TMA265/MMA600
Numerical Linear Algebra
Computer Exercise 3

Maitreya Dave [*] Alan Ali Doosti [†]

November 6, 2020

Abstract

Implementation of least square method and perceptron algorithm were performed on a data set with the MATLAB software in order to study the impact of altered parameters of the aforementioned algorithms.

[*]Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, SE-42196 Gothenburg, Sweden, e-mail: maitreya@student.chalmers.se

[†]Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, SE-42196 Gothenburg, Sweden, e-mail: payama@student.chalmers.se

Contents

List of Figures

1 Introduction

The aim of the report is to evaluate the performance of some numerical methods regarding classification problem. In this problem, the methods of least squares and perceptron learning were implemented with regularization.

2 Theory, methods and algorithms

A data set with two coordinates (which will be denoted x and y respectively) and a class set constituting the input values will be subject to classification where decision lines will be generated to distinguish each class. A brief explanation of classification using least squares, the perceptron learning algorithms and balancing principle will be provided.

2.1 Methodology

1. The first step to do is to write a compute function which can generate a decision line using least squares method and the perceptron learning algorithm (polynomial order 1 and 2).

2. Then an option is added of passing regularization parameter γ to the compute function. An initial approximate value $\gamma_0 = 0.5$ is directly passed and will be updated using the balancing principle and therefore an implement is performed on that.

3. To check for miss-classification, the iris dataset is split into a ratio of 70% for training and 30% for testing purposes.

4. To understand the effect of learning rate, a counter is added to find how many iterations it takes for the algorithm to reach the solution or max iteration limit which is set to 10^6.

5. Since the data selected from the iris dataset is linearly separable, computations is also performed on the seals dataset as demonstrated in the paper [1].

2.2 Least Squares for Classification

Following is the Non-regularized least-squares problem

$$\min_{w} \frac{1}{2}||y(\omega) - t||_2^2 = \min_{w} \frac{1}{2}||t - f(x,\omega)||_2^2 = \min_{w} \frac{1}{2}||t - \omega^T \phi(x)||_2^2 \quad (1)$$

where ω are the weights, t is the target function value and $y(\omega)$ is the input data. Here the function $f(x, \omega) = \omega_0 + \omega^T \phi(x)$, where $\phi(x)$ is known as the basis functions with $\phi_0(x) = 1$.

The design matrix of such a problem is as follows:

$$A = \begin{bmatrix} 1 & \phi_1(x_1) & \phi_2(x_2) & \dots & \phi_M(x_1) \\ 1 & \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_M(x_2) \\ 1 & \phi_1(x_3) & \phi_2(x_3) & \dots & \phi_M(x_3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(x_N) & \phi_2(x_N) & \dots & \phi_M(x_M) \end{bmatrix} \quad (2)$$

and so forth, the LS problem is finally written as

$$\min_{w} \frac{1}{2}||A\omega - t||_2^2 \quad A \,\epsilon\, R^{N*M} \text{ where } N > M, t \,\epsilon R^N,\, \omega \,\epsilon\, R^M \quad (3)$$

Now when taking regularization into consideration the following problem had to be solved:

$$\min_{w} \frac{1}{2}||A\omega - t||_2^2 + \frac{\gamma}{2}||\omega||_2^2 \quad A \,\epsilon\, R^{N*M} \text{ where } N > M, t \,\epsilon R^N,\, \omega \,\epsilon\, R^M \quad (4)$$

The difference in equation:(3) and equation:(4) is of the regularization parameter γ. Obtaining the optimum value of this parameter γ will be mentioned in further subsection. To solve this problem the approach was to only find the gradient of the equation:(4). After computing the gradient one arrives to the following solution:

$$(A^T A + \gamma I)\omega = A^T t \Longrightarrow \omega = (A^T A + \gamma I)^{-1}(A^T t) \quad (5)$$

The above solution for ω is valid since it forms a system of M linear equations with M unknowns.

2.3 Perceptron Learning

Perceptron learning is a binary classifier which follows a continous update strategy for its weight's such that an optimum decision line (two-dimensional) is formed. A perceptron is computed in the following way:

$$y(x,\omega) = sign(\omega^T x) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} \omega_0 + \omega_i x_i > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The decision boundary for prediction or estimation is then computed as follows:

$$\omega^T x = 0 \tag{7}$$

Expanding this learning algorithm for polynomial of second order is also possible. The second order polynomial function would thus look something as follows:

$$\omega_0 + \omega_1(x_1) + \omega_2(x_2) + \omega_3(x_1)^2 + \omega_4(x_1)(x_2) + \omega_5(x_2)^2 = 0 \tag{8}$$

Using simple substitution technique the above 2^{nd} order polynomial can be converted to a linear equation in the following way:

$$z_1 = x_1 \tag{9a}$$
$$z_2 = x_2 \tag{9b}$$
$$z_3 = (x_1)^2 \tag{9c}$$
$$z_4 = (x_1)(x_2) \tag{9d}$$
$$z_5 = (x_2)^2 \tag{9e}$$

The same steps will be followed as for the perceptron learning algorithm of first order. But now to compute the decision lines, one needs to recover the value of x_2 which is embedded in a quadratic equation (as we know $\omega_{i=0,1,\ldots,5}$ and x_1). To compute x_2, the following can be performed:

$$
\begin{align}
0 &= \omega_5(x_2)^2 + (\omega_2 + \omega_4 x_1)(x_2) + (\omega_0 + \omega_1 x_1 + \omega_3 x_1^2) \tag{10a}\\
0 &= a(x_2)^2 + b(x_2) + c \tag{10b}\\
a &= \omega_5 \tag{10c}\\
b &= \omega_2 + \omega_4 x_1 \tag{10d}\\
c &= \omega_0 + \omega_1 x_1 + \omega_3 x_1^2 \tag{10e}\\
x_2 &= -\frac{b \pm \sqrt{b^2 - 4ac}}{2 * a} \tag{10f}
\end{align}
$$

To implement Regularization for perceptron learning algorithm, the regularization parameter γ will be introduced as follows:

$$
\begin{gathered}
F(\omega) = \min_{w} \frac{1}{2}||t - y(x,\omega)||_2^2 + \frac{\gamma}{2}||\omega||_2^2 \quad A \,\epsilon\, R^{N*M} \\
\text{where } N > M, t \,\epsilon R^N,\, \omega \,\epsilon\, R^M
\end{gathered} \tag{11}
$$

To solve equation (11) the gradient is taken and the solution will be

$$
F'(\omega) = (y - t)x + \gamma\omega = 0
$$

2.4 Balancing Principle

Balancing or Lepskii principle is used here to determine a stable value of γ given a starting approximate γ_0 which could be computed such that a-priori rule is satisfied. In this particular problem an assumed value of γ_0 is set to 0.5. The tolerance level θ, which defines the accepted difference in γ_k and γ_{k+1} is set to 0.01, i.e. $|\gamma_{k+1} - \gamma_k| \leq 0.01$.

