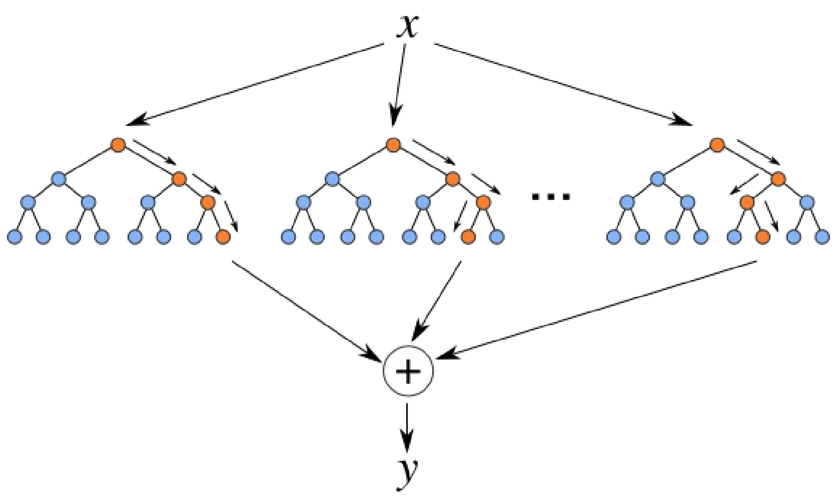
R-course:
Machine Learning using R

Random Forest

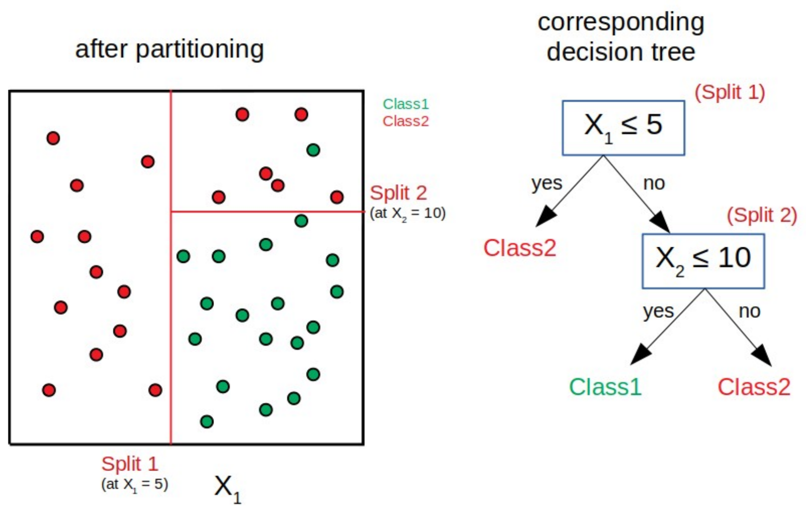


Yannick Rothacher

Zürich, 2021

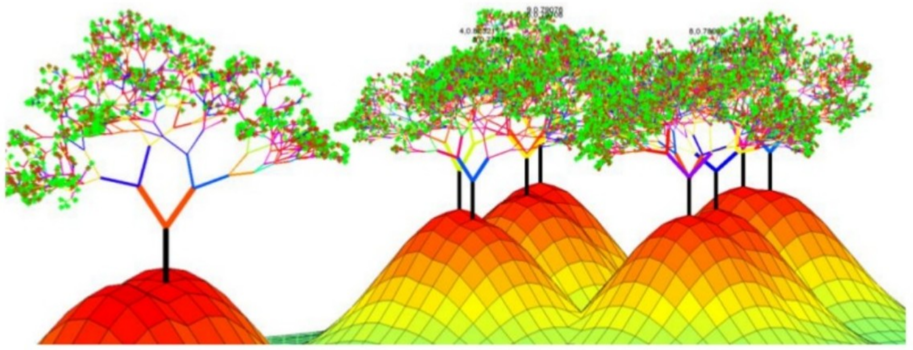
Recap: Decision trees

- Last lecture we saw how decision trees can be used for **classification** and **regression**
- Looked at the different methods for **variable** and **split selection** in rpart() and ctree() function
- One advantage of decision trees is their **good interpretability**
- **However**: If trees are allowed to grow deep they show large variability and low bias (overfitting the data)
- What can be done to prevent overfitting?



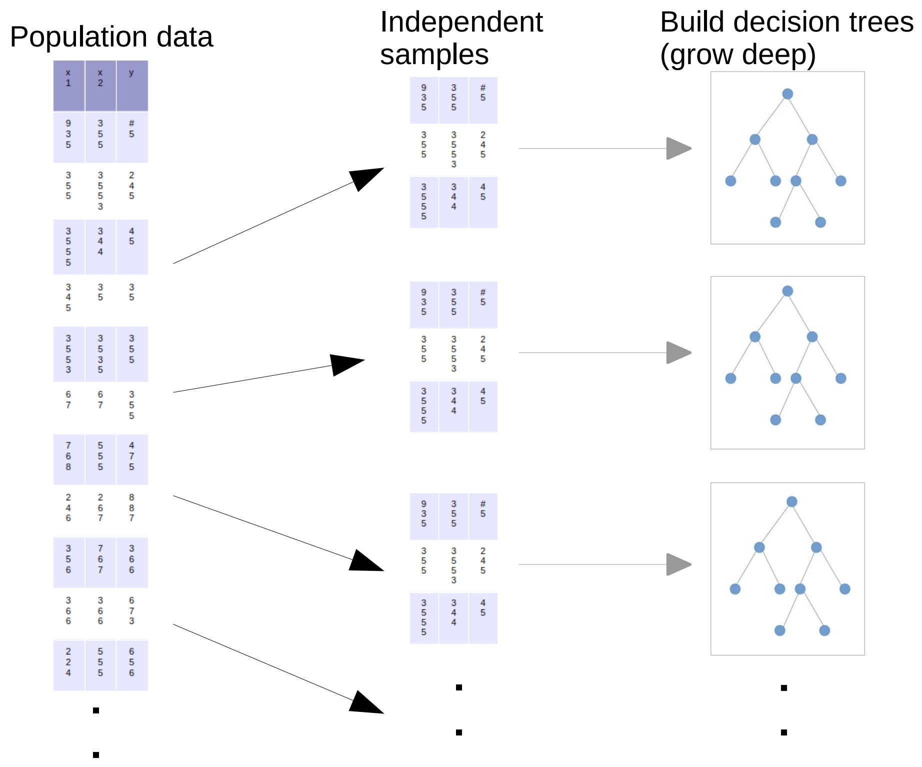
Ensemble methods

- Why not combine multiple classifiers to improve prediction?
- Given some training data and a test-observation:
  - You could apply multiple, different classifiers to the data (logistic regression, **KNN**, **decision tree**, neural network, …) and incorporate each classifier's prediction into your final prediction of the test observation (e.g. majority vote)
- **Ensemble methods** are based on a similar intuition: Combine multiple (simple) classifiers of the same type to improve performance
- With regard to decision trees there are two ensemble methods, which are mostly applied:
  - Bagging
  - Random Forest



