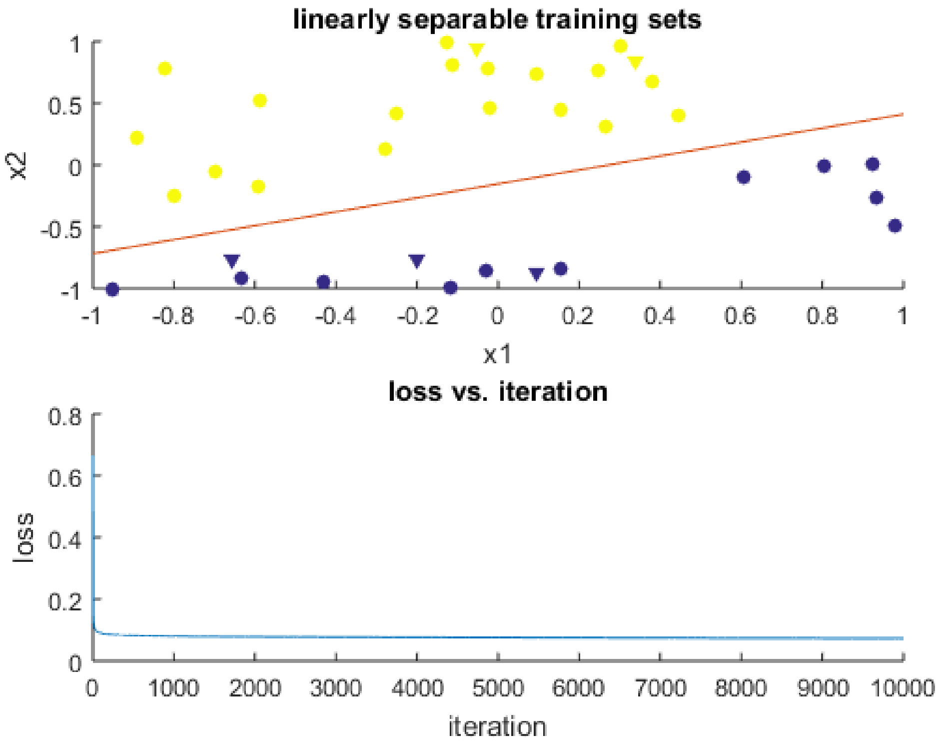
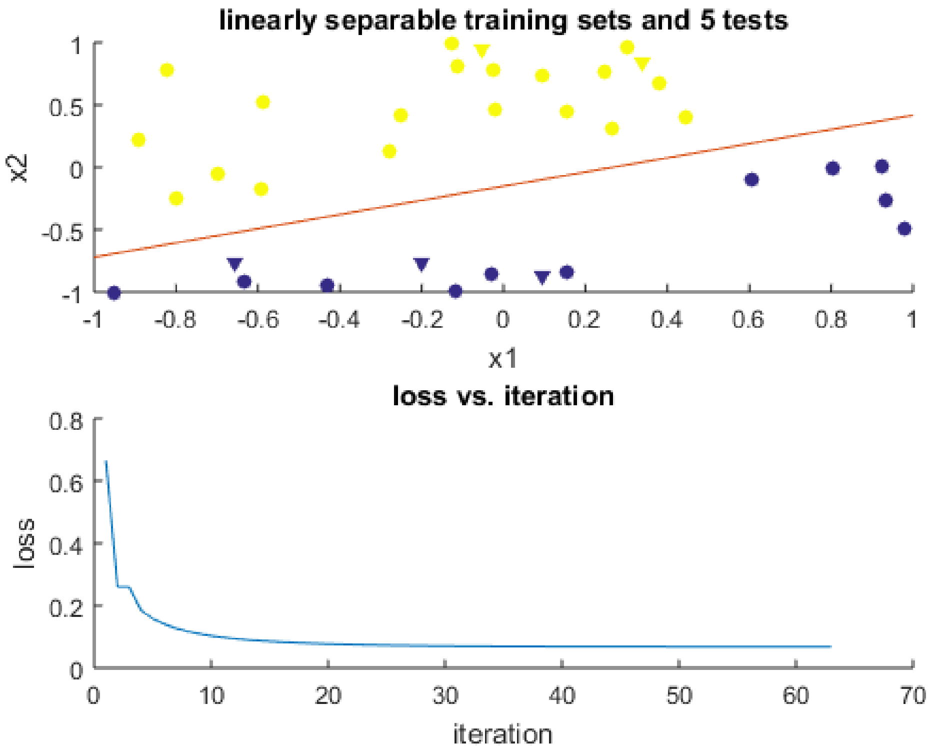
HW3 Report

