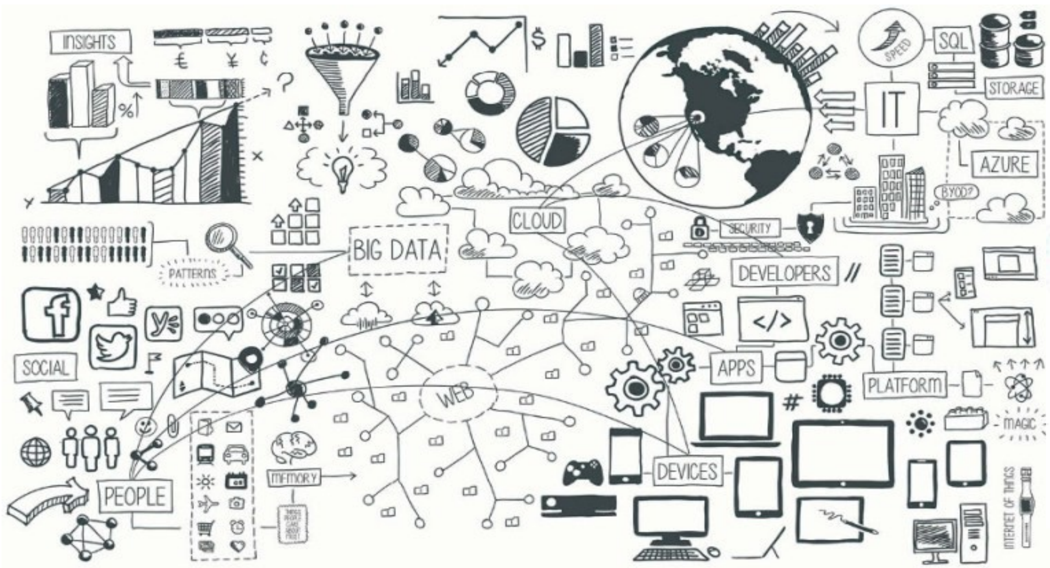
Data Mining (Minería de Datos)

Ensemble Methods: Bagging and Random Forests

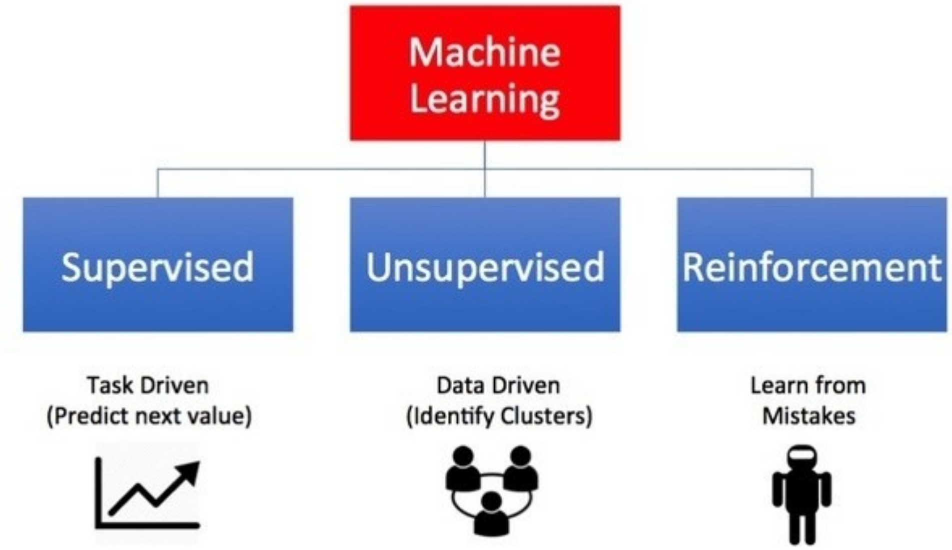


Sixto Herrera

Grupo de Meteorología

Univ. de Cantabria – CSIC
MACC / IFCA

Types of Machine Learning



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

Oct	29	Presentación, introducción y perspectiva histórica
	30	Paradigmas, problemas canónicos y data challenges
	31	Reglas de asociación
Nov	4	Practica: Reglas de asociación
	6	Evaluación, sobreajuste y crossvalidación
	11	Practica: Crossvalidación
	13	Árboles de clasificación y decisión
	18	Practica: Árboles de clasificación
	20	Técnicas de vecinos cercano (k-NN)
	25	Práctica: Vecinos cercanos
	27	Comparación de Técnicas de Clasificación.
Dic	2	Árboles de clasificación y regresión (CART)
	4	Práctica: Árboles de clasificación y regresión (CART)
	9	Practica: El paquete CARET
	11	**Ensembles: Bagging and Boosting**
	13	**Random Forests**
	16	**Gradient boosting**
	18	Práctica: XAI-Explainable Artificial Intelligence
Ene	8	Reducción de dimensión no lineal
	13	Reducción de dimensión no lineal
	15	Técnicas de agrupamiento
	20	Práctica: Técnicas de agrupamiento
	22	Predicción Condicionada
	24	Sesión de refuerzo/repaso.
	29	**Examen**

Types of Machine Learning

Machine

Oct	29	Presentación, introducción y perspectiva histórica
	30	Paradigmas, problemas canónicos y data challenges
	31	Reglas de asociación
Nov	4	Practica: Reglas de asociación
	6	Evaluación, sobreajuste y crossvalidación

Biomedical Signal Processing and Control 52 (2019) 456–462

Contents lists available at ScienceDirect

Biomedical Signal Processing and Control

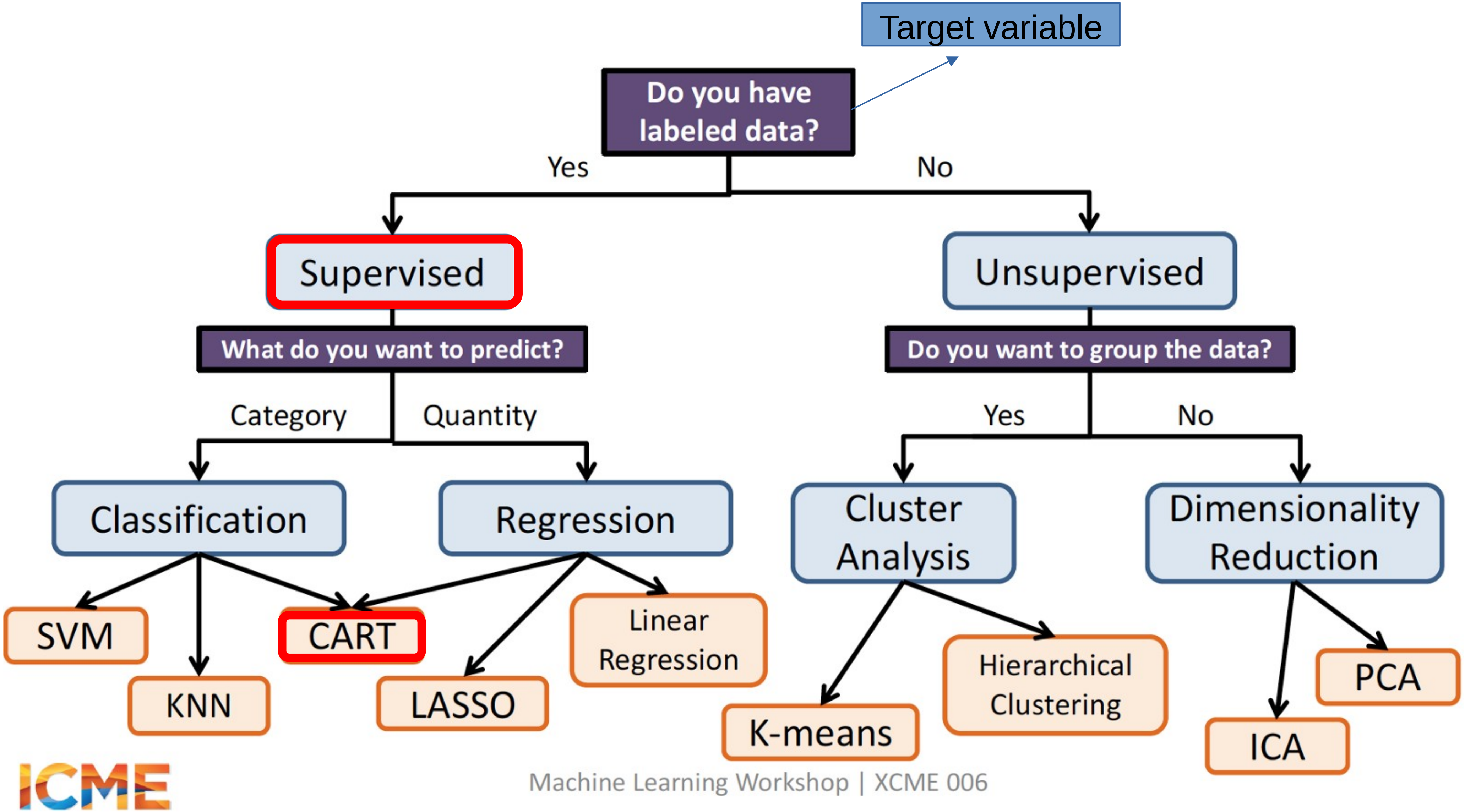
journal homepage: www.elsevier.com/locate/bspc

Decision tree and random forest models for outcome prediction in antibody incompatible kidney transplantation

Torgyn Shaikhina[a], Dave Lowe[b], Sunil Daga[d,e], David Briggs[c], Robert Higgins[e], Natasha Khovanova[a,*]

	24	Sesión de refuerzo/repaso.
	29	**Examen**

SYLLABUS



https://www.kdnuggets.com/2022/07/boosting-machine-learning-algorithms-overview.html

- **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 without the need to create dummy variables.
  - In general, they work fine on both classification and regression problems
- **Cons:**
  - Trees don’t have the same prediction accuracy as some of the more complicated approaches that we’ll see in this course.
  - Trees fail to deal with linear relationships.
  - Lack of smoothness.
  - Decision trees are very interpretable – as long as they are short. The number of terminal nodes increases quickly with depth.
  - Trees are also quite unstable. A few changes in the training dataset can create a completely different tree -> Trees are prone to overfitting.

