Classification trees and RandomForest

浙江大学爱丁堡大学联合学院
ZJU-UoE Institute

Tim Gorman, Flickr, CC-BY-ND-2.0

Nicola Romanò
nicola.romano@ed.ac.uk

Learning objectives

At the end of this lecture you should be able to:

- Explain the usage of binary trees as a classification method
- Explain how RandomForests improves on binary tree classification
- Explain the concept of *bagging* and how can it improve classification

- R workshop #6 will show you how to use these techniques more in detail

Binary trees

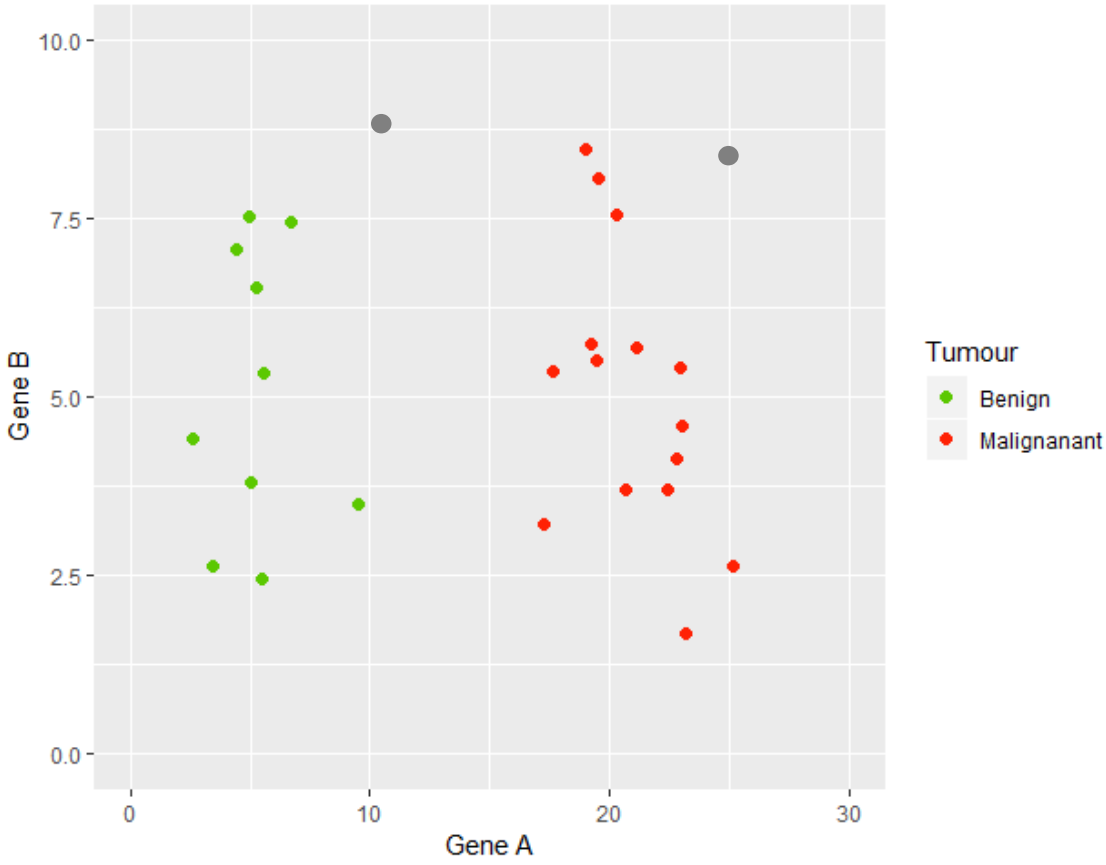
One of the simplest method of classification

Can be used to predict binary data (classification tree) or continuous data (regression tree)

Altogether known as **CART** (Classification And Regression Trees) (Breiman et al. 1984)

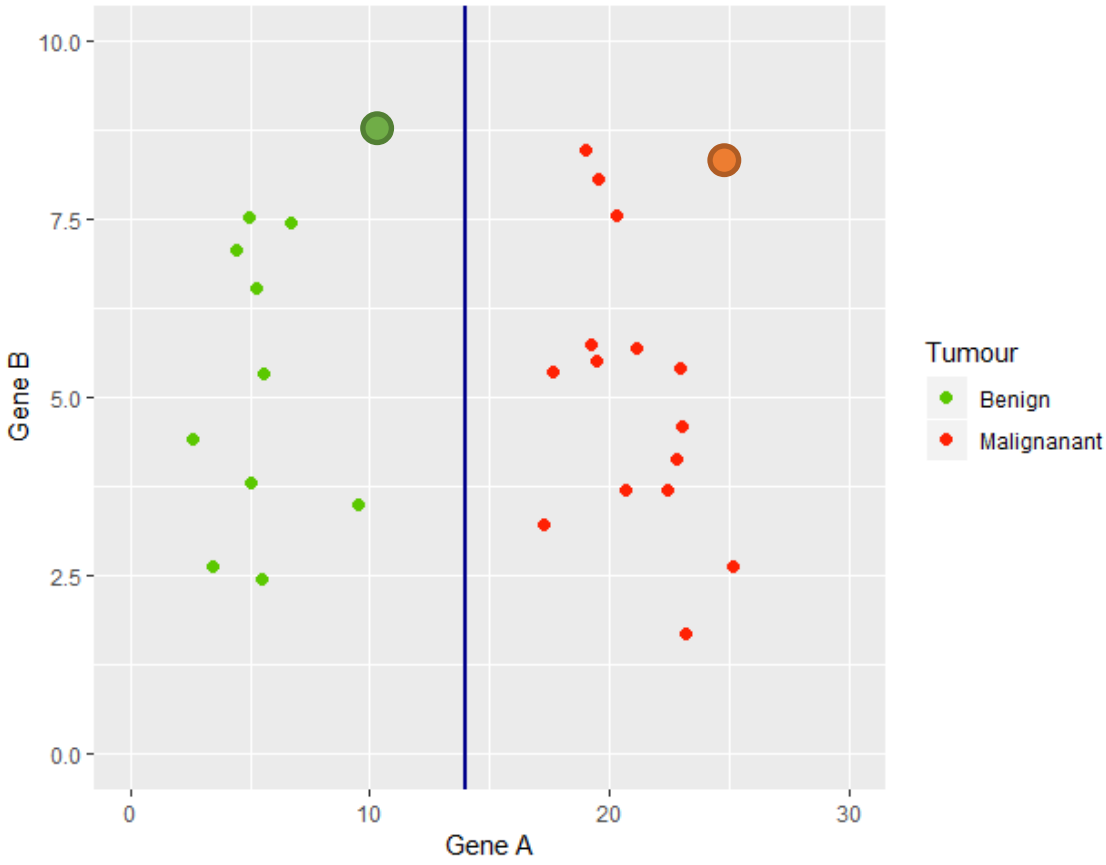
Work by partitioning data into two subsets, in a recursive way, until all data is classified.

