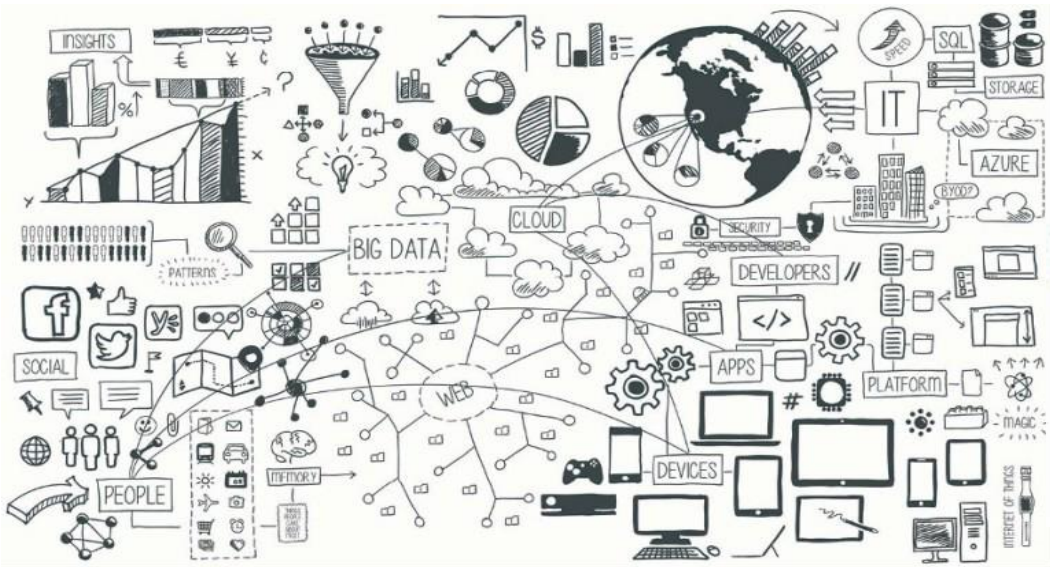
Data Mining (Minería de Datos)

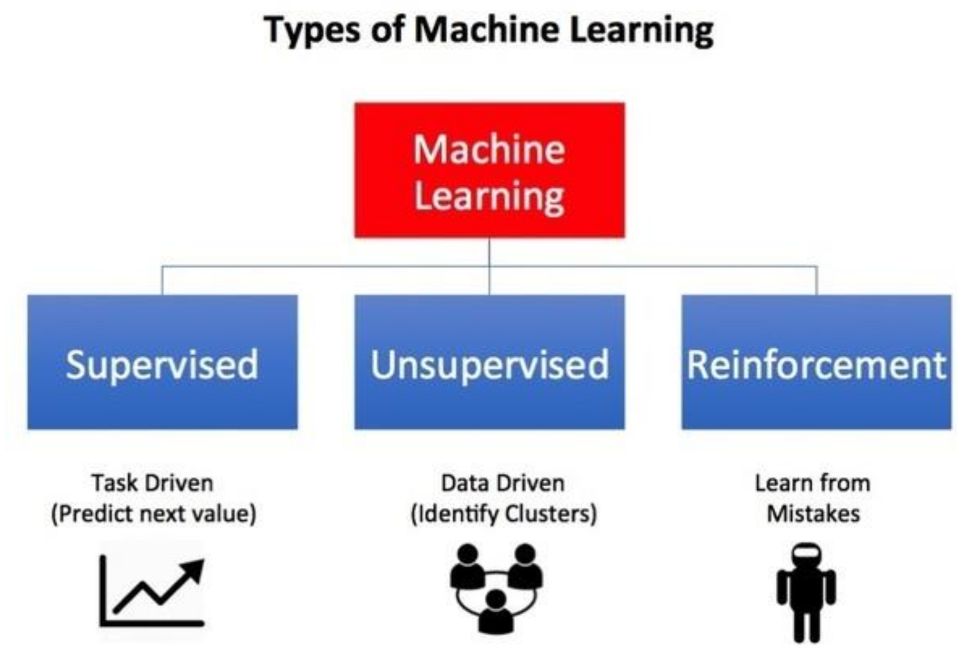
Ensembles: Bagging and boosting



Rodrigo Manzanas

Grupo de Meteorología

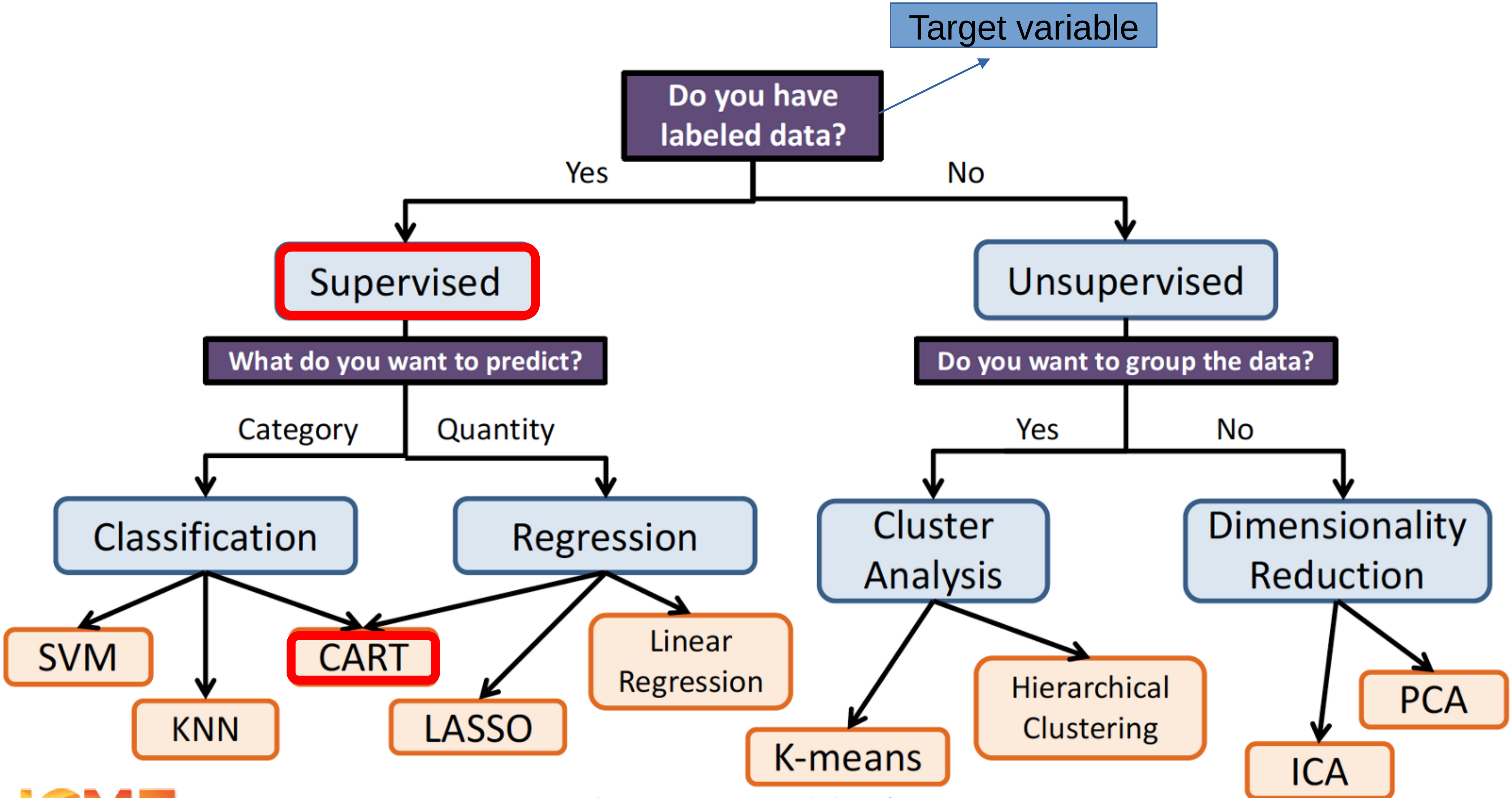
Univ. de Cantabria – CSIC
MACC / IFCA



NOTA: Las líneas de código de R en esta presentación se muestran sobre un fondo gris

