Politecnico di Torino

ICT for health

Academic Year 2016/2017

Professors:
Monica Visintin
Guido Pagana

Author:
Gaetano Moceri

Contents

1 Introduction

The purpose of this report is to explain how laboratory activities within the "ICT for Health" subject have been faced. Each chapter is related to a single topic. In order to allow readability, within each section there will be just an introduction and the explaination of what it was done by introducing some significant code lines and graphs. The whole set of codes and plots is placed at the end of each descriptive section.

2 Linear regression - PCR - Neural Networks

2.1 Introduction

Linear regression and Principal Component Regression (PCR) are machine learning techniques which allow the prevision of some future values starting from a set of known samples. This machine learning approach is called supervised learning and performes predictive analysis. PCR is a more complex technique: regression is carried out after a data manipulation that leads to the selection of a subset of the initial dataset (Principal Component Analysis). Neural networks will be treated in the following paragraphs. The dataset comes from UCI Machine Learning repository (link dataset).

The aim of the laboratory is performing linear regression on feature 5 and feature 7 of the initial dataset, which are respectively clinician's motor linearly interpolated UPDRS score and a measured voice parameter (jitter in percentage). The expected result is that measured feature will regress better with respect to the interpolated one.

Data cleaning and normalization Is the very first operation to be done on the raw data. It is a sort of filtering unreliable and wrong or missing data and guarantees data correctness. The "dataLoading" function aims to perform this task.

Then, it is convenient to normalize data measured in different scales in order to make featuers coherent with each other. From a pratical point of view, normalizing a dataset means remove averages and make variances equal to one.
Click 2.3 for the code.

2.2 Linear regression

From the normalized dataset, it is possible to apply machine learning algorithms.

Linear regression in particular, models the relationship between one or more scalar dependent variables, called regressors, and a dependent variable, called regressand. The focus is on finding a linear function that links the regressors and the regressand. Linear function consists in finding some weights which link the dependent variable with the independent ones. Parameters of this linear function are estimated starting from data and different methodologies can be used.

Minimum Square Error This method calculates the best-fitting line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line: if a

point lies on the fitted line exactly, then its vertical deviation is 0, so error is null. Minimum square error (MSE) is an estimation method which minimizes the square error of the observed values and the fitted ones. From a conceptual point of view, the error function is derived in order to find the set of parameters which minize it. The graphs show the output of the MSE algorithm. Figure 1 deals with the interpolated feature. While figure 2 shows the result for feature 7. Click 2.3 for the code.

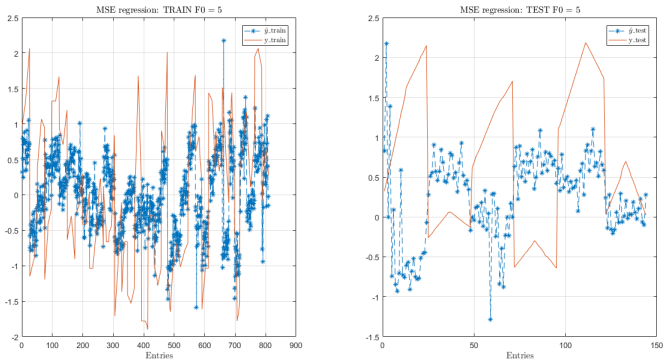


Figure 1: MSE estimation plots for feature 5

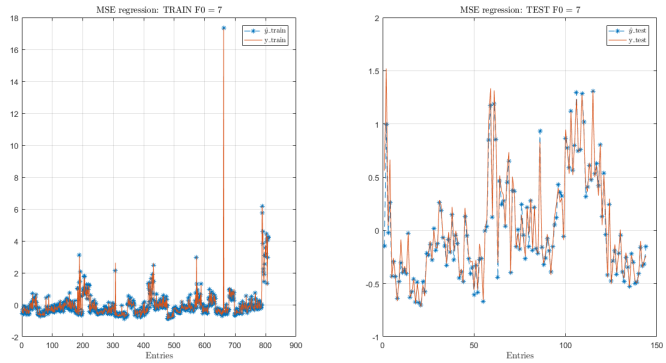


Figure 2: MSE estimation plots for feature 7

As it was expected, feature 5 is very bad fitted because it deals with the physician judgment: it is uncorrelated with the other features and is difficoult to be predicted. The errors in both train and test phase show a very high variance and regression is (see below for the other graphs). On the other side, feature 7 is related to voice characteristics: in this case prediction is very accurated and regression is much more linear than before.

Gradient Algorithm Since MSE technique requires the train matrix inversion, which could require a lot of computational and time resources in the case of a lot of matrix entries, an iterative

solution is suited. In particular, gradient algorithm (GA) allows to find local minimum of a function from an initial guess of the coefficients. Each iteration, a "jump" toward the minimum is performed: the direction of the jump comes from the gradient (negative), while the length is given by a learning coefficient greater than zero. Learning coefficient should be properly choosen: if it is too small there could be a lot of iterations before reaching a good solution. If it is too large, jumps may not reach an optimal solution and remaining in a loop around the local minimum. A stopping condition should block the iterations when the distance from the previous solution is smaller than a threshold. GA result seems very similar to the MSE one, but computational time is almost 20 times larger from the learning coefficient and threshold choosen values. Click 2.3 for the code.

Steepest Descent Steepest descent (SD) is an iterative algorithm very similar to the gradient algorithm, discussed previously. The difference is that here the step size is adaptive: learning coefficient is updated each iteration. While the same threshold and the same stopping condition used in gradient algorithm are applied in steepest descent. Click 2.3 for the code.

Conclusions From the graphs generated, (6, 7, 4, 5, 2, 8, 9) the results of the three algorithm discussed are very similar each other, in terms of errors estimation. The main difference is the computational time: with the configurated parameters, MSE and SD have a comparable execution time, while GA execution time is 10-15 time higher. For both features, it has also been computated the iterations number for the iterative algorithms: SD has about 1000 iterations, while GA performs 100 times the number of SD iterations, maybe due to the fact that SD adapts the step size by updating the learning coefficient.

2.3 Principal Component Regression

Principal component regression (PCR) is a kind of regression performed from a simplified version of the dataset. The "simplification" process is actuated by principal component analysis (PCA) where initial features are projected in a new space through an orthonormal basis computed from covariance matrix of the initial train dataset. Then, new features are statistically indepentent. Click 2.3 for the code.

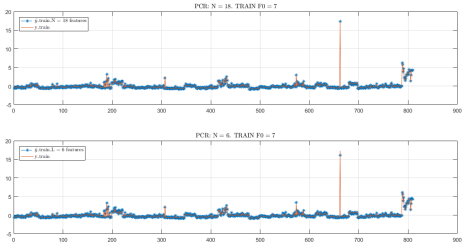


Figure 3: PCR estimation for both features

From the basis, eigenvectors with the corresponding highest eigenvalues are more "important" and correspond to the variables with the highest variance: eigenvectors with lower eigenvalues have to be descarded because the corresponding features are less meaningful: pratically, it is possible to discard them in order to simplify the dataset. In this case, the 90% of the eigenvalues total sum is considered, corresponding to "L" features. Performances are still similar to the other algorithms described (MSE and SD), but now the set of coefficients is more regular and from figure 3 is possible to see how this method underestimate the big outlier, in the inner graph.

For very complex dataset, PCA is fundamental because it allows feature extraction and consequentely data simplification. Another advantage of PCR is that it is very robust against outliers, as told before, but on the other side, since a part of data is going to be "thrown away", performance, in terms of error are worse with respect to the other algorithms.
Below, you find all scripts and plots for the regression task.

Listing 1: UPDRS Analysis - Main

```
close all
clear all
clc

load('updrs.mat');
updrs = parkinsonsupdrs;
nOfDays = 180;
nOfPatients = 42;
trainingPatients = 36;
set(groot,'DefaultLegendInterpreter','latex')
set(groot,'DefaultTextInterpreter','latex')

% ========== DATA LOADING & NORMALIZATION ==========
% ==================================================================================


% dataLoading function is devoted to "data cleaning" procedure: from the
% imported raw data (updrs) it deletes empty features, averages daily
% measurements per patient and sorts days per patient.
updrsNew = dataLoading(updrs, nOfPatients, nOfDays);

% The obtained matrix must be normalized through "matNorm" function. The
% output is train matrix and test matrix, normalized.
[data_train_norm, data_test_norm] = matNorm(updrsNew, trainingPatients);

consideredFeatures = [2:4 8:22];
% consideredFeatures = 5:22;
trainMatrix = data_train_norm(:, consideredFeatures);
testMatrix = data_test_norm(:, consideredFeatures);
x_train = trainMatrix;
x_test = testMatrix;
% Residual features are: (2,3,4,8,9):end

F0 = [5 7]; % Vector containing analysed features.
```

```
34
35 tic
36 % ========================== MSE ESTIMATION ==========================
37 % ====================================================================
38 for ii = 1:length(F0)
39     % MSECoefficients function returns MSE coefficients and the test
           and
40     % train vectors.
41     [y_train, y_test, aHatMSE] = MSECoefficients(data_train_norm, ...
42         data_test_norm , x_train, F0(ii));
43
44     y_train_hat = x_train * aHatMSE;        % Trying TRAINING data.
45     y_test_hat = x_test * aHatMSE;          % Trying TESTING data.
46
47     estimPlot(y_train, y_train_hat, y_test, y_test_hat, F0(ii), aHatMSE
           , 'MSE');
48 end
49 timeElapsedMSE = toc
50
51 % ========================= GRADIENT ALGORITHM ========================
52 % ====================================================================
53 rng('default');
54 M = length(aHatMSE);        % Number of features
55 threshold = 10^-6;       % Treshold for the stopping condition
56 gamma = 10^-5;           % Speed of convergence
57 % countGA vector contains the number of iterations per each F0:
58 countGA = zeros(1, length(F0));
59
60 tic
61 for ii = 1:length(F0)
62
63     [y_train, y_test, aHatGA, countGA(ii)] = GACoefficients(
           data_train_norm, ...
64         data_test_norm, M, x_train, gamma, threshold, F0(ii));
65
66     % In aHatFinal there is the final set of coefficients a(i+1):
67     y_train_hat = x_train * aHatGA;
68     y_test_hat = x_test * aHatGA;
69
70     estimPlot(y_train, y_train_hat, y_test, y_test_hat, F0(ii), aHatGA,
           'GA');
71 end
72 timeElapsedGA = toc
73
74 % ======================== STEPEST DESCENT ===========================
75 % ====================================================================
```

```
76  rng('Default');
77  % Vector containing the number of iterations per each F0
78  countSD = zeros(1, length(F0));
79
80  tic
81  for ii = 1:length(F0)
82      [y_train, y_test, aHatSD, countSD(ii)] = SDCoefficients(
        data_train_norm, ...
83          data_test_norm, M, x_train, threshold, F0(ii));
84
85      y_train_hat = x_train * aHatSD;
86      y_test_hat = x_test * aHatSD;
87
88      estimPlot(y_train, y_train_hat, y_test, y_test_hat, F0(ii), aHatSD,
            'SD');
89  end
90  timeElapsedSD = toc
91
92  % ================================ PCR ================================
93  % =====================================================================
94  F = M;  % Numero di features originali
95  N = length(x_train(:, 1));
96  tic
97  for ii = 1:length(F0)
98
99      [y_train, y_test, aHatPCR] = MSECoefficients(data_train_norm, ...
100         data_test_norm , x_train, F0(ii));
101
102     percentage = 0.9;
103     % L is the new number of features considered
104     [aHatPCRL, L] = PCRCoefficients(data_train_norm, data_test_norm, N,
            ...
105         x_train, percentage, F0(ii));
106
107     % Result computing for both N and L features considered:
108     y_train_hat_Nfeature = x_train * aHatPCR;
109     y_test_hat_Nfeature = x_test * aHatPCR;
110     y_train_hat_Lfeature = x_train * aHatPCRL;
111     y_test_hat_Lfeature = x_test * aHatPCRL;
112
113     % Task 1
114     figure, subplot(2,2,1)
115     plot(y_train_hat_Nfeature, '--*'), hold on, grid on, plot(y_train)
116     legend(['$\hat{y}$\_train\_N = ', num2str(F), ' features'], ...
117         'y\_train', 'Location', 'northwest')
118     title(['PCR: N = ', num2str(F), '. TRAIN F0 = ', ...
```

```
        num2str(F0(ii))])
    subplot(2,2,2)
    plot(y_train_hat_Lfeature, '--*'), hold on, grid on, plot(y_train)
    legend(['$\hat{y}$\_train\_L = ', num2str(L), ' features'], ...
        'y\_train', 'Location', 'northwest')
    title(['PCR: N = ', num2str(L), '. TRAIN F0 = ', ...
        num2str(F0(ii))])

    % Task 2
    subplot(2,2,3)
    plot(y_test_hat_Nfeature, '--*'), hold on, grid on, plot(y_test)
    legend(['$\hat{y}$\_test\_N = ', num2str(F), ' features'], 'y\_test
        ')
    title(['PCR: N = ', num2str(F), '. TEST F0 = ', ...
        num2str(F0(ii))])
    subplot(2,2,4)
    plot(y_test_hat_Lfeature, '--*'), hold on, grid on, plot(y_test)
    legend(['$\hat{y}$\_test\_L = ', num2str(L), ' features'], 'y\_test
        ')
    title(['PCR: N = ', num2str(L), '. TEST F0 = ', ...
        num2str(F0(ii))])

    % Task 3
    errTrainPCRN = y_train - y_train_hat_Nfeature;
    meanTrainPCRN = mean(errTrainPCRN);
    varTrainPCRN = var(errTrainPCRN);
    figure, subplot(2,2,1)
    hist(errTrainPCRN, 50), grid on
    title(['TRAIN for N = ', num2str(F), ...
        '. F0 = ', num2str(F0(ii)), '. var = ', num2str(varTrainPCRN),
            ...
        '. mean = ', num2str(meanTrainPCRN)])
    subplot(2,2,2)
    errTrainPCRL = y_train - y_train_hat_Lfeature;
    meanTrainPCRL = mean(errTrainPCRL);
    varTrainPCRL = var(errTrainPCRL);
    hist(errTrainPCRL, 50), grid on
    title(['TRAIN for N = ', num2str(L), ...
        '. F0 = ', num2str(F0(ii)), '. var = ', num2str(varTrainPCRL),
            ...
        '. mean = ', num2str(meanTrainPCRL)])

    % Task 4
    errTestPCRN = y_test - y_test_hat_Nfeature;
    meanTestPCRN = mean(errTestPCRN);
    varTestPCRN = var(errTestPCRN);
```

```
161       subplot(2,2,3)
162       hist(errTestPCRN, 50), grid on
163       title(['TEST for N =  ', num2str(F), ...
164           '. F0 = ', num2str(F0(ii)), '. var = ', num2str(varTestPCRN),
                ...
165           '. mean = ', num2str(meanTestPCRN)])
166       subplot(2,2,4)
167       errTestPCRL = y_test - y_test_hat_Lfeature;
168       meanTestPCRL = mean(errTestPCRL);
169       varTestPCRL = var(errTestPCRL);
170       hist(y_test - y_test_hat_Lfeature, 50), grid on
171       title(['TEST for N = ', num2str(L), ...
172           '. F0 = ', num2str(F0(ii)), '. var = ', num2str(varTestPCRL),
                ...
173           '. mean = ', num2str(meanTestPCRL)])
174
175       % Task 5
176       estimPlotPCR(aHatPCR, aHatPCRL, F0(ii), y_train, ...
177           y_train_hat_Nfeature, y_train_hat_Lfeature)
178   end
179   timeElapsedPCR = toc
```

