LABORATORY EXPERIENCE #2
(PCA and PCR)

Introduction

In this laboratory it is required to repeat the previous experiment performing **Principal Component Regression (PCR),** an algorithm that implements feature extraction through **Principal Component Analysis (PCA)** before proceeding with the regression phase.
Feature extraction allows to select a subset of the features (measurements) used in the regression in order to keep the ones that give the greater contribution, or to avoid using the ones that are difficult or costly to obtain, and anyway to **reduce the dimension** of the problem and the contribution of eventual **outliers** in the dataset.

Data preparation

The same procedure as Laboratory #1.

Algorithms implementation and results

The algorithm requires to set the parameter related to the percentage of variance that we want to keep after rescaling the regressor. The variances are given by the eigenvalues of the covariance matrix Rx of the regressor and they are indicators of the amount of contribution brought by each feature to the prediction of the unknown quantity.
The **feature extraction** algorithm will select the first L features which eigenvalues allow to reach the objective percentage. Decreasing the value of this parameter is it possible to decrease significantly the number of features to be evaluated in the regression, but the mean value of the error will increase consequently: a percentage of p=90% allows to use only 3 features in the PCR algorithm, but the error in the prediction is ten times bigger than the error using p=99%, which allows to reduce the problem using only 8 features rather then 17, and giving performances comparable to the algorithms performed in the previous laboratory experience.

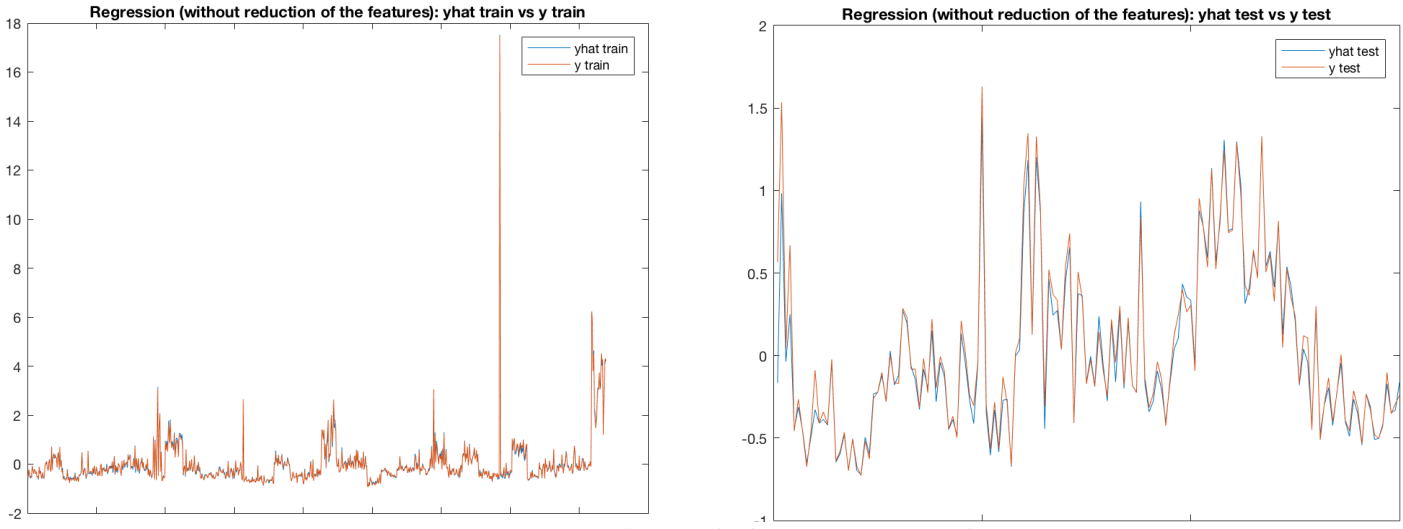
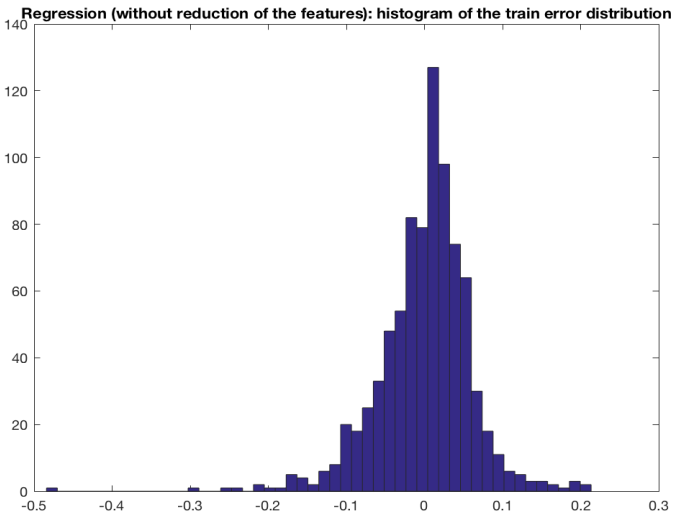


