EECS 4404

Assignment 2

Name: Bochao Wang

Student ID: 215237902

Prism: bochao

Date: March. 14th

1. Gradient computation

Let $X \in \mathbb{R}^{d*d}$ be some matrix. Consider the function $f:\ \mathbb{R}^d \to \mathbb{R}$ defined by $f(w) = w^T X w$, show that the gradient of this function with repect to w is $\bigtriangledown f(w) = w^T(X + X^T)$

- Proof:
    - $\bigtriangledown f(w) = \begin{pmatrix} \frac{\partial f}{\partial w_1}, & \frac{\partial f}{\partial w_2}, & \dots & \frac{\partial f}{\partial w_d} \end{pmatrix}$
    - $f(w) = \begin{bmatrix} w_1 & w_2 & w_3 & \dots & w_d \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1d} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2d} \\ x_{31} & x_{32} & x_{33} & \dots & x_{3d} \\ \vdots & & & & \vdots \\ x_{d1} & x_{d2} & x_{d3} & \dots & x_{dd} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_d \end{bmatrix}$
    - $f(w) = \begin{bmatrix} \sum_{i=1}^d w_i x_{i1}, & \sum_{i=1}^d w_i x_{i2}, & \dots & \sum_{i=1}^d w_i x_{id} \end{bmatrix} w = \sum_{j=1}^d \sum_{i=1}^d w_i x_{ij} w_j$
    - Simplify the function $f(w) = \sum_{j=1}^d (x_{jj} w_j^2 + \sum_{i \neq j} w_i x_{ij} w_j)$
    - we can compute the partial derivative with element k, $\frac{\partial}{\partial w_k}[\sum_{j=1}^d (x_{jj} w_j^2 + \sum_{i \neq j} w_i x_{ij} w_j)] = \sum_{j=1}^d (\frac{\partial}{\partial w_k} x_{jj} w_j^2 + \sum_{i \neq j} \frac{\partial}{\partial w_k} w_i x_{ij} w_j) = 2x_{kk} w_{kk} + \sum_{i \neq k} x_{ki} w_i + \sum_{i \neq k} w_i x_{ik}$
    - $\because x_{kk} w_{kk} = x_{kk} w_{kk}$ Then, $\frac{\partial}{\partial w_k}[\sum_{j=1}^d (x_{jj} w_j^2 + \sum_{i \neq j} w_i x_{ij} w_j)] = \sum_{i=1}^d x_{ki} w_i + \sum_{i=1}^d w_i x_{ik}$
    - $\bigtriangledown f(w) = \begin{bmatrix} \sum_{i=1}^d x_{1i} w_i + \sum_{i=1}^d w_i x_{i1}, & \sum_{i=1}^d x_{2i} w_i + \sum_{i=1}^d w_i x_{i2}, & \dots & \sum_{i=1}^d x_{di} w_i + \sum_{i=1}^d w_i x_{id} \end{bmatrix}$
    - $\bigtriangledown f(w) = \begin{bmatrix} \sum_{i=1}^d w_i x_{i1}, & \sum_{i=1}^d w_i x_{i2}, & \dots & \sum_{i=1}^d w_i x_{id} \end{bmatrix} + \begin{bmatrix} \sum_{i=1}^d x_{1i} w_i, & \sum_{i=1}^d x_{2i} w_i, & \dots & \sum_{i=1}^d x_{di} w_i \end{bmatrix}$
    - $\begin{bmatrix} \sum_{i=1}^d w_i x_{i1}, & \sum_{i=1}^d w_i x_{i2}, & \dots & \sum_{i=1}^d w_i x_{id} \end{bmatrix} = w^T X$
    - $\begin{bmatrix} \sum_{i=1}^d x_{1i} w_i, & \sum_{i=1}^d x_{2i} w_i, & \dots & \sum_{i=1}^d x_{di} w_i \end{bmatrix} = w^T X^T$
    - Thus, $\bigtriangledown f(w) = w^T X + w^T X^T = w^T(X + X^T)$

2. Stochastic Gradient Descent

Recall the logistic loos function, point wise defined as

$l^{logist}(y_{\boldsymbol{w}}, (\boldsymbol{x}, t)) = ln(1 + exp(-t < \boldsymbol{w}, \boldsymbol{x} >))$

We know that empirical logistic loss over a dataset

$\mathcal{L}^{logist}(y_{\boldsymbol{w}}) = \frac{1}{N} \sum_{n=1}^N ln(1 + exp(-t_n < \boldsymbol{w}, \boldsymbol{x}_n >))$

(a) Compute the gradient with respect to w of the logistic loss $l^{logist}(y_{\boldsymbol{w}}, (\boldsymbol{x}, t))$ on a data point (x, t).

