10. Decision Trees and Random Forests

10.1 Overview

Decision trees recursively split the input observations into partitions until the observations in a given partition are uniform. These regions will be rectangular in two dimensions as see in Figure 10.1 and cuboid in higher dimensions. Notice that the decision boundaries are linear, and aligned with the axes. The algorithm is greedy and does not go back and reconsider earlier splits. Once the vertical split at about 125 is made in the figure below, the only considerations at that point is whether or not to further divide the regions to the right and to the left. Decision trees have been around since the 80s and can be used for either classification or regression. They are not among the best performing algorithms but have the advantage that they are highly interpretable and give insight into the data.

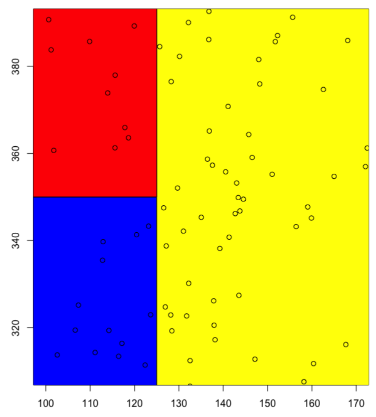


Figure 10.1: Decision Regions

10.2 Decision Trees in R

There are several packages with decision tree functions in R, among them rpart and tree. The latter seems to be a little more robust and prints better trees so we will use that for our examples. However, the online example uses both rpart and tree so you can compare. Figure 10.2 shows the output decision tree, which has 6 leaf nodes.

Code 10.2.1 — Decision Tree. Iris Data

```
library(tree)
tree_iris2 <- tree(Species~., data=iris)
plot(tree_iris2)
text(tree_iris2, cex=0.5, pretty=0)
```

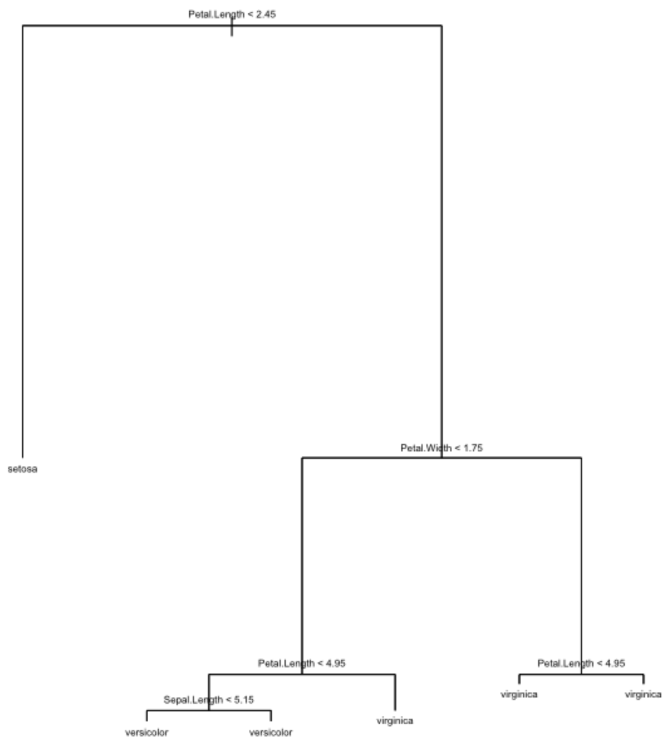


Figure 10.2: Decision Tree for Iris

If you type the tree name at the console you get a tree representing branches by indendation. This also includes the split data at each level. The summary() function on the model also gives some additional information. In the online notebook we also ran the algorithm with the same train set we have used before with iris and got 3 misclassified observations for a mean accuracy of 94%, but then we know that is typical for the Iris data set, having run it through many algorithms.

Check Your Understanding 10.1 — Decision Tree. Using the Wine Data.

Try the following:

- Load the wine_all.csv data and divide it into 80-20 train and test.
- Perform logistic regression to predict type from all other predictors.
- Perform knn on a scaled version of the data, again predicting type.
- Create a decision tree and evaluate its results. Note the Quick Reference 10.8.2 at the end of the chapter which shows how to use predict() for decision trees.
- Print your tree.
- Compare the results of the 3 algorithms and consider the advantage and disadvantages of each.

■

10.3 The Algorithm

Since decision trees can be used for regression or classification we will look at these separately, beginning with regression.

