



EFC2 - Exercise 2

Rafael Claro Ito (R.A.: 118430)

October 2019

1 Source files

All code cited and all figures showed here can be found at the following GitHub repository:
<https://github.com/ito-rafael/IA006C-MachineLearning/tree/master/efc2>
In this repository, one can found the following files:

- Jupyter Notebook
 - efc2_pre-ex1.ipynb
 - efc2_ex1_binary_classification.ipynb
 - efc2_ex2_multiclass_classification.ipynb
 - efc2_ex2_knn.ipynb
- L^AT_EX
 - efc2.tex

The notebook “efc2_pre-ex1” plots the histograms for the exercise 1 and it is used for data visualization. It shows the input features histograms for the raw data and after a data standardization. Also, it shows the correlation between these data.

The notebook “efc2_ex1_binary_classification” effectively implements the logistic regression used to perform a binary classification proposed in exercise 1.

The notebooks “efc2_ex2_multiclass_classification” and “efc2_ex2_knn” implements the algorithms to perform a multiclass classification proposed in exercise 2. The former one uses the softmax approach while the latter one implements the K-Nearest Neighbors (kNN) algorithm.

2 Part 1 - Binary Classification

2.1 a) Input features characteristics analysis considering the histograms and correlation measures between them.

The first thing we did for this item, was to plot the histograms of the data before and after standardization. The histograms for the raw data can be seen in Figure 2 and the histograms for the standardized data can be seen in Figure 3. To perform the standardization, the StandardScaler class provided by the scikit-learn library was used.

Figure 4 shows the histograms of the input features separated according to the labels. The orange plot represents the label 0 (female class) and the blue plot represents the label 1 (male class).

As we can see in Figure 4, some features may be visually very informative about the class. For example, features “mean fun”, “Q25” and “IQR” can be very informative and show the classes in a distinct way. Features like “sd”, “sp ent”, “smf” and “mode” can bring some information about the class, but they are not so obvious, since both classes are showed in a specific value of the histograms. In some cases, it is not possible to separate the classes only by visually analyzing the histograms of the features in a straightforward way, like in the case of the features “Q75”, “maxfun” or “modindx”, since the data has values in a similar range for both classes.

