



Intelligent Systems

Project

Developing of a neural system, a Mamdany-type fuzzy inference system and a Sugeno-type fuzzy inference system for the identification of a volunteer position/activity

Francesco Paolo Culcasi
30/05/2016

This page intentionally left blank.

SUMMARY

Summary	iii
1 - Project specifications	1
1.1 - Data description	1
1.2 - Objective.....	2
2 - Data analysis	3
2.1 – Data initialization.....	3
2.2 – Data cleaning	4
2.2.1 – Signal smoothing	4
2.2.2 – Normalization	5
2.3 - Feature extraction.....	5
2.4 – Feature selection.....	7
2.4.1 – Sequential feature selection.....	7
3 - Neural Network	9
3.1 – Defining the problem	9
3.2 – Train the Neural Network.....	9
3.3 – Network outputs evaluation	9
3.2.1 – Smaller necessary interval	10
Appendix A - Data pre-processing	13
A.1 init.m	13
A.2 split.m	15
A.3 cleaning.m	16
A.4 featureEx.m	17
A.5 my_autocorr.m	19
A.6 my_fft.m.....	19
A.7 featureSel.m	19
Appendix B –Neural network	20
B.1 nn.m	20
B.2 cfmatrix2.m	21

1 - PROJECT SPECIFICATIONS

The project requires the analysis of a set of medical data. The application context is dermatology and, in particular, the compression therapy by means of bandages in the treatment of venous ulcers of the leg.

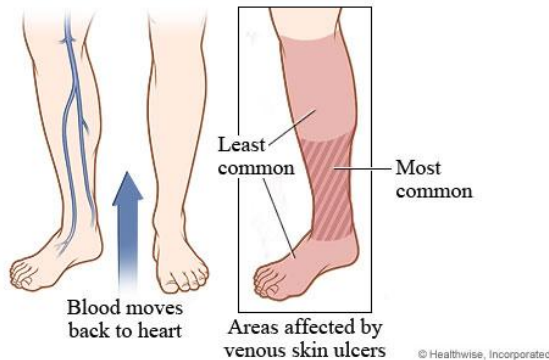


FIGURE 1 - COMPRESSION THERAPY

In dermatology, venous ulcers are very frequent lesion of the skin of the legs, due to a compromised functioning of the blood circulation. To repair these ulcers, the blood circulation must be enhanced by exploiting the calf muscle pump which allows the blood to be moved back to the heart. The basic way to re-establish the blood circulation is the compression therapy.

In the compression therapy, the doctor applies a compression bandage on the leg of the patient providing a given pressure of the ulcer area. The pressure applied by the bandage allows to repair the ulcer, typically, in a few months.

The pressure applied by the bandage, and thus the efficiency of the therapy, depends on several factors:

- The type of bandage (depending, e.g., on the elasticity);
- The correct application of the bandage: an appropriate pressure should be applied in several parts of the leg according to the position of the ulcer, and the pressure should remain constant as much as possible between bandage applications;
- The activities performed by the patient during the therapy (e.g., walking, standing, etc.).

1.1 - DATA DESCRIPTION

The data refer to 10 volunteers. A compression bandage was applied to their calf. Three sensors were applied in three different position to measure sub-bandage pressure. Each volunteer wears the bandage for 12 minutes. During this time, the volunteer perform different activities or maintains their positions. Each activity/position is performed/maintained for about 3 minutes. The sensors measure the pressure with the sampling time of about 82 ms. The positions/activities taken into account are the following:

- A. Supine position
- B. Dorsiflexion standing
- C. Walking
- D. Stair climbing

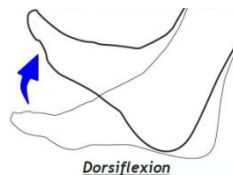


FIGURE 2 - DORSIFLEXION ACTIVITY

The data are organized as follows. One folder for each volunteer is provided. In each folder there are 4 files, one for each position/activity performed. Each file contains the measurements of the three sensors (in the first three columns), and the corresponding sampling time (in the last column). Please, note that the sensor measurements are electrical resistance and are measured in Ohms.

1.2 - OBJECTIVE

The objective of the project is twofold: on the one hand, we want to identify the position/activity of the volunteer by analyzing the pressure of the bandage, i.e., to distinguish among supine position, dorsiflexion standing, walking and stair climbing; on the other hand, we want to find out the least temporal interval that is necessary and sufficient to recognize the position/activity.

2 - DATA ANALYSIS

Given the data organization decrypted in the previous chapter, each signal coming from a sensor is a set of approximately 2000 samples. Rather than give all these data as input of our system, we prefer to appropriately represent the signal in terms of a reduced number of features that can summarize the information deriving from the original.

For this reason on the following paragraphs we are going to evaluate the signal after a preliminar feature extraction phase.

2.1 – DATA INITIALIZATION

At the starting point we have to deal with raw signals that are quite difficult to analyze. This implies a pre-processing phase with the purpose to get some structures in order to ease the computations on data ([Appendix A.1](#)). In the pressure measurement of the activity we observe that the three different sensors may show different shapes and shifts due to their position on the calf sub-bandage. For this reason we also introduced a fourth “virtual” sensor, obtained as the sum of the former three, to summarize the overall measurement.

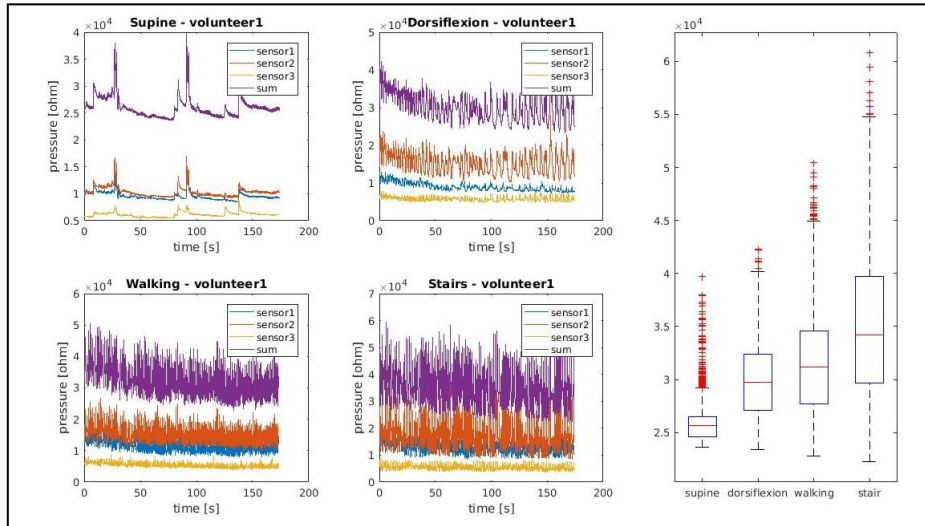


FIGURE 1 SIGNALS AND BOXPLOTS OF POSITION/ACTIVITIES MEASURED ON VOLUNTEER 1

Moreover our data should be sliced to obtain several pieces of the signal with smaller duration than the 3 minute signal for each activity. This has twofold profit: we split every signal to find out the least temporal interval that is necessary and sufficient to recognize the position/activity and even more we have much more signals that we can use to train/test/valuate our neural network ([Appendix A.2](#)).