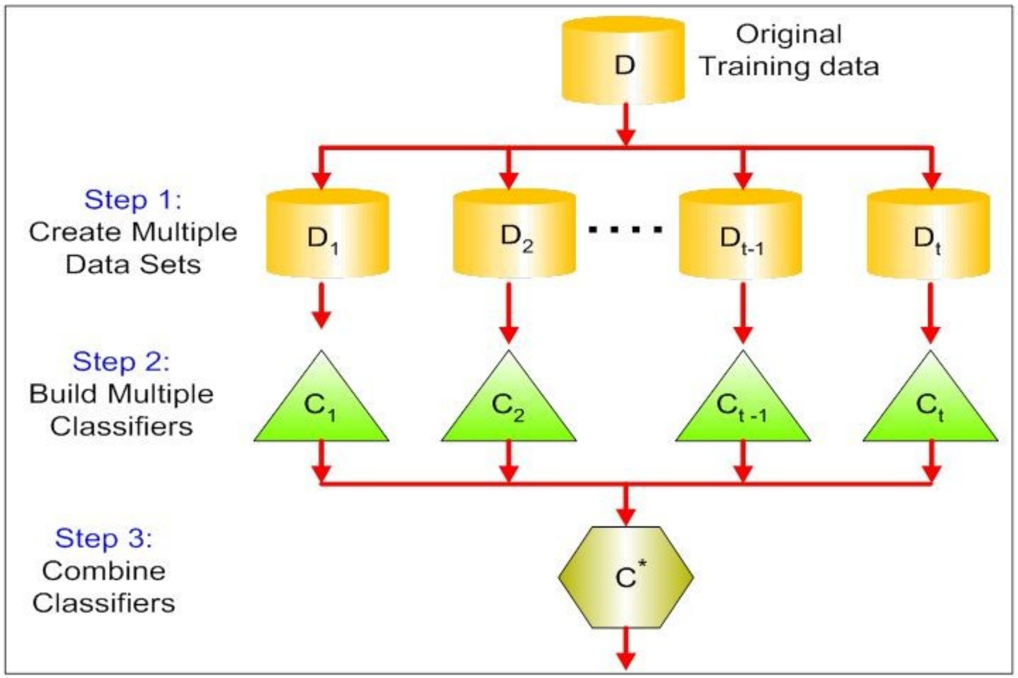
https://www.kdnuggets.com/2022/08/decision-trees-random-forests-explained.html

- **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 without the need to create dummy variables.
  - In general, they work fine on both classification and regression problems
- **Cons:**
  - Trees don't have the same prediction accuracy as some of the more complicated approaches that we'll see in this course.
  - Trees fail to deal with linear relationships.
  - Lack of smoothness.
  - Decision trees are very interpretable – as long as they are short. The number of terminal nodes increases quickly with depth.
  - Trees are also quite unstable. A few changes in the training dataset can create a completely different tree → **Trees are prone to overfitting** → **By aggregating many trees, the predictive performance of trees can be substantially improved.**

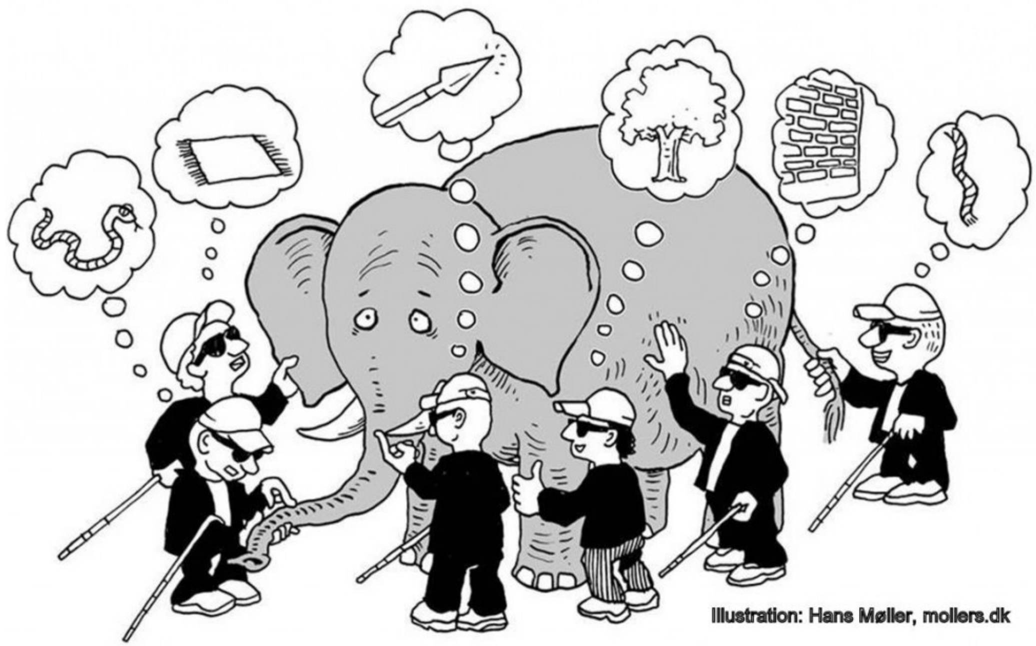
Ensemble Methods: Bagging (Random Forest), Boosting (Adaboost and Grandient Boosting) and Stacking

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

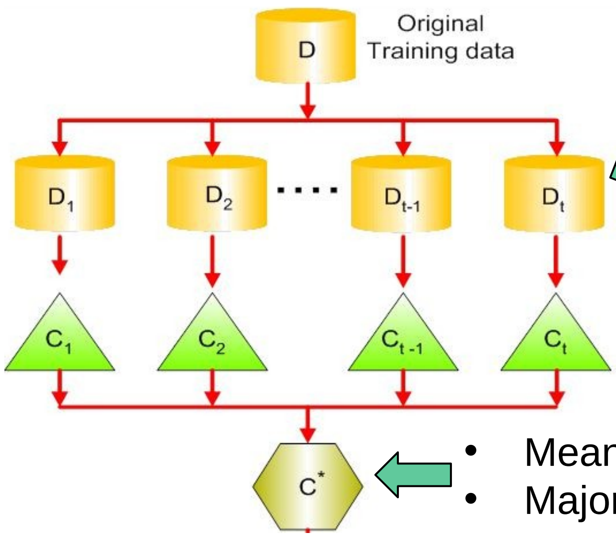
Ensemble learning

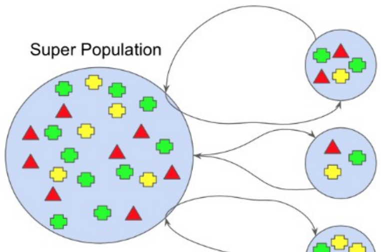
Ensemble approaches are typically used with CART.

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 **performance** improved.





bootstrapping

- Mean (prediction)
- Majority voting (classification)

Weak learners

High bias and low variance leads to overfitting

Low degree if freedom models
e.g. low depth trees

Ensemble learning

Ensemble approaches are typically used with CART.

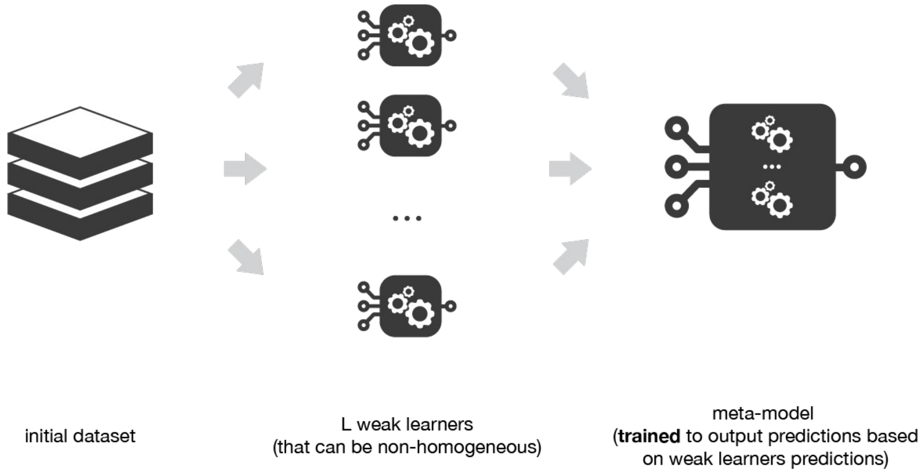
Pros

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.



Heterogenous Weak Learners

Stacking considers heterogeneous weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak models predictions.

https://stats.stackexchange.com/questions/290701/how-to-stack-machine-learning-models-in-r

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

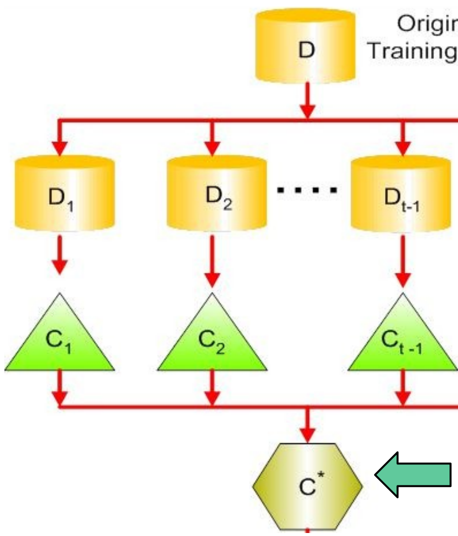
Ensemble learning

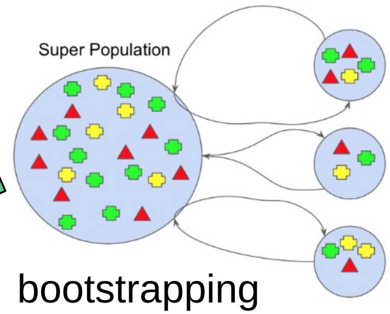
Ensemble approaches are typically used with CART.

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.

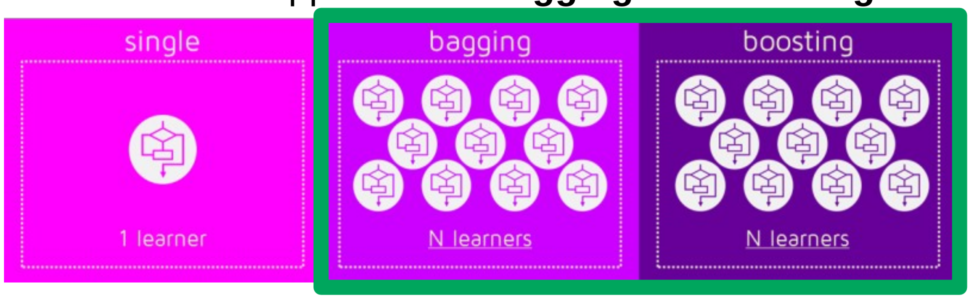




- Mean (prediction)
- Majority voting (classification)

