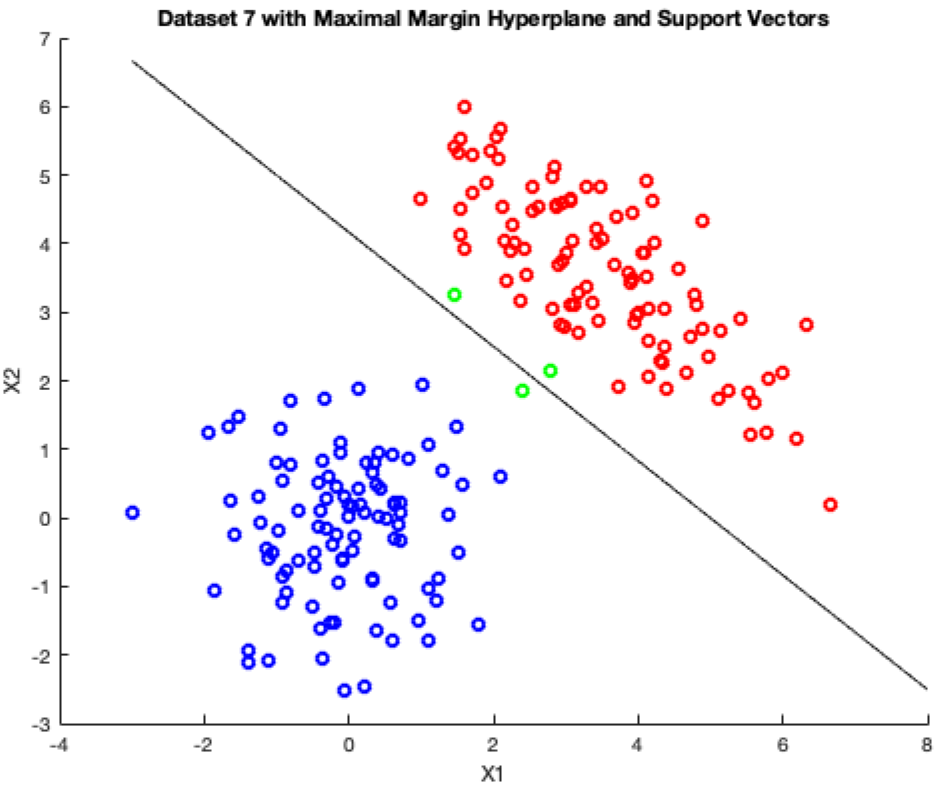
```
% IDS/ACM/CS 158: Fundamentals of Statistical Learning
% PS5, Problem 1: Maximal Margin Hyperplane
% Author: Michael Li, mlli@caltech.edu
%-----------------------------------------------------------------------
clear;

D = readmatrix('dataset7.csv');
X = D(:, 1:end-1);
ys = D(:,end);
g_plus = D(ys == 1, 1:end-1);
g_minus = D(ys == -1, 1:end-1);

N = size(D,1);
p = size(D(1,1:end-1), 2);
X_with_bias = [ones(N,1),X];

% Primal values for beta
primal_margin = quadprog(eye(p+1), zeros(p+1, 1), ys.*X_with_bias*-1,
 -1+zeros(N, 1));
fprintf("\nPrimal Maximal Margin Hyperplane Beta: \n")
disp(primal_margin)

x = linspace(-3, 8, 10000);
f=@(x) (-primal_margin(2) / primal_margin(3))*x - (primal_margin(1) /
 primal_margin(3));
Y=f(x);

% Dual approach for beta
H = (ys*transpose(ys)) .* (X*transpose(X));
dual_margin = quadprog(H, -1*ones(1,N), zeros(1,N), 0, transpose(ys),
 0, zeros(N,1), 10^10*ones(N,1));

% Find beta from lambdas
support_vecs = X(abs(dual_margin) > 10^-5, :);
beta = sum(dual_margin .* ys .* X, 1);
beta0 = -1/2 * (min(beta*transpose(g_plus)) +
 max(beta*transpose(g_minus)));
dual_beta = [beta0 beta];
fprintf("\nDual Maximal Margin Hyperplane Beta: \n")
disp(dual_beta)

% plot
figure
hold on
plot(x, Y, 'k')
plot(g_plus(:,1), g_plus(:, 2), 'or')
plot(g_minus(:,1), g_minus(:, 2), 'ob')
plot(support_vecs(:,1), support_vecs(:,2), 'og')
title('Dataset 7 with Maximal Margin Hyperplane and Support Vectors')
xlabel('X1')
ylabel('X2')
```

```
% primal margin hyperplane beta = [-13.6254, 2.7269, 3.2707]
% dual margin hyperplane beta = [-13.6254, 2.7269, 3.2707]
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint
 tolerance.


Primal Maximal Margin Hyperplane Beta:
  -13.6254
    2.7269
    3.2707


Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint
 tolerance.


Dual Maximal Margin Hyperplane Beta:
  -13.6254    2.7269    3.2707
```




*Published with MATLAB® R2019a*

2\. a. $H_{\hat{\beta}_0, \hat{\beta}} = \{X : \hat{\beta}_0 + \hat{\beta}^T X = 0\} \subset \mathbb{R}^p$

From Lecture 14 Page 81 $\hat{d} = \frac{1}{\|\hat{\beta}\|}$

We want to shift all $X$ a distance $\hat{d}$ in the direction orthogonal to $\beta$ $\rightarrow$ $X^{\pm} = X \pm \frac{\hat{d}\hat{\beta}}{\|\beta\|}$ $d$

