ROCHESTER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING
CMPE 677 Machine Intelligence
HW #5

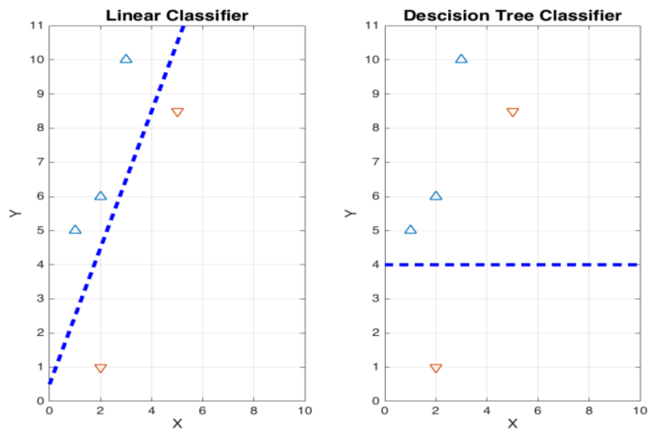
Name:_________Kaiwen Zheng______________

Due 11:55pm, 10/13/2017, submit via Dropbox. Work alone. All questions pertain to Matlab/Octav.

1. (5 pts) A decision tree with a single node can be viewed as a linear classifier. Consider a problem with *n* samples, each having two dimensions: $x^{(i)} \epsilon \{x_1^{(i)},x_2^{(i)}\}$, *i*=1…*n*, and two classes *c* ϵ {'+', '-'}. Draw a plot, showing *n* samples, $n \leq 5$, for which a linear classifier (read as mx+b classifier) can do a perfect job splitting the two classes, but our single node decision tree cannot split these classes? Given *n* points, what is the maximum number of tree nodes to do perfect classification?

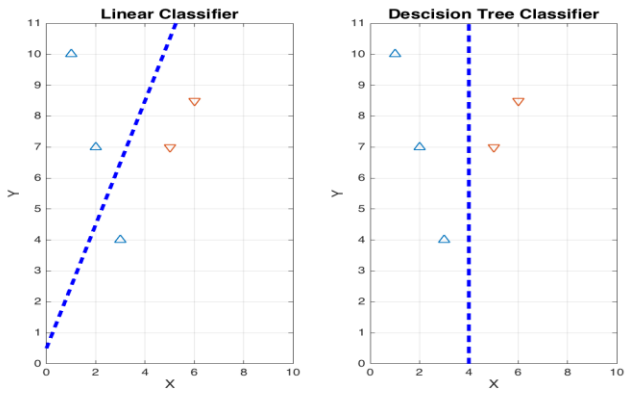
Ans:

Given n points, what is the maximum number of tree nodes to do perfect classification is n-1.



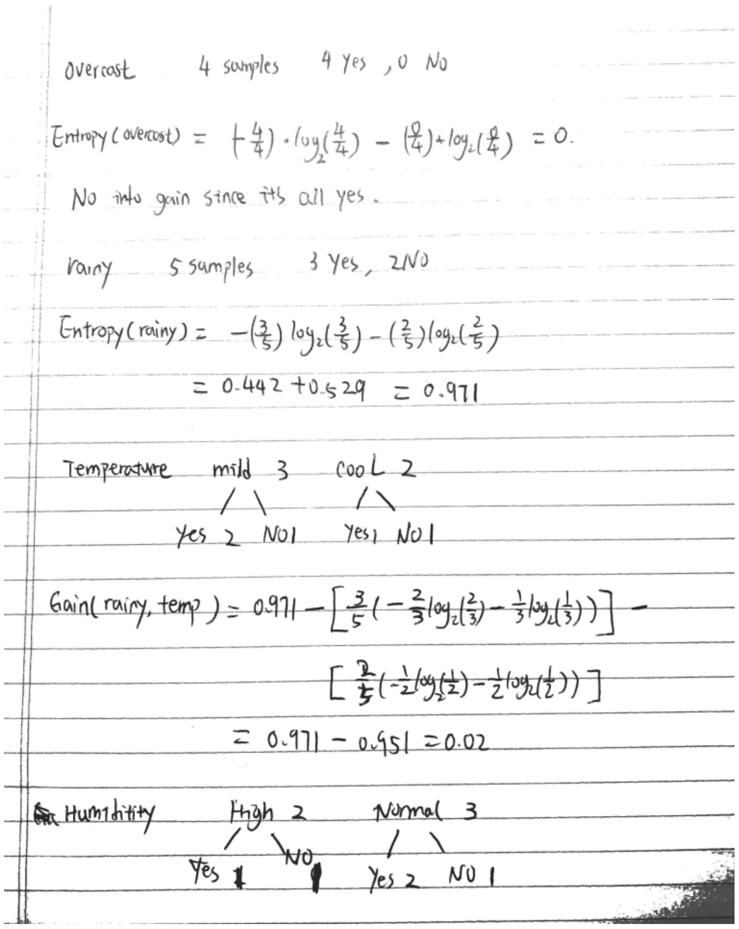
2. (5 pts) If we allow our decision tree to arbitrary number of nodes (splits), it can handle more complex datasets. Once again, we have *n* samples, each having two dimensions: $x^{(i)} \epsilon \{x_1^{(i)},x_2^{(i)}\}$, *i*=1…*n*, and two classes *c* ϵ {'+', '-'}. Draw a plot, showing *n* samples, $n \leq 5$, for which a linear classifier (read as mx+b classifier) cannot do a perfect job splitting two classes, but our multiple node decision tree can split these classes?