Homogenous Weak Learners

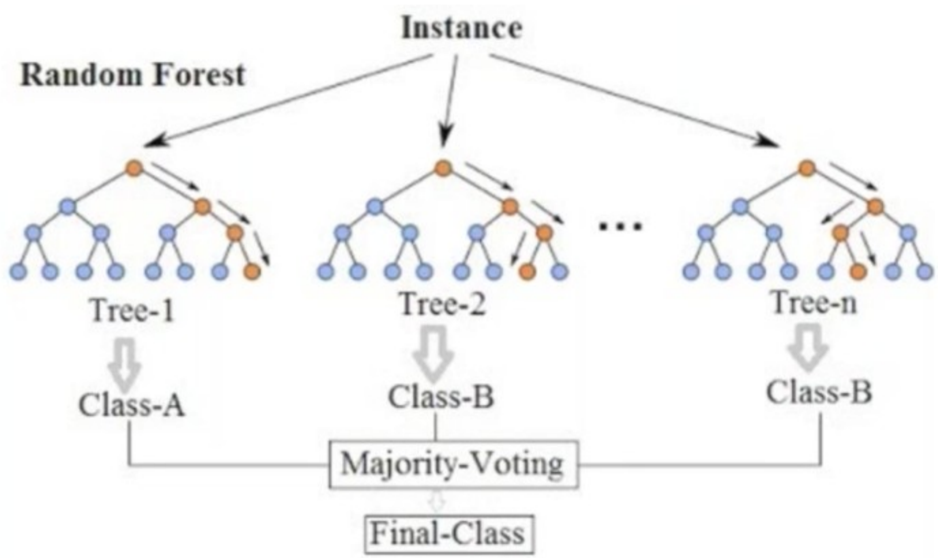
Most used approaches: **Bagging** and **boosting**

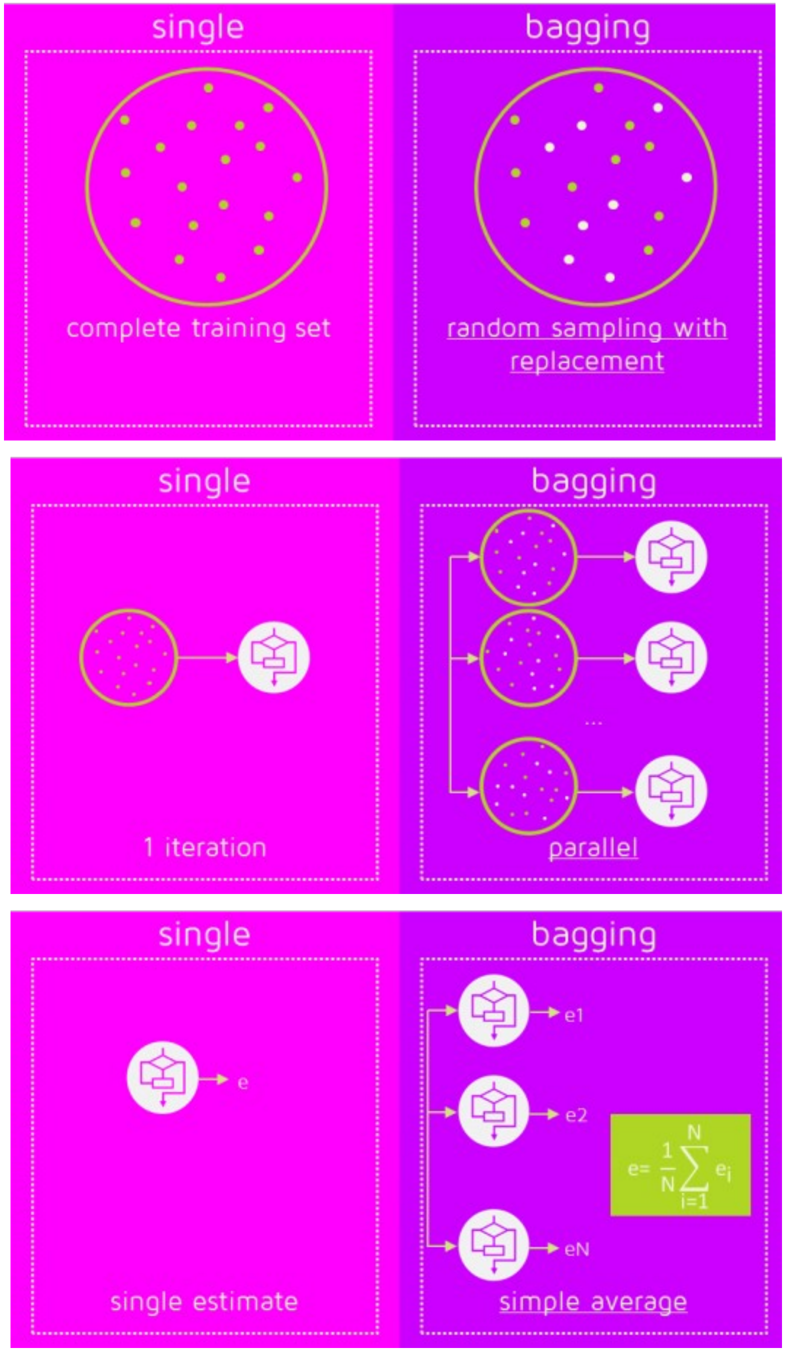


Bagging

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.**
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…).





Ensemble learning

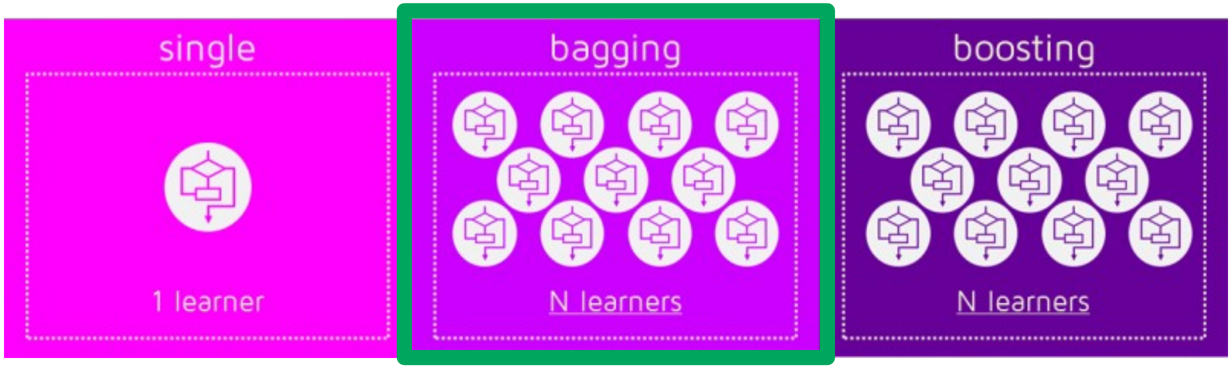
Ensemble approaches are typically used with CART.

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.

Weak learners



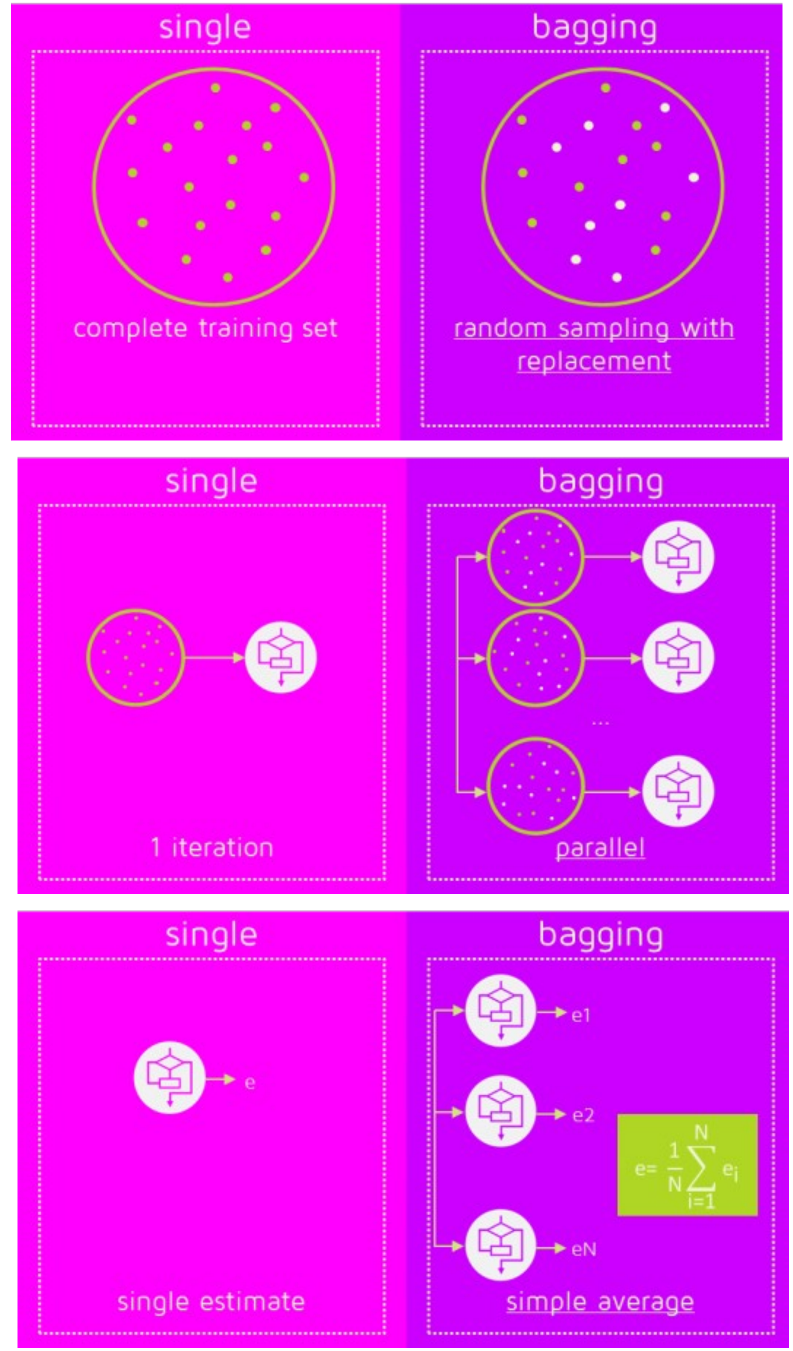
Low bias and high variance

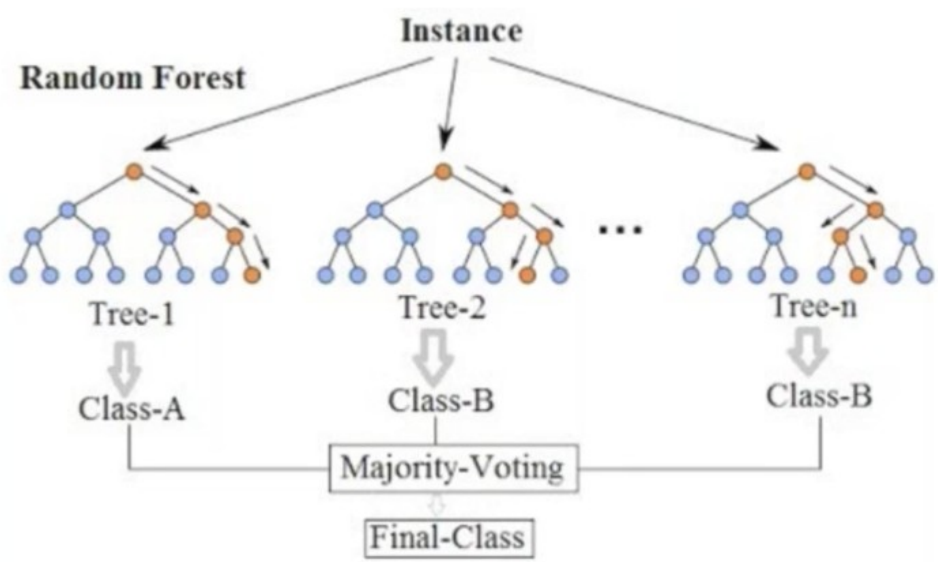
High bias and low variance

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

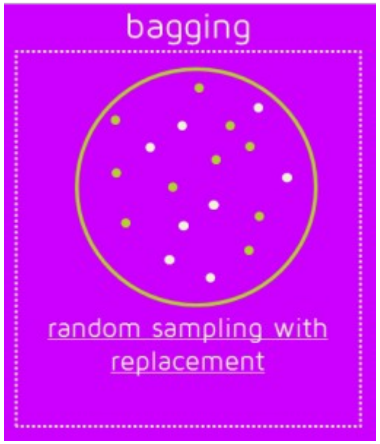
Random forest (RF) is an improvement over bagged trees.

