## Contents



## Classification of digit 1 from the rest using Linear Regression

```
close all;
clear;
clc;
digitTobeClassified = 1;
```

## Read the training data from txt file

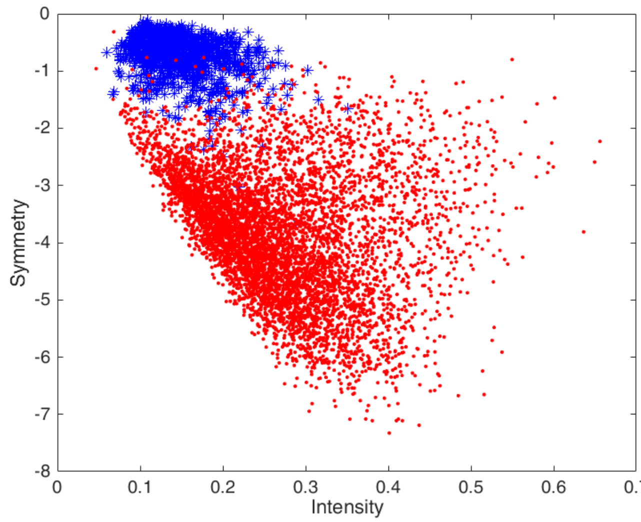
```
fileID = fopen('features_train.txt','r');        % open file
formatSpec = '%f %f %f';                         % specifying the reading format
sizeA=[3 inf];                                   % specifying the size of the data matrix
training_data = fscanf(fileID,formatSpec,sizeA); % reading the data matrix
fclose(fileID);

% Getting the size of the matrix data
training_data_length=length(training_data);
```

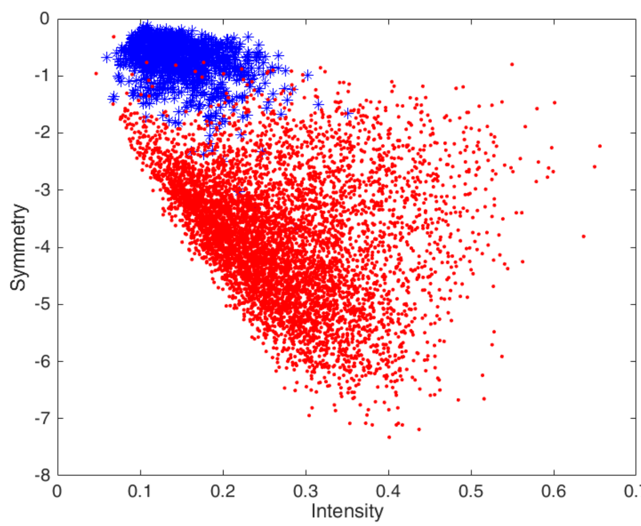
## Read the validation data from txt file

```
fileID = fopen('features_test.txt','r');        % open file
formatSpec = '%f %f %f';                        % specifying the reading format
sizeA=[3 inf];                                  % specifying the size of the data matrix
testing_data = fscanf(fileID,formatSpec,sizeA); % reading the data matrix
fclose(fileID);

% Getting the size of the data matrix
testing_data_length = length(testing_data);
```

## Plot the distribution of digitTobeClassified and the rest

```
figure(1)
logicalIdForDigitTobeClassified = (training_data(1,:) == digitTobeClassified);
digitTobeClassifiedData = training_data(:,logicalIdForDigitTobeClassified);
otherData = training_data(:,~logicalIdForDigitTobeClassified);
plot(digitTobeClassifiedData(2,:),digitTobeClassifiedData(3,:),'b*');
hold on;
plot(otherData(2,:),otherData(3,:),'r.');
hold off;
xlabel('Intensity')
ylabel('Symmetry')

% Create duplicate figure 2
copyobj(figure(1),0);
```




### Linear Regression Model Parameters

Generate the data matrix A

```
A = [training_data(2,:)' training_data(3,:)' ones(length(training_data),1)];

% Generate the label matrix b
b = -ones(length(training_data),1);
b(training_data(1,:) == digitTobeClassified) = 1;
```

### Least Square optimal solution

```
fprintf('******************************************************************\n');
fprintf('### Least Square Optimal Solution using Normal Equation: \n');
fprintf('******************************************************************\n');
xStarLeastSqr = (A'*A)\A'*b %#ok
```

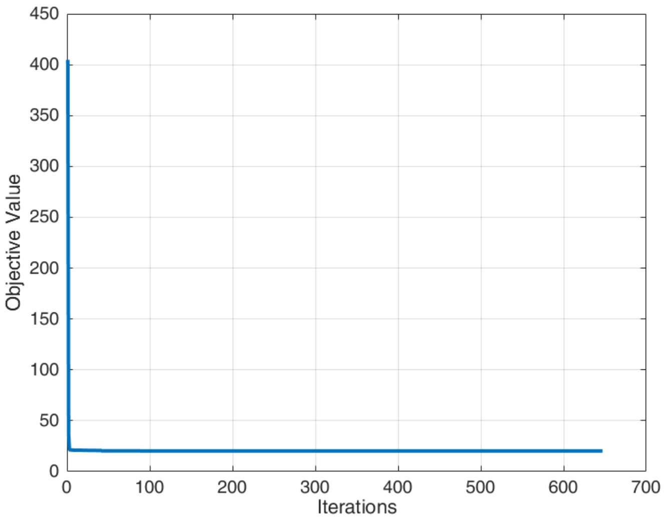
```
******************************************************************
### Least Square Optimal Solution using Normal Equation:
******************************************************************

xStarLeastSqr =

   -0.9675
    0.3009
    0.5459
```

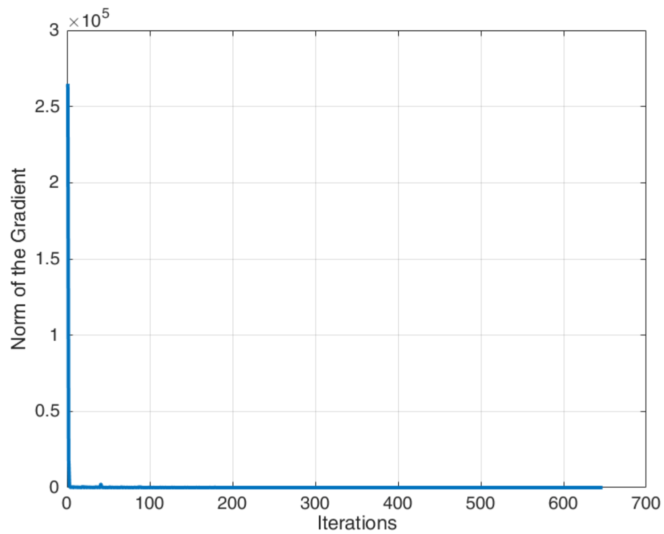
### Solve the optimization problem to reduce the errornorm using steepest descent