Ans:



3. (15 pts) Using ID3 classification trees along with the “ID3 Example” in the class slides, expand the node branches for Outlook=“Overcast” and Outlook=”Rain” one more split, showing what feature (if any) to split on next. As in class notes, use Entropy and Gain. Remember, once a feature is used in a split, it is no longer available for further splits in ID3. What feature is split on in the Overcast and Rain nodes? Show all work neatly.

Ans:



overcast 4 samples 4 yes , 0 No

Entropy (overcast) $= (-\frac{4}{4}) \cdot \log_2(\frac{4}{4}) - (\frac{0}{4}) * \log_2(\frac{0}{4}) = 0.$

No info gain since it's all yes.

rainy 5 samples 3 yes, 2NO

Entropy(rainy) $= -(\frac{3}{5}) \log_2(\frac{3}{5}) - (\frac{2}{5}) \log_2(\frac{2}{5})$

$= 0.442 + 0.529 = 0.971$

Temperature mild 3 cooL 2

Yes 2 NO1 Yes1 NO1

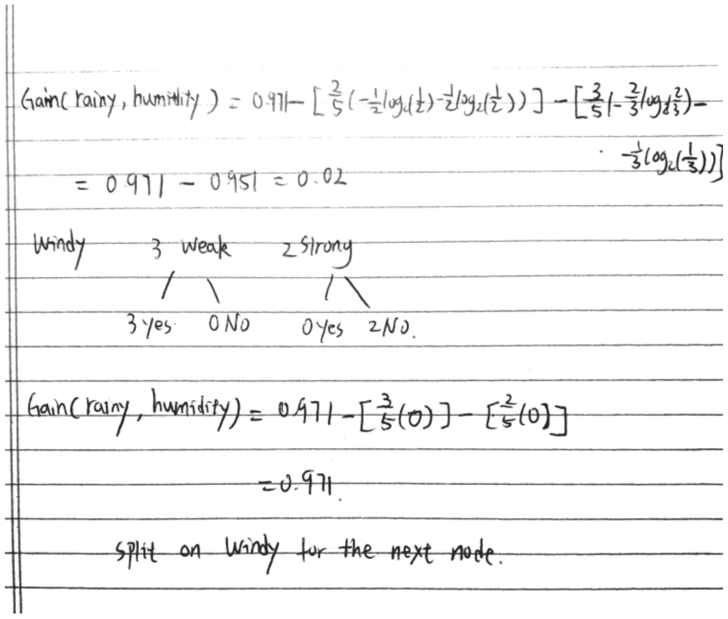
Gain(rainy, temp) $= 0.971 - \left[\frac{3}{5}\left(-\frac{2}{3}\log_2(\frac{2}{3}) - \frac{1}{3}\log_2(\frac{1}{3})\right)\right] -$

$\left[\frac{2}{5}\left(-\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2})\right)\right]$

$= 0.971 - 0.951 = 0.02$

Humiditity High 2 Normal 3

Yes 1 NO 1 Yes 2 NO 1



Gain(rainy, humidity) $= 0.971 - \left[\frac{2}{5}\left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right)-\frac{1}{2}\log_2\left(\frac{1}{2}\right)\right)\right] - \left[\frac{3}{5}\left(-\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right)\right)\right]$

$= 0.971 - 0.951 = 0.02$

Windy 3 weak 2 Strong

3 yes 0 No 0 yes 2 No.

Gain(rainy, humidity) $= 0.971 - \left[\frac{3}{5}(0)\right] - \left[\frac{2}{5}(0)\right]$

$= 0.971.$

split on Windy for the next node.

