Intelligent Systems

Project

Developing of a neural system, a Mamdani-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

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.

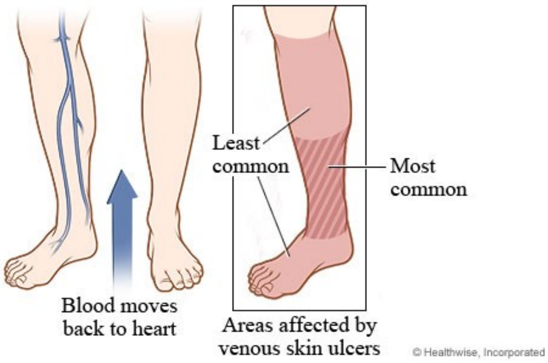

FIGURE 1 - COMPRESSION THERAPY

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

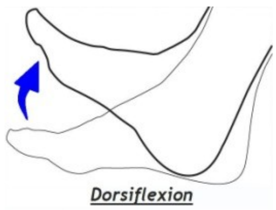

FIGURE 2 - DORSIFLEXION ACTIVITY

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.

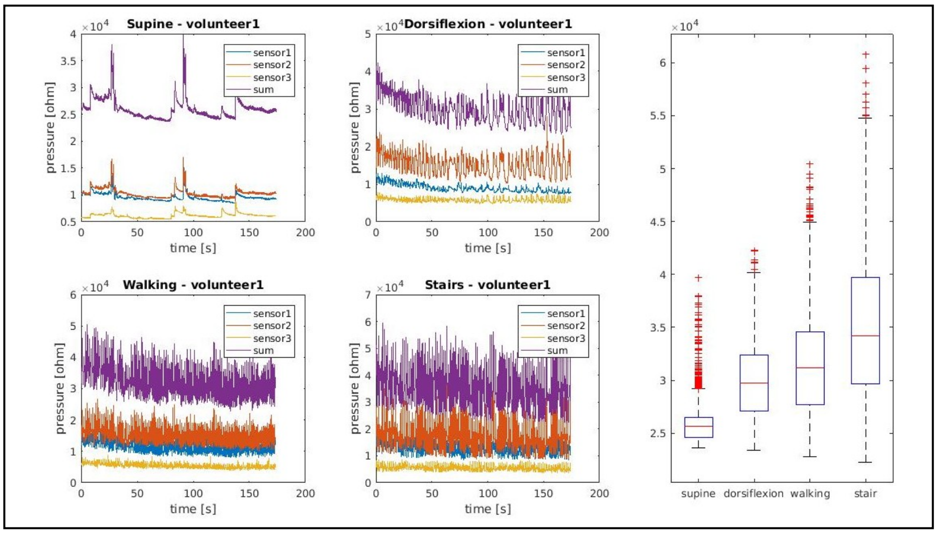

FIGURE 1 *SIGNALS AND BOXPLOTS OF POSITION/ACTIVITIES MEASURED ON VOLUNTEER 1*

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:

$$N = 12.2Hz \times M\,sec$$

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.

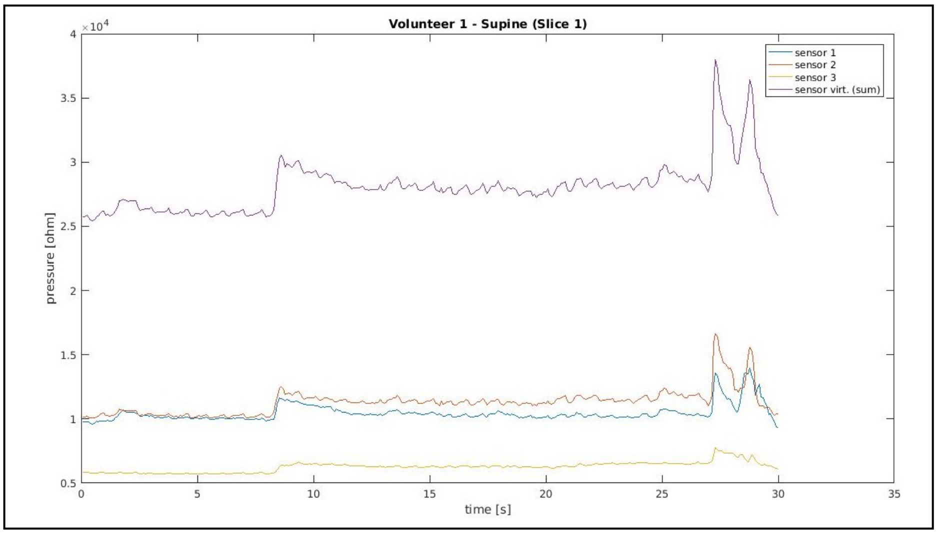

FIGURE 2 *FIRST 30-SECONDS-LONG SLICE OF THE SUPINE SIGNALS FROM VOLUNTEER 1*

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*).

