Data Mining with R

Decision Trees and Random Forests

Hugh Murrell

reference books

These slides are based on a book by Graham Williams:

Data Mining with Rattle and R,
The Art of Excavating Data for Knowledge Discovery.

for further background on decision trees try Andrew Moore's slides from: *http://www.autonlab.org/tutorials*

and as always, **wikipedia** is a useful source of information.

classification

A Major Data Mining Operation

Given one attribute in a data frame try to predict its value by means of other available attributes in the frame.

Applies to predicting categorical attributes

- Categorical attribute: a column which takes on two or more but a finite number of discrete values.
- Real attribute: a column of real numbers.

Today's lecture is about:

- Information Gain for measuring association between attributes
- Using information gain to **learn** a decision tree classifier from data

example data set

For purposes of demonstration we will again make use of the classic **iris** data set from R's datasets collection.



The `Species` attribute in the iris dataset is categorical.

```
> summary(iris$Species)
```

```
    setosa versicolor  virginica
        50         50         50
```

The other four iris attributes are real valued descriptors.
Can we use these real valued attributes to predict iris species?

decision trees

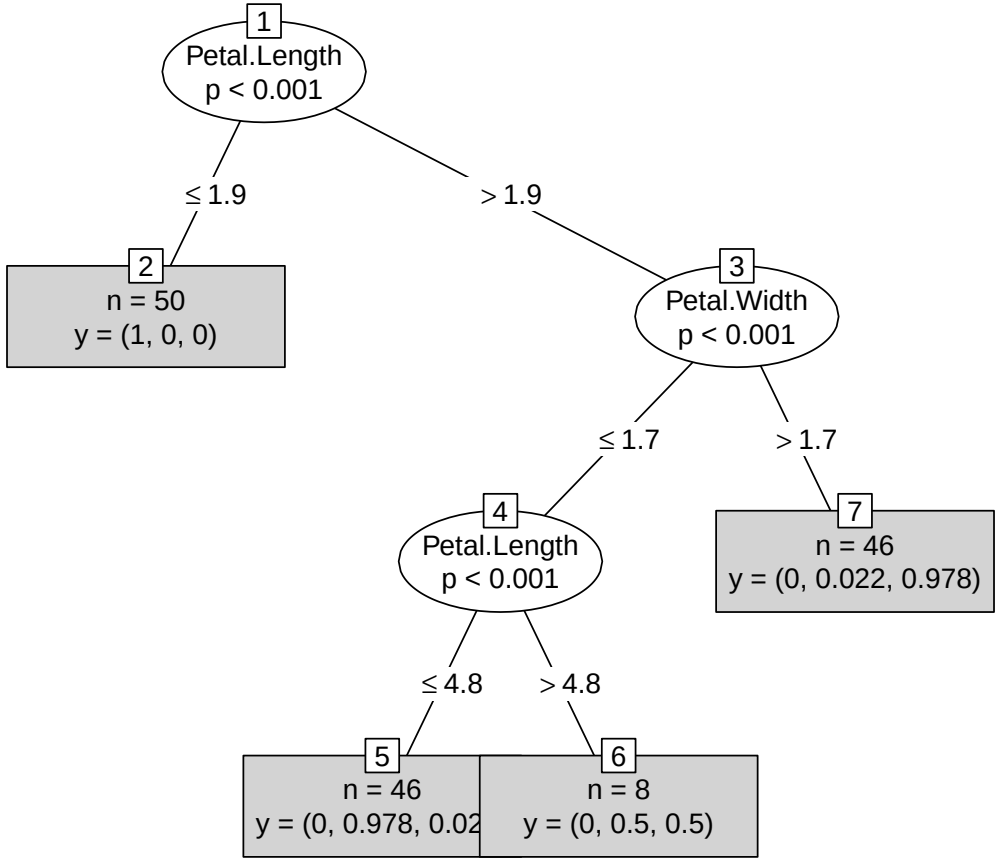
A decision tree uses the traditional tree structure from your second year data structures module.

It starts with a single root node that splits into multiple branches, leading to further nodes, each of which may further split or else terminate as a leaf node.

Associated with each nonleaf node will be a test that determines which branch to follow.

The leaf nodes contain the **decisions**.

a decision tree for the iris dataset



decision tree topologies

There are variations to the basic decision tree structure for representing knowledge.

Some approaches limit trees to two splits at any one node to generate a binary decision tree.

