Table of Contents

Setup Workspace

```
% 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);

% 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;
```

Read the testing 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);
```

```
% 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;
```

Least Square optimal solution - Only for Reference

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

Steepest descent with Armijo stepsize rule for linear regression

```
fprintf('**********************************************************************
\n');
fprintf('### Starting Steepest descent 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 \n',r);
        fprintf('Feval:%5.0f\n',nFeval);
        fprintf('OldObj:%5.5e\n',obj);
        fprintf('NewObj:%5.5e\n',newobj);
        fprintf('ReductionInObj:%5.5e\n',obj-newobj);
        fprintf('GradientNorm:%5.2f\n',norm(gradient));
        fprintf('x(1):%.4d | x(2):%.4d | x(3):%.4d\n',x);
        fprintf('------------------------------------------------------
\n');
    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 \n',r);
fprintf('Feval:%5.0f\n',nFeval);
fprintf('OldObj:%5.5e\n',obj);
fprintf('NewObj:%5.5e\n',newobj);
fprintf('ReductionInObj:%5.5e\n',obj-newobj);
fprintf('GradientNorm:%5.2f\n',norm(gradient));
```

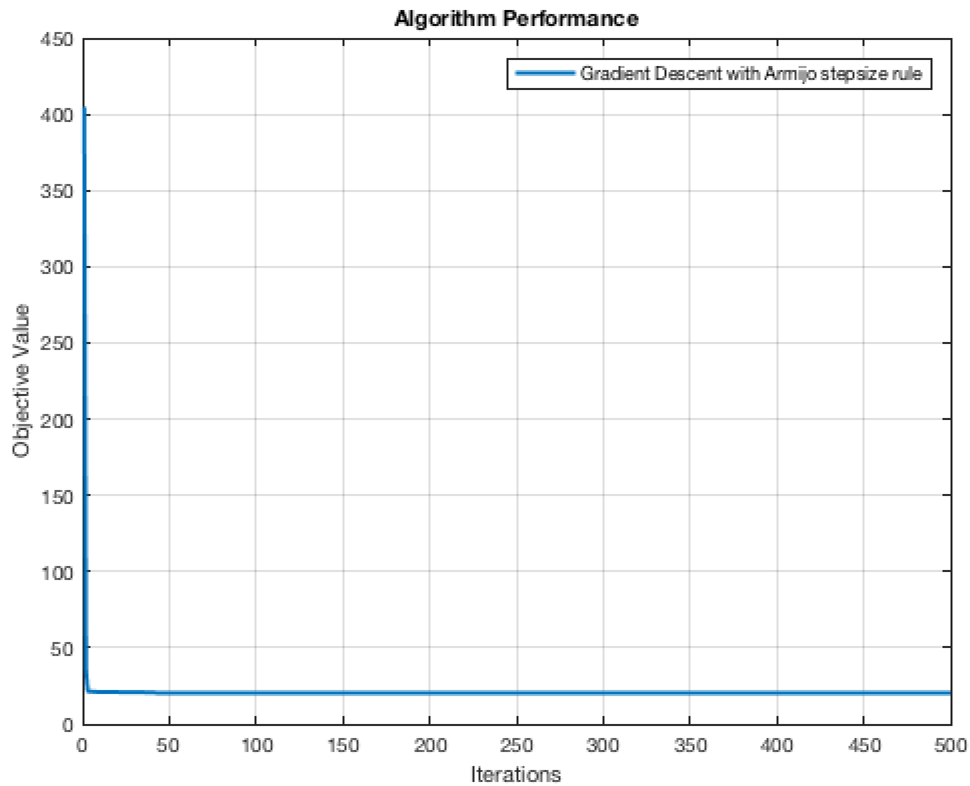
```
fprintf('x(1):%.4d | x(2):%.4d | x(3):%.4d\n',x);
fprintf('-----------------------------------------------------\n');

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

% Display optimal solution
xStarGradientDescent = x %#ok

% Plot the reduction in gradient norm and objective reduction
figure(1)
plot(1:r,objForPlotting(1:r),'LineWidth',2);
xlabel('Iterations');ylabel('Objective Value');grid on;
legend('Gradient Descent with Armijo stepsize rule');
title('Algorithm Performance');
set(gca,'XLim',[0 500]);

figure(2)
plot(1:r,GradientNormForPlotting(1:r),'LineWidth',2);
xlabel('Iterations');ylabel('Norm of the Gradient');grid on;
legend('Gradient Descent with Armijo stepsize rule');
title('Algorithm Performance');
set(gca,'XLim',[0 500]);

% Plot states for gradient descent
figure
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 saperating boundry
createFigureAndPlotTrainingData
hold on;
equationOflineGD = @(a1,a2) (xStarGradientDescent(1)*a1 +...
    xStarGradientDescent(2)*a2 + xStarGradientDescent(3));
h = fimplicit(equationOflineGD,[get(gca,'XLim'),get(gca,'YLim')]);
title('Training Data: Classification boundry with Gradient Descent');
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;

% Visulize how the line changes as algorithm progresses
createFigureAndPlotTrainingData
hold on;
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,[get(gca,'XLim'),get(gca,'YLim')]);
        set(h,'LineWidth',2,'Color','green','LineStyle','--');
        hold on;
```

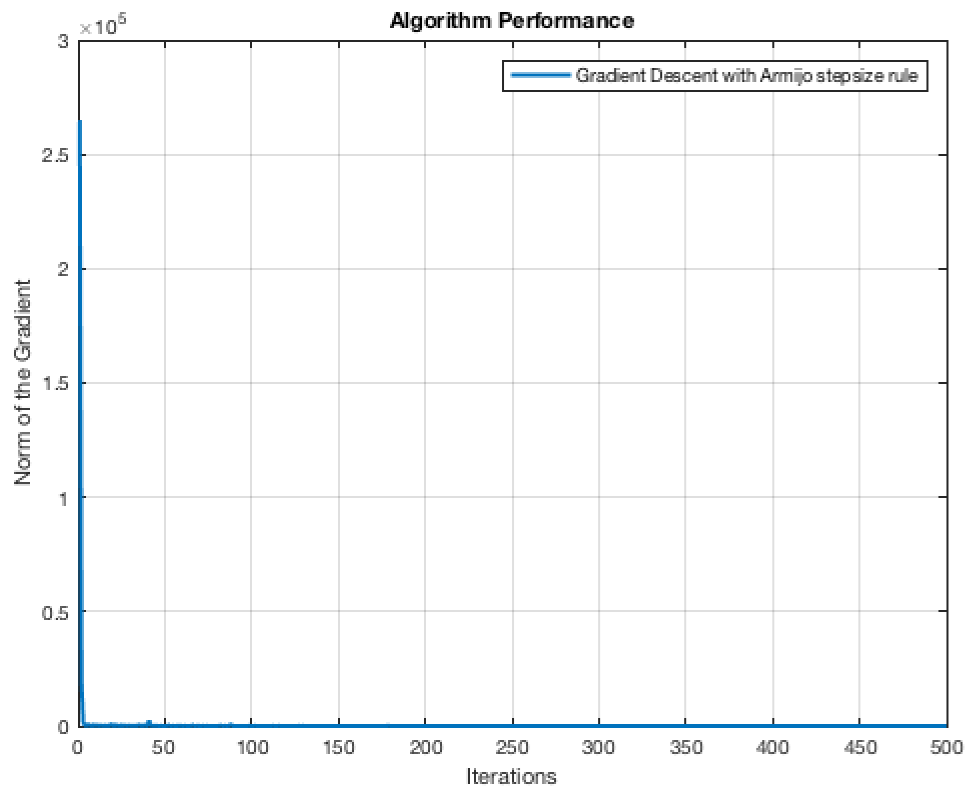
```
    end
end
h = fimplicit(equationOflineGD,[get(gca,'XLim'),get(gca,'YLim')]);
set(h,'LineWidth',3,'Color','magenta');
title('Training Data: Change in classification boundry: Gradient
 Descent');
grid on;
hold off;

% Use the optimal model derived from gradient descent to classify a
 digit
% in the test/validation data Compute Av*xStarGradientDescent
bClassifierTest = Av*xStarGradientDescent;
bTest = sign(bClassifierTest);

% Print Confusion Matrices
printConfusion(bTest,bTrue);

% Plot the results in figure
createFigureAndPlotTestingData
h = fimplicit(equationOflineGD,[get(gca,'XLim'),get(gca,'YLim')]);
title('Testing Data: Classification boundry with Gradient Descent');
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;

