Homework 4

Zachary DeStefano, 15247592

CS 273A: Winter 2015

Due: February 24, 2015

Problem 1

Part a

This is the plot of class 0 versus class 1 using the SVM solver.
The value of b was -17.2697
The w vector was $(6.3572, -5.3693)$

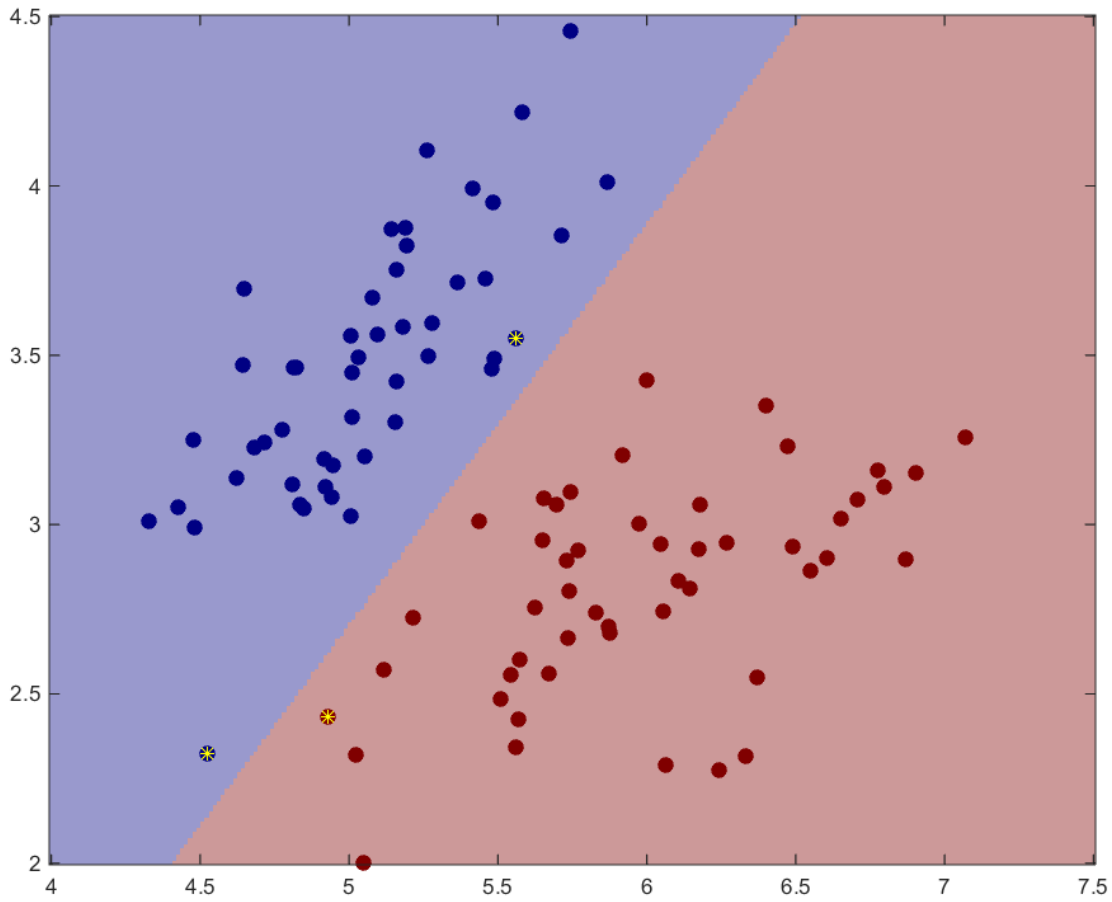

Figure 1: Classification Plot with support vectors starred in yellow

Here is the code to obtain the previous plot. The details of how I transformed into a version compatible with quadprog are in the comments.

```
iris=load('data/iris.txt'); % load the text file
X = iris(:,1:2); Y=iris(:,end); % get first two features

% get class 0 vs 1
XA = X(Y<2,:); YA=Y(Y<2);
sizeXA = size(YA);
numFeatures = sizeXA(1);

Yvar = YA.*2 - 1; %turns 0 and 1 into -1 and 1
%%
%{
we need to minimize sum w_i^2
subject to y(w*x+b) >= 1

The x vector for quadprog is as follows:
b
w_1
w_2

The minimizing matrix H would then be:
0 0 0
0 2 0
0 0 2

f would then be all zeros

I take the negative of the constraint to get
-y(i)*b + -y*w_1*x_1(i) + -y*w_2*x_2(i) <= -1
and that form is compatible with quadprog

each row of A for quadprog is as follows then:
-y(i) -y(i)*x_1(i) -y(i)*x_2(i)

each row of b for quadprog is just -1

%}

%H,f for quadprog as described above
H = [0 0 0; 0 2 0;0 0 2];
f = [0 0 0];

%gets the A matrix for quadprog
Ainit = [ones(numFeatures,1) XA];
A = -1.*Ainit.*(repmat(Yvar,1,3));

%gets the b column
b = ones(numFeatures,1).*-1;

theta = quadprog(H,f,A,b);
```

This is the code to obtain α, verify it, and then plot the classification bounday.

```
%%

%{
We have to change the optimization function to
min alpha>=0 1/2 sum{alpha_i alpha_j y(i) y(j) K_ij} - sum alpha_i

Details below on what the variables become in order to
    compute the quadratic program in this form
%}