10.3.1 Regression

In linear regression our goal was to minimize RSS over all the data. In decision trees, we want to minimize RSS within each region, where $\hat{y}_r$ is the mean response for the training observation in region r.

$$\sum_{r=1}^{R} \sum_{i \in r} (y_i - \hat{y}_r)^2 \tag{10.1}$$

This is computationally infeasible in part because the data space could be divided into virtually infinite regions. Instead, a top-down, greedy approach is used to partition the data. To start, all predictors are examined to see if they make good splits in the data, and for each predictor the numerical value at which to split must be determined. Our first split will divide the data into regions r1 and r2. We will consider predictor X_1 which has a split value $split_1$, such that observations in r1 are less than the split value and other observations are in r2.

$$RSS_{X_1} = \sum_{i \in r_1} (y_i - \hat{y}_{r1}) + \sum_{i \in r_2} (y_i - \hat{y}_{r2}) \tag{10.2}$$

This splitting process is repeated until a stopping threshold is reached.

10.3.2 Classification

The same recursive, top-down, greedy algorithm is used for classification except that RSS is replaced by counts of classes in regions. A measure such as accuracy is not sufficient for splitting regions so other measures such as entropy or the Gini index are used. These are discussed in the mathematical foundations section. The goal of either metric is to measure the purity of the regions, how homogenous they are.

10.3.3 Qualitative data

The discussion of the algorithm for regression and classification assumed quantitative data. When splitting qualitative data, a binary feature would make a natural split between the two values. Features with more than 2 values would assign one or more values to one branch and the remaining ones to the other. Decision trees can handle quanlitative data without having to create dummy variables.

10.4 Mathematical Foundations

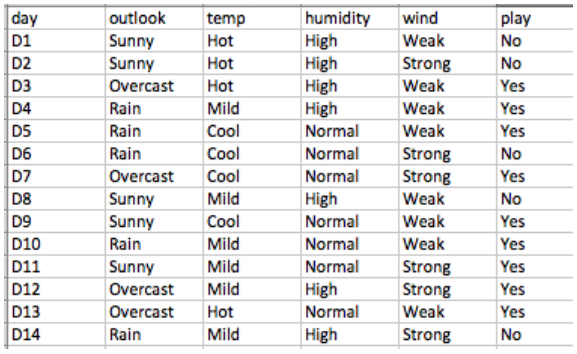
In this section we give further information about entropy, information gain, and the Gini index. We start with entropy.

10.4.1 Entropy

First we are going to talk about entropy, a measure of homogeneity of observations, and information gain, the expected reduction in entropy. The formula for entropy is given below. H() is the traditonal symbol for entropy. Notice that the probability is multiplied by the log of the probability for each class. These are summed and then negated.

$$H(data) = -\sum_{c \in C} p_c log_2 p_c \tag{10.3}$$

The probability of a class is just the fraction of data observations in that class. Let's look at the tennis data set as a simple example for these calculations.



day	outlook	temp	humidity	wind	play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Figure 10.3: Tennis Data Set

The entropy of the entire data set is determined by counting 9/14 days when we did play and 5/14 when we did not:

$$H(tennis) = -9/14 \times log_2 9/14 - 5/14 \times log_2 5/14 = .94 \tag{10.4}$$

How do we interpret entropy? The x-axis in Figure 10.4 indicates the probability. At .5 probability, entropy is maximized. There is a lot of confustion in the data, it is the opposite of homogenous. At higher or lower probabilities, we see than entropy is lower. Lower entropy means the data is more homogenous. The probability of play in the tennis data is 0.64 and our entropy was .94, very high.

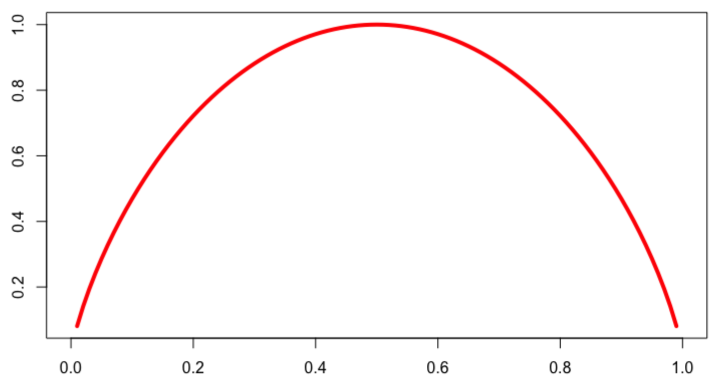


Figure 10.4: Entropy

Now let’s say we want to split on wind. Wind has two levels: weak and strong. When wind is weak we have 6 examples where we play and 2 where we do not. Let’s compute the entropy.

$$H(wind = weak) = -6/8 \times log_2 6/8 - 2/8 \times log_2 2/8 = .811 \tag{10.5}$$

Guess what the entropy of wind=strong is? Since we have an even split play=3 and not-play=3 we know our entropy is 1. In order to compute the entropy of wind we need to combine the entropy or wind=weak and of wind=strong in a formula that weights it by the relative size of these two groups.