*************************************************************
### Starting Steepest descent 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):3.5939e-01
----------------------------------------------------
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):4.4547e-01
----------------------------------------------------
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):5.1084e-01
----------------------------------------------------
Iter:  136
```

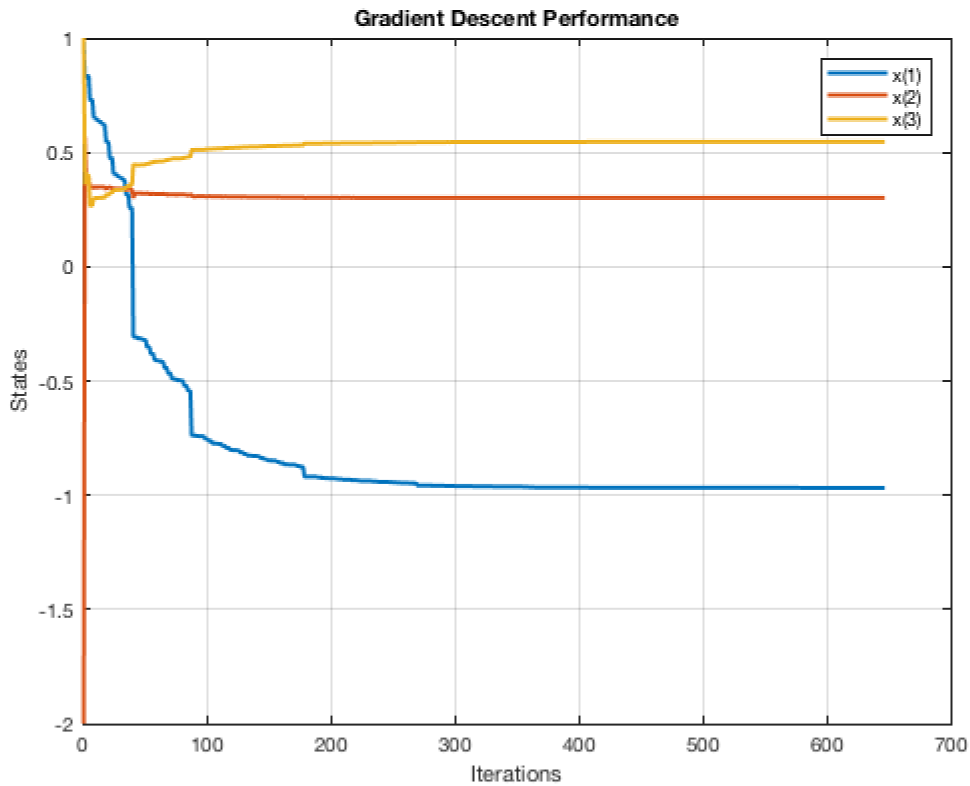
```
Feval:  732
OldObj:2.02638e+01
NewObj:2.02638e+01
ReductionInObj:3.31888e-05
GradientNorm: 6.83
x(1):-8.2734e-01 | x(2):3.0487e-01 | x(3):5.2440e-01
-----------------------------------------------------
Iter:  181
Feval:  973
OldObj:2.02587e+01
NewObj:2.02587e+01
ReductionInObj:2.04505e-06
GradientNorm: 2.46
x(1):-9.1854e-01 | x(2):3.0239e-01 | x(3):5.3838e-01
-----------------------------------------------------
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):5.4128e-01
-----------------------------------------------------
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):5.4430e-01
-----------------------------------------------------
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):5.4493e-01
-----------------------------------------------------
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):5.4558e-01
-----------------------------------------------------
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):5.4570e-01
```

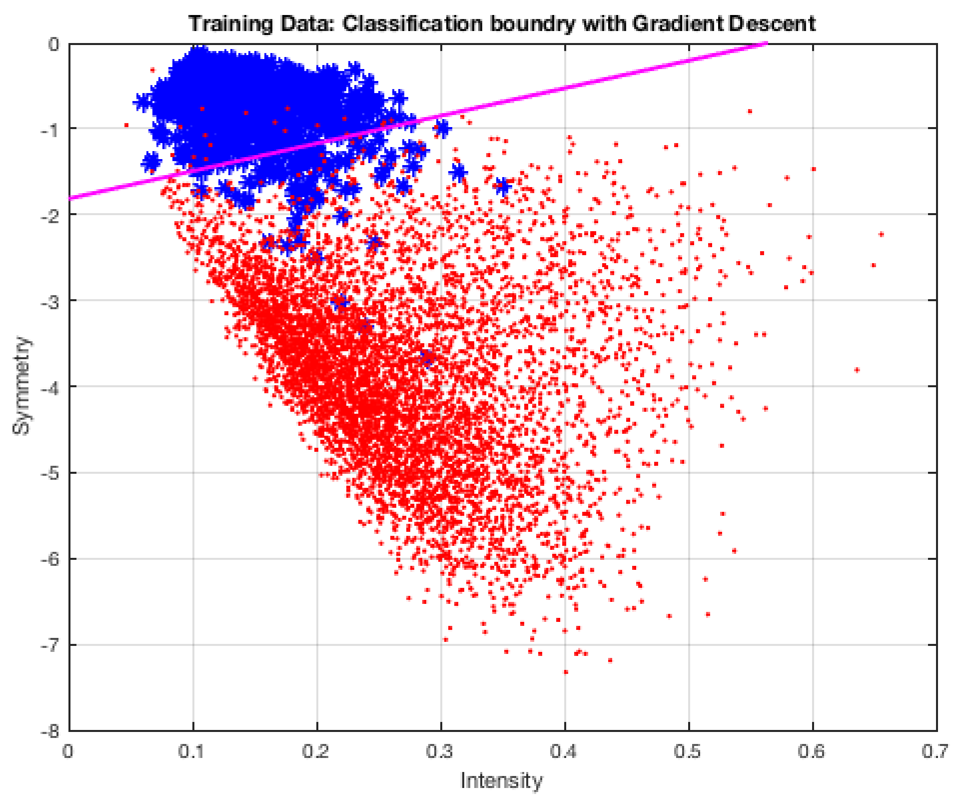
```
--------------------------------------------------
Iter:  451
Feval: 2422
OldObj:2.02580e+01
NewObj:2.02580e+01
ReductionInObj:8.35421e-11
GradientNorm: 0.03
x(1):-9.6695e-01 | x(2):3.0087e-01 | x(3):5.4585e-01
--------------------------------------------------
Iter:  496
Feval: 2663
OldObj:2.02580e+01
NewObj:2.02580e+01
ReductionInObj:5.64551e-10
GradientNorm: 0.10
x(1):-9.6715e-01 | x(2):3.0087e-01 | x(3):5.4588e-01
--------------------------------------------------
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):5.4591e-01
--------------------------------------------------
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):5.4592e-01
--------------------------------------------------
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):5.4592e-01
--------------------------------------------------
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):5.4593e-01
--------------------------------------------------

xStarGradientDescent =

   -0.9674
    0.3009
```

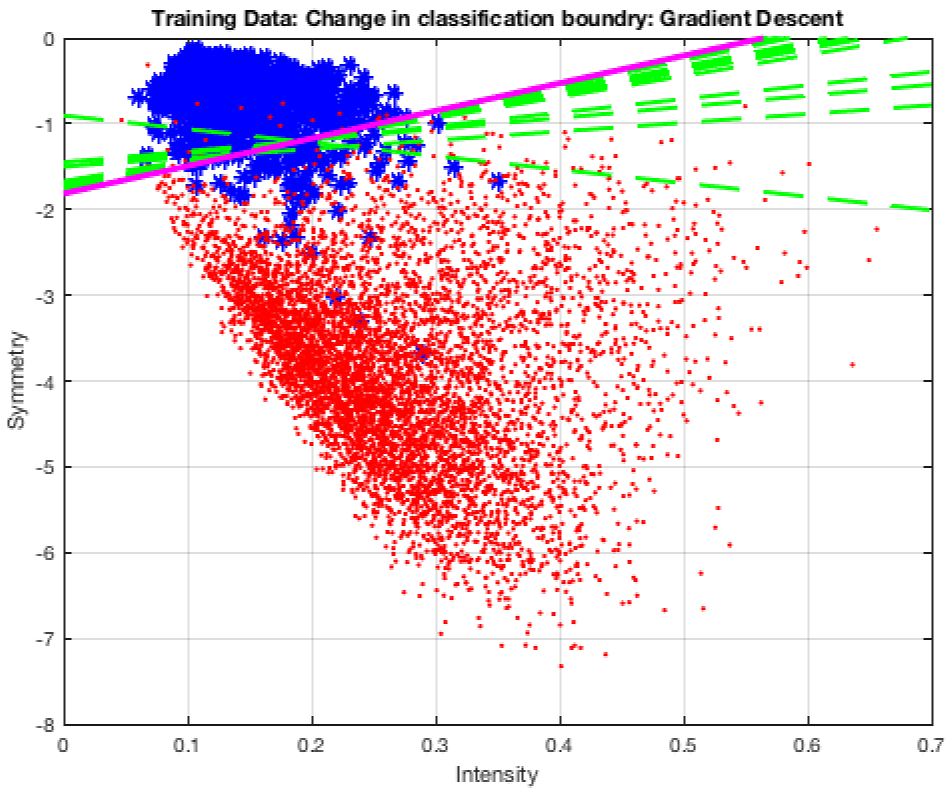
```
    0.5459

truePositive =

   226

falsePositive =

     8

falseNegative =

    38

trueNegative =

        1735

truePositiveRate =

    0.8561

trueNegativeRate =

    0.9954

accuracy =

    0.9771
```


Algorithm Performance
Gradient Descent with Armijo stepsize rule
Objective Value
Iterations
0
50
100
150
200
250
300
350
400
450
500

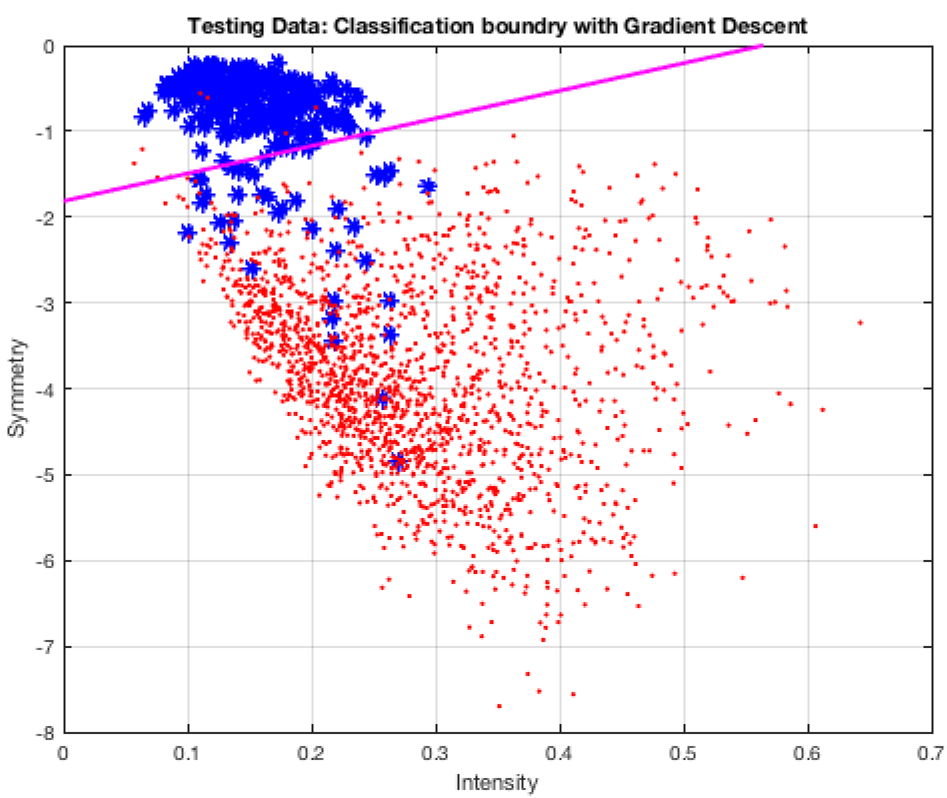

Algorithm Performance
Gradient Descent with Armijo stepsize rule
$\times 10^5$
Norm of the Gradient
Iterations
0
0.5
1
1.5
2
2.5
3
50
100
150
200
250
300
350
400
450
500


Gradient Descent Performance
x(1)
x(2)
x(3)
States
Iterations
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700


Training Data: Classification boundry with Gradient Descent
Symmetry
Intensity
0
-1
-2
-3
-4
-5
-6
-7
-8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7



Training Data: Change in classification boundry: Gradient Descent
Symmetry
Intensity
0
-1
-2
-3
-4
-5
-6
-7
-8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7



Testing Data: Classification boundry with Gradient Descent
Symmetry
Intensity
0
-1
-2
-3
-4
-5
-6
-7
-8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

Coordinate descent for linear regression

