University of Pisa

Master's Degree in Computer Engineering

Intelligent Systems' project

Marco Micera
Minou Manafi Varkiani

Intelligent Systems

Academic Year 2016-2017

Contents

1 Introduction **2**
1.1 Problem scenario . . . 2

2 Data editing **3**
2.1 Data Description . . . 3
2.2 Dataset Normalization . . . 3

3 Pattern Recognition with Neural Networks **5**
3.1 Specifications . . . 5
3.2 First Level . . . 5
3.2.1 Feature selection . . . 5
3.2.2 Structure . . . 6
3.3 Second Level . . . 6
3.3.1 Feature Selection . . . 6
3.3.2 Structure . . . 6
3.4 Neural network creation code . . . 7
3.5 Performances . . . 9

4 Mamdani Fuzzy Classifiers **10**
4.1 Requirements . . . 10
4.2 Horizontal scenario - First Level . . . 10
4.2.1 Feature Extraction . . . 10
4.2.2 Rules . . . 11
4.3 Horizontal Scenario - Second Level . . . 11
4.3.1 Feature Selection and Feature Extraction . . . 11
4.3.2 Rules . . . 12
4.4 Mamdani Fuzzy Classifiers - Vertical scenario . . . 12
4.5 Testing and Results . . . 12

5 Adaptive Neuro-Fuzzy Inference System (ANFIS) Classifiers **13**
5.1 Requirements . . . 13
5.2 Horizontal scenario - First Level . . . 13
5.2.1 Input Structure . . . 13
5.2.2 Graphs . . . 13
5.3 Horizontal scenario - Second Level . . . 14
5.3.1 Input Structure . . . 14
5.3.2 Graphs . . . 14
5.4 Vertical scenario . . . 15
5.4.1 Input Structure . . . 15
5.4.2 Graphs . . . 15

6 Clustering with Self-Organization Maps (SOM) **16**
6.1 Requirements . . . 16
6.2 Horizontal scenario - First Level . . . 16
6.3 Horizontal scenario - Second Level . . . 18
6.4 Vertical scenario . . . 18
6.5 Performances . . . 19

7 Observations **19**

1 Introduction

During this project we designed and built several systems that are able to recognize 4 different actions in a restricted area, using an RFID tag: 3 dynamic actions and one static action.

Dynamic actions:

1. *enter* in the restricted area;
2. *exit* from the restricted area;
3. *walk in parallel to the gate* inside the restricted area without going out.

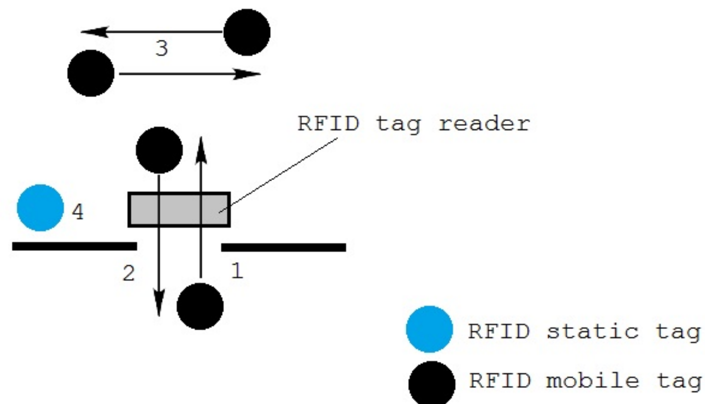
Static action:

4. *stand still* close to the restricted area.

These actions are performed by people equipped with an RFID tag oriented horizontally or vertically with respect to the RFID reader.

1.1 Problem scenario

The way data was collected is shown in the following scheme:



When the RFID reader is turned on, it receives the signals from the two RFID tags collocated in the area: the fixed static tag and the mobile tag.
Four different experiments were performed. In each experiment the moving person performs one of the three possible dynamic actions. Simultaneously, a static tag is placed near the doorstep to simulate a person standing still. In addition, in each experiment, the tag orientation was also taken into account. The orientation of the mobile tag can be horizontal or vertical. The orientation of the static tag is horizontal in all the experiments. Each experiment was repeated 10 times, and the signals from the static and mobile tags were simultaneously collected.