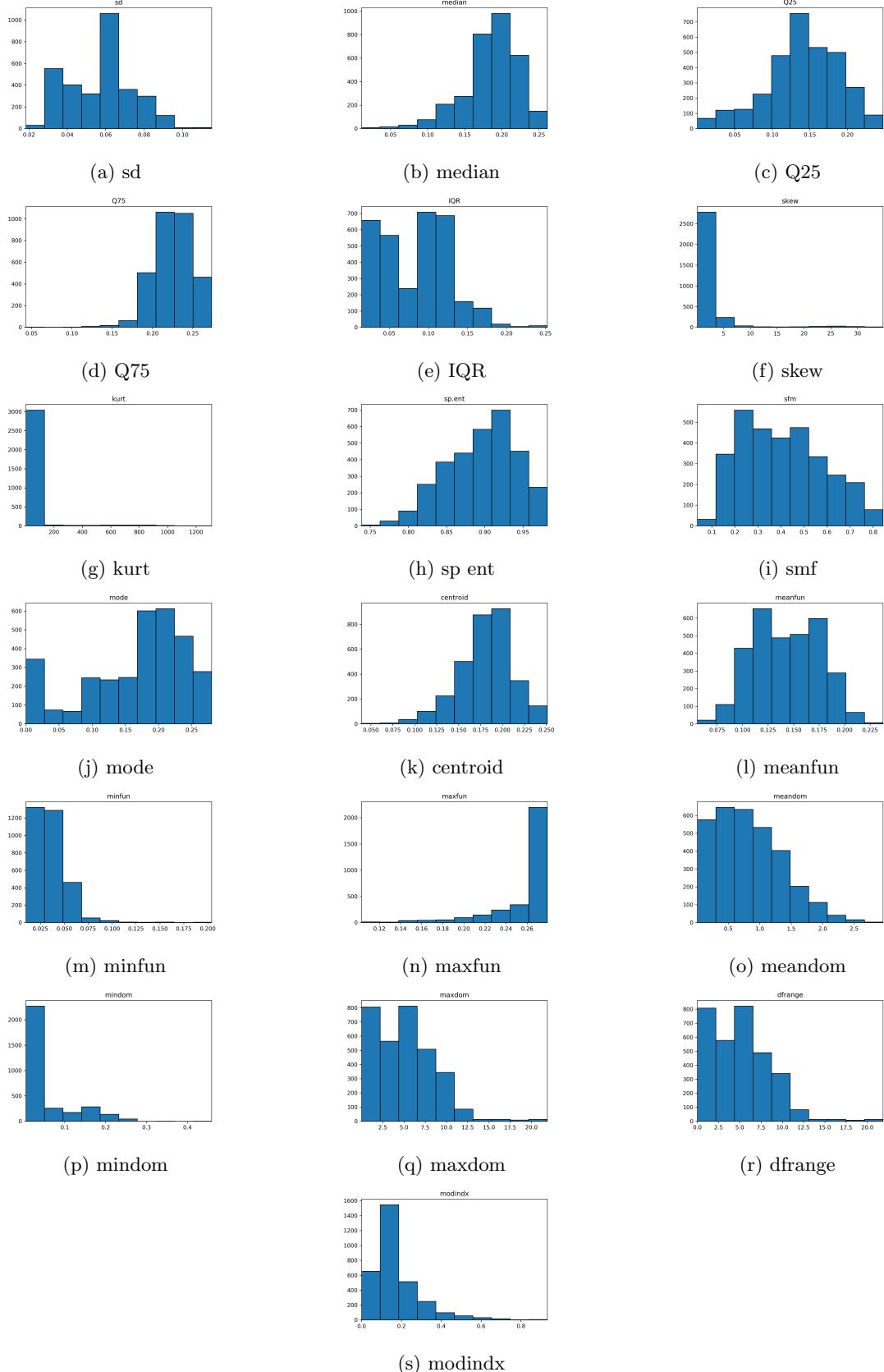


Figure 2: Histograms of raw input features

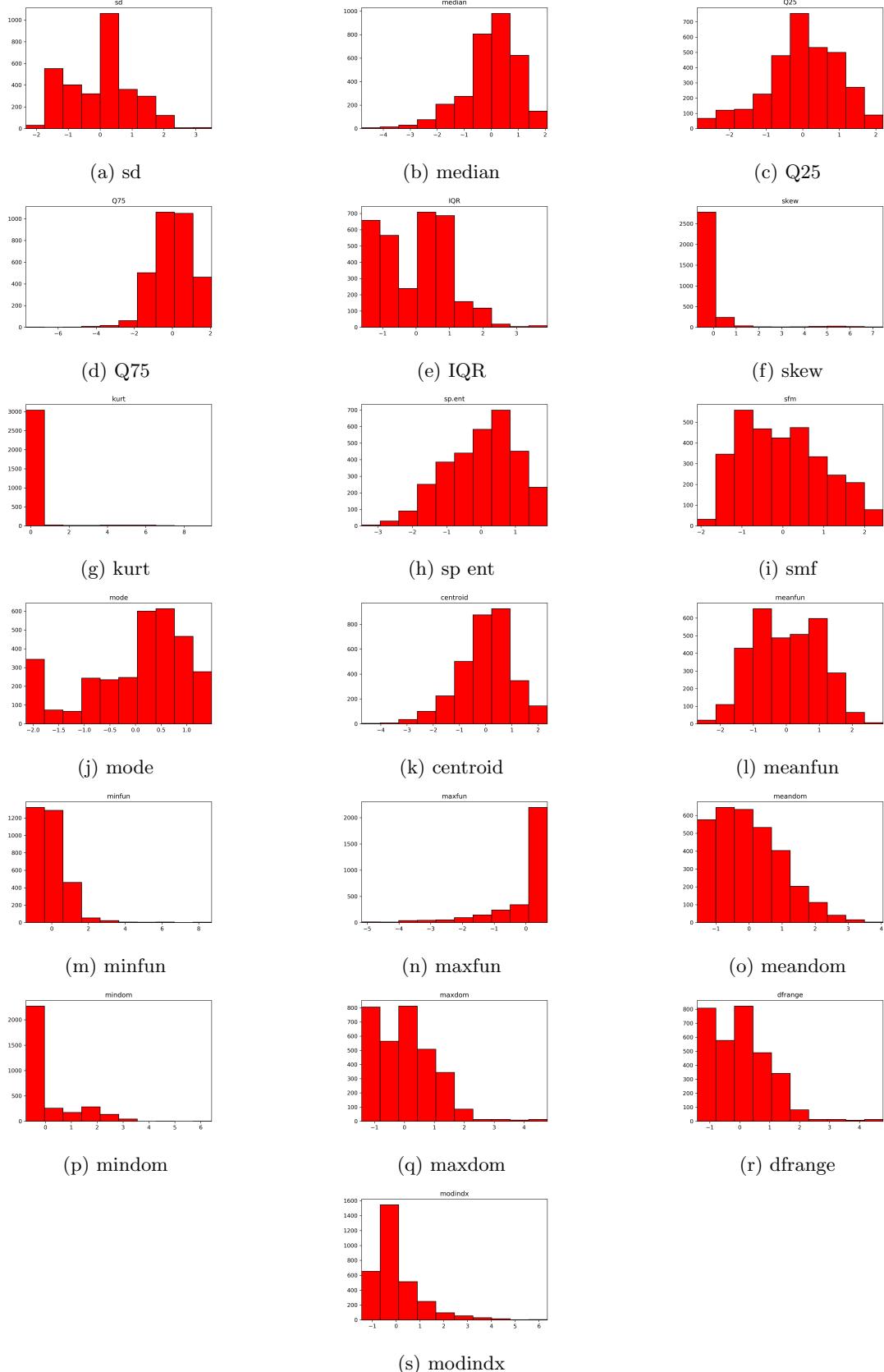


Figure 3: Histograms of input features after data standardization

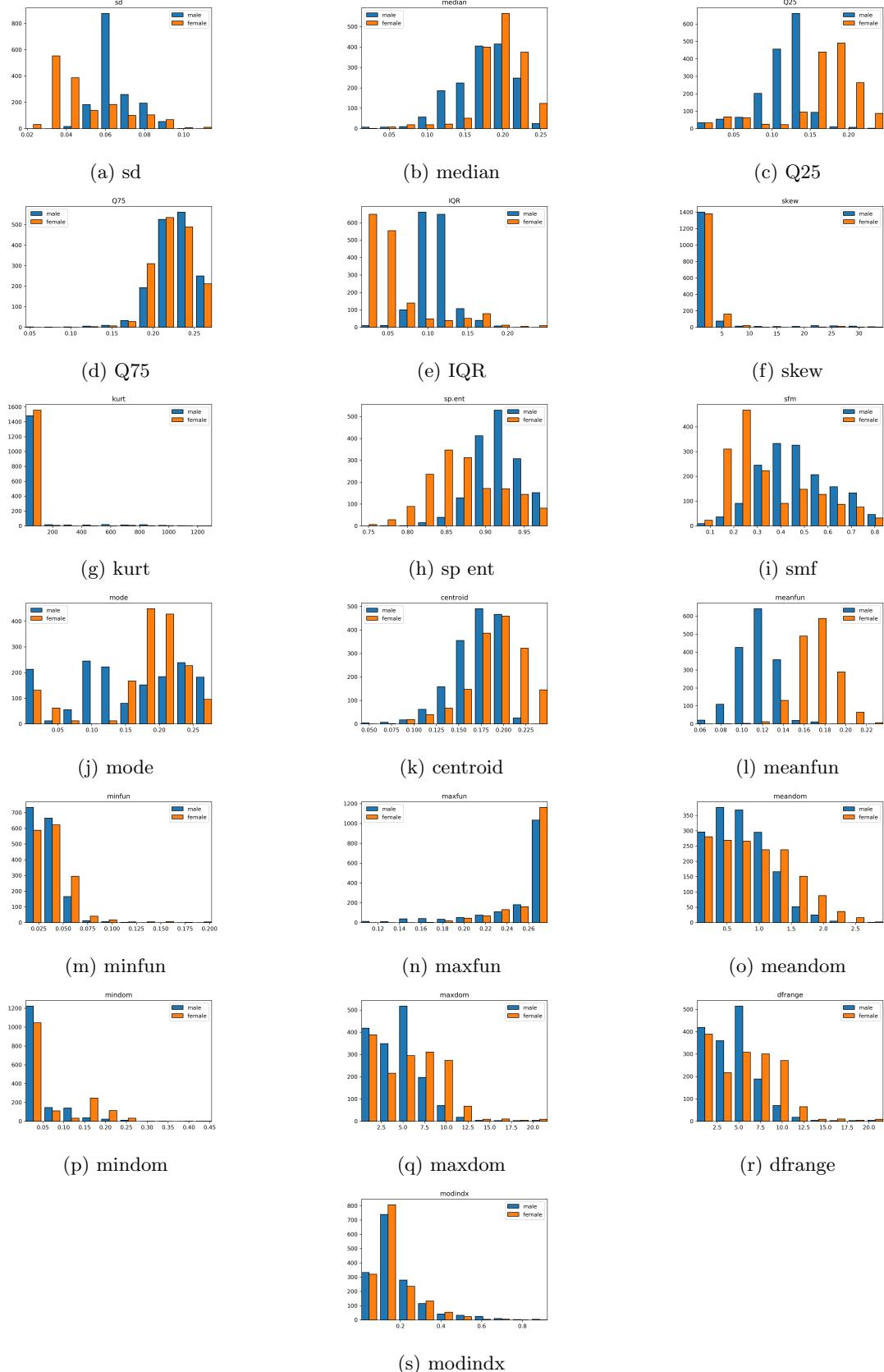


Figure 4: Histograms of input features after data standardization

In the Figure 5 we can see the correlation matrix, with values between 0 and 1, mapped in colors. We can see from this image, that there are cases where a feature is highly

correlated to another and also cases in which no significative correlation was measured. For example, features like “skew” and “kurt”, “maxdom” and “dfrange”, “centroid”, “Q25” and “median” have a high correlation. In fact, if we look into these two first pairs of features, we can see that the shape of the histograms are very similar.