```
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;
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 \n',iter);
        fprintf('OldObj:%5.5e\n',OldObj);
        fprintf('NewObj:%5.5e\n',NewObj);
        fprintf('ReductionInObj:%5.5e\n',ReductionInObj);
        fprintf('x(1):%.4d | x(2):%.4d | x(3):%.4d\n',updatedState);
        fprintf('-----------------------------------------------------
\n');
    end

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

% Print Optimal Solution
xStarCoordinateDescent = stateForPlotting(:,iter) %#ok

% Plot the boundry
createFigureAndPlotTrainingData
hold on;
equationOflineCD = @(a1,a2) (xStarCoordinateDescent(1)*a1 +...
    xStarCoordinateDescent(2)*a2 + xStarCoordinateDescent(3));
h = fimplicit(equationOflineCD,[get(gca,'XLim'),get(gca,'YLim')]);
title('Training Data: Classification boundry with Coordinate
 Descent');
set(h,'LineWidth',2,'Color','magenta');
```

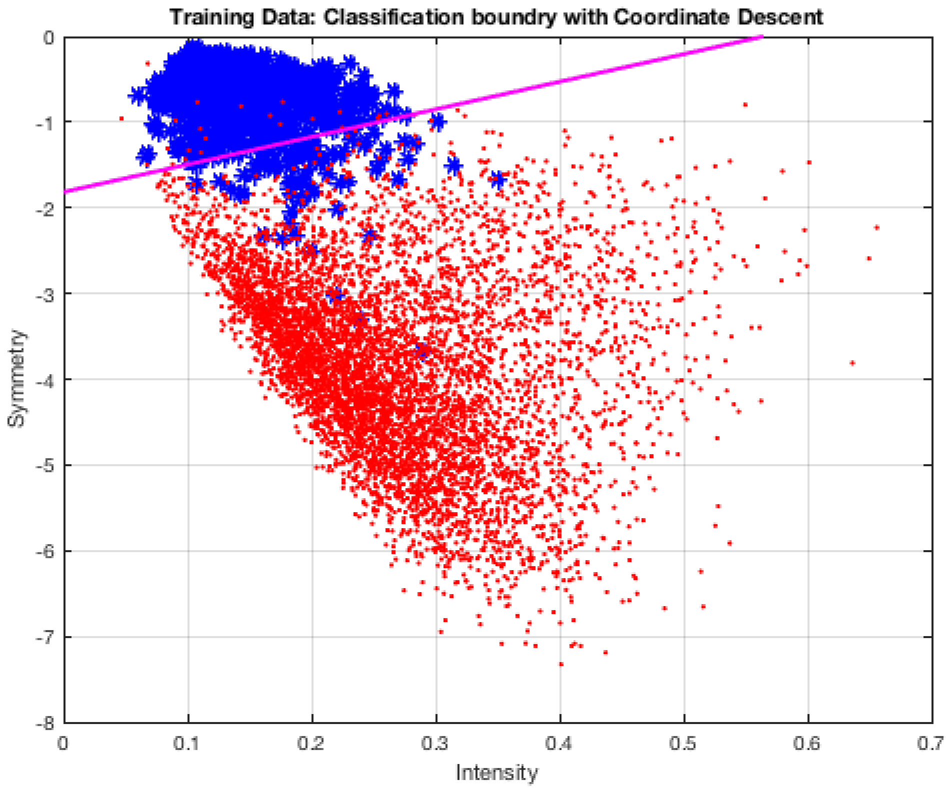
```
grid on;
hold off;

% Visulize how the line changes as algorithm progresses
createFigureAndPlotTrainingData
for i = 1:iter
    if(mod(i,10)==1 || i==1)
        hold on;
        eqOflineCD = @(a1,a2) (stateForPlotting(1,i)*a1 +...
            stateForPlotting(2,i)*a2 + stateForPlotting(3,i));
        h = fimplicit(eqOflineCD,[get(gca,'XLim'),get(gca,'YLim')]);
        set(h,'LineWidth',2,'Color','green','LineStyle','--');
    end
end
h = fimplicit(equationOflineCD,[get(gca,'XLim'),get(gca,'YLim')]);
set(h,'LineWidth',3,'Color','magenta');
title('Training Data: Change in classification boundry: Coordinate
 Descent');
grid on;
hold off;

% Plot the reduction in gradient norm and objective reduction
figure(1)
hold on;
plot(1:iter,objForPlotting(1:iter),'LineWidth',2);
hold off;
legend('Gradient Descent with Armijo stepsize rule','Coordinate
 Descent');
title('Algorithm Performance');
set(gca,'XLim',[0 500]);

figure(2)
hold on;
plot(1:iter,GradientNormForPlotting(1:iter),'LineWidth',2);
hold off;
legend('Gradient Descent with Armijo stepsize rule','Coordinate
 Descent');
title('Algorithm Performance');
set(gca,'XLim',[0 500]);

% Plot coordinate descent performance
figure
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')

% Use the optimal model derived from coordinate descent to classify a
 digit
% in the test/validation data
% Compute Av*xStarGradientDescent
bClassifierTest = Av*xStarCoordinateDescent;
bTest = sign(bClassifierTest);
```

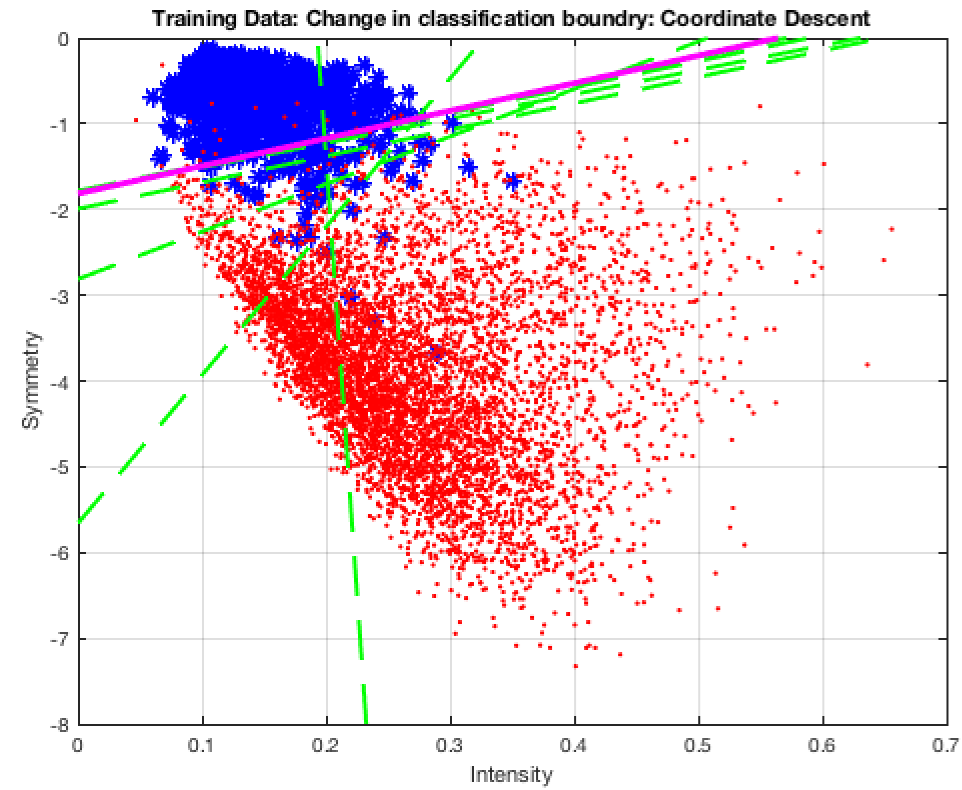
```
% Print Confusion Matrices
printConfusion(bTest,bTrue);