2 Data editing

2.1 Data Description

The dataset consists of:

- 30 different samples corresponding to dynamic actions, belonging to the three possible classes *(enter, exit, walk in parallel)*, in the horizontal scenario;
- 30 different samples corresponding to dynamic actions, belonging to the three possible classes *(enter, exit, walk in parallel)*, in the vertical scenario;
- 60 different samples corresponding to the static action *(standing still)*.

There are 60 .mat files. Each .mat file refers to an action performed with a given tag orientation, and it contains the signal information related to:

1. The dynamic action collected with the mobile tag;
2. The static action collected with the static tag.

Inside each .mat file the only relevant information is the 'Inventario' structure.
Inside the *Inventario* structure there are two relevant data structures:

- *Tag98BD*, which refers to the mobile tag
- *Tag98B6*, which refers to the static tag

Both tags have the same internal content:

- *RSSiTag* vector corresponding to the amplitude values (in dBm) of the RSSI signal received;
- *RNSITag* vector corresponding to the noise values (in dBm) of the RSSI signal received;
- *FaseTag* vector corresponding to the values of the phase of the signal (in degrees);
- *TempoTag* corresponding to the sampling times of the signal (in ms).

2.2 Dataset Normalization

One important part of the pre-processing activity is the dataset normalization. First of all, we created a new array input matrix: each sample consists in 30 averages, computed from the relative initial input samples. This allows us to have constant size inputs. Then, we normalized all samples values from 0 to 1.

```
1
2 function result = createMatrix(mean_values)
3     % For every mat file contained in the Misure_VarcoRFID
        folder
4     file_names = dir('Misure_VarcoRFID/*.mat');
```

```
5       result = [];
6
7       % At first, dynamic samples
8       for s = 1:length(file_names)
9           file = (importdata(strcat('Misure_VarcoRFID/',
                file_names(s).name)));
10          result = [result; normalizeInput(file.Inventario.
                Tag598BD.RSSITag, mean_values).'];
11      end
12
13      % Then, static samples
14      for s = 1:length(file_names)
15          file = (importdata(strcat('Misure_VarcoRFID/',
                file_names(s).name)));
16          result = [result; normalizeInput(file.Inventario.
                Tag598B6.RSSITag, mean_values).'];
17      end
18  end
```

```
1   function normalized_array = normalizeInput(input_array,
        mean_values)
2
3           input_array_size = length(input_array);
4       values_per_average = floor(input_array_size/mean_values)
            ;
5       normalized_array = zeros(mean_values, 1);
6
7       avg_start = 1;
8       for current_average = 1:mean_values
9           if(current_average == mean_values)
10              avg_end = input_array_size - avg_start + 1;
11          else
12              avg_end = values_per_average;
13          end
14
15          normalized_array(current_average) =
16                  mean(input_array(avg_start:avg_start+avg_end
                        -1,1));
17          avg_start = avg_start + avg_end;
18      end
19
20           mx = max(normalized_array);
21       mn = min(normalized_array);
22       for i = 1:mean_values
23           normalized_array(i) = (normalized_array(i) - mn)/(
                 mx - mn);
24       end
25  end
```

3 Pattern Recognition with Neural Networks

3.1 Specifications

The aim is to build hierarchical classifiers organized on two levels.
The first level is made by a 2-class classifier which is able to distinguish between dynamic static actions. This classifier takes as input a signal sample and returns as output the corresponding class label (*static* or *dynamic*). The second level is made by a 3-class classifier which is able to distinguish among the 3 dynamic actions. This classifier takes as input a signal sample which has been previously classified as dynamic in the first level, and it produces as output the corresponding class label (*enter*, *exit* or *walk in parallel*).

