EFC2 - Exercise 2

Rafael Claro Ito (R.A.: 118430)

September 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
- LaTeX
  - 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.

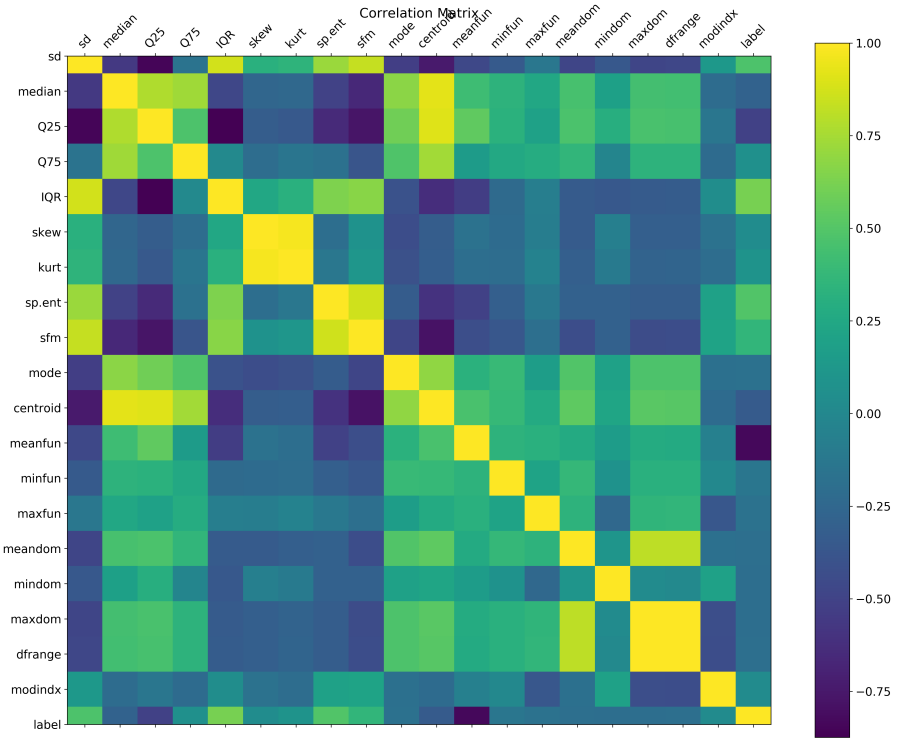
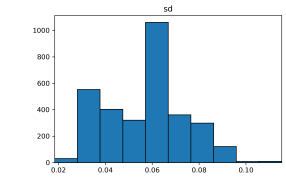


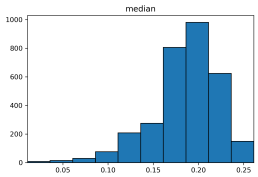
Figure 4: Correlation between input features

In the Figure 4 we can see the correlation matrix, with values between 0 and 1, mapped in colors.

