Computational Statistics Programming Assignment

Name: Ravi Kumar Choudhary
Roll No: 13MA20033
Coding Language: MATLAB

Problem 1: PCA Regression

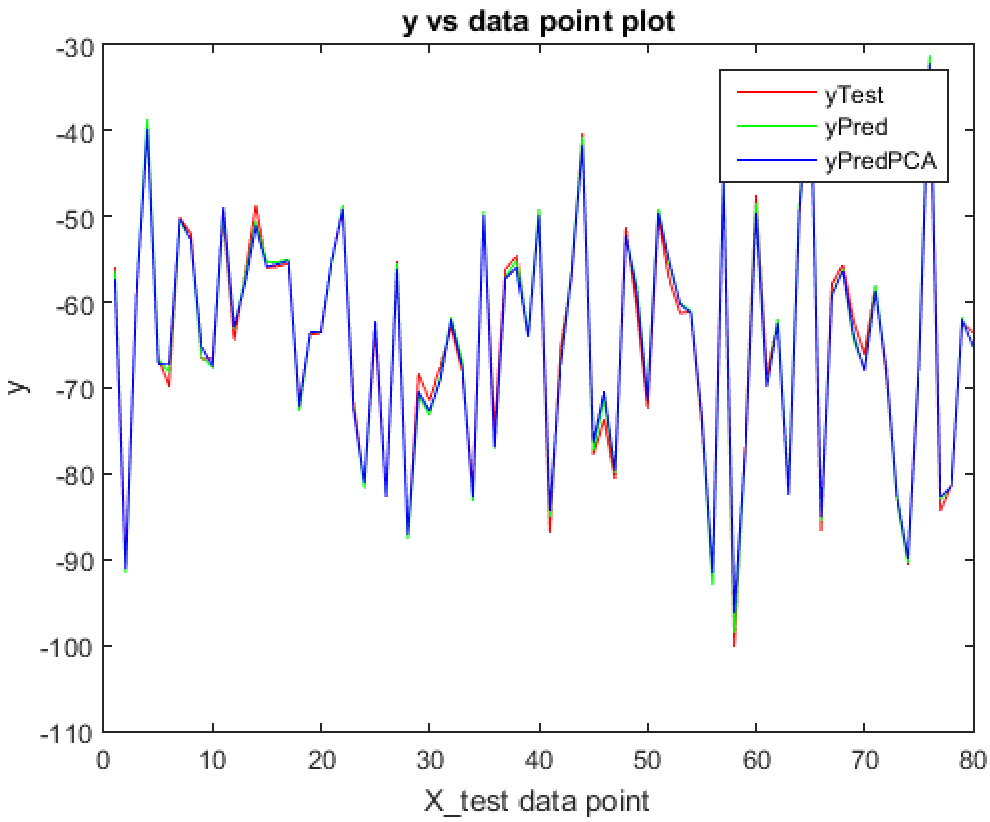
```
%% loading data
clc

x1 = csvread('x1.csv');
x2 = csvread('x2.csv');
x3 = csvread('x3.csv');
x4 = csvread('x4.csv');
y = csvread('y.csv');

X = [x1, x2, x3, x4];
n = size(X,1);
p = size(X,2);

%% new data matrix after adding column of ones
X_new = [ones(n,1), X];
p_new = size(X_new, 2);

%% split the data into traing and testing matrix by (60% and 40%)
X_train = X_new(1:120, :);
X_test = X_new(121:end, :);
y_train = y(1:120, :);
y_test = y(121:end, :);

%% finding the beta hat for training data
beta_hat = X_train\y_train;
% display(beta_hat);

%% now applying svd decomposition on X_new
[U,S,V] = svd(X_train);

%% get the diag entry of S
sigma = diag(S);
% display(sigma);

%% compute the sigma ratios
sigma_sum = sum(sigma);
% display(sigma_sum);

%% creating a vector of size p-1 to store sigma ratios
sigma_ratio = zeros(p_new-1, 1);
sigma_ratio(1,1) = sigma(1,1);

for i = 2:p_new-1
    sigma_ratio(i,1) = sigma_ratio(i-1,1) + sigma(i,1);
end
```

```
sigma_ratio = sigma_ratio/sigma_sum;
% display(sigma_ratio);

%% calculating informative part and non informative part (removing lower
sigma)
r = size(sigma,1) - 2;
U_I = U(:,1:r);
S_I = S(1:r,1:r);
V_I = V(:, 1:r);

