MATH5743M: Statistical Learning - Lecture 7

Dr Seppo Virtanen, s.virtanen@leeds.ac.uk

Semester 2: 7 Mar 2022

Outline

Decision trees

- intuitive and interpretable sequence of decisions (i.e., a tree) based on inputs for classification
- multiple inputs feasible but complicate interpretability and representation

Example; Classify dog breed based on weight and age

Figure 1: A German Shepherd (GS) and a Jack Russel Terrier (JR). The typical weight of a German Shepherd is about 25kg, the typical weight of a Jack Russel is about 5kg

Example; A desicion tree

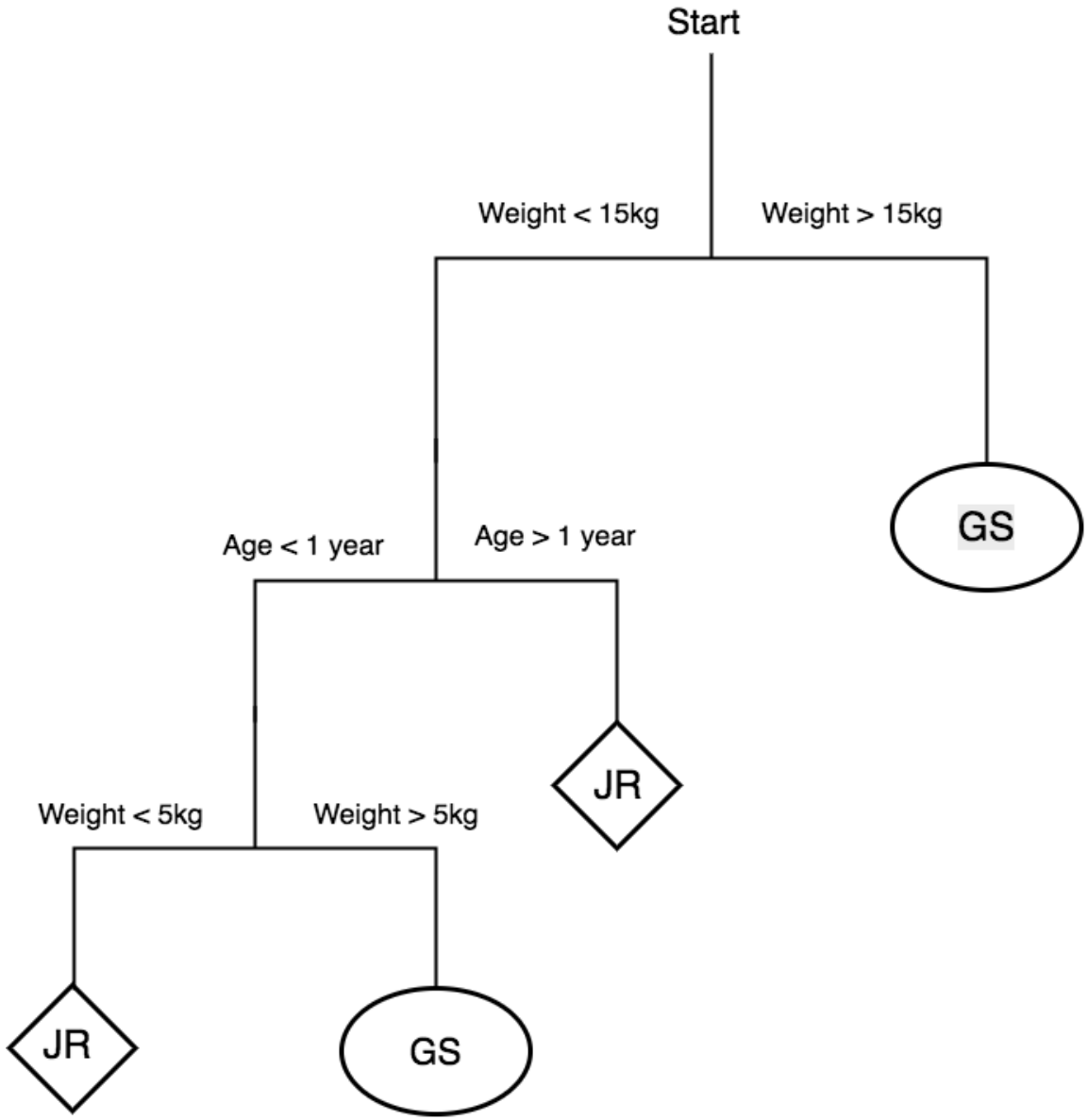


Figure 2: A decision tree representing the decision-making process for

How to train/learn a decision tree?