Oct	30	Aplazada (sesión de refuezo)
Nov	6	Presentación, introducción y perspectiva histórica
	8	Paradigmas, problemas canónicos y data challenges
	13	Reglas de asociación
	15	Práctica: Reglas de asociación
	20	Evaluación, sobreajuste y cross-validación
	22	Práctica: Cross-validación
	27	Árboles de clasificación
	29	Práctica: Árboles de clasificación
		T01. Datos discretos
Dic	4	Técnicas de vecinos cercano (k-NN)
	11	Práctica: Vecinos cercanos
	13	Reducción de dimensión lineal
	18	Práctica: LDA y PCA
	20	Reducción no lineal
		T02. Clasificación
Ene	8	Árboles de clasificación y regresión (CART)
	10	Práctica: CART
	15	Ensembles: Bagging and Boosting
	17	Práctica: Random forests
		T03. Predicción
	22	Práctica: Gradient boosting
	24a	Técnicas de agrupamiento
	24b	Práctica: Técnicas de agrupamiento
	29a	Práctica: El paquete CARET
	29b	Examen

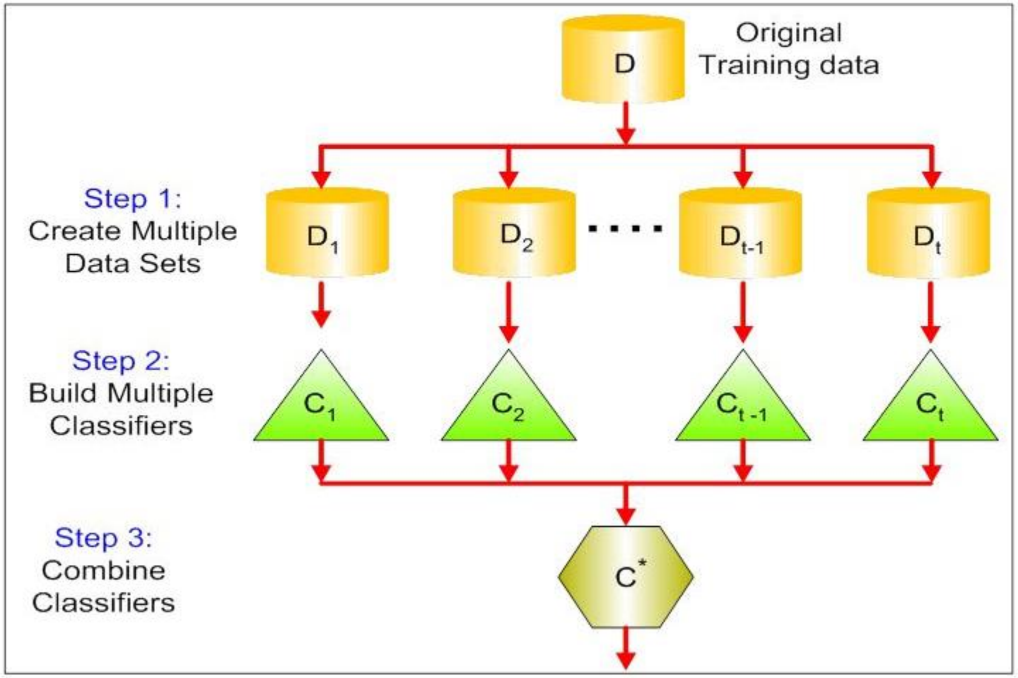
Calendar



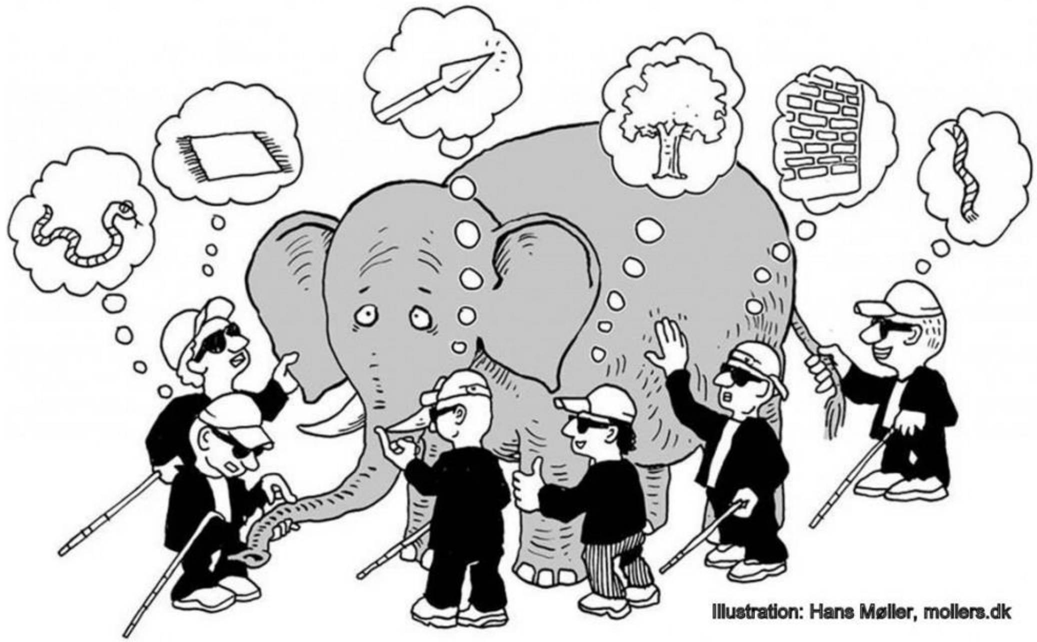
Machine Learning Workshop | XCME 006

Ensemble learning

Ensemble learning is a supervised approach in which the basic idea is to generate multiple weak models on a training dataset and combining them to generate a strong model which improves the ***stability*** and the ***performance*** of the individual models.



The wisdom of the crowd



Fable of blind men and elephant
https://en.wikipedia.org/wiki/Blind_men_and_an_elephant

Ensemble learning

Ensemble approaches are typically used with CART.

Pros
Trees are very easy to explain (even easier than linear regression)
Trees can be plotted graphically, and are easily interpreted
Trees can easily handle qualitative predictors
They work fine on both classification and regression problems

Cons
Poor prediction accuracy (compared with other approaches)
Instability when changing the train/test partition (cross-validation is key)

By aggregating many trees, the instability of the trees can be reduced and their predictive performance substantially improved.

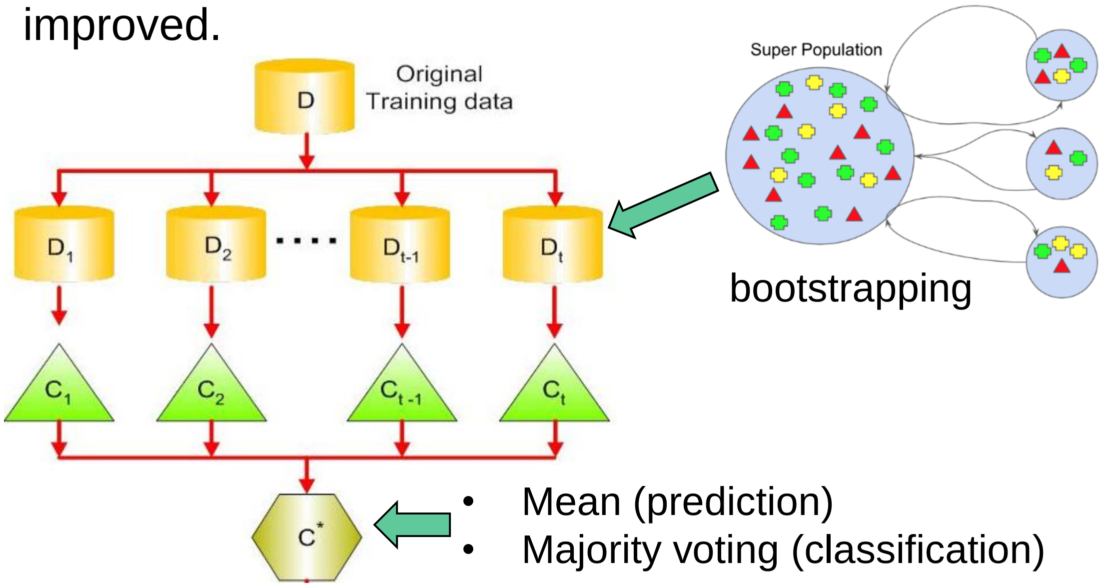
Ensemble learning

Ensemble approaches are typically used with CART.

Pros
Trees are very easy to explain (even easier than linear regression)
Trees can be plotted graphically, and are easily interpreted
Trees can easily handle qualitative predictors
They work fine on both classification and regression problems

Cons
Poor prediction accuracy (compared with other approaches)
Instability when changing the train/test partition (cross-validation is key)

By aggregating many trees, the **instability** of the trees can be reduced and their **performance** improved.