% Plot the results in figure
createFigureAndPlotTestingData
hold on;
h = fimplicit(equationOflineCD,[get(gca,'XLim'),get(gca,'YLim')]);
title('Testing Data: Classification boundry with Coordinate Descent');
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;
```

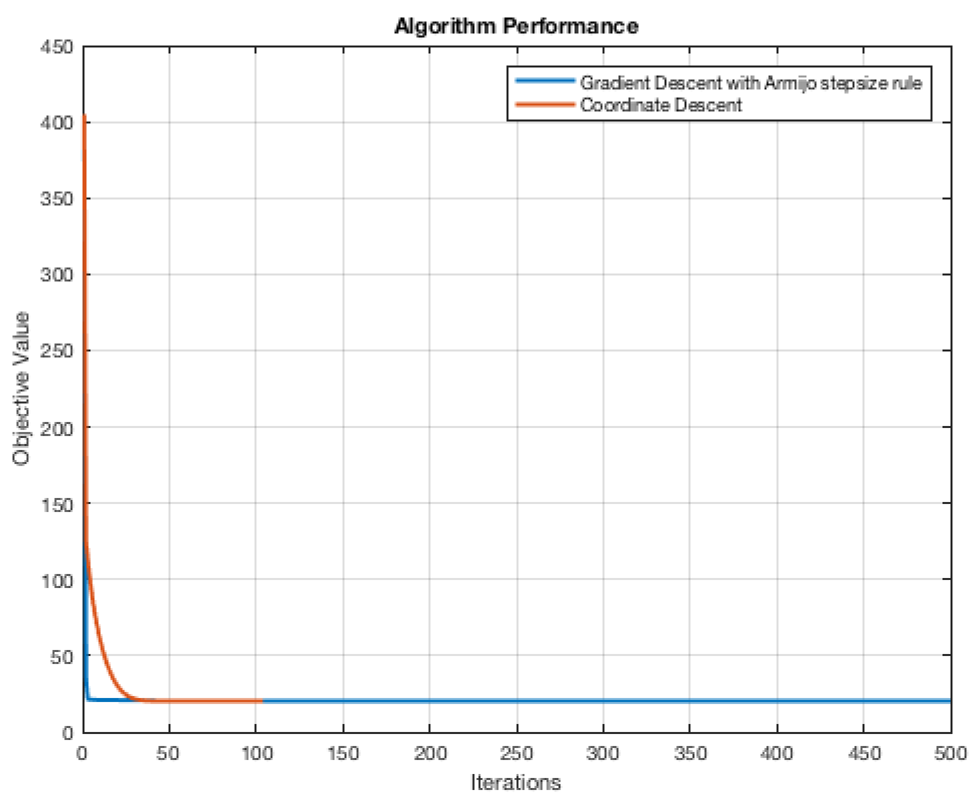
```
***************************************************************
### Starting coordinate descent:
***************************************************************
Iter:   11
OldObj:6.11981e+01
NewObj:5.65403e+01
ReductionInObj:4.65785e+00
x(1):-1.6240e+01 | x(2):-7.9865e-02 | x(3):3.1367e+00
------------------------------------------------------
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
```

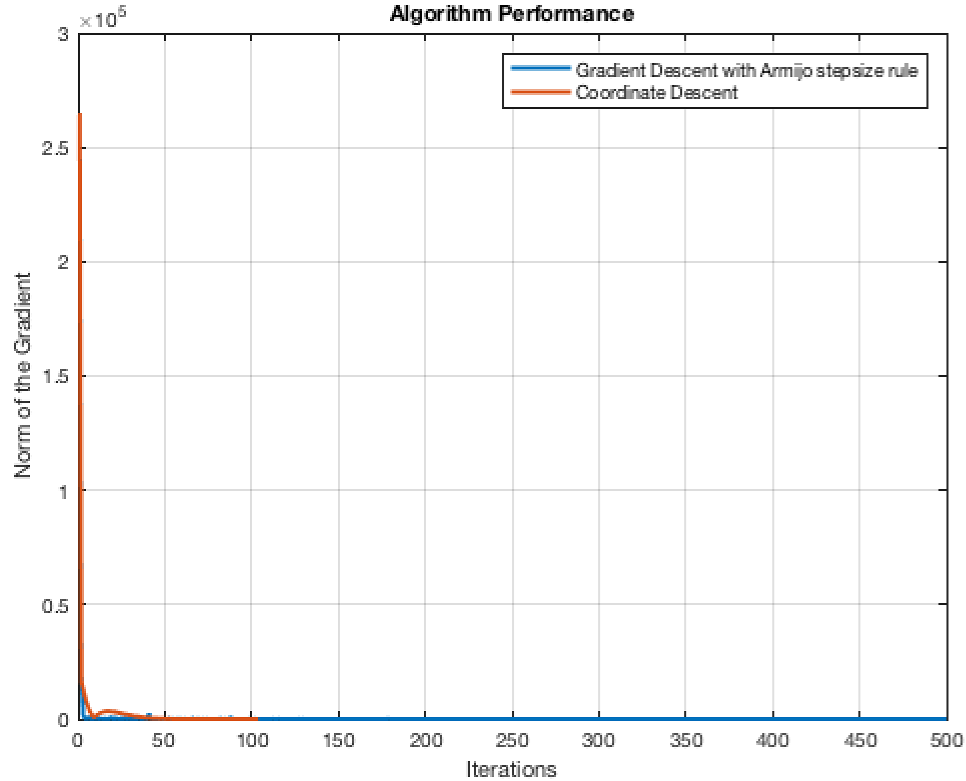
```
ReductionInObj:2.63490e-05
x(1):-9.4839e-01 | x(2):3.0098e-01 | x(3):5.4150e-01
----------------------------------------------------
Iter:   81
OldObj:2.02581e+01
NewObj:2.02581e+01
ReductionInObj:4.08474e-06
x(1):-9.6252e-01 | x(2):3.0070e-01 | x(3):5.4414e-01
----------------------------------------------------
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


truePositive =

   226


falsePositive =

     8


falseNegative =

    38


trueNegative =

        1735


truePositiveRate =

    0.8561
```

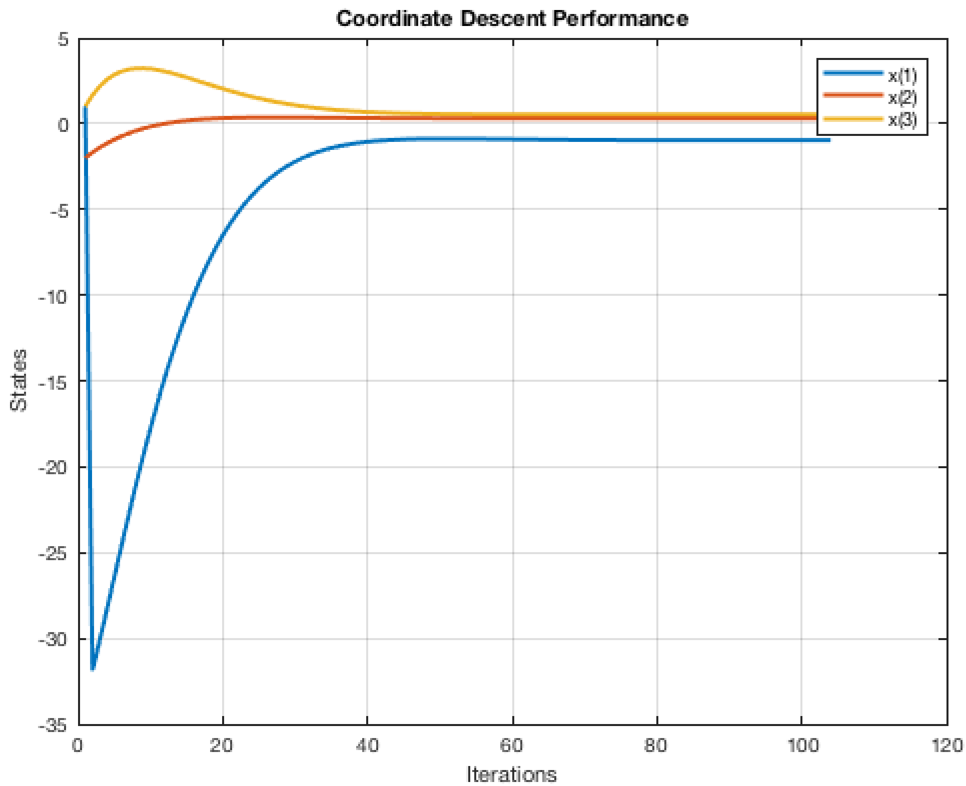
```
trueNegativeRate =

    0.9954


accuracy =

    0.9771
```

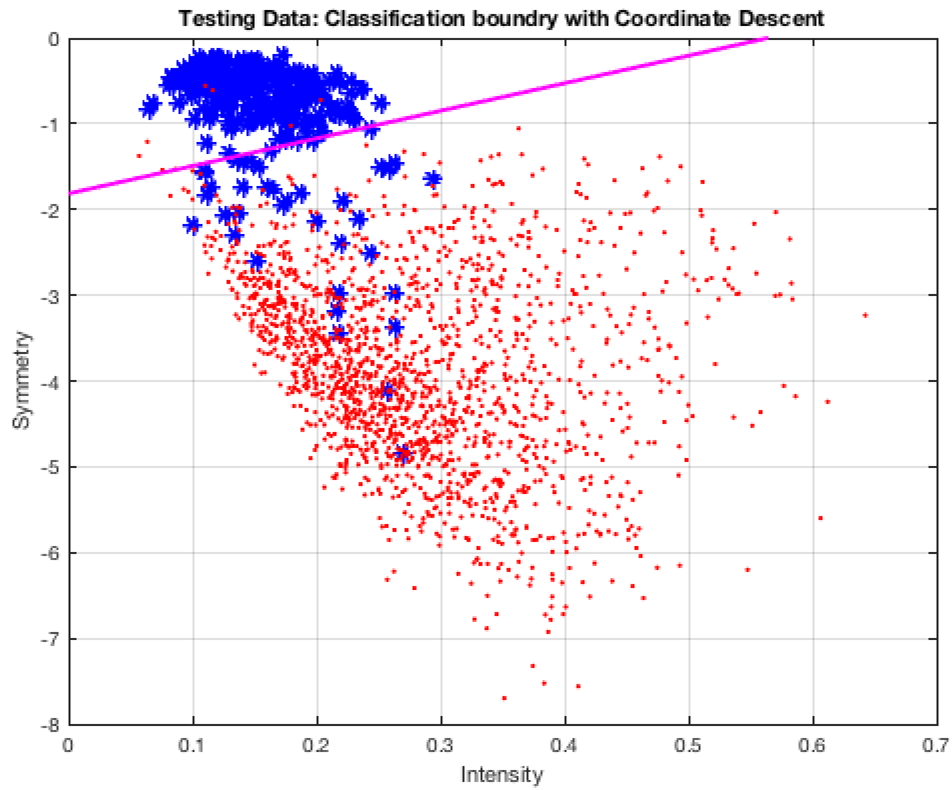



Training Data: Change in classification boundry: Coordinate Descent
Symmetry
Intensity
0
-1
-2
-3
-4
-5
-6
-7
-8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


Algorithm Performance
Gradient Descent with Armijo stepsize rule
Coordinate Descent
Objective Value
Iterations
450
400
350
300
250
200
150
100
50
0
0
50
100
150
200
250
300
350
400
450
500


Algorithm Performance
Gradient Descent with Armijo stepsize rule
Coordinate Descent
Norm of the Gradient
Iterations


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



Support Vector Machine using 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
```

