Machine Learning HW 3

R04922034 吳軒衡

1.

ED[Ein(Wlin)] = sig^2\*(1-(d+1)/N) > 0.008

0.01\*(1-9/N) > 0.008

1-9/N > 0.8

9/N < 0.2

N > 9 \* 5 = 45 => N = 46

2.

a, d, e

a,e:

let v be any vector in R^n

Ht = (Xt)t \*(XtX)^(-t)\* (X)t = X \* (XtX)^(-1) Xt = H

H \* H = X \* (XtX)^(-1) Xt \* X \* (XtX)^(-1) Xt = X \* (XtX)^-1 \* Xt = H => H^n = H for n >= 1

<Hv,Hv> = vt\*Ht\*H\*v = vt\*H\*H\*v = vt\*H\*v >=0 since <Hv,Hv> = ||Hv||^2 >=0

=> vt\*H\*v >=0 for all possible v => H is positive semi-definite

d:

H = X\*(XtX)^-1\*Xt

HX = X

let v be eigenvector of X such that Xv = cv for some constant c

(HX)v = H(Xv) = H(cv) = Xv = cv

let q = cv

HXv = H(Xv) = H(cv) = Hq = q => q is an eigenvector of H with corresponding eigenvalues = 1

=> For all eigenvectors v of X, we can construct an eigenvector q of H with eigenvalues =1 by mutiplying the v with its corresponding eigenvalue

=> There are d+1 eigenvalues = H are 1 (X is of R^N x (d+1))

3.

a,b,e

[[sign(wtx)!=y]] =1 >0 when wrong

=0 when right

-y\*wt\*x > 0 when wrong

-ywtx < 0 when right

a):

err(w) = max(0,1-ywtx), when right , count =0 , the 0 in max gaurantees's upperbound

, when wrong, count =1 , 1-ywtx = -ywtx +1 > 1 is also upperbound

b):

err(w) = (max(0,1-ywtx))^2, when right, count =0 , the 0 in max still gaurantees's upperbound

, when wrong, count =1 , 1-ywtx = -ywtx +1 > 1 => (1-ywtx )^2 > 1 still upperbound

c):

err(w) = max(0,-ywtx) when wrong, count =1 , -ywtx >0 but might be less than 1

d):

err(w) = theta(-ywtx) = 1 / (1 + e^(-ywtx)) , when wrong , count =1; 1/1+e^(-ywtx) wher e^(-ywtx) >1 => err(w) < 1/2 not upperbound

e):

err(w) = exp(-ywtx), when right , count =0; exp(negative) >= 0 is upperbound

when wrong, count =1 , exp(something >=0 ) >= 1 is upperbound

4.

b,d,e:

ywtx = yxtw => d(ywtx)dw = yx

ywtx continuous over w and differentiable

for err(w) of a , b , c that consists of max of two functions, we need to consider the point where the two components match

that is 1-ywtx =0

The combination of continuous functions are continuous

a). from 0's point of view , derivative = 0

from 1-ywtx's point of view, derivative = -yx need not be 0

b). from 0^2's point of view , derivative = 0

from (1-ywtx)^2's point of view, by chain rule, the derivative is

2\*(1-ywtx)\*(-yx) where yx is constant and 1-ywtx =0 > derivative = 0

=> differentiable

c). from -ywtx's point of view, derivative = -yx need not be 0

d). v = -ywtx

theta(s) = 1/(1+exp(-s))

err(w) = theta(-ywtx)

d err(w) / dw = d theta(v) / dv \* (dv/dw)

= (v - ln(1+exp(-v)) ) \* (yx)

e).

d exp(v) /dw = d(exp(v)) / dv \* ( dv/dw) = exp(v) \* dv/dw =(-ywtx) \* (-yx)

5.

let err(w) = max(0,-ywtx)

wt+1 = wt - d err(w) / dw

where d err(w) /dw = 0 if -ywtx < 0 , that is , when the prediciton is right

= -yx if -ywtx > 0 , that is , when prediciton is wrong and update is needed

So for SGD requires

wt+1 = wt - (-yx) = wt+yx when wrong

no update when right

This result is the same as the update in PLA .

6.7.

