MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

MACHINE LEARNING LAB

HALF-TERM ACTIVITIES REPORT:

Methodology and Tools for Developing Machine Learning Models

Case Study: Obstructive Sleep Apnea

Jaime Pérez Sánchez

Course 2019/20

Table of Contents

1. Methodology

1.1. Guidelines for Machine Learning Models Development

The very first step in any project is to define the problem we want to solve. However, this report focuses on the development of the Machine Learning models to address these problems. Therefore, we will assume that the problem has already been defined in the report of the subject "Predictive and Descriptive Learning". The main structure of every Machine Learning project can be synthesized the following 6 steps, summarized in Figure 1.

1. **Data Acquisition:** It is the process of collecting the data that will be used to train the algorithms. This is a critical step that will determine how good our model will work, the more and better data we get, the better our model will perform.
2. **Data Cleaning, processing and Exploratory Data Analysis:** Before training the Machine Learning models we must ensure that the data is correctly transformed and cleaned so that it can be interpreted by the models. An Exploratory Data Analysis (EDA) is also important in order to extract the main characteristics and correlations of the dataset (typically with visualizations), to be able to formulate preliminary hypotheses that could lead to new data collections and experiments.
3. **Choosing Evaluation Metrics and Protocols:** In order to select and improve our model, we must establish specific metrics that capture the objective of the problem we want to solve. In addition, we must define the protocols to accomplish this evaluation of the model (typically splitting the dataset into train and test sets, and Cross-Validation techniques).
4. **Model Selection and Training:** In this step we choose Machine Learning models suitable for our kind of problem, and train them with the data obtained from the previous phases. There is a great variety of algorithms for very different problems.
5. **Model Testing:** Once the model has been trained, we check its performance with the metrics and protocols previously chosen. In this step we compare the performance of the different chosen algorithms.
6. **Hyperparameter tuning and Model Deployment:** After we have tested which algorithms solve the problems most satisfactorily, there are many hyperparameters (of both the model and the preprocessing of the data) that we can adjust to obtain more precise results. After this step and if the results are satisfactory, we will be able to deploy the model.

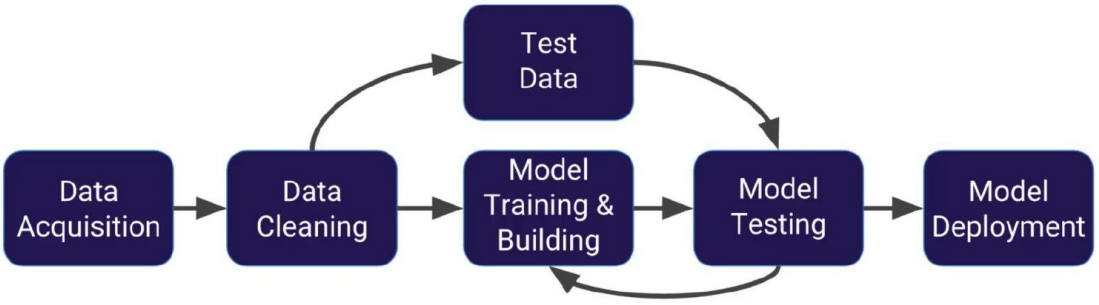

Figure 1. Overview of a Machine Learning Project Structure

1.2. Implementation Schedule

Our objective is the development of an Obstructive Sleep Apnea (OSA) predictive model. We will follow a methodology based on the one explained in the previous section. The first dataset used has been provided in the Moodle website of the subject and contains basic clinical information such as: age, weight, height, cervical perimeter... Also, the target variable that we have to predict: Apnea-Hypopnea Index (AHI); and other data that may be risk factors such as whether the patient is a smoker, other diseases, etc…

First, the data will be cleaned in order to deal with missing information, its transformation in so that it can be introduced into the models, and to decide which variables will be used to make the predictions. For a better generalization and confidence in the obtained results, the cross-validation method **K-Fold** will be used. Through this method, we will divide the dataset into **5** random sets of the same size (20%), and evaluated one by one taking the rest as training data in each case. Therefore, the evaluation metrics will be conducted with the predictions of the entire dataset. Figure 2 shows a schematic illustration of this.