Listing 2: Data cleaning function

```
1   function [updrsNew] = dataLoading(updrs, nOfPatients, nOfDays)
2   % This function fixes the original matrix downloaded from the website:
3   % - updrs --> original matrix
4   % - nOfPatients --> number of patients in the dataset
5   % - nOfDays --> number of days of observation
6   % - updrsNew --> new matrix with the requested features
7   % [updrsNew] = dataLoading(updrs, nOfPatients, nOfDays)
8
9       count = 1;
10      lung = 0;
11      for patient = 1:nOfPatients
12          %patientIndex = [];
13          timeArray = [];
14          patientIndex = find(updrs(:, 1) == patient);    % Index-Patient
                vector
15
16          for k = 1:length(patientIndex)
17              % timeArray --> is the vector containing all the days
                    rounded to
18              % the closest INTEGER number related to the "patient"-th
                    patient.
19              timeArray(k, 1) = floor(updrs(patientIndex(k), 4));
20          end
```

```
21
22          % In this cycle I evaluate the mean for each patient: I scroll
                every
23          % day from day 1 to day 180: in this way I do not consider
                negative
24          % days.
25          for days = 1:nOfDays
26              %timeIndex = [];
27              sumRow = zeros(1, 22);
28              timeIndex = find(timeArray == days);     % Contains the
                    indices of
29                                                       % the days index
30
31              if ~isempty(timeIndex)
32                  for (ii = 1:length(timeIndex))
33                      sumRow = sumRow + updrs(timeIndex(ii) + lung, :);
34                  end
35                  sumRow = sumRow ./ length(timeIndex);    % <-- Contains
                        the
36                                                           % vector of the
                                                                means
37                  updrsNew(count, :) = sumRow;     % <-- Final matrix for
                        the
38                                                   % patient "patient"-th
39                  count = count + 1;
40              end
41          end
42          % lungh contains the index of the last element related to the
43          % "patient"-th patiens.
44          lung = patientIndex(length(patientIndex));
45      end
46  end
```

Listing 3: Data normalization function

```
1  function [trainNorm, testNorm] = matNorm(updrsNew, tr)
2  % This function returns the train and test NORMALIZED matrices.
3  % updrsNew           --> sorted matrix, output of dataLoading function
4  % tr                 --> number of patients used as train dataset
5  % trainNorm/testNorm --> output normalized matrixes
6  % [trainNorm, testNorm] = matNorm(updrsNew, tr)
7
8      trainIndex = find(updrsNew(:, 1) <= tr);
9      data_train = updrsNew(1:length(trainIndex), :);
10     data_test = updrsNew(length(trainIndex)+1:end, :);
11     m_data_train = mean(data_train, 1);
12     v_data_train = var(data_train, 1);
```

```
    trainDim = length(data_train(:, 1));
    testDim = length(data_test(:, 1));
    onesMatrixTrain = ones(trainDim, 1);
    onesMatrixTest = ones(testDim, 1);
    meanMatrixTrain = onesMatrixTrain * m_data_train;
    meanMatrixTest = onesMatrixTest * m_data_train;
    varMatrixTrain = onesMatrixTrain * v_data_train;
    varMatrixTest = onesMatrixTest * v_data_train;
    trainNorm = (data_train - meanMatrixTrain) ./ sqrt(varMatrixTrain);
    testNorm = (data_test - meanMatrixTest) ./ sqrt(varMatrixTest);

end
```

Listing 4: MSE coefficients function

```
function [y_tr, y_te, coef] = MSECoefficients(nMatTr, nMatrTe, trMat,
    Index)
% This function finds MSE coefficients for the following inputs:
% - nMatTrain --> normalized training matrix
% - nMatTest --> normalized testing matrix
% - y_tr --> feature used for training
% - y_te --> feature to be tested
% - coef --> MSE coefficients
% [y_tr, y_te, coef] = MSECoefficients(nMatTr,nMatrTe, trMat, Index)

    y_tr = nMatTr(:, Index);  % Feature to be estimated (y_train)
    y_te = nMatrTe(:, Index);   % Feature to be tested (y_test)
    coef = pinv(trMat) * y_tr;
end
```

Listing 5: Plot function

```
function [] = estimPlot(tr, trH, te, teH, featureIndex, coeff, str)
% tr --> vector y_train
% thH --> vector y_train_hat (ESTIMATED)
% te --> vector y_test
% teH --> vector y_test_hat (ESTIMATED)
% featureIndex --> index of the feature
% coeff --> MSE coefficients
% estimPlot(tr, trH, te, teH, featureIndex, coeff)

    v_ = sort(tr);
    a = v_(1);
    b = v_(end);
    l = length(tr);
%     x_ax = linspace(a, b, l);
%     y = x_ax;

```

```
17      figure, subplot(2,3,1)
18      plot(trH, '--*'), hold on, grid on, plot(tr, '')
19      title([str, ' regression: TRAIN F0 = ', num2str(featureIndex)])
20      legend('$\hat{y}$\_train', 'y\_train'), xlabel('Entries')
21
22      subplot(2,3,4)
23      plot(teH, '--*'), hold on, grid on, plot(te, '')
24      title([str, ' regression: TEST F0 = ', num2str(featureIndex)])
25      legend('$\hat{y}$\_test', 'y\_test'), xlabel('Entries')
26
27      subplot(2,3,2)
28      plot(tr, trH, 'o'), grid on,
29  %     plot(x_ax, y, 'linewidth', 2)
30      title(['Regression for F0 = ', num2str(featureIndex)])
31      xlabel('y\_train'), ylabel('$\hat{y}$\_train')
32
33      subplot(2,3,5)
34      plot(coeff), grid on
35      title(['Coefficients w for F0 = ', num2str(featureIndex)])
36
37      errTrainMSE = tr - trH;
38      varTrainMSE = var(errTrainMSE);
39      meanTrainMSE = mean(errTrainMSE);
40      subplot(2,3,3)
41      hist(errTrainMSE, 50), grid on
42      title(['TRAIN prediction error F0 = ', num2str(featureIndex), ...
43          '. Var = ', num2str(varTrainMSE), '. Mean = ', num2str(
                meanTrainMSE)])
44
45      errTestMSE = te - teH;
46      varTestMSE = var(errTestMSE);
47      meanTestMSE = mean(errTestMSE);
48      subplot(2,3,6)
49      hist(errTestMSE, 50), grid on
50      title(['TEST prediction error for F0 = ', num2str(featureIndex),
            ...
51          '. Var = ', num2str(varTestMSE), '. Mean = ', num2str(
                meanTestMSE)])
52  end
```

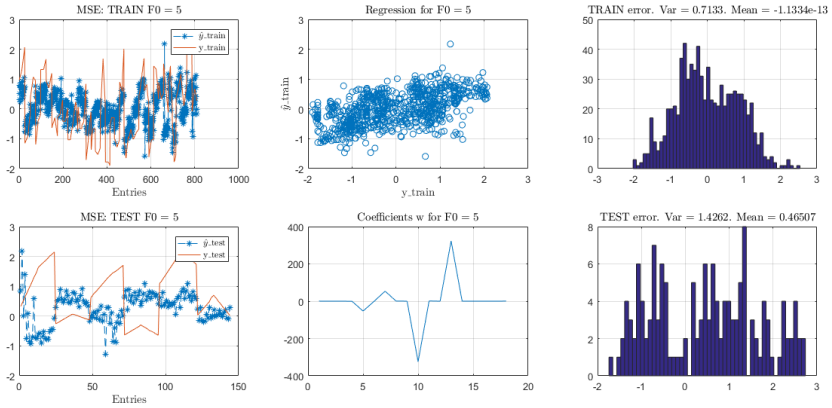


Figure 4: MSE estimation plots for feature 5

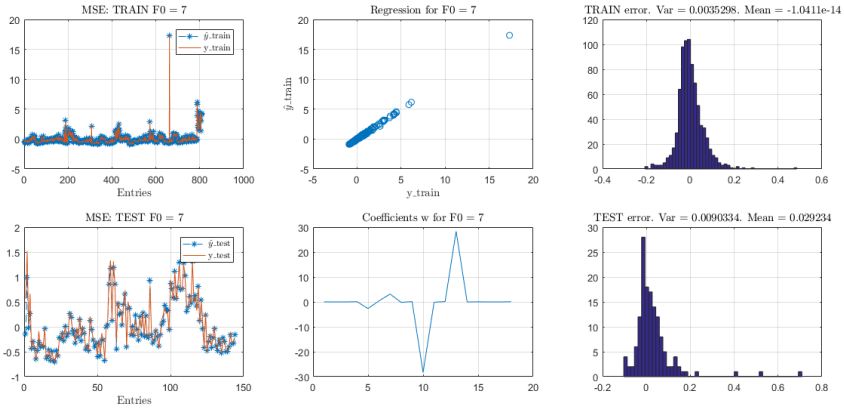


Figure 5: MSE estimation plots for feature 7

Listing 6: GA coefficients function

```
function [y_tr, y_te, coeff, count] = GACoefficients(nMatTr, nMatTe,
    ...
    nF, trMat, g, t, Index)
% - nMatTrain --> normalized training matrix
% - nMatTest --> normalized testing matrix
% - g --> gamma
% - t --> threshold
% - nF --> number of features
% - trMat --> train matrix
% [y_tr, y_te, coeff, count] = GACoefficients(nMatTr, nMatTe, ...
```

```
%     nF, trMat, g, t, Index)

      y_tr = nMatTr(:, Index);
      y_te = nMatTe(:, Index);
      aHatInitialGA = rand(nF, 1);
      gradientGA = (-2 * (trMat)' * y_tr) + (2 * (trMat)' * trMat * ...
          aHatInitialGA);
      aHatFinalGA = aHatInitialGA - (g * gradientGA);
      ii = 1;
      count = 0;
      % aHatInitial = a(i)
      % aHatFinal = a(i + 1) --> Final coefficients vector
      while norm(aHatFinalGA - aHatInitialGA) > t
          count(ii) = count(ii) + 1;
          aHatInitialGA = aHatFinalGA;
          gradientGA = (-2 * trMat' * y_tr) + (2 * (trMat)' * trMat * ...
              aHatInitialGA);
          aHatFinalGA = aHatInitialGA - (g * gradientGA);
      end
      coeff = aHatFinalGA;
end
```

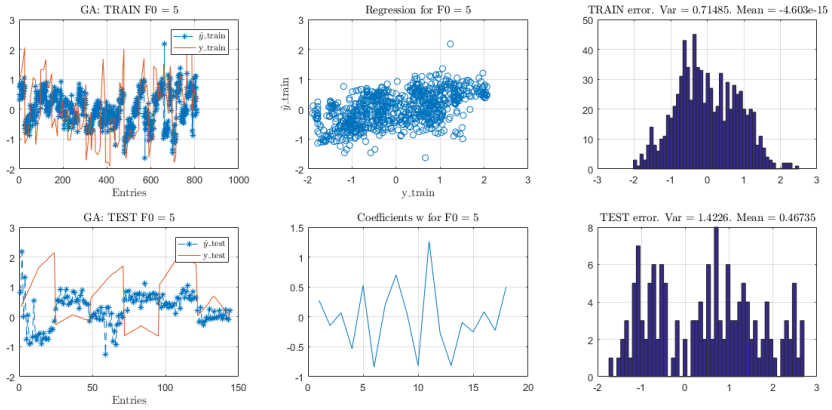


Figure 6: GA estimation plots for feature 5

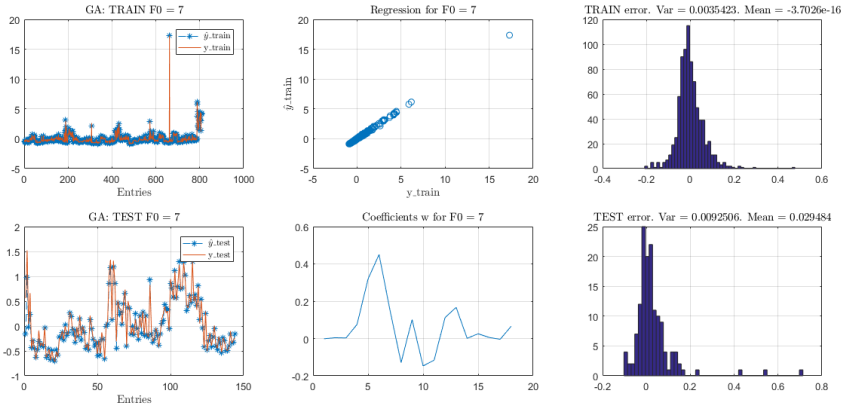


Figure 7: GA estimation plots for feature 7

Listing 7: SD coefficients function

```
function [y_tr, y_te, coeff, count] = SDCoefficients(nMatTr, nMatTe,
    ...
    nF, trMat, t, Index)

    y_tr = nMatTr(:, Index);
    y_te = nMatTe(:, Index);

    aHatInitialSD = rand(nF, 1);
    gradientSD = (-2 * (trMat)' * y_tr) + (2 * (trMat)' * ...
                 trMat * aHatInitialSD);
    hessianAHat = 4 * (trMat') * trMat;
    g = ((norm(gradientSD)^2) / (gradientSD' * hessianAHat * ...
             gradientSD));
    aHatFinalSD = aHatInitialSD - (g * gradientSD);
    ii = 1;
    count = 0;

    while norm(aHatFinalSD - aHatInitialSD) > t
        count(ii) = count(ii) + 1;
        aHatInitialSD = aHatFinalSD;
        gradientSD = (-2 * (trMat') * y_tr) + (2 * (trMat)' *  ...
                     trMat * aHatInitialSD);
        g = ((norm(gradientSD)^2) / (gradientSD' * hessianAHat * ...
                 gradientSD));
        aHatFinalSD = aHatInitialSD - (g * gradientSD);
    end
    coeff = aHatFinalSD;
end
```