By Puxin Xu

1. Gradient descent (GD) tested on 2-demensional data set



2. Accelerated gradient descent (AGD) tested on 2-demensional data set



3. Gradient descent (GD) tested on digits

Gradient Descent				
Lamda	01_loss	hinge_loss	Iteration Times	Running Time(s)
0.01	0.012242627	0.036313883	10000	87.38636366
0.1	0.010016694	0.02538688	1830	17.20646443
1	0.013912076	0.023542781	128	2.117978633
10	0.076238175	0.116608984	48	1.064812748

4. Accelerated gradient descent (AGD) tested on digits

Accelerated gradient descent				
Lamda	01_loss	hinge_loss	Iteration Times	Running Time(s)
0.01	0.012242627	0.036506195	601	7.664221403
0.1	0.010016694	0.025387141	549	8.539486435
1	0.013912076	0.023543015	317	7.434504734
10	0.076238175	0.116609761	133	4.284422498

Comparing the testing accuracy with that of the solution produced by Mosek:

Mosek		
Linear		
C	01_loss	hinge_loss
0.01	0.010573178	0.020183827
0.1	0.014468559	0.045595318
1	0.021146355	0.10685606
10	0.020033389	0.143022965

Obviously, on the 2-demensional data set, the result is reasonable and AGD is much more efficient GD. The result of digits is too reasonable by comparing the testing accuracy.

The reason why the iterate times of AGD are more than of GD when lamda is 1 and 10 is that, I use the same initial µ whatever lamda is. When lamda is large enough, the efficiency of AGD becomes negative.

Gradient descent and accelerated Gradient descent Coding:

```
% Gradient descent
function [w,loss,iteration] = GD()
global x % n-1 * m
global y % 1 * m
global lamda
global u
n = size(x,1)+1;
m = length(y);
xd = [x;ones(1,m)];
w = 0.1*ones(1,n); % default w 1*n
loss = [];
for i = 1:10000
    u = max(20/i,0.01);
    fx = sum((log(1+exp(w*(-repmat(y,n,1).*xd)))),2)/m +
lamda/2*norm(w)^2;
```

```
    gradient = sum((-
repmat(y,n,1).*xd)./repmat((1+exp(w*(repmat(y,n,1).*xd))),n,1),2)/m +
lamda*w';
    v = w - u*gradient';
    Q = fx + gradient'*(v-w)' + norm(w-v)^2/2/u;
    fxv = sum((log(1+exp(v*(-repmat(y,n,1).*xd)))),2)/m +
lamda/2*norm(v)^2;
    if Q < fxv
        while Q < fxv
            u = u /2;
            v = w - u*gradient';
            Q = fx + gradient'*(v-w)' + norm(w-v)^2/2/u;
            fxv = sum((log(1+exp(v*(-repmat(y,n,1).*xd)))),2)/m +
lamda/2*norm(v)^2;
        end
    end
    if norm(gradient) <= 10^-4
        break
    end
    w = v;
    loss =[loss,fx];
end
iteration = length(loss);




% Accelerated gradient descent
function [w,loss,iteration] = AGD()
global x % n-1 * m
global y % 1 * m
global lamda
global u
n = size(x,1)+1;
m = length(y);
xd = [x;ones(1,m)];
w = 0.1*ones(1,n);% default w 1*n
v = 0.1*ones(1,n);% default w 1*n
loss = [];
t = 0;
tplus = (1+sqrt(1+4*t^2))/2;
r = (1-t)/tplus;
t = tplus;
for i = 1:10000
```

```
    u = max(20/i,0.01);
    fw = sum((log(1+exp(w*(-repmat(y,n,1).*xd)))),2)/m +
lamda/2*norm(w)^2;
    gradient = sum((-
repmat(y,n,1).*xd)./repmat((1+exp(v*(repmat(y,n,1).*xd))),n,1),2)/m +
lamda*v';
    wplus = v - u*gradient';
    fv = sum((log(1+exp(v*(-repmat(y,n,1).*xd)))),2)/m +
lamda/2*norm(v)^2;
    Q = fv + gradient'*(wplus-v)' + norm(v-wplus)^2/2/u;
    fwplus = sum((log(1+exp(wplus*(-repmat(y,n,1).*xd)))),2)/m +
lamda/2*norm(wplus)^2;
    if Q < fwplus
        while Q < fwplus
            u = u /2;
            wplus = v - u*gradient';
            fv = sum((log(1+exp(v*(-repmat(y,n,1).*xd)))),2)/m +
lamda/2*norm(v)^2;
            Q = fv + gradient'*(wplus-v)' + norm(v-wplus)^2/2/u;
            fwplus = sum((log(1+exp(wplus*(-repmat(y,n,1).*xd)))),2)/m
+ lamda/2*norm(wplus)^2;
        end
    end
    if norm(gradient) <= 10^-4
        break
    end
    vplus = (1-r)*wplus + r*w;
    tplus = (1+sqrt(1+4*t^2))/2;
    r = (1-t)/tplus;
    t = tplus;
    w = wplus;
    v = vplus;
    loss =[loss,fw];
end
iteration = length(loss);
```