Problem 1: Perceptron Learning (30 pts)

## **1. Standard subgradient descent with the step size Gamma= 1 for each iteration.**

# **Code:**

perceptron\_data = importdata('perceptron.data',',');

X = perceptron\_data(:,1:end-1);

Y = perceptron\_data(:,end);

w = zeros(1,size(X,2));

b = 0;

w\_first3 = zeros(3,size(X,2));

b\_first3 = zeros(3,1);

grad\_w = ones(size(w));

grad\_b = ones(size(b));

p\_loss = Inf;

iter = 0;

max\_iter = 1000;

% step size

gamma = 1;

loss\_history = [];

while iter < max\_iter

pred = (X \* w.') + b ;

loss\_each = -1 \* ( Y .\* pred);

incorrect = loss\_each >= 0;

p\_loss = sum(loss\_each .\* incorrect) ;

loss\_history = [loss\_history ,p\_loss] ;

grad\_w = -1 \* (sum((incorrect .\* Y) .\* X));

grad\_b = -1 \*(sum(incorrect .\* Y));

% check if all our gradients are zero, stop if they are

if (grad\_b == 0) && (all(grad\_w==0))

break

end

iter = iter + 1;

w = w - gamma \* grad\_w;

b = b - gamma \* grad\_b;

% to note the first 3 values

if iter <= 3

w\_first3(iter,:) = w;

b\_first3(iter,:) = b;

end

end

plot(loss\_history(2:end));

w\_first3

b\_first3

iter

w

b

**Results:**

**Standard Gradient Descent ; Step size = 1**

First 3 iterations weight vectors =

1.0e+03 \*

1.2790 0.4601 -0.1086 -1.6723

1.3073 0.4327 -0.0276 -1.5238

1.2552 0.4255 0.0188 -1.4347

First 3 iterations bias values =

-354

-493

-625

**Total number of iterations = 46**

**Final weights**

**w = [685.7993 , 243.8995, 8.2420, -797.6251]**

**Final bias**

**b = -1485**

## **2. Stochastic subgradient descent where exactly one component of the sum is chosen to approximate the gradient at each iteration. Instead of picking a random component at each iteration, you should iterate through the data set starting with the first element, then the second, and so on until the Mth element, at which point you should start back at the beginning again.**

## **Again, use the step size t = 1.**

# **Code:**

perceptron\_data = importdata('perceptron.data',',');

X = perceptron\_data(:,1:end-1);

Y = perceptron\_data(:,end);

w = zeros(1,size(X,2));

b = 0;

w\_first3 = zeros(3,size(X,2));

b\_first3 = zeros(3,1);

grad\_w = ones(size(w));

grad\_b = ones(size(b));

p\_loss = Inf;

iter = 0;

max\_iter = Inf;

gamma = 1;

loss\_history = [];

idx = 1;

while iter < max\_iter

pred = (X \* w.') + b ;

loss\_each = -1 \* ( Y .\* pred);

incorrect = loss\_each >= 0;

if sum(incorrect) == 0

break

end

p\_loss = sum(loss\_each .\* incorrect) ;

loss\_history = [loss\_history ,p\_loss] ;

if idx > size(X,1)

idx = mod(idx,size(X,1)) + 1

end

% go to the next incorrect sample

if incorrect(idx) == 0

offset = find(incorrect(idx:end),1);

if isempty(offset)

idx = 1;

offset = find(incorrect(idx:end),1);

end

idx = idx + offset - 1;

end

grad\_w = -1 \* (Y(idx) \* incorrect(idx) \* X(idx,:));

grad\_b = -1 \* (Y(idx) \* incorrect(idx));

% check if all our gradients are zero, stop if they are

% For the sake of uniformity, we use this instead in assignment

w = w - gamma \* grad\_w;

b = b - gamma \* grad\_b;

iter = iter + 1;

idx = idx + 1;

if iter <= 3

w\_first3(iter,:) = w;

b\_first3(iter,:) = b;