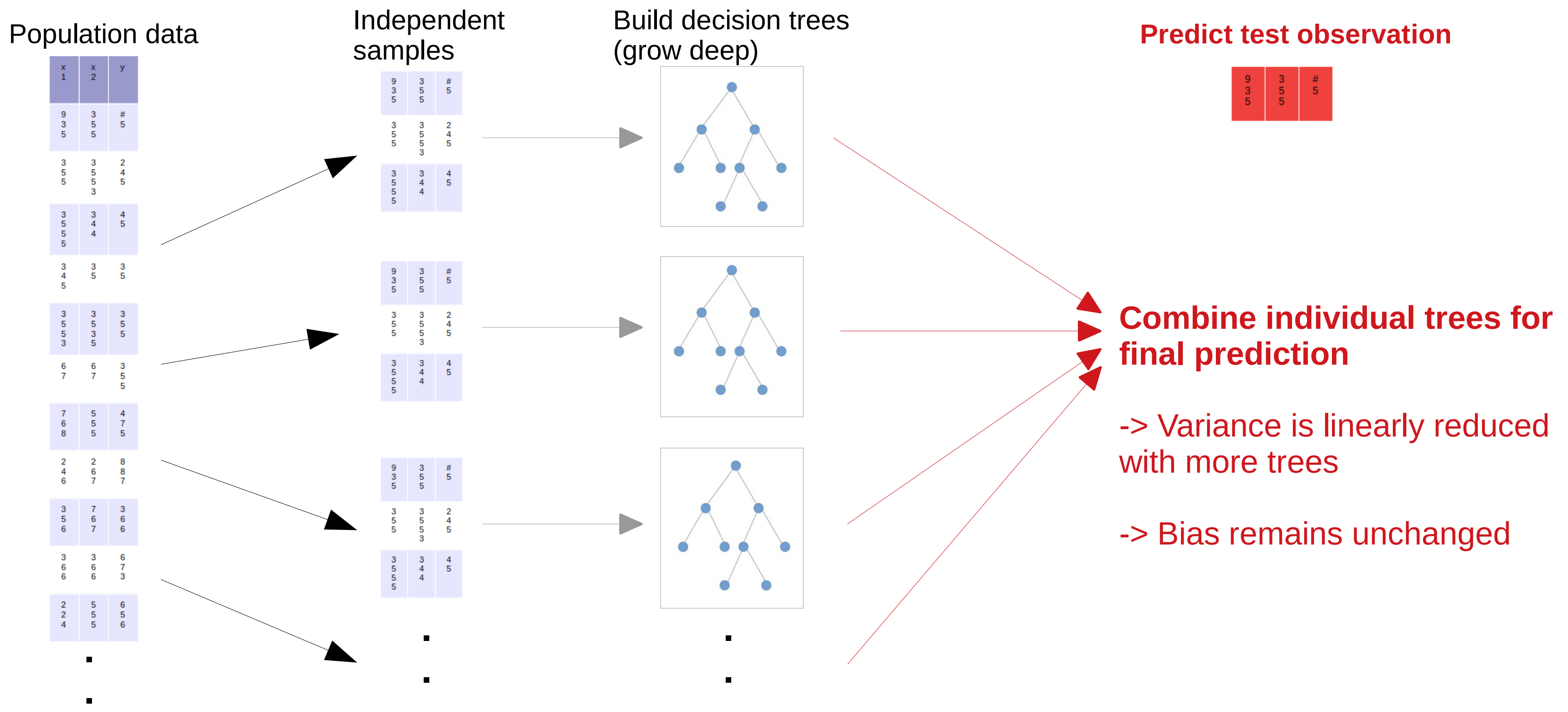
Ensemble methods: Bagging

- **Bagging** (bootstrap aggregating) is an ensemble method typically applied to decision trees (but can be applied to all classifiers)
- Main idea: Fully grown decision trees have **low bias** but **high variance**...
  - ...it would be nice to have multiple independent samples from the same population to **reduce variance** by taking the mean of prediction



Ensemble methods: Bagging

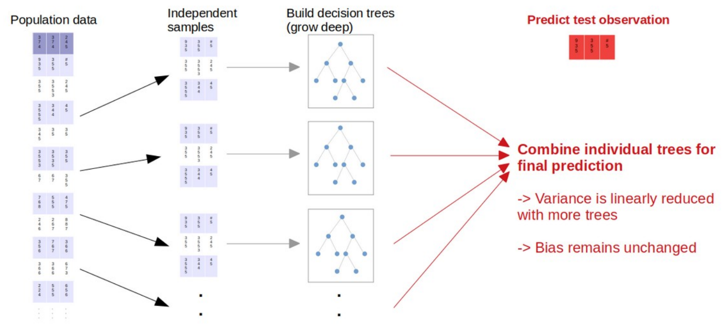
- **Bagging** (bootstrap aggregating) is an ensemble method typically applied to decision trees (but can be applied to all classifiers)
- Main idea: Fully grown decision trees have **low bias** but **high variance**...
  - ...it would be nice to have multiple independent samples from the same population to **reduce variance** by taking the mean of prediction

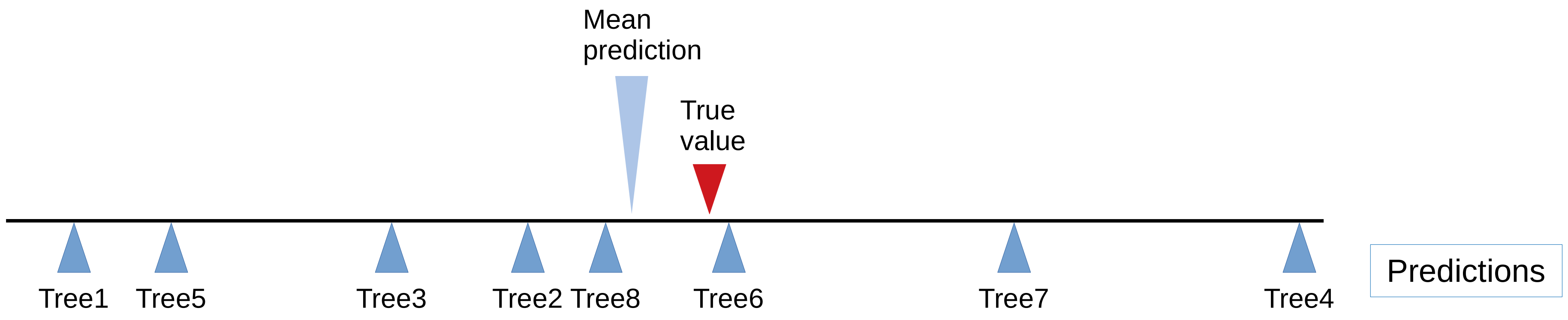


Population data
Independent samples
Build decision trees (grow deep)
Predict test observation
Combine individual trees for final prediction
-> Variance is linearly reduced with more trees
-> Bias remains unchanged

Ensemble methods: Bagging

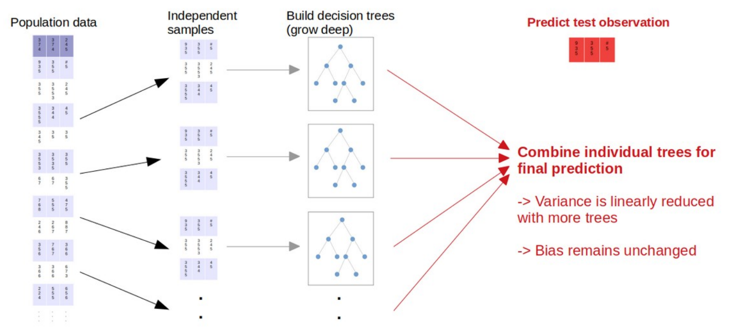
- **Bagging** (bootstrap aggregating) is an ensemble method typically applied to decision trees (but can be applied to all classifiers)
- Main idea: Fully grown decision trees have **low bias** but **high variance**...
  - ...it would be nice to have multiple independent samples from the same population to **reduce variance** by taking the mean of prediction
- By averaging the individual trees the variance of the estimation is reduced

