Vincent Purcell

Professor Cohen

ECE 487 – Pattern Recognition

8 October 2019

Homework 2

Problems: 3.1 p 142, 3.1 & 3.2 p 146

3.1 Explain why the perceptron cost function is a continuous piecewise linear function.

First one must look at the perceptron algorithm at its base to understand how it is continuous and piecewise linear function. The perceptron algorithm takes a data set of two classes and then attempts to define a plane that attempts to create a dividing threshold to classify testing points. That plane can be defined as $w^{*T}x=0$, such that

$$\begin{aligned} w^{*T}x &> 0 \quad \forall x \in \omega_1 \\ w^{*T}x &< 0 \quad \forall x \in \omega_2 \end{aligned}$$

A cost function is chosen in order to approach a problem as an optimization task, that perception cost function can be defined as

$$J(w) = \sum_{x \in Y} (\delta_x w^T x)$$

That function *J(w)* is a continuous piecewise linear function. The weight parameters, *w*, change in a continuous fashion through each iteration of the algorithm. It is a smooth descent and one could describe the descent as piecewise. The plane makes piecewise adjustments towards an optimal division line once there is a change in the number of misclassified vectors.

Page 93 of Pattern Recognition Text

References

- Pattern Recognition - AP (2008) - Theodoridis S., Koutroumbas K.

Contents

Vincent Purcell - HW 1 - ECE487

```
clear; clc; close all;
```

Problem 3.1

```
rng('default'); rng(0);
% Initializing the variables used for functions
m = [[-5;0],[5;0]];
S(:,:,1) = [1,0;0,1];
S(:,:,2) = [1,0;0,1];
S(:,:,3) = [1,0;0,1];
P = [0.5,0.5];
N = 200;

% Generated Gaussian classes
[X1,~] = generate_gauss_classes(m,S,P,N);
[X1_p,~] = generate_gauss_classes(m,S,P,N);
y = [-ones(1,100),ones(1,100)];
```

Evalute X1 with Various Algorithms

```
X1 = [X1 , ones(1,200)'];
w_perce_1 = perce(X1',y,rand(3,1));
w_sserr_1 = SSErr(X1',y);
w_lms_1 = LMSalg(X1(:,1:2)',y,rand(2,1));

% Classify using SVM algorithms and compute error
y_est_perce_1 = svm_classifier(X1',w_perce_1);
perce_err_1 = compute_error(y,y_est_perce_1);

y_est_sserr_1 = svm_classifier(X1',w_sserr_1);
sserr_err_1 = compute_error(y,y_est_sserr_1);

y_est_lms_1 = svm_classifier_LMS(X1',w_lms_1);
lms_err_1 = compute_error(y,y_est_lms_1);
```

Evalute X1 Prime with Various Algorithms

```
X1_p = [X1_p , ones(1,200)'];
w_perce_1_p = perce(X1_p',y,rand(3,1));
w_sserr_1_p = SSErr(X1_p',y);
w_lms_1_p = LMSalg(X1_p(:,1:2)',y,rand(2,1));

% Classify using SVM algorithms and compute error
y_est_perce_1_p = svm_classifier(X1_p',w_perce_1_p);
perce_err_1_p = compute_error(y,y_est_perce_1_p);

y_est_sserr_1_p = svm_classifier(X1_p',w_sserr_1_p);
sserr_err_1_p = compute_error(y,y_est_sserr_1_p);

y_est_lms_1_p = svm_classifier_LMS(X1_p',w_lms_1_p);
lms_err_1_p = compute_error(y,y_est_lms_1_p);
```

Problem 3.2

```
rng('default'); rng(0);
% Initializing the variables used for functions
m = [[-2;0],[2;0]];
S(:,:,1) = [1,0;0,1];
S(:,:,2) = [1,0;0,1];
S(:,:,3) = [1,0;0,1];
P = [0.5,0.5];
N = 200;

% Generated Gaussian classes
[X2,~] = generate_gauss_classes(m,S,P,N);
[X2_p,~] = generate_gauss_classes(m,S,P,N);
y = [-ones(1,100),ones(1,100)];
```

