

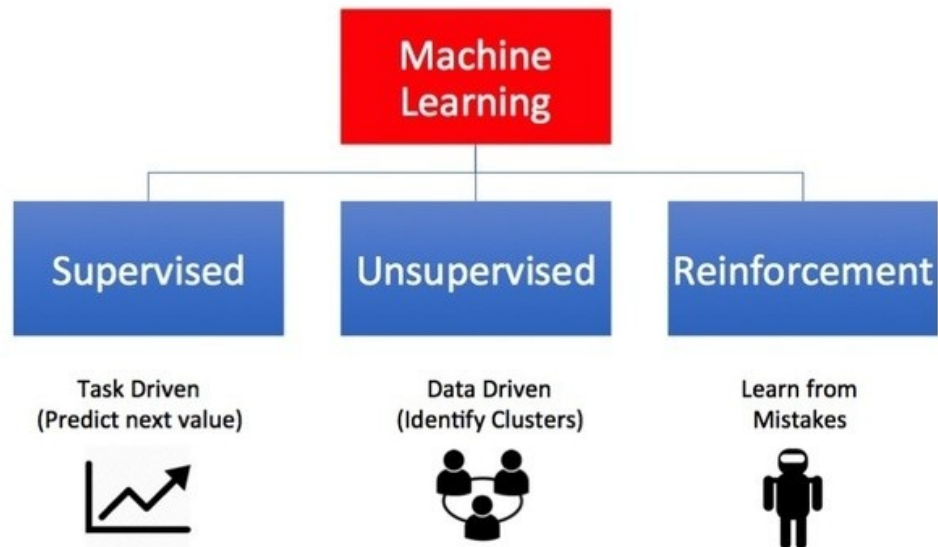
Ensembles: Gradient Boosting



Grupo de Meteorología

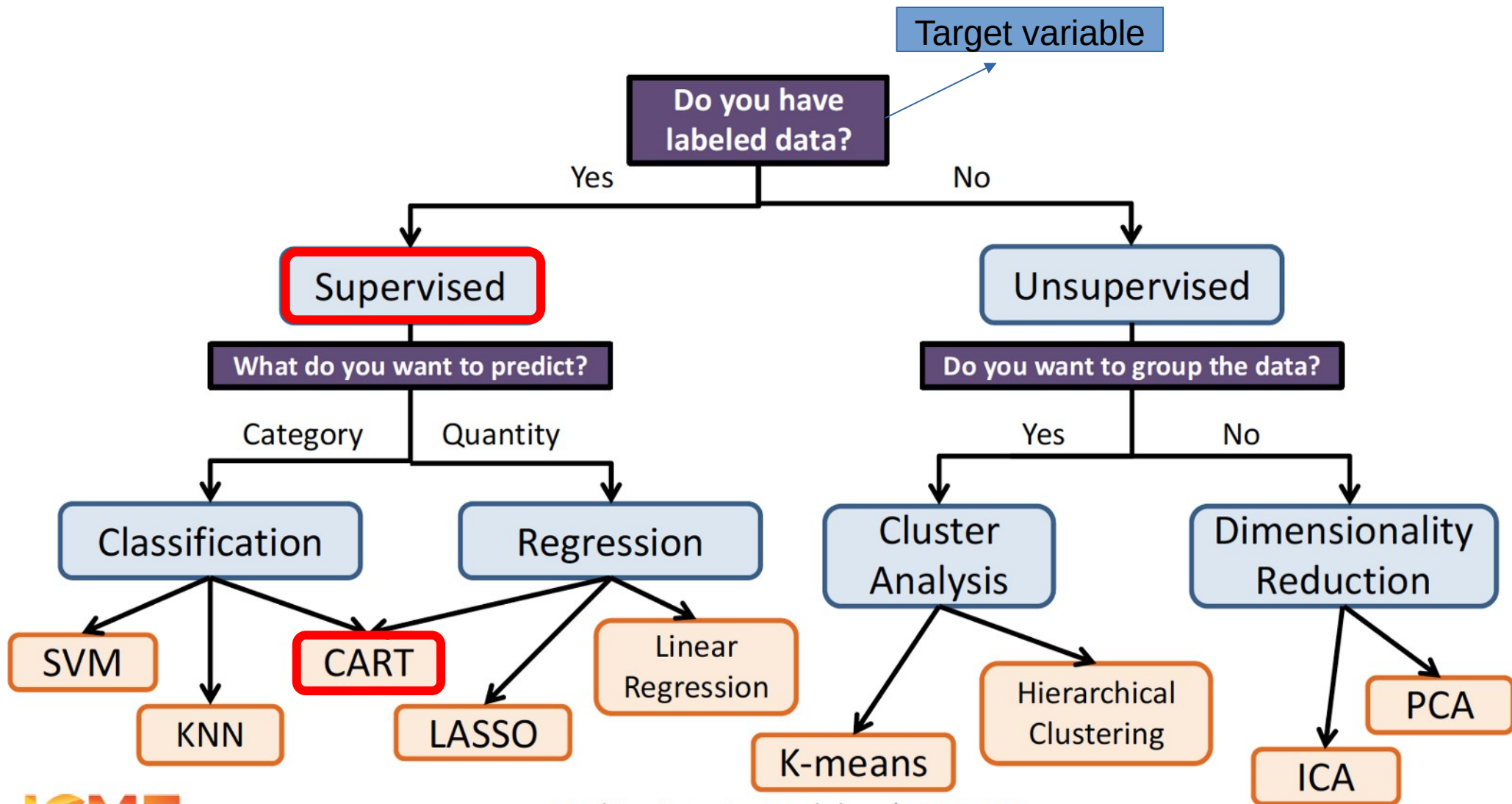