%this lets us construct the dot product matrix K
Kmat = XA*XA';

%this constructs the matrix H for the program
yMat = Yvar*Yvar';
Hmatrix = yMat.*Kmat;

%this is the f vector
fVec = -ones(numFeatures,1);

%this is the A matrix, which will also be Aeq
Amat = Yvar';

%this is the b value for the input
bVal = 0;

%the lower bound is zero
LBvec = zeros(numFeatures,1);

%run quadprog on dual form finally
alpha = quadprog(Hmatrix,fVec,[],[],Amat,bVal,LBvec);

%%

%this part verifies the alpha solution and then plots the
%       boundary and support vectors

%gets b,w from theta
bFromTheta = theta(1);
wFromTheta = theta(2:3);

%due to floating part error, this is just to indicate almost zero
epsilon = 0.001;

%I find the support vectors here
%the alphas that are greater than 0 indicate support vectors
supportVecInds = find(alpha>epsilon);
supportVecs = XA(supportVecInds,:);
supportVecsY = Yvar(supportVecInds);

%verify w obtained from alpha
wFromAlpha = (alpha.*Yvar)'*XA;
```

```
diffBetweenW = sum(abs(wFromAlpha'-wFromTheta));
assert(diffBetweenW < epsilon); %no assertion failed occured, so this is true

%verify b obtained from alpha
bFromAlpha = mean(supportVecsY-supportVecs*wFromAlpha');
diffBetweenB = abs(bFromAlpha-bFromTheta);
assert(diffBetweenB < epsilon); %no assertion fail

%plots the classification boundary finally
learnerA=logisticClassify();
learnerA=setClasses(learnerA, unique(YA));
learnerA=setWeights(learnerA, theta');
plotClassify2D(learnerA,XA,YA);
hold on
plot(supportVecs(:,1),supportVecs(:,2),'y*');
```

Problem 2

Part a

To calculate the entropy, I did the following

$$H(y) = p(y=1)log_2(\frac{1}{p(y=1)}) + p(y=-1)log_2(\frac{1}{p(y=-1)})$$

The entropy ends up being 0.9743

Part b

You should split on feature 2 first. Here is the information gain for all the variables:

