EE4350 - Applied Convex Optimization
Linear Support Vector Machine

1 Introduction

The problem Support Vector Machine(SVM) solves can be simply described as a binary classification problem. Given a training dataset of N points of the form

$$\{\boldsymbol{X_1}, y_1\}, \{\boldsymbol{X_2}, y_2\}, \cdots, \{\boldsymbol{X_N}, y_N\}$$

where label y_i (either -1 or 1) indicates which class the point belongs to. The purpose is to find a hyperplane which divides all the points into two classes. Later this classifier can be used for dividing test dataset. The hyperplane can be expressed as follow

$$\boldsymbol{\omega_1^T x} + \omega_0 = 0$$

There may exist several hyperplanes that can divide the points into two groups. SVM is to determine the optimal classifier for the problem where the distance d from the hyperplane to nearest data point on each side is maximized.

$$d = \frac{\mid \boldsymbol{\omega_1^T x} + \omega_0 \mid}{\parallel \boldsymbol{\omega_1} \parallel}$$

In the meantime some constraints must be satisfied in order to find the optimal hyperplane. According to the label of each point, the hyperplane can divide all the points correctly and fulfill the margin d if it satisfies the following equation

$$\begin{cases} (\boldsymbol{\omega_1^T x_i} + \omega_0)/ \parallel \boldsymbol{\omega_1} \parallel \geq d & \forall y_i = 1 \\ (\boldsymbol{\omega_1^T x_i} + \omega_0)/ \parallel \boldsymbol{\omega_1} \parallel \leq -d & \forall y_i = -1 \end{cases}$$

Divide both sides of the inequality by d and obtain

$$\begin{cases} \boldsymbol{\omega_1^T x_i} + \omega_0 \geq 1 & \text{for } y_i = 1 \\ \boldsymbol{\omega_1^T x_i} + \omega_0 \leq -1 & \text{for } y_i = -1 \end{cases}$$

Thus the mathematics model of SVM optimal problem is obtained as follows

$$min_{\omega_1, \omega_0} \frac{1}{2} \parallel \boldsymbol{\omega_1} \parallel^2$$

$$s.t. \quad y_i(\boldsymbol{\omega_1^T x_i} + \omega_0) \geq 1, i = 1, 2, \ldots, m$$

2 Performance of off-the-shelf solver(CVX)

CVX is used here to implement the proposed convex optimization problem. In this section, firstly the algorithm of CVX is introduced. Then training result and testing result are shown in figures. Finally, performance of CVX is discussed using number of iterations and CPU time.

2.1 Interior point methods

Interior point methods are often used for solving linear and nonlinear convex optimization problem. Given an optimization problem, CVX transforms the problem that you give it into the standard Linear Programming,second Order Cone Programming, Semidefinite Programming (LP/SOCP/SDP) form and then the solver uses a primal-dual interior point method to solve the problem.
Primal-dual interior-point methods aim to compute approximately points on the central path. Primal-dual interior point method takes one Newton step per iteration which makes it more efficient and accurate. A series of equality constrained problems have the from

$$minimize \quad f_0(x) - (1/t)\Sigma_{i=1}^{m} \log(-f_i(x))$$

$$subject \quad to \quad \boldsymbol{A}x = b$$

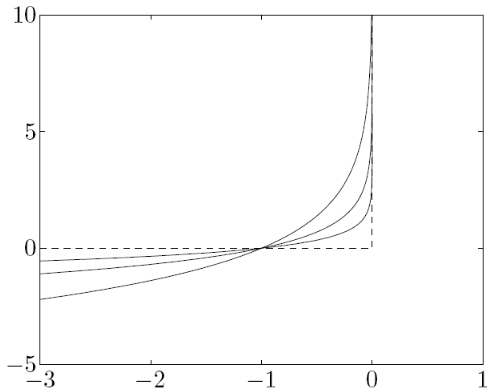

Figure 1: Breif description of interior point method

As t becomes larger, the approximation becomes better.

2.2 Implementation and analysis

A MATLAB framework called "CVX" is used here for data training and testing.