- Mean (prediction)
- Majority voting (classification)

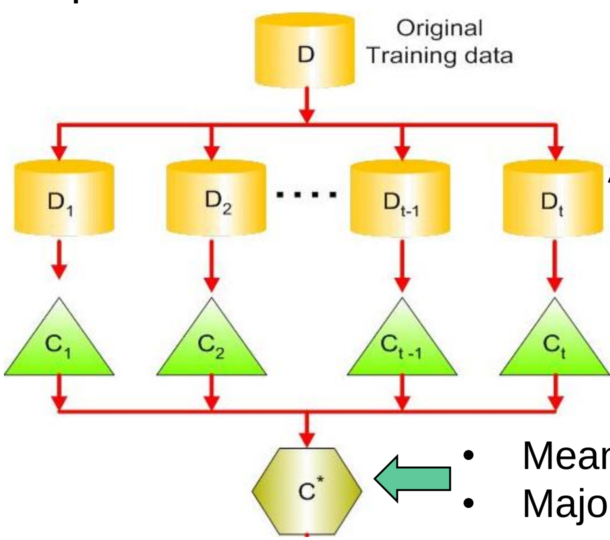
Ensemble learning

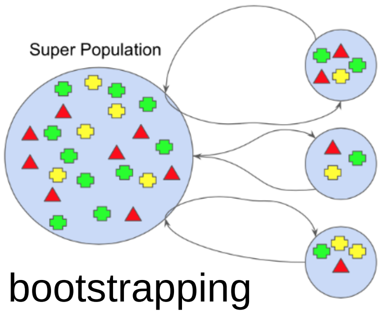
Ensemble approaches are typically used with CART.

Pros
Trees are very easy to explain (even easier than linear regression)
Trees can be plotted graphically, and are easily interpreted
Trees can easily handle qualitative predictors
They work fine on both classification and regression problems

Cons
Poor prediction accuracy (compared with other approaches)
Instability when changing the train/test partition (cross-validation is key)

By aggregating many trees, the **instability** of the trees can be reduced and their **performance** improved.

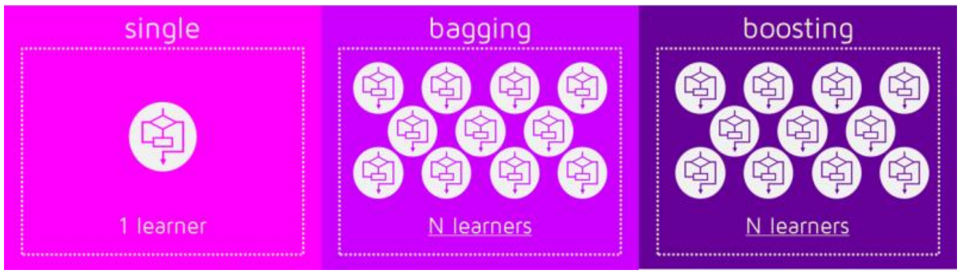




bootstrapping

- Mean (prediction)
- Majority voting (classification)

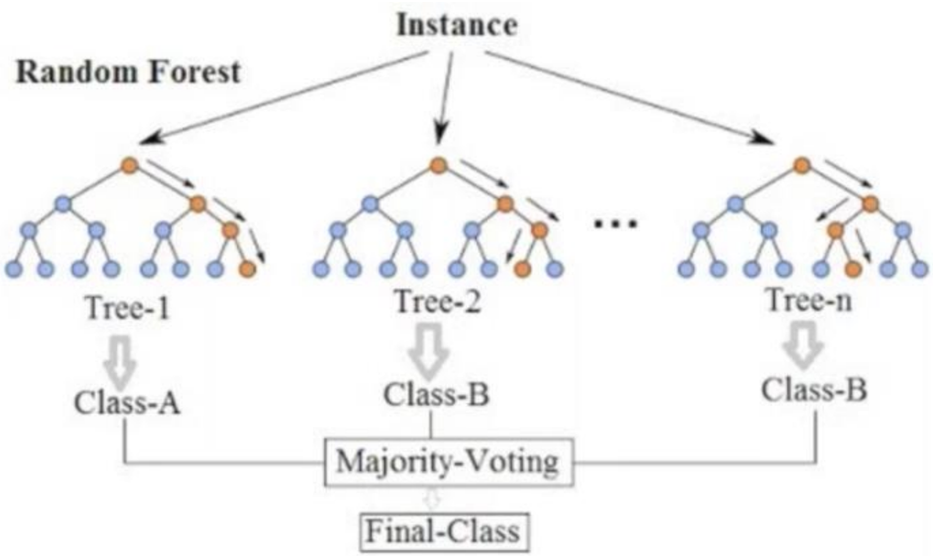
Most used approaches: **Bagging** and **boosting**

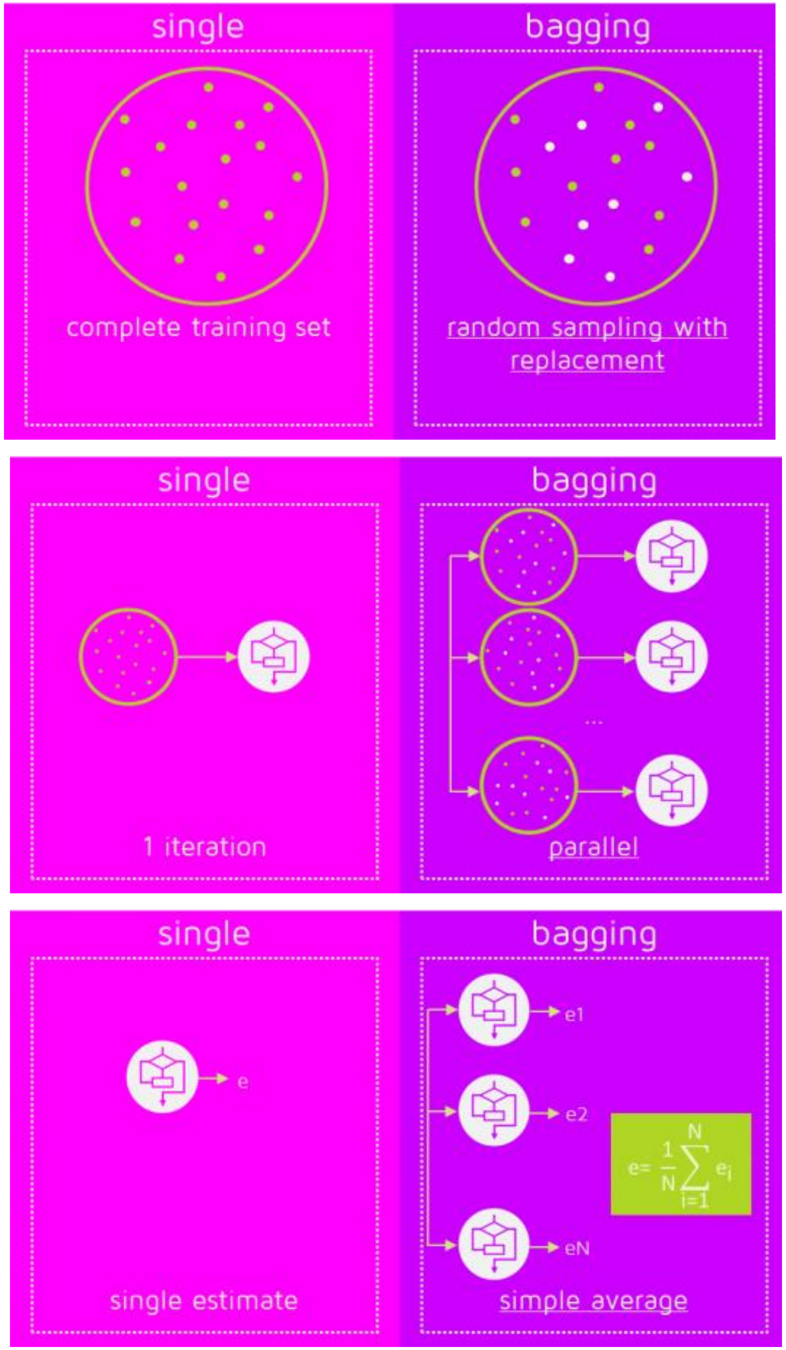


Bagging

Simple and powerful ensemble method.

1) Suppose there are N observations for training. M (only parameter to be chosen) subsamples are selected randomly with replacement (bootstrapping).
2) Using these bootstrapped subsamples, M individual trees are created **in pararell.** Each of these trees is fully grown and not pruned (we do not care about overfitting in bagging). These trees will have very low bias, but there will be a high variability among them.
3) A prediction for new input data is given based on the predictions resulting from the M individual trees (e.g. as the mean value, for majority voting…).