There are several possible algorithms for creating decision trees from data.

We are going to focus on the most commonly used method:

- a tree is grown by recursively creating one split after another, based on the values of input variables, so as to maximise the probability that we classify the output variables correctly.

Example

- We want to create a single split that best divides the data into two distinct species.
- For each split, imagine that you have n_A items of class A, and n_B of class B (with corresponding proportions $\rho_A = n_A/(n_A + n_B)$ and $\rho_B = 1 - \rho_A$.
- To calculate how good a split is, we introduce a quantity called the 'Gini Impurity' (G).

Gini impurity

- What is the probability that if you pick an item at random, and classify it as either A or B with probabilities corresponding to the proportions of each item, that you will make a mistake?
- We can calculate it like this: to make a mistake we must choose an item of class A (with probability ρ_A) and then misclassify it with probability ρ_B, or vice versa.

$$G = \rho_A \rho_B = \rho_A(1 - \rho_A) \tag{1}$$

- If the set is 'pure', i.e. all of one class, then $G = 0$. Alternatively if the set is half A and half B then $G = 0.25$, which is the maximum possible value.

Example

- We will try splitting the data according to the weight of the dog.
- To identify the best first split to make in our growing tree we need to find the value of 'weight' where, if we split the data there, we minimise the *average Gini Impurity* of the resulting two leaves.

Example

Lets imagine this data is as below

```
##    weight   age breed
## 1       4  4.00    JR
## 2       3  5.00    JR
## 3       5  0.25    GS
## 4       8  0.50    GS
## 5       8  6.00    JR
## 6      15  7.00    GS
## 7      20 10.00    GS
## 8      25  8.00    GS
## 9       5  7.00    JR
## 10     30  6.00    GS
## 11     12 10.00    JR
## 12     22  9.00    GS
```

Example

If we split the data at weight=16

```
##    weight   age breed
## 1       4  4.00    JR
## 2       3  5.00    JR
## 3       5  0.25    GS
## 4       8  0.50    GS
## 5       8  6.00    JR
## 6      15  7.00    GS
## 9       5  7.00    JR
## 11     12 10.00    JR
```

```
##    weight age breed
## 7      20  10    GS
## 8      25   8    GS
## 10     30   6    GS
## 12     22   9    GS
```

Example

Lets find the proportion of JR in group 1 and group 2 and calculate the Gini Impurity of each

```
p1 = mean(group1$breed == 'JR')
p2 = mean(group2$breed == 'JR')
G1 = p1*(1-p1)
G2 = p2*(1-p2)
```

Example

Finally record the (weighted) average Gini Impurity of the two groups. The weighting is important: we need to average the two impurities based on how big each new subset is:

```
G = (G1*dim(group1)[1]+G2*dim(group2)[1])/
  (dim(group1)[1]+dim(group2)[1])
print(G)
```

```
## [1] 0.15625
```

Example

What about we choose to split at weight=14? We can repeat the same procedure, with the new split value

```
idx = which(dog_data$weight < 14)
group1 = dog_data[idx, ]
group2 = dog_data[-idx, ]
p1 = mean(group1$breed == 'JR')
p2 = mean(group2$breed == 'JR')
G1 = p1*(1-p1)
G2 = p2*(1-p2)
G = (G1*dim(group1)[1]+G2*dim(group2)[1])/
  (dim(group1)[1]+dim(group2)[1])
print(G)
```

```
## [1] 0.1190476
```

