

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

1	Introduction	4
2	Theory, methods and algorithms	4
2.1	Methodology	4
2.2	Least Squares for Classification	5
2.3	Perceptron Learning	6
2.4	Balancing Principle	8
3	Numerical Results	9
3.1	Comparison for Least Squares	9
3.2	Comparison for Perceptron Learning (Order = 1)	10
3.3	Comparison for Perceptron Learning (Order = 2)	11
3.4	Comparison for All 3 Algorithms: Iris Dataset	12
3.5	Comparison for All 3 Algorithms: Seals Dataset	13
4	Conclusion	14
4.1	Varying Learning Rate	14
4.2	Best Classification Algorithm	14
4.3	Failure of Perceptron Learning Algorithm	15
A	[MATLAB Code] Main Code	17
B	[MATLAB Code] Balancing Principle	22
C	[MATLAB Code] Compute on Iris Dataset	23
D	[MATLAB Code] Compute on Seals Dataset	28
E	[MATLAB Code] Load Iris Dataset	32
F	[MATLAB Code] Load Seals Dataset	33
G	[MATLAB Console Output]	35

List of Figures

1	Non-Regularized vs Regularized Comparison for Least Squares on 2 Datasets	9
2	Non-Regularized vs Regularized Comparison for Perceptron Learning (Order = 1) on 2 Datasets	10
3	Non-Regularized vs Regularized Comparison for Perceptron Learning (Order = 2) on 2 Datasets	11
4	Comparison of All 3 Algorithms on Iris Dataset	12
5	Comparison of All 3 Algorithms on Seal Dataset	13
6	Effect of Varying Learning Rate on Iris.	14

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:

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

and so forth, the LS problem is finally written as

$$\min_w \frac{1}{2} \|A\omega - t\|_2^2 \quad A \in R^{N \times M} \text{ where } N > M, t \in R^N, \omega \in R^M \quad (3)$$

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

$$\min_w \frac{1}{2} \|A\omega - t\|_2^2 + \frac{\gamma}{2} \|\omega\|_2^2 \quad A \in R^{N \times M} \text{ where } N > M, t \in R^N, \omega \in R^M \quad (4)$$

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:

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

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 continuous update strategy for its weight's such that an optimum decision line (two-dimensional) is formed. A perceptron is computed in the following way:

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

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

$$\omega^T x = 0 \quad (7)$$

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

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

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

$$z_1 = x_1 \quad (9a)$$

$$z_2 = x_2 \quad (9b)$$

$$z_3 = (x_1)^2 \quad (9c)$$

$$z_4 = (x_1)(x_2) \quad (9d)$$

$$z_5 = (x_2)^2 \quad (9e)$$

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 $\omega_{i=0,1,\dots,5}$ and x_1). To compute x_2 , the following can be performed:

$$0 = \omega_5(x_2)^2 + (\omega_2 + \omega_4x_1)(x_2) + (\omega_0 + \omega_1x_1 + \omega_3x_1^2) \quad (10a)$$

$$0 = a(x_2)^2 + b(x_2) + c \quad (10b)$$

$$a = \omega_5 \quad (10c)$$

$$b = \omega_2 + \omega_4x_1 \quad (10d)$$

$$c = \omega_0 + \omega_1x_1 + \omega_3x_1^2 \quad (10e)$$

$$x_2 = -\frac{b \pm \sqrt{b^2 - 4ac}}{2 * a} \quad (10f)$$

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

$$F(\omega) = \min_w \frac{1}{2} ||t - y(x, \omega)||_2^2 + \frac{\gamma}{2} ||\omega||_2^2 \quad A \in R^{N * M} \quad (11)$$

where $N > M, t \in R^N, \omega \in R^M$

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. $|\gamma_{k+1} - \gamma_k| \leq 0.01$.

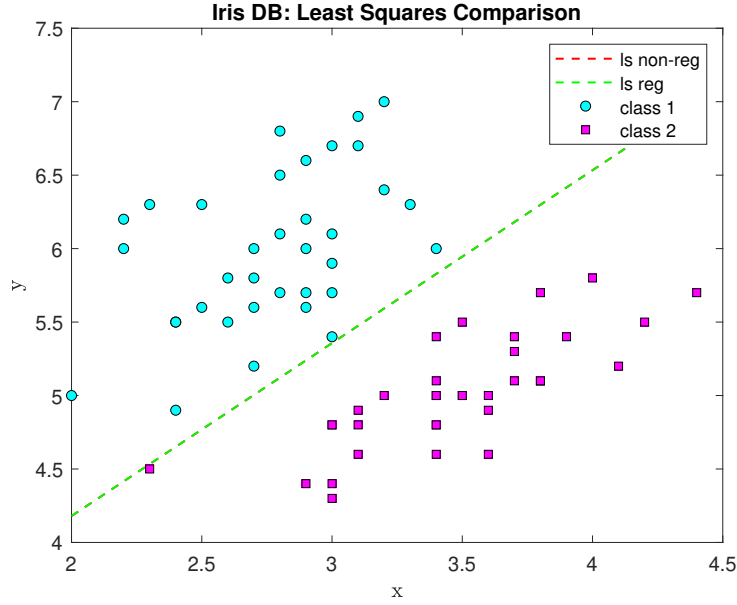
To compute the γ parameter one has to solve another minimization problem in order to compute ω_γ i.e. one uses the Value function

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

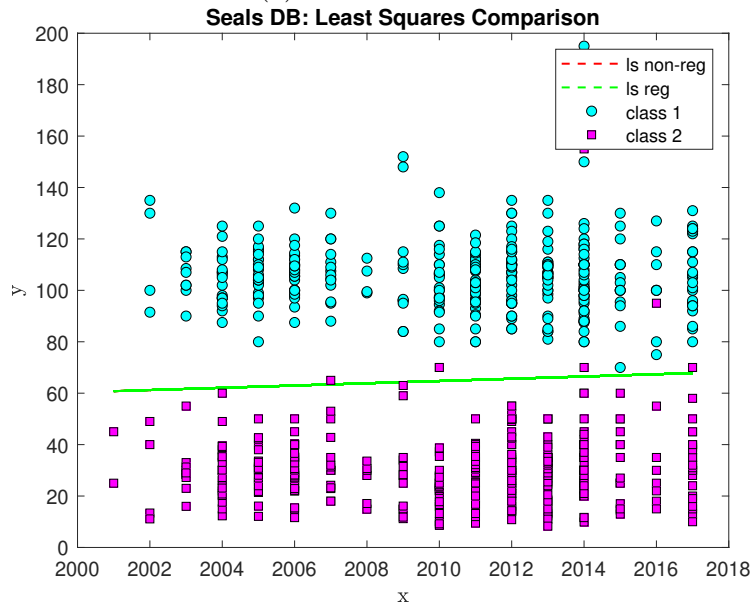
From this value function, ω is computed and used to update γ . One thing to note here is that after the stabilized regularization parameter has been calculated; it is being used while updating the weight vector in the perceptron learning algorithm but γ is not applied on ω_0 since it is the bias.