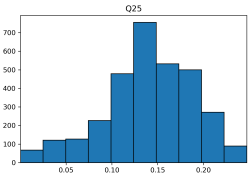
H



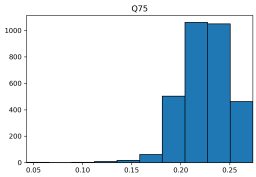
(a) sd



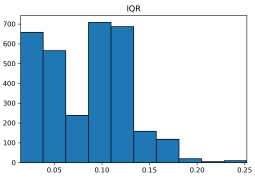
(b) median



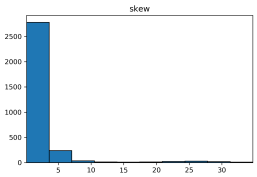
(c) Q25



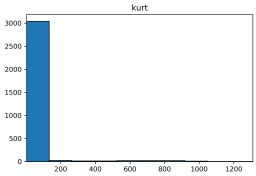
(d) Q75



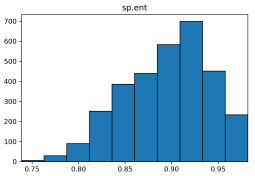
(e) IQR



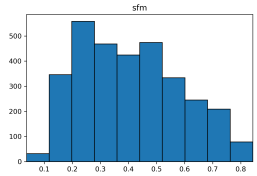
(f) skew



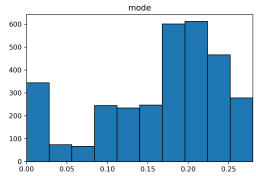
(g) kurt



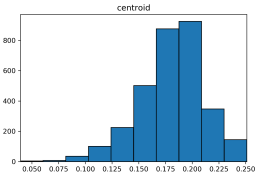
(h) sp ent



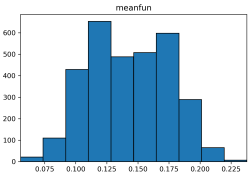
(i) smf



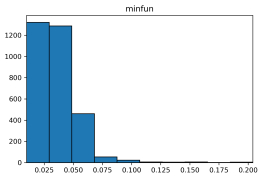
(j) mode



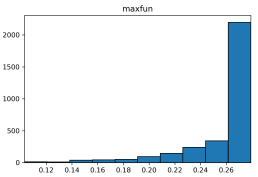
(k) centroid



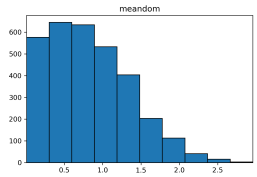
(l) meanfun



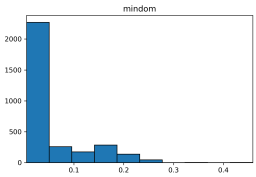
(m) minfun



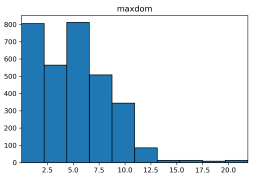
(n) maxfun



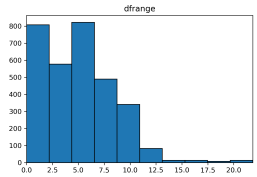
(o) meandom



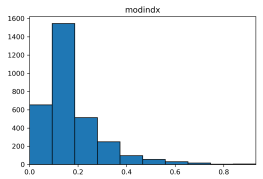
(p) mindom



(q) maxdom



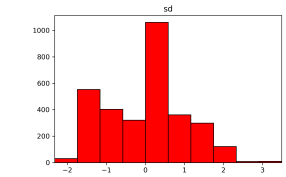
(r) dfrange



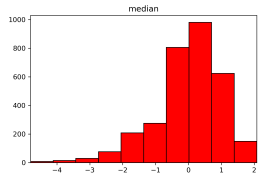
(s) modindx

Figure 2: Histograms of raw input features

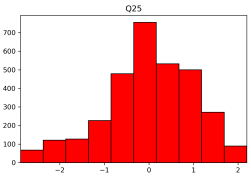
H



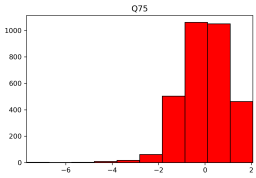
(a) sd