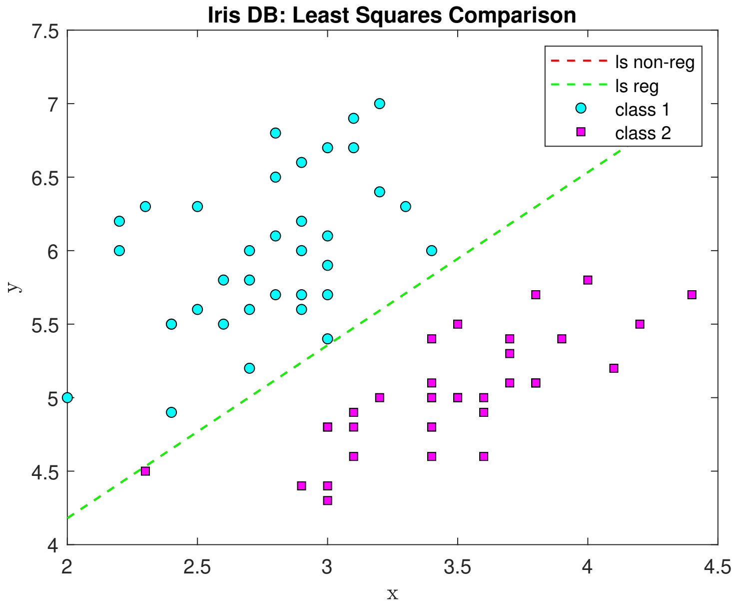
To compute the γ parameter one has to solve another minimization problem in order to compute ω_γ i.e. one uses the Value function

$$F(\gamma) = \inf_{\omega} J_\gamma(\omega) = \inf_{\omega} \frac{1}{2}||y(\omega) - t||_2^2 + \gamma||\omega||$$

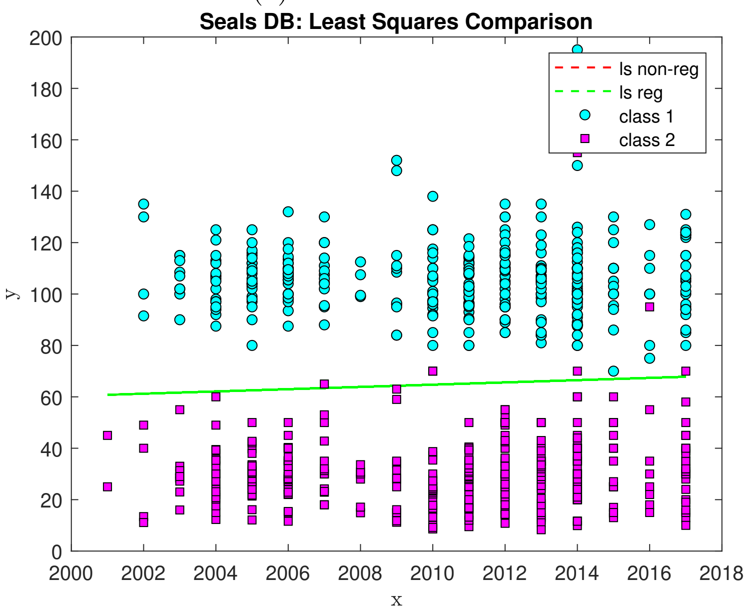
From this value function, ω is computed and used to update γ. One thing to note here is that after the stabilized regularization parameter has been calculated; it is being used while updating the weight vector in the perceptron learning algorithm but γ is not applied on ω_0 since it is the bias.

3 Numerical Results

3.1 Comparison for Least Squares



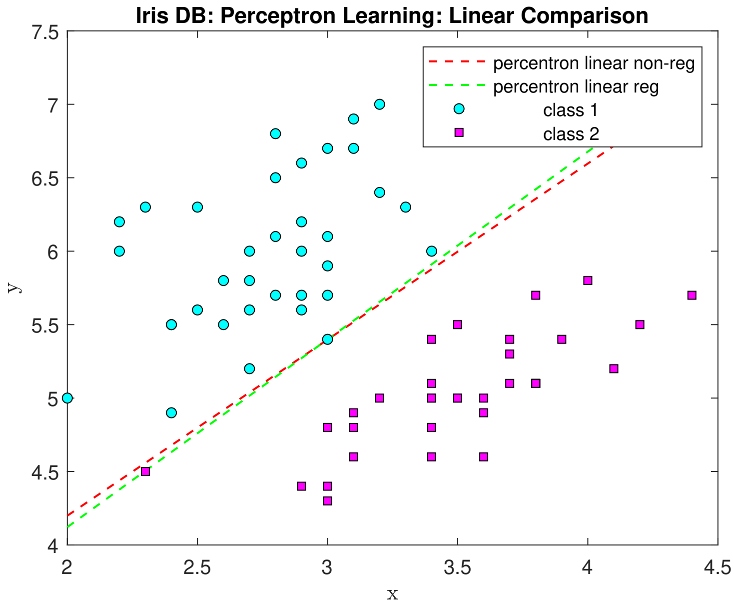
(a) On Iris Dataset



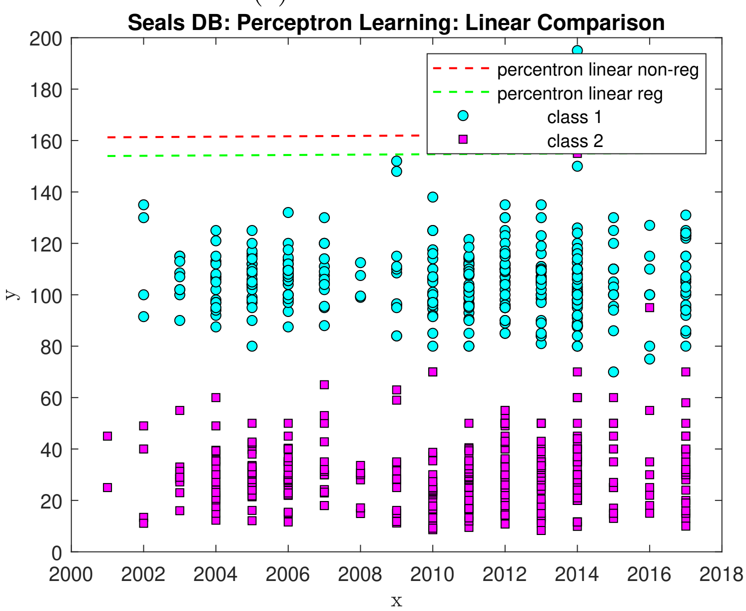
(b) On Seals Dataset

Figure 1: Non-Regularized vs Regularized Comparison for Least Squares on 2 Datasets