Figure 1: Regression results vs. real values in train and test phases (regression without reduction of features)



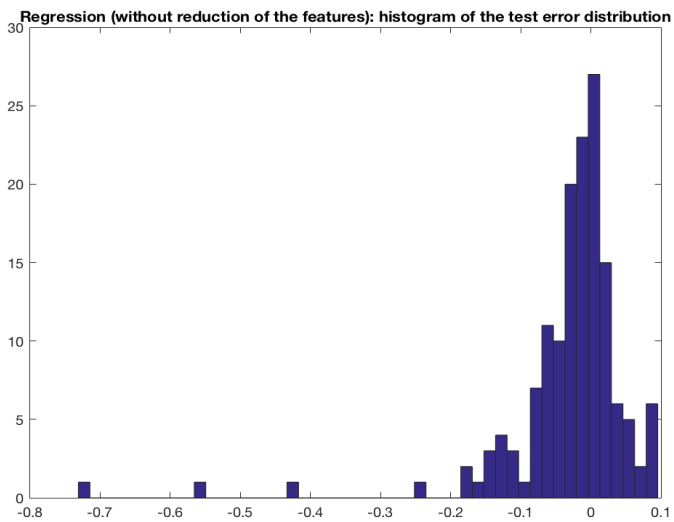
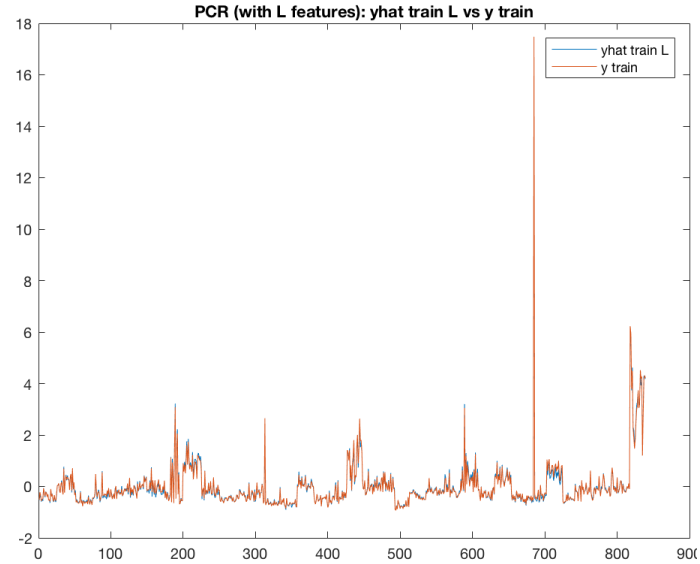


Figure 2: Comparison of the distribution of errors in train and test phase (regression without reduction of features)



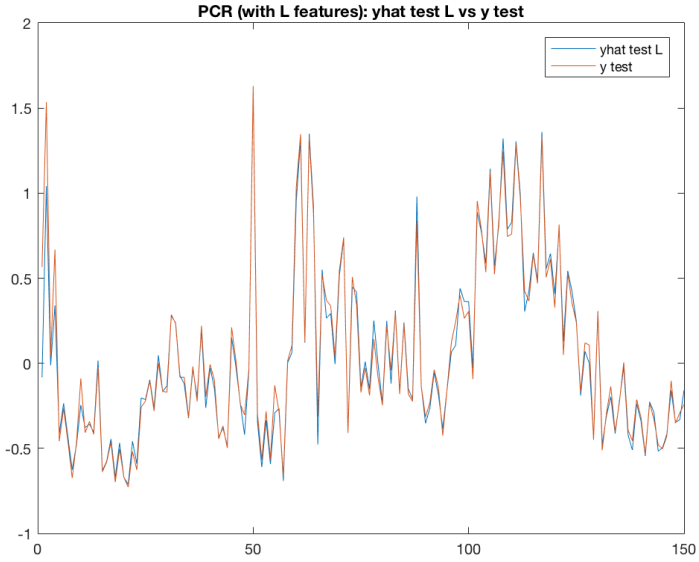
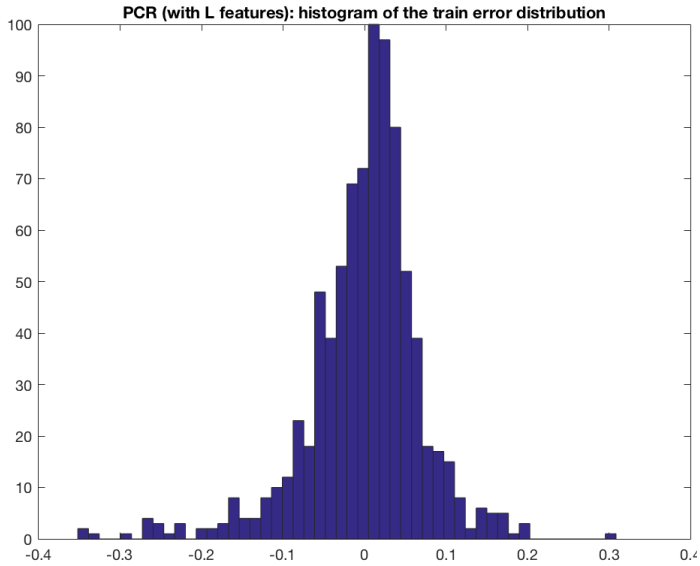


