DATA2002

Decision trees and random forests

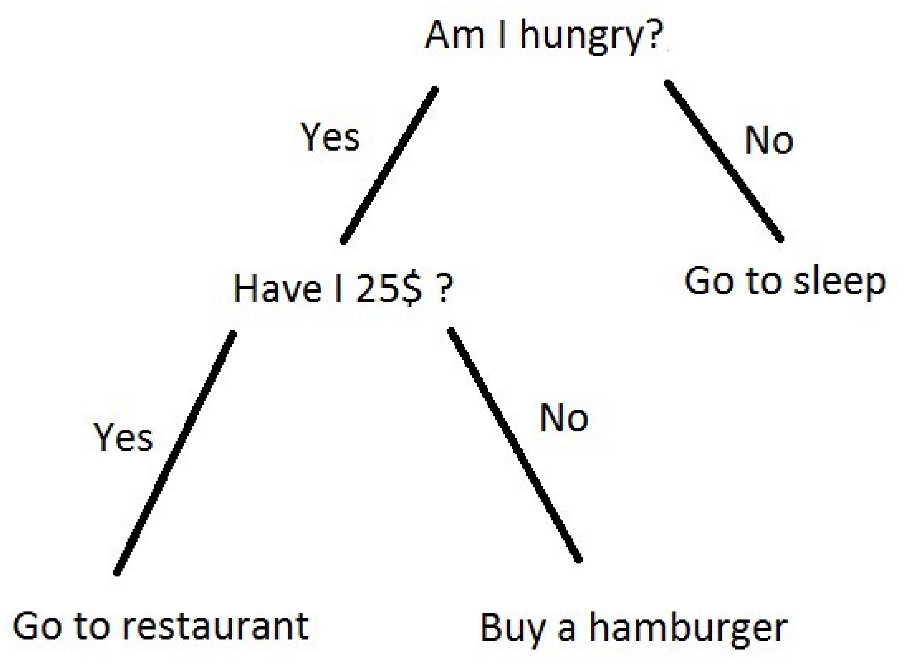
Garth Tarr

THE UNIVERSITY OF SYDNEY

Decision trees

Random forests

Decision trees



A decision tree determines the predicted outcome based on series of questions and conditions.

When making Decision Trees, there are several factors we take into consideration.

- What features do we make our decisions on?
- What is the threshold for classifying each question into a **yes** or **no** answer?

Image Source: Egor Dezhic

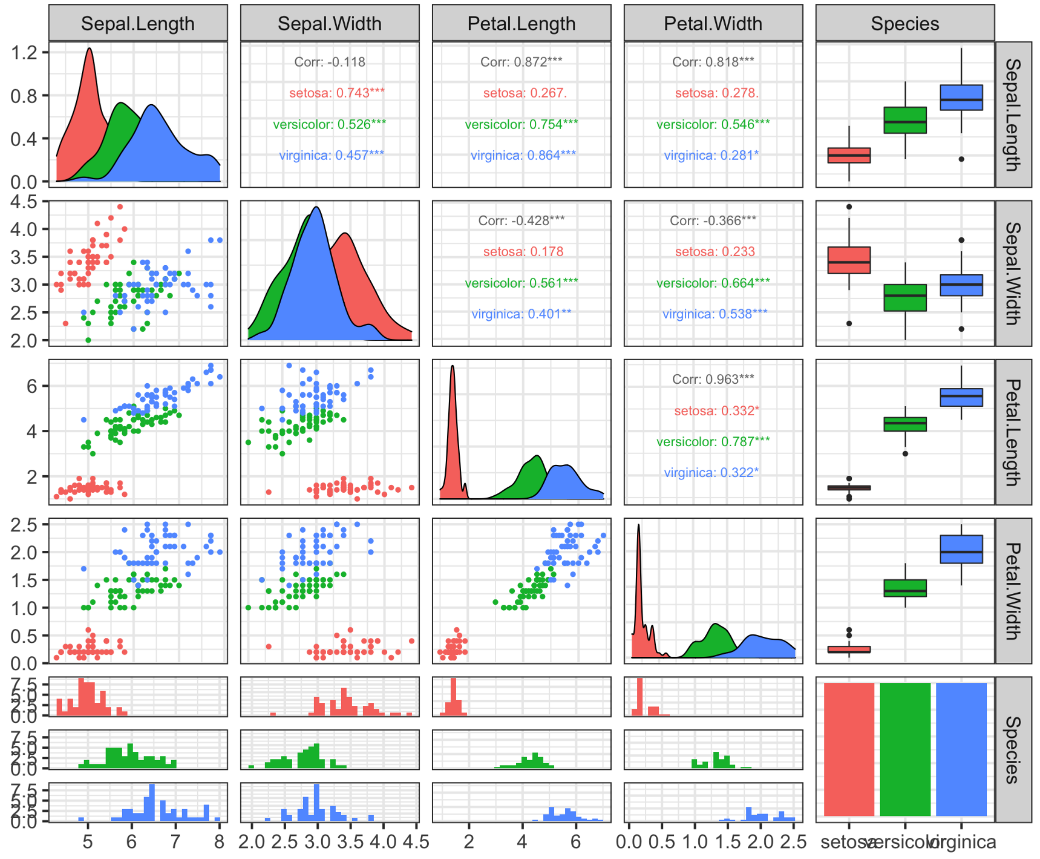
Consider the `iris` dataset

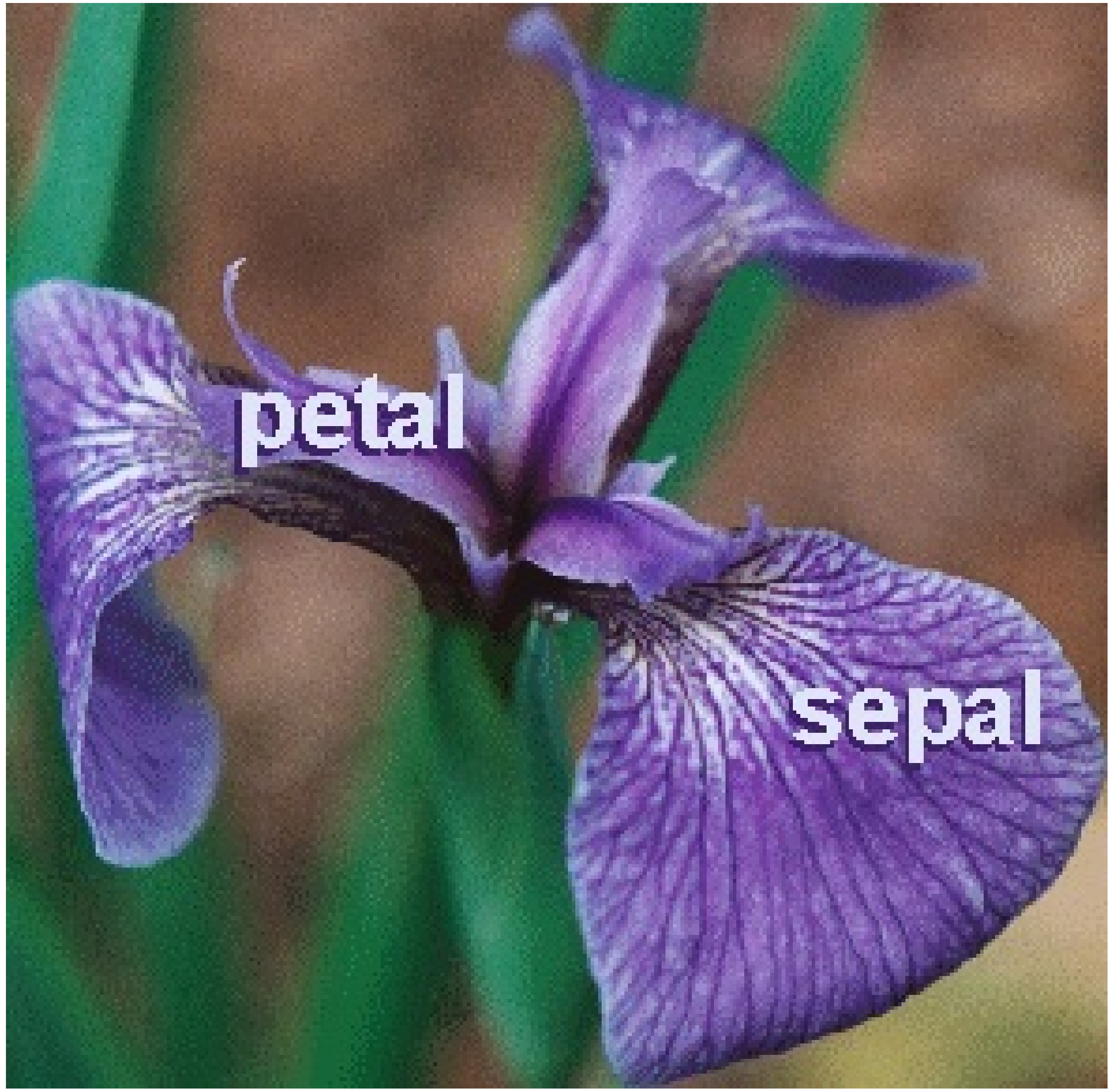
```r
library(tidyverse)
glimpse(iris)
```

```
## Rows: 150
## Columns: 5
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4,
## $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9,
## $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7,
## $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4,
## $ Species      <fct> setosa, setosa, setosa, setosa
```