- Solve:
    - $l^{logist}(y_{\boldsymbol{w}}, (\boldsymbol{x}, t)) = ln(1 + exp(-t * \sum_i^N w_i x_i))$
    - $\bigtriangledown f(\boldsymbol{w}) = \begin{pmatrix} \frac{\partial f}{\partial w_1}, & \frac{\partial f}{\partial w_2}, & \dots & \frac{\partial f}{\partial w_d} \end{pmatrix}$
    - $\bigtriangledown l^{logist}(y_{\boldsymbol{w}}, (\boldsymbol{x}, t)) = \Bigg(\frac{1}{1+exp(-t*\sum_i^N w_i x_i)} \frac{\partial(1+exp(-t*\sum_i^N w_i x_i))}{\partial w_1}, \quad \frac{1}{1+exp(-t*\sum_i^N w_i x_i)} \frac{\partial(1+exp(-t*\sum_i^N w_i x_i))}{\partial w_2}, \quad \dots, \quad \frac{1}{1+exp(-t*\sum_i^N w_i x_i)} \frac{\partial(1+exp(-t}{\partial u}$
    - we compute the k element:
    - $\frac{\partial(1+exp(-t\sum_i^N w_i x_i))}{\partial w_k} = e^{-t\sum_i^N w_i x_i} \frac{\partial(-t\sum_i^N w_i x_i)}{\partial w_k} = e^{-t\sum_i^N w_i x_i}(-t x_k) = e^{-t<\boldsymbol{w},\boldsymbol{x}>}(-t x_k)$
    - $\bigtriangledown l^{logist}(y_{\boldsymbol{w}}, (\boldsymbol{x}, t)) = \Bigg(\frac{e^{-t<\boldsymbol{w},\boldsymbol{x}>}}{1+exp(-t*\sum_i^N w_i x_i)}(-t x_1), \quad \frac{e^{-t<\boldsymbol{w},\boldsymbol{x}>}}{1+exp(-t*\sum_i^N w_i x_i)}(-t x_2), \quad \dots, \quad \frac{e^{-t<\boldsymbol{w},\boldsymbol{x}>}}{1+exp(-t*\sum_i^N w_i x_i)}(-t x_d) \Bigg)$
    - $\bigtriangledown l^{logist}(y_{\boldsymbol{w}}, (\boldsymbol{x}, t)) = \Bigg(\frac{e^{-t<\boldsymbol{w},\boldsymbol{x}>}}{1+exp(-t<\boldsymbol{w},\boldsymbol{x}>)}(-t x_1), \quad \frac{e^{-t<\boldsymbol{w},\boldsymbol{x}>}}{1+exp(-t<\boldsymbol{w},\boldsymbol{x}>)}(-t x_2), \quad \dots, \quad \frac{e^{-t<\boldsymbol{w},\boldsymbol{x}>}}{1+exp(-t<\boldsymbol{w},\boldsymbol{x}>)}(-t x_d) \Bigg)$
    - $\bigtriangledown l^{logist}(y_{\boldsymbol{w}}, (\boldsymbol{x}, t)) = \frac{e^{-t<\boldsymbol{w},\boldsymbol{x}>}}{1+exp(-t<\boldsymbol{w},\boldsymbol{x}>)}(-t\boldsymbol{x})$

(b) Describe Stochastic Gradient Descent with a fixed stepsize η with respect to the logistic loss