4. (15 pts) This problem introduces classification tree support. Study this baseline code:

```
clear ; close all; clc

% Load Training Data- Andrew Ng Machine Learning MOOC
load('ex3data1.mat'); % training data stored in arrays X, y (this data was given in hwk #4)
n = size(X, 1);
num_labels =  length(unique(y));          % 10 labels, from 1 to 10   (note  "0" is mapped to label
10)

% Randomly select 100 data points to display
rng(2000);  %random number generator seed
rand_indices = randperm(n);
sel = X(rand_indices(1:100), :);

%displayData(sel);
%print -djpeg95 hwk4_4.jpg

Xdata =X;

%ClassificationTree
tree = ClassificationTree.fit(Xdata,y);
maxprune = max(tree.PruneList);
treePrune = prune(tree,'level',maxprune-3);
view(treePrune,'mode','graph');
pred = predict(tree,Xdata);
subplot(2,1,1);
hold off; plot(y,'g','linewidth',2); hold on; plot(pred,'b','linewidth',2);
subplot(2,1,2);
plot(y-pred,'k','linewidth',3);
mse = (1/length(y))*sum((y-pred).^2);  %0.7964
fprintf('MSE for regular classification tree: %f \n ',mse);
```

```
%BaggedTree
rng(2000);  %random number generator seed
t = ClassificationTree.template('MinLeaf',1);
bagtree = fitensemble(Xdata,y,'Bag',10,t,'type','classification');
ypred = predict(bagtree,Xdata);  %really should test with a test set here
figure
subplot(2,1,1);
hold off; plot(y,'g','linewidth',2); hold on; plot(ypred,'b','linewidth',2);
subplot(2,1,2);
plot(y-ypred,'k','linewidth',3);
msebag = (1/length(y))*sum((y-ypred).^2);  %0.0884
fprintf('MSE for bagged classification tree: %f \n ',msebag);
```

Complete the two new methods to the sample classification code below for the cases ‘ClassificationTree’, and ‘BaggedTree’ using the parameters as above (and make sure you use rng(2000) for BaggedTree). Report the accuracy and confidence matrix for each. (No need to implement or show plots.) Show your code which should now support LogisticRegression, KNN, ClassificationTree, and BaggedTree.
Sample classification code:

```
clear ; close all; clc

% Load Training Data- Andrew Ng Machine Learning MOOC
load('ex3data1.mat'); % training data stored in arrays X, y
n = size(X, 1);
num_labels =  length(unique(y));          % 10 labels, from 1 to 10   (note  "0" is mapped to label
10)

% Randomly select 100 data points to display
rng(2000);  %random number generator seed
rand_indices = randperm(n);
sel = X(rand_indices(1:100), :);

%displayData(sel);
%print -djpeg95 hwk4_4.jpg

Xdata = [ones(n, 1) X];
% the matlab functions you want to use are crossvalind.m and confusionmat.m_
% Xdata- A vector of feature, nxD, one set of attributes for each dataset sample
% y- A vector of ground truth labels, nx1 (each class has a unique integer value), one label for
each dataset sample
% numberOfFolds- the number of folds for k-fold cross validation
numberOfFolds=5;
rng(2000);  %random number generator seed
CVindex = crossvalind('Kfold',y, numberOfFolds);

method='ClassificationTree'

lambda = 0.1;
for i = 1:numberOfFolds
   TestIndex = find(CVindex == i);
   TrainIndex = find(CVindex ~= i);
```

```
    TrainDataCV = Xdata(TrainIndex,:);
    TrainDataGT =y(TrainIndex);

    TestDataCV = Xdata(TestIndex,:);
    TestDataGT = y(TestIndex);

    %
    %build the model using TrainDataCV and TrainDataGT
    %test the built model using TestDataCV
    %
    switch method
        case 'LogisticRegression'
            % for Logistic Regression, we need to solve for theta
            % Initialize fitting parameters
            all_theta = zeros(num_labels, size(Xdata, 2));

            for c=1:num_labels
                % Set Initial theta
                initial_theta = zeros(size(Xdata, 2), 1);
                % Set options for fminunc
                options = optimset('GradObj', 'on', 'MaxIter', 50);

                % Run fmincg to obtain the optimal theta
                % This function will return theta and the cost
                [theta] = ...
                    fmincg (@(t)(costFunctionLogisticRegression(t, TrainDataCV, (TrainDataGT == c),
lambda)), ...
                    initial_theta, options);

                all_theta(c,:) = theta;
            end

            % Using TestDataCV, compute testing set prediction using
            % the model created
            % for Logistic Regression, the model is theta
            % Insert code here to see how well theta works...
            all_pred = sigmoid(TestDataCV*all_theta');
            [maxVal,maxIndex] = max(all_pred,[],2);
            TestDataPred=maxIndex;

        case 'KNN'
            [idx, dist] = knnsearch(TrainDataCV,TestDataCV,'k',3);
            TestDataPred=mode([TrainDataGT(idx(:,1)) TrainDataGT(idx(:,2))  TrainDataGT(idx(:,3))
]')';

         case 'ClassificationTree'


         case 'BaggedTree'
            rng(2000);  %random number generator seed


        otherwise
```