E = @(u,v) exp(u)+exp(2\*v)+exp(u\*v) + u^2 - 2\*u\*v+ 2\*v^2 - 3\*u-2\*v

dEdu =@(u,v) exp(u)+v\*exp(u\*v)+2\*u-2\*v-3;

dEdv= @(u,v) 2\*exp(2\*v)+u\*exp(u\*v)-2\*u+4\*v-2;

eta = 0.01;

u = 0;

v = 0;

for i =1 : 5

fprintf('u= %d, v =%d du = %d dv = %d \n',u,v,dEdu(u,v),dEdv(u,v));

new\_u = u - eta\*dEdu(u,v);

new\_v = v - eta\*dEdv(u,v);

u = new\_u;

v = new\_v;

end

fprintf('u= %d, v =%d du = %d dv = %d \n',u,v,dEdu(u,v),dEdv(u,v));

u= 0, v =0 du = -2 dv = 0

u= 2.000000e-02, v =0 du = -1.939799e+00 dv = -2.000000e-02

u= 3.939799e-02, v =2.000000e-04 du = -1.881220e+00 dv = -3.779752e-02

u= 5.821018e-02, v =5.779752e-04 du = -1.824220e+00 dv = -5.358309e-02

u= 7.645238e-02, v =1.113806e-03 du = -1.768758e+00 dv = -6.753046e-02

u= 9.413996e-02, v =1.789111e-03 du = -1.714795e+00 dv = -7.979840e-02

ans =

2.8250

gradient at (0,0) is (-2,0)

E(u,v) after 5 updates is 2.8250

8.

E = @(u,v) exp(u)+exp(2\*v)+exp(u\*v) + u^2 - 2\*u\*v+ 2\*v^2 - 3\*u-2\*v

f(x) = f(x0) + f'(x0)\*(x-x0) + f''(x0) \*(x-x0)^2 / 2!

let x = [u+du,v+dv]

x0 = [u,v]

E(x) = E(x0) + (x-x0)t \* gradient(E)(x0) + (x-x0)t Hessian(E)(x0) (x-x0) /2!

dEdu =@(u,v) exp(u)+v\*exp(u\*v)+2\*u-2\*v-3;

dEdv= @(u,v) 2\*exp(2\*v)+u\*exp(u\*v)-2\*u+4\*v-2;

d2E/du^2 = @(u,v) exp(u)+v^2\*exp(u\*v)+2

d2E/dudv = @(u,v) exp(u\*v)+v\*u\*exp(u\*v)-2

d2E/dv^2 = @(u,v) 4\*exp(2\*v)+u^2\*exp(u\*v)+4

E(u+du,v+dv) = E(u,v) + [du,dv] \* [dEdu(u,v) dEdv(u,v)]' + [du,dv]\*H(du,dv)\*[du,dv]'/2

E(0,0) = 1+1+1+0-0+0-0-0 =3

dE/du(0,0) = 1+ 0+2-2-3 = -2

dE/dv(0,0) = 2+0-0+0-2 =0

d2E/du^2 = 1+0+2 =3

d2E/dudv = 1+0-2 = -1

d2E/dv^2 = 4+0+4 =8

[du dv] \* [dE/du(u,v) dE,dv(u,v)] = -2\*du

H(du,dv) = [3 -1 ,-1 8]

E(u+du,v+dv) = 3 - 2\*du + (3\*du^2-2\*du\*dv+8\*dv^2) /2

= 1.5\*du^2+4\*dv^2-1\*du\*dv-2\*du+0\*dv+3

=> ANS = (1.5,4,-1,-2,0,3)

9.

E(x+s) = E(x)-(s)'gradient(E)(x) + s'\*Hessian(E)(x)\*s /2

dE(x+s)/ds = gradient(E)(x) + Hessian(E)(x)\*s = 0

=> s = -inv(Hessian(E)(x))\* gradient(E)(x)

= -inv( gradient (gradient (E))(x))\*gradient(E)(x)

10.