- Solve:
    1. $In : data\ D = ((\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \dots, (\boldsymbol{x}_N, t_N))$
    2. $Parameters : \eta,\ T,\ n$
    3. $Initialize : \boldsymbol{w}_0 = 0$
    4. $For\ i = 0, \dots, T$
    5. Choose a random data point $(\boldsymbol{x}^*, t^*)$ form D
    6. so that $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta * \frac{e^{-t^*<\boldsymbol{w_i},\boldsymbol{x}^*>}}{1+exp(-t^*<\boldsymbol{w_i},\boldsymbol{x}^*>)}(-t^*\boldsymbol{x}^*)$
    7. $Output : \boldsymbol{w} = \frac{1}{n}\sum_{k=T-n}^T \boldsymbol{w}^k$
    - T is the number of updates
    - η is fixed stepsize

    - n compute average of last n iteration

(c) Compare the derived algorithm to the SGD algorithm we derived for SVM. What is similar? What is different? On what type of data would you expect to see different behavior?

- Solve:
    - Similar :
        - The part (b) algrothim and we derived for SVM all use the gradient descent method to update and find the optimal $\boldsymbol{w}$ so that minimize the loss.
        - Both algorithms would sample a data point $(\boldsymbol{w}_i, t_i)$ uniformly at the random from dataset D to compute the gradient to update
        - Both of them are convex optimal problems
        - Both algorithms compute the average $\boldsymbol{w}$ of last n iterations.
        - Both of them would be initialized $\boldsymbol{w}_0 = 0$
    - Different:
        - part (b) use fixed stepsize η, SVM we derived use stepsize sequence $\eta * j = \frac{1}{\lambda * j}$ where j is current step number. It means in SVM, the stepsize would less and less in each step, it would more accuracy to be closed minimum.
        - Part(b) consider the gradient of logistic loss, because logistic loss would be differenviable in every point, the gradient and subgtadient would be same. However, SVM consider the subgradient of hinge loss, because the point at $t_i < \boldsymbol{w}, x_i >$ would not be differentiable. The logistic loss would close to 0, but never be 0; the hinge loss would be 0 if $t_i < \boldsymbol{w}, x_i > \ > 1$, it means that the suport vectors would determine the $\boldsymbol{w}$, but Part(b) algrothm every points, which classify correctlly, also update the $\boldsymbol{w}$.
        - soft SVM allows sparse support vectors. It means soft SVM would tolerant the missclassified point in the margin.
        - The gradient of hinge loss function is a piecewise-defined function, but the gradient of logistic loss is comtinuous.
    - The subgradient of SVM is 0, if $t_i < \boldsymbol{w}, x_i > \ > 1$, it means if the point is far from the margin and classified correctly, these points woud not update the $\boldsymbol{w}$, but the SGD of logistic loss would always update the $\boldsymbol{w}$. Thus, if the dataset contains most points where are in the $t_i < \boldsymbol{w}, x_i > \ > 1$ i.e. almost point would far from the margin and classify correctly. In such dataset, SVM would update very few steps, but SGD of logistics loss would always update.

3. SoftSVM optimization

(a) Implement SGD for SoftSVM.

`soft_svm.m`

```
function [w, hi_loss, bi_loss] = soft_svm(D,T,lambda, n)
% if n=0, only print last step w, otherwise print average of
% last n iteration

% Initialize
[N, clo] = size(D);
x = D(:,1:clo-1); % features
t = D(:,clo); % label {-,+}
theta=zeros(1,clo-1);

hi_loss = zeros(T,1);
bi_loss = zeros(T,1);

% store `w` in each iteration
ws=zeros(T,clo-1);

w_j = (1/(lambda))*theta; % w_0

for j = 1:T
    % Choose the i in uniform distribution
    i = unidrnd(N);

    if t(i) * dot(w_j, x(i,:)) < 1

        theta = theta + t(i)*(x(i,:));

    end

    % update `w`
    w_j = (1/(j*lambda))*theta;

    % store the `w` in `ws`
     ws(j,:) = w_j;

    % Track the emprical and hinge loss
    hi_loss(j,:) = emp_loss(w_j, D, 'hinge');
    bi_loss(j,:) = emp_loss(w_j, D, 'binary');
end

if(n ~= 0)
    w = (1/n) * sum(ws((T-n:T),:));
else
    w = w_j;
end
```