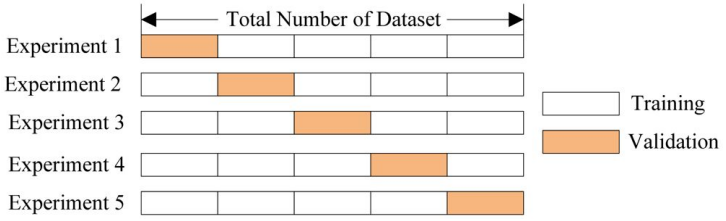

Figure 2. Cross-validation K-Fold diagram

The following evaluation metrics will be used for the **regression** models:

- **Coefficient of determination (R^2):** The proportion of the variance in the dependent variable that is predictable from the independent variables.

$$R^2 = \frac{\sum_{j=1}^{n} \left(y_j - x_j \right)^2}{\sum_{j=1}^{n} \left(y_j - \bar{y}_j \right)^2}$$

- **Maximum Absolute Error (MaxAE):** The maximum error, in absolute value, between predicted and actual values.

$$MáxAE = máx\{ \left| y_j - x_j \right|, \forall j \in \left[1, n \right] \}$$

- **Mean Absolute Error ± Standard Deviation (MAE ± STD):** MAE is the mean error, in absolute value, between predicted and actual values. And STD is the amount of variation or dispersion of a set of values, in this case, the errors between predicted and actual values.

$$MAE = \frac{1}{n} \sum_{j=1}^{n} \left| y_j - x_j \right| \qquad STD = \sqrt{\frac{1}{n} \sum_{j=1}^{n} \left(y_j - \mu \right)^2}$$

- **Root Mean Square Error (RMSE):** The root of the squared mean of the errors between predicted and actual values.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(y_j - x_j\right)^2}$$

The following evaluation metrics will be used for the **classification** models:

- **Precision and Recall:** Precision is the fraction of relevant instances among the retrieved instances, while recall is the fraction of the total amount of relevant instances that were actually retrieved.

$$Precision = \frac{n^{\circ}\ True\ Positives}{n^{\circ}\ True\ Positives\ +\ n^{\circ}\ False\ Positives}$$

$$Recall = \frac{n^{\circ}\ True\ Positives}{n^{\circ}\ True\ Positives\ +\ n^{\circ}\ False\ Negatives}$$

- **F1 score:** The single weighted value of accuracy and recall.

$$F_1 = 2 \times \frac{Precision\ \times\ Recall}{Precision\ +\ Recall}$$

- **Balanced Accuracy:** The weighted mean of the recall obtained in each class.

$$Balanced\ Accuracy = \frac{\sum_{j=0}^{N} w_{(class\ j)} \times recall_{(class\ j)}}{N_{classes}}$$

- **Confusion Matrix Plot:** The root of the squared mean of the errors between predicted and actual values.

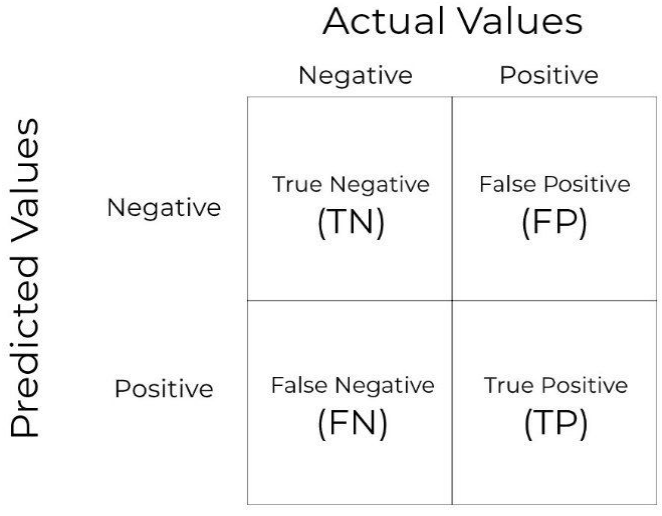

Figure 2. Correlation matrix diagram

- **ROC AUC Curve:** ROC (Receiver Operating Characteristic) illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. AUC (Area Under the Curve) is the metric computed for the ROC curve plot. The Y axis is the True Positives ratio, and the X axis is the False Positives ratio. An illustrative example is shown below.

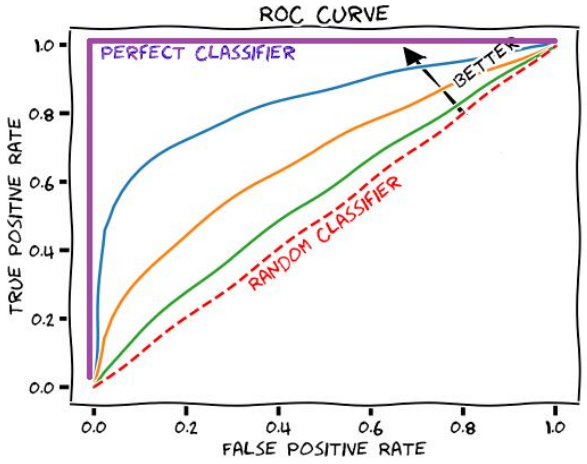

Figure 3. ROC curve diagram

In a preliminary analysis it appears that we have insufficient data to make a reliable predictive model. Although, if this data is sufficiently representative we could draw interesting conclusions about the risk factors of the OSA, which could help doctors to diagnose the disease more accurately and earlier.