How can we construct a **decision tree** to determine the `Species` of `iris` given petal and sepal measurements?

```r
library(GGally)
ggpairs(iris, mapping = aes(col = Species)) + t
```




petal
sepal

Trees in R

- We can use the **rpart** package (recursive partitioning) to build our decision tree.
- It uses a formula structure, much like `lm()` and `glm()` to identify the dependent variable (what we're trying to predict) and the explanatory variables (what information do we have available to help prediction).

```
library(rpart)
tree = rpart(Species ~ ., data = iris, method = "class")
```

- The formula `Species ~ .` in the code above says we want to predict `Species` using all the variables in the data frame `iris`.
- Specifying `method = "class"` means we want to build a classification (decision) tree.

```r
tree
```

```
## n= 150
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Width< 1.75 54   5 versicolor (0.00000000 0.90740741 0.09259259) *
##     7) Petal.Width>=1.75 46   1 virginica (0.00000000 0.02173913 0.97826087) *
```

```r
summary(tree)
```

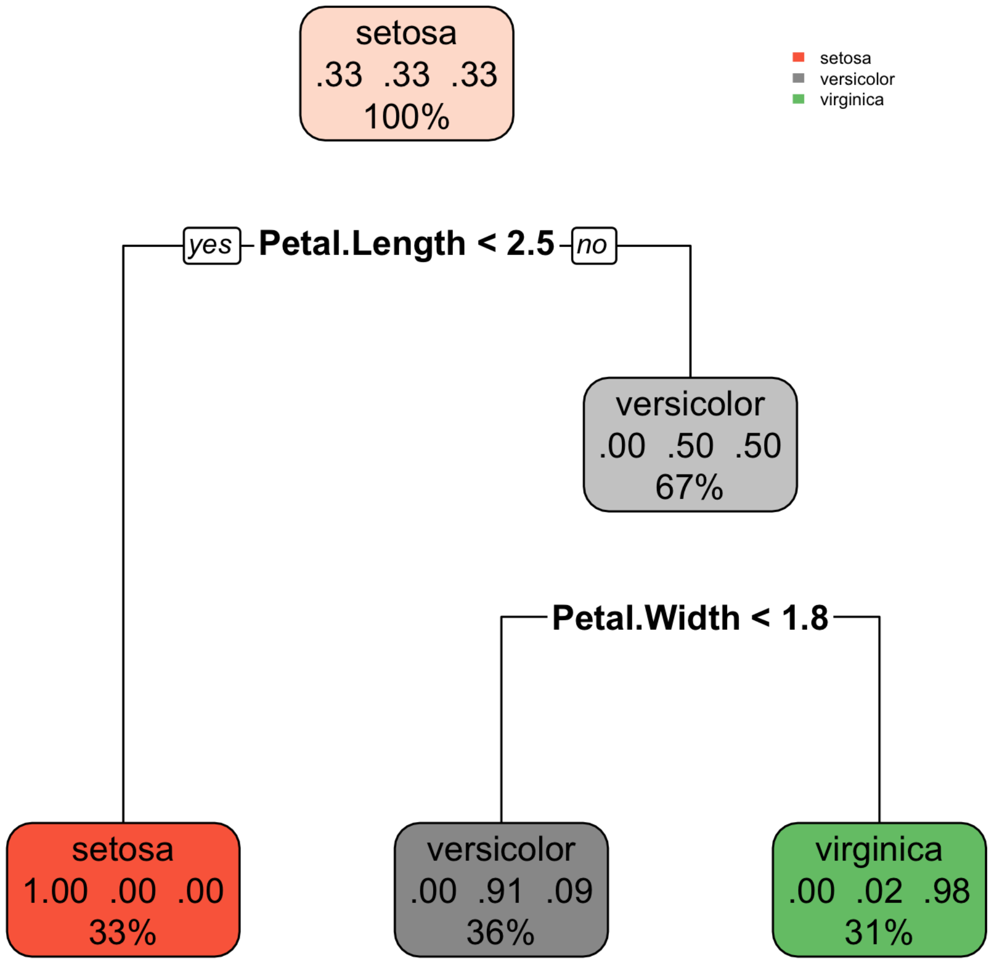
```
## Call:
## rpart(formula = Species ~ ., data = iris, method = "class")
##   n= 150
##
##     CP nsplit rel error xerror       xstd
## 1 0.50      0      1.00   1.16 0.05127703
## 2 0.44      1      0.50   0.72 0.06118823
```

Trees in R

```r
library(rpart)
tree = rpart(
  Species ~ .,
  data = iris,
  method = "class")

library(rpart.plot)
rpart.plot(tree)
```

- Using the `rpart.plot()` function from the **rpart.plot** library, we can visualise the results of our classification tree.
- Each node shows the predicted class, the predicted probability of each class, and the percentage of observations in each node.

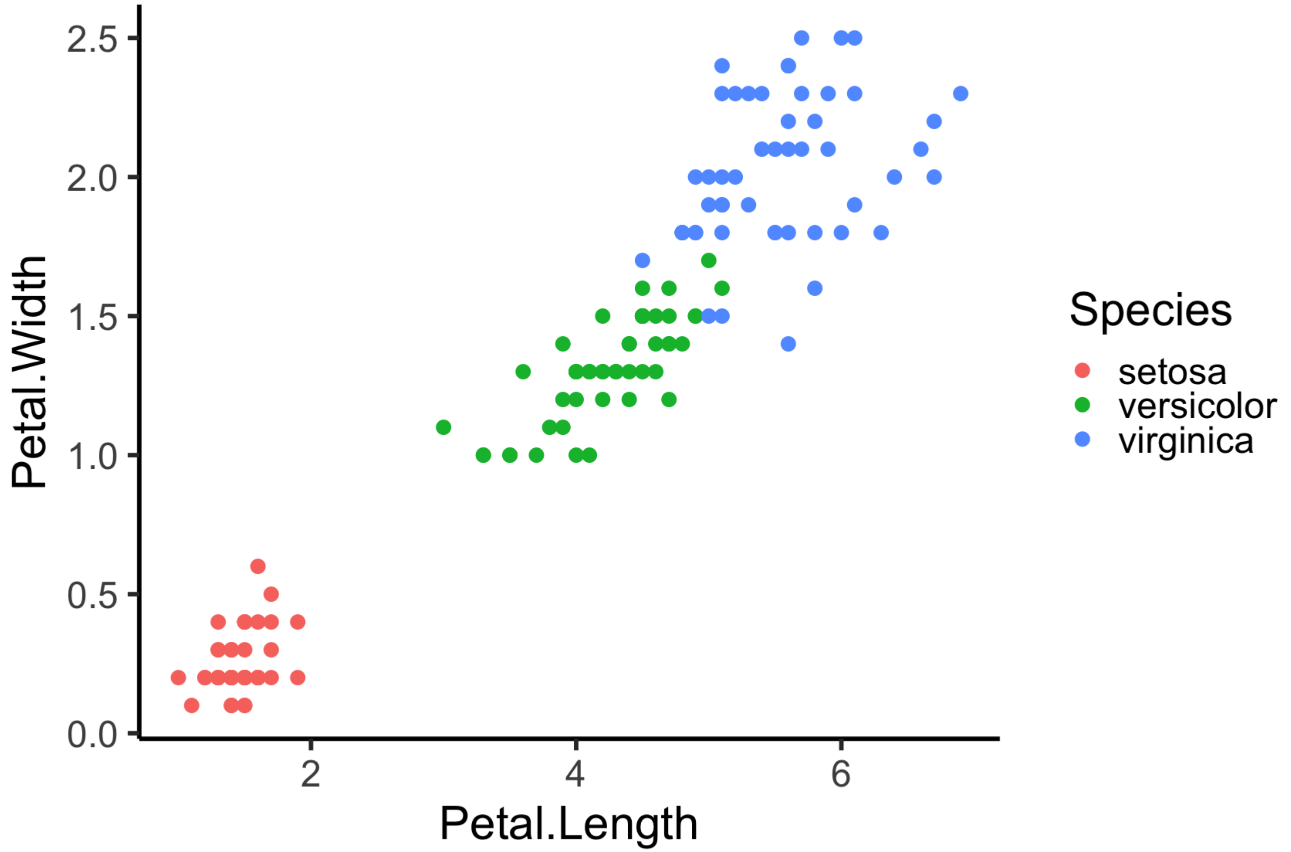


Is it magic?

