PG Diploma in Data Analytics

Course: Predictive Analytics - II

Instructor: Prof. G. Srinivasaraghavan

PGDDA-Lecture Notes/PA-II/4

Tree Models

Contents

1 Learning Objectives **4**

2 Appendix: Legend and Conventions, Notation **5**

3 Introduction **6**

4 Algorithms for Decision Tree Construction **11**

4.1 Core Idea . . . 12

4.2 Homogeneity Measures . . . 14

4.3 Gini Index . . . 14

4.4 Information Gain / Entropy-based . . . 15

4.5 Variance of the Target Variable . . . 18

5 Truncation and Pruning of Decision Trees **18**

5.1 Decision Tree Stopping Criteria (Truncation) . . . 19

5.1.1 χ^2 (Chi-Square) Stopping Criterion . . . 19

5.2 Decision Tree (Post)-Pruning . . . 21

5.3 Rule Post-Pruning . . . 21

5.4 Other Implementation Specifics . . . 23

5.4.1 Continuous Valued Attributes . . . 23

5.4.2 Handling Attributes with Different Costs . . . 23

5.4.3 Handling Missing Values . . . 24

List of Figures

1 Bank Marketing Decision Tree 6

2 Heart HDecision Tree 7

3 Multiway Split 9

4 Decision Tree Regression 9

5 Overfit Decision Tree 11

6 Heart Decision Tree Perturbed 12

7 Example for Homogeneity Measures 14

8 χ^2 Stopping Criterion 20

9 Pruned Heart Tree 23

List of Tables

1 Learning Objectives

After this module you are expected to be familiar with Decision Trees as a tool in your Machine Learning toolbox. Specifically at the end of this module you are expected to be:

1. clear about what decision trees are and why they are useful
   (a) Basic concepts and Structure of a decision tree
   (b) Pros and Cons
2. familiar with some of the algorithms used for constructing decision trees
   (a) Decision tree Algorithms in general
   (b) Specific measures used for measuring homogeneity
   (c) Stopping Criteria
   (d) Pruning strategies
3. able to build tree models in R and evaluate their suitability for your problem

2 Appendix: Legend and Conventions, Notation

There are several parts of the test that have been highlighted (in boxes). The document uses three kinds of boxes.

Example 0: Title

One of the running examples in the document. The reader must be able to reproduce the observations in these examples with the corresponding R code provided in Section **??**.

Title

Good to remember stuff. These boxes typically highlight the key take-aways from this document.

Title

Things that are good to be aware of. This is primarily targeted at the reader who wishes to explore further and is curious to know the underlying (sometimes rather profound) connections with other fields, alternative interpretations, proofs of some statements, etc. However one could safely skip these boxes completely if you are only interested in the primary content/message of this document.

Title

`Code snippets. Implementation Tips. Do's and Dont's.`

$\boldsymbol{x}$	A *vector* $\boldsymbol{x}$
x	A scalar quantity x
term	Refer to the glossary as a quick reference for 'term'. These are typically prerequisite concepts that the reader is expected to be familiar with.
`code`	Code snippets, actual names of library functions, etc.

3 Introduction

Decision Trees naturally represent the way we make decisions. Think of a machine learning model as a decision making engine that takes a decision on any given input object (data point). Imagine a doctor making a decision (the diagnosis) on whether a patient is suffering from a particular condition given the patient data, an insurance company making a decision on whether claims on a particular insurance policy needs to be paid out or not given the policy and the claim data, a company deciding on which role an applicant seeking a position in the company is eligible to apply for, based on the past track record and other details of the applicant, etc.. Solutions to each of these can be thought of as machine learning models trying to mimic the human decision making.

Refer to Figures 1 and 2 for a couple of examples built from representative UCI datasets. The Bank Marketing dataset (Figure 1) consists of data "is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed." The Heart dataset (Figure 2) consists of data about various cardiac parameters along with an indicator column that says whether the person has a heart disease or not.

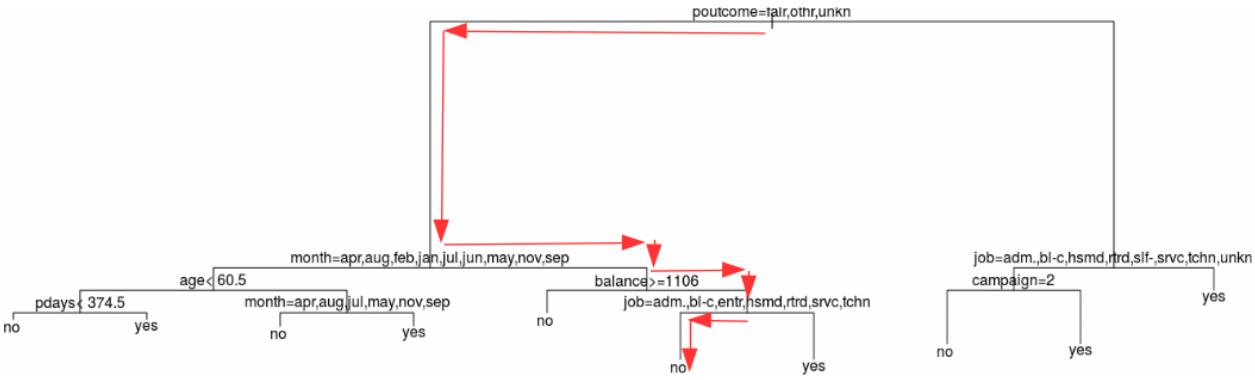

Figure 1: Bank Marketing Decision Tree built from the UCI dataset at `https://archive.ics.uci.edu/ml/datasets/Bank+Marketing`

R Code for Figure 1

```
library(rpart)
bank <- read.csv('bankdata.csv')

# Cast the cols to the correct types
cols <- c(1,6,13:15)
bank[,cols] <- as.integer(as.character(unlist(bank[,cols])))
cols <- c(2:5,7:9,11,13,16)
bank[,cols] <- lapply(bank[,cols], factor)

# Remove unnecessary cols
```

```
bank$day <- NULL
bank$duration <- NULL

# Build the decision tree
banktree <- rpart(y ~ .,  data=bank, method='class',
                  control=rpart.control(minsplit=65, cp=0.001))
plot(banktree)
text(banktree, pretty=TRUE)
```

Figure 2: Heart Decision Tree built from the UCI dataset on Heart Disease at `https://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29`

R Code for Figure 2

```
library(rpart)
heart <- read.csv('heartdata.csv')

# Cast the cols to the correct types
cols <- c(1,4,5,8,10,12)
heart[,cols] <- as.numeric(as.character(unlist(heart[,cols])))
cols <- c(2,3,6,7,9,13)
heart[,cols] <- lapply(heart[,cols], factor)
cols <- c(11)
heart[,cols] <- as.integer(as.character(unlist(heart[,cols])))

# Build the decision tree
hearttree <- rpart(heart.disease ~ .,  data=heart, method='class')
plot(hearttree)
text(hearttree, pretty=TRUE)
```