3.2 First Level

3.2.1 Feature selection

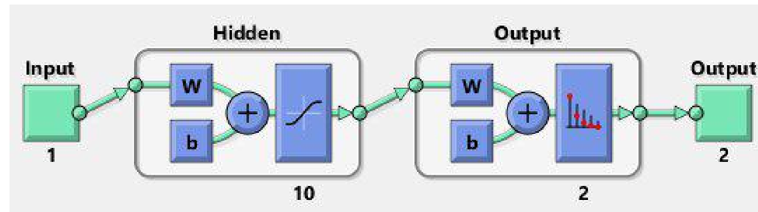
In order to choose the best subset of the provided features, we opted for the **sequentialfs()** Matlab function: this function selects a feature subset from the data matrix **ds_samples** (which stands for *dynamic/static samples matrix*) that best predicts data in **ds_target** (which stands for *dynamic/static target matrix*), by sequentially selecting features that won't provide further improvement in prediction. Rows of **ds_samples** correspond to observations (samples); columns correspond to variables (features). **ds_target** is a column vector containing response values or class labels for each observation in X.
We organized **ds_samples** as follows: the first 60 rows contain dynamic samples, and last 60 rows contain static ones.
The **ds_target** matrix structure is reported below:

```
%% Dynamic/static target matrix (order: dinamic, static)
ds_target = [  % samples by rows
    1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0;
    1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0;
    1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0;
    1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0;
    1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0;
    1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
];
```

The **sequentialfs()** function returns the first input column as the only selected feature: this means that the first neural network only need the first feature to correctly distinguish among dynamic and static sample.

3.2.2 Structure

The following scheme is provided by the **view()** function.



3.3 Second Level

3.3.1 Feature Selection

The sample and target matrix now change: they only have to contain those samples representing dynamic inputs.

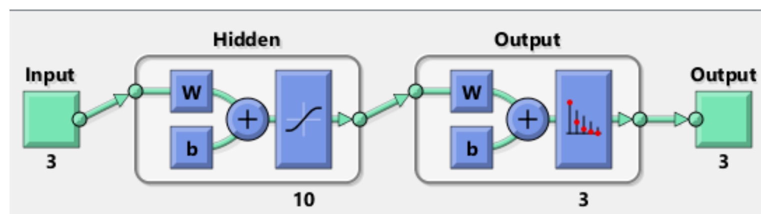
```
%% Dynamic samples matrix
d_samples = ds_samples(1:60, :);

%% Dynamic target matrix (order: in, out, pass)
d_target = [ % samples by rows
    1 0 0; 1 0 0; 1 0 0; 1 0 0; 1 0 0;
    1 0 0; 1 0 0; 1 0 0; 1 0 0; 1 0 0;
    0 1 0; 0 1 0; 0 1 0; 0 1 0; 0 1 0;
    0 1 0; 0 1 0; 0 1 0; 0 1 0; 0 1 0;
    0 0 1; 0 0 1; 0 0 1; 0 0 1; 0 0 1;
    0 0 1; 0 0 1; 0 0 1; 0 0 1; 0 0 1;
    1 0 0; 1 0 0; 1 0 0; 1 0 0; 1 0 0;
    1 0 0; 1 0 0; 1 0 0; 1 0 0; 1 0 0;
    0 1 0; 0 1 0; 0 1 0; 0 1 0; 0 1 0;
    0 1 0; 0 1 0; 0 1 0; 0 1 0; 0 1 0;
    0 0 1; 0 0 1; 0 0 1; 0 0 1; 0 0 1;
    0 0 1; 0 0 1; 0 0 1; 0 0 1; 0 0 1;
];
```

The **sequentialfs()** function selects 3 features for the **d_samples** matrix.

3.3.2 Structure

This image is provided by the **view()** function.