%% calculating X informative and X non informative
X_I = U_I * S_I * V_I';
X_N = X_train - X_I;

%% calculating Wr

Wr = X_train * V_I;
% display(size(Wr));

%% calculating gamma hat for Wr
gamma_hat = Wr\y_train;

%% calculating beta hat pca using gamma hat
beta_hat_pca = V_I * gamma_hat;

%% testing prediction

y_pred = X_test * beta_hat;
y_pred_pca = X_test * beta_hat_pca;

% display(y_pred);
% display(y_pred_pca);

%% ploting prediction and actual value on testing data plot
test_data_point = 1:size(y_test,1);
test_data_point = test_data_point';

plot(test_data_point, y_test , 'r-');
hold on;
plot(test_data_point, y_pred, 'g-');
hold on
plot(test_data_point, y_pred_pca, 'b-');
hold off

title('y vs data point plot');
xlabel('X\_test data point') % x-axis label
ylabel('y') % y-axis label
legend('yTest','yPred', 'yPredPCA');

%% calculating covariance of beta hat
cov_beta_hat = cov(beta_hat);
display(cov_beta_hat);

%% calculating covariance of beta hat pca
cov_beta_hat_pca = cov(beta_hat_pca);
display(cov_beta_hat_pca);

%%  calculating rmse
```

```
rmse_test_pred = sqrt(mean((y_test-y_pred).^2));
rmse_test_pred_pca = sqrt(mean((y_test-y_pred_pca).^2));
display(rmse_test_pred);
display(rmse_test_pred_pca);

%% save the prediction in separate csv file
csvwrite('y_pred.csv', y_pred);
csvwrite('y_pred_pca.csv', y_pred_pca);

%% from the out put clearly cov(beta_hat) >= cov(beta_hat_pca)
```