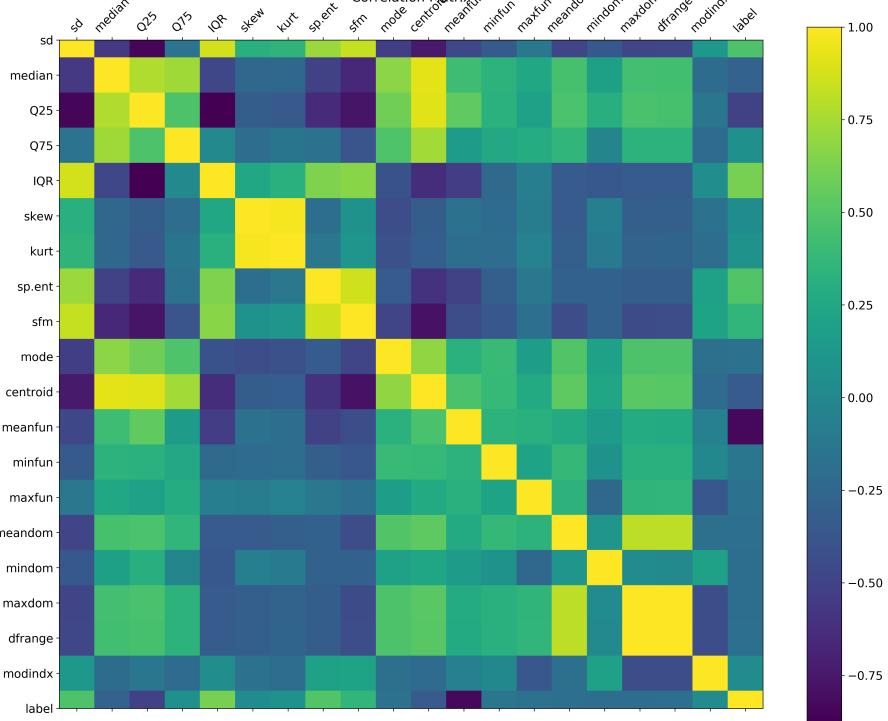


Figure 5: Correlation between input features

On the other hand, features like “sd” and “Q25”, “IQR” and Q25, “centroid” and “sfm”, presents a low correlation. An intuition about this can be inferred from the fact that for all these pair of features, the shape of the plots in the histograms are kind of mirrored. For example, for the “IQR” feature the orange data is plotted before the blue data, while in “Q25” the opposite happens. One thing to be noted is that the matrix presents a symmetry with respect to the diagonal of this matrix, indicating that the correlation of A with respect to B is the same of the correlation of B with respect to A.

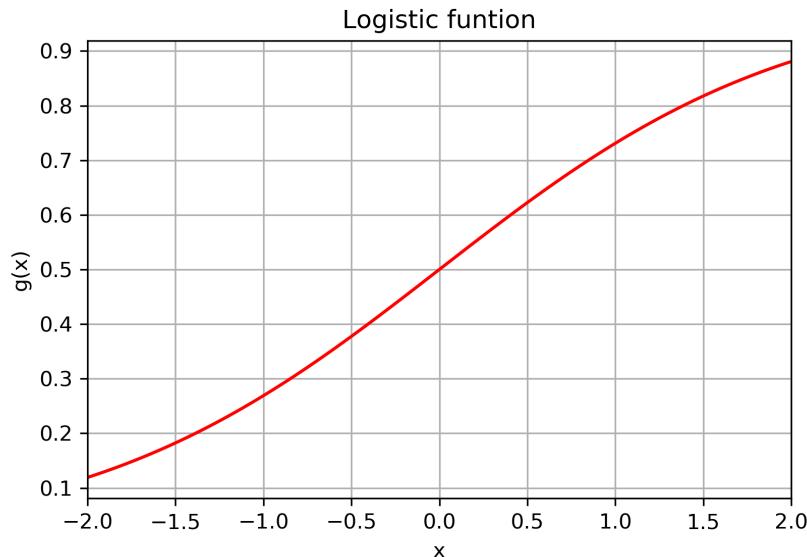


Figure 6: Logistic function

In the first notebook we also plotted the logistic function, used in the model proposed for binary classification. The Figure 6 shows the logistic function in an interval of -2 and 2.

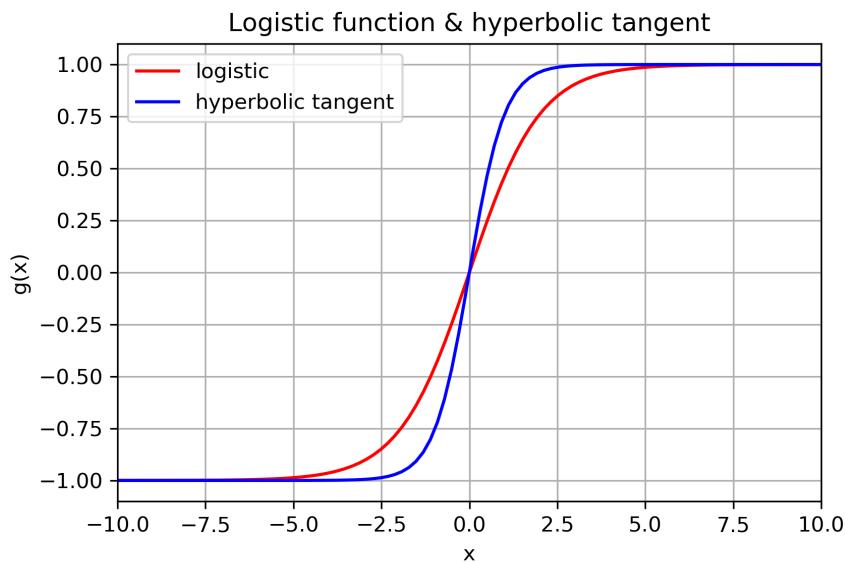


Figure 7: Comparasion between hyperbolic tangent and logistic function (scaled and with offset)

Before actually starting the binary classification proposed in exercise 1, one fact can be noted when comparing the logistic function (used in this exercise) and the non-linear function (hyperbolic tangent) used to transform the data in the exercise 2 of the previous list of exercises EFC1. The shape of these functions are very similar. Just to visualize the shape of the functions, the two functions are plotted in the same Figure 7. Here, the logistic function was multiplied by two, and subtracted by one in order to show the similar shape with the hyperbolic tangent (but with a different scale).

2.2 b) Logistic regression, ROC and F1-score curves.

In this subsection the binary classifier is actually coded. Here we are using the *shuffle* function provided by the scikit-learn library. First, we separate the data according to the label. We do this to ensure that we have the same rate of both classes in the training set and also in the test set. The data in both groups are then shuffled.

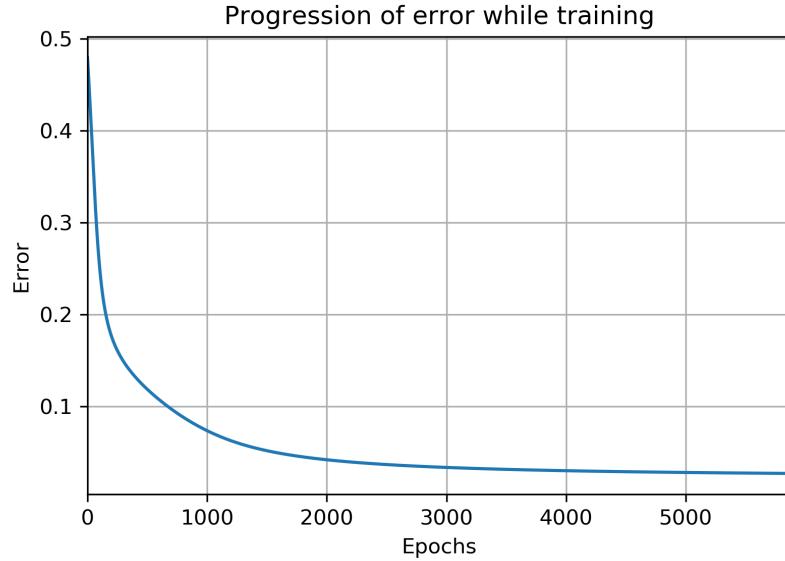


Figure 8: Error progression in training

The training step is performed until the difference between successive iteration errors is under a provided tolerance. RMSE was chosen to be the error criteria. We are using a tolerance of 1.10^{-6} , which means that when the error between the real and the predicted class for a iteration is less than the error in the previous iteration minus this tolerance, we should stop the training process. The error progression with the number of epochs is shown in Figure 8. We are considering the whole batch of samples for each step during the training.