E = @(u,v) exp(u)+exp(2\*v)+exp(u\*v) + u^2 - 2\*u\*v+ 2\*v^2 - 3\*u-2\*v;

gE = @(u,v)[exp(u)+v\*exp(u\*v)+2\*u-2\*v-3; 2\*exp(2\*v)+u\*exp(u\*v)-2\*u+4\*v-2; ];

HE = @(u,v)[ exp(u)+v^2\*exp(u\*v)+2 exp(u\*v)+v\*u\*exp(u\*v)-2;exp(u\*v)+v\*u\*exp(u\*v)-2 4\*exp(2\*v)+u^2\*exp(u\*v)+4];

x = [0 ; 0];

for i =1 : 5

dx = - inv(HE(x(1),x(2)))\*gE(x(1),x(2));

fprintf('u =%d , v= %d ,du = %d v = %d\n',x(1),x(2),dx(1),dx(2));

x = x+dx;

end

E(x(1),x(2))

u =0 , v= 0 ,du = 6.956522e-01 v = 8.695652e-02

u =6.956522e-01 , v= 8.695652e-02 ,du = -8.188995e-02 v = -1.584862e-02

u =6.137622e-01 , v= 7.110790e-02 ,du = -1.949361e-03 v = -6.078377e-04

u =6.118129e-01 , v= 7.050006e-02 ,du = -1.142618e-06 v = -5.142346e-07

u =6.118117e-01 , v= 7.049955e-02 ,du = -4.467273e-13 v = -2.802453e-13

ans = 2.3608

11.

X = [ 1 1; 1 -1; -1 -1 ; -1 1; 0 0 ; 1 0]

X = X';

A = []

gen\_row = @(x) [ 1 x(1) x(2) x(1)\*x(2) x(1)\*x(1) x(2)\*x(2)]

for x = X

A = [A ;gen\_row(x)];

end

A =[

1 1 1 1 1 1

1 1 -1 -1 1 1

1 -1 -1 1 1 1

1 -1 1 -1 1 1

1 0 0 0 0 0

1 1 0 0 1 0

]

inv(A) = [

0 0 0 0 1.0000 0

0.2500 0.2500 -0.2500 -0.2500 0 0

0.2500 -0.2500 -0.2500 0.2500 0 0

0.2500 -0.2500 0.2500 -0.2500 0 0

-0.2500 -0.2500 0.2500 0.2500 -1.0000 1.0000

0.5000 0.5000 0 0 0 -1.0000

]

Since inv(A) exists,

for any labeling y , we can solve the equation A\*w = y for w by

w = inv(A)\* y => Thus the all six points can be shattered

12.

Such transformation will result in N by N matrix Z where zij = 1 iff i == j => an Identity Matrix

Thus , for solving an equation Z w = y

It is equivlant to solving I w =y , where the trivial solution is take w = y

That is , for every given y, the hypothesis can shatter that labeling using weight w = y

=> The VC dimension of this hypothesis is infinity

13.

f = @(x) sign(x(1)^2+x(2)^2-0.6)

error\_rate\_sum = 0;

error\_history = [];

for t = 1:1000

X = rand(1000,1) \* 2 -1;

Y = rand(1000,1) \*2 -1;

D = [X Y];

Z = [];

for i = 1:1000

z = f(D(i,:));

if(rand()<=0.1)

z = -z;

end

Z = [Z ;z];

end

D = [ones(1000,1) D];

W = inv(D'\*D)\*D'\*Z;

OUT = D\*W;

err = 0;

for i = 1:1000

if(sign(Z(i))~=sign(OUT(i)))

err= err+1;