How does it work? In this tree, we only need to consider two variables.

```
p1 = iris %>% ggplot() +
  aes(x = Petal.Length,
      y = Petal.Width,
      colour = Species) +
  geom_point(size = 4) +
  theme_classic(base_size = 30)
```

```
p1
```



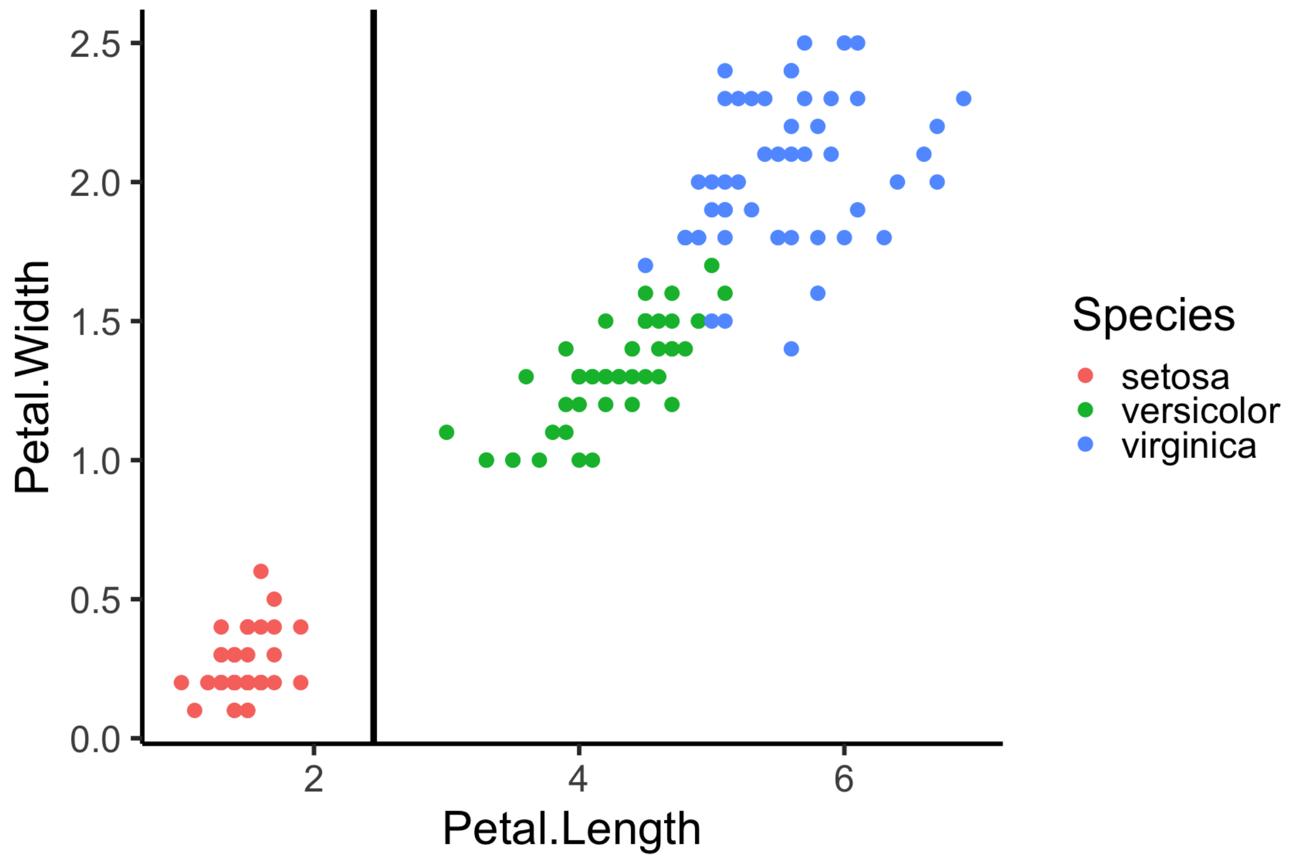
Is it magic?

The first branch is done to "best" split the data to create the most "pure" (homogenous) partitions.

```
Petal.Length < 2.45
```

```
p2 = p1 +
  geom_vline(
    aes(xintercept=2.45),
    size = 2
  )
```

```
p2
```



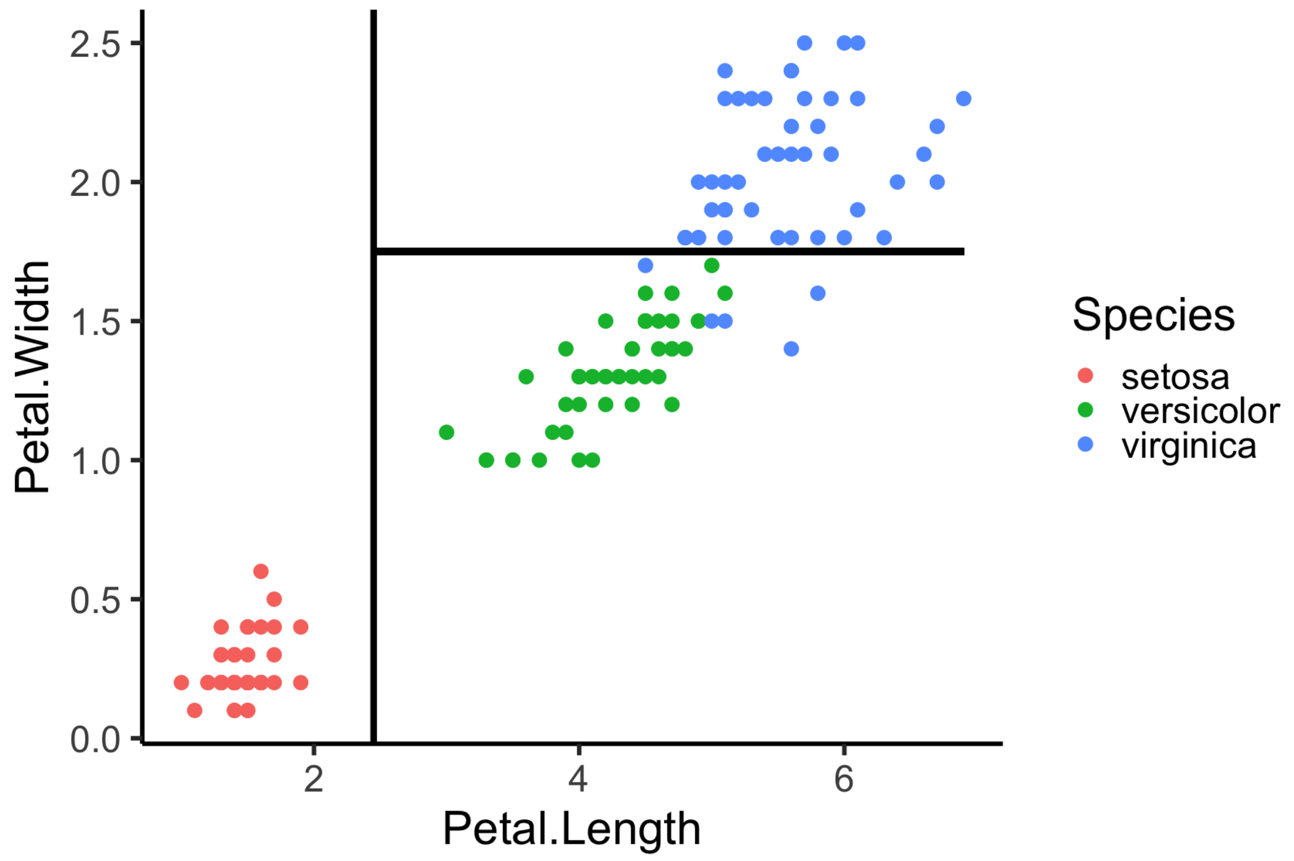
Is it magic?

The next branch applies to observations that have `Petal.Length > 2.45` and it tries to find the next best split of the data.