`emp_loss.m`

```
function emr_loss = emp_loss(w, D, flag)
% L_D(w)=(1/N)sum(loss(w,(x_i, t_i)))
[N,clo] = size(D);
l=0;
for i = 1:N
    x_i = D(i,1:clo-1);
    t_i = D(i,clo);
    if(strcmpi(flag, 'hinge'))
        l = l + hinge_loss(w, x_i, t_i);
    elseif(strcmpi(flag, 'binary'))
        l = l + binary_loss(w, x_i, t_i);
    end
end
emr_loss = (1/N) * l;
```

`hinge_loss.m`

```
function h_loss = hinge_loss(w, x_i, t_i)
% hinge_loss(w,(x,t)) = max{0, 1-t<w,x>}
if t_i * dot(w,x_i) > 1
    h_loss = 0;
else
    h_loss = 1 - t_i * dot(w,x_i);
end
```

`binary_loss.m`

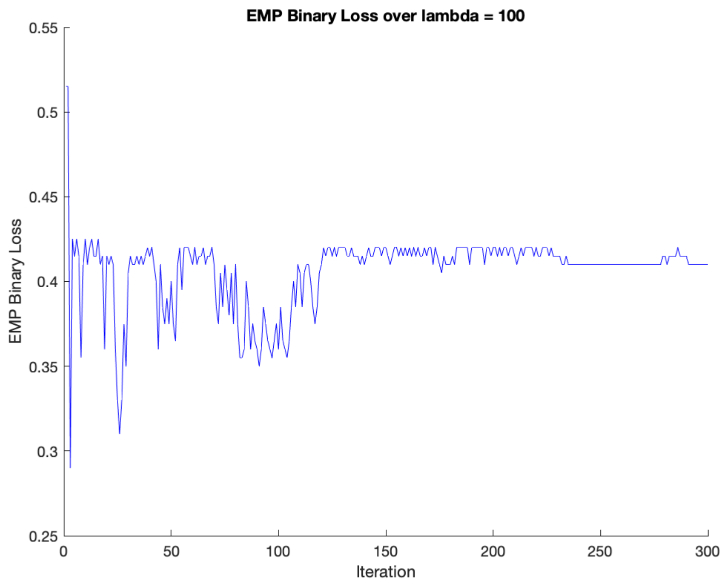
```
function b_loss = binary_loss(w, x_i, t_i)
% binary_loss(w,(x,t)) = 0[y(x) = t]
if t_i * dot(w,x_i) >= 0
    b_loss = 0;
else
    b_loss = 1;
end
```

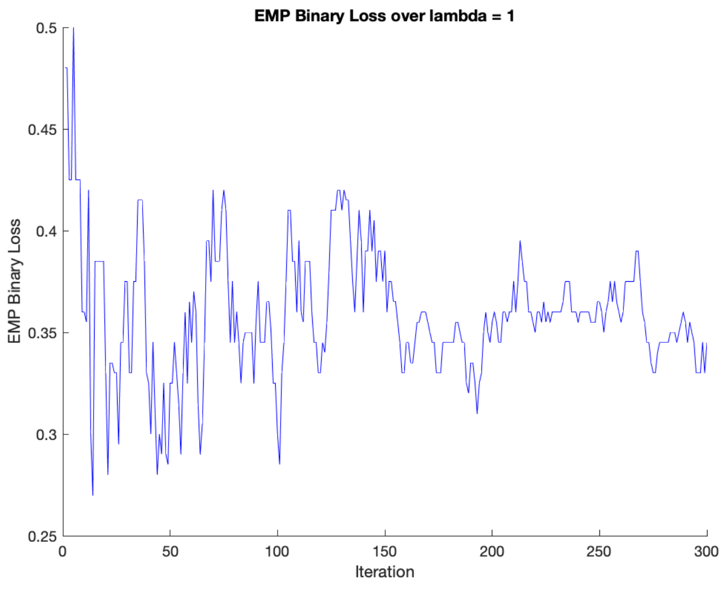
(b) Run the optimization method with various values for the regularization parameter $\lambda = \{100, 10, 1, .1, .01, .001\}$ on the data.