Feature	Information Gain
1	0.0245
2	0.5059
3	-0.0097
4	0.0930
5	-0.0095

Code for a and b

This is the code to complete Part A and B. The code for the getEntropy function will be shown later.

```
%%
%sets up the data
xyData=[
0 0 1 1 0 -1;
1 1 0 1 0 -1;
0 1 1 1 1 -1;
1 1 1 1 0 -1;
0 1 0 0 0 -1;
1 0 1 1 1 1;
0 0 1 0 0 1;
1 0 0 0 0 1;
1 0 1 1 0 1;
1 1 1 1 1 -1];

yVec = xyData(:,6);
xVals = xyData(:,1:5);
numVals = size(yVec);

%%
%Part A, calculates the entropy
probClass1 = size(find(yVec==1))/numVals;
entropy = getEntropy(probClass1);

%Part B, calculates the information gain for each variable
information = zeros(1,5);
for xCol = 1:5
    xData = xVals(:,xCol);
    probX1 = size(find(xData==1))/numVals;
    probX0 = 1-probX1;
    yDataForX1 = yVec(xData==1);
    yDataForX0 = yVec(xData==0);

    probClass1ForX1 = size(find(yDataForX1==1))/size(yDataForX1);
    newEntropyX1 = getEntropy(probClass1ForX1);

    probClass1ForX0 = size(find(yDataForX0==1))/size(yDataForX0);
    newEntropyX0 = getEntropy(probClass1ForX0);

    information(xCol) = probX1*(entropy-newEntropyX1)+probX0*(entropy-newEntropyX0);
end

[gain,colNum0] = max(information);

%information gain for all variables
information

%feature to split on initially
colNum0
```

