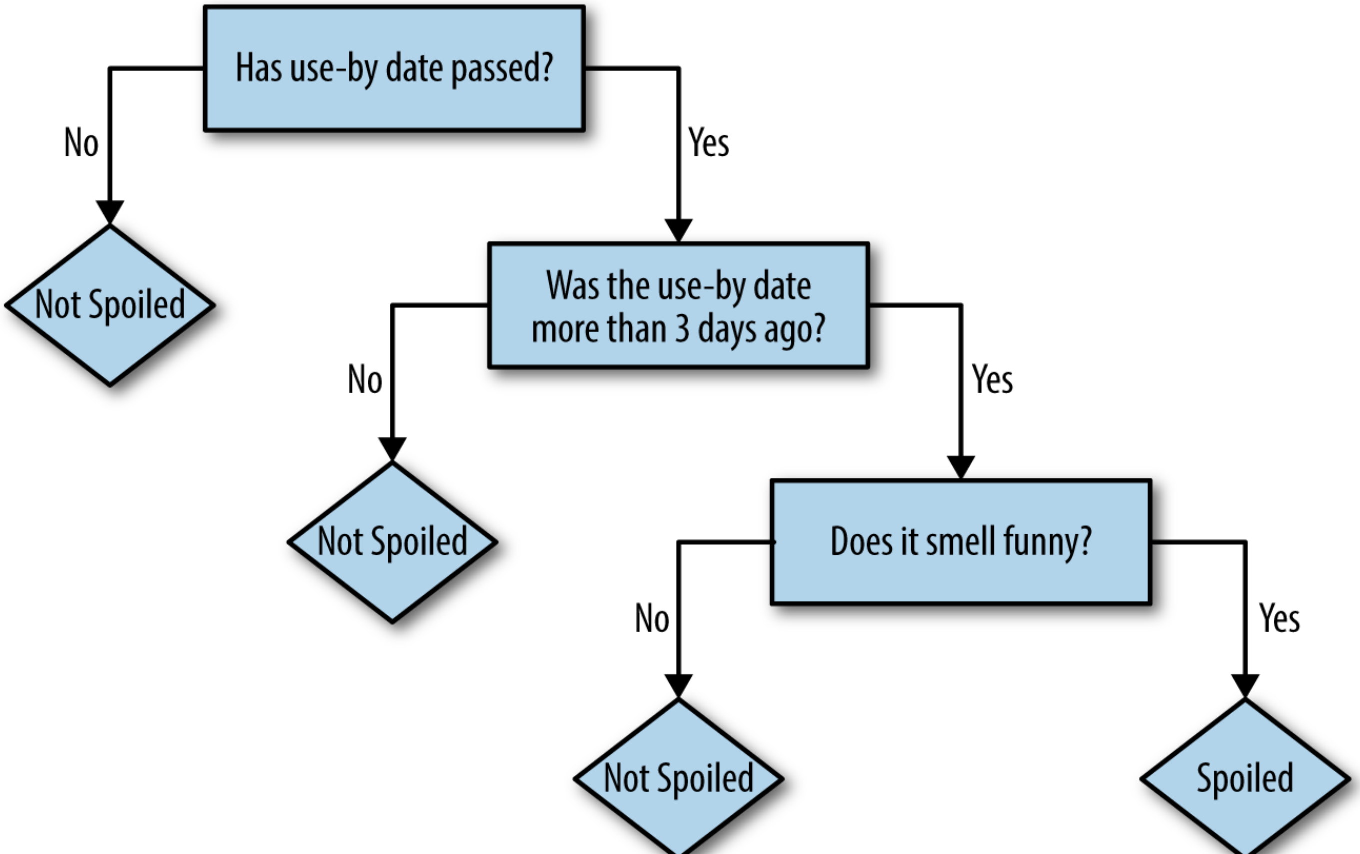
Decision Tree

Decision Trees

- Set of rules that helps us make a decision
- set of if then examples

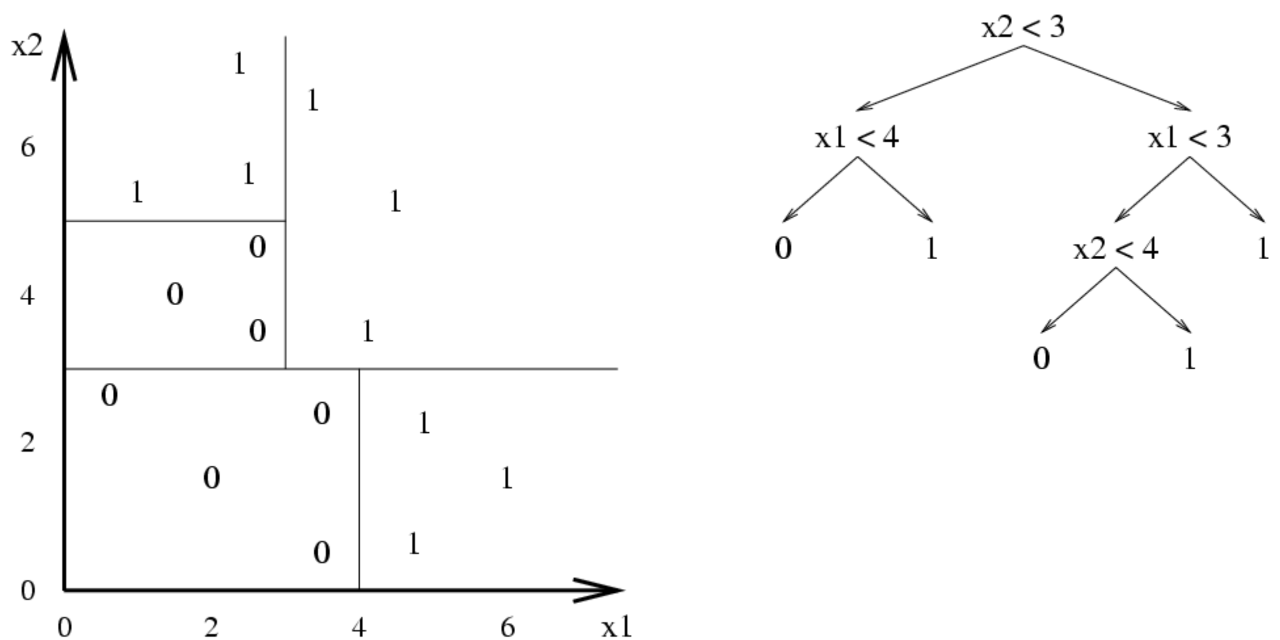


How it works

We partition data into homogeneous rectangles

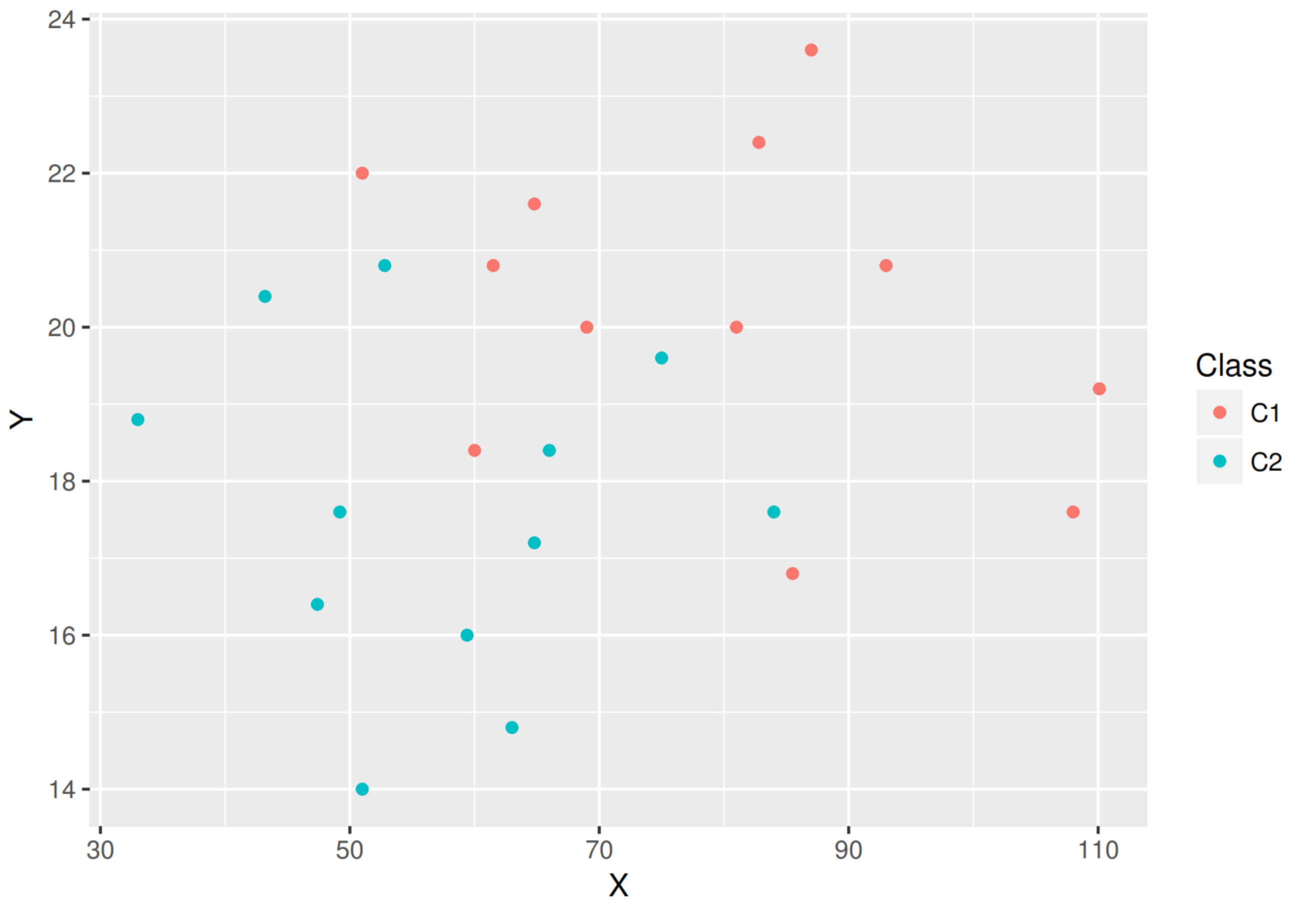
Decision Tree Decision Boundaries

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the K classes.