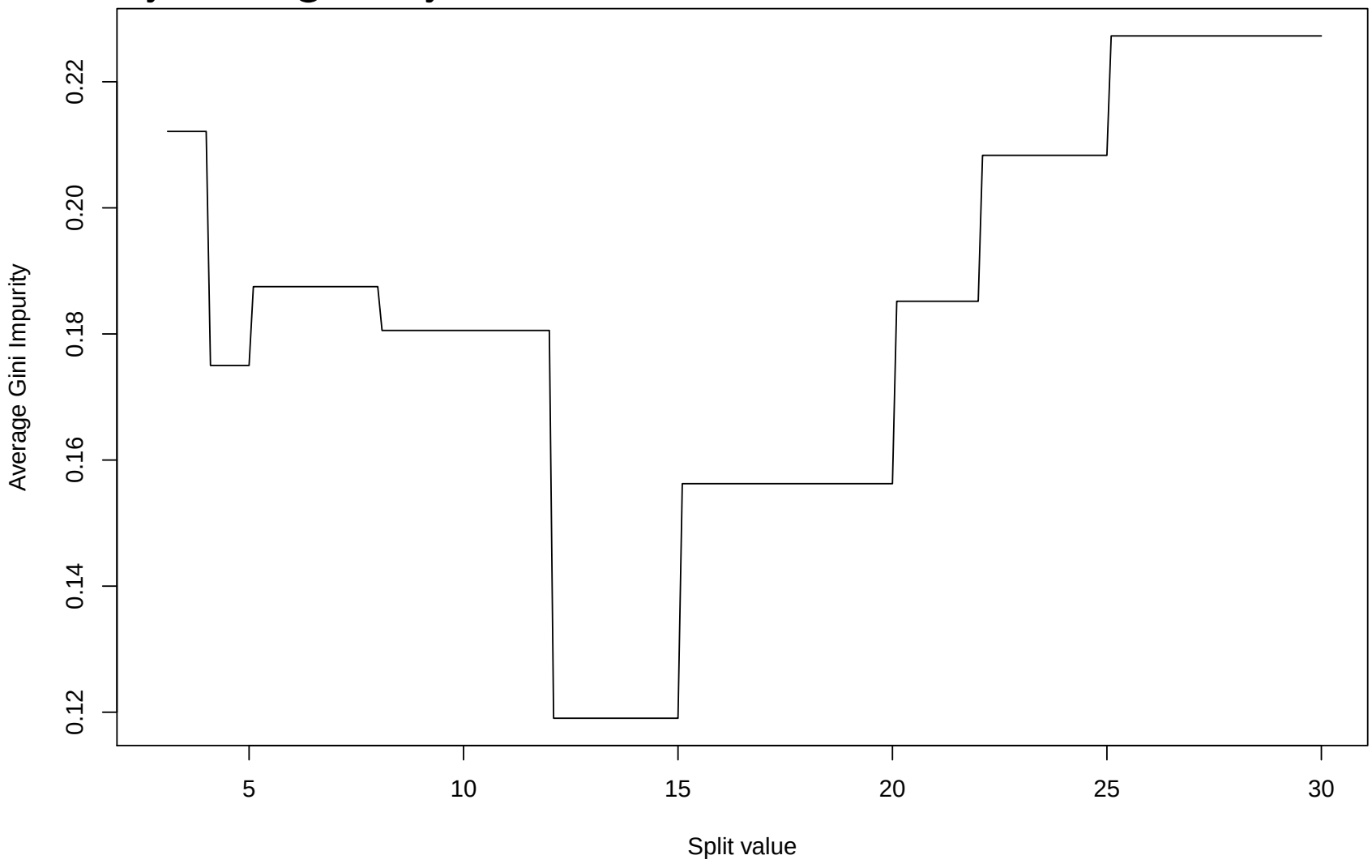
Example

Create a function that returns the average Gini Impurity. The function needs to know which input to use when splitting, and we will give it a default of 'weight':

```
average_G <- function(split_value, data, input='weight'){
  idx = which(data[, input] < split_value)
  group1 = data[idx, ]
  group2 = data[-idx, ]
  p1 = mean(group1$breed == 'JR')
  p2 = mean(group2$breed == 'JR')
  G1 = p1*(1-p1)
  G2 = p2*(1-p2)
  G = (G1*dim(group1)[1]+G2*dim(group2)[1])/
    (dim(group1)[1]+dim(group2)[1])
  return(G)
}
```

Example

Now we can map out the mean value of G as a function of the split value by testing many different values

