_rm[train_ind_rm,]
df_OSA_rm.test=df_OSA_rm[-train_ind_rm,]

# For 10-Fold crossvalidation
train.control <- trainControl(method = "cv",
                              number = 10,
                              savePredictions=TRUE)
```

5.1 Regression

We have started the regression problem by defining the metrics that will allow us to evaluate the models. These functions get the model, the training data and the test data. As shown below, we compute *MSE*, *MAE* and *R-Squared* metrics. Finally, we developed two more functions, one that printed these metrics to get a fast response and another one that printed the metrics as a latex row, so it could be copied fast and easy, avoiding typing mistakes.

```
get_metrics <- function(model, df.train, df.test, attrs, modelName) {
  pred_train = predict(model, df.train)
  pred_test  = predict(model, df.test)

  MSE_Predict_train <- mean((df.train$AHI - pred_train)^2)
  MSE_Predict_test <- mean((df.test$AHI - pred_test)^2)
  MSE_Naive_test <- mean((df.test$AHI - mean(df.test$AHI))^2)

  MAE_Predict_train <- mean(abs(df.train$AHI - pred_train))
  MAE_Predict_test <- mean(abs(df.test$AHI - pred_test))
  MAE_Naive_test <- mean(abs(df.test$AHI - mean(df.test$AHI)))

  R_Squared_train <- 1-MSE_Predict_train/MSE_Naive_test
  R_Squared_test <- 1-MSE_Predict_test/MSE_Naive_test

  res <- data.frame(Model=modelName, attrs=attrs,
                    MSE=MSE_Predict_test,
                    MAE=MAE_Predict_test,
```

```
                        R_Squared=R_Squared_test)
  return(res)
}
```

In order to train the machine learning model, *caret* framework has been used. It provides a wide range of models at the same time that allows hyperparameter tune-fining and search optimums with a grid search. Below, we present examples of the training for different models. As we can see it is quite simple to train the models just by setting the training data, the model we want to train and the training control (10-Fold cross-validation). Additionally, we can observe that:

1. When developing a grid search, it is important to set *tuneLength* parameter to maintain the commitment between finding the optimum and computational cost.
2. It is also possible to define a custom tune grid, as we can observe for the SVM model.
3. Preprocessing stages are efficient and easy to configure, as we can observe for the SVM model.

It is important to highlight the importance of the *set.seed* function, as it fixes the random number generation, so we can replicate the same results we got previously. I decided to fix a *set.seed* sentence at the beginning of the training for each family of models.

```
set.seed(1)
lm_all.model.train <- train(AHI ~Cervical + Weight + BMI + Gender + Height +
                                  Snorer + Smoker + Age,
                            data = df_OSA.train,
                            method = "lm", trControl=train.control)

lasso.model.train <- train(AHI ~Cervical + Weight + BMI + Gender + Height +
                                 Snorer + Smoker + Age,
                           data = df_OSA.train,
                           method = "lasso", trControl=train.control,
                           tuneLength = 81)

rf_model <- train(AHI ~Cervical + Weight + BMI + Gender + Height +
                        Snorer + Smoker + Age,
                  data = df_OSA.train,
                  method = "rf", trControl=train.control,
                  tuneLength=10)

svm_model <- train(AHI ~Cervical + Weight + BMI + Gender + Height +
                         Snorer + Smoker + Age,
                   data = df_OSA.train,
                   method = "svmLinear", trControl=train.control,
                   preProcess = c("center","scale"),
                   tuneGrid = expand.grid(C = seq(0.1, 2, length = 20)))
```

Finally, the *get_metrics* function was used to fill a new data frame that contains the metrics for all the models trained in this section, a total of 16 models. Although, this new

data frame is not tidy, therefore, using *dplyr* functions *mutate* and *gather* we tidy it. At this point, we are able to plot the results using *ggplot2*. The tidy and plotting code is completely present below. There we create a figure with two plots, one representing Model versus RMSE and the other representing Model versus MAE.

```
df_comparison = data.frame(Model=c(), attrs=c(),
                           MSE=c(),
                           MAE=c(),
                           R_Squared=c())

df_comparison <- rbind(df_comparison,
        get_metrics(
            lm_all.model.train,
            df_OSA.train,
            df_OSA.test,
            "All", "Linear Regression"))