Without getting into the domain details of each of the terms in the datasets (please visit the respective urls for descriptions of each of the attributes), in fact the decision trees can be

interpreted quite naturally. In the Bank Marketing example shown in Figure 1 the leaf nodes (bottom) are labelled `yes` (the customer will subscribe for a term deposit) or `no` (the customer will not subscribe for a term deposit). The decision tree predicts that if the outcome of the call with the customer was `fail`, the contact month was `march`, if the current balance exceeds `$1106` but the customer is `unemployed` then he/she will not subscribe to a term deposit — the path indicated by the red arrows in Figure 1. Note that every node (split junction) in the tree represents a test on some attribute of the data. As a matter of convention we go to the left part of the tree if the test passes, else go the right subtree. The example given above represents the path `left->right->right->left` starting from the top (the root). For the heart dataset the leaf nodes (bottom) are labelled 1 (no heart disease) or 2 (has heart disease). The decision tree model predicts that if a person has `thal` of type 3 (normal), `pain.type` other than `{1,2,3}` and the number of blood vessels `flouroscopy.coloured` more than 0.5, then the person has heart disease. The example given above represents the path `left->right->right` starting from the top (the root). In general in a decision tree:

- The leaf nodes represent the final decisions.
- Each intermediate node represents a simple test on one of the attributes.
- The path from the root to a leaf corresponds to a conjunction of tests at each of the nodes on the path. We say a test data point 'follows a path' on a decision tree if it passes all the tests on the path in the decision tree. The branches out of an intermediate node are exclusive — the test data point can follow exactly one branch out of every intermediate node it encounters.
- The prediction by a decision tree on a data point is the one corresponding to the leaf at which the path followed by the data point ends.
- There could be multiple leaves representing the same class (decision). For example in the heart disease example, this simply means that a person does not have heart disease if: (`thal=3` and `pain.type in {1,2,3}`) or (`thal=3` and `pain.type not in {1,2,3}` and `flouroscopy.coloured<0.5`) or (`thal!=3` and `flouroscopy.coloured<0.5` and `exercise.angina=0` and `age>=51`). Note that when the `thal` is normal, by and large the heart is normal. So in some sense the `thal` type is a major indicator of heart disease (this is apparent from the length of the leader lines from the root node). The last condition (1 leaf on the right branch) may seem a little counter-intuitive. An abnormal `thal` (right branch) is probably expected at age beyond 51 and so is not considered heart diseased, whereas at an age below 51 would be considered heart disease. In general the decision by tree is a value y represented by some of the leaves if the OR of the conditions corresponding to the paths from the root to each of the leaves with value y, is true for the given data point.

We generally assume, at least for explanation, the decision trees we consider are binary — every intermediate node has exactly two children. This is not a restriction since any more general tree can be converted into an equivalent binary tree. In practice however splits on attributes that have too many distinct values (for example a continuous valued attribute) are usually implemented as binary splits and splits on attributes with not many distinct values are implemented as multi-way splits. Figure 3 illustrates multiway split on an attribute A.

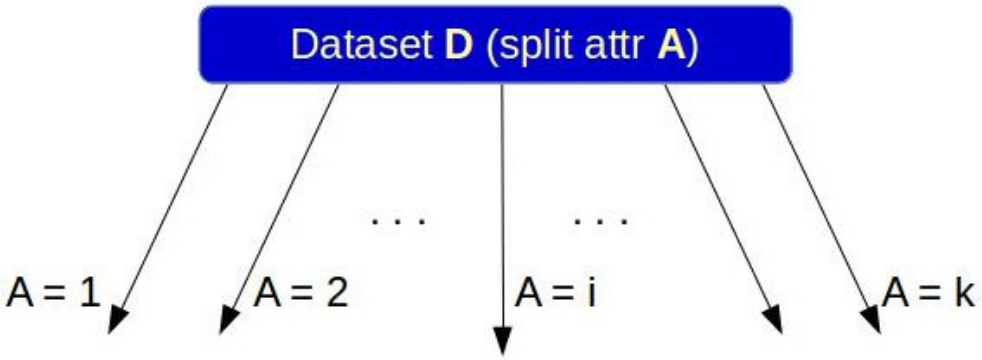

Figure 3: Multiway Split

The examples we have given are of the binary classification kind. However it is easy to see that this extends to multiclass classification as well with no change whatsoever to our description of a decision tree given above — the leaves would simply represent various class labels. It is also possible to extend decision trees to regression. Consider the dataset shown in Figure 4. It is a simple synthetic dataset where the y-value is just a constant with some noise thrown in three ranges of x-values — $0 < x \leq 1000$, $1000 < x \leq 2000$ and $2000 < x \leq 3000$. The decision tree identifies these three ranges and assigns the average y-value to each range.

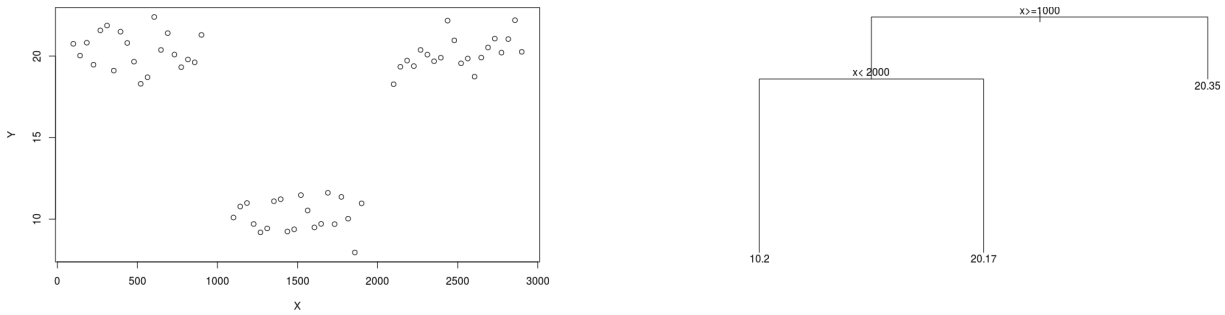

Figure 4: Decision Tree Regression

R Code for Figure 4

```
library(rpart)
seglength <- 1000
delta <- 100
segpoints <- 20
yref <- 10

# Generate the synthetic dataset and plot it
df <- data.frame("x"=numeric(),"y"=numeric())
```

```
for (i in c(0:2)) {
  xvals <- seq(i*seglength+delta, (i+1)*seglength-delta, length.out=segpoints)
  ybase <- if (i == 1) yref else 2*yref
  yvals <- ybase + rnorm(segpoints)
  newrows <- cbind(xvals, yvals)
  df <- rbind(df, newrows)
}
names(df) <- c('x', 'y')
plot(df$x, df$y, ylab='Y', xlab='X')

# Construct the decision tree and plot the tree
dftree <- rpart(y ~ x,  data=df, method='anova')
plot(dftree)
text(dftree)
```