end

end

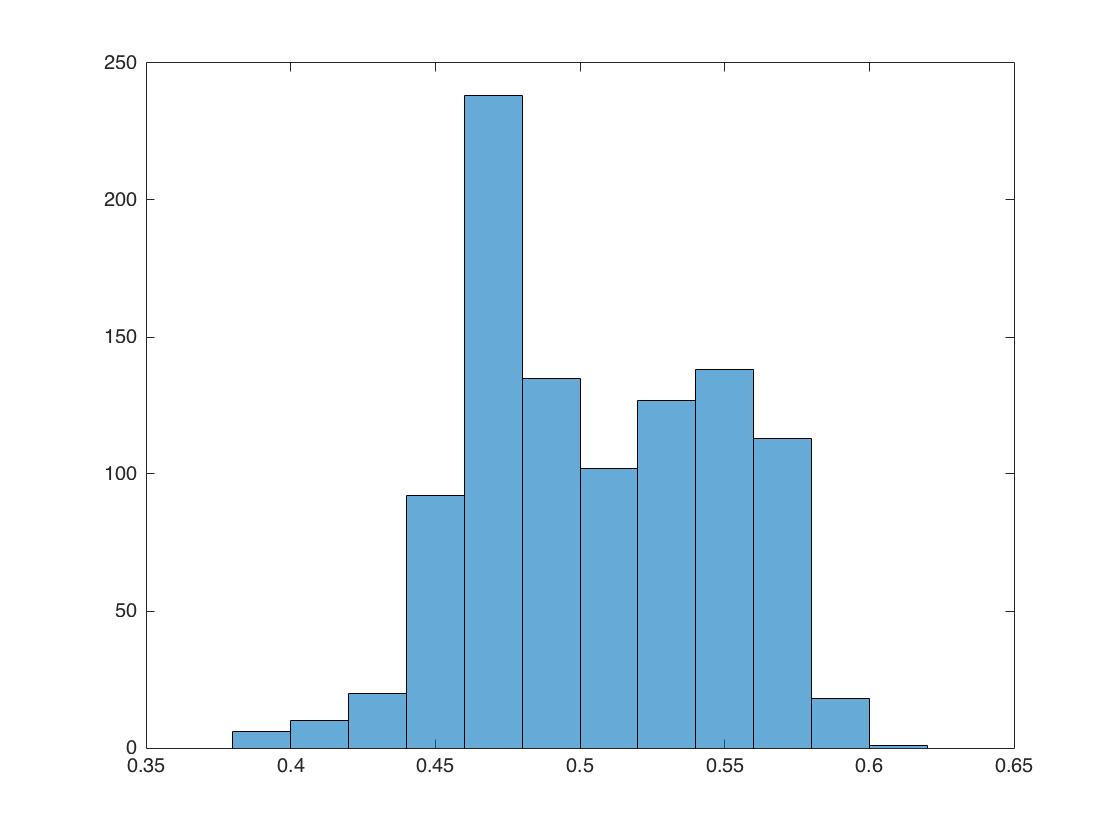
error\_rate\_sum = error\_rate\_sum+err/1000;

error\_history = [error\_history; err/1000];

end

histogram(error\_history)

error\_rate\_sum/1000



average in-sample error = 0.5047

14.

f = @(x) sign(x(1)^2+x(2)^2-0.6)

g = @(x) [1 x(1) x(2) x(1)\*x(2) x(1)\*x(1) x(2)\*x(2)]

error\_rate\_sum = 0;

Ws = [];

for t = 1:1000

X = rand(1000,1) \* 2 -1;

Y = rand(1000,1) \*2 -1;

D = [X Y];

Z = [];

G = [];

for i = 1:1000

z = f(D(i,:));

if(rand()<=0.1)

z = -z;

end

Z = [Z ;z];

G = [G; g(D(i,:))];

end

W = inv(G'\*G)\*G'\*Z;

OUT = G\*W;

err = 0;

for i = 1:1000

if(sign(Z(i))~=sign(OUT(i)))

err= err+1;

end

end

Ws = [Ws W];

error\_rate\_sum = error\_rate\_sum+err/1000;

end

error\_rate\_sum/1000

weight = [];

