UNIVERSITÀ DI PISA

Artificial Intelligence and Data Engineering

Data Mining and Machine Learning

User Guide

TEAM MEMBER:
Emanuele Tinghi

Contents

Chapter 1

Introduction

The purpose of this project is to build a machine learning model that can recognize the position on the chair (Left, Right, Center) of a given person.

The raw data belong to four testers:

- Leo
- Lorenzo
- Carlotta
- Irene

The data extracted from a tester and a position must be placed in *Data/Position/Tester_name*. However, for what regards the files needed to execute the code on chapters 4 and 5, the preprocessed data are already located inside the "Data_unificati" folder.

The chair used to obtain the data has two sensors, which returns two variables each. Sensor XX returns:

- ***SXX(DB)***: modulus of the signal coming from sensor XX
- ***SXX(DEG)***: phase of the signal from sensor XX

1.1 Used Approach

The approach used to address this problem is the following:

1. From each file (time instant) we extract the peak frequency and the corresponding value of modulus and phase for each sensor (vector of 6 numerical characteristics).
2. Each acquisition is divided into time windows of 5 seconds. This window is then analyzed, attempting to classify it consistently with the acquisition from which it was extracted.

3. The weight of the tester and some aggregate statistics of the 6 numerical characteristics are saved for each window (mean and variance).

4. The models are trained and their performances are evaluated in Leave-One-Subject-Out cross-validation.

Chapter 2

Exploration

The exploration phase of this project has been carried out in the notebook "Data_Exploration.ipynb" inside the 'Script' folder.

2.1 Utility Functions

The first section is composed by the definition of the functions used to plot

- ***plot_single_sample***: takes as input a dataframe containing the information of a single csv file and plots them
- ***plot_four_samples***: takes as inputs four dataframes and the corresponding labels (the tester from which the data are taken and his/her position on the chair) and plots them one against the other
- ***prepare_data_for_plot_imShow***: takes as input a path and a variable to be plotted and returns a dataframe that contains all the information needed to make a heat map. In this map on the x-axis we have the frequency, on the y-axis we have the time instance (to which the data belong), and the color represents the modulus of the signal at that specific point.
- ***heat_map_plot***: takes as inputs a path and a variable to display and shows the heat map of the testers, regarding those inputs

2.2 Single Sample

The second section of this notebook is used to see what the single sample looks like. Executing the code will result in the following graph being displayed:

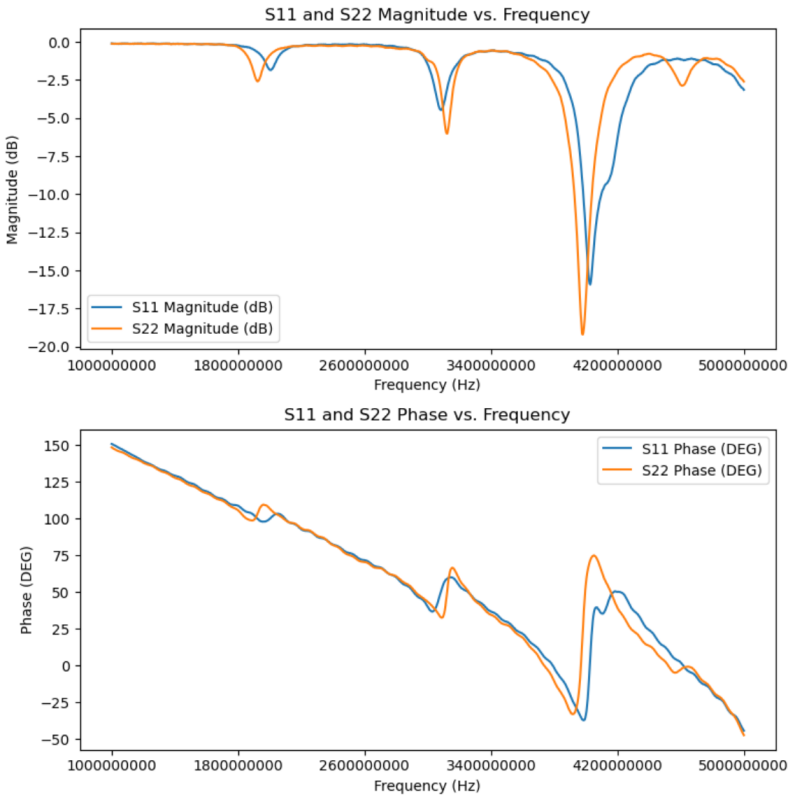

Figure 2.1: Right/Irene/Data (1).csv

2.3 Multiple Samples

The third section of the notebook is used to see more than one sample at a time. It will display a plot containing the graph of the four testers in a user-specified position:

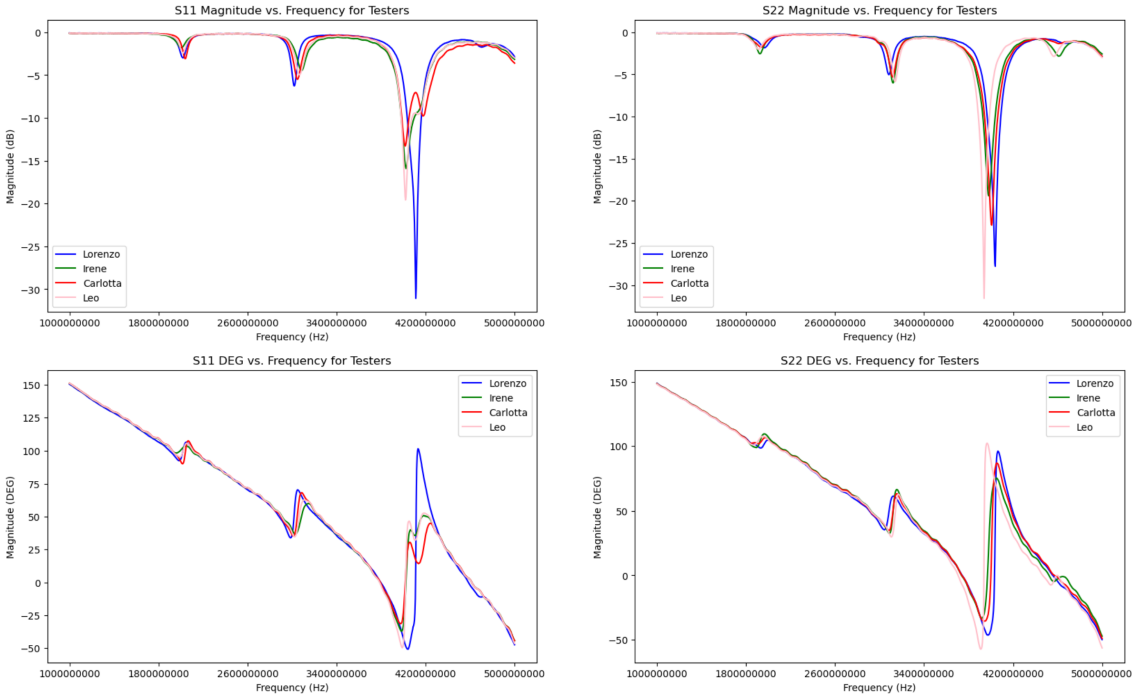

Figure 2.2: Samples from Right Position

Looking at this picture we can see that there are some differences between the four testers, even if three of them show similar plots

2.4 Heat Maps

The fourth section is for the creation of the heat maps. Those are useful tools to understand if, given a variable and a position on the chair, the trend is approximately the same for every person.

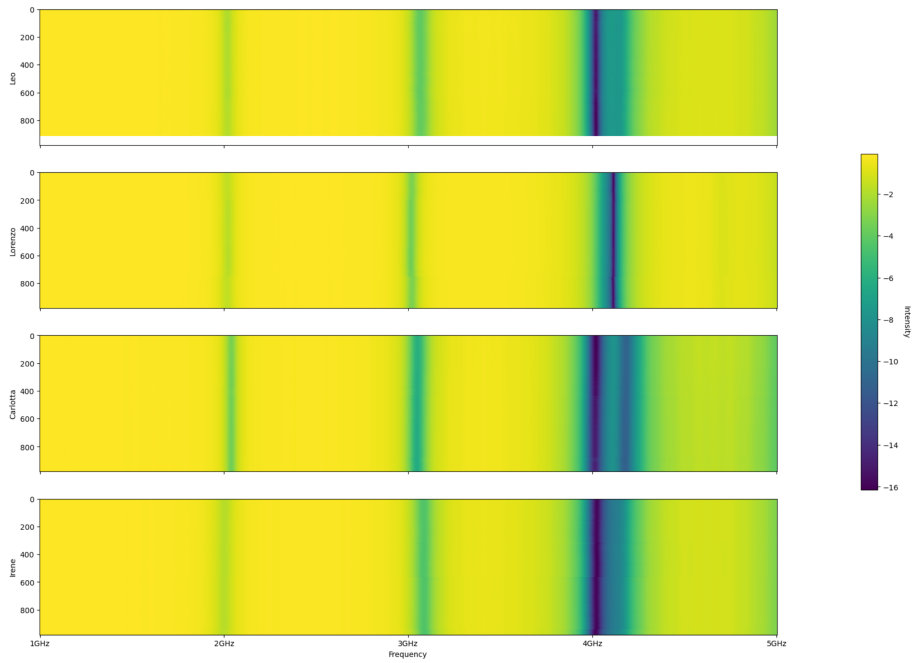

Figure 2.3: Heatmap for S11(DB) for right position

As it can be seen from this example the samples have some similarities, even if, also in this case, the samples of Lorenzo differ significantly from the others. The other heatmaps can be found in the "img" folder.