3.4 Neural network creation code

```
function [net_1, net_2] = createNN()
    %% Creating the first neural network

    global ds_sf ds_target;

    % Samples by columns
    ds_sf_by_columns = ds_sf';
    ds_target_by_columns = ds_target';

    % Creating it
    trainFcn = 'trainscg';
    hiddenLayerSize = 10;
    net_1 = patternnet(hiddenLayerSize, trainFcn);

    % Training it
    net_1.trainParam.showWindow = false; % no GUI
    [net_1, training_record] = train(net_1, ds_sf_by_columns
        , ds_target_by_columns);

    % Testing it
    result = net_1(ds_sf_by_columns);
    errors = gsubtract(ds_target_by_columns, result);
    % performance = perform(net, target, result)
    % target_indexes = vec2ind(target);
    % result_indexes = vec2ind(result);
    % percentErrors = sum(target_indexes ~= result_indexes)/
        numel(target_indexes);

    % View the Network
     view(net_1)

    % Plots
     figure, plotperform(training_record)
     figure, plottrainstate(training_record)
     figure, ploterrhist(errors)
     figure, plotconfusion(ds_target_by_columns, result)
     figure, plotroc(ds_target_by_columns, result)

    %% Creating the second neural network

    global d_sf d_target;

    % Samples by columns
    d_sf_by_columns = d_sf';
    d_target_by_columns = d_target';

    % Creating it
    trainFcn = 'trainscg';
    hiddenLayerSize = 20;
    net_2 = patternnet(hiddenLayerSize, trainFcn);

    % Training it
```

```
51      net_2.trainParam.showWindow = false; % no GUI
52      [net_2, training_record] = train(net_2, d_sf_by_columns,
            d_target_by_columns);
53
54      % Testing it
55      result = net_2(d_sf_by_columns);
56      errors = gsubtract(d_target_by_columns, result);
57      % performance = perform(net, target, result)
58      % target_indexes = vec2ind(target);
59      % result_indexes = vec2ind(result);
60      % percentErrors = sum(target_indexes ~= result_indexes)/
          numel(target_indexes);
61
62      % View the Network
63       view(net_2)
64
65      % Plots
66       figure, plotperform(training_record)
67       figure, plottrainstate(training_record)
68       figure, ploterrhist(errors)
69       figure, plotconfusion(d_target_by_columns, result)
70       figure, plotroc(d_target_by_columns, result)
71
72      %% Cleaning
73      clear criteriaFunctionHandler;
74  end
```

3.5 Performances

We created some different networks and we compared the results to select the best one in order to find the best trade off between accuracy of results and minimum computational costs. We simulated the nets with three different number of hidden layers (10, 15 and 20) using the train function.

For the first layer, 10 hidden neurons are enough to ensure the best performance:

For the second layer we obtained the following confusion matrices:

a) 10 hidden layers *b) 15 hidden layers* *c) 20 hidden layers*

We chose the third option *(hiddenLayerSize = 20)* because of the great increase in terms of performances (from $\sim 76\%$ to $\sim 93\%$).

4 Mamdani Fuzzy Classifiers

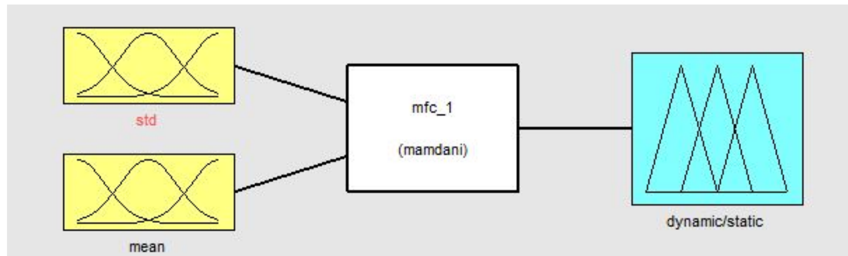
4.1 Requirements

The objective is to build two fuzzy hierarchical classifiers (one for the *horizontal* and one for the *vertical scenario*), organized on two levels as explained before. This scenario is composed by two levels: one for the dynamic/static classification and one for the dynamic actions classification.[1]

For this part, we used the *Fuzzy Logic Designer app* of the *Fuzzy Logic toolbox*, as required.

