10-601 Machine Learning, Fall 2011: Homework 1

Machine Learning Department
Carnegie Mellon University

Due: September 23, 5pm

Instructions There are 2 questions on this assignment. The second question involves coding, so start early. Please submit your completed homework to Sharon Cavlovich (GHC 8215) by 5pm friday, Sept 23. Submit your homework as 2 **separate** sets of pages, one for each question (so the TA's can easily split it up for grading). Include your name and email address on each set.

1 Basic Probability Review [Mladen Kolar, 20 points]

1. This problem reviews basic concepts from probability.

   (a) [2 points] A biased die has the following probabilities of landing on each face:

face	1	2	3	4	5	6
P(face)	.1	.1	.2	.2	.4	0

   I win if the die shows even. What is the probability that I win? Is this better or worse than a fair die (i.e., a die with equal probabilities for each face)?

   Solution:
   $P[even] = P(2) + P(4) + P(6) = 0.1 + 0.2 + 0 = 0.3$ This is *worse* than a fair die which has probability 0.5 to land on an even number.

   (b) [2 points] Suppose P(snow today) = 0.30, P(snow tomorrow) = 0.60, P(snow today and tomorrow) = 0.25. Given that it snows today, what is the probability it will snow tomorrow?

   Solution:

$$P[\text{snow tomorrow}|\text{snow today}] = \frac{P[\text{tomorrow} \wedge \text{today}]}{P[\text{today}]} = \frac{0.25}{0.30} = \frac{5}{6}.$$

   (c) [2 points] Give a formula for $P(G|\neg H)$ in terms of $P(G)$, $P(H)$ and $P(G \wedge H)$ only. Here H and G are boolean random variables.

   Solution:

$$P[G|\neg H] = \frac{P[G \wedge \neg H]}{P[\neg H]} = \frac{P[G] - P[G \wedge H]}{1 - P[H]}.$$

(d) [2 points] Recall that the expected value $E[X]$ for a random variable X is

$$E[X] = \sum_{x \in Values(X)} P(X = x)\; x$$

where Values(X) is the set of values X may take on. Similarly, the expected value of any function f of random variable X is

$$E[f(X)] = \sum_{x \in Values(X)} P(X = x)\; f(x)$$

Now consider the function below, which we call the "indicator function"

$$\delta(X = a) := \begin{cases} 1 & \text{if } X = a \\ 0 & \text{if } X \neq a \end{cases}$$

Let X be a random variable which takes on the values $3, 8$ or 9 with probabilities p_3, p_8 and p_9 respectively. Calculate $E[\delta(X = 8)]$.

Solution:

$$E[\delta(X = 8)] = \sum_{x \in \{3,8,9\}} p_x \delta(x = 8) = p_3 * 0 + p_8 * 1 + p_9 * 0 = p_8$$

(e) [2 points] A box contains w white balls and b black balls. A ball is chosen at random. The ball is then replaced, along with d more balls of the same color (as the chosen ball). Then another ball is drawn at random from the box. Show that the probability that the second ball is white does not depend on d.

Solution:

$$\begin{aligned} P[\text{2nd white}] &= P[\text{2nd white}|\text{1st white}]P[\text{1st white}] \\ &\quad + P[\text{2nd white}|\text{1st black}]P[\text{1st black}] \\ &= \frac{w+d}{w+b+d} * \frac{w}{w+b} + \frac{w}{w+b+d}\frac{b}{w+b} \\ &= \frac{(w+b+d)w}{(w+b+d)(w+b)} \\ &= \frac{w}{w+b} \end{aligned}$$

which does not depend on d.

2. The remaining questions examine the relationship between entropy, mutual information, and independence. Recall the following definitions:

- Entropy: $H(X) = -\sum_{x \in \text{Values}(X)} P(X = x) \log_2 P(X = x) \quad = -E[\log_2 P(X)]$
- Joint entropy: $H(X, Y) = -\sum_{x \in \text{Values}(X)} \sum_{y \in \text{Values}(Y)} P(X = x, Y = y) \log_2 P(X = x, Y = y) \quad = -E[\log_2 P(X, Y)]$
- Conditional entropy: $H(Y|X) = -\sum_{x \in \text{Values}(X)} \sum_{y \in \text{Values}(Y)} P(X = x, Y = y) \log_2 P(Y = y|X = x) \quad = -E[\log_2 P(Y|X)]$
- Mutual information: $I(X; Y) = \sum_{x \in \text{Values}(X)} \sum_{y \in \text{Values}(Y)} P(X = x, Y = y) \log_2 \frac{P(X=x,Y=y)}{P(X=x)P(Y=y)}$