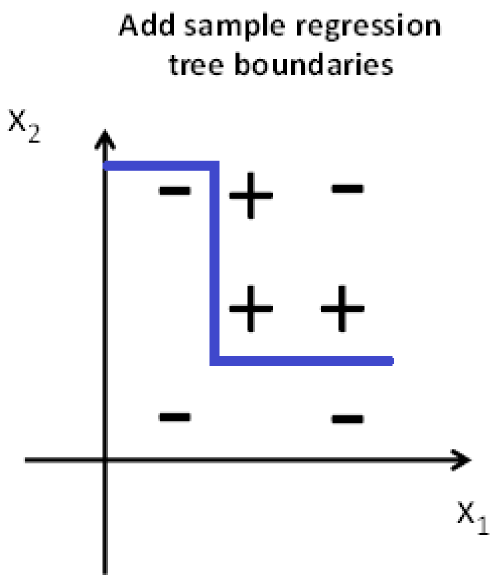
```
            error('Unknown classification method')
    end

    predictionLabels(TestIndex,:) =double(TestDataPred);
end

confusionMatrix = confusionmat(y,predictionLabels);
accuracy = sum(diag(confusionMatrix))/sum(sum(confusionMatrix));

fprintf(sprintf('%s: Lambda = %d, Accuracy = %6.2f%%%%% \n',method, lambda,accuracy*100));
fprintf('Confusion Matrix:\n');
[r c] = size(confusionMatrix);
for i=1:r
    for j=1:r
        fprintf('%6d ',confusionMatrix(i,j));
    end
    fprintf('\n');
end
```

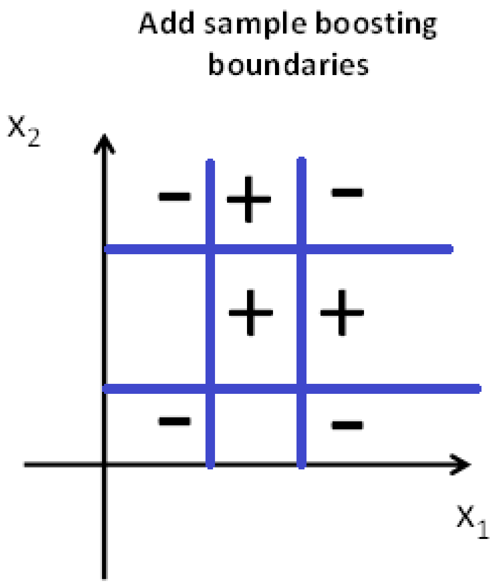
Ans (show accuracy and confidence for ClassificationTree and BaggedTree; show code):

```
ClassificationTree: Lambda = 1.000000e-01, Accuracy =  77.16%

Confusion Matrix:

   432     10     14      6      6      3     12     14      2      1

    16    352     27     16     14      9     21     29      8      8

    12     31    349      5     31      1     21     33     16      1

    10     11      8    388     14     21      7     13     25      3

    17     15     34     13    343     15      6     33     19      5

     8     20      4     12     20    412      1     17      1      5

    11     14      6      8     10      1    418      6     25      1

    21     40     26     14     17     11      2    353     16      0

     6      7     14     27     22      5     31     23    361      4

     0     13      6      2      7      6      7      6      3    450



BaggedTree: Lambda = 1.000000e-01, Accuracy =  89.24%

Confusion Matrix:
   479    4    3    1    2    1    1    9    0    0

     4  431    7    9    5   11   10   11    2   10

     5   12  416    4   20    5   10   17    9    2

     4    6    0  446    3   11    0    3   25    2

     2    2   19    7  433    8    1   13   11    4

     2    9    0    5   11  468    0    2    0    3

     7    7    3   12    1    0  449    3   16    2

     8    9   20    5   13    7    1  423   11    3

     2    2    8   20    2    4    9    7  440    6

     1    4    1    2    4    5    2    4    0  477
```

```
case 'ClassificationTree'
    tree = ClassificationTree.fit(TrainDataCV,TrainDataGT);
    maxprune = max(tree.PruneList);
    treePrune = prune(tree,'level',maxprune-3);
    view(treePrune,'mode','graph');
    TestDataPred = predict(tree,TestDataCV);
case 'BaggedTree'
    rng(2000); %random number generator seed
    t = ClassificationTree.template('MinLeaf',1);
    bagtree =
fitensemble(TrainDataCV,TrainDataGT,'Bag',10,t,'type','classification')
;
    TestDataPred = predict(bagtree,TestDataCV); %really should test
with a test set here
```

5. (5 points) Regression trees perform classification by continually subdividing the training space into sub-groups. Boosting classifiers use thresholds which cover all space, and determine classification using a sum of weighted thresholds. The figures below show a dataset of seven samples, each having an x_1 and x_2 parameter. On the figure on the left, draw sample boundaries that a regression tree might use. On the figure on the right, draw sample boundaries that a boosting classifier might use.