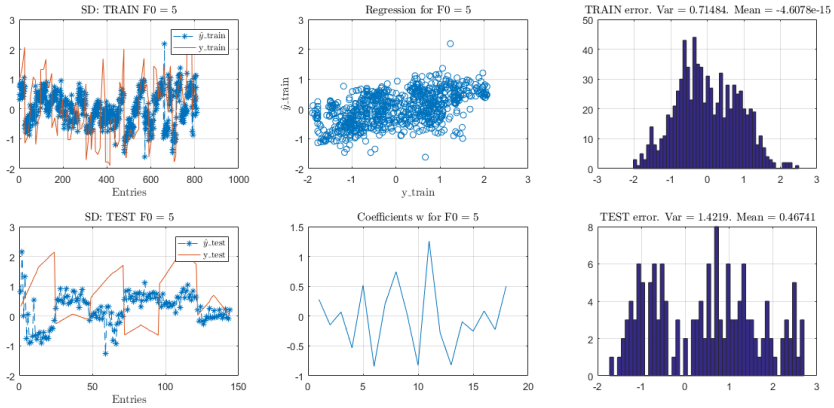


Figure 8: SD estimation plots for feature 5

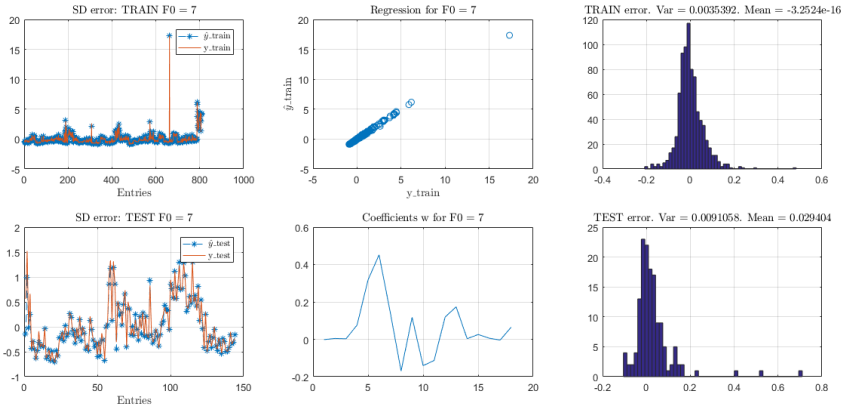


Figure 9: SD estimation plots for feature 7

Listing 8: PCR coefficients function

```
function [coeff, L] = PCRCoefficients(nMatTr, nMatTe, ...
    nRows, trMat, perc, index)

    y_tr = nMatTr(:, index);
    y_te = nMatTe(:, index);

    R = (1/nRows) * (trMat') * trMat;    % Covariance matrix
    [U, Lambda] = eig(R);
    Lambdas = diag(Lambda);
    P = sum(Lambdas);
```

```
%     Z = x_train * U;     % We map initial features on orthogonal
    vectors
    somma = 0;
    L = 0;
    while somma < perc * P
        L = L + 1;
        somma = somma + Lambdas(L);
    end
    LambdaL = Lambda(1:L, 1:L);
%     LambdasL = diag(LambdaL);
    UL = U(:, 1:L);
    aHatPCRL = (1/nRows) * UL * (inv(LambdaL)) * (UL') * (trMat') *
        y_tr;
    coeff = aHatPCRL;
end
```

Listing 9: PCR plot function

```
function [] = estimPlotPCR(c_F, c_L, index, y_tr, y_tr_N, y_tr_L)

    F = length(c_F);
    L = length(c_L);
%
%     v_ = sort(y_tr);
%     a_ = v_(1);
%     b_ = v_(end);
%     l_ = length(y_tr);
%     x_ax = linspace(a_, b_, l_);
%     y_ = x_ax;

    figure, subplot(2,2,1:2)
    plot(c_F), grid on, hold on, plot(c_L, 'o')
    legend(['$\hat{a}$\_N with N = ', num2str(F), ' features'], ...
        ['$\hat{a}$\_L with L = ', num2str(L), ' features'], ...
        'Location', 'northwest')
    title(['PCR: Coefficients for F0 = ', num2str(index)])

    subplot(2,2,3)
    plot(y_tr, y_tr_N, 'o'), grid on, hold on,
%     plot(x_ax, y_, 'linewidth', 2)
    title(['PCR: Regression for N = ', num2str(F), ...
        ' features and F0 = ', num2str(index)])
    xlabel('y\_train'), ylabel('$\hat{y}$\_train')
    subplot(2,2,4)
    plot(y_tr, y_tr_L, 'o'), grid on, hold on,
%     plot(x_ax, y_, 'linewidth', 2)
    title(['PCR: Regression for N = ', num2str(L), ...
```

```
30              ' features and F0 = ', num2str(index)])
31          xlabel('y\_train'), ylabel('$\hat{y}$\_train')
32      end
```

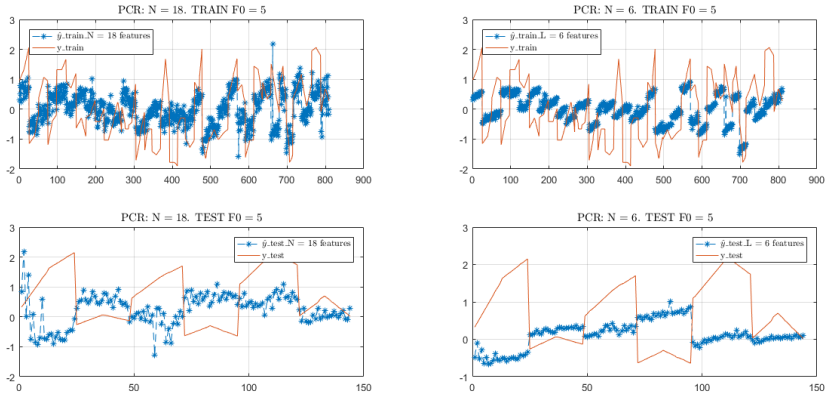


Figure 10: PCR estimation plots for feature 5

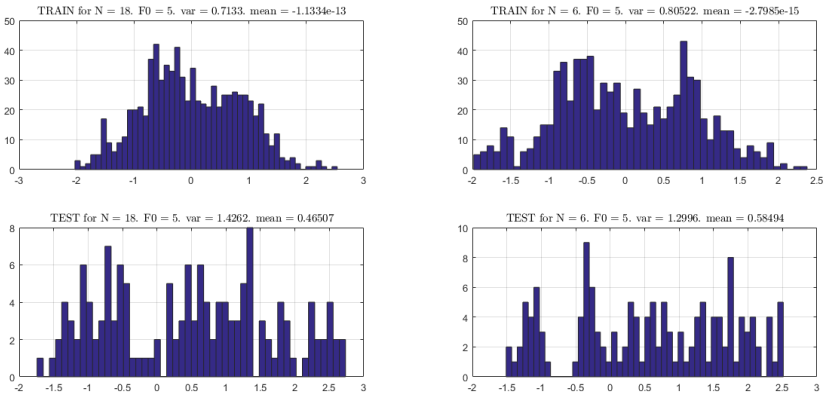


Figure 11: PCR error plots for feature 5

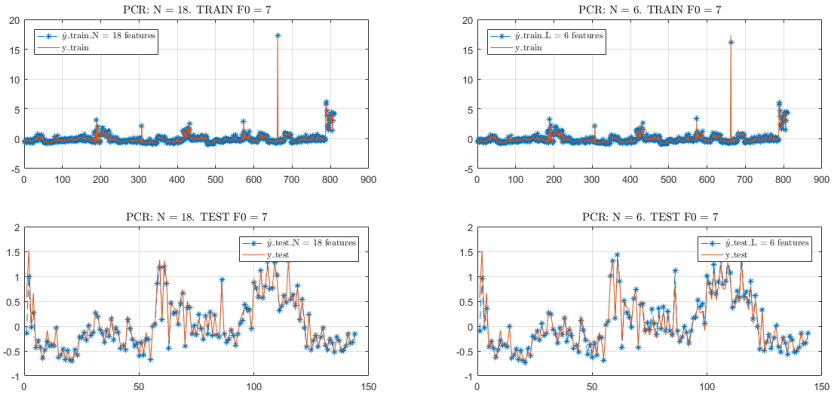


Figure 12: PCR estimation plots for feature 7

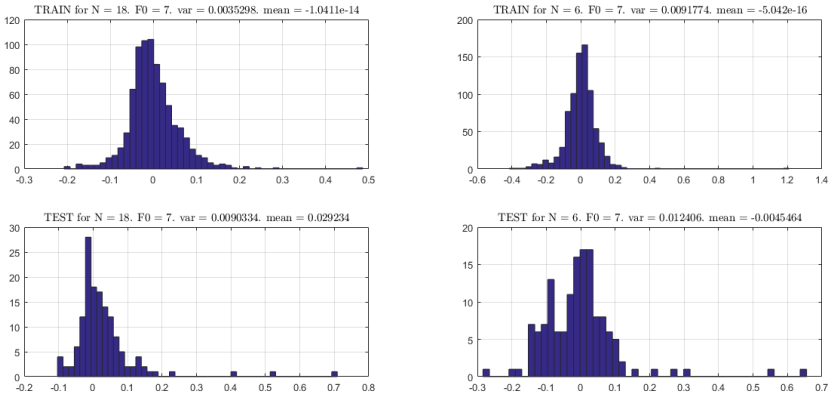


Figure 13: PCR error plots for feature 7

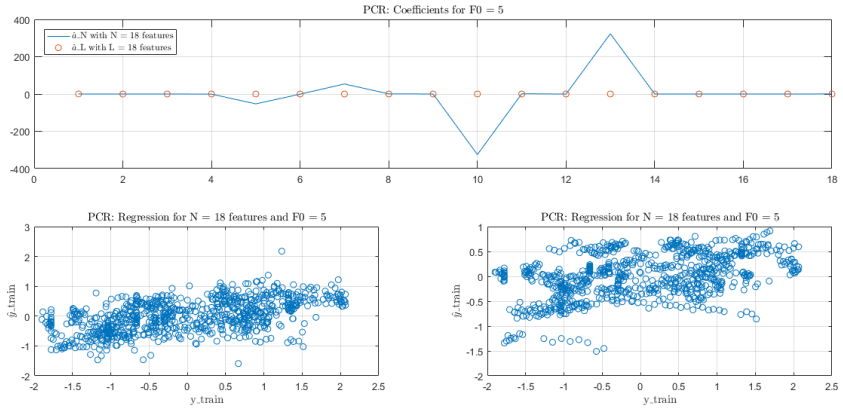


Figure 14: PCR regression plots for feature 5

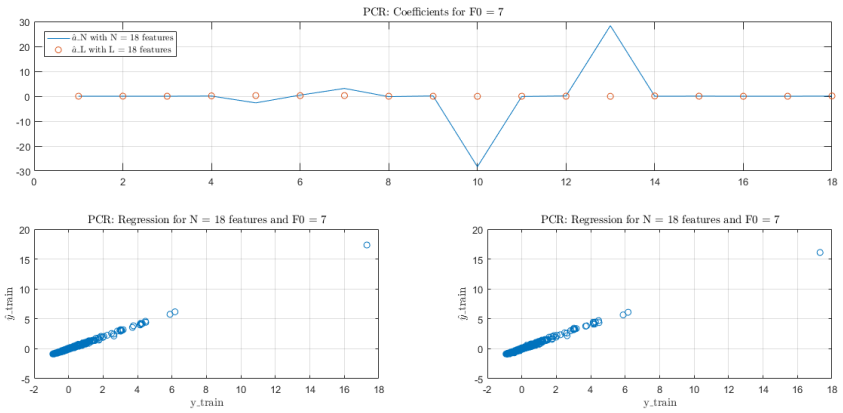


Figure 15: PCR regression plots for feature 7

2.4 Neural networks - Introduction

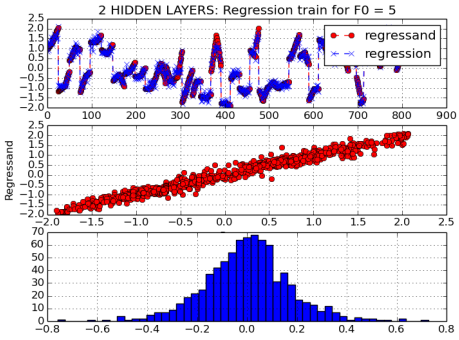
Regression task was also performed by using neural networks. For this exercise it has been used Python programming language with Google Tensorflow library. Initial dataset is already normalized. Typically, neural networks use a gradient algorithm approach by applying backpropagation method: Tensorflow allows to do it very easily. Tensorflow approach is divided in two parts: building the neural network "hardware" and runing the computational graph.
In order to compare performances with MATLAB ones, the parameters are the same. Check 2.4.2 for the whole code. A menu was implemented in order to allow the user to choose if to perform a no hidden layers regression or a two hidden layers regression. Code is shown here 2.4.2.