We notice from the signals that it has been sampled with a sampling period of ~82ms (or with a 12.2Hz sampling frequency). Therefore we can compute the number N of samples needed to form a M -seconds-long piece of the signal with the following formula:

$$N = 12.2Hz \times M \text{ sec}$$

A vector with several of these values has been defined to faster switch between different temporal intervals in our project. The values are computed for the following intervals: 3sec, 5sec, 10sec, 12sec, 15sec, 20sec, 30sec, 1min, 2min.

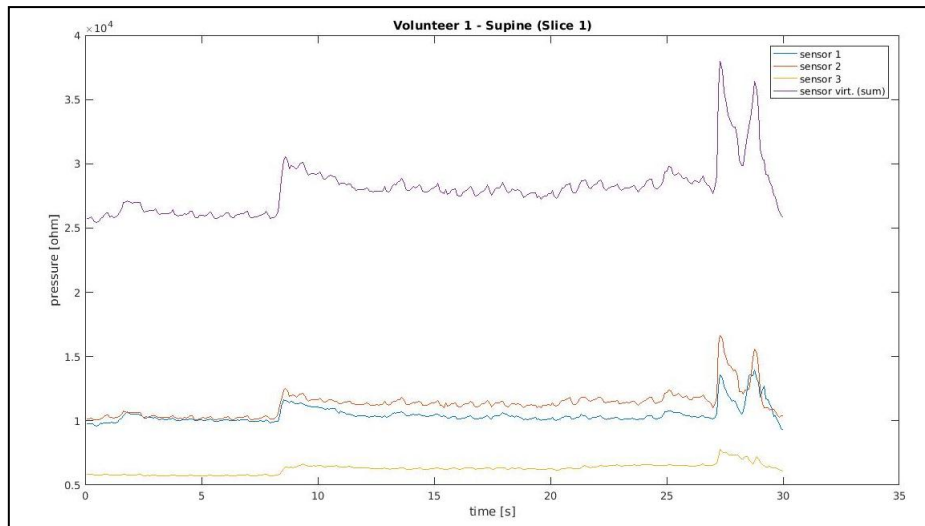


FIGURE 2 FIRST 30-SECONDS-LONG SLICE OF THE SUPINE SIGNALS FROM VOLUNTEER 1

2.2 – DATA CLEANING

The signals we are going to processing are obtained from pressure sensors. As like as every real world signal they are affected from noise that deteriorates the signal.

For this reason it is a good rule to apply some cleaning techniques before any feature extraction.

2.2.1 – SIGNAL SMOOTHING

One of the most common smoothing technique is the use of Savitzky-Golay filter. Savitzky-Golay filter is a digital filter that can be applied to a set of digital data points for the purpose of smoothing the data, that is, to increase the signal-to-noise ratio without greatly distorting the signal. This is achieved, in a process known as convolution, by fitting successive sub-sets of adjacent data points with a low-degree polynomial by the method of linear least squares.

For our project we used a third-order polynomial and a frame length of 7 ([Appendix A.3](#)).

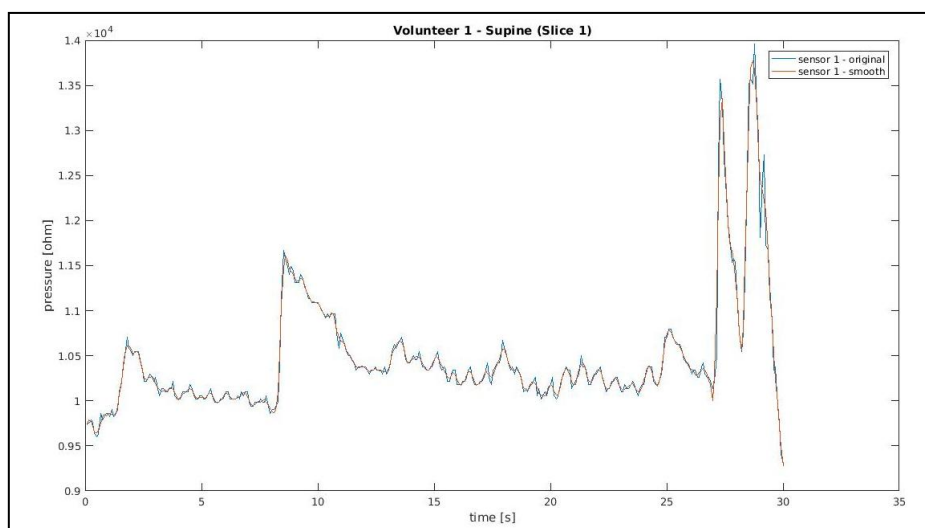


FIGURE 3 SAVITZKY-GOLAY FILTER APPLIED TO A SIGNAL SLICE

2.2.2 – NORMALIZATION

Another important point to deal with is the following: signals may have different scale and shift (see **Figure 2**). This is a huge problem since we would like to compare signals. To solve the problem we apply a normalization phase.

Since our dataset may contain some outliers we use Z-score normalization of the signal X , that is not affected from outliers. It can be described with the formula below:

$$Z = \frac{X - \mu}{\sigma}$$

where we define with μ and σ respectively the signal mean and standard deviation.

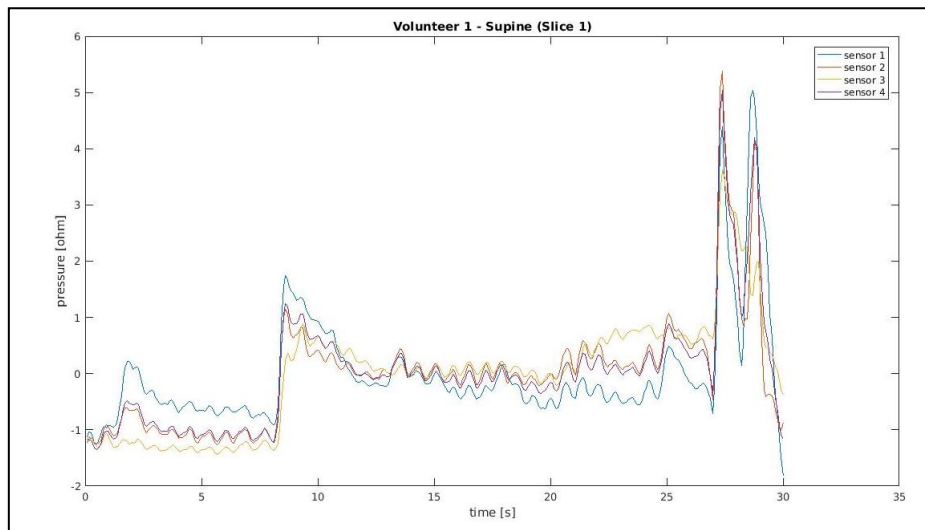


FIGURE 4 NORMALIZED SIGNALS

2.3 - FEATURE EXTRACTION

The goal is to reduce the number of inputs sufficient to distinguish among positions/activities. In order to do that we start analyzing signals shape.