6. This problem continues the continual improvement of our cross validation classification code you have been developing. We will turn our Matlab script into a function, write scripts to call this function, and implement the AdaBoost algorithm using a decision stump as a weak classifier. We will test our code on the Bupa Liver Disorder dataset. This dataset predicts whether an individual has a liver disorder based upon several blood tests and level of alcohol consumption. The columns are {mean corpuscular volume, alkaline phosphoase, alamine aminotransferase, aspartate aminotransferase, gamma-glutamyl transpeptidase, number of half-pint alcoholic beverages drunk per day} This problem is divided into several sections, please complete them sequentially.

6.a (5pts) Download bupa.data and classify677_hwk5.m files from MyCourses, they are part of hwk5files_forMycourses.zip. Note that classify677_hwk5.m is functional version of the cross validation classification function developed in hwk4. Starting with the sample code (note- you have to change the 'cd' and 'addpath' lines of code to match the paths on your system), verify you get the accuracy and confusion matrices as shown.

```
% load the bupa data
% cd to location where hwk5 files are
cd C:\Users\rwpeec\Desktop\rwpeec\rit\CMPE-677_MachineIntelligence\hwk\hwk5
% add path to where hwk4 files are
addpath C:\Users\rwpeec\Desktop\rwpeec\rit\CMPE-677_MachineIntelligence\hwk\hwk4
close all ; clear all;
b = load('bupa.data');
X = b(:,1:end-1);
y = b(:,end);

options.method = 'LogisticRegression';
options.numberOfFolds = 5;
options.lambda = 0.1;
[confusionMatrix,accuracy] =  classify677_hwk5(X,y,options);
```

```
%should get:
% LogisticRegression: Accuracy =  67.83%
% Confusion Matrix:
%     77     68
%     43    157
% Note: Matlab 2016a gives:
%     77     68
%     45    155
% Note: Matlab 2016b gives:
%     75     70          --or--       76     69
%     45    155                       45    155
```

Do not continue until your accuracy and confusion matrix matches as above.
Ans: Show listing with correct accuracy and confusion matrix

LogisticRegression: Accuracy = 67.25%

Confusion Matrix:

```
   76    69

   44   156
```

6.b) (10 points) AdaBoost uses a greedy algorithm to determine a series of weak classifiers, one weak classifier per each iteration. Each weak classifier in our example will be a threshold, where a threshold consists of {feature,T,β}, where the feature is the column of X we are splitting on, T is a threshold value to split that feature, and β={-1,+1} determines whether positive samples are on the left or right of T. Each iteration uses the same exact data to select the next threshold, but it weights these samples differently for each iteration. During each iteration, the classifier is trained to minimize the weighted classification error:

$$\sum_i D(i) * I\left(f_m\left(x^{(i)}\right) \neq y^{(i)}\right) \tag{1}$$

Where $D(i)$ is the weight of sample $\{x^{(i)},y^{(i)}\}$, I is the indicator function, and $f_m()$ is the boosting decision function:

$$f_m(x) = \beta(-1)I\left(x^{(i)} \geq T\right) + \beta(1)I\left(x^{(i)} < T\right) \tag{2}$$