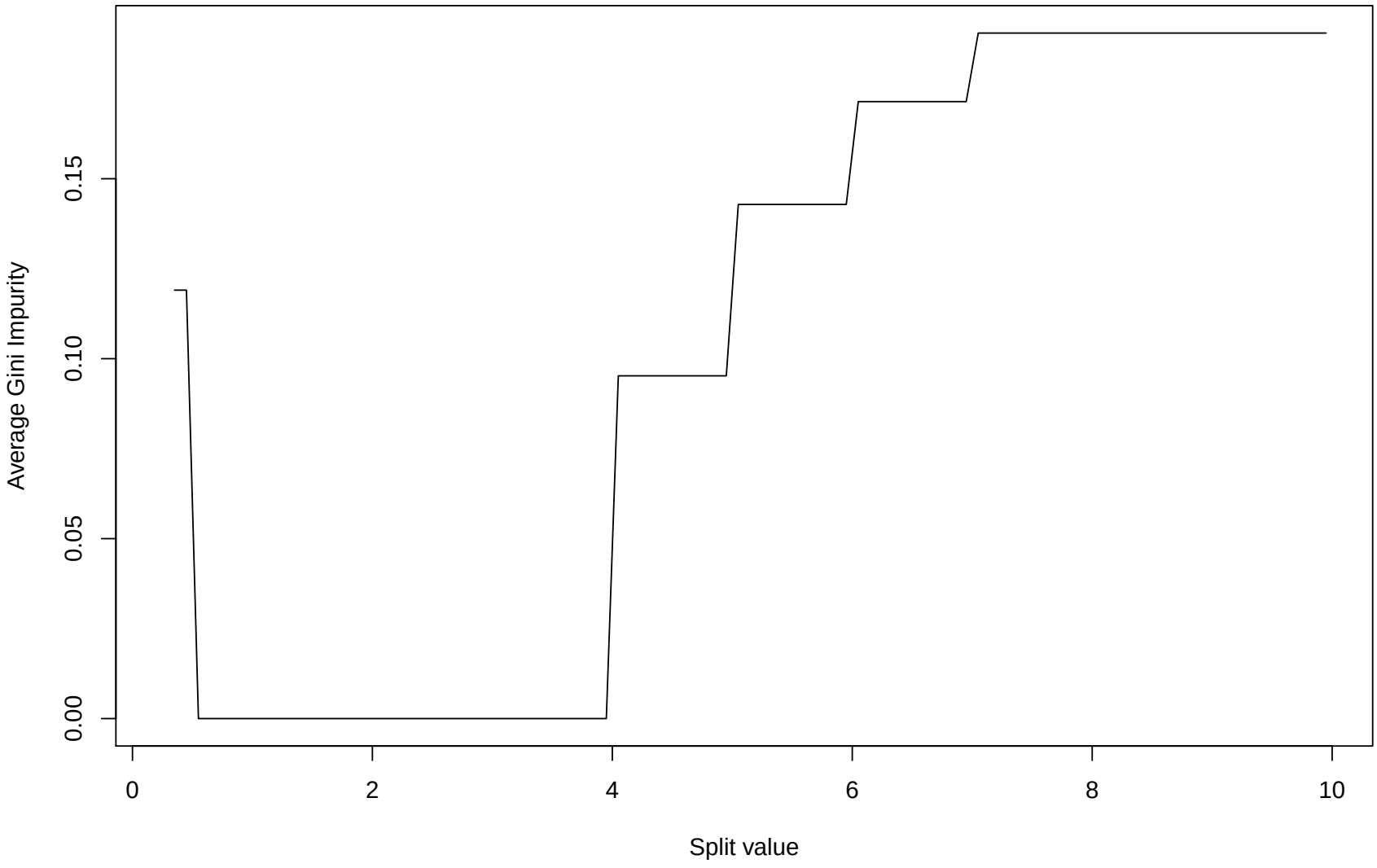


Example

A single split at weight=13, 14 or 15 gives a very good classification – one group contained only German Shepherds, the other only Jack Russels and two German Shepherd puppies.

Consider partitioning the group that remains mixed – the one containing both Jack Russels and German Shepherds. We repeat the procedure above on this subset, but rather than using weight, we will now try splitting the group according to age and only apply this to dogs below 15kg (group1).

Example



Example

- We can see from these results that any split between age $= 0.5$ and age $= 4$ gives an average Gini impurity of zero, i.e. a perfect split. We can choose any value in this range, so let's choose 2.25 years as the mid point.
- This means that our tree is now complete.
- First we ask whether the dog is more than 14kg: if so, we class it as a German Shepherd; if not we look at the dog's age.
- Those below 14kg and older than 2.25 years are Jack Russels, while those below 14kg but younger than 2.25 years are German Shepherds.

Decision tree

Decision trees are good at *partitioning* the input space into distinct areas that need not be monotonic, but they are less good at capturing gradually changing functions.