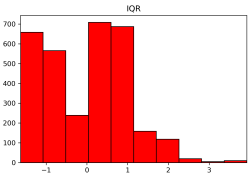
(b) median



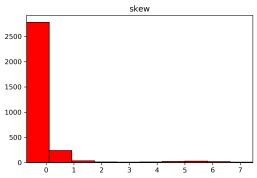
(c) Q25



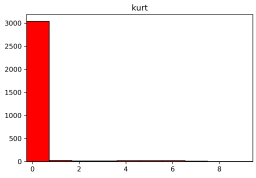
(d) Q75



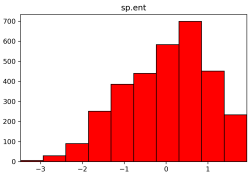
(e) IQR



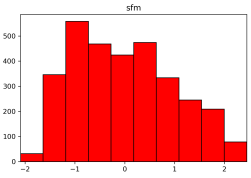
(f) skew



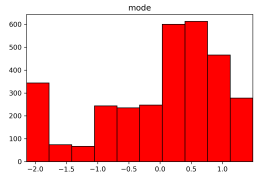
(g) kurt



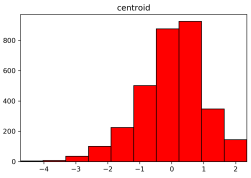
(h) sp ent



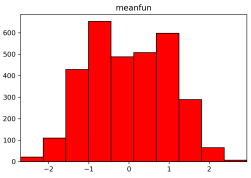
(i) smf



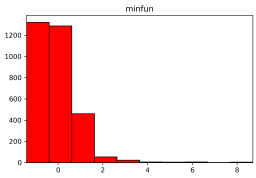
(j) mode



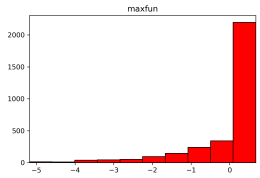
(k) centroid



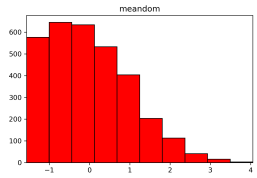
(l) meanfun



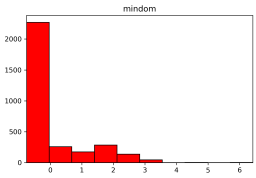
(m) minfun



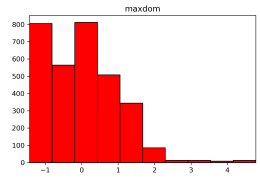
(n) maxfun



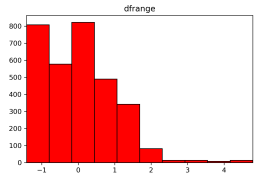
(o) meandom



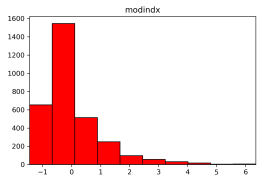
(p) mindom



(q) maxdom



(r) dfrange



(s) modindx

Figure 3: Histograms of input features after data standardization

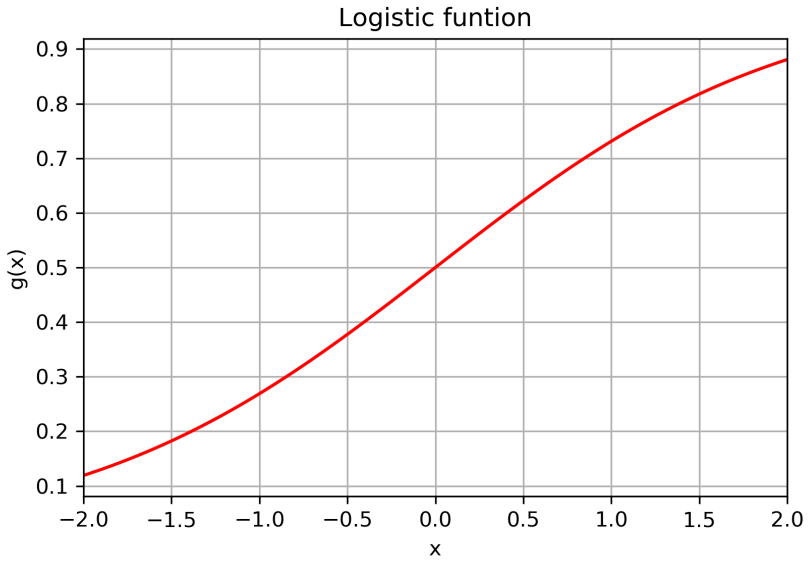


Figure 5: Logistic function

In the first notebook we also plotted the logistic function, used in the model proposed for binary classification. The Figure 5 shows the logistic function in an interval of -2 and 2.