Bagging: Random forest

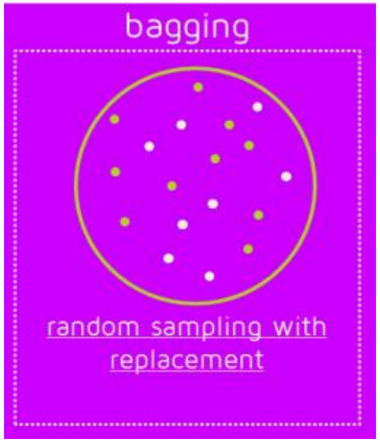
Random forest (RF) is an improvement over bagged trees.

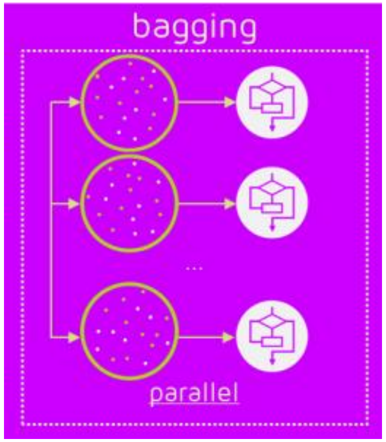
In CART, when selecting a split point, the learning algorithm is allowed to look through all predictor variables (p) in order to make the most division. Therefore, even with bagging, the individual trees can have a lot of structural similarities and in turn provide highly correlated predictions. However, ensemble methods work better if the predictions from the submodels are uncorrelated or at best weakly correlated.

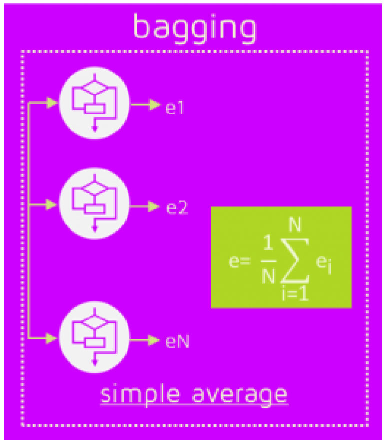
To solve this issue, in RF the learning algorithm is limited to a number of randomly selected predictors (m) at each splitting. Although m must be properly tuned, typical values for this parameter are:

m = sqrt(p), for classification problems
m = p/3, for prediction problems

Often, RF improve substantially the performance of individual trees.



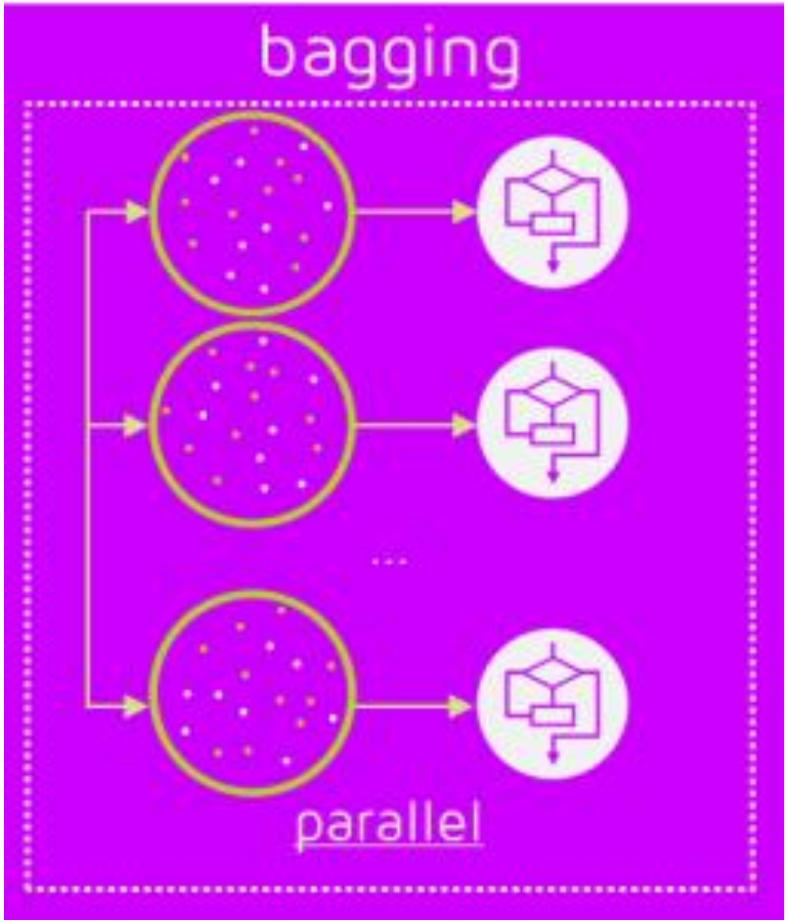




Bagging: Random forest

Estimated performance (test error)
For each bootstrap sample taken from the training data, there will be samples left behind that were not included. These samples are called Out-Of-Bag samples or OOB.

When averaged over all trees, the performance on these OBB provides a good estimate of the test error that may be expected.



Variable Importance
While the bagged trees are constructed, we can calculate how much the error drops for a variable at each split point.

These error drops can be averaged across all trees, providing thus an estimate of the importance of each input variable.

Random forest in R

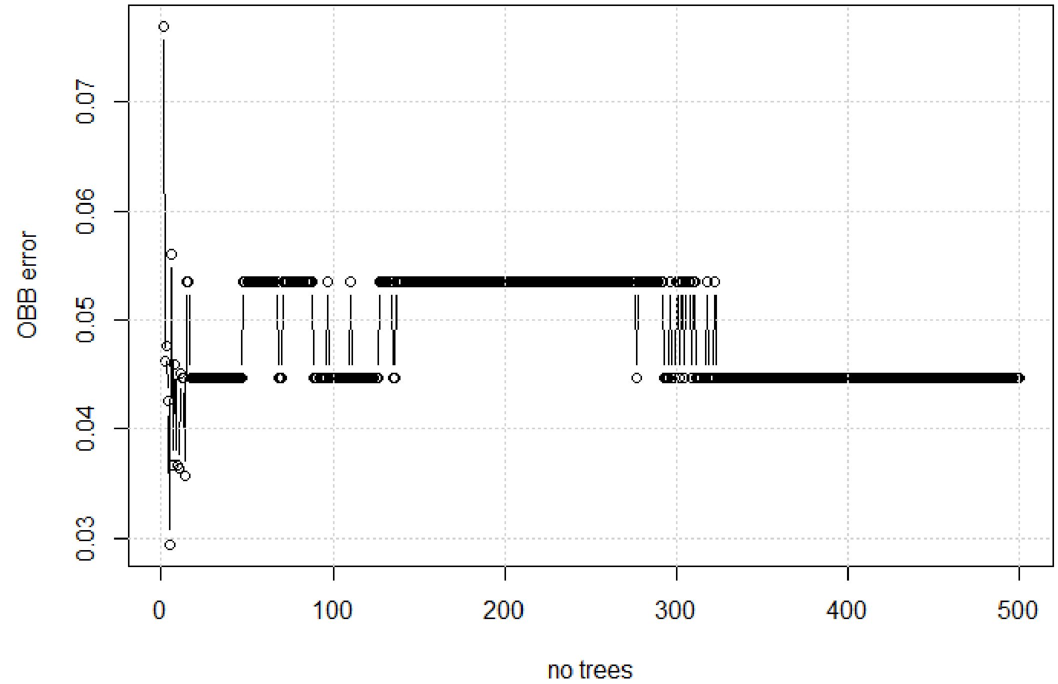
Classification problem (iris)

```
rm(list = ls())
Install.packages("randomForest")
library(randomForest)
```

```
n = nrow(iris)
# train/test partition
indtrain = sample(1:n, round(0.75*n))  # indices for train
indtest = setdiff(1:n, indtrain)  # indices for test
```

```
# RF
rf = randomForest(Species ~., iris , subset = indtrain)
# RF configuration: no. of trees? no. of predictors
considered at each node?
rf
```

```
# OOB error
plot(rf$err.rate[, 1], type = "b", xlab = "no trees",
ylab = "OBB error")
grid()
```




```
# prediction for test
pred = predict(rf, iris[indtest, ])
# accuracy
sum(diag(table(pred, iris$Species[indtest]))) / length(indtest)
```

```
# comparison with a single tree
library(tree)
t = tree(Species ~., iris, subset = indtrain)
# prediction for test
pred.t = predict(t, iris[indtest, ], type = "class")
# accuracy
sum(diag(table(pred.t, iris$Species[indtest]))) / length(indtest)
```