3 Numerical Results

3.1 Comparison for Least Squares



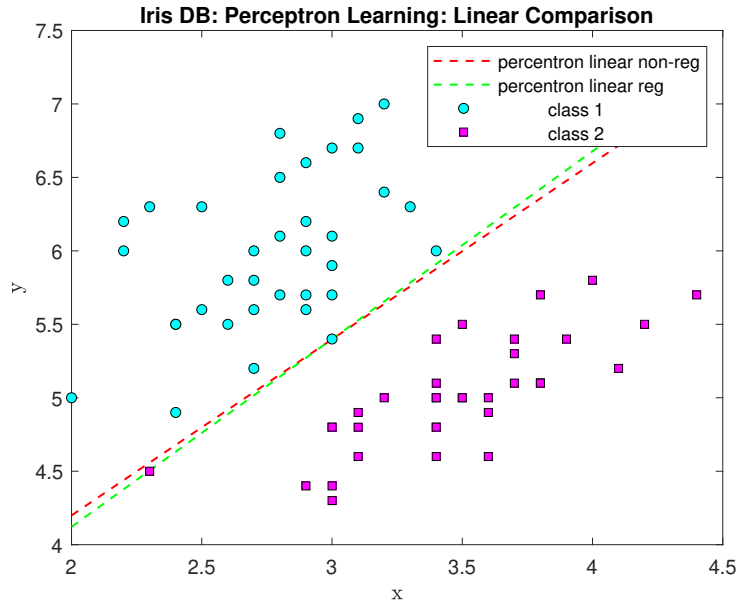
(a) On Iris Dataset



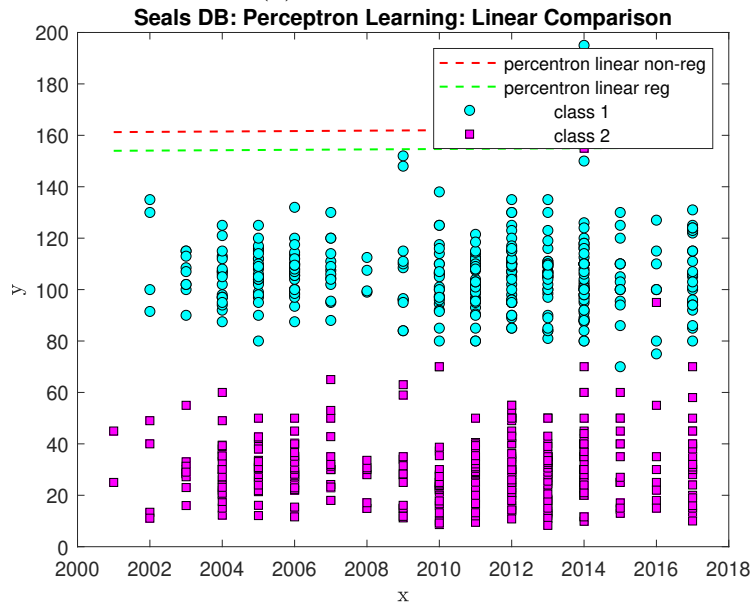
(b) On Seals Dataset

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

3.2 Comparison for Perceptron Learning (Order = 1)



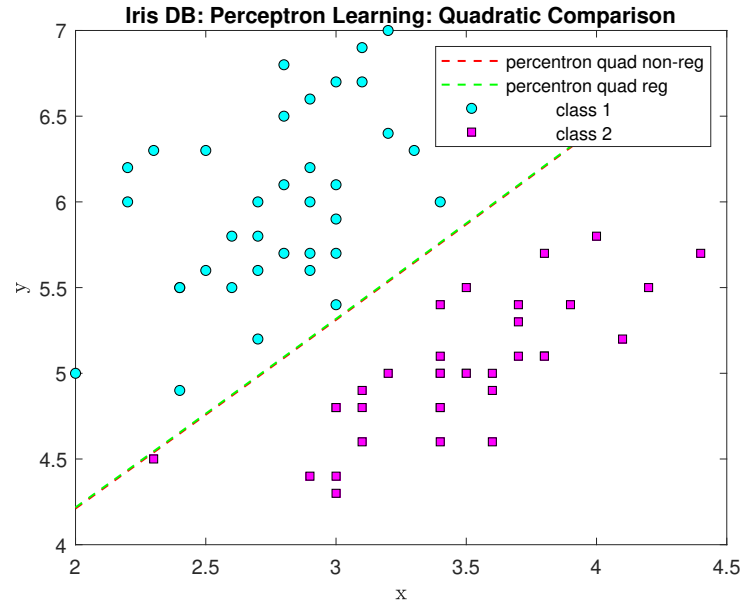
(a) On Iris Dataset