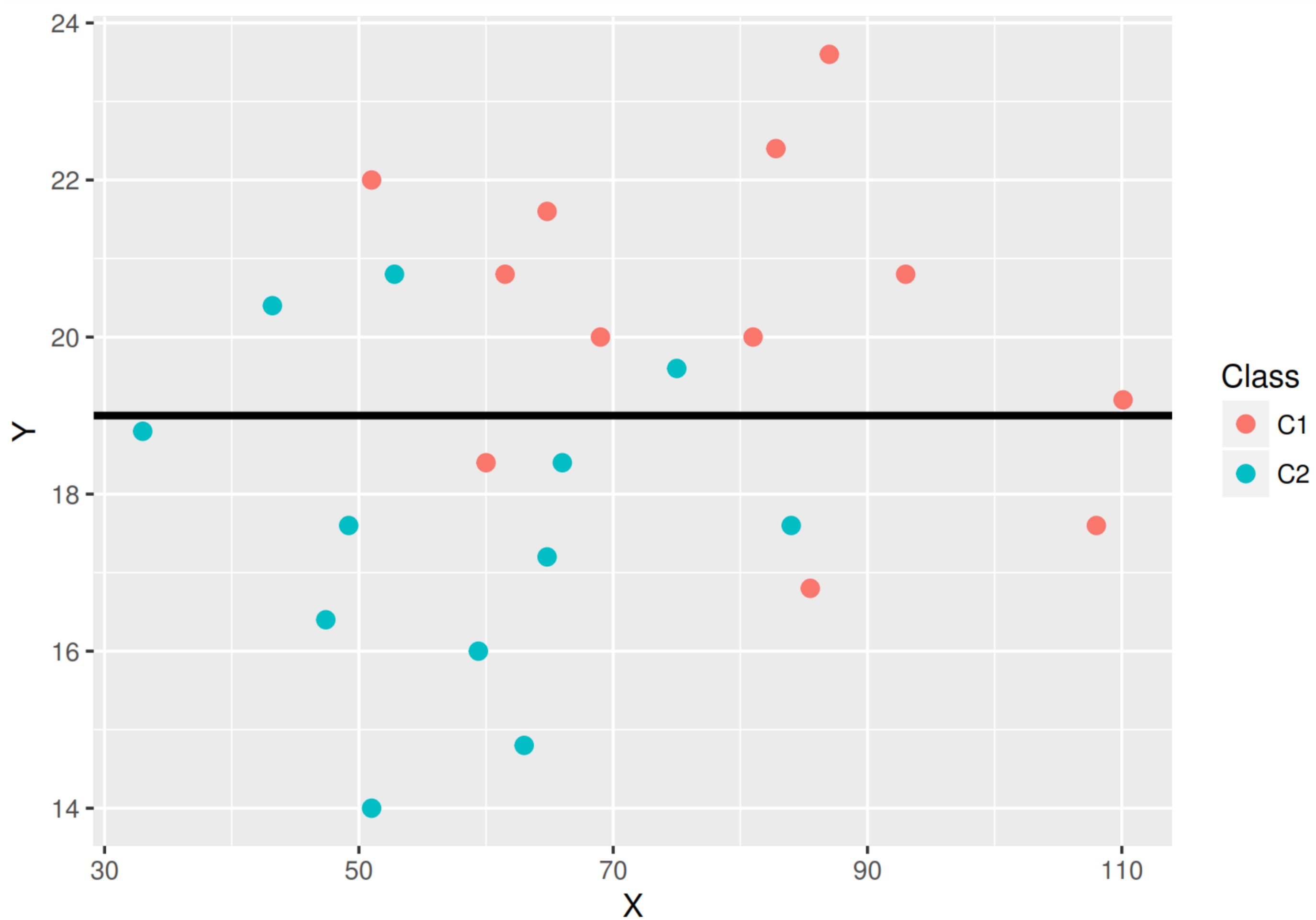
Example

Two variables: X and Y and variable Class – needs to be predicted



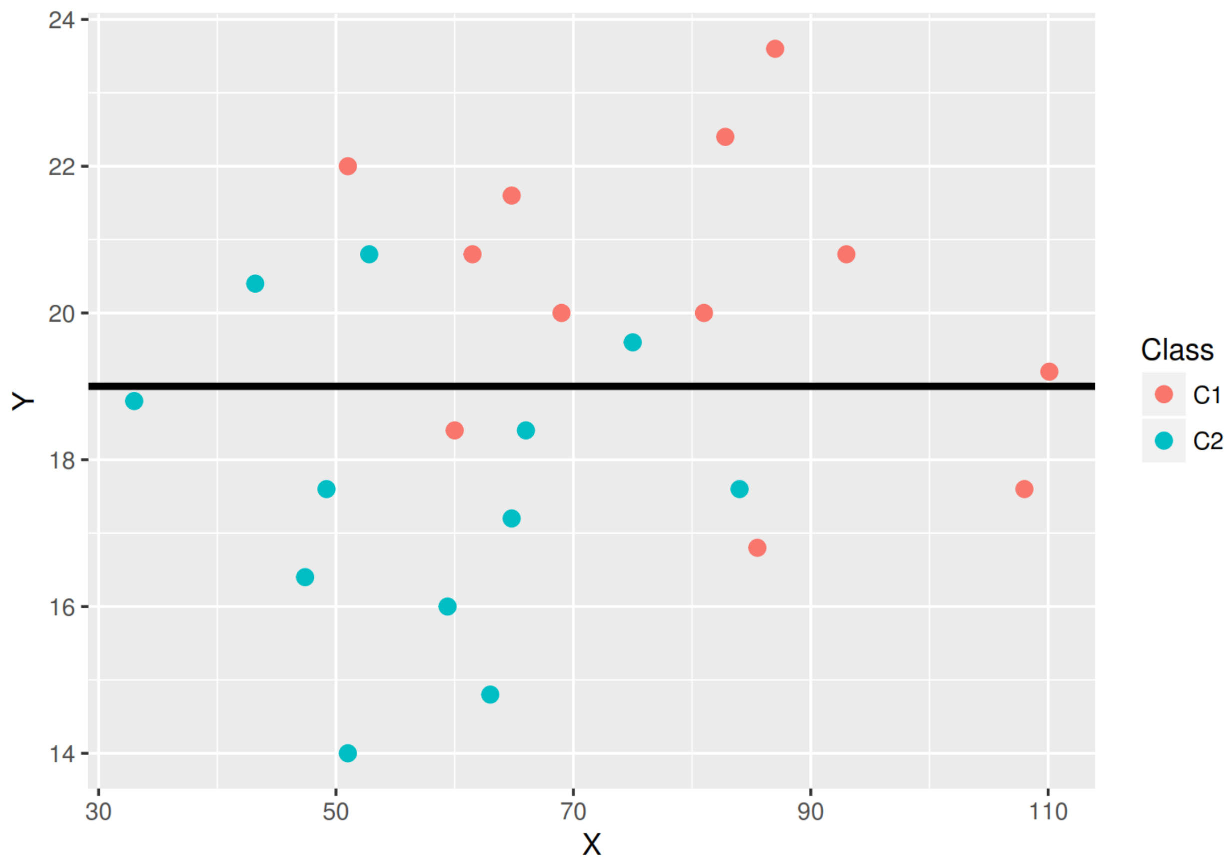
Example: Split 1

$Y \geq 19$ and $Y < 19$



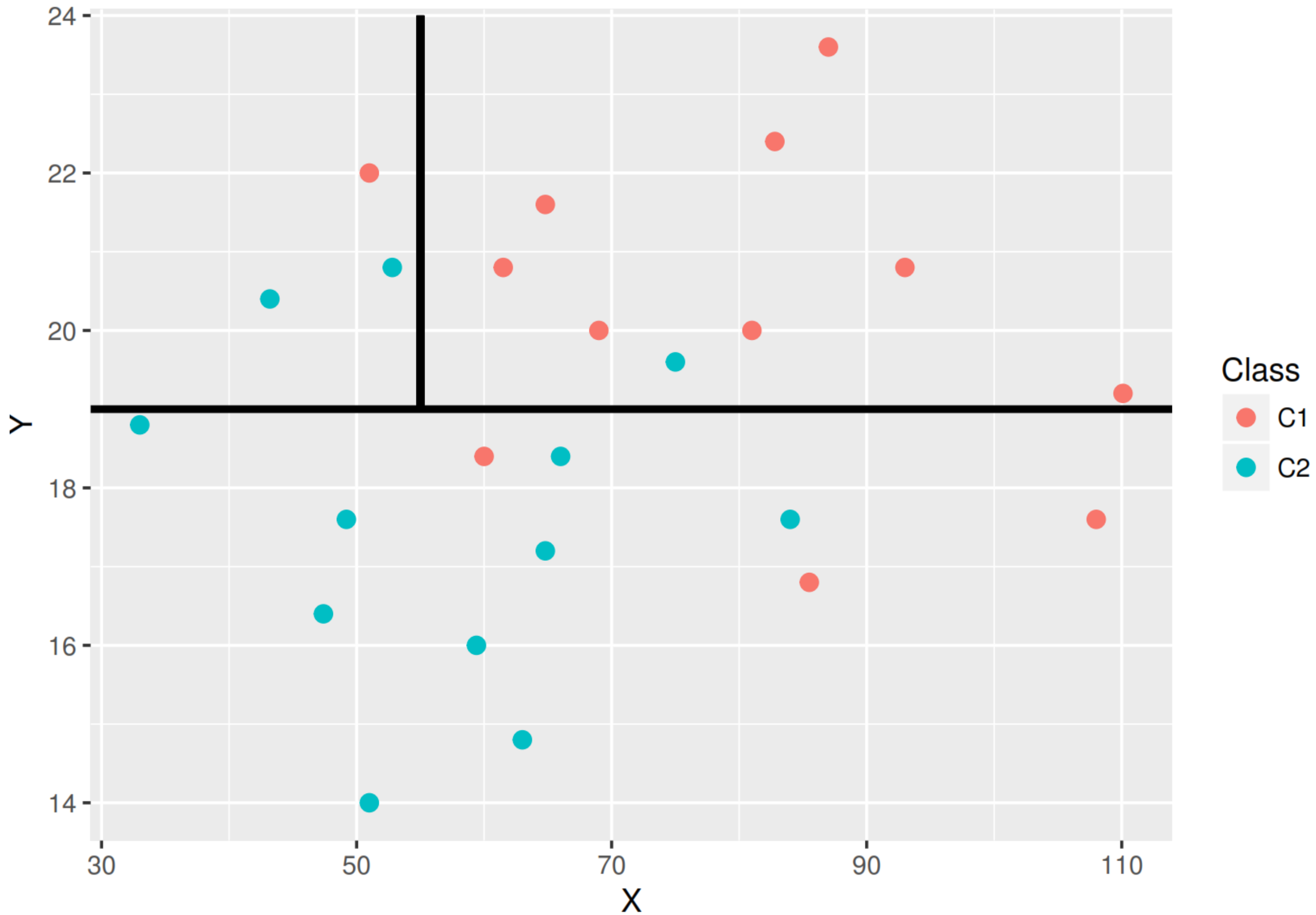
Example: Split N 1

Split N1: $Y \geq 19$ and $Y < 19$. Lets look at the prevailing class at each rectangle
IF $Y \geq 19$ THEN C1
IF $Y < 19$ THEN C2