(a) [3 points] Using the definitions of the entropy, joint entropy, and conditional entropy, prove the following chain rule for the entropy:

$$H(X,Y) = H(Y) + H(X|Y).$$

Solution:

$$\begin{aligned}
H(X,Y) &= -\sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2 P(X=x,Y=y)\\
&= -\sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2 P(X=x)P(Y=y|X=x)\\
&= -\sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2 P(X=x)\\
&\quad -\sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2 P(Y=y|X=x)\\
&= -\sum_{x\in\text{Values}(X)} P(X=x)\log_2 P(X=x)\\
&\quad -\sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2 P(Y=y|X=x)\\
&= H(X) + H(Y|X).
\end{aligned}$$

(b) [5 points] Prove the following identities that relate the mutual information and entropy:

i. $I(X;Y) = H(X) - H(X|Y)$
ii. $I(X;Y) = H(Y) - H(Y|X)$
iii. $I(X;Y) = I(Y;X)$
iv. $I(X;X) = H(X)$.

Solution:

i.

$$\begin{aligned}
I(X;Y) &= \sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2\frac{P(X=x,Y=y)}{P(X=x)P(Y=y)}\\
&= \sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2\frac{P(X=x,Y=y)}{P(X=x)}\\
&= -\sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2 P(X=x)\\
&\quad +\sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2 P(X=x|Y=y)\\
&= -\sum_{x\in\text{Values}(X)} P(X=x)\log_2 P(X=x)\\
&\quad -\left(-\sum_{x\in\text{Values}(X)}\sum_{y\in\text{Values}(Y)} P(X=x,Y=y)\log_2 P(X=x|Y=y)\right)\\
&= H(X) - H(X|Y)
\end{aligned}$$

ii. $I(X;Y) = H(Y) - H(Y|X)$ follows by symmetry.

iii. $I(X;Y) = I(Y;X)$ follows by symmetry.

iv. $I(X;X) = H(X) - H(X|X) = H(X)$.

(c) [2 points] Recall that two random variables X and Y are *independent* if

$$\text{for all } x \in \text{Values}(X) \text{ and all } y \in \text{Values}(Y), \quad P(X=x, Y=y) \;=\; P(X=x)P(Y=y).$$

If variables X and Y are independent, is $I(X;Y) = 0$? If yes, prove it. If no, give a counter example.

Solution:

Since variables X and Y are independent

$$\begin{aligned}
I(X;Y) &= \sum_{x\in\text{Values}(X)} \sum_{y\in\text{Values}(Y)} P(X=x, Y=y)\log_2 \frac{P(X=x,Y=y)}{P(X=x)P(Y=y)} \\
&= \sum_{x\in\text{Values}(X)} \sum_{y\in\text{Values}(Y)} P(X=x, Y=y)\log_2 \frac{P(X=x)P(Y=y)}{P(X=x)P(Y=y)} \\
&= \sum_{x\in\text{Values}(X)} \sum_{y\in\text{Values}(Y)} P(X=x, Y=y)\log_2 1 \\
&= 0.
\end{aligned}$$

2 Binary Decision Trees [William Bishop, 40 points]

One very interesting application area of machine learning is in making medical diagnoses. In this problem you will train and test a binary decision tree to detect breast cancer using real world data. *You may use any programming language you like.*

2.1 The Dataset

We will use the Wisconsin Diagnostic Breast Cancer (WDBC) dataset[1]. The dataset consists of 569 samples of biopsied tissue. The tissue for each sample is imaged and 10 characteristics of the nuclei of cells present in each image are characterized. These characteristics are

1. Radius
2. Texture
3. Perimeter
4. Area
5. Smoothness
6. Compactness
7. Concavity
8. Number of concave portions of contour
9. Symmetry
10. Fractal dimension

[1]Original dataset available at `http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`.

Each of the 569 samples used in the dataset consists of a feature vector of length 30. The first 10 entries in this feature vector are the mean of the characteristics listed above for each image. The second 10 are the standard deviation and last 10 are the largest value of each of these characteristics present in each image.

Each sample is also associated with a label. A label of value 1 indicates the sample was for malignant (cancerous) tissue. A label of value 0 indicates the sample was for benign tissue.

