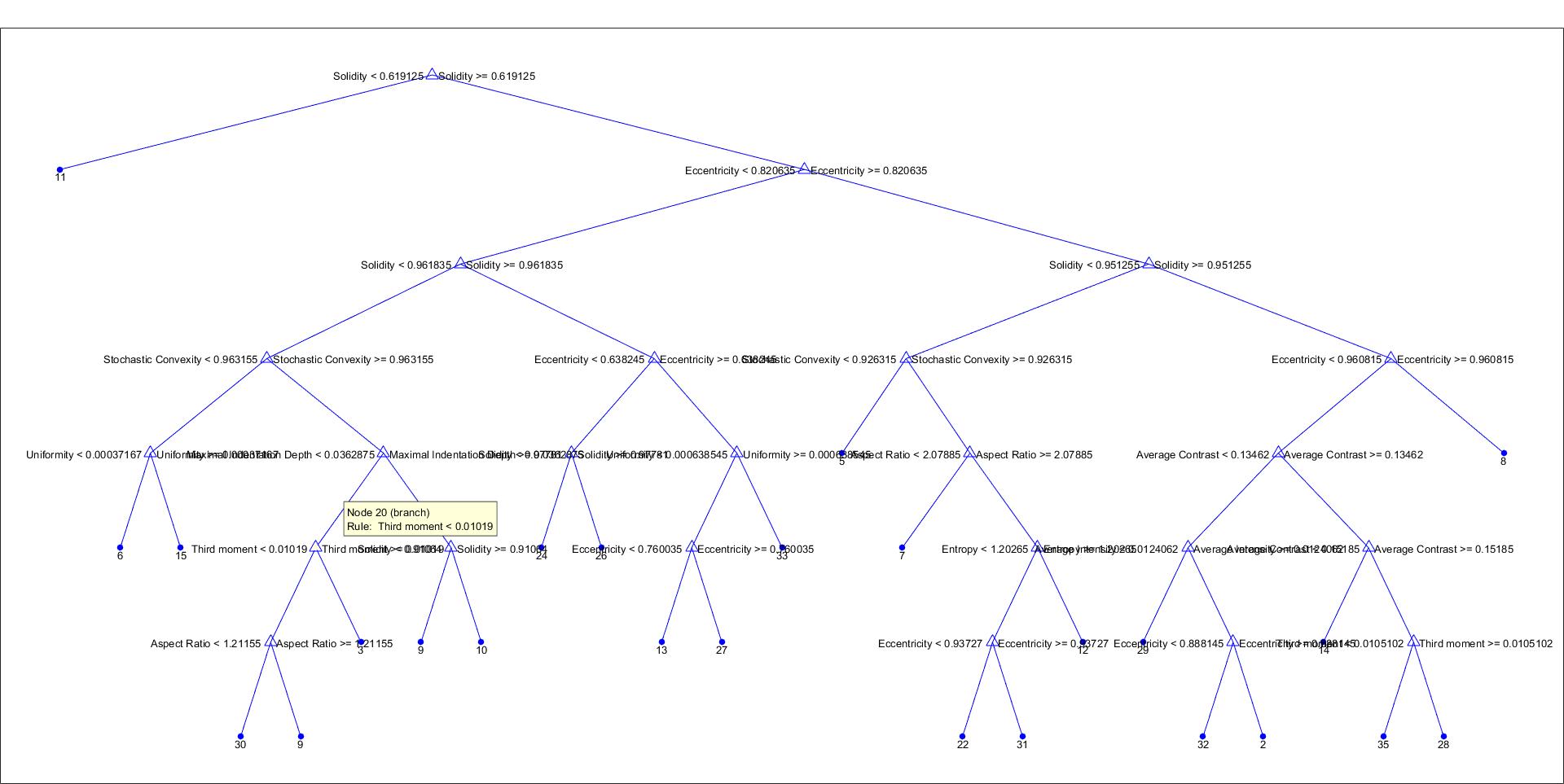
Decision trees are useful tools in classifying objects given existing data on features of similar objects and their classes. Decision trees provide a visual look at how each feature of an object determines the likelihood that it will be a part of a given class. A decision tree can help identify which features of an object is the greatest indicator and has the greatest effect on its classification.

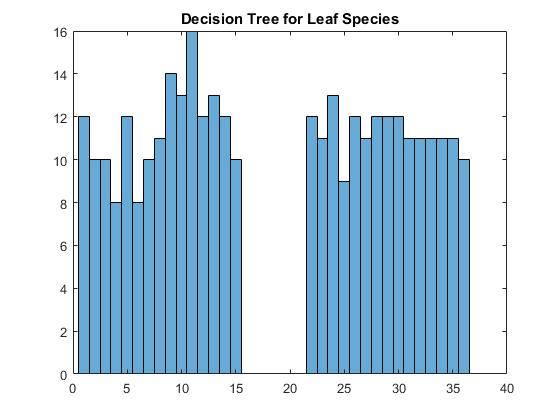
Using a data set which includes 14 features of 340 leaves, we can perform a classic two-fold cross-validation nearest neighbor exercise and yield an error-rate, which will tell us how much failure we can expect when trying to properly classify a leaf of unknown class. In two-fold cross-validation, half of the data is chosen as the training set for the model, while the other half is chosen as the test set. Without any normalization of the data, the nearest-neighbor error rate of two-fold cross validation is 0.44706, which is high enough that we cannot reliably classify the leaves. Normalizing the data with zero-one normalization and z-score normalization does not improve our situation, since we yield error-rates that of 0.92941 and 0.92353, respectively.