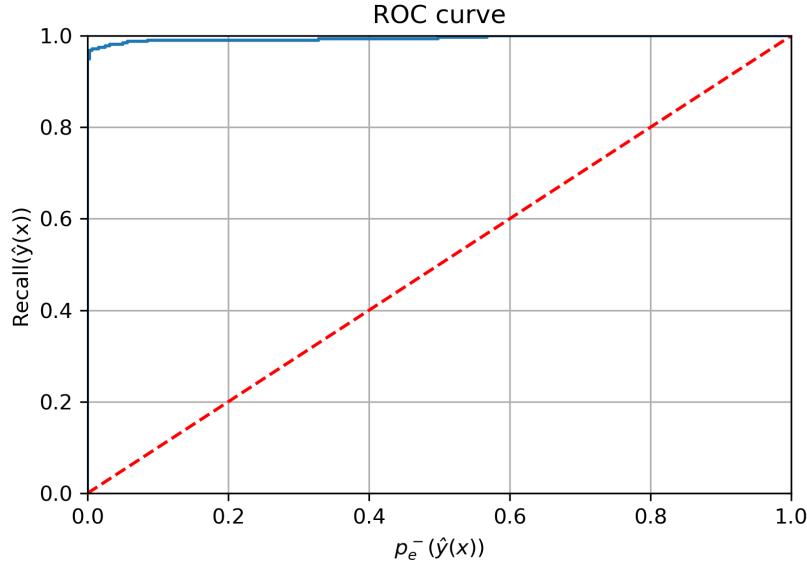


Figure 9: ROC curve

Now we are going to plot the receiver operating characteristic (ROC) curve and the F1-score for the classifier. For this, we are using the metrics provided by the scikit-learn library. The ROC curve is plotted with the recall (true positive rate, tpr) being the y axis and the false positive rate (fpr) being the x axis. The ROC curve can be seen in the Figure 9. We can see that the ROC curve is located at a high part of the graph as well it is to the left, indicating a good performance.

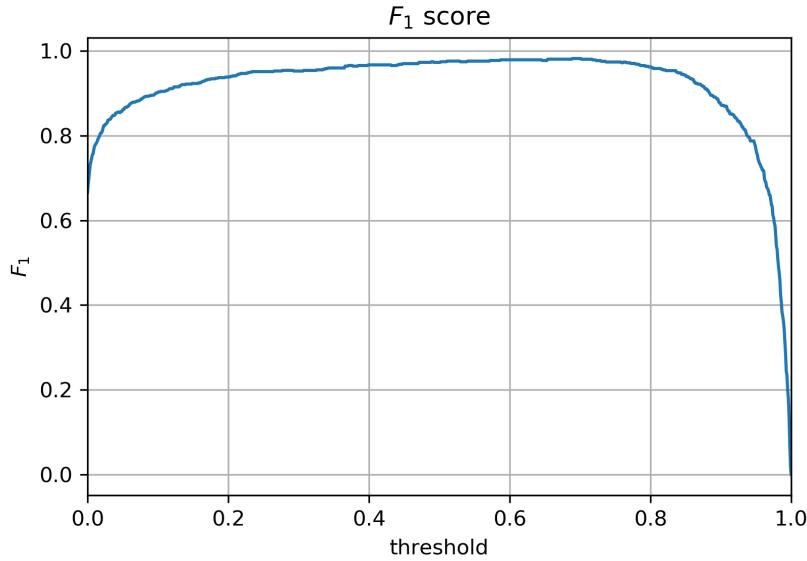


Figure 10: F1-score curve

We now change the threshold used to classify a sample with the positive or negative class. We used values of threshold from 0 to 1 with step of 0.001, i.e., we are using 1000 different thresholds values. For each value of threshold we calculated the F1-score. Figure 10 shows this plot.

2.3 c) Threshold, confusion matrix and accuracy.

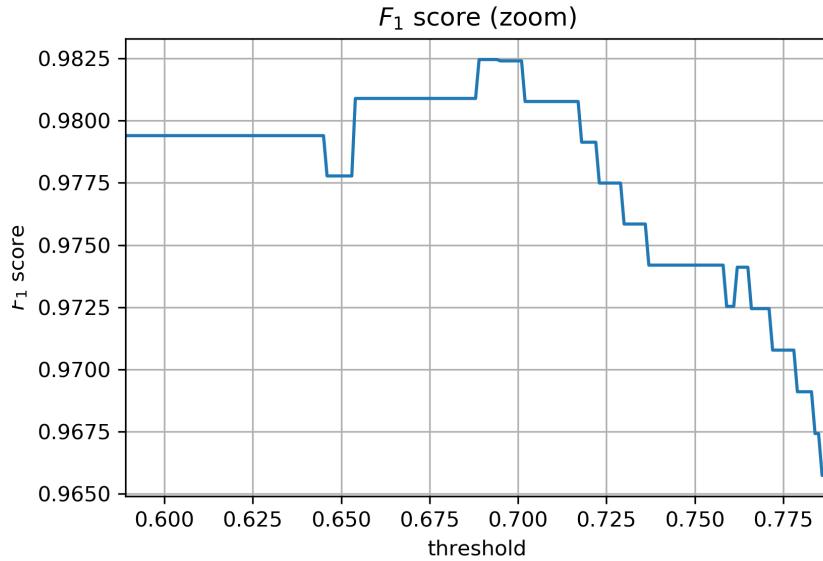


Figure 11: Zoom in F1-score curve

If we give a zoom in Figure 10 highlighting the points near the maximum of the curve, as shown in Figure 11, we can choose the most appropriate threshold value. Knowing that the F1-score can be interpreted as a weighted average of the precision and recall (with 1 being the best value and 0 being the worst), we then choose for the threshold which maximizes the F1-score. This gives us a threshold of 0.689.