$X = X^+ - \frac{d\beta}{\|\beta\|}$

$X = X^- + \frac{d\beta}{\|\beta\|}$

$$\boxed{\begin{aligned} H^+ &= \{X^+ : \hat{\beta}_0 + \beta^T(X^+ - \tfrac{d\beta}{\|\beta\|}) = 0\} \\ H^- &= \{X^- : \hat{\beta}_0 + \beta^T(X^- + \tfrac{d\beta}{\|\beta\|}) = 0\} \end{aligned}}$$

$$H^+ = \{X^+ : \hat{\beta}_0 + \beta^T(X^+ - \tfrac{\beta}{\|\beta\|^2}) = 0\}$$
$$H^- = \{X^- : \hat{\beta}_0 + \beta^T(X^- + \tfrac{\beta}{\|\beta\|^2}) = 0\}$$

$$\boxed{\begin{aligned} H^+ &= \{X^+ : \hat{\beta}_0 + \beta^T X^+ = 1\} \\ H^- &= \{X^- : \hat{\beta}_0 + \beta^T X^- = -1\} \end{aligned}}$$

```
% IDS/ACM/CS 158: Fundamentals of Statistical Learning
% PS5, Problem 2: Soft Margin Hyperplane
% Author: Michael Li, mlli@caltech.edu
%-----------------------------------------------------------------------
clear;

D = readmatrix('dataset8.csv');
X = D(:, 1:end-1);
ys = D(:,end);
g_plus = D(ys == 1, 1:end-1);
g_minus = D(ys == -1, 1:end-1);

% C = .1
C = .1;
N = size(D,1);
p = size(D(1,1:end-1), 2);

% solve dual problem using C constraint
H = (ys*transpose(ys)) .* (X*transpose(X));
dual_margin = quadprog(H, -1*ones(1,N), zeros(1,N), 0, transpose(ys),
 0, zeros(N,1), (1/C)*ones(N,1));

% finding beta using lambdas
support_vecs = D(abs(dual_margin) > 10^-5, :);
support_plus = support_vecs(support_vecs(:,3)==1, 1:end-1);
support_minus = support_vecs(support_vecs(:,3)==-1, 1:end-1);
beta = sum(dual_margin .* ys .* X, 1);
beta0 = -1/2 * (max(beta*transpose(support_plus)) +
 min(beta*transpose(support_minus)));
dual_beta = [beta0 beta];
fprintf("\nNumber of Support Vectors C=.1 are %i\n",
 size(support_vecs, 1))
% C=.1 has 8 support vectors

% get points for decision boundary and margins
x = linspace(-3, 5, 10000);
f=@(x) (-dual_beta(2) / dual_beta(3))*x - (dual_beta(1) /
 dual_beta(3));
Y=f(x);

g=@(x) (-dual_beta(2) / dual_beta(3))*x - ((dual_beta(1) - 1) /
 dual_beta(3));
Z=g(x);

h=@(x) (-dual_beta(2) / dual_beta(3))*x - ((dual_beta(1) + 1) /
 dual_beta(3));
P=h(x);

% plot
figure
hold on
plot(x, Y, 'k')
```

```
plot(x, Z, '--k')
plot(x, P, '--k')
plot(g_plus(:,1), g_plus(:, 2), 'or')
plot(g_minus(:,1), g_minus(:, 2), 'ob')
plot(support_vecs(:,1), support_vecs(:,2), 'xk')
title('Dataset 8 with Soft Margin Hyperplane C=.1')
xlabel('X1')
ylabel('X2')

% C = 10
C = 10;

% solve dual problem using C constraint
H = (ys*transpose(ys)) .* (X*transpose(X));
dual_margin = quadprog(H, -1*ones(1,N), zeros(1,N), 0, transpose(ys),
 0, zeros(N,1), (1/C)*ones(N,1));

% finding beta using lambdas
support_vecs = D(abs(dual_margin) > 10^-5, :);
support_plus = support_vecs(support_vecs(:,3)==1, 1:end-1);
support_minus = support_vecs(support_vecs(:,3)==-1, 1:end-1);
beta = sum(dual_margin .* ys .* X, 1);
beta0 = -1/2 * (max(beta*transpose(support_plus)) +
 min(beta*transpose(support_minus)));
dual_beta = [beta0 beta];
% disp(dual_beta)
fprintf("\nNumber of Support Vectors C=10 are %i\n",
 size(support_vecs, 1))
% C=10 has 27 support vectors

% get points for decision boundary and margins
x = linspace(-3, 5, 10000);
f=@(x) (-dual_beta(2) / dual_beta(3))*x - (dual_beta(1) /
 dual_beta(3));
Y=f(x);
g=@(x) (-dual_beta(2) / dual_beta(3))*x - ((dual_beta(1) - 1) /
 dual_beta(3));
Z=g(x);
h=@(x) (-dual_beta(2) / dual_beta(3))*x - ((dual_beta(1) + 1) /
 dual_beta(3));
P=h(x);

% plot
figure
hold on
plot(x, Y, 'k')
plot(x, Z, '--k')
plot(x, P, '--k')
plot(g_plus(:,1), g_plus(:, 2), 'or')
plot(g_minus(:,1), g_minus(:, 2), 'ob')
plot(support_vecs(:,1), support_vecs(:,2), 'xk')
title('Dataset 8 with Soft Margin Hyperplane C=10')
xlabel('X1')
ylabel('X2')
```