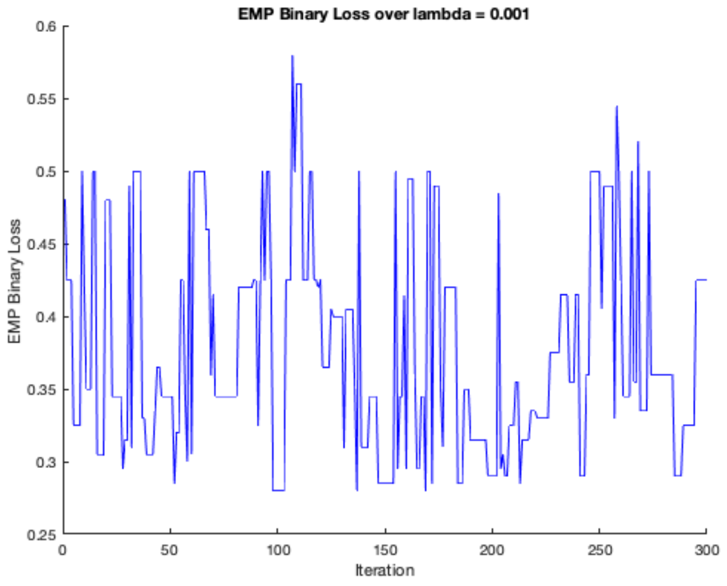
`main.m`

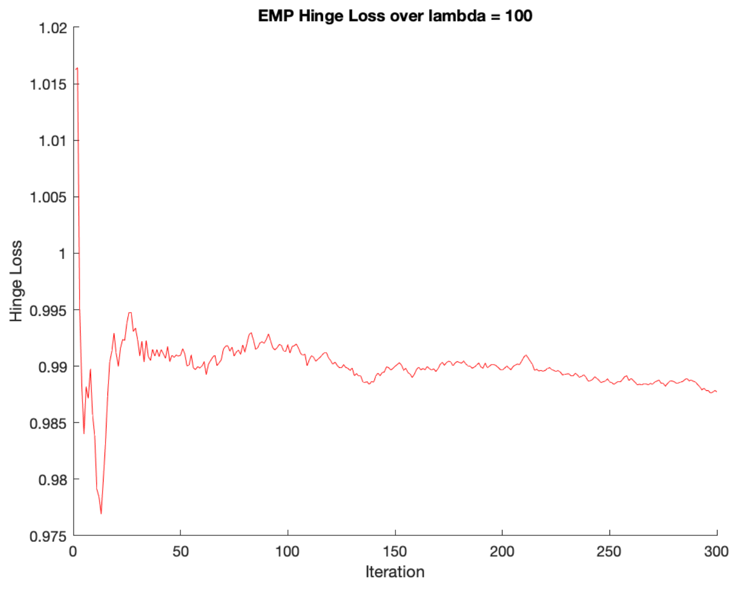
```
% PART (b) Run with lambda = {100, 10, 1, .1, .01, .001} on the data
% and print empiral hinge loss and empiral binary loss
clc;clear;close all;
dataset = load_data("bg.txt");

% add bias
[N,C] = size(dataset);
bias = ones(N,1);
```