10.4.2 Information Gain

Information gain is the reduction in uncertainty we would get given a certain choice. It is the starting entropy minus the entropy of the split. In our case, it is the entropy of the entire data set (.94) minus the weighted entropy of wind. The formula is shown below, where f represents each value of a feature and N is the total count of items.

$$IG(split) = H(data) - \sum_f \frac{|f|}{N} H(split_f) \tag{10.6}$$

$$IG(split_wind) = .94 - 8/14 * .811 - 6/14 * 1 = 0.048 \tag{10.7}$$

Running similar IG statistics releals that IG(outlook) = 0.246, IG(humidity) = 0.151, and IG(temperature) = 0.029. It turns out that outlook has the highest IG, so we would choose outlook for the decision tree split.

10.4.3 Gini index

The Gini index is mathematically similar to entropy and likewise measures homogeneity. For a given region, for each class in the region, Gini sums the products of the class probability times 1 minus the probability.

$$Gini = 1 - \sum_{k \in K} p_k^2 \tag{10.8}$$

Let's look at the Gini index for the data set.
$Gini(tennis) = 1 - (9/14)^2 - (5/14)^2 = 0.459$
The maximum Gini index is 0.5 so 0.459 is high. Let's look at the Gini index for wind.
$Gini(wind = strong) = 1 - (3/6)^2 - (3/6)^2 = 0.5$
$Gini(wind = weak) = 1 - (6/8)^2(2/8)^2 = .375$
The Gini index will give comparable results to information gain, but we want a minimal Gini and a maximum IG.

10.5 Tree Pruning

Decision trees are highly sensitive to variations in the data. For this reason if we grow decision trees out fully on a training set they are likely to overfit. One strategy to reduce overfitting is to grow the tree all the way out, then prune it back to get a subtree. T_0 represents the fully grown tree. There are $|T|$ subtrees in which some internal node has been collapsed to combine its children into one region. Let Q represent the criterion for tree building for either regression or classification. Then subtrees can be indexed by lambda:

$$Subtree_t = Q + \lambda |T| \tag{10.9}$$

The optimal subtree can be found by cross validation or using a held out validation set. If lamda = 0 our subtree is the original tree. As lambda increases, the subtree is smaller and less tuned to the training data.

There is no guarantee that a pruned tree will generalize better to new data for improved performance. However the pruned tree may be more interpretable since it should have fewer branches.

The code below shows how to use cv.tree() to try various trees. We plot the deviance by tree size. It looks like the best tree is 5. Deviance is lower at a size of 8 but we are concerned that this may overfit the data. After performing cross validation, we can prune it with the prune.tree() function, and print the pruned tree.

Code 10.5.1 — Decision Tree. Cross validation.

```
cv_tree <- cv.tree(tree1)
plot(cv_tree$size, cv_tree$dev, type='b')
tree_pruned <- prune.tree(tree1, best=5)
plot(tree_pruned)
text(tree_pruned, pretty=0)
```