*Minimum found that satisfies the constraints.*

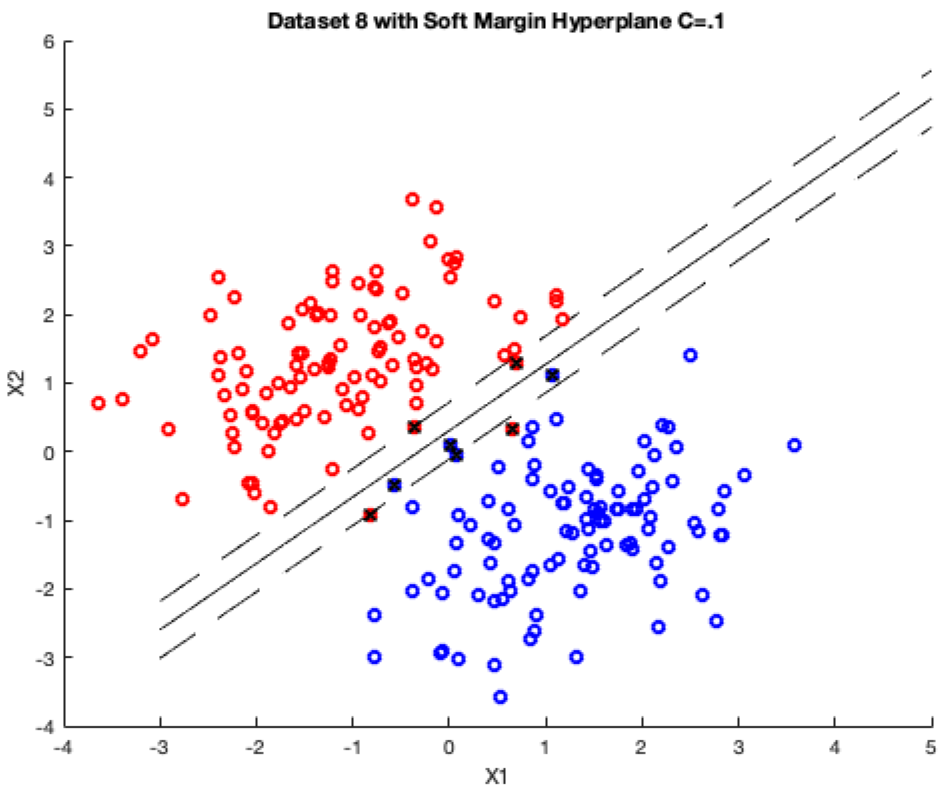
*Optimization completed because the objective function is non-decreasing in*
*feasible directions, to within the value of the optimality tolerance,*
*and constraints are satisfied to within the value of the constraint tolerance.*

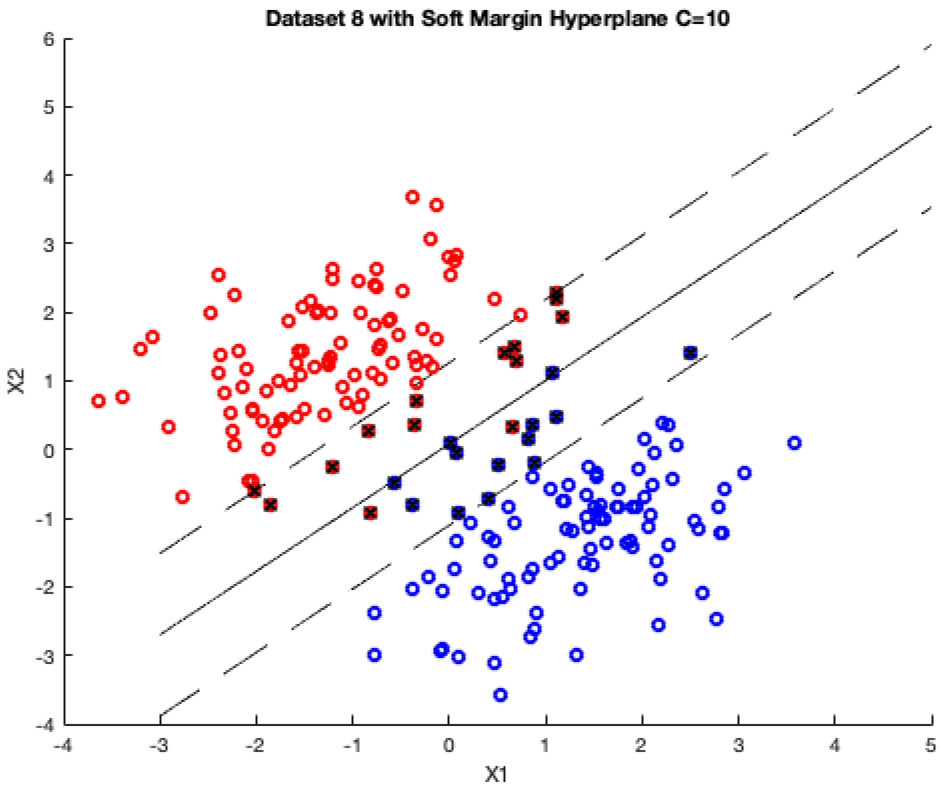
*Number of Support Vectors C=.1 are 8*

*Minimum found that satisfies the constraints.*

*Optimization completed because the objective function is non-decreasing in*
*feasible directions, to within the value of the optimality tolerance,*
*and constraints are satisfied to within the value of the constraint tolerance.*