Evalute X2 with Various Algorithms

```
X2 = [X2 , ones(1,200)'];
w_perce_2 = perce(X2',y,rand(3,1));
w_sserr_2 = SSErr(X2',y);
w_lms_2 = LMSalg(X2(:,1:2)',y,rand(2,1));

% Classify using SVM algorithms and compute error
y_est_perce_2 = svm_classifier(X2',w_perce_2);
perce_err_2 = compute_error(y,y_est_perce_2);

y_est_sserr_2 = svm_classifier(X2',w_sserr_2);
sserr_err_2 = compute_error(y,y_est_sserr_2);

y_est_lms_2 = svm_classifier_LMS(X2',w_lms_2);
lms_err_2 = compute_error(y,y_est_lms_2);
```

Evalute X2 Prime with Various Algorithms

```
X2_p = [X2_p , ones(1,200)'];
w_perce_2_p = perce(X2_p',y,rand(3,1));
w_sserr_2_p = SSErr(X2_p',y);
w_lms_2_p = LMSalg(X2_p(:,1:2)',y,rand(2,1));

% Classify using SVM algorithms and compute error
y_est_perce_2_p = svm_classifier(X2_p',w_perce_2_p);
perce_err_2_p = compute_error(y,y_est_perce_2_p);

y_est_sserr_2_p = svm_classifier(X2_p',w_sserr_2_p);
sserr_err_2_p = compute_error(y,y_est_sserr_2_p);

y_est_lms_2_p = svm_classifier_LMS(X2_p',w_lms_2_p);
lms_err_2_p = compute_error(y,y_est_lms_2_p);
```

Plot Figures

Plot X1

```
figure;
hold on
% Plot X1 x,y values
scatter(X1(1:100,1),X1(1:100,2),10,'o','r')
scatter(X1(101:200,1),X1(101:200,2),10,'x','b')
% Plot lines equated from w vector parameters of various algorithms
line(X1(:,1),plot_w_equation(X1',w_perce_1),'Color','black')
line(X1(:,1),plot_w_equation(X1',w_sserr_1),'Color','green')
line(X1(:,1),plot_w_equation_LMS(X1',w_lms_1),'Color','magenta')
legend('Class 1','Class 2','Perceptron','SSerr','LMS')
title('Problem 3.1 - X1')
xlim([-8 8])
ylim([-4 4])
a = gca; % get the current axis;
% set the width of the axis (the third value in Position)
% to be 60% of the Figure's width
a.Position(3) = 0.6;
text1 = {"Computed Error","Perce: " + num2str(perce_err_1) ...
```

```
    ,"SSerr: " + num2str(sserr_err_1),"LMS: " + num2str(lms_err_1)};
annotation('textbox',[0.75 0 .4 .5],'String',text1,'FitBoxToText','on')
hold off
```

Plot X1 Prime

```
figure;
hold on
% Plot X1 Prime x,y values
scatter(X1_p(1:100,1),X1_p(1:100,2),10,'o','r')
scatter(X1_p(101:200,1),X1_p(101:200,2),10,'x','b')
% Plot lines equated from w vector parameters of various algorithms
line(X1_p(:,1),plot_w_equation(X1_p',w_perce_1_p),'Color','black')
line(X1_p(:,1),plot_w_equation(X1_p',w_sserr_1_p),'Color','green')
line(X1_p(:,1),plot_w_equation_LMS(X1_p',w_lms_1_p),'Color','magenta')
legend('Class 1','Class 2','Perceptron','SSerr','LMS')
title('Problem 3.2 - X1 Prime')
xlim([-8 8])
ylim([-4 4])
a = gca; % get the current axis;
% set the width of the axis (the third value in Position)
% to be 60% of the Figure's width
a.Position(3) = 0.6;
text1 = {"Computed Error","Perce: " + num2str(perce_err_1_p) ...
    ,"SSerr: " + num2str(sserr_err_1_p),"LMS: " + num2str(lms_err_1_p)};
annotation('textbox',[0.75 0 .4 .5],'String',text1,'FitBoxToText','on')
hold off
```