2.4.1 No hidden layers regression

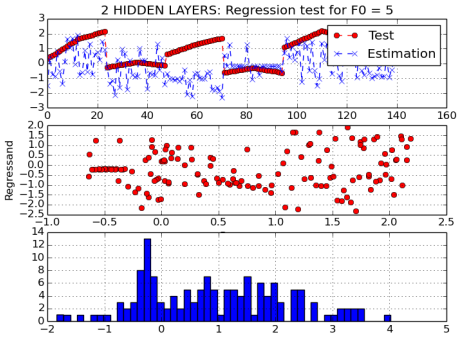
In no hidden layers case, as it was expected, feature 5 is still very badly fitted with a huge error variance, which is very similar to the MATLAB case shonw in 6. While feature 7, as it was expected again, seems to be very well estimated on both train an test phase. Error histogram is quite similar to the MATLAB gradient algorithm analysis in 7. It is reasonable because the learning coefficient is exactly the same, and the iterations number is of the same order of magnitude.

2.4.2 Two hidden layers regression

In this case, neural network implements two hidden layers. The first one has 17 nodes and the second one 10. Here, training performances seems very good in feature 5 training context, as it is shown in the figure 16a beside: regression is quite good and errors histogram presents a very low variance. But when the model is tested, result gets worse than the "no hidden layers" case. It could be a clear example of overfitting: it means that the model learns perfectly the train dataset features, but it is not able to predict the new observations. In this case, the model found is not usable. This happens when, for example, parameters number is big with respect to the observations or when the training runs for too much time. Cross-validation may be a very usefull method to prevent overfitting.



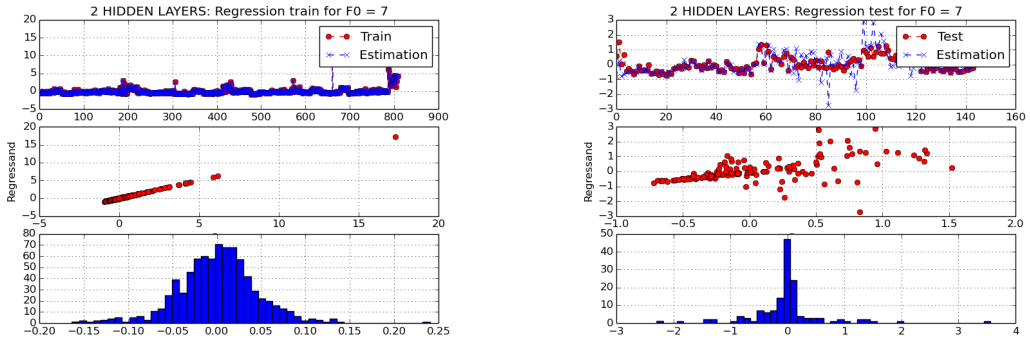
(a) ANN training for F 5



(b) ANN testing for F 5

Figure 16: Regression plots for F 5 with 2 hidden layers

The same discussion is valid for feature 7: the training phase is very good and the error histogram shows that error is enormously small (Fig.17). But when the model is tested, it gets worse than the "no hidden layers" case because of overfitting, again. In general, introducing some new hidden layers in neural networks, gets worse results. For what concern the neurons number, there is an empirical rule saying that "the optimal size of the hidden layer is usually between the size of the input and size of the output layers".



(a) ANN training for F 7

(b) ANN testing for F 7

Figure 17: Regression plots for F 7 with 2 hidden layers

Listing 10: Neural network regression

```
import tensorflow as tf
import numpy as np
import scipy.io as scio
import matplotlib.pyplot as plt
#plt.close("all")

# ================================ DATA LOADING
# Data come from MATLAB elaboration, hence have already been properly
# normalized
dataLoad = scio.loadmat("DataNorm.mat")
dataTrainLoad = scio.loadmat("DataTrainNorm.mat")
dataTestLoad = scio.loadmat("DataTestNorm.mat")

dataNormOriginal = dataLoad.get("DataNorm")
dataTrainNorm = dataTrainLoad.get("data_train_norm")
dataTestNorm = dataTestLoad.get("data_test_norm")
excludedFeatures = [0, 3, 4, 5, 6]  # Features to be excluded
x_train = np.delete(dataTrainNorm, excludedFeatures, 1)
x_test = np.delete(dataTestNorm, excludedFeatures, 1)
nSamplesTrain = len(x_train[:, 0])
nSamplesTest = len(x_test[:, 0])
```

```
22 nFeatures = len(x_train[0, :])
23 regression = [4]      # Features to be regressed
24 hiddenNodes1 = 17
25 hiddenNodes2 = 10
26 flag = True
27
28 while flag:
29     print("\nNeural Network application\n")
30     inp = str(input("Insert a preference\n1 for NO hidden layers\n2 for
       \
31 TWO hidden layes\n3 for delete graphs\n4 Exit\n>>> "))
32
33     # ============================== NO HIDDEN NODE CASE
34     # ================================================
35     if inp == "1":
36         for i in range(len(regression)):
37             y_train = dataTrainNorm[:, regression[i]]   # Regressand
                feature
38             y_train = np.reshape(y_train, (nSamplesTrain, 1))
39             y_test = dataTestNorm[:, regression[i]]   # Regressand
                feature
40             y_test = np.reshape(y_test, (nSamplesTest, 1))
41
42             # ========================== PLACEHOLDERS AND VARIABLES
43             # Placeholders are input "container": when application runs
                the value of the
44             # inputs are overwritten over placeholders. Then,
                optimization begins cycle by
45             # cycle.
46             # Initial settings
47             tf.set_random_seed(1234)   # in order to get always the
                same results
48             learningRate = 10e-5        # Learning rate for the
                gradient algorithm
49             xPlaceholder = tf.placeholder(tf.float32, None)
50             # desired output: it is just SPACE
51             yPlaceholder = tf.placeholder(tf.float32, None)
52
53             # "Hardware" neural network structure --> NO HIDDEN NODES
54             w = tf.Variable(tf.random_normal(shape=[nFeatures, 1], mean
                =0.0, \
55                 stddev=1.0, dtype=tf.float32, name="weights"))
56             b = tf.Variable(tf.random_normal(shape=[1, 1], mean=0.0, \
57                 stddev=1.0, dtype=tf.float32, name="biases"))
58             # Activation function --> OUTPUT
59             y = tf.matmul(xPlaceholder, w) + b
```

```
60
61              # ============================== OPTIMIZATION STRUCTURE
62              # Objective function is the reduction of square error
63              cost = tf.reduce_sum(tf.squared_difference(y, yPlaceholder,
                    \
64                  name="objective_function"))
65              optim = tf.train.GradientDescentOptimizer(learningRate, \
66                  name="GradientDescent")
67
68              # Minimize the objective function changing w and b
69              optim_op = optim.minimize(cost, var_list=[w, b])
70
71              # Variables initialization
72              init=tf.global_variables_initializer()
73              #--- run the learning machine
74              sess = tf.Session() # Each graph must have its own session
75              sess.run(init)
76
77              # ============================== GRADIENT ALGORITHM
78              for i in range(100000):
79                  # Data Generation
80                  xGen = x_train
81                  yGen = y_train
82                  yGen = np.reshape(yGen, (nSamplesTrain, 1))
83
84                  # Data for feeding placeholder
85                  train_data = {xPlaceholder : xGen, yPlaceholder : yGen}
86                  sess.run(optim_op, feed_dict=train_data)
87
88              # Output of the neural network is evaluated:
89              yEvaluation = y.eval(feed_dict = train_data, session = sess
                    )
90              #yProva = y.eval(feed_dict = test_data, session = sess)
91
92              test_data = {xPlaceholder : x_test}
93              yHat_test = sess.run(y, feed_dict = test_data)
94
95              # ============================== RESULT PLOTS
96              # TRAINING
97              plt.figure()
98              plt.subplot(311)
99              plt.title("NO HIDDEN LAYERS: Regression train for \
100 F0 = "  + str(regression[0]+1))
101             plt.plot(y_train, "ro--", label = "Train")
102             plt.plot(yEvaluation, "bx--", label = "Estimation")
103             plt.grid(which = "major", axis = "both")
```

```
104             plt.legend(), plt.show()
105
106             plt.subplot(312)
107             plt.plot(y_train, yEvaluation, "ro")
108             plt.xlabel("Regressor"), plt.ylabel("Regressand")
109             plt.grid(which = "major", axis = "both")
110             plt.legend(), plt.show()
111
112             plt.subplot(313)
113             plt.hist(y_train - yEvaluation, bins = 50)
114             plt.grid(which = "major", axis = "both"), plt.show()
115
116             # TESTING
117             plt.figure()
118             plt.subplot(311)
119             plt.title("NO HIDDEN LAYERS: Regression test for \
120 F0 = "  + str(regression[0]+1))
121             plt.plot(y_test, "ro--", label = "Test")
122             plt.plot(yHat_test, "bx--", label = "Estimation")
123             plt.grid(which = "major", axis = "both")
124             plt.legend(), plt.show()
125
126             plt.subplot(312)
127             plt.plot(y_test, yHat_test, "ro")
128             plt.xlabel("Regressor"), plt.ylabel("Regressand")
129             plt.grid(which = "major", axis = "both")
130             plt.legend(), plt.show()
131
132             plt.subplot(313)
133             plt.hist(y_test - yHat_test, bins = 50)
134             plt.grid(which = "major", axis = "both"), plt.show()
135
136     # ============================ HIDDEN NODE CASE
137     # ================================================
138     elif inp == "2":
139         for i in range(len(regression)):
140             y_train = dataTrainNorm[:, regression[i]]   # Regressand
                    feature
141             y_train = np.reshape(y_train, (nSamplesTrain, 1))
142             y_test = dataTestNorm[:, regression[i]]   # Regressand
                    feature
143             y_test = np.reshape(y_test, (nSamplesTest, 1))
144
145             tf.set_random_seed(1234)        # in order to get always
                    the same results
146             learningRate = 10e-5
```

```
147             xPlaceholder = tf.placeholder(tf.float32,None)
148             # desired output: it is just SPACE
149             yPlaceholder = tf.placeholder(tf.float32, None)
150
151             # "Hardware" neural network structure --> There are 17
                    HIDDEN NODES:
152             w1 = tf.Variable(tf.random_normal(shape=[nFeatures,
                    hiddenNodes1], \
153                  mean=0.0, stddev=1.0, dtype=tf.float32, name="weights")
                         )
154             b1 = tf.Variable(tf.random_normal(shape=[1, hiddenNodes1],
                    \
155                  mean=0.0, stddev=1.0, dtype=tf.float32, name="biases"))
156             a1 = tf.matmul(xPlaceholder, w1) + b1       # Activation
                    function --> OUTPUT
157             z1 = tf.nn.tanh(a1)     # NON-LINEARITY
158             w2 = tf.Variable(tf.random_normal(shape=[hiddenNodes1,
                    hiddenNodes2], \
159                  mean=0.0, stddev=1.0, dtype=tf.float32, name="weights2
                         "))
160             b2 = tf.Variable(tf.random_normal([1,hiddenNodes2], mean
                    =0.0, \
161                  stddev=1.0, dtype=tf.float32, name="biases2"))
162             a2 = tf.matmul(z1, w2) + b2    # neural network output
163             z2 = tf.nn.tanh(a2)
164             w3 = tf.Variable(tf.random_normal(shape=[hiddenNodes2, 1],
                    \
165                  mean=0.0, stddev=1.0, dtype=tf.float32, name="weights2
                         "))
166             b3 = tf.Variable(tf.random_normal(shape=[1,1], mean=0.0, \
167                  stddev=1.0, dtype=tf.float32, name="biases2"))
168             y = tf.matmul(z2, w3) + b3
169             cost = tf.reduce_sum(tf.squared_difference(y, yPlaceholder,
                         \
170                  name="objective_function"))
171             optim = tf.train.GradientDescentOptimizer(learningRate, \
172                  name="GradientDescent")
173             optim_op = optim.minimize(cost, var_list=[w1, b1, w2, b2,
                    w3, b3])
174             init = tf.initialize_all_variables()
175             sess = tf.Session() # Each graph must have its own session
176             sess.run(init)
177
178             # ================================ GRADIENT ALGORITHM
179             for k in range(100000):
180                 # Data Generation
```

```
                xGen = x_train
                yGen = y_train
                yGen = np.reshape(yGen, (nSamplesTrain, 1))
                # Data for feeding placeholder
                train_data = {xPlaceholder : xGen, yPlaceholder: yGen}
                sess.run(optim_op, feed_dict=train_data)

            # Output of the neural network is evaluated:
            yEvaluation = y.eval(feed_dict = train_data, session = sess
                )

            test_data = {xPlaceholder : x_test}
            yHat_test = sess.run(y, feed_dict = test_data)

            # ========================== RESULT PLOTS
            # TRAINING
            plt.figure()
            plt.subplot(311)
            plt.title("2 HIDDEN LAYERS: Regression train for F0 = " + \
str(regression[0] + 1))
            plt.plot(y_train, "ro--", label = "Train")
            plt.plot(yEvaluation, "bx--", label = "Estimation")
            plt.legend()
            plt.xlabel("case number")
            plt.grid(which = "major", axis = "both")
            plt.legend(), plt.show()

            plt.subplot(312)
            plt.plot(y_train, yEvaluation, "ro")
            plt.xlabel("Regressor"), plt.ylabel("Regressand")
            plt.grid(which = "major", axis = "both")
            plt.legend(), plt.show()

            plt.subplot(313)
            plt.hist(y_train - yEvaluation, bins = 50)
            plt.grid(which = "major", axis = "both"), plt.show()

            # TESTING
            plt.figure()
            plt.subplot(311)
            plt.title("2 HIDDEN LAYERS: Regression test for F0 = " + \
str(regression[0] + 1))
            plt.plot(y_test, "ro--", label = "Test")
            plt.plot(yHat_test, "bx--", label = "Estimation")
            plt.legend()
            plt.xlabel("case number")
```

```
226                 plt.grid(which = "major", axis = "both")
227                 plt.legend(), plt.show()
228 
229                 plt.subplot(312)
230                 plt.plot(y_test, yHat_test, "ro")
231                 plt.xlabel("Regressor"), plt.ylabel("Regressand")
232                 plt.grid(which = "major", axis = "both")
233                 plt.legend(), plt.show()
234 
235                 plt.subplot(313)
236                 plt.hist(y_test - yHat_test, bins = 50)
237                 plt.grid(which = "major", axis = "both"), plt.show()
238 
239 
240     elif inp == "3":
241         plt.close("all")
242         print("Graphs deleted")
243 
244     elif inp == "4":
245         print("Closing...")
246         #plt.close("all")
247         flag = False
248 
249 # This function is used to restore the initial situazion of the network
        : it
250 # blows away all Variables, tensors and placeholders.
251 tf.reset_default_graph()
```