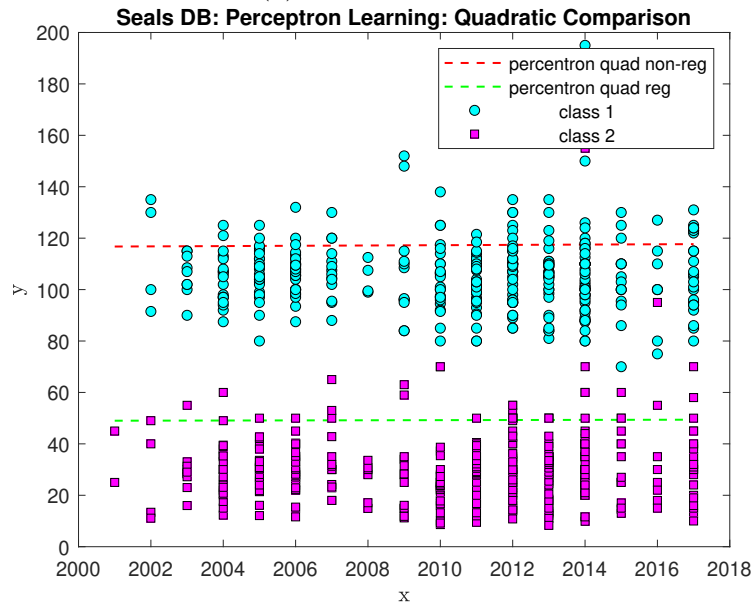
(b) On Seals Dataset

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

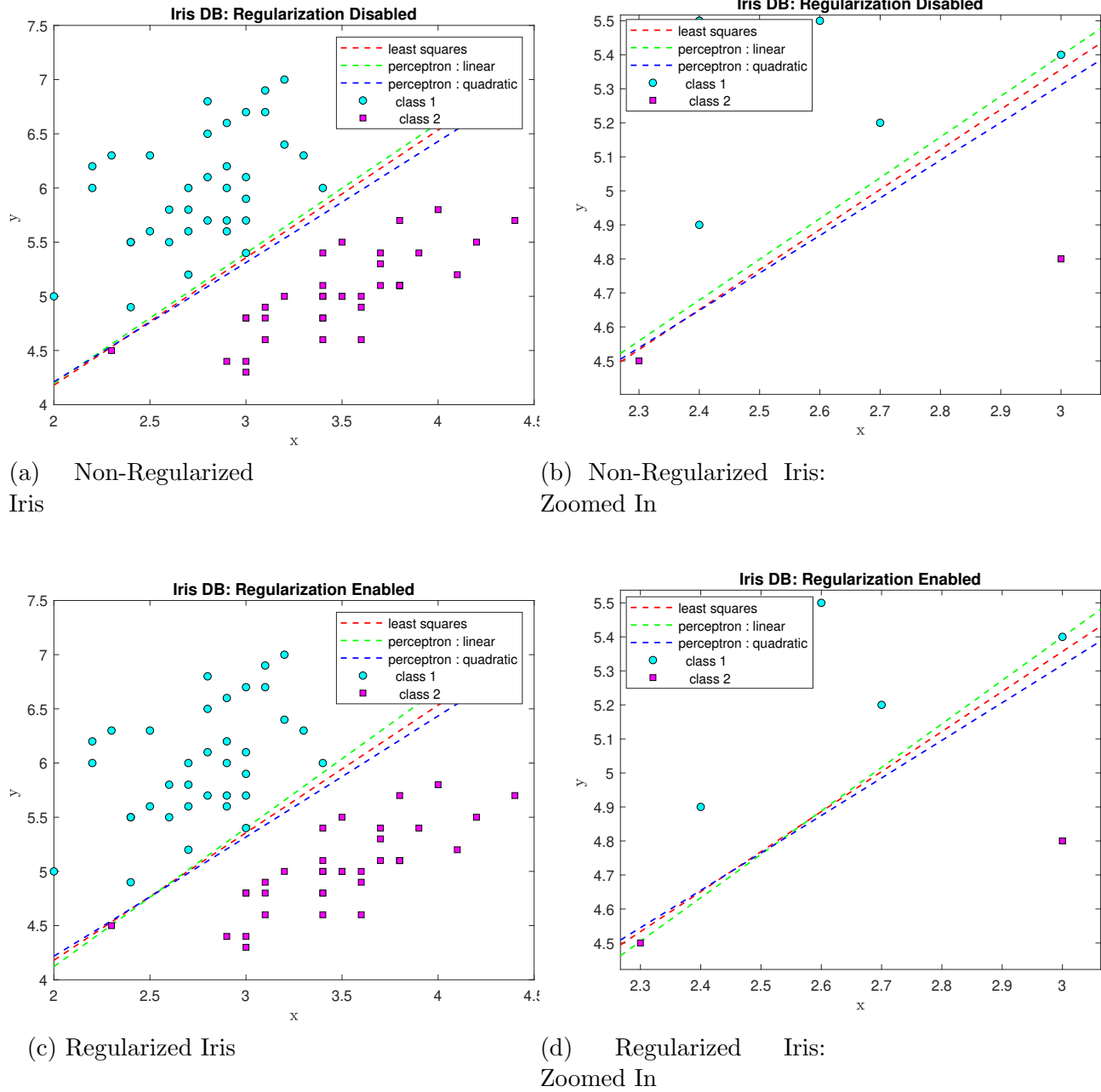


Figure 4: Comparison of All 3 Algorithms on Iris Dataset

3.5 Comparison for All 3 Algorithms: Seals Dataset

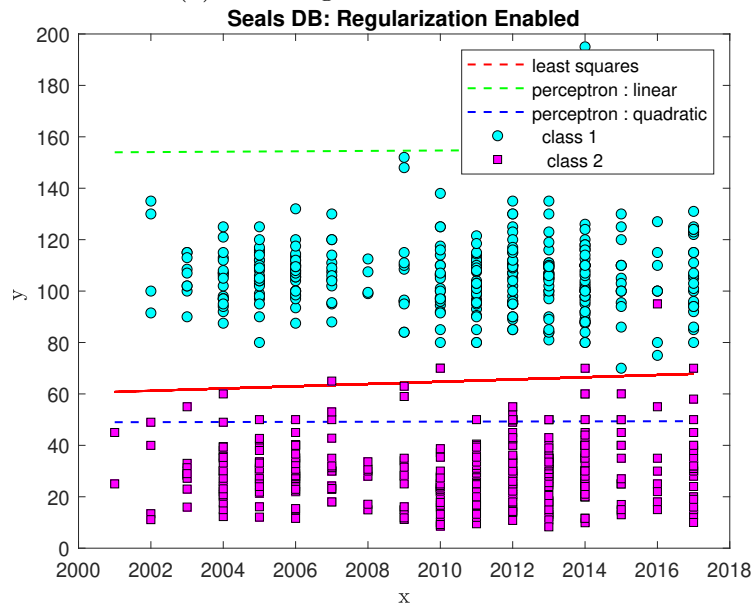
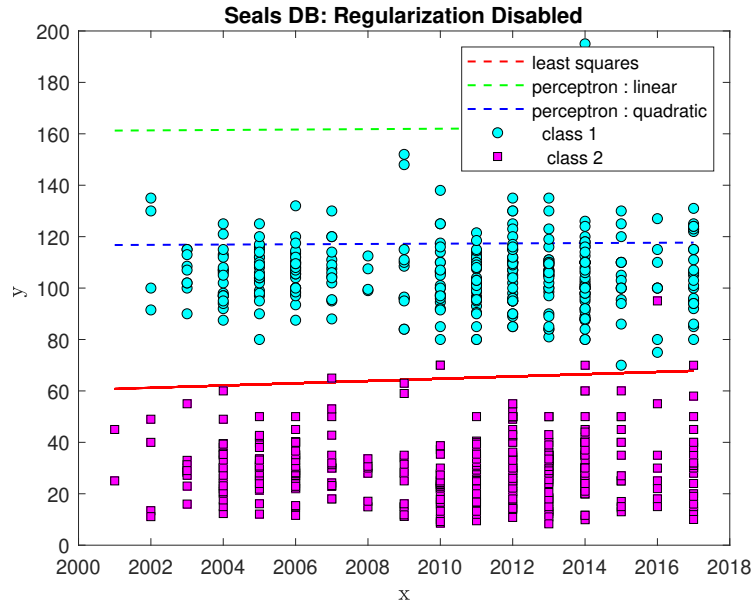