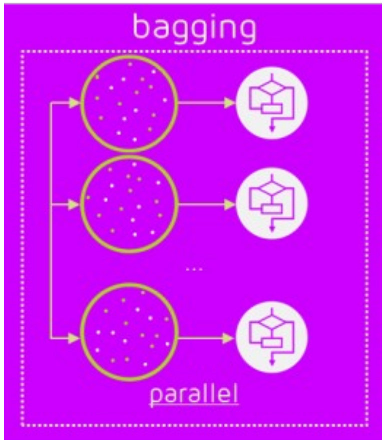
In CART, when selecting a split point, the learning algorithm is allowed to look through all predictor variables (p) in order to make the most division. Therefore, even with bagging, the individual trees can have a lot of structural similarities and in turn provide highly correlated predictions. However,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

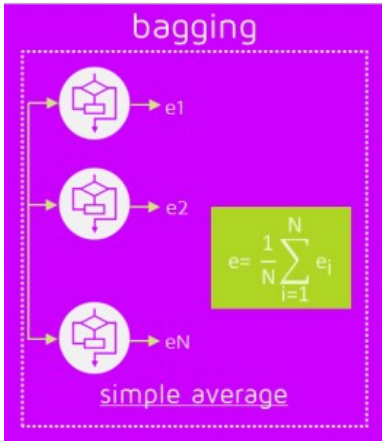
To solve this issue, in RF the learning algorithm is limited to a number of randomly selected predictors (m) at each splitting. Although m must be properly tuned, typical values for this parameter are:

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

Often, RF improve substantially the performance of individual trees.



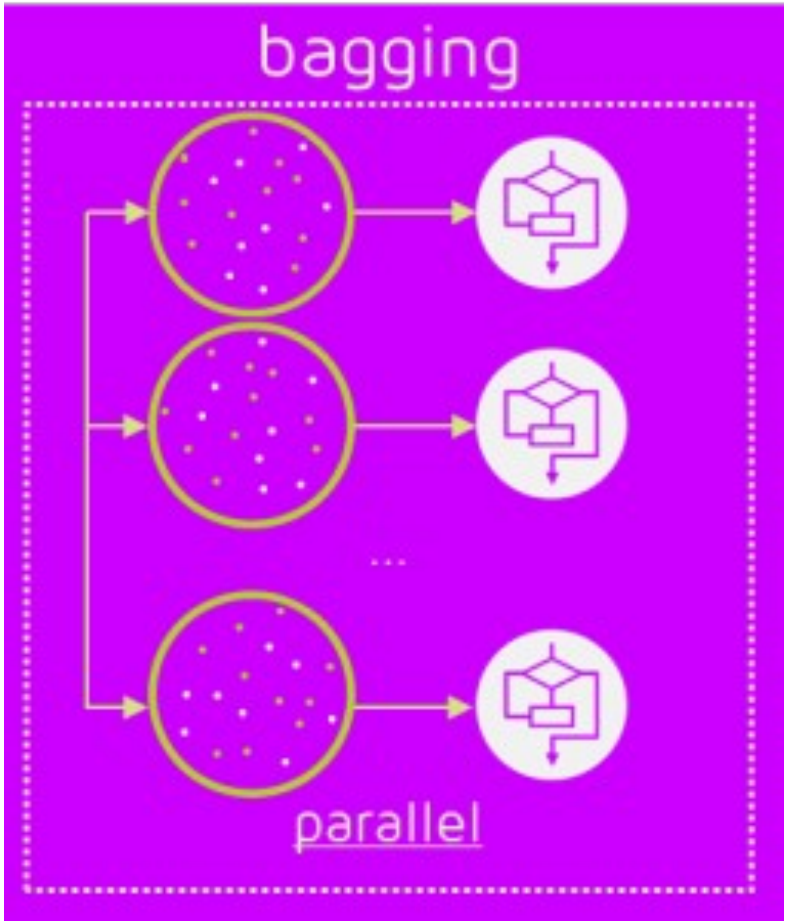




Bagging: Random forest

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 OOB 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.

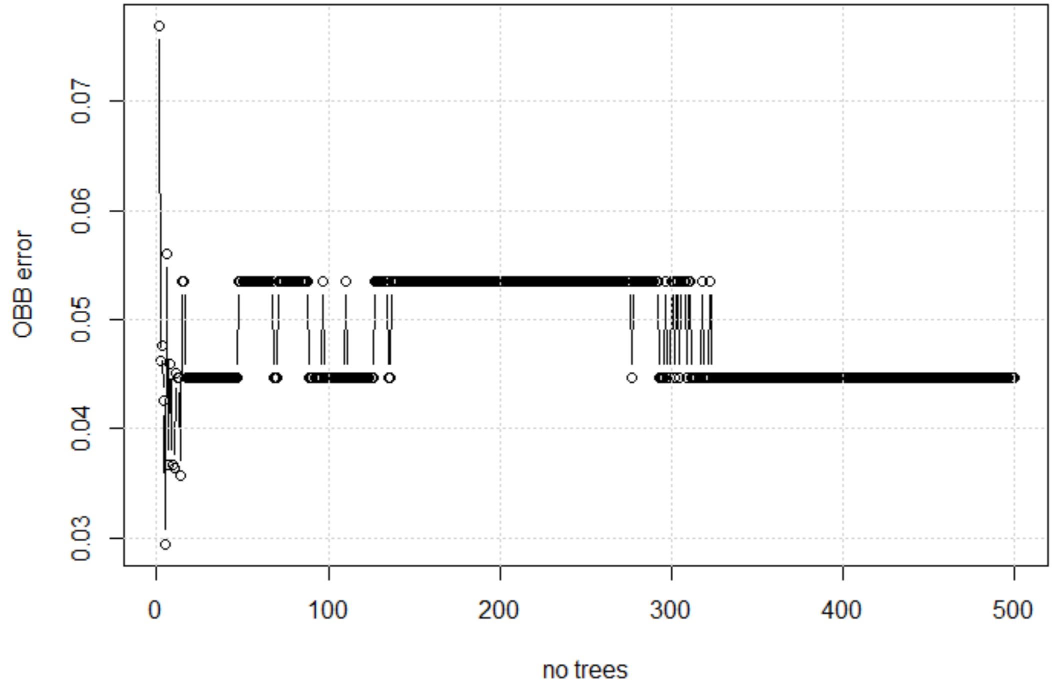
Random forest in R
Classification problem (iris)

```
rm(list = ls())
Install.packages("randomForest")
library(randomForest)
set.seed(42)
```

```
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 = "OOB 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)
```

Then, why random forest instead trees?

Random forest in R
Classification problem (iris)

Then, why random forest instead trees?

```
rfValue <- NULL
treeValue <- NULL
for (i in c(1:100)){
        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 <- randomForest(Species ~., iris , subset = indtrain, ntree=250)
        # prediction for test
        pred <- predict(rf, iris[indtest, ])
        # accuracy
        rfValue <- c(rfValue,sum(diag(table(pred, iris$Species[indtest]))) / length(indtest))
        # comparison with a single tree
        t = tree(Species ~., iris, subset = indtrain)
        # prediction for test
        pred.t = predict(t, iris[indtest, ], type = "class")
        # accuracy
        treeValue <- c(treeValue,sum(diag(table(pred.t, iris$Species[indtest]))) / length(indtest))
}
mean(treeValue)
mean(rfValue)
sd(treeValue)
sd(rfValue)
```

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, ]
```

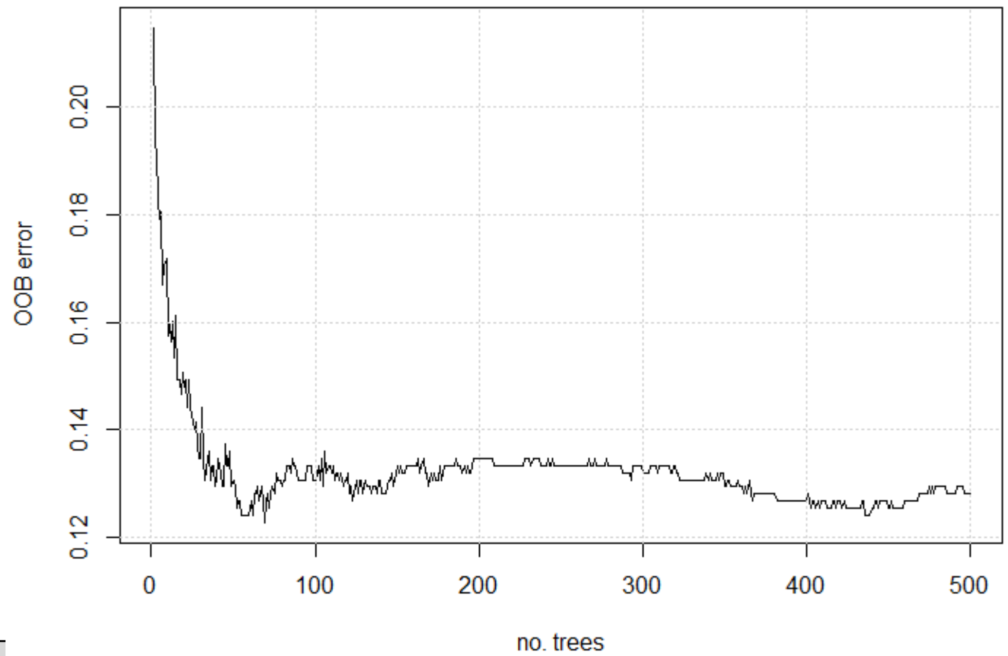
```
# train/test partition
indtrain = sample(1:n, round(0.75*n))  # indices for train
indtest = setdiff(1:n, indtrain)  # indices for test
```