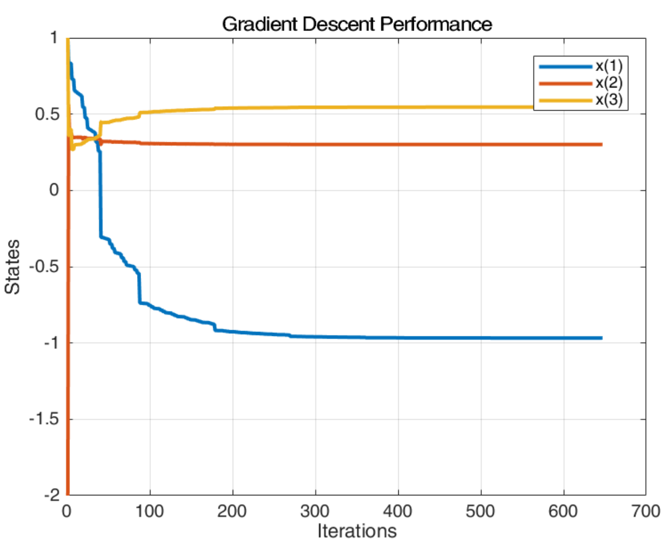
```
fprintf('****************************************************************\n');
fprintf('### Starting Steepest descent method with Armijo step size rule: \n');
fprintf('****************************************************************\n');

% Initial guess for the algorithm
x0 = [1;-2;1];

% Objective Function
f = @(x) (0.5*norm(A*x-b));

% Gradient Function
g = @(x) (A'*(A*x-b));

% Steepest Descent with Armijo stepsize rule
x = x0;
sigma = 0.00001;
beta = 0.1;
s = 1;
epsilon = 1e-3;

nFeval = 1;
r = 1;
MAX_ITER = 10000;
obj = f(x);
gradient = g(x);

objForPlotting = zeros(1,MAX_ITER);
objForPlotting(r) = obj;
GradientNormForPlotting = zeros(1,MAX_ITER);
GradientNormForPlotting(r) = norm(gradient);
stateForPlotting = zeros(3,MAX_ITER);
stateForPlotting(:,r) = x0;

while norm(gradient) > epsilon && r < MAX_ITER
    % Steepest descent direction i.e. -grad
    direction = -gradient;

    % Start with stepsize = s
    alpha = s;
    newobj = f(x + alpha*direction);
    nFeval = nFeval+1;

    % Armijo stepsize rule check i.e. do we have sufficient descent?
    while (newobj-obj) > alpha*sigma*gradient'*direction
        alpha = alpha*beta;
        newobj = f(x + alpha*direction);
        nFeval = nFeval+1;
    end

    % Update the next state
    x = x + alpha*direction;

    % Print Status every 45 iterations
    if(mod(r,45)==1)
        fprintf('Iter:%5.0f | Feval:%5.0f | OldObj:%5.5e | NewObj:%5.5e | ReductionInObj:%5.5e | GradientNorm:%5.2f | x(1):%.4d | x(2):%.4d | x(3):%.4d\n',...
            r,nFeval,obj,newobj,obj-newobj,norm(gradient),x);
    end
    obj = newobj;
    gradient = g(x);
    r = r+1;
    stateForPlotting(:,r) = x;
    objForPlotting(r) = obj;
    GradientNormForPlotting(r) = norm(gradient);
end

% Print the final iteration
fprintf('Iter:%5.0f | Feval:%5.0f | OldObj:%5.5e | NewObj:%5.5e | ReductionInObj:%5.5e | GradientNorm:%5.2f | x(1):%.4d | x(2):%.4d | x(3):%.4d\n',...
    r,nFeval,obj,newobj,obj-newobj,norm(gradient),x);

% Check MAX_ITER
if r == MAX_ITER
    fprintf('Maximum iteration limit reached.\n');
end

% Plot the reduction in gradient norm and objective reduction
figure(3)
plot(1:r,objForPlotting(1:r),'LineWidth',2);
xlabel('Iterations');ylabel('Objective Value');grid on;
figure(4)
plot(1:r,GradientNormForPlotting(1:r),'LineWidth',2);
xlabel('Iterations');ylabel('Norm of the Gradient');grid on;
figure(5)
plot(1:r,stateForPlotting(:,1:r),'LineWidth',2)
xlabel('Iterations');ylabel('States');grid on;
legend('x(1)','x(2)','x(3)');
title('Gradient Descent Performance')

% Plot the boundry
figure(1)
hold on;
xStarGradientDescent = x %#ok
```

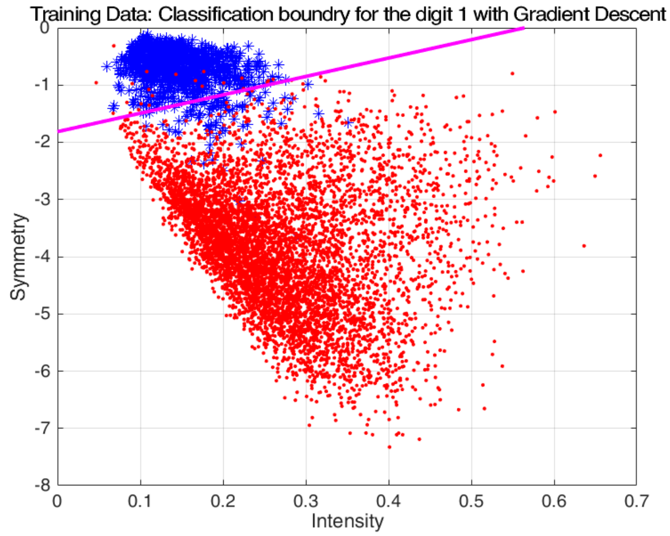
```
equationOflineGD = @(a1,a2) (xStarGradientDescent(1)*a1 + xStarGradientDescent(2)*a2 + xStarGradientDescent(3));
currentAxes = gca;
h = fimplicit(equationOflineGD,[currentAxes.XLim,currentAxes.YLim]);
title(sprintf('Training Data: Classification boundry for the digit %d with Gradient Descent',digitTobeClassified));
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;
% Visulize how the line changes as algorithm progresses
figure(2)
hold on;
currentAxes = gca;
for i = 1:r
    if(mod(i,20)==1 || i==1)
        eqOflineGD = @(a1,a2) (stateForPlotting(1,i)*a1 + stateForPlotting(2,i)*a2 + stateForPlotting(3,i));
        h = fimplicit(eqOflineGD,[currentAxes.XLim,currentAxes.YLim]);
        set(h,'LineWidth',2,'Color','green','LineStyle','--');
        hold on;
    end
end
h = fimplicit(equationOflineGD,[currentAxes.XLim,currentAxes.YLim]);
set(h,'LineWidth',3,'Color','magenta');
title(sprintf('Training Data: Change in the classification boundry with Gradient Descent'));
grid on;
hold off;
```