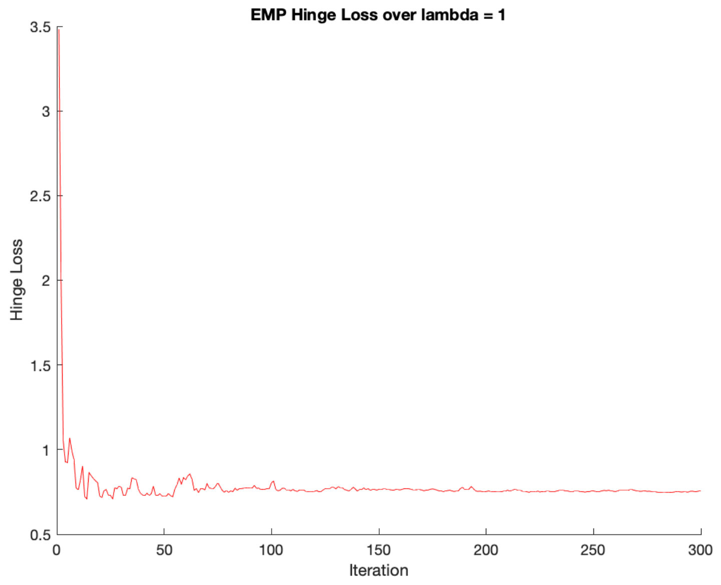
```
dataset = [dataset(:,1:C-1) bias dataset(:,C)];

num_updates = 300;
s_lambda = [100, 10, 1, .1, .01, .001];
n = 1;

for i = 1:length(s_lambda)
    [w,hi_loss,bi_loss] = soft_svm(dataset, num_updates, s_lambda(i), n);

    figure
    title(['EMP Hinge Loss over lambda = ', num2str(s_lambda(i))])
    hold on
%     set(gca, 'YScale', 'log'); % set y-axis as log
    plot(hi_loss, 'Color', 'red');
    xlabel('Iteration');
    ylabel('Hinge Loss');
    hold off

    figure
    title(['EMP Binary Loss over lambda = ', num2str(s_lambda(i))])
    hold on
%     set(gca, 'YScale', 'log'); % set y-axis as log
    plot(bi_loss, 'Color', 'blue');
    xlabel('Iteration');
    ylabel('EMP Binary Loss');
    hold off
end

fprintf("part(b): Finished! Please press enter to continue!\n")
pause;
clc;close all;
```

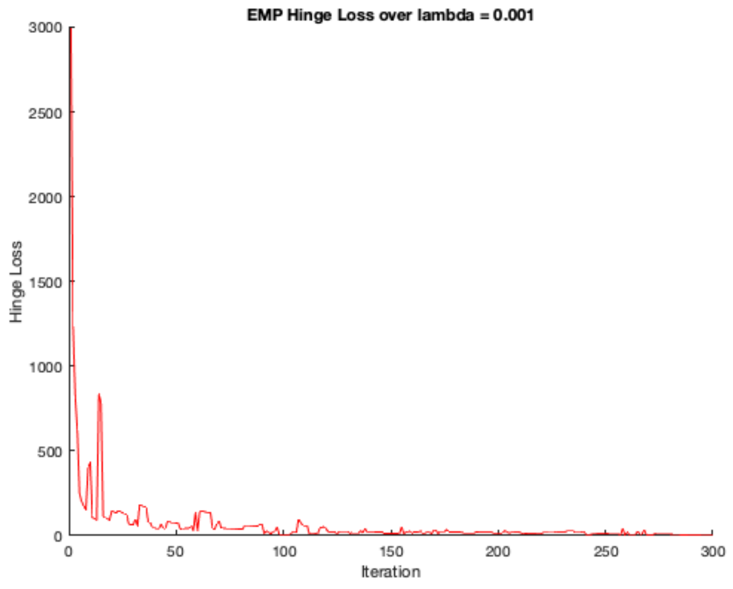



EMP Binary Loss over lambda = 1
EMP Binary Loss
Iteration
0.25
0.3
0.35
0.4
0.45
0.5
0
50
100
150
200
250
300


EMP Binary Loss over lambda = 0.001
EMP Binary Loss
Iteration
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0
50
100
150
200
250
300


EMP Hinge Loss over lambda = 100
Hinge Loss
Iteration
1.02
1.015
1.01
1.005
1
0.995
0.99
0.985
0.98
0.975
0
50
100
150
200
250
300


EMP Hinge Loss over lambda = 1
Hinge Loss
Iteration
3.5
3
2.5
2
1.5
1
0.5
0
50
100
150
200
250
300



(c) Discuss the plots. Are the curves monotone? Are they approximately monotone? Why or why not? How does the choice of λ affect the optimization? How would you go about finding a linear predictor of minimal binary loss?