Figure 5: Comparison of All 3 Algorithms on Seal Dataset

4 Conclusion

4.1 Varying Learning Rate

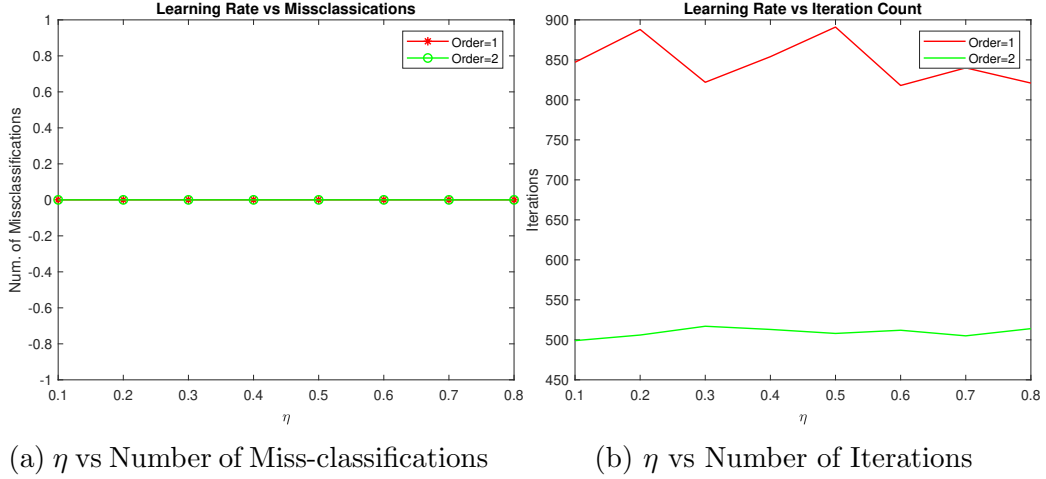


Figure 6: Effect of Varying Learning Rate on Iris.

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 whereas for order of 1, there is deviation of $\pm x \in (50, 75)$. 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"

```
1 %% Clear All
2 clc; %clear all; close all;
3
4 %% Load Datasets: IRIS and Seals
5 [xi, yi, ci, xti, yti, cti, cl_x1i, cl_y1i, cl_x2i,
   cl_y2i] = load_iris;
6 [xs, ys, cs, hyps, cl_x1s, cl_y1s, cl_x2s, cl_y2s, sl, sw
   ] = load_seals;
7
8 %% Set Parameters
9 plot_iris = 1;           % set to 1 to plot iris result
10 plot_seal = 1;          % set to 1 to plot seal results
11 plot_varylr = 1;        % set to 1 to plot results of
    varying learning rate on iris
12 lr = 0.5;              % learning rate
13 cntLt = 1e6;           % count limit for perceptron
    learning
14 % For Regularization => gamma_0 : 0.5 ; it passed
    directly when calling function
15
16 %% Compute on IRIS: Regularization Disabled
17 fprintf('-----IRIS Dataset-----\n');
18 fprintf('-----Regularization Disabled-----\n');
19 fprintf('-----\n');
20 [x1i, y1i, x2i, y2i, y3_1i, y3_2i, counts_i, missClassNumi,
    missClassRatei] = compute_iris(xi, yi, ci, xti, yti,
    cti, lr, 0);
21
22 %% Compute on IRIS: Regularization Enabled
23 fprintf('-----IRIS Dataset-----\n');
24 fprintf('-----Regularization Enabled-----\n');
25 fprintf('-----\n');
26 [x1ir, y1ir, x2ir, y2ir, y3_1ir, y3_2ir, countsir,
    missClassNumir, missClassRateir] = compute_iris(xi, yi,
    ci, xti, yti, cti, lr, 0.5);
27
28 %% Compute on Seals: Regularization Disabled
29 fprintf('-----Seals Dataset-----\n');
30 fprintf('-----Regularization Disabled-----\n');
31 fprintf('-----\n');
32 [y1s, x2s, y2s, y3_1s, y3_2s] = compute_seal(xs, ys, cs,
    hyps, 0, 0.5, cntLt, sl, sw);
33
34 %% Compute on Seals: Regularization Enabled
35 fprintf('-----Seals Dataset-----\n');
```

```

36 fprintf('-----Regularization Enabled-----\n');
37 fprintf('-----\n');
38 [y1sr, x2sr, y2sr, y3_1sr, y3_2sr] = compute_seal(xs, ys,
    cs, hyps, 0.5, 0.5, cntLt, sl, sw);
39
40 %% Vary Learning Rate
41 fprintf('-----\n');
42 fprintf('----- Varying Learning Rate -----\n');
43 fprintf('-----On IRIS-----\n');
44
45 vary_lr = 0.1:0.1:0.8;
46 itr_l = zeros(size(vary_lr));
47 itr_q = itr_l; mc_l = itr_l; mc_q = itr_l;
48
49 for j = 1:length(vary_lr)
50     [~, ~, ~, ~, ~, ~, lr_c, lr_num, ~] = compute_iris(xi
    , yi, ci, xti, yti, cti, vary_lr(j), 0);
51     itr_l(j) = lr_c(1); itr_q(j) = lr_c(2);
52     mc_l(j) = lr_num(1); mc_q(j) = lr_num(2);
53 end
54
55 %% Legends for Plotting
56 ls_legend = {'ls non-reg', 'ls reg', 'class 1', 'class 2'
    };
57 pll_legend = {'percentron linear non-reg', 'percentron
    linear reg', 'class 1', 'class 2'
    };
58 plq_legend = {'percentron quad non-reg', 'percentron quad
    reg', 'class 1', 'class 2'};
59
60 %% Plot Results
61 if plot_iris == 1
62     pre = 'Iris DB: ';
63     plot_comparison(1, x1i, y1i, x1ir, y1ir, cl_x1i,
    cl_x2i, cl_y1i, cl_y2i, ls_legend, sprintf('%sLeast
    Squares Comparison',pre), 'iris_');
64     plot_comparison(2, x2i, y2i, x2ir, y2ir, cl_x1i,
    cl_x2i, cl_y1i, cl_y2i, pll_legend, sprintf('%
    sPerceptron Learning: Linear Comparison',pre), 'iris_
    ');
65     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_');
66
67     plot_all(4, x1i, y1i, x2i, y2i, sort(xi), y3_1i,
    cl_x1i, cl_x2i, cl_y1i, cl_y2i, sprintf('%
    sRegularization Disabled',pre), 'iris_');