Program Output:

cov_beta_hat = 89.8813
cov_beta_hat_pca = 11.6655
rmse_test_pred = 1.0761
rmse_test_pred_pca = 1.2514



Problem 2: Recursive Least Square Regression 1

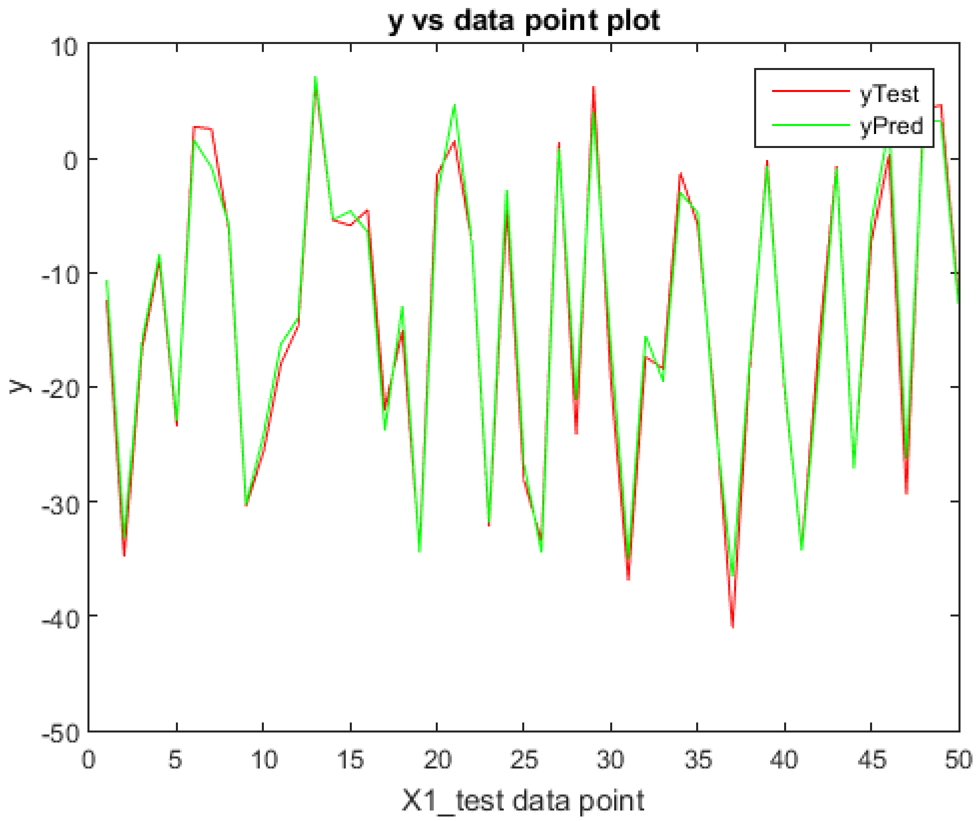
```
%% loading data
clc
X1_test = csvread('Problem1_Input_Test.csv');
X1_train = csvread('Problem1_Input_Training.csv');

Y1_train = csvread('Problem1_Output_Training.csv');
Y1_test = csvread('Problem1_Output_Test.csv');

n = size(X1_train, 1);
```

```
m = size(X1_train, 2);

% display(n)
% display(m)

%% adding ones to first column
X1_train = [ones(n,1) X1_train];
X1_test = [ones(n,1) X1_test];

%% calculating beta hat for train data
beta_hat = pinv((X1_train'*X1_train))*X1_train'*Y1_train;

%% prediction on beta hat
Y1_pred = X1_train*beta_hat;

%% calculate sum of square error
mse_train_pred = sum((Y1_train - Y1_pred).^2)/n;
rmse_train_pred = sqrt(mse_train_pred);
display(rmse_train_pred)

%% applying recursive least square model
% formula beta_hat_n_plus_1 = beta_hat_n + K_n*e_n
M_n = pinv(X1_train'*X1_train);
Y1_test_pred = zeros(n, 1);

lambda = 1;
beta_hat_n = beta_hat;

for i = 1:n
    M_n_plus_1 = M_n - (M_n * X1_test(i,:)' * X1_test(i,:) * M_n)
./((1/lambda) + X1_test(i,:) * M_n * X1_test(i,:)');

    K_n = (1/lambda) .* (M_n_plus_1 * X1_test(i,:)');
    e_n = (Y1_test(i) - X1_test(i,:) * beta_hat_n);

    beta_hat_n_plus_1 = beta_hat_n + K_n .* e_n;

    Y1_test_pred(i) = X1_test(i,:) * beta_hat_n_plus_1;
    % update beta hat n+1
    M_n = M_n_plus_1;
    beta_hat_n = beta_hat_n_plus_1;
end

%% calculate rmse for test data
mse_test_pred = sum((Y1_test - Y1_test_pred).^2)/n;
rmse_test_pred = sqrt(mse_test_pred);
display(rmse_test_pred)

%% ploting prediction and actual value on testing data plot
test_data_point = 1:size(Y1_test,1);
test_data_point = test_data_point';

plot(test_data_point, Y1_test , 'r-');
hold on;
plot(test_data_point, Y1_test_pred, 'g-');
hold off
title('y vs data point plot');
xlabel('X1\_test data point') % x-axis label
```

```
ylabel('y') % y-axis label
legend('yTest','yPred');
```