For example, lets look at a function that is neatly partitioned:

$$P(Y = 1) = \begin{cases} 1, & \text{if } 0.25 < X < 0.75 \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

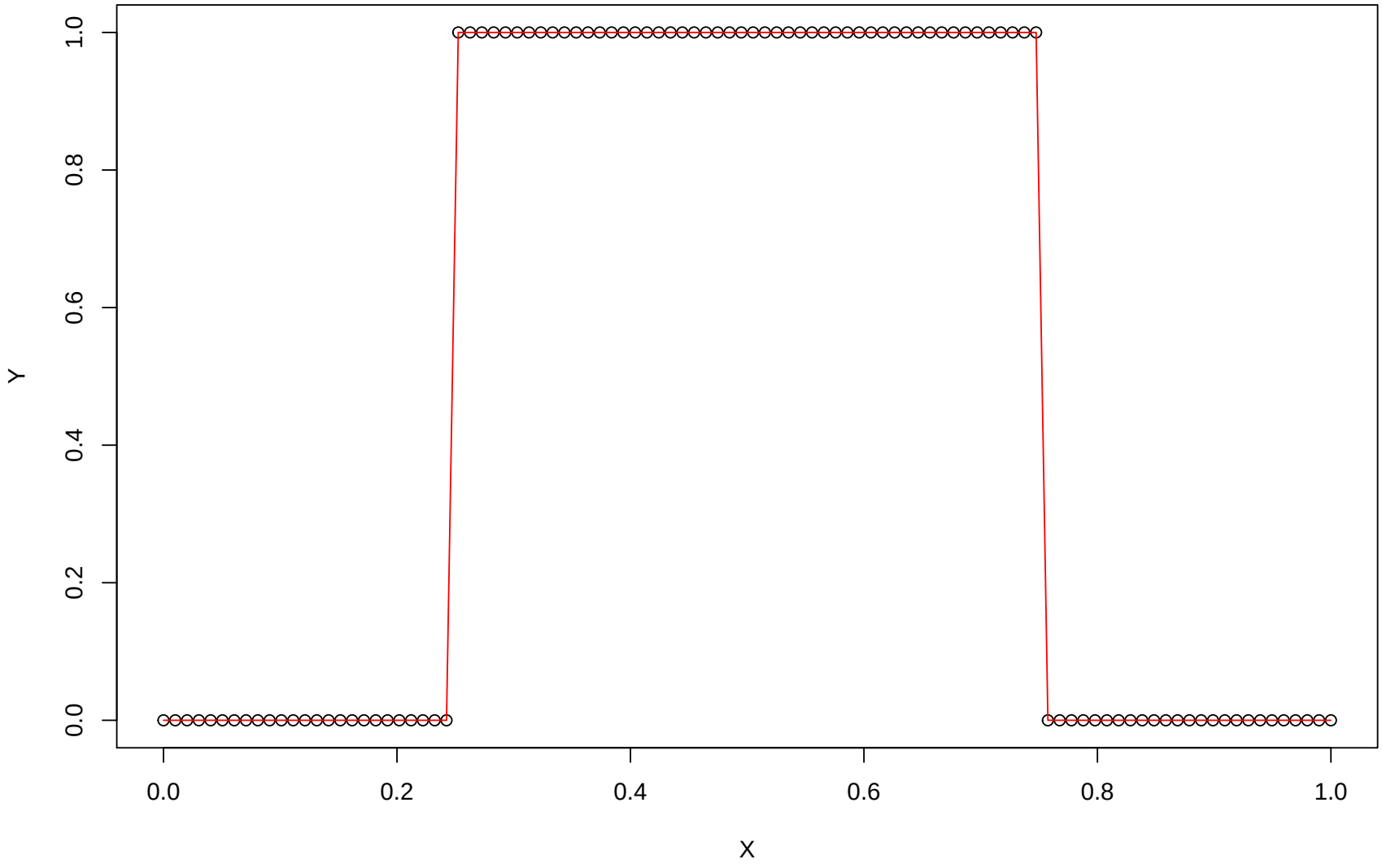
that is, Y is always equal to 1 when X is between 0.25 and 0.75, and zero otherwise: a 'top hat' function.

Example

```r
#define the probability function
myfunction <- function(x){0+1*(x>0.25 & x < 0.75)}

#Generate some data
X = seq(0,1, length.out=100)
P = myfunction(X)
Y = as.numeric(runif(100)<P)
mydata = data.frame(X=X, Y=Y)
```

Example



rpart in R

To fit a decision tree we need to use a package called **rpart**.