```

```

68     plot_all(5, x1ir, y1ir, x2ir, y2ir, sort(xi), y3_1ir,
        cl_x1i, cl_x2i, cl_y1i, cl_y2i, sprintf('%
        sRegularization Enabled',pre), 'iris_');
69 end
70
71 if plot_seal == 1
72     j = 1;
73     if plot_iris == 1
74         j = 5;
75     end
76     pre = 'Seals DB: ';
77     plot_comparison(j+1, xs, y1s, xs, y1sr, cl_x1s,
        cl_x2s, cl_y1s, cl_y2s, ls_legend, sprintf('%sLeast
        Squares Comparison',pre), 'seal_');
78     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_
        ');
79     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_');
80
81     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_');
82     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_');
83 end
84
85 if plot_varylr == 1
86     k = 11;
87     figure(k);
88     plot(vary_lr, mc_l, '-* r', 'linewidth', 1); hold on;
89     plot(vary_lr, mc_q, '-o g', 'linewidth', 1); hold off
90 ;
91     legend({'Order=1', 'Order=2'});
92     title('Learning Rate vs Missclassifications');
93     xlabel('\eta');
94     ylabel('Num. of Missclassifications');
95     saveas(gcf, sprintf('vary_lr_lab30_%d',k), 'epsc');
96     saveas(gcf, sprintf('vary_lr_lab30_%d',k), 'png');
97
98     figure(k+1);
99     plot(vary_lr, itr_l, 'r', 'linewidth', 1); hold on;
100    plot(vary_lr, itr_q, 'g', 'linewidth', 1); hold off;
101    title('Learning Rate vs Iteration Count');
102    legend({'Order=1', 'Order=2'});

```

```

102     xlabel('\eta');
103     ylabel('Iterations');
104     saveas(gcf, sprintf('vary_lr_lab30_%d',k+1), 'eps');
105     saveas(gcf, sprintf('vary_lr_lab30_%d',k+1), 'png');
106 end
107
108 %% Plotting Functions
109 function plot_comparison(i, x1, y1, x1_r, y1_r, cl_x1,
    cl_x2, cl_y1, cl_y2, l, t, pre)
110     figure(i);
111     plot(x1, y1, '-- r', 'linewidth', 1); hold on;
112     plot(x1_r, y1_r, '-- g', 'linewidth', 1); hold on;
113     plot(cl_x1, cl_y1, "ko", "MarkerSize", 5, "
    MarkerFaceColor", "c"); hold on;
114     plot(cl_x2, cl_y2, "ks", "MarkerSize", 5, "
    MarkerFaceColor", "m"); hold off;
115     title(t); legend(l); xlabel("x", "Interpreter", "
    Latex"); ylabel("y", "Interpreter", "Latex");
116     saveas(gcf, sprintf('%s_lab30_%d',pre,i), 'eps');
117     saveas(gcf, sprintf('%s_lab30_%d',pre,i), 'png');
118 end
119
120 function plot_all(i, x1, y1, x2, y2, x3, y3, cl_x1, cl_x2
    , cl_y1, cl_y2, t, pre)
121     figure(i);
122
123     % decision line: least squares
124     plot(x1, y1, '-- r', 'linewidth', 1); hold on;
125
126     % decision line : perceptron "linear" learning
    algorithm
127     plot(x2, y2, '-- g', 'linewidth', 1); hold on;
128
129     % decision line : perceptron "quadratic" learning
    algorithm
130     plot(x3, y3, '-- b', 'linewidth', 1); hold on;
131
132     % data
133     plot(cl_x1, cl_y1, "ko", "MarkerSize", 5, "
    MarkerFaceColor", "c"); hold on;
134     plot(cl_x2, cl_y2, "ks", "MarkerSize", 5, "
    MarkerFaceColor", "m"); hold off;
135
136     % plot details
137     legend('least squares', 'perceptron : linear', '
    perceptron : quadratic', 'class 1', 'class 2');
138     xlabel("x", "Interpreter", "Latex");
139     ylabel(" y ", "Interpreter", "Latex");
140     title(t);

```

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

B [MATLAB Code] Balancing Principle

Filename: *"balancing_principle.m"*

```
1 function gamma_updated = balancing_principle(gamma, A, t)
2     %% Implementation
3     gamma_updated = 0;
4     C = 1;                                % zero-crossing
5     method
6     theta = 0.01;                          % tolerance
7     count = 1;
8     check_condition = 1;
9     while (check_condition ~= 0)
10        % Step 2: Compute Value Function 'J' to obtain
11        % omega 'w'
12        % J = 0.5 * ((norm( (Aw - t), 2)^2) + gamma*(
13        norm(w, 2))^2);
14        % 0.5 [ (2A')*(Aw - t) + (2*gamma*w) ] = 0
15        % (A')Aw - (A')t + gamma*w = 0
16        % ((A')A + gamma*I) w = (A')t
17        % w = inv((A')A + gamma*I) * (A')t
18        w = ( (A')*A + gamma*eye(size((A')*A)) ) \ ((A')*
19        t);
20        phi_bar = (A')*(A*w - t);
21        psi_bar = 2*w;
22
23        % Step 3: Update Regularization Parameter
24        gamma_updated = C * (norm( phi_bar )^2 / norm(
25        psi_bar )^2);
26        count = count+1;
27        if (abs(gamma_updated - gamma) > theta)
28            gamma = gamma_updated;
29        else
30            check_condition = 0;
31        end
32        if count > 100
33            break;
34        end
35    end
36    fprintf('Regularization Iteration: %d\ngamma: %.12f\n', count, gamma_updated);
37 end
```