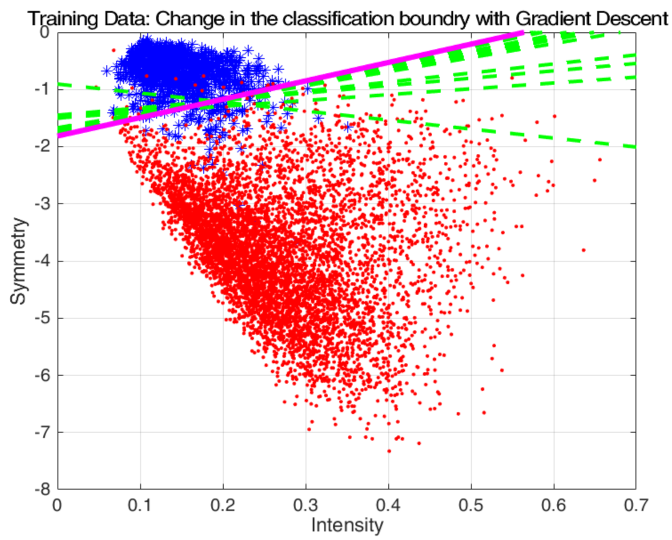
```
*****************************************************************
### Starting Steepest descent method with Armijo step size rule:
*****************************************************************
Iter:    1 | Feval:    7 | OldObj:4.04761e+02 | NewObj:3.65969e+01 | ReductionInObj:3.68164e+02 | GradientNorm:264884.85 | x(1):8.2407e-01 | x(2):5.6419e-01 | x(3):
Iter:   46 | Feval:  250 | OldObj:2.03822e+01 | NewObj:2.03817e+01 | ReductionInObj:5.39858e-04 | GradientNorm:91.81 | x(1):-3.1471e-01 | x(2):3.2027e-01 | x(3):4.4
Iter:   91 | Feval:  491 | OldObj:2.02733e+01 | NewObj:2.02732e+01 | ReductionInObj:1.09520e-04 | GradientNorm:42.27 | x(1):-7.3955e-01 | x(2):3.0768e-01 | x(3):5.1
Iter:  136 | Feval:  732 | OldObj:2.02638e+01 | NewObj:2.02638e+01 | ReductionInObj:3.31888e-05 | GradientNorm: 6.83 | x(1):-8.2734e-01 | x(2):3.0487e-01 | x(3):5.2
Iter:  181 | Feval:  973 | OldObj:2.02587e+01 | NewObj:2.02587e+01 | ReductionInObj:2.04505e-06 | GradientNorm: 2.46 | x(1):-9.1854e-01 | x(2):3.0239e-01 | x(3):5.3
Iter:  226 | Feval: 1215 | OldObj:2.02583e+01 | NewObj:2.02583e+01 | ReductionInObj:2.03938e-06 | GradientNorm: 5.78 | x(1):-9.3726e-01 | x(2):3.0175e-01 | x(3):5.4
Iter:  271 | Feval: 1456 | OldObj:2.02581e+01 | NewObj:2.02581e+01 | ReductionInObj:4.30185e-07 | GradientNorm: 2.69 | x(1):-9.5691e-01 | x(2):3.0117e-01 | x(3):5.4
Iter:  316 | Feval: 1697 | OldObj:2.02581e+01 | NewObj:2.02581e+01 | ReductionInObj:3.05987e-08 | GradientNorm: 0.33 | x(1):-9.6098e-01 | x(2):3.0104e-01 | x(3):5.4
Iter:  361 | Feval: 1938 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:3.42561e-06 | GradientNorm: 7.74 | x(1):-9.6520e-01 | x(2):3.0093e-01 | x(3):5.4
Iter:  406 | Feval: 2181 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:6.33058e-10 | GradientNorm: 0.08 | x(1):-9.6600e-01 | x(2):3.0090e-01 | x(3):5.4
Iter:  451 | Feval: 2422 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:8.35421e-11 | GradientNorm: 0.03 | x(1):-9.6695e-01 | x(2):3.0087e-01 | x(3):5.4
Iter:  496 | Feval: 2663 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:5.64551e-10 | GradientNorm: 0.10 | x(1):-9.6715e-01 | x(2):3.0087e-01 | x(3):5.4
Iter:  541 | Feval: 2904 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:1.23279e-10 | GradientNorm: 0.05 | x(1):-9.6735e-01 | x(2):3.0086e-01 | x(3):5.4
Iter:  586 | Feval: 3145 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:1.04730e-10 | GradientNorm: 0.00 | x(1):-9.6739e-01 | x(2):3.0086e-01 | x(3):5.4
Iter:  631 | Feval: 3386 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:1.02283e-11 | GradientNorm: 0.00 | x(1):-9.6743e-01 | x(2):3.0086e-01 | x(3):5.4
Iter:  647 | Feval: 3468 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:0.00000e+00 | GradientNorm: 0.00 | x(1):-9.6744e-01 | x(2):3.0086e-01 | x(3):5.4

xStarGradientDescent =

   -0.9674
    0.3009
    0.5459
```