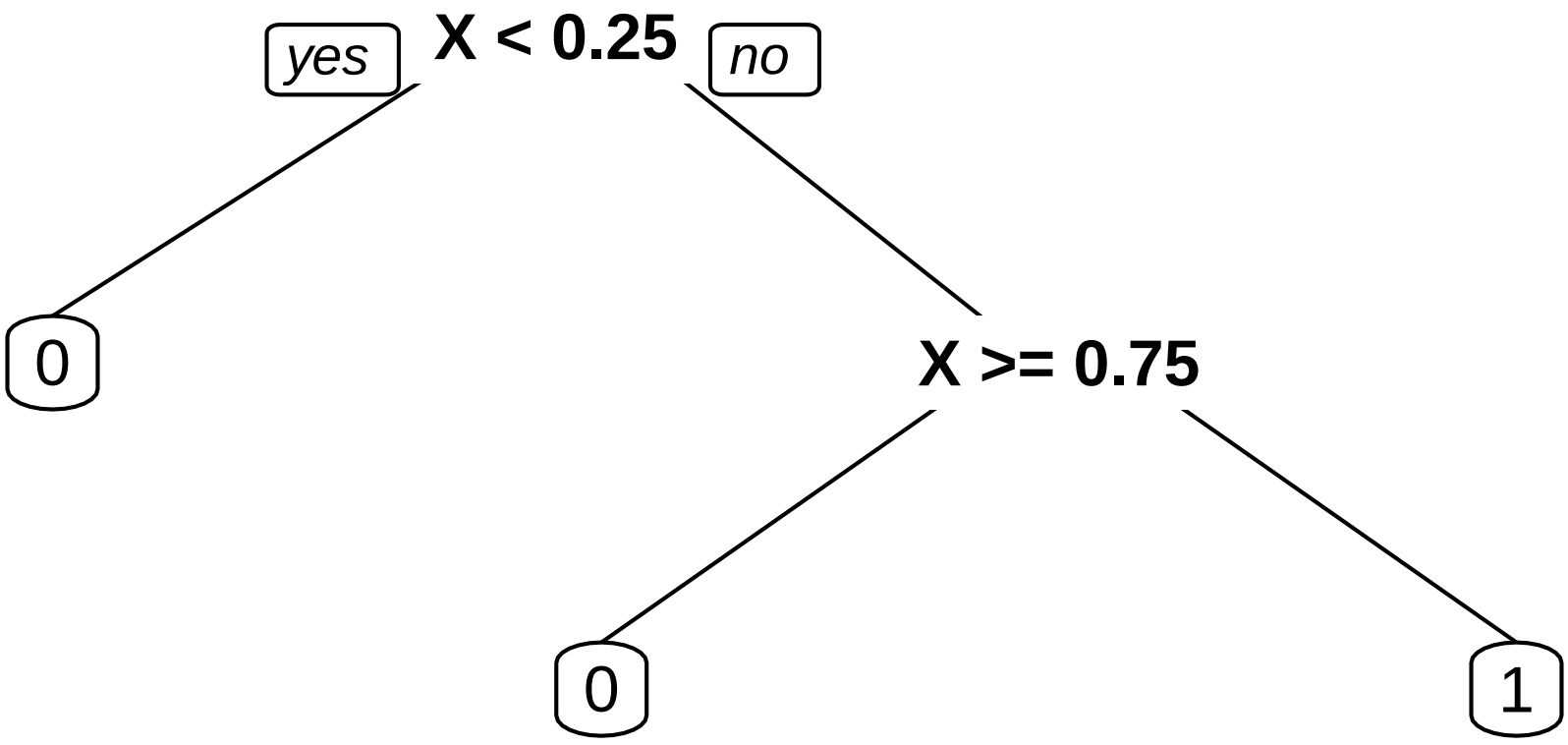
Similarly to our use of **glm**, we need to provide a *formula* that specifies which outputs and inputs to use from the supplied data, like this:

```
#Training a decision tree with formula Y ~ X using mydata
library(rpart)
mytree = rpart(Y ~ X, data=mydata, method='class')
```

rpart in R

Now visualise the tree using the function **prp**:

```
#Visualising a decision tree
library(rpart.plot)
prp(mytree)
```



What about more complex problems?

- The resulting tree may be a lot more complex than those we've seen previously – there are a lot more decision points. This is a natural consequence of the problem becoming harder, but it means that the model is increasingly difficult to interpret just by looking at the tree.
- Decision trees are often quite poor classifiers by themselves. In later lectures we will look at how they can be improved.
- Decision tree cannot easily model smooth variations in the probability, as it fundamentally works by partitioning the space.
- Decision tree models are very *unstable*. Slightly different data sets can produce very different trees.