Plot X2

```
figure;
hold on
% Plot X2 x,y values
scatter(X2(1:100,1),X2(1:100,2),10,'o','r')
scatter(X2(101:200,1),X2(101:200,2),10,'x','b')
% Plot lines equated from w vector parameters of various algorithms
line(X2(:,1),plot_w_equation(X2',w_perce_2),'Color','black')
line(X2(:,1),plot_w_equation(X2',w_sserr_2),'Color','green')
line(X2(:,1),plot_w_equation_LMS(X2',w_lms_2),'Color','magenta')
legend('Class 1','Class 2','Perceptron','SSerr','LMS')
title('Problem 3.2 - X2')
xlim([-5 5])
ylim([-3 3])
a = gca; % get the current axis;
% set the width of the axis (the third value in Position)
% to be 60% of the Figure's width
a.Position(3) = 0.6;
text1 = {"Computed Error","Perce: " + num2str(perce_err_2) ...
    ,"SSerr: " + num2str(sserr_err_2),"LMS: " + num2str(lms_err_2)};
annotation('textbox',[0.75 0 .4 .5],'String',text1,'FitBoxToText','on')
hold off
```

Plot X2 Prime

```
figure;
hold on
% Plot X2 Prime x,y values
scatter(X2_p(1:100,1),X2_p(1:100,2),10,'o','r')
scatter(X2_p(101:200,1),X2_p(101:200,2),10,'x','b')
% Plot lines equated from w vector parameters of various algorithms
line(X2_p(:,1),plot_w_equation(X2_p',w_perce_2_p),'Color','black')
line(X2_p(:,1),plot_w_equation(X2_p',w_sserr_2_p),'Color','green')
line(X2_p(:,1),plot_w_equation_LMS(X2_p',w_lms_2_p),'Color','magenta')
legend('Class 1','Class 2','Perceptron','SSerr','LMS')
title('Problem 3.2 - X2 Prime')
xlim([-5 5])
ylim([-3 3])
a = gca; % get the current axis;
% set the width of the axis (the third value in Position)
% to be 60% of the Figure's width
a.Position(3) = 0.6;
text1 = {"Computed Error","Perce: " + num2str(perce_err_2_p) ...
    ,"SSerr: " + num2str(sserr_err_2_p),"LMS: " + num2str(lms_err_2_p)};
annotation('textbox',[0.75 0 .4 .5],'String',text1,'FitBoxToText','on')
hold off
```