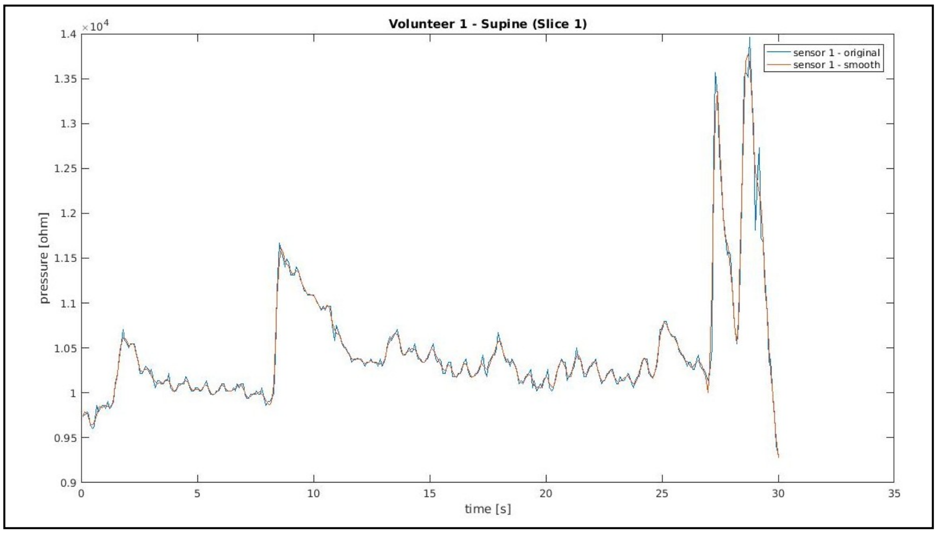

FIGURE 3 *SAVITZKY-GOLAY FILTER APPLIED TO A SIGNAL SLICE*

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 belove:

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

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

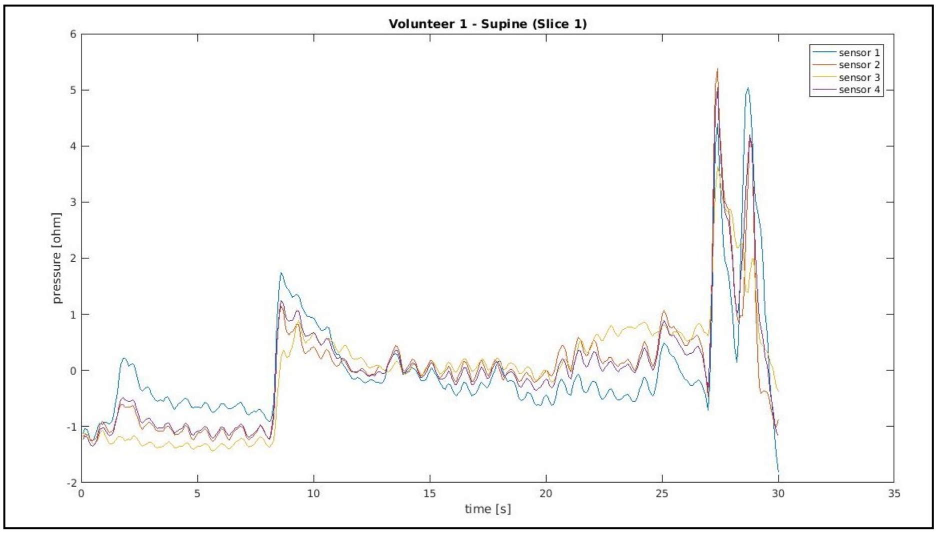

FIGURE 4 *NORMALIZED SIGNALS*

2.3 – FEATURE EXTRACTION

The signals, as whole, are quite difficult to analyze, than our goal is to represent them in terms of a reduced set of features. 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 significatively 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 (Δ=1,2,…,20).

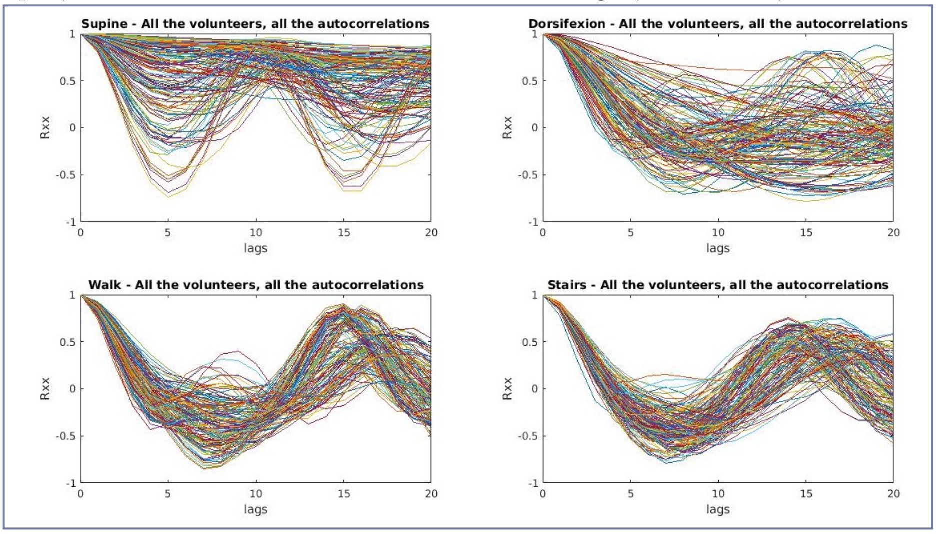

FIGURE 5 *AUTOCORRELATIONS FOR ALL THE ACTIVITIES OF EVERY VOLUNTEER*

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

- **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,
$$X_n = \sum_{k=0}^{N-1} x_k e^{-i\frac{2\pi}{N}kq} \ \ (\mathrm{q} = 0,1,\dots,\mathrm{N}-1)$$
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,
$$P_X = \sum_{n=-\infty}^{+\infty} |X_n|^2.$$