Program Output:

rmse_train_pred = 1.5712
rmse_test_pred = 1.6827



Problem 3: Recursive Least Square Regression 2

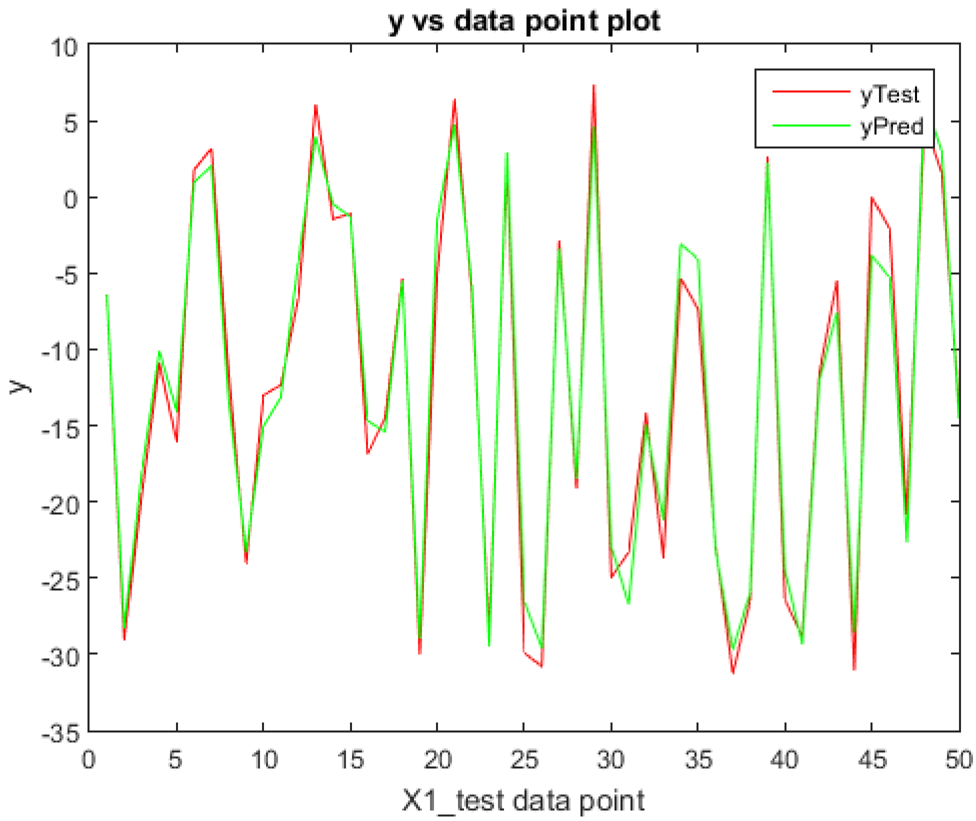
```
%% loading data
clc
X1_test = csvread('Problem1_Input_Test.csv');
X1_train = csvread('Problem1_Input_Training.csv');

Y1_train = csvread('Problem2_Output_Training.csv');
Y1_test = csvread('Problem2_Output_Test.csv');

n = size(X1_train, 1);
m = size(X1_train, 2);

% display(n)
% display(m)

%% adding ones to first column
X1_train = [ones(n,1) X1_train];
X1_test = [ones(n,1) X1_test];

%% calculating beta hat for train data
```

```
beta_hat = pinv((X1_train'*X1_train))*X1_train'*Y1_train;

%% prediction on beta hat
Y1_pred = X1_train*beta_hat;

%% calculate sum of square error
mse_train_pred = sum((Y1_train - Y1_pred).^2)/n;
rmse_train_pred = sqrt(mse_train_pred);
display(rmse_train_pred)

%% applying recursive least square model
% formula beta_hat_n_plus_1 = beta_hat_n + K_n*e_n
M_n = pinv(X1_train'*X1_train);
Y1_test_pred = zeros(n, 1);

lambda = 1;
beta_hat_n = beta_hat;

for i = 1:n
    M_n_plus_1 = M_n - (M_n * X1_test(i,:)' * X1_test(i,:) * M_n)
./((1/lambda) + X1_test(i,:) * M_n * X1_test(i,:)');

    K_n = (1/lambda) .* (M_n_plus_1 * X1_test(i,:)');
    e_n = (Y1_test(i) - X1_test(i,:) * beta_hat_n);

    beta_hat_n_plus_1 = beta_hat_n + K_n .* e_n;

    Y1_test_pred(i) = X1_test(i,:) * beta_hat_n_plus_1;
    % update beta hat n+1
    M_n = M_n_plus_1;
    beta_hat_n = beta_hat_n_plus_1;
end

%% calculate rmse for test data
mse_test_pred = sum((Y1_test - Y1_test_pred).^2)/n;
rmse_test_pred = sqrt(mse_test_pred);
display(rmse_test_pred)

%% ploting prediction and actual value on testing data plot
test_data_point = 1:size(Y1_test,1);
test_data_point = test_data_point';

plot(test_data_point, Y1_test , 'r-');
hold on;
plot(test_data_point, Y1_test_pred, 'g-');
hold off
title('y vs data point plot');
xlabel('X1\_test data point') % x-axis label
ylabel('y') % y-axis label
legend('yTest','yPred');
```