Part C

Here is the decision tree for Part C

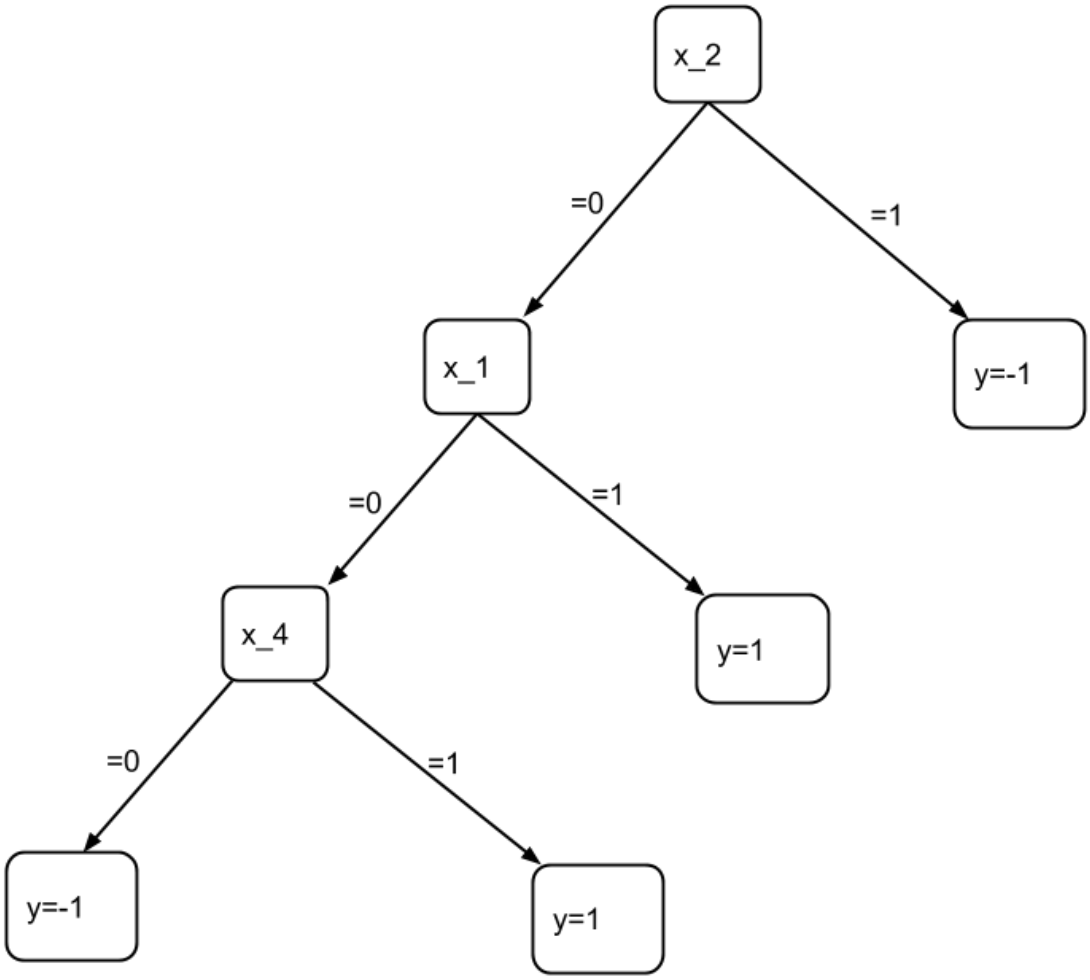

Figure 2: Decision Tree

Here is the code used in Part C. It relies on the data set up from Part A.

```
%%

%Part C, calculates the decision tree

%This does the first split
[newXDataClass0, newXDataClass1,newYvecClass0, newYvecClass1...
    ,maxInfoGain,colNum ] = getDecTreeSplit( xVals,yVec );

%{
    In the first split, we use x_2
    When x_2 = 1, y=-1 so we don't need to traverse furthur down that node
    We will now split x_2=0
%}

[newXData2Class0, newXData2Class1,newYvec2Class0, newYvec2Class1...
    ,maxInfoGain2,colNum2 ] = getDecTreeSplit( newXDataClass0,newYvecClass0 );

%{
  This tells us to split on x_1
  When x_2=0,x_1=1, it happens that y=1 so we do not traverse furthur down
    We will now split x_2=0,x_1=0
%}

[newXData3Class0, newXData3Class1,newYvec3Class0, newYvec3Class1...
    ,maxInfoGain3,colNum3 ] = getDecTreeSplit( newXData2Class0,newYvec2Class0 );

%{
    This tells us to split on x_4
    When x_2=0,x_1=0,x_4=0 y=1
    When x_2=0,x_1=0,x_4=1 y=-1

    We are now done
%}
```

Matlab functions written for Problem 2

This is the code for the getEntropy function

```
function [ entropy ] = getEntropy( probClass1)
%GETENTROPY Summary of this function goes here
%   Detailed explanation goes here

probClass0 = 1-probClass1;
entropy = 0;
if probClass1 > 0
   entropy = entropy + probClass1*log(probClass1);
end
if probClass0 > 0
   entropy = entropy + probClass0*log(probClass0);