`Petal.width < 1.75`

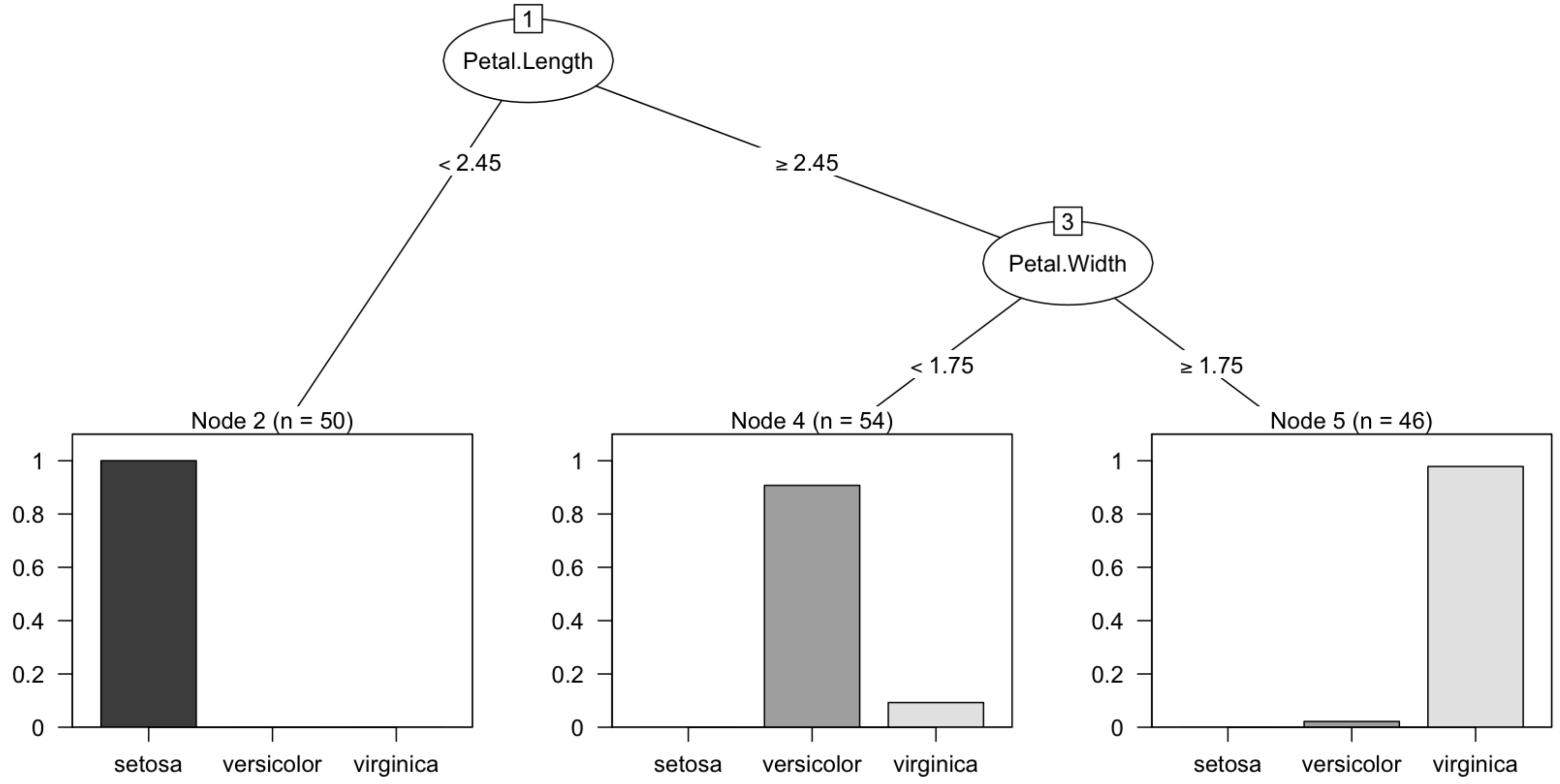
```
p3 = p2 +
  geom_segment(
    aes(x = 2.45, y = 1.75,
        xend = 6.9, yend = 1.75),
    size = 2,
    colour = "black"
  )
```

```
p3
```



Alternative visualisation with partykit

```r
# install.packages("partykit") # A Toolkit for Recursive Partytioning
library(partykit)
plot(as.party(tree))
```



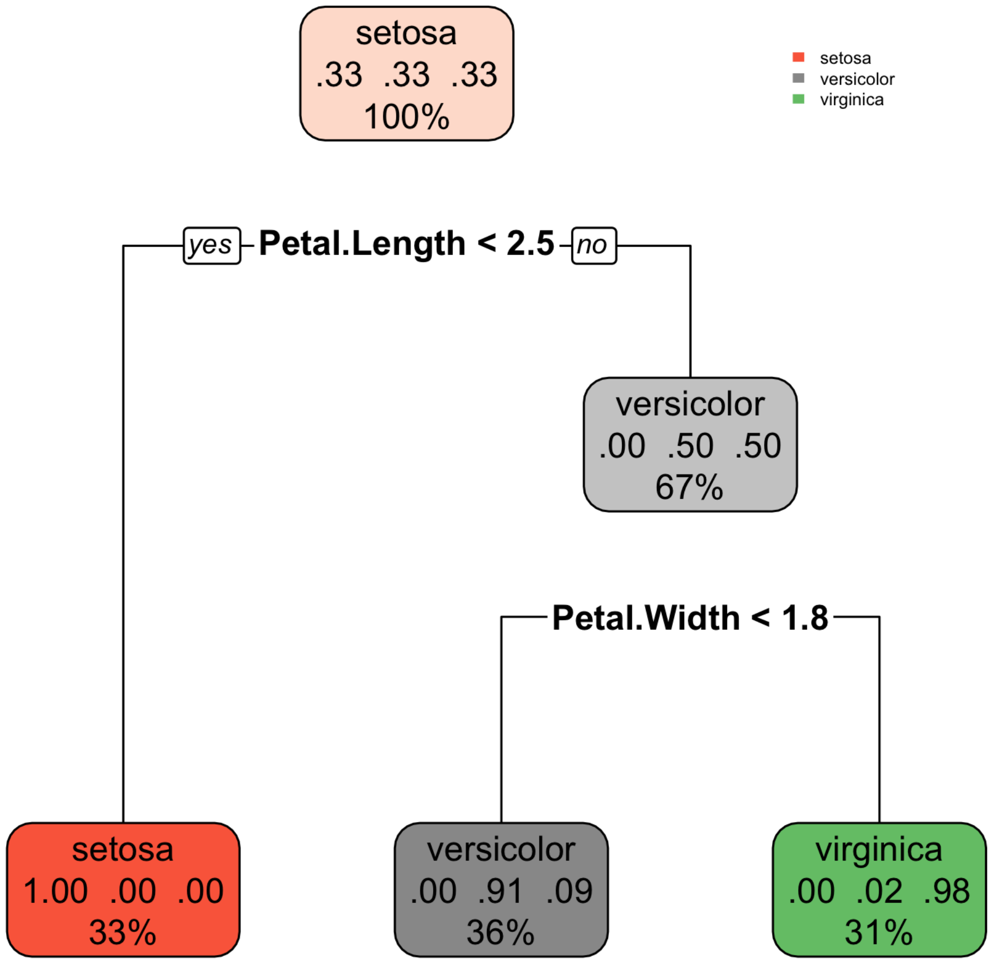
Making a prediction

How would we predict the species of a flower with

- Sepal.Length = 5.0
- Sepal.Width = 3.9
- Petal.Length = 1.4
- Petal.Width = 0.3

How about:

- Sepal.Length = 5.0
- Sepal.Width = 3.9
- Petal.Length = 3.4
- Petal.Width = 0.3



Making a prediction

This was our fitted model:

```
tree <- rpart(Species ~ ., data = iris, method = "class")
```

Using `predict()`, we can predict the class of a new data point, much like for `lm()` and `glm()` objects.

```
new_data = data.frame(Sepal.Length = c(5.0, 5.0),
                      Sepal.Width = c(3.9, 3.9),
                      Petal.Length = c(1.4, 3.4),
                      Petal.Width = c(0.3,0.3))
new_data
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5         3.9          1.4         0.3
## 2            5         3.9          3.4         0.3
```

```
predict(tree, new_data, type = "class")
```

```
##          1          2
##     setosa versicolor
## Levels: setosa versicolor virginica
```

Assessing in sample performance in the iris data

```r
library(caret)
predicted_species = predict(tree, type = "class")
confusionMatrix(
  data = predicted_species,
  reference = iris$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         49         5
##   virginica       0          1        45
##
## Overall Statistics
##
##                Accuracy : 0.96
##                  95% CI : (0.915, 0.9852)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.94
##
```