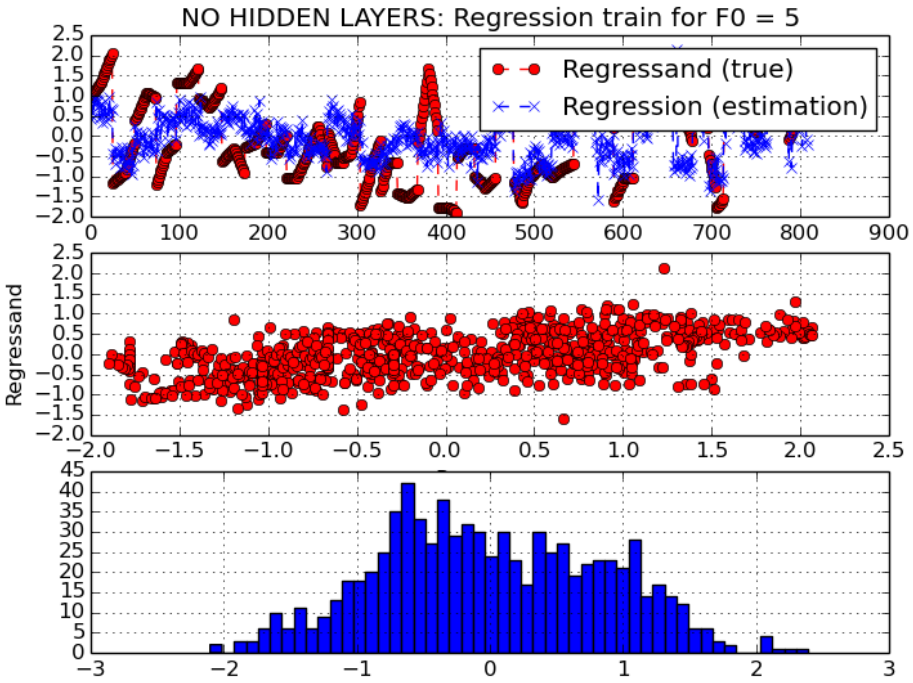


Figure 18: ANN regression F = 5 with no HL

3 Classification

3.1 Introduction

Classification problems focus on identifying to which set of categories a new observation belongs to. As linear regression, classification algorithms are considered instances of supervised learning. From a conceptual point of view, classification techniques try to recognize patterns and regularities in train dataset labeled with true classes. Each class is conceptually summarized by a representative value, called centroid. In general, a new observation belongs to the class at the minimum distance with respect to the others.

Dataset is from UCI Machine learning repository (link dataset). The aim of this exercise is performing 2 classification algorithms and testing their performances. Dataset contains 279 attributes related to ECG track. Last column is contains patient's scores, going from 1 to 16, where 1 suggests healthy patients, from 2 to 15 it indicates different levels of arrythmia, while 16 refers to an unclassified status of the sickness.
The code is placed at the end of clustering section(4.3) because of an easier results readability.

Data cleaning and normalization Since dataset contains some missing or wrong data, a cleaning process is the first thing to do, made by deleting empy features (columns). Next, as it has been said above, normalization must be done in order to get attributes coherent with each other through removing the mean and setting variance to one.

3.2 Binary classification

The first task requires binary classification. Considered classes are 1, which is related to healty patients, and 2, that gathers all sick patients and unclassified ones. From the knowledge of the classes, it is possible to choose the decision function that maps the observed values to the regions. There are several kinds of classification algorithms, and it is important keeping in mind that different decision criteria give rise to different decision regions and, of course, different results.

3.2.1 Minimum distance criterion

The idea behind classification is that the instances are mapped into a plan, a 3D space, or a multidimensional space. Per each istance, a distance is computed per each class. From a conceptual point of view, distances are like geometric displacements between instances and centroids. Minimum distance criterion assigns the current instance to the nearest class: this classifier is also called nearest-neighbour classifier. Pratically, a distances distance vector has been compute per each class, and the analysis focuses on finding all minimum distances.

3.2.2 Bayesian approach

It is a slightly different approach with respect to the previous one. Bayesian approach maximizes the posterior probability and, again, minimizes the distance from the centroid. This technique works with statistically independent Gaussian random variables; for this reason first of all we have to perform K-L transformation in order to "whitening" dataset features and PCA in order to cut away all very low components in terms of variance and to get statistically independent Gaussian

random variables. While minimum distance criterion supposes the prior probabilities equivalent per each feature (0.5 per both classes), Bayesian approach considers the true prior probabilities, so it is needed to compute them, compute their logarithm and subtract to the new distance vector.

3.2.3 Neural network - Tensorflow

Classification has been also performed by using, again, Google framework Tensorflow. The neural network built has two hidden layers: the first one it has 257 hidden nodes, the second one has 128 nodes. The target function consists in reduction of the summation of the squared difference between the estimation given by doctors and the neural network output. The neural network configuration (in terms of iterations number and learning rate) is the same of the linear regression case. Data is already normalized from MATLAB script. Activation function for this neural network is sigmoid: it is defined between 0 and 1 and it is suitable for binary classification. Results seem to be very good in terms of sensitivity, specificity and correct detection (see Conclusions subsection for details). If the iteration number is increased, performances get a little bit better, but on the other side the computation time becomes huge due to the fact that this is the CPU version of TensorFlow, and not the GPU one (which requires much more installation problems).

3.3 Conclusions

Performances for the analysed methods have been evaluated in terms of specificity and sensitivity (and correct detection percentage, in Tensorflow case). Bayes approach behaves better than the MDC. If the threshold for the PCA is increased, sensitivity and specificity are definitely improved. PCA threshold should be a trade off between dataset dimension and performances to be got, because in case of huge datasets, it may be mandatory reducing its dimension. Neural network approach seems to be the best one: but in this case, if learning rate is increased, some problems appears to the output which becomes a NaN vector. While if iteration number is decreased, performances get worse.

	Sensitivity	Specificity
Minimum Distance	68.5%	83.7%
Bayes approach	80%	94.7%
Neural network	94.6%	97.2%

3.4 Classification 16 classes

The same process made above is made for the original 16 classes. Instead of having 2 distances vectors, now there are 16 vectors. Performances are now evaluated in terms of true detection. Figure 19 shows the confusion matrix that groups the detections and the original classes.

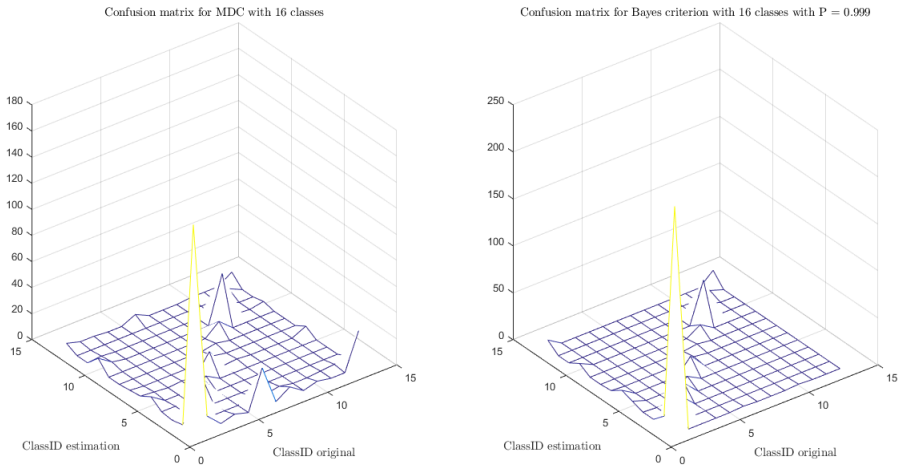


Figure 19: Confusion matrix for 16 classes classification

The plot shows the improvement brought by the bayesian method with respect to MCD. Each instance in the diagonal is a correct detection, elsewhere the estimation is different from the true class. As it has been discussed above, if the percentage of the eigenvalues grows, perfomances improve.

	Minimum Distance	Bayes approach
True detection	67.3%	89.6%

4 Clustering

4.1 K-means introduction

Differentely from classification, clustering techniques is an instance of unsupervised learning. The dataset is still the same of the classification one. Now, it is considered without the known arrythmia classes column. The aim of clustering is to explore data, find some common patterns and group them into clusters with similar characteristics. The analysed clustering algorithm, called K-means, is still distance based: during the assignment, a measurement is associated to a centroid if it is closer to it than to the other ones. During the update, the centroid of the region will be the mean value of all the points associated to that region. It goes on iteratively until the convergence, or a stopping condition.

4.2 Hard K-means 2 clusters

The first task aims to apply hard K-means to the normalized dataset with 2 clusters starting from 2 different initial centroid positions: the classification one and random. Instead of distances, the used criterion is maximum a posteriori, where posterior probability is maximized. It starts by randomly placing k centroids in a multidimensional dimensional space, setting variance values to 1 and prior

probabilities to 1/k per each cluster (initial step), then run through the dataset and find the nearest centroid per each observation (assignment step). Then, within each cluster, we recompute centroid position by averaging all points belonging to it, prior probabilities, distances per each instance and variances (update step). Algorithm converges when every point does not change from two different iterations. The complexity of the algorithm is proportional to the number of iterations times the number of clusters times the number of instances times dataset dimensions. In the end, performances will be assessed by computing the difference of the distance vectors and comparing the clustering with the classification obtained in the previous laboratory.

Stating from classification distances Initial centroid positions are the same positions found in classification problem. It has been observed that if only an iteration is computer, distance vectors are of course the same of the ones obtained with the MDC approach (with prior probabilities locked to 1/k). Results are a little bit poorer than previous case because clustering takes into account unknown features extracted from data, and it could clusters patients in a different way; while doctors are only interested in arrythmia diagnosis, so they may weight more some features than others, creating an important difference between classes and clusters. The number of iteration has been set to 10: increasing this value, it has been seen that performances in terms of sensitivity and specificity do not change anymore.

Starting from random distances For random distances vectors, things change a little bit with respect to the previous case. For a small number of iterations, sensitivity and specificity appears to be very small, as it is reasonable to think about. Each time script runs, result is always different because of the different starting points.

Silhouette plots was done in order to show how a datum is properly assigned to a cluster: one of the two cluster appears to be more "correct" than the other one for both feature 5 and 7 (Figure 22).

In order to show graphically the clusters, a PCA has been performed and data set has been reduced to 2 dimensions: the result is not that good because there is not a clear distintction among the clusters, maybe because of PCA itself. A possible technique to perform a graphical visualization is t-distributed stochastic neighbor embedding (t-SNE).

4.3 K-means 4 clusters

Clustering was repeated following the same algorithm, but considering 4 clusters. Two different initial distance vectors was chosen, and again results.