3.2 Comparison for Perceptron Learning (Order = 1)



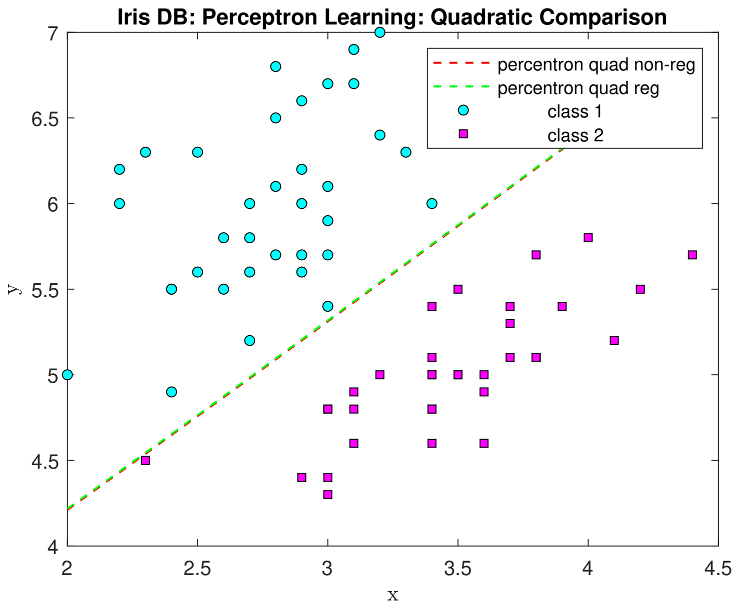
(a) On Iris Dataset



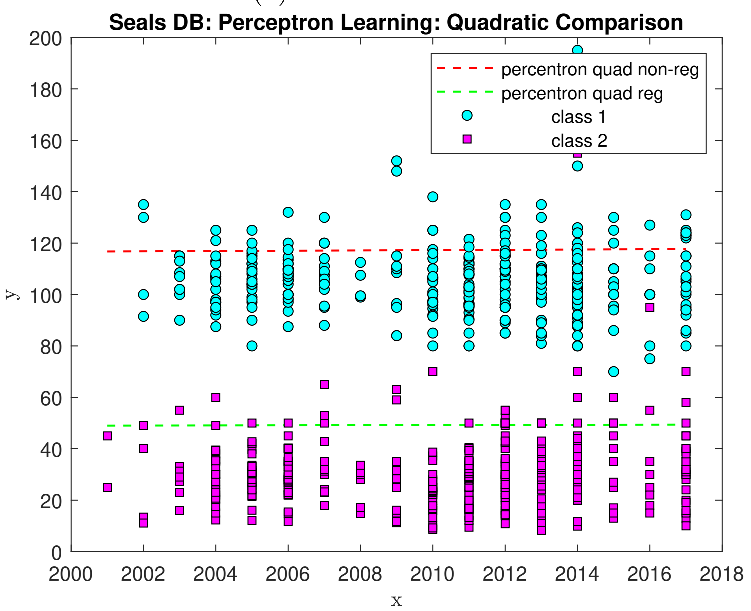
(b) On Seals Dataset

Figure 2: Non-Regularized vs Regularized Comparison for Perceptron Learning (Order = 1) on 2 Datasets

3.3 Comparison for Perceptron Learning (Order = 2)



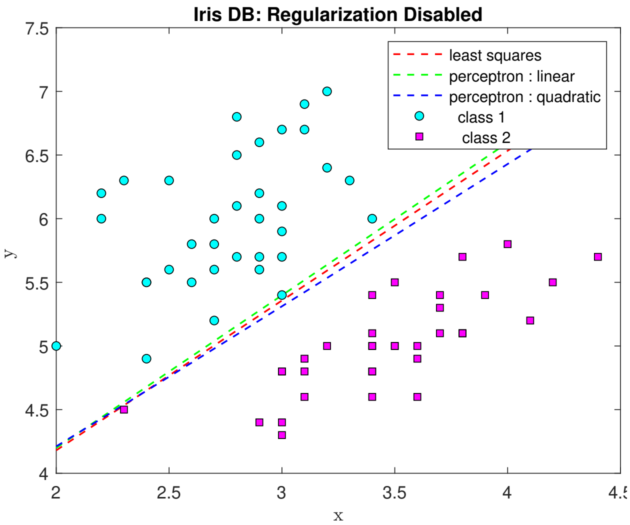
(a) On Iris Dataset



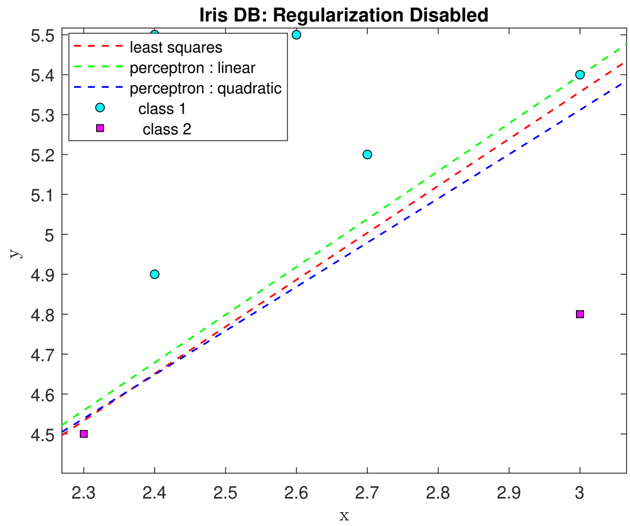
(b) On Seals Dataset

Figure 3: Non-Regularized vs Regularized Comparison for Perceptron Learning (Order = 2) on 2 Datasets

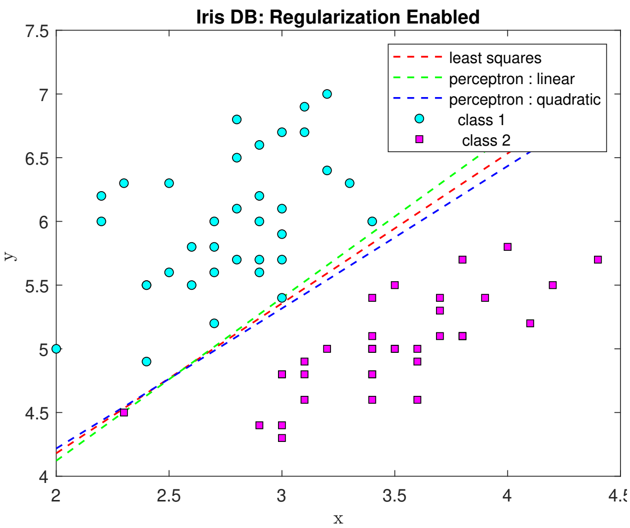
3.4 Comparison for All 3 Algorithms: Iris Dataset