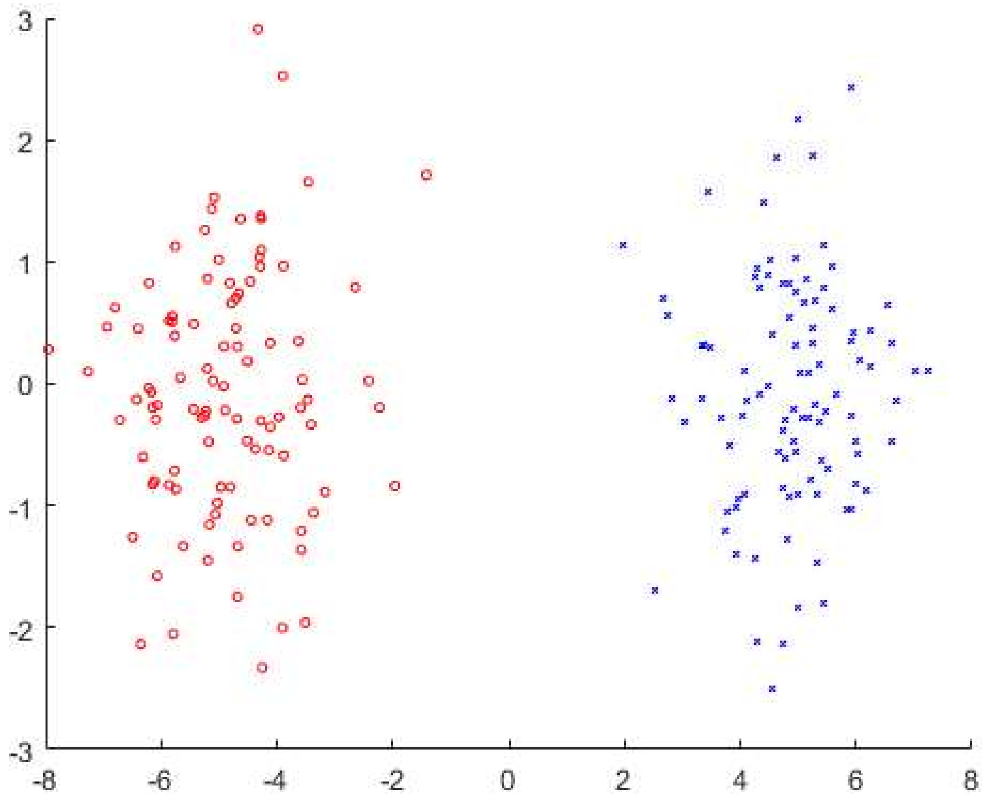
Classification Functions

Function that will output estimated classes (either -1 or 1) based on the input w parameter

```
function y_est = svm_classifier(X,w)
    test_var = -(w(1)/w(2)).*X(1,:)-(w(3)/w(2))-X(2,:);
    y_est = zeros(1,length(X));
    for i=1:length(y_est)
        if (-w(1)/w(2)) < 0
            if test_var(i) > 0
                y_est(i) = -1;
            else
                y_est(i) = 1;
            end
        else
            if test_var(i) < 0
                y_est(i) = -1;
            else
                y_est(i) = 1;
            end
        end
    end
end

function y_est = svm_classifier_LMS(X,w)
    test_var = -(w(1)/w(2)).*X(1,:)-X(2,:);
    y_est = zeros(1,length(X));
    for i=1:length(y_est)
        if (-w(1)/w(2)) < 0
            if test_var(i) > 0
                y_est(i) = -1;
            else
                y_est(i) = 1;
```

```
            end
        else
            if test_var(i) < 0
                y_est(i) = -1;
            else
                y_est(i) = 1;
            end
        end
    end
end
```



Generate Line from Parameter Function

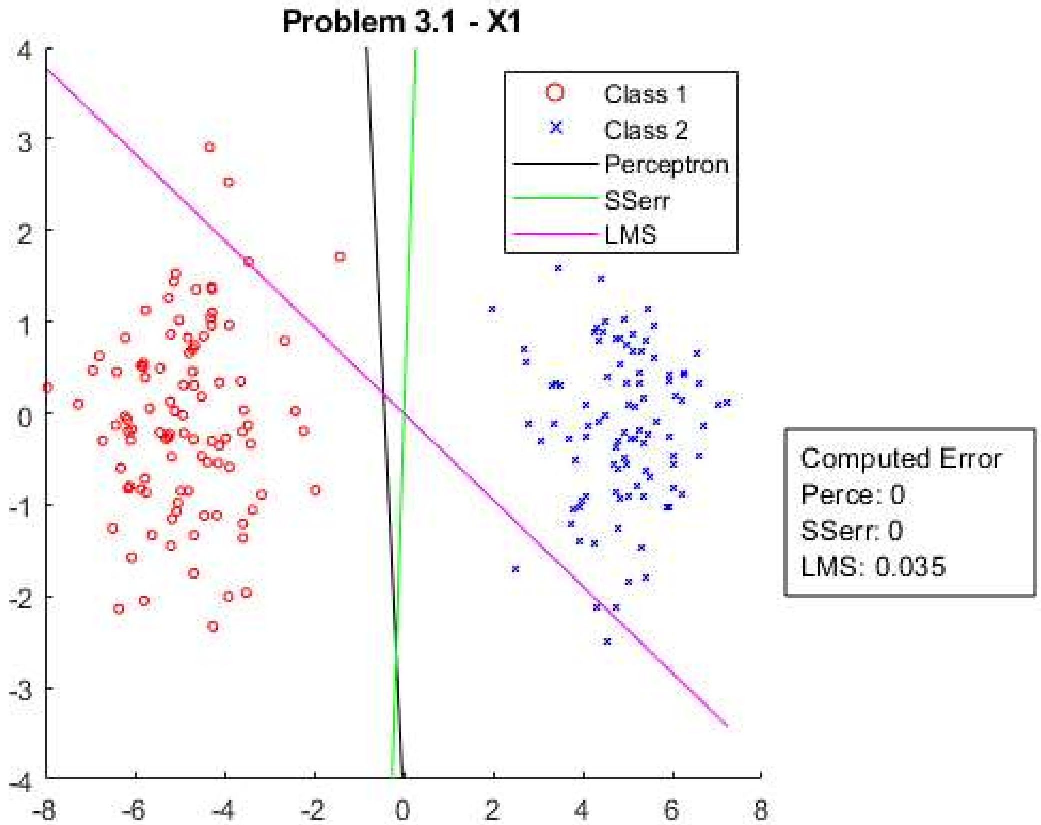
Take an input w parameter vector and X values and calculate the line plot from the resulting equation