## Tidy the new data frame
df_comparison <- df_comparison %>%
  mutate(RMSE = sqrt(MSE)) %>%
  gather("metric", "value", MSE, MAE, R_Squared, RMSE) %>%
  mutate(metric=as.factor(metric))

p1 <- df_comparison %>%
  filter(metric=="RMSE") %>%
  ggplot(aes(x=reorder(Model,value), y=value, group = attrs, color=attrs)) +
  geom_line() + geom_point() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1)) +
  labs(x="Model", y="RMSE")+
  ylim(16.7,18.7) + theme(legend.position = "none")


p2 <- df_comparison %>%
  filter(metric=="MAE") %>%
  ggplot(aes(x=reorder(Model,value), y=value, group = attrs, color=attrs)) +
  geom_line() + geom_point() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1)) +
  labs(x="Model", y="MAE") + theme(legend.position = "none")

combined <- p1 + p2 & theme(legend.position = "bottom")
combined + plot_layout(guides = "collect")
```

5.2 Classification

In the classification problem the same structure was followed than for the regression problem: training and test data were split, metric computation functions were defined, models were trained, results were evaluated.

In order to get classification models' metric, we must first get the confusion matrix. Then we can compute both accuracy and kappa coefficients with their confidence intervals following the formulas explained in the PRDL report, which appear also below.

$$acc = \frac{1}{S}\sum_{i=1}^{N} CM_{i,i} \tag{1}$$

$$kappa = \frac{P_o - P_e}{1 - P_e} \quad \begin{cases} P_o = \frac{1}{S}\sum_{i=1}^{N} CM_{i,i} = acc \\ P_e = \frac{1}{S^2}\sum_{i=1}^{N}(\sum_{j=1}^{N} CM_{i,j} \cdot \sum_{j=1}^{N} CM_{j,i}) \end{cases} \tag{2}$$

$$\sigma_{acc} = \sqrt{\frac{acc \cdot (1 - acc)}{N}} \tag{3}$$

$$\sigma_{kappa} = \sqrt{\frac{P_o(1 - P_o)}{(1 - P_e)^2}} \tag{4}$$

```r
get_metrics <- function(model, df.train, df.test, lvs, model_name) {

  pred_test <- predict(model, df.test)

  df.test$predicted <- pred_test
  samples.test <- nrow(df.test)

  df_data.test <- df.test %>%
    mutate(goodbad = ifelse(OSA == predicted, "good", "bad")) %>%
    group_by(OSA, predicted, goodbad) %>%
    summarise(N = n(), rate = n()/samples.test)

  tb_data.test <- df_data.test %>%
    select(OSA, predicted, N) %>%
    spread(OSA, N) %>%
    ungroup %>%
    select(-predicted) %>%
    mutate_all(~replace(., is.na(.), 0))
  tb_data.test <- as_tibble(tb_data.test)

  accuracy <- 0
  for (i in 1:nrow(tb_data.test)) {
    accuracy<-accuracy+tb_data.test[i,i]
  }
  accuracy <- as.numeric(accuracy / samples.test)

  Pe <- 0
  for (i in 1:nrow(tb_data.test)) {
    Pe <- Pe + sum(tb_data.test[i,1:nrow(tb_data.test)])*sum(tb_data.test[1:
        nrow(tb_data.test),i])
  }
  Pe <- Pe /samples.test^2

  kappa <- (accuracy-Pe)/(1-Pe)
```

```
34
35    ci_acc <- 1.96 * sqrt(accuracy*(1-accuracy)/samples.test)
36    ci_kappa <- 1.96 * sqrt(accuracy*(1-accuracy)/(1-Pe)/(1-Pe)/samples.test)
37
38    res <- data.frame(
39      Model=model_name, lvs=lvs,
40      acc=accuracy, ci_acc=ci_acc,
41      kappa=kappa, ci_kappa=ci_kappa)
42    return(res)
43  }
```

For training classification models, *caret* was also used. As we can see below, in most cases we used different models (*method* parameter) than for regression, as there are models that are especially built for classification and orders for regression, such as logistic regression and linear regression. Nevertheless, others can develop both tasks with minimal changes, such as random forest ('*rf*'). Additionally, we can easily check how the concepts used on regression were also applied here: training control (*trControl*), grid search tune-fining (*tuneGrid*) and pre-processing task (*preProcess*).

```
1  set.seed(99)
2  regularization_model <- train(OSA ~Cervical + Weight + BMI + Gender + Height +
3                                  Snorer + Smoker + Age, data = df_OSA.train,
4                                method = "regLogistic",
5                                trControl=train.control)
6
7  knn_model <- train(OSA ~Cervical + Weight + BMI + Gender + Height +
8                       Snorer + Smoker + Age,
9                       data = df_OSA.train,
10                      method = "knn",
11                      tuneLength=30,
12                      preProcess = c("center","scale"))
13
14 rf_model <- train(OSA ~Cervical + Weight + BMI + Gender + Height +
15                     Snorer + Smoker + Age, data = df_OSA.train,
16                 method = "rf",
17                 trControl=train.control,
18                 tuneLength=10)
19
20 svm_model <- train(OSA ~Cervical + Weight + BMI + Gender + Height +
21                      Smoker + Age,
22                      data = df_OSA.train,
23                      method = "svmLinear",
24                      trControl=train.control,
25                      preProcess = c("center","scale"),
26                      tuneGrid = expand.grid(C = seq(0.1, 2, length = 20)))
```

Finally, we create a new data frame with the metrics obtained for each model trained in this stage, a total of 16 models. After, we must tidy the data frame in order to plot with *ggplot2*. As done for the classification problem. The two plots done in this section represent Model versus accuracy and Model versus kappa coefficients, considering confidence intervals computed before.