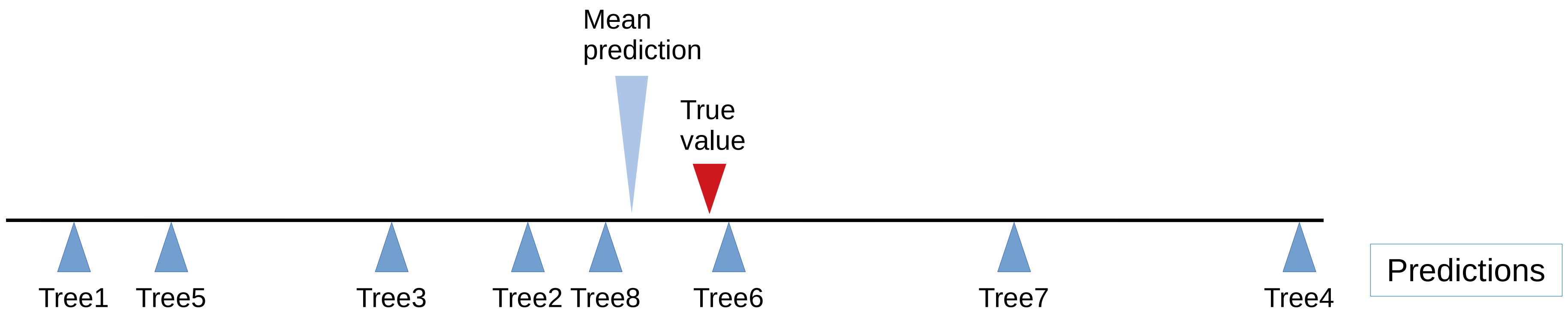

Population data
Independent samples
Build decision trees (grow deep)
Predict test observation
Combine individual trees for final prediction
-> Variance is linearly reduced with more trees
-> Bias remains unchanged



Ensemble methods: Bagging

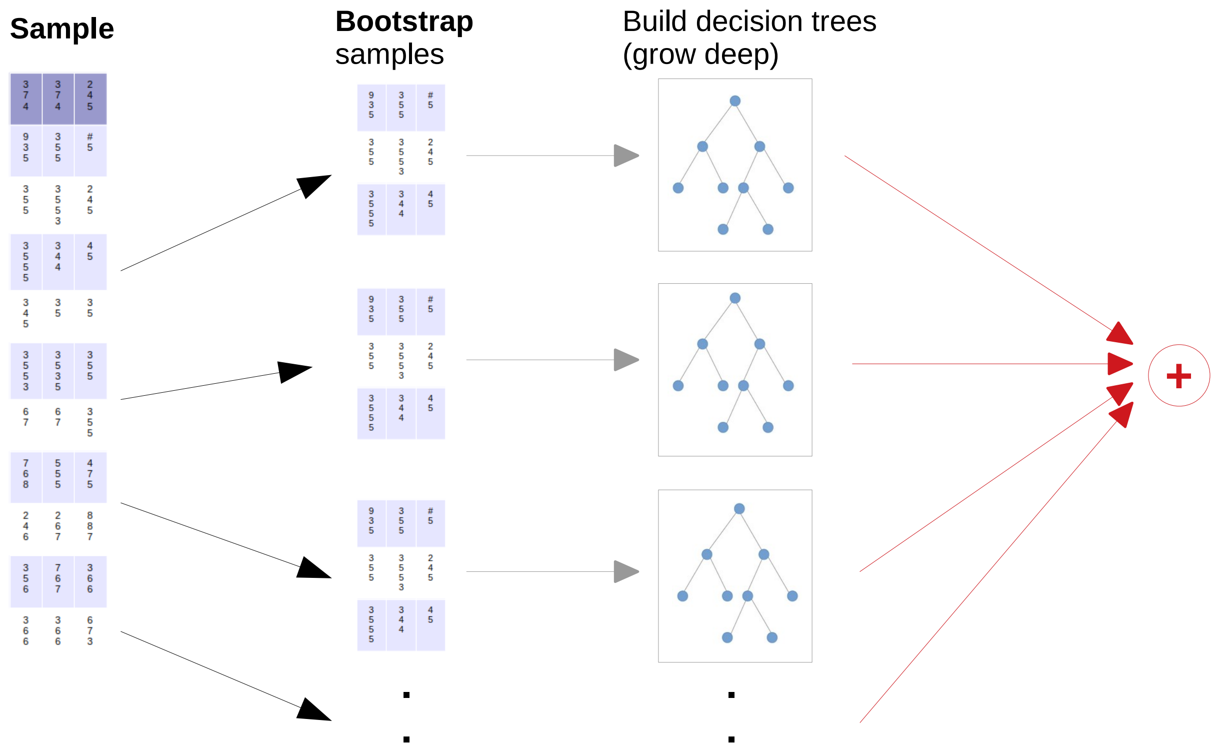
- **Bagging** (bootstrap aggregating) is an ensemble method typically applied to decision trees (but can be applied to all classifiers)
- Main idea: Fully grown decision trees have **low bias** but **high variance**...
  - ...it would be nice to have multiple independent samples from the same population to **reduce variance** by taking the mean of prediction
- By averaging the individual trees the variance of the estimation is reduced
- **However**: In reality we usually only have **one training sample** from the population


Population data
Independent samples
Build decision trees (grow deep)
Predict test observation
Combine individual trees for final prediction
-> Variance is linearly reduced with more trees
-> Bias remains unchanged



Ensemble methods: Bagging

- In reality we usually only have only **one sample** from the population
  - Solution: We now take **bootstrap samples** from our training data
  - Grow deep decision trees on bootstrap samples and combine the trees for prediction



Ensemble methods: Bagging

- In reality we usually only have only **one sample** from the population
  - Solution: We now take **bootstrap samples** from our training data (originally suggested by Breiman 1996)
  - Grow deep decision trees on bootstrap samples and combine the trees for prediction
- Bootstrap samples are samples drawn with replacement:

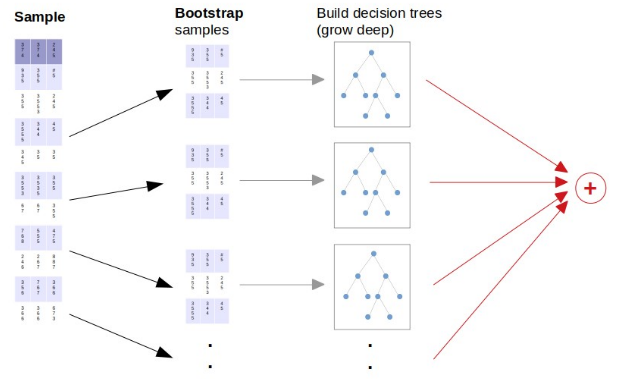
Original data

4	5	22	3	99	67	43	1	21	2

Bootstrap samples (same size as original data)

3	5	67	5	43	67	21	2	99	3
2	4	4	67	21	1	1	67	99	3
5	5	21	99	1	2	22	99	67	5

.
.



Breiman L: Bagging Predictors. Machine Learning 1996, 24(2):123-140.

Ensemble methods: Bagging

- In reality we usually only have only **one sample** from the population
  - Solution: We now take **bootstrap samples** from our training data (originally suggested by Breiman 1996)
  - Grow deep decision trees on bootstrap samples and combine the trees for prediction
- Bootstrap samples are samples drawn with replacement:

Original data

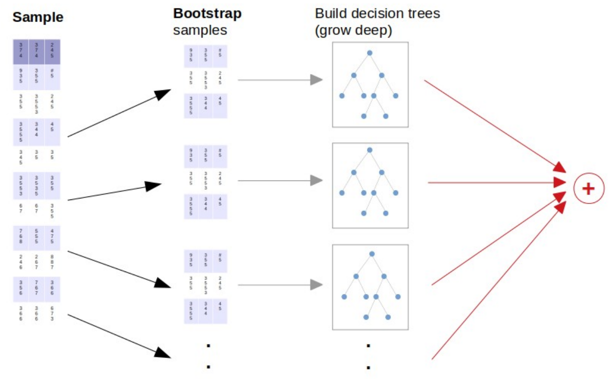
4	5	22	3	99	67	43	1	21	2

Bootstrap samples (same size as original data)

3	5	67	5	43	67	21	2	99	3
2	4	4	67	21	1	1	67	99	3
$\boxed{5}$	$\boxed{5}$	21	99	1	2	22	99	67	$\boxed{5}$

Same observation can appear multiple times!

.
.



Breiman L: Bagging Predictors. Machine Learning 1996, 24(2):123-140.