This dataset has already been broken up into training, validation and test sets for you and is available in the compressed archive for this problem on the class website. The names of the files are "trainX.csv", "trainY.csv", "validationX.csv", "validationY.csv", "testX.csv" and "testY.csv." The file names ending in "X.csv" contain feature vectors and those ending in "Y.csv" contain labels. Each file is in *comma separated value* format where each row represents a sample.

2.2 Programming

2.2.1 Learning a Binary Decision Tree [10 Points]

As discussed in class and the reading material, to learn a binary decision tree we must determine which feature attribute to select as well as the threshold value to use in the split criterion for each non-leaf node in the tree. This can be done in a recursive manner, where we first find the optimal split for the root node using all of the training data available to us. We then split the training data according to the criterion selected for the root node, which will leave us with two subsets of the original training data. We then find the optimal split for each of these subsets of data, which gives the criterion for splitting on the second level children nodes. We recursively continue this process until the subsets of training data we are left with at a set of children nodes are pure (i.e., they contain only training examples of one class) or the feature vectors associated with a node are all identical (in which case we can not split them) but their labels are different.

In this problem, you will implement an algorithm to learn the structure of a tree. The optimal splits at each node should be found using the information gain criterion discussed in class. While you are free to write your algorithm in any language you choose, if you use the provided MATLAB code included in the compressed archive for this problem on the class website, you only need to complete one function - `computeOptimalSplit.m`. This function is currently empty and only contains comments describing how it should work. Please complete this function so that given any set of training data it finds the optimal split according to the information gain criterion.

Include a printout of your completed `computeOptimalSplit.m` along with any other functions you needed to write with your homework submission. If you choose to not use the provided MATLAB code, please include a printout of all the code you wrote to train a binary decision tree according to the description given above.

Note: While there are multiple ways to design a decision tree, in this problem we constrain ourselves to those which simply pick *one* feature attribute to split on. Further, we restrict ourselves to performing only binary splits. In other words, each split should simply determine if the value of a particular attribute in the feature vector of a sample is less than or equal to a threshold value or greater than the threshold value.

Note: Please note that the feature attributes in the provided dataset are continuously valued. There are two things to keep in mind with this.

First, this is slightly different than working with feature values which are discrete because it is no longer possible to try splitting at every possible feature value (since there are an infinite number of possible feature values). One way of dealing with this is by recognizing that given a set of training data of N points, there are only $N - 1$ places we could place splits for the data (if we constrain ourselves to binary splits). Thus, the approach you should take in this function is to sort the training data by feature value and then test split values that are the mean of ordered training points. For example, if the points to split between were $1, 2, 3$, you would test two split values - 1.5 and 2.5.

Second, when working working with feature values that can only take on one of two values, once we split using one feature attribute, there is no point in trying to split on that feature attribute later. (Can you think of why this would be?) However, when working with continuously valued data, this is no longer the case, so your splitting algorithm should consider splitting on all feature attributes at every split.