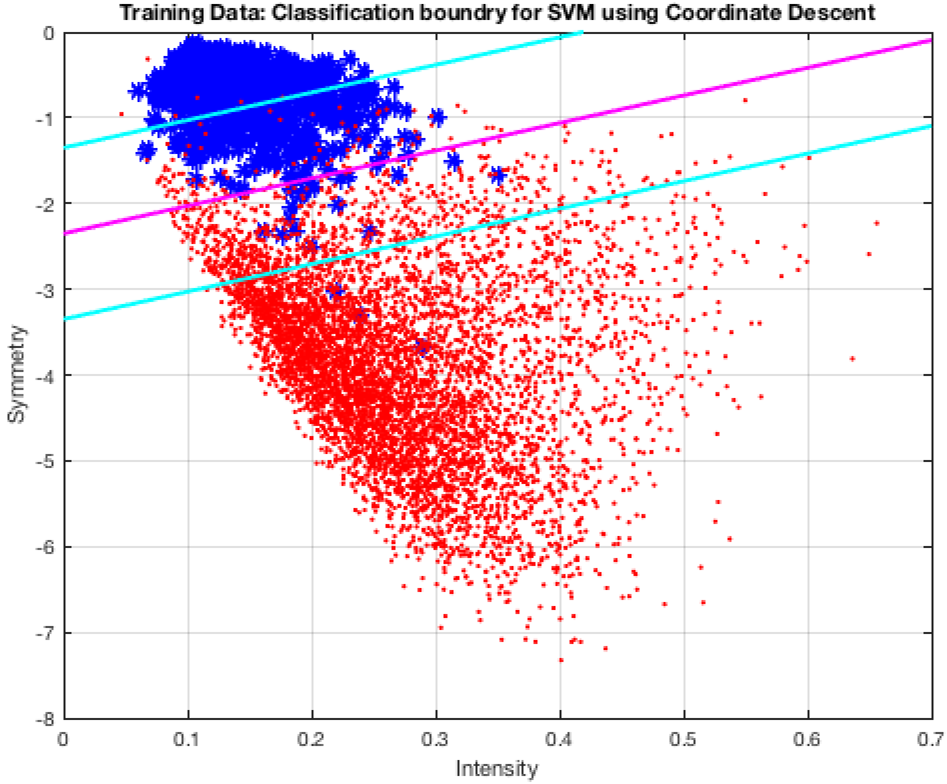
```
        xt = xt + (lambdanew(i)-lambdaold(i))*b(i)*A(i,:)';

        % 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 \n',iter);
    fprintf('OldObj:%5.5e\n',OldObj);
    fprintf('NewObj:%5.5e\n',NewObj);
    fprintf('IncreeseInObj:%5.5e\n',increaseInObj);
    fprintf('x(1):%.4d | x(2):%.4d | x(3):%.4d\n',newState);
    fprintf('-----------------------------------------------------\n')

    % 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;
```

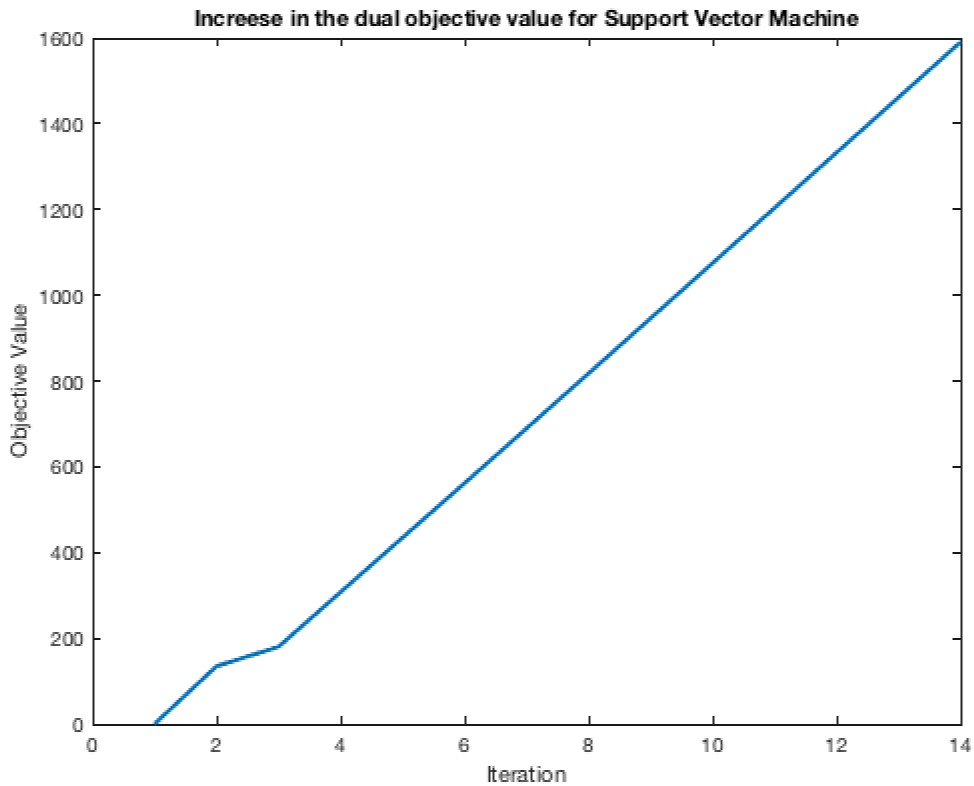
```
    end
end

% Print the optimal solution with SVM
xStarSVMCD = xt %#ok<NOPTS>

% Plot the SVM boundry
createFigureAndPlotTrainingData
hold on;
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));
h = fimplicit(equationOflineSVMCD,[get(gca,'XLim'),get(gca,'YLim')]);
h1 = fimplicit(equationOflineSVMCDBound1,
[get(gca,'XLim'),get(gca,'YLim')]);
h2 = fimplicit(equationOflineSVMCDBound2,
[get(gca,'XLim'),get(gca,'YLim')]);
title('Training Data: Classification boundry for SVM using Coordinate
 Descent');
set(h,'LineWidth',2,'Color','magenta');
set(h1,'LineWidth',2,'Color','cyan');
set(h2,'LineWidth',2,'Color','cyan');
grid on;
hold off;

% Plot how dual objective increeses as iteration progresses
figure
plot(1:iter,dualobjForPlotting(1:iter),'LineWidth',2);
ylabel('Objective Value')
xlabel('Iteration')
title('Increese in the dual objective value for Support Vector
 Machine');

% Use the optimal model derived from SVM to classify a digit in the
% test/validation data
% Compute Av*xStarGradientDescent
bClassifierTest = Av*xStarSVMCD;
bTest = sign(bClassifierTest);

% Print Confusion Matrices
printConfusion(bTest,bTrue);

% Plot the results in figure
createFigureAndPlotTestingData
h = fimplicit(equationOflineSVMCD,[get(gca,'XLim'),get(gca,'YLim')]);
title('Testing Data: Classification boundry for SVM using Coordinate
 Descent');
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;
```

```
*************************************************************
### Starting Support Vector Machine using Coordinate Descent:
*************************************************************
Iter:    1
OldObj:0.00000e+00
NewObj:1.34818e+02
IncreeseInObj:1.34818e+02
x(1):-3.5205e+00 | x(2):1.8691e+00 | x(3):3.9633e+00
----------------------------------------------------
Iter:    2
OldObj:1.34818e+02
NewObj:1.80109e+02
IncreeseInObj:4.52918e+01
x(1):-4.5679e+00 | x(2):2.3734e+00 | x(3):4.1247e+00
----------------------------------------------------
Iter:    3
OldObj:1.80109e+02
NewObj:3.07502e+02
IncreeseInObj:1.27393e+02
x(1):-5.1333e+00 | x(2):1.9417e+00 | x(3):4.3664e+00
----------------------------------------------------
Iter:    4
OldObj:3.07502e+02
NewObj:4.34541e+02
IncreeseInObj:1.27039e+02
x(1):-5.5357e+00 | x(2):1.9706e+00 | x(3):4.4801e+00
----------------------------------------------------
Iter:    5
OldObj:4.34541e+02
NewObj:5.61611e+02
IncreeseInObj:1.27069e+02
x(1):-5.8131e+00 | x(2):1.9829e+00 | x(3):4.5492e+00
----------------------------------------------------
Iter:    6
OldObj:5.61611e+02
NewObj:6.89359e+02
IncreeseInObj:1.27749e+02
x(1):-6.0406e+00 | x(2):1.9905e+00 | x(3):4.6028e+00
----------------------------------------------------
Iter:    7
OldObj:6.89359e+02
NewObj:8.17544e+02
IncreeseInObj:1.28185e+02
x(1):-6.1862e+00 | x(2):1.9883e+00 | x(3):4.6286e+00
----------------------------------------------------
Iter:    8
OldObj:8.17544e+02
NewObj:9.46007e+02
IncreeseInObj:1.28462e+02
x(1):-6.2748e+00 | x(2):1.9849e+00 | x(3):4.6417e+00
----------------------------------------------------
Iter:    9
OldObj:9.46007e+02
NewObj:1.07477e+03
```