Ensemble methods: Bagging

- In reality we usually only have only **one sample** from the population
  - Solution: We now take **bootstrap samples** from our training data (originally suggested by Breiman 1996)
  - Grow deep decision trees on bootstrap samples and combine the trees for prediction
- Bootstrap samples are samples drawn with replacement:

Original data

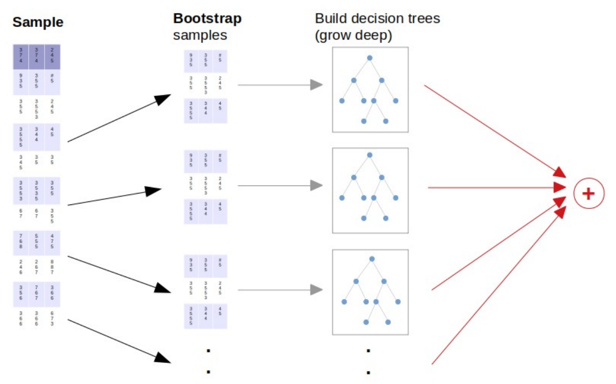
4	5	22	3	99	67	43	1	21	2

Bootstrap samples (same size as original data)

3	5	67	5	43	67	21	2	99	3
2	4	4	67	21	1	1	67	99	3
5	5	21	99	1	2	22	99	67	5

Same observation can appear multiple times!

⋮

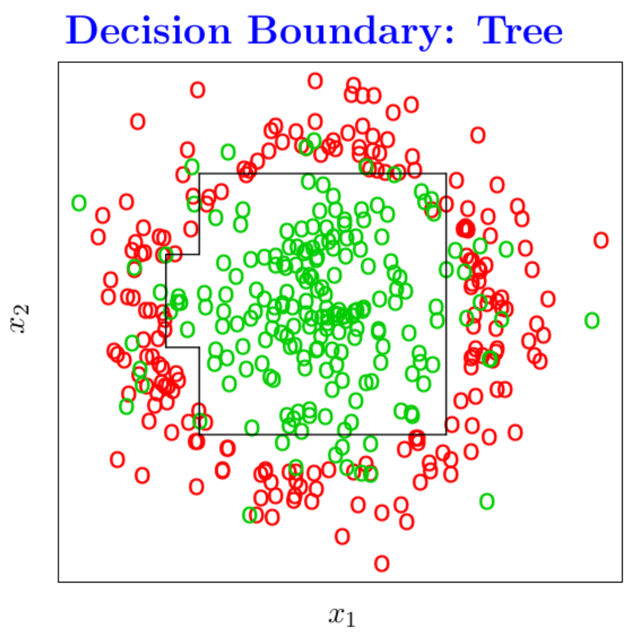


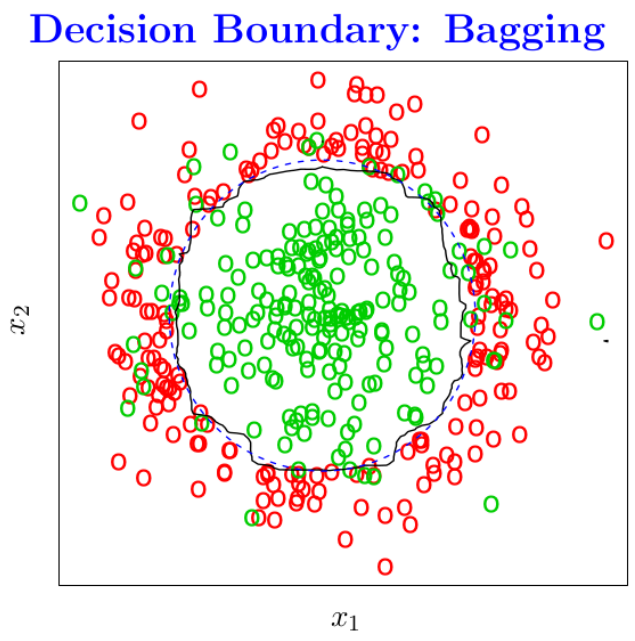
Bagging shows better performance than one decision tree applied to original data (why?)

Note: Individual bootstrap samples are not independent!

-> Variance is only reduced sublinearly

Ensemble methods: Bagging

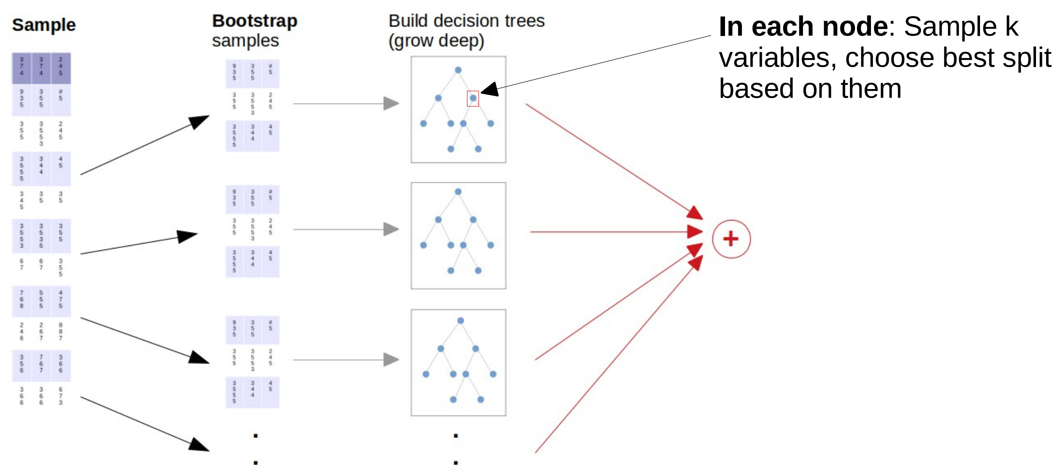




Quelle: Ji Zhu, University of Michigan

Ensemble methods: Random Forest

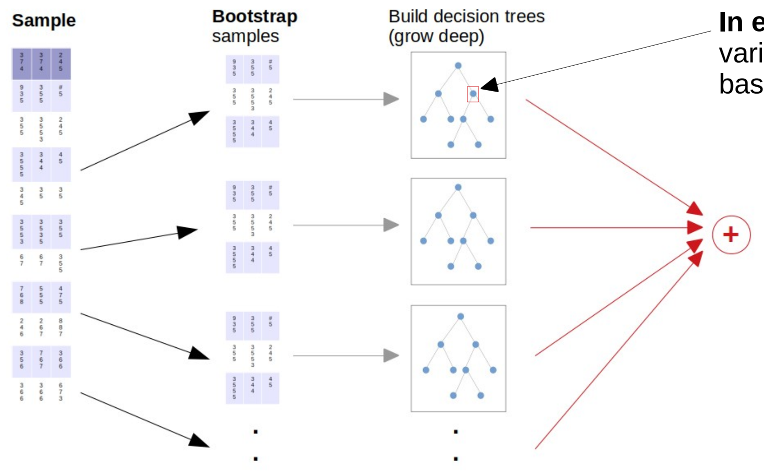
- **Random forest** is an extension of bagging
  - Main idea: Do not only sample from the data but also from the **variables**, which are used for splitting
- In random forest the trees are generated in the following way
  - Generate bootstrap samples from original data (same like bagging)
  - Build a decision tree on each bootstrap sample, but...
  - ... at each node of a decision tree, **randomly select k variables**, which are evaluated for splitting. Choose the best split (using only the k variables).


Sample
Bootstrap samples
Build decision trees (grow deep)
In each node: Sample k variables, choose best split based on them
+

Ensemble methods: Random Forest

- **Random forest** is an extension of bagging
  - Main idea: Do not only sample from the data but also from the **variables**, which are used for splitting
- In random forest the trees are generated in the following way
  - Generate bootstrap samples from original data (same like bagging)
  - Build a decision tree on each bootstrap sample, but...
  - ... at each node of a decision tree, **randomly select k variables**, which are evaluated for splitting. Choose the best split (using only the k variables).