From **Figure 1**, representing the 4 activities of the volunteer 1, it become immediately evident that signals belonging on different activities can be discriminated by the following temporal characteristics:

- **Max/Min:** the maximum and minimum values of each signals may be significantly important;
- **Standardized moments:**
 - **1st std. moment (mean):** after the normalization phase the mean value has been nullified;
 - **2nd std. moment (variance):** the same as for mean value.
 - **3rd std. moment (skewness):** is a measure of the asymmetry, computed as
$$\widetilde{\mu}_3 = \frac{E[(X - \mu)^3]}{\sqrt{(E[(X - \mu)^2])^3}};$$
 - **4th std. moment (kurtosis):** is a measure of the "tailedness", measured as:

$$\widetilde{\mu}_4 = \frac{E[(X - \mu)^4]}{\sqrt{(E[(X - \mu)^2])^4}};$$

- **Autocorrelation:** is the correlation of a signal with a delayed copy of itself as a function of delay, and can be obtained with

$$R_{xx}(\Delta) = \frac{E[(X_t - \mu)(X_{t+\Delta} - \mu)]}{\sigma^2}$$

For this project we take in consideration the first 20 lags ($\Delta=1,2,\dots,20$).

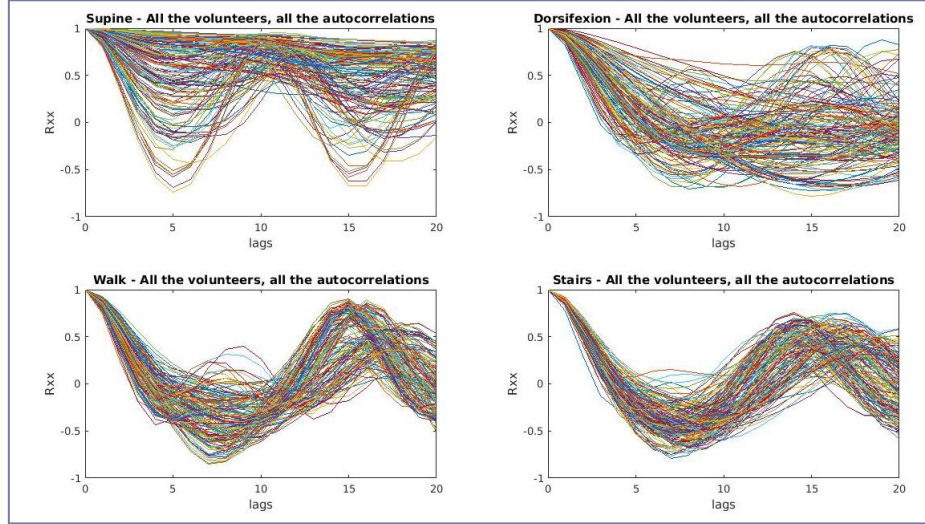


FIGURE 5 AUTOCORRELATIONS FOR ALL THE ACTIVITIES OF EVERY VOLUNTEER

In addition, performing some spectral analysis, we can infer common frequential features for spectra for the same positions/activities (see **Figure 6**). We can introduce some frequency features that may look appropriated:

- **Fundamental frequency** (from now on f_0): achieved by detecting the maximum peak abscissa of the signals' spectrum, obtained by Fast Fourier Transform (FFT) formula,

$$X_n = \sum_{k=0}^{N-1} x_k e^{-i\frac{2\pi}{N}kq} \quad (q = 0, 1, \dots, N-1)$$

To faster compute f_0 we can approximate it with the maximum value of the FFT;

- **Amplitude of the f_0 peak** (amp);
- **Power Spectral Density (PSD)**: obtained from the spectrum through the formula,

$$P_X = \sum_{n=-\infty}^{+\infty} |X_n|^2.$$

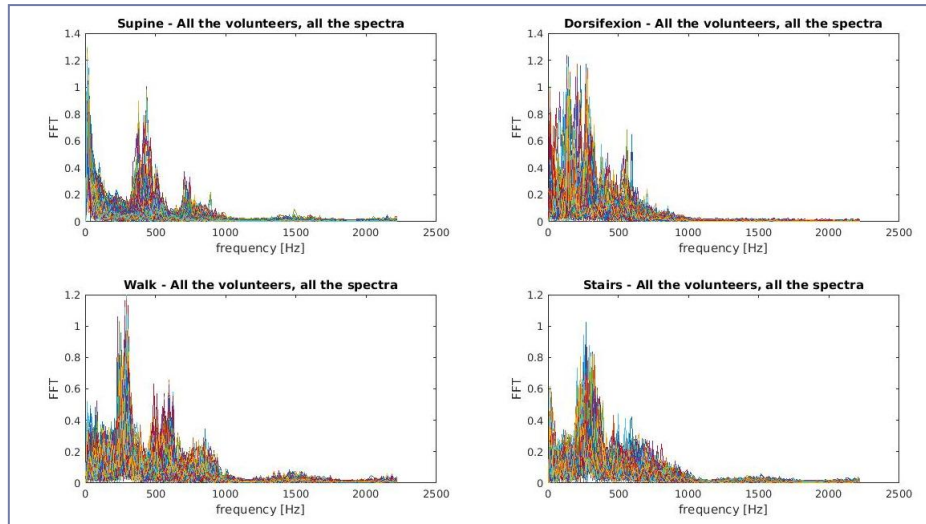


FIGURE 6 FAST FOURIER TRANSFORMS FOR EVERY VOLUNTEER SEPARATED ON ACTIVITY/POSITION

The feature extraction phase is showed in the [Appendix A.4](#).

2.4 – FEATURE SELECTION

So far we have obtained 25 features: max, min, skewness, kurtosis, autocorrelation (20), f0, amp and PSD. But we have to compute these for each of the 4 sensors (3 + 1 virtual). Then we have 108 features.

They are still a lot of inputs for the system, thus we have to select only the more significant for the activity classification.

2.4.1 – SEQUENTIAL FEATURE SELECTION

To obtain the most important features to discriminate among positions/activities we can take advantage of the tools MATLAB makes available.

One of this is “sequentialfs”, that selects a subset of features from the input data matrix basing on the the result of the criterion function ([Appendix A.7](#)). Every detail can be found in the MATLAB documentation.

3 - NEURAL NETWORK

Neural networks are good at recognizing patterns. MATLAB provides two ways to solve this kind of problems:

- Use the “nprtool” GUI;
- Use a command-line solution.

It is generally best to start with the GUI, and then to use the GUI to automatically generate command-line scripts. All the reasoning in the sections below take form in the script showed in [Appendix B.1](#).

Before using either method, the first step is to define the problem by selecting a data set.

3.1 – DEFINING THE PROBLEM

To define a pattern recognition problem, we arrange a set of input features vectors (the ones selected in paragraph 2.4 – Feature selection) as columns in a matrix. Then we arrange another set of vectors so that they indicate the classes to which the input vectors are assigned. Target vector have 4 elements (we have 4 activities to classify), where for each target vector, one element is 1 and the others are 0.

3.2 – TRAIN THE NEURAL NETWORK

To train the neural network we have to ask a preliminary question:

“What’s the best number of hidden neurons?”

In order to answer, we write down some script code to perform the following computations:

- For more values for the number n, number of hidden neurons:
 - Train the network 10 times;
 - Compute the average performance (Mean Square Error);
- Choice as final number of hidden neurons the one that produces the best performance value (least MSE).