Objective Value
Iterations


Norm of the Gradient
Iterations


Gradient Descent Performance
x(1)
x(2)
x(3)
States
Iterations



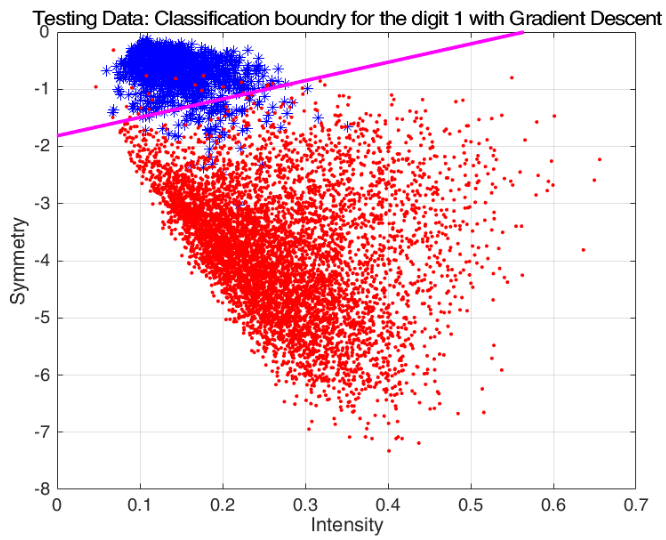

**Use the optimal model derived from gradient descent to classify a digit in the test/validation data**

Generate the data matrix Av for validation

```
Av = [testing_data(2,:)' testing_data(3,:)' ones(testing_data_length,1)];

% Generate the label matrix bv for validation
bTrue = -ones(testing_data_length,1);
bTrue(testing_data(1,:) == digitTobeClassified) = 1;

% Compute Av*xStarGradientDescent
bClassifierTest = Av*xStarGradientDescent;
bTest = sign(bClassifierTest);

% Find out quality measures for the classifier
truePositive = sum((bTest == 1) & (bTrue == 1))%#ok
falsePositive = sum((bTest == 1) & (bTrue == -1))%#ok
falseNegative = sum((bTest == -1) & (bTrue == 1))%#ok
trueNegative = sum((bTest == -1) & (bTrue == -1))%#ok
truePositiveRate = truePositive/(truePositive+falseNegative) %#ok Sensitivity
trueNegativeRate = trueNegative/(trueNegative+falsePositive) %#ok Specificity

% Correctly classified labels
accuracy = (truePositive+trueNegative)/(truePositive+falsePositive+falseNegative+trueNegative)%#ok

% Plot the results in figure 5
figure(6);
logicalIdForDigitTobeClassified = (training_data(1,:) == digitTobeClassified);
digitTobeClassifiedData = training_data(:,logicalIdForDigitTobeClassified);
otherData = training_data(:,~logicalIdForDigitTobeClassified);
plot(digitTobeClassifiedData(2,:),digitTobeClassifiedData(3,:),'b*');
hold on;
plot(otherData(2,:),otherData(3,:),'r.');
xlabel('Intensity')
ylabel('Symmetry')
```

```
currentAxes = gca;
h = fimplicit(equationOflineGD,[currentAxes.XLim,currentAxes.YLim]);
title(sprintf('Testing Data: Classification boundry for the digit %d with Gradient Descent',digitTobeClassified));
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;
```

```
truePositive =

   226


falsePositive =

     8


falseNegative =

    38


trueNegative =

        1735


truePositiveRate =

    0.8561


trueNegativeRate =

    0.9954


accuracy =

    0.9771
```



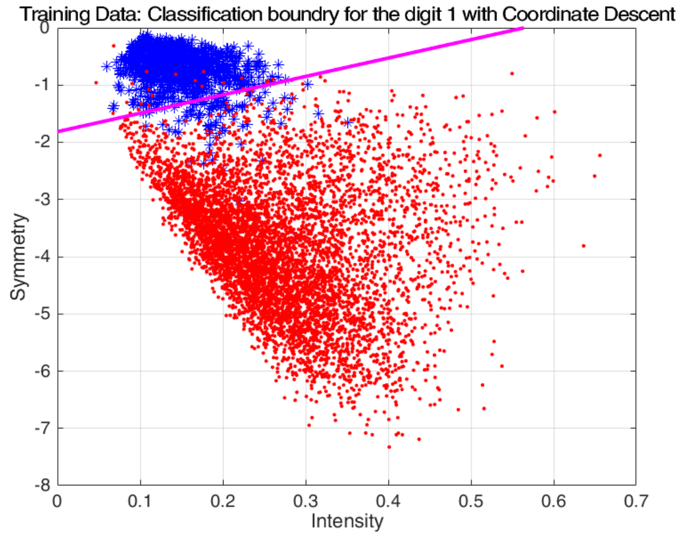

### Solve the optimization problem to reduce the errornorm using coordinate discent

```
fprintf('*****************************************************************\n');
fprintf('### Starting coordinate descent: \n');
fprintf('*****************************************************************\n');

% Initial guess for the algorithm
x0 = [1;-2;1];

% Objective Function
f = @(x) (0.5*norm(A*x-b));

% Gradient Function
g = @(x) (A'*(A*x-b));

% Coordinate Descent
r = 1;
iter = r;
alpha = 0.01;
```