Can also use **sub-samples** instead of bootstrap samples. Bootstrap sampling can also induce bias in variable selection.

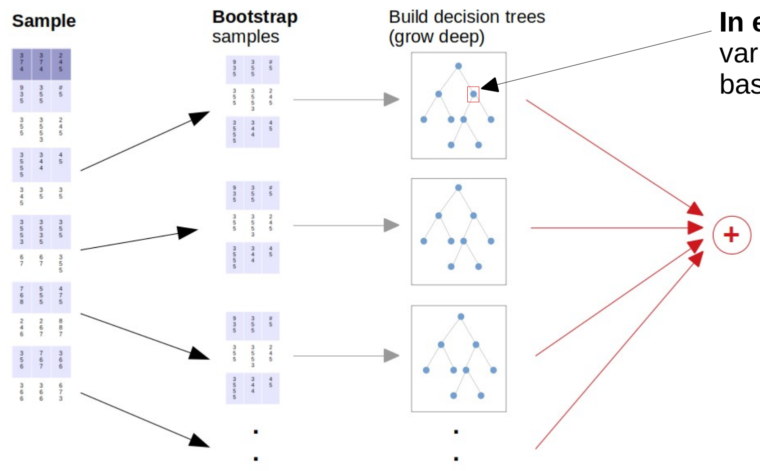


In each node: Sample k variables, choose best split based on them

Bias of bootstap sampling in RF:
Strobl, C., Boulesteix, A., Zeileis, A. et al. Bias in random forest variable importance measures: Illustrations, sources and a solution. BMC Bioinformatics 8, 25 (2007)

Ensemble methods: Random Forest

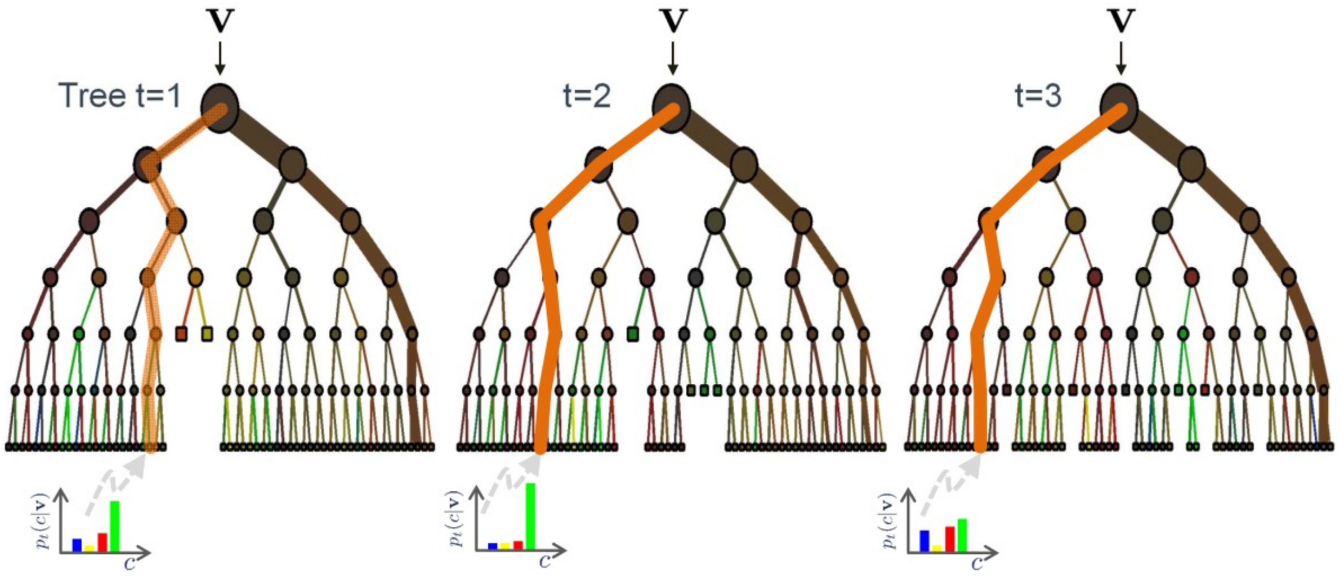
- What is the point of sampling the variables?
  - Sampling variables de-correlates the individual trees, makes them more diverse (this increases variance reduction)
  - **Intuition**: By restricting the variable selection at each node, some variables are incorporated in the analysis which might otherwise never be considered
  - This can reveal interactions in the data, which would otherwise not be detected



In each node: Sample k variables, choose best split based on them

Ensemble methods: Random Forest

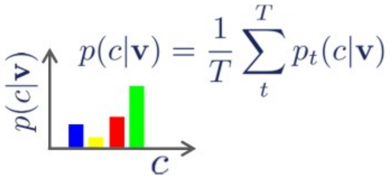
- Prediction of a new observation same like in bagging:
  - Run observation through all trees and incorporate each tree's prediction in the final prediction (e.g. majority vote for classification or mean for regression)
- Example of aggregating the tree's results for **classification**:



Source: Microsoft

Winner category: Category that was predicted most often (green)

Average probability:



$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$$

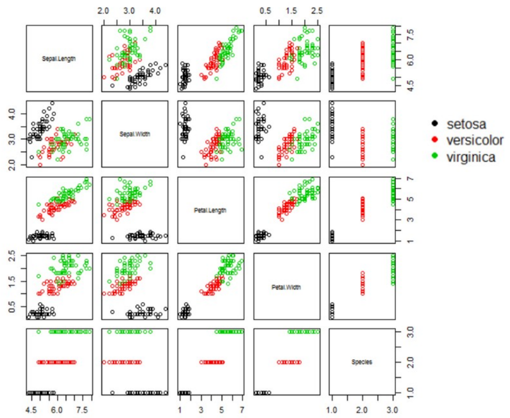
Random Forest in R

- We will use the **cforest()** function from the package "party" (unbiased variable selection)

```
library(party)
cfor_ctr <- cforest_unbiased(ntree = 500, mtry = 2)
(rf.iris <- cforest(Species~., data = iris, controls = cfor_ctr))
       Random Forest using Conditional Inference Trees


Number of trees:  500


Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  150
# honest cross classification (OOB-error)
(confT <- table(iris$Species, predict(rf.iris, OOB = TRUE)))
             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         3
  virginica       0          4        46
diag(confT) <- 0
sum(confT)/nrow(iris)
[1] 0.04666667
varimp(rf.iris)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
 0.051127273  0.002109091  0.314800000  0.264909091
```



Random Forest in R

- We will use the **cforest()** function from the package "party" (unbiased variable selection)

Define the options/parameters for random forest

ntree: How many trees are grown?

mtry: How many variables are sampled at each node?

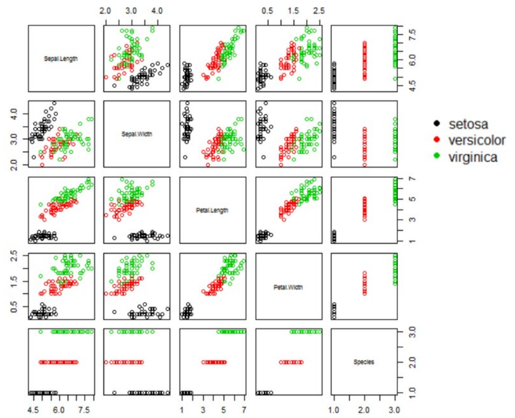
```
library(party)
cfor_ctr <- cforest_unbiased(ntree = 500, mtry = 2)
(rf.iris <- cforest(Species~., data = iris, controls = cfor_ctr))
       Random Forest using Conditional Inference Trees


Number of trees:  500


Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  150
# honest cross classification (OOB-error)
(confT <- table(iris$Species, predict(rf.iris, OOB = TRUE)))
             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         3
  virginica       0          4        46
diag(confT) <- 0
sum(confT)/nrow(iris)
[1] 0.04666667
varimp(rf.iris)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
 0.051127273  0.002109091  0.314800000  0.264909091
```

What is OOB? ...