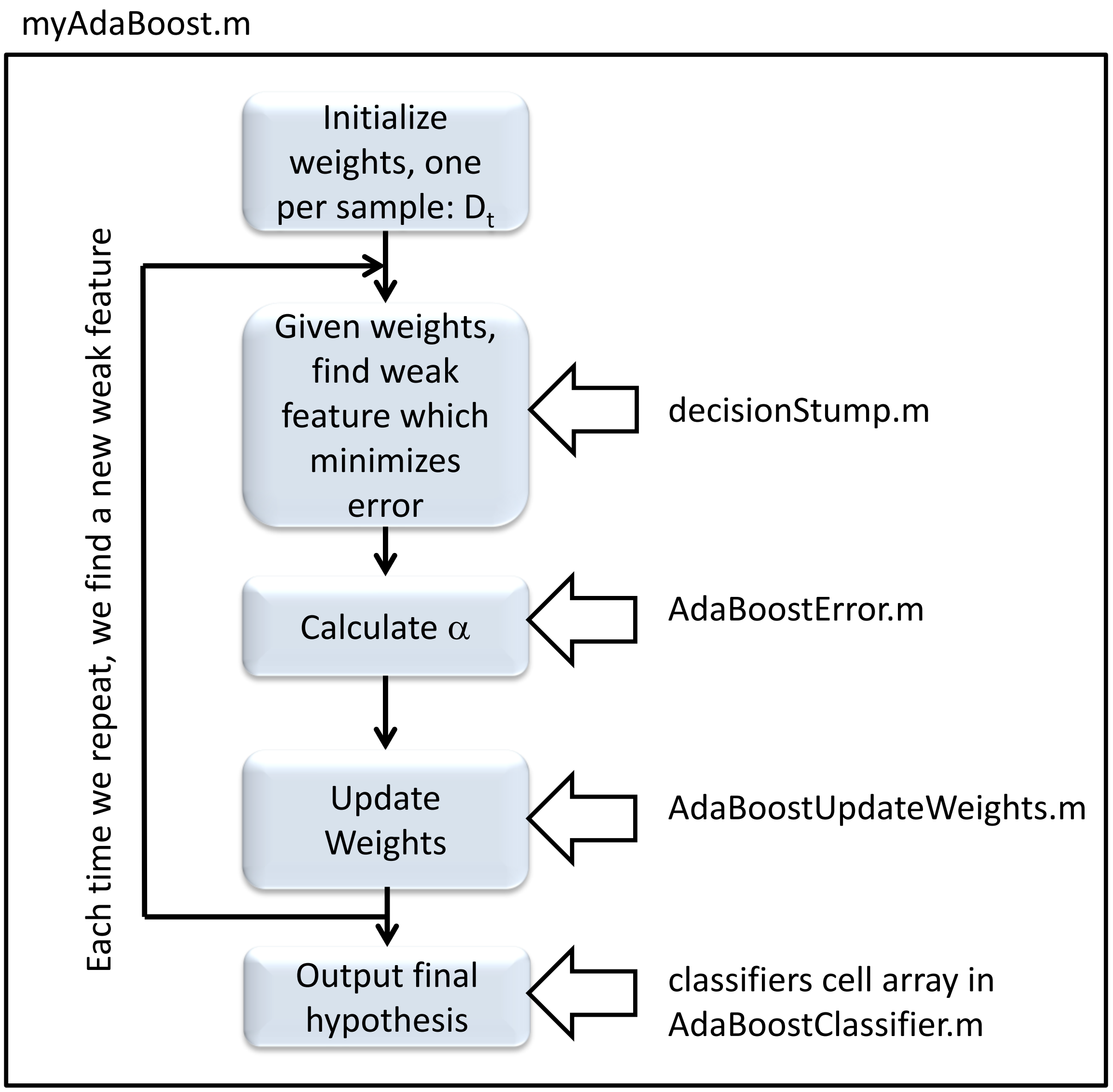
A decision stump is a decision tree with a single node which corresponds to a single threshold on one feature at a time. During each iteration, each feature of X is evaluated across all values of T and β. The combination of {feature,T,β} that minimizes the classification error is chosen during each iteration in a greedy fashion.

Download the functions myAdaBoost.m, decisionStump.m, AdaBostError.m, AdaBoostUpdateWeights.m, and AdaBoostClassifier.m from MyCourses (part of hwk5files_forMycourses.zip).

- myAdaBoost.m implements the AdaBoost algorithm core functionality, you do not need to modify this file

- decisionStump.m iterates through all possible threshold possibilities during each round to find the one with minimum error on the weighted training data, you do not need to modify this file. decisionStump.m returns h1, which is a structure element containing our threshold information {feature,T,β}, as well as a performance error.
- AdaBostError.m, AdaBoostUpdateWeights.m, and AdaBoostClassifier.m are template functions provided. You will have to modify each to make myAdaBoost.m work properly. The convention <insert code here> will be used inside these m-files whenever you need to insert code in those functions.

Study the 'AdaBoost Algorithm' and 'AdaBoost Algorithm- Example' slides from the lecture notes. You will implement this code in Matlab. The Matlab code will be organized as follows:


myAdaBoost.m
Initialize weights, one per sample: D_t
Given weights, find weak feature which minimizes error
Calculate α
Update Weights
Output final hypothesis
decisionStump.m
AdaBoostError.m
AdaBoostUpdateWeights.m
classifiers cell array in AdaBoostClassifier.m
Each time we repeat, we find a new weak feature

Our first task will be to update the prediction and alpha values of the AdaBoost error function AdaBoostError.m. For the prediction, we can implement (2) as:
predict = (2*(Xdata(:,feature) < T) -1)*polarity

For the alpha function, we have:

$$\alpha = \frac{1}{2}\left(log\left(1-\frac{\sum weightedMistakes}{\sum weights}\right) - log\left(\frac{\sum weightedMistakes}{\sum weights}\right)\right) \qquad (3)$$