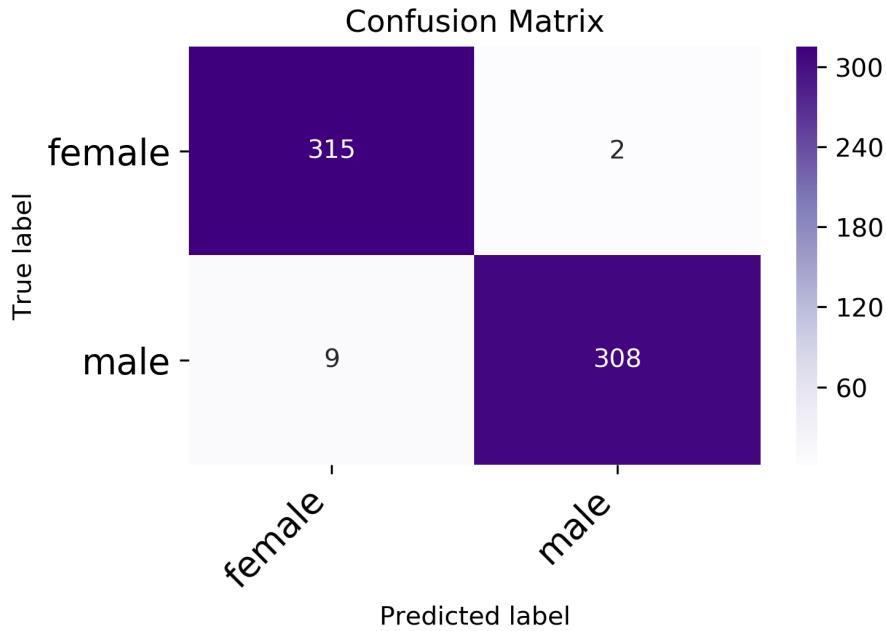


Figure 12: Confusion Matrix for binary classification (logistic regression)

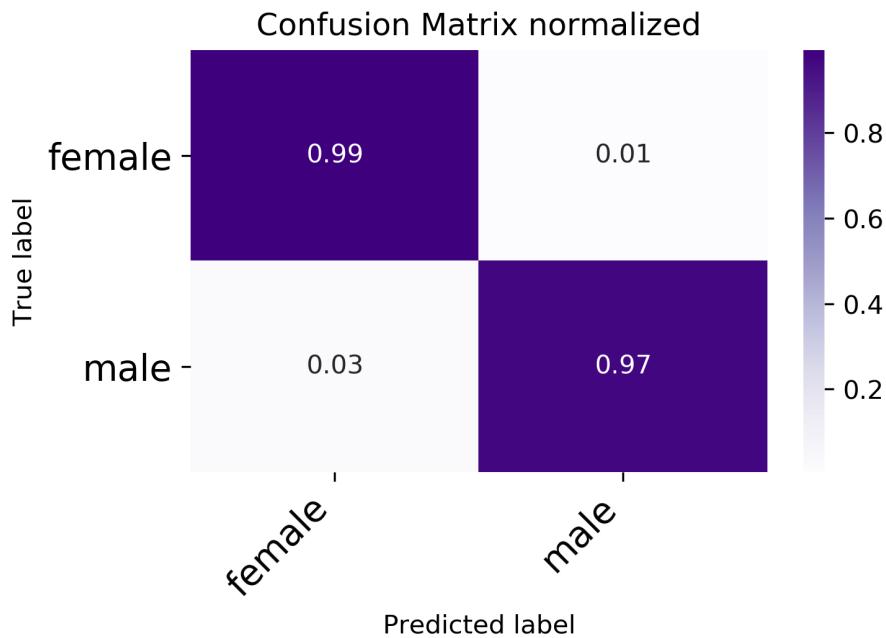


Figure 13: Confusion Matrix normalized for binary classification (logistic regression)

Using the chosen threshold (with value 0.689) we now plot the confusion matrix. The Figure 12 shows the confusion matrix with raw values and the Figure 13 shows the one with normalized values.

Finally, we can calculate the accuracy for this classifier, which gives us a value of 0.9826, indicating a good performance.

3 Part 2 - Multiclass Classification

We start this exercise by performing a one-hot encoding with the data. This way we can represent the categorical variables in a binary way. This encoding is defined as follows:

output:

- 1 – caminhada
- 2 – subindo escadas
- 3 – descendo escadas
- 4 – sentado
- 5 – em pé
- 6 – deitado

one-hot encoding:

- $[1 \ 0 \ 0 \ 0 \ 0 \ 0]^T$: walking
- $[0 \ 1 \ 0 \ 0 \ 0 \ 0]^T$: climbing stairs
- $[0 \ 0 \ 1 \ 0 \ 0 \ 0]^T$: going down stairs
- $[0 \ 0 \ 0 \ 1 \ 0 \ 0]^T$: seated
- $[0 \ 0 \ 0 \ 0 \ 1 \ 0]^T$: standing
- $[0 \ 0 \ 0 \ 0 \ 0 \ 1]^T$: lying

3.1 a) Logistic regression (using softmax approach)

The approach used to implement the multiclass classifier was the softmax function. All operations involved was performed in the matrix way, in order to try to improve the computational costs involved.

A weight array with $K+1$ elements (in this case we have $K = 561$ features) is randomly initiated according to a uniform distribution between -1 and 1 for each class, as can be seen:

$$\mathbf{w}_C = \begin{bmatrix} w_0^{(C)} & w_1^{(C)} & \dots & w_K^{(C)} \end{bmatrix}^T$$

As we have six different classes, the \mathbf{W} matrix can be generated as follows:

$$\mathbf{W} = [\mathbf{w}_0 \ \mathbf{w}_1 \ \dots \ \mathbf{w}_K] = \begin{bmatrix} w_0^{(0)} & w_0^{(1)} & \dots & w_0^{(C)} \\ w_1^{(0)} & w_1^{(1)} & \dots & w_1^{(C)} \\ \vdots & \vdots & \ddots & \vdots \\ w_K^{(0)} & w_K^{(1)} & \dots & w_K^{(C)} \end{bmatrix}$$

The stop criteria adopted was a tolerance of 2.10^{-2} for the error in a given iteration. The function used to calculate the error was the zero-one loss, where a 0 is assigned when the class was correctly classified, and 1 when it was not. We then calculate the zero-one loss for all data in the training set, sum all of them and divide by the number of inputs (7352 in this case).

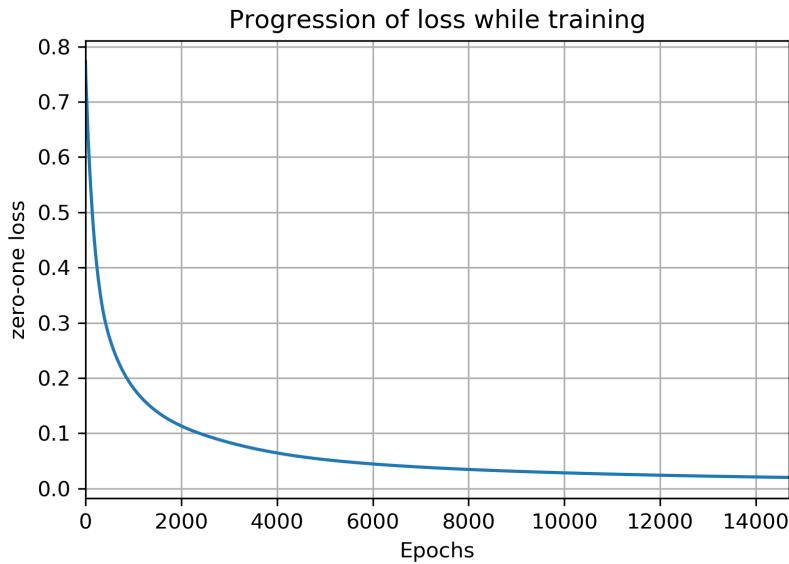


Figure 14: Zero-one loss curve

The variation of the error with the number of epochs is shown in the Figure 14.