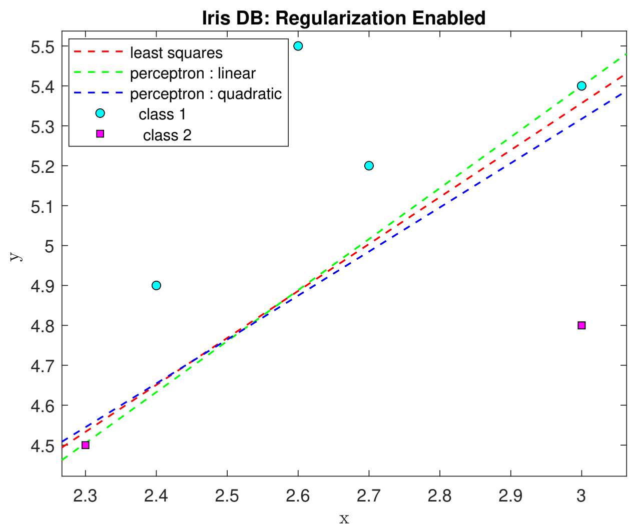
(a) Non-Regularized Iris



(b) Non-Regularized Iris: Zoomed In



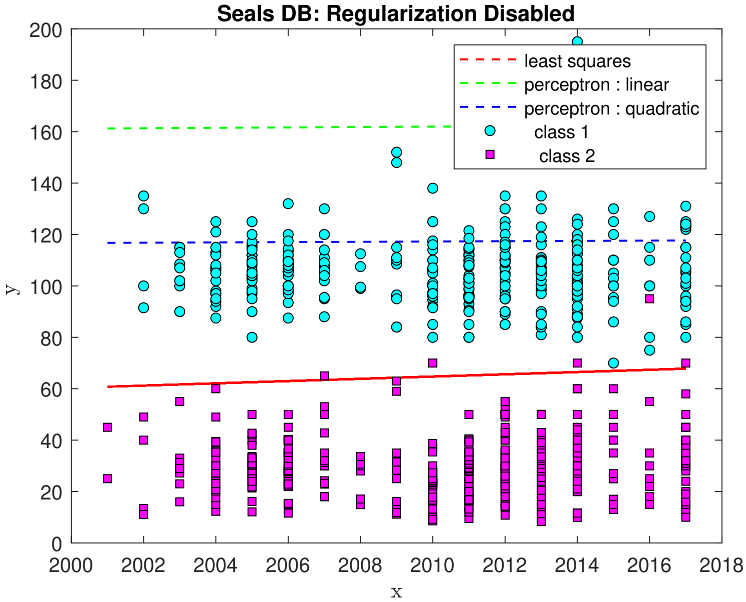
(c) Regularized Iris



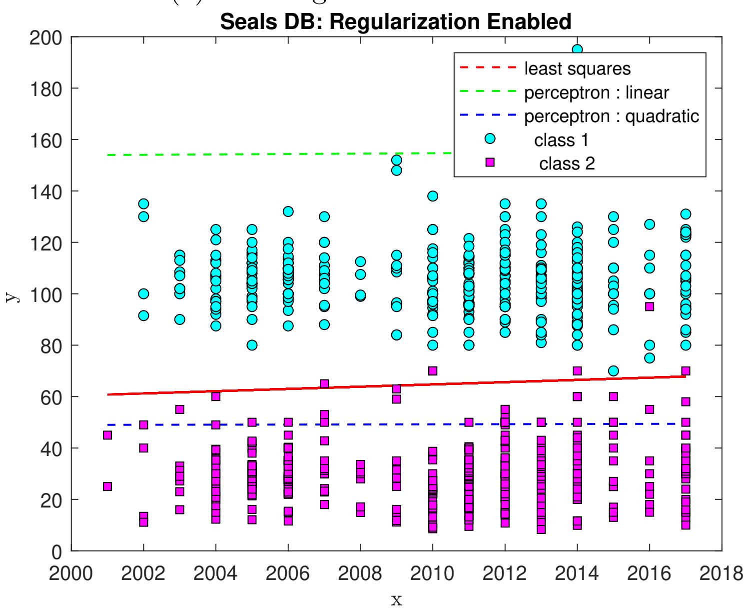
(d) Regularized Iris: Zoomed In

Figure 4: Comparison of All 3 Algorithms on Iris Dataset

3.5 Comparison for All 3 Algorithms: Seals Dataset



(a) Non-Regularized Seal Result

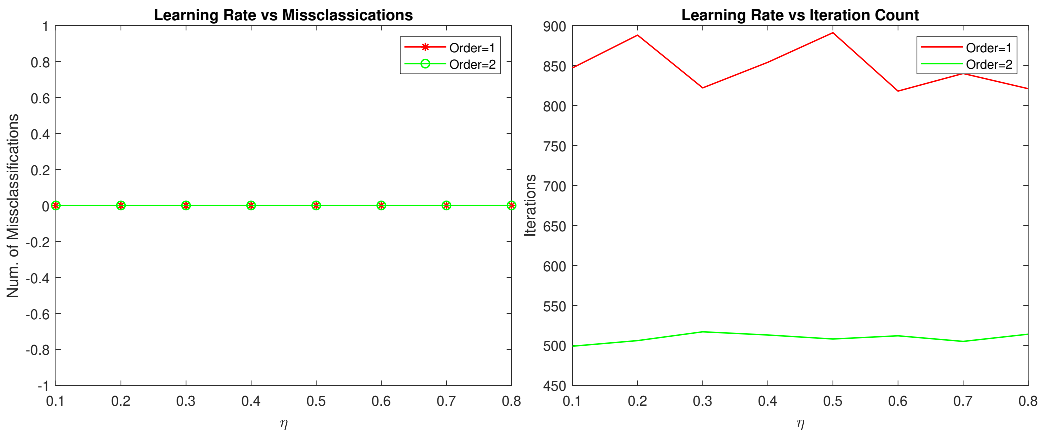


(b) Regularized Seal Result

Figure 5: Comparison of All 3 Algorithms on Seal Dataset

4 Conclusion

4.1 Varying Learning Rate



(a) η vs Number of Miss-classifications (b) η vs Number of Iterations

Figure 6: Effect of Varying Learning Rate on Iris.

Since the data selected from the iris dataset is linearly separable, all three algorithms are able to classify the data extremely well. Hence, one cannot see any change in the number of miss-classifications for perceptron learning algorithm. While there are some patterns in the variation for number of required iterations of the learning algorithm, they seem to be confined to a particular range. In case of order of 2, the algorithm shows little to no deviation in the required numbers of iterations wheres for order of 1, there is deviation of $\pm\ x\ \epsilon\ (50, 75)$. Differences and more interesting results could be obtained on more challenging datasets.

4.2 Best Classification Algorithm

The best algorithm considering both the iris as well as the seals dataset is the least squares algorithm. The reason for this is that perceptron learning fails when the data is not linearly separable, while least squares can still give us a reasonable solution. Considering the amount of computation required, the perceptron algorithm may necessarily not have an end point and every new iteration may deliver a better result but that is not the case for least squares. While the actual computation time required for was not calculated its fair to say that, it did not matter when computing for iris dataset.

4.3 Failure of Perceptron Learning Algorithm

Perceptron learning works best for linearly separable data and certainly fails for non-linear data. While there are certain special cases where by using some transformation one can still use this algorithm, for example if the data has a circular (two-dimensional) or spherical (three-dimensional) structure, using Polar to Cartesian transformation we could be able to make the data linearly separable and then use the algorithm. The failure of the algorithm can be seen in figure:(2b) and figure:(3b), The decision lines for order of 1 and 2 do not classify the dataset at all even with regularization order of 1 fails; while order of 2 shows substantial improvement.

References

[1] L, Beilina. *Numerical analysis of least squares and perceptron learning for classification problems*, Open Journal of Discrete Applied Mathematics, 2020.