A simple example



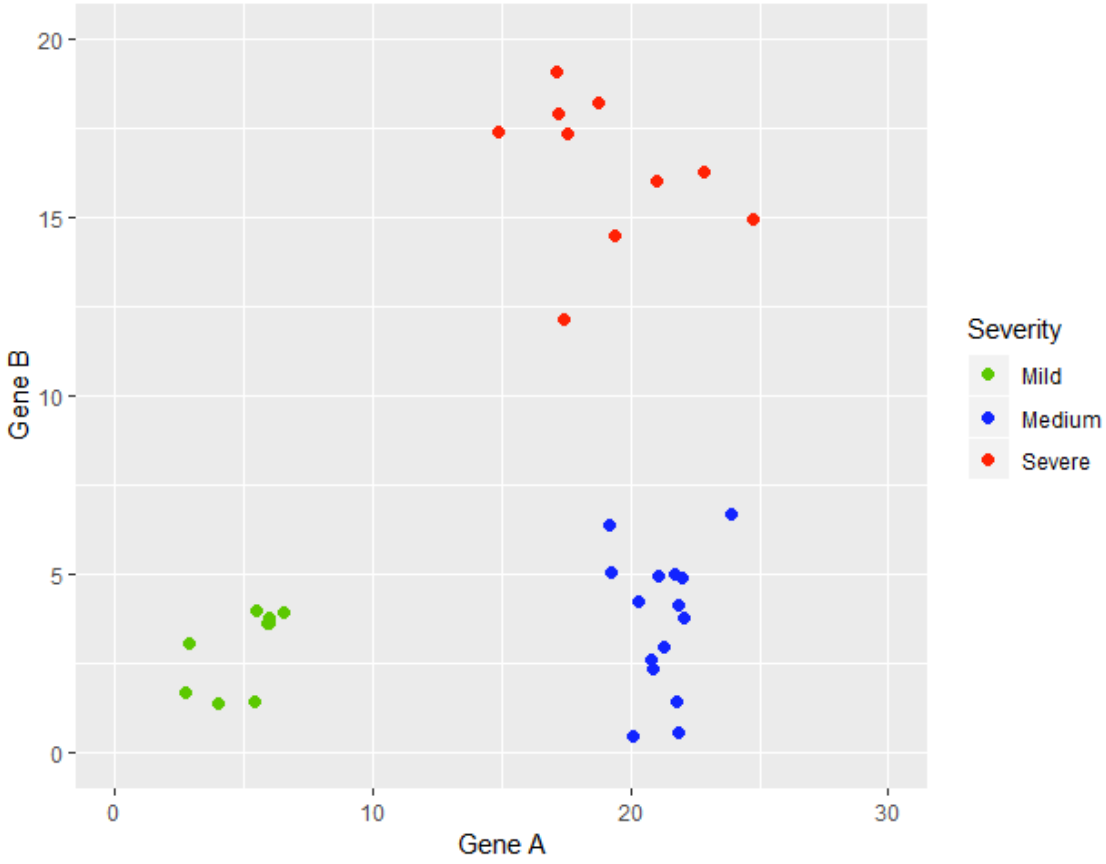
What is the easiest way to classify benign vs malignant tumours from levels of A and B?

A simple example



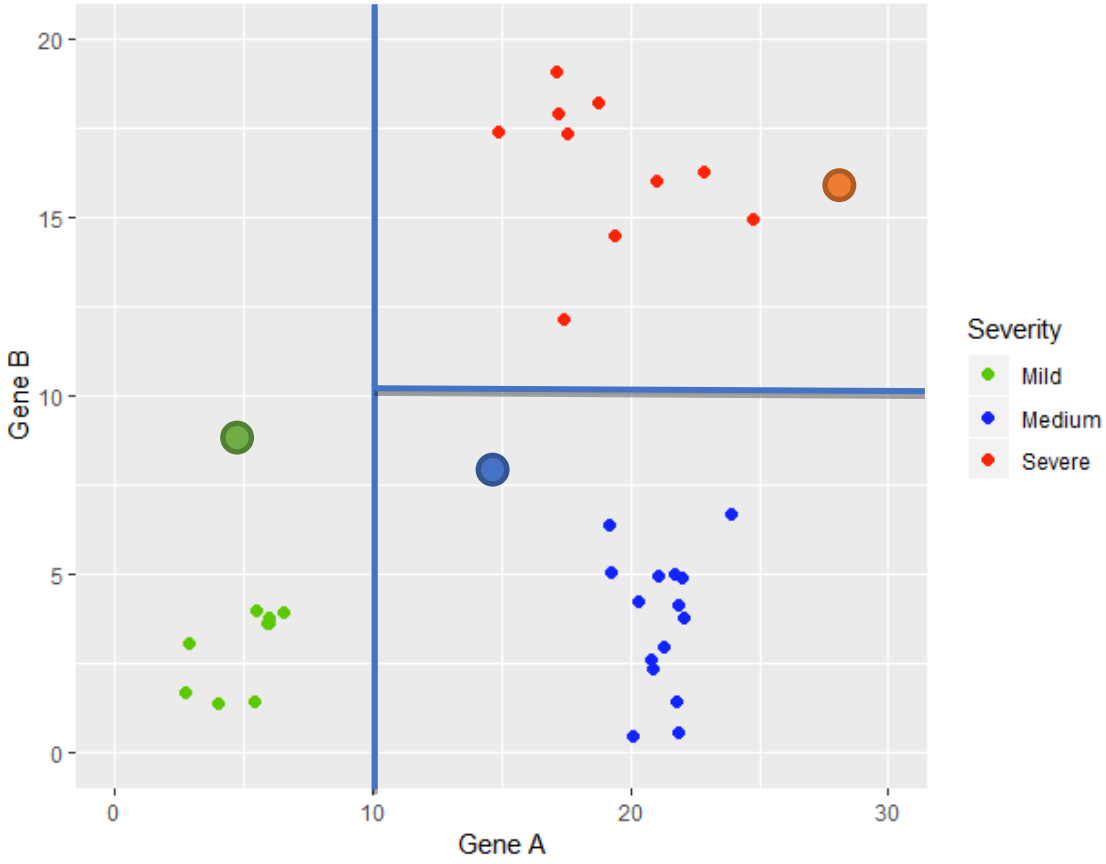
If A >= 14 → malignant
If A < 14 → benign