```
# 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()
```



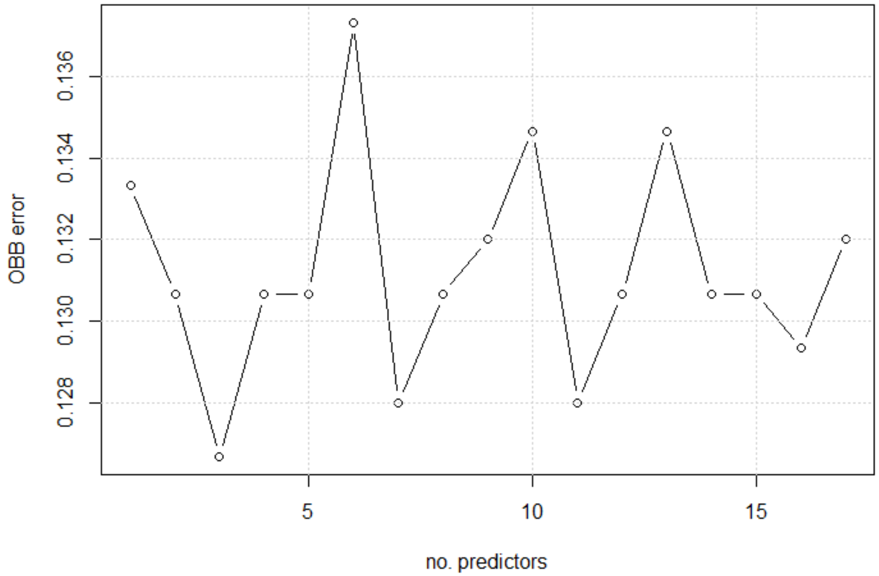

```
# test error?
pred = predict(rf, df.occ[indtest, ])
1 - sum(diag(table(pred, df.occ$y.occ[indtest]))) /
length(indtest)  # error (1-accuracy)
```

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)
```

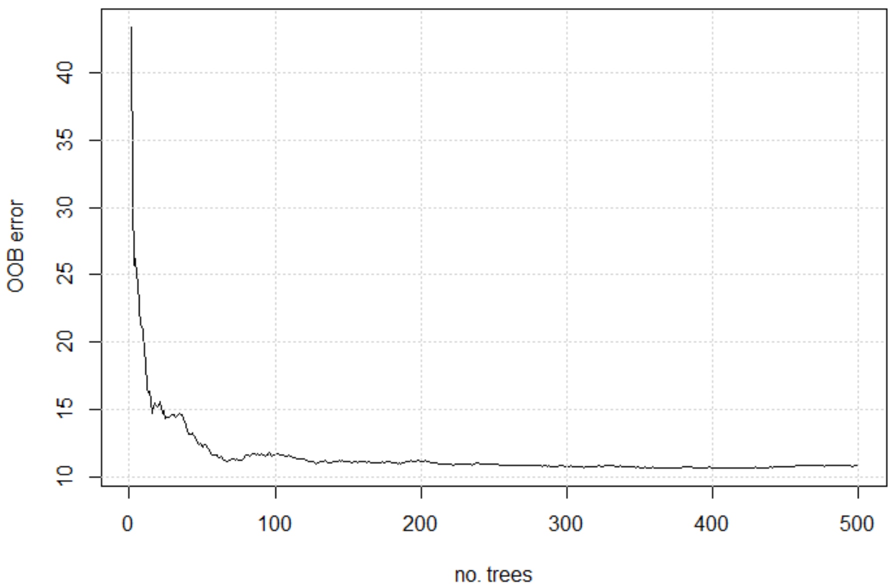
Random forest in R

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?
```

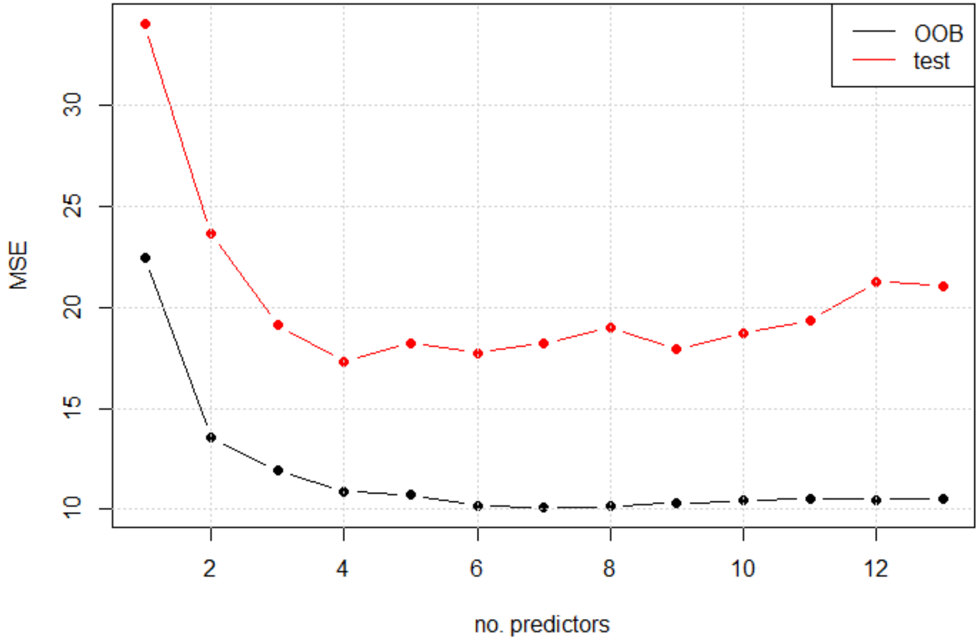
```
# OOB error?
plot(rf$mse, type = "l", xlab = "no. trees",
ylab = "OOB error"); grid()
```



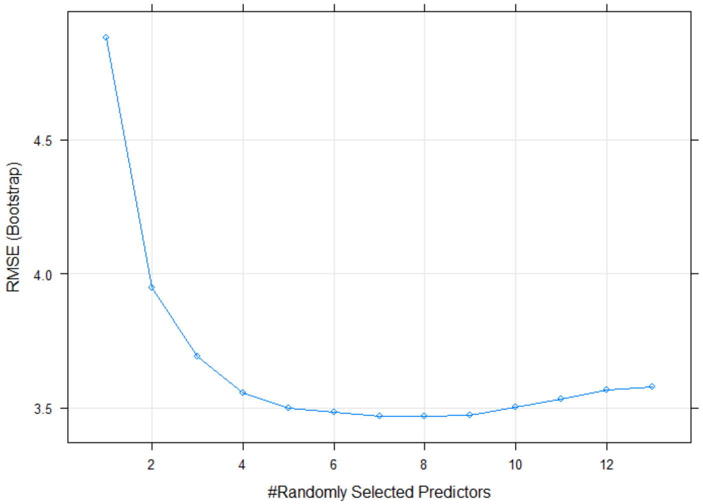

```
## fitting mtry
ntree = which(rf$mse == min(rf$mse))

# 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)

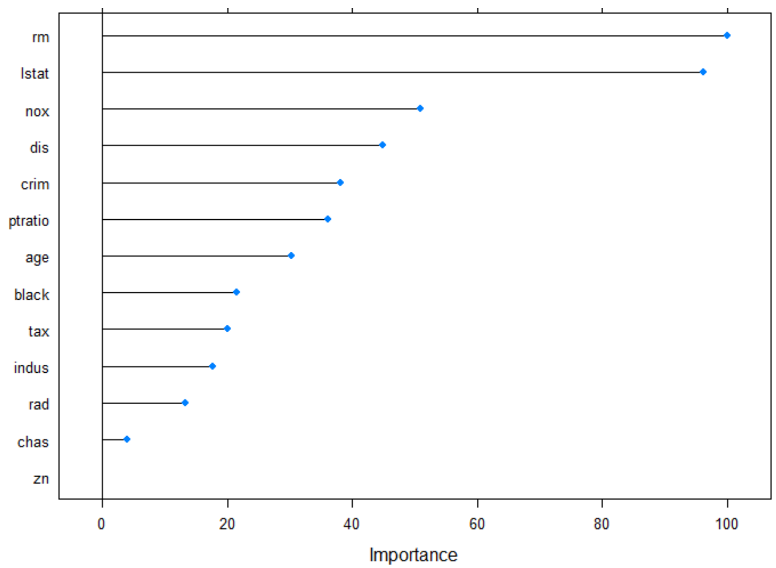
```
# OOB vs. test errors
matplot(1:13 , cbind(err.oob, err.test), type = "b", pch = 19 ,
        lty = 1, col = c("black", "red"),
        ylab = "MSE", xlab = "no. predictors")
grid()
legend("topright", c("OOB", "test"), lty = 1, col = c("black", "red"))
```




```
## 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)
```

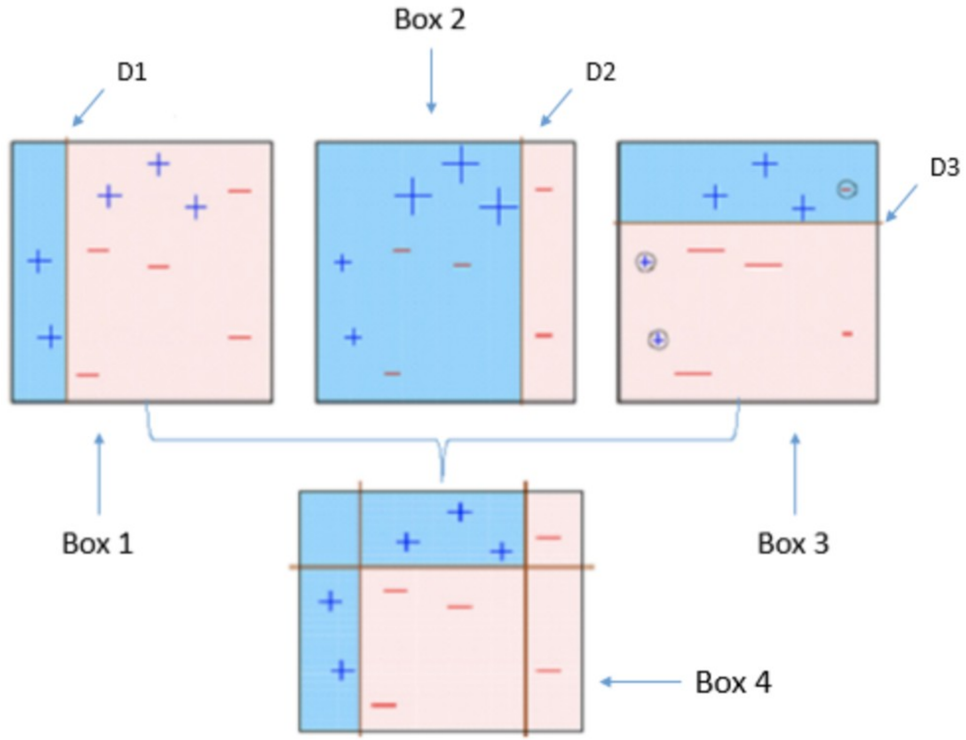



```
## predictors' importance
rf.opt = train(medv ~., Boston, subset = indtrain,
                method = "rf", ntree = ntree,
                tuneGrid = expand.grid(mtry = 6),
                importance = T)
plot(varImp(rf.opt))
```