In the decision tree on the previous slide the decision variables are real valued and one real number is used to generate the decision split.

For categoric data a binary decision may involve partitioning the levels of the variable into two groups.

Another approach is to have a branch corresponding to each of the levels of a categoric variable to generate a multiway tree.

partitioning the dataset

Decision tree induction is accomplished using a recursive partitioning approach.

We want to find a decision variable that can be used to split the dataset into two smaller datasets.

At each step we have identify a question, that we use to partition the data. The resulting two datasets then correspond to the two branches of the tree emanating from that node.

For each branch, we identify a new question and partition appropriately, building our representation of the knowledge we are discovering from the data.

We repeatedly partition the dataset and applying the same process to each of the smaller datasets; thus it is a **recursive partitioning** algorithm.

the decision tree induction algorithm is greedy

At each stage of the process, we try to find the **best** variable and split to partition the data.

That decision may not be the best to make in the overall context of building this decision tree, but once we make that decision, we stay with it for the rest of the tree.

This is generally referred to as a **greedy** approach and may not result in the best overall decision tree.

At each split the **goal** is to increase the **homogeneity** of each of the two resulting datasets with respect to the target variable.

What do we mean by **homogeneity**?

the split decision

We use **information gain** for deciding between alternative splits.

The concept comes from *information theory* and is related to *entropy* (the amount of disorder in a system).

We discuss the information gain here in terms of a categorical target variable, but the concept generalises to real valued target variables for regression purposes.

entropy

disorder relates to how *mixed* our dataset is with respect to the values of the target variable.

If the dataset contains only observations that all have the same value for the target variable then there is no disorder (entropy is 0).

If the values of the target variable are equally distributed across the observations then the dataset contains the maximum amount of disorder (entropy is 1).

Datasets containing different mixtures of the values of the target variable will have a measure of entropy between 0 and 1.

entropy ...

From an information theory perspective, we interpret an entropy measure of 0 as indicating that we need no further information in order to classify a specific observation within the dataset. All observations belong to the same class.

Conversely, an entropy measure of 1 suggests we need the maximal amount of extra information in order to classify our observations into one of the available classes.

entropy ...

The entropy of a sample D with respect to a target variable of k possible classes, C_i, is defined as:

$$H(D) = -\sum_{i=1}^{k} P(C_i|D) \log_k(P(C_i|D))$$

where the probability of class C_i in D is obtained directly from the dataset according to:

$$P(C_i|D) = \frac{\text{number observations in } D \text{ with label } C_i}{\text{total number of observations in } D}$$

entropy ...

Note that if the observations are evenly split amongst all k classes then:

$$H(D) = -\sum_{i=1}^{k} \frac{1}{k} \log_k(\frac{1}{k}) = 1$$

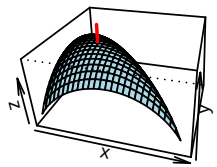
and note that if all the observations are from one class then

$$H(D) = 0$$

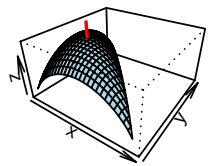
which is why we choose entropy as our measure of disorder.

the entropy surface for a 3 class target variable

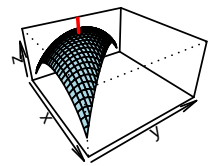
theta = 15 deg



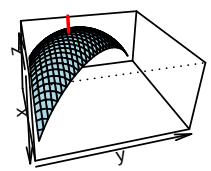
theta = 35 deg



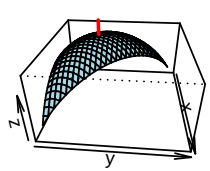
theta = 55 deg



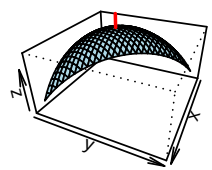
theta = 75 deg



theta = 95 deg



theta = 115 deg



information gain

We now define the weighted entropy of a decision/split as follows:

$$H(D_L|D_R) = \frac{|D_L|}{|D|}H(D_L) + \frac{|D_R|}{|D|}H(D_R)$$

and then the **information gain** for the split is:

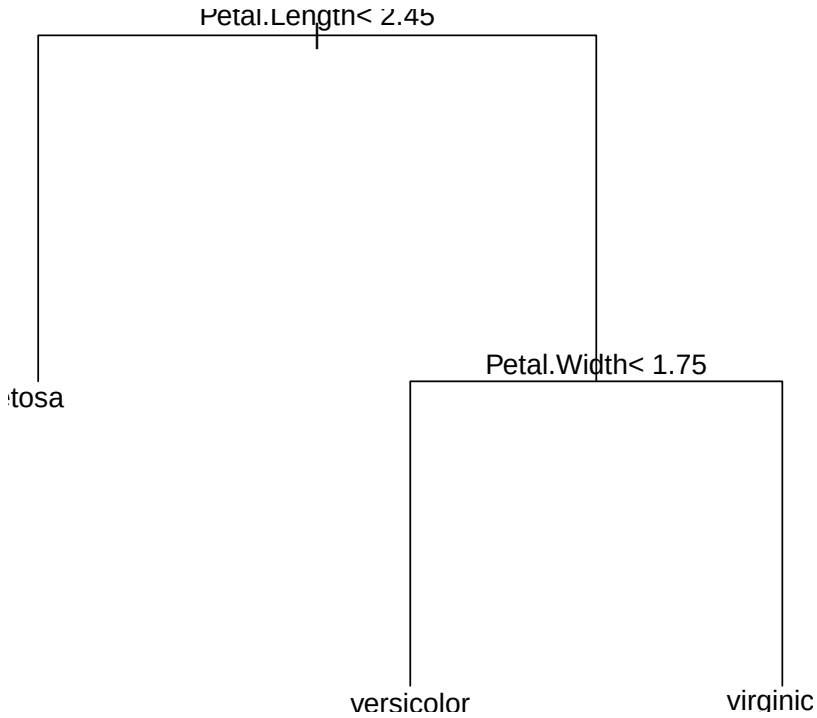
$$IG(D, D_L, D_R) = H(D) - H(D_L|D_R)$$

In other words IG is the expected reduction in entropy caused by knowing the value a attribute.

On the next slide we present the `rpart` package which uses maximum information gain to obtain best split at each node.

recursive partitioning with rpart

```
> library(rpart)                          # load package
> target <- Species ~ .                   # target variable
> dt <- rpart(target, data=iris)          # build tree
> plot(dt); text(dt)                      # draw the tree
```



overtraining

Of course it is always possible to generate a perfect decision tree by continuing to split nodes until only one sample is left in the dataset thus ensuring a correct classification.

This **overtraining** approach usually results in an uniterpretable decision tree and so a package like `rpart` allows the user to set parameters that control the depth of the decision tree via pruning techniques.
For example try running `rpart` on the iris dataset with the following control parameter settings:

```
> control = rpart.control(cp = 0.0, minsplit=0)
```