Model selection

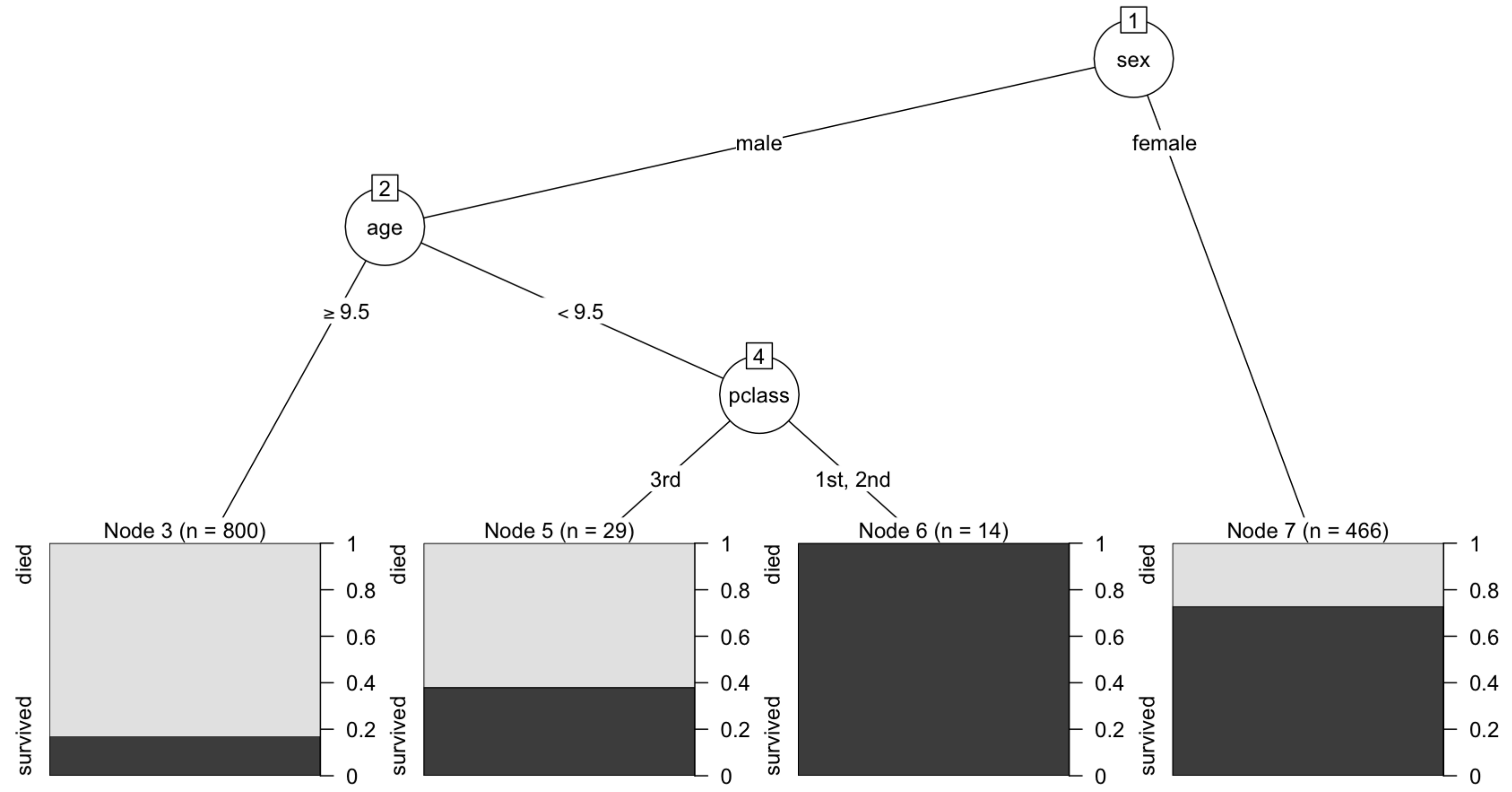
- How did our tree know to know to only use two splits?
- There is a **complexity parameter** that can be used to determine if a proposed new split *sufficiently* improves the predictive power or not.
- A choice is made whether to keep or "prune" a proposed new branch.
- Default is that a branch should decrease the error by 1%.
- This helps to avoid **overfitting**.

Titanic tree

```r
data("Titanicp", package = "vcdExtra")
titanic_tree = rpart(survived ~ sex + age + pclass, data = Titanicp, method = "class")
titanic_tree
```

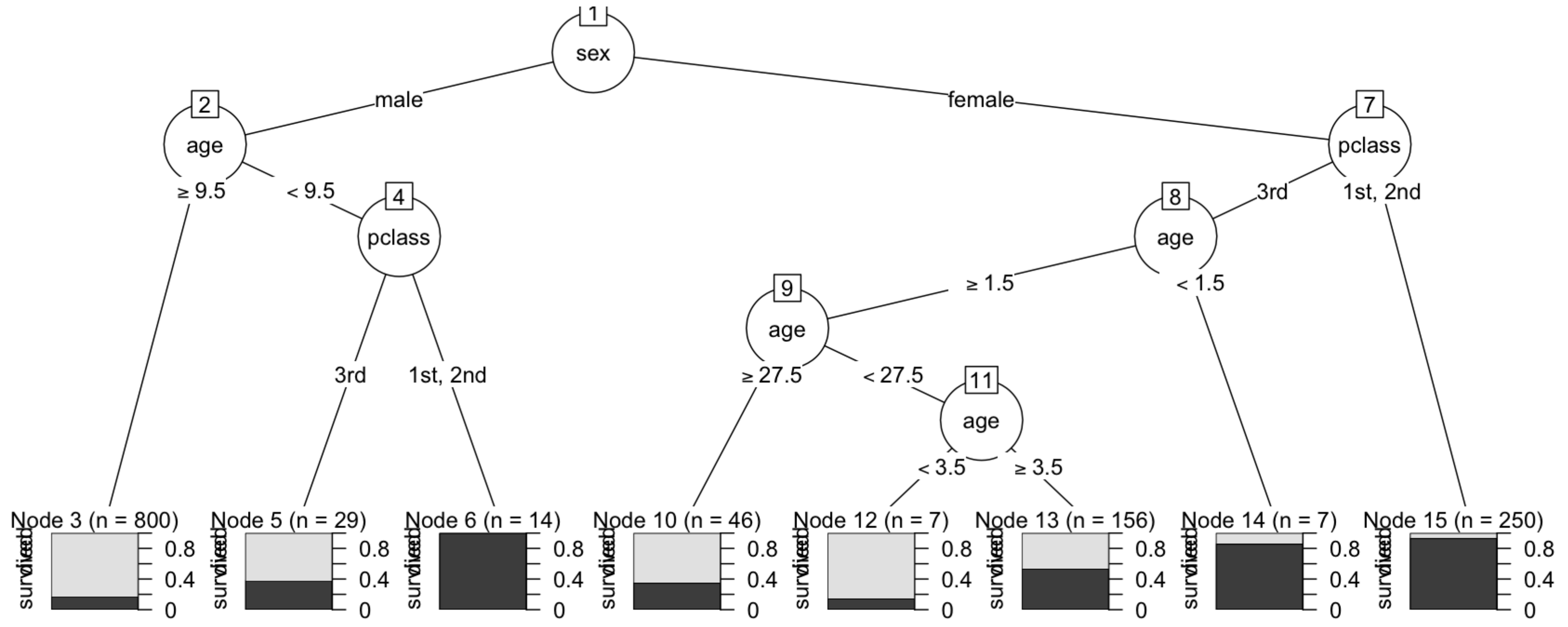
```
## n= 1309
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1309 500 died (0.6180290 0.3819710)
##    2) sex=male 843 161 died (0.8090154 0.1909846)
##      4) age>=9.5 800 136 died (0.8300000 0.1700000) *
##      5) age< 9.5 43  18 survived (0.4186047 0.5813953)
##       10) pclass=3rd 29  11 died (0.6206897 0.3793103) *
##       11) pclass=1st,2nd 14   0 survived (0.0000000 1.0000000) *
##    3) sex=female 466 127 survived (0.2725322 0.7274678) *
```

```
plot(as.party(titanic_tree))
```