4.2 Horizontal scenario - First Level

The schema for the first level of the horizontal scenario is the following:

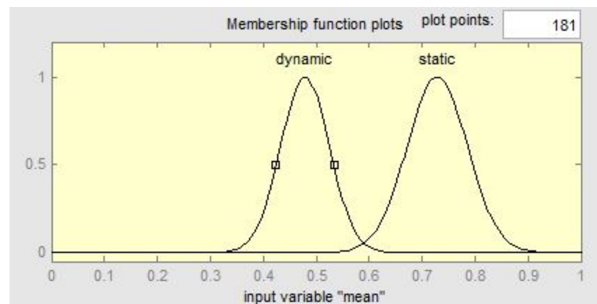


4.2.1 Feature Extraction

For each input signal, we extracted the following features:

- *Standard Deviation*
- *Mean Value*

Subsequently we calculated, for each feature column (we have 90 values for each one), the standard deviation and the mean value and we used this values to create input membership functions, that will discriminate against static and dynamic signals.



[1]The static tag is horizontal in all the experiments

4.2.2 Rules

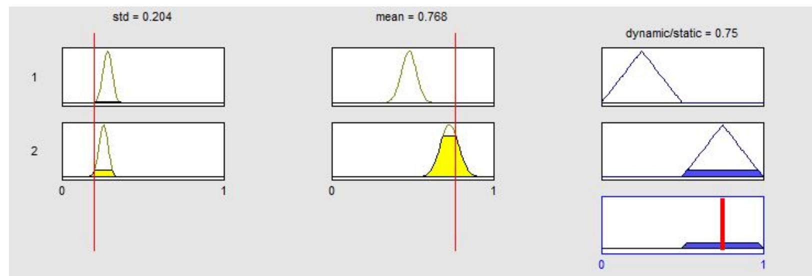
Through the following rules it was possible to compute the output:

- *if (std is dynamic) and (mean is dynamic), then (Out is dynamic)*
- *if (std is static) and (mean is static), then (Out is static)*

These rules were codifed on Matlab in a m + n + 2 column matrix, with m inputs and n outputs. Each column between m + n columns contains a numeric encoding that refers to the membership function index concerning that variable. The m + n + 1 column refers to the weight that should be applied to the rule: generally, it is a number between 0 and 1. Otherwise, the m + n + 2 column is referred to the driver before the rule: it is set 1 if there is an "AND", or 2 if there is an "OR".

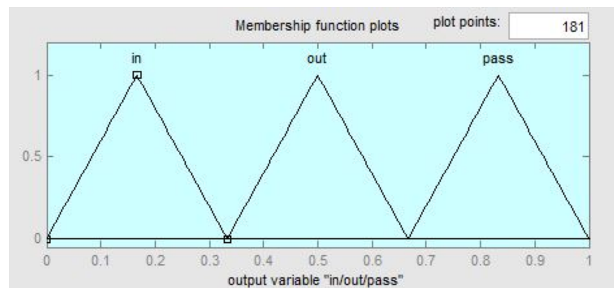
$$rulelist = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix} \tag{1}$$

Depending on the input sample membership degree, the outcome changes accordingly to the rules previously shown.



4.3 Horizontal Scenario - Second Level

The general schema for the second level of the horizontal scenario is the same as before.
The output has now the following shape:



4.3.1 Feature Selection and Feature Extraction

For this level (which is more complex than the first one) we computed the features *standard deviation* and *mean value* only by using the columns returned by the *sequentialfs* used in the previous part for the neural networks.

4.3.2 Rules

The rules for the second level are the following:

- *if (std is in) and (mean is in) then (Out is in)*
- *if (std is out) and (mean is out) then (Out is out)*
- *if (std is pass) and (mean is pass) then (Out is pass)*

Rules matrix:

$$rulelist = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 3 & 3 & 1 \end{bmatrix} \tag{2}$$

4.4 Mamdani Fuzzy Classifiers - Vertical scenario

The vertical scenario is the same as second level horizontal scenario, mutatis mutandis.

4.5 Testing and Results

We obtained the following result *(right results/total number of inputs)*:

- First level of horizontal scenario: $\sim 98\%$
- Second level of horizontal scenario: $\sim 88\%$
- Vertical scenario : $\sim 50\%$

We obtained these results by performing 10 different tests and by calculating the resulting mean.