What is varimp? ...



Out-of-bag error (OOB)

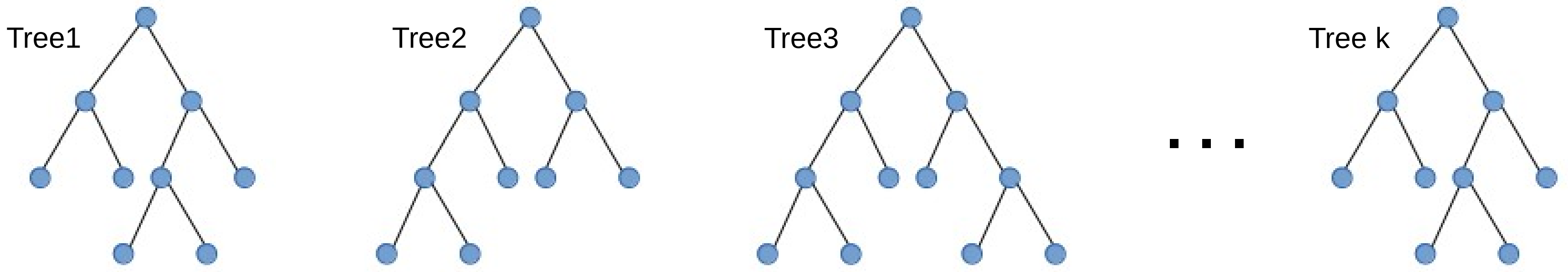
- Random Forest comes with its own integrated evaluation tool!
- Each tree is fitted to a bootstrap sample (or sub-sample) of original data
  - Thus, every tree in the forest has only seen a part of the data
- To calculate the **OOB-error**:
  - Predict each observation in the data using only the trees, which have not seen the observation when they were generated

Out-of-bag error (OOB)

- Random Forest comes with its own integrated evaluation tool!
- Each tree is fitted to a bootstrap sample (or sub-sample) of original data
  - Thus, every tree in the forest has only seen a part of the data
- To calculate the **OOB-error**:
  - Predict each observation in the data using only the trees, which have not seen the observation when they were generated

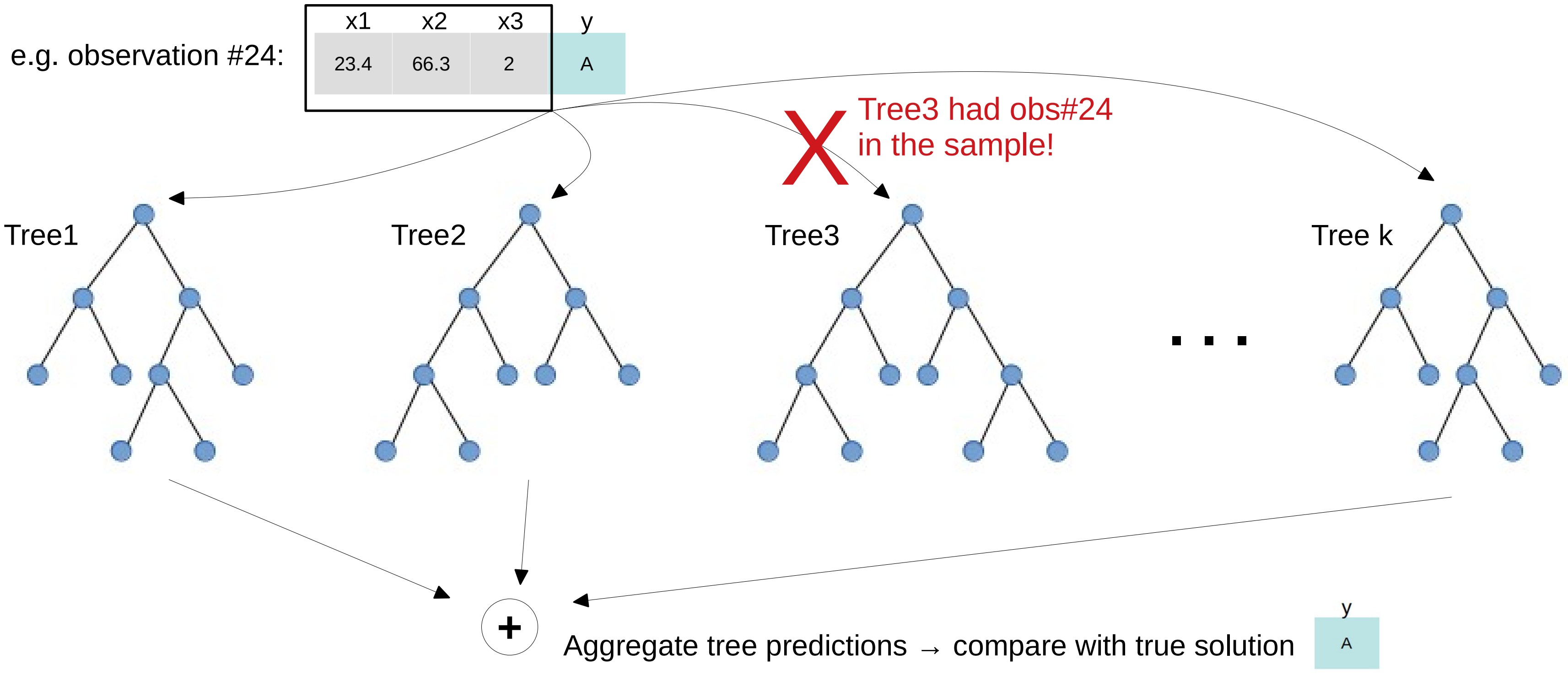
e.g. observation #24:

x1	x2	x3	y
23.4	66.3	2	A



Out-of-bag error (OOB)

- Random Forest comes with its own integrated evaluation tool!
- Each tree is fitted to a bootstrap sample (or sub-sample) of original data
  - Thus, every tree in the forest has only seen a part of the data
- To calculate the **OOB-error**:
  - Predict each observation in the data using only the trees, which have not seen the observation when they were generated


e.g. observation #24:
x1
x2
x3
y
23.4
66.3
2
A
Tree3 had obs#24 in the sample!
Tree1
Tree2
Tree3
Tree k
...
+
Aggregate tree predictions → compare with true solution
y
A

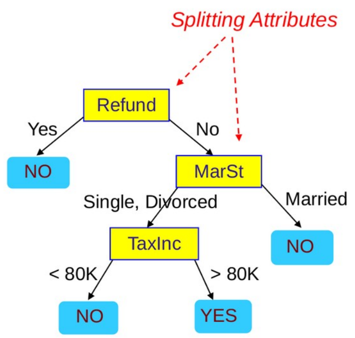
Out-of-bag error (OOB)

- Create OOB confusion table to look at the OOB-performance (see R-code slide)
- Use predict()-function and set OOB=TRUE, give no "newdata" argument

```
rf.iris <- cforest(Species~., data = iris, controls =
cfor_ctr)
(confT <- table(iris$Species, predict(rf.iris, OOB = TRUE)))
             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         3
  virginica       0          4        46
diag(confT) <- 0
sum(confT)/nrow(iris)
[1] 0.04666667
```

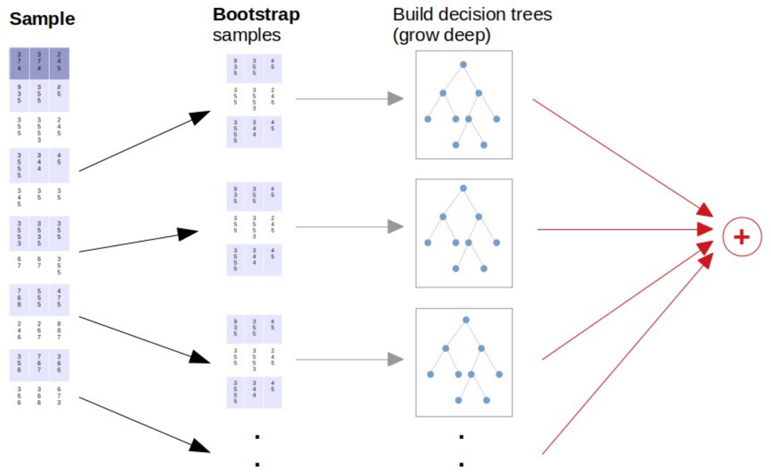
Interpretability of Random Forest

- One advantage of decision trees are their good interpretability
  - Can easily track how each variable affects the prediction
- In Random Forest we somewhat lose the interpretability
  - Difficult to track how each variable is active in each of the e.g. 500 trees…
- How can we assess which variables are **important** for prediction?