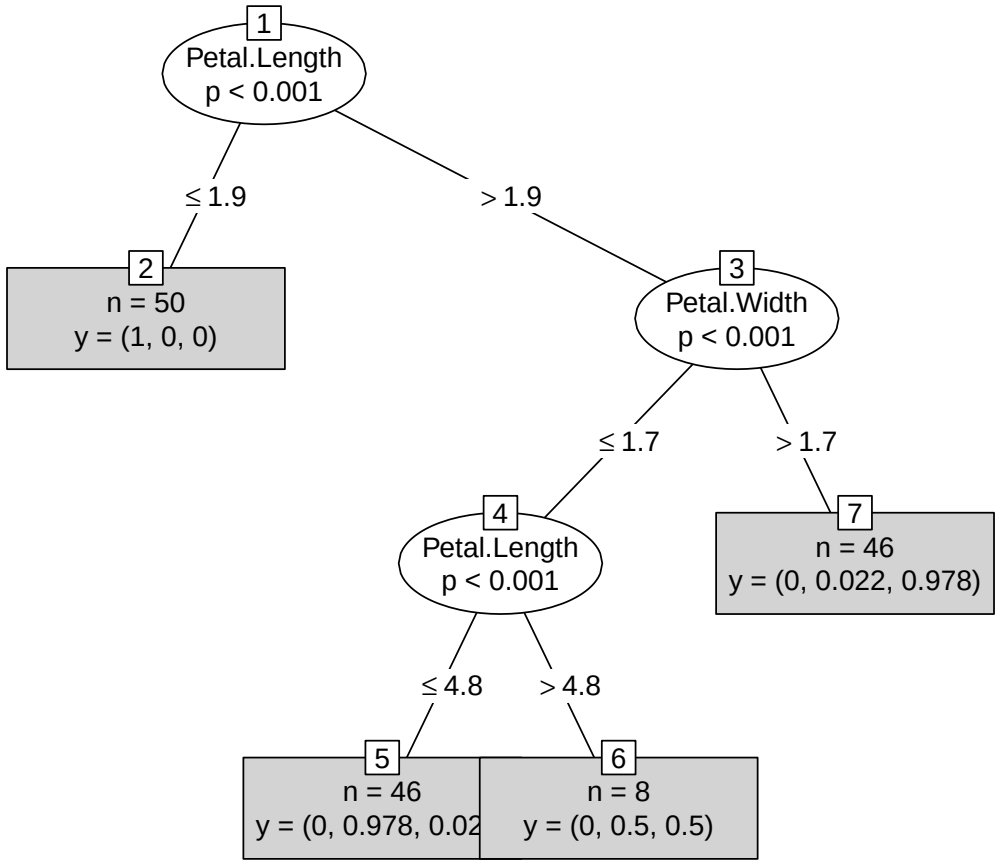
conditional inference trees

Hothorn et al. (2006) introduced an improvement to the `rpart` approach for building a decision tree, called conditional inference trees. Conditional inference trees address the overfitting and variable selection biases of `rpart` by making use of statistical *p values*.

A conditional inference tree for the iris dataset can be generated using the `party` package as follows:

```
> library(party)
> target <- Species ~ .
> cdt <- ctree(target, iris)
> plot(cdt,type="simple")
```

a conditional inference tree for the iris dataset



predicting target variables using a decision tree

The success of the tree in predicting its own training data can then be examined using:

```
> table(predict(cdt), iris$Species)
```

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	49	5
virginica	0	1	45

Prediction on the training set is not an accepted method for rating a machine learning algorithm. One should split your dataset into training and testing subsets, then train your algorithm on the training subset, and then test its prediction powers on the testing subset.

train and test conventions

For example we could split, train and test a decision tree for the iris dataset as follows:

```
> train <- c(sample(50,25),
+            50 + sample(50,25),
+            100 + sample(50,25))
> iris_train <- iris[train,]
> iris_test <- iris[-train,]
> target <- Species ~ .
> cdt <- ctree(target, iris_train)
> table(predict(cdt,newdata=iris_test), iris_test$Spec
```

	setosa	versicolor	virginica
setosa	25	0	0
versicolor	0	24	5
virginica	0	1	20

random forests

Building a single decision tree provides a simple model of the world, but it is often too simple or too specific.

Many models working together are often better than one model doing it all.

In building a single decision tree there is often very little difference in choosing between alternative variables.

Two or more variables might not be distinguishable in terms of their ability to partition the data into more homogeneous datasets.

The *random forest* algorithm builds all equally good trees and then combines them into one model, resulting in a better overall model.

random forests ...

In a random forest each decision tree is built to its maximal depth. The individual decision trees are not pruned.

Each individual tree will overfit the data, but this is outweighed by the multiple trees using different variables and (over) fitting the data differently.

The randomness used by a random forest algorithm is in the selection of both observations and variables.

In building a single decision tree in the forest the algorithm considers a random subset of the observations from the training dataset. Also, at each node in the process of building the decision tree, only a small fraction of all of the available variables are considered when determining how to best partition the dataset.

the random forest package

```
> library(randomForest)
> target <- Species ~ .
> rf <- randomForest(target, data=iris,
+                    ntree=1000, proximity=TRUE)
> table(predict(rf), iris$Species)
```

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	47	4
virginica	0	3	46

```
> 100*(sum(predict(rf)==iris$Species)/nrow(iris))
[1] 95.33333
```

Naturally, due to the random nature of the algorithm, we get different success rates each time the model is built.

exercises

Download the *seeds* dataset from my website.

Use the `party` package to derive a decision tree that predicts `Variety` from the **full** data set. Compute the success rate of your decision tree on the **full** data set.

Split the dataset **sensibly** into training and testing subsets.

Make use of the `party` package to create a decision tree from the training set and use it to predict `Variety` on the test set.

Compute the success rate of your decision tree on the **test** data set.

Comment on differences in the two success rates.

exercise continued ...

e-mail your solution to me as a single R script by:

6:00 am, Monday the 11th April.

Make sure that the first line of your R script is a comment statement containing your name, student number and the week number for the exercise. (in this case week 06).

Use: `dm06-STUDENTNUMBER.R` as the filename for the script.

Use: `dm06-STUDENTNUMBER` as the subject line of your email.

There will be no extensions. No submission implies no mark.