5 Adaptive Neuro-Fuzzy Inference System (ANFIS) Classifiers

5.1 Requirements

The objective here is to build two fuzzy hierarchical classifiers (one for the *horizontal* and one for the *vertical scenario*), organized on two levels as explained before.

5.2 Horizontal scenario - First Level

We used the *Neuro-Fuzzy Designer app* of the *Fuzzy Logic toolbox.*

5.2.1 Input Structure

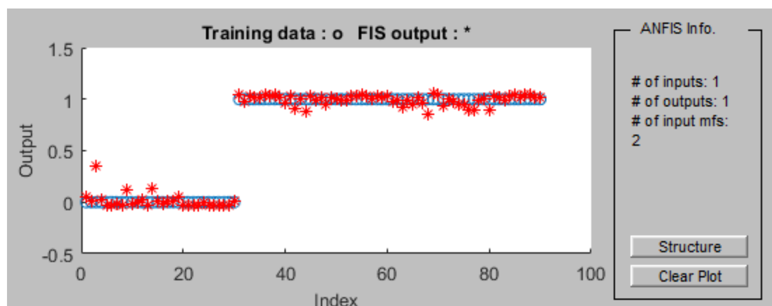
To train the FIS, we begin by loading a training data set that contains the desired input-output data of the system.

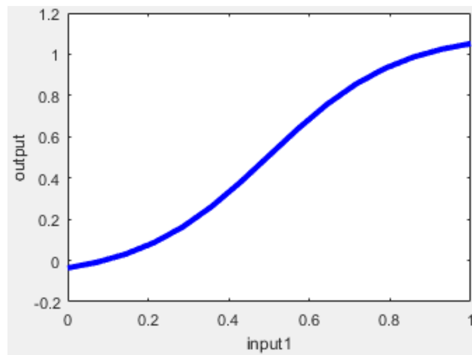
```
%% Dynamic/static horizontal target matrix
anfis_ds_h_target = [
    0; 0; 0; 0; 0; 0; 0; 0; 0; 0; % dynamic
    0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
    0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
    1; 1; 1; 1; 1; 1; 1; 1; 1; 1; % static
    1; 1; 1; 1; 1; 1; 1; 1; 1; 1;
    1; 1; 1; 1; 1; 1; 1; 1; 1; 1;
    1; 1; 1; 1; 1; 1; 1; 1; 1; 1;
    1; 1; 1; 1; 1; 1; 1; 1; 1; 1;
    1; 1; 1; 1; 1; 1; 1; 1; 1; 1;
];



%% Dynamic/static horizontal selected features + targets
    matrix
anfis_ds_h_sf = [ds_h_samples(:, find(ds_h_sf_indexes))
    anfis_ds_h_target];
```