By building a decision tree in MATLAB, we are able to cut our error-rate down drastically, reducing it to 0.0029. An example of the decision tree is shown in Figure 1 below:



*Figure 1: Decision Tree*. The Decision Tree shown above indicates the path that the computer program takes to estimate which class the object belongs in. It compares values of each parameter to the rule it has determined for each parameter, and makes decisions one parameter at a time.

Figure 2 below shows the frequency for each class present on the decision tree. Note that classes 16 through 21 did not have any data.



*Figure 2: Decision Tree object frequency per leaf species.*

Figure 3 below shows the frequency for each class from the original data. Note that classes 16 through 21 also did not have any data, which are likely what contributed to the decision tree results to reflect an absence of classifications for those classes as well.

Note how similar Figure 2 and Figure 3 appear to be, just by the fact that the graphs follow a similar, but not identical shape. There is a global maximum at the 12th class, a local maximum at class 5 surrounded by local minima for class 4 and 6, and of course the missing classes in the center.

*Figure 3: Leaves per Species, original data set.* This graph shows the number of leaves in the original data set for each class.

Normalization is not performed for a decision tree because the data is analyzed by the decision tree one feature at a time, such that it is independent of influence from other features. With the low error rate we witnessed from the decision tree, this is acceptable.

Using the decision tree, I identified the features that were the least relevant to the classification, and those were Elongation, Solidity, and Average Contrast. We call this pruning of the decision tree. I removed these features from the training data set, and the new error-rate for nearest-neighbor classification was 0.44118.

I have attached code for the nearest neighbor and decision tree programs used, respectively.

Nearest Neighbor Code

function Leaf\_NN

TRAIN = load('leaf.csv');

TRAIN = TRAIN(randperm(size(TRAIN,1)),:);

TRAIN(:,2)=[]; % column 2 not useful

TRAIN(:,4)=[]; % column 4 (Elongation) not useful

TRAIN(:,4)=[]; % column 4 (Solidity) not useful

TRAIN(:,8)=[]; % column 8 (Average COntrast) not useful

TEST = TRAIN(1:170,:); % pull out half the data set since 2-fold cross-validation

TRAIN = TRAIN(171:end,:);

TRAIN\_class\_labels = TRAIN(:,1); % Pull out the class labels.

TRAIN(:,1) = []; % Remove class labels from training set.

TEST\_class\_labels = TEST(:,1); % Pull out the class labels.

TEST(:,1) = []; % Remove class labels from testing set.

correct = 0; % Initialize the number we got correct

for i = 1 : length(TEST\_class\_labels) % Loop over every instance in the test set

classify\_this\_object = TEST(i,:);

this\_objects\_actual\_class = TEST\_class\_labels(i);

predicted\_class = Classification\_Algorithm(TRAIN,TRAIN\_class\_labels, classify\_this\_object);

if predicted\_class == this\_objects\_actual\_class

correct = correct + 1; % we got one more correct

end;

disp([int2str(i), ' out of ', int2str(length(TEST\_class\_labels)), ' done']) % Report progress

end;

%%%%%%%%%%%%%%%%% Create Report %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp(['The dataset you tested has ', int2str(length(unique(TRAIN\_class\_labels))), ' classes'])

disp(['The training set is of size ', int2str(size(TRAIN,1)),', and the test set is of size ',int2str(size(TEST,1)),'.'])

disp(['The time series are of length ', int2str(size(TRAIN,2))])

disp(['The error rate was ',num2str((length(TEST\_class\_labels)-correct )/length(TEST\_class\_labels))])

%%%%%%%%%%%%%%%%% End Report %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Here is a sample classification algorithm, it is the simple (yet very competitive) one-nearest

% neighbor using the Euclidean distance.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function predicted\_class = Classification\_Algorithm(TRAIN,TRAIN\_class\_labels,unknown\_object)

best\_so\_far = inf;

for i = 1 : length(TRAIN\_class\_labels)

compare\_to\_this\_object = TRAIN(i,:);

distance = sqrt(sum((compare\_to\_this\_object - unknown\_object).^2)); % Euclidean distance

if distance < best\_so\_far

predicted\_class = TRAIN\_class\_labels(i);

best\_so\_far = distance;

end

end;

Decision Tree Code

function LeafDT()

LEAF = load('leaf.csv') % Load the data

LEAF(:,2) = []; % do not need column 2

meas = LEAF(:,2:15);

species = LEAF(:,1);

n = size(meas,1); % How many instances do we have?

%rng(1) % Seed the random number generator for reproducibility

idxTrn = false(n,1); % Initialize a vector of indices to a train subset

idxTrn(randsample(n,round(0.5\*n))) = true; % Training set logical indices

idxVal = idxTrn == false; % Validation set logical indices

% Learn a tree ONLY on the idxTrn subset: Call it Md1, as in Model 1

Mdl = fitctree(meas(idxTrn,:),species(idxTrn),'PredictorNames',{'Eccentricity', 'Aspect Ratio', 'Elongation', 'Solidity', 'Stochastic Convexity', 'Isoperimetric Factor', 'Maximal Indentation Depth', 'Lobedness', 'Average Intensity', 'Average Contrast', 'Smoothness', 'Third moment', 'Uniformity', 'Entropy'});

view(Mdl,'Mode','graph') % Let us see the tree we learned

for i = 1:30

% Classify ONLY the idxVal subset

label = predict(Mdl,meas(idxVal,:)); % Predict (classify) the test data, on the trained model

[label,species(idxVal)] % Echo the predicted and then true labels side by side

numMisclass(i) = sum(~strcmp(label,species(idxVal))) % How many did we get wrong?

end;

disp(sum(numMisclass)/30/n)

histogram(species)

title('Decision Tree for Leaf Species')

end