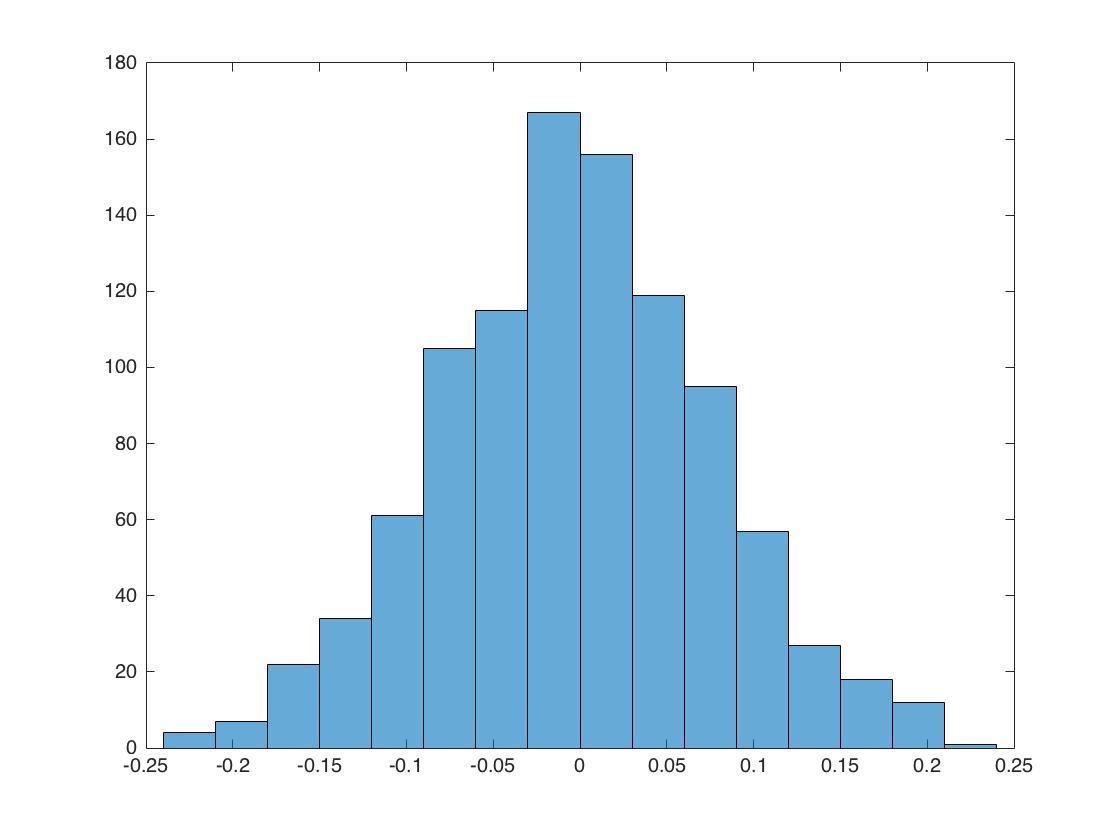
for i = 1:6

weight = [weight ; sum(Ws(i,:))/1000 ];

end

histogram(Ws(4,:));

weight(4)



average w3 = -0.0025

15.

f = @(x) sign(x(1)^2+x(2)^2-0.6)

g = @(x) [1 x(1) x(2) x(1)\*x(2) x(1)\*x(1) x(2)\*x(2)]

error\_rate\_sum = 0;

Ws = [];

error\_history = [];

for t = 1:1000

X = rand(1000,1) \* 2 -1;

Y = rand(1000,1) \*2 -1;

Xout = rand(1000,1) \* 2 -1;

Yout = rand(1000,1) \*2 -1;

D = [X Y];

Dout = [Xout Yout];

Z = [];

Zout = [];

G = [];

Gout = [];

for i = 1:1000

z = f(D(i,:));

zout = f(Dout(i,:));

if(rand()<=0.1)

z = -z;

end

if(rand()<=0.1)

zout = -zout;

end

Zout = [Zout ; zout];

Z = [Z ;z];

G = [G; g(D(i,:))];

Gout = [Gout; g(Dout(i,:))];

end

W = inv(G'\*G)\*G'\*Z;

OUT = Gout\*W;

err = 0;

for i = 1:1000

if(sign(Zout(i))~=sign(OUT(i)))

err= err+1;

end

end

Ws = [Ws W];