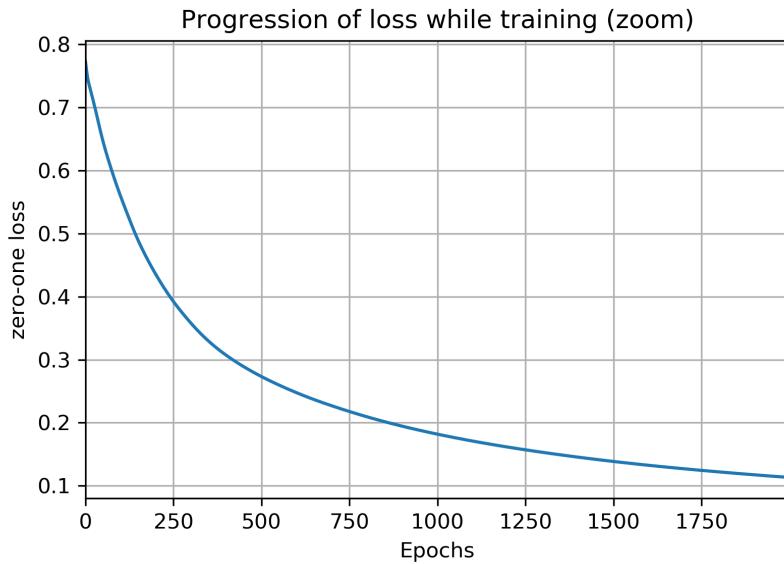


Figure 15: Zoom in zero-one loss curve

In the Figure 15 we can see a zoom of the previous curve focusing in the beginning of the curve. As we can see, the initial error is very high (bigger than 0.9) due to the randomly generated weights.

Now it is time to measure the performance of the classifier. To plot the confusion matrix we are using the “`plot_confusion_matrix`” described in the notebook, and the “`f1_score`” function provided by the scikit-learn library.

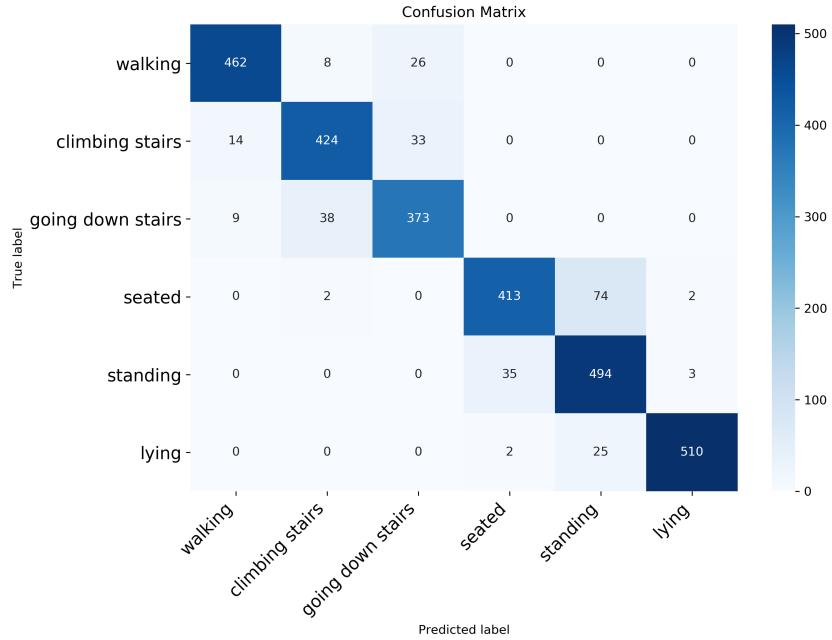


Figure 16: Confusion Matrix using softmax model

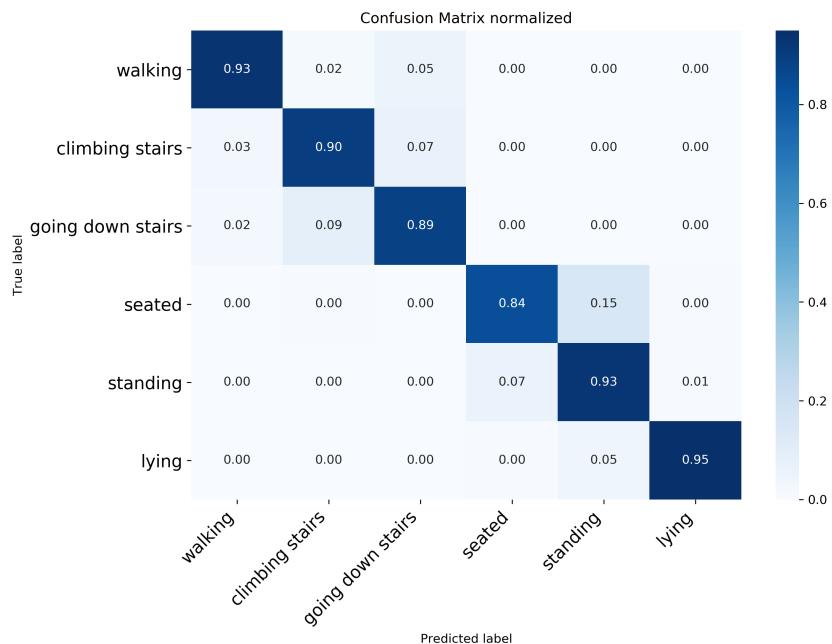


Figure 17: Confusion Matrix with normalized values using softmax model

Figures 16 and 17 shows the confusion matrix with raw and normalized values, respectively.

Having the confusion matrix, we can calculate metrics to measure the performance of the classifier and compare with the k-Nearest Neighbor, the next model we will validate. The zero-one loss and the F1-score obtained are:

- Zero-one loss = 0.0929
- F1-score micro = 0.9080
- F1-score macro = 0.9071

3.2 b) k-Nearest Neighbors

For the k-Nearest Neighbors (kNN) method, we use a lazy learning method. For this, the first thing we should do is to calculate somehow the distance between the new data and all the data from the training set. The distance metric used here was the Minkowski.

Minkowski distance:

$$d(x, y) = \left(\sum_{i=1}^K |x_i - y_i|^p \right)^{1/p}$$

For $p = 2$, we have the Euclidian distance:

$$d(x, y) = \sqrt{\sum_{i=1}^K |x_i - y_i|^2}$$

In order to reduce the calculation costs, the distance between all data from the training set and all data in the validation set were calculated in the matrix form. A distance matrix D was created for this, as can be seen as follows:

D : distances matrix
 D shape: ($M \times N$)

$$D = \begin{bmatrix} d(x_{T_0}, x_{D_0}) & d(x_{T_0}, x_{D_1}) & \dots & d(x_{T_0}, x_{D_N}) \\ d(x_{T_1}, x_{D_0}) & d(x_{T_1}, x_{D_1}) & \dots & d(x_{T_1}, x_{D_N}) \\ \vdots & \vdots & \ddots & \vdots \\ d(x_{T_M}, x_{D_0}) & d(x_{T_M}, x_{D_1}) & \dots & d(x_{T_M}, x_{D_N}) \end{bmatrix}$$

Where:

- x_{T_M} is the M^{th} sample of training set
- x_{D_N} is the N^{th} sample of data set
- $d(x_{T_M}, x_{D_N})$ is the Minkowski distance between the points x_{T_M} and x_{D_N}