Program Output:

rmse_train_pred = 1.5605

rmse_test_pred = 1.8010



<u>Program 4: Locally Weighted Regression</u>

```
%% loading data
clc
X1_test = csvread('Problem1_Input_Test.csv');
X1_train = csvread('Problem1_Input_Training.csv');

Y3_train = csvread('Problem3_Output_Training.csv');
Y3_test = csvread('Problem3_Output_Test.csv');

n = size(X1_train, 1);
m = size(X1_train, 2);

% display(n)
% display(m)

%% adding ones to first column
X1_train = [ones(n,1) X1_train];
X1_test = [ones(n,1) X1_test];

%% initialize y_test_pred
Y3_test_pred = ones(n, 3);

%% calculating weight for each test data
d = zeros(n, 1);
w = zeros(n, 3);

%% i for iterating through test data and j for train data
phi = 1;
epsilon = 0.0001;
```

```
for i = 1:n
   for j = 1:n
      % d(j) =  dot(X1_train(j, :), X1_test(i, :))/(norm(X1_train(j,
:))*norm(X1_test(i, :)));
      d(j) = norm(X1_train(j,:) - X1_test(i,:));
      % w(j) = exp((-d(j)^2));
   end
   % normalizing d
   d = (d - mean(d))/std(d);
   d = abs(d);
   w(:,1) = d <= 1;
   w(:,2) = exp((-d.^2)/(2*phi));
   w(:,3) = exp(-d.^2);
   for k = 1:n
      if d(k) > epsilon
          w(k,3) = 1/d(k);
      else
          w(k,3) = 10;
      end
   end

   % for first test data we got d and w, now compute beta_hat
   % display(d);

   for l = 1:3
       V = diag(w(:, l));
       Inv_V = V;
       beta_hat = pinv(X1_train' * Inv_V * X1_train) * X1_train' * Inv_V *
Y3_train;
       % display(beta_hat);
       Y3_test_pred(i, l) = X1_test(i, :) * beta_hat;
   end

end
display(Y3_test_pred);

%% calculate rmse for test data
rmse_test_pred = zeros(3,1);

for m = 1:3
    mse_test_pred = mean((Y3_test - Y3_test_pred(:,m)).^2);
    rmse_test_pred(m) = sqrt(mse_test_pred);
end
display(rmse_test_pred);

%% ploting prediction and actual value on testing data plot
test_data_point = 1:size(Y3_test,1);
test_data_point = test_data_point';

x0=10;
y0=10;
width=550;
height=400;
set(gcf,'units','points','position',[x0,y0,width,height])

for p = 1:3
    subplot(3,1,p);
    plot(test_data_point, Y3_test , 'r-');
```

```
    hold on;
    plot(test_data_point, Y3_test_pred(:,p), 'g-');
    hold off
    if p == 1
        title('y vs data point plot for weight 1');
    elseif p == 2
        title('y vs data point plot for weight 2');
    else
        title('y vs data point plot for weight 3');
    end
    xlabel('X1\_test data point') % x-axis label
    ylabel('y') % y-axis label
    legend('yTest','yPred');
end
```

Program Output:

rmse_test_pred =

0.7919
0.7661
0.7983