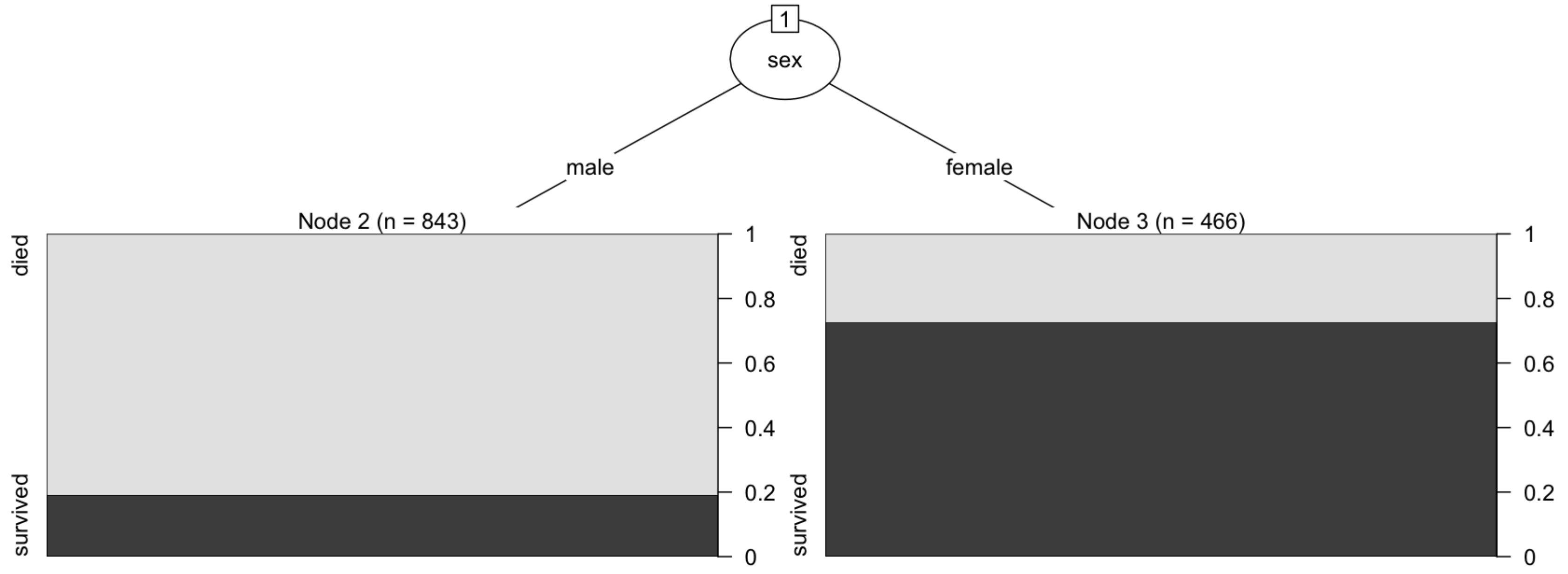
What if we in lower the complexity parameter threshold, so that each new branch only needs to decrease the error by 0.9%?

```
titanic_tree0.9 = rpart(survived ~ sex + age + pclass, data = Titanicp, method = "class",
                        control = rpart.control(cp = 0.009))
plot(as.party(titanic_tree0.9))
```



What if we in increase the complexity parameter threshold, so that each new branch needs to decrease the error by 2%?

```
titanic_tree2 = rpart(survived ~ sex + age + pclass, data = Titanicp, method = "class",
                      control = rpart.control(cp = 0.02))
plot(as.party(titanic_tree2))
```



Evaluating (in-sample) performance

1% (default)

```
titanic_1_pred = predict(titanic_tree, type = "class")
confusionMatrix(data=titanic_1_pred, reference = Titanicp$survived)$table
```

```
##           Reference
## Prediction died survived
##   died      682      147
##   survived  127      353
```

```
confusionMatrix(data=titanic_1_pred, reference = Titanicp$survived)$overall[1]
```

```
##  Accuracy
## 0.7906799
```

Evaluating (in-sample) performance

0.9%

```
titanic_0.9_pred = predict(titanic_tree0.9, type = "class")
confusionMatrix(data=titanic_0.9_pred,
                reference = Titanicp$survived)$table
```

```
##            Reference
## Prediction died survived
##   died      718      164
##   survived   91      336
```

```
confusionMatrix(data=titanic_0.9_pred,
                reference = Titanicp$survived)$overall[1]
```

```
##  Accuracy
## 0.8051948
```

Evaluating (in-sample) performance

2%

```r
titanic_2_pred = predict(titanic_tree2, type = "class")
confusionMatrix(data=titanic_2_pred,
                reference = Titanicp$survived)$table
```

```
##           Reference
## Prediction died survived
##   died      682      161
##   survived  127      339
```

```r
confusionMatrix(data=titanic_2_pred,
                reference = Titanicp$survived)$overall[1]
```

```
##  Accuracy
## 0.7799847
```

Performance benchmarking

```
table(Titanicp$survived)
```

```
##
##     died survived
##      809      500
```

What if our prediction model was just that everyone died? The accuracy would be:

```
809/(809+500)
```

```
## [1] 0.618029
```

When considering performance, we should take into account that a "null" model might appear to give quite good performance when we have unbalanced group sizes.

Evaluating (out-of-sample) performance

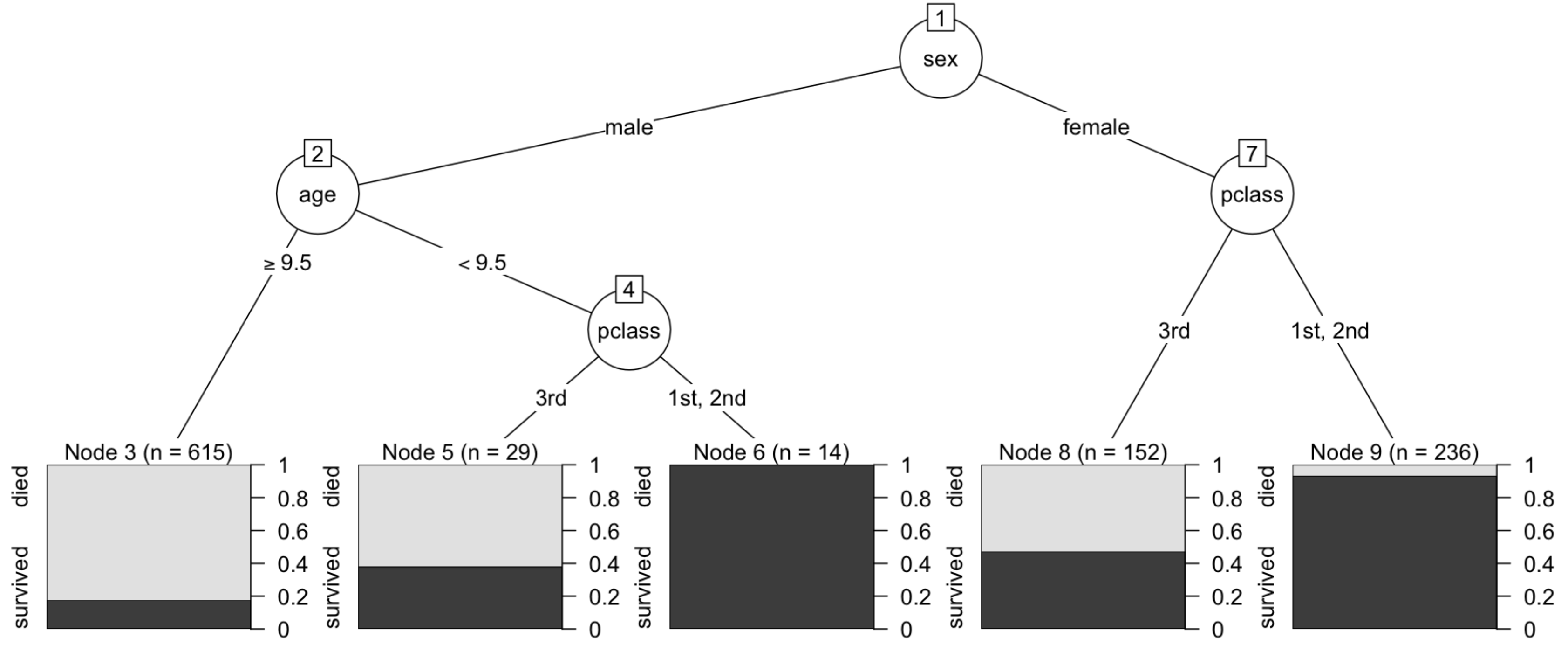
```r
titanic_complete = Titanicp %>% select(survived, sex, age, pclass) %>% drop_na()
train(survived ~ sex + age + pclass, data = titanic_complete,
      method = "rpart", trControl = trainControl(method = "cv", number = 10))
```

```
## CART
##
## 1046 samples
##    3 predictor
##    2 classes: 'died', 'survived'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 942, 941, 942, 941, 941, 941, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.01639344  0.7696021  0.5002840
##   0.01873536  0.7648311  0.4910097
##   0.45901639  0.6748002  0.2449885
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final value used for the model was cp = 0.01639344.
```

Final model

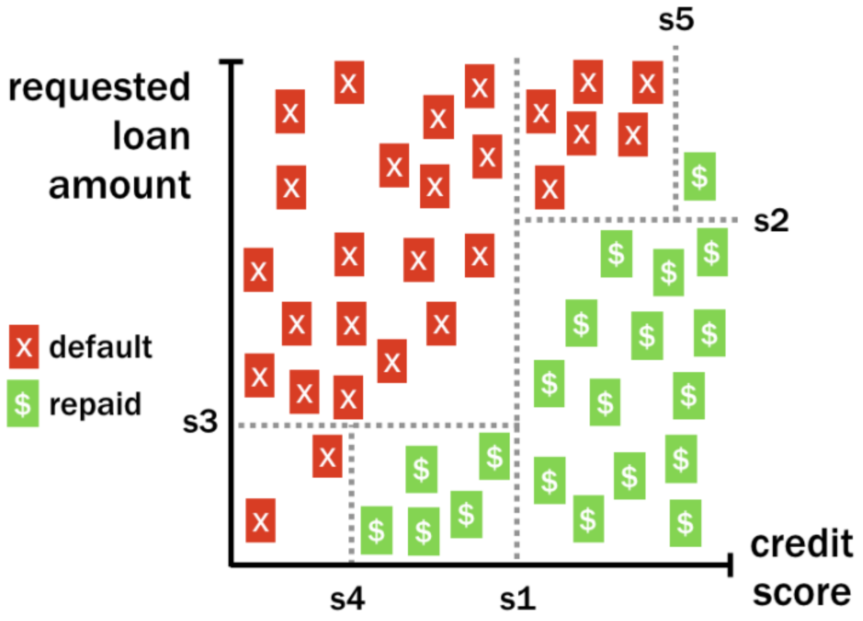
The CV procedure suggested 1.6% for the complexity parameter.

```
titanic_final = rpart(survived ~ sex + age + pclass, data = titanic_complete,
                      control = rpart.control(cp = 0.016))
plot(as.party(titanic_final))
```