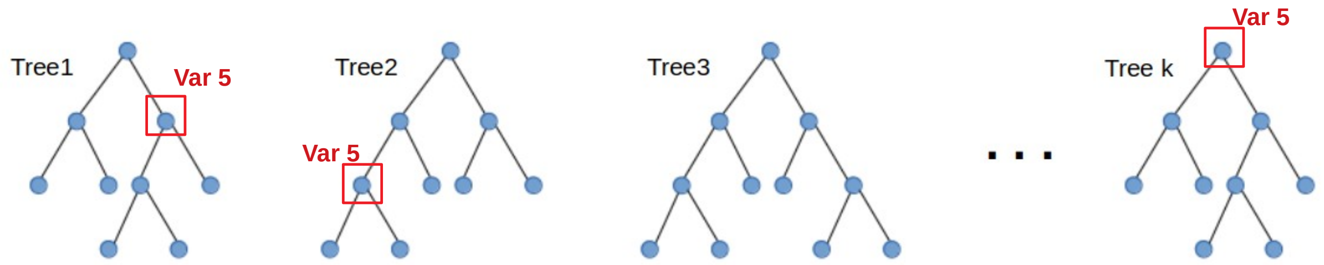
Model: Decision Tree

vs.



Variable importance in RF

- How could variable importance be **scored**?
  - Simple idea: Count the number of times a variable is chosen for splitting throughout all trees
- Two more elaborate approaches:
- **Variable importance 1**: Average score improvement at splits
  - For each node where variable x was used for splitting, record the achieved improvement in the splitting score (e.g. Gini-index -> **Gini-importance score**)
  - Variable x's importance 1 score is the average of all score improvements (can be weighted with the number of data points in each node)

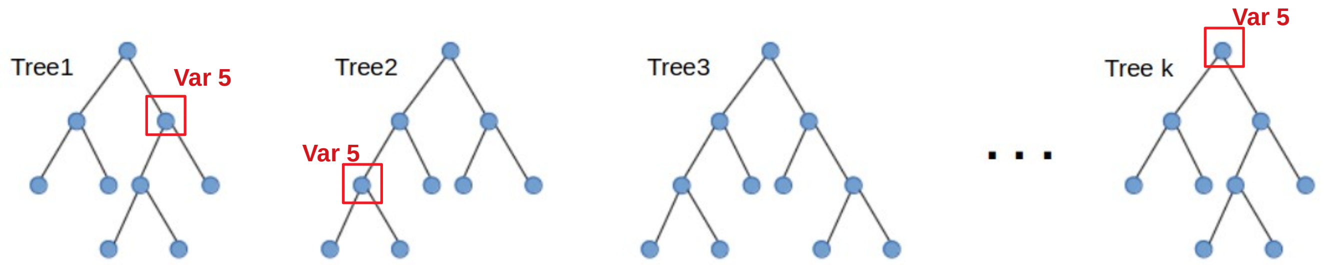


Variable importance in RF

- How could variable importance be **scored**?
  - Simple idea: Count the number of times a variable is chosen for splitting throughout all trees
- Two more elaborate approaches:

Caution: Gini-Importance is biased if different types of variables are present

- **Variable importance 1**: Average score improvement at splits
  - For each node where variable x was used for splitting, record the achieved improvement in the splitting score (e.g. Gini-index -> **Gini-importance score**)
  - Variable x's importance 1 score is the average of all score improvements (can be weighted with the number of data points in each node)



Variable importance in RF

- **Variable importance 2**: Permutation importance
- **Permutation importance** is a very intuitive importance score
- Main idea: Mix up the values of variable x to break up any meaningful relation between x and the target variable (permutation)
  - Check how much the performance (usually the OOB-error) drops after permutation of x

Original data

X1 (Predictor 1)	X2 (Predictor 2)	X3 (Predictor 3)	Y (Target var)
2.3	33.1	67	A
5.1	35.8	70.3	B
3.3	34.0	96	A
2.8	37.7	85	C
1.3	38.3	84.9	A

→ Calculate **OOB-error**

vs.

X1 permuted

X1 (Predictor 1)	X2 (Predictor 2)	X3 (Predictor 3)	Y (Target var)
5.1	33.1	67	A
2.3	35.8	70.3	B
2.8	34.0	96	A
1.3	37.7	85	C
3.3	38.3	84.9	A

→ Calculate **OOB-error**

Permutation importance:
Difference in OOB-error

With stronger relation between x1 and target variable, the more should the OOB-error grow

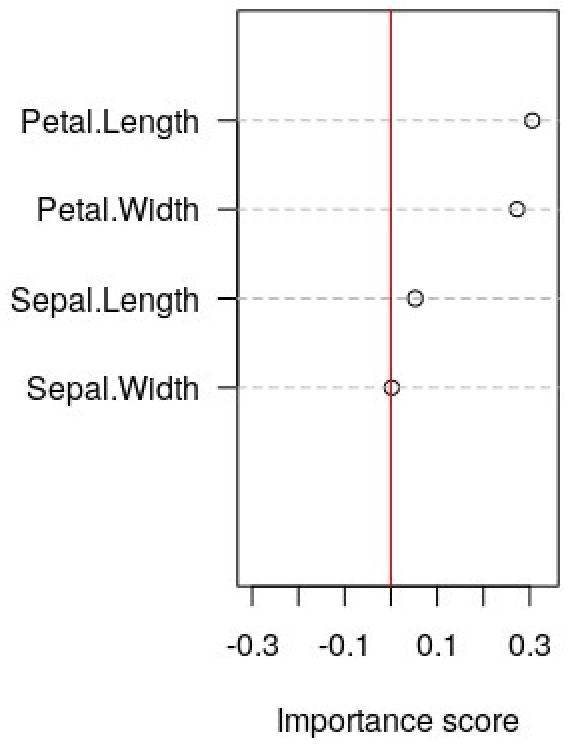
Variable importance with R

- To calculate the variable importance after fitting a Random Forest use the **varimp()**-function (default is permutation importance score)

```
rf.iris <- cforest(Species~., data = iris, controls = cfor_ctr)
varimp(rf.iris)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
 0.052727273  0.001854545  0.305636364  0.272727273

# Function to plot importance scores nicely
implot <- function(rf.fit){
  par(mar=c(9,9,9,9))
  scores <- sort(varimp(rf.fit))
  plot(x = scores, y = 1:length(scores) ,
       xlim=c(-max(scores), max(scores)*1.1),
       ylim=c(-1,(length(scores)+1)),
       yaxt='n', ylab='', xlab='Importance score')
  axis(2, at=1:length(scores), labels = names(scores), las=1)
  abline(h=1:length(scores), lty=2, col='grey')
  abline(v=0, col='red')
}

implot(rf.iris)
```