Decision Trees provide a natural interpretation in terms of the typical decision making process that we adopt when we make manual decisions. It is easy to think of an implicit rule for some decision making in terms of an (possibly nested) `if-then-else` rule. Notice that, that's exactly what a decision tree is — a nested `if-then-else`. Consider a decision tree for a binary classification problem and let $P_1, \ldots, P_k$ represent the logical conditions corresponding to the paths from the root to the leaves that give the answer YES. Then the behaviour of the decision tree is precisely:

if (P_1 or P_2 or ... or P_k) decision=YES else decision=NO

This is one of the biggest advantages of a decision tree — the predictions made by a decision tree model are very easily interpretable; the 'why' for a certain decision can be answered by just following the path taken by the decision tree leading up to the decision leaf.

Another key advantage of a decision tree is that it does not assume anything specific about the nature of the values in the dataset. It can seamlessly handle all kinds of data — numeric, categorical, strings, boolean, and so on. Unlike many other methods in ML it also does not require that the columns are normalized to bring all the numeric columns in the data to more-or-less the same scale. Decision tree algorithms are also relatively immune to outliers and missing values in the data.

Decision trees often give us an idea of the relative importance among the explanatory attributes that are used for prediction. For instance in the figures of decision trees shown above the 'length' of the leader lines from each of the nodes to its children is an indication of relative importance. This also roughly corresponds to the level at which a test for that attribute occurred in the decision tree — closer it is to the root, relatively more important it is.

Decision trees are typically immune to noise in the data — noise might 'shift' the split point a little bit, however it very unlikely the quality of the partitions and hence the decisions will change substantially due to presence of noise.

Decision trees do have their drawbacks. The prominent ones are:

- Decision trees tend to overfit the data. If allowed to grow without keeping the tree complexity under check the tree goes out of its way to 'fit' each data point. So some pruning

strategy is a must as part of any realistic implementation of decision trees. It is easy to see this by changing the call to `rpart` in the code given above for generating Figure 2 to

```
hearttree <- rpart(heart.disease ~ .,  data=heart, method='class',
                   control=rpart.control(minsplit=1, cp=0.001))
```

This will make the decision tree fit every data point perfectly with zero residual error (call `residue(.)` on the `rpart` object returned to check this). Figure 5 shows the overfit decision tree. Can you think of a way to create, for any given dataset, a decision tree that will make zero error on that dataset?

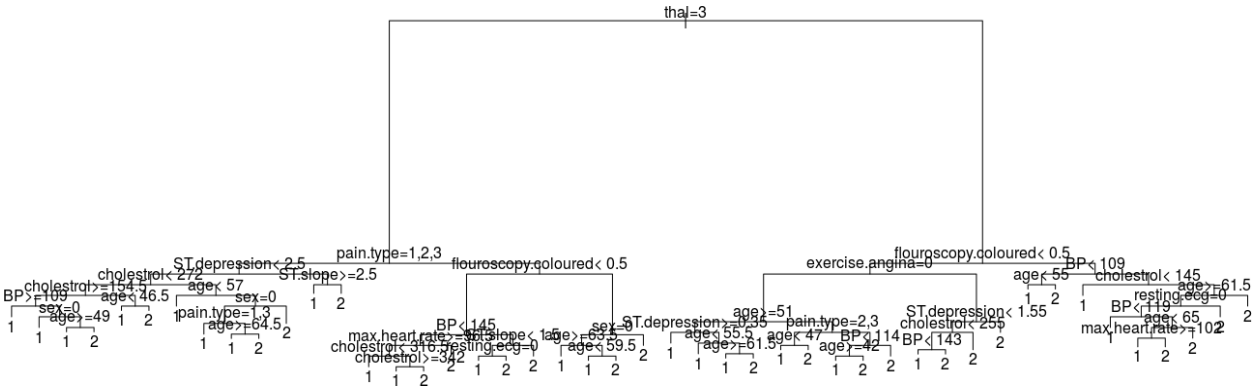

Figure 5: Overfit Decision Tree for the Heart Dataset

- Decision trees tend to be very unstable. See Figure 6 that shows the decision tree generated from a random 90% sample from the original dataset. Notice that the tree looks very different from the one in Figure 2 that was generated from the full dataset. This is a demonstration of the instability of decision tree models — small changes in the dataset can change the decision tree (the model) significantly. Recall from our discussion as part of model selection that generalizability suffers when the model variance is high (model is unstable). So in a way the instability of the decision tree and its tendency to overfit are related and are symptoms of a broader phenomenon.

4 Algorithms for Decision Tree Construction

We explore some algorithms that are used for decision tree construction. The problem we are trying to solve is: given a dataset (say for supervised classification) in the tabular form — each row represents a data point and each column represents an attribute along with a special column for the target attribute — how do we construct a decision tree that represents a succinct 'rule' to 'explain' the target attribute values from the explanatory attributes.