Random forest in R

Classification problem (rain/no rain)

```
load(“…/meteo.RData")
# keeping only 1000 days for this example
n = 1000 y = y[1:n]
x = x[1:n, ]
```

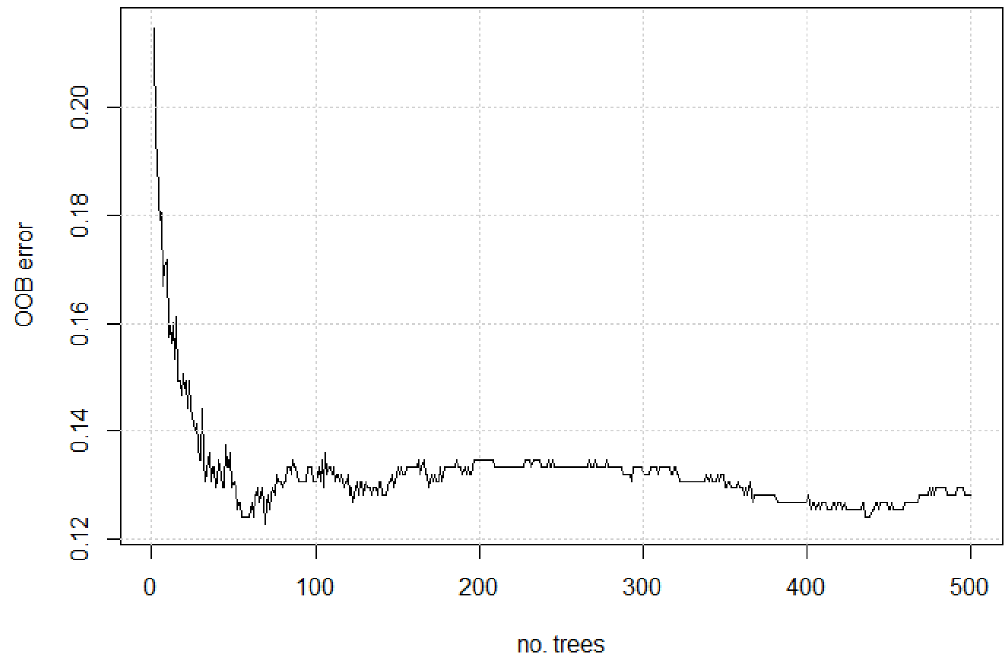
```
# train/test partition
indtrain = sample(1:n, round(0.75*n))  # indices for train
indtest = setdiff(1:n, indtrain)  # indices for test
```

```
# binary occurrence (1/0)
occ = y
occ[which(y < 1)] = 0
occ[which(y >= 1)] = 1
```

```
# dataframe for occurrence
df.occ = data.frame(y.occ = as.factor(occ), predictors = x)
```

```
# RF
rf = randomForest(y.occ ~., df.occ, subset = indtrain)
# RF configuration: no. of trees? no. of predictors considered at each node?
rf
```

```
# OOB error?
plot(rf$err.rate[, 1], type = "l", xlab = "no. trees", ylab = "OOB error")
grid()
```



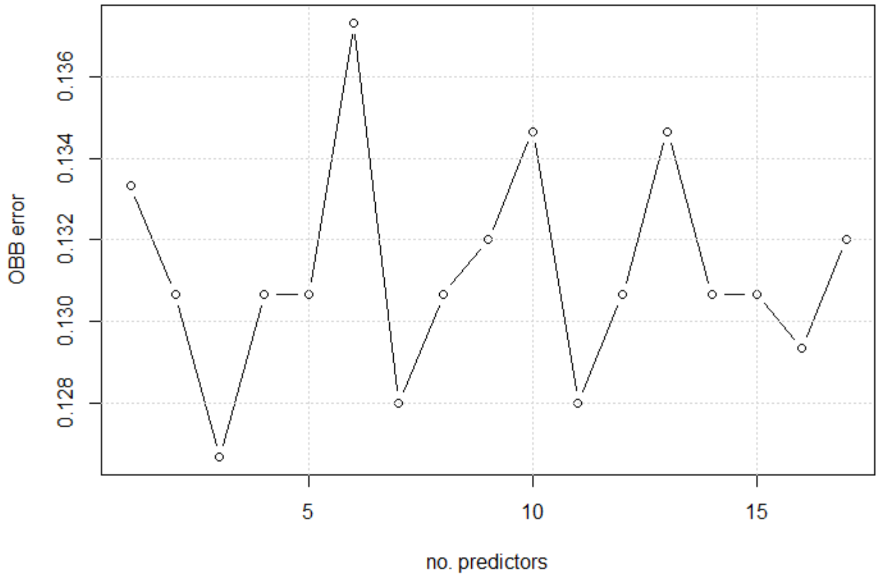

```
# test error?
pred = predict(rf, df.occ[indtest, ])
1 - sum(diag(table(pred, df.occ$y.occ[indtest]))) / length(indtest)  # error (1-accuracy)
```

Random forest in R

Classification problem (rain/no rain)

```
## fitting the optimum number of predictors considered
at each node (mtry)
ntree = which(rf$err.rate[,1] == min(rf$err.rate[,1]))
```

```
# OOB error?
err.oob = c()
for (mtry in 1:17) {
  rf.mtry = randomForest(y.occ ~., df.occ, subset = indtrain,
ntree = ntree,  mtry = mtry)
  err.oob[mtry] = rf.mtry$err.rate[ntree, 1]
}
plot(err.oob, type = "b", xlab = "no. predictors", ylab = "OBB
error")
grid()
```




```
## results for optimum RF
mtry = 3  # optimum value
rf.opt = randomForest(y.occ ~., df.occ, subset = indtrain,
ntree = ntree, mtry = mtry)
```

```
# OOB error for optimum RF?
rf.opt
```

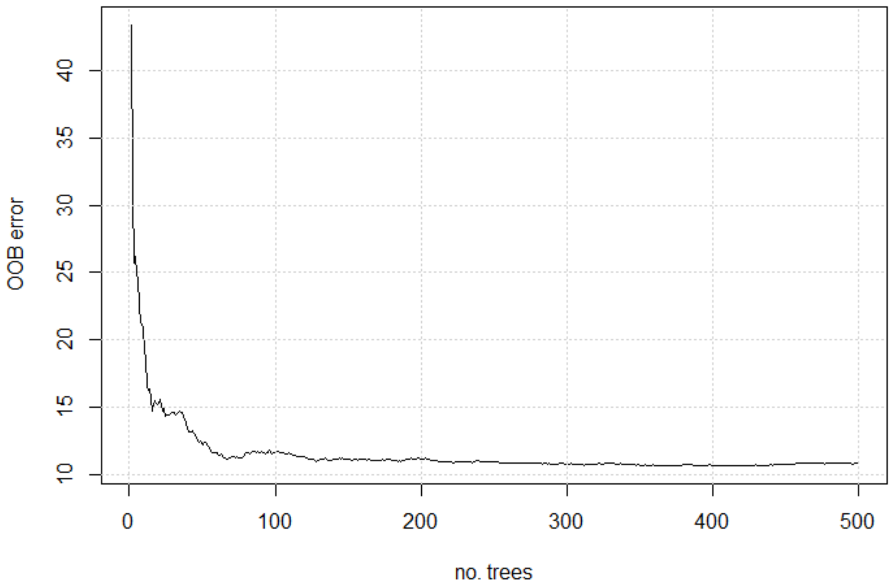
```
# test error for optimum RF?
pred = predict(rf.opt, df.occ[indtest, ])
1 - sum(diag(table(pred, df.occ$y.occ[indtest]))) /
length(indtest)
```

Random forest in R
Prediction problem (Boston)

```
library(MASS)

n = nrow(Boston)
# train/test partition
indtrain = sample(1:n, round(0.75*n))  # indices for train
indtest = setdiff(1:n, indtrain)  # indices for test
```

```
# RF
rf = randomForest(medv ~., Boston , subset = indtrain)
# RF configuration?
```

```
# OOB error?
plot(rf$mse, type = "l", xlab = "no. trees",
ylab = "OOB error"); grid()
```