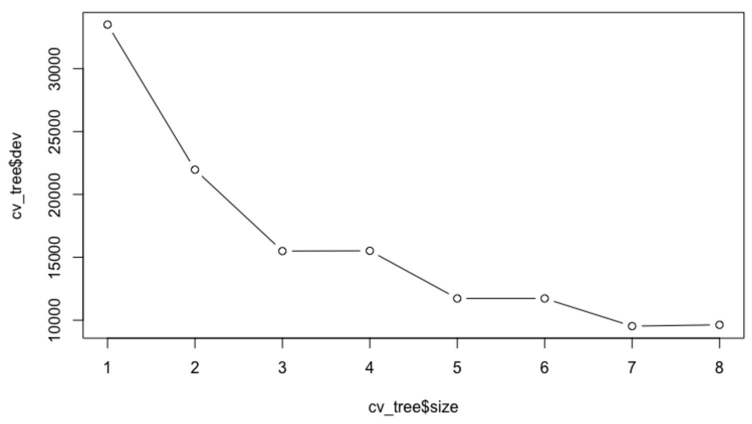


Figure 10.5: Cross Validation on Decision Tree

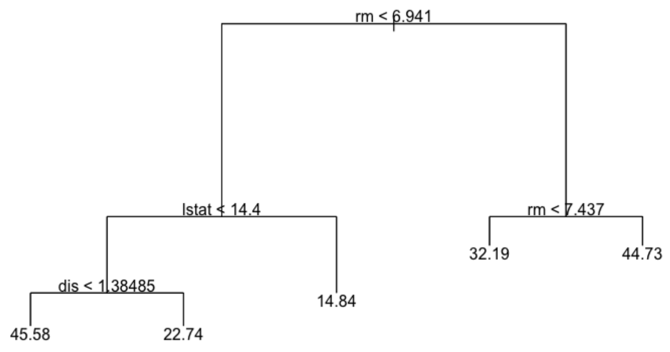


Figure 10.6: Pruned Decision Tree

Check Your Understanding 10.2 — Pruning a Decision Tree. Using the Wine Data. Try the following:

- Using the cross-validation technique demonstrated in this chapter, prune th decision tree you created for the wine data.
- Print the tree.
- Evaluate your results on the same test data as before.
- How does accuracy compare with the unpruned tree? Which tree do you prefer, and why?

10.6 Random Forests

In previous chapters we discussed the bootstrap, repeated sampling from a data set. Bagging is short for bootstrap aggregation and the main idea is to repeatedly sample the data to overcome the variance in a technique. As we have discussed, decision trees have high variance. The random forest builds on the idea of bagging but the trees are decorrelated as follows. At each split, a random subset of predictors is selected from all the predictors and one is chosen. A typical size of the subset is the square root of the number of predictors. Suppose there is a strong predictor that many bagged trees would choose first. The random forest approach prevents trees from chosing the same predictors in the same order. This allows new trees to be discovered that may outperform trees choosing the strongest predictor first. In effect, the down side of the greedy algorithm is mitigated by restricting the choice of predictors.

10.7 Cross validation, Bagging, Random Forests in R

In the github for the website is an example of a regression tree on the Boston housing market. For comparison purposes we first built a linear regression model using all predictors. The correlation of the predicted versus the test median home value was .9 and the root mean squard error was 4.35, meaning that the home price was off by about \$4,350. Running the tree algorithm on the same train/test data had a correlation of .89 and an rmse of 4.46. Very similar results to the linear regression model although linear regression was slightly better. The following code example shows how to perform cross validation on the tree and the output graph in Figure 10.5 indicates that a tree size of 5 should give good results, perhaps without overfitting. Next the code shows that we prune the tree. The pruned tree is shown in Figure 10.6. The pruned tree has 5 leaf nodes compared to 8 in the unpruned tree which you can view online. The pruned tree had a correlation of 0.85 and an rmse of 5.18. This shows a reduced performance but has the advantage of being more interpretable with fewer leaf nodes. Next we try bagging and random Forest on the same data. The argument `importance` tells the random Forest algorithm to consider predictor importance. Having this variable set to TRUE essentially performs bagging. We see in the output, shown below the code example, that 500 trees were created, trying 4 variables at each split. The test correlation on the bagged trees jumped to 0.96 and the rmse decreased to 2.87, meaning that we were off on the home price by only an average of \$2,870.

Code 10.7.1 — Random Forest. Bagging.

```
library(randomForest)
set.seed(1234)
tree_bagged <- randomForest(medv~., data=train, importance=TRUE)
tree_bagged
```

```
Call: randomForest(formula=medv ~ .,data=train,importance=TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4
          Mean of squared residuals: 10.84628
                    % Var explained: 86.83
```

In the online notebook you can see that we also tried the random forest by removing the importance=TRUE argument. The results were very similar to the results from bagging but bagging very slightly outperformed the random Forest. Recall that the difference between bagging and random Forest is that the random Forest is discouraged from predicting the strongest predictor first so that the trees are more unique. In the Boston data, rm was the strongest predictor on all the models so it seems that choosing this as the first split is the best choice. The strongest predictor may not always be the best initial split, however.

Check Your Understanding 10.3 — Baggind and Random Forest. Using the Wine Data.