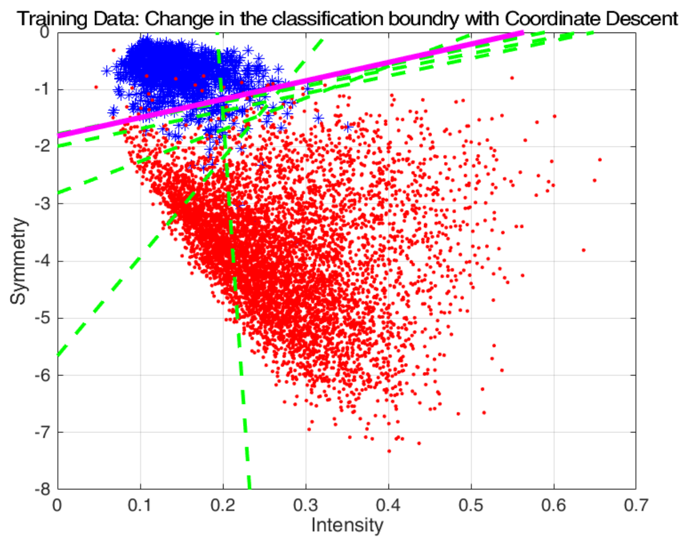
```
MAX_ITER = 100000;
x = zeros(3,MAX_ITER);
epsilon = sqrt(eps);

x(:,r) = x0;
obj = f(x(:,r));
gradient = g(x(:,r));

objForPlotting = zeros(1,MAX_ITER);
objForPlotting(r) = obj;
GradientNormForPlotting = zeros(1,MAX_ITER);
GradientNormForPlotting(r) = norm(gradient);
stateForPlotting = zeros(3,MAX_ITER);
stateForPlotting(:,r) = x(:,r);
prevState = stateForPlotting(:,r);
ReductionInObj = 1;

% Stopping Criteria for the algorithm
while norm(ReductionInObj) > epsilon
    for i = 1:3
        for j = 1:3
            if ~isequal(i,j)
                % If i~=j then just copy the values.
                x(j,r+1) = x(j,r);
            else
                % That means i==j
                % Remove the ith/jth column and call it Aj
                Aj = A;
                Aj(:,i)=[];
                xj = x(:,r);
                xj(i) = [];

                % Update the x(i,r+1)
                x(i,r+1) = A(:,i)'*A(:,i)\A(:,i)'*(b-Aj*xj);
            end
        end
        % Increment the r
        r = r + 1;

        % If we reach maximum limit then break the loop
        if r == MAX_ITER
            break;
        end
    end

    % Increment the iteration count and save the plotting state after
    % iteration
    prevState = stateForPlotting(:,iter);
    iter = iter + 1;
    stateForPlotting(:,iter) = x(:,r);
    GradientNormForPlotting(iter) = norm(g(x(:,r)));
    updatedState = stateForPlotting(:,iter);
    OldObj = f(prevState);
    NewObj = f(updatedState);
    ReductionInObj = OldObj-NewObj;
    objForPlotting(iter) = NewObj;

    % Print Status
    if(mod(iter,10)==1)
        fprintf('Iter:%5.0f | OldObj:%5.5e | NewObj:%5.5e | ReductionInObj:%5.5e | x(1):%.4d | x(2):%.4d | x(3):%.4d\n',...
            iter,OldObj,NewObj,ReductionInObj,updatedState);
    end

    % Check MAX_ITER break the outer loop
    if r == MAX_ITER
        fprintf('Maximum iteration limit reached.\n');
        break;
    end
end

xStarCoordinateDescent = stateForPlotting(:,iter) %#ok

% Plot the boundry
figure(7)
plot(digitTobeClassifiedData(2,:),digitTobeClassifiedData(3,:),'b*');
hold on;
plot(otherData(2,:),otherData(3,:),'r.');
xlabel('Intensity')
ylabel('Symmetry')
equationOflineCD = @(a1,a2) (xStarCoordinateDescent(1)*a1 + xStarCoordinateDescent(2)*a2 + xStarCoordinateDescent(3));
currentAxes = gca;
h = fimplicit(equationOflineCD,[currentAxes.XLim,currentAxes.YLim]);
title(sprintf('Training Data: Classification boundry for the digit %d with Coordinate Descent',digitTobeClassified));
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;
% Visulize how the line changes as algorithm progresses
% Create duplicate figure 8
copyobj(figure(7),0);
hold on;
currentAxes = gca;
```

```
for i = 1:iter
    if(mod(i,10)==1 || i==1)
        eqOflineCD = @(a1,a2) (stateForPlotting(1,i)*a1 + stateForPlotting(2,i)*a2 + stateForPlotting(3,i));
        h = fimplicit(eqOflineCD,[currentAxes.XLim,currentAxes.YLim]);
        set(h,'LineWidth',2,'Color','green','LineStyle','--');
        hold on;
    end
end
h = fimplicit(equationOflineCD,[currentAxes.XLim,currentAxes.YLim]);
set(h,'LineWidth',3,'Color','magenta');
title(sprintf('Training Data: Change in the classification boundry with Coordinate Descent'));
grid on;
hold off;

% Plot the reduction in gradient norm and objective reduction
figure(3)
hold on;
plot(1:iter,objForPlotting(1:iter),'LineWidth',2);
legend('Gradient Descent','Coordinate Descent');
title('Algorithm Performance')
fig3axis = gca;
set(fig3axis,'XLim',[0 120])

figure(4)
hold on;
plot(1:iter,GradientNormForPlotting(1:iter),'LineWidth',2);
legend('Gradient Descent','Coordinate Descent');
title('Algorithm Performance')
fig4axis = gca;
set(fig4axis,'XLim',[0 120],'YLim',[0 300000/4])

figure(9)
plot(1:iter,stateForPlotting(:,1:iter),'LineWidth',2)
xlabel('Iterations');ylabel('States');grid on;
legend('x(1)','x(2)','x(3)');
title('Coordinate Descent Performance')
```

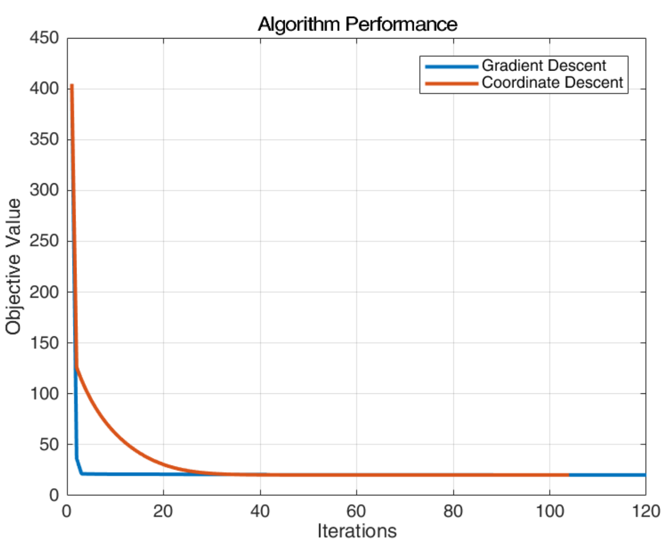
```
*****************************************************************
### Starting coordinate descent:
*****************************************************************
Iter:   11 | OldObj:6.11981e+01 | NewObj:5.65403e+01 | ReductionInObj:4.65785e+00 | x(1):-1.6240e+01 | x(2):-7.9865e-02 | x(3):3.1367e+00
Iter:   21 | OldObj:3.05074e+01 | NewObj:2.89247e+01 | ReductionInObj:1.58270e+00 | x(1):-5.8450e+00 | x(2):3.3810e-01 | x(3):1.9139e+00
Iter:   31 | OldObj:2.17061e+01 | NewObj:2.14122e+01 | ReductionInObj:2.93970e-01 | x(1):-2.0052e+00 | x(2):3.6182e-01 | x(3):1.0175e+00
Iter:   41 | OldObj:2.03825e+01 | NewObj:2.03537e+01 | ReductionInObj:2.88073e-02 | x(1):-1.0146e+00 | x(2):3.3022e-01 | x(3):6.5787e-01
Iter:   51 | OldObj:2.02669e+01 | NewObj:2.02649e+01 | ReductionInObj:1.96932e-03 | x(1):-8.8257e-01 | x(2):3.1041e-01 | x(3):5.5683e-01
Iter:   61 | OldObj:2.02590e+01 | NewObj:2.02588e+01 | ReductionInObj:1.64396e-04 | x(1):-9.1602e-01 | x(2):3.0295e-01 | x(3):5.3995e-01
Iter:   71 | OldObj:2.02582e+01 | NewObj:2.02582e+01 | ReductionInObj:2.63490e-05 | x(1):-9.4839e-01 | x(2):3.0098e-01 | x(3):5.4150e-01
Iter:   81 | OldObj:2.02581e+01 | NewObj:2.02581e+01 | ReductionInObj:4.08474e-06 | x(1):-9.6252e-01 | x(2):3.0070e-01 | x(3):5.4414e-01
Iter:   91 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:4.44482e-07 | x(1):-9.6679e-01 | x(2):3.0076e-01 | x(3):5.4543e-01
Iter:  101 | OldObj:2.02580e+01 | NewObj:2.02580e+01 | ReductionInObj:3.36636e-08 | x(1):-9.6762e-01 | x(2):3.0082e-01 | x(3):5.4584e-01

xStarCoordinateDescent =

   -0.9677
    0.3008
    0.5459
```