1.3. Experimental Setup

This project has been conducted on the student's personal computer and in an Anaconda's virtual environment [1]. The main specifications of the PC are:

- **Host:** Lenovo Ideapad 700-15ISK
- **OS:** Arch Linux x86_64
- **CPU:** Intel i5-6300HQ (4 cores) @3.2 GHz
- **RAM:** 16 GB

It has been decided that this project would be developed in **Python** (version 3.7) programming language. The main reasons for this choice are: previous experience with the language from the developer, simpler syntax, large number of libraries for Machine Learning development (with smoothly difficult curves) and simplicity of integration in case of model deployment. In the following we will briefly describe the Python libraries used during the project.

- **NumPy:** [2] Fundamental package for scientific computing, including powerful N-Dimensional array objects and functions (essential for Machine Learning).

- **Pandas:** [3] Data structures management and data analysis library. Provides a high-performance and easy-to-use framework to manage and manipulate datasets.

- **Pandas Profiling:** [4] Package in a layer over *Pandas*, that generates profile reports from a DataFrame for a quick data analysis.
- **Scikit-learn:** [5] Fundamental Machine Learning library that includes a large number of algorithms (classification, regression, clustering...) together with preprocessing and evaluation functions.

- **XGBoost and CatBoost:** [6] [7] Optimized distributed gradient boosting libraries designed to be highly efficient, flexible and portable. Provides a parallel tree boosting that solve many data scientist problems in a fast and accurate way.
- **Matplotlib:** [8] 2D plotting library which produces publications quality figures in a variety of hard copy formats and interactive environments across platforms.

- **Seaborn:** [9] Data visualization library over Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics.

All the developed code is available in the student's GitHub profile [10] [11] and in Google Collaboratory [12] [13].

2. Data Wrangling

2.1. Data Description

The analysis of the data has been carried out with the "pandas-profiling" library. When executing the following lines of code, it generates a report, from a *Pandas* DataFrame, in a ".html" file with a basic analysis of all variables and their correlations.

```
import pandas
import pandas_profiling
report = df_tmp.profile_report(title='OSA_report')
report.to_file(output_file="OSA_report.html")
```

2.2. Data Preparation

The preparation and transformation of the data so that it can be fed to the predictive models, has been done with the *Pandas* and *NumPy* libraries.

```
import pandas as pd
import numpy as np
```

2.2.1. Missing Values

The main actions carried out in this section have been: replacing the "ns" values with NaNs, replacing the -1 values with NaNs, and removing all rows containing NaN values.

```
df_tmp1 = df_tmp1.replace('ns', np.nan)
df_tmp1 = df_tmp1.replace(-1, np.nan)
df_final = df_tmp1.dropna()
```

2.2.2. Label Encoding 'Smoker'

The main actions carried out in this section have been: replacing the "ns" values with NaNs, replacing "si (poco)" values for "poco", setting the variable 'Smoker' as 'category' and label encoding this category.

```
df_tmp1 = df_tmp1.replace('ns', np.nan)
df_tmp1 = df_tmp1.replace('si (poco)', 'poco')
df_tmp1['Smoker'] = df_tmp1['Smoker'].astype('category')
df_tmp1['Smoker'] = df_tmp1['Smoker'].cat.codes
```

2.2.3. One Hot Encoding 'Gender'

The main actions carried out in this section have been: adding the two new columns with One Hot Encoding of 'Gender' and dropping the variable 'Gender'.

```
df_tmp1 = pd.concat([df_tmp1,pd.get_dummies(df_tmp1['Gender'])],axis=1)
df_tmp1 = df_tmp1.drop(['Gender'],axis=1)
```

2.2.4. Setting ‘Patient’ to Index

The actions carried out in this section have been: replacing the repeated label (P0363) and setting the ‘Patient’ column as index..

```
# Show which rows contain the repeated label
df_tmp1[df_tmp1['Patient']=='P0363']
# Replace the label on one of the rows
df_tmp1.iloc[362] = df_tmp1.iloc[362].replace('P0363','P9999')
# Set as index
df_tmp1.set_index('Patient', inplace=True)
```

2.2.5. Computing ‘BMI’

The BMI has been computed from the Weight and Height data.

```
df_final['BMI'] = df_final['Weight']/((df_final['Height']/100)**2)
```

2.2.6. Computing ‘log(AHI+1)’

As explained in the report of ‘Predictive and Descriptive Learning’, the justification for adding 1 to the AHI before making the logarithm is to avoid that the 0 values are transformed into ‘-Infinity’. Finally, this column will not be used in the models because when analyzing the results, it did not represent an appreciable improvement.

```
# Save AHI column to test accuracy of the predictions
np_AHI = df_final['AHI'].to_numpy()
# Compute log(AHI)
df_final['log_AHI'] = df_final['AHI'].apply(lambda row:np.log10(row+1))
# Drop 'AHI' column
df_final = df_final.drop(['AHI'],axis=1)
```