*Number of Support Vectors C=10 are 27*






*Published with MATLAB® R2019a*

```
% IDS/ACM/CS 158: Fundamentals of Statistical Learning
% PS5, Problem 3: Support Vector Machine
% Author: Michael Li, mlli@caltech.edu
%-----------------------------------------------------------------------
clear;

D = readmatrix('dataset9.csv');
X = D(:, 1:end-1);
ys = D(:,end);
g_plus = D(ys == 1, 1:end-1);
g_minus = D(ys == -1, 1:end-1);

% C = 0
C = 0;
N = size(D,1);
p = size(D(1,1:end-1), 2);
K = ones(size(ys*transpose(ys)));

% generate kernel matrix and find lambdas
for i=1:length(X)
    for j=1:length(X)
        K(i, j) = gaussKernel(X(i,:), X(j,:));
    end
end

H = (ys*transpose(ys)) .* K;
svm = quadprog(H, -1*ones(1,N), zeros(1,N), 0, transpose(ys), 0,
 zeros(N,1), (1/C)*ones(N,1));

% find support vectors
support_vecs = D(abs(svm) > 10^-5, :);
support_plus = support_vecs(support_vecs(:,3)==1, 1:end-1);
support_minus = support_vecs(support_vecs(:,3)==-1, 1:end-1);
nonzero_lambs = svm(abs(svm) > 10^-5, :);
lambs_plus = nonzero_lambs(support_vecs(:,3)==1);
lambs_minus = nonzero_lambs(support_vecs(:,3)==-1);

% find beta0
b0_max = 0;
for i=1:length(support_plus)
    tot = 0;
    for j=1:length(support_plus)
        tot = tot + lambs_plus(j) * gaussKernel(support_plus(j,:),
 support_plus(i,:));
    end

    if tot > b0_max
        b0_max = tot;
    end
end

b0_min = 10^100;
```

```
for i=1:length(support_minus)
    tot = 0;
    for j=1:length(support_minus)
        tot = tot + -lambs_minus(j) * gaussKernel(support_minus(j,:),
 support_minus(i,:));
    end

    if tot < b0_max
        b0_min = tot;
    end
end

b0 = -1/2*(b0_max + b0_min);

fprintf("\nNumber of Support Vectors C=0 are %i\n", size(support_vecs,
 1))
% For C=0, there are 30 Support Vectors

% get points for boundary
x1 = -2:.004:2;
x2 = -2:.004:2;
boundary_x_0 = [];
boundary_y_0 = [];

% test each point
for i=x1
    for j=x2
        tot = b0;
        for k=1:length(support_vecs)
            tot = tot + support_vecs(k,3) * nonzero_lambs(k) *
 gaussKernel(support_vecs(k, 1:end-1), [i j]);
        end

        if abs(tot) < .04
            boundary_x_0 = [boundary_x_0 i];
            boundary_y_0 = [boundary_y_0 j];
        end
    end
end

% C=1
C = 1;
N = size(D,1);
p = size(D(1,1:end-1), 2);
K = ones(size(ys*transpose(ys)));

% generate kernel matrix and find lambdas
for i=1:length(X)
    for j=1:length(X)
        K(i, j) = gaussKernel(X(i,:), X(j,:));
    end
end

H = (ys*transpose(ys)) .* K;
```

```
svm = quadprog(H, -1*ones(1,N), zeros(1,N), 0, transpose(ys), 0,
 zeros(N,1), (1/C)*ones(N,1));

% find support vectors
support_vecs = D(abs(svm) > 10^-5, :);
support_plus = support_vecs(support_vecs(:,3)==1, 1:end-1);
support_minus = support_vecs(support_vecs(:,3)==-1, 1:end-1);
nonzero_lambs = svm(abs(svm) > 10^-5, :);
lambs_plus = nonzero_lambs(support_vecs(:,3)==1);
lambs_minus = nonzero_lambs(support_vecs(:,3)==-1);

% find beta0
b0_max = 0;
for i=1:length(support_plus)
    tot = 0;
    for j=1:length(support_plus)
        tot = tot + lambs_plus(j) * gaussKernel(support_plus(j,:),
 support_plus(i,:));
    end

    if tot > b0_max
        b0_max = tot;
    end
end

b0_min = 10^100;
for i=1:length(support_minus)
    tot = 0;
    for j=1:length(support_minus)
        tot = tot + -lambs_minus(j) * gaussKernel(support_minus(j,:),
 support_minus(i,:));
    end

    if tot < b0_max
        b0_min = tot;
    end
end

b0 = -1/2*(b0_max + b0_min);

fprintf("\nNumber of Support Vectors C=1 are %i\n", size(support_vecs,
 1))
% For C=1, there are 61 Support Vectors

% get points for boundary
x1 = -2:.004:2;
x2 = -2:.004:2;
boundary_x = [];
boundary_y = [];

% test each point
for i=x1
    for j=x2
        tot = b0;
```

```
        for k=1:length(support_vecs)
            tot = tot + support_vecs(k,3) * nonzero_lambs(k) *
 gaussKernel(support_vecs(k, 1:end-1), [i j]);
        end

        if abs(tot) < .04
            boundary_x = [boundary_x i];
            boundary_y = [boundary_y j];
        end
    end
end

% plot
figure
hold on
plot(g_plus(:,1), g_plus(:, 2), 'or')
plot(g_minus(:,1), g_minus(:, 2), 'ob')
plot(boundary_x_0, boundary_y_0)
plot(boundary_x, boundary_y)
legend('+ Class', '- Class', 'C=1 Boundary', 'C=0 Boundary')
title('Dataset 9 with SVM')
xlabel('X1')
ylabel('X2')

function res = gaussKernel(x, y)
    res = exp(-(norm(x-y))^2);
end

% Clearly there's something wrong with my code. I'm not sure if the
 lambdas
% are just incorrect or if i'm calculating the decision boundaries
% incorrectly, but I think if I were to guess, that using the decision
% boundary for C=1 makes more sense because the training data is
% definitely not all encompassing. C=0 overfits the data for sure and
 would
% not generalize well to other cases not seen in this data.
```