Modify the prediction and alpha lines of AdaBoostError.m. You can test AdaBoostError.m with:

```
%%  develop adaboost
cd C:\Users\rwpeec\Desktop\rwpeec\rit\CMPE-677_MachineIntelligence\hwk\hwk5
close all ; clear all;
b = load('bupa.data');
Xdata = b(:,1:end-1);
y = b(:,end);

%turn ground truth labels into {-1,+1}
yList = unique(y);
if yList(1) ~= -1
   y(y==yList(1))=-1;
   y(y==yList(2))= 1;
end

%form train and test sets
TrainXdata = Xdata(1:200,:);
TrainGT = y(1:200);
TestXdata = Xdata(201:end,:);
TestGT = y(201:end);

%number of features
adaboost_numFeatures=500;

%for every feature, find the best threshold
D = size(TrainXdata,2);  %number of features , fn
n = size(TrainXdata,1);  %number of samples, fs
nlist = [1:n]';

%Each sample gets a weight r
weights = TrainGT.*0+1./n;   %equal weights to start

 %each iteration creates a classifier.
h1=decisionStumpW(weights,TrainXdata,TrainGT)    ;                %train base learner
[errorAmt, alpha] = AdaBoostError(weights, h1,TrainXdata, TrainGT); %determine alpha
%errorAmt = 0.3750, alpha = 0.2554
```

Ans: show you completed AdaBoostError.m function

```
function [errorAmt, alpha,predict] = AdaBoostError(weights,
classifier,localXdata, localGT)
% function [errorAmt, alpha] = AdaBoostError(weights,
classifier,localXdata, localGT)
%  Compute AdaBoost Error across a set of training samples
% Input:
%   weights- one weight per input sample, nx1
%   classifier- a struct containing the fields feature, thresh, and
polarity
%   localXdata- the X values of input sample, nxD
%   localGT- the ground truth for each input sample, nx1
% Output:
%   errorAmt- current error
%   alpha- AdaBoost alpha value
%   predict- prediction class [-1,+1] of each input sample
%
%  CMPE-677, Machine Intelligence
%  Base code by R. Ptucha, Andrew Gallagher 2014
%  Rochester Institute of Technology

%make a predition using the classifier on our data
%predict = (2*(Xdata(:,feature) < T) -1)*polarity
predict = (2*(localXdata(:,classifier.feature)<classifier.thresh)-
1)*classifier.polarity;

%form a [0,1] vector mistakes, mistakes will have one entry for each
%training sample
mistakes = (predict~=localGT);              % 1 for mistakes

%weigh these mistakes
wmistakes = mistakes.*weights;          %weighted mistakes

%sum of weighted mistakes
errsum = sum(wmistakes);

%relative error
errorAmt = errsum/sum(weights);

%alpha as per AdaBoost
alpha = 0.5*log((1-errorAmt)./errorAmt);
```

6.c (5 pts) After each iteration, AdaBoost needs to update its weights. Modify the function AdaBoostUpdateWeights.m by calling AdaBoostError.m to generate an α and prediction value, then calculating the new weights using:

$$weights' = weights * e^{(-\alpha PG)} \tag{4}$$

Where P is the prediction values from AdaBoostError.m, and G is the ground truth values from each sample.

You can test this code, by continuing from the last step with the commands:

```
h1.alpha = alpha;

[newWeights,zz] = AdaBoostUpdateWeights(weights,h1,TrainXdata,TrainGT);    %update the
weights
% newWeights(1:5)
%     0.0067
%     0.0040
%    0.0040
%    0.0040
%    0.0067
%zz = 0.9682
```

Ans: Include your AdaBoostUpdateWeights.m function here

```
function [newweights,zt] = AdaBoostUpdateWeights(inweights, classifier,
localXdata, localGT)
%function [newweights,zt] = AdaBoostUpdateWeights(inweights,
classifier, localXdata, localGT)
%  Update weights for each iteration of AdaBoost
% Input:
%   inweights- one weight per input sample, nx1
%   classifier- a struct containing the fields feature, thresh, and
polarity
%   localXdata- the X values of input sample, nxD
%   localGT- the ground truth for each input sample, nx1
% Output:
%   newweights- updated weights, one weight per input sample
%   zt- sum of newweights
%
%  CMPE-677, Machine Intelligence
%  Base code by R. Ptucha, Andrew Gallagher 2014
%  Rochester Institute of Technology


%get the weighting of the classifier
%call AdaBoostError to get alpha and predict
[err, alpha,predict] = AdaBoostError(inweights, classifier,localXdata,
localGT);


%calculate new weights
newweights = inweights.*exp(-alpha.*predict.*localGT);


%normalize new weights
zt = sum(newweights);
newweights = newweights./zt;
```

6.d (10 points) You are done writing code to learn AdaBoost classifier, now you need to create code to apply the classifier to test data points. Modify the function AdaBoostClassifier.m, such that given an input vector of Xdata it will apply the following pseudo code:

```
Foreach classifier threshold
    1. Extract the classifier from the vector of classifiers
    2. Make a classification prediction
    3. Keep track of a running sum of predictions
End
```

To make a classification prediction, you will apply the same exact equation you used in AdaBoostError.m, but you also need to multiply by the alpha value from the extracted classifier