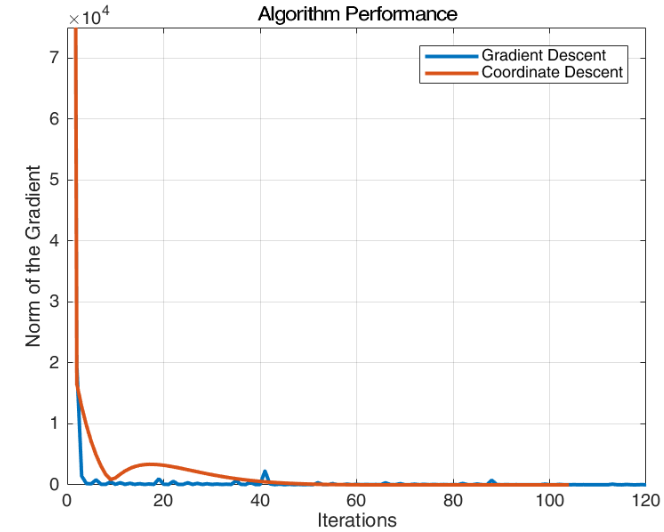
Training Data: Classification boundry for the digit 1 with Coordinate Descent
Symmetry
Intensity

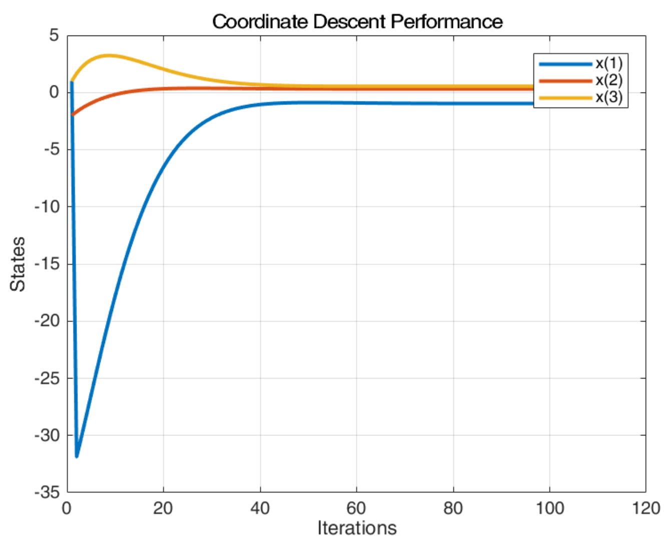



Training Data: Change in the classification boundry with Coordinate Descent
Symmetry
Intensity

Algorithm Performance
Gradient Descent
Coordinate Descent
Objective Value
Iterations



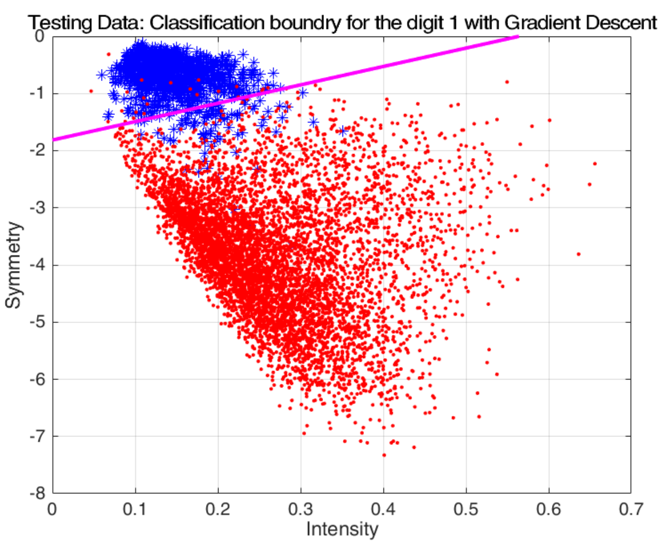

### Use the optimal model derived from coordinate descent to classify a digit in the test/validation data

Generate the data matrix Av for validation

```
Av = [testing_data(2,:)' testing_data(3,:)' ones(testing_data_length,1)];

% Generate the label matrix bv for validation
bTrue = -ones(testing_data_length,1);
bTrue(testing_data(1,:) == digitTobeClassified) = 1;

% Compute Av*xStarGradientDescent
bClassifierTest = Av*xStarCoordinateDescent;
bTest = sign(bClassifierTest);

% Find out quality measures for the classifier
truePositive = sum((bTest == 1) & (bTrue == 1))%#ok
falsePositive = sum((bTest == 1) & (bTrue == -1))%#ok
falseNegative = sum((bTest == -1) & (bTrue == 1))%#ok
trueNegative = sum((bTest == -1) & (bTrue == -1))%#ok
truePositiveRate = truePositive/(truePositive+falseNegative) %#ok Sensitivity
trueNegativeRate = trueNegative/(trueNegative+falsePositive) %#ok Specificity

% Correctly classified labels
accuracy = (truePositive+trueNegative)/(truePositive+falsePositive+falseNegative+trueNegative)%#ok

% Plot the results in figure 5
figure(6);
logicalIdForDigitTobeClassified = (training_data(1,:) == digitTobeClassified);
digitTobeClassifiedData = training_data(:,logicalIdForDigitTobeClassified);
otherData = training_data(:,~logicalIdForDigitTobeClassified);
plot(digitTobeClassifiedData(2,:),digitTobeClassifiedData(3,:),'b*');
hold on;
plot(otherData(2,:),otherData(3,:),'r.');
xlabel('Intensity')
ylabel('Symmetry')
```

```
currentAxes = gca;
h = fimplicit(equationOflineCD,[currentAxes.XLim,currentAxes.YLim]);
title(sprintf('Testing Data: Classification boundry for the digit %d with Gradient Descent',digitTobeClassified));
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;
```

```
truePositive =

   226


falsePositive =

     8


falseNegative =

    38


trueNegative =

        1735


truePositiveRate =

    0.8561


trueNegativeRate =

    0.9954


accuracy =

    0.9771
```