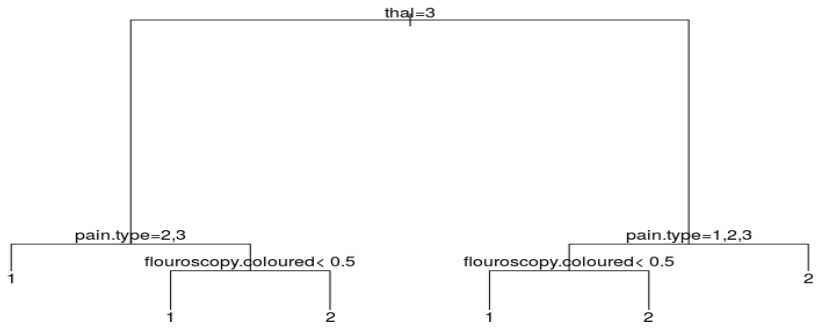

Figure 6: Decision Tree for a slightly perturbed Heart Dataset

4.1 Core Idea

It is instructive in this case to consider a simple, almost trivial, example before we generalize it to understand what lies behind decision tree construction algorithms. Consider a dataset in which all the values in the target attribute are identical (some constant value y — same class throughout or same value in case of regression). What rule would best capture this dataset? The answer is trivial — the identity rule that assigns the given constant value y to the target attribute of any data point, no matter what the values of the explanatory attributes are. This is admittedly too trivial. Suppose the heart dataset that we looked at in the previous section, is such that everyone with blood pressure above 150 has heart disease and those with BP below 150 are fine. Then clearly the BP is the only unambiguous indicator of heart disease. So the decision tree will be one with just one node in it: given data for a new person, check the person's BP and decide YES if it happens to be above 150 and decide NO otherwise. This is the tree with the root checking if the BP is > 150 and two leaves as the children of the root — left child for YES and the right child for NO. Notice that once we split the training data points into those with BP> 150 and the rest, the two partitions are now homogeneous — the value of the target variable becomes uniform in each of the two partitions and we can therefore assign the same value to any other data point that falls in that bucket. So in general what all decision tree algorithms do is the following: recursively partition the dataset into subgroups by splitting on some attribute every time till the remaining partitions are all nearly homogeneous (pure). The splits become the intermediate nodes and the final partitions are made leaves with the value of the homogeneous target attribute attached to it. Where the different algorithms differ is in their definition of *homogeneity* / *purity*. There are also subtlelities involved in the way the different algorithms carry out pruning of the trees to avoid overfitting.

There are few broad thumbrules that are easy to recognize in any reasonable measure of homogeneity.

- If the target attribute values are all identical — this is as homogeneous as the list can be. So the homogeneity measure must be maximum for this case.

- A dataset with an overwhelming majority of one value for the target attribute (say 90% values are 1 and the rest 10% are 0) is more homegeneous than a dataset where the target attribute values are 50-50.
- When every value for the target attribute is equally likely, the homogeneity must be minimum

All decision tree algorithms essentially work in a 'greedy' manner — at every level trying to find a split that will partition the dataset in such a way that maximizes the 'homogeneity' in the partitions induced by the split. In general therefore any decision tree algorithm can be fit into the following pseudocode pattern. Let us denote tests that can be carried out on the i^{th} attribute as C_{ij} — $C_{i1}, \ldots, C_{ik}$ are the different tests that can be carried out on the attribute value. Let's denote the test C_{ij} done on a data point $\boldsymbol{d}$ as $C_{ij}(\boldsymbol{d})$ returning a boolean value. The generic template for a decision tree algorithm is given in Algorithm 1.

Algorithm 1 Generic Decision Tree Algorithm

procedure GROWDECISIONTREE(dataset D)
  Initialize the decision tree T to an empty tree
  h = Homogeneity(D)
  if $h >$ threshold **then**
    ▷ The current partition is adequately homogeneous
    ▷ partitioning further is not worth it
    T = Leaf Node with the majority/average value in the current partition
  else
    for Tests C_{ij}, across attributes i, and tests on i (indexed with j) **do**
      $L_{ij} = \{\text{all } \boldsymbol{d} \in D \text{ such that } C_{ij}(\boldsymbol{d}) \text{ is true}\}$
      $w^L_{ij}, h^L_{ij} = |L_{ij}|, \text{Homogeneity}\left(L_{ij}\right)$
      $R_{ij} = \{\text{all } \boldsymbol{d} \in D \text{ such that } C_{ij}(\boldsymbol{d}) \text{ is false}\}$
      $w^R_{ij}, h^R_{ij} = |R_{ij}|, \text{Homogeneity}\left(R_{ij}\right)$
      ▷ L_{ij}, R_{ij} form the partition of D induced by C_{ij}.
    end for
    $C^* = C_{ij}$ for which difference between $\left(w^L_{ij} * h^L_{ij} + w^R_{ij} * h^R_{ij}\right)$ and h is maximum
    ▷ C^* maximizes the increase in (weighted) homogeneity among the child partitions
    Make C^* the root of T — let L^* and R^* be the partitions induced by C^*
    Make GrowDecisionTree(L^*) the left child of C^*
    Make GrowDecisionTree(R^*) the right child of C^*
  end if
  return T
end procedure

Notice that the name of the common R package for building decision trees is `rpart` which is an acronymn for *recursive partitioning*, which is another term for the generic scheme described above. Yet another term that is often used for this procedure is *Top-Down Inductive Inference*. We discuss some of the specific homogeneity measures that are used in the following section.

4.2 Homogeneity Measures

In this section we look at the commonly used homogeneity measures used in decision tree algorithms. To illustrate the measures described in this section we use a simple hypothetical example of people in an organization and we want to build a model for who among them plays football. Each employee has two explanatory attributes — Gender and Age. The target attribute is whether they play football. Figure 7 illustrates this dataset — the numbers against P and N indicate the numbers of employees who play football and those who don't respectively, for each combination of gender and age.

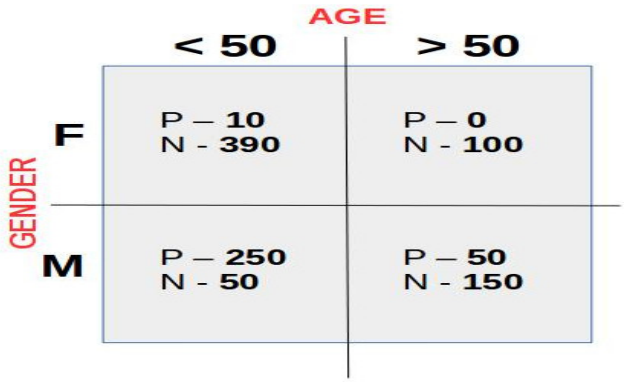

Figure 7: Example for Homogeneity Measures

4.3 Gini Index

Gini Index uses the probability of finding a data point with one label as an indicator for homogeneity — if the dataset is completely homogeneous, then the probability of finding a datapoint with one of the labels is 1 and the probability of finding a data point with the other label is zero. An empirical estimate of the probability p_i of finding a data point with label i (assuming the target attribute can take say k distinct values) is just the ratio of the number of data points with label i to the total number of data points. It must be that $\sum_{i=1}^{k} p_i = 1$. For binary classification problems the probabilities for the two classes become p and $(1-p)$. Gini Index is then defined as:

$$Gini = \sum_{i=1}^{k} p_i^2$$

Note that the Gini index is maximum when $p_i = 1$ for exactly one of the classes and all others are zero. So higher the Gini index higher the homogeneity. In a Gini based decision tree algorithm, we therefore find the split that maximizes the weighted sum (weighted by the size of the partition) of the Gini indices of the two partitions created by the split. For the example in Figure 7:

1. **Split on gender**: the two partitions will have 10/500 and 300/500 as the probabilities of finding a football player respectively. Each partition is half the total population. Therefore

$$Gini = \frac{1}{2}\left(\left(\frac{1}{50}\right)^2 + \left(\frac{49}{50}\right)^2\right) + \frac{1}{2}\left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2\right) = 0.7404$$

2. **Split on Age**: the two partitions will have 260/700 and 50/250 as the probabilities, and 700 and 300 as the sizes respectively, giving us a Gini index of:

$$Gini = 0.7\left(\left(\frac{26}{70}\right)^2 + \left(\frac{44}{70}\right)^2\right) + 0.3\left(\left(\frac{1}{5}\right)^2 + \left(\frac{4}{5}\right)^2\right) = 0.5771$$

Therefore we would first need to split on the gender — this split gives a higher GINI index for the partitions. Gini index can only be used on classification problems where the target attribute is categorical.