Figure 3: Regression results vs. real values in train and test phase (PCR)



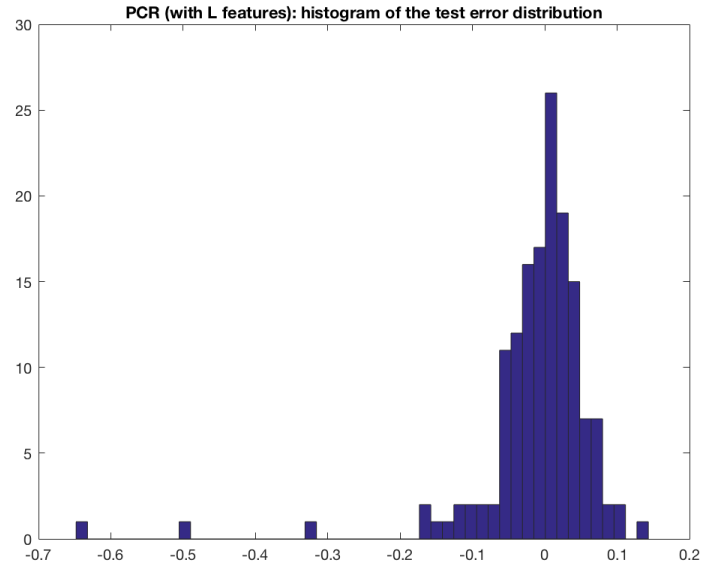


Figure 4: Comparison of the distribution of errors in train and test phase (PCR)

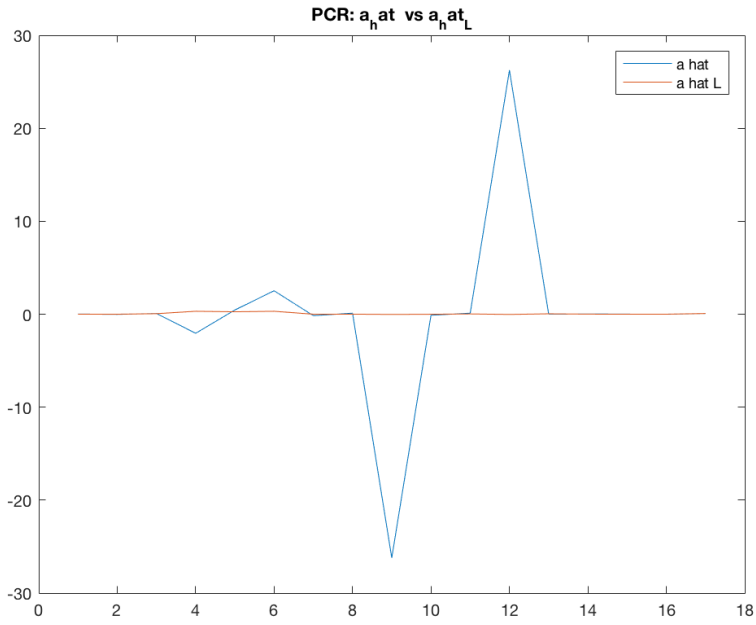


Figure 5: Comparison of the coefficients in regression without reduction of features and PCR with L features