Code for computeOptimalSplit.m

```
% Given a set of training data, this is a function to select the single
% best attribute to perform a binary split with, as measured by information
% gain, and the particular threshold to perform that split at.
%
% Usage: [attr, thresh] = computeOptimalSplit(x, y)
%
% Inputs:
%
%   x - a S by D matrix, where S is the number of samples and D is the
%   length of a feature vector.   x(s,:) gives the feature vector for
%   sample s.
%
%   y - an S by 1 array.  y(s) gives the label for sample s.
%
% Outputs:
%
%   attr - this is the index of the best attribute to perform a split at.
%
%   thresh - this is the threshold at which the split should be performed.
%
%   Thus, the splitting rule for sample s will be:
%
%       split into class 1,      if x(s,attr) <= thresh,
%       split into class 2,      otherwise
%
% Example:
%
%  If:
%
%       x = [1 2 3 4 5 6 7 8 9; 2 2 2 2 2 2 2 2 2]'
%
%       y = [0 0 0 0 0 1 1 1 1]'
%
%
%   and we call:
%
%      [attr, thresh] = computeOptimalSplit(x, y)
%
%   then:
%
%       attr = 1
%
%       thresh = 5.5
%
function [attr, thresh] = computeOptimalSplit(x, y)

% Get some basic information about input
nSmps = size(x,1);
```

```
nAttrs = size(x,2);

% Create cell to hold possible threshold values for each attribute
possThreshVlsCell = cell(1, nAttrs);

% Create cell to hold conditional entropies achieved on training data
% for each possible threshold value for each attribute
possThreshCondEntropiesCell = cell(1, nAttrs);

for a = 1:nAttrs
    attrVls = x(:,a);
    uniqueAttrVls = unique(attrVls);

    % Get possible threshold values
    if length(uniqueAttrVls) > 1
        possThreshVls = diff(uniqueAttrVls)/2 + uniqueAttrVls(1:end-1);
    else
        possThreshVls = uniqueAttrVls;
    end
    nPossThreshVls = length(possThreshVls);
    possThreshVlsCell{a} = possThreshVls;
    possThreshCondEntropies = nan(nPossThreshVls,1);

    % Determine conditional entropy for each possible threshold value
    for t = 1:nPossThreshVls

        % Find samples above and below the current threshold
        indsLessThanOrEqual = attrVls <= possThreshVls(t);
        indsAbove = attrVls > possThreshVls(t);

        % Get current sample entropy
        pLessThanOrBelow  = sum(indsLessThanOrEqual)/nSmps;
        possThreshCondEntropies(t) = pLessThanOrBelow*computeSmpEntropy(y(indsLessThanOrEqual)) + ...
            (1 - pLessThanOrBelow)*computeSmpEntropy(y(indsAbove));

    end

    possThreshCondEntropiesCell{a} = possThreshCondEntropies;

end

% Pick optimal attribute to split on and the optimal threshold to split
% that attribute at
bestCE = inf;
attr = nan;
thresh = nan;

bestAVls = nan(1, nAttrs);
for a = 1:nAttrs
    curCEVals = possThreshCondEntropiesCell{a};
    [curBest, bestThreshInd] = min(curCEVals);

    bestAVls(a) = curBest;
    if curBest < bestCE
```

```
        bestCE = curBest;
        attr = a;
        thresh = possThreshVlsCell{a}(bestThreshInd);
    end
end

% This is a function to compute the entropy of a sample.
%
% Usage: e = computeSmpEntropy(vls)
%
% Inputs:
%
%   vls - a 1-D array of sample labels.
%
% Outputs:
%
%   e - the computed entropy of the sample.
%
% Author: wbishop@cs.cmu.edu
%
function e = computeSmpEntropy(vls)


if length(vls) > 1
    uniqueVls = unique(vls);
    nUniqueVlInstances = histc(vls, uniqueVls);
    vlProbs = nUniqueVlInstances/sum(nUniqueVlInstances);
    e = -sum(vlProbs.*log2(vlProbs));
else
    e = 0;
end
```

2.2.2 Pruning a Binary Decision Tree [10 Points]

The method of learning the structure and splitting criterion for a binary decision tree described above terminates when the training examples associated with a node are all of the same class or there are no more possible splits. In general, this will lead to overfitting. As discussed in class, pruning is one method of using validation data to avoid overfitting.

In this problem, you will implement an algorithm to use validation data to greedily prune a binary decision tree in an iterative manner. Specifically, the algorithm that we will implement will start with a binary decision tree and perform an exhaustive search for the single node for which removing it (and its children) produces the largest increase (or smallest decrease) in classification accuracy as measured using validation data. Once this node is identified, it and its children are removed from the tree, producing a new tree. This process is repeated, where we iteratively prune one node at a time until we are left with a tree which consists only of the root node[2].

In this problem, you will implement a function which starts with a tree and selects the single best node to remove to produce the greatest increase (or smallest decrease) in classification accuracy as measured with validation data. If you are using MATLAB, this means you only need to complete the empty function `pruneSingleGreedyNode.m`. Please see the comments in that function for details on what you should implement. Note, we suggest that you make use of the provided MATLAB functiona `pruneAllNodes.m` which

[2]In practice, you can often simply continue the pruning process until the validation error fails to increase by a predefined amount. However, for illustration purposes, we will continue until there is only one node left in the tree.

will return a listing of all possible trees that can be formed by removing a single node from a base tree and `batchClassifyWithDT.m` which will classify a set of samples given a decision tree.

Please include your version of `pruneSingleGreedyNode.m` along with any other functions you needed to write with your homework. If not using MATLAB, please attach the code for a function which performs the same function described for `pruneSingleGreedyNode.m`.