Then we automatically generate the command-line script from the graphical tool “nprtool” for the Neural Network with chosen number of hidden neurons. The settings used for the division of data is the following: 70% training, 15% validation and 15% testing.

3.3 – NETWORK OUTPUTS EVALUATION

Once we have got our network trained, we can use it to compute some network outputs. These can be used to evaluate its “goodness”. These information can be showed in diagrams such as **confusion matrix** and **ROC plots** (respectively **Figure 7** and **Figure 8**). Some of the information we can scavenge from these diagrams are: **Accuracy** $\frac{TP+TN}{P+N}$, **Precision** $\frac{TP}{TP+FP}$, **Sensitivity** (or Recall) $\frac{TP}{P}$, **Miss rate** $\frac{FN}{P}$, **Specificity** $\frac{TN}{N}$, **Fall-out** (false alarm) $\frac{FP}{N}$.

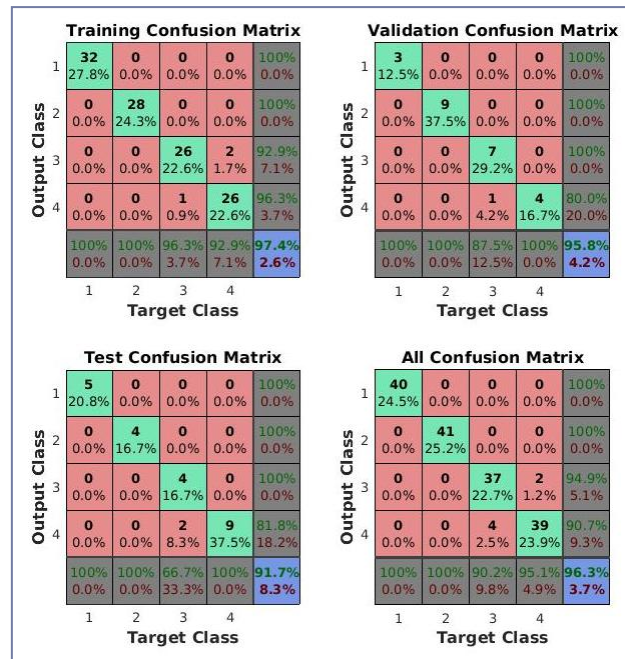


FIGURE 7 CONFUSION MATRICES FOR A 30-SECONDS-LONG SIGNALS

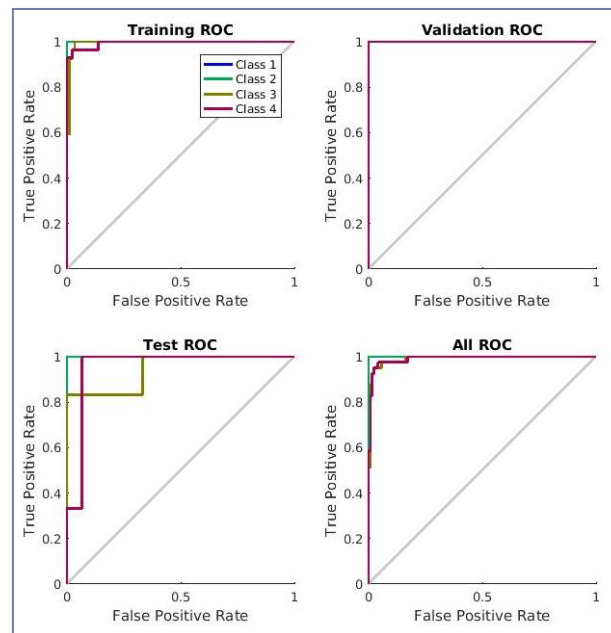


FIGURE 8 ROC PLOTS

3.2.1 – SMALLER NECESSARY INTERVAL

Now that we have described the procedure to produce the neural network, we can apply it for different time intervals, in order to find the least necessary time interval. There is not a “best” time interval because, obviously, it is relative to the performance we are asking to the neural network.

We run a script ([Appendix B.1](#)) that is able to choice the best number of hidden neurons for the neural network, by running 10 times the training with a number “ n ” of hidden neurons, and

computing the mean error as measure of performance. Than we choose the best, and use it to run 10 times the training. Among the latest network we choice the one with best performance.

The following **Table 1** shows the results obtained for every interval.

Intervals	Performances	Classes			
		Supine	Dorsiflexion	Walking	Stair climbing
3sec	Accuracy (ACC)	96,25	90,51	87,76	85,00
	Precision (PPV)	93,42	79,73	73,75	72,41
	Sensitivity (TPR)	91,19	83,72	79,71	64,56
	Specificity (TNR)	97,90	92,81	90,46	91,81
	Miss rate (FNR)	8,81	16,28	20,29	35,44
	Fall-out (FPR)	2,10	7,19	9,54	8,19
	Model Accuracy	79,77			
5sec	Accuracy (ACC)	97,27	93,79	89,17	86,29
	Precision (PPV)	94,50	88,34	77,17	73,21
	Sensitivity (TPR)	94,50	86,75	80,66	71,21
	Specificity (TNR)	98,19	96,15	92,01	91,31
	Miss rate (FNR)	5,50	13,25	19,34	28,79
	Fall-out (FPR)	1,81	3,85	7,99	8,69
	Model Accuracy	83,26			
10sec	Accuracy (ACC)	99,53	95,75	91,34	89,13
	Precision (PPV)	98,73	92,36	83,02	77,78
	Sensitivity (TPR)	99,36	90,62	82,50	79,25
	Specificity (TNR)	99,58	97,47	94,32	92,44
	Miss rate (FNR)	0,64	9,38	17,50	20,75
	Fall-out (FPR)	0,42	2,53	5,68	7,56
	Model Accuracy	87,87			
12sec	Accuracy (ACC)	98,46	95,97	93,09	91,36
	Precision (PPV)	96,15	92,31	88,00	81,62
	Sensitivity (TPR)	97,66	91,60	83,97	84,73
	Specificity (TNR)	98,73	97,44	96,15	93,59
	Miss rate (FNR)	2,34	8,40	16,03	15,27
	Fall-out (FPR)	1,27	2,56	3,85	6,41
	Model Accuracy	89,44			
15sec	Accuracy (ACC)	99,25	98,26	96,27	95,77
	Precision (PPV)	98,98	97,00	93,00	90,38
	Sensitivity (TPR)	97,98	96,04	92,08	93,07
	Specificity (TNR)	99,67	99,00	97,67	96,68
	Miss rate (FNR)	2,02	3,96	7,92	6,93
	Fall-out (FPR)	0,33	1,00	2,33	3,32
	Model Accuracy	94,78			
20sec	Accuracy (ACC)	99,65	98,94	96,47	96,47
	Precision (PPV)	100,00	97,22	94,20	91,78
	Sensitivity (TPR)	98,57	98,59	91,55	94,37

	Specificity (TNR)	100,00	99,06	98,11	97,17
	Miss rate (FNR)	1,43	1,41	8,45	5,63
	Fall-out (FPR)	0,00	0,94	1,89	2,83
	Model Accuracy	95,76			
30sec	Accuracy (ACC)	100,00	100,00	99,39	99,39
	Precision (PPV)	100,00	100,00	100,00	97,62
	Sensitivity (TPR)	100,00	100,00	97,56	100,00
	Specificity (TNR)	100,00	100,00	100,00	99,18
	Miss rate (FNR)	0,00	0,00	2,44	0,00
	Fall-out (FPR)	0,00	0,00	0,00	0,82
	Model Accuracy	99,39			
1min	Accuracy (ACC)	100,00	97,73	95,45	97,73
	Precision (PPV)	100,00	91,67	100,00	91,67
	Sensitivity (TPR)	100,00	100,00	81,82	100,00
	Specificity (TNR)	100,00	96,97	100,00	96,97
	Miss rate (FNR)	0,00	0,00	18,18	0,00
	Fall-out (FPR)	0,00	3,03	0,00	3,03
	Model Accuracy	95,45			

TABLE 1 PERFORMANCE OF NEURAL NETWORKS VERSUS VARIOUS TIME INTERVALS

It is noteworthy that with an interval large 20 seconds it is possible to reach an accuracy of about 95%. With an 30 seconds interval the accuracy rise to 99%, and the first 2 classes are classified without errors.