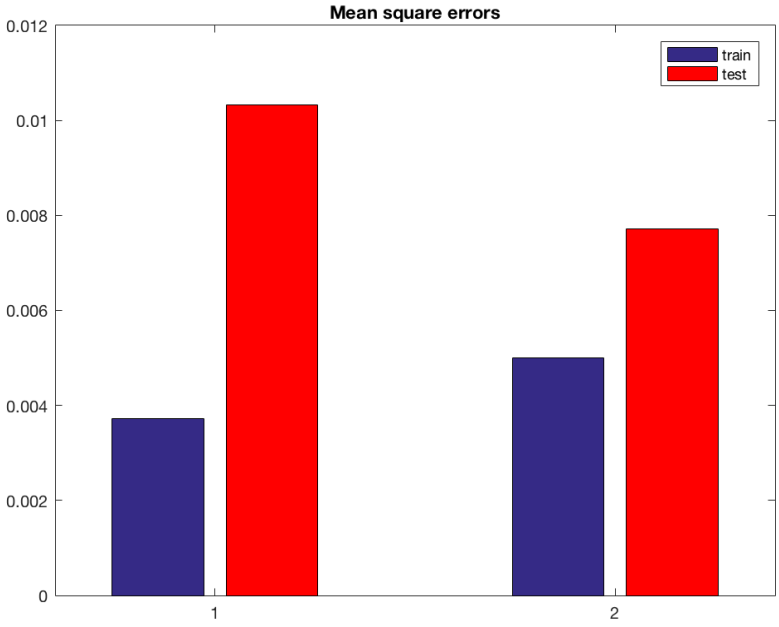


Figure 6: Comparison of the mean squared errors between regression without reduction of features and PCR with L features

Table of Contents

Data preparation

```
clear all
close all

load('updrs.mat')

A=parkinsonsupdrs;

num_patients=max(A(:,1));

new_matrix=ones(1,22); % we define the matrix in which we insert
 grouped values

for ind=1:num_patients
    sub_rows=find(A(:,1)==ind); % indexes of the matrix corresponding
 to the 'ind' patient
    pat=A(sub_rows, :); % sub_matrix of 'ind' patient
    time=sort(abs(fix(pat(:,4)))); % we define an array of the days in
 which measurements were taken for patient 'ind'
    % we had to take the integer part of that values and make the
 absolute
    % value because there were some negative time values. Then I
 ordered
    % them by increasing order
    time=unique(time); % we delete duplicates in the array of days
    for i=1:length(time) %we iterate within the days
        day=time(i);
        time_rows=find(A(:,1)==ind & abs(fix(A(:,4)))==day); % we find
 the rows with all the measurements taken on patient 'ind' in the day
 'day'
        pat_time=A(time_rows, :); % we build a matrix of the
 measurements of patient 'ind' on day 'day'
        pat_time(:,4)=abs(fix(pat_time(:,4))); % we adjust the time
 column in order to have integer day indexes
        mean_row=mean(pat_time,1);
        new_matrix=[new_matrix;mean_row];
    end
end

pazienti=new_matrix(2:end,:);


%-------  we build data_train and data_test matrices-----
```

```
% data_train=ones(1,22);
% data_test=ones(1,22);
% for i=1:length(new_matrix(:,1))
%     if new_matrix(i,1)<=36
%         data_train=[data_train;new_matrix(i,:)];
%     else
%         data_test=[data_test;new_matrix(i,:)];
%     end
% end
%
% data_train=data_train(2:end,:);
% data_test=data_test(2:end,:);

data_train = pazienti(pazienti(:,1)<37,:);
data_test = pazienti(pazienti(:,1)>36,:);


% ---- applying normalization to data_train matrix

m_data_train=mean(data_train,1);
v_data_train=var(data_train,1);
s_v_data_train=sqrt(v_data_train);

% data_train_norm=ones(length(data_train(:,1)),22);
% for i=1:length(data_train(:,1))
%     data_train_norm(i,1:4)=data_train(i,1:4);
%     for f=5:22
%         data_train_norm(i,f)=(data_train(i,f)-m_data_train(f))/
s_v_data_train(f);
%     end
% end

o = ones(size(data_train,1),1);
data_train_norm = data_train;
data_train_norm(:,5:end) = (data_train(:,5:end) -
 o*m_data_train(:,5:end)) ./ (o*s_v_data_train(:,5:end));

mean_train_norm = mean(data_train_norm,1); % verify that normalization
 is effective
var_train_norm = var(data_train_norm,1);


% ---- applying normalization to data_test matrix

% data_test_norm=ones(length(data_test(:,1)),22);
% for i=1:length(data_test(:,1))
%     data_test_norm(i,1:4)=data_test(i,1:4);
%     for f=5:22
%         data_test_norm(i,f)=(data_test(i,f)-m_data_train(f))/
s_v_data_train(f);
%     end
% end

o = ones(size(data_test,1),1);
```

```
data_test_norm = data_test;
data_test_norm(:,5:end) = (data_test(:,5:end) -
 o*m_data_train(:,5:end)) ./ (o*s_v_data_train(:,5:end));

mean_test_norm = mean(data_train_norm,1); % verify that normalization
 is effective
var_test_norm = var(data_train_norm,1);

% ---- PERFORM REGRESSION ----------

F0=7; % we have to try with feature 7 and feature 5

y_train=data_train_norm(:,F0);
X_train=data_train_norm;
X_train(:,F0)=[];

y_test=data_test_norm(:,F0);
X_test=data_test_norm;
X_test(:,F0)=[];

X_T=X_train(:,5:end).';
```