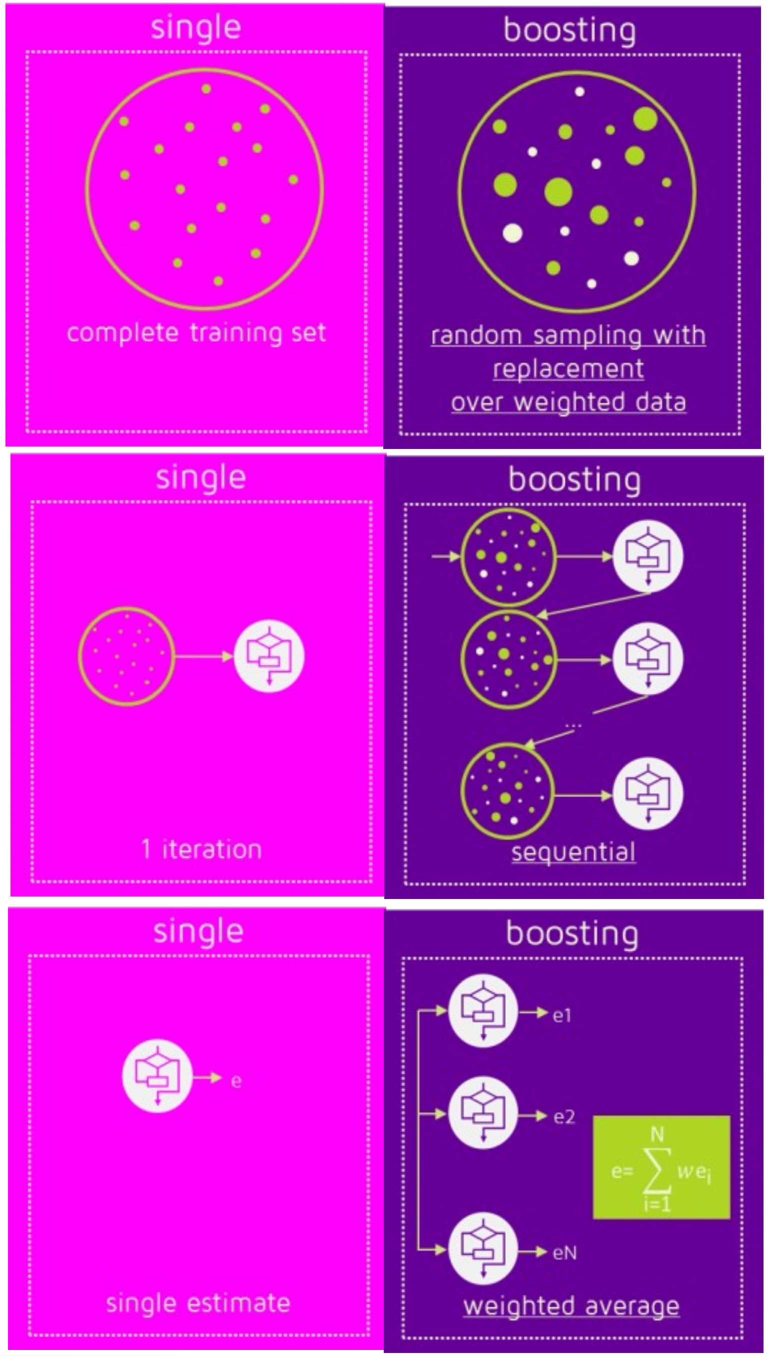


Boosting

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.



Ensemble learning

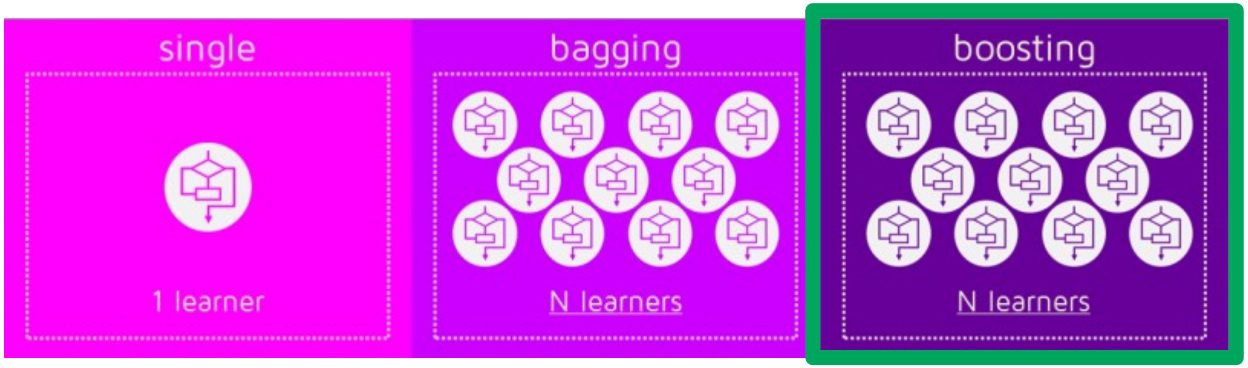
Ensemble approaches are typically used with CART.

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.

Weak learners



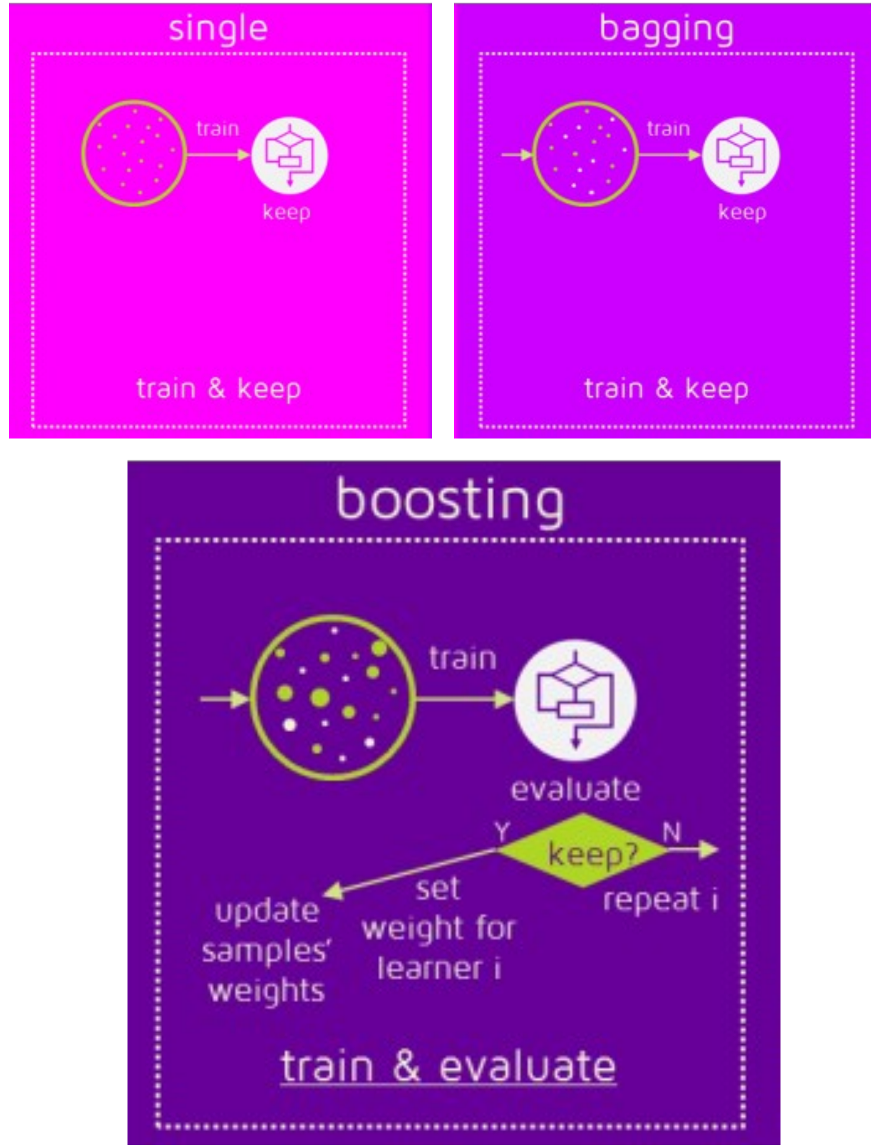
Low bias and high variance

High bias and low variance

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.

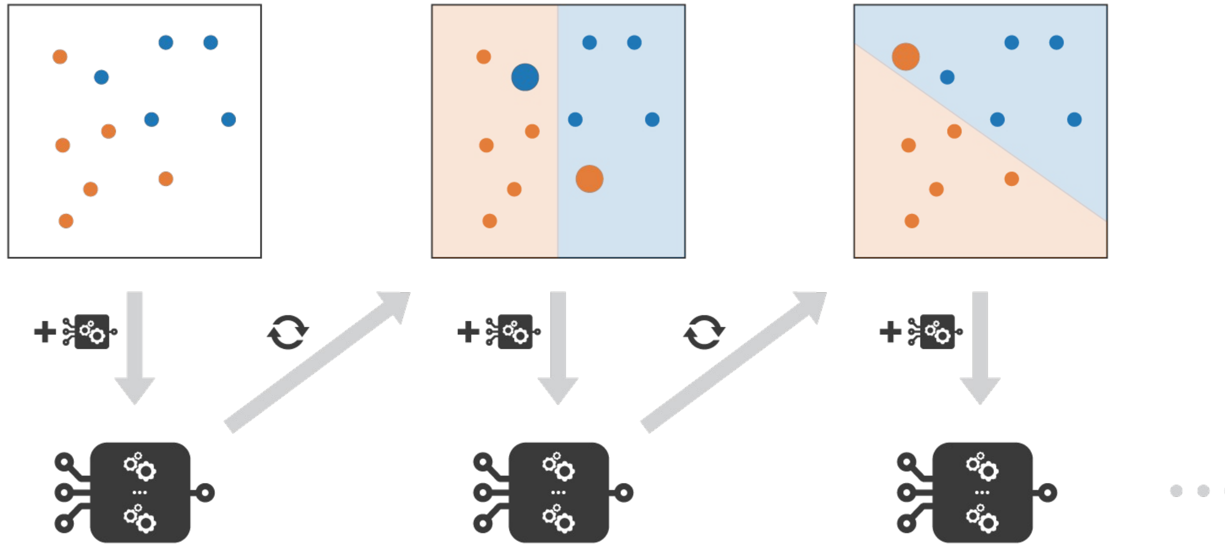
Several alternatives for boosting exist with different ways to determine the weights to use in the next training step and as well as in the final combination stage: *LPBoost, XGBoost, GradientBoost, BrownBoost…*