5.2.2 Graphs





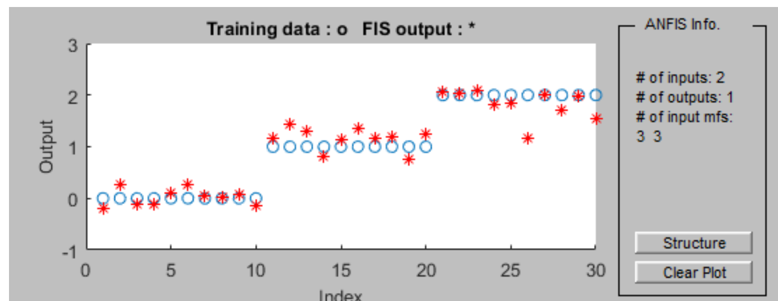
5.3 Horizontal scenario - Second Level

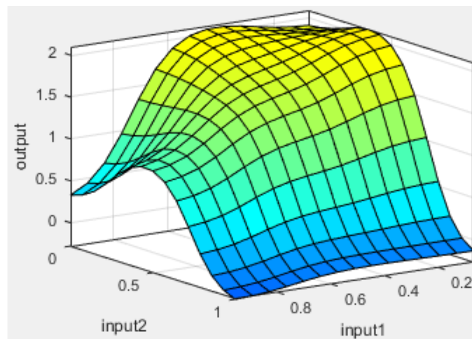
5.3.1 Input Structure

```
%% Dynamic horizontal target matrix
anfis_d_h_target = [
    0; 0; 0; 0; 0; 0; 0; 0; 0; 0; % in
    1; 1; 1; 1; 1; 1; 1; 1; 1; 1; % out
    2; 2; 2; 2; 2; 2; 2; 2; 2; 2; % pass
];

%% Dynamic horizontal selected features + targets matrix
anfis_d_h_sf = [d_h_samples(:, find(d_h_sf_indexes))
    anfis_d_h_target];
```

5.3.2 Graphs





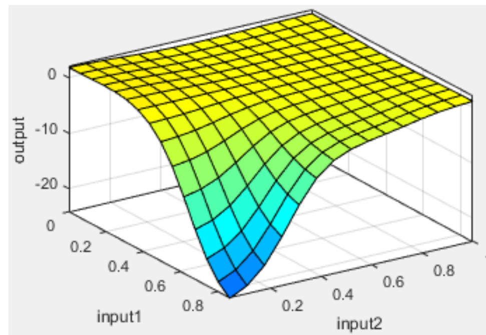
5.4 Vertical scenario

5.4.1 Input Structure

```
%% Vertical target matrix
anfis_v_target = anfis_d_h_target;

%% Vertical selected features + targets matrix
anfis_v_sf = [v_samples(:, find(v_sf_indexes))
    anfis_v_target];
```

5.4.2 Graphs



6 Clustering with Self-Organization Maps (SOM)

6.1 Requirements

For each scenario *(horizontal and vertical)*, we want to build two clustering models:

1. the first in sable to distinguish between *static* and *dynamic* actions,
2. the second is able to distinguish among the three dynamic activities *(enter, exit, walk in parallel)*.

6.2 Horizontal scenario - First Level

We used the *Neural Clustering App*. The network creation is contained in the **som_ds_h_creation.m** file, which is reported below:

```
%% Creating a Self-Organizing Map for dynamic/static
    discrimination
global som_net_1 som_1_clusters;
som_net_1 = selforgmap([10 10]);

%% Choosing Plot Functions
som_net_1.plotFcns = {
    'plotsomtop','plotsomnc','plotsomnd', ...
    'plotsomplanes', 'plotsomhits', 'plotsompos'
};

%% Training
global ds_h_samples;
[som_net_1, ~] = train(som_net_1, ds_h_samples');

%% Clustering

% SOM weights
som_1_weights = som_net_1.IW{1, 1};

% 'ward': Inner squared distance (minimum variance algorithm
    ),
% appropriate for Euclidean distances only
som_1_linkages = linkage(som_1_weights, 'ward');

% som_dendrogram = dendrogram(som_1_linkages, 0);

% Clusters creation
som_1_clusters = cluster(som_1_linkages, 'maxclust', 2);

%% som_net_1 plots
% figure, plotsomtop(som_net_1)
% figure, plotsomnc(som_net_1)
% figure, plotsomnd(som_net_1)
% figure, plotsomplanes(som_net_1)
% figure, plotsomhits(som_net_1, ds_h_samples')
% figure, plotsompos(som_net_1, ds_h_samples')
```