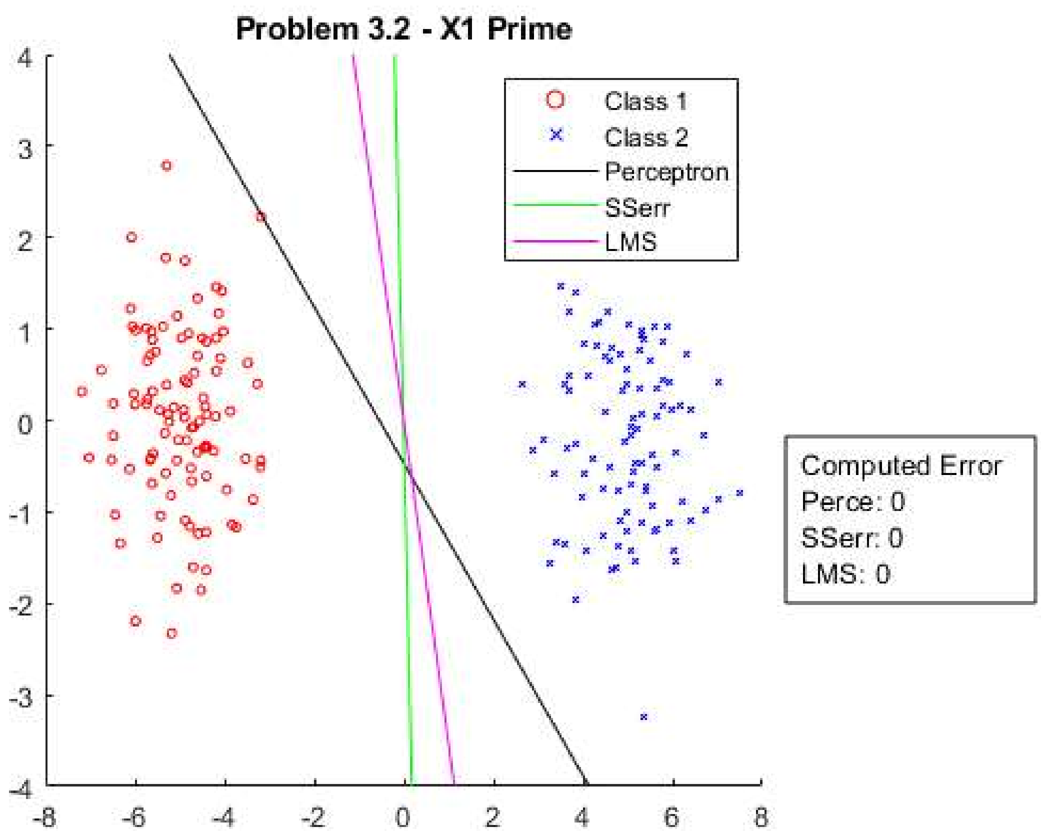
Function for SSerr and Perceptron functions