Try the following:

- Using the unpruned decision tree you created earlier, try bagging.
- How many trees were created?
- The results from bagging talk about an OOB estimate of error. Research what that means and write a 1-2 sentence explanation.
- Now try random forest.
- Compare the output of the bagging and the random forest. What do you observe?
- Compare the results of bagging and random Forest to your earlier results in terms of accuracy.
- Besides accuracy, what other considerations would cause you to prefer one algorithm over the others?

■

10.8 Summary

As you can see in the Boston example online, linear regression and the regression tree got fairly similar results. Linear regression will typically outperform decision trees when the underlying function is linear. However if the relationship between the predictors and the target is not linear and complex, decision trees will probably perform better. In the Check Your Understanding examples, you observed that decision trees can perform similarly to logistic regression, particularly with bagging or random forests.

Decision trees are considered to be non-parametric algorithms. Decision trees have an advantage in being easily interpretable. Decison trees have low bias and high variance. Decision trees are sensitive to the distribution of predictors so slightly different data can result in very different trees. This high variance can be overcome with bagging or random forests. Pruning the tree may help it generalize to new data by holding back from overfitting but this is no guarantee of improving performance.

10.8.1 Quick Reference

Reference 10.8.1 Build Decision Tree

```
library(tree)
tree1 <- tree(y~., data=df)
plot(tree1)
text(tree1, cex=0.5, pretty=0)
```

Reference 10.8.2 Evaluate Decision Tree

```
# remove type="class" for regression
tree_pred <- predict(tree1, newdata=test, type="class")
table(tree_pred, test$type)
mean(tree_pred == test$type)
```

Reference 10.8.3 Cross Validation and Pruning

```
# cross validate to find  best
cv_tree <- cv.tree(tree1)
plot(cv_tree$size, cv_tree$dev, type='b')
# prune to best
tree_pruned <- prune.tree(tree1, best=5)
```

Reference 10.8.4 Bagging or Random Forest

```
library(randomForest)
set.seed(..)
# importance=TRUE means bagging instead of random forest
tree_bagged <- randomForest(y~., data=df, importance=TRUE)
tree_bagged
```

10.8.2 Lab

Problem 10.1 — Practice Decision Trees on the Abalone Data. Try the following:

1. Use the abalone data available on the github. Load the data, providing names as done in the labs for Chapter 3 and 4.
2. Divide the data into train and test sets.
3. Perform linear regression and evaluate correlation and mse on the test set.
4. Perform knn regression and compute correlation and mse. You will need to convert sex from a factor to integer before scaling. Copy your train and test sets to a new name so the original train and test are preserved.
5. Create a decision tree, print the tree, and evaluate on the test data.
6. Try bagging and compute your metrics on the test data.
7. Compare the results on all models and discuss why you think the best model was able to outperform the others.

Problem 10.2 — Classification on the Heart Data. Try the following:

- Load the processed.hungardian.data.csv from the git hub. Make sure class is a factor.
- Remove columns 11:13 because they have too many NAs, and use complete.cases() to get rid of remaining NAs.
- Split the data into 80-20 train and test.
- Create a logistic regression model on the data and evaluate its accuracy on the test data.
- Create an additional train and test set that are normalized for knn.
- Run knn with k=10 and evaluate its performance on the test data.
- Create a decision tree, print it, and evaluate its performance.
- Try the random forest algorithm and evaluate its performance.
- Comment on which algorithm performed best.

Problem 10.3 — Classification on the Glass Data. Try the following:

- Load the Glass data set from package mlbench. Research this data set and write a brief description of the columns.
- Divide the data into 80-20 train-test.
- Create scaled versions of the train and test data for kNN. Try various values for k and see how accurate the algorithm can get.
- Create a decision tree, print it, and evaluate its accuracy on the test data.
- Try random forest on the data and evaluate its accuracy.
- Comment on which cut you think is best, and why.
- Discuss what we would have to do with the data to use logistic regression.

10.8.3 Exploring Concepts

Problem 10.4 Compare how linear regression makes predictions compared to decision trees.

Problem 10.5 Compare how kNN makes predictions compared to decision trees.

Problem 10.6 Explain why linear regression has high bias and decision trees have high variance.

10.8.4 Going Further

Professor Lior Rokach from Tel-Aviv University has shared his overview of decision trees here: `http://www.ise.bgu.ac.il/faculty/liorr/hbchap9.pdf`