Listing 11: Arrythmia analysis - Main

```
clear all
close all
clc

% LEGEND:
% arrhythmia         --> original data
% classIdOriginal    --> Original column with patients classes
% y                  --> Feature matrix WITHOUT classes
```

```
% yNorm              --> Feature matrix WITHOUT classes
% pi16               --> Vector containing the prob related to each
    class
% xxxNClassesMDC     --> Related to MINIMUM DISTANCE CRITERION N classes
% xxxNClassesBay     --> Related to BAYES CRITERION N classes
% xxxMeans           --> Matrix ORDERED with the means of the features
    per
                        % each class
set(groot,'DefaultLegendInterpreter','latex')
set(groot,'DefaultTextInterpreter','latex')
% ============================== MATRIX LOADING
    ==============================
load('arrhythmia.mat')

% ============================== DATA PREPARING
    ==============================
arr = arrhythmia;   % Original dataset
nOfPatients = length(arr(:, 1));
maxClassId = max(arr(:, end));
classIdOriginal = arr(:, end);
% Substitution of the last column with 1-2 classes
results = arr(:, end);
results(results > 1) = 2;
arr(:, end) = results;

% This command deletes all colomns that don't carry information (empty)
arr(:, ~any(arr,1)) = [];
class_id2Classes = arr(:, end);     % Class ID (just 2 classes)
y = arr(:, 1:(end - 1));    % Features matrix without class ID colomn
F = length(y(1, :));
meanY = mean(y, 1);         % Vector containing  the MEAN per feature
varY = var(y, 1);           % Vector containing the VARIANCE per
    feature

% NORMALIZATION of the features matrix
meanYMatrix = ones(nOfPatients, 1) * meanY;
varYMatrix = ones(nOfPatients, 1) * varY;
yNorm = (y - meanYMatrix) ./ sqrt(varYMatrix);

indexClass1 = find(class_id2Classes == 1);   % classID 1 indices
indexClass2 = find(class_id2Classes == 2);   % classID 2 indices
y1 = yNorm(indexClass1, :);     % Class 1 matrix, WITHOUT CLASSID
y2 = yNorm(indexClass2, :);     % Class 2 matrix, WITHOUT CLASSID
x1 = mean(y1, 1);   % Row vector of the mean per each feature for class
    1
```

```
48 x2 = mean(y2, 1);   % Row vector of the mean per each feature for class
       2
49
50 % Matrix with x1 and x2: first row contains the mean of the features of
       the
51 % FIRST class, second row contains the mean of the features of the
       SECOND
52 % class:
53 xMeans = [x1; x2];
54
55 % ================================ 2 CLASSES
       ================================
56 % ======================== MINIMUM DISTANCE CRITERION
       ========================
57 eny = diag(yNorm * yNorm');
58 enx = diag(xMeans * xMeans');   % Region centroids
59 dotProd = yNorm * xMeans';
60 [U, V] = meshgrid(enx, eny);
61
62 % Matrix containing distance of EACH patient (row) from each class
63 distance2Classes = U + V - 2*dotProd;
64 [val, ind] = min(distance2Classes, [], 2);  % <-- MIN DISTANCE
       EVALUATION
65 est_class_id_2ClassesMDC = ind;
66
67 % ------------------------ Sensitivity & Specificity
       ------------------------
68 [specificityMDC, sensitivityMDC] = prob2Class(est_class_id_2ClassesMDC,
       ...
69     class_id2Classes);
70
71 figure, subplot(2,1,1)
72 plot(est_class_id_2ClassesMDC, 'o'), hold on, grid on
73 plot(class_id2Classes, '*'), title(['MDC plot 2 classes: Sensitivity =
       ', ...
74     num2str(sensitivityMDC), ' Specificity = ', num2str(specificityMDC)
         ])
75 legend('Class ID estimation', 'Class ID true')
76
77 % ================================ 2 CLASSES
       ================================
78 % ========================== BAYESIAN CRITERION
       ==========================
79 % First of all I evaluate the prior probability for each class.
80 pi1 = length(indexClass1) / nOfPatients;   % Prob that hyp 1 is
       correct
```

```
81  pi2 = length(indexClass2) / nOfPatients;     % Prob that hyp 2 is
        correct
82  R = (1/nOfPatients) * (yNorm') * (yNorm);    % Covariance matrix
83  [U2Classes, Lambda2Classes] = eig(R);  % U2Classes columns are
        EIGENVECTORS
84  Lambdas2Classes = diag(Lambda2Classes);
85  sommaLambdas2Classes = sum(Lambdas2Classes);
86  P2Classes = 0.99;    % We take the 99% of the total number of
        eigenvalues
87  somm2Classes = 0;
88  ii = 0;
89
90  while somm2Classes < P2Classes * sommaLambdas2Classes
91      ii = ii + 1;
92      somm2Classes = somm2Classes + Lambdas2Classes(ii);
93  end
94
95  UL2Classes = U2Classes(:, 1:ii);     % I just take the 99% of
        eigenvectors
96  z2Classes = yNorm * UL2Classes;
97
98  % z2Classes is the projection of original features in an new orthoGONAL
99  % space. We need to re-normalize again to get orthoNORMAL vectors of
100 % features.
101 % ------------------------- z normalization
        ------------------------------------
102 zMean2Classes = ones(nOfPatients, 1) * mean(z2Classes, 1);
103 zVar2Classes = ones(nOfPatients, 1) * var(z2Classes, 1);
104 % zNorm2Classes is now the orthonormal set of features:
105 zNorm2Classes = (z2Classes - zMean2Classes) ./ sqrt(zVar2Classes);
106
107 % I now perform again the minimum distance criterion with the new set
        of
108 % features:
109 w1 = mean(zNorm2Classes(indexClass1, :), 1);
110 w2 = mean(zNorm2Classes(indexClass2, :), 1);
111 wMeans = [w1; w2];
112 eny = diag(zNorm2Classes * zNorm2Classes');
113 enx = diag(wMeans * wMeans');
114 dotProd = zNorm2Classes * wMeans';
115 [U, V] = meshgrid(enx, eny);
116 % This matrix contains the distance of every patient (row) from the
117 % relative class (column):
118 distance2ClassesBay = U + V - 2*dotProd;
119 % est_class_id_2ClassesBay will be a vector containing the estimation
        of
```

```
120 % the minimum distance of each patient (row) from class 1 (column 1)
    and
121 % class 2 (column 2):
122
123 pi1Vector = ones(nOfPatients, 1) * (2 * log(pi1));
124 pi2Vector = ones(nOfPatients, 1) * (2 * log(pi2));
125 class1 = distance2ClassesBay(:, 1) - pi1Vector;
126 class2 = distance2ClassesBay(:, 2) - pi2Vector;
127 distance2ClassesBay = [class1 class2];
128 [value, est_class_id_2ClassesBay] = min(distance2ClassesBay, [], 2);
129 % [value, est_class_id_2ClassesBay] = min(distance2ClassesBay, [], 2);
130 % est_class_id_2ClassesBay = est_class_id_2ClassesBay';
131
132 % ---------------------- Sensitivity & Specificity ----------------------
133 [specificity2ClassesBay, sensitivity2ClassesBay] = prob2Class(
    est_class_id_2ClassesBay, ...
134     class_id2Classes);
135
136 subplot(2,1,2), plot(est_class_id_2ClassesBay, 'o'), hold on, grid on
137 plot(class_id2Classes, '*'), title(['Bayes criterion plot 2 classes:
    Sensitivity = ', ...
138     num2str(sensitivity2ClassesBay), ' Specificity = ', ...
139     num2str(specificity2ClassesBay)])
140 legend('Class ID estimation', 'Class ID true')
141
142 % ============================ 16 CLASSES
    ================================
143 % ====================== MINIMUM DISTANCE CRITERION
    ========================
144 % Prior probability for each class is stored in pi16Classes:
145 for aa = 1:maxClassId
146     indexes = find(classIdOriginal == aa);
147     pi16Classes(aa) = length(indexes) / nOfPatients;
148     xMean16Classes(aa, :) = mean(yNorm(indexes, :), 1);
149 end
150
151 eny16Classes = diag(yNorm * yNorm');
152 enx16Classes = diag(xMean16Classes * xMean16Classes');
153 dotProd16Classes = yNorm * xMean16Classes';
154 [U16Classes, V16Classes] = meshgrid(enx16Classes, eny16Classes);
155 % distance16Classes stores the distance of each patient from each class
156 distance16Classes = U16Classes + V16Classes - 2*dotProd16Classes;
157
158 % Performance evaluation: I distinguish the true detection from the
    false
159 % detection:
```

```
160 trueDetection16ClassesMDC = 0;
161 falseDetection16ClassesMDC = 0;
162 [minVal, est_class_id_16ClassesMDC] = min(distance16Classes, [], 2);
163 diff16Classes = est_class_id_16ClassesMDC - classIdOriginal;
164 trueDetection16ClassesMDC = length(find(diff16Classes == 0));
165 falseDetection16ClassesMDC = length(find(diff16Classes ~= 0));
166
167 percTrueDetection16ClassesMDC = trueDetection16ClassesMDC / nOfPatients
        ;
168 percFalseDetection16ClassesMDC = falseDetection16ClassesMDC /
        nOfPatients;
169
170 figure, plot(est_class_id_16ClassesMDC, 'o'), hold on, grid on
171 plot(classIdOriginal, '*'), title(['MDC plot 16 classes: True detection
        = ', ...
172     num2str(percTrueDetection16ClassesMDC), ' False detection = ', ...
173     num2str(percFalseDetection16ClassesMDC)])
174 legend('Class ID estimation', 'Class ID true')
175
176 % ========================== 16 CLASSES
        ==================================
177 % ========================== BAYESIAN CRITERION
        ============================
178 % Covariance matrix for the original dataset:
179 R16Classes = (1/nOfPatients) * (yNorm') * (yNorm);
180 [U16Classes, Lambda16Classes] = eig(R16Classes);
181 Lambdas16Classes = diag(Lambda16Classes);
182 sommaLambdas16Classes = sum(Lambdas16Classes);
183 P16Classes = 0.999;
184 somm16Classes = 0;
185 ii16Classes = 0;
186
187 while somm16Classes < P16Classes * sommaLambdas16Classes
188     ii16Classes = ii16Classes + 1;
189     somm16Classes = somm16Classes + Lambdas16Classes(ii16Classes);
190 end
191
192 UL16Classes = U16Classes(:, 1:ii16Classes);
193 z16Classes = yNorm * UL16Classes;
194 zMean16Classes = ones(nOfPatients, 1) * mean(z16Classes, 1);
195 zVar16Classes = ones(nOfPatients, 1) * var(z16Classes, 1);
196 zNorm16Classes = (z16Classes - zMean16Classes) ./ ...
197     sqrt(zVar16Classes);
198
199 % Matrix sorting:
200 for aa = 1:maxClassId
```

```
201         indici16Classes = find(classIdOriginal == aa);
202         wMean16ClassesBay(aa, :) = mean(zNorm16Classes(indici16Classes, :),
                1);
203     end
204
205     eny16ClassesBay = diag(zNorm16Classes * zNorm16Classes');
206     enx16ClassesBay = diag(wMean16ClassesBay * wMean16ClassesBay');
207     dotProd16ClassesBay = zNorm16Classes * wMean16ClassesBay';
208     [U16ClassesBay, V16ClassesBay] = meshgrid(enx16ClassesBay,
            eny16ClassesBay);
209     distanceBay16ClassesBay = U16ClassesBay + V16ClassesBay - 2*
            dotProd16ClassesBay;
210     pi16Matrix = ones(nOfPatients, 1) * (2 * log(pi16Classes));
211     bayesDist = distanceBay16ClassesBay - pi16Matrix;
212     [minimum, est_class_id_16ClassesBay] = min(bayesDist, [], 2);
213
214     probC = 0;
215
216     for gg = 1:16
217         estimatedPatients = 0;
218         for oo = 1:nOfPatients
219             if est_class_id_16ClassesBay(oo) == gg && classIdOriginal(oo)
                    == gg
220                 estimatedPatients = estimatedPatients + 1;
221             end
222         end
223         truePatients = length(find(classIdOriginal == gg));
224         if truePatients == 0
225         else
226             % Probability of right decision:
227             probC = probC + (estimatedPatients / truePatients) *
                    pi16Classes(gg);
228         end
229     end
230     % ------------------------------------ FIGURES
            ------------------------------------------
231     figure, subplot(2,1,1)
232     plot(est_class_id_16ClassesMDC, '*'), hold on, grid on
233         plot(classIdOriginal, 'o'), legend('ClassID estimation', 'ClassID
                true')
234     title(['Class detection MDC plot: true detection = ', ...
235         num2str(percTrueDetection16ClassesMDC * 100), ' %'])
236
237     subplot(2,1,2)
238     plot(est_class_id_16ClassesBay, '*'), hold on, grid on
```

```
239     plot(classIdOriginal, 'o'), legend('ClassID estimation', 'ClassID
            true')
240 title(['Class detection Bayesian criterion plot: true detection = ',
        ...
241     num2str(probC * 100), ' %'])
242
243 confMat16ClassesMDC = confusionmat(classIdOriginal,
        est_class_id_16ClassesMDC);
244 confMat16ClassesBay = confusionmat(classIdOriginal,
        est_class_id_16ClassesBay);