*Minimum found that satisfies the constraints.*

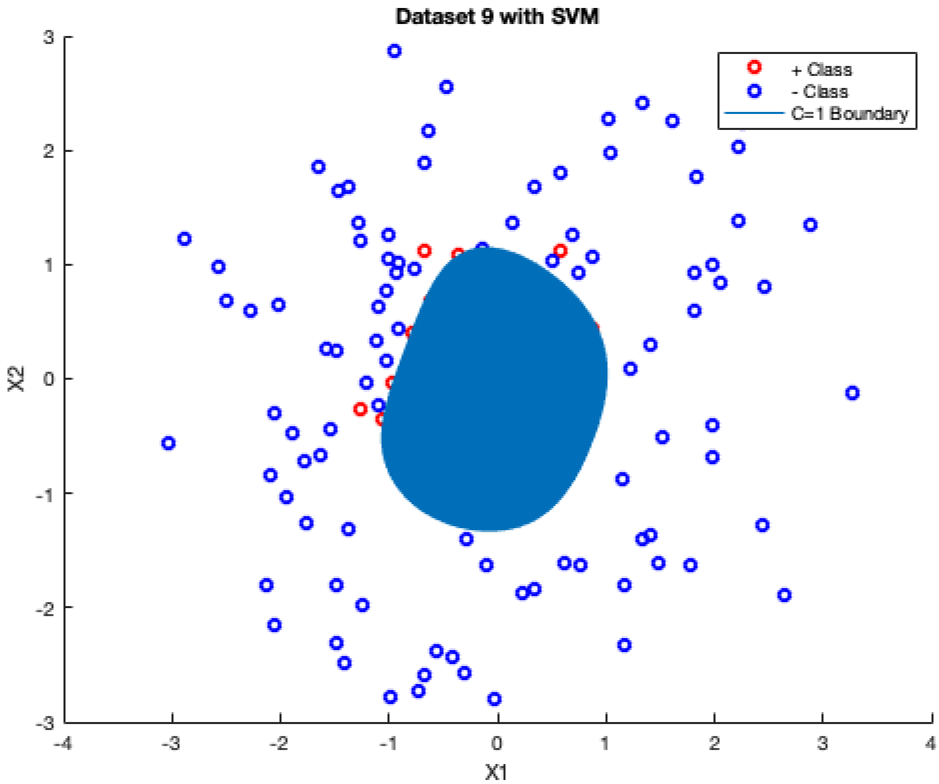
*Optimization completed because the objective function is non-decreasing in*
*feasible directions, to within the value of the optimality tolerance,*
*and constraints are satisfied to within the value of the constraint tolerance.*

*Number of Support Vectors C=0 are 30*

*Minimum found that satisfies the constraints.*

*Optimization completed because the objective function is non-decreasing in*
*feasible directions, to within the value of the optimality tolerance,*

```
and constraints are satisfied to within the value of the constraint
 tolerance.


Number of Support Vectors C=1 are 61
Warning: Ignoring extra legend entries.
```