2.5 Smoothing

The last section of the notebook is used to see how the presence of smoothing modifies the signal. The test has been made with Simple Moving Average or Exponential Moving Average.

Chapter 3

Preprocessing

The preprocessing phase of this project has been carried out in the notebook "Data_Preprocessing.ipynb" inside the 'Script' folder

3.1 Utility Functions

The first section contains some function used in the notebook, that are:

- ***load_data***: takes as inputs a path and loads in a dataframe the tuples ('Freq(Hz)S11', 'S11(DB)', 'S11(DEG)') and ('Freq(Hz)S22', 'S22(DB)', 'S22(DEG)'), that are information regarding the point of maximum S11(DB)/ S22(DB) and the corresponding frequency and phase. If needed, smoothing can be disbaled by setting to False the variable smoothing_present (the default value is True).

- ***calculate_window_information***: takes as inputs a dataset, a window_size (chosen equal to 5 seconds), the step_size (each window differs from the others by 10 elements), the weight of the tester, the class to which the data belongs and the number of instances from the beginning and from the end to skip (Lorenzo has this value set to 500 to obtain approximately the same number of windows as the others). This function calculates the window information, such as the mean and the variance of the variables inside the window.

- ***scatter_plot***: takes as inputs a dataframe, two variables and a tester name and uses those variables as axis to plot the points in the dataframe (every class has its own color).

3.2 Preprocessing

In this part of the notebook the final datasets are calculated, one for each tester.

In detail, each tester loads the data of right/left/centered positions and calculates, for each one, a dataset composed by windows' information.

The three dataframes are then merged in another dataframe that is exported as "tester_name.csv".

3.3 Graphs

In this section there are code cells to make graphs such as box-plots and scatter-plots.

Chapter 4

Models

The classifiers are created in the notebook "Classification_Pipeline.ipynb", inside the 'Script' folder. It is assumed that in the dataset used for the model training/testing, the window size is 50, the step size is 10, the smoothing is present and the number of instances to skip is the default one (with the exception of Lorenzo that skips 500 instances)

4.1 Preparatory Code Cells

In the first cells the dataset are loaded and divided into a feature matrix (X, 868x13) and a target matrix (y, 868x1). Also the groups to which those windows belongs is extracted and is stored inside the matrix groups (868x1).

It is then controlled the order in which the testers are analyzed in Leave-One-Subject-Out

4.2 Models

The best models obtained are the ones based on KNN, Random Forest and AdaBoost. They are derived in the following way:

1. For what concerns scaling, RobustScaler was used, while for feature selection SelectKBest(k=2) was chosen. For the validation Leave-One-Subject-Out was implemented.

2. Each model is tested using different parameters combinations to see which one works better.

3. The obtained parameter are used and the model is derived. It is possible to see the performances of the model for every tester and the mean accuracy/f-score (both in case the smoothing was used and not used)

4.3 Other Tested Models with worse Performance

Other models such as SVC, GradientBoosting, and SGD (with and without RBF-Sampler) were tested. However, these models had worse performance than the others so they were not considered in the next steps.

Chapter 5

Performance Evaluation

This section can be found in the notebook "Performance_evaluation.ipynb", inside the 'SCript' folder. The values to be tested with Wilcoxon signed test are obtained concatenating the results of 200 iterations (for each model) of the test f-score. This is done in the first sections of the notebook.

5.1 Utility Functions

- ***tree_to_code***: takes as input a decision tree and the two best variables (according to selectKBest()) and translates the tree in a series of rules.
- ***scatter_plot***: takes as inputs a dataframe, two variables and the tester name and uses those variables as axis to plot the points in the dataframe (every class has its own color).

5.2 Perfomance evaluation

The f-score results of KNN, AdaBoost Classifier and Random Forest Classifier are compared using Wilcoxon Signed Test with alpha equal to 0.05. The results to compare are obtained repeating the validation using different random states (for a total of 800 summaries for each classifier).

The comparison between AdaBoost and Random Forest gives as result that AdaBoost is the model that performs better because the Wilcoxon test returns a p-value of approximately 0.0111 (the null hypothesis can be rejected), together with the fact that the AdaBoost mean value for the f-score is higher than the one from Random Forest.

The comparison between AdaBoost and KNN instead, given a p-value of approximately 0.243, doesn't allow to reject the null hypothesis.

5.3 Estimator Rules in AdaBoost Classifier

It is possible then to extract a tree from the classifier and to see it in the form of rules by using the function tree_to_code(). It is then possible to check why a certain tester performs well or not by checking those rules against the his/her scatterplot (plot with respect to the same features).