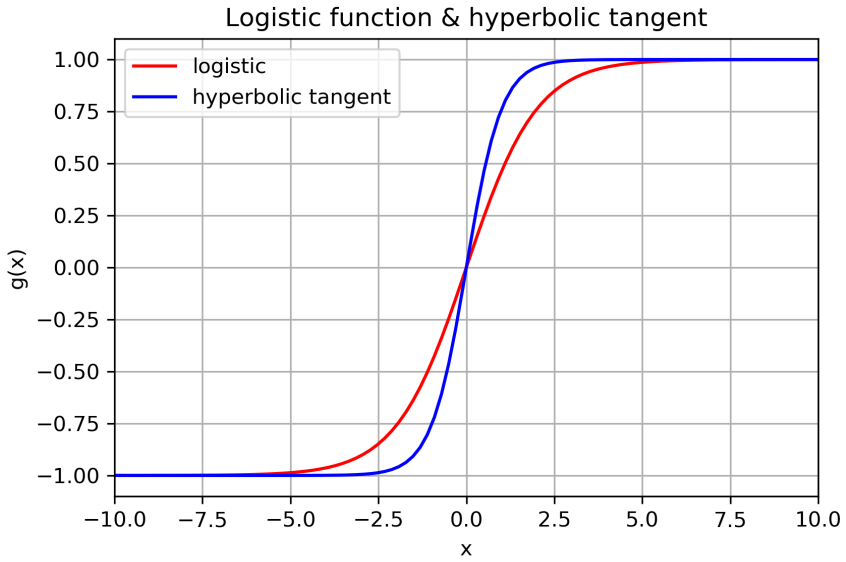


Figure 6: Comparasion between hyperbolic tangent and logistic function(scaled and with offset)

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 6. Here, the logistic function was multiplied by two, and subtracted by 1 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 built. 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.

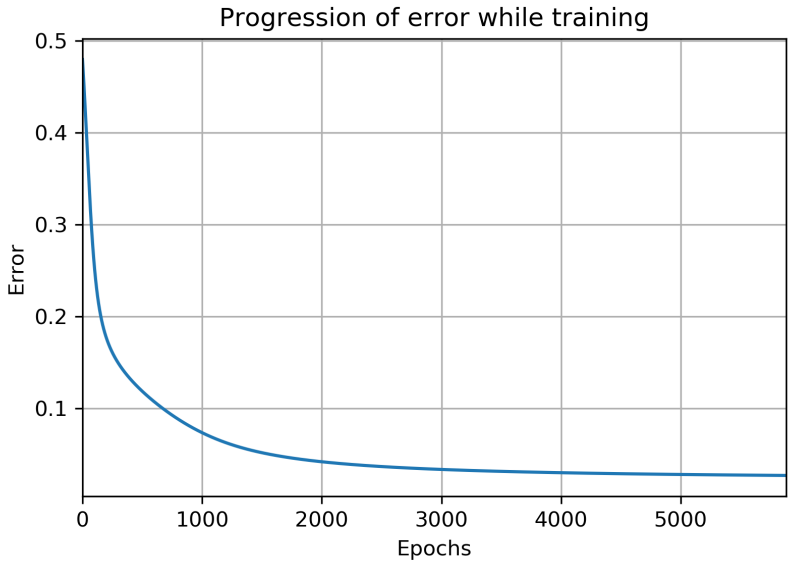


Figure 7: Error progression in training

The training step is performed until the difference between successive iteration errors is under a provided tolerance. 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 7. We are considering the whole batch of samples for each step during the training.

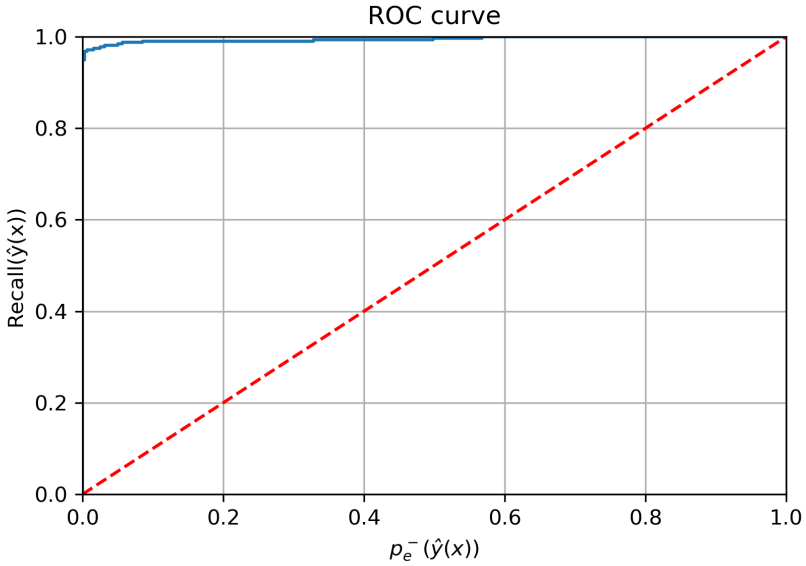


Figure 8: ROC curve

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 8. 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.