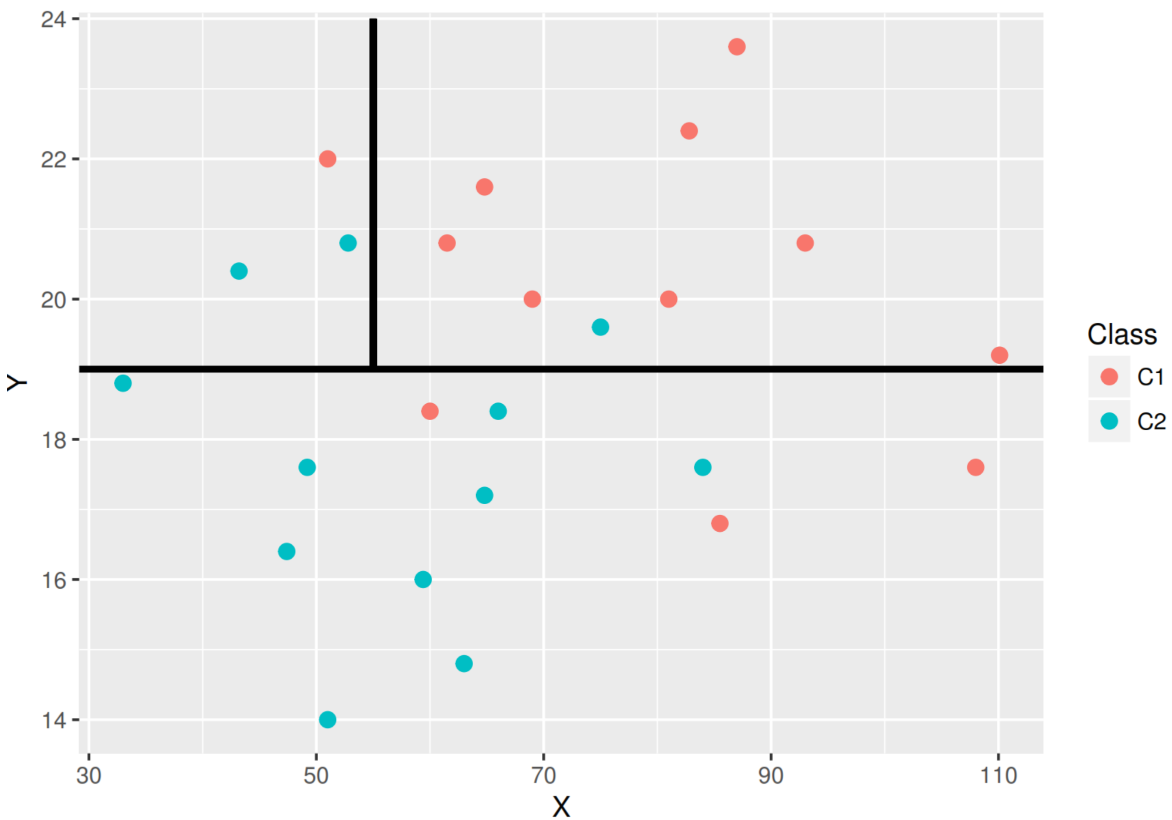
Split N 2

Split N2: $X \leq 55$ and $X > 55$



Split N 2

Split N2: $X \leq 55$ and $X > 55$



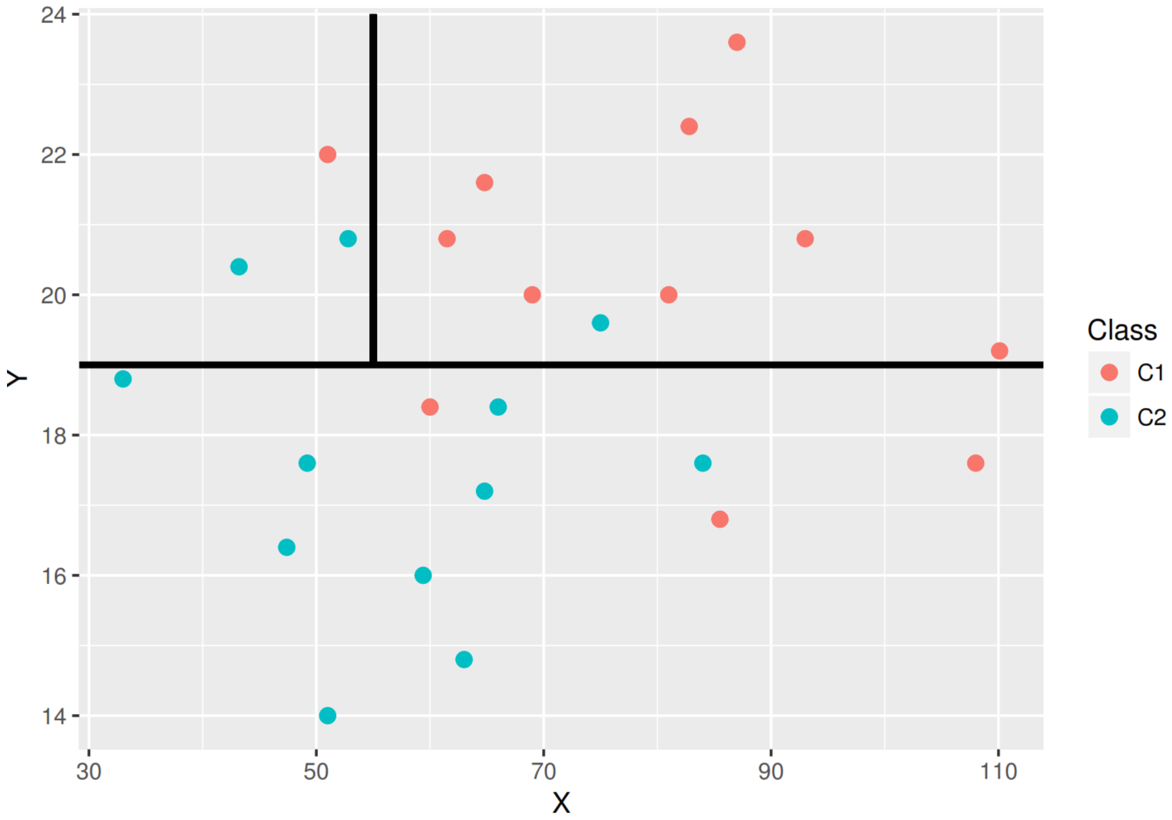
Split N 2

Classification rules:

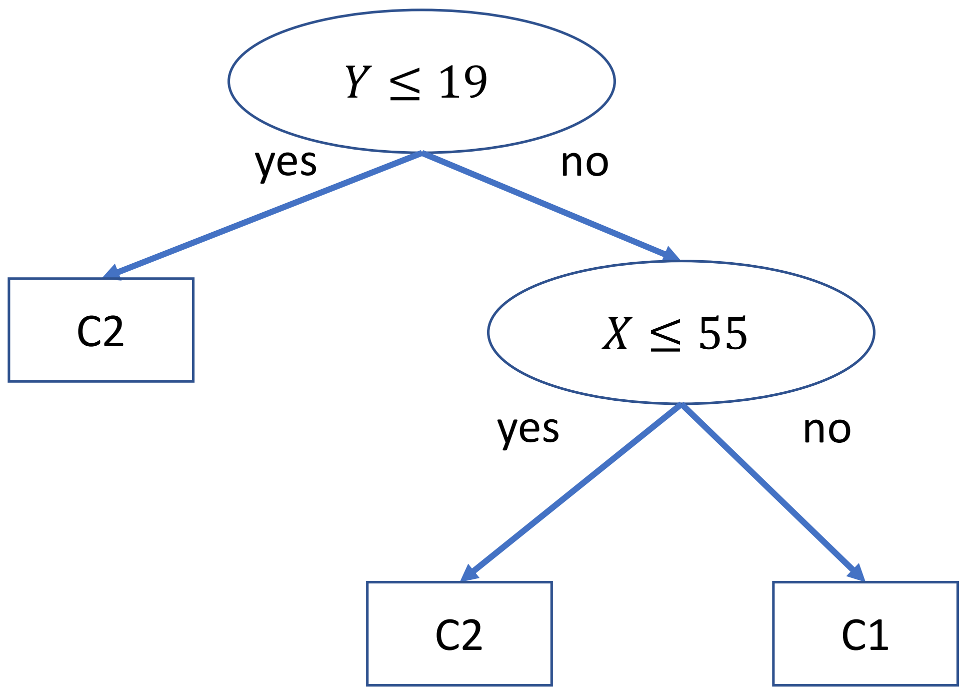
IF $Y \leq 19$ THEN C2

IF $Y > 19$ and $X \leq 55$ THEN C2

IF IF $Y > 19$ and $X > 55$ THEN C1



The Decision Tree

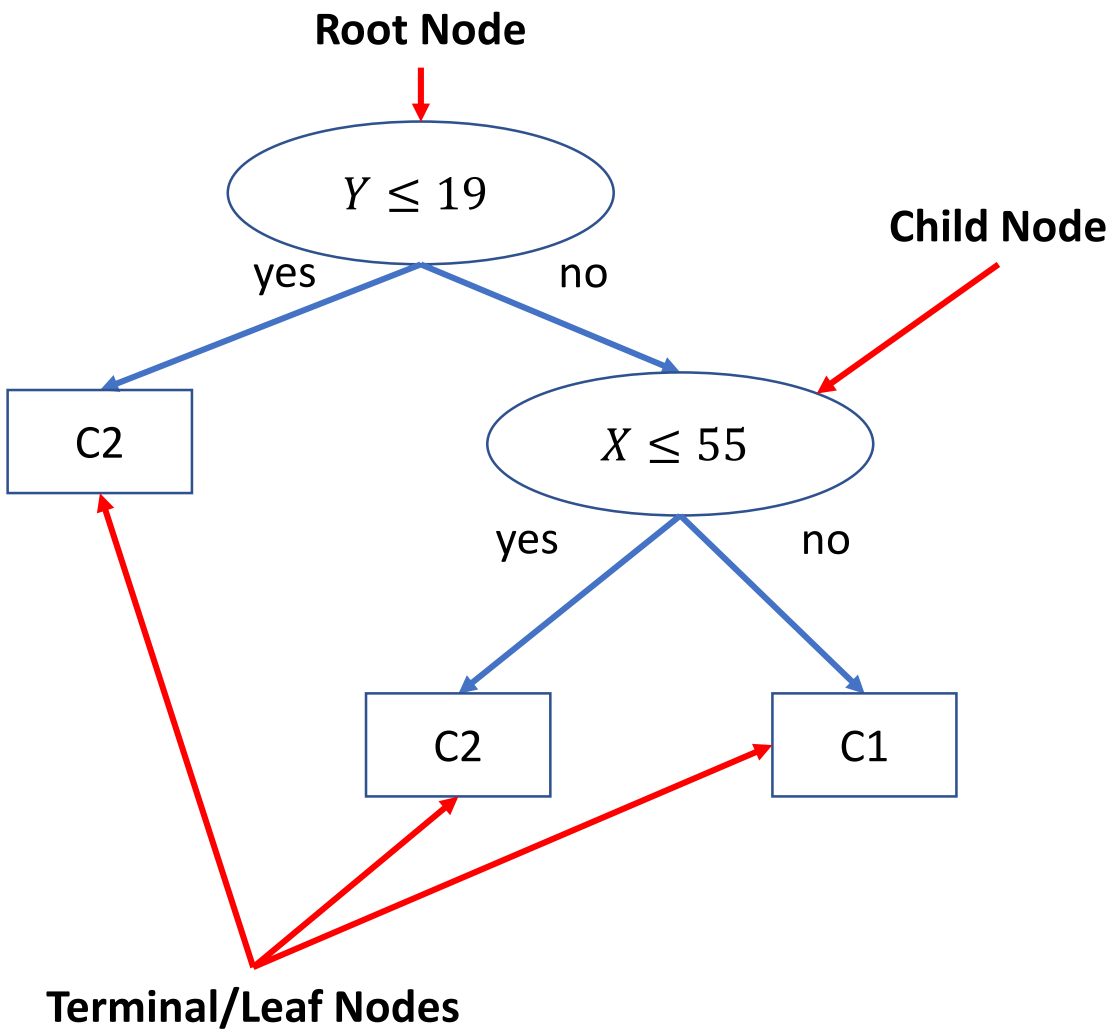


- **Decision Tree has three types of nodes**:
  - **Root Node** : top (or left-most) node with no incoming edges and zero or more outgoing edges.
  - **Child or Internal Node** : descendent node which has exactly one incoming edge and two or more outgoing edges.
  - **Leaf Node** : terminal node which has exactly one incoming edge and no outgoing edges.
- In Decision Tree, each **leaf node** is assigned a class label.
- The **rules or branches** are the unique path (edges) with a set of conditions (attribute) that divide the observations into smaller subset.

Read down tree to derive rule

E.g., If lot size < 19, and if income > 84.75, then class = "owner"

The Decision Tree



Recursive partitioning: Repeatedly split the records into two parts so as to achieve maximum homogeneity/purity within the new parts, or decrease the impurity.

Pruning the tree: Simplify the tree by pruning peripheral branches to avoid overfitting

Recursive Partitioning Steps

- Pick one of the predictor variables, x_i
- Pick a value of $x_{i,}$ say s_i, that divides the training data into two (not necessarily equal) portions
- Measure how "pure" or homogeneous each of the resulting portions are

  "Pure" = containing records of mostly one class
- Algorithm tries different values of $x_{i,}$ and s_i to maximize purity in initial split
- After you get a "maximum purity" split, repeat the process for a second split, and so on

What is impurity

Saying that the split is pure we mean that all records belong to the same class.

Two main measures of impurity

- Gini index
- Entropy

Gini Index

Gini Index for rectangle *A* containing *m* records

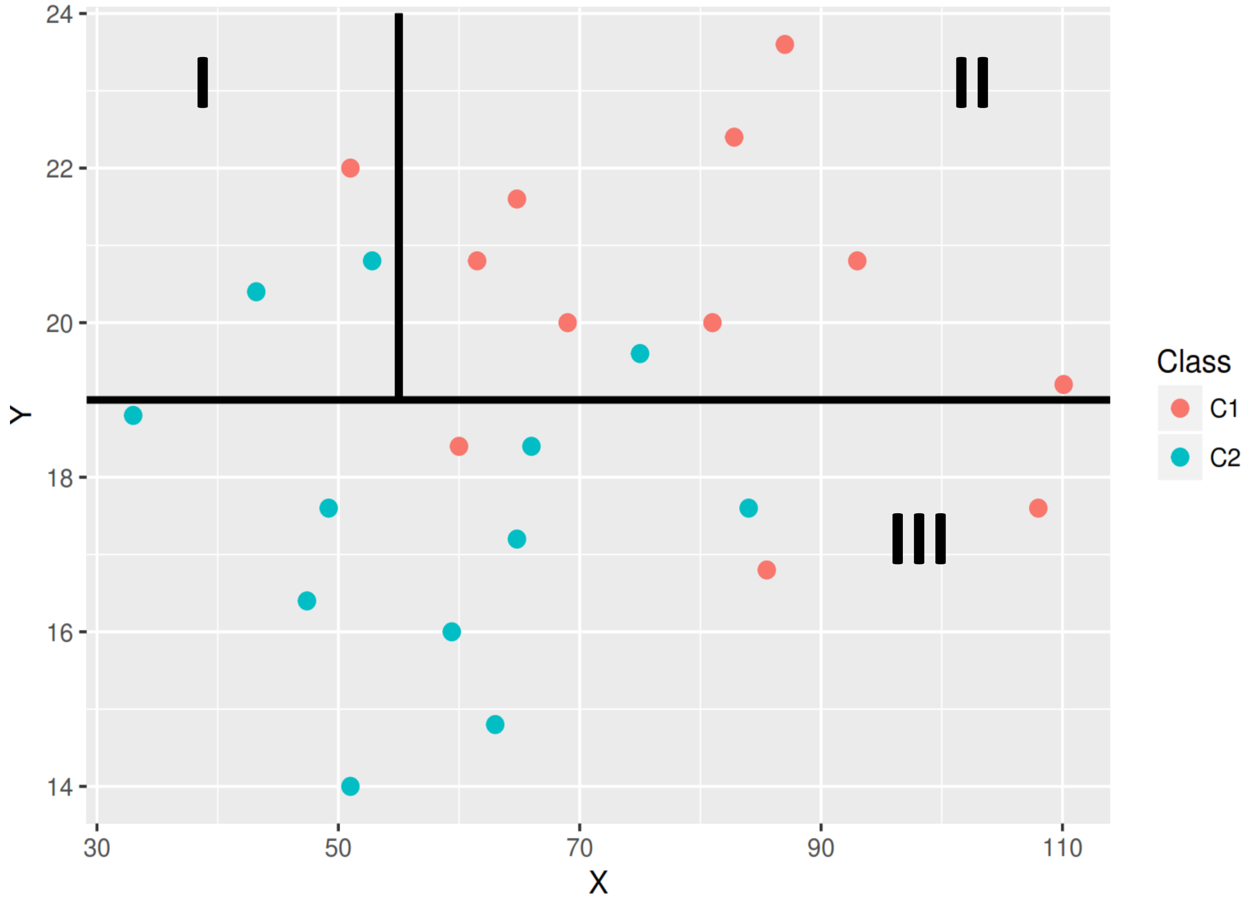
$$I(A) = 1 - \sum_{k=1}^{m} p_k^2$$

p = proportion of cases in rectangle *A* that belong to class *k*

- I(A) = 0 when all cases belong to same class
- Max value when all classes are equally represented (= 0.50 in binary case)

Gini Index

Calculate Gini index for rectangles I, II and III



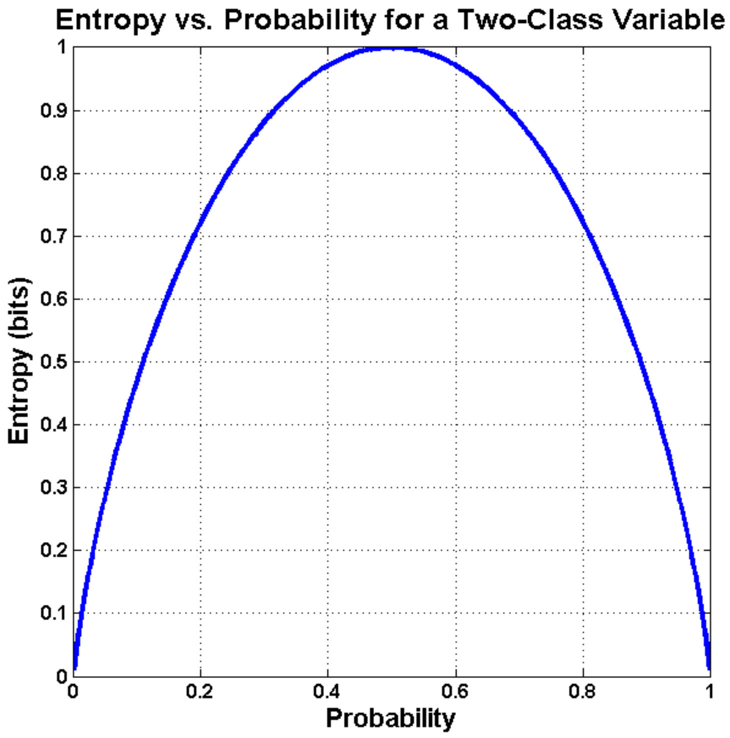
Entropy

$$entropy\,(A) = -\sum_{k=1}^{m} p_k \, log_2(p_k)$$

p = proportion of cases (out of *m*) in rectangle *A* that belong to class *k*

- Entropy ranges between 0 (most pure) and $\log_2(m)$ (equal representation of classes)