```
1
2  df_comparison <- data.frame(Model=c(), lvs=c(),
3                              acc=c(), ci_acc=c(),
4                              kappa=c(), ci_kappa=c())
5
6  df_comparison <- rbind(df_comparison,
7      get_metrics(regularization_model,
8                  df_OSA.train, df_OSA.test,
9                  "All", "Regularized LR"))
10
11 df_comparison <- df_comparison %>%
12   gather("metric", "value", acc, kappa) %>%
13   mutate(ci = ifelse(metric=="acc", ci_acc,ci_kappa)) %>%
14   select(Model, lvs, metric, value, ci)
15
16 p1 <- df_comparison %>%
17   filter(lvs=="All", metric=="acc") %>%
18   ggplot(aes(x=reorder(Model,value), y=value, color = metric)) +
19   geom_line() + geom_point() +
20   geom_errorbar(aes(ymin=value-ci, ymax=value+ci), width=.4) +
21   theme(axis.text.x = element_text(angle = 30, vjust = 0.5)) +
22   labs(x="Model", y="Accuracy")+
23   ylim(0,1) +
24   theme(legend.position = "none")
25
26 p2 <- df_comparison %>%
27   filter(lvs=="All", metric=="kappa") %>%
28   ggplot(aes(x=reorder(Model,value), y=value, color = metric)) +
29   geom_line() + geom_point() +
30   geom_errorbar(aes(ymin=ifelse(value-ci>0,value-ci,0), ymax=value+ci), width
         =.4) +
31   theme(axis.text.x = element_text(angle = 30, vjust = 0.5)) +
32   labs(x="Model", y="kappa")+
33   ylim(0,1) +
34   theme(legend.position = "none")
```

6 Conclusions

In this project, R language has been used in order to analyse, manipulate and visualize data. To this aim, *tydiverse* packages *dplyr* and *ggplot2* must be highlighted. *ggplot2* provides a large set of visualization options, such as histograms, bar plots, box plots, violin plots, etc... The knowledge on this package allows you to easily create figures that represent information effectively. *dplyr* provides a set of functions to manipulate the data. Functions such as filter, select or mutate, highly helps us to preprocess data before plotting it or saving it.

Additionally, a concept I learnt thanks to this tool is tidy data. In tidy data, all column headers are variables while each row is an observation. Below we show an example of tidying data: as we can see, MSE, MAE and R-Squared are values instead of variable names. The package that provides the functions (mainly *gather* and *spread*) to make these transformations is *tidyr* which also belongs to *tydiverse*. It is important to say that, python library, pandas has also equivalents to these functions, f.e. *pandas.melt* and *pandas.pivot*.

Model	MSE	MAE	R-Squared
Naive model	370	14.29	0
LM all	297	12.26	0.1980
LM subset	306	12.58	0.1726

(a) Messy data

Model	Metric	Value
Naive model	MSE	370
Naive model	MAE	14.29
Naive model	R-Squared	0
LM all	MSE	297
LM all	MAE	12.26
LM all	R-Squared	0.1980
LM subset	MSE	306
LM subset	MAE	12.58
LM subset	R-Squared	0.1726

(b) Tidy data

Figure 1: Messy versus tidy data example.

Finally, in the project, we have trained and evaluated a total of 32 machine learning models. In order to initialize and fit them with the data, *caret* package has been used. It provides a set of flexible and efficient functionalities, such as defining training control, preprocessing, hyperparameter tuning and a lot of different models. Nevertheless, the knowledge of the package and its possibilities is essential to use it, although, after it gets pretty fast-forward. For the evaluation, customized functions that computed selected metrics were developed. Therefore, knowledge about data-flow and basic functions in R has been learnt.

As an overall conclusion, we can say that we have learnt sufficient resources to develop an entire machine learning procedure in R. Nevertheless, more extensive or exhaustive experience could be achieved. Therefore, I expect to be able to develop future ML projects better and more effectively, both in R and in other languages.

References

[1] M. C. L. Eduardo López Gonzalo and L. H. Gómez, "Prdl & mllb activities," p. 40, 2021.

[2] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow.* O'Reilly Media, Inc, 2017.