2.2.7. Computing ‘OSA’ for Classification Models

As explained in the report of ‘Predictive and Descriptive Learning’, for the classification models we intend to predict if the male patients are healthy (i.e. AHI ≤ 10 encoding as a 0) or have severe OSA (i.e. AHI ≥ 30 encoding as a 1). The patients between these values are set as NaNs for further elimination.

```
df_tmp1['OSA'] = np.where(df_tmp1['AHI'] <= 10, 0,
                 np.where(df_tmp1['AHI'] >= 30, 1,
                 np.nan))
```

3. Exploratory Data Analysis (EDA)

The EDA has been carried out mainly with the *Seaborn* and *Matplotlib* libraries, performing the operations on DataFrames of the *Pandas* library.

```
import seaborn as sns
import matplotlib.pyplot as plt
```

- **Plotting Missing Values:** In order to visualize the missing values and help us to understand the dataset, a Heatmap has been used with a DataFrame that only indicates if the values it contains is NaN or not. An example of the result can be found in Figure 4.

```
sns.heatmap(df_tmp1.isnull(),cmap='Blues',cbar=False)
```

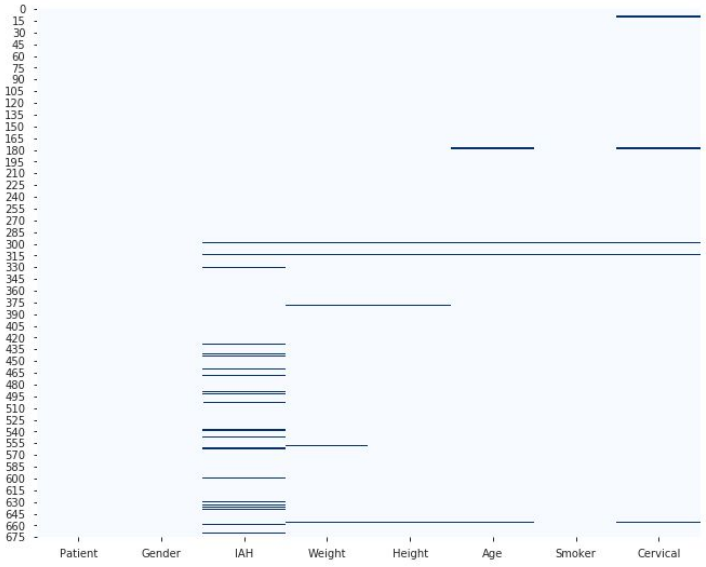

Figure 4. Plotting missing values with Heatmap

- **Plotting Correlation Matrix:** In this step, we must first calculate the correlation matrix and then visualize it with a Heatmap. In Figure 5 we can find an example of the result.

```
# Compute the correlation matrix
corr_matrix = df_final.corr()
# Plot
sns.heatmap(corr,center=0,square=True,linewidths=.5,annot=True)
```

- **Plotting Linear Regression, Histograms and KDE:** To visualize the relationship and statistical distributions of two variables we can use the *jointplot* method of the *Seaborn* library. An example of the result is shown in Figure 6.

```
sns.jointplot('Cervical','AHI',kind='reg',data=OSA_df)
```

Figure 5. Correlation matrix Heatmap

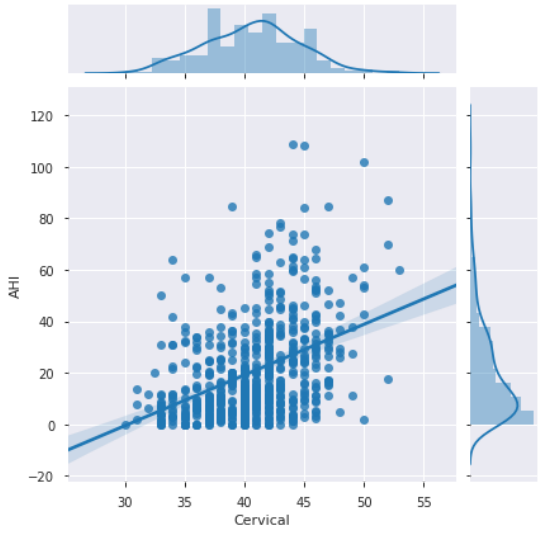

Figure 6. Linear regression, histograms and KDE between two variables with *jointplot*

- **Plotting Categorical Variables:** A very visual approach for analyzing categorical variables is the *violinplot* method of the *Seaborn* library. In Figure 7 we can see an example of the result, where the lines inside the violins indicate the quartiles of the statistical distribution. In this case we are also analyzing the categorical variable 'Gender' that is displayed on the sides of the violin.

```
sns.violinplot(x='Smoker',y='AHI',hue='Gender',
               split=True,inner='quart',palette='Set2')
```