245
246 figure, subplot(1,2,1),
247 mesh(confMat16ClassesMDC), title('Confusion matrix for MDC with 16
        classes')
248 xlabel('ClassID original'), ylabel('ClassID estimation')
249 subplot(1,2,2), mesh(confMat16ClassesBay)
250 title(['Confusion matrix for Bayes criterion with 16 classes with P = '
        ,...
251     num2str(P16Classes)])
252 xlabel('ClassID original'), ylabel('ClassID estimation')
253
254 % ================================ HARD K-MEANS
        ============================
255 % ================================ 2 CLUSTERS
        ================================
256 % Starting from xMeans already evaluated above.
257 % LO SCOPO E' QUELLO DI MINIMIZZARE LA VARIANZA INTRA-CLUSTER PER OGNI
258 % CLUSTER: OGNI CLUSTER VIENE IDENTIFICATO DA UN CENTROIDE
259 % L'initial step dovrebbe essere una scelta casuale di K clusters.
        Quindi
260 % inizializzare un vettore di nOfPatients righe per K colonne dove ogni
261 % elemento contiene la distanza!!!
262 yNorm;        % <-- Normalized data
263 k = 2;
264 x_k = xMeans;          % <-- INITIAL GUESS: step 1
265 sigma_k = ones(1, k);
266 pi_k = [1/k 1/k];
267 count = 0;
268
269 while count < 10
270     for ii = 1:k
271         matrix = yNorm - (ones(nOfPatients, 1) * x_k(ii, :));
272         norma = sqrt(sum(abs(matrix).^2,2));
273         MAP_values(:, ii) = pi_k(ii) .* exp(-(norma) ./ (2*sigma_k(ii))
                ) / ...
274             (2*pi*sigma_k(ii))^(F/2);
```

```
275         end
276         % Assignment step
            ------------------------------------------------------------------
277         [p_, assignm_step] = max(MAP_values, [], 2);
278
279         % Update step
            ------------------------------------------------------------------
280         for ii = 1:k
281             new_indexes = find(assignm_step == ii);
282             N(ii) = length(new_indexes);
283             pi_k(ii) = N(ii) / nOfPatients; % prior probabilities UPDATED
284             w = yNorm(new_indexes, :);
285             if ii == 1
286                 w_1 = yNorm(new_indexes, :);
287             else
288                 w_2 = yNorm(new_indexes, :);
289             end
290             x_k(ii, :) = mean(w, 1);     % x_k UPDATED
291             matrix2 = w - (ones(N(ii), 1) * x_k(ii, :));
292             norma2 = sqrt(sum(abs(matrix2).^2,2));
293             norma2 = sum(norma2);
294             sigma_k(ii) = norma2 / (F * (N(ii) - 1));    % variance UPDATED
295 %           if count == 5
296 %               err1(ii, :) = abs(xMeans(ii, :) - x_k(ii, :));
297 %           end
298         end
299         count = count + 1;
300     end
301
302     [specificity_kM, sensitivity_kM] = prob2Class(assignm_step, ...
303         class_id2Classes);
304     figure, plot(assignm_step, 'o'), hold on, grid on
305     plot(class_id2Classes, '*'), title(['k-means 2 classes: Sensitivity = '
            , ...
306         num2str(sensitivity_kM), ' Specificity = ', num2str(specificity_kM)
                ])
307     legend('Class ID estimation', 'Class ID true')
308
309     figure, subplot(2,1,1)
310     plot(xMeans(1, :)), hold on, grid on, plot(x_k(1, :))
311     title('Difference between initial and final distances - Cluster 1')
312     legend('Initial distances', 'Final distances')
313     subplot(2,1,2)
314     plot(xMeans(2, :)), hold on, grid on, plot(x_k(2, :))
315     title('Difference between initial and final distances - Cluster 2')
316     legend('Initial distances', 'Final distances')
```

```
317
318 figure
319 % This graph evaluates the consistance of clusters: it tells how well
      each
320 % object lies within each cluster.
321 [silh2, h] = silhouette(yNorm, assignm_step, 'cityblock');
322 grid on
323 savg = grpstats(silh2,assignm_step);
324 title('Silhouette plot - From classification results')
325 %
326 % [coeffy1, scorey1] = pca(y1);
327 % [coeffy1w, scorey1w] = pca(w_1);
328 % [coeffy2, scorey2] = pca(y2);
329 % [coeffy2w, scorey2w] = pca(w_2);
330 % numDimen = 2;
331 % scorey1red = scorey1(:, 1:numDimen);
332 % a = scorey1red;
333 % scorey1redw = scorey1w(:, 1:numDimen);
334 % media1Init = mean(scorey1red, 1);
335 % media1Final = mean(scorey1redw, 1);
336 % scorey2red = scorey2(:, 1:numDimen);
337 % b = scorey2red;
338 % scorey2redw = scorey2w(:, 1:numDimen);
339 % media2Init = mean(scorey2red, 1);
340 % media2Final = mean(scorey2redw, 1);
341 %
342 % figure, plot(scorey1red(:, 1), scorey1red(:, 2), 'rx', ...
343 %     scorey2red(:, 1), scorey2red(:, 2), 'bo'), grid on
344 % title('Patient clustering'), xlabel('x'), ylabel('y')
345 %
346 % figure, plot(media1Init(:, 1), media1Init(:, 2), 'kx'), ...
347 %     hold on, plot(media2Init(:, 1), media2Init(:, 2), 'ko')
348 % hold on, plot(media1Final(:, 1), media1Final(:, 2), 'rx'), ...
349 %     hold on, plot(media2Final(:, 1), media2Final(:, 2), 'ro'), grid
      on
350 % legend('Initial cluster 1', 'Initial cluster 2', 'Final cluster 1',
      ...
351 %     'Final cluster 2', 'Location', 'Best'), xlabel('x'), ylabel('y')
352 % title('Centroid positions - From classification results')
353
354 % Clustering starting from random distances vectors
      ----------------------
355 yNorm;          % <-- Normalized data
356 k = 2;
357 x_k = rand(k, F);          % <-- INITIAL GUESS: step 1
358 sigma_k = ones(1, k);
```

```
pi_k = [1/k 1/k];
count = 0;

while count < 100
    for ii = 1:k
        matrix = yNorm - (ones(nOfPatients, 1) * x_k(ii, :));
        norma = sqrt(sum(abs(matrix).^2,2));    % Norma della
            situazione
        MAP_values(:, ii) = pi_k(ii) .* exp(-(norma) ./ (2*sigma_k(ii))
            ) / ...
             (2*pi*sigma_k(ii))^(F/2);
    end
    % Assignment step
    ----------------------------------------------------------------
    % MAX ITERA SULLE COLONNE QUINDI OTTENIAMO IL MASSIMO PER OGNI RIGA
    [p_, assignm_step] = max(MAP_values, [], 2);

    % Update step
    ----------------------------------------------------------------
    for ii = 1:k
        new_indexes = find(assignm_step == ii);
        N(ii) = length(new_indexes);
        pi_k(ii) = N(ii) / nOfPatients; % prior probabilities UPDATED
        w = yNorm(new_indexes, :);
        if ii == 1
            w_1 = yNorm(new_indexes, :);
        else
            w_2 = yNorm(new_indexes, :);
        end
        x_k(ii, :) = mean(w, 1);     % x_k UPDATED
        matrix2 = w - (ones(N(ii), 1) * x_k(ii, :));
        norma2 = sqrt(sum(abs(matrix2).^2,2));
        norma2 = sum(norma2);
        sigma_k(ii) = norma2 / (F * (N(ii) - 1));     % variance UPDATED
    end
    count = count + 1;

end

[specificity_kM, sensitivity_kM] = prob2Class(assignm_step, ...
    class_id2Classes);
figure, plot(assignm_step, 'o'), hold on, grid on
plot(class_id2Classes, '*'), title(['k-means 2 clusters: Sensitivity =
    ', ...
    num2str(sensitivity_kM), ' Specificity = ', num2str(specificity_kM)
        ])
```

```
399 legend('Class ID estimation', 'Class ID true')
400
401 figure, subplot(2,1,1)
402 plot(xMeans(1, :)), hold on, grid on, plot(x_k(1, :))
403 title('Difference between initial and final distances (random) -
      Cluster 1')
404 legend('Initial distances', 'Final distances')
405 subplot(2,1,2)
406 plot(xMeans(2, :)), hold on, grid on, plot(x_k(2, :))
407 title('Difference between initial and final distances (random)- Cluster
      2')
408 legend('Initial distances', 'Final distances')
409
410 figure
411 [silh2, h] = silhouette(yNorm, assignm_step, 'cityblock');
412 grid on
413 savg = grpstats(silh2,assignm_step);
414 title('Silhouette plot - From random initial distances')
415
416 % [coeffy1, scorey1] = pca(y1);
417 % [coeffy1w, scorey1w] = pca(w_1);
418 % [coeffy2, scorey2] = pca(y2);
419 % [coeffy2w, scorey2w] = pca(w_2);
420 % numDimen = 2;
421 % scorey1red = scorey1(:, 1:numDimen);
422 % scorey1redw = scorey1w(:, 1:numDimen);
423 % media1Init = mean(scorey1red, 1);
424 % media1Final = mean(scorey1redw, 1);
425 % scorey2red = scorey2(:, 1:numDimen);
426 % scorey2redw = scorey2w(:, 1:numDimen);
427 % media2Init = mean(scorey2red, 1);
428 % media2Final = mean(scorey2redw, 1);
429 %
430 % figure, plot(scorey1red(:, 1), scorey1red(:, 2), 'rx', ...
431 %     scorey2red(:, 1), scorey2red(:, 2), 'bo'), grid on, hold on,
432 % plot(a(:, 1), a(:, 2), 'gx', b(:, 1), b(:, 2), 'yo')
433 %
434 % title('Patient clustering'), xlabel('x'), ylabel('y')
435 % figure, plot(media1Init(:, 1), media1Init(:, 2), 'kx'), ...
436 %     hold on, plot(media2Init(:, 1), media2Init(:, 2), 'ko')
437 % hold on, plot(media1Final(:, 1), media1Final(:, 2), 'rx'), ...
438 %     hold on, plot(media2Final(:, 1), media2Final(:, 2), 'ro'), grid
      on
439 % legend('Initial cluster 1', 'Initial cluster 2', 'Final cluster 1',
      ...
440 %     'Final cluster 2', 'Location', 'Best'), xlabel('x'), ylabel('y')
```

```
441 % title('Centroid positions - From random initial distances')
442
443 % ================================== HARD K-MEANS
      ==========================
444 % ================================== 4 CLUSTERS
      ==========================
445 yNorm;       % <-- Normalized data
446 k = 4;
447 x_k1 = rand(k, F);          % <-- INITIAL GUESS: step 1
448 x_k2 = rand(k, F);
449 sigma_k1 = ones(1, k);
450 sigma_k2 = ones(1, k);
451 pi_k1 = [1/k 1/k 1/k 1/k];
452 pi_k2 = [1/k 1/k 1/k 1/k];
453 count = 0;
454
455 while count < 10
456     for ii = 1:k
457         matrix1 = yNorm - (ones(nOfPatients, 1) * x_k1(ii, :));
458         norma1 = sqrt(sum(abs(matrix1).^2,2));
459         MAP_values1(:, ii) = pi_k1(ii) .* exp(-(norma1) ./ (2*sigma_k1(
              ii))) / ...
460             (2*pi*sigma_k1(ii))^(F/2);
461
462         matrix2 = yNorm - (ones(nOfPatients, 1) * x_k2(ii, :));
463         norma2 = sqrt(sum(abs(matrix2).^2,2));
464         MAP_values2(:, ii) = pi_k2(ii) .* exp(-(norma2) ./ (2*sigma_k2(
              ii))) / ...
465             (2*pi*sigma_k2(ii))^(F/2);
466     end
467     % Assignment step
        ---------------------------------------------------------------
468     % MAX ITERA SULLE COLONNE QUINDI OTTENIAMO IL MASSIMO PER OGNI RIGA
469     [p_, assignm_step1] = max(MAP_values1, [], 2);
470     [p_, assignm_step2] = max(MAP_values2, [], 2);
471
472     % Update step
        ---------------------------------------------------------------
473     for ii = 1:k
474         new_indexes1 = find(assignm_step1 == ii);
475         N1(ii) = length(new_indexes1);
476         pi_k1(ii) = N1(ii) / nOfPatients; % prior probabilities UPDATED
477         w1 = yNorm(new_indexes1, :);
478         x_k1(ii, :) = mean(w1, 1);     % x_k UPDATED
479         matrix12 = w1 - (ones(N1(ii), 1) * x_k1(ii, :));
480         norma12 = sqrt(sum(abs(matrix12).^2,2));
```

```
481             norma12 = sum(norma12);
482             sigma_k1(ii) = norma12 / (F * (N1(ii) - 1));    % variance
                    UPDATED
483
484             new_indexes2 = find(assignm_step2 == ii);
485             N2(ii) = length(new_indexes2);
486             pi_k2(ii) = N2(ii) / nOfPatients; % prior probabilities UPDATED
487             w2 = yNorm(new_indexes2, :);
488             x_k2(ii, :) = mean(w2, 1);    % x_k UPDATED
489             matrix22 = w2 - (ones(N2(ii), 1) * x_k2(ii, :));
490             norma22 = sqrt(sum(abs(matrix22).^2,2));
491             norma22 = sum(norma22);
492             sigma_k2(ii) = norma22 / (F * (N2(ii) - 1));    % variance
                    UPDATED
493         end
494         count = count + 1;
495     end
496
497     figure, subplot(1,2,1)
498     [silh2, h] = silhouette(yNorm, assignm_step1, 'cityblock');
499     grid on
500     savg = grpstats(silh2,assignm_step1);
501     title('Silhouette plot - From random initial distances')
502
503     subplot(1,2,2)
504     [silh2, h] = silhouette(yNorm, assignm_step2, 'cityblock');
505     grid on
506     savg = grpstats(silh2,assignm_step2);
507     title('Silhouette plot - From random initial distances')
```