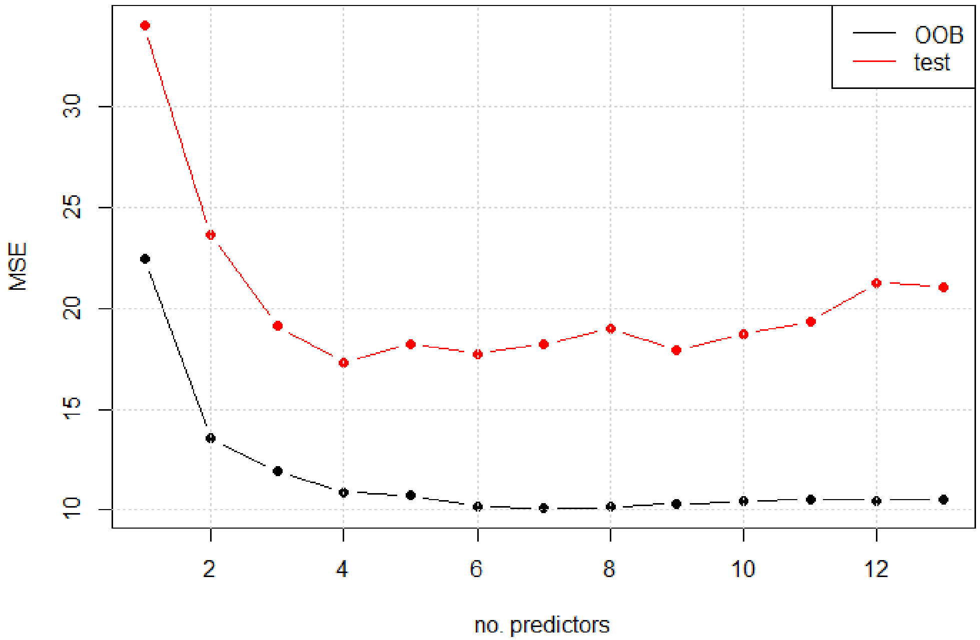



```
## fitting mtry
ntree = which(rf$mse == min(rf$mse))

# OOB error?
err.oob = c()
for (mtry in 1:13) {
  rf.mtry = randomForest(medv ~., Boston, subset = indtrain,
ntree = ntree, mtry = mtry)
  err.oob[mtry] = rf.mtry$mse[ntree]
}

# test error?
err.test = c()
for (mtry in 1:13) {
  rf.mtry = randomForest(medv ~., Boston, subset = indtrain,
ntree = ntree, mtry = mtry)
  pred.mtry = predict(rf.mtry, Boston[indtest, ])
  err.test[mtry] = mean((pred.mtry - Boston$medv[indtest])^2)
}
```

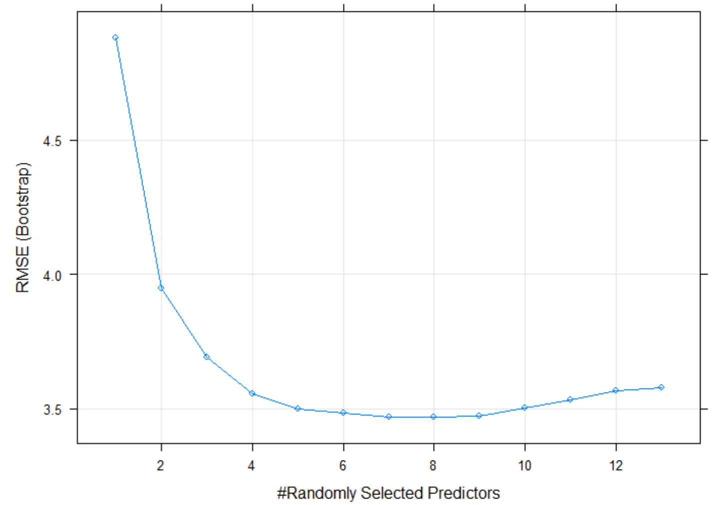
Random forest in R

Prediction problem (Boston)

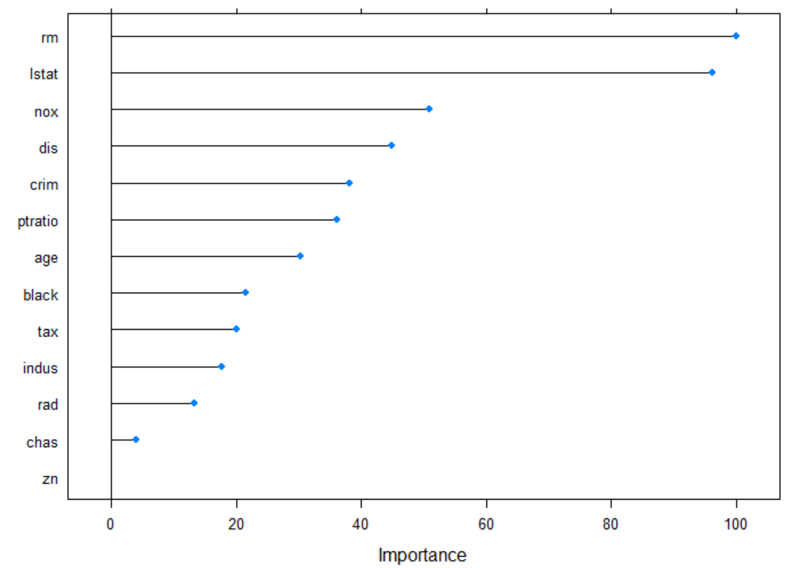
```
# OOB vs. test errors
matplot(1:13 , cbind(err.oob, err.test), type = "b", pch = 19 ,
        lty = 1, col = c("black", "red"),
        ylab = "MSE", xlab = "no. predictors")
grid()
legend("topright", c("OOB", "test"), lty = 1, col = c("black", "red"))
```




```
## fitting mtry with caret
library(caret)
rf.caret = train(medv ~., Boston, subset = indtrain,
          method = "rf", ntree = ntree,
          tuneGrid = expand.grid(mtry = 1:13))
plot(rf.caret)
```

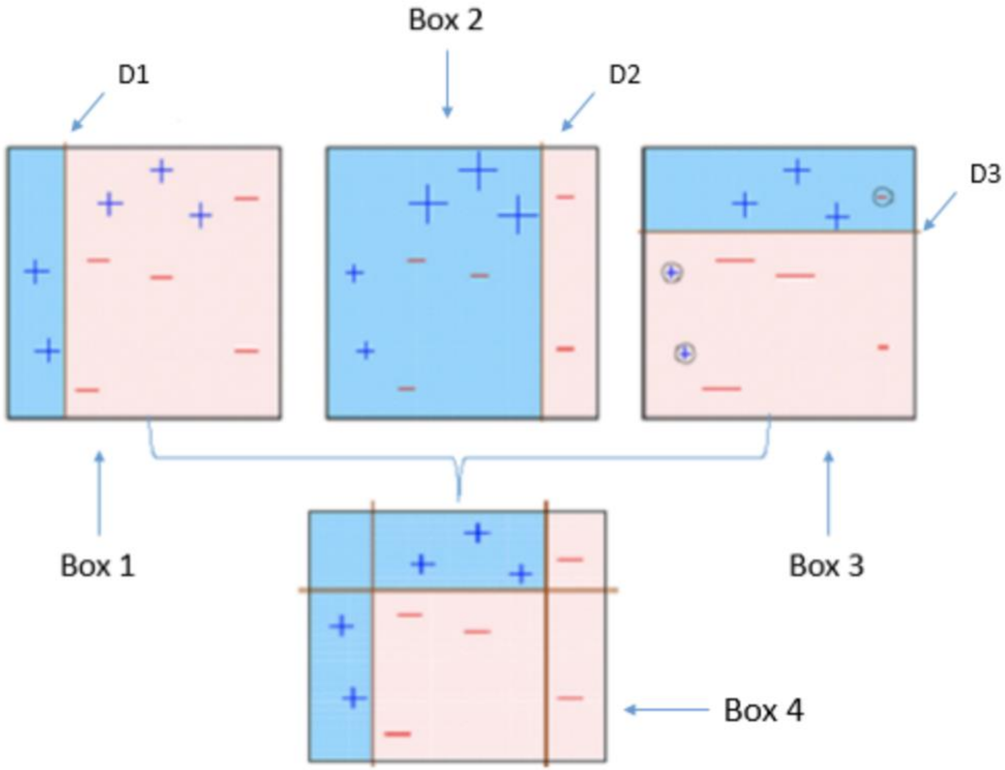



```
## predictors' importance
rf.opt = train(medv ~., Boston, subset = indtrain,
              method = "rf", ntree = ntree,
              tuneGrid = expand.grid(mtry = 6),
              importance = T)
plot(varImp(rf.opt))
```