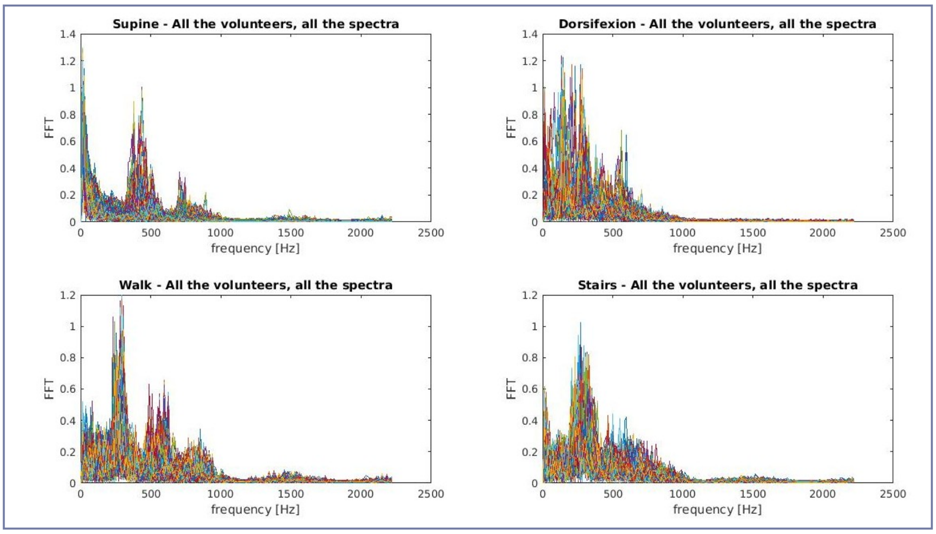

FIGURE 6 *FAST FOURIER TRANSFORMS FOR EVERY VOLUNTEER SEPARATED ON ACTIVITY/POSITION*

The feature extraction phase is showed in the *Appendix A.4*.

2.4 – FEATURE SELECTION

So far we have obtained 29 features: max, min, first and third quartile, 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 116 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 s ***Figure 8***). Some of the information we can scavenge from these diagrams are:

- **Accuracy (ACC)**: Overall, how often is the classifier correct?
  - $\frac{TP+TN}{P+N}$

- **Precision (PPV)**: How many times the classifier is correct in predicting a class over its overall predictions of that class?
    - $\frac{TP}{TP+FP}$
- **Sensitivity** or Recall **(TPR)**: How many times the classifier predicts the correct class among all the overall actual occurrences of that class?
    - $\frac{TP}{TP+FN}$
- **Miss rate (FNR)**: How many times the classifier does not predict a class?
    - $\frac{FN}{TP+FN} = 1 - TPR$
- **Specificity (TNR)**: When the classifier is correct at not classify an input as a class over the whole not-members of that class?
    - $\frac{TN}{TN+FP}$
- **Fall-out** (false alarm): When the classifier predicts a class, wrongly?
    - $\frac{FP}{TN+FP}$.

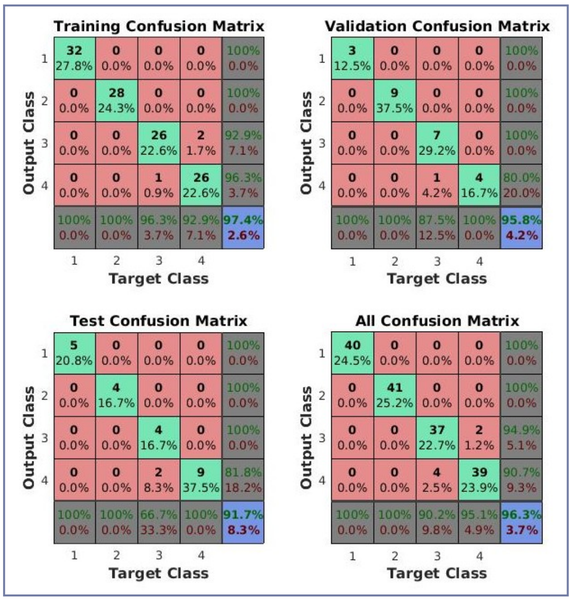

FIGURE 7 *CONFUSION MATRICES*

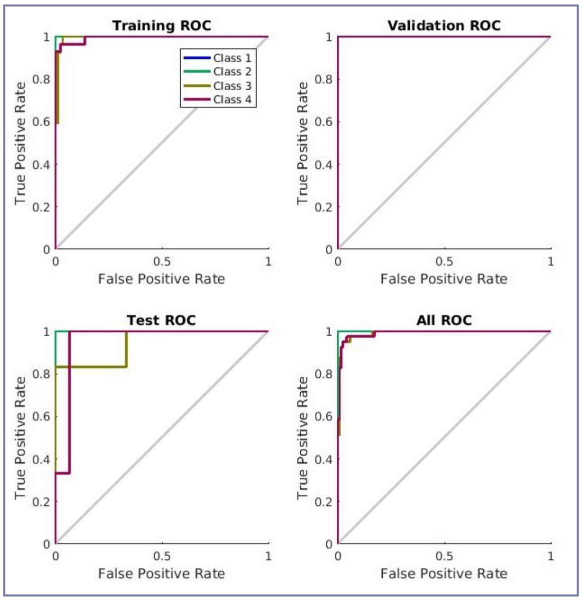

FIGURE 8 *ROC PLOTS*

3.2.1 – SMALLER NECESSARY TIME 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. Then 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 VARING TIME INTERVALS*

It is noteworthy that with an interval large 20 seconds it is possible to reach an accuracy of about 95%. With a 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. This can be attributed to the lower number of signal pieces available to train the network.