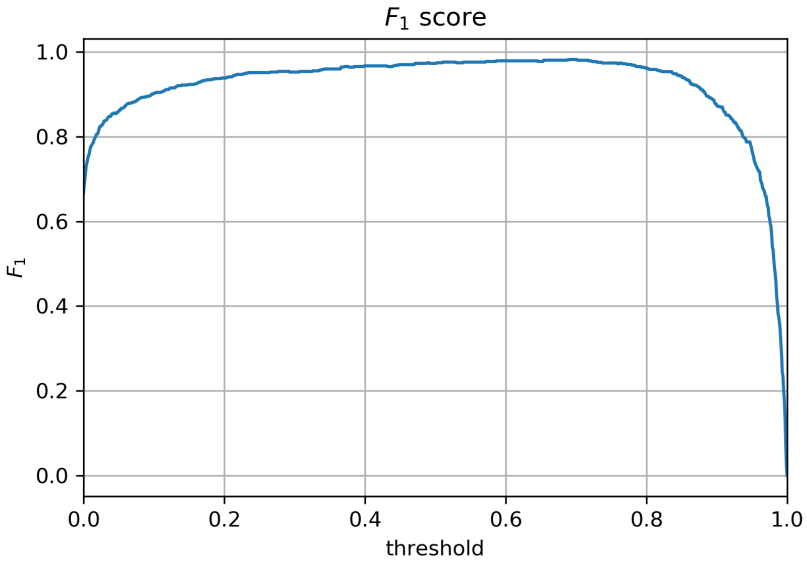


Figure 9: F1-score curve

We now change the threshold used to classify a sample with the positive or negative class. We use values of threshold from 0 to 1 with step of 0.001, i.e., we are using 1000 thresholds values. For each value of threshold we calculate the F1-score. Figure 9 shows this plot.

2.3 c) Threshold, confusion matrix and accuracy.

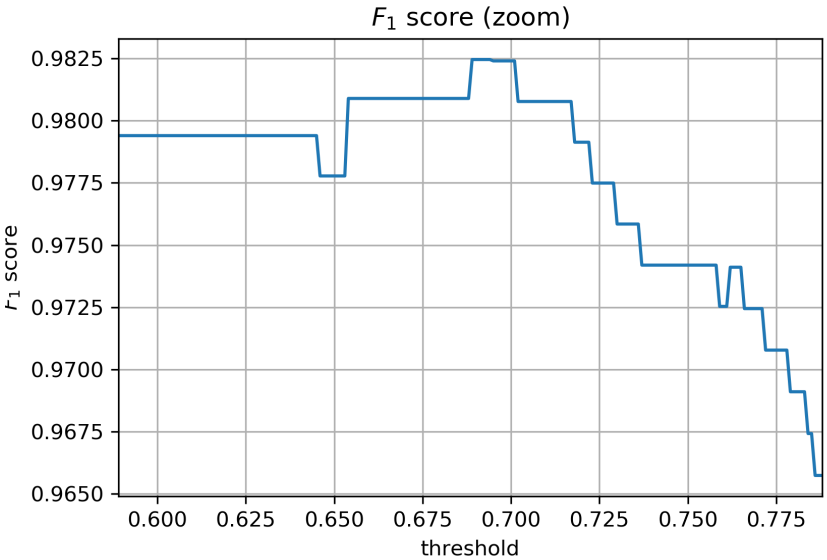


Figure 10: Zoom in F1-score curve

If we give a zoom in Figure 9 highlighting the points near the maximum of the curve, as shown in Figure 10, 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.

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 enconding:

$[1\ 0\ 0\ 0\ 0\ 0]^T$: 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. 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:

$$\boldsymbol{w_C} = \begin{bmatrix} w_0^{(C)} & w_1^{(C)} & ... & w_K^{(C)} \end{bmatrix}^T$$

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

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_0} & \boldsymbol{w_1} & ... & \boldsymbol{w_K} \end{bmatrix} = \begin{bmatrix} w_0^{(0)} & w_0^{(1)} & ... & w_0^{(C)} \\ w_1^{(0)} & w_1^{(1)} & ... & w_1^{(C)} \\ \vdots & \vdots & \ddots & \vdots \\ w_K^{(0)} & w_K^{(1)} & ... & w_K^{(C)} \end{bmatrix}$$

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).