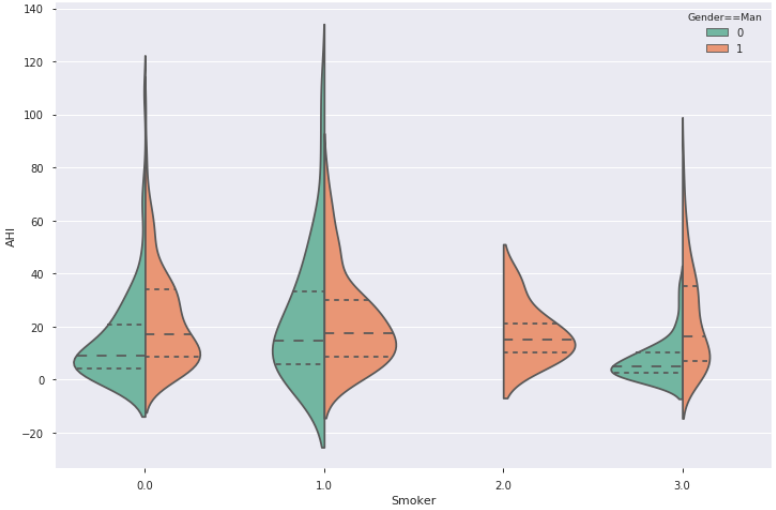

Figure 7. Plotting categorical variables with *violinplot*

- **Matrix of Scatter Plots:** To visualize all the relationships between several variables you can use a scatter plots matrix with the *pairplot* method of the *Seaborn* library. In this case we can also visualize the distributions according to some categorical variable (e.g. 'Gender'). In Figure 8 we can see an example of the result.

```
sns.pairplot(OSA_df,hue='Gender')
```

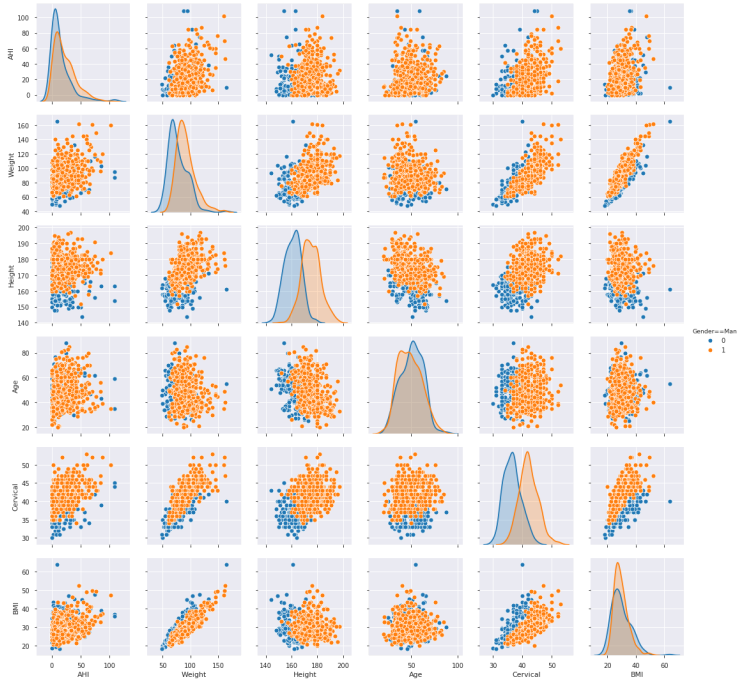

Figure 8. Matrix of scatter plots with *pairplot*

- **Bar Plots:** To represent an estimation of central tendency and confidence intervals for a numeric variable as a function of another one, we can use the *barplot* method of the *Seaborn* library. In Figure 9 we can observe an example of the result.

```
sns.barplot(x='OSA', y='BMI', data=OSA_df)
```

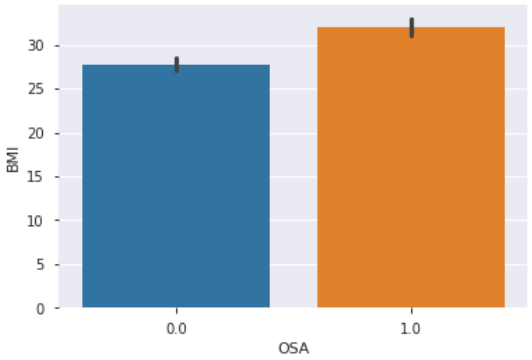

Figure 9. Bar plot example

- **Count Plots:** If we wish to perform a similar analysis to that obtained with Bar Plots, but with a single variable and taking into account the frequency of appearance in the data, we can use the countplot method of the Seaborn library. In this case we have also added a categorical variable ('OSA') to the analysis, so that we can study its distribution. In Figure 10 we can observe an example of the result.

```
sns.countplot(x='Smoker', data=OSA_df, hue='OSA')
```