### Solve the classification problem by Support vector machine

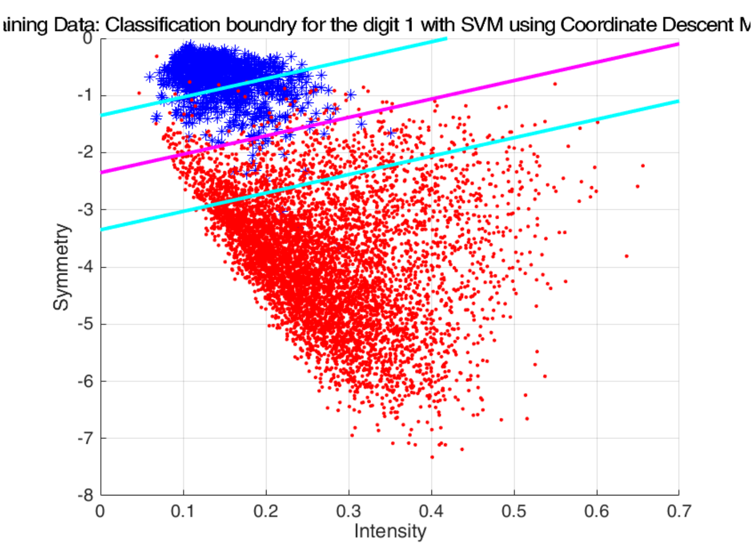
Optimization algorithm used here is coordinate descent

```
fprintf('*****************************************************************\n');
fprintf('### Starting Support Vector Machine using Coordinate Descent: \n');
fprintf('*****************************************************************\n');

% Fix c initially
c = 100;

% Start with r = 1
r = 1;

% Number of iterations are 1 initially
iter = r;

% Maximum iterations
MAX_ITER = 10000;
```

```
% Allocate memory for the primal variable
x = zeros(3,MAX_ITER);

% Allocate memory for fullgradient
fullgradient = zeros(training_data_length,MAX_ITER);

% Allocate memory for dual varialbe
lambda = zeros(training_data_length,MAX_ITER);

% Stopping critera
epsilon = 1e-2;

% Generate the random vector lambda for initial state
% 0 <= lambda <= c
lambda(:,r) = zeros(training_data_length,1);

% Dual objective to be maximized
dualObj = @(xt,lambda) sum(lambda) - 0.5*norm(xt)^2;

% Projection on to 0 <= x <= c
proj = @(x) max(min(x,c),0);

% ith Gradient Function
ithgrad = @(i,calulatedPrimal) 1-(b(i)*A(i,:)*calulatedPrimal);

% Full gradient
fullgrad = @(x) 1 - b.*A*x;

% Dual to primal variable update equation
primalClaculate = @(lambdaIn) sum((lambdaIn.*b).*A);

% Initial x0 is given by sum of all lambda(i)*b(i)*A(i,:)
% i.e. Update the initial guess for the primal variable
x(:,r) = primalClaculate(lambda(:,r));
fullgradient(:,r) = fullgrad(x(:,r));

% Plotting varialbes
dualobjForPlotting = zeros(1,MAX_ITER);
dualobjForPlotting(r) = dualObj(x(:,r),lambda(:,r));

while true
    xt = x(:,iter);
    lambdaold = lambda(:,iter);
    OldObj = dualObj(xt,lambdaold);
    lambdanew = lambdaold;

    for i = 1:training_data_length
        % Compute ith gradient
        grad = ithgrad(i,xt);

        % Update single coordinate at a time
        % If i==j then update the lambda vector i.e. ith coordinate
        lambdanew(i) = proj(lambdaold(i) + (b(i)^2 * norm(A(i,:)')^2)\(grad));

        % Update (r+1)th Primal based on lambda
        xt = xt + (lambdanew(i)-lambdaold(i))*b(i)*A(i,:)';

        % Update fullgradient
        fullgradient (i,r) = grad;

        % Increment the r
        r = r + 1;

        % If we reach maximum limit then break the loop
        if r == MAX_ITER
            break;
        end
    end

    % Store the primal solution after one iteration
    x(:,iter+1) = xt;
    deltax = x(:,iter) - x(:,iter+1);
    lambda(:,iter+1) = lambdanew;

    % Dual Objective will be increesing
    NewObj = dualObj(xt,lambdanew);

    % save the plotting state after iteration
    prevState = x(:,iter);
    newState = x(:,iter+1);

    % Dual objective should be increasing
    increaseInObj = NewObj-OldObj;
    dualobjForPlotting(iter+1) = NewObj;

    % Print Status
    fprintf('Iter:%5.0f | dualOldObj:%5.5e | dualNewObj:%5.5e | IncreeseInObj:%5.5e | x(1):%.4d | x(2):%.4d | x(3):%.4d\n',...
        iter,OldObj,NewObj,increaseInObj,newState);

    % Increment the iteration count
```

```
    iter = iter + 1;

    % Stopping creteria
    if norm(deltax) < epsilon
        if norm(proj(lambda(:,iter) - fullgradient(:,iter))-lambda(:,iter)) < epsilon % Stopping creteria
            break;
        end
    end

    % Check MAX_ITER break the outer loop
    if iter == MAX_ITER
        fprintf('Maximum iteration limit reached.\n');
        break;
    end
end

xStarSVMCD = xt;

% Plot the SVM boundry
figure(10)
hold on;
plot(digitTobeClassifiedData(2,:),digitTobeClassifiedData(3,:),'b*');
plot(otherData(2,:),otherData(3,:),'r.');
equationOflineSVMCD = @(a1,a2) (xStarSVMCD(1)*a1 + xStarSVMCD(2)*a2 + xStarSVMCD(3));
equationOflineSVMCDBound1 = @(a1,a2) (xStarSVMCD(1)*a1 + xStarSVMCD(2)*(a2+1) + xStarSVMCD(3));
equationOflineSVMCDBound2 = @(a1,a2) (xStarSVMCD(1)*a1 + xStarSVMCD(2)*(a2-1) + xStarSVMCD(3));
currentAxes = gca;
h = fimplicit(equationOflineSVMCD,[currentAxes.XLim,currentAxes.YLim]);
h1 = fimplicit(equationOflineSVMCDBound1,[currentAxes.XLim,currentAxes.YLim]);
h2 = fimplicit(equationOflineSVMCDBound2,[currentAxes.XLim,currentAxes.YLim]);
title(sprintf('Training Data: Classification boundry for the digit %d with SVM using Coordinate Descent Method',digitTobeClassified));
set(h,'LineWidth',2,'Color','magenta');
set(h1,'LineWidth',2,'Color','cyan');
set(h2,'LineWidth',2,'Color','cyan');
xlabel('Intensity')
ylabel('Symmetry')
grid on;
hold off;
```

```
*****************************************************************
### Starting Support Vector Machine using Coordinate Descent:
*****************************************************************
Iter:    1 | dualOldObj:0.00000e+00 | dualNewObj:1.34818e+02 | IncreeseInObj:1.34818e+02 | x(1):-3.5205e+00 | x(2):1.8691e+00 | x(3):3.9633e+00
Iter:    2 | dualOldObj:1.34818e+02 | dualNewObj:1.80109e+02 | IncreeseInObj:4.52918e+01 | x(1):-4.5679e+00 | x(2):2.3734e+00 | x(3):4.1247e+00
Iter:    3 | dualOldObj:1.80109e+02 | dualNewObj:3.07502e+02 | IncreeseInObj:1.27393e+02 | x(1):-5.1333e+00 | x(2):1.9417e+00 | x(3):4.3664e+00
Iter:    4 | dualOldObj:3.07502e+02 | dualNewObj:4.34541e+02 | IncreeseInObj:1.27039e+02 | x(1):-5.5357e+00 | x(2):1.9706e+00 | x(3):4.4801e+00
Iter:    5 | dualOldObj:4.34541e+02 | dualNewObj:5.61611e+02 | IncreeseInObj:1.27069e+02 | x(1):-5.8131e+00 | x(2):1.9829e+00 | x(3):4.5492e+00
Iter:    6 | dualOldObj:5.61611e+02 | dualNewObj:6.89359e+02 | IncreeseInObj:1.27749e+02 | x(1):-6.0406e+00 | x(2):1.9905e+00 | x(3):4.6028e+00
Iter:    7 | dualOldObj:6.89359e+02 | dualNewObj:8.17544e+02 | IncreeseInObj:1.28185e+02 | x(1):-6.1862e+00 | x(2):1.9883e+00 | x(3):4.6286e+00
Iter:    8 | dualOldObj:8.17544e+02 | dualNewObj:9.46007e+02 | IncreeseInObj:1.28462e+02 | x(1):-6.2748e+00 | x(2):1.9849e+00 | x(3):4.6417e+00
Iter:    9 | dualOldObj:9.46007e+02 | dualNewObj:1.07477e+03 | IncreeseInObj:1.28767e+02 | x(1):-6.3243e+00 | x(2):1.9816e+00 | x(3):4.6473e+00
Iter:   10 | dualOldObj:1.07477e+03 | dualNewObj:1.20362e+03 | IncreeseInObj:1.28843e+02 | x(1):-6.3501e+00 | x(2):1.9824e+00 | x(3):4.6533e+00
Iter:   11 | dualOldObj:1.20362e+03 | dualNewObj:1.33253e+03 | IncreeseInObj:1.28917e+02 | x(1):-6.3691e+00 | x(2):1.9832e+00 | x(3):4.6580e+00
Iter:   12 | dualOldObj:1.33253e+03 | dualNewObj:1.46146e+03 | IncreeseInObj:1.28923e+02 | x(1):-6.3819e+00 | x(2):1.9836e+00 | x(3):4.6611e+00
Iter:   13 | dualOldObj:1.46146e+03 | dualNewObj:1.59041e+03 | IncreeseInObj:1.28949e+02 | x(1):-6.3890e+00 | x(2):1.9838e+00 | x(3):4.6626e+00
```