A simple example



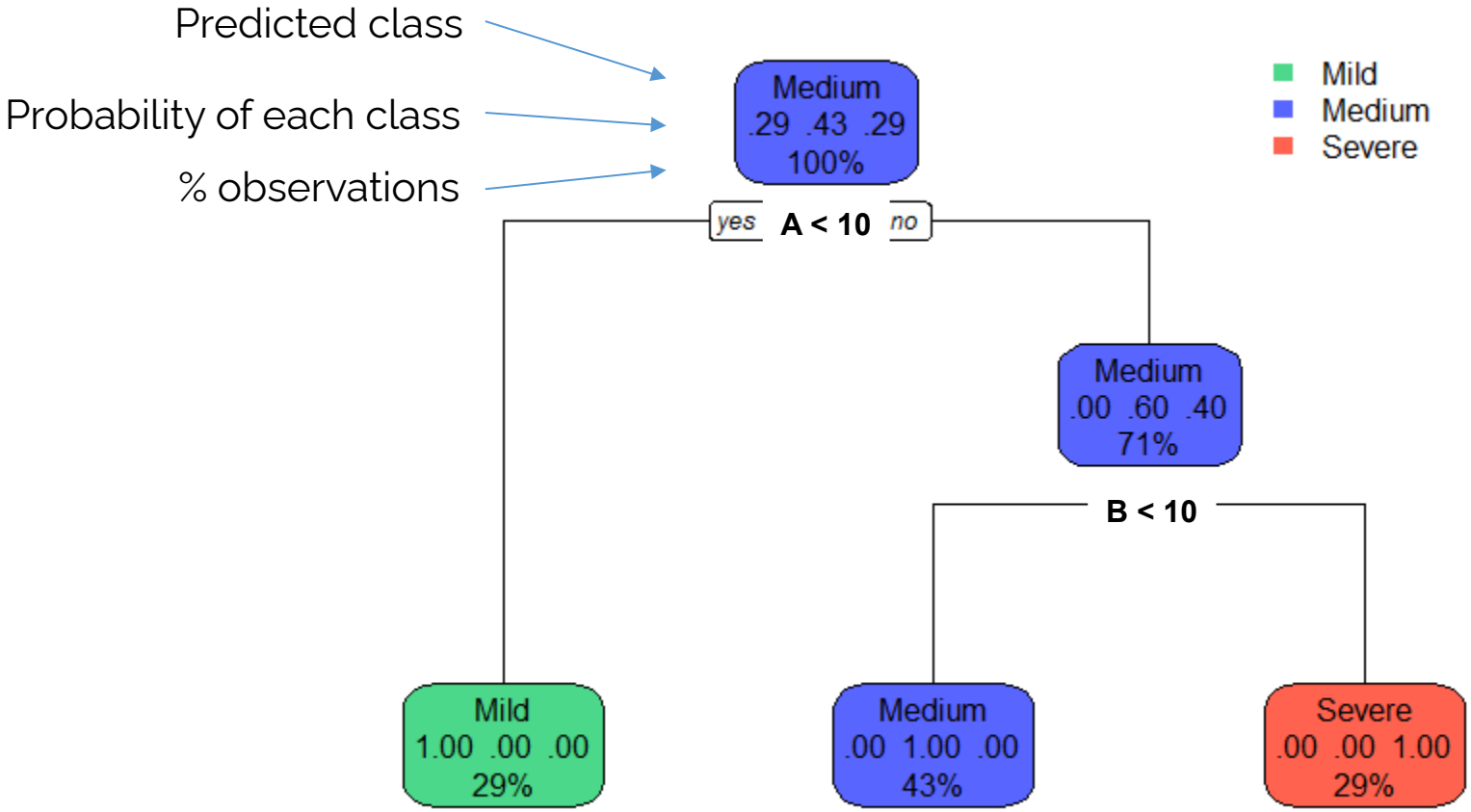
What is the easiest way to classify the severity of the disease?

A simple example



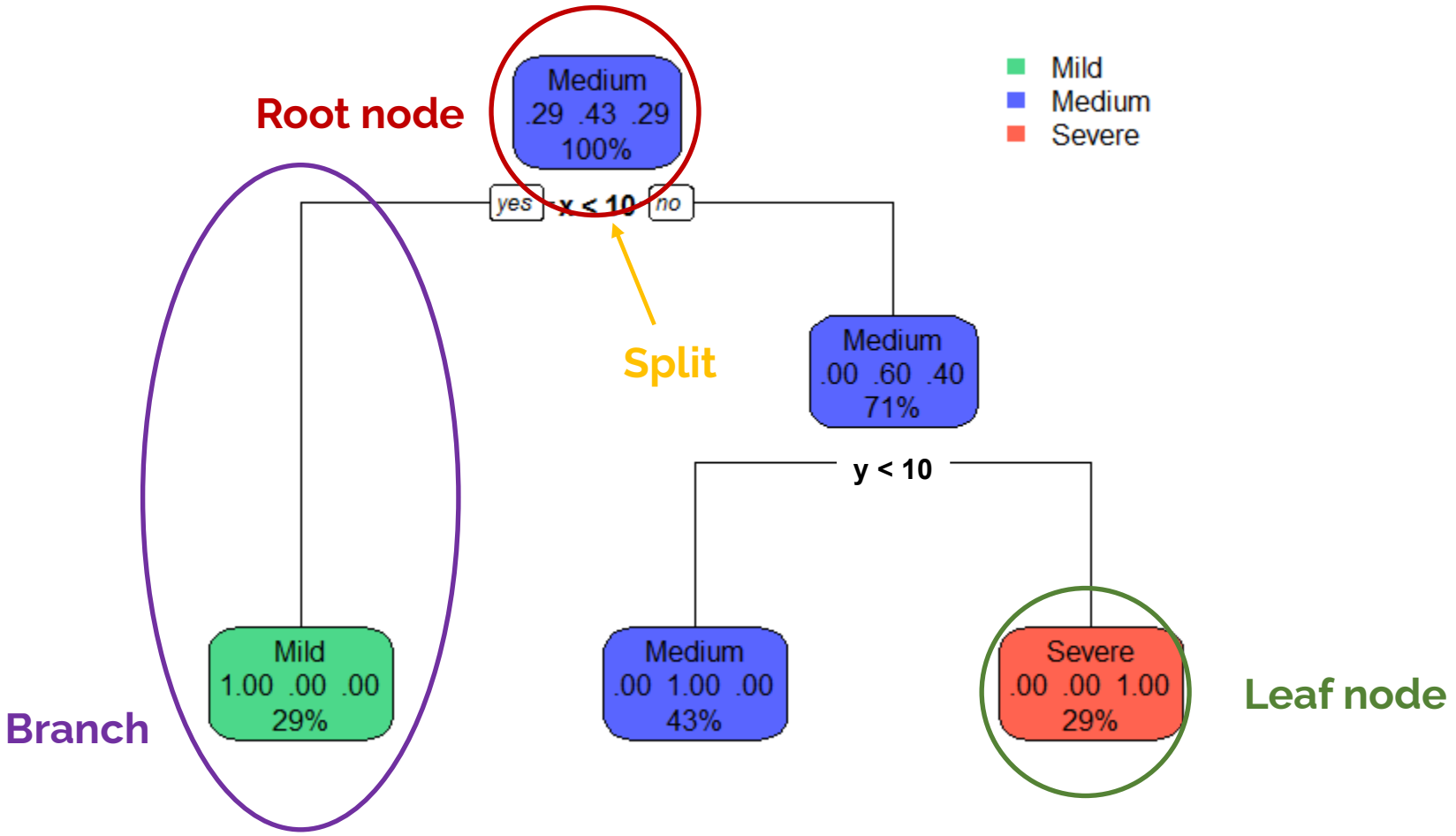
If A < 10 → mild
If A >= 10
    If B < 10 → medium
    If B >= 10 → severe