C [MATLAB Code] Compute on Iris Dataset

Filename: "compute_iris.m"

```
1 function [x1, y1, x2, y2, y3_1, y3_2, counts,
missClassNum, missClassRate] = compute_iris(x, y,
class, xt, yt, ct, lr, gamma_0)
2 %% Function: LeastSquares Problem Solver => min ||A*
omega - t||, A =[1; x; y], t = class
3 A = [ones(numel(x), 1) x' y'];
4 if gamma_0 ~= 0
5     B = (A') * A;
6     gamma_updated = balancing_principle(gamma_0, A,
class');
7     B = B + gamma_updated*(eye(size(B)));
8     wls = B\((A')*(class'));
9 else
10    wls = A\(class');
11 end
12
13 y1 = zeros(1,2);
14 x1(1)= min(x); x1(2)=max(x);
15
16 for i=1:1:2
17     y1(i) = ((0.5-wls(1))/wls(3)) - ((wls(2)/wls(3))
*x1(i));
18 end
19
20 m = ( y1(1,2) - y1(1,1) ) / (x1(2) - x1(1));
21 pred = ct';
22
23 mx = mean(x(1,:));
24 my = mean(y(1,:));
25 b = my/mx;
26
27 for i = 1:length(xt)
28     if (yt(1,i) - ((m*xt(1,i)) + b) ) > 0 % point
is above line
29         pred(i) = 1;
30     elseif (yt(1,i) - ((m*xt(1,i)) + b) ) < 0 % point
is below line
31         pred(i) = 0;
32     end
33 end
34
35 missClassNumLS = (length(pred)-sum(pred == ct'));
36 missClassRateLS = missClassNumLS/length(pred);
37 fprintf("Number of MissClassified Points : %d\
nMissClassification Rate: %.3f\n", missClassNumLS,
```

```

missClassRateLS);
38
39 %% Function: Perceptron Learning Algorithm: d = 1
40 % initialize parameters
41 rng(1, 'philox');
42 w1 = randn(3, 1);
43 hyp = zeros(size(x,2), 1)';
44 best_w1 = w1;
45 count = 0;
46 pred = ct;
47
48 if gamma_0 ~= 0
49     gamma_updated = balancing_principle(gamma_0, [
ones(numel(x), 1) x' y'], class');
50 else
51     gamma_updated = 0;
52 end
53
54 while sum(class ~= hyp)
55     for i = 1:size(x,2)
56         if (w1(1) + w1(2)*x(i) + w1(3)*y(i)) > 0
57             hyp(i) = 1;
58         else
59             hyp(i) = 0;
60         end
61         % save current weights
62         best_w1 = w1;
63
64         % update weights and gamma
65         w1(1) = w1(1) + lr*(class(i) - hyp(i));
66         w1(2) = w1(2) + lr*((class(i) - hyp(i)) * x
(i)) + (gamma_updated*best_w1(2)) );
67         w1(3) = w1(3) + lr*((class(i) - hyp(i)) * y
(i)) + (gamma_updated*best_w1(2)) );
68     end
69
70     count = count+1;
71     if count > 1e10
72         break
73     end
74 end
75
76 y2 = zeros(1,2);
77 x2(1)= min(x);    x2(2)=max(x);
78
79 for i=1:1:2
80     y2(i) = (-best_w1(1)/best_w1(3)) - ((best_w1(2)/
best_w1(3))*x2(i));
81 end

```



```

82
83     counts(1) = count-1;
84     fprintf('Iterations: %d\n', count-1);
85
86     for i=1:length(xt)
87         Arow = [1 xt(i) yt(i)];
88         if (Arow*best_wl) > 0
89             pred(1,i) = 1;
90         else
91             pred(1,i) = 0;
92         end
93     end
94
95     missClassNumL = (length(pred)-sum(pred == ct));
96     missClassRateL = missClassNumL/length(pred);
97
98     fprintf("Number of MissClassified Points : %d\
nMissClassification Rate: %.3f\n", missClassNumL,
missClassRateL);
99
100    %% Function: Perceptron Learning Algorithm: d = 2
101    % initialize parameters
102    rng(1, 'philox');
103    wq = randn(6, 1);
104    hyp = zeros(size(x,2), 1)';
105    best_wq = wq;
106    count = 0;
107    pred = ct;
108    z1 = pred;
109    z2 = z1;
110
111    if gamma_0 ~= 0
112        gamma_updated = balancing_principle(gamma_0, [
ones(numel(x), 1) x' y'], class');
113    else
114        gamma_updated = 0;
115    end
116
117    while sum(class ~= hyp)
118        for i=1:size(x,2)
119            if wq(1) + wq(2)*x(i) + wq(3)*y(i) + ...
120                wq(4)*x(i)*x(i) + wq(5)*x(i)*y(i) + ...
121                wq(6)*y(i)*y(i) > 0
122                hyp(i)=1;
123            else
124                hyp(i)=0;
125            end
126            % save current weights
127            best_wq = wq;

```

```

128         % update weights
129         wq(1) = wq(1) + lr*(class(i) - hyp(i));
130         wq(2) = wq(2) + lr*((class(i) - hyp(i)) *x(
131 i)) + (gamma_updated*best_wq(2)) );
132         wq(3) = wq(3) + lr*((class(i) - hyp(i)) *y(
133 i)) + (gamma_updated*best_wq(3)) );
134         wq(4) = wq(4) + lr*((class(i) - hyp(i)) *x(
135 i)*x(i)) + (gamma_updated*best_wq(4)) );
136         wq(5) = wq(5) + lr*((class(i) - hyp(i)) *x(
137 i)*y(i)) + (gamma_updated*best_wq(5)) );
138         wq(6) = wq(6) + lr*((class(i) - hyp(i)) *y(
139 i)*y(i)) + (gamma_updated*best_wq(6)) );
140     end
141
142     count = count+1;
143     if count > 1e10
144         break
145     end
146 end
147
148 q_a = best_wq(6);
149 x = sort(x);
150
151 y3_1 = zeros(1, size(x,2));
152 y3_2 = zeros(1, size(x,2));
153
154 for i = 1:size(x,2)
155     q_b = best_wq(5)*x(i) + best_wq(3);
156     q_c = best_wq(1) + best_wq(2)*x(i) + best_wq(4)*x
157 (i)*x(i);
158     D = q_b*q_b - 4*q_a*q_c;
159     y3_1(i) = (-q_b + sqrt(D))/(2*q_a);
160     y3_2(i) = (-q_b - sqrt(D))/(2*q_a);
161 end
162
163 counts(2) = count-1;
164 fprintf('Iterations: %d\n', count-1);
165
166 qa = best_wq(6);
167 for i=1:length(xt)
168     qb = best_wq(5)*xt(i) + best_wq(3);
169     qc = best_wq(1) + best_wq(2)*xt(i) + best_wq(4)*
170 xt(i)*xt(i);
171     D = qb*q_b - 4*q_a*q_c;
172     z1(i) = (-qb + sqrt(D))/(2*qa);
173     z2(i) = (-qb - sqrt(D))/(2*qa);
174     if (yt(i) > z1(i))
175         pred(i) = 1;
176     else

```

```

170         pred(i) = 0;
171     end
172 end
173
174     missClassNumQ = (length(pred)-sum(pred == ct));
175     missClassRateQ = missClassNumQ/length(pred);
176
177     fprintf("Number of MissClassified Points : %d\
nMissClassification Rate: %.3f\n", missClassNumQ,
missClassRateQ);
178
179     missClassNum = [missClassNumLS, missClassNumL,
missClassNumQ];
180     missClassRate = [missClassRateLS, missClassRateL,
missClassRateQ];
181 end