4.4 Information Gain / Entropy-based

The idea is to use the notion of *entropy* which is a central concept in information theory. Entropy quantifies the degree of disorder in the data. Entropy is always a positive number between zero and 1. Another interpretation of entropy is in terms of information content. A completely homogeneous dataset has no information content in it (there is nothing non-trivial to be learnt from the dataset) whereas a dataset with a lot of disorder has a lot of latent information waiting to be learnt. Assume the dataset consists of only categorical attributes, both the explanatory variables and the class variable. Again in terms of the probabilities of finding data points belonging to various classes, entropy for a dataset D is defined as

$$\mathscr{E}[D] = -\sum_{i=1}^{k} p_i \log_2 p_i$$

Notice that the entropy is zero if and only if for some i, $p_i = 1$ and all the other $p_j = 0$ — i.e., when the dataset is completely homogeneous. Consider a k-valued attribute A of the dataset. Suppose we partition the dataset into groups where each group $D_{A=i}$ consists of all the data points for which the attribute A has value i, for each $1 \leq i \leq k$. The weighted average entropy if we partition the dataset based on the values of A is

$$\mathscr{E}[D_A] = \sum_{i=1}^{k}\left(\left(\frac{|D_{A=i}|}{|D|}\right)\mathscr{E}[D_{A=i}]\right)$$

This is also the expected entropy of the partition if the dataset is split on the different values of attribute A. This corresponds to a multiway split — partitioning the dataset into groups, each of which is filtered on one value of the splitting attribute. Entropy based algorithms therefore, at each state, find the attribute on which the data needs to be split to make the entropy of the partition minimum. In practice a slightly modified measure called *Information Gain* is used. Information Gain, denoted $Gain(D, A)$, is the expected reduction in entropy for the collection of data points D if we filter on a specific value of the attribute A. Information Gain is a direct homogeneity measure — higher the information gain, higher the homogeneity achieved. Information Gain $Gain(D, A)$ is defined as

$$Gain(D, A) = \mathscr{E}[D] - \mathscr{E}[D_A] = \mathscr{E}(D) - \sum_{i=1}^{k}\left(\left(\frac{|D_{A=i}|}{|D|}\right)\mathscr{E}[D_{A=i}]\right)$$

In entropy-based methods, the partition is carried out on the attribute that maximizes the information gain. Another interpretation of $Gain(D, A)$ is that it is a measure of the amount of

'information' we get about the target column if we partition on attribute A. $Gain(D, A)$ is also the number of bits saved by knowing the value of the attribute A when encoding the target value of an arbitrary data point in D.

Let's examine our example in Figure 7 using entropy and information gain. For the example $\mathscr{E}[D] = -0.31(\log_2 0.31) - 0.69(\log_2 0.69) = 0.8932$.

1. **Split on gender**: the two partitions will have 10/500 and 300/500 as the probabilities of finding a football player respectively. Each partition is half the total population. Therefore

$$\mathscr{E}[D_{Gen}] = -\frac{1}{2}\left(0.02(\log_2 0.02) + 0.98(\log_2 0.98) + 0.6(\log_2 0.6) + 0.4(\log_2 0.4)\right) = 0.5562$$

$$Gain(D, Gen) = \mathscr{E}[D] - \mathscr{E}[D_{Gen}] = 0.337$$

2. **Split on Age**: the two partitions will have 260/700 and 50/250 as the probabilities, and 700 and 300 as the sizes respectively, giving us an entropy for the partition as:

$$\begin{aligned}\mathscr{E}\left[D_{Age}\right] &= -0.7\left(\frac{13}{35}\left(\log_2 \frac{13}{35}\right) + \frac{22}{35}\left(\log_2 \frac{22}{35}\right)\right) - 0.3\left(0.2(\log_2 0.2) + 0.8(\log_2 0.8)\right) \\ &= 0.8828\end{aligned}$$

$$Gain(D, Age) = \mathscr{E}[D] - \mathscr{E}\left[D_{Age}\right] = 0.0104$$

Gender split gives a much higher information gain. So we must first split on gender.

An interesting property of Entropy as a homogeneity measure is that it can be computed in stages. Suppose our target variable can take one of c values — it is a c-class classification problem. It is natural to think of the classification task in stages — first distinguish class 1 from the rest, then distinguish 2 from among others that are not 1, and so on. There are broadly two ways one could construct the decision tree (i) by first looking at class 1 vs the rest (minimizing the entropy at each stage treating this as a binary classification problem), and then constructing the subtree for distinguishing class 2 from the remaining, and so on, (ii) trying to minimize the overall entropy (taking all the classes together) at every state. Ideally the two decision trees should not be very different. It turns out that Information Gain as a homogeneity measure indeed preserves this equivalence — in fact none of the other measures do. This is often termed the *multistage property*, entropy being the only measure that has the multistage property.

Multistage Property of Entropy

To see why the multistage property is true, we only need to verify that the entropy calculated with all the classes, is the sum of the entropy calculated for one class vs the rest and the entropy calculated on the rest weighted by the size. Let the dataset be denoted D, the entropy of D computed as class 1-vs-rest as $\mathscr{E}_1[D]$, and the subset of D with data points having class label other than 1 as $D_{\neq 1}$. Also let the set of data points in D with class label i be denoted $D_{=i}$. Formally we need to show that

$$\mathscr{E}[D] = \mathscr{E}_1[D] + \left(\frac{|D_{\neq 1}|}{|D|}\right)\mathscr{E}\left[D_{\neq 1}\right]$$

The proof follows. Note that p_i (probability of finding class i) for D is $\frac{|D_{=i}|}{|D|}$.