Binary trees



If A < 10 → mild
If A >= 10
    If B < 10 → medium
    If B >= 10 → severe

Some nomenclature

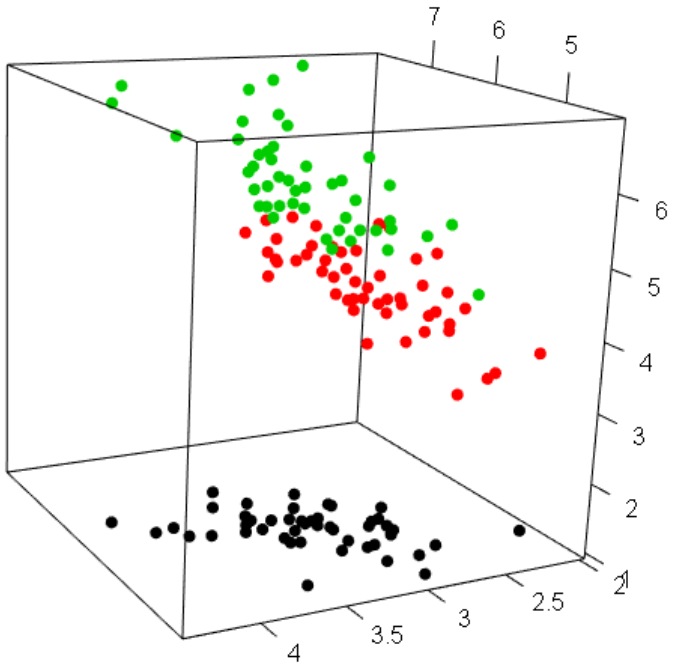


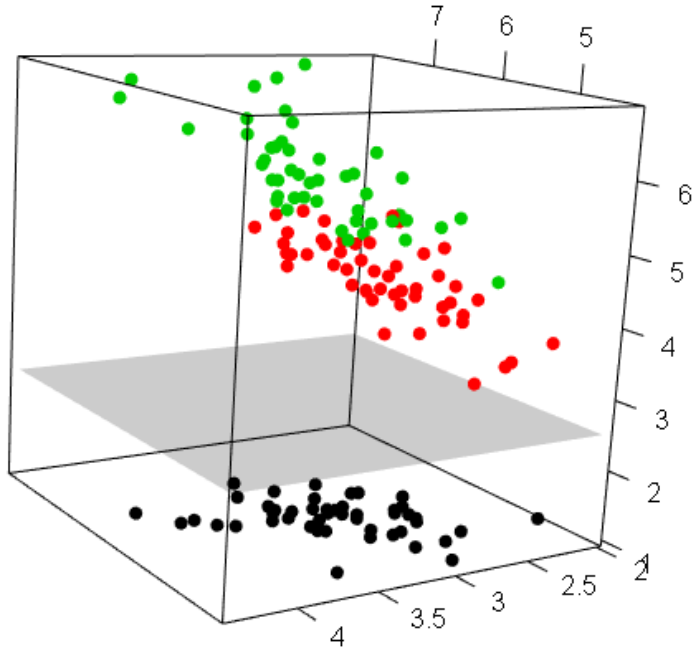
If x < 10 → mild
If x >= 10
    If y < 10 → medium
    If y >= 10 → severe

More complex/real situations

What if we have multiple descriptors?

Our binary splits will become (hyper)planes



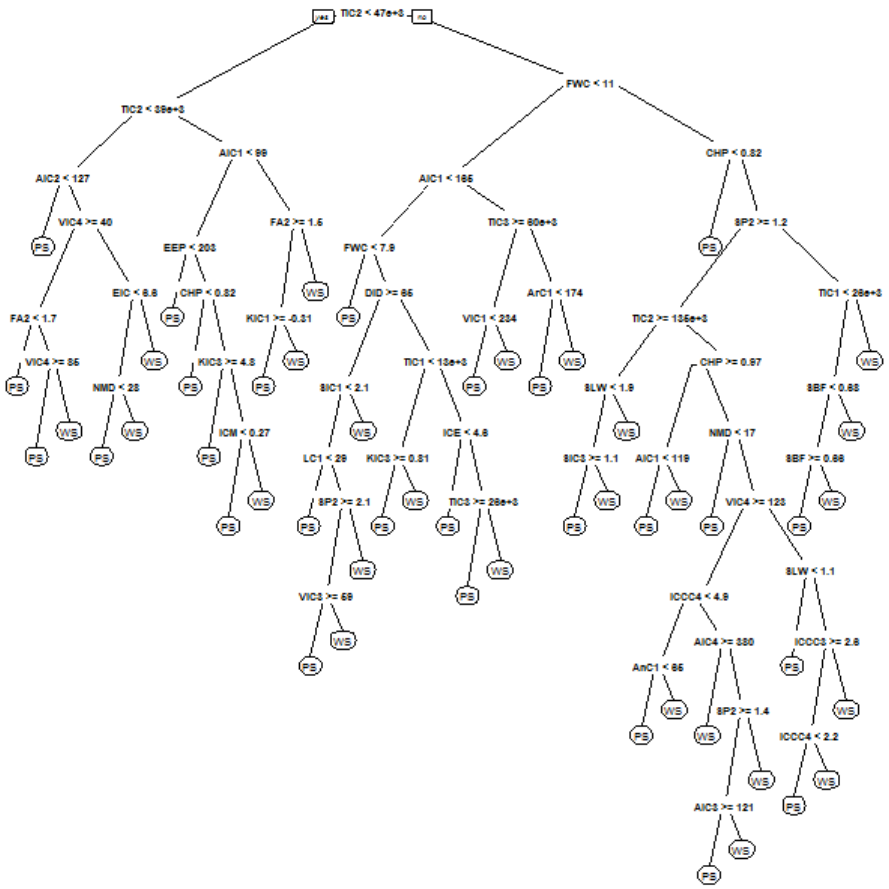


Creating a binary tree in R

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

# Generate the model
# Just pass a formula and the dataset!
model <- rpart(Output ~ Predictor1 + Predictor2, data = mydata)

# Plots the model
rpart.plot(model)
```

Pruning

Sometimes the resulting tree can be too complex, with the risk of overfitting
Pruning allows to “cut back” some splits and simplify the tree, generally improving prediction



How does rpart choose a split?

The **r**ecursive **part**itioning approach

1. Find the variable which best splits the data

2. Split the data on that variable and repeat the process for each branch

3. Stop when:

    a) The subgroups reach a minimum size
    or
    b) No improvements can be made
    This depends on the cp parameter that is related to the probability of increasing misclassification if making a new split

4. Prune back the tree to improve classification rate

Pruning

Two main ways of pruning an rpart tree

- Specifying the control parameter in rpart
  - Can control several aspects of how the tree is generated
  - minsplit → minimum number of observations to try and split a node
  - minbucket → minimum number of observation in a leaf
  - cp → complexity parameter
  - … others left to you to find out!