Code for pruneSingleGreedyNode

```
% This is a function that given a set of validation data and a binary
% decision tree will select the non-leaf node of the tree to remove such
% that validation accuracy increases by the greatest amount.  (Note that
% sometimes removing any non-leaf node can only hurt things.  In this case,
% this function selects the non-leaf node that will result in the smallest
% decrease in validation accuracy).
%
% Usage: dTOut = pruneSingleGreedyNode(x, y, dT)
%
% Inputs:
%
%   x - a S by D matrix, where S is the number of samples and D is the
%   length of a feature vector.   x(s,:) gives the feature vector for
%   validation sample s.
%
%   y - an S by 1 array.  y(s) gives the label for validation sample s.
%
%   dT - the base tree to prune a node from.
%
% Outputs:
%
%   dTOut - the pruned tree.
%
function dTOut = pruneSingleGreedyNode(x, y, dT)

nSmps = length(y);

% Get all possible trees that can be formed be removing a single node from
% the curren tree
allPossibleTrees = pruneAllNodes(dT);

% Get error on provided validation data for each tree
nTrees = length(allPossibleTrees);
treePCorrect = nan(1, nTrees);
treeNLeafs = nan(1, nTrees);
for t = 1:nTrees
    yHat = batchClassifyWithDT(x, allPossibleTrees{t});
    treePCorrect(t) = sum(yHat == y)/nSmps;

    curTreeStats = gatherTreeStats(allPossibleTrees{t});
    treeNLeafs(t) = curTreeStats.nLeafs;
end

[blah, bestInd] = max(treePCorrect);
```

```
% Check if there were multiple trees with the best accuracy; in that case
% pick the tree with the smallest number of leafs
sameAcTrees = find(treePCorrect == treePCorrect(bestInd));

disp(num2str(treeNLeafs(sameAcTrees)));

[blah, minNLeafsInd] = min(treeNLeafs(sameAcTrees));
bestInd = sameAcTrees(minNLeafsInd);

dTOut = allPossibleTrees{bestInd};
```

2.3 Data Analysis

2.3.1 Training a Binary Decision Tree [5 Points]

In this section, we will make use of the code that we have written above. We will start by training a basic decision tree. Please use the training data provided to train a decision tree. (In MATLAB, assuming you have completed the `computeOptimalSplit.m` function, the function `trainDT.m` can do this training for you.)

Please specify the total number of nodes and the total number of leaf nodes in the tree. (In MATLAB, the function `gatherTreeStats.m` will be useful). **Also, please report the classification accuracy (percent correct) of the learned decision tree on the provided training and testing data.** (In MATLAB, the function `batchClassifyWithDT.m` will be useful).

Answer

- There are 29 total nodes and 15 leafs in the unpruned tree.
- Training accuracy is 100% and test accuracy is 92.98%.

2.3.2 Pruning a Binary Decision Tree [8 Points]

Now we will make use of the pruning code we have written. Please start with the tree that was just trained in the previous part of the problem and make use of the validation data to iteratively remove nodes in the greedy manner described in the section above. Please continue iterations until a degenerate tree with only a single root node remains. For each tree that is produced, please calculate the classification accuracy for that tree on the training, validation and testing datasets.

After collecting this data, please plot a line graph relating classification accuracy on the test set to the number of leaf nodes in each tree (so number of leaf nodes should be on the X-axis and classification accuracy should be on the Y-Axis). Please add to this same figure, similar plots for percent accuracy on training and validation data. The number of leaf nodes should range from 1 (for the degenerate tree) to the the number present in the unpruned tree. The Y-axis should be scaled between 0 and 1. [5 Points]

Correct Plot The correct plot is shown below.