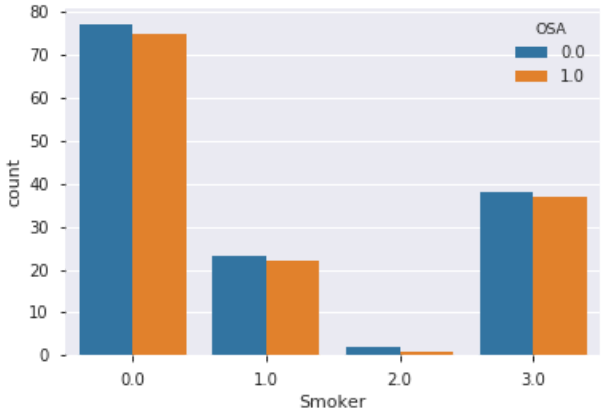

Figure 10. Count plot example

4. Machine Learning Models

The methodology followed in the implementation of Machine Learning models has been to try to automate the process as much as possible. The main library used is *Scikit-Learn*, which provides both the implemented models and the preprocessing methods. In the following we will study the implementation of data preprocessing methods and the families of the implemented algorithms, both for the problem of regression and classification.

4.1. Data Preprocessing

In this project three preprocessing methods have been tested, in addition to the raw data, to introduce into the models. For each method, two scalers are created to facilitate data de-transformation.

- **Raw Data:** In order to facilitate the obtaining of results and error metrics, the dataset is divided into *features* and *target* columns.

```
# Get all the columns from the dataframe.
columns = OSA_df.columns.tolist()
# Filter the columns to remove ones we don't want.
features = [c for c in columns if c not in ['AHI']]
# Store the variable we'll be predicting on.
target = 'AHI'
# Split the dataset between features and target
X = OSA_df.loc[:, features]
y = OSA_df.loc[:, target]
# For further transformations we create a reshaped y column
y_reshaped = np.reshape(y.values, (y.shape[0], 1))
```

- **Polynomial Features:** Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to an specified degree.

```
from sklearn.preprocessing import PolynomialFeatures
# Create Polynomial transformer with 2 degrees
poly = PolynomialFeatures(2)
# Apply transformation
X_poly = poly.fit_transform(X)
```

- **Standard Scaler:** Standardize features by subtracting the mean and scaling to unit variance, of each variable.

```
from sklearn.preprocessing import StandardScaler
# Create Scalers
Scaler_std_X = StandardScaler()
Scaler_std_y = StandardScaler()
# Apply transformation
X_std = Scaler_std_X.fit_transform(X)
y_std = Scaler_std_y.fit_transform(y)
```

- **MinMax Scaler:** Transforms features by scaling each variable to a given range.

```
from sklearn.preprocessing import MinMaxScaler
# Create Scalers
Scaler_minmax_X = MinMaxScaler()
Scaler_minmax_y = MinMaxScaler()
# Apply transformation
X_minmax = Scaler_minmax_X.fit_transform(X)
y_minmax = Scaler_minmax_y.fit_transform(y)
```

4.2. Regression

As explained before, the objective in the implementation of the models has been to automate it as much as possible. For this purpose, a dictionary has been created with most of the regression models implemented in *Scikit-Learn* library. In addition to a for loop where the models are executed with K-Fold validation and results metrics are obtained. The selected hyperparameters for each model have been omitted to simplify the explanation of the code.

```
from sklearn import linear_model, kernel_ridge, svm, neighbors,
                    gaussian_process, tree, ensemble, neural_network
import xgboost, catboost
model_list = {"Linear_Regression": linear_model.LinearRegression(),
       "Ridge_Regression": linear_model.Ridge(),
       "Lasso": linear_model.Lasso(),
       "ElasticNet": linear_model.ElasticNet(),
       "LARS": linear_model.Lars(),
       "Lasso_LARS": linear_model.LassoLars(),
       "OMP": linear_model.OrthogonalMatchingPursuit(),
       "Bayesian_Ridge": linear_model.ARDRegression(),
       "Bayesian_ARD": linear_model.ARDRegression(),
       "Passive_Aggressive":linear_model.PassiveAggressiveRegressor(),
       "RANSAC": linear_model.RANSACRegressor(),
       "Theil_Sen_Regressor": linear_model.TheilSenRegressor(),
       "Huber_Regressor": linear_model.HuberRegressor(),
       "Kernel_Rigde": kernel_ridge.KernelRidge(),
       "SVM_Linear": svm.LinearSVR(),
       "SVM_C-support": svm.SVR(),
       "SVM_Nu-support": svm.NuSVR(),
       "SGD_Regression": linear_model.SGDRegressor(),
       "K-neighbors": neighbors.KNeighborsRegressor(),
       "K-neighbors_Radius": neighbors.RadiusNeighborsRegressor(),
       "Gaussian_Process":gaussian_process.GaussianProcessRegressor(),
       "PLS_Regressor": cross_decomposition.PLSRegression(),
       "Decision_Tree": tree.DecisionTreeRegressor(),
       "Gradient_Boosting": ensemble.GradientBoostingRegressor(),
       "Bagging_Regressor": ensemble.BaggingRegressor(),
       "Random_Forest": ensemble.RandomForestRegressor(),
       "Extra_Tree": tree.ExtraTreeRegressor(),
       "AdaBoost": ensemble.AdaBoostRegressor(),
       "MLP": neural_network.MLPRegressor(),
       "XGBoost": xgboost.XGBRegressor(),
       "CatBoost": catboost.CatBoostRegressor()}
```