Decision tree weaknesses

- Decision trees can become very complex very quickly - without a complexity penalty, it will happily continue until perfect classification (likely massively overfitting the data).
- The selected tree might be very sensitive to the complexity penalty
- Can only make decisions parallel to axes:



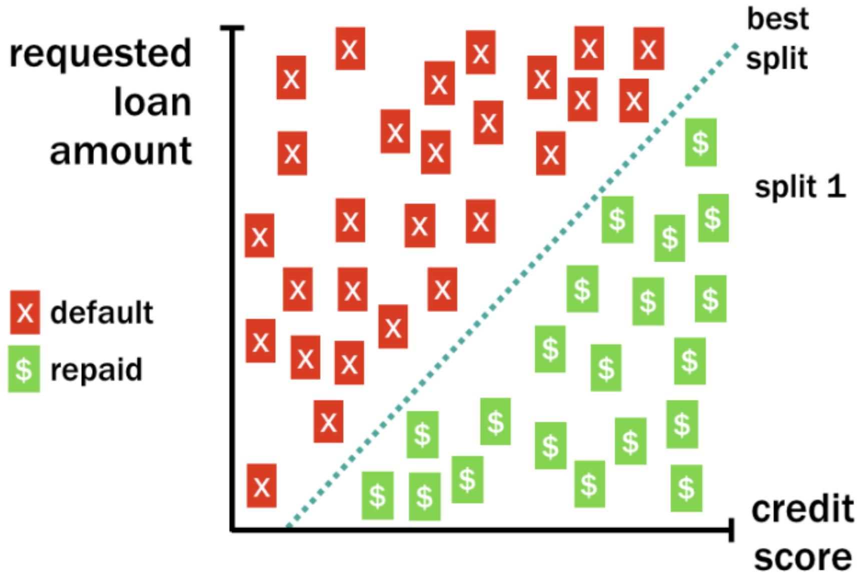
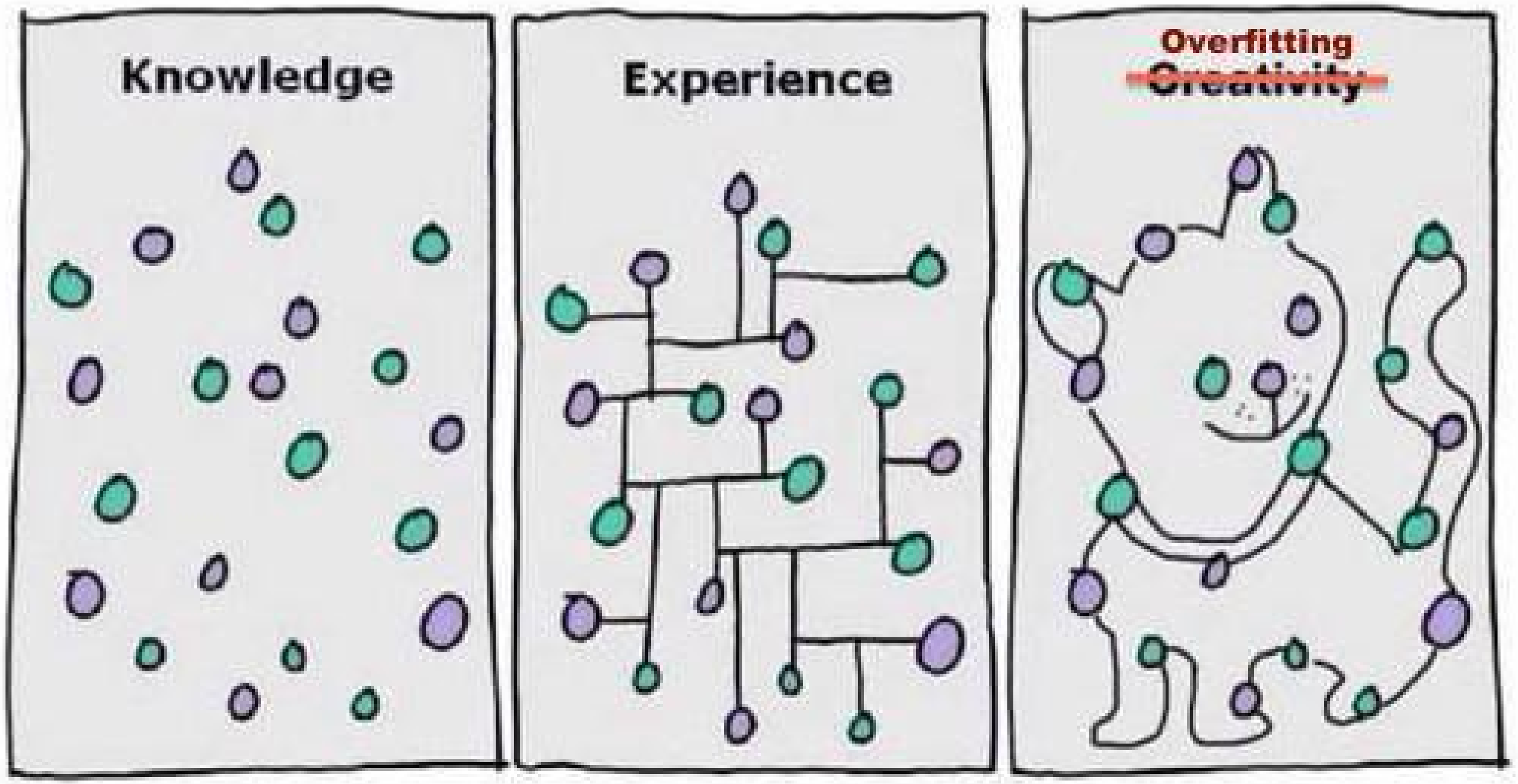


Image source: https://campus.datacamp.com/courses/supervised-learning-in-r-classification/chapter-4-classification-trees?ex=5

A single tree is prone to overfitting



When you **overfit** you're modelling noise rather than signal.

Random forests

Random forests

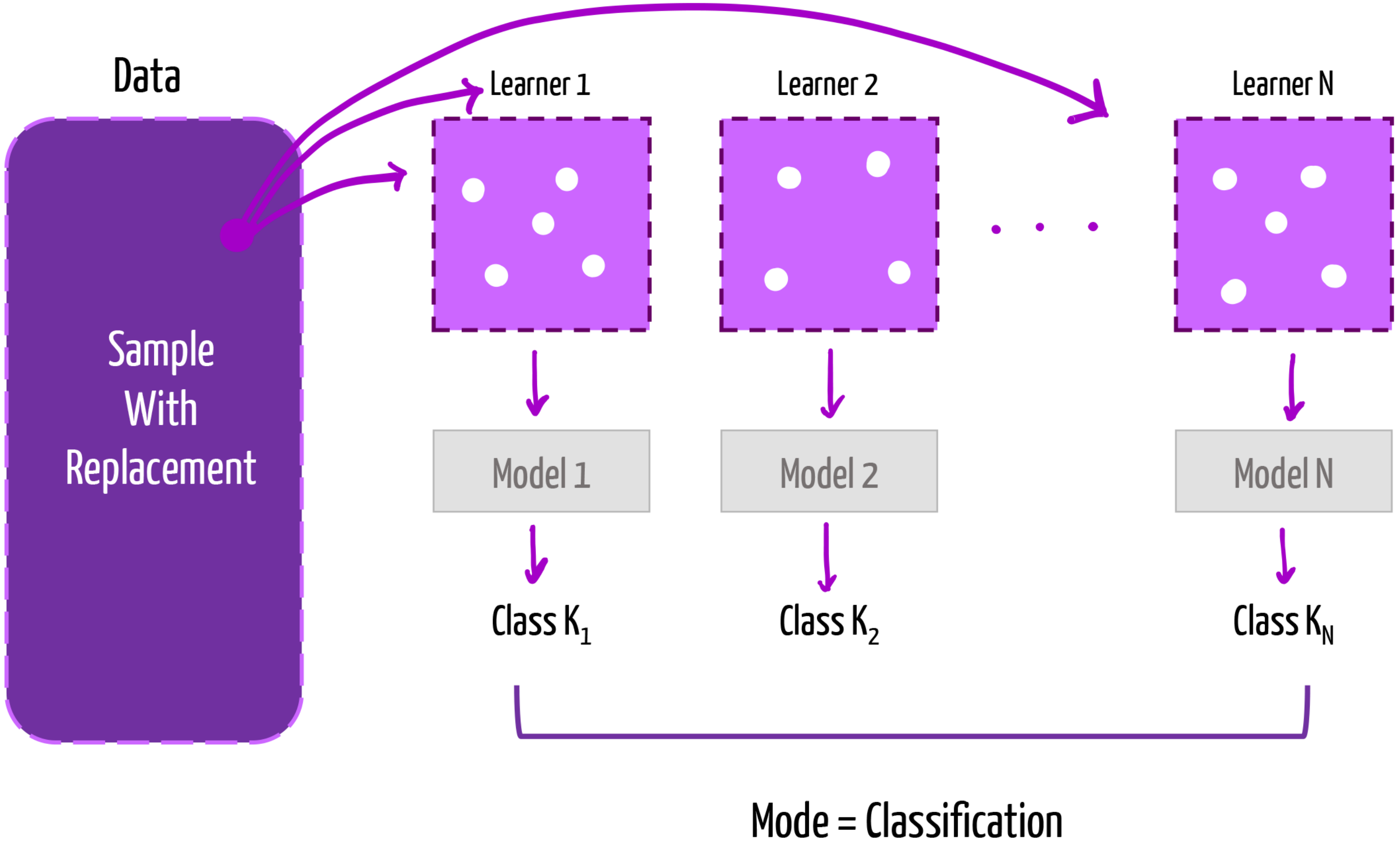
In a **random forest** we grow many trees. Each one learns from different (sub)samples of observations and different combinations of variables.