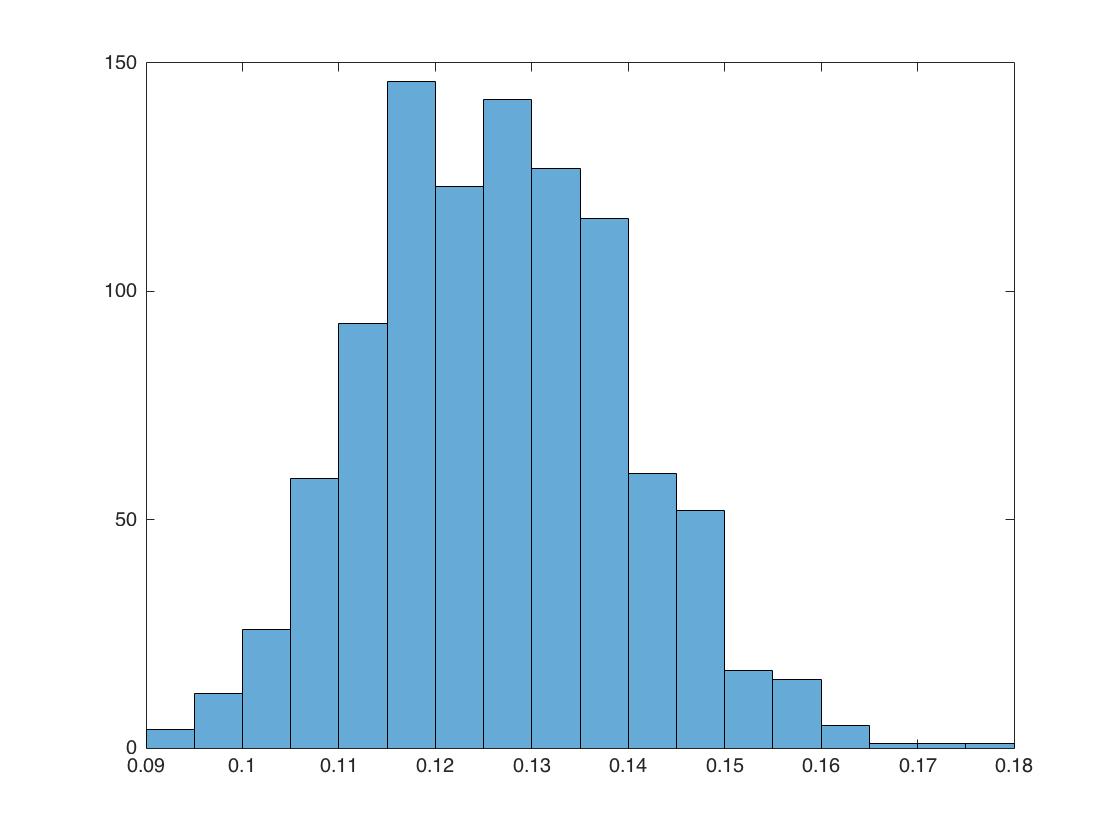
error\_rate\_sum = error\_rate\_sum+err/1000;

error\_history = [error\_history ; err/1000];

end

error\_rate\_sum/1000

histogram(error\_history);



average out of sample error = 0.1260

16.

minimizing negative log likelihood

Ein = -log (hy1(x) \* hy2(x) \* hy3(x) \* ... hyn(x))

= -log hy1(x) + log (hy2(x)) + ... log(hyn(x))

= -sigma (n=1...N)(log exp (wynt\*x)) - N \* log(sigma(i=1...K)(exp(wit\*x)))

= -sigma (n=1...N)(wynt\*x) - N \* log(simga(i=1...K)(exp(wit\*x)))

= sigma (n = 1...N ) (log(sigma(i=1...K)(exp(wit\*xn)))-wynt\*xn)

17.

d log(sigma(i=1...K)(exp(wit\*xn)))-wynt\*xn / dwi

= (sigma(i=1...K) (exp(wit\*xn)))^-1 \* xn\*exp(wit\*xn) - [[yn==i]]xn

= xn\*(hi(xn)-[[yn==i]])

=> gradient(Ein) = sigma(n=1...N) xn\*(hi(xn)-[[yn==i]])

18.

tt = load('hw3\_train.dat');

row\_size = size(tt,1);

feature\_size = size(tt,2)-1;

X = tt(:,1:feature\_size)';

Y =tt(:,feature\_size+1);

err = @(y,w,x) (-y\*x)/(1+exp(y\*w'\*x))

w = zeros(feature\_size,1);

for t = 1:2000

grad = zeros(feature\_size,1);

for i = 1 : row\_size

grad = grad+err(Y(i,1),w,X(:,i));

end

delta = grad/row\_size;

w = w-0.001\*delta;

end

tt = load('hw3\_test.dat');

row\_size = size(tt,1);

feature\_size = size(tt,2)-1;

X = tt(:,1:feature\_size)';

Y =tt(:,feature\_size+1);

count = 0;

for i = 1:row\_size

if(sign(w'\*X(:,i))~=sign(Y(i,1)))

count =count+1;

end

end

w

count / row\_size

w= [

-0.0111

0.0423

-0.0311

0.0166

-0.0351

0.0141

0.0497

-0.0206

0.0263

0.0705

0.0209

-0.0184

-0.0072

0.0476

0.0594

0.0628

-0.0457

0.0622

-0.0146

-0.0333

]

Eout = 0.4717

19.

tt = load('hw3\_train.dat');

row\_size = size(tt,1);

feature\_size = size(tt,2)-1;

X = tt(:,1:feature\_size)';