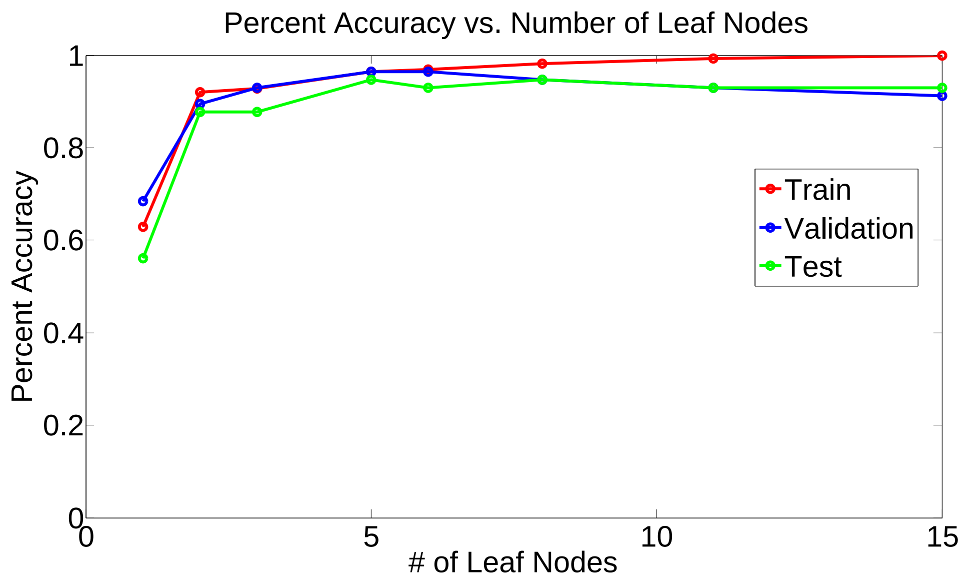

Figure 1: Plot of decode accuracy as the number of leaf nodes on a decision tree is pruned from 15 (right) to 1 (left) on training, validation and test data.

Please comment on what you notice and how this illustrates over fitting. Include the produced figure and any code you needed to write to produce the figure and calculate intermediate results with your homework submission. [3 Points]

Answer Overfitting is evident: as the number of leafs in the decision tree grows, performance on the training set of data increases. However, after a certain point, adding more leaf nodes (after 5 in this case) detrimentally affects performance on test data as the more complicated decision boundaries that are formed essentially reflect noise in the training data.

Code to produce plot is shown below.

```
% This is the main script to produce the requested plots for the HW 1
% decision tree problem.

%load hwData;

% Load our data
load('validationX.csv');
load('validationY.csv');

load('trainX.csv');
load('trainY.csv');

load('testX.csv');
load('testY.csv');
%% ========================================================================
%                    Train the basic decision tree
%  ========================================================================
basicDT = trainDT(trainX, trainY);

%% ========================================================================
%       Prune the basic tree in a greedy manner using validaiton data
% ========================================================================
prunedTrees = {basicDT};
curTree = basicDT;
while ~curTree.isLeaf
   curTree = pruneSingleGreedyNode(validationX, validationY, curTree);
   prunedTrees{end +1} = curTree;
end
nPrunedTrees = length(prunedTrees);

%% ========================================================================
%             Gather results on train, validation and test error
% ========================================================================

nLeafs = nan(1, nPrunedTrees);
trainPCorrect = nan(1, nPrunedTrees);
validationPCorrect = nan(1, nPrunedTrees);
testPCorrect = nan(1, nPrunedTrees);

for p = 1:length(prunedTrees)

    curPrunedTree = prunedTrees{p};

    curTreeStats = gatherTreeStats(curPrunedTree);
    nLeafs(p) = curTreeStats.nLeafs;

    yHatTrain = batchClassifyWithDT(trainX, curPrunedTree);
    yHatValidation = batchClassifyWithDT(validationX, curPrunedTree);
    yHatTest = batchClassifyWithDT(testX, curPrunedTree);

    trainPCorrect(p) = sum(yHatTrain == trainY)/length(trainY);
    validationPCorrect(p) = sum(yHatValidation == validationY)/length(validationY);
    testPCorrect(p) = sum(yHatTest == testY)/length(testY);

end

[blah, sortOrder] = sort(nLeafs);
nLeafs = nLeafs(sortOrder);
prunedTrees = prunedTrees(sortOrder);
trainPCorrect = trainPCorrect(sortOrder);
validationPCorrect = validationPCorrect(sortOrder);
testPCorrect = testPCorrect(sortOrder);

%% ========================================================================
%   Make plots of train, validaiton and test error vs. number of leaf nodes
% ========================================================================

plot(nLeafs, trainPCorrect, 'ro-', 'LineWidth', 4, 'MarkerSize', 10);
hold on;
plot(nLeafs, validationPCorrect, 'bo-', 'LineWidth', 4, 'MarkerSize', 10);
plot(nLeafs, testPCorrect, 'go-', 'LineWidth', 4, 'MarkerSize', 10);
hold off;
xlabel('# of Leaf Nodes', 'FontSize', 40);
ylabel('Percent Accuracy', 'FontSize', 40);
l = legend('Train', 'Validation', 'Test');
```

```
set(l, 'FontSize', 40);
title('Percent Accuracy vs. Number of Leaf Nodes', 'FontSize', 40);
set(gca, 'FontSize', 40);
set(gca, 'YLim', [0, 1]);
```