A random forest is constructed by:

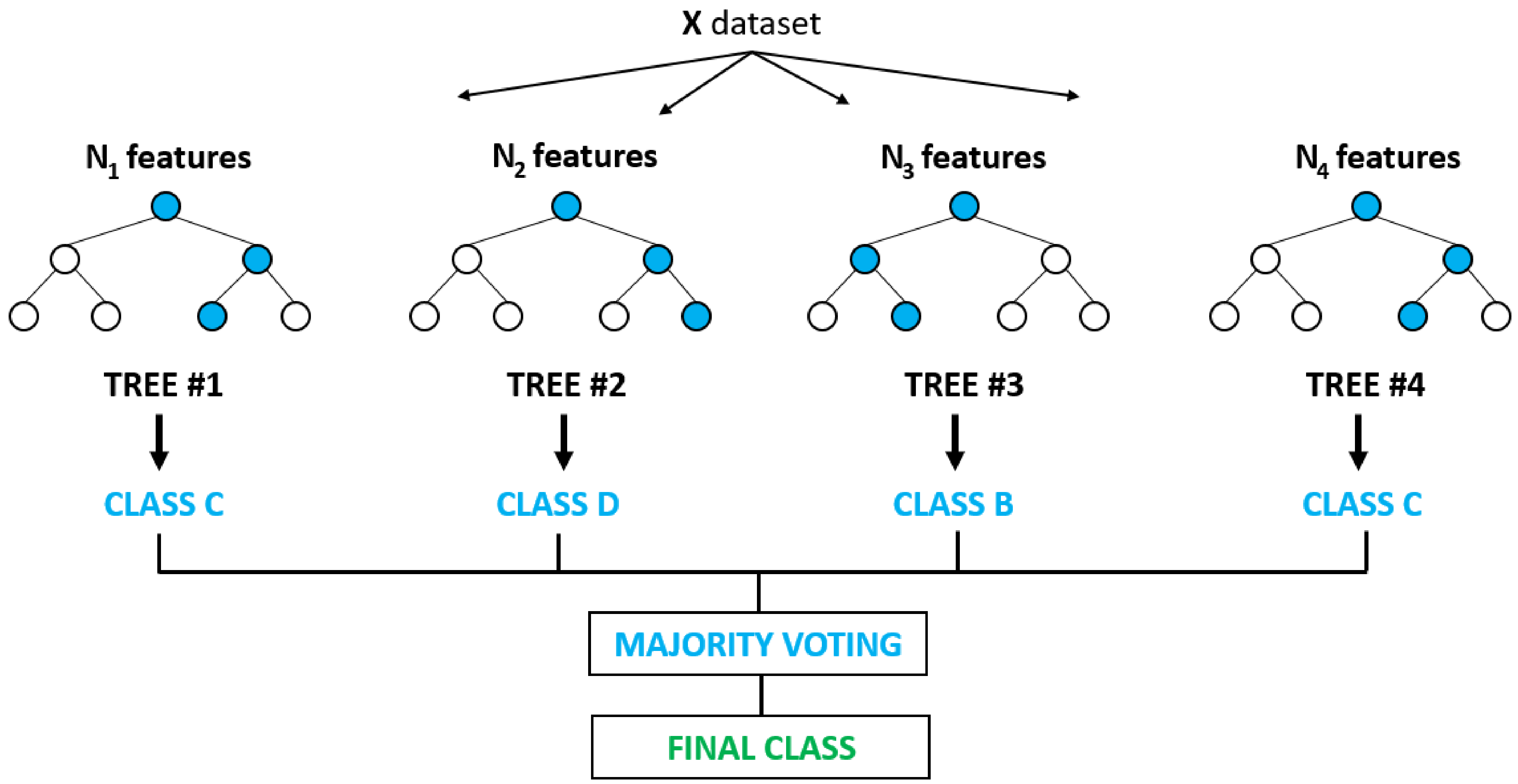
1. Choosing the number of decision trees to grow and the number of variables to consider in each tree.
2. Randomly selecting the rows of the data frame **with replacement**.
3. Randomly selecting the appropriate number of variables from the data frame.
4. Building a decision tree on the resulting data set.
5. Repeating this procedure a large number of times.
6. A prediction is be made by **majority rule**, i.e. running your new observation through all the trees in the forest and seeing which class is predicted most often.

- The `randomForest::randomForest()` function in R defaults to 500 trees each trained on `sqrt(p)` variables where `p` is the number of predictors in the full data set.

Ensemble learning: bagging (bootstrap aggregating)



Ensemble decision making



Source: GSS

randomForest in R

```r
library(randomForest)
iris_rf <- randomForest(Species ~ ., iris)
iris_rf
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = iris)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 5.33%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          5        45        0.10
```

Using the `randomForest()` function, we can train our ensemble learning using the same formula we passed to `rpart`.

`randomForest` in R

```r
predict(iris_rf, new_data)
```

```
##      1      2
## setosa setosa
## Levels: setosa versicolor virginica
```

Again, we can use the same `new_data` values in the `predict` function as before to reach the same prediction we did with `rpart`.

Titanic random forest

```r
titanic_rf = randomForest(survived ~ sex + age + pclass, titanic_complete)
titanic_rf
```

```
## 
## Call:
##  randomForest(formula = survived ~ sex + age + pclass, data = titanic_complete) 
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
## 
##         OOB estimate of  error rate: 20.75%
## Confusion matrix:
##          died survived class.error
## died      567       52  0.08400646
## survived  165      262  0.38641686
```

```r
importance(titanic_rf)
```

```
##        MeanDecreaseGini
## sex           122.29587
## age            21.97414
## pclass         40.82804
```

Note: OOB is short for: Out-of-bag

For more details on decision trees see Hastie, Tibshirani, and Friedman (2009; section 9.2) and James, Witten, Hastie, et al. (2017; chapter 8).

References

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning*. 2nd ed. Springer Series in Statistics. New York, NY, USA: Springer. URL: https://web.stanford.edu/~hastie/ElemStatLearn/.

James, G., D. Witten, T. Hastie, and R. Tibshirani (2017). *An Introduction to Statistical Learning: With Applications in R*. New York: Springer. URL: https://www-bcf.usc.edu/~gareth/ISL/.

Jed Wing, M. K. C. from, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team, M. Benesty, et al. (2018). *caret: Classification and Regression Training*. R package version 6.0-80. URL: https://CRAN.R-project.org/package=caret.

Liaw, A. and M. Wiener (2002). "Classification and Regression by randomForest". In: *R News* 2.3, pp. 18-22. URL: https://CRAN.R-project.org/doc/Rnews/.

Therneau, T. and B. Atkinson (2018). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-13. URL: https://CRAN.R-project.org/package=rpart.

Acknowledgements

Special thanks to Sarah Romanes for providing content inspiration from her R Ladies talks on machine learning:

- sarahromanes.github.io/r-ladies-ML-1
- sarahromanes.github.io/r-ladies-ML-2