4 – FUZZY INFERENCE SYSTEM

In this chapter we develop a Mamdani-type Fuzzy Inference System (FIS) with the help of the MATLAB *Fuzzy Logic Toolbox Graphical User Interface Tools*.

4.1 – FUZZY INFERENCE SYSTEM

For the definition of the problem, we first run the command "`fuzzy`" on MATLAB, in order to start the *FIS editor*.

From the showing GUI we introduce inputs and outputs of the FIS.

4.1.1 – FEATURE SELECTION

Since, this time, the choice of the features lies on our ability to write down the membership functions, we prefer to plot the Probability Mass Function (PMF) of each feature, so that it's easier for us to take the features in which different activities differ the most.

From the plots we have chosen the following features: .

[PMF of the chosen features].

The resulting system is a 4inputs-1output.

[FIS system]

4.2 – MEMBERSHIP FUNCTIONS

From the PMFs chosen in the previous paragraph, we use the *Membership Function Editor* to write following membership functions.

4.2.1 – FEATURE

LINGUISTIC VARIABLES

[MF1]

4.2.1 – FEATURE

LINGUISTIC VARIABLES

[MF2]

4.2.1 – FEATURE

LINGUISTIC VARIABLES

[MF3]

4.2.1 – FEATURE

LINGUISTIC VARIABLES

[MF4]

4.3 – RULES

Now we can write the rules that our FIS have to use in order to infer the desired output. In this regard the *Rule Editor* helps us.

The rules we are going to use are the following:

- A
- A
- A
- A

[Rule Editor]

APPENDIX A – DATA PRE-PROCESSING

A.1 INIT.M

```
%% 01 Data initialization - init.m
% - Data are grouped by activity/position and collected in a structure.
addpath('gitProjects/intelligent-system/');
load('ProjectWS.mat');

% This variable disable plots, so to fast computations.
showPlots = false;

% Neglect first row, it's just a row of zeros for every activity.
supine = {V01A(2:end,:), V02A(2:end,:), V03A(2:end,:), V04A(2:end,:),...
    V05A(2:end,:), V06A(2:end,:), V07A(2:end,:), V08A(2:end,:), ...
    V09A(2:end,:), V10A(2:end,:)};

dorsiflexion = {V01B(2:end,:), V02B(2:end,:), V03B(2:end,:), ...
    V04B(2:end,:), V05B(2:end,:), V06B(2:end,:), V07B(2:end,:), ...
    V08B(2:end,:), V09B(2:end,:), V10B(2:end,:)};

walking = {V01C(2:end,:), V02C(2:end,:), V03C(2:end,:), V04C(2:end,:),...
    V05C(2:end,:), V06C(2:end,:), V07C(2:end,:), V08C(2:end,:), ...
    V09C(2:end,:), V10C(2:end,:)};

stair = {V01D(2:end,:), V02D(2:end,:), V03D(2:end,:), V04D(2:end,:),...
    V05D(2:end,:), V06D(2:end,:), V07D(2:end,:), V08D(2:end,:), ...
    V09D(2:end,:), V10D(2:end,:)};

% Build the structure.
Struct = struct('supine',supine,'dorsiflexion',dorsiflexion,'walking',...
    walking,'stair',stair);

for i=1:10
    % - Create a new signal as the sum of the three existing components
    %   and append at the other components.
    Struct(i).supine = ...
        [Struct(i).supine(:,4), Struct(i).supine(:,1:3), ...
        sum(Struct(i).supine(:,1:3),2)];
    Struct(i).dorsiflexion = ...
        [Struct(i).dorsiflexion(:,4), Struct(i).dorsiflexion(:,1:3), ...
        sum(Struct(i).dorsiflexion(:,1:3),2)];
    Struct(i).walking = ...
        [Struct(i).walking(:,4), Struct(i).walking(:,1:3), ...
        sum(Struct(i).walking(:,1:3),2)];
    Struct(i).stair = ...
        [Struct(i).stair(:,4), Struct(i).stair(:,1:3), ...
        sum(Struct(i).stair(:,1:3),2)];

    % - Plot of each volunteer's activity/position signal and boxplot
    %   to better show patterns, time features, probability distribution.
    if showPlots
        % full screen figure
        figure('units','normalized','outerposition',[0 0 1 1]);

        subplot(2,3,1);
        plot(Struct(i).supine(:,1),Struct(i).supine(:,2:5));
        title(strcat('Supine - volunteer ',num2str(i)));
        legend('sensor1','sensor2','sensor3','sum');
        xlabel('time [s]');
        ylabel('pressure [ohm]');

        subplot(2,3,2);
        plot(Struct(i).dorsiflexion(:,1),Struct(i).dorsiflexion(:,2:5));
        title(strcat('Dorsiflexion - volunteer ',num2str(i)));
        legend('sensor1','sensor2','sensor3','sum');
        xlabel('time [s]');
        ylabel('pressure [ohm]');

        subplot(2,3,4);
        plot(Struct(i).walking(:,1),Struct(i).walking(:,2:5));
        title(strcat('Walking - volunteer ',num2str(i)));
```

```
        legend('sensor1','sensor2','sensor3','sum');
        xlabel('time [s]');
        ylabel('pressure [ohm]');

        subplot(2,3,5);
        plot(Struct(i).stair(:,1),Struct(i).stair(:,2:5));
        title(strcat('Stairs - volunteer ',num2str(i)));
        legend('sensor1','sensor2','sensor3','sum');
        xlabel('time [s]');
        ylabel('pressure [ohm]');

        subplot(2,3,[3 6]);
        var = [Struct(i).supine(:,5)' Struct(i).dorsiflexion(:,5)'...
            Struct(i).walking(:,5)' Struct(i).stair(:,5)'];
        grp = [zeros(1,length(Struct(i).supine)), ...
            ones(1,length(Struct(i).dorsiflexion)), ...
            2.*ones(1,length(Struct(i).walking)), ...
            3.*ones(1,length(Struct(i).stair))];
        boxplot(var,grp,'Labels',{'supine','dorsiflexion','walking',...
            'stair'});
    end
end
```