Types of Machine Learning



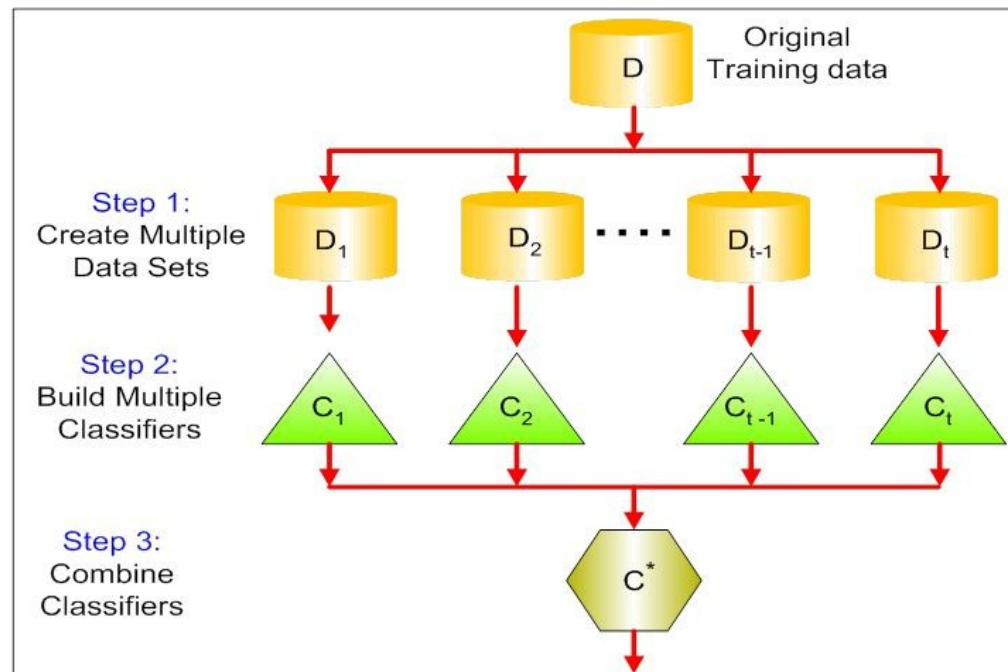
NOTA: Las líneas de código de R en esta presentación se muestran sobre un fondo gris.