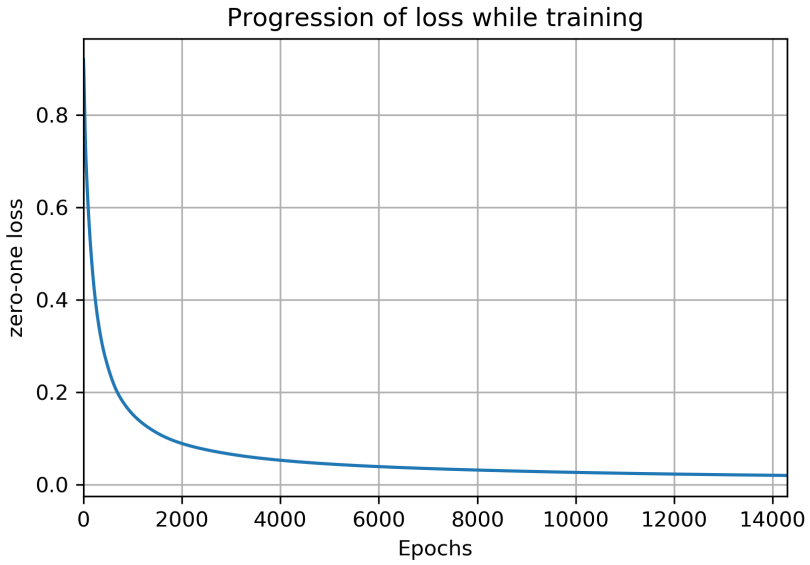


Figure 11: Zero-one loss curve

The variation of the error with the number of epochs is shown in the Figure 11.

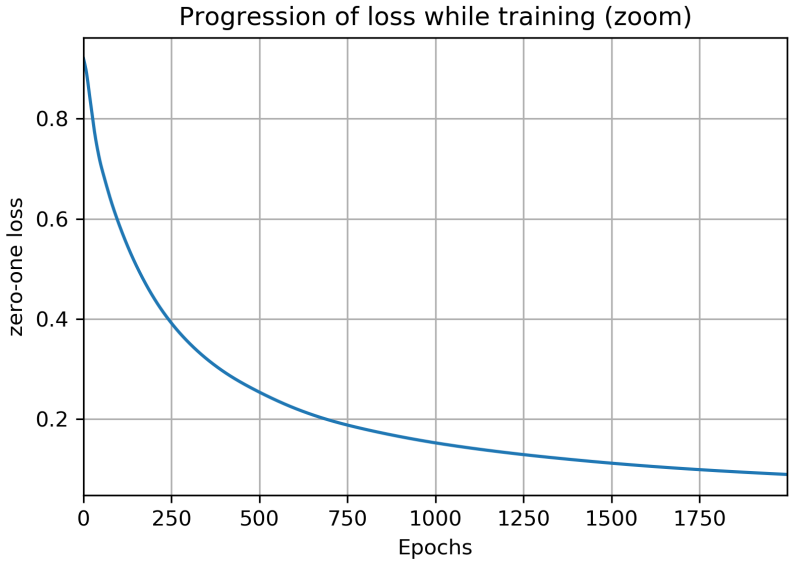


Figure 12: Zoom in zero-one loss curve

In the Figure 12 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 do this, we again are using the metrics provided by the scikit-learn library. We are using the "*confusion_matrix*" and "*f1_score*" functions.

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 test 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 x N)

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

Where:

- x_{T_M} is the Mth sample of training set
- x_{D_N} is the Nth 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}