```

D [MATLAB Code] Compute on Seals Dataset

Filename: "compute_seal.m"

```
1 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)  
2 % Generate Vandemorte Matrix  
3 fprintf('1. Generating Vandemorte Matrix\n');  
4 m = size(Seal_w, 1);  
5 d = 1;  
6 A = [];  
7 for i=1:1:m  
8     for j=1:1:d+1  
9         A(i,j)=power(x(i),j-1);  
10        if isnan(Seal_l(i))  
11            y(i)= 0;  
12        end  
13    end  
14 end  
15  
16 sch = 0;  
17  
18 for i=m+1:1:2*m  
19     sch = sch+1;  
20     for j=1:1:d+1  
21         A(i,j)=power(x(i),j-1);  
22         if isnan(Seal_w(sch))  
23             y(i)= 0;  
24         end  
25     end  
26 end  
27  
28 %  
29 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
30 %  
31 %  
32 %  
33 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
34 %  
35 %  
36 %  
37 %  
38 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
21
```

```

39     w = B\((A')*(y'));
40 else
41     w = A\((y'));
42 end
43
44 ls_y = A*w;
45
46 %
47 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
48 %%%
49 %%%                                PERCEPTRON LINEAR ALGO
50 %%%
51 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
52
53 fprintf('3. Computing Solution by Perceptron "d = 1"
54 Learning Algorithm\n');
55 w1 = randn(3, 1);
56 best_w1 = w1;
57 count = 0;
58 hyp1 = hyp;
59
60 if gamma_0 ~= 0
61     gamma_updated = balancing_principle(gamma_0, A, y
62 ');
63 else
64     gamma_updated = 0;
65 end
66
67 while sum(class ~= hyp1)
68     for i = 1:size(x,2)
69         if (w1(1) + w1(2)*x(i) + w1(3)*y(i)) > 0
70             hyp1(i) = 1;
71         else
72             hyp1(i) = 0;
73         end
74         % save current weights
75         best_w1 = w1;
76
77         % update weights and gamma
78         w1(1) = w1(1) + lr*(class(i) - hyp1(i));
79         w1(2) = w1(2) + lr*((class(i) - hyp1(i)) *
80 x(i)) + (gamma_updated*best_w1(2));
81         w1(3) = w1(3) + lr*((class(i) - hyp1(i)) *
82 y(i)) + (gamma_updated*best_w1(2));
83     end
84 end

```

```

80         count = count+1;
81         if count > CountLimit
82             break
83         end
84     end
85
86     pll_y = zeros(1,2);
87     pll_x(1)= min(x);    pll_x(2)=max(x);
88
89     for i=1:1:2
90         pll_y(i) = (-best_wl(1)/best_wl(3)) - ((best_wl
91 (2)/best_wl(3))*pll_x(i));
92     end
93
94     fprintf('Iterations: %d\n', count-1);
95
96     %
97     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
98
99     %%%
100                                PERCEPTRON QUAD ALGO
101     %%%
102     %
103     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104
105     fprintf('4. Computing Solution by Perceptron "d = 2"
106 Learning Algorithm\n');
107     wq = randn(6, 1);
108     hyp2 = hyp;
109     best_wq = wq;
110     count = 0;
111
112     if gamma_0 ~= 0
113         gamma_updated = balancing_principle(gamma_0, A, y
114 ');
115     else
116         gamma_updated = 0;
117     end
118
119     while sum(class ~= hyp2)
120         for i=1:size(x,2)
121             if wq(1) + wq(2)*x(i) + wq(3)*y(i) + ...
122                 wq(4)*x(i)*x(i) + wq(5)*x(i)*y(i) + ...
123                 wq(6)*y(i)*y(i) > 0
124                 hyp2(i)=1;
125             else
126                 hyp2(i)=0;
127             end
128         end
129     end

```

```

122         % save current weights
123         best_wq = wq;
124         % update weights
125         wq(1) = wq(1) + lr*(class(i) - hyp2(i));
126         wq(2) = wq(2) + lr*( ((class(i) - hyp2(i)) *x
(i)) + (gamma_updated*best_wq(2)) );
127         wq(3) = wq(3) + lr*( ((class(i) - hyp2(i)) *y
(i)) + (gamma_updated*best_wq(3)) );
128         wq(4) = wq(4) + lr*( ((class(i) - hyp2(i)) *x
(i)*x(i)) + (gamma_updated*best_wq(4)) );
129         wq(5) = wq(5) + lr*( ((class(i) - hyp2(i)) *x
(i)*y(i)) + (gamma_updated*best_wq(5)) );
130         wq(6) = wq(6) + lr*( ((class(i) - hyp2(i)) *y
(i)*y(i)) + (gamma_updated*best_wq(6)) );
131     end
132
133     count = count+1;
134     if count > CountLimit
135         break
136     end
137 end
138
139 q_a = best_wq(6);
140 x = sort(x);
141
142 plq_y1 = zeros(1, size(x,2));
143 plq_y2 = zeros(1, size(x,2));
144
145 for i = 1:size(x,2)
146     q_b = best_wq(5)*x(i) + best_wq(3);
147     q_c = best_wq(1) + best_wq(2)*x(i) + best_wq(4)*x
(i)*x(i);
148     D = q_b*q_b - 4*q_a*q_c;
149     plq_y1(i) = (-q_b + sqrt(D))/(2*q_a);
150     plq_y2(i) = (-q_b - sqrt(D))/(2*q_a);
151 end
152 fprintf('Iterations: %d\n', count-1);
153 end

```

E [MATLAB Code] Load Iris Dataset

Filename: *"load_iris.m"*

```
1 function [x, y, class, xt, yt, ct, cl_x1, cl_y1, cl_x2,  
   cl_y2] = load_iris  
2     data = csvread("iris.csv");  
3  
4     Xiris = data(:, 1)'; Yiris = data(:, 2)'; Ciris =  
   data(:, 3)';  
5     x = Xiris(1, :); y = Yiris(1, :); class = Ciris(1, :)  
   ;  
6  
7     sortIdx = randperm(length(x));  
8     x = x(sortIdx);  
9     y = y(sortIdx);  
10    class = class(sortIdx);  
11  
12    testSizeIndx = 70;  
13    xt = x(1,testSizeIndx+1:100); yt = y(1,testSizeIndx  
+1:100); ct = class(1,testSizeIndx+1:100);  
14    x = x(1,1:testSizeIndx); y = y(1,1:testSizeIndx);  
   class = class(1,1:testSizeIndx);  
15  
16    cl_x1 = x(class==1); cl_y1 = y(class==1);  
17    cl_x2 = x(class==0); cl_y2 = y(class==0);  
18  
19    clt_x1 = xt(ct==1); clt_y1 = yt(ct==1);  
20    clt_x2 = xt(ct==0); clt_y2 = yt(ct==0);  
21 end
```