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import r2_score, max_error, mean_absolute_error,
                            mean_squared_error
for i in model_list:
     # Obtain predictions with K-Fold (5) validation from each model
     y_pred = cross_val_predict(model_list[i], X, y, cv=5)
     # Obtain errors array
     errors = y - y_pred
     # Compute evaluation metrics on the results
     print('\033[1m' + str(i) + '\033[0m') #Print model name
     print('Variance score R²: %.3f' % r2_score(y, y_pred))
     print('Max Error: %.2f' % max_error(y, y_pred))
     print('Mean Absolute Error ± STD: %.2f ± %.2f' %
                    (mean_absolute_error(y, y_pred), np.std(errors)))
     print('Root Mean Squared Error: %.2f' %
                                np.sqrt(mean_squared_error(y, y_pred)))
     # Plot scatter with measured and predicted values
     fig, ax = plt.subplots()
     ax.scatter(y, y_pred)
     ax.plot([y.min(),y.max()],[y_pred.min(),y_pred.max()],'k--',lw=3)
     ax.set_xlabel('Measured')
     ax.set_ylabel('Predicted')
     plt.show()
```

4.3. Classification

In analogy to the regression case and in order to automate the process of execution and evaluation of the models, a dictionary has been created with most of the classification models implemented in the *Scikit-learn* library. In addition to a for loop where the models are executed and the results are obtained. The selected hyperparameters for each model have been omitted to simplify the explanation of the code.

```
from sklearn import linear_model, naive_bayes, svm, neural_networks
                    neighbors, gaussian_process, ensemble, tree
import xgboost, catboost
model_list = {
      "Logistic_Regression": linear_model.LogisticRegression(),
      "Ridge_Classifier": linear_model.RidgeClassifier(),
      "SGD_Classifier": linear_model.SGDClassifier(),
      "Perceptron": linear_model.Perceptron(),
      "Passive_Aggressive":linear_model.PassiveAggressiveClassifier(),
      "NaiveBayes_Bernoulli": naive_bayes.BernoulliNB(),
      "NaiveBayes_Multinomial": naive_bayes.MultinomialNB(),
      "SVM_Linear": svm.LinearSVC(),
      "SVM_C-support": svm.SVC(),
      "SVM_Nu-support": svm.NuSVC(),
      "K-neighbors": neighbors.KNeighborsClassifier(),
      "K-neighbors_Radius": neighbors.RadiusNeighborsClassifier(),
      "Neighbors_Nearest-Centroid": neighbors.NearestCentroid(),
      "Gaussian_Process":gaussian_process.GaussianProcessClassifier(),
      "AdaBoost": ensemble.AdaBoostClassifier(),
```

```
      "Bagging_Classifier": ensemble.BaggingClassifier(),
      "Ensemble_Extra_Trees": ensemble.ExtraTreesClassifier(),
      "Gradient_Boost": ensemble.GradientBoostingClassifier(),
      "Random_Forest": ensemble.RandomForestClassifier(),
      "Decision_Tree": tree.DecisionTreeClassifier(),
      "Extra_Tree": tree.ExtraTreeClassifier(),
      "MLP": neural_network.MLPClassifier(),
      "XGBoost": xgboost.XGBClassifier(),
      "CatBoost": catboost.CatBoostClassifier()
      }
```

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import classification_report, f1_score,
                            balanced_accuracy_score, confusion_matrix
for i in model_list:
     # Obtain predictions with K-Fold (5) validation from each model
     y_pred = cross_val_predict(model_list[i], X, y, cv=5)
     target_names = ['Healthy', 'Severe']
     # Compute evaluation metrics on the results
     print('\033[1m' + str(i) + '\033[0m')
     print('Classification Report')
     print(classification_report(y,y_pred, target_names=target_names))
     print('Balanced Accuracy: %.3f' % balanced_accuracy_score(y,
                                                              y_pred))
     print('f1 score: %.3f' % f1_score(y, y_pred, average='weighted'))
     # Plot the confusion matrix
     print('Confusion matrix')
     cm = confusion_matrix(y, y_pred)
     plt.legend(target_names)
     hm = sns.heatmap(cm, annot=True, cmap='plt.cm.Blues',
                    xticklabels=target_names, yticklabels=target_names)
     plt.show()
```

Once the results have been obtained, we have visualized the ROC curve (Receiver Operating Characteristic) for each K-Fold of the model that offers the best scores, using the code of this example from *Scikit-learn* [14]. Obtaining a figure like the one shown in the following. The analysis and visualization of all the results obtained (both regression and classification) can be found in this jupyter notebook [15] [16].