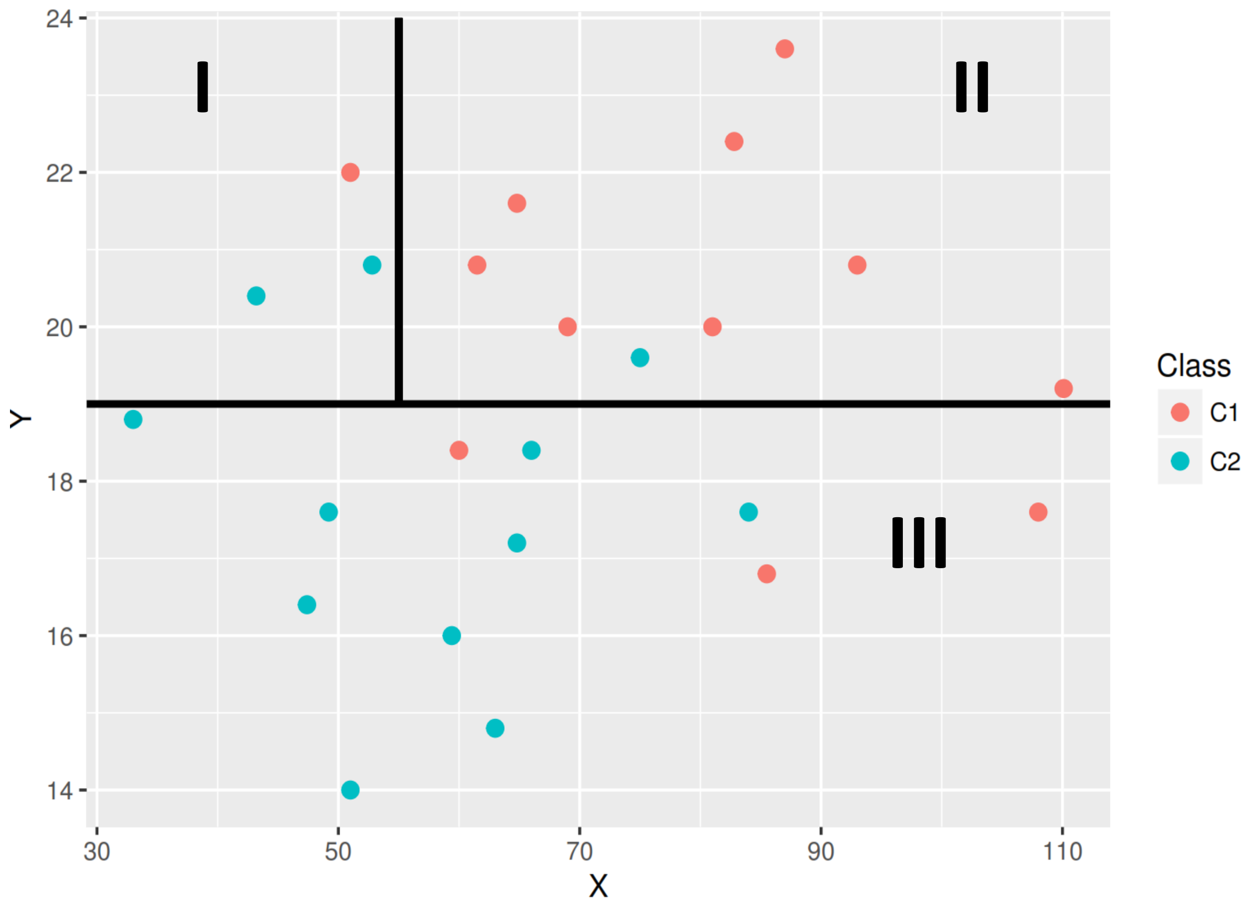
Measures of impurity: Entropy



•Entropy ranges between 0 (most pure) and $\log_2(0.5)$ (equal representation of classes)

Entropy

Calculate entropy for rectangles I, II and III



Impurity and Recursive Partitioning

- Obtain overall impurity measure (weighted avg. of individual rectangles)
- At each successive stage, compare this measure across all possible splits in all variables
- Choose the split that reduces impurity the most
- Chosen split points become nodes on the tree

Decision tree partitioning: Measuring impurity

Parent Node	Count
Class = 0	4
Class = 1	14

Node N1	Count
Class = 0	0
Class = 1	6

Node N2	Count
Class = 0	1
Class = 1	5

Node N3	Count
Class = 0	3
Class = 1	3

Calculate Gini index and entropy for each node

Decision tree partitioning: Measuring impurity

Parent Node	Count
Class = 0	4
Class = 1	14

Node N1	Count
Class = 0	0
Class = 1	6
Gini	0
Entropy	0

Node N2	Count
Class = 0	1
Class = 1	5
Gini	0.278
Entropy	0.650

Node N3	Count
Class = 0	3
Class = 1	3
Gini	0.5
Entropy	1

Information gain

- To determine how well a test condition performs, we need to compare the degree of impurity of the parent node (before splitting) and the child node (after splitting).
- The larger the different, the better the test condition.
- The **gain** Δ, is a criterion that can be used to determine the goodness of a split.

- $\Delta = I(Parent) - I(Split)$

Gain from splitting

Example :

	Parent
C0	6
C1	6
Gini = 0.5	

Gini :
$1 - (6/12)^2 - (6/12)^2$
= 0.5

Suppose there are two ways (A and B) to split the data into smaller subset.

A

	N1
C0	4
C1	3

	N2
C0	2
C1	3

B

	N1
C0	1
C1	4

	N2
C0	5
C1	2

Which one is better?

Information Gain

A

	N1
C0	4
C1	3

Gini Index: **0.4898**

	N2
C0	2
C1	3

Gini Index: **0.480**

Weighted Average of Gini Index:
[(7/12) x **0.4898**] + [(5/12) x **0.480**]
= 0.486

Gain, Δ = 0.5 - 0.486 = **0.014**

B

	N1
C0	1
C1	4

Gini Index: **0.320**

	N2
C0	5
C1	2

Gini Index: **0.4082**

Weighted Average of Gini Index:
[(5/12) x **0.320**] + [(7/12) x **0.4082**]
= 0.3715

Gain, Δ = 0.5 - 0.3715 = **0.1285**

Splitting by option B is better

Doing in R

Doing in R

we will use the following libraries

- rpart
- rpart.plot – to plot the trees

Doing in R

```r
weather<-read.csv("weather.csv")
str(weather)
```

```
## 'data.frame':    366 obs. of  21 variables:
##  $ MinTemp      : num  8 14 13.7 13.3 7.6 6.2 6.1 8.3 8.8 8.4 ...
##  $ MaxTemp      : num  24.3 26.9 23.4 15.5 16.1 16.9 18.2 17 19.5 22.8 ...
##  $ Rainfall     : num  0 3.6 3.6 39.8 2.8 0 0.2 0 0 16.2 ...
##  $ Evaporation  : num  3.4 4.4 5.8 7.2 5.6 5.8 4.2 5.6 4 5.4 ...
##  $ Sunshine     : num  6.3 9.7 3.3 9.1 10.6 8.2 8.4 4.6 4.1 7.7 ...
##  $ WindGustDir  : Factor w/ 16 levels "E","ENE","ESE",..: 8 2 8 8 11 10 10 1 9 1 ...
##  $ WindGustSpeed: int  30 39 85 54 50 44 43 41 48 31 ...
##  $ WindDir9am   : Factor w/ 16 levels "E","ENE","ESE",..: 13 1 4 15 11 10 10 10 1 9 ...
##  $ WindDir3pm   : Factor w/ 16 levels "E","ENE","ESE",..: 8 14 6 14 3 1 3 1 2 3 ...
##  $ WindSpeed9am : int  6 4 6 30 20 20 19 11 19 7 ...
##  $ WindSpeed3pm : int  20 17 6 24 28 24 26 24 17 6 ...
##  $ Humidity9am  : int  68 80 82 62 68 70 63 65 70 82 ...
##  $ Humidity3pm  : int  29 36 69 56 49 57 47 57 48 32 ...
##  $ Pressure9am  : num  1020 1012 1010 1006 1018 ...
##  $ Pressure3pm  : num  1015 1008 1007 1007 1018 ...
##  $ Cloud9am     : int  7 5 8 2 7 7 4 6 7 7 ...
##  $ Cloud3pm     : int  7 3 7 7 7 5 6 7 7 1 ...
##  $ Temp9am      : num  14.4 17.5 15.4 13.5 11.1 10.9 12.4 12.1 14.1 13.3 ...
##  $ Temp3pm      : num  23.6 25.7 20.2 14.1 15.4 14.8 17.3 15.5 18.9 21.7 ...
##  $ RainToday    : Factor w/ 2 levels "No","Yes": 1 2 2 2 2 1 1 1 1 2 ...
##  $ RainTomorrow : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 1 1 1 2 1 ...
```

Doing in R

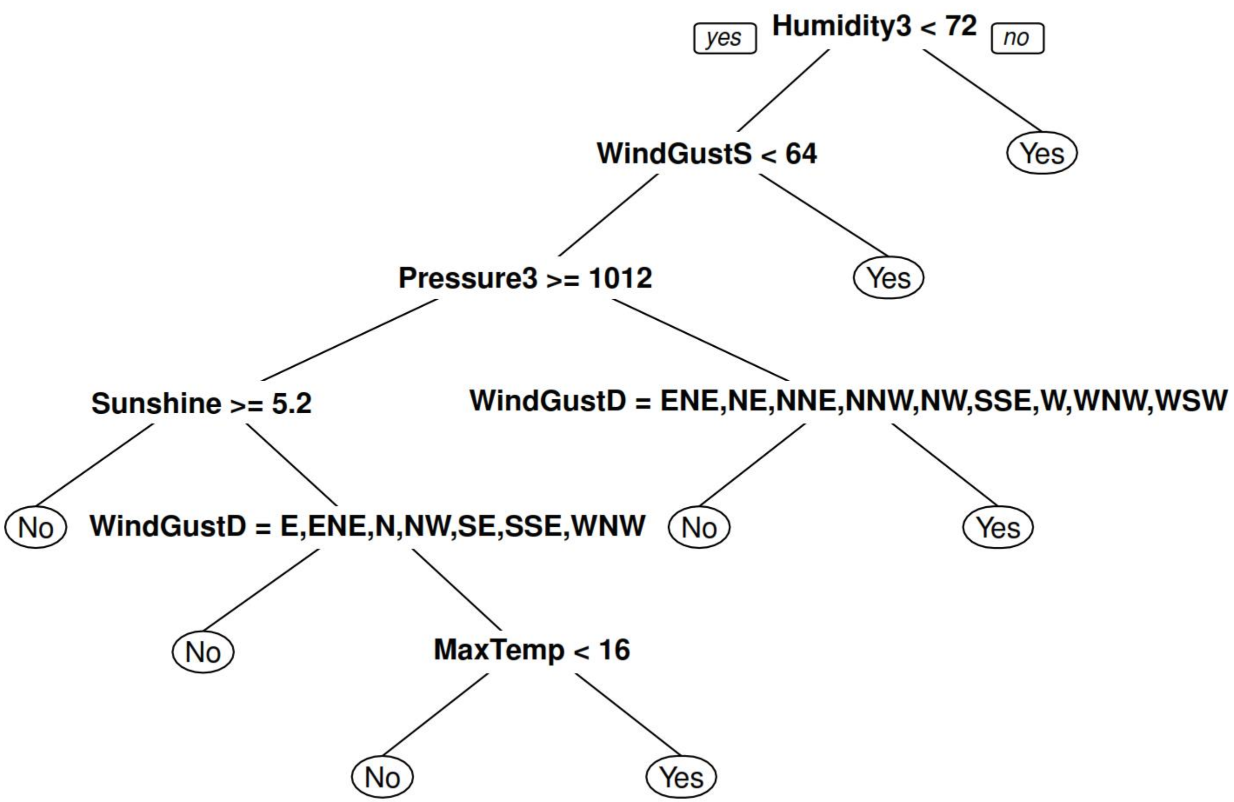
Specify the formula, with independent and dependent variables

```
library(rpart)
model<-rpart(RainTomorrow~., data=weather)
```