A [MATLAB Code] Main Code

Filename: *"lab3_0.m"*

```matlab
%% Clear All
clc; %clear all; close all;

%% Load Datasets: IRIS and Seals
[xi, yi, ci, xti, yti, cti, cl_x1i, cl_y1i, cl_x2i, cl_y2i] = load_iris;
[xs, ys, cs, hyps, cl_x1s, cl_y1s, cl_x2s, cl_y2s, sl, sw] = load_seals;

%% Set Parameters
plot_iris = 1;          % set to 1 to plot iris result
plot_seal = 1;          % set to 1 to plot seal results
plot_varylr = 1;        % set to 1 to plot results of varying learning rate on iris
lr = 0.5;               % learning rate
cntLt = 1e6;            % count limit for perceptron learning
% For Regularization => gamma_0 : 0.5 ; it passed directly when calling function

%% Compute on IRIS: Regularization Disabled
fprintf('---------------IRIS Dataset----------------\n');
fprintf('----------Regularization Disabled----------\n');
fprintf('-------------------------------------------\n');
[x1i, y1i, x2i, y2i, y3_1i, y3_2i, countsi, missClassNumi, missClassRatei] = compute_iris(xi, yi, ci, xti, yti, cti, lr, 0);

%% Compute on IRIS: Regularization Enabled
fprintf('---------------IRIS Dataset----------------\n');
fprintf('----------Regularization  Enabled----------\n');
fprintf('-------------------------------------------\n');
[x1ir, y1ir, x2ir, y2ir, y3_1ir, y3_2ir, countsir, missClassNumir, missClassRateir] = compute_iris(xi, yi, ci, xti, yti, cti, lr, 0.5);

%% Compute on Seals: Regularization Disabled
fprintf('---------------Seals Dataset---------------\n');
fprintf('----------Regularization Disabled----------\n');
fprintf('-------------------------------------------\n');
[y1s, x2s, y2s, y3_1s, y3_2s] = compute_seal(xs, ys, cs, hyps, 0, 0.5, cntLt, sl, sw);

%% Compute on Seals: Regularization Enabled
fprintf('---------------Seals Dataset---------------\n');
```

```
fprintf('----------Regularization  Enabled----------\n');
fprintf('-------------------------------------------\n');
[y1sr, x2sr, y2sr, y3_1sr, y3_2sr] = compute_seal(xs, ys, cs, hyps, 0.5, 0.5, cntLt, sl, sw);

%% Vary Learning Rate
fprintf('-------------------------------------------\n');
fprintf('---------- Varying Learning Rate ----------\n');
fprintf('-----------------On IRIS-------------------\n');

vary_lr = 0.1:0.1:0.8;
itr_l = zeros(size(vary_lr));
itr_q = itr_l; mc_l = itr_l; mc_q = itr_l;

for j = 1:length(vary_lr)
    [~, ~, ~, ~, ~, ~, lr_c, lr_num, ~] = compute_iris(xi, yi, ci, xti, yti, cti, vary_lr(j), 0);
    itr_l(j) = lr_c(1); itr_q(j) = lr_c(2);
    mc_l(j) = lr_num(1); mc_q(j) = lr_num(2);
end

%% Legends for Plotting
ls_legend = {'ls non-reg', 'ls reg', 'class 1', 'class 2'};
pll_legend = {'percentron linear non-reg', 'percentron linear reg', '           class 1', '           class 2'};
plq_legend = {'percentron quad non-reg', 'percentron quad reg', '           class 1', '           class 2'};

%% Plot Results
if plot_iris == 1
    pre = 'Iris DB: ';
    plot_comparison(1, x1i, y1i, x1ir, y1ir, cl_x1i, cl_x2i, cl_y1i, cl_y2i, ls_legend, sprintf('%sLeast Squares Comparison',pre), 'iris_');
    plot_comparison(2, x2i, y2i, x2ir, y2ir, cl_x1i, cl_x2i, cl_y1i, cl_y2i, pll_legend, sprintf('%sPerceptron Learning: Linear Comparison',pre), 'iris_');
    plot_comparison(3, sort(xi), y3_1i, sort(xi), y3_1ir, cl_x1i, cl_x2i, cl_y1i, cl_y2i, plq_legend, sprintf('%sPerceptron Learning: Quadratic Comparison',pre), 'iris_');

    plot_all(4, x1i, y1i, x2i, y2i, sort(xi), y3_1i, cl_x1i, cl_x2i, cl_y1i, cl_y2i, sprintf('%sRegularization Disabled',pre), 'iris_');
```

```
    plot_all(5, x1ir, y1ir, x2ir, y2ir, sort(xi), y3_1ir, cl_x1i, cl_x2i, cl_y1i, cl_y2i, sprintf('%sRegularization Enabled',pre), 'iris_');
end

if plot_seal == 1
    j = 1;
    if plot_iris == 1
        j = 5;
    end
    pre = 'Seals DB: ';
    plot_comparison(j+1, xs, y1s, xs, y1sr, cl_x1s, cl_x2s, cl_y1s, cl_y2s, ls_legend, sprintf('%sLeast Squares Comparison',pre), 'seal_');
    plot_comparison(j+2, x2s, y2s, x2sr, y2sr, cl_x1s, cl_x2s, cl_y1s, cl_y2s, pll_legend, sprintf('%sPerceptron Learning: Linear Comparison',pre), 'seal_');
    plot_comparison(j+3, sort(xs), y3_1s, sort(xs), y3_1sr, cl_x1s, cl_x2s, cl_y1s, cl_y2s, plq_legend, sprintf('%sPerceptron Learning: Quadratic Comparison',pre), 'seal_');

    plot_all(j+4, xs, y1s, x2s, y2s, sort(xs), y3_1s, cl_x1s, cl_x2s, cl_y1s, cl_y2s, sprintf('%sRegularization Disabled',pre), 'seal_');
    plot_all(j+5, xs, y1sr, x2sr, y2sr, sort(xs), y3_1sr, cl_x1s, cl_x2s, cl_y1s, cl_y2s, sprintf('%sRegularization Enabled',pre), 'seal_');
end

if plot_varylr == 1
    k = 11;
    figure(k);
    plot(vary_lr, mc_l, '-* r', 'linewidth', 1); hold on;
    plot(vary_lr, mc_q, '-o g', 'linewidth', 1); hold off;
    legend({'Order=1', 'Order=2'});
    title('Learning Rate vs Missclassications');
    xlabel('\eta');
    ylabel('Num. of Missclassifications');
    saveas(gcf, sprintf('vary_lr_lab30_%d',k), 'epsc');
    saveas(gcf, sprintf('vary_lr_lab30_%d',k), 'png');

    figure(k+1);
    plot(vary_lr, itr_l, 'r', 'linewidth', 1); hold on;
    plot(vary_lr, itr_q, 'g', 'linewidth', 1); hold off;
    title('Learning Rate vs Iteration Count');
    legend({'Order=1', 'Order=2'});
```

```
    xlabel('\eta');
    ylabel('Iterations');
    saveas(gcf, sprintf('vary_lr_lab30_%d',k+1), 'epsc');
    saveas(gcf, sprintf('vary_lr_lab30_%d',k+1), 'png');
end

%% Plotting Functions
function plot_comparison(i, x1, y1, x1_r, y1_r, cl_x1, cl_x2, cl_y1, cl_y2, l, t, pre)
    figure(i);
    plot(x1, y1, '-- r', 'linewidth', 1); hold on;
    plot(x1_r, y1_r, '-- g', 'linewidth', 1); hold on;
    plot(cl_x1, cl_y1, "ko", "MarkerSize", 5, "MarkerFaceColor", "c"); hold on;
    plot(cl_x2, cl_y2, "ks", "MarkerSize", 5, "MarkerFaceColor", "m"); hold off;
    title(t); legend(l); xlabel("x", "Interpreter", "Latex"); ylabel("y", "Interpreter", "Latex");
    saveas(gcf, sprintf('%s_lab30_%d',pre,i), 'epsc');
    saveas(gcf, sprintf('%s_lab30_%d',pre,i), 'png');
end

function plot_all(i, x1, y1, x2, y2, x3, y3, cl_x1, cl_x2, cl_y1, cl_y2, t, pre)
    figure(i);

    % decision line: least squares
    plot(x1, y1, '-- r', 'linewidth', 1); hold on;

    % decision line : perceptron "linear" learning algorithm
    plot(x2, y2, '-- g', 'linewidth', 1); hold on;

    % decision line : perceptron "quadratic" learning algorithm
    plot(x3, y3, '-- b', 'linewidth', 1); hold on;

    % data
    plot(cl_x1, cl_y1, "ko", "MarkerSize", 5, "MarkerFaceColor", "c"); hold on;
    plot(cl_x2, cl_y2, "ks", "MarkerSize", 5, "MarkerFaceColor", "m"); hold off;

    % plot details
    legend('least squares', 'perceptron : linear', 'perceptron : quadratic', ' class 1', '  class 2');
    xlabel("x", "Interpreter", "Latex");
    ylabel(" y ", "Interpreter", "Latex");
    title(t);
```

```
    saveas(gcf, sprintf('%s_lab30_%d',pre,i), 'epsc');
    saveas(gcf, sprintf('%s_lab30_%d',pre,i), 'png');
end
```