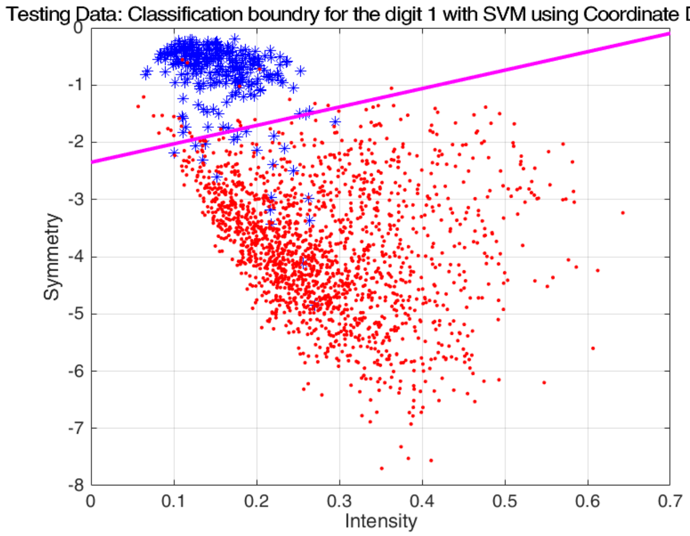



### Use the optimal model derived from SVM to classify a digit in the test/validation data

Generate the data matrix Av for validation

```
Av = [testing_data(2,:)' testing_data(3,:)' ones(testing_data_length,1)];

% Generate the label matrix bv for validation
bTrue = -ones(testing_data_length,1);
bTrue(testing_data(1,:) == digitTobeClassified) = 1;

% Compute Av*xStarGradientDescent
bClassifierTest = Av*xStarSVMCD;
bTest = sign(bClassifierTest);

% Find out quality measures for the classifier
truePositive = sum((bTest == 1) & (bTrue == 1))%#ok
falsePositive = sum((bTest == 1) & (bTrue == -1))%#ok
falseNegative = sum((bTest == -1) & (bTrue == 1))%#ok
trueNegative = sum((bTest == -1) & (bTrue == -1))%#ok
truePositiveRate = truePositive/(truePositive+falseNegative) %#ok Sensitivity
trueNegativeRate = trueNegative/(trueNegative+falsePositive) %#ok Specificity

% Correctly classified labels
accuracy = (truePositive+trueNegative)/(truePositive+falsePositive+falseNegative+trueNegative)%#ok

% Plot the results in figure 5
figure(11);
logicalIdForDigitTobeClassified = (testing_data(1,:) == digitTobeClassified);
digitTobeClassifiedData = testing_data(:,logicalIdForDigitTobeClassified);
otherData = testing_data(:,~logicalIdForDigitTobeClassified);
plot(digitTobeClassifiedData(2,:),digitTobeClassifiedData(3,:),'b*');
hold on;
plot(otherData(2,:),otherData(3,:),'r.');
xlabel('Intensity')
ylabel('Symmetry')
currentAxes = gca;
h = fimplicit(equationOflineSVMCD,[currentAxes.XLim,currentAxes.YLim]);
title(sprintf('Testing Data: Classification boundry for the digit %d with SVM using Coordinate Descent',digitTobeClassified));
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;
```

```
truePositive =

   242


falsePositive =

    26


falseNegative =

    22


trueNegative =

        1717


truePositiveRate =

    0.9167


trueNegativeRate =

    0.9851


accuracy =

    0.9761
```




*Published with MATLAB® R2017a*