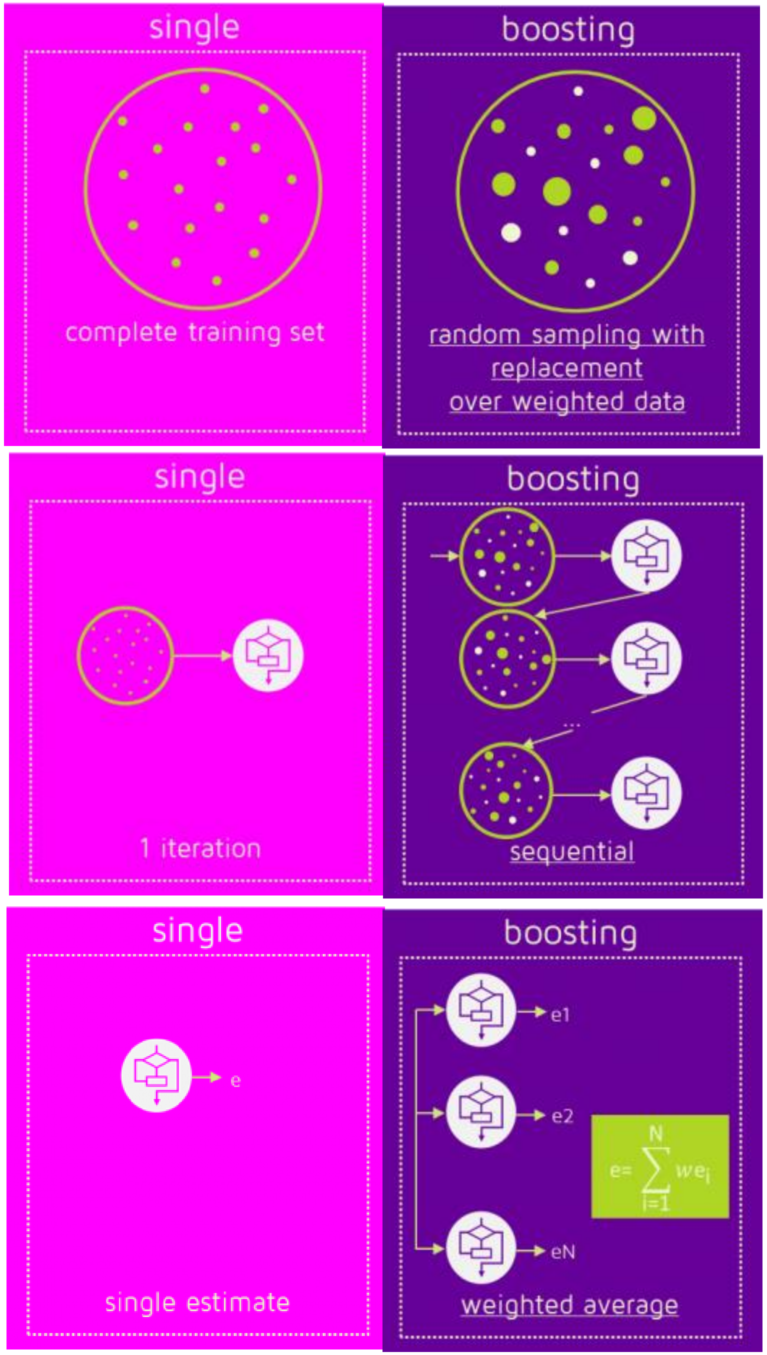


Boosting

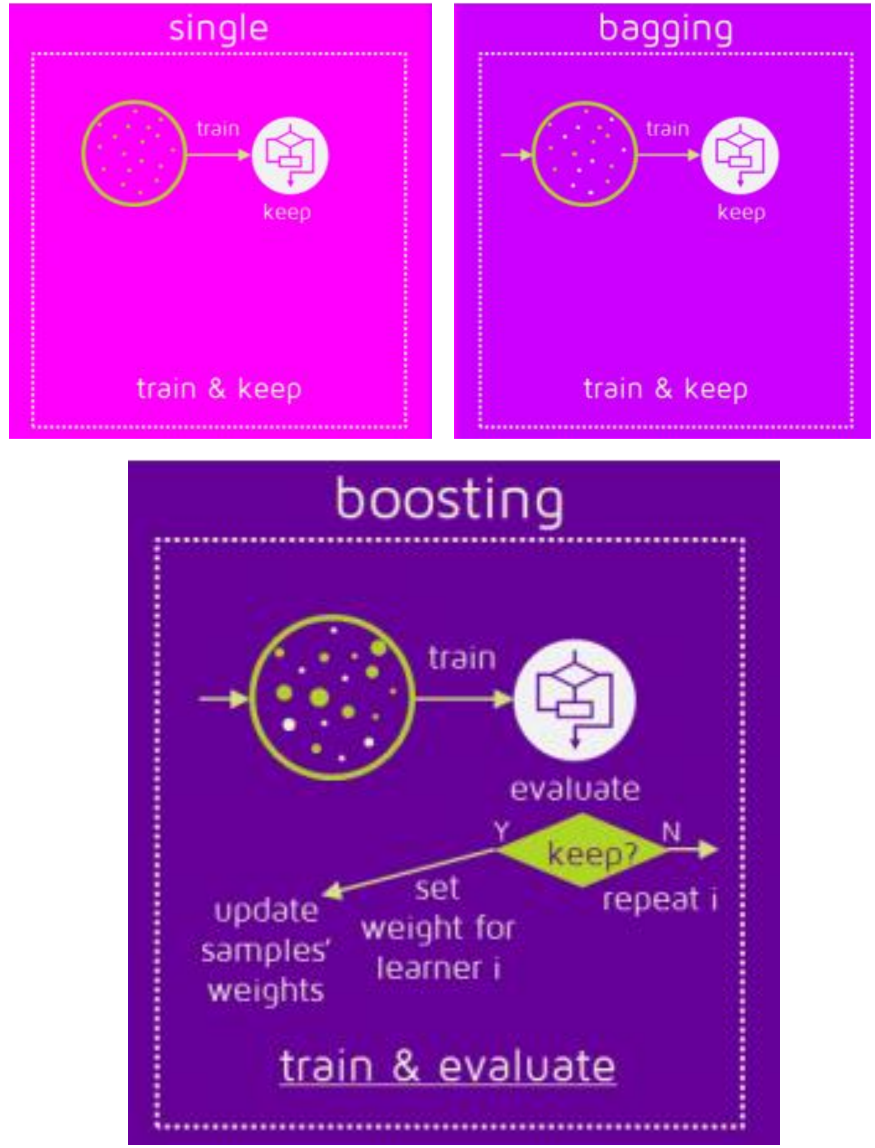
We saw that bagging operates in **parallel**. Differently, boosting procedures are **sequential**; i.e., each model run determines which elements the next model will focus on.



The algorithm allocates weights to each resulting model, depending on their individual performance. As in bagging, predictions for a new input data are based on the predictions resulting from the individual models, but taking into account these weights.