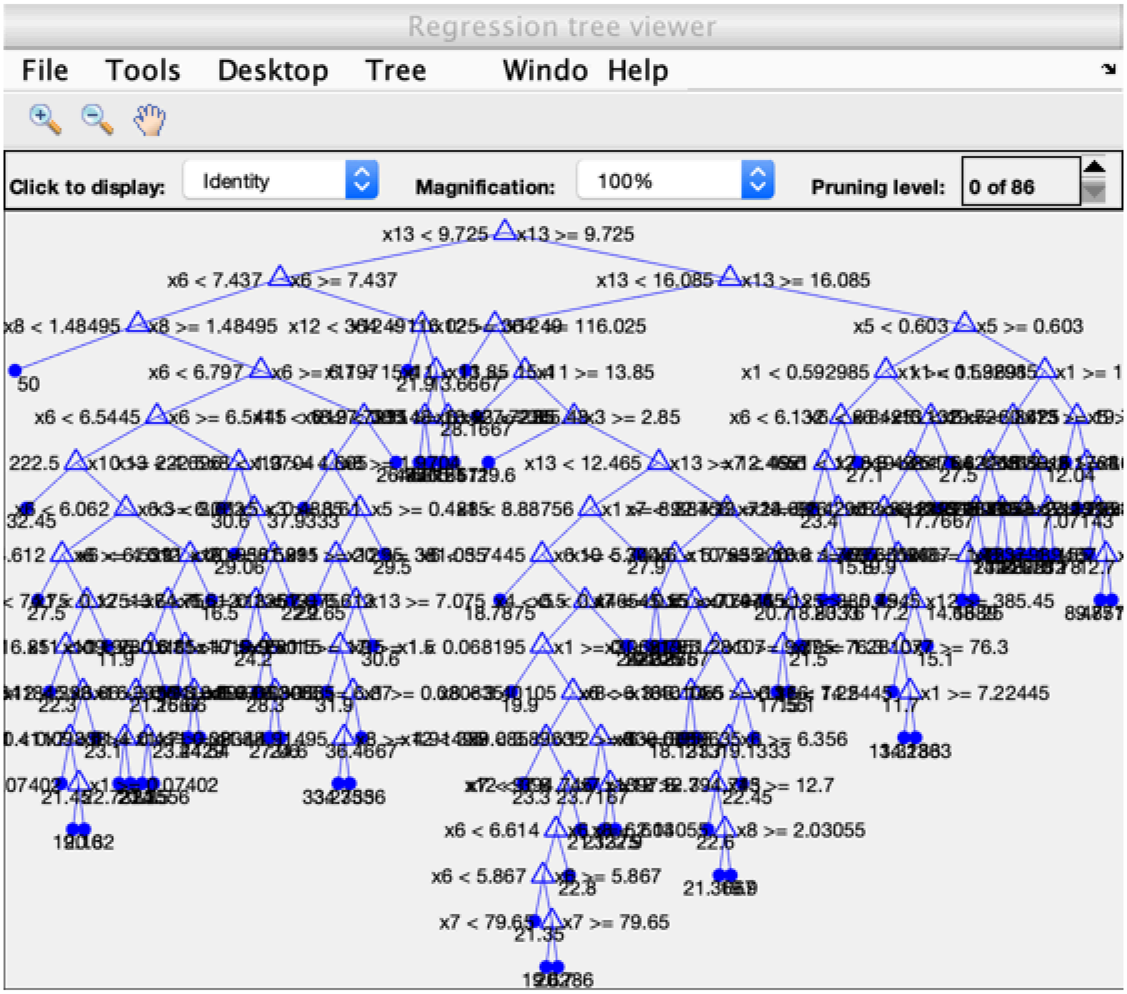


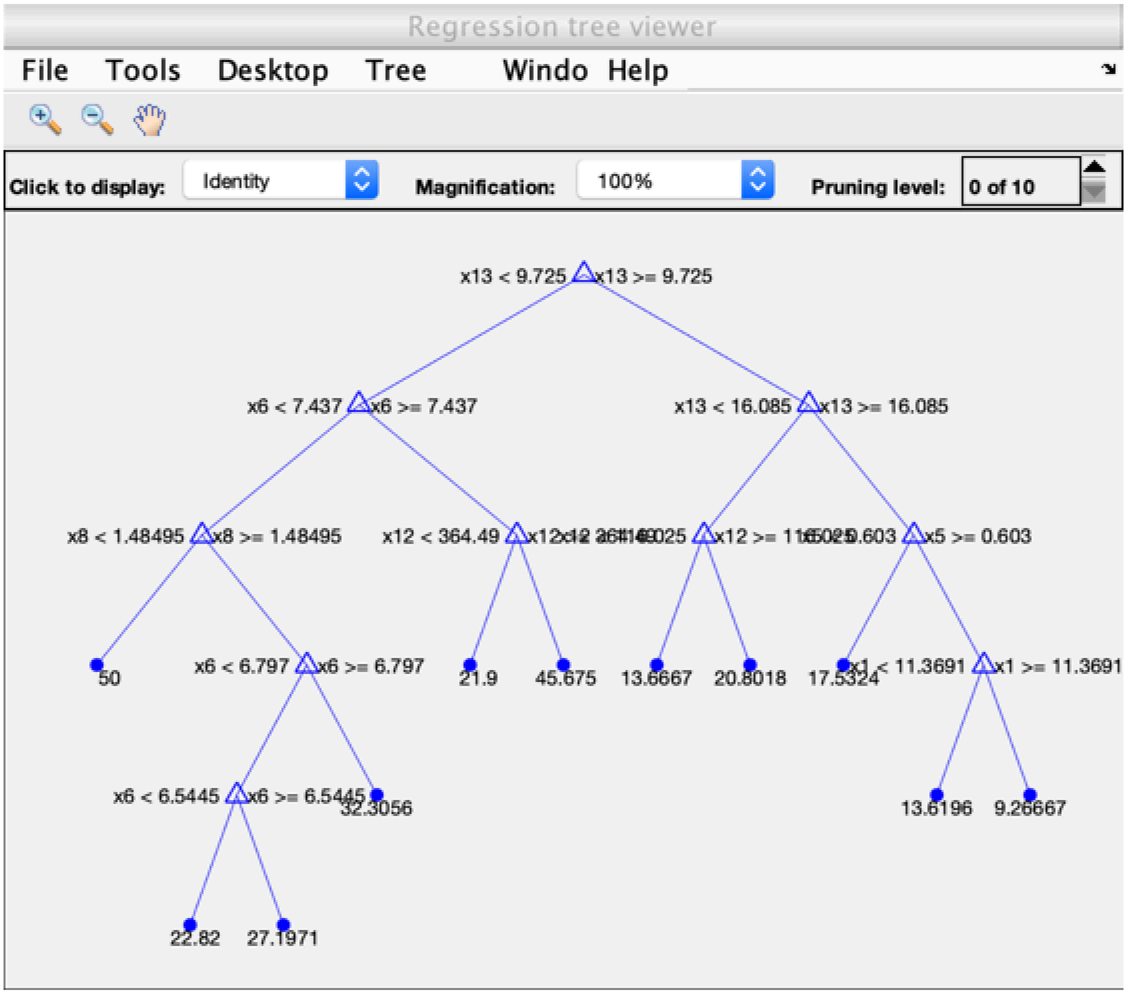

*Published with MATLAB® R2019a*

```matlab
% IDS/ACM/CS 158: Fundamentals of Statistical Learning
% PS5, Problem 4: Regression Trees for Boston Housing Data
% Author: Michael Li, mlli@caltech.edu
%-----------------------------------------------------------------------
clear;

% Boston housing Data
train = readmatrix('Boston_train.csv');
test = readmatrix('Boston_test.csv');
TO = fitrtree(train(:,1:end-1), train(:,end));
view(TO, 'Mode', 'graph')

train_preds = predict(TO, train(:,1:end-1));
test_preds = predict(TO, test(:, 1:end-1));

train_err = norm(train(:,end) - train_preds)^2 / length(train);
test_err = norm(test(:,end) - test_preds)^2 / length(test);

fprintf("Training Error for T_0: %s\n", train_err);
fprintf("Test Error for T_0: %s\n", test_err);
% The training error for T0 is 2.1707
% the test error is for T0 12.9867

alphas = linspace(0, 2, 21);
best = [];
lowest_err = 10^10;

% loop over each alpha and run loocv
for a = alphas
    err = 0;
    tree = [];
    % for each datapoint leave it out and test error
    for i = 1:length(train)
        x = repmat(train, 1);
        x(i,:) = [];
        test_x = train(i,:);

        tree = fitrtree(x(:,1:end-1), x(:,end));
        tree = prune(tree, 'Alpha', a);
        pred = predict(tree, test_x(1, 1:end-1));
        err = err + (test_x(1, end) - pred)^2;
    end

    err = err / length(train);

    % if error is lower than previous, replace
    if err < lowest_err
        best = a;
        lowest_err = err;
    end
end
```

```
% refit best alpha on all data
T_best = fitrtree(train(:,1:end-1), train(:,end));
T_best = prune(T_best, 'Alpha', best);
view(T_best, 'Mode', 'graph');

train_preds = predict(T_best, train(:,1:end-1));
test_preds = predict(T_best, test(:, 1:end-1));

train_err = norm(train(:,end) - train_preds)^2 / length(train);
test_err = norm(test(:,end) - test_preds)^2 / length(test);
fprintf("\nBest Alpha: %s\n", best);
fprintf("Training Error for T_best: %s\n", train_err);
fprintf("Test Error for T_best: %s\n", test_err);
% Optimal value of pruning parameter is .7
% The training error for T is 10.21203
% the test error for T is 9.607979
```

```
Training Error for T_0: 2.170719e+00
Test Error for T_0: 1.298670e+01

Best Alpha: 7.000000e-01
Training Error for T_best: 1.021203e+01
Test Error for T_best: 9.607979e+00
```