Listing 12: 2 Classes probabilities

```
1   function [specificity, sensitivity] = prob2Class(estim, trueV)
2
3       truePositive = 0;
4       trueNegative = 0;
5       falsePositive = 0;
6       falseNegative = 0;
7       nOfPatients = length(estim(:, 1));
8
9       for ii = 1:nOfPatients
10          if estim(ii) == 1 && trueV(ii) == 1
11              trueNegative = trueNegative + 1;
12          elseif estim(ii) == 2 && trueV(ii) == 2
13              truePositive = truePositive + 1;
14          elseif estim(ii) == 2 && trueV(ii) == 1
15              falsePositive = falsePositive + 1;
```

```
16            elseif estim(ii) == 1 && trueV(ii) == 2
17                falseNegative = falseNegative + 1;
18            end
19        end
20
21        sensitivity = truePositive / (truePositive + falseNegative);
22    %     falseNegativeProb = 1 - sensitivity;
23        specificity = trueNegative / (trueNegative + falsePositive);
24    %     falsePositiveProb = 1 - specificity;
25
26    end
```

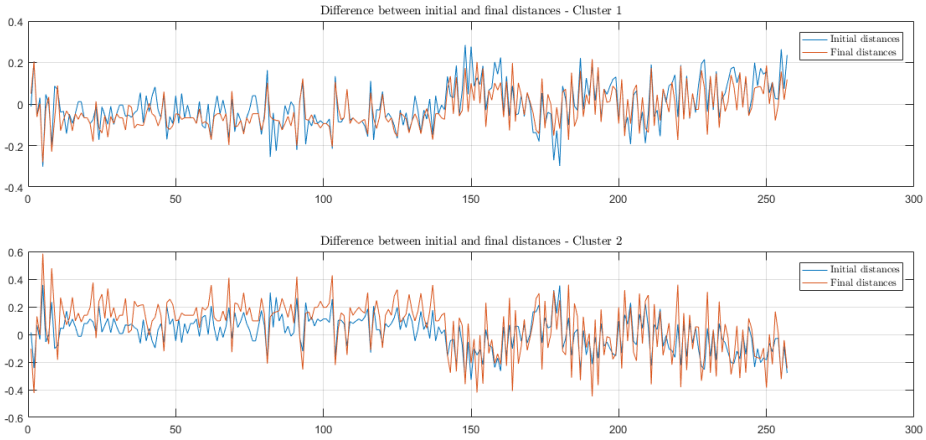


Figure 20: Hard K-means - distance vectors from classification ones

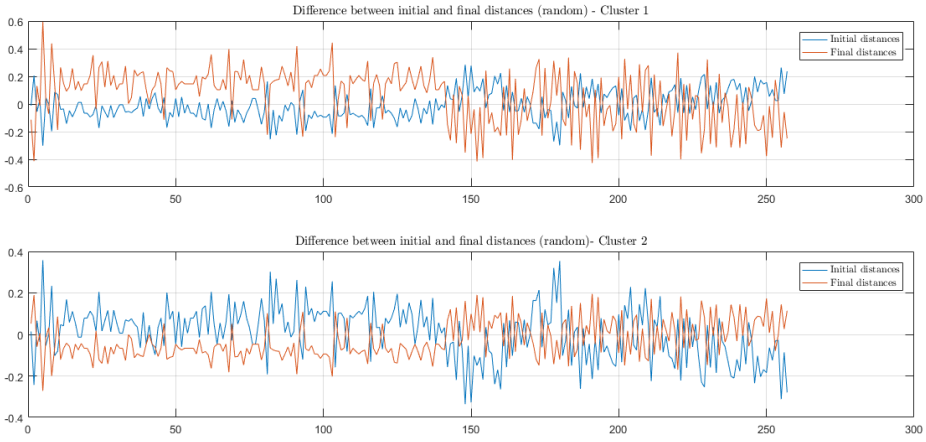


Figure 21: Hard K-means - distance vectors from random ones

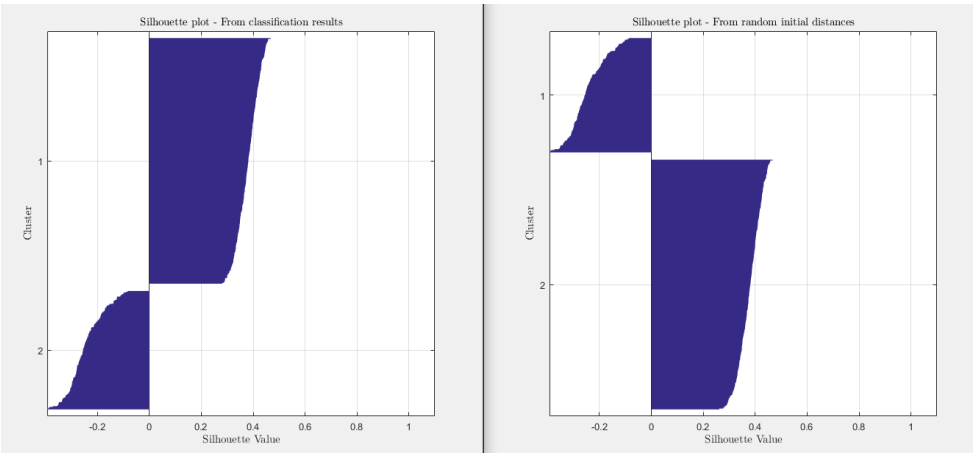


Figure 22: Silhouette plot for 2 clusters

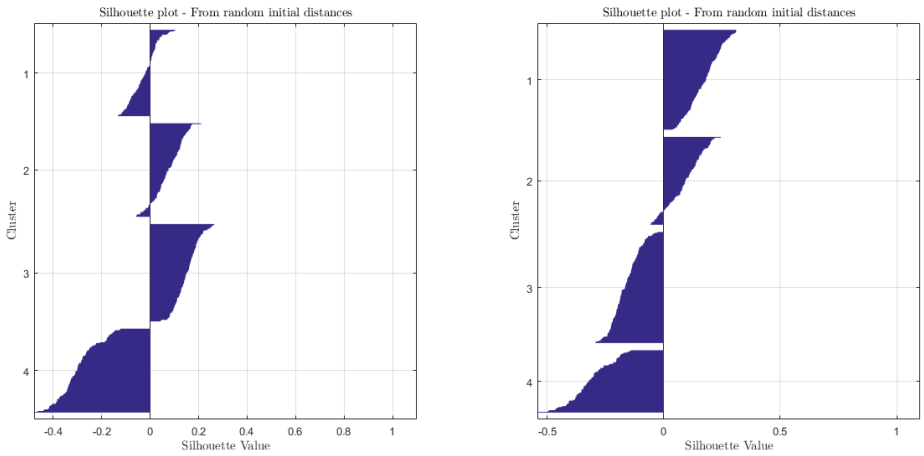


Figure 23: Silhouette plot for 4 clusters

5 Hierarchical trees

5.1 Introduction

The aim of this exercise deals with the application of hierarchical clustering on chronic kidney disease UCI dataset. Hierarchical clustering focuses on creation of a clusters structure. Basically, the used strategy is "agglomerative": it has a bottom up approach. It means that, initially, each observation forms a cluster; then, per each two observations, a distance is computed. The two minimum distance observations will be clustered in the same object, and so on until a unique cluster is got (root). This method has basically two main advantages:

- No need for clusters initial guess
- Dendrogram (a binary tree) allows to choose the suitable depth and change the number of clusters.

Distances between objects is a very crucial element of hierarchical clustering because there are a lot of criteria which can be used, and sometimes may produce very different results. Performances of different clusterings are measured by evaluating the sum of squared error: the less it is, the better is the result.

For this exercise, this dataset presents some non-numerical features, such that it is not possible to perform the canonical operations. For this reason, each of the nominal features has to be converted into numerical values.

5.2 Hierarchical clustering

On the obtained dataset, a hierarchical clusering has been performed by using a set of MATLAB functions:

- pdist: returns a vector containing the distance bewteen each two observations. A parameter is to be set in order to choose what kind of distance criterion
- linkage: generates the tree specifying how to measure the distances between clusters
- dendrogram: shows the resulting tree generated by linkage function
- cluster: returns the vector containing the resulting cluster per each observation

In this analysis, pdist distance is the default one (euclidean distance). While four different linkage distance parameters was choosen: single, average, centroid and complete. For all of them, clustering results are exactly the same, with a correct detection with respect the doctors classification of 80.5%, so there is not any difference in terms of error probability. Graph 24 show the different dendrograms.

5.3 Hierarchical classification

For hierarchical classification, was used "fitctree" function which allows binary classification decision tree for multiclass classification (see graph 25). From the initial dataset, the most meaningful feature are used to split it into subranges that generate new branches of the tree, and so on until all features

are analyzed. It should be convenient to perform Karhunen-Loeve decomposition in order to get some uncorrelated features, but due to some NaN entry, this is not possible. However, the tree gives the rules for the decision regions: in this case the correct detection probability is 82.5%. Even if this evaluation is a little bit compromised by the presence of some "NaN" entries.
Click 5.3 for checking the used code.

Listing 13: Hierarchical trees

```
clear all
close all
clc

% ============================== DATA LOADING ================================
%  ============================================================================

data = load('data2.mat');
chron = data.chron;
N = length(chron(:, 1)); % Number of patients
F = length(chron(1, :));  % Number of features
keylist={'normal','abnormal','present','notpresent','yes','no','good',
    ...
    'poor','ckd','notckd','?',''};
keymap=[0,1,0,1,0,1,0,1,2,1,NaN,NaN];

for ii = 1:N
    for aa = 1:F
        c = strtrim(chron{ii, aa});
        % check stores the comparison vector between c and keylist
        check = strcmp(c, keylist);
        if sum(check) == 0
            b(ii, aa) = str2double(c);
        else
            % if there is a match between keylist and check, substitute
                the
            % corresponding value
            index = find(check == 1);   % Indice corrispondente alla
                keylist
            b(ii, aa) = keymap(index);
        end
    end
end

% ============================ HIERARCHICAL CLUSTERING ========================
%  ============================================================================
```

```
32 %
     ==========================================================================

33 k = 2;
34 est = chron(:, end);
35 kidney = b(:, 1:(end-1));
36 numEst = b(:, end);
37
38 distance = pdist(kidney);
39 tree1 = linkage(distance);
40 tree2 = linkage(distance, 'average');
41 tree3 = linkage(distance, 'centroid');
42 tree4 = linkage(distance, 'complete');
43 T1 = cluster(tree1, 'maxclust', k);
44 T2 = cluster(tree2, 'maxclust', k);
45 T3 = cluster(tree3, 'maxclust', k);
46 T4 = cluster(tree4, 'maxclust', k);
47 p = 0;
48
49 error1 = T1 - b(:, end);
50 det1 = (length(find(error1 == 0)) / N) * 100;
51 error2 = T2 - b(:, end);
52 det2 = (length(find(error2 == 0)) / N) * 100;
53 error3 = T3 - b(:, end);
54 det3 = (length(find(error3 == 0)) / N) * 100;
55 error4 = T4 - b(:, end);
56 det4 = (length(find(error4 == 0)) / N) * 100;
57
58 figure, subplot(2,2,1), dendrogram(tree1, p),
59 title(['single - true detection = ', num2str(det1), '%'])
60 subplot(2,2,2), dendrogram(tree2, p), title(['average - true detection
     = ', ...
61     num2str(det2), '%'])
62 subplot(2,2,3), dendrogram(tree3, p), title(['centroid - true detection
     = ', ...
63     num2str(det3), '%'])
64 subplot(2,2,4), dendrogram(tree4, p), title(['complete - true detection
     = ', ...
65     num2str(det4), '%'])
66
67 % figure, plot(error1), grid on
68 % figure, plot(error2), grid on
69 % figure, plot(error3), grid on
70 % figure, plot(error4), grid on
71 % ========================= HIERARCHICAL CLASSIFICATION
     ======================
```

```
72 %
   ==========================================================================

73 tc = fitctree(kidney, est);
74 view(tc, 'Mode', 'graph');
75
76 for aa = 1:N
77     if kidney(aa, 15) < 13.05
78         if kidney(aa, 16) < 44.5
79             classification(aa) = 2;
80         else
81             classification(aa) = 1;
82         end
83     else
84         if kidney(aa, 3) < 1.0175
85             classifcation(aa) = 2;
86         else
87             if kidney(aa, 4) < 0.5
88                 classification(aa) = 1;
89             else
90                 classification(aa) = 2;
91             end
92         end
93     end
94 end
95
96 correctProb = classification' - numEst;
97 correctProb = length(correctProb(correctProb == 0)) / N;
```

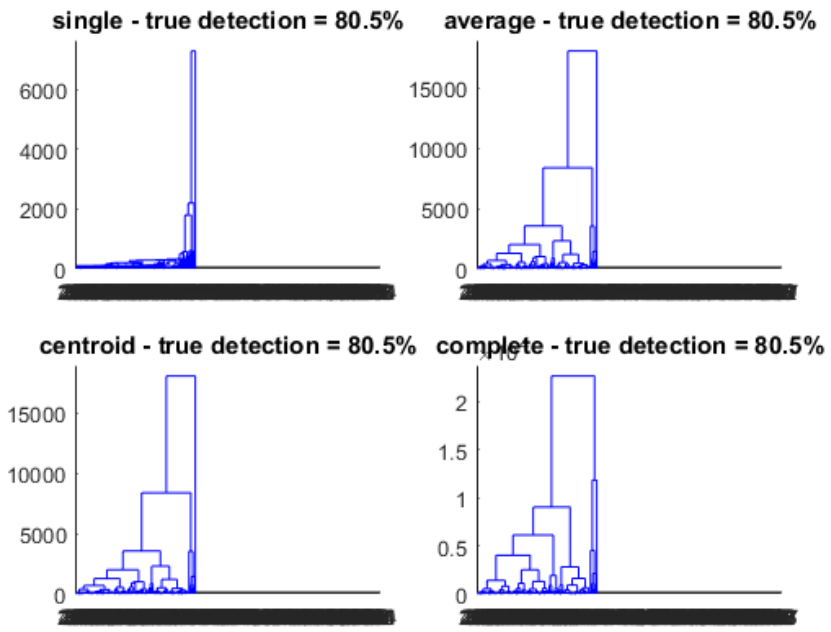


Figure 24: Dendrograms

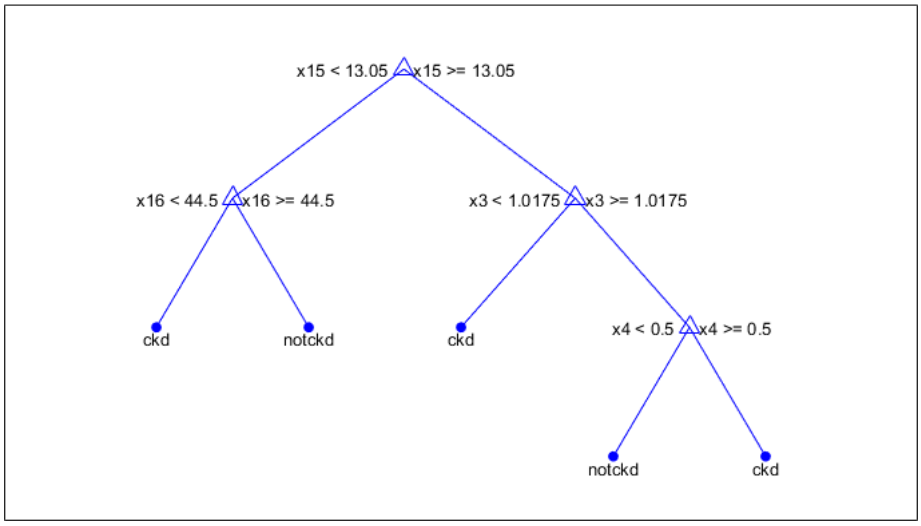


Figure 25: Decision tree

6 Neural networks: Tensorflow

6.1 Introduction

This exercise recalls the very first one: dataset is about Parkinson disease affected patients' voice. Regression is performed on the same features of the first section, but now has been used a neural network built thanks to a Google open source library called Tensorflow. The core of Tensorflow approach is divided in two parts: building and runing the computational graph.
Normalized data was imported thanks to the Scipy library, and all parameters have been set equal to the MATLAB case in order to compare obtained results.

List of Figures

Listings