Doing in R

Plot using rpart.plot library

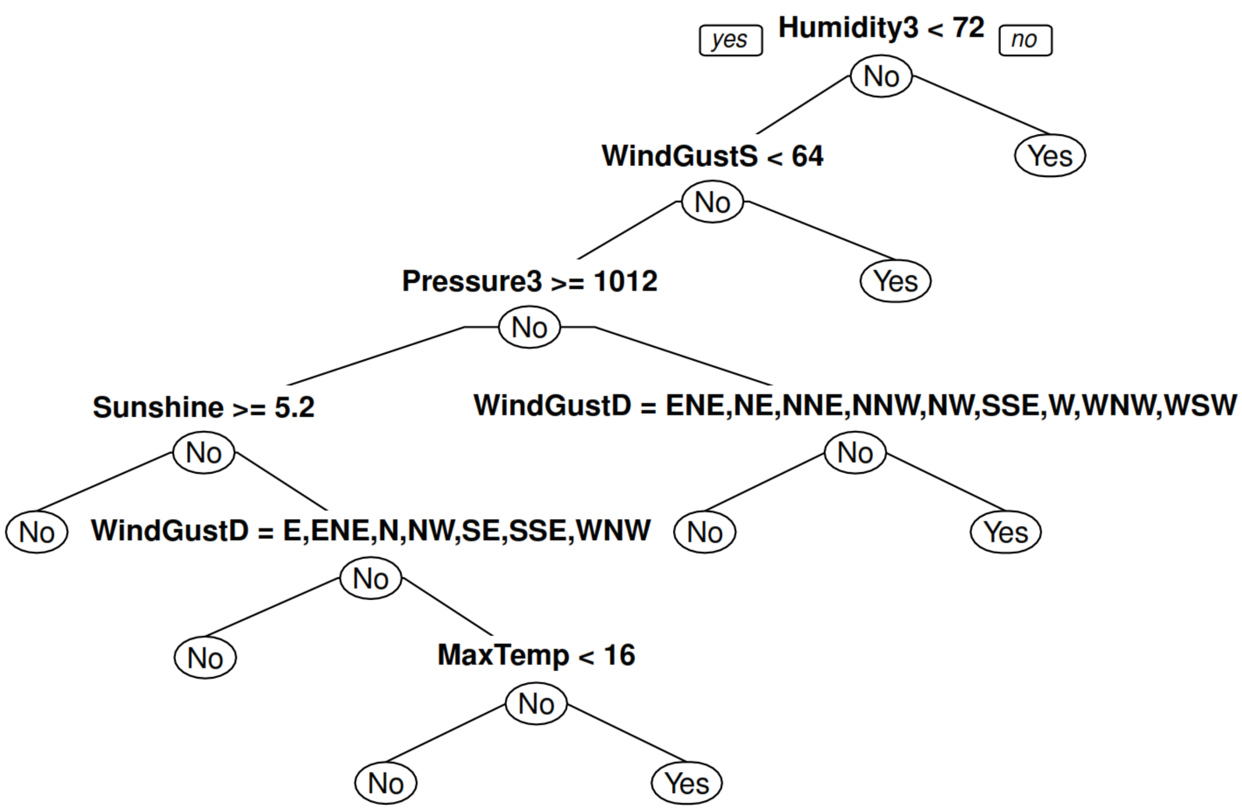
```
library(rpart.plot)
prp(model)
```



Doing in R

You can change the type argument to get different layouts for the tree. Look for help for more info: ?rpart.plot::prp

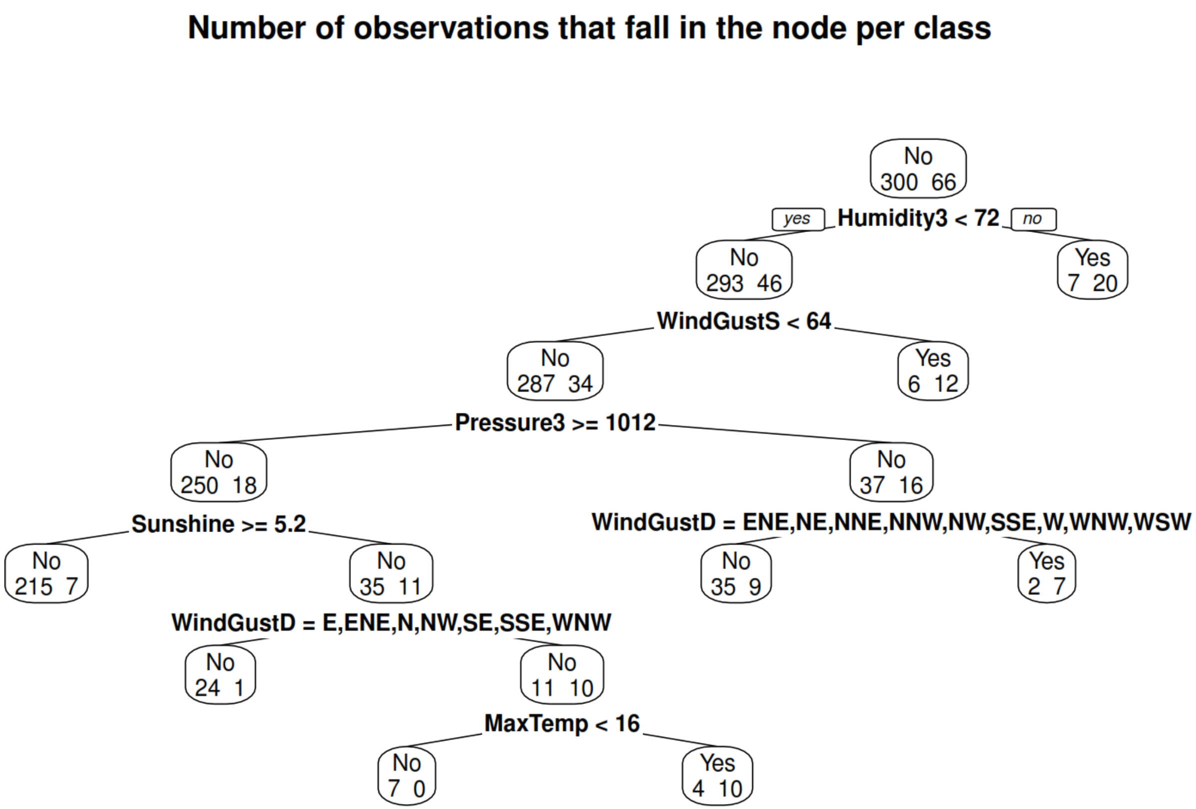
```
prp(model, type=1)
```



Doing in R

You can add extra information to the plot using argument extra

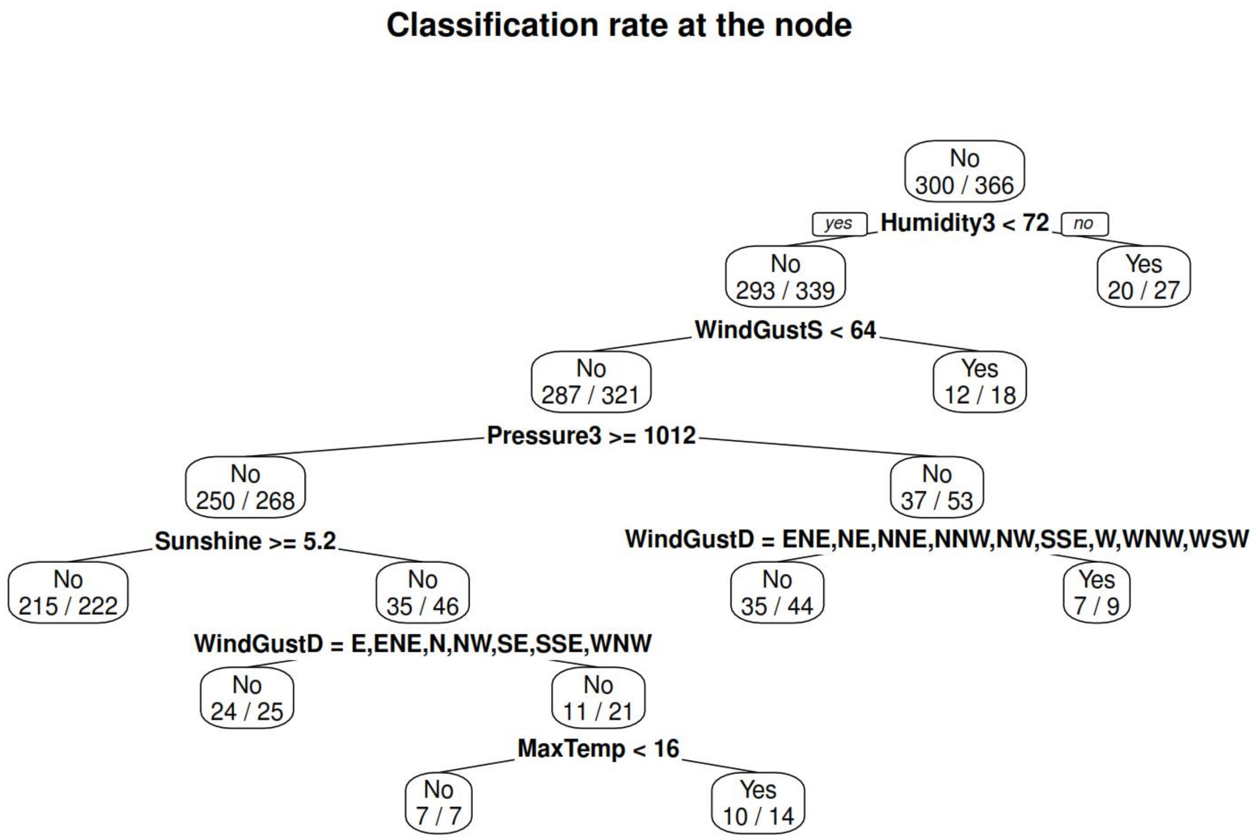
```
prp(model, type=2, extra=1, main="Number of observations that fall in the node per class")
```