A.2 SPLIT.M

```
%% 02 Data splitting - split.m
% - We need to split every signal to find out the least temporal interval
%   that is necessary and sufficient to recognize the position/activity.

% N.B. We are using a sampling period of ~82ms (12.2Hz).
% Number of samples based on the interval size:
% N1 = 12.2Hz * 3sec = 36.6;    %3sec
% N2 = 61;                      %5sec
% N3 = 122;                  %10sec
% N4 = 144.4;                %12sec
% N5 = 183;                     %15sec
% N6 = 244;                          %20sec
% N7 = 366;                          %30sec
% N8 = 732;                          %1min
% N9 = 1464;                    %2min - not enough signals to train NN
% N = 2196;                     %3min - NO SIGNAL HAS A 3min LONG TRACE
N = [37 61 122 145 183 244 366 732];

% The index only need to use one choice among the upper showed.
index = 6;
clear A B C D;
A = Struct(1).supine(1:N(index), 2:5);
B = Struct(1).dorsiflexion(1:N(index), 2:5);
C = Struct(1).walking(1:N(index), 2:5);
D = Struct(1).stair(1:N(index), 2:5);

% If the piece of signal cannot fill entirely the number of samples
% needed, it wont be used:
% "length(Struct(i).<activity>)/N(j)" is used just for its integer part so
% to truncate incomplete pieces of signal.
for i=1:10
    for k=2:length(Struct(i).supine)/N(index)
        A = [A Struct(i).supine(1+N(index)*(k-1):k*N(index), 2:5)];
    end
    for k=2:length(Struct(i).dorsiflexion)/N(index)
        B = [B Struct(i).dorsiflexion(1+N(index)*(k-1):k*N(index), 2:5)];
    end
    for k=2:length(Struct(i).walking)/N(index)
        C = [C Struct(i).walking(1+N(index)*(k-1):k*N(index), 2:5)];
    end
    for k=2:length(Struct(i).stair)/N(index)
        D = [D Struct(i).stair(1+N(index)*(k-1):k*N(index), 2:5)];
    end
end

if showPlots
    figure, plot(0.082*(1:size(A,1)),A(:,1:4))
    title('Volunteer 1 - Supine (Slice 1)')
    xlabel('time [s]')
    ylabel('pressure [ohm]')
    legend('sensor 1', 'sensor 2', 'sensor 3', 'sensor virt. (sum)')
end
```

A.3 CLEANING.M

```
%% 03 Data cleaning - cleaning.m
% Clean the signal from noise by mean of a smoothing filter
% (Savitzky-Golay Filter), sgolayfilt(X,K,F):
%  - K=3, third-order polynomial;
%  - F=7, just an odd value greater than the piece of signal;

clear smoothA;
smoothA = sgolayfilt(A,3,7);
if showPlots
    figure, plot(0.082*(1:size(smoothA,1)),[A(:,1) smoothA(:,1)])
    title('Volunteer 1 - Supine (Slice 1)');
    xlabel('time [s]');
    ylabel('pressure [ohm]');
    legend('sensor 1 - original', 'sensor 1 - smooth');
end

clear smoothB;
smoothB = sgolayfilt(B,3,7);

clear smoothC;
smoothC = sgolayfilt(C,3,7);

clear smoothD;
smoothD = sgolayfilt(D,3,7);

% Use Z-score normalization for each signal, in order to be able to compare
% signals.
smoothA = zscore(smoothA);
smoothB = zscore(smoothB);
smoothC = zscore(smoothC);
smoothD = zscore(smoothD);

if showPlots
    figure, plot(0.082*(1:size(smoothA)),smoothA(:,1:4))
    title('Volunteer 1 - Supine (Slice 1)');
    xlabel('time [s]');
    ylabel('pressure [ohm]');
    legend('sensor 1', 'sensor 2', 'sensor 3', 'sensor 4');
end
```