Nov	2	Presentación, introducción y perspectiva histórica
	4	Paradigmas, problemas canonicos y data challenges
	9	Reglas de asociación
	11	Practica: Reglas de asociación
	16	Evaluación, sobreajuste y crossvalidacion
	18	Practica: Crossvalidacion
	23	Árboles de clasificacion y decision
	25	Practica: Árboles de clasificación
		T01. Datos discretos
	30	Técnicas de vecinos cercano (k-NN)
Dic	2	Práctica: Vecinos cercanos
	9	Comparación de Técnicas de Clasificación.
14	16	Reducción de dimensión no lineal
		T02. Clasificación
	17	Árboles de clasificación y regresion (CART)
	20	Práctica: Árboles de clasificación y regresion (CART)
	21	Practica: El paquete CARET
		T03. Prediccion
Ene	11	Ensembles: Bagging and Boosting
	13	Random Forests y Gradient boosting
	14	Técnicas de agrupamiento
	20	Técnicas de agrupamiento
	24	Predicción Condicionada
	26	Sesión de refuerzo/repaso.
Ene	27	Examen

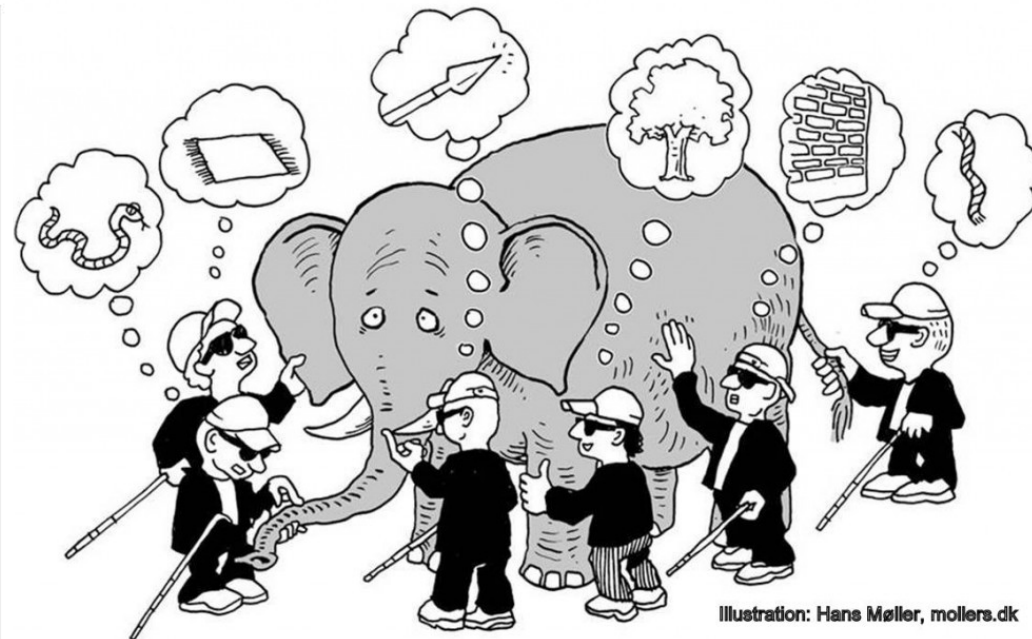


Ensemble learning

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

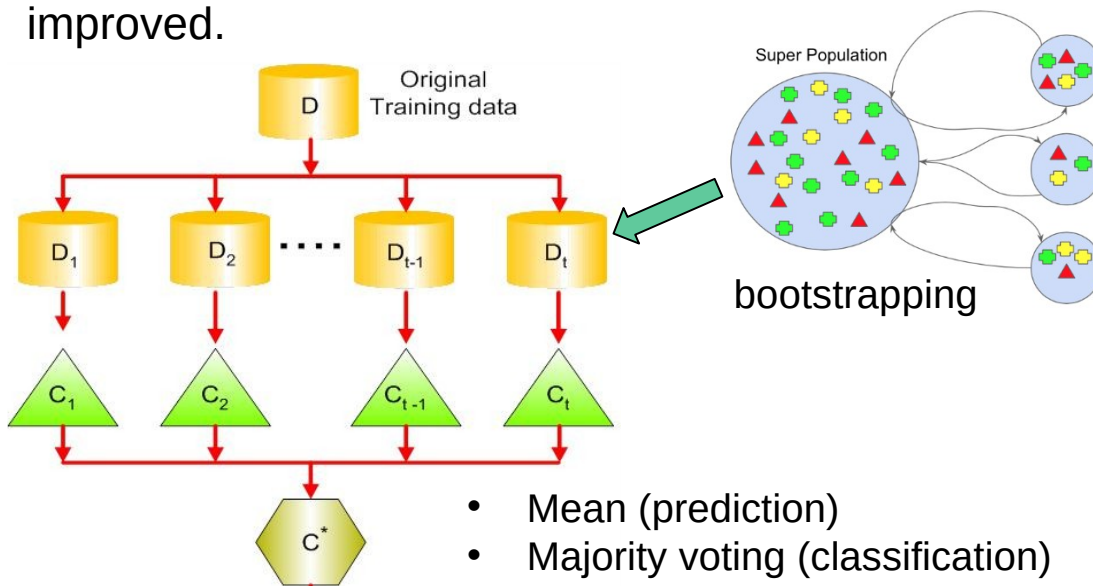
They work fine on both classification and regression problems

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.



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

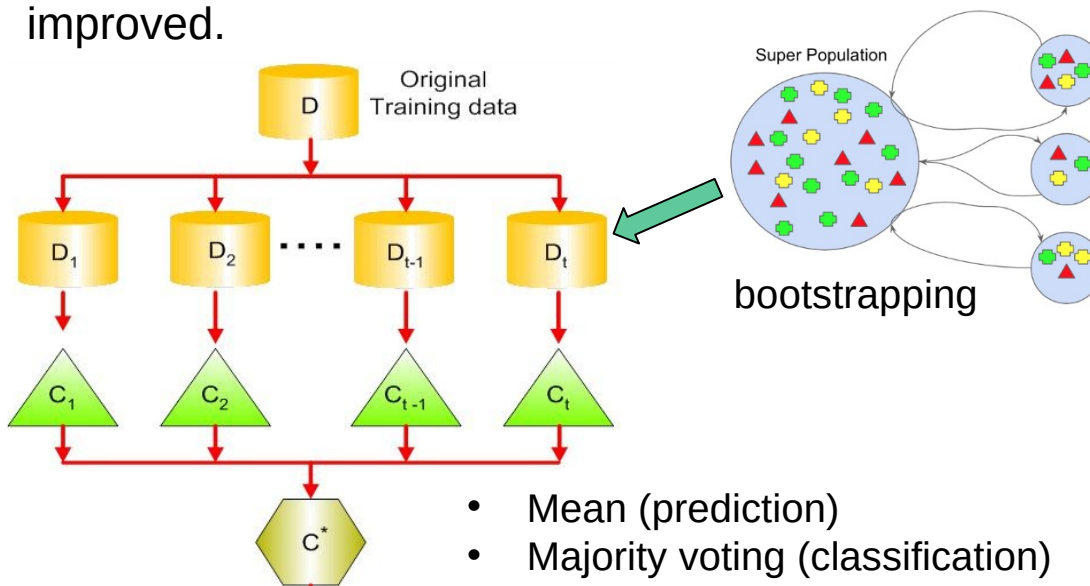
They work fine on both classification and regression problems

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 degree of freedom models
e.g. fully developed 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