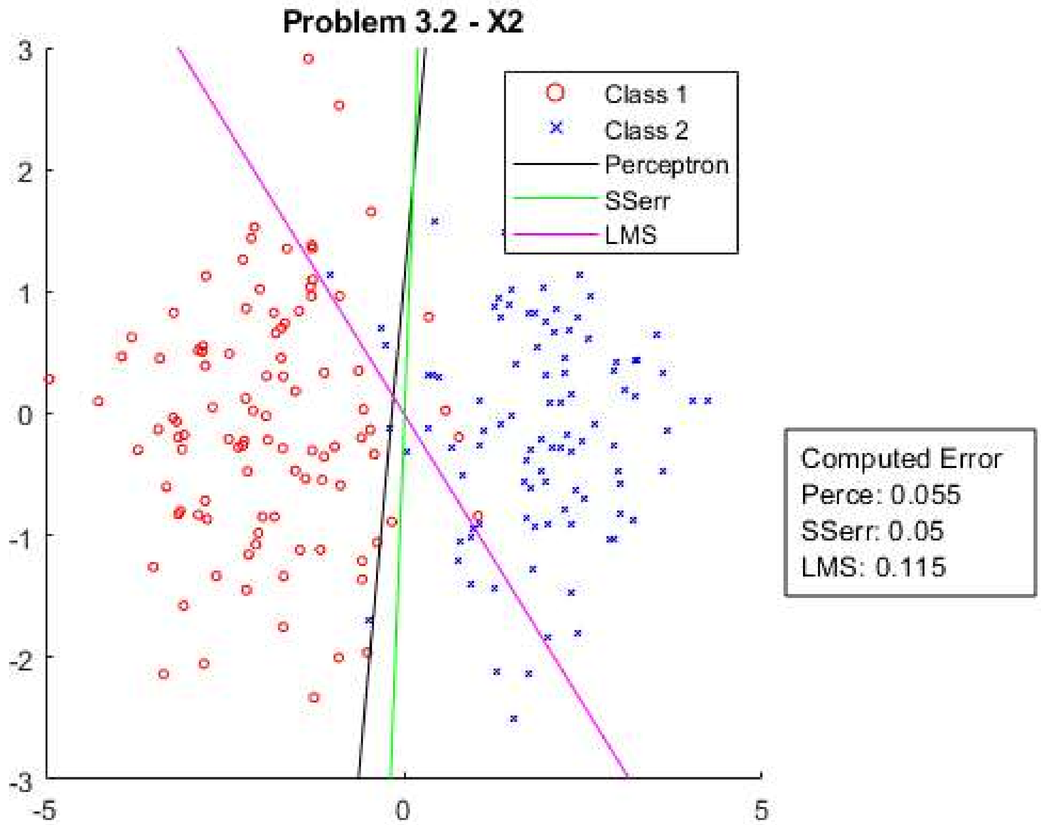
```
function y = plot_w_equation(X,w)
    y = -(w(1)/w(2)).*X(1,:)-(w(3)/w(2));
end
```

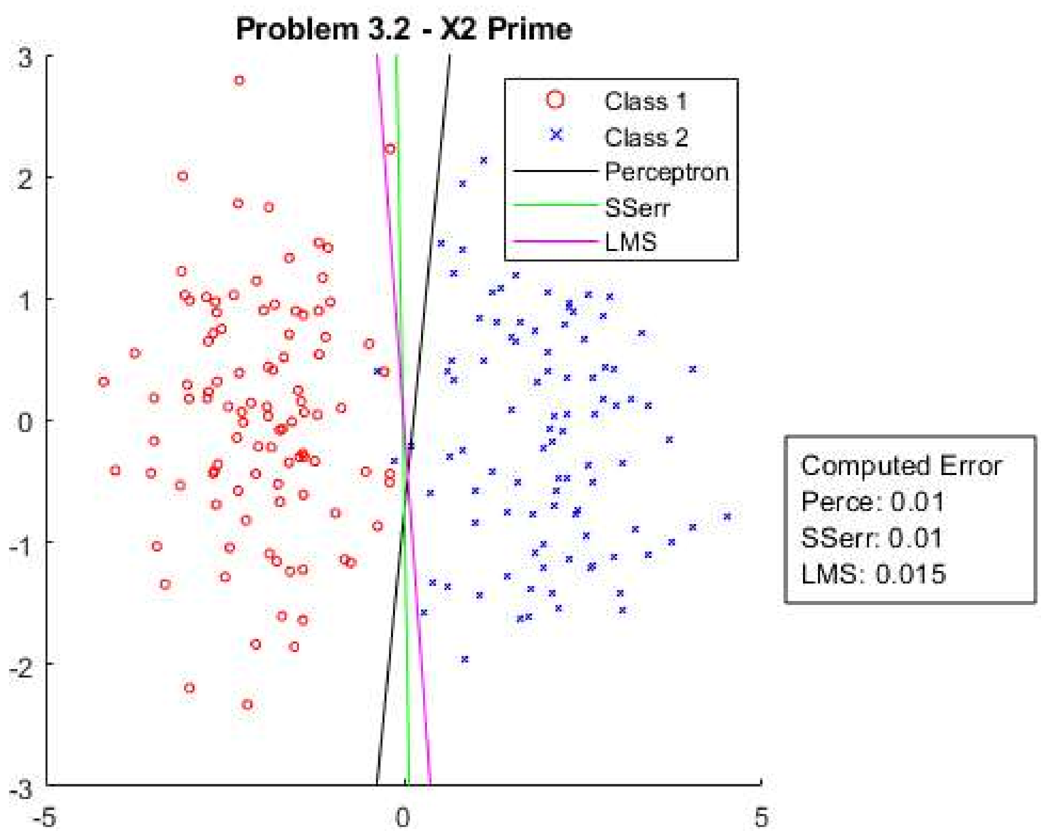
Function for LMS Algorithm

```
function y = plot_w_equation_LMS(X,w)
    y = -(w(1)/w(2)).*X(1,:);
end
```