It is also important to notice how with the 1 minute long intervals the accuracy decrease, mainly due to the less number of signal pieces available to train the network.

APPENDIX A - DATA PRE-PROCESSING

A.1 INIT.M

```
1 %% 01 Data initialization - init.m
2 % - Data are grouped by activity/position and collected in a structure.
3 addpath('gitProjects/intelligent-system/');
4 load('ProjectWS.mat');
5
6 % This variable disable plots, so to fast computations.
7 showPlots = false;
8
9 % Neglect first row, it's just a row of zeros for every activity.
10 supine = {V01A(2:end,:), V02A(2:end,:), V03A(2:end,:), V04A(2:end,:), ...
11           V05A(2:end,:), V06A(2:end,:), V07A(2:end,:), V08A(2:end,:), ...
12           V09A(2:end,:), V10A(2:end,:)};
13
14 dorsiflexion = {V01B(2:end,:), V02B(2:end,:), V03B(2:end,:), ...
15               V04B(2:end,:), V05B(2:end,:), V06B(2:end,:), V07B(2:end,:), ...
16               V08B(2:end,:), V09B(2:end,:), V10B(2:end,:)};
17
18 walking = {V01C(2:end,:), V02C(2:end,:), V03C(2:end,:), V04C(2:end,:), ...
19           V05C(2:end,:), V06C(2:end,:), V07C(2:end,:), V08C(2:end,:), ...
20           V09C(2:end,:), V10C(2:end,:)};
21
22 stair = {V01D(2:end,:), V02D(2:end,:), V03D(2:end,:), V04D(2:end,:), ...
23         V05D(2:end,:), V06D(2:end,:), V07D(2:end,:), V08D(2:end,:), ...
24         V09D(2:end,:), V10D(2:end,:)};
25
26 % Build the structure.
27 Struct = struct('supine',supine,'dorsiflexion',dorsiflexion,'walking',...
28               walking,'stair',stair);
29
30 for i=1:10
31     % - Create a new signal as the sum of the three existing components
32     % and append at the other components.
33     Struct(i).supine = ...
34         [Struct(i).supine(:,4), Struct(i).supine(:,1:3), ...
35         sum(Struct(i).supine(:,1:3),2)];
36     Struct(i).dorsiflexion = ...
37         [Struct(i).dorsiflexion(:,4), Struct(i).dorsiflexion(:,1:3), ...
38         sum(Struct(i).dorsiflexion(:,1:3),2)];
39     Struct(i).walking = ...
40         [Struct(i).walking(:,4), Struct(i).walking(:,1:3), ...
41         sum(Struct(i).walking(:,1:3),2)];
42     Struct(i).stair = ...
43         [Struct(i).stair(:,4), Struct(i).stair(:,1:3), ...
44         sum(Struct(i).stair(:,1:3),2)];
45
46     % - Plot of each volunteer's activity/position signal and boxplot
47     % to better show patterns, time features, probability distribution.
48     if showPlots
49         % full screen figure
50         figure('units','normalized','outerposition',[0 0 1 1]);
51
52         subplot(2,3,1);
53         plot(Struct(i).supine(:,1),Struct(i).supine(:,2:5));
54         title(strcat('Supine - volunteer ',num2str(i)));
55         legend('sensor1','sensor2','sensor3','sum');
56         xlabel('time [s]');
57         ylabel('pressure [ohm]');
58
59         subplot(2,3,2);
60         plot(Struct(i).dorsiflexion(:,1),Struct(i).dorsiflexion(:,2:5));
61         title(strcat('Dorsiflexion - volunteer ',num2str(i)));
62         legend('sensor1','sensor2','sensor3','sum');
63         xlabel('time [s]');
64         ylabel('pressure [ohm]');
65
66         subplot(2,3,4);
67         plot(Struct(i).walking(:,1),Struct(i).walking(:,2:5));
68         title(strcat('Walking - volunteer ',num2str(i)));
```

```

69     legend('sensor1','sensor2','sensor3','sum');
70     xlabel('time [s]');
71     ylabel('pressure [ohm]');
72
73     subplot(2,3,5);
74     plot(Struct(i).stair(:,1),Struct(i).stair(:,2:5));
75     title(strcat('Stairs - volunteer ',num2str(i)));
76     legend('sensor1','sensor2','sensor3','sum');
77     xlabel('time [s]');
78     ylabel('pressure [ohm]');
79
80     subplot(2,3,[3 6]);
81     var = [Struct(i).supine(:,5) ' Struct(i).dorsiflexion(:,5)'...
82           Struct(i).walking(:,5) ' Struct(i).stair(:,5)'];
83     grp = [zeros(1,length(Struct(i).supine)), ...
84           ones(1,length(Struct(i).dorsiflexion)), ...
85           2.*ones(1,length(Struct(i).walking)), ...
86           3.*ones(1,length(Struct(i).stair))];
87     boxplot(var,grp,'Labels',{'supine','dorsiflexion','walking',...
88           'stair'});
89     end
90 end