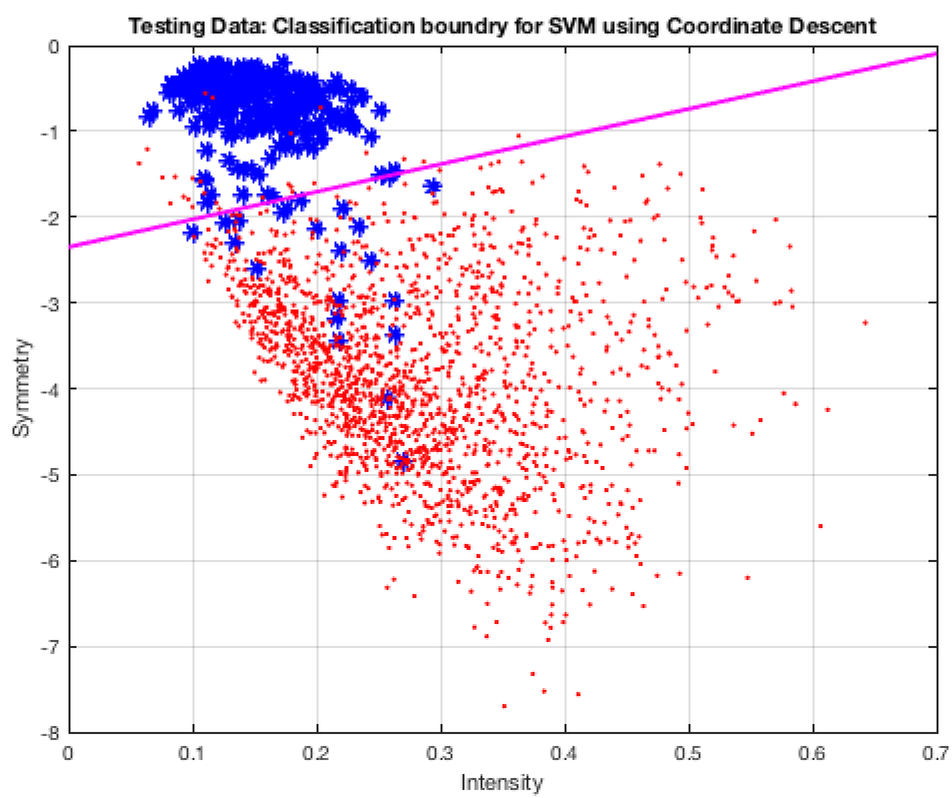
```
IncreeseInObj:1.28767e+02
x(1):-6.3243e+00 | x(2):1.9816e+00 | x(3):4.6473e+00
-----------------------------------------------------
Iter:   10
OldObj:1.07477e+03
NewObj:1.20362e+03
IncreeseInObj:1.28843e+02
x(1):-6.3501e+00 | x(2):1.9824e+00 | x(3):4.6533e+00
-----------------------------------------------------
Iter:   11
OldObj:1.20362e+03
NewObj:1.33253e+03
IncreeseInObj:1.28917e+02
x(1):-6.3691e+00 | x(2):1.9832e+00 | x(3):4.6580e+00
-----------------------------------------------------
Iter:   12
OldObj:1.33253e+03
NewObj:1.46146e+03
IncreeseInObj:1.28923e+02
x(1):-6.3819e+00 | x(2):1.9836e+00 | x(3):4.6611e+00
-----------------------------------------------------
Iter:   13
OldObj:1.46146e+03
NewObj:1.59041e+03
IncreeseInObj:1.28949e+02
x(1):-6.3890e+00 | x(2):1.9838e+00 | x(3):4.6626e+00
-----------------------------------------------------

xStarSVMCD =

   -6.3890
    1.9838
    4.6626


truePositive =

   242


falsePositive =

    26


falseNegative =

    22


trueNegative =

        1717
```

```
truePositiveRate =

    0.9167

trueNegativeRate =

    0.9851

accuracy =

    0.9761
```





Increese in the dual objective value for Support Vector Machine
Objective Value
Iteration
0
200
400
600
800
1000
1200
1400
1600
0
2
4
6
8
10
12
14



Testing Data: Classification boundry for SVM using Coordinate Descent
Symmetry
Intensity
0
-1
-2
-3
-4
-5
-6
-7
-8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

Newton's method with constant stepsize rule

```
fprintf('*************************************************************
\n');
fprintf(['### Starting Newton method with constant stepsize',...
    ' rule for linear regression problem: \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));

% Hessian Computation
h = A'*A;

% Newton's method with constant stepsize rule
x = x0;
% Stepsize is inverse of the maximum eigen value of hessian, algorithm
% stops at 828493 iteration
alpha =  1/max(eig(h)); %#ok<NASGU>
% Selecting higher stepsize makes algorithm to converge faster
alpha = 0.01;

epsilon = sqrt(eps);
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;
ReductionInObj = 1;

while abs(ReductionInObj) > epsilon
    % Steepest descent direction i.e. -grad
    direction = -h\gradient;

    % Update the next state, Newton's Iteration
    x = x + alpha*direction;

    % Compute New Objective value
    newobj = f(x);
    ReductionInObj = obj-newobj;
```

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

    % For the next iteration
    obj = newobj;
    gradient = g(x);
    r = r+1;

    % For plotting
    stateForPlotting(:,r) = x;
    objForPlotting(r) = obj;
    GradientNormForPlotting(r) = norm(gradient);
end

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

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

% Optimal Solution using Newton's method
xStarNewton = x %#ok<NOPTS>

% Plot the reduction in gradient norm and objective reduction
figure(1)
hold on;
plot(1:r,objForPlotting(1:r),'LineWidth',2);
legend('Gradient Descent with Armijo stepsize rule',...
    'Coordinate Descent','Newton Method with Constant stepsize');
title('Algorithm Performance');
set(gca,'XLim',[0 500]);

figure(2)
hold on;
plot(1:r,GradientNormForPlotting(1:r),'LineWidth',2);
legend('Gradient Descent with Armijo stepsize rule',...
```

```
    'Coordinate Descent','Newton Method with Constant stepsize');
title('Algorithm Performance');
set(gca,'XLim',[0 500]);

figure
plot(1:r,stateForPlotting(:,1:r),'LineWidth',2)
xlabel('Iterations');ylabel('States');grid on;
legend('x(1)','x(2)','x(3)');
title('Newton Method Performance')

% Plot the boundry
createFigureAndPlotTrainingData
hold on;
equationOflineNewton = @(a1,a2) (xStarNewton(1)*a1 + ...
    xStarNewton(2)*a2 + xStarNewton(3));
h = fimplicit(equationOflineNewton,[get(gca,'XLim'),get(gca,'YLim')]);
title(sprintf('Training Data: Classification boundry with Newton
 Method'));
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;

% Visulize how the line changes as algorithm progresses
createFigureAndPlotTrainingData
for i = 1:r
    if(mod(i,20)==1 || i==1)
        hold on;
        eqOflineNewton = @(a1,a2) (stateForPlotting(1,i)*a1 +...
            stateForPlotting(2,i)*a2 + stateForPlotting(3,i));
        h = fimplicit(eqOflineNewton,
[get(gca,'XLim'),get(gca,'YLim')]);
        set(h,'LineWidth',2,'Color','green','LineStyle','--');
    end
end
h = fimplicit(equationOflineGD,[get(gca,'XLim'),get(gca,'YLim')]);
set(h,'LineWidth',3,'Color','magenta');
title('Training Data: Change in classification boundry with Newton
 Method');
grid on;
hold off;

% Use the optimal model derived from SVM to classify a digit in the
% test/validation data
% Compute Av*xStarGradientDescent
bClassifierTest = Av*xStarNewton;
bTest = sign(bClassifierTest);

% Print Confusion Matrices
printConfusion(bTest,bTrue);

% Plot the results in figure
createFigureAndPlotTestingData
h = fimplicit(equationOflineSVMCD,[get(gca,'XLim'),get(gca,'YLim')]);
```