Problem 3.1 - X1
Class 1
Class 2
Perceptron
SSerr
LMS
Computed Error
Perce: 0
SSerr: 0
LMS: 0.035


Problem 3.2 - X1 Prime
Class 1
Class 2
Perceptron
SSerr
LMS
Computed Error
Perce: 0
SSerr: 0
LMS: 0





Functions Received From Textbook

```
The following functions were received from the Textbook
Pattern Recognition - Theodoridis, Koutroumbas
```

Generate Gaussian Classes Function

Received from page 80 of the Text

```
function [X,y]=generate_gauss_classes(m,S,P,N)
    [l,c]=size(m);
    X=[];
    y=[];
    for j=1:c
    % Generating the [p(j)*N)] vectors from each distribution
        t=mvnrnd(m(:,j),S(:,:,j),fix(P(j)*N));
        % The total number of points may be slightly less than N
        % due to the fix operator
        X=[X;t];
        y=[y ones(1,fix(P(j)*N))*j];
    end
end
```

Compute Classification Error Function

Received from page 83 of the Text

```
function clas_error=compute_error(y,y_est)
    [q,N]=size(y); % N= no. of vectors
    c=max(y); % Determining the number of classes
    clas_error=0; % Counting the misclassified vectors
    for i=1:N
        if(y(i)~=y_est(i))
            clas_error=clas_error+1;
        end
    end
    % Computing the classification error
    clas_error=clas_error/N;
end
```

Perceptron Algorithm Function

Page 145 from the text

```
function w=perce(X,y,w_ini)
    [l,N]=size(X);
    max_iter=10000; % Maximum allowable number of iterations
    rho=0.05; % Learning rate
    w=w_ini; % Initialization of the parameter vector
    iter=0; % Iteration counter
    mis_clas=N; % Number of misclassified vectors
    while (mis_clas>0) && (iter<max_iter)
        iter=iter+1;
        mis_clas=0;
        gradi=zeros(l,1);% Computation of the "gradient"
```

```
        % term
        for i=1:N
            if((X(:,i)'*w)*y(i)<0)
                mis_clas=mis_clas+1;
                gradi=gradi+rho*(-y(i)*X(:,i));
            end
        end
        w=w-rho*gradi; % Updating the parameter vector
    end
end
```

Sum of Squares error

Page 145 from the text

```
function w=SSErr(X,y)
    w=inv(X*X')*(X*y');
end
```

LMS Algorithm function

page 146 from the text

```
function w=LMSalg(X,y,w_ini)
    [l,N]=size(X);
    rho=0.1; % Learning rate initialization
    w=w_ini; % Initialization of the parameter vector
    for i=1:N
        w=w+(rho/i)*(y(i)-X(:,i)'*w)*X(:,i);
    end
end
```