2.3.3 Drawing a Binary Decision Tree [2 Points]

One of the benefits of decision trees is the classification scheme they encode is easily understood by humans. **Please select the binary decision tree from the pruning analysis above that produced the highest accuracy on the validation dataset and diagram it. (In the even that two trees have the same accuracy on validation data, select the tree with the smaller number of leaf nodes). When stating the feature attributes that are used in splits, please use the attribute names (instead of index) listed in the dataset section of this problem.** (If using the provided MATLAB code, the function `trainDT` has a section of comments which describes how you can interpret the structure used to represent a decision tree in the code.)

Hint: The best decision tree as measured on validation data for this problem should not be too complicated, so if drawing this tree seems like a lot of work, then something may be wrong.

Diagram of correct tree The correct diagram is shown below.

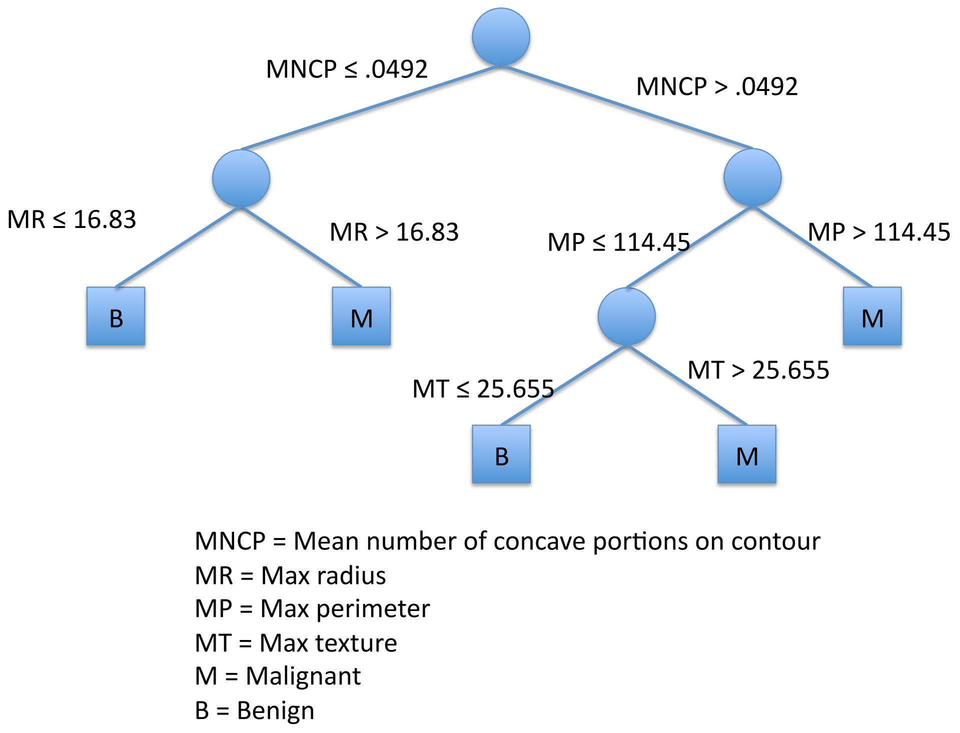

Figure 2: A diagram of the decision tree which produced the highest accuracy on validation data. (Note: This tree produced the same accuracy as a second tree, but the second tree had a greater number of leaf nodes.

2.4 An Alternative Splitting Method [5 Points]

While information gain is one criterion to use when estimating the optimal split, it is by no means the only one. Consider instead using a criterion where we try to minimize the weighted misclassification rate.

Formally, assume a set of D data samples $\{< \vec{x}^{(i)}, y^{(i)} >\}_{i=1}^{D}$, where $y^{(i)}$ is the label of sample i and $\vec{x}^{(i)}$ is the feature vector for sample i. Let $x(j)^{(i)}$ refer to the value of the j^{th} attribute of the feature vector for data point i.