Ans: Include your AdaBoostClassifier.m function here

```
function [predict] = AdaBoostClassifier(classifier,localXdata)
% function [pred] = AdaBoostClassifier( classifier,localXdata)
%  Compute AdaBoost Classification across a set of training samples
% Input:
%   classifier- a struct containing the fields feature, thresh, and
%                  polarity, one entry per threshold
%   localXdata- the X values of input sample, nxD
% Output:
%   predict- prediction class [-1,+1] of each input sample
%
%  CMPE-677, Machine Intelligence
%  Base code by R. Ptucha, Andrew Gallagher 2014
%  Rochester Institute of Technology

%for every feature, find the best threshold
D = size(localXdata,2);             %number of features
n = size(localXdata,1);             %number of samples
nh = length(classifier);             %the number of classifiers.
totalPred = zeros(n,1);

%for each classifier, make a guess on the input X data
for i = 1:1:nh
   % apply the classifier
   % extract the current iterations classifier
   cc = classifier(i);


    %Make a prediction for each feature, threshold value, and polarity
    % The equation here is identical to to AdaBoostError.m, but now we
also
    % need to multiply by the alpha value in our classifier entry
    predict = (2*(localXdata(:,cc.feature)<cc.thresh)-
1)*cc.polarity*cc.alpha;



    %keep running sum of predict
    %totalPred = <insert code here>
    totalPred = totalPred+predict;
end

% now convert threshold to [-1 1]
% Convert totalPred positive values to +1 and negative totalPred values
to -1
predict = (totalPred>0)*2-1;
```

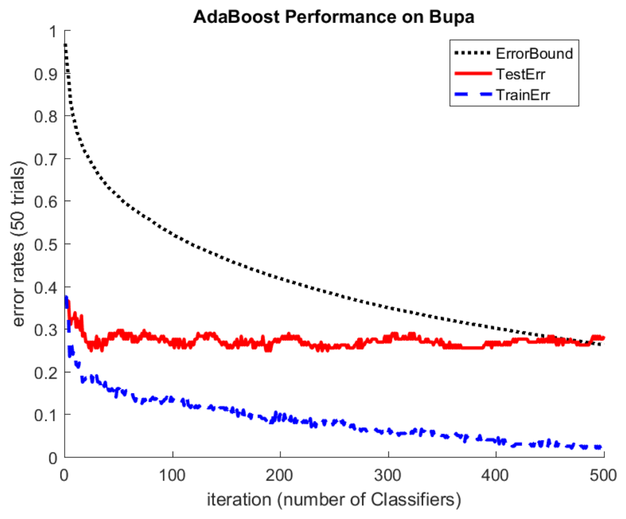
6.e (10 pts) You now will test your completed AdaBoost function myAdaBoost.m. Use the following test code:

```
cd C:\Users\rwpeec\Desktop\rwpeec\rit\CMPE-677_MachineIntelligence\hwk\hwk5
close all ; clear all;
b = load('bupa.data');
Xdata = b(:,1:end-1);
y = b(:,end);

%turn ground truth labels into {-1,+1}
yList = unique(y);
if yList(1) ~= -1
   y(y==yList(1))=-1;
   y(y==yList(2))= 1;
end

%form train and test sets
TrainXdata = Xdata(1:200,:);
TrainGT = y(1:200);
TestXdata = Xdata(201:end,:);
TestGT = y(201:end);

%number of features
adaboost_numFeatures=500;
[classifiers, errors, pred] =
myAdaBoost(TrainXdata,TrainGT,adaboost_numFeatures,TestXdata,TestGT)

meanTrain = errors.train;
meanTest = errors.test;
meanEB = errors.eb;


figure
hold on
x = 1:1:adaboost_numFeatures;
plot(x, meanEB,'k:',x,meanTest,'r-',x,meanTrain,'b--','LineWidth',2);
legend('ErrorBound','TestErr','TrainErr','Location','Best');
xlabel 'iteration (number of Classifiers)'
ylabel 'error rates (50 trials)'
title 'AdaBoost Performance on Bupa'
%print -dpng hwk5_10e.png
```

Insert a copy of the figure here at 50% and explain what is happening with model complexity during each iteration, and why the training error appears to have a different slope than the testing error. What is the best number of classification features?

Ans:

Training error keeps decreasing with iterations while testing error fluctuates. This means that the classifier is overfitting the data. The best number of features is 20 – 30 because there is a dip in test error.



7. (15 pts) Call classify677_hwk5.m with:

```
b = load('bupa.data');
X = b(:,1:end-1);
y = b(:,end);

options.method = 'Adaboost';
options.numberOfFolds = 5;
options.adaboost_numFeatures=500;
[confusionMatrix,accuracy] =  classify677_hwk5(X,y,options);
```

What is the Accuracy and confusion matrix? By looking at the plot generated when you run this code, how might you increase the accuracy in this problem?

Ans:

As the number of features increases, the test error seems to increase as well. Reducing number of features can increase the accuracy.

Adaboost: Accuracy = 73.91%

Confusion Matrix:

98	**47**
43	**157**