2.2.1 Training and testing results

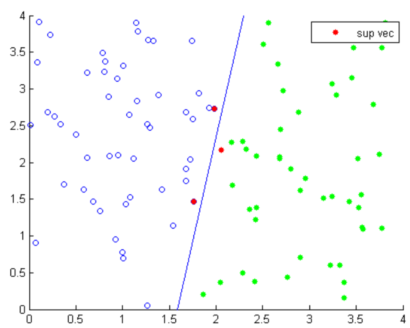

Figure 2: train set and result by CVX

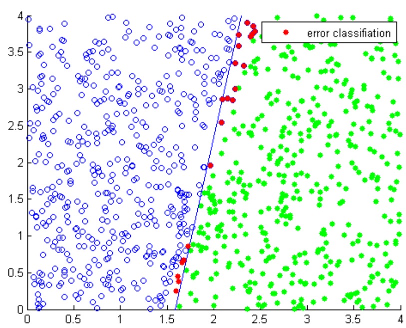

Figure 3: test set and result by CVX

where error rate equals to the number of misclassified samples/the number of total samples.

Table 1: Performance evaluation

number of iterations	CPU time	$\boldsymbol{\omega_1}$	ω_0	error rate
11	1.2763 s	[11.2556;-1.9739]	-17.8885	0.0233

2.2.2 Performance evaluation

Parameters including number of iterations,number of variables and equality constraints are given in the CVX calculation report. CPU time is obtained using tic-toc instruction because it is noticed that there is time difference between tic-toc instruction and cputime instruction. Regardless of the specification of CPU,tic / toc is the direct code which tests the actual execution time.

3 Low-complexity solution for primal problem

3.1 Sub-gradient method for constrained optimization

The algorithm we implement for solving this problem is sub-gradient method for constrained optimization, and the details about this algorithm are in the textbook.We just give a brief introduction. The standard form of inequality constrained problem is

$$minimize \quad f_0(x)$$

$$s.t. \quad f_i(x) \leq 0, \quad i = 1, 2, \ldots, m$$

After transforming our problem to standard form, we get

$$min_{\omega_1,\omega_0} \frac{1}{2} \parallel \boldsymbol{\omega_1} \parallel^2$$

$$min_{\omega_1,\omega_0} \frac{1}{2} \boldsymbol{\omega_1^T} \boldsymbol{\omega_1}$$

$$s.t. \quad -y_i(\boldsymbol{\omega_1^T} \boldsymbol{x_i} + \omega_0) + 1 \leq 0, i = 1, 2, \ldots, m$$

Then for simplification we take

$$\boldsymbol{\omega} = [\boldsymbol{\omega_1}, \omega_0]$$

After each iteration, we have to update the variables following the form:

$$\boldsymbol{\omega}^{(k+1)} = \boldsymbol{\omega}^{(k)} - \alpha_k g^{(k)}$$

where α_k is a step size and $g^{(k)}$ is a sub-gradient of the objective function or one of the constraint functions at $x^{(k)}$.More specifically, we take

$$g^{(k)} \in \begin{cases} [\boldsymbol{\omega_1}, 0] & f_i(\boldsymbol{\omega}^{(k)}) \leq 0, i = 1, 2, \ldots, m \\ [-x_j y_i, -y_j] & f_j(\boldsymbol{\omega}^{(k)}) > 0 \end{cases}$$

$$\alpha_k = \begin{cases} c \qquad x^{(k)} feasible \\ (f_i(x^{(k)}) + E) / \parallel \boldsymbol{g^{(k)}} \parallel_2^2 \quad x^{(k)} infeasible \end{cases}$$

Where c is a constant,we take c = 1/k and E is a small positive margin.

3.2 Implementation and analysis

The body matlab code for the implementation is

```
tic
while k<MAX_ITERS
    if (k>100) %check if the program can stop
        if((abs(a0(end)-a0(end-1))<diff &&
        \abs(a1(end)-a1(end-1))<diff &&
        \abs(a2(end)-a2(end-1)) < diff))
```

```
                break;
            end
        end
      % feasibility check; fval: the value of first constrain which is not satisfied
      [fval,ind] = min(labels_train.*(X_train * w1 + w0)-1);
       % subgradient and step size computation
      if( fval < 0 ) % infeasible
         g = [(X_train(ind,:)*labels_train(ind)), labels_train(ind)]
         alpha = (fval+E)/norm(g)^2;
      else % feasible
         g = [w1' , 0]; alpha = 1/k;
      end
      %store the values of each iteration
      a0(end+1)=w1(1); a1(end+1)=w1(2); a2(end+1)=w0;
      % subgradient update
      w1 = w1 - alpha*g(1:2)'
      w0 = w0 - alpha*g(3)
      k = k + 1;
end
toc
```