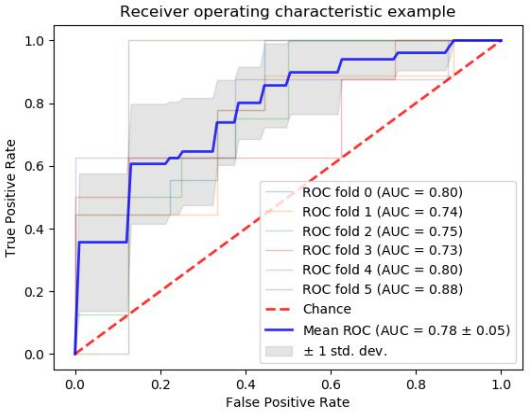

Figure 11. ROC curve with K-Fold example

5. Conclusions

In this project a comprehensive methodology has been developed to predict Obstructive Sleep Apnea (OSA) using Machine Learning models. This methodology, widely used and studied, is based on *state of the art* Machine Learning model development guidelines. It is described in detail in the first section of this report.

For the development of this project has been used Python programming language and fundamental libraries for Machine Learning problems and data processing (such as Pandas, Numpy, Scikit-Learn ...). All this provides a very powerful and user-friendly framework for model development. Jupyter Notebooks have been used in an Anaconda virtual environment to program all the code and display the results in the same file. The development of Feature Engineering methods using the patient voice frequency dataset is proposed as a future line of work.

My experience in this field due to my work as a researcher at the Polytechnic University of Madrid (Department of Electronic Engineering), has made that the development of this project has been carried out without outstanding issues. All the code of the project is available in my Github account [10] [11] and in Google Collab [12] [13]. All the results and visualizations can be found in this jupyter notebook [15] [16].

6. References

[1] «Anaconda Distribution», [Online]. Available: https://www.anaconda.com/distribution/ [Accessed 10 November 2019]

[2] «NumPy Library, [Online]. Available: https://numpy.org/ [Accessed 10 November 2019]

[3] «Pandas Library, [Online]. Available: https://pandas.pydata.org/ [Accessed 10 November 2019]

[4] «Pandas-Profiling Library, [Online]. Available: https://pandas-profiling.github.io/pandas-profiling/docs/ [Accessed 10 November 2019]

[5] «Scikit-Learn Library», [Online]. Available: https://scikit-learn.org/stable/ [Accessed 10 November 2019]

[6] «XGBoost Library», [Online]. Available: https://xgboost.readthedocs.io/ [Accessed 10 November 2019]

[7] «CatBoost Library», [Online]. Available: https://catboost.ai/ [Accesed 10 November 2019]

[8] «Matplotlib Library», [Online]. Available: https://matplotlib.org/ [Accessed 10 November 2019]

[9] «Seaborn Library», [Online]. Available: https://seaborn.pydata.org/ [Accessed 10 November 2019]

[10] Jupyter Notebook with regression approach code, [Online]. Available: https://github.com/jaimeperezsanchez/Machine_Learning_Lab/blob/master/CaseStudy_OSA/CODE_Python/Notebooks/1_Regression_OSA_Extra.ipynb [Accessed 10 November 2019]

[11] Jupyter Notebook with classification approach code, [Online]. Available: https://github.com/jaimeperezsanchez/Machine_Learning_Lab/blob/master/CaseStudy_OSA/CODE_Python/Notebooks/2_Classification_OSA.ipynb [Accessed 10 November 2019]

[12] Jupyter Notebook with regression approach code on Google Collab, [Online]. Available: https://drive.google.com/file/d/1_BaVajc6zaqngJLSN_vzwwn9lS5qS2NP/view?usp=sharing [Accessed 10 November 2019]

[13] Jupyter Notebook with classification approach code on Google Collab, [Online]. Available: https://drive.google.com/file/d/1LdWIC2MSEiHzccP1Tq0qRCYCJ8PeTt0P/view?usp=sharing [Accessed 10 November 2019]

[14] «Receiver Operating Characteristic (ROC) with cross validation», [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html#sphx-glr-auto-examples-model-selection-plot-roc-crossval-py [Accessed 10 November 2019]

[15] Jupyter Notebook with model results and analysis, [Online]. Available: https://github.com/jaimeperezsanchez/Machine_Learning_Lab/blob/master/CaseStudy_OSA/CODE_Python/Notebooks/Results.ipynb [Accessed 10 November 2019]

[16] Jupyter Notebook with model results and analysis on Google Collab, [Online]. Available: https://drive.google.com/file/d/1UvLXJFKYAQ1LJ038HtDrU7gbajTcDI3Z/view?usp=sharing [Accessed 10 November 2019]