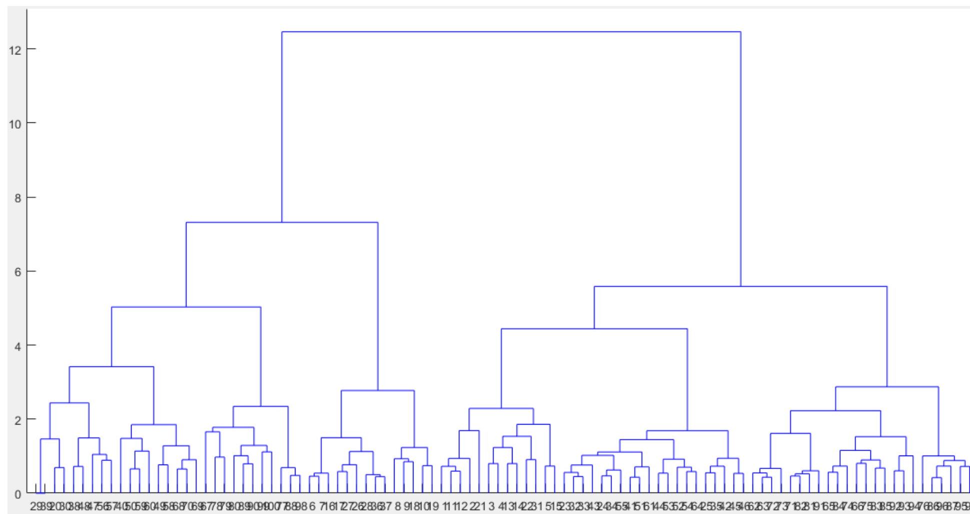
After training the SOM, we obtained the network's weights thanks to this code line:

```
som_1_weights = som_net_1.IW{1, 1};
```

Then, thanks to the **linkage()** function, we obtained the **som_1_linkages** matrix in which the agglomerative hierarchical cluster tree formation process is reported.

```
som_1_linkages = linkage(som_1_weights, 'ward');
```

The *'ward'* parameters specifies the algorithm for computing distance between clusters. In this case, *'ward'* stands for the inner squared distance (minimum variance algorithm): the Matlab documentation states that this is appropriate for Euclidean distances only. The **som_1_linkages** matrix can be better visualized with a dendrogram:



With the following instruction, 2 clusters can be created (one for dynamic action, the other one for static ones):

```
% Clusters creation
som_1_clusters = cluster(som_1_linkages, 'maxclust', 2);
```

The **som_1_clusters** matrix then contains, for each SOM neuron (a hundred of them), the corresponding output cluster identifier. For this reason, during the input computation, this instruction is performed:

```
som_ds_h_output = som_1_clusters(find(som_net_1(input.')),
    1);
```

The following code represents the first level of the SOM for horizontal samples:

```
% Dynamic/static Self-Organizing Map

function result = som_ds_h(input)
    %% Checking input dimension
    if size(input, 1) ~= 1
```

```
6           error('Wrong input row size');
7       elseif size(input, 2) ~= 30
8           error('Wrong input column size');
9       end
10
11      %% Input computation
12      global som_net_1 som_1_clusters;
13      som_ds_h_output = som_1_clusters(find(som_net_1(input.')
            ), 1);
14      result = som_ds_h_output;
15
16      if som_ds_h_output == 1
17          disp('The given input belongs to the first cluster.'
                );
18      elseif som_ds_h_output == 2
19          disp('The given input belongs to the second cluster.
                ');
20      else
21          disp('The given input does not belong to any cluster
                .');
22      end
23
24      %% Cleaning
25      clear training_record;
26  end
```

A simple test script may consists in the following code:

```
1  % Testing the SOM for all possible dynamic/static input
       samples
2
3  global ds_h_samples;
4
5  som_ds_h_creation;
6  for i = 1:size(ds_h_samples, 1)
7      ans = som_ds_h(ds_h_samples(i, :));
8  end
9  clear i ans;
```

6.3 Horizontal scenario - Second Level

All operations are made for the second level: this time, the classifier has to distinguish among *entering*, *exiting* and *passing* actions. This means that the SOM training is performed just with the relative samples, and then 3 clusters are formed.

6.4 Vertical scenario

This case is completely equivalent to the second level of the horizontal scenario, as in the vertical one, no static actions are registered.

6.5 Performances

- First level of the horizontal scenario: $\sim 89\%$
- Second level of the horizontal scenario: $\sim 100\%$
- Vertical scenario: $\sim 77\%$

7 Observations

We have noticed that vertical scenario performances are not good enough compared to the horizontal ones.
This could mean that vertical samples were not correctly measured, or that they could have been affected by some errors.
Therefore, their use do not certainly bring any advantage in the action decision process.