```

A.2 SPLIT.M

```

1 %% 02 Data splitting - split.m
2 % - We need to split every signal to find out the least temporal interval
3 %   that is necessary and sufficient to recognize the position/activity.
4
5 % N.B. We are using a sampling period of ~82ms (12.2Hz).
6 % Number of samples based on the interval size:
7 % N1 = 12.2Hz * 3sec = 36.6;      %3sec
8 % N2 = 61;                       %5sec
9 % N3 = 122;                       %10sec
10 % N4 = 144.4;                     %12sec
11 % N5 = 183;                       %15sec
12 % N6 = 244;                       %20sec
13 % N7 = 366;                       %30sec
14 % N8 = 732;                       %1min
15 % N9 = 1464;                      %2min - not enough signals to train NN
16 % N = 2196;                       %3min - NO SIGNAL HAS A 3min LONG TRACE
17 N = [37 61 122 145 183 244 366 732];
18
19 % The index only need to use one choice among the upper showed.
20 index = 6;
21 clear A B C D;
22 A = Struct(1).supine(1:N(index), 2:5);
23 B = Struct(1).dorsiflexion(1:N(index), 2:5);
24 C = Struct(1).walking(1:N(index), 2:5);
25 D = Struct(1).stair(1:N(index), 2:5);
26
27 % If the piece of signal cannot fill entirely the number of samples
28 % needed, it wont be used:
29 % "length(Struct(i).<activity>)/N(j)" is used just for its integer part so
30 % to truncate incomplete pieces of signal.
31 for i=1:10
32     for k=2:length(Struct(i).supine)/N(index)
33         A = [A Struct(i).supine(1+N(index)*(k-1):k*N(index), 2:5)];
34     end
35     for k=2:length(Struct(i).dorsiflexion)/N(index)
36         B = [B Struct(i).dorsiflexion(1+N(index)*(k-1):k*N(index), 2:5)];
37     end
38     for k=2:length(Struct(i).walking)/N(index)
39         C = [C Struct(i).walking(1+N(index)*(k-1):k*N(index), 2:5)];
40     end
41     for k=2:length(Struct(i).stair)/N(index)
42         D = [D Struct(i).stair(1+N(index)*(k-1):k*N(index), 2:5)];
43     end
44 end
45
46 if showPlots
47     figure, plot(0.082*(1:size(A,1)),A(:,1:4))
48     title('Volunteer 1 - Supine (Slice 1)')
49     xlabel('time [s]')
50     ylabel('pressure [ohm]')
51     legend('sensor 1', 'sensor 2', 'sensor 3', 'sensor virt. (sum)')
52 end

```

A.3 CLEANING.M

```

1 %% 03 Data cleaning - cleaning.m
2 % Clean the signal from noise by mean of a smoothing filter
3 % (Savitzky-Golay Filter), sgolayfilt(X,K,F):
4 % - K=3, third-order polynomial;
5 % - F=7, just an odd value greater than the piece of signal;
6
7 clear smoothA;
8 smoothA = sgolayfilt(A,3,7);
9 if showPlots
10     figure, plot(0.082*(1:size(smoothA,1)),[A(:,1) smoothA(:,1)])
11     title('Volunteer 1 - Supine (Slice 1)');
12     xlabel('time [s]');
13     ylabel('pressure [ohm]');
14     legend('sensor 1 - original', 'sensor 1 - smooth');
15 end
16
17 clear smoothB;
18 smoothB = sgolayfilt(B,3,7);
19
20 clear smoothC;
21 smoothC = sgolayfilt(C,3,7);
22
23 clear smoothD;
24 smoothD = sgolayfilt(D,3,7);
25
26 % Use Z-score normalization for each signal, in order to be able to compare
27 % signals.
28 smoothA = zscore(smoothA);
29 smoothB = zscore(smoothB);
30 smoothC = zscore(smoothC);
31 smoothD = zscore(smoothD);
32
33 if showPlots
34     figure, plot(0.082*(1:size(smoothA)),smoothA(:,1:4))
35     title('Volunteer 1 - Supine (Slice 1)');
36     xlabel('time [s]');
37     ylabel('pressure [ohm]');
38     legend('sensor 1', 'sensor 2', 'sensor 3', 'sensor 4');
39 end

```

A.4 FEATUREEX.M

```

1  %% 04 Feature extraction - featureEx.m
2  % Obtain a set of temporal feature starting from the N(j) samples of the
3  % signal.
4  clear fftA fftB fftC fftD;
5
6  clear featuresA;
7  featuresA = [max(smoothA); min(smoothA); skewness(smoothA); ...
8             kurtosis(smoothA)];
9  clear featuresB;
10 featuresB = [max(smoothB); min(smoothB); skewness(smoothB); ...
11            kurtosis(smoothB)];
12 clear featuresC;
13 featuresC = [max(smoothC); min(smoothC); skewness(smoothC); ...
14            kurtosis(smoothC)];
15 clear featuresD;
16 featuresD = [max(smoothD); min(smoothD); skewness(smoothD); ...
17            kurtosis(smoothD)];
18
19 % Autocorrelation: computed at lags 0,1,2, ... T= min[20,length(y)-1]
20 clear RxxA;
21 RxxA = my_autocorr(smoothA); % See my_autocorr.m
22 clear RxxB;
23 RxxB = my_autocorr(smoothB);
24 clear RxxC;
25 RxxC = my_autocorr(smoothC);
26 clear RxxD;
27 RxxD = my_autocorr(smoothD);
28
29 if showPlots
30     figure('units','normalized','outerposition',[0 0 1 1]);
31
32     subplot(2,2,1);
33     plot(0:20,RxxA);
34     title('Supine - All the volunteers, all the autocorrelations');
35     xlabel('lags');
36     ylabel('Rxx');
37
38     subplot(2,2,2);
39     plot(0:20,RxxB);
40     title('Dorsiflexion - All the volunteers, all the autocorrelations');
41     xlabel('lags');
42     ylabel('Rxx');
43
44     subplot(2,2,3);
45     plot(0:20,RxxC);
46     title('Walk - All the volunteers, all the autocorrelations');
47     xlabel('lags');
48     ylabel('Rxx');
49
50     subplot(2,2,4);
51     plot(0:20,RxxD);
52     title('Stairs - All the volunteers, all the autocorrelations');
53     xlabel('lags');
54     ylabel('Rxx');
55 end
56
57 % first value, autocorr with lags=0 is always equal to 1
58 % (it's irrelevant)
59 featuresA = [featuresA; RxxA(2:end,:)];
60 featuresB = [featuresB; RxxB(2:end,:)];
61 featuresC = [featuresC; RxxC(2:end,:)];
62 featuresD = [featuresD; RxxD(2:end,:)];
63
64 % Frequential features:
65 % - Fundamental frequency f0
66 % - Power Spectral Density PSD
67
68 % Define the frequency domain
69 f = 12.2*(0:N(index)/2-1);
70
71 % Compute the single-sided spectrum
72 fftA = my_fft(smoothA,N(index)); % See my_fft.m
73 fftB = my_fft(smoothB,N(index));
74 fftC = my_fft(smoothC,N(index));

```

```

75 fftD = my_fft(smoothD,N(index));
76
77 if(showPlots)
78     figure('units','normalized','outerposition',[0 0 1 1]);
79
80     subplot(2,2,1);
81     plot(f,fftA);
82     title('Supine - All the volunteers, all the spectra');
83     xlabel('frequency [Hz]');
84     ylabel('FFT');
85
86     subplot(2,2,2);
87     plot(f,fftB);
88     title('Dorsiflexion - All the volunteers, all the spectra');
89     xlabel('frequency [Hz]');
90     ylabel('FFT');
91
92     subplot(2,2,3);
93     plot(f,fftC);
94     title('Walk - All the volunteers, all the spectra');
95     xlabel('frequency [Hz]');
96     ylabel('FFT');
97
98     subplot(2,2,4);
99     plot(f,fftD);
100    title('Stairs - All the volunteers, all the spectra');
101    xlabel('frequency [Hz]');
102    ylabel('FFT');
103 end
104
105 % The max amplitude should be a good approximation for the fundamental
106 % frequency
107 [amp, x] = max(fftA);
108 [~, x2] = findpeaks(fftA, 'MinPeakProminence', 0.7*max(fftA));
109
110 % PSD
111 PSD = sum(fftA.^2);
112 % figure
113 % plot(f,fftA(:, [1 11 111]));
114 % hold on;
115 % plot(f(x([1 11 111])),y([1 11 111]),'rv');
116 featuresA = [featuresA; f(x); amp; PSD];
117
118 clear amp x;
119 [amp, x] = max(fftB);
120 featuresB = [featuresB; f(x); amp; sum(fftB.^2)];
121 clear amp x;
122 [amp, x] = max(fftC);
123 featuresC = [featuresC; f(x); amp; sum(fftC.^2)];
124 clear amp x;
125 [amp, x] = max(fftD);
126 featuresD = [featuresD; f(x); amp; sum(fftD.^2)];
127
128 %% Rotate matrix
129 % Since we use 4 columns to represent 3 sensor signals + 1 "virtual"
130 % sensor (the sum), we move all the features of the same piece of signal
131 % on the same column.
132 newFeaturesA = rotate_features(featuresA);
133 newFeaturesB = rotate_features(featuresB);
134 newFeaturesC = rotate_features(featuresC);
135 newFeaturesD = rotate_features(featuresD);