B [MATLAB Code] Balancing Principle

Filename: *"balancing_principle.m"*

```
function gamma_updated = balancing_principle(gamma, A, t)
    %% Implementation
    gamma_updated = 0;
    C = 1;                              % zero-crossing method
    theta = 0.01;                       % tolerance
    count = 1;
    check_condition = 1;
    while (check_condition ~= 0)
        % Step 2: Compute Value Function 'J' to obtain omega 'w'
        % J = 0.5 * ((norm( (Aw - t), 2)^2) +  gamma*(norm(w, 2))^2);
        % 0.5 [ (2A')*(Aw - t) + (2*gamma*w) ] = 0
        % (A')Aw - (A')t + gamma*w = 0
        % ((A')A + gamma*I) w = (A')t
        % w = inv((A')A + gamma*I) * (A')t
        w = ( (A')*A + gamma*eye(size((A')*A)) ) \ ((A')*t);
        phi_bar = (A')*(A*w - t);
        psi_bar = 2*w;

        % Step 3: Update Regularization Parameter
        gamma_updated = C * (norm( phi_bar )^2 / norm( psi_bar )^2);
        count = count+1;
        if (abs(gamma_updated - gamma) > theta)
            gamma = gamma_updated;
        else
            check_condition = 0;
        end
        if count > 100
            break;
        end
    end
    fprintf('Regularization Iteration: %d\ngamma: %.12f\n', count, gamma_updated);
end
```