```
# Minimum 10 elements for a split
my.tree <- rpart(Output ~ .,
                 control = rpart.control(minsplit = 10))

# Minimum 10 elements for a split
# Minimum 3 elements in each leaf
my.tree2 <- rpart(Output ~ .,
                              control = rpart.control(minsplit = 10,
                                                      minbucket = 3))
```

- Using the prune function on an existing tree. Needs a complexity parameter cp

```
my.tree.pruned <- prune(my.tree, cp = 0.1)
```

Predicting risk of kyphosis

81 children with corrective spinal surgery

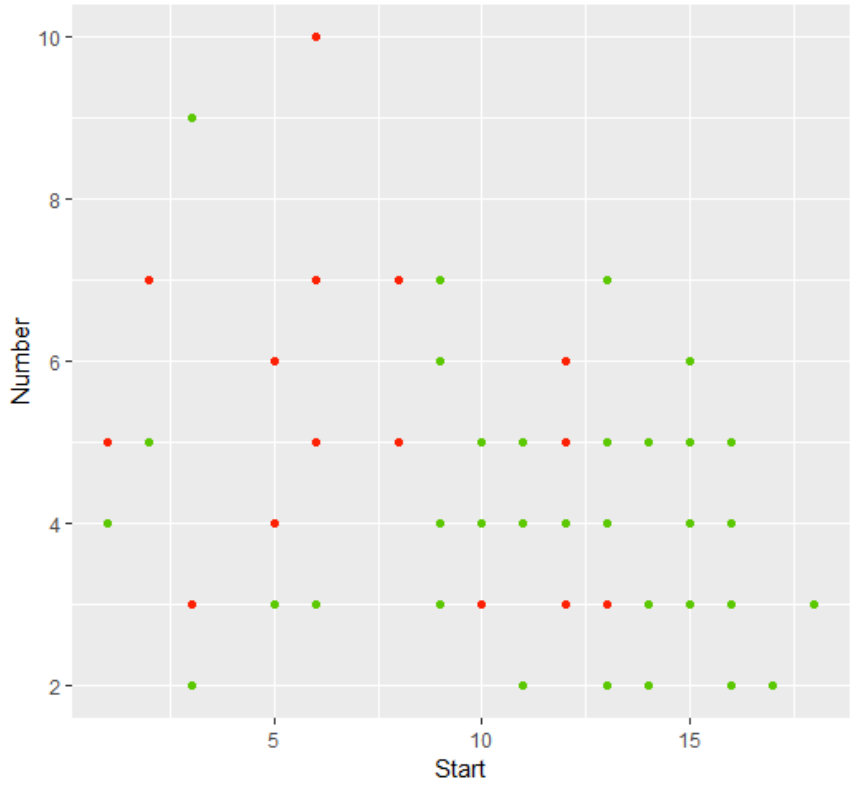
Kyphosis (excessive convex spine curvature) is a possible side effect of corrective surgery

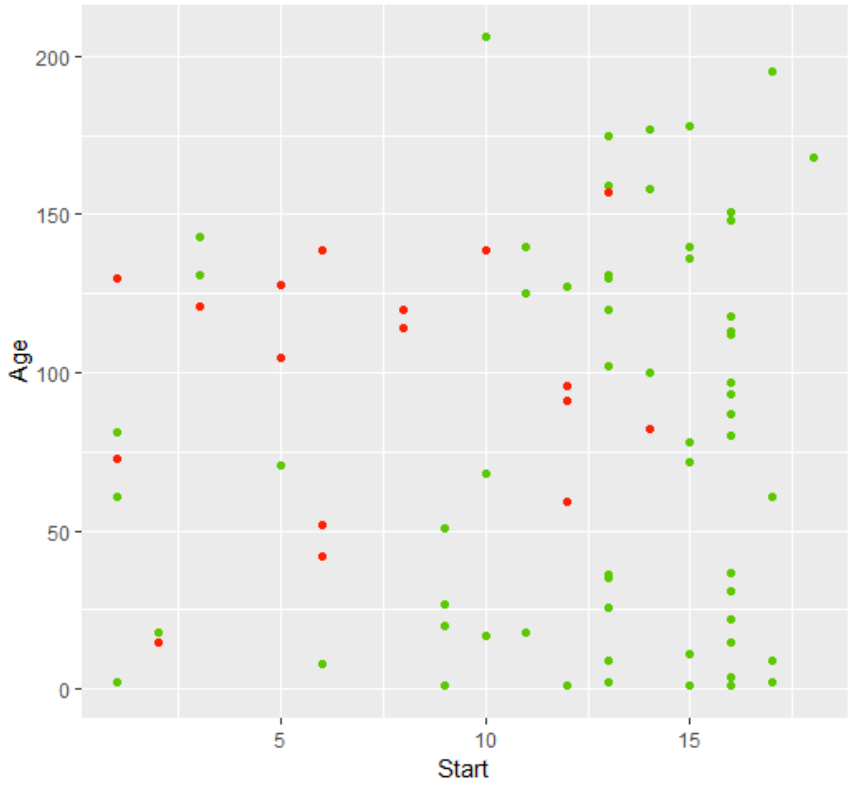
Can we predict the risk of kyphosis?

Dataset from 81 patients

- **Age** (in months) at time of surgery
- **Number** of vertebrae involved in the surgery
- **Start**: first vertebra operated on

A quick look at the data





Difficult to manually define a split (generally the case with real-life data!)

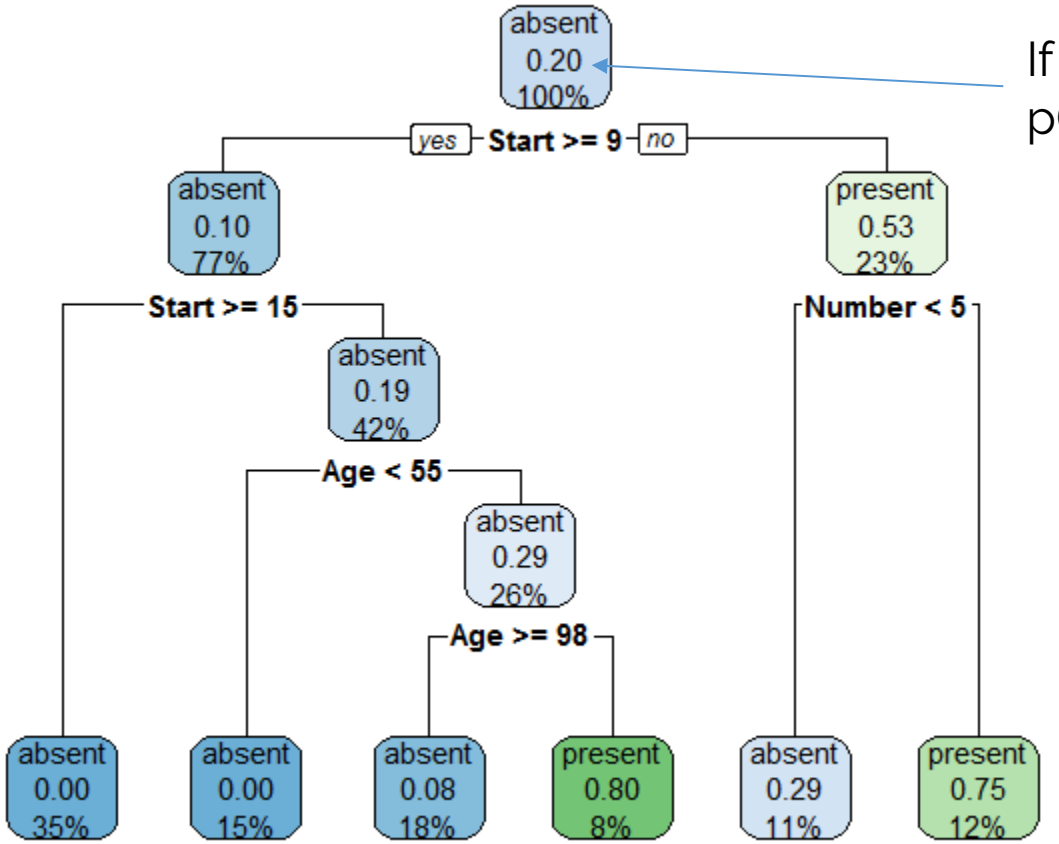
Creating a training set