```

A.5 MY_AUTOCORR.M

```

1 % My extension of autocorr(), in order to compute autocorr on a matrix of
2 % more than 1 signal, where each column is a different signal
3 function Rxx = my_autocorr (matrix)
4     % If "matrix" has only a column, I only need to compute correlation once
5     Rxx = autocorr(matrix(:,1));
6
7     for k=2:size(matrix,2)
8         Rxx = [Rxx autocorr(matrix(:,k))];
9     end
10 end

```

A.6 MY_FFT.M

```

1 function f = my_fft (matrix, ind)
2     if(mod(ind,2)~=0), ind = ind-1; end
3     f = fft(matrix);
4     f = abs(f./ind);
5     f(ind/2+1:end, :) = [];
6     f= f.*2;
7 end

```

A.7 FEATURESEL.M

```

1 %% 05 Feature Selection
2 % Sequential feature selection
3
4 num_features = 6;
5
6 % Classify each activity with a different class:
7 % Y = 0 => supine
8 % Y = 1 => dorsiflexion
9 % Y = 2 => walking
10 % Y = 3 => stairs
11 sizeA = size(newFeaturesA,2);
12 sizeB = size(newFeaturesB,2);
13 sizeC = size(newFeaturesC,2);
14 sizeD = size(newFeaturesD,2);
15 X = [newFeaturesA    newFeaturesB    newFeaturesC    newFeaturesD]';
16 Y = [zeros(sizeA,1); ones(sizeB,1); 2*ones(sizeC,1); 3*ones(sizeD,1)];
17
18 f = @(xtrain, ytrain, xtest, ytest) ...
19     sum(ytest ~= classify(xtest, xtrain, ytrain));
20 if showPlots
21     opts = statset('display','iter');
22     [fs, history] = sequentialfs(f,X,Y,'nfeatures',num_features,...
23         'options',opts);
24 else
25     [fs, history] = sequentialfs(f,X,Y,'nfeatures',num_features);
26 end

```

APPENDIX B – NEURAL NETWORK

B.1 NN.M

```
1 %% 06 Neural network
2
3 % "What's the best number of hidden neurons?"
4 inputs = X(:,fs)';
5 targets = zeros(4,size(Y,1));
6 targets(1,1:sizeA) = 1;
7 targets(2,sizeA+1:sizeA+sizeB) = 1;
8 targets(3,sizeA+sizeB+1:sizeA+sizeB+sizeC) = 1;
9 targets(4,sizeA+sizeB+sizeC+1:end) = 1;
10
11 n1 = 1; % lowest number of hidden neurons
12 n2 = 10; % highest number of hidden neurons
13 performances = zeros(10,1);
14 regressions = zeros(10,4);
15 meanPerformance = zeros(n2-n1+1,1);
16 meanRegression = zeros(n2-n1+1,4);
17
18 for n = n1:1:n2,
19     % Create a Pattern Recognition Network
20     hiddenLayerSize = n;
21
22     for k=1:10,
23         net = patternnet(hiddenLayerSize);
24         % Setup Division of Data for Training, Validation, Testing
25         net.divideParam.trainRatio = 70/100;
26         net.divideParam.valRatio = 15/100;
27         net.divideParam.testRatio = 15/100;
28
29         % hide window: speed up computations
30         net.trainParam.showWindow = false;
31
32         % Train the Network
33         [net,~] = train(net,inputs,targets);
34
35         % Test the Network
36         outputs = net(inputs);
37         performances(k) = perform(net,targets,outputs);
38         [regressions(k,:),~,~] = regression(targets,outputs);
39     end
40     meanRegression(n,:) = mean(regressions);
41     meanPerformance(n) = mean(performances);
42 end
43
44 if showPlots
45     figure, plot(n1:n2, meanPerformance, 'r-o');
46     title('MSE: less is better');
47     ylabel('mean square error');
48     xlabel('# of hidden neurons');
49     legend('mean performance');
50
51     figure, plot(n1:n2, meanRegression, 'g-o');
52     title('R-value: correlation between output and targets');
53     ylabel('Regression coefficient');
54     xlabel('# of hidden neurons');
55     legend('mean regression');
56 end
57
58 %% Train the neural network
59 % Best among the networks with chosen number of hidden neurons
60 [~,hiddenLayerSize] = min(meanPerformance);
61 p = inf;
62 for k=1:10,
63     net_temp = patternnet(hiddenLayerSize);
64     % Setup Division of Data for Training, Validation, Testing
65     net_temp.divideParam.trainRatio = 70/100;
66     net_temp.divideParam.valRatio = 15/100;
67     net_temp.divideParam.testRatio = 15/100;
```



```

69 % hide window: speed up computations
70 net_temp.trainParam.showWindow = false;
71
72 % Train the Network
73 [net_temp,tr_temp] = train(net_temp,inputs,targets);
74
75 % Test the Network
76 outputs = net_temp(inputs);
77 p_temp = perform(net_temp,targets,outputs);
78 if(p_temp < p),
79     best_net = net_temp;
80     p = p_temp;
81     tr = tr_temp;
82 end
83 end
84
85 %% Print evaluations
86
87 outputs = best_net(inputs);
88 errors = gsubtract(targets,outputs);
89 performance = perform(best_net,targets,outputs);
90
91 % View the Network
92 % view(best_net)
93
94 % Plots
95 % figure, plotperform(tr)
96 % figure, plottrainstate(tr)
97 % figure, plotconfusion(targets,outputs)
98 % figure, ploterrhist(errors)
99
100 [actual,~,~] = find(targets);
101 [~,predict] = max(outputs);
102 cfmatrix2(actual',predict,[1 2 3 4], 1, 1);