Variable importance with R

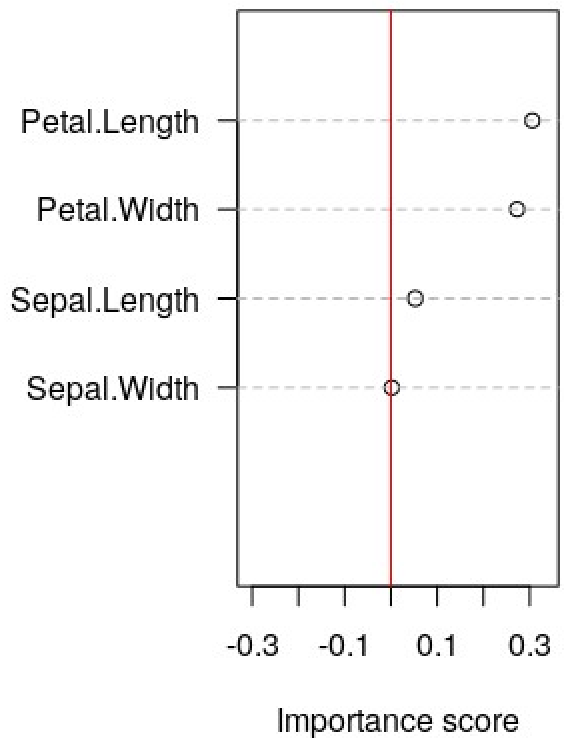
- To calculate the variable importance after fitting a Random Forest use the **varimp()**-function (default is permutation importance score)

Permutation importance scores can also be negative! → Why?

```
rf.iris <- cforest(Species~., data = iris, controls = cfor_ctr)
varimp(rf.iris)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
 0.052727273  0.001854545  0.305636364  0.272727273

# Function to plot importance scores nicely
implot <- function(rf.fit){
  par(mar=c(9,9,9,9))
  scores <- sort(varimp(rf.fit))
  plot(x = scores, y = 1:length(scores) ,
       xlim=c(-max(scores), max(scores)*1.1),
       ylim=c(-1,(length(scores)+1)),
       yaxt='n', ylab='', xlab='Importance score')
  axis(2, at=1:length(scores), labels = names(scores), las=1)
  abline(h=1:length(scores), lty=2, col='grey')
  abline(v=0, col='red')
}

implot(rf.iris)
```

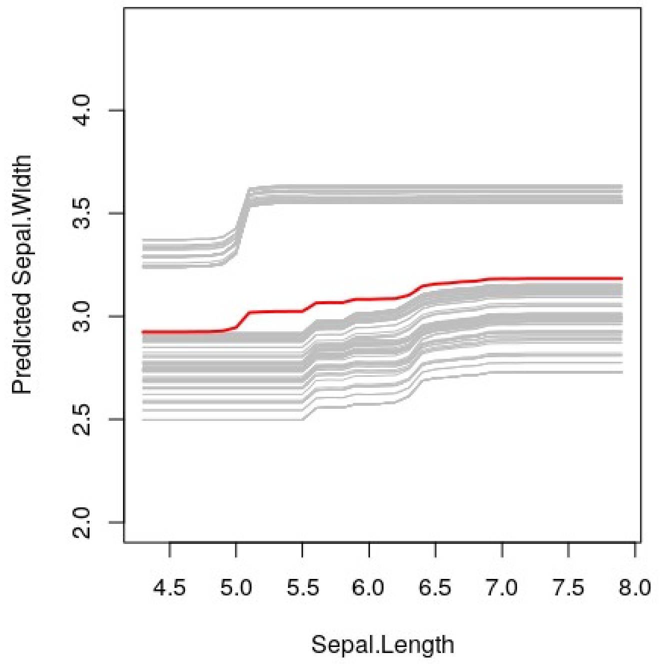


Because importance scores are based on random permutation we should fix the random seed when calculating them

Dependency plots in Random Forest

- Dependency plots are another method to "bring light" into what is happening in the forest
- Main idea: See how the prediction of the target variable changes when only **one predictor** is shifted in its value
- Example of **partial dependency plot** on RF-regression on the iris data (target variable: Sepal.width):

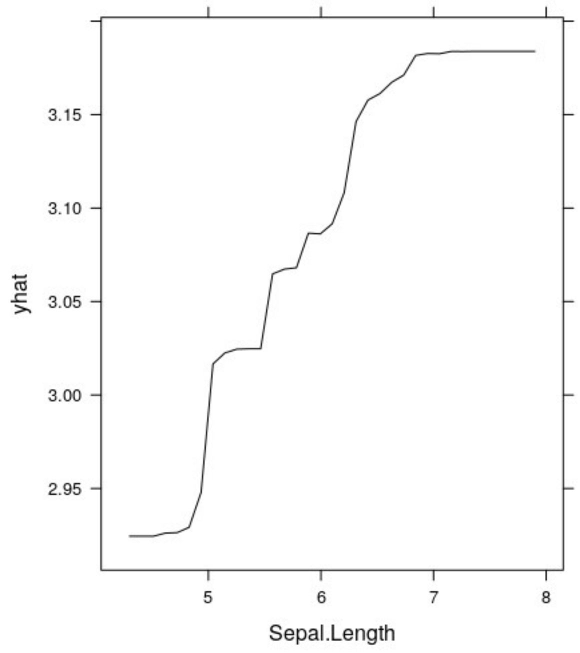
```
cfor_ctr <- cforest_unbiased(ntree = 500, mtry = 2)
rf.SepWid <- cforest(Sepal.Width~., data = iris, controls = cfor_ctr)
steps <- seq(min(iris$Sepal.Length), max(iris$Sepal.Length), by = 0.1)
predic <- matrix(NA, nrow=nrow(iris), ncol = length(steps))
for (i in 1:nrow(iris)){
  obs <- iris[i,]
  obs$Sepal.Width <- NULL # Remove column
  for(s in 1:length(steps)){
    obs$Sepal.Length <- steps[s]
    predic[i,s] <- predict(rf.SepWid, newdata=obs)
  }
}
plot(NULL, xlim=c(min(iris$Sepal.Length), max(iris$Sepal.Length)),
     ylim=c(min(iris$Sepal.Width), max(iris$Sepal.Width)),
     xlab='Sepal.Length', ylab='Predicted Sepal.Width')
for(l in 1:nrow(predic)){
  lines(x = steps, y = predic[l,], col='grey')
}
lines(x=steps, y=apply(predic, 2, mean), col='red', lwd=2)
```



Partial dependency plots with "pdp"

- "**pdp**" is an R package to construct partial dependency plots
- Can be applied to different Machine Learning methods
  - Also works for Random Forests fitted with cforest()

```
cfor_ctr <- cforest_unbiased(ntree = 500, mtry = 2)
rf.SepWid <- cforest(Sepal.Width~., data = iris,
             controls = cfor_ctr)
#install.packages("pdp")
library(pdp)
partial(rf.SepWid, pred.var = "Sepal.Length", plot = TRUE)
```



- Meaning of curve (same like in previous slide):
  - Mean predicted Sepal.Width for varying Sepal.Length
    (averaged over all observed constellations of other co-predictors)

Summary Random Forest

- **Advantages:**
  - Random Forest is a very effective ML method (good performance in ML-competitions)
  - Is non-parametric, poses no assumption regarding distribution of variables or residuals
  - Comes with included performance evaluation (OOB-error)
  - Random Forest does not tend to overfit
  - Can handle thousands of input variables
  - Can handle lots of noise variables with only few relevant variables
  - We can derive variable importance measures
  - Random Forest can detect strong and local interactions
  - Is robust against outliers
  - It can handle imbalanced data (e.g. using stratified bootstrap sampling)
  - ...

Summary Random Forest

- **Disadvantages:**
  - Less interpretability, feeling of a "black box"
  - Can not extrapolate predictions beyond the range of training data well
  - Takes more time to train than e.g. decision tree

Further reading

- Strobl, C., Malley, J., & Tutz, G. (2009). **An introduction to recursive partitioning: Rationale, application, and characteristics of classification and regression trees, bagging, and Random Forests**. Psychological Methods, 14(4), 323–348. https://doi.org/10.1037/a0016973