Boosting

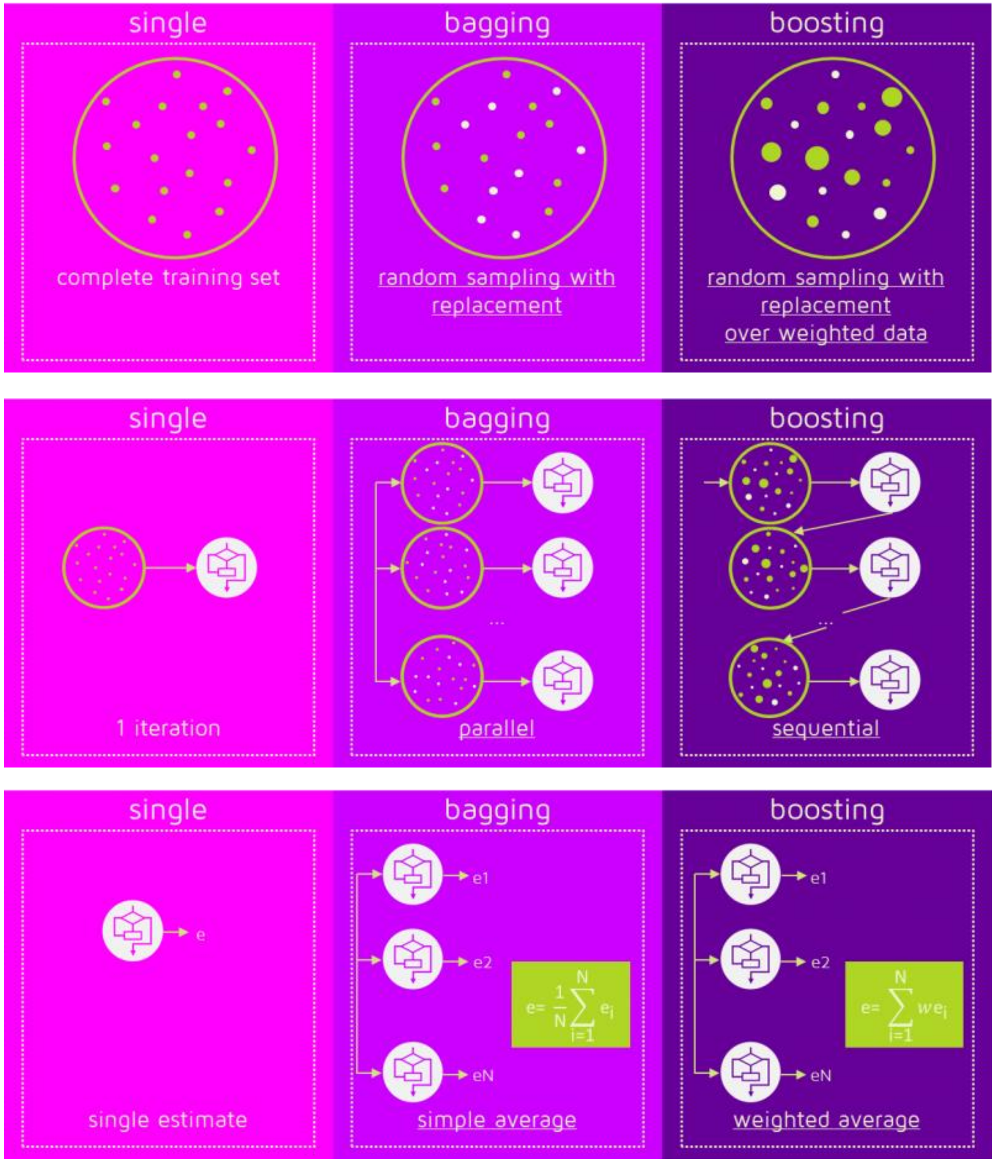


Some boosting techniques include an extra-condition to keep or discard an individual model. For example, in *AdaBoost* (the most popular), an error less than 50% is required to maintain the model; otherwise, the iteration is repeated until achieving a model better than a random guess.

Several alternatives for boosting exist with different ways to determine the weights to use in the next training step and as well as in the final combination stage: *LPBoost, XGBoost, GradientBoost, BrownBoost…*

Bagging and boosting

Similarities	Differences
Both are ensemble methods to get N learners from 1 learner…	… but, while they are built independently for Bagging, Boosting tries to add new models that do well where previous models fail.
Both generate several training data sets by random sampling…	… but only Boosting determines weights for the data to tip the scales in favor of the most difficult cases.
Both make the final decision by averaging the N learners (or taking the majority of them)…	… but it is an equally weighted average for Bagging and a weighted average for Boosting, more weight to those with better performance on training data.
Both are good at reducing variance and provide higher stability…	… but only Boosting tries to reduce bias. On the other hand, Bagging may solve the over-fitting problem, while Boosting can increase it.



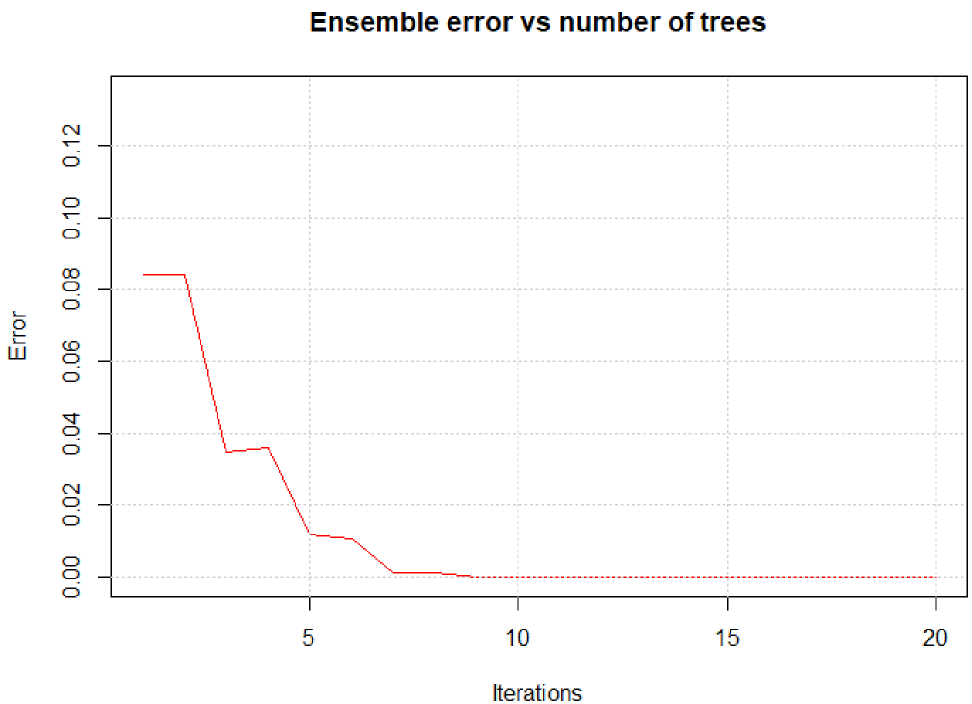
Boosting (AdaBoost) in R

Classification problem (rain/no rain)

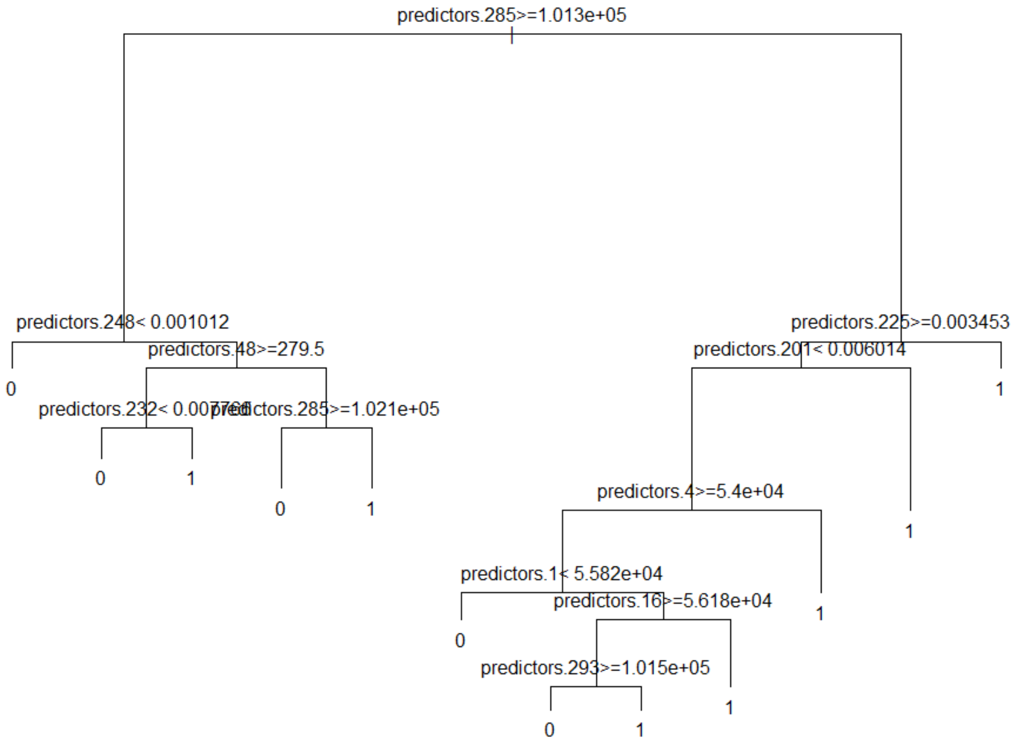
```
install.packages("adabag")
library(adabag)
```

```
# AdaBoost with 20 trees (mfinal)
ab = boosting(y.occ ~., df.occ[indtrain, ], mfinal = 20)
```

```
# train errors as a function of number of trees
plot(errorevol(ab, df.occ[indtrain, ]))
grid()
```




```
# we can pick and draw individual trees
plot(ab$trees[[1]])
text(ab$trees[[1]], pretty = F)
```




```
## prediction for test
pred.ab = predict(ab, df.occ[indtest, ])
# test error
1 - sum(diag(table(pred.ab$class, df.occ$y.occ[indtest]))) / length(indtest)
```