```

B.2 CFMATRIX2.M

```

1 %% https://it.mathworks.com/matlabcentral/fileexchange/21212-confusion-matrix---matching-
2 matrix-along-with-precision--sensitivity--specificity-and-model-accuracy
3
4 function [confmatrix] = cfmatrix2 ...
5     (actual, predict, classlist, per, printout)
6 % CFMATRIX2 calculates the confusion matrix for any prediction
7 % algorithm ( prediction algorithm generates a list of classes to which
8 % each test feature vector is assigned );
9 %
10 % Outputs: confusion matrix
11 %
12 %           Actual Classes
13 %           p           n
14 %   _____|_____|_____
15 %   Predicted p'|_____|_____
16 %   Classes  n'|       |       |
17 %
18 %           Also the TP, FP, FN and TN are output for each class based
19 %           on http://en.wikipedia.org/wiki/Confusion_matrix
20 %           The Precision, Sensitivity and Specificity for each class
21 %           have also been added in this update along with the overall
22 %           accuracy of the model ( ModelAccuracy ).
23 ...
24 % If classlist not entered: make classlist equal to all
25 % unique elements of actual
26 if (nargin < 2)
27     error('Not enough input arguments. Need atleast two vectors as input');
28 elseif (nargin == 2)
29     classlist = unique(actual); % default values from actual
30     per = 0;
31     printout = 1;
32 elseif (nargin == 3)
33     per = 0; % default is numbers and input 1 or higher for percentage
34     printout = 1;
35 elseif (nargin == 4)
36     printout = 1; % default is silent output ( 0 ); one or higher printsout
37 elseif (nargin > 5)

```

```

86     error('Too many input arguments. ');
87 end
88
89
90 if (length(actual) ~= length(predict))
91     error('First two inputs need to be vectors with equal size. ');
92 elseif ((size(actual,1) ~= 1) && (size(actual,2) ~= 1))
93     error('First input needs to be a vector and not a matrix');
94 elseif ((size(predict,1) ~= 1) && (size(predict,2) ~= 1))
95     error('Second input needs to be a vector and not a matrix');
96 end
97 format short g;
98 n_class = length(classlist);
99 confmatrix = zeros(n_class);
100 line_two = '-----';
101 line_three = '_____|';
102
103 for i = 1:n_class
104     for j = 1:n_class
105         m = (predict == classlist(i) ...
106             & actual == classlist(j));
107         confmatrix(i,j) = sum(m);
108     end
109     line_two = strcat(line_two,'---',num2str(classlist(i)),'-----');
110     line_three = strcat(line_three,'_____|');
111 end
112
113 TPFNFNTN = zeros(4, n_class);
114 Accuracy = zeros(1, n_class);
115 Precision = zeros(1, n_class);
116 Sensitivity = zeros(1, n_class);
117 Specificity = zeros(1, n_class);
118 MissRate = zeros(1, n_class);
119 Fall_Out = zeros(1, n_class);
120
121 temps1 = sprintf('    TP  ');
122 temps2 = sprintf('    FP  ');
123 temps3 = sprintf('    FN  ');
124 temps4 = sprintf('    TN  ');
125 temps5 = sprintf('Accur. ');
126 temps6 = sprintf('Preci. ');
127 temps7 = sprintf('Sensi. ');
128 temps8 = sprintf('Speci. ');
129 temps9 = sprintf('MissR. ');
130 temps10 = sprintf('FallO. ');
131
132 for i = 1:n_class
133     % TP
134     TPFNFNTN(1, i) = confmatrix(i,i);
135     temps1 = strcat(temps1,sprintf(' | %3d \t',TPFNFNTN(1, i)));
136
137     % FP
138     TPFNFNTN(2, i) = sum(confmatrix(i,:))-confmatrix(i,i);
139     temps2 = strcat(temps2,sprintf(' | %3d \t',TPFNFNTN(2, i)));
140
141     % FN
142     TPFNFNTN(3, i) = sum(confmatrix(:,i))-confmatrix(i,i);
143     temps3 = strcat(temps3,sprintf(' | %3d \t',TPFNFNTN(3, i)));
144
145     % TN
146     TPFNFNTN(4, i) = sum(confmatrix(:)) - sum(confmatrix(i,:)) - ...
147         sum(confmatrix(:,i)) + confmatrix(i,i);
148     temps4 = strcat(temps4,sprintf(' | %3d \t',TPFNFNTN(4, i)));
149
150     % Accuracy(class) = (TP(class)+TN(class))/all
151     Accuracy(i) = (TPFNFNTN(1, i)+TPFNFNTN(4, i))/sum(confmatrix(:))*100;
152     temps5 = strcat(temps5,sprintf(' | %3.2f \t',Accuracy(i)));
153
154     % Precision(class) = TP(class) / ( TP(class) + FP(class) )
155     Precision(i) = TPFNFNTN(1, i) / sum(confmatrix(i,:))*100;
156     temps6 = strcat(temps6,sprintf(' | %3.2f \t',Precision(i)));
157
158     % Sensitivity(class) = Recall(class) = TruePositiveRate(class)
159     % = TP(class) / ( TP(class) + FN(class) )
160     Sensitivity(i) = TPFNFNTN(1, i) / sum(confmatrix(:,i))*100;
161     temps7 = strcat(temps7,sprintf(' | %3.2f \t',Sensitivity(i)));

```

```

162
163 % Specificity ( mostly used in 2 class problems )=
164 % TrueNegativeRate(class)
165 % = TN(class) / ( TN(class) + FP(class) )
166 Specificity(i) = TPFNFNTN(4, i) / ( TPFNFNTN(4, i) + TPFNFNTN(2, i) ) * 100;
167 temps8 = strcat(temps8, sprintf(' | %3.2f \t', Specificity(i) ));
168
169 % Miss rate = FN(class) / ( TP(class) + FN(class) )
170 MissRate(i) = TPFNFNTN(3, i) / sum(confmatrix(:, i)) * 100;
171 temps9 = strcat(temps9, sprintf(' | %3.2f \t', MissRate(i) ));
172
173 % Fall-out = FP(class) / ( TN(class) + FP(class) )
174 Fall_Out(i) = TPFNFNTN(2, i) / ( TPFNFNTN(4, i) + TPFNFNTN(2, i) ) * 100;
175 temps10 = strcat(temps10, sprintf(' | %3.2f \t', Fall_Out(i) ));
176
177 end
178
179 ModelAccuracy = sum(diag(confmatrix)) / sum(confmatrix(:)) * 100;
180 temps11 = sprintf('Model Accuracy is %1.2f ', ModelAccuracy);
181
182 if (per > 0) % ( if > 0 implies true; < 0 implies false )
183     confmatrix = (confmatrix ./ length(actual)) * 100;
184 end
185
186 if ( printout > 0 ) % ( if > 0 printout; < 0 no printout )
187     disp('-----');
188     disp('          Actual Classes');
189     disp(line_two);
190     disp('Predicted|          ');
191     disp('  Classes|          ');
192     disp(line_three);
193
194     for i = 1:n_class
195         temps = sprintf('          %d          ', i);
196         for j = 1:n_class
197             temps = strcat(temps, sprintf(' | %3.1f ', confmatrix(i, j)));
198         end
199         disp(temps);
200         clear temps
201     end
202     disp('-----');
203
204     disp('-----');
205     disp('          Actual Classes');
206     disp(line_two);
207     disp(temps1); disp(temps2); disp(temps3); disp(temps4);
208     disp(temps5); disp(temps6); disp(temps7); disp(temps8);
209     disp(temps9); disp(temps10);
210     disp('-----');
211     disp(temps11);
212     disp('-----');
213 end
214 clear temps1 temps2 temps3 temps4 temps5 temps6 temps7 temps8 temps9 temps10 temps11

```