Regression tree viewer
File
Tools
Desktop
Tree
Windo
Help
Click to display:
Identity
Magnification:
100%
Pruning level:
0 of 86
x13 < 9.725
x13 >= 9.725
x6 < 7.437
x6 >= 7.437
x13 < 16.085
x13 >= 16.085
x8 < 1.48495
x8 >= 1.48495
x5 < 0.603
x5 >= 0.603
50
x6 < 6.797
x1 < 0.592985
x6 < 6.5445
x13 < 12.465
x13 >= 7.075
x1 >= 7.22445
x8 >= 2.03055
x6 < 6.614
x6 < 5.867
x7 < 79.65
x7 >= 79.65

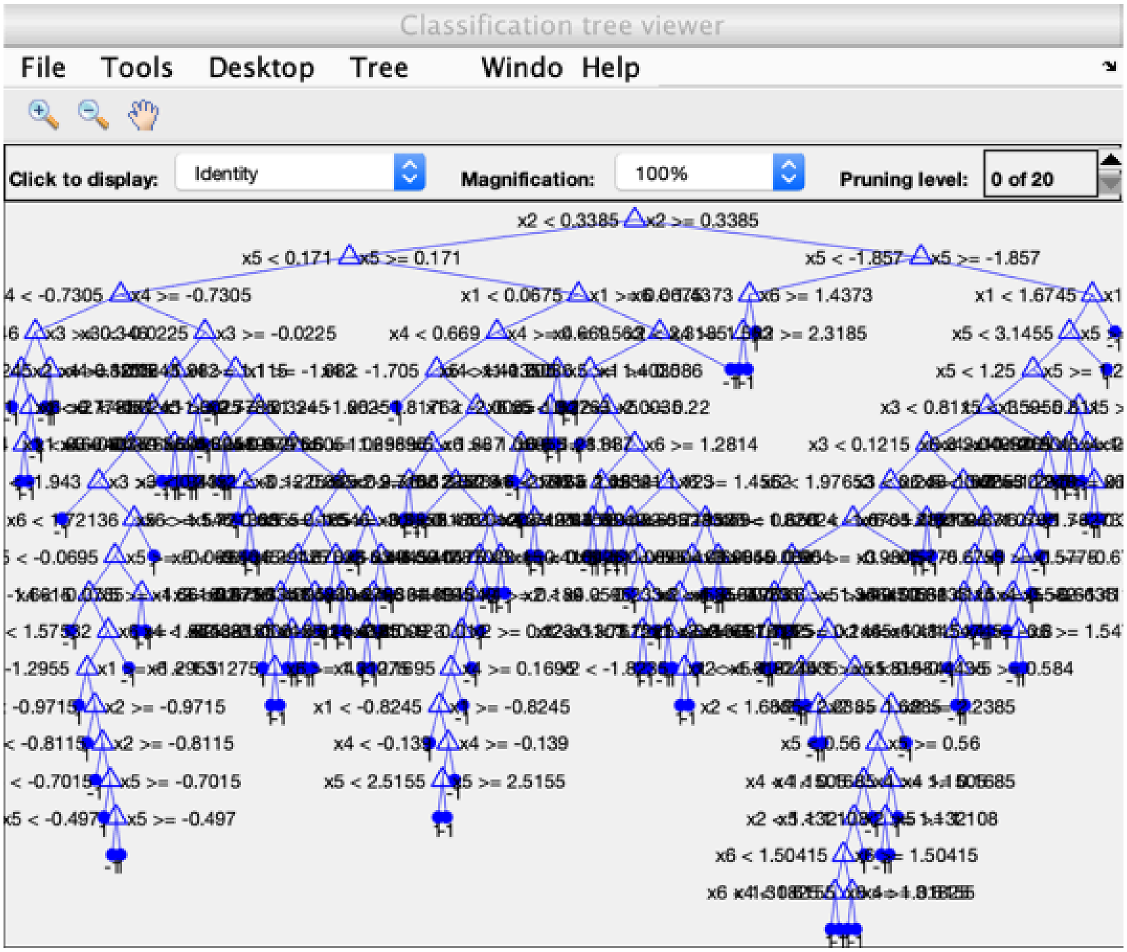


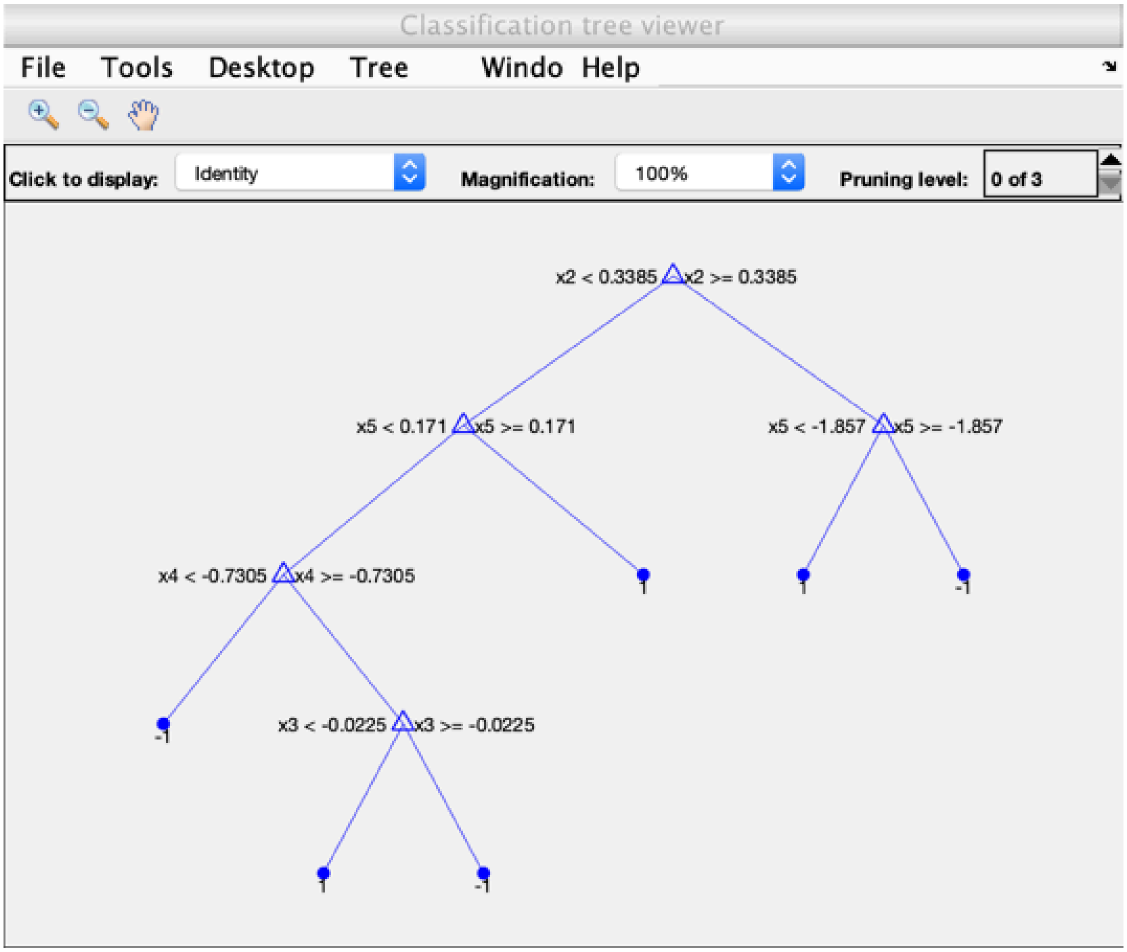

*Published with MATLAB® R2019a*

```matlab
% IDS/ACM/CS 158: Fundamentals of Statistical Learning
% PS5, Problem 5: Classification Trees for Stock Market Data
% Author: Michael Li, mlli@caltech.edu
%-----------------------------------------------------------------------
clear;

% Stock market Data
train = readmatrix('stock_market_train.csv');
test = readmatrix('stock_market_test.csv');
TO = fitctree(train(:,1:end-1), train(:,end));
view(TO, 'Mode', 'graph')

train_preds = predict(TO, train(:,1:end-1));
test_preds = predict(TO, test(:, 1:end-1));

train_err = mean(train(:,end) ~= train_preds);
test_err = mean(test(:,end) ~= test_preds);

fprintf("Training Error for T_0: %s\n", train_err);
fprintf("Test Error for T_0: %s\n", test_err);

% The training error is .118
% the test error is .476

% loop over each alpha and run loocv
alphas = linspace(0, .04, 41);
best = [];
lowest_err = 1;

for a = alphas
    err = 0;
    tree = [];
    % for each datapoint leave it out and test error
    for i = 1:length(train)
        x = repmat(train, 1);
        x(i,:) = [];
        test_x = train(i,:);