Adaptative Boosting (AdaBoost)

Step 1: All the observations have the **same weights**

initial setting: all the observations have the same weight



Repeat (1:L):

a) Fit the weak model considering the observations weights.
b) Evaluate the weak learner to obtain its coefficient.
c) Update the strong learner adding the weak learner.
d) Update the obervations weights

Result: A strong learner is obtained as a simple linear combination of weak learners weighted by coefficients expressing the performance of each learner. Variants of this algorithm could be obtained by modifying the loss function (e.g. logit for classification or L2 for regression).

$$s_L(.) = \sum_{l=1}^{L} c_l \times w_l(.) \qquad \text{where } c_l\text{'s are coefficients and } w_l\text{'s are weak learners}$$

$$(c_l, w_l(.)) = \underset{c,w(.)}{\arg\min} E(s_{l-1}(.) + c \times w(.)) = \underset{c,w(.)}{\arg\min} \sum_{n=1}^{N} e(y_n, s_{l-1}(x_n) + c \times w(x_n))$$

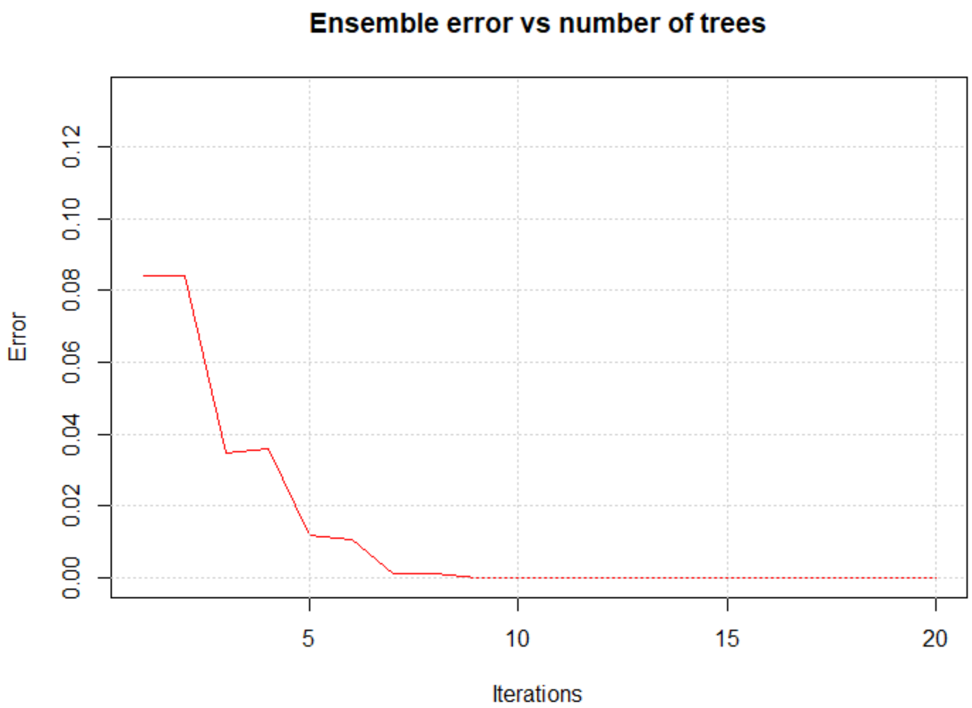
Boosting (AdaBoost) in R

Classification problem (rain/no rain)

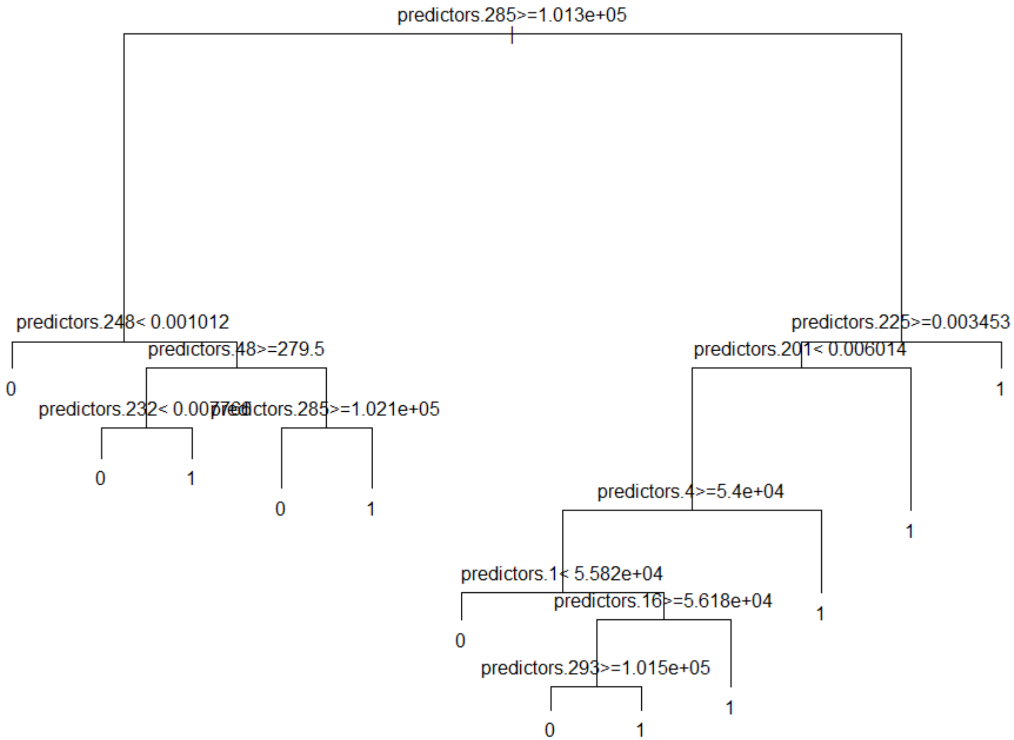
```
install.packages("adabag")
library(adabag)
```

```
# 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()
```



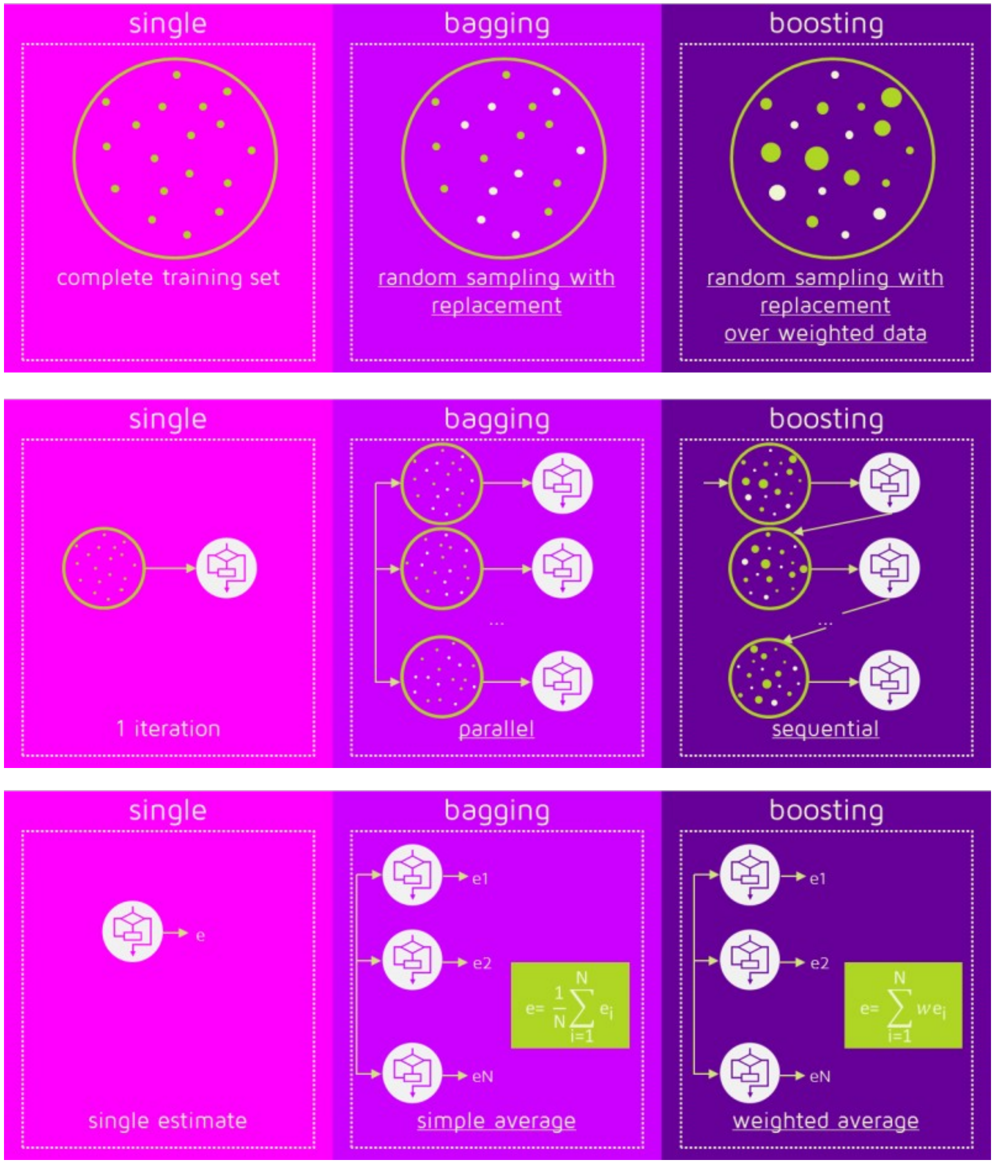

```
# we can pick and draw individual trees
plot(ab$trees[[1]])
text(ab$trees[[1]], pretty = F)
```




```
## 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)
```

Bagging and boosting

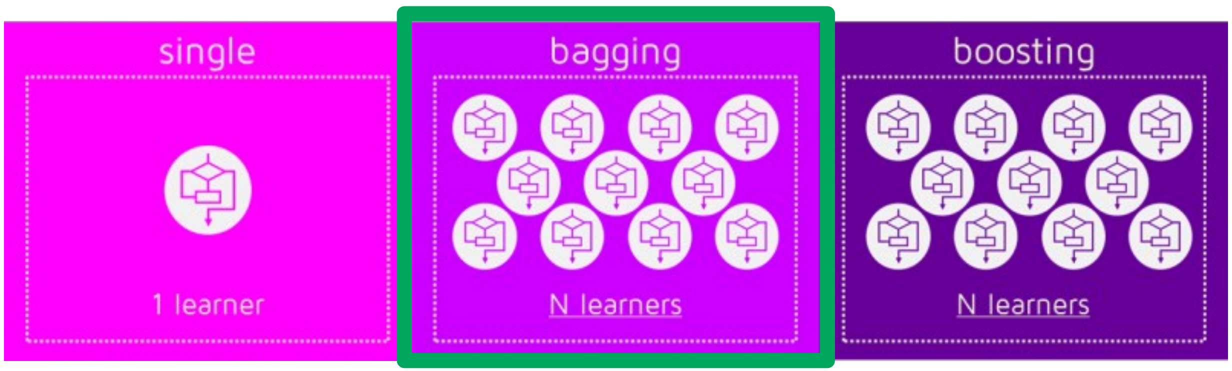
Similarities	Differences
Both are ensemble methods to get N learners from 1 learner…	… but, while they are built independently for Bagging, Boosting tries to add new models that do well where previous models fail.
Both generate several training data sets by random sampling…	… but only Boosting determines 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.