end
entropy = -1*entropy/log(2);

end
```

This is the code for the getDecTreeSplit function

```
function [ newXDataClass0, newXDataClass1,newYvecClass0, newYvecClass1,maxInfoGain,colNum ]...
    = getDecTreeSplit( xVals,yVec )
%GETDECTREESPLIT Summary of this function goes here
%   Detailed explanation goes here

sizeX = size(xVals);
numVals = size(yVec);
numFeatures = sizeX(2);

probClass1 = size(find(yVec==1))/numVals;

if probClass1 <= 0
    newXDataClass0 = xVals;
    newYvecClass0 = yVec;
    colNum = 0;
   return
elseif probClass1 >= 1
    newXDataClass1 = xVals;
    newYvecClass1 = yVec;
    colNum = 0;
    return
end

%calculates entropy.
entropy = getEntropy(probClass1);

maxInfoGain=-Inf;

for xCol = 1:numFeatures
    xData = xVals(:,xCol);
    probX1 = size(find(xData==1))/numVals;
    probX0 = 1-probX1;
    yDataForX1 = yVec(xData==1);
    yDataForX0 = yVec(xData==0);

    probClass1ForX1 = size(find(yDataForX1==1))/size(yDataForX1);
    newEntropyX1 = getEntropy(probClass1ForX1);

    probClass1ForX0 = size(find(yDataForX0==1))/size(yDataForX0);
    newEntropyX0 = getEntropy(probClass1ForX0);

    information = probX1*(entropy-newEntropyX1)+probX0*(entropy-newEntropyX0);
    if(information > maxInfoGain)
       maxInfoGain = information;
       colNum = xCol;
       newXDataClass0 = xVals(xData==0,:);
       newXDataClass1 = xVals(xData==1,:);
       newYvecClass0 = yVec(xData==0);
       newYvecClass1 = yVec(xData==1);
    end
end
```