end

end

**Results:**

**Stochastic Gradient Descent, Stepsize = 1**

First 3 iterations weight vectors =

4.6175 2.4697 1.9677 -1.8134

3.4532 0.1694 2.6280 -4.6471

0.4561 4.9290 -2.2589 -4.6515

First 3 iterations bias values =

-1

-2

-3

**Total number of iterations = 8694**

**Final weights**

**w = [149.2771, 52.5335, 1.6717, -172.8919]**

**Final Bias**

**b = -322**

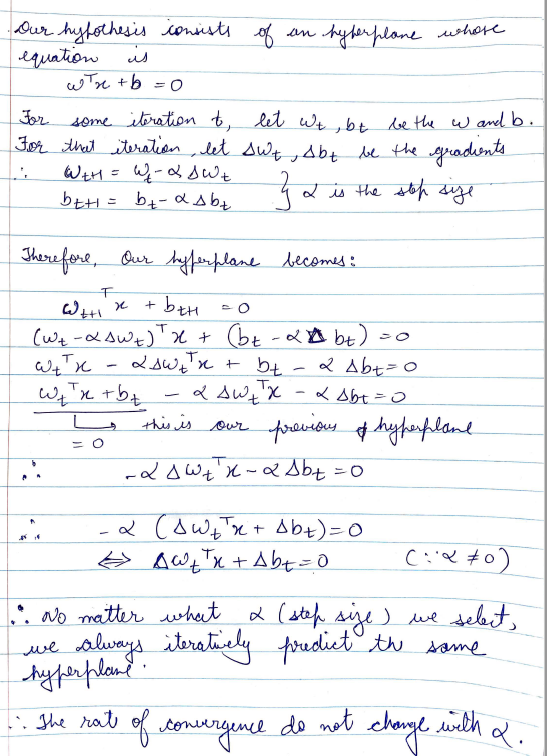
**3. How does the rate of convergence change as you change the step size? Provide some example step sizes to back up your statements.**

**Answer:**

For the standard as well as stochastic gradient descent, for the perceptron algorithm, The code takes the same number of iterations to converge with step sizes 0.001, 1 and 1000.

The rate of convergence doesn’t change with the step size because our hypothesis is scale invariant in w and b.

i.e.



**4. What is the smallest, in terms of number of data points, two-dimensional data set containing both class labels on which the algorithm, with step size one, fails to converge? Use this example to explain why the method may fail to converge more generally.**

**Answer:**

The smallest number of data points, in a 2-D data set containing both class labels on which the algorithm fails to converge is 2.

i.e.

if for some input x1,x2 we get both +ve and -ve label, the algorithm will fail to converge.

(1,1) -> 1

(1,1) -> -1

In general, If we contain an ambiguous dataset where in we get both the labels for same inputs, the algorithm fails to converge.

Also, if no linear separator is possible for the data in our feature space, the algorithm fails to converge.

If we don’t have ambiguous data, that if no input has both the labels then the minimum number of points that wouldn’t converge are **3 co-linear points with an opposite label in between.**

Problem 4: Support Vector Machines

For this problem, consider the data set (mystery.data) attached to this homework that, like Problem

2, contains four numeric attributes per row and the fifth entry is the class variable (either + or

-). Find a perfect classi\_er for this data set using support vector machines. Your solution should

explain the optimization problem that you solved and provide the learned parameters, the optimal

margin, and the support vectors.

**CODE**

mystery\_data= importdata('temp.data',',');

X\_1 = mystery\_data(:,1:end-1);

Y = mystery\_data(:,end);

% second degree polynomial of X

X\_2 = [];

for i = 1:4

for j = i:4

X\_2 = [X\_2 X\_1(:,i).\* X\_1(:,j)];

end

end

X = X\_2;

O = ones(size(X,1),1);

A = [O X];

A = -Y.\*A;

b = - O;

f = zeros(size(A,2),1);

H = eye(size(A,2));

H(1,1) = 0;

[w,fval,exitflag,output,lambda] = quadprog(H,f,A,b);

weights = w(2:end);

bias = w(1);

pred = sign(X \* weights + bias);

diff = abs(Y - pred);

if sum(diff) == 0

disp("Perfect seperator found!");

dist = abs(X \* weights + bias);

sidelines = dist -1;

support\_vectors = find(sidelines<1e-10);

support\_vectors

Margin = 1/ norm(weights);

disp("Optimal Marign: ");

Margin

weights

bias

else

disp("Perfect seperator NOT found! Incorrect predictions = %d",sum(diff)/2);

end

**Results**

Optimization Problem:

As we couldn’t find a linear separator for our input space, we apply a transformation and convert our input to a degree-2 polynomial with each 2-degree term possible.

In the new space, we are able to find a linear separator and its shown below.

These weights are for the following terms:

[ x1^2 , x1x2, x1x3, x1x4, x2^2, x2x3, x3x4, x3^2, x3x4, x4^2]

**weights :**

**[ 378.5014, 0.2089, 85.9106, -123.0933, -0.5346, 14.1634, -0.9070, 7.0084, 44.5379, -258.2477]**

**bias = 96.5406**

**support\_vectors\_ids = [8, 44, 347, 526, 836, 898]**

**Margin = 0.0021**