$$\begin{aligned}
\mathscr{E}_1[D] &= -\frac{|D_{=1}|}{|D|}\log_2\left(\frac{|D_{=1}|}{|D|}\right) - \frac{|D_{\neq 1}|}{|D|}\log_2\left(\frac{|D_{\neq 1}|}{|D|}\right) \\
&= -\frac{|D_{=1}|}{|D|}\log_2\left(\frac{|D_{=1}|}{|D|}\right) - \left(\frac{|D_{\neq 1}|}{|D|}\log_2\left(\frac{1}{|D|}\right)\right) - \frac{|D_{\neq 1}|}{|D|}\log_2(|D_{\neq 1}|) \\
&= -\frac{|D_{=1}|}{|D|}\log_2\left(\frac{|D_{=1}|}{|D|}\right) - \left(\sum_{i=2}^{c}\frac{|D_{=i}|}{|D|}\log_2\left(\frac{1}{|D|}\right)\right) - \frac{|D_{\neq 1}|}{|D|}\log_2(|D_{\neq 1}|) \\
\mathscr{E}\left[D_{\neq 1}\right] &= -\sum_{i=2}^{c}\frac{|D_{=i}|}{|D_{\neq 1}|}\log_2\left(\frac{|D_{=i}|}{|D_{\neq 1}|}\right) \\
\left(\frac{|D_{\neq 1}|}{|D|}\right)\mathscr{E}\left[D_{\neq 1}\right] &= -\sum_{i=2}^{c}\frac{|D_{=i}|}{|D|}\log_2\left(\frac{|D_{=i}|}{|D_{\neq 1}|}\right) = -\sum_{i=2}^{c}\frac{|D_{=i}|}{|D|}\log_2\left(|D_{=i}|\right) + \sum_{i=2}^{c}\frac{|D_{=i}|}{|D|}\log_2\left(|D_{\neq 1}|\right) \\
&= -\sum_{i=2}^{c}\frac{|D_{=i}|}{|D|}\log_2\left(|D_{=i}|\right) + \frac{|D_{\neq 1}|}{|D|}\log_2\left(|D_{\neq 1}|\right)
\end{aligned}$$

It is now easy to verify that

$$\mathscr{E}_1[D] + \left(\frac{|D_{\neq 1}|}{|D|}\right)\mathscr{E}\left[D_{\neq 1}\right] = -\sum_{i=1}^{c}\frac{|D_{=i}|}{|D|}\log_2\left(\frac{|D_{=i}|}{|D|}\right) = \mathscr{E}[D]$$

One issue with Information Gain as a homogeneity measure is that it tends to favour attributes that have a large number of distinct values. Most datasets have an `ID` attribute that is unique to every data point. Splitting on the `ID` attribute will trivially reduce every partition to a single datapoint and the entropy of the partition is clearly 0. The issue in general is that splitting a dataset on an attribute that has a large number distinct values will split the dataset into too many tiny partitions. As the partition becomes smaller, the likelihood of it being very disorderly reduces. Hence it will naturally result in a partition with a low entropy. However this is a sign of overfitting not a split that will result in a good generalizable rule. To address this issue we use the notion of *Intrinsic Information* in an attribute — this measures "the information we obtain about a data point when the value of the attribute is specified". Notice that the intrinsic information of an `ID` attribute is very high since giving the `ID` tells us everything about the data point — recall that the `ID` is a unique identifier for every data point. So higher the intrinsic information, less useful it is likely to be as a splitting attribute. Intrinsic Information of a k-valued attribute A for a dataset D is defined as

$$\mathscr{I}(D, A) = -\sum_{i=1}^{k}\left(\frac{|D_{A=i}|}{|D|}\right)\log_2\left(\frac{|D_{A=i}|}{|D|}\right)$$

Notice the similarity of this definition with that of entropy — entropy (for decision tree construction) is defined over the target attribute values. A few special cases can be useful to gain intuition about information gain. $\mathscr{I}(D, A) = 0$ whenever $|D_{A=i}| = |D|$ for some i — this means all the data points have the same value i for attribute A. Clearly splitting on A in this case does tell us anything more about the dataset than what we knew already. If $|D_{A=i}| = 0$ for some i (there are no data points with $A = i$) then that term becomes 0 in the summation, and hence can be ignored. Note that $0\log 0$ is taken as 0 — this is easily proved using limits. $\mathscr{I}(D, A)$ is maximum when $|D_{A=i}| = 1$ for all $1 \leq i \leq k$ (incidentally this means $k = |D|$) — all partitions

over the values of attribute A are singletons and $\mathscr{I}(D,A) = \log_2 k$. This definition of $\mathscr{I}(D,A)$ is therefore consistent with the intuitive description of Intrinsic Information we gave earlier.

To avoid the bias of Information Gain towards attributes with a large number of distinct values, we use a modified measure called *Gain Ratio* defined as

$$GR(D,A) = \frac{Gain(D,A)}{\mathscr{I}(D,A)}$$

This measure would 'penalize' attributes with high intrinsic information. To compute the gain ratio for the example in Figure 7

$$\begin{aligned}
\mathscr{I}(D,Gen) &= 1 \quad \text{(Gender splits } D \text{ into exactly two halves)} \\
Gain(D,Gen) &= 0.337 \quad \text{(from our previous calculation)} \\
GR(D,Gen) &= 0.337 \\
\mathscr{I}(D,Age) &= -0.7\log_2 0.7 - 0.3\log_2 0.3 = 0.8813 \\
Gain(D,Age) &= 0.0104 \quad \text{(from our previous calculation)} \\
GR(D,Age) &= 0.0104/0.8813 = 0.0118
\end{aligned}$$

4.5 Variance of the Target Variable

If the target attribute is continuous — regression problem — then *variance* of the target attribute is used as a measure of homogeneity. Lower the variance, more homogeneous the data is. The target variable, say X say takes values X_i for the i^{th} data point in the dataset D. The mean value of the variable X is $\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n}$. The variance is then given by

$$Var[D] = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2$$

If a split partitions the dataset D with n data points into D_1 and D_2 having n_1 and n_2 data points respectively ($n_1 + n_2 = n$), then the variance after the split is given by the weighted average of the variances of the partitions

$$\frac{n_1}{n}Var[D_1] + \frac{n_2}{n}Var[D_2]$$

We therefore try to find a split that minimizes the variance after the split. Notice that in the example in Figure 4, choosing a split at say 1000 or 2000 reduces the variance of one of the partitions to just the variance of the noise. Any other split will increase the variance after the split considerably.

5 Truncation and Pruning of Decision Trees

We have seen earlier that decision trees have a strong tendency to overfit the data. So practical uses of the decision tree must necessarily incorporate some 'regularization' measures to ensure the decision tree built does not become more complex than is necessary and starts to overfit. There are broadly two ways of regularization on decision trees:

1. Truncate the decision tree during the training (growing) process preventing it from degenerating into one with one leaf for every data point in the training dataset. One or more stopping criteria are used to decide if the decision tree needs to be grown further.

2. Let the tree grow to any complexity. However add a post-processing step in which we prune the tree in a bottom-up fashion starting from the leaves. It is more common to use pruning strategies to avoid overfitting in practical implementations.

We describe some popular stopping criteria and pruning strategies in the following subsections.