```
# Get 65 numbers from 1 to 81 (the number of patients)

training.id <- sample(1:81, size = 65)

# Subset our data into a training set and a test set
training <- kyphosis[training.id,]
test <- kyphosis[-training.id,]
```

We can now train our tree on the training set and test it on the test set.

Creating the tree

```
model <- rpart(Kyphosis ~ ., data = training,
               control = rpart.control(minsplit = 10, minbucket = 4))
rpart.plot(model)
```



If outcome is binary p(Yes) is shown

Exploring the results

Confusion matrix – training set

```
table(actual = training$Kyphosis,
      pred = predict(model, training, type = "class"))

         pred
actual    absent present
  absent      49       3
  present      3      10
```

~90% correctly classified

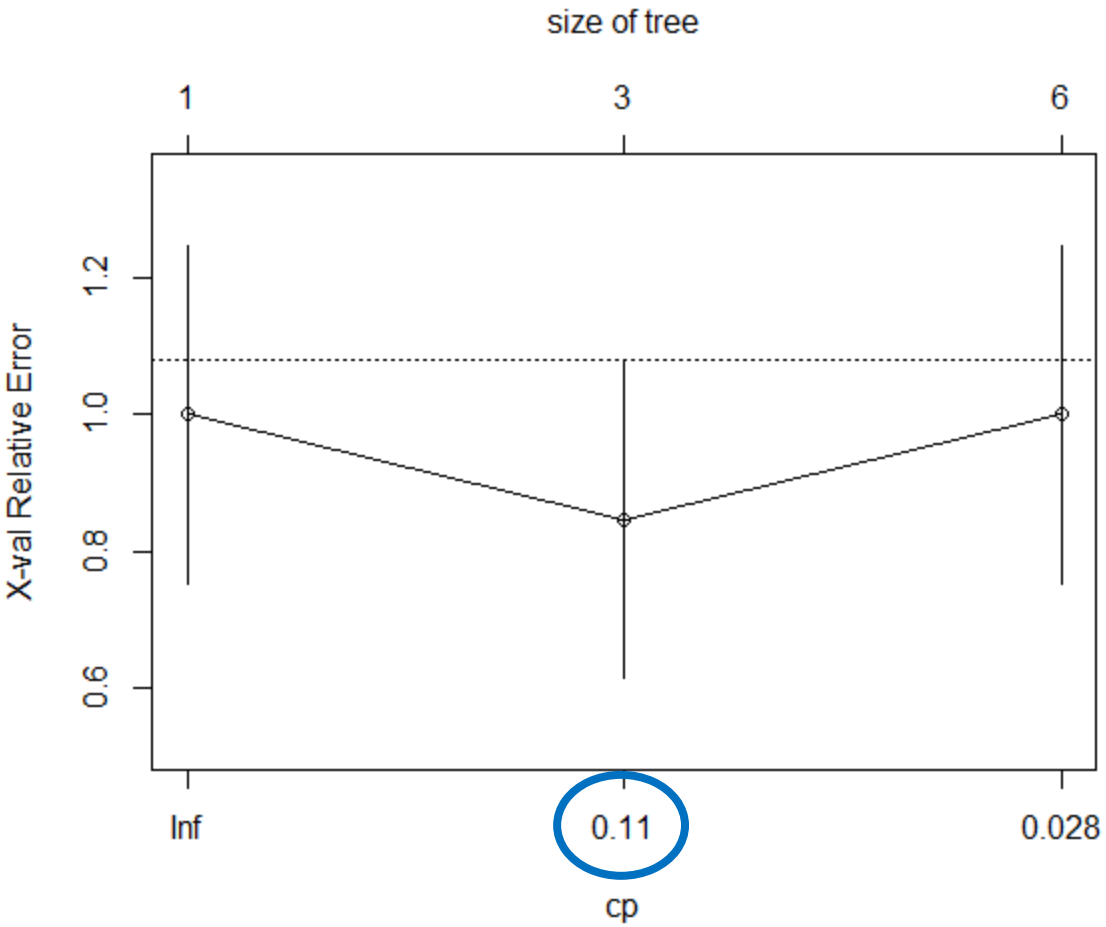
Confusion matrix – test set