Y =tt(:,feature\_size+1);

err = @(y,w,x) (-y\*x)/(1+exp(y\*w'\*x))

w = zeros(feature\_size,1);

for t = 1:2000

grad = zeros(feature\_size,1);

for i = 1 : 1000

grad = grad+err(Y(i,1),w,X(:,i));

end

delta = grad/1000;

w = w-0.01\*delta;

end

tt = load('hw3\_test.dat');

row\_size = size(tt,1);

feature\_size = size(tt,2)-1;

X = tt(:,1:feature\_size)';

Y =tt(:,feature\_size+1);

count = 0;

for i = 1:row\_size

if(sign(w'\*X(:,i))~=sign(Y(i,1)))

count =count+1;

end

end

w

count / row\_size

w = [

-0.1894

0.2659

-0.3538

0.0407

-0.3798

0.0195

0.3337

-0.2642

0.1347

0.4912

0.0870

-0.2557

-0.1632

0.3004

0.3999

0.4319

-0.4625

0.4320

-0.2081

-0.3697

]

Eout = 0.2207

20.

tt = load('hw3\_train.dat');

row\_size = size(tt,1);

feature\_size = size(tt,2)-1;

X = tt(:,1:feature\_size)';

Y =tt(:,feature\_size+1);

err = @(y,w,x) (-y\*x)/(1+exp(y\*w'\*x))

w = zeros(feature\_size,1);

random\_index = randperm(row\_size);

for t = 1:2000

grad = err(Y(random\_index(1,mod(t,row\_size)+1)),w,X(:,mod(t,row\_size)+1));

delta = grad;

w = w-0.001\*delta;

end

tt = load('hw3\_test.dat');

row\_size = size(tt,1);

feature\_size = size(tt,2)-1;

X = tt(:,1:feature\_size)';

Y =tt(:,feature\_size+1);

count = 0;

for i = 1:row\_size

if(sign(w'\*X(:,i))~=sign(Y(i,1)))

count =count+1;

end

end

w

count / row\_size

w =[

0.0137

0.0087

0.0064

-0.0013

0.0094

0.0088

0.0051

0.0052

0.0072

0.0270

0.0333

-0.0018

0.0283

0.0078

0.0231

0.0218

0.0043

0.0227

0.0026

0.0221

]

Eout = 0.4770

21.

hty = h(x1)\*y1+...+h(xn)\*yn = sigma(n=1..N) (h(xn)yn)

RMSE^2 = 1/N \* sigma(n=1,N) (yn^2 - 2\*yn\*h(xn)+h(xn)^2)

N\*RMSE^2 = sigma yn^2 - 2 \* sigma \* yn\*h(xn) + sigma h(xn)^2

sigma yn\*h(xn) = 1/2 \* (N\*RMSE^2-sigma yn^2 - sigma h(xn)^2)

since xn is known, h(xn) is known for given , we only need to know sigma yn^2 to compute hty

One way to do so is to first apply a constant hypothesis h =0

such that RMSE(h)^2 \* N = simga (yn-0)^2 = sigma(yn)^2

Thus for any h, we can do two queries , one for h , one for 0 to get hty

22.

N\*RMSE^2 = sigma(n= 1...N) (yn-H(xn))^2

= sigma(n= 1...N) (yn^2 - 2\*yn\*H(xn)+H(xn)^2)

= sigma(n= 1...N) yn^2 - 2\*sigma(n=1...N) yn \*H(xn) + sigma(n=1...N) H(xn)^2

yn\*H(xn) = yn\* sigma(k=1...K) wk \* hk(xn)

= yn\*w1 h1(xn) + yn\*w2 \* h2(xn) ... yn\*wk\*hk(xn)

sigma yn\*H(xn)

= y1\*w1\*h1(x1)+y1\*w2\*h2(x1)... y1\*wk\*hk(x1)

+ y2\*w1\*h1(x2)+y2\*w2\*h2(x2)... y2\*wk\*hk(x2)

+ ...

+ yn\*w1\*h1(xn)+yn\*w2\*h2(xn)... yn\*wk\*hk(xn)

= w1 \* (sigma yn\*h1(xn)) + w2 \* ( sigma yn \* h2(xn))+ ... wk\*(sigma yn\*hk(xn))

since xn is known, H(xn) is known , sigma H(xn)^2 is known

from problem 19, we know that we can get sigma yn^2 by one query

the only thing left to known is sigma yn\*(H(xn)) , which needs to compute sigma yn\*hk(xn) for all k in K

==> Total k+1 queries are needed (the sigma yn ^2 is also used to compute the value of yn \* hk (xn))