Listing 1: implementation of sub-gradient method

When the difference of variables between two successive steps are smaller than $diff$, the program will end even if it has not reached the maximum iteration.For getting the processing time, we use tic-toc method to record the running time of the body codes.In our implementation, and we set $diff = 0.01$ and E=0.001.Then three different Maximum Iterations are chosen for testing.

3.2.1 Tests for different settings

Table 2: Comparison of the different settings about Maximum Iteration

Max Iteration	**real number of iteration**	**processing time**	$\boldsymbol{\omega_1}$	ω_0	**error rate**
1000	1000	0.099506 s	[4.1606,-1.0974]	-6.7018	0.0389
3000	3000	0.241276 s	[8.4533,-1.6283]	-13.4685	0.0189
10000	7770	0.632488 s	[10.9415,-1.9346]	-17.3929	0.0233

Where the error rate refers to the rate of classification error gained by applying the SVM trained by train set to test set.we can see from the table that the results under third setting are closer to the optimal results obtained by using CVX.

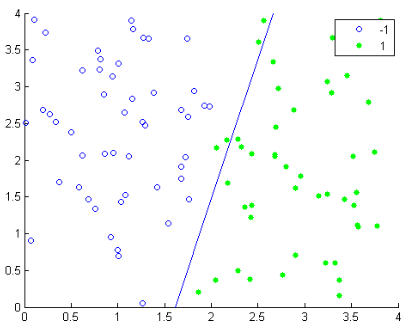

Figure 4: train set and SVM for max iteration=1000

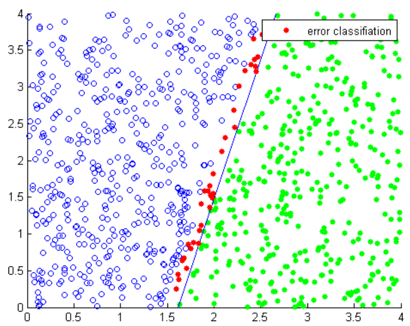

Figure 5: test set and SVM for max iteration=1000

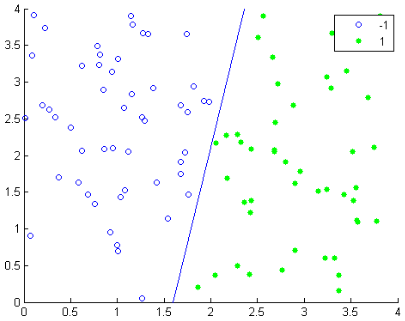

Figure 6: train set and SVM for max iteration=3000

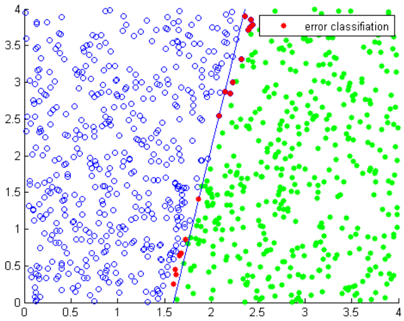

Figure 7: test set and SVM for max iteration=3000

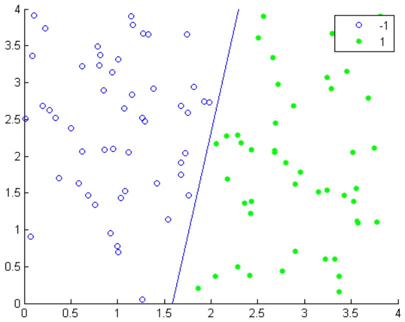

Figure 8: train set and SVM for max iteration=10000

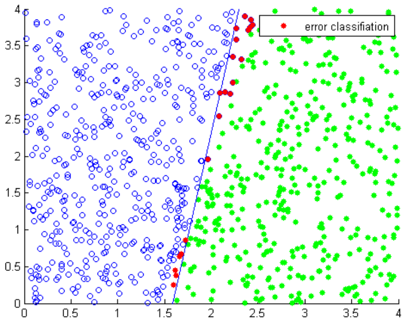

Figure 9: test set and SVM for max iteration=10000

3.2.2 Convergence

The mathematical proof of convergence of sub-gradient method is illustrated in textbook, here we just show the convergence with plots directly.