Problem 3

Part a

The validation MSE that I obtained was 0.7133.
This is the code I used to obtain it:

```
%%
Xte = load('kaggle/kaggle.X1.test.txt');
Xtr = load('kaggle/kaggle.X1.train.txt');
Ytr = load('kaggle/kaggle.Y.train.txt');

%%

%Part A
[Xtrain,Xvalid,Ytrain,Yvalid] = splitData(Xtr,Ytr,0.8);
dt = treeRegress(Xtrain,Ytrain, 'maxDepth',20);
mseTree = mse(dt,Xvalid,Yvalid);
```

Part b

At the maxDepth parameter increases, the model grows increasingly complex. This is due to the fact that more nodes can be added to the decision tree. Just like with other models, it does begin to overfit as we add complexity. The lowest validation MSE is obtained when maxDepth is 8 and after that overfitting seems to begin. The validation MSE at 8 ends up being 0.4355. Here is the plot which illustrates that below.

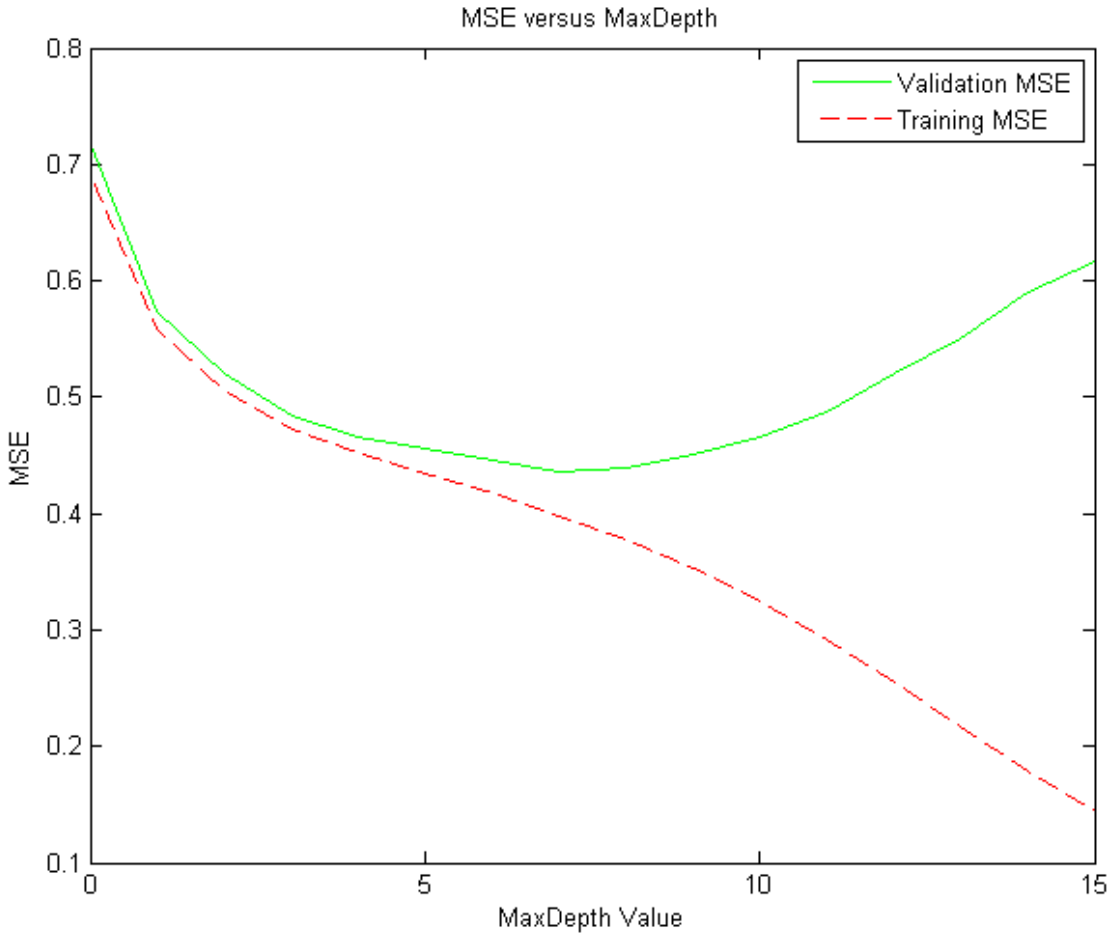

Figure 3: MSEs for the Decision Trees with varying maxDepth

Here is the code used for Part b

```
%%

%Part B
validMSE = ones(1,16);
trainMSE = ones(1,16);
index = 1;
for depth = 0:15
    dt = treeRegress(Xtrain,Ytrain, 'maxDepth',depth);
    trainMSE(index) = mse(dt,Xtrain,Ytrain);
    validMSE(index) = mse(dt,Xvalid,Yvalid);
    index = index + 1;
end

plot(0:15,validMSE,'g-')
hold on
plot(0:15,trainMSE','r--')
xlabel('MaxDepth Value');
ylabel('MSE');
title('MSE versus MaxDepth');
legend('Validation MSE','Training MSE');
[minMSE,indexOfMinMSE] = min(validMSE);
depthOfMinMSE = indexOfMinMSE + 1;
```

Part c

With the minParent parameter, the model grows decreasingly complex as minParent grows. This is due to the fact that the number of data points required for a new split increases, hence less splits will be made. Thus it goes from overfitting to underfitting as minParent grows. It is thus overfitting in the first few steps and underfitting in the last few steps. The best validation MSE is obtained in the middle when minParent is equal to 2^9 with an MSE of 0.4306. Here is the plot which illustrates this:

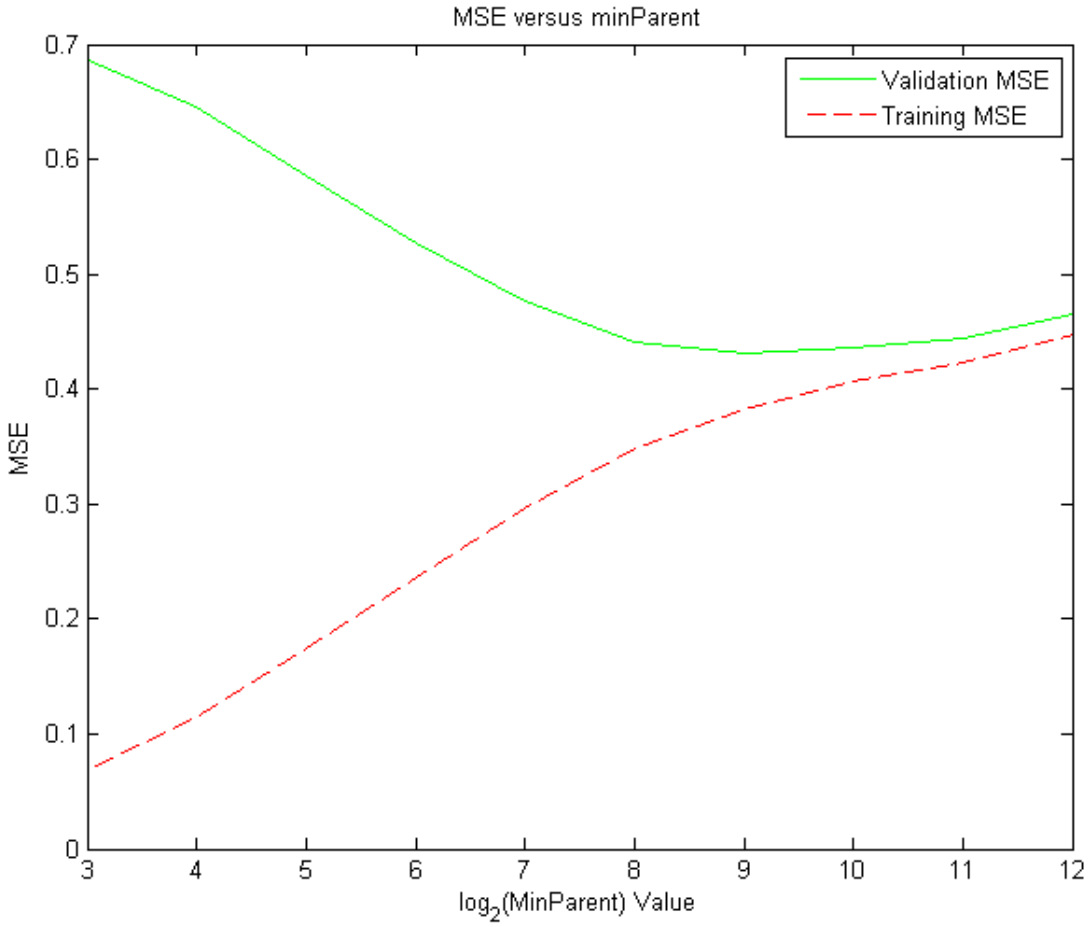

Figure 4: MSEs for the Decision Trees with varying minParent

Here is the code used for Part c

```
%%

%Part C

minParentVals = 2.^(3:12);
validMSE2 = zeros(1,10);
trainMSE2 = zeros(1,10);
for val = 1:10
    dt = treeRegress(Xtrain,Ytrain,'maxDepth',20,'minParent',minParentVals(val));
    trainMSE2(val) = mse(dt,Xtrain,Ytrain);
    validMSE2(val) = mse(dt,Xvalid,Yvalid);
end

plot(3:12,validMSE2,'g-');
hold on
plot(3:12,trainMSE2,'r--');
xlabel('log_2(MinParent) Value');
ylabel('MSE');
title('MSE versus minParent');
legend('Validation MSE','Training MSE');
[minMSE2,indexOfMinMSE2] = min(validMSE2);
```

Part d

I attempted 3 different calls to treeRegress with the following parameters:
1. maxDepth=9, minParent=2^9
2. maxDepth=20, minParent=2^9
3. maxDepth=9

With call 1, the Kaggle score was 0.65140
With call 2, the Kaggle score was 0.64843, which was the best one of the three
With call 3, the Kaggle score was 0.66217

Here is the code for part d

```
%%

%Part D

%Call 1: Kaggle score was 0.65140
%dt = treeRegress(Xtr,Ytr,'maxDepth',9,'minParent',2^9);

%Call 2: Kaggle score was 0.64843, the best one
dt = treeRegress(Xtr,Ytr,'maxDepth',20,'minParent',2^9);

%Call 3: Kaggle score was 0.66217
%dt = treeRegress(Xtr,Ytr,'maxDepth',9);

Yhat = predict(dt,Xte);

fh = fopen('kagglePrediction.csv','w');  % open file for upload
fprintf(fh,'ID,Prediction\n');      % output header line
for i=1:length(Yhat),
    fprintf(fh,'%d,%d\n',i,Yhat(i));  % output each prediction
end;
fclose(fh);                         % close the file
```

Problem 4

Part a

I decided to do Option 2 and employ Gradient Boosting to try to enhance my predictions. Here are the errors I got for 1,5,10,25 learners:

Number of Learners	Training MSE	Validation MSE
1	0.6203	0.6461
5	0.4933	0.4386
10	0.4487	0.4136
25	0.4283	0.3923

Here is the plot of training and validation MSE:

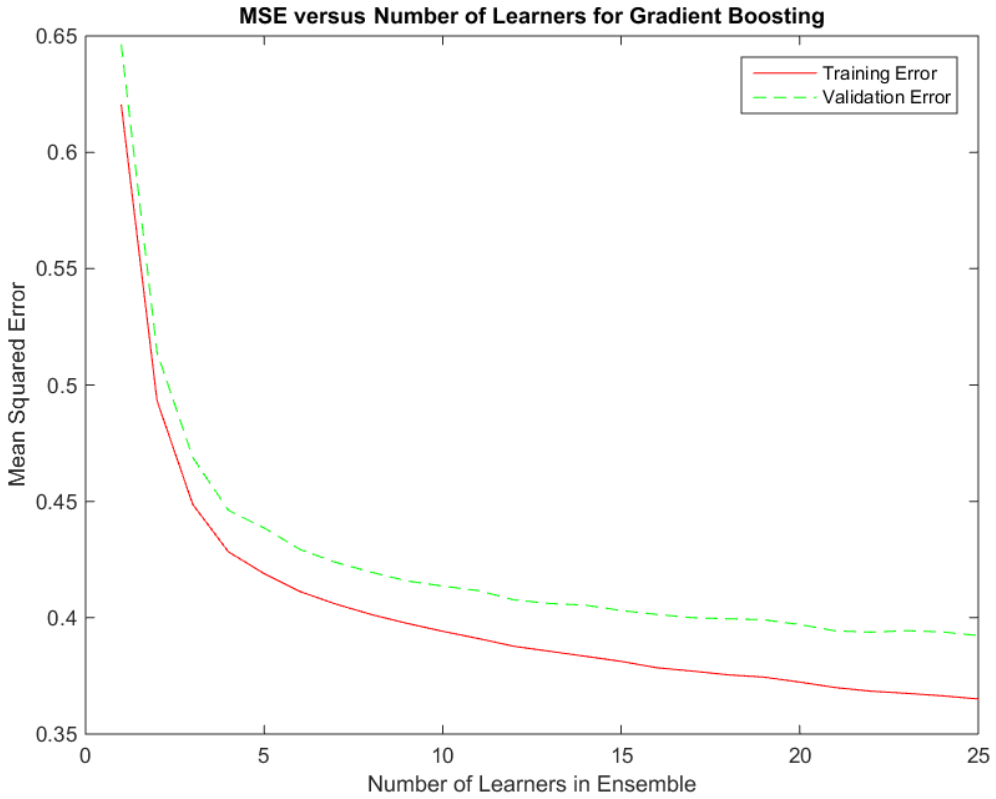

Figure 5: MSE versus number of learners

Here is the code for Part A, which uploaded the data, trained the ensembles, and then graphed the results.

```
%%

%run this when loading on a new computer
Xte = load('kaggle/kaggle.X1.test.txt');
Xtr = load('kaggle/kaggle.X1.train.txt');
Ytr = load('kaggle/kaggle.Y.train.txt');
save('kaggleData.mat','Xte','Ytr','Xtr');

%%
%run this if above was already run on this computer
load('kaggleData.mat');

%%
[Xtrain,Xvalid,Ytrain,Yvalid] = splitData(Xtr,Ytr,0.8);

%number of ensembles
N = 200;

mseTraining = zeros(1,N);
mseValidation = zeros(1,N);

%alpha values
alpha = 0.25*ones(1,N);
dt = cell(1,N);

predictY = 0;
curY = 0;

for k=1:N,

 grad = 2*(curY - Ytrain);
 dt{k} = treeRegress(Xtrain,grad,'maxDepth',3);
 curY = curY - alpha(k) * predict(dt{k}, Xtrain);

 %find training MSE at k
 mseTraining(k) = mean((curY-Ytrain).^2);

 %find validation MSE
 predictY = predictY - alpha(k)*predict(dt{k}, Xvalid);
 mseValidation(k) = mean((Yvalid-predictY).^2);

end;

%%
plot(mseTraining,'r-');
hold on
plot(mseValidation,'g--');
xlabel('Number of Learners in Ensemble');
ylabel('Mean Squared Error');
legend('Training Error','Validation Error');
title('MSE versus Number of Learners for Gradient Boosting');
```

Part b

After having over 200 learners trained and seeing the validation error, the lowest validation error was reached at 0.3787 when there were 82 learners. I have decided to retrain on all the test data and have it learn 100 learners before running it on the test data.

After uploading to Kaggle, my score was 0.61760.

Here is the code for Part B, which trained the ensembles on all the training data and then made the Kaggle file. It uses some of the variables in Part A that were created to hold the data.

```
%%

%train on all the test data

N=100; %new number of ensembles
curY=0;
predictY=0;
for k=1:N,

 %train the k-th decision tree
 grad = 2*(curY - Ytr);
 dt{k} = treeRegress(Xtr,grad,'maxDepth',3);
 curY = curY - alpha(k) * predict(dt{k}, Xtr);

 %boost current prediction using that k-th decision tree
 predictY = predictY - alpha(k)*predict(dt{k}, Xte);

end;

%%
fh = fopen('kagglePrediction.csv','w');  % open file for upload
fprintf(fh,'ID,Prediction\n');      % output header line
for i=1:length(predictY),
    fprintf(fh,'%d,%d\n',i,predictY(i));  % output each prediction
end;
fclose(fh);                         % close the file
```