Doing in R

extra=2

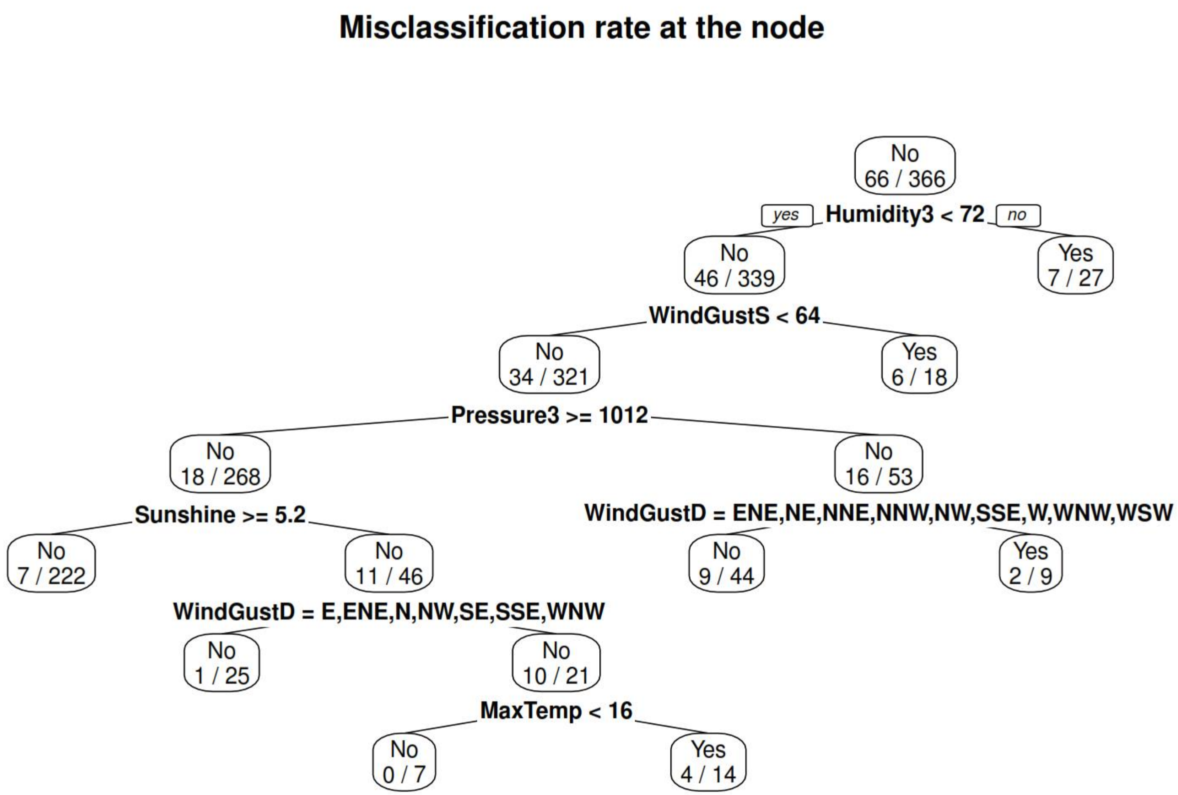
```
prp(model, type=2, extra=2, main="Classification rate at the node")
```



Doing in R

extra=3

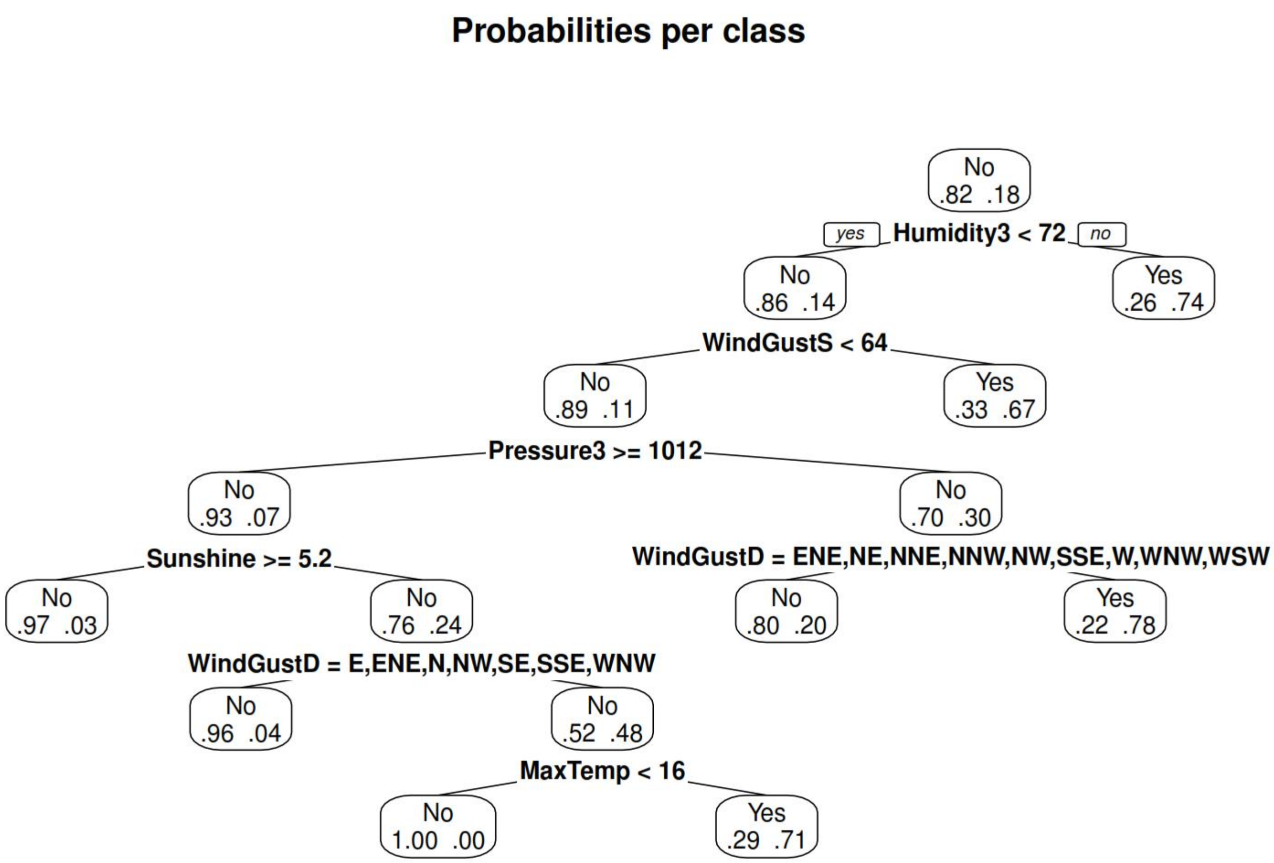
```
prp(model, type=2, extra=3, main="Misclassification rate at the node")
```



Doing in R

extra=4

```
prp(model, type=2, extra=4, main="Probabilities per class")
```



Doing in R

Look at the decision rules using rattle package

```
library(rattle)
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

##
## Attaching package: 'rattle'

## The following object is masked _by_ '.GlobalEnv':
##
##     weather
```

```
asRules(model)
```

```
##
##  Rule number: 19 [RainTomorrow=Yes cover=9 (2%) prob=0.78]
##    Humidity3pm< 71.5
##    WindGustSpeed< 64
##    Pressure3pm< 1012
##    WindGustDir=E,ESE,N,S,SW
##
##  Rule number: 3 [RainTomorrow=Yes cover=27 (7%) prob=0.74]
##    Humidity3pm>=71.5
##
##  Rule number: 71 [RainTomorrow=Yes cover=14 (4%) prob=0.71]
##    Humidity3pm< 71.5
##    WindGustSpeed< 64
##    Pressure3pm>=1012
##    Sunshine< 5.15
##    WindGustDir=ESE,NE,NNE,NNW,S
##    MaxTemp>=15.8
##
##  Rule number: 5 [RainTomorrow=Yes cover=18 (5%) prob=0.67]
##    Humidity3pm< 71.5
##    WindGustSpeed>=64
##
```

Doing in R

Rules

- Ignore the rule number
- The predicted class is Yes, it covers 2% of the data, overall 9 cases in the terminal node, probability of Yes is 0.78

```
Rule number: 19 [RainTomorrow=Yes cover=9 (2%) prob=0.78]
  Humidity3pm< 71.5
  WindGustSpeed< 64
  Pressure3pm< 1012
  WindGustDir=E,ESE,N,S,SW
```

Doing in R

- The terminal node predicts No,
- covers 7% of the data (25 cases)
- prob=0.04, probability of **Yes** is 0.04, for **No** is 0.96

```
Rule number: 34 [RainTomorrow=No cover=25 (7%) prob=0.04]
  Humidity3pm< 71.5
  WindGustSpeed< 64
  Pressure3pm>=1012
  Sunshine< 5.15
  WindGustDir=E,ENE,N,NW,SE,SSE,WNW
```