C [MATLAB Code] Compute on Iris Dataset

Filename: *"compute_iris.m"*

```
function [x1, y1, x2, y2, y3_1, y3_2, counts, missClassNum, missClassRate] = compute_iris(x, y, class, xt, yt, ct, lr, gamma_0)
    %% Function:  LeastSquares Problem Solver => min ||A*omega - t||, A =[1; x; y], t = class
    A = [ones(numel(x), 1) x' y'];
    if gamma_0 ~= 0
        B = (A') * A;
        gamma_updated = balancing_principle(gamma_0, A, class');
        B = B + gamma_updated*(eye(size(B)));
        wls = B\((A')*(class'));
    else
        wls = A\(class');
    end

    y1 = zeros(1,2);
    x1(1)= min(x); x1(2)=max(x);

    for i=1:1:2
        y1(i) =  ((0.5-wls(1))/wls(3)) - ((wls(2)/wls(3))*x1(i));
    end

    m = ( y1(1,2) - y1(1,1) ) / (x1(2) - x1(1));
    pred = ct';

    mx = mean(x(1,:));
    my = mean(y(1,:));
    b = my/mx;

    for i = 1:length(xt)
        if (yt(1,i) - ((m*xt(1,i)) + b) ) > 0     % point is above line
            pred(i) = 1;
        elseif (yt(1,i) - ((m*xt(1,i)) + b) ) < 0 % point is below line
            pred(i) = 0;
        end
    end

    missClassNumLS = (length(pred)-sum(pred == ct'));
    missClassRateLS = missClassNumLS/length(pred);
    fprintf("Number of MissClassified Points : %d\nMissClassification Rate: %.3f\n", missClassNumLS,
```

```
missClassRateLS);

%% Function: Perceptron Learning Algorithm: d = 1
% initialize parameters
rng(1, 'philox');
wl = randn(3, 1);
hyp = zeros(size(x,2), 1)';
best_wl = wl;
count = 0;
pred = ct;

if gamma_0 ~= 0
    gamma_updated = balancing_principle(gamma_0, [ones(numel(x), 1) x' y'], class');
else
    gamma_updated = 0;
end

while sum(class ~= hyp)
    for i = 1:size(x,2)
        if (wl(1) + wl(2)*x(i)  + wl(3)*y(i)) > 0
            hyp(i) = 1;
        else
            hyp(i) = 0;
        end
        % save current weights
        best_wl = wl;

        % update weights and gamma
        wl(1) = wl(1) + lr*(class(i) - hyp(i));
        wl(2) = wl(2) + lr*( ((class(i) - hyp(i)) * x(i)) + (gamma_updated*best_wl(2)) );
        wl(3) = wl(3) + lr*( ((class(i) - hyp(i)) * y(i)) + (gamma_updated*best_wl(2)) );
    end

    count = count+1;
    if count > 1e10
        break
    end
end

y2 = zeros(1,2);
x2(1)= min(x);    x2(2)=max(x);

for i=1:1:2
    y2(i) = (-best_wl(1)/best_wl(3)) - ((best_wl(2)/best_wl(3))*x2(i));
end
```

```
counts(1) = count-1;
fprintf('Iterations: %d\n', count-1);

for i=1:length(xt)
    Arow = [1 xt(i) yt(i)];
    if (Arow*best_wl) > 0
        pred(1,i) = 1;
    else
        pred(1,i) = 0;
    end
end

missClassNumL = (length(pred)-sum(pred == ct));
missClassRateL = missClassNumL/length(pred);

fprintf("Number of MissClassified Points : %d\nMissClassification Rate: %.3f\n", missClassNumL, missClassRateL);

%% Function: Perceptron Learning Algorithm: d = 2
% initialize parameters
rng(1, 'philox');
wq = randn(6, 1);
hyp = zeros(size(x,2), 1)';
best_wq = wq;
count = 0;
pred = ct;
z1 = pred;
z2 = z1;

if gamma_0 ~= 0
    gamma_updated = balancing_principle(gamma_0, [ones(numel(x), 1) x' y'], class');
else
    gamma_updated = 0;
end

while sum(class ~= hyp)
    for i=1:size(x,2)
        if wq(1) + wq(2)*x(i)  + wq(3)*y(i)  + ...
            wq(4)*x(i)*x(i)  + wq(5)*x(i)*y(i) + ...
            wq(6)*y(i)*y(i)  > 0
            hyp(i)=1;
        else
            hyp(i)=0;
        end
        % save current weights
        best_wq = wq;
```

```
            % update weights
            wq(1) = wq(1) + lr*(class(i) - hyp(i));
            wq(2) = wq(2) + lr*( ((class(i) - hyp(i)) *x(i)) + (gamma_updated*best_wq(2)) );
            wq(3) = wq(3) + lr*( ((class(i) - hyp(i)) *y(i)) + (gamma_updated*best_wq(3)) );
            wq(4) = wq(4) + lr*( ((class(i) - hyp(i)) *x(i)*x(i)) + (gamma_updated*best_wq(4)) );
            wq(5) = wq(5) + lr*( ((class(i) - hyp(i)) *x(i)*y(i)) + (gamma_updated*best_wq(5)) );
            wq(6) = wq(6) + lr*( ((class(i) - hyp(i)) *y(i)*y(i)) + (gamma_updated*best_wq(6)) );
        end

        count = count+1;
        if count > 1e10
            break
        end
    end

    q_a = best_wq(6);
    x = sort(x);

    y3_1 = zeros(1, size(x,2));
    y3_2 = zeros(1, size(x,2));

    for i = 1:size(x,2)
        q_b = best_wq(5)*x(i) + best_wq(3);
        q_c = best_wq(1) + best_wq(2)*x(i) + best_wq(4)*x(i)*x(i);
        D  = q_b*q_b - 4*q_a*q_c;
        y3_1(i) = (-q_b + sqrt(D))/(2*q_a);
        y3_2(i) = (-q_b - sqrt(D))/(2*q_a);
    end

    counts(2) = count-1;
    fprintf('Iterations: %d\n', count-1);

    qa = best_wq(6);
    for i=1:length(xt)
        qb = best_wq(5)*xt(i) + best_wq(3);
        qc = best_wq(1) + best_wq(2)*xt(i) + best_wq(4)*xt(i)*xt(i);
        D  = qb*qb - 4*q_a*qc;
        z1(i) = (-qb + sqrt(D))/(2*qa);
        z2(i) = (-qb - sqrt(D))/(2*qa);
        if (yt(i) > z1(i))
            pred(i) = 1;
        else
```

```
            pred(i) = 0;
        end
    end

    missClassNumQ = (length(pred)-sum(pred == ct));
    missClassRateQ = missClassNumQ/length(pred);

    fprintf("Number of MissClassified Points : %d\nMissClassification Rate: %.3f\n", missClassNumQ, missClassRateQ);

    missClassNum = [missClassNumLS, missClassNumL, missClassNumQ];
    missClassRate = [missClassRateLS, missClassRateL, missClassRateQ];
end
```

D [MATLAB Code] Compute on Seals Dataset

Filename: *"compute_seal.m"*

```
function [ls_y, pll_x, pll_y, plq_y1, plq_y2] = compute_seal(x, y, class, hyp, gamma_0, lr, CountLimit, Seal_l, Seal_w)
    % Generate Vandemorte Matrix
    fprintf('1. Generating Vandemorte Matrix\n');
    m = size(Seal_w, 1);
    d = 1;
    A = [];
    for i=1:1:m
        for j=1:1:d+1
            A(i,j)=power(x(i),j-1);
            if isnan(Seal_l(i))
                y(i)= 0;
            end
        end
    end

    sch = 0;

    for i=m+1:1:2*m
        sch = sch+1;
        for j=1:1:d+1
            A(i,j)=power(x(i),j-1);
            if isnan(Seal_w(sch))
                y(i)= 0;
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%
    %%%                    LEAST SQUARES
    %%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fprintf('2. Computing Least Squares\n');
    if gamma_0 ~= 0
        B = (A') * A;
        gamma_updated = balancing_principle(gamma_0, A, y');
        B = B + gamma_updated*(eye(size(B)));
```

```
        w = B\((A')*(y'));
    else
        w = A\(y');
    end

    ls_y = A*w;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%
    %%%                 PERCEPTRON LINEAR ALGO
    %%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fprintf('3. Computing Solution by Perceptron "d = 1" Learning Algorithm\n');
    wl = randn(3, 1);
    best_wl = wl;
    count = 0;
    hyp1 = hyp;

    if gamma_0 ~= 0
        gamma_updated = balancing_principle(gamma_0, A, y');
    else
        gamma_updated = 0;
    end

    while sum(class ~= hyp1)
        for i = 1:size(x,2)
            if (wl(1) + wl(2)*x(i)  + wl(3)*y(i)) > 0
                hyp1(i) = 1;
            else
                hyp1(i) = 0;
            end
            % save current weights
            best_wl = wl;

            % update weights and gamma
            wl(1) = wl(1) + lr*(class(i) - hyp1(i));
            wl(2) = wl(2) + lr*( ((class(i) - hyp1(i)) * x(i)) + (gamma_updated*best_wl(2)) );
            wl(3) = wl(3) + lr*( ((class(i) - hyp1(i)) * y(i)) + (gamma_updated*best_wl(2)) );
        end