5.1 Decision Tree Stopping Criteria (Truncation)

There are several ways to truncate decision trees before they start to overfit.

1. **Minimum Size of the Partition for a Split**: Stop partitioning further when the current partition is small enough.

2. **Minimum Change in Homogeneity Measure**: Do not partition further when even the best split causes an insignificant change in the purity measure (difference between the current purity and the purity of the partitions created by the split).

3. **Limit on Tree Depth**: If the current node is farther away from the root than a threshold, then stop partitioning further.

4. **Minimum Size of the Partition at a Leaf**: If any of partitions from a split has fewer than this threshold minimum, then do not consider the split. Notice the subtle difference between this condition and the minimum size required for a split.

5. **Maxmimum number of leaves in the Tree**: If the current number of the bottom-most nodes in the tree exceeds this limit then stop partitioning.

Most of these can be set as part of the `rpart.control` in the call to the `rpart` function in the `rpart` package of R. Please see the documentation of `rpart` at `https://cran.r-project.org/web/packages/rpart/rpart.pdf`. See an example of the use of `rpart.control` in the code used to generate Figure 1. A serious disadvantage of the truncation approach to controlling overfitting is that it is often hard to set the appropriate thresholds and limits of the kind described above. It takes a lot of trial-and-error to get the correct combination of these parameters for optimal performance.

5.1.1 χ^2 (Chi-Square) Stopping Criterion

The χ^2 test checks if there is a statistically significant correlation between a potential explanatory attribute and the target attribute. Correlation indicates if knowing specific values or ranges of values for the explanatory attribute will tell us something significant about the target variable. If the explanatory attribute and the target attribute are completely uncorrelated, then

clearly splitting on that explanatory attribute is hardly of any use. So the χ^2 test is used to decide if we need to split on an attribute or not — often as a filter that considers only features that pass the χ^2 test for splitting on, before picking the best one based on some homogeneity measure.

The idea behind the χ^2 test is that if the splitting attribute and the target attribute are uncorrelated then the distribution of target labels before the split and the distribution after the split must be more-or-less identical. Consider a data set D with n points in which j data points belong to class j, assuming a c-valued class/target variable. Refer Figure 8. Clearly $n = \sum_{j=1}^{c} n_j$. Suppose we split on a k-valued attribute A — let the partitions obtained have m_i data points in the partition $D_{A=i}$ that has value i for the attribute A.

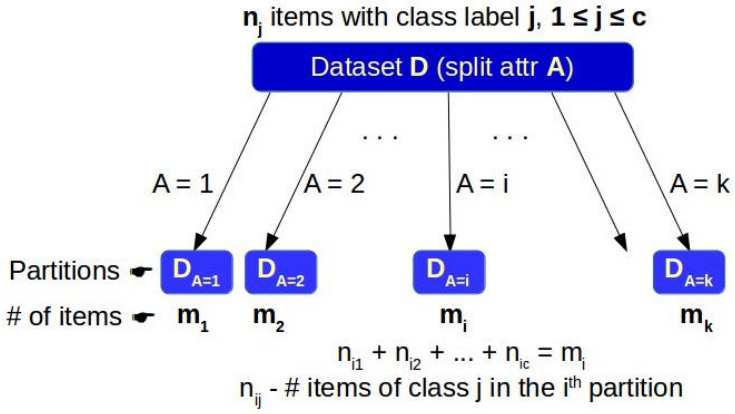

Figure 8: χ^2 Stopping Criterion

Also let the number of items with class label j in $D_{A=i}$ be denoted as n_{ij}. Note that

$$|D_{A=i}| = m_i = \sum_{j=1}^{c} n_{ij}$$

The fraction of data points in D with label j is n_j/n. If the attribute A is uncorrelated with the target then we would expect (more-or-less) the same fraction of data items in each of the $D_{A=i}$ partitions to have class label j. Similar observation must hold for each class label j. Therefore if the attribute A and the target are uncorrelated we expect roughly $e_{ij} = \left(\frac{n_j}{n}\right) m_i$ items in $D_{A=i}$ to have class label j. To measure the strength of the correlation between attribute A and the target attribute, we see how different are the n_{ij}s (the actual numbers of items) from the expected values e_{ij}. The correlation is therefore measured as

$$C = \sum_{i=1}^{k} \sum_{j=1}^{c} \frac{(n_{ij} - e_{ij})^2}{e_{ij}}$$

Clearly lower the value of C more uncorrelated the attribute A is with the target attribute. Using elementary statistics (Central Limit Theorem) it is possible to show that the random variable C must have a χ^2 distribution with $(k-1)(c-1)$ degrees of freedom. If A is a binary attribute and it is a binary classification problem ($k = c = 2$) then C is $\chi^2(1)$ — one degree of freedom.

Similar what we did in Time-Series analysis to test if a series is white noise, we check, under the null hypothesis that "the attribute A is uncorrelated with the target attribute", the probability of *encountering a statistic at least as extreme as C which was observed.* We therefore estimate the probability $Pr(X \geq C)$, given that X follows a χ^2 distribution with the appropriate degrees of freedom. If this happens to be less than the significance level (commonly 0.05), we reject the null hypothesis. For the case $k = c = 2$ we reject the null hypothesis if $C > 3.8415$ — in other words we cannot believe anymore that we could have got such a C value even 'by chance'.

Applying the χ^2 test to the example in Figure 7, we first observe that $n_P = 310, n_N = 690$.

1. **Split on gender**: $n_{FP} = 10$, $n_{FN} = 490$, $n_{MP} = 300$ and $n_{MN} = 200$. Each partition is half the total population. Therefore $e_{FP} = e_{MP} = 155$ and $e_{FN} = e_{MN} = 345$.

$$C = \frac{(10-155)^2}{155} + \frac{(490-345)^2}{345} + \frac{(300-155)^2}{155} + \frac{(200-345)^2}{345} = 393.1744 > 3.8415$$

2. **Split on Age**: $n_{>50,P} = 50$, $n_{>50,N} = 250$, $n_{<50,P} = 260$ and $n_{<50,N} = 440$. The two partitions are of sizes $m_{>50} = 300$ and $m_{<50} = 700$. Therefore $e_{>50,P} = 0.7 * 310 = 217$, $e_{>50,N} = 0.7 * 690 = 483$, $e_{<50,P} = 0.3 * 310 = 93$ and $e_{<50,N} = 0.3 * 690 = 207$.

$$C = \frac{(260-217)^2}{217} + \frac{(440-483)^2}{483} + \frac{(50-93)^2 93}{+} \frac{(250-207)^2}{207} = 41.1630 > 3.8415$$

   We have enough evidence that both the attributes, Age and Gender, are not uncorrelated with the target attribute — Play or not-Play football. So both Age and Gender ar eligible attributes to split on.