- **Pros:**
  - Ensembling is a proven method for improving the accuracy of the model and works in most of the cases.
  - It is the key ingredient for winning almost all of the machine learning hackathons.
  - Ensembling makes the model more robust and stable thus ensuring decent performance on the test cases in most scenarios.
  - You can use ensembling to capture linear and simple as well non-linear complex relationships in the data. This can be done by using two different models and forming an ensemble of two.
- **Cons:**
  - Ensembling reduces the model interpretability and makes it very difficult to draw any crucial business insights at the end.
  - It is time-consuming and thus might not be the best idea for real-time applications.
  - The selection of models for creating an ensemble is an art which is really hard to master.

https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/

```
## Example: https://machinelearningmastery.com/machine-learning-ensembles-with-r/
# Load libraries
library(mlbench)
library(caret)
library(caretEnsemble)

# Load the dataset: This dataset describes high-frequency antenna returns from high energy particles in the atmosphere and whether
# the return shows structure or not. The problem is a binary classification that contains 351 instances and 35 numerical attributes.
# https://archive.ics.uci.edu/ml/datasets/Ionosphere
data(Ionosphere)
dataset <- Ionosphere
dataset <- dataset[,-2] # The second variable is constant and it has been removed
dataset$V1 <- as.numeric(as.character(dataset$V1)) # It is a factor but it is converted to numeric
head(dataset)
```

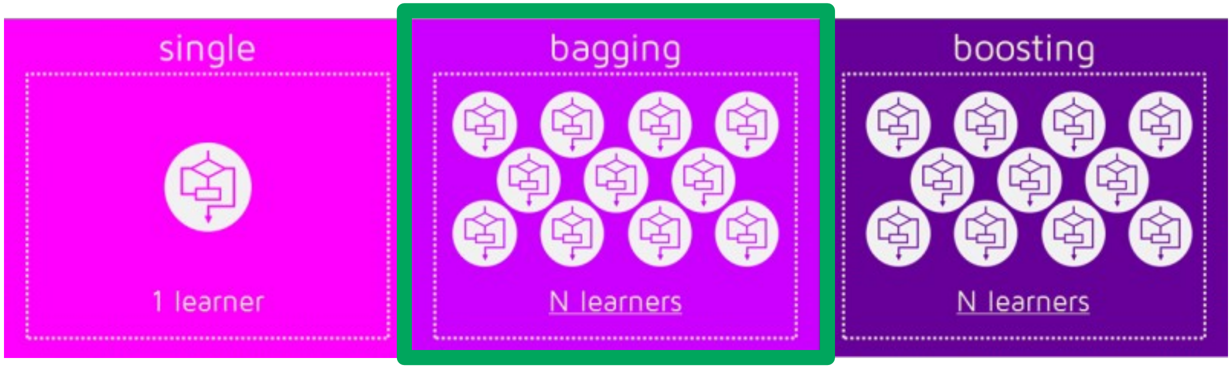


Weak learners

Low bias and high variance

High bias and low variance

```
## Example: https://machinelearningmastery.com/machine-learning-ensembles-with-r/
## Example of Bagging algorithms
control <- trainControl(method="repeatedcv", number=10, repeats=3)
seed <- 7
metric <- "Accuracy"
# Bagged CART
set.seed(seed)
fit.treebag <- train(Class~., data=dataset, method="treebag", metric=metric, trControl=control)
# Random Forest
set.seed(seed)
fit.rf <- train(Class~., data=dataset, method="rf", metric=metric, trControl=control)
# summarize results
bagging_results <- resamples(list(treebag=fit.treebag, rf=fit.rf))
summary(bagging_results)
dotplot(bagging_results)
```



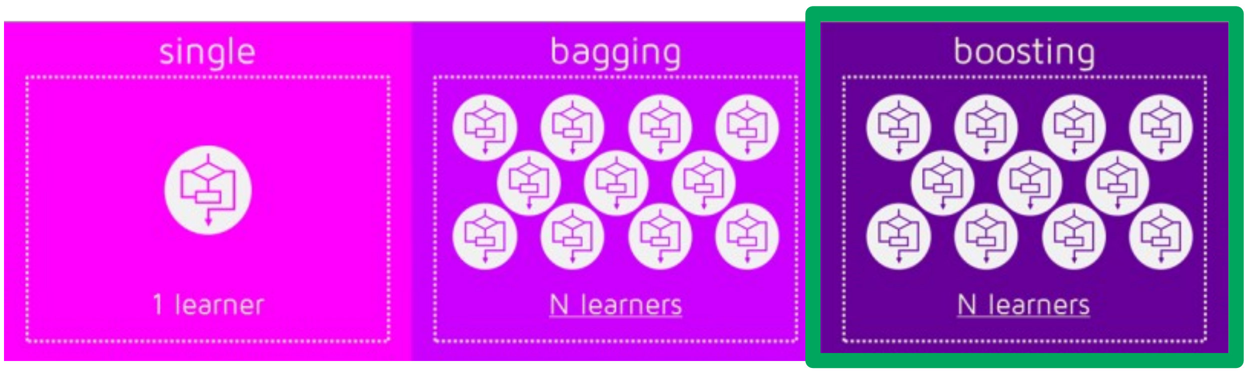
Weak learners

Low bias and high variance

High bias and low variance

```
## Example: https://machinelearningmastery.com/machine-learning-ensembles-with-r/
## Example of Boosting Algorithms
control <- trainControl(method="repeatedcv", number=10, repeats=3)
seed <- 7
metric <- "Accuracy"
# C5.0
set.seed(seed)
fit.c50 <- train(Class~., data=dataset, method="C5.0", metric=metric, trControl=control)
# Stochastic Gradient Boosting
set.seed(seed)
fit.gbm <- train(Class~., data=dataset, method="gbm", metric=metric, trControl=control, verbose=FALSE)
# summarize results
boosting_results <- resamples(list(c5.0=fit.c50, gbm=fit.gbm))
summary(boosting_results)
dotplot(boosting_results)
```

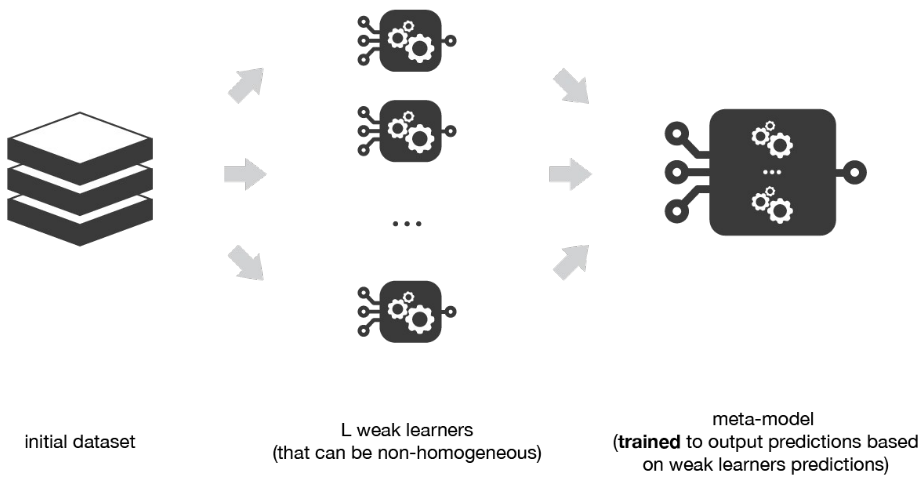
Weak learners



Low bias and high variance

High bias and low variance

```
# Example of Stacking algorithms
# create submodels
control <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE, classProbs=TRUE)
algorithmList <- c('rpart', 'glm', 'knn') ## algorithmList <- c('lda', 'rpart', 'glm', 'knn', 'svmRadial')
set.seed(seed)
models <- caretList(Class~., data=dataset, trControl=control, methodList=algorithmList)
results <- resamples(models)
summary(results)
dotplot(results)

# correlation between results
modelCor(results)
splom(results)
```



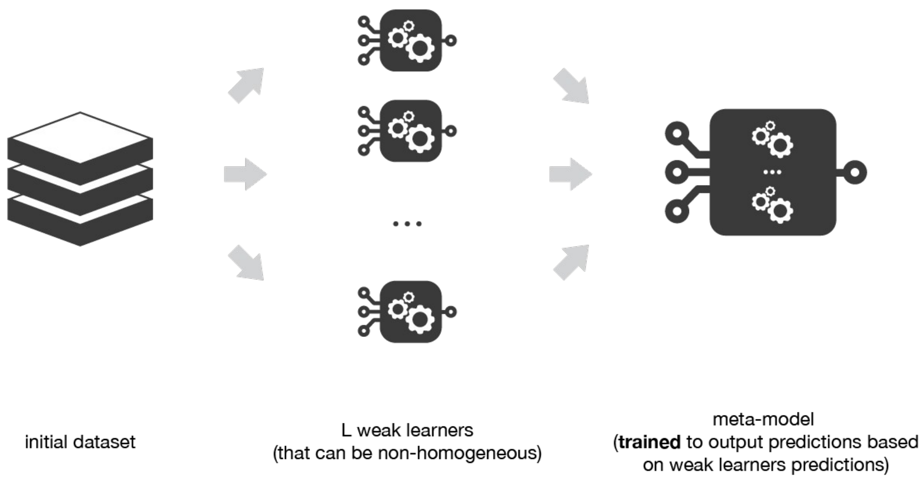
Heterogenous Weak Learners

Stacking considers heterogeneous weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak models predictions.

https://stats.stackexchange.com/questions/290701/how-to-stack-machine-learning-models-in-r

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

```
# Example of Stacking algorithms
# correlation between results
modelCor(results)
splom(results)

# stack using glm
stackControl <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE, classProbs=TRUE)
set.seed(seed)
stack.glm <- caretStack(models, method="glm", metric="Accuracy", trControl=stackControl)
print(stack.glm)

# stack using random forest
set.seed(seed)
stack.rf <- caretStack(models, method="rf", metric="Accuracy", trControl=stackControl)
print(stack.rf)
```



Heterogenous Weak Learners

Stacking considers heterogeneous weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak models predictions.

https://stats.stackexchange.com/questions/290701/how-to-stack-machine-learning-models-in-r

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205