F [MATLAB Code] Load Seals Dataset

Filename: *"load_seals.m"*

```
1 function [x, y, class, hyp, cl_x1, cl_y1, cl_x2, cl_y2,  
   seal_length, seal_weight] = load_seals  
2     seal_table = readtable('DatabaseGreySeal.xlsx');  
3     seal_year = seal_table.(3);  
4     seal_weight = seal_table.(10);  
5     seal_length = str2double(seal_table.(11));  
6  
7     m = size(seal_weight,1);  
8  
9     x=zeros(1,2*m);  
10    y=zeros(1,2*m);  
11  
12    sch1=0;  sch2=0;  
13  
14    class = zeros(1,m);  
15    hyp = zeros(1,m);  
16  
17    for i=1:1:m  
18  
19        sch1 = sch1 +1;  
20        cl_y1(sch1)= seal_length(i);  
21        cl_x1(sch1) = seal_year(i);  
22        x(i) = seal_year(i);  
23  
24        sch2 = sch2 +1;  
25        cl_y2(sch2)= seal_weight(i);  
26        cl_x2(sch2) = seal_year(i);  
27        y(i) = seal_length(i);  
28  
29        class(sch1) = 1;  
30        hyp(sch1)= 0;  
31    end  
32  
33    i=0; sch3=0;  
34  
35    for i=m+1:1:2*m  
36        sch3 = sch3 +1;  
37        y(i) = seal_weight(sch3);  
38        x(i) = seal_year(sch3);  
39  
40        class(i) = 0;  
41        hyp(i)= 1;  
42    end  
43  
44    %     testSize = 0;
```

```
45 %     testSizeIndx = round(length(x)*(1-testSize));
46 %     xt = x(1,testSizeIndx+1:end); yt = y(1,testSizeIndx
    +1:end); ct = class(1,testSizeIndx+1:end);
47 %     x = x(1,1:testSizeIndx); y = y(1,1:testSizeIndx);
    class = class(1,1:testSizeIndx);
48 end
```

G [MATLAB Console Output]

Filename: "OUTPUTLOG.m"

```
1 Warning: Table variable names were modified to make them
   valid MATLAB identifiers. The original names are saved
   in the
2 VariableDescriptions property.
3 -----IRIS Dataset-----
4 -----Regularization Disabled-----
5 -----
6 Number of MissClassified Points : 0
7 MissClassification Rate: 0.000
8 Iterations: 891
9 Number of MissClassified Points : 0
10 MissClassification Rate: 0.000
11 Iterations: 508
12 Number of MissClassified Points : 0
13 MissClassification Rate: 0.000
14 -----IRIS Dataset-----
15 -----Regularization Enabled-----
16 -----
17 Regularization Iteration: 4
18 gamma: 0.000000238419
19 Number of MissClassified Points : 0
20 MissClassification Rate: 0.000
21 Regularization Iteration: 4
22 gamma: 0.000000238419
23 Iterations: 687
24 Number of MissClassified Points : 0
25 MissClassification Rate: 0.000
26 Regularization Iteration: 4
27 gamma: 0.000000238419
28 Iterations: 511
29 Number of MissClassified Points : 0
30 MissClassification Rate: 0.000
31 -----Seals Dataset-----
32 -----Regularization Disabled-----
33 -----
34 1. Generating Vandemorte Matrix
35 2. Computing Least Squares
36 3. Computing Solution by Perceptron "d = 1" Learning
   Algorithm
37 Iterations: 1000000
38 4. Computing Solution by Perceptron "d = 2" Learning
   Algorithm
39 Iterations: 1000000
40 -----Seals Dataset-----
41 -----Regularization Enabled-----
```

```

42 -----
43 1. Generating Vandemorte Matrix
44 2. Computing Least Squares
45 Regularization Iteration: 4
46 gamma: 0.000000238419
47 3. Computing Solution by Perceptron "d = 1" Learning
    Algorithm
48 Regularization Iteration: 4
49 gamma: 0.000000238419
50 Iterations: 1000000
51 4. Computing Solution by Perceptron "d = 2" Learning
    Algorithm
52 Regularization Iteration: 4
53 gamma: 0.000000238419
54 Iterations: 1000000
55 -----
56 ----- Varying Learning Rate -----
57 -----On IRIS-----
58 Number of MissClassified Points : 0
59 MissClassification Rate: 0.000
60 Iterations: 847
61 Number of MissClassified Points : 0
62 MissClassification Rate: 0.000
63 Iterations: 499
64 Number of MissClassified Points : 0
65 MissClassification Rate: 0.000
66 Number of MissClassified Points : 0
67 MissClassification Rate: 0.000
68 Iterations: 888
69 Number of MissClassified Points : 0
70 MissClassification Rate: 0.000
71 Iterations: 506
72 Number of MissClassified Points : 0
73 MissClassification Rate: 0.000
74 Number of MissClassified Points : 0
75 MissClassification Rate: 0.000
76 Iterations: 822
77 Number of MissClassified Points : 0
78 MissClassification Rate: 0.000
79 Iterations: 517
80 Number of MissClassified Points : 0
81 MissClassification Rate: 0.000
82 Number of MissClassified Points : 0
83 MissClassification Rate: 0.000
84 Iterations: 854
85 Number of MissClassified Points : 0
86 MissClassification Rate: 0.000
87 Iterations: 513
88 Number of MissClassified Points : 0

```

```

89 MissClassification Rate: 0.000
90 Number of MissClassified Points : 0
91 MissClassification Rate: 0.000
92 Iterations: 891
93 Number of MissClassified Points : 0
94 MissClassification Rate: 0.000
95 Iterations: 508
96 Number of MissClassified Points : 0
97 MissClassification Rate: 0.000
98 Number of MissClassified Points : 0
99 MissClassification Rate: 0.000
100 Iterations: 818
101 Number of MissClassified Points : 0
102 MissClassification Rate: 0.000
103 Iterations: 512
104 Number of MissClassified Points : 0
105 MissClassification Rate: 0.000
106 Number of MissClassified Points : 0
107 MissClassification Rate: 0.000
108 Iterations: 840
109 Number of MissClassified Points : 0
110 MissClassification Rate: 0.000
111 Iterations: 505
112 Number of MissClassified Points : 0
113 MissClassification Rate: 0.000
114 Number of MissClassified Points : 0
115 MissClassification Rate: 0.000
116 Iterations: 821
117 Number of MissClassified Points : 0
118 MissClassification Rate: 0.000
119 Iterations: 514
120 Number of MissClassified Points : 0
121 MissClassification Rate: 0.000
122 Warning: Imaginary parts of complex X and/or Y arguments
    ignored
123 > In lab3_final>plot_comparison (line 111)
124   In lab3_final (line 79)
125 Warning: Imaginary parts of complex X and/or Y arguments
    ignored
126 > In lab3_final>plot_all (line 130)
127   In lab3_final (line 81)
128 >>

```