A.4 FEATUREEX.M

```
%% 04 Feature extraction - featureEx.m
% Obtain a set of temporal feature starting from the N(j) samples of the
% signal.
clear fftA fftB fftC fftD;

clear featuresA;
featuresA = [max(smoothA); min(smoothA); skewness(smoothA); ...
    kurtosis(smoothA)];
clear featuresB;
featuresB = [max(smoothB); min(smoothB); skewness(smoothB); ...
    kurtosis(smoothB)];
clear featuresC;
featuresC = [max(smoothC); min(smoothC); skewness(smoothC); ...
    kurtosis(smoothC)];
clear featuresD;
featuresD = [max(smoothD); min(smoothD); skewness(smoothD); ...
    kurtosis(smoothD)];

% Autocorrelation: computed at lags 0,1,2, ... T= min[20,length(y)-1]
clear RxxA;
RxxA = my_autocorr(smoothA); % See my_autocorr.m
clear RxxB;
RxxB = my_autocorr(smoothB);
clear RxxC;
RxxC = my_autocorr(smoothC);
clear RxxD;
RxxD = my_autocorr(smoothD);

if showPlots
    figure('units','normalized','outerposition',[0 0 1 1]);

    subplot(2,2,1);
    plot(0:20,RxxA);
    title('Supine - All the volunteers, all the autocorrelations');
    xlabel('lags');
    ylabel('Rxx');

    subplot(2,2,2);
    plot(0:20,RxxB);
    title('Dorsifexion - All the volunteers, all the autocorrelations');
    xlabel('lags');
    ylabel('Rxx');

    subplot(2,2,3);
    plot(0:20,RxxC);
    title('Walk - All the volunteers, all the autocorrelations');
    xlabel('lags');
    ylabel('Rxx');

    subplot(2,2,4);
    plot(0:20,RxxD);
    title('Stairs - All the volunteers, all the autocorrelations');
    xlabel('lags');
    ylabel('Rxx');
end

% first value, autocorr with lags=0 is always equal to 1
% (it's irrelevant)
featuresA = [featuresA; RxxA(2:end,:)];
featuresB = [featuresB; RxxB(2:end,:)];
featuresC = [featuresC; RxxC(2:end,:)];
featuresD = [featuresD; RxxD(2:end,:)];

% Frequential features:
% - Fundamental frequency f0
% - Power Spectral Density PSD

% Define the frequency domain
f = 12.2*(0:N(index)/2-1);

% Compute the single-sided spectrum
fftA = my_fft(smoothA,N(index)); % See my_fft.m
fftB = my_fft(smoothB,N(index));
fftC = my_fft(smoothC,N(index));
```

```
fftD = my_fft(smoothD,N(index));

if(showPlots)
    figure('units','normalized','outerposition',[0 0 1 1]);

    subplot(2,2,1);
    plot(f,fftA);
    title('Supine - All the volunteers, all the spectra');
    xlabel('frequency [Hz]');
    ylabel('FFT');

    subplot(2,2,2);
    plot(f,fftB);
    title('Dorsifexion - All the volunteers, all the spectra');
    xlabel('frequency [Hz]');
    ylabel('FFT');

    subplot(2,2,3);
    plot(f,fftC);
    title('Walk - All the volunteers, all the spectra');
    xlabel('frequency [Hz]');
    ylabel('FFT');

    subplot(2,2,4);
    plot(f,fftD);
    title('Stairs - All the volunteers, all the spectra');
    xlabel('frequency [Hz]');
    ylabel('FFT');
end

% The max amplitude should be a good approximation for the fundamental
% frequency
[amp, x] = max(fftA);
% [~, x2] = findpeaks(fftA, 'MinPeakProminence', 0.7*max(fftA));

% PSD
PSD = sum(fftA.^2);
% figure
% plot(f,fftA(:,[1 11 111]));
% hold on;
% plot(f(x([1 11 111])),y([1 11 111]),'rv');
featuresA = [featuresA; f(x); amp; PSD];

clear amp x;
[amp, x] = max(fftB);
featuresB = [featuresB; f(x); amp; sum(fftB.^2)];
clear amp x;
[amp, x] = max(fftC);
featuresC = [featuresC; f(x); amp; sum(fftC.^2)];
clear amp x;
[amp, x] = max(fftD);
featuresD = [featuresD; f(x); amp; sum(fftD.^2)];

%% Rotate matrix
% Since we use 4 columns to represent 3 sensor signals + 1 "virtual"
% sensor (the sum), we move all the features of the same piece of signal
% on the same column.
newFeaturesA = rotate_features(featuresA);
newFeaturesB = rotate_features(featuresB);
newFeaturesC = rotate_features(featuresC);
newFeaturesD = rotate_features(featuresD);
```

A.5 MY_AUTOCORR.M

```
% My extension of autocorr(), in order to compute autocorr on a matrix of
% more than 1 signal, where each column is a different signal
function Rxx = my_autocorr (matrix)
        % If "matrix" has only a column, I only need to compute correlation once
        Rxx = autocorr(matrix(:,1));

        for k=2:size(matrix,2)
                Rxx = [Rxx autocorr(matrix(:,k))];
        end
end
```

A.6 MY_FFT.M

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

A.7 FEATURESEL.M

```
%% 05 Feature Selection
% Sequential feature selection

num_features = 6;

% Classify each activity with a different class:
% Y = 0 => supine
% Y = 1 => dorsiflexion
% Y = 2 => walking
% Y = 3 => stairs
sizeA = size(newFeaturesA,2);
sizeB = size(newFeaturesB,2);
sizeC = size(newFeaturesC,2);
sizeD = size(newFeaturesD,2);
X = [newFeaturesA     newFeaturesB   newFeaturesC     newFeaturesD]';
Y = [zeros(sizeA,1);  ones(sizeB,1); 2*ones(sizeC,1); 3*ones(sizeD,1)];

f = @(xtrain, ytrain, xtest, ytest) ...
    sum(ytest ~= classify(xtest, xtrain, ytrain));
if showPlots
    opts = statset('display','iter');
    [fs, history] = sequentialfs(f,X,Y,'nfeatures',num_features,...
        'options',opts);
else
    [fs, history] = sequentialfs(f,X,Y,'nfeatures',num_features);
end
```