```
title('Testing Data: Classification boundry for SVM using Coordinate
 Descent');
set(h,'LineWidth',2,'Color','magenta');
grid on;
hold off;
```

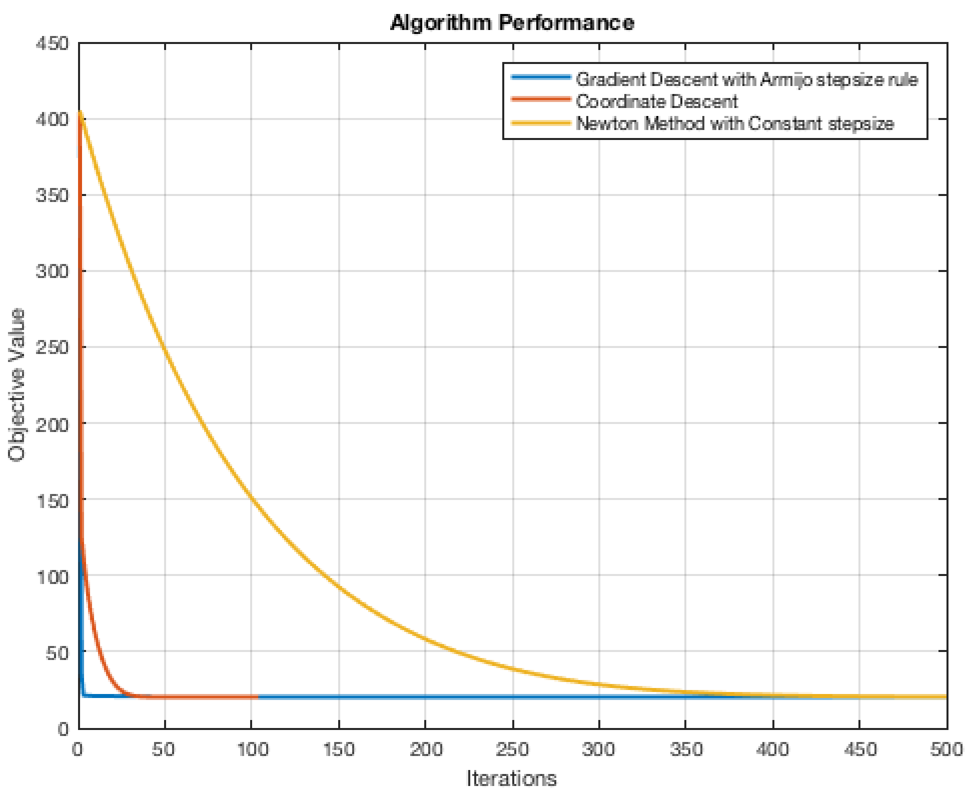
```
**************************************************************
### Starting Newton method with constant stepsize rule for linear
 regression problem:
**************************************************************
Iter:    1
OldObj:4.04761e+02
NewObj:4.00724e+02
ReductionInObj:4.03742e+00
GradientNorm:264884.85
x(1):9.8033e-01 | x(2):-1.9770e+00 | x(3):9.9546e-01
------------------------------------------------------
Iter:   46
OldObj:2.57977e+02
NewObj:2.55413e+02
ReductionInObj:2.56378e+00
GradientNorm:168515.89
x(1):2.7169e-01 | x(2):-1.1483e+00 | x(3):8.3191e-01
------------------------------------------------------
Iter:   91
OldObj:1.64864e+02
NewObj:1.63240e+02
ReductionInObj:1.62362e+00
GradientNorm:107207.37
x(1):-1.7913e-01 | x(2):-6.2106e-01 | x(3):7.2787e-01
------------------------------------------------------
Iter:  136
OldObj:1.06042e+02
NewObj:1.05021e+02
ReductionInObj:1.02153e+00
GradientNorm:68203.77
x(1):-4.6593e-01 | x(2):-2.8565e-01 | x(3):6.6168e-01
------------------------------------------------------
Iter:  181
OldObj:6.92494e+01
NewObj:6.86164e+01
ReductionInObj:6.32958e-01
GradientNorm:43390.25
x(1):-6.4840e-01 | x(2):-7.2274e-02 | x(3):6.1957e-01
------------------------------------------------------
Iter:  226
OldObj:4.67459e+01
NewObj:4.63665e+01
ReductionInObj:3.79308e-01
GradientNorm:27604.25
x(1):-7.6448e-01 | x(2):6.3476e-02 | x(3):5.9278e-01
------------------------------------------------------
Iter:  271
OldObj:3.35961e+01
```

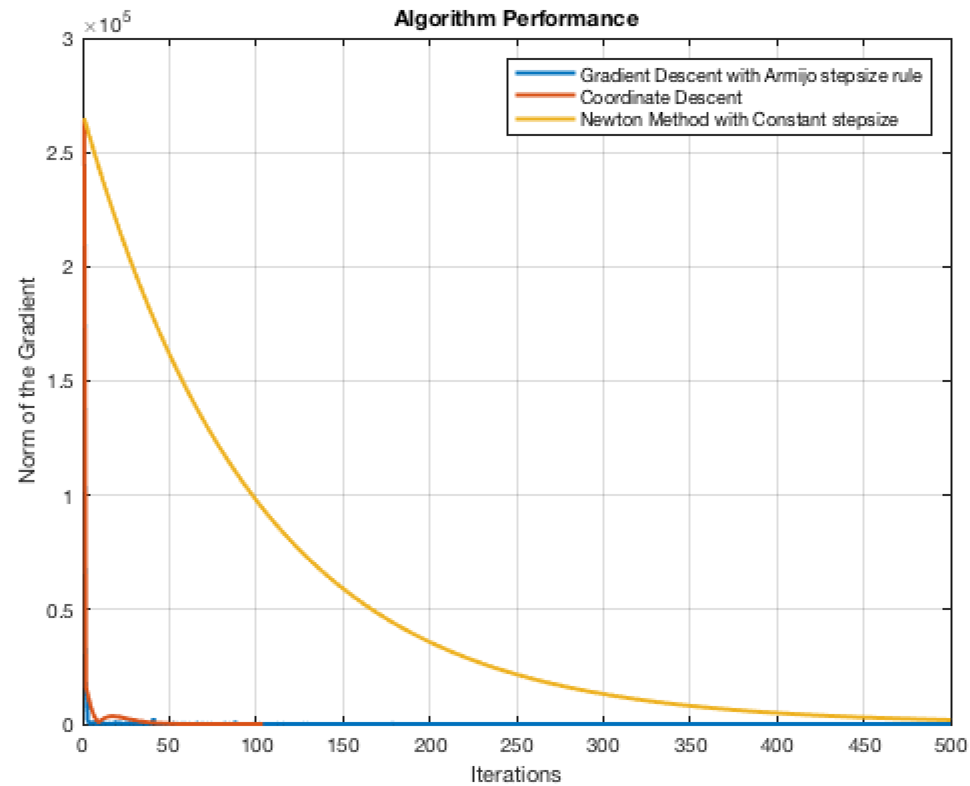
```
NewObj:3.33827e+01
ReductionInObj:2.13417e-01
GradientNorm:17561.42
x(1):-8.3832e-01 | x(2):1.4984e-01 | x(3):5.7573e-01
-----------------------------------------------------
Iter:  316
OldObj:2.64785e+01
NewObj:2.63691e+01
ReductionInObj:1.09474e-01
GradientNorm:11172.32
x(1):-8.8530e-01 | x(2):2.0478e-01 | x(3):5.6489e-01
-----------------------------------------------------
Iter:  361
OldObj:2.29794e+01
NewObj:2.29284e+01
ReductionInObj:5.10053e-02
GradientNorm:7107.67
x(1):-9.1519e-01 | x(2):2.3973e-01 | x(3):5.5799e-01
-----------------------------------------------------
Iter:  406
OldObj:2.14012e+01
NewObj:2.13790e+01
ReductionInObj:2.21527e-02
GradientNorm:4521.80
x(1):-9.3421e-01 | x(2):2.6197e-01 | x(3):5.5360e-01
-----------------------------------------------------
Iter:  451
OldObj:2.07283e+01
NewObj:2.07191e+01
ReductionInObj:9.25422e-03
GradientNorm:2876.70
x(1):-9.4630e-01 | x(2):2.7612e-01 | x(3):5.5081e-01
-----------------------------------------------------
Iter:  496
OldObj:2.04497e+01
NewObj:2.04459e+01
ReductionInObj:3.79602e-03
GradientNorm:1830.12
x(1):-9.5400e-01 | x(2):2.8512e-01 | x(3):5.4903e-01
-----------------------------------------------------
Iter:  541
OldObj:2.03358e+01
NewObj:2.03343e+01
ReductionInObj:1.54489e-03
GradientNorm:1164.29
x(1):-9.5890e-01 | x(2):2.9084e-01 | x(3):5.4790e-01
-----------------------------------------------------
Iter:  586
OldObj:2.02896e+01
NewObj:2.02889e+01
ReductionInObj:6.26677e-04
GradientNorm:740.71
x(1):-9.6201e-01 | x(2):2.9449e-01 | x(3):5.4719e-01
-----------------------------------------------------
```

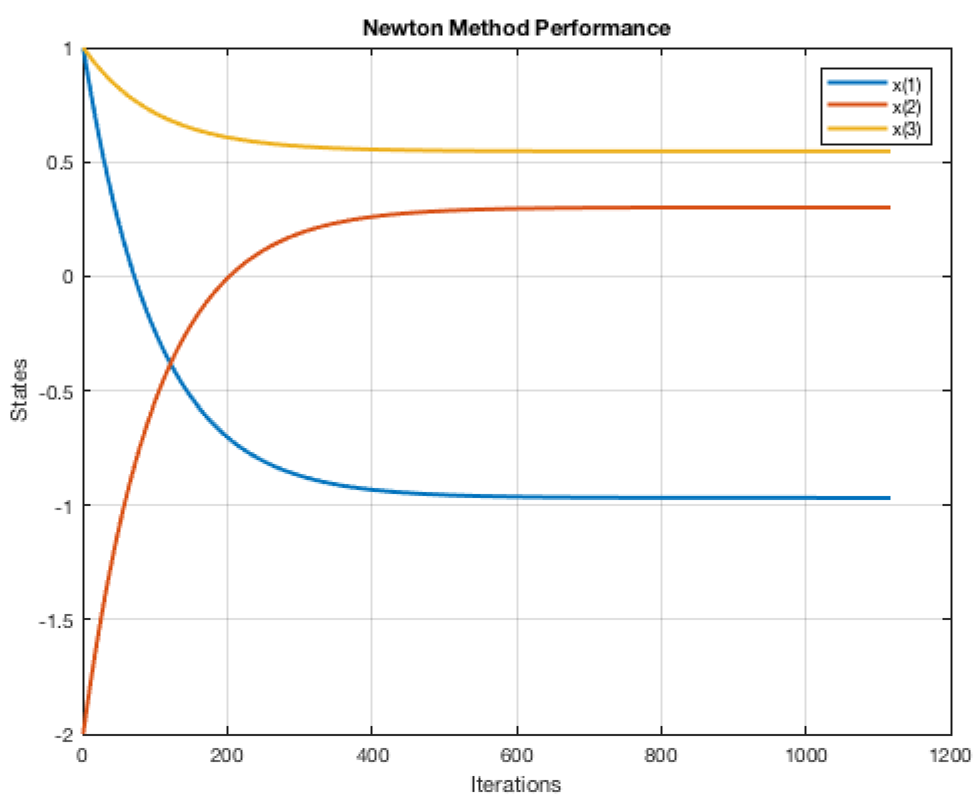
```
Iter:  631
OldObj:2.02708e+01
NewObj:2.02706e+01
ReductionInObj:2.53868e-04
GradientNorm:471.23
x(1):-9.6399e-01 | x(2):2.9680e-01 | x(3):5.4673e-01
-----------------------------------------------------
Iter:  676
OldObj:2.02632e+01
NewObj:2.02631e+01
ReductionInObj:1.02787e-04
GradientNorm:299.79
x(1):-9.6525e-01 | x(2):2.9828e-01 | x(3):5.4644e-01
-----------------------------------------------------
Iter:  721
OldObj:2.02601e+01
NewObj:2.02601e+01
ReductionInObj:4.16073e-05
GradientNorm:190.72
x(1):-9.6606e-01 | x(2):2.9922e-01 | x(3):5.4625e-01
-----------------------------------------------------
Iter:  766
OldObj:2.02589e+01
NewObj:2.02589e+01
ReductionInObj:1.68408e-05
GradientNorm:121.33
x(1):-9.6657e-01 | x(2):2.9981e-01 | x(3):5.4613e-01
-----------------------------------------------------
Iter:  811
OldObj:2.02584e+01
NewObj:2.02584e+01
ReductionInObj:6.81620e-06
GradientNorm:77.19
x(1):-9.6689e-01 | x(2):3.0019e-01 | x(3):5.4606e-01
-----------------------------------------------------
Iter:  856
OldObj:2.02582e+01
NewObj:2.02582e+01
ReductionInObj:2.75876e-06
GradientNorm:49.11
x(1):-9.6710e-01 | x(2):3.0043e-01 | x(3):5.4601e-01
-----------------------------------------------------
Iter:  901
OldObj:2.02581e+01
NewObj:2.02581e+01
ReductionInObj:1.11656e-06
GradientNorm:31.24
x(1):-9.6723e-01 | x(2):3.0059e-01 | x(3):5.4598e-01
-----------------------------------------------------
Iter:  946
OldObj:2.02581e+01
NewObj:2.02581e+01
ReductionInObj:4.51910e-07
GradientNorm:19.88
```

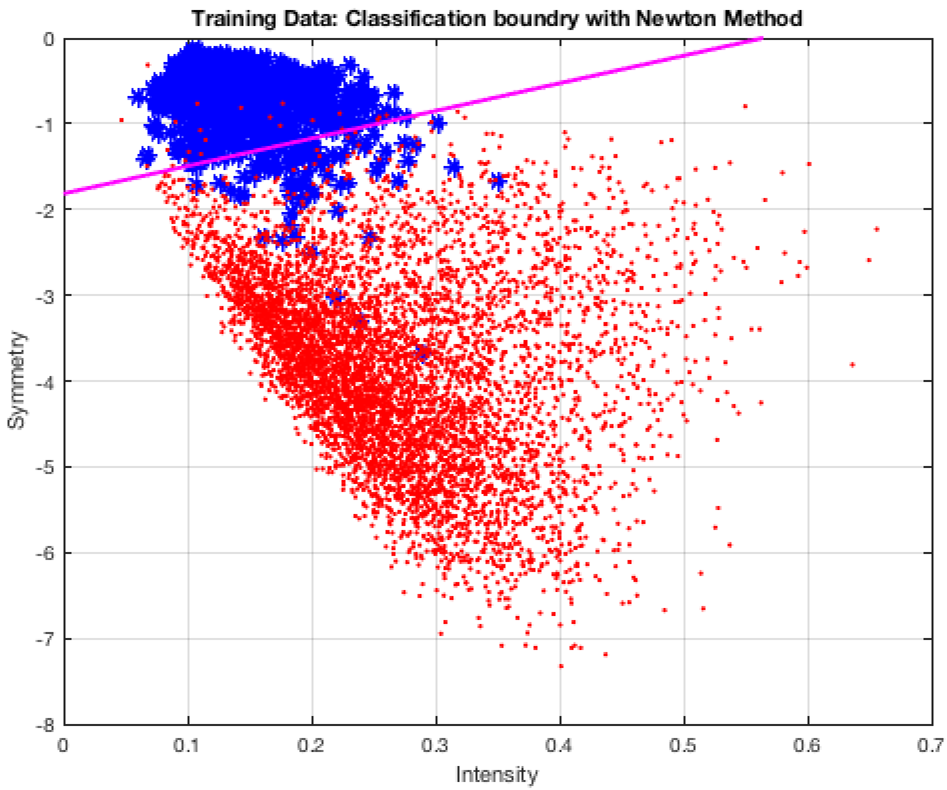
```
x(1):-9.6731e-01 | x(2):3.0068e-01 | x(3):5.4596e-01
----------------------------------------------------
Iter:  991
OldObj:2.02581e+01
NewObj:2.02581e+01
ReductionInObj:1.82902e-07
GradientNorm:12.64
x(1):-9.6737e-01 | x(2):3.0075e-01 | x(3):5.4595e-01
----------------------------------------------------
Iter: 1036
OldObj:2.02580e+01
NewObj:2.02580e+01
ReductionInObj:7.40265e-08
GradientNorm: 8.04
x(1):-9.6740e-01 | x(2):3.0079e-01 | x(3):5.4594e-01
----------------------------------------------------
Iter: 1081
OldObj:2.02580e+01
NewObj:2.02580e+01
ReductionInObj:2.99609e-08
GradientNorm: 5.12
x(1):-9.6742e-01 | x(2):3.0081e-01 | x(3):5.4594e-01
----------------------------------------------------
Iter: 1117
OldObj:2.02580e+01
NewObj:2.02580e+01
ReductionInObj:1.48258e-08
GradientNorm: 3.56
x(1):-9.6743e-01 | x(2):3.0082e-01 | x(3):5.4593e-01
----------------------------------------------------