Regression without reducing number of features

```
N = size(X_train,1);
R = (1/N) * X_T * X_train(:,5:end);
[U, A] = eig(R); % [U,A] = eig(R) returns diagonal matrix A of
 eigenvalues
                % and matrix U whose columns are the corresponding
 right
                % eigenvectors, so that R*U = U*A.

%----without reducing the number of features---%

a_hat= (1/N) * U * inv(A) * U.' * X_T * y_train;
a_hat_classic= inv(X_T*X_train(:,5:end))*X_T*y_train; % it's the same
 of a_hat

y_hat_train = X_train(:,5:end) * a_hat;
y_hat_test = X_test(:,5:end) * a_hat;

error_PCR_train=y_hat_train-y_train;
error_PCR_test=y_hat_test-y_test;

MSE_train = mean(error_PCR_train.^2);
MSE_test = mean(error_PCR_test.^2);

figure
plot(y_hat_train)
hold on
plot(y_train)
```

```
hold off
title('Regression (without reduction of the features): yhat train vs y
 train')
legend('yhat train','y train')

figure
plot(y_hat_test)
hold on
plot(y_test)
hold off
title('Regression (without reduction of the features): yhat test vs y
 test')
legend('yhat test','y test')

figure
hist(error_PCR_train,50)
title('Regression (without reduction of the features): histogram of
 the train error distribution')

figure
hist(error_PCR_test,50)
title('Regression (without reduction of the features): histogram of
 the test error distribution')
```

Principal Component Regression (PCR) using L features

```
P = sum(diag(A)); % taking the sum of the eigenvalues
percentage = 0.99; % with 0.9 the MSE error in test phase was huge.
                   % With 0.99 it is lower than the PCR without
 reduction
                   % of features and allows to take only 8 features
 from
                   % the initial 17
new_P = percentage * P; % defining the amount of "information" we want
 to keep


cumulative_P = cumsum(diag(A)); % function that evaluates the
 cumulative
                                 % sum of each element of the diagonal
 of A
L = length(find(cumulative_P<new_P)); % determines the first L
 features
                                 % that contribut to obtain new_P
 amount
                                 % of "information"

U_L = U(:,1:L); % we only consider the first L features
A_L = A(1:L,1:L);
```

```
a_hat_L = 1/N * U_L * inv(A_L) * U_L.' * X_T *y_train;

y_hat_train_L = X_train(:,5:end) * a_hat_L;
y_hat_test_L = X_test(:,5:end) * a_hat_L;

error_PCR_train_L=y_hat_train_L-y_train;
error_PCR_test_L=y_hat_test_L-y_test;

MSE_train_L = mean(error_PCR_train_L.^2);
MSE_test_L = mean(error_PCR_test_L.^2);


figure
plot(y_hat_train_L)
hold on
plot(y_train)
hold off
title('PCR (with L features): yhat train L vs y train')
legend('yhat train L','y train')

figure
plot(y_hat_test_L)
hold on
plot(y_test)
hold off
title('PCR (with L features): yhat test L vs y test')
legend('yhat test L','y test')

figure
hist(error_PCR_train_L,50)
title('PCR (with L features): histogram of the train error
 distribution')

figure
hist(error_PCR_test_L,50)
title('PCR (with L features): histogram of the test error
 distribution')

figure
plot(a_hat)
hold on
plot(a_hat_L)
hold off
title('PCR: a_hat  vs a_hat_L')
legend('a hat','a hat L')


mses=[MSE_train,MSE_test;MSE_train_L,MSE_test_L]
figure
c = categorical({'MSE' 'PCR (L features)'});
b=bar(c,mses);
b(2).FaceColor='red';
title('Mean square errors')
legend('train','test')
```

Published with MATLAB® R2016a