Now, to pick a split criterion, we pick a feature attribute, a, and a threshold value, t,to use in the split. Let:

$$p_{\text{below}}(a,t) = \frac{1}{D}\sum_{i=1}^{D} \mathbb{I}\left(x(a)^{(i)} \leq t\right)$$

$$p_{\text{above}}(a,t) = \frac{1}{D}\sum_{i=1}^{D} \mathbb{I}\left(x(a)^{(i)} > t\right)$$

and let:

$$l_{\text{below}}(a,t) = \text{Mode}\left(\{y_i\}_{i:x(a)^{(i)} \le t}\right)$$
$$l_{\text{above}}(a,t) = \text{Mode}\left(\{y_i\}_{i:x(a)^{(i)} > t}\right)$$

The split that minimizes the weighted misclassification rate is then the one which minimizes:

$$O(a,t) = p_{\text{below}}(a,t) \sum_{i:x(a)^{(i)} \le t} \mathbb{I}\left(y^{(i)} \neq l_{\text{below}}(a,t)\right) + p_{\text{above}}(a,t) \sum_{i:x(a)^{(i)} > t} \mathbb{I}\left(y^{(i)} \neq l_{\text{above}}(a,t)\right)$$

Please modify the code for your `computeOptimalSplit.m` **(or equivalent function if not using MATLAB) to perform splits according to this criterion. Attach the code of your modified function when submitting your homework.**

Code for new version of computeOptimalSplit

```
% Given a set of training data, this is a function to select the single
% best attribute to perform a binary split with, percent accuracy,
% and the particular threshold to perform that split at.
%
% Usage: [attr, thresh] = computeOptimalSplitFromError(x, y)
%
% Inputs:
%
%   x - a S by D matrix, where S is the number of samples and D is the
%   length of a feature vector.   x(s,:) gives the feature vector for
%   sample s.
%
%   y - an S by 1 array.  y(s) gives the label for sample s.
%
% Outputs:
%
%   attr - this is the index of the best attribute to perform a split at.
%
%   thresh - this is the threshold at which the split should be performed.
%
%   Thus, the splitting rule for sample s will be:
%
%       split into class 1,      if x(s,attr) <= thresh,
%       split into class 2,      otherwise
%
%
function [attr, thresh] = computeOptimalSplitFromError(x, y)

% Get some basic information about input
nSmps = size(x,1);
nAttrs = size(x,2);

% Create cell to hold possible threshold values for each attribute
```

```
possThreshVlsCell = cell(1, nAttrs);

% Create cell to hold errors achieved on training data
% for each possible threshold value for each attribute
possThreshPErrorCell = cell(1, nAttrs);

for a = 1:nAttrs
    attrVls = x(:,a);
    uniqueAttrVls = unique(attrVls);

    % Get possible threshold values
    if length(uniqueAttrVls) > 1
        possThreshVls = diff(uniqueAttrVls)/2 + uniqueAttrVls(1:end-1);
    else
        possThreshVls = uniqueAttrVls;
    end
    nPossThreshVls = length(possThreshVls);
    possThreshVlsCell{a} = possThreshVls;
    possThreshPError = nan(nPossThreshVls,1);

    % Determine conditional entropy for each possible threshold value
    for t = 1:nPossThreshVls

        % Find samples above and below the current threshold
        indsLessThanOrEqual = attrVls <= possThreshVls(t);
        indsAbove = attrVls > possThreshVls(t);

        % Get current sample error
        pLessThanOrBelow  = sum(indsLessThanOrEqual)/nSmps;
        possThreshPError(t) = pLessThanOrBelow*computeSmpError(y(indsLessThanOrEqual)) + ...
            (1 - pLessThanOrBelow)*computeSmpError(y(indsAbove));

    end
    possThreshPErrorCell{a} = possThreshPError;

end

% Pick optimal attribute to split on and the optimal threshold to split
% that attribute at
bestE = inf;
attr = nan;
thresh = nan;
for a = 1:nAttrs
    curPEVals = possThreshPErrorCell{a};
    [curBest, bestThreshInd] = min(curPEVals);
    if curBest < bestE
        bestE = curBest;
        attr = a;
        thresh = possThreshVlsCell{a}(bestThreshInd);
    end
end


function e = computeSmpError(vls)
```

```
if ~isempty(vls)
    smpMode = mode(vls);
    e = sum(vls ~= smpMode);
else
    e = 0;
end
```

After modifying `computeOptimalSplit.m`, please retrain a decision tree (without doing any pruning). **In your homework submission, please indicate the total number of nodes and total number of leaf nodes in this tree. How does this compare with the tree that was trained using the information gain criterion?**

Answer The new tree has 16 leafs and 31 nodes. The new tree has 1 more leaf and 2 more nodes than the original tree.