APPENDIX B –NEURAL NETWORK

B.1 NEUR_NETW.M

```
%% 06 Neural network

% "What's the best number of hidden neurons?"
inputs = X(:,fs)';
targets = zeros(4,size(Y,1));
targets(1,1:sizeA) = 1;
targets(2,sizeA+1:sizeA+sizeB) = 1;
targets(3,sizeA+sizeB+1:sizeA+sizeB+sizeC) = 1;
targets(4,sizeA+sizeB+sizeC+1:end) = 1;

n1 = 1;  % lowest number of hidden neurons
n2 = 10; % highest number of hidden neurons
performances = zeros(10,1);
regressions = zeros(10,4);
meanPerformance = zeros(n2-n1+1,1);
meanRegression = zeros(n2-n1+1,4);

for n = n1:1:n2,
    % Create a Pattern Recognition Network
    hiddenLayerSize = n;

    for k=1:10,
        net = patternnet(hiddenLayerSize);
        % Setup Division of Data for Training, Validation, Testing
        net.divideParam.trainRatio = 70/100;
        net.divideParam.valRatio = 15/100;
        net.divideParam.testRatio = 15/100;

        % hide window: speed up computations
        net.trainParam.showWindow = false;

        % Train the Network
        [net,~] = train(net,inputs,targets);

        % Test the Network
        outputs = net(inputs);
        performances(k) = perform(net,targets,outputs);
        [regressions(k,:),~,~] = regression(targets,outputs);
    end
    meanRegression(n,:) = mean(regressions);
    meanPerformance(n) = mean(performances);
end

if showPlots
    figure, plot(n1:n2, meanPerformance, 'r-o');
    title('MSE: less is better');
    ylabel('mean square error');
    xlabel('# of hidden neurons');
    legend('mean performance');

    figure, plot(n1:n2, meanRegression, 'g-o');
    title('R-value: correlation between output and targets');
    ylabel('Regression coefficent');
    xlabel('# of hidden neurons');
    legend('mean regression');
end

%% Train the neural network
% Best among the networks with chosen number of hidden neurons
[~,hiddenLayerSize] = min(meanPerformance);
p = inf;
for k=1:10,
    net_temp = patternnet(hiddenLayerSize);
    % Setup Division of Data for Training, Validation, Testing
    net_temp.divideParam.trainRatio = 70/100;
    net_temp.divideParam.valRatio = 15/100;
    net_temp.divideParam.testRatio = 15/100;

```

```
    % hide window: speed up computations
    net_temp.trainParam.showWindow = false;

    % Train the Network
    [net_temp,tr_temp] = train(net_temp,inputs,targets);

    % Test the Network
    outputs = net_temp(inputs);
    p_temp = perform(net_temp,targets,outputs);
    if(p_temp < p),
        best_net = net_temp;
        p = p_temp;
        tr = tr_temp;
    end
end

%% Print evaluations

outputs = best_net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(best_net,targets,outputs);

% View the Network
% view(best_net)

% Plots
% figure, plotperform(tr)
% figure, plottrainstate(tr)
% figure, plotconfusion(targets,outputs)
% figure, ploterrhist(errors)

[actual,~,~] = find(targets);
[~,predict] = max(outputs);
cfmatrix2(actual',predict,[1 2 3 4], 1, 1);
```