3.2.1 Uniform weights

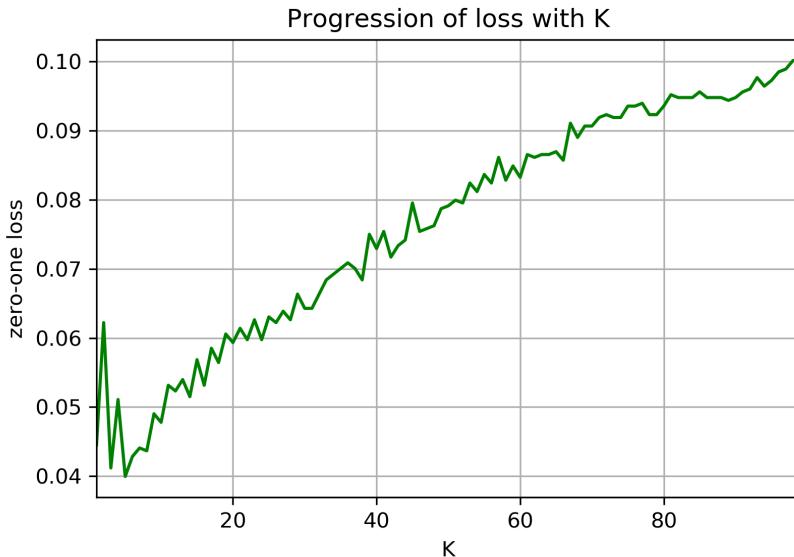


Figure 18: Zero-one loss curve (uniform weights)

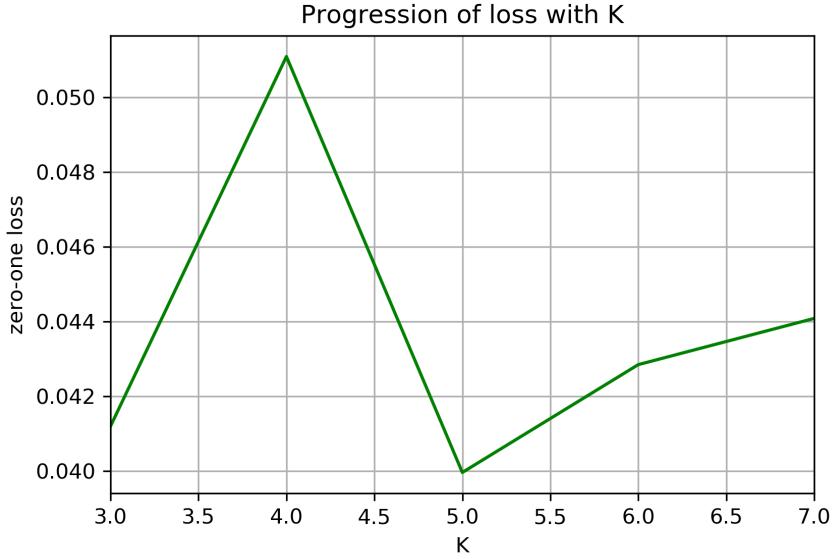


Figure 19: Zero-one loss curve (uniform weights)

During the decision step two approaches were used. The first one presented here uses uniform weights, i.e., each k neighbors have the same weight when voting for their own class. The result for this approach can be seen in the Figure 18. In Figure 19 we can see a zoom in the previous plot showing the value of k optimum = 5, which minimizes the loss function.

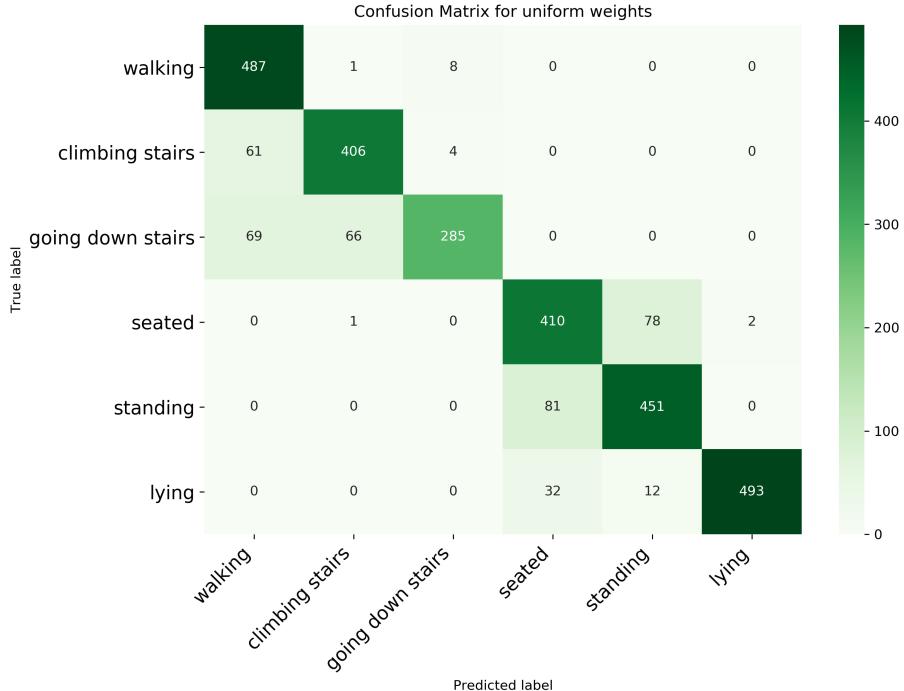


Figure 20: Confusion Matrix for uniform weights

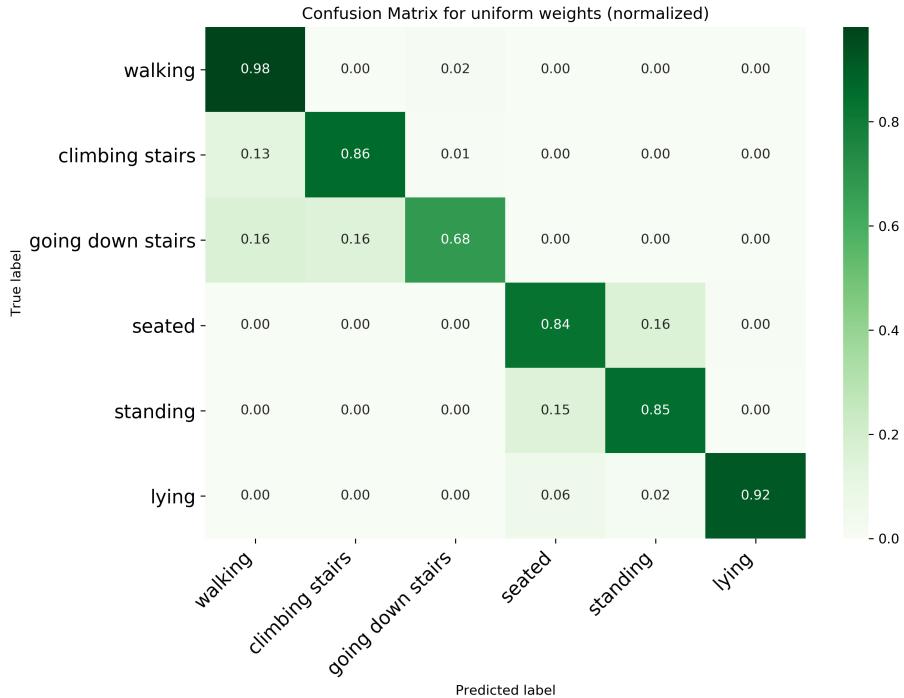


Figure 21: Confusion Matrix normalized for uniform weights

After finding the best value for the hyperparameter $k = 5$, we now apply the model to the test data. The results can be seen in Figures 20 and 21, which shows the confusion matrix with raw values and with normalized values, respectively.