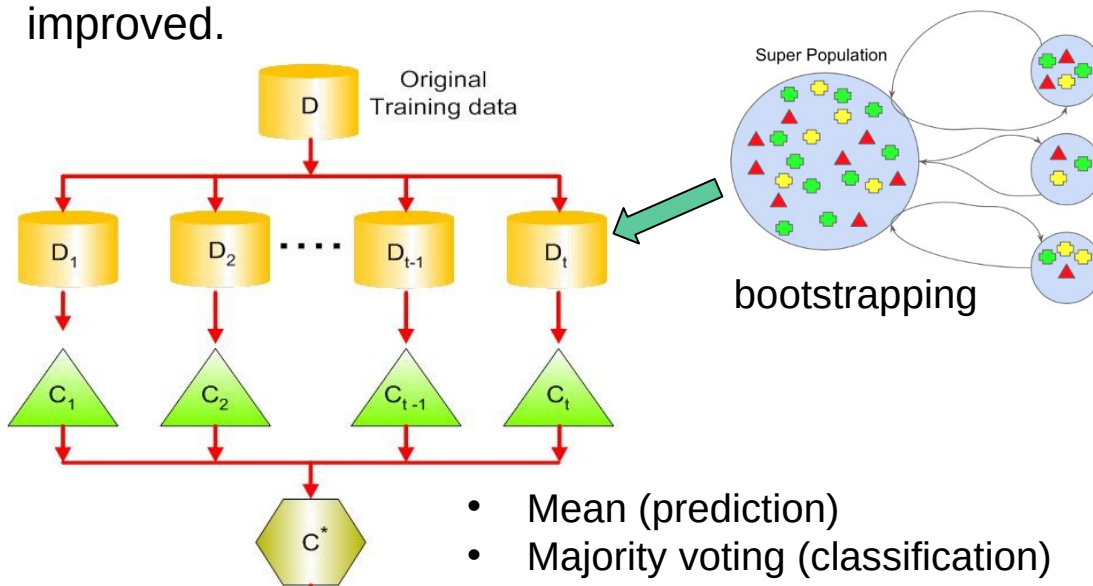
They work fine on both classification and regression problems

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.



Weak learners

High bias and low variance

Low degree of 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

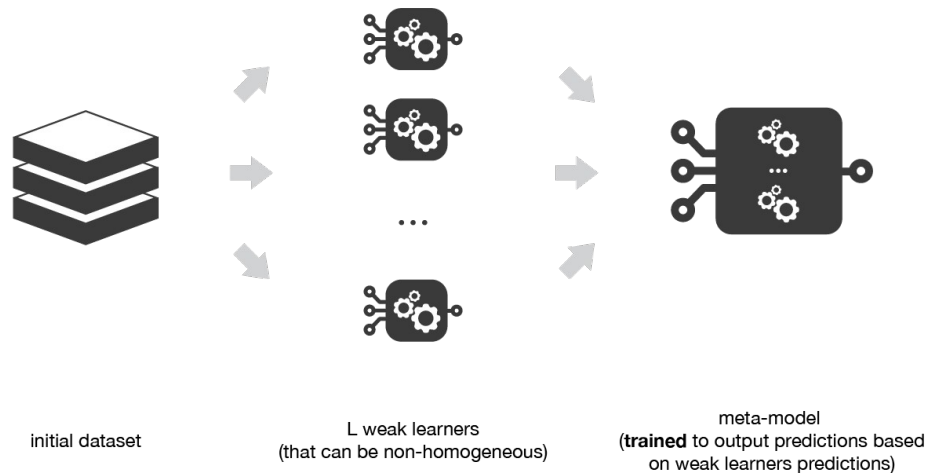
They work fine on both classification and regression problems

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->

<https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a20>

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

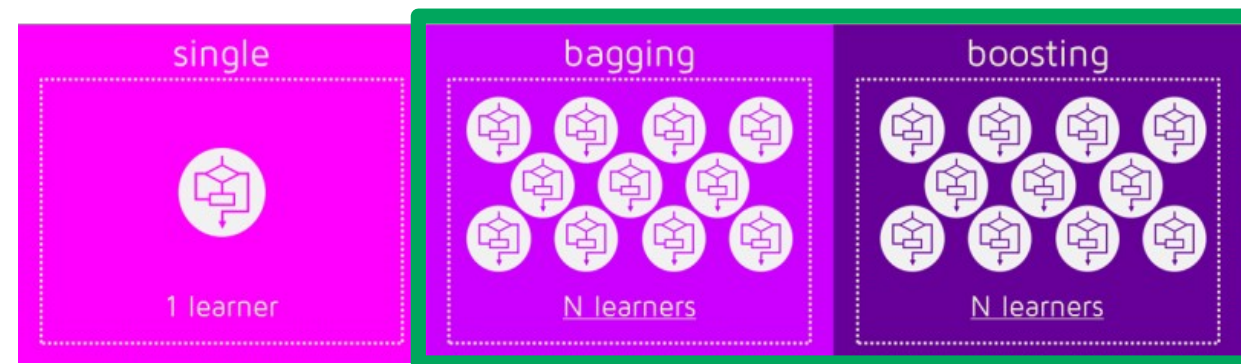
Poor prediction accuracy (compared with other approaches)