B.2 CFMATRIX2.M

```
%% https://it.mathworks.com/matlabcentral/fileexchange/21212-confusion-matrix---matching-
matrix-along-with-precision--sensitivity--specificity-and-model-accuracy

function [confmatrix] = cfmatrix2 ...
    (actual, predict, classlist, per, printout)
% CFMATRIX2 calculates the confusion matrix for any prediction
% algorithm ( prediction algorithm generates a list of classes to which
% each test feature vector is assigned );
%
% Outputs: confusion matrix
%
%                  Actual Classes
%                    p       n
%              ___|_____|______|
%    Predicted  p'|     |      |
%      Classes  n'|     |      |
%
%            Also the TP, FP, FN and TN are output for each class based
%            on http://en.wikipedia.org/wiki/Confusion_matrix
%            The Precision, Sensitivity and Specificity for each class
%            have also been added in this update along with the overall
%            accuracy of the model ( ModelAccuracy ).
…
% If classlist not entered: make classlist equal to all
% unique elements of actual
if (nargin < 2)
   error('Not enough input arguments. Need atleast two vectors as input');
elseif (nargin == 2)
    classlist = unique(actual); % default values from actual
    per = 0;
    printout = 1;
elseif (nargin == 3)
    per = 0; % default is numbers and input 1 or higher for percentage
    printout = 1;
elseif (nargin == 4)
    printout = 1; % default is silent output ( 0 ); one or higher printsout
elseif (nargin > 5)
   error('Too many input arguments.');
end


if (length(actual) ~= length(predict))
    error('First two inputs need to be vectors with equal size.');
elseif ((size(actual,1) ~= 1) && (size(actual,2) ~= 1))
    error('First input needs to be a vector and not a matrix');
elseif ((size(predict,1) ~= 1) && (size(predict,2) ~= 1))
    error('Second input needs to be a vector and not a matrix');
end
format short g;
n_class = length(classlist);
confmatrix = zeros(n_class);
line_two =       '----------';
line_three =     '_________|';

for i = 1:n_class
    for j = 1:n_class
        m = (predict == classlist(i) ...
           & actual  == classlist(j));
        confmatrix(i,j) = sum(m);
    end
    line_two = strcat(line_two,'---',num2str(classlist(i)),'-----');
    line_three = strcat(line_three,'__________');
end

TPFPFNTN    = zeros(4, n_class);
Accuracy    = zeros(1, n_class);
Precision   = zeros(1, n_class);
Sensitivity = zeros(1, n_class);
Specificity = zeros(1, n_class);
MissRate    = zeros(1, n_class);
Fall_Out    = zeros(1, n_class);

temps1 = sprintf('    TP  ');
temps2 = sprintf('    FP  ');
```

```
temps3 = sprintf('    FN  ');
temps4 = sprintf('    TN  ');
temps5 = sprintf('Accur.  ');
temps6 = sprintf('Preci.  ');
temps7 = sprintf('Sensi.  ');
temps8 = sprintf('Speci.  ');
temps9 = sprintf('MissR.  ');
temps10= sprintf('FallO.  ');

for i = 1:n_class
    % TP
    TPFPFNTN(1, i) = confmatrix(i,i);
    temps1 = strcat(temps1,sprintf(' |   %3d \t',TPFPFNTN(1, i)));

    % FP
    TPFPFNTN(2, i) = sum(confmatrix(i,:))-confmatrix(i,i);
    temps2 = strcat(temps2,sprintf(' |   %3d \t',TPFPFNTN(2, i) ));

    % FN
    TPFPFNTN(3, i) = sum(confmatrix(:,i))-confmatrix(i,i);
    temps3 = strcat(temps3,sprintf(' |   %3d \t',TPFPFNTN(3, i) ));

    % TN
    TPFPFNTN(4, i) = sum(confmatrix(:)) - sum(confmatrix(i,:)) -...
        sum(confmatrix(:,i)) + confmatrix(i,i);
    temps4 = strcat(temps4,sprintf(' |   %3d \t',TPFPFNTN(4, i) ));

    % Accuracy(class) = (TP(class)+TN(class))/all
    Accuracy(i) = (TPFPFNTN(1, i)+TPFPFNTN(4, i))/sum(confmatrix(:))*100;
    temps5 = strcat(temps5,sprintf(' |   %3.2f \t',Accuracy(i) ));

    % Precision(class) = TP(class) / ( TP(class) + FP(class) )
    Precision(i) = TPFPFNTN(1, i) / sum(confmatrix(i,:))*100;
    temps6 = strcat(temps6,sprintf(' |   %3.2f \t',Precision(i) ));

    % Sensitivity(class) = Recall(class) = TruePositiveRate(class)
    % = TP(class) / ( TP(class) + FN(class) )
    Sensitivity(i) = TPFPFNTN(1, i) / sum(confmatrix(:,i))*100;
    temps7 = strcat(temps7,sprintf(' |   %3.2f \t',Sensitivity(i) ));

    % Specificity ( mostly used in 2 class problems )=
    % TrueNegativeRate(class)
    % = TN(class) / ( TN(class) + FP(class) )
    Specificity(i) = TPFPFNTN(4, i) / ( TPFPFNTN(4, i) + TPFPFNTN(2, i) )*100;
    temps8 = strcat(temps8,sprintf(' |   %3.2f \t',Specificity(i) ));

    % Miss rate = FN(class) / ( TP(class) + FN(class) )
    MissRate(i) = TPFPFNTN(3, i)/sum(confmatrix(:,i))*100;
    temps9 = strcat(temps9,sprintf(' |   %3.2f \t',MissRate(i) ));

    % Fall-out = FP(class) / ( TN(class) + FP(class) )
    Fall_Out(i) = TPFPFNTN(2, i)/( TPFPFNTN(4, i) + TPFPFNTN(2, i) )*100;
    temps10 = strcat(temps10,sprintf(' |   %3.2f \t',Fall_Out(i) ));

end

ModelAccuracy = sum(diag(confmatrix))/sum(confmatrix(:))*100;
temps11 = sprintf('Model Accuracy is %1.2f ',ModelAccuracy);

if (per > 0) % ( if > 0 implies true; < 0 implies false )
    confmatrix = (confmatrix ./ length(actual)).*100;
end

if ( printout > 0 ) % ( if > 0 printout; < 0 no printout )
    disp('------------------------------------------');
    disp('             Actual Classes');
    disp(line_two);
    disp('Predicted|                    ');
    disp('  Classes|                    ');
    disp(line_three);

    for i = 1:n_class
        temps = sprintf('       %d              ',i);
        for j = 1:n_class
            temps = strcat(temps,sprintf(' |    %3.1f    ',confmatrix(i,j)));
        end
```

```
            disp(temps);
            clear temps
        end
        disp('-----------------------------------------');

        disp('-----------------------------------------');
        disp('             Actual Classes');
        disp(line_two);
        disp(temps1); disp(temps2); disp(temps3); disp(temps4);
        disp(temps5); disp(temps6); disp(temps7); disp(temps8);
        disp(temps9); disp(temps10);
        disp('-----------------------------------------');
        disp(temps11);
        disp('-----------------------------------------');
    end
    clear temps1 temps2 temps3 temps4 temps5 temps6 temps7 temps8 temps9 temps10 temps11
```

APPENDIX C –FUZZY INFERENCE SYSTEM

C.1 – FUZZY_SYS.M