- Solve:
  - No, the curves are not strictly monotone. Because in each iteration, the SGD for SoftSVM pick random data point to compute the subgradient, so the loss may not decreasing. For example, sometimes, it may pick a noice data point, the loss would larger than previous iteration; sometimes, it may pick a same point as previous iteration, the loss would very small. Thus, it not monotone.
  - Yes, they are approximately monotone. The trend of loss would montonely decreasing. Because gradient method would always give the direction which decreasing fast. The loss would decrease globally.
  - Because stepsize sequence $\eta^j = \frac{1}{\lambda * j}$ and the λ is on the denominator, when λ is larger, the stepsize would be small, and multiply the iteration number, the stepsize would smaller and smaller, it would close to the minimum slower, and it may need more iterations to close the minimum points . However, if λ is very small, stepsize would very big initially, so it would overstep, it may just go across the optimal solution and increase the loss. Also, if λ is very larger, the function would be simpler.
  - Because minimize the binary loss computation would very hard, we use its upper bounded loss i.e. hinge loss to substitute. To find a linear predictor of minimal binary loss, we use soft SVM and track the binary loss. we need try and modify the several different λ and T the minimum loss and obtain the weight in the SVM would also for linear predictor of minimal binary loss.

(e-f) Multi-class predictor

`main.m`

```
% PART (d-e) Spilt the data set and train three binary linear predictors
clc;clear;close all;

% load the dataset and spilt
D = load('seeds_dataset.txt');
% + bias
[N,C] = size(D);
bias = ones(N,1);
D = [D(:,1:C-1) bias D(:,C)];

[N,C] = size(D);
[D_1, D_2, D_3] = spiltDateset(D); % split and add bias cloumn
num_updates = 10000;
lambda = 0.01;
n = 1;
binary_losses=zeros(3,1);
% ws = zeros(3,C-1);
ws = zeros(3,C-1); % + bias

```

```

min_loss = N;
min_ws_loss = zeros(3,1);


for i = 1:50

    [w_1,~,bi_loss_1] = soft_svm(D_1, num_updates, lambda, n);
    binary_losses(1) = emp_loss(w_1, D_1, 'binary');
    ws(1,:) = w_1;

    [w_2,~,bi_loss_2] = soft_svm(D_2, num_updates, lambda, n);
    binary_losses(2) = emp_loss(w_2, D_2, 'binary');
    ws(2,:) = w_2;

    [w_3,~,bi_loss_3] = soft_svm(D_3, num_updates, lambda, n);
    binary_losses(3) = emp_loss(w_3, D_3, 'binary');
    ws(3,:) = w_3;

    DR = D(:,C);
    for j = 1:N
        x_j = D(j, 1:C-1);
        t_j = D(j,C);
        [~, I] = max([dot(ws(1,:),x_j), dot(ws(2,:),x_j), dot(ws(3,:),x_j)]);
        DR(j,1) = I;
    end
    Diff = DR(:,1) ~= D(:,C);
    loss = sum(Diff)

    if (loss < min_loss)
        min_loss = loss;
        min_ws_loss = binary_losses;
        opt_DR = DR;
        opt_ws = ws;
    end
end
```

`spiltDateset.m`

```
function [D_1, D_2, D_3] = spiltDateset(D)
[~,C] = size(D);
% each spilt D into two part {-1,1}
D_1 = D;
D_1(D(:,C)==1, C) = 1;
D_1(D(:,C)~=1, C) = -1;

D_2 = D;
D_2(D(:,C)==2, C) = 1;
D_2(D(:,C)~=2, C) = -1;

D_3 = D;
D_3(D(:,C)==3, C) = 1;
D_3(D(:,C)~=3, C) = -1;
```

- In last part, we divide the dataset into three different dataset as follow the part (e) requierments, and we modify the λ and T, compute each $\boldsymbol{w}$ for three dataset and track the binary loss. We select the optimal $\boldsymbol{w}$, which has least binary loss. Then use part (f) function, reclassify the dataset and compute the loss. Repeat this process several times, each time store the optimal weight, binary loss and final loss.
- We obtain the best weight as following:

w								
w_1	-0.9099421	2.03256985	0.51723937	0.58956456	1.47053108	-1.1926746	-3.9665291	0.36748049
w_2	8.29929124	-7.257732	-1.2886759	-3.238241	-1.6755155	1.62626611	-0.3327964	-1.4819588
w_3	-4.3032284	2.12446638	0.34679696	1.27768143	-0.0472385	1.70518543	2.35108058	0.50693703

	w_1	w_2	w_3
Binary Loss	0.09047619	0.02857143	0.03333333

Mutlti-class error: $10/210 = 0.045$