        tree = fitctree(x(:,1:end-1), x(:,end));
        tree = prune(tree, 'Alpha', a);
        pred = predict(tree, test_x(1, 1:end-1));
        err = err + (pred ~= test_x(1, end));
    end

    err = err / length(train);

    % if error is lower than previous, replace
    if err < lowest_err
        best = a;
        lowest_err = err;
    end
end
```

```
% refit best alpha on all data
T_best = fitctree(train(:,1:end-1), train(:,end));
T_best = prune(T_best, 'Alpha', best);
view(T_best, 'Mode', 'graph');

train_preds = predict(T_best, train(:,1:end-1));
test_preds = predict(T_best, test(:, 1:end-1));

train_err = mean(train(:,end) ~= train_preds);
test_err = mean(test(:,end) ~= test_preds);
fprintf("\nBest Alpha: %s\n", best);
fprintf("Training Error for T_best: %s\n", train_err);
fprintf("Test Error for T_best: %s\n", test_err);
% Optimal value of pruning parameter is .01
% The training error for T is .401
% the test error for T is .584
```

```
Training Error for T_0: 1.180000e-01
Test Error for T_0: 4.760000e-01

Best Alpha: 1.000000e-02
Training Error for T_best: 4.010000e-01
Test Error for T_best: 5.840000e-01
```


Classification tree viewer
File Tools Desktop Tree Windo Help
Click to display:
Identity
Magnification:
100%
Pruning level:
0 of 20
x2 < 0.3385
x2 >= 0.3385
x5 < 0.171
x5 >= 0.171
x5 < -1.857
x5 >= -1.857
x4 >= -0.7305
x3 >= -0.0225
x4 < 0.669
x1 < 1.6745
x5 < 3.1455
x5 < 1.25
x1 < -0.8245
x4 < -0.139
x4 >= -0.139
x5 < 2.5155
x5 >= 2.5155
x2 >= -0.9715
x2 >= -0.8115
x5 >= -0.7015
x5 >= -0.497
x6 < 1.50415




*Published with MATLAB® R2019a*