```

```
        count = count+1;
        if count > CountLimit
            break
        end
    end

    pll_y = zeros(1,2);
    pll_x(1)= min(x);    pll_x(2)=max(x);

    for i=1:1:2
        pll_y(i) = (-best_wl(1)/best_wl(3)) - ((best_wl(2)/best_wl(3))*pll_x(i));
    end

    fprintf('Iterations: %d\n', count-1);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%
    %%%                  PERCEPTRON QUAD ALGO
    %%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fprintf('4. Computing Solution by Perceptron "d = 2" Learning Algorithm\n');
    wq = randn(6, 1);
    hyp2 = hyp;
    best_wq = wq;
    count = 0;

    if gamma_0 ~= 0
        gamma_updated = balancing_principle(gamma_0, A, y');
    else
        gamma_updated = 0;
    end

    while sum(class ~= hyp2)
        for i=1:size(x,2)
            if wq(1) + wq(2)*x(i)  + wq(3)*y(i)  + ...
                wq(4)*x(i)*x(i)  + wq(5)*x(i)*y(i) + ...
                wq(6)*y(i)*y(i)  > 0
                hyp2(i)=1;
            else
                hyp2(i)=0;
            end
```

```
            % save current weights
            best_wq = wq;
            % update weights
            wq(1) = wq(1) + lr*(class(i) - hyp2(i));
            wq(2) = wq(2) + lr*( ((class(i) - hyp2(i)) *x(i)) + (gamma_updated*best_wq(2)) );
            wq(3) = wq(3) + lr*( ((class(i) - hyp2(i)) *y(i)) + (gamma_updated*best_wq(3)) );
            wq(4) = wq(4) + lr*( ((class(i) - hyp2(i)) *x(i)*x(i)) + (gamma_updated*best_wq(4)) );
            wq(5) = wq(5) + lr*( ((class(i) - hyp2(i)) *x(i)*y(i)) + (gamma_updated*best_wq(5)) );
            wq(6) = wq(6) + lr*( ((class(i) - hyp2(i)) *y(i)*y(i)) + (gamma_updated*best_wq(6)) );
        end

        count = count+1;
        if count > CountLimit
            break
        end
    end

    q_a = best_wq(6);
    x = sort(x);

    plq_y1 = zeros(1, size(x,2));
    plq_y2 = zeros(1, size(x,2));

    for i = 1:size(x,2)
        q_b = best_wq(5)*x(i) + best_wq(3);
        q_c = best_wq(1) + best_wq(2)*x(i) + best_wq(4)*x(i)*x(i);
        D  = q_b*q_b - 4*q_a*q_c;
        plq_y1(i) = (-q_b + sqrt(D))/(2*q_a);
        plq_y2(i) = (-q_b - sqrt(D))/(2*q_a);
    end
    fprintf('Iterations: %d\n', count-1);
end
```

E [MATLAB Code] Load Iris Dataset

Filename: *"load_iris.m"*

```matlab
function [x, y, class, xt, yt, ct, cl_x1, cl_y1, cl_x2, cl_y2] = load_iris
    data = csvread("iris.csv");

    Xiris = data(:, 1)'; Yiris = data(:, 2)'; Ciris = data(:, 3)';
    x = Xiris(1, :); y = Yiris(1, :); class = Ciris(1, :);

    sortIdx = randperm(length(x));
    x = x(sortIdx);
    y = y(sortIdx);
    class = class(sortIdx);

    testSizeIndx = 70;
    xt = x(1,testSizeIndx+1:100); yt = y(1,testSizeIndx+1:100); ct = class(1,testSizeIndx+1:100);
    x = x(1,1:testSizeIndx); y = y(1,1:testSizeIndx); class = class(1,1:testSizeIndx);

    cl_x1 = x(class==1); cl_y1 = y(class==1);
    cl_x2 = x(class==0); cl_y2 = y(class==0);

    clt_x1 = xt(ct==1); clt_y1 = yt(ct==1);
    clt_x2 = xt(ct==0); clt_y2 = yt(ct==0);
end
```

F [MATLAB Code] Load Seals Dataset

Filename: *"load_seals.m"*

```
function [x, y, class, hyp, cl_x1, cl_y1, cl_x2, cl_y2,
    seal_length, seal_weight] = load_seals
    seal_table = readtable('DatabaseGreySeal.xlsx');
    seal_year = seal_table.(3);
    seal_weight = seal_table.(10);
    seal_length = str2double(seal_table.(11));

    m = size(seal_weight,1);

    x=zeros(1,2*m);
    y=zeros(1,2*m);

    sch1=0;  sch2=0;

    class = zeros(1,m);
    hyp =   zeros(1,m);

    for i=1:1:m

        sch1 = sch1 +1;
        cl_y1(sch1)= seal_length(i);
        cl_x1(sch1) = seal_year(i);
        x(i) =  seal_year(i);

        sch2 = sch2 +1;
        cl_y2(sch2)= seal_weight(i);
        cl_x2(sch2) = seal_year(i);
        y(i) = seal_length(i);

        class(sch1) = 1;
        hyp(sch1)= 0;
    end

    i=0; sch3=0;

    for i=m+1:1:2*m
        sch3 = sch3 +1;
        y(i) = seal_weight(sch3);
        x(i) = seal_year(sch3);

        class(i) = 0;
        hyp(i)= 1;
    end

%     testSize = 0;
```

```
%      testSizeIndx = round(length(x)*(1-testSize));
%      xt = x(1,testSizeIndx+1:end); yt = y(1,testSizeIndx+1:end); ct = class(1,testSizeIndx+1:end);
%      x = x(1,1:testSizeIndx); y = y(1,1:testSizeIndx); class = class(1,1:testSizeIndx);
end
```

G [MATLAB Console Output]

Filename: *"OUTPUTLOG.m"*

```
Warning: Table variable names were modified to make them
    valid MATLAB identifiers. The original names are saved
     in the
VariableDescriptions property.
---------------IRIS Dataset----------------
----------Regularization Disabled----------
-------------------------------------------
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 891
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 508
Number of MissClassified Points : 0
MissClassification Rate: 0.000
---------------IRIS Dataset----------------
----------Regularization  Enabled----------
-------------------------------------------
Regularization Iteration: 4
gamma: 0.000000238419
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Regularization Iteration: 4
gamma: 0.000000238419
Iterations: 687
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Regularization Iteration: 4
gamma: 0.000000238419
Iterations: 511
Number of MissClassified Points : 0
MissClassification Rate: 0.000
---------------Seals Dataset---------------
----------Regularization Disabled----------
-------------------------------------------
1. Generating Vandemorte Matrix
2. Computing Least Squares
3. Computing Solution by Perceptron "d = 1" Learning
    Algorithm
Iterations: 1000000
4. Computing Solution by Perceptron "d = 2" Learning
    Algorithm
Iterations: 1000000
---------------Seals Dataset---------------
----------Regularization  Enabled----------
```

```
-------------------------------------------
1. Generating Vandemorte Matrix
2. Computing Least Squares
Regularization Iteration: 4
gamma: 0.000000238419
3. Computing Solution by Perceptron "d = 1" Learning
   Algorithm
Regularization Iteration: 4
gamma: 0.000000238419
Iterations: 1000000
4. Computing Solution by Perceptron "d = 2" Learning
   Algorithm
Regularization Iteration: 4
gamma: 0.000000238419
Iterations: 1000000
-------------------------------------------
---------- Varying Learning Rate ----------
-----------------On IRIS-------------------
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 847
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 499
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 888
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 506
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 822
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 517
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 854
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 513
Number of MissClassified Points : 0
```

```
MissClassification Rate: 0.000
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 891
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 508
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 818
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 512
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 840
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 505
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 821
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Iterations: 514
Number of MissClassified Points : 0
MissClassification Rate: 0.000
Warning: Imaginary parts of complex X and/or Y arguments
    ignored
> In lab3_final>plot_comparison (line 111)
  In lab3_final (line 79)
Warning: Imaginary parts of complex X and/or Y arguments
    ignored
> In lab3_final>plot_all (line 130)
  In lab3_final (line 81)
>>
```