```
table(actual = test$Kyphosis,
         pred = predict(model, test, type = "class"))

         pred
actual    absent present
  absent      11       1
  present      2       2
```

~81% correctly classified

The complexity value

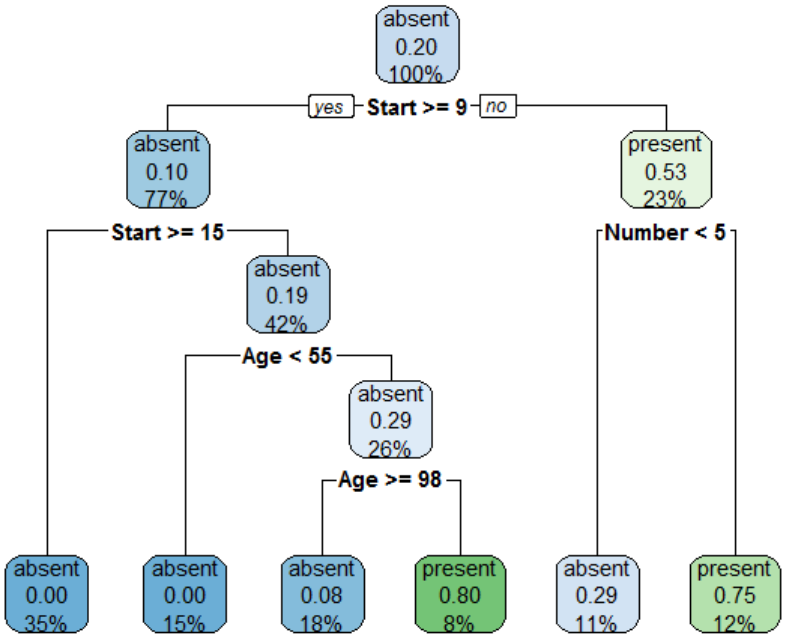
```
plotcp(model)
```



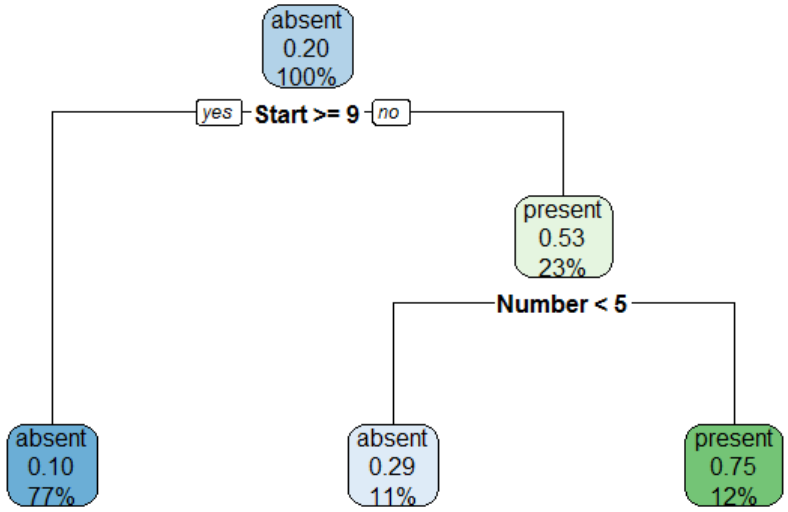
Pruning back the tree

```
model.pruned <- prune(model, 0.11)
```

Original tree



Pruned tree

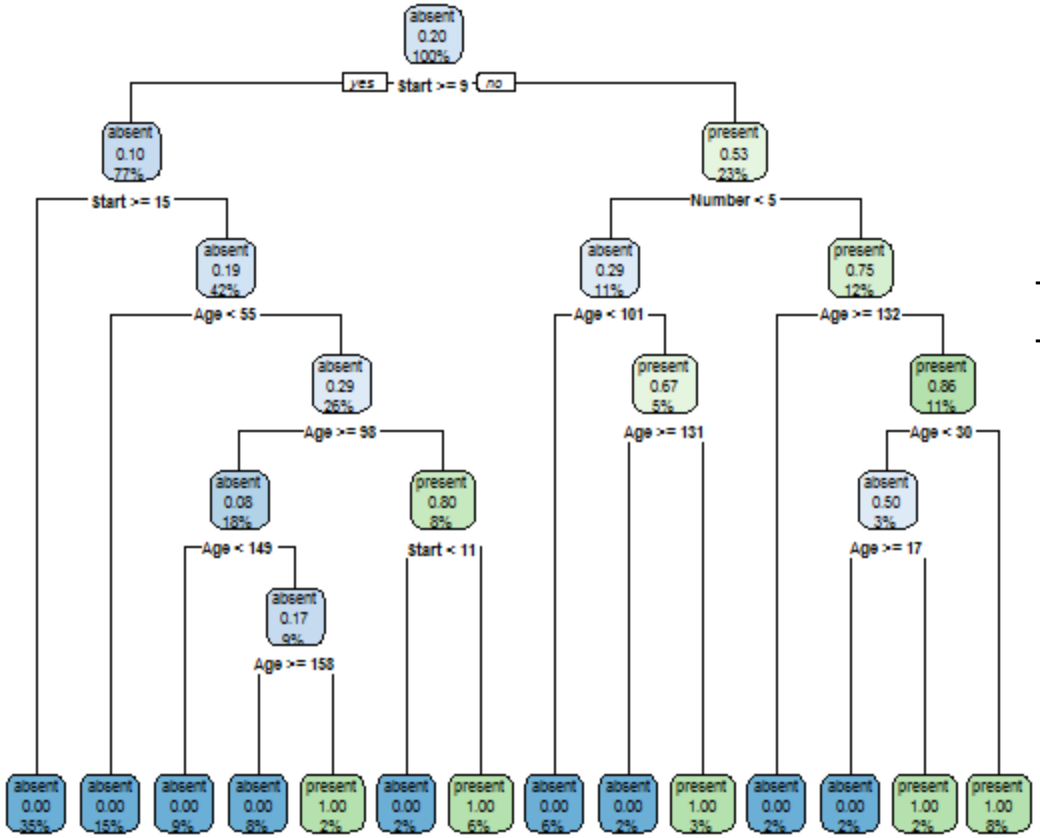


~81% accuracy in both trees for the test set

The problem with overfitting

In theory I could have perfect prediction on the training set by doing:

```
overfitted.model <- rpart(Kyphosis ~ ., data = training,
                          control = rpart.control(minsplit = 1,
                                                  minbucket = 1))
```



Training: 100% correctly classified
Test: 62.5% correctly classified

OVERFITTING!

The problem with ~~overfitting~~ binary trees

Binary classification trees are

- Easy to build
- Easy to interpret
- Easy to use

- **NOT ACCURATE**

The solution - Random Forests

The basic idea: bootstrap binary trees!

Bagging (**b**ootstrap **agg**regat**ing**)

Random Forest combines the simplicity of binary tree with high accuracy!

We create a large number of trees from a bootstrapped sample of the data.

We can now hold a “vote” between all the trees to predict our new data (aggregating)

Step 1 – create a bootstrap sample

Real data

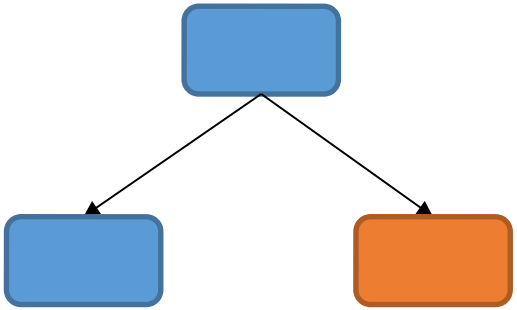
Age	Sex	HR	Chol	Angina type	Chest pain	Heart disease
34	M	118	182	0	3	1
51	M	140	299	1	2	0
58	F	130	197	0	0	1
59	M	160	273	0	3	0
35	F	138	183	2	1	1
43	M	150	247	0	0	1

Bootstrapped data (1 possible sample)

Age	Sex	HR	Chol	Angina type	Chest pain	Heart disease
43	M	150	247	0	0	1
59	M	160	273	0	3	0
34	M	118	182	0	3	1
35	F	138	183	2	1	1
43	M	150	247	0	0	1

Step 2- create a tree

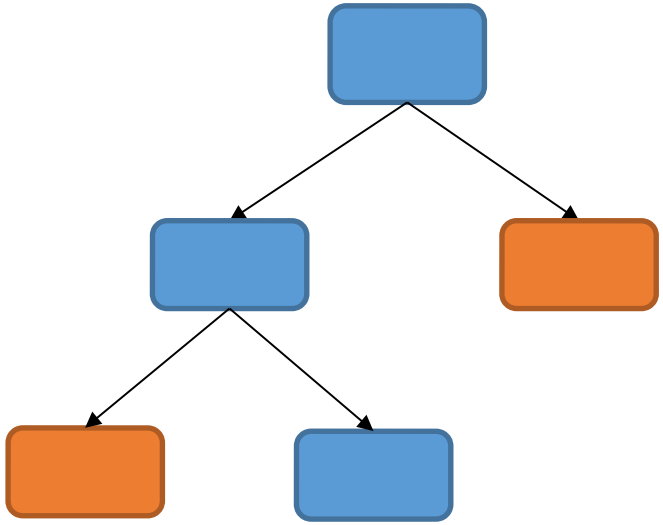
We create a tree only using a **random subset of classifiers** at each step



Age	Sex	HR	Chol	Angina type	Chest pain

Step 2- create a tree

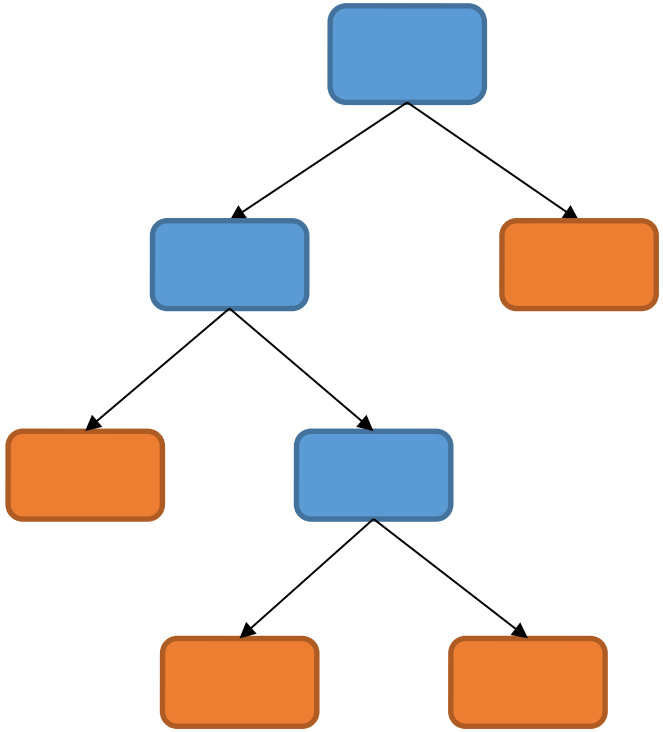
We create a tree only using a **random subset of classifiers** at each step



Age	Sex	HR	Chol	Angina type	Chest pain

Step 2- create a tree

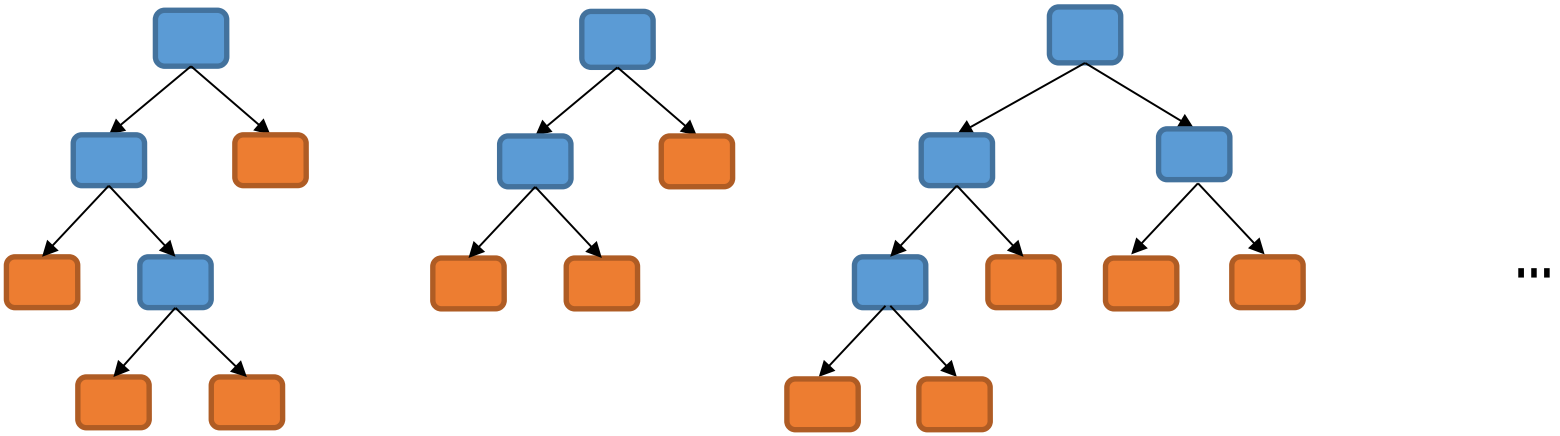
We create a tree only using a **random subset of classifiers** at each step



Age	Sex	HR	Chol	Angina type	Chest pain

We have a (random) forest!

Now repeat with a lot (hundreds) of bootstrap samples



Now vote!

New data

Age	Sex	HR	Chol	Angina type	Chest pain
41	F	135	180	1	0

Tree	Pred
1	1
2	0
3	0
…	…
200	0

YES: 22
NO: 178

By majority vote we say that the patient is not at risk

Accuracy of the Random Forest

For each tree, some of the data is not included in the bootstrap sample

This is called **out-of-bag (OOB) sample**

We can use these data to test each tree

The OOB error is the % of OOB observation that is misclassified

We can test the forest with different numbers of variable at each split and see which one has the least OOB error!

Summary

- Classification trees are one of the simplest classifiers
- Very easy to use but prone to overfitting
- Bagging (bootstrap aggregating) can help reduce overfitting
- Random Forests employs bagging on regression trees, and allow improved classificatior