We then use metrics to measure the performance of the classifier. The results obtained are:

- Zero-one loss = 0.1274
- F1-score micro = 0.8592
- F1-score macro = 0.8557

3.2.2 Distance weights

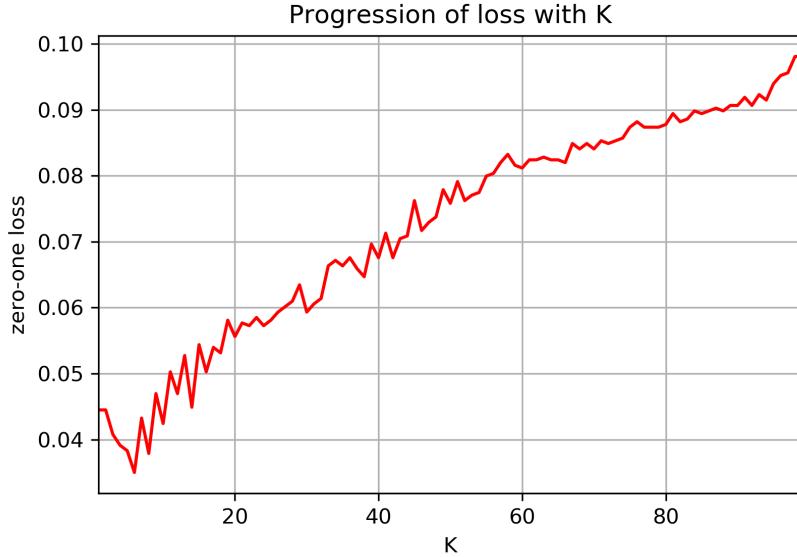


Figure 22: Zero-one loss curve (inversely proportional to distance weights

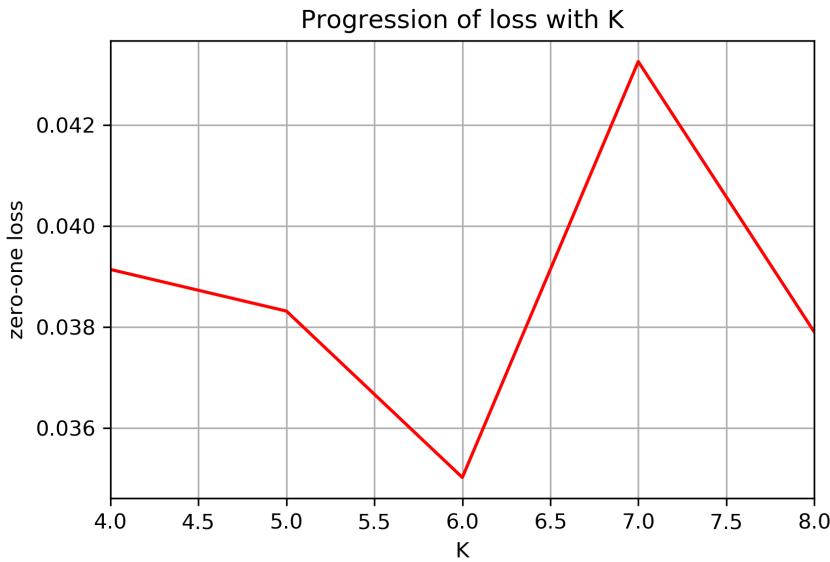


Figure 23: Zero-one loss curve (inversely proportional to distance weights

The second approach uses distance weights. In this case, each vote is pondered according to the inverse of the distance. This means that when predicting the class for a new input, the class of a very similar data in the training set has more relevance than a data not so close to this new input. The result for this approach can be seen in the Figure 22. In Figure 23 we can see a zoom in the previous plot showing the value of k optimum = 6, which minimizes the loss function.

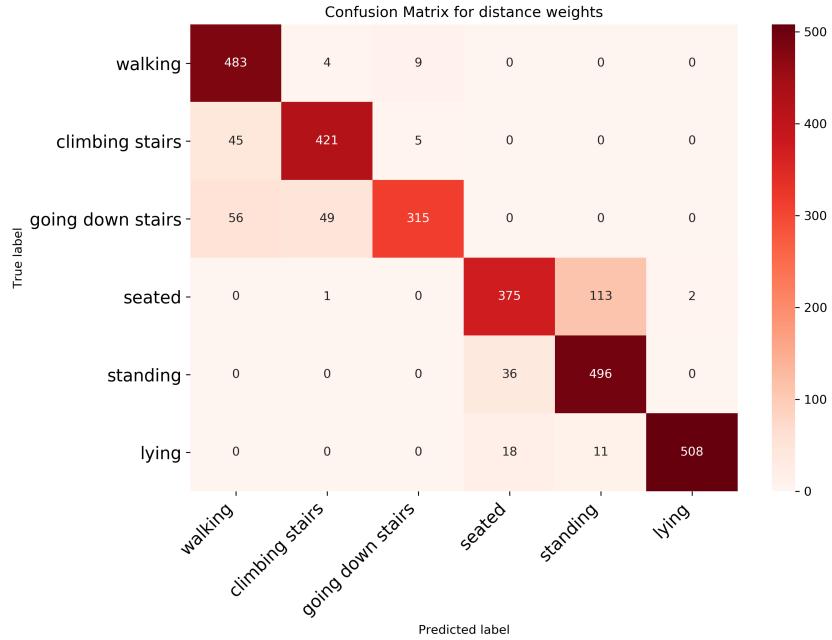


Figure 24: Confusion Matrix for uniform weights



Figure 25: Confusion Matrix for distance weights

After finding the best value for the hyperparameter $k = 6$, we now apply the model to the test data. The results can be seen in Figures 24 and 25, which shows the confusion matrix with raw values and with normalized values, respectively.

The last thing we did was to use metrics to measure the performance of the classifier. The results obtained are:

- Zero-one loss = 0.1030
- F1-score micro = 0.8816
- F1-score macro = 0.8787

3.3 Results and conclusion

	softmax	kNN	
		distance	uniform
Zero-one loss	0.0920	0.1030	0.1274
F1-score: micro	0.9080	0.8816	0.8592
F1-score: macro	0.9071	0.8787	0.8557

Table 1: Comparison of results between softmax and kNN models for multiclass classification

We now can compare the performance between the two models proposed in a multiclass classification scenario. For the k-Nearest Neighbors, two approaches were used: one considering uniform weights for the neighbors and another one considering distance weights. Table 1 shows a summary of the results obtained:

As expected, the softmax model is more robust. It presents the smallest error and the highest value for the F1-score, both micro and micro, showing that the overall classification is good (micro) and also the classification in each individual class (macro).

The kNN model, however, it does not present bad results when comparing to the softmax model. Actually, the results are quite reasonable, close to the ones obtained with the softmax function, which indicates that depending on the dataset and the problem, it can be an easy to implement and low cost alternative, considering that there is no training in this method, but only distance measures and comparison between them.

Still talking about the kNN model, when comparing the two approaches used, the distance weights has a superior performance than the uniform weights, even though the number of neighbors used are similar (5 for the uniform approach and 6 for the distance approach, chose by a cross-validation method).