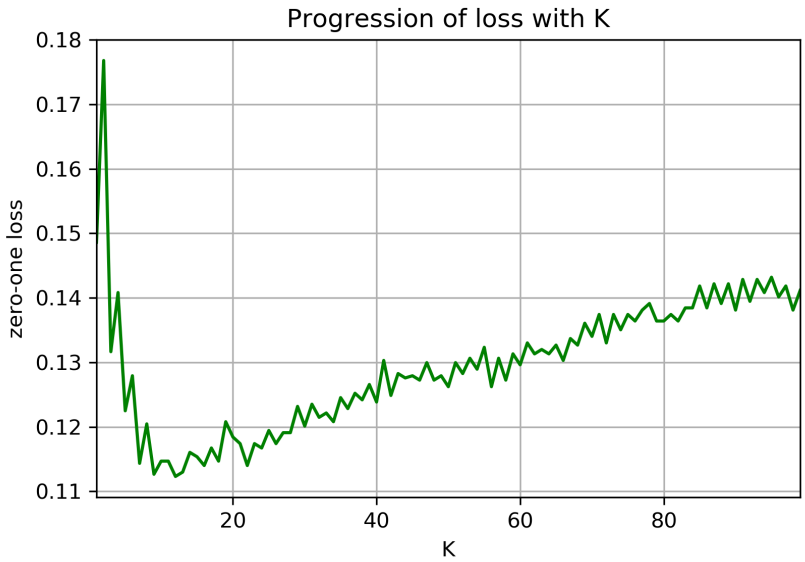


Figure 13: Zero-one loss curve (uniform weights

Figure 14: 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 13. In Figure 14 we can see a zoom in the previous plot showing the value of k optimum = AAAAAAAAA.

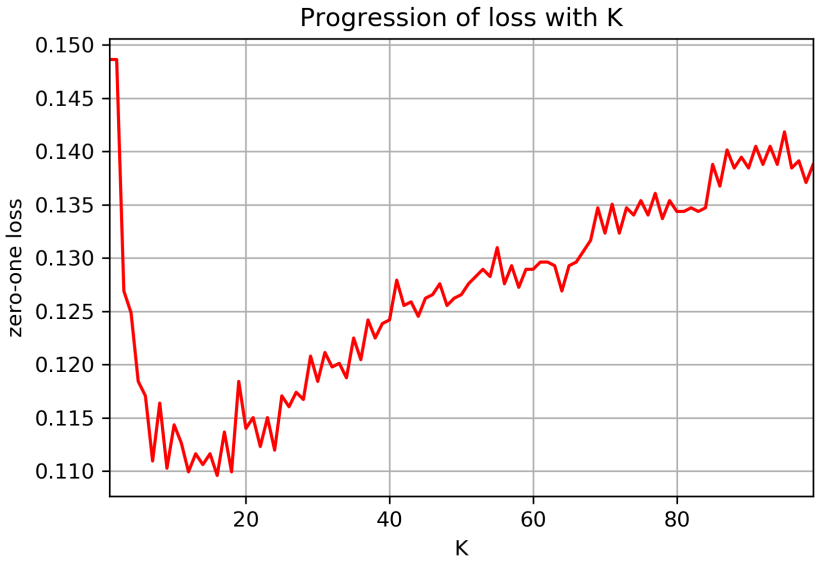


Figure 15: Zero-one loss curve (inversely proportional to distance weights

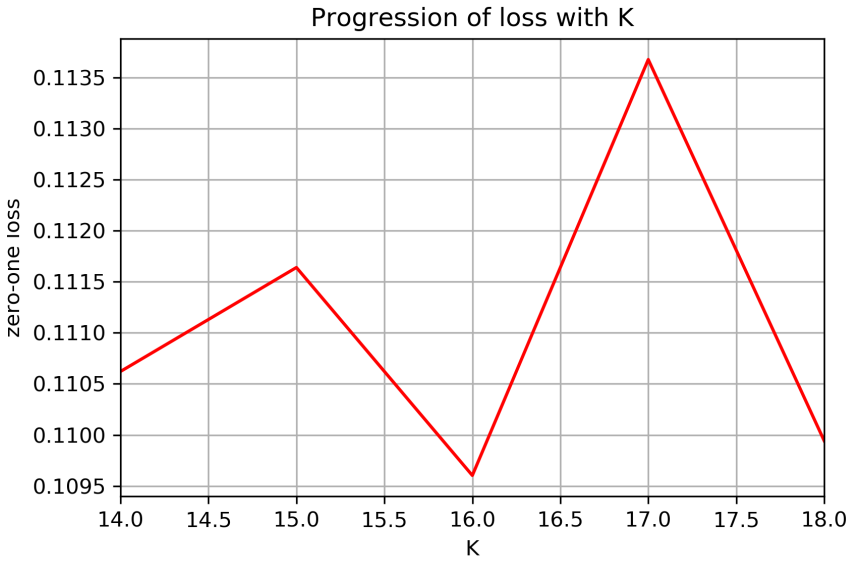


Figure 16: Zero-one loss curve (inversely proportional to distance weights

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 15. In Figure 14 we can see a zoom in the previous plot showing the value of k optimum = AAAAAAAAA.

COMPARASION BETWEEN softmax and kNN COMPARASION BETWEEN DISTANCE AND UNIFORM