5.2 Decision Tree (Post)-Pruning

One popular approach to pruning is to use a validation set — a set of labelled data points, typically kept aside from the original training dataset. This method called *reduced-error* pruning, considers every one of the test (non-leaf) nodes for pruning. Pruning a node means removing the entire subtree below the node, making it a leaf, and assigning the majority class (or the average of the values in case it is regression) among the training data points that pass through that node. A node in the tree is pruned only if the decision tree obtained after the pruning has an accuracy that is no worse on the validation dataset than the tree prior to pruning. This ensures that parts of the tree that were added due to accidental irregularities in the data are removed, as these irregularities are not likely to repeat. The pruning is carried out iteratively as summarized in Algorithm 2.

5.3 Rule Post-Pruning

This is used in one of the most popular decision tree algorithms (currently C5.0 — its earlier avatars were ID3 and C4.5). The idea is to convert the fully grown tree into a collection of rules. We have already discussed the correspondence between rules and decision trees earlier in this document. There are a couple of reasons why working with rules instead of the decision tree during pruning may be better.

Algorithm 2 Reduced-Error Pruning of a Decision Tree

```
1: procedure PRUNEDECISIONTREE(validation dataset V, fully grown decision tree T)
2:     ϵ = Accuracy(T, V)
3:     Terminate = False
4:     while not Terminate do
5:         ϵ* = ϵ; x* = NULL
6:         for all non-leaf nodes x in T do
7:             if α = Accuracy(T − Subtree(x), V) ≥ ϵ* then
8:                 ϵ* = α; x* = x
9:             end if
10:        end for
11:        if x* ≠ NULL then    ▷ Pruning at x* gives maximum increase in validation accuracy
12:            ϵ = ϵ*; T = T − Subtree(x*)
13:        else
14:            Terminate = True
15:        end if
16:    end while
17:    return T
18: end procedure
```

1. Using the rules allows taking pruning decisions on different occurrences of the same 'condition' (this corresponds to one node in the tree, but can reappear multiple times in different rules) independently depending on the context in which they are used. We are not constrained by the structure of the tree anymore.

2. Rules are more readable than the tree itself.

Rule post-pruning works as follows:

1. The fully grown decision tree is converted into a collection of rules — each rule corresponding to the path from the root to a leaf.

2. Prune any of the conditions in any of the rules if it does not result in the validation performance worsening.

3. Sort the pruned rules in the decreasing order of their estimated accuracy. Apply the rules in this order when taking decision on a new test instance.

For the heart example we constructed the fully grown tree in Figure 5. We can now try pruning it to get back a generalizable model. Assuming the variable `hearttree` contains the fully grown tree, we can prune it by simply calling the `prune` function as shown in the code snippet below. The pruned tree is shown in Figure 9.

```
hp <- prune(hearttree, cp=0.03)
plot(hp)
text(hp)
```

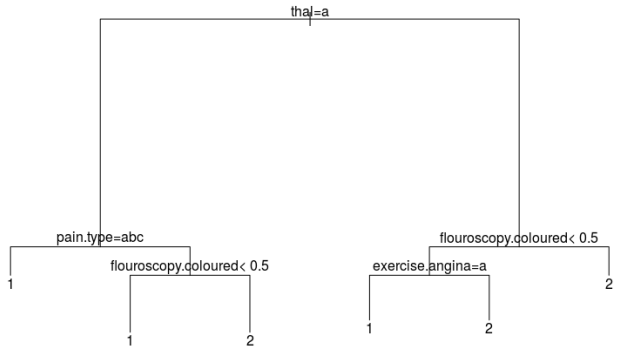

Figure 9: Pruned Tree for the Example in Figure 5

5.4 Other Implementation Specifics

We wrap up this module with a short discussion of some specifics regarding the implementation of decision tree algorithms.

5.4.1 Continuous Valued Attributes

Some special care needs to be taken to handle continuous valued attributes. There are two broad ways in which continuous valued attributes are evaluated for possible splits.

1. Discretize the values into a fixed set of 'buckets' (intervals) and treat it like a nominal (categorical) attribute.
2. The second is a more computationally costly but more accurate way of evaluating splits on a continuous valued attribute. Suppose the data set is sorted on the attribute. Consider all possible binary splits at midpoints between the attribute values of consecutive data points in the sorted order. Evaluate the information gain (or equivalent measure) at each split and pick the best. Treat the gain from this best split as the gain from the attribute.

Also at the leaves, instead of assigning the average of the target attribute values of the training samples at a leaf, we could fit a linear model among those data points and store the model at the leaf. For any test point, the prediction will use the linear model at the leaf at which the test point ends up. These are also referred to as *Model Trees*, *Regression Trees* being the ones which assign the average value at a leaf.

5.4.2 Handling Attributes with Different Costs

In a health care scenario for example, a decision tree for diagnosis may involve tests of different kinds — temperature, biopsy, blood tests, CAT scan, etc. If the decision tree has say CAT scan

high up in the tree, then using the tree would imply that most decisions would require the patient to undergo a CAT scan. This is clearly an unrealistic ask, even though a CAT scan may be a far better indicator for accurate diagnosis. This is simply because the 'cost' (monetary, accessibility, discomfort, time involved, ...) is too high to warrant the test unless absolutely necessary, as a last resort. So in such situations it may be unrealistic to use just information gain or some such homogeneity measure as the sole parameter on which to split the dataset next, during training. One way to incorporate this is to explicitly penalize splitting on attributes that have high cost. Assuming the cost associated with the attribute A is $C(A)$, a modified information gain to take care of the cost could be

$$Gain(D,A) = \frac{(Gain(D,A))^2}{C(A)}$$

During training therefore at each stage, we try to find the attribute that maximizes this modified gain ratio.

5.4.3 Handling Missing Values

Missing values are to be expected in any realistic data. Fortunately it is relatively straightforward to handle missing values for decision tree learning. Missing values are a problem only when we have to split on an attribute which has missing values. In any case the core algorithm remains the same. In case we need to split on an attribute with missing values, we need to decide which subtree will a data point with missing value for this attribute belong to. The näive way to handle this will be to just replace the missing value by the average of the values of that attribute among the data points at that node and belonging to the same class. A more sophisticated way is to estimate the probability of the attribute values corresponding to the split (again from the data points at that node and belonging to the same class). Then the data point is split into fractional instances, one piece for each subtree under the current node. Homogeneity measures such as Information Gain are computed based on the sum of the weights and not just the counts. A similar strategy is adopted during prediction as well.

Most of the parameters and strategies described in this module are provided as control parameters in many of the popular implementations of decision trees in R and other environments. Refer to the manuals of some of these implementations for more details. References to these are given in the additional reading and references below.