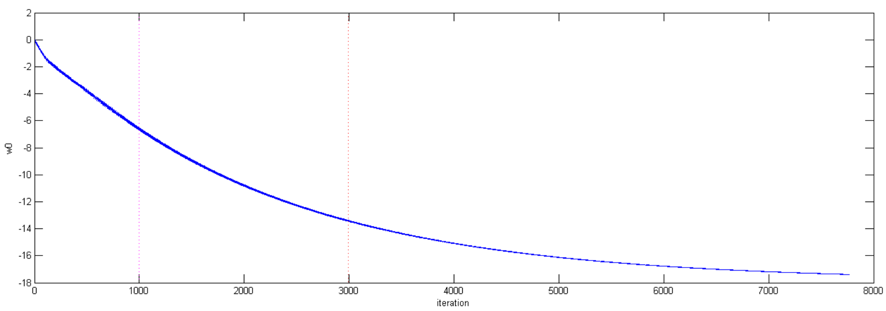

Figure 10: convergence of ω_0

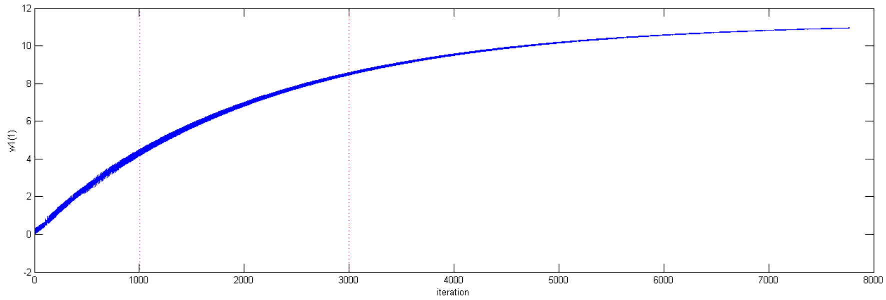

Figure 11: convergence of $\boldsymbol{\omega_1(1)}$

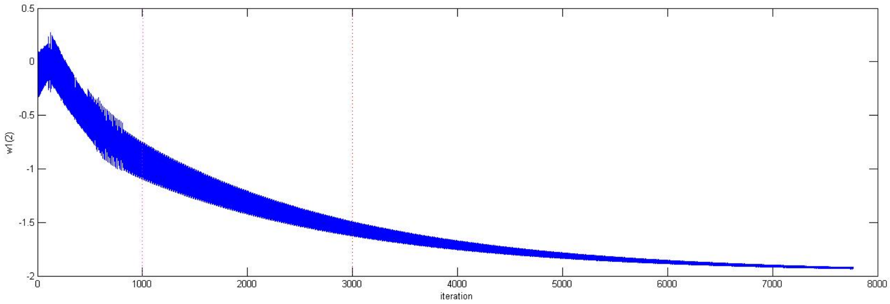

Figure 12: convergence of $\boldsymbol{\omega_1(2)}$

4 Conclusion

In conclusion,mature CVX tool is able to solve the given problem in limited iterations(11 times) and get the optimal solution.By contrast, the low-complexity algorithm we implement ed needs much more iterations(7770 times) before getting the solution because it is first-order methods which means that the difference between two steps is smaller.However, our algorithm needs less processing time compared with CVX tool for this problem to get the same result.Interestingly, from the results of table 2, we can find that the best solution(the third one) under the training data set does not perform best.This is called over-fitting which is an important issue in machine learning.