xStarNewton =

   -0.9674
    0.3008
    0.5459


truePositive =

   226


falsePositive =

     8


falseNegative =

    38


trueNegative =
```

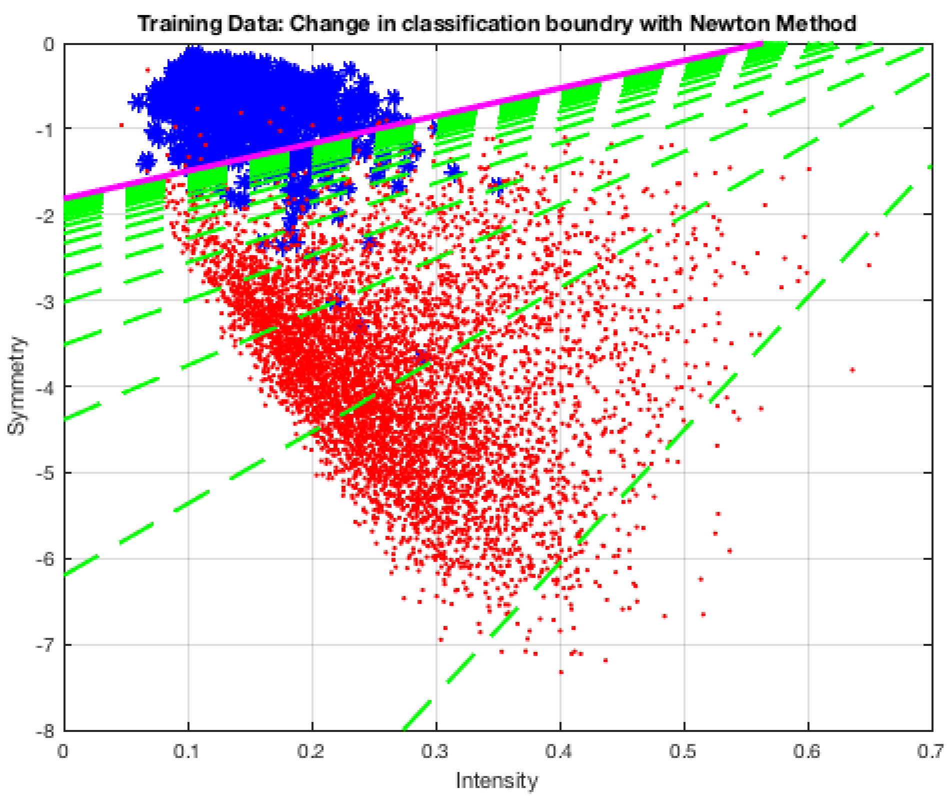
```
    1735

truePositiveRate =

    0.8561

trueNegativeRate =

    0.9954

accuracy =

    0.9771
```

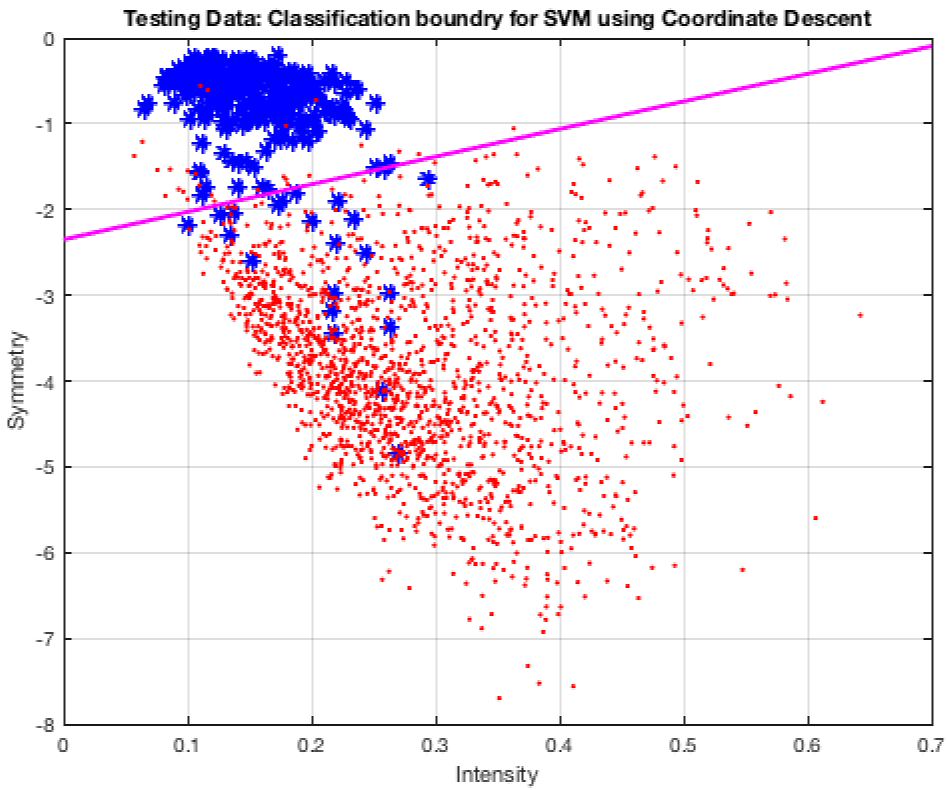



Algorithm Performance
×10^5
Gradient Descent with Armijo stepsize rule
Coordinate Descent
Newton Method with Constant stepsize
Norm of the Gradient
Iterations


Newton Method Performance
x(1)
x(2)
x(3)
States
Iterations


Training Data: Classification boundry with Newton Method
Symmetry
Intensity
0
-1
-2
-3
-4
-5
-6
-7
-8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


Training Data: Change in classification boundry with Newton Method
Symmetry
Intensity
0
-1
-2
-3
-4
-5
-6
-7
-8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7



Published with MATLAB® R2017b