Doing in R

Controlling the tree

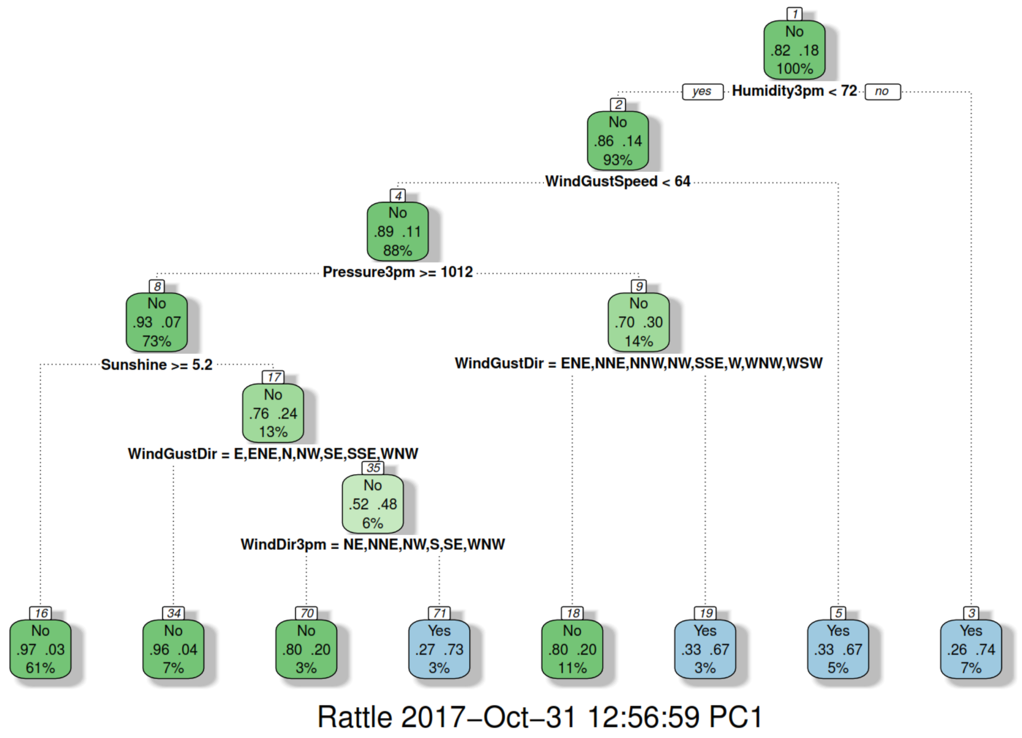
- by default Gini coefficient is the impurity measure
- Tree is pruned using Complexity parameter
- Other parameters to control tree growth
    - **minsplit:** the minimum number of observations that must exist in a node in order for a split to be attempted
    - **minbucket:** the minmum number of observations in any terminal node,
    - look for more at help: ??rpart.control

Doing in R

```
model<-rpart(RainTomorrow~.-RISK_MM, data=weather,
             control=rpart.control(minsplit=10, minbucket=10))
```

fancyRpartPlot from library rattle

```
fancyRpartPlot(model)
```



Credit dataset

```
credit<-read.csv("credit.csv")
library(caret)
```

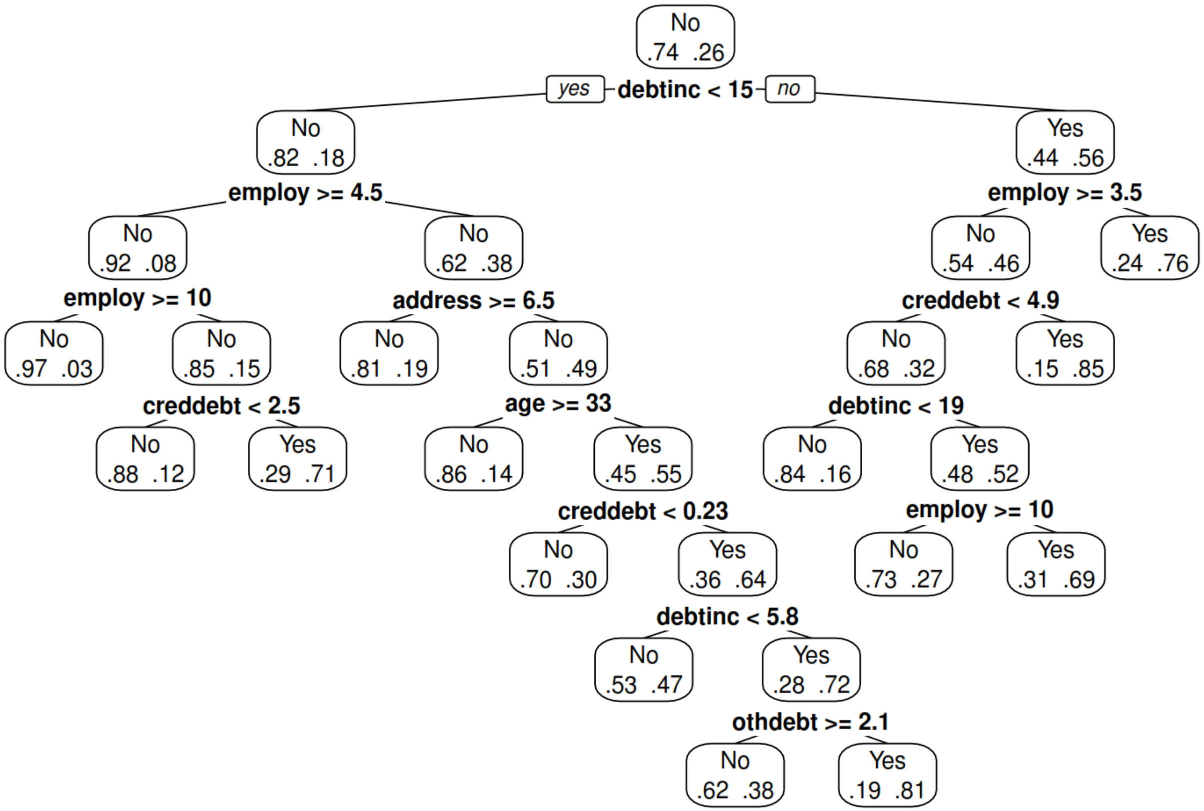
```
## Loading required package: lattice
```

```
index<-createDataPartition(credit$default, p=0.8, list=F)
Train<-credit[index,]
Test<-credit[-index,]
```

Credit dataset

```
model_c<-rpart(default~., data=Train)
```

```
prp(model_c, type=2, extra=4)
```



Credit dataset

```
pred_class<-predict(model_c, Test, type="class")
pred_class[1:20]
```

```
##  10  15  16  25  36  38  40  43  44  48  53  62  66  68  77  79  82  84
## Yes  No  No Yes Yes  No  No  No  No  No  No  No Yes  No  No  No  No  No
##  86  99
##  No  No
## Levels: No Yes
```

Credit dataset

```r
confusionMatrix(pred_class, Test$default, positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  93  15
##        Yes 10  21
##
##                Accuracy : 0.8201
##                  95% CI : (0.7461, 0.8801)
##     No Information Rate : 0.741
##     P-Value [Acc > NIR] : 0.01823
##
##                   Kappa : 0.5093
##  Mcnemar's Test P-Value : 0.42371
##
##             Sensitivity : 0.5833
##             Specificity : 0.9029
##          Pos Pred Value : 0.6774
##          Neg Pred Value : 0.8611
##              Prevalence : 0.2590
##          Detection Rate : 0.1511
##    Detection Prevalence : 0.2230
##       Balanced Accuracy : 0.7431
##
##        'Positive' Class : Yes
##
```

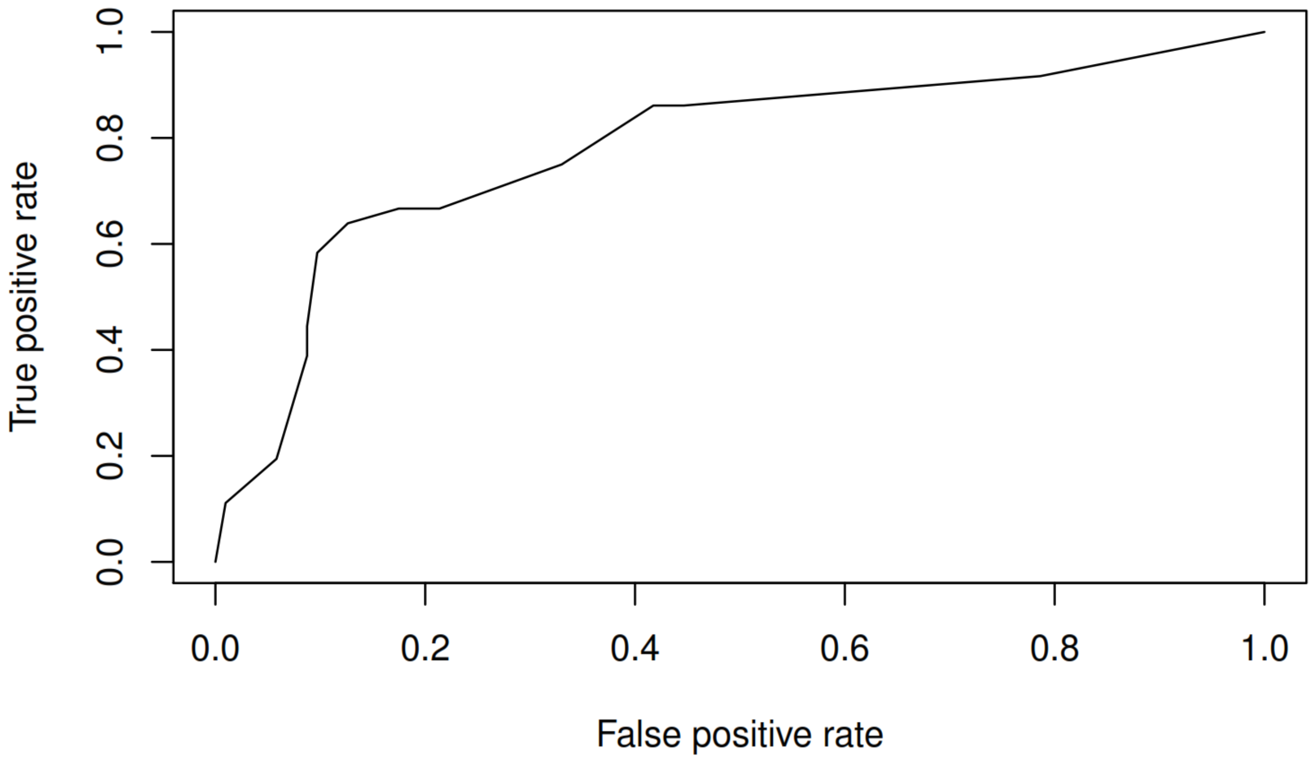
Credit dataset

```
pred_prob<-predict(model_c, Test)
pred_prob[1:10,]
```

```
##            No        Yes
## 10 0.2439024 0.75609756
## 15 0.9736842 0.02631579
## 16 0.8823529 0.11764706
## 25 0.2439024 0.75609756
## 36 0.1500000 0.85000000
## 38 0.9736842 0.02631579
## 40 0.9736842 0.02631579
## 43 0.8823529 0.11764706
## 44 0.9736842 0.02631579
## 48 0.8571429 0.14285714
```

Credit dataset

```
P_Test <- prediction(pred_prob[,2], Test$default)
perf <- performance(P_Test,"tpr","fpr")
plot(perf)
```



Credit dataset

```
performance(P_Test,"auc")@y.values
```

```
## [[1]]
## [1] 0.7815534
```