A matlab codes for low-complexity algorithm implementation

```
% Primal SVM problem:
%    minimize     1/2 * ||w||^2
%    subject to  y(i)*(w' * x(i) +b) >=1   i=1,2,......,m
clear all;
load('linear_svm.mat')
[num,dim]=size(X_train);%num : number of data
[num2,dim2]=size(X_test);
%yi = labels_train
%xi = X_train
%--------------------solve by cxv tool------------------%
cvx_begin
    variable w(dim);
    variable b;
    minimize( 1/2*norm(w) );
    subject to
        labels_train.*( X_train * w + b) -1 >= 0;
cvx_end

%-------------------------------------------------------%
a0=[];%store the values of w1(1)
a1=[];%store the values of w1(2)
a2=[];%store the values of w0
disp('Starting projected subgradient algorithm for the primal problem...')
k = 1;
MAX_ITERS = 15000;
num_infeasible=0;
num_feasible=0
% initial point
w1 = zeros(dim,1);
w0=0;
E=0.001
diff = 0.001
%------------------solved by subgradient method---------------%
tic
while k<MAX_ITERS
   if (k>100)
       if((abs(a0(end)-a0(end-1))
       \<diff && abs(a1(end)-a1(end-1))
       \<diff && abs(a2(end)-a2(end-1)) < diff))
           break;
       end
   end
  % feasibility check
  [fval,ind] = min(labels_train.*(X_train * w1 + w0)-1);
   % subgradient and step size computation
  if( fval < 0 ) % infeasible
    g = [(X_train(ind,:)*labels_train(ind)), labels_train(ind)]
    alpha = (fval+E)/norm(g)^2;
  else % feasible
    g = [w1' , 0];
    alpha = 1/k;
  end
  a0(end+1)=w1(1);
  a1(end+1)=w1(2);
  a2(end+1)=w0;
  % subgradient update
```

```
  w1 = w1 - alpha*g(1:2)'
  w0 = w0 - alpha*g(3)
  k = k + 1;
end
toc
%--------------------------------------------------------------%
figure (1)
plot( [1:length(a0)], a0 )
figure (2)
plot( [1:length(a1)], a1 )
figure (3)
plot( [1:length(a2)], a2 )
figure (4)
x2=-5:5;
x1=-(w1(1)/w1(2))*x2-w0/w1(2);
xmin=0;
xmax=4;
ymin=0;
ymax=4
gscatter(X_test(:,1),X_test(:,2),labels_test,'bg','o.',[5,15]);
hold on
plot(x2,x1,'b')
axis([xmin xmax ymin ymax])
out1 = labels_test.*( X_test * w1 + w0);
ind1 = find( out1 <= 0);
for i = 1:length(ind1)
    gscatter(X_test(ind1(i),1),X_test(ind1(i),2),'error classifiation');
end

[num_err dim3]=size(ind1)
err_rate = num_err/num2
figure(5)

out3 = labels_train.*( X_train * w1 + w0);
out3=round(out3*10)/10;
ind3 = find( out3 == 1);

gscatter(X_train(:,1),X_train(:,2),labels_train,'bg','o.',[5,15]);
hold on
plot(x2,x1,'b')
axis([xmin xmax ymin ymax])
```

B matlab codes for solving the problem by CVX

```
clear all;
load('linear_svm.mat')
[num,dim]=size(X_train);
[num2,dim2]=size(X_test);
tic
cvx_begin
    variable w(dim);
    variable b;
    minimize( 1/2*norm(w) );
    subject to
        labels_train.*( X_train * w + b) -1 >= 0;
cvx_end
toc
% find support vector
out = labels_train.*( X_train * w + b);
out=round(out*100)/100;
ind = find( out == 1);
% draw support vector
figure,
x2=-5:5;
x1=-(w(1)/w(2))*x2-b/w(2);
xmin=0;
xmax=4;
ymin=0;
ymax=4
gscatter(X_train(:,1),X_train(:,2),labels_train,'bg','o.',[5,15]);
hold on
plot(x2,x1,'b')
axis([xmin xmax ymin ymax])
hold on
for i = 1:length(ind)
    gscatter(X_train(ind(i),1),X_train(ind(i),2),'sup vec');
end
% test svm
figure (2)
gscatter(X_test(:,1),X_test(:,2),labels_test,'bg','o.',[5,15]);
hold on
plot(x2,x1,'b')
axis([xmin xmax ymin ymax])
% find support vector
out1 = labels_test.*( X_test * w + b);
out1=round(out1*100)/100;
ind1 = find( out1 < 0);
for i = 1:length(ind1)
    gscatter(X_test(ind1(i),1),X_test(ind1(i),2),'error classifiation');
end
[num_err dim3]=size(ind1)
err_rate = num_err/num2
```