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.

Homogenous Weak Learners

Bagging and Boosting



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.

Homogenous Weak Learners

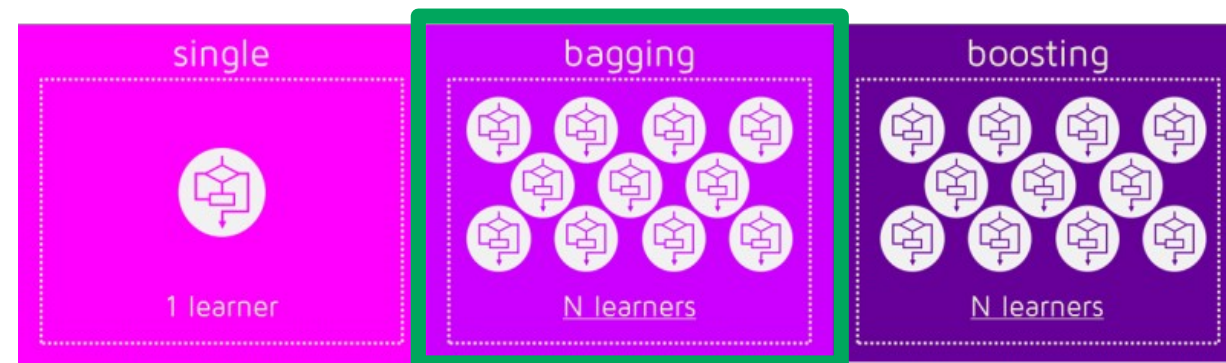
Low bias and high variance



High degree of freedom models
e.g. fully developed trees



Random Forests



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.

Homogenous Weak Learners

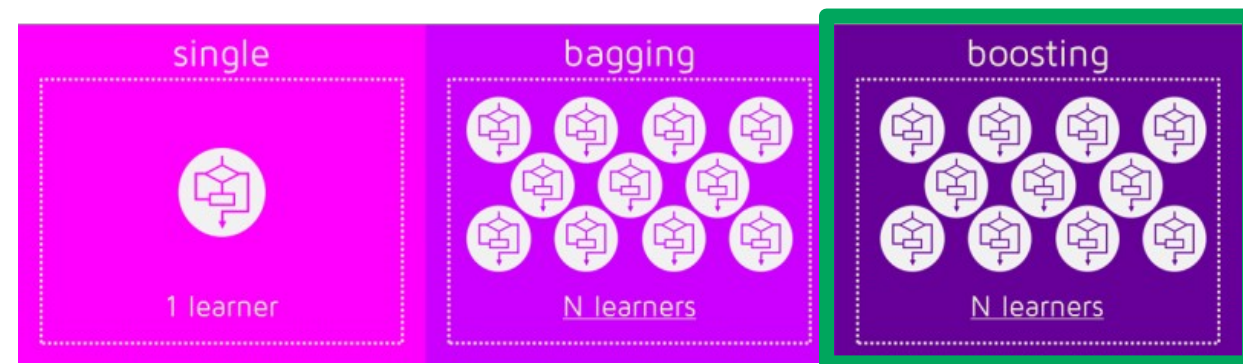
High bias and low variance



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



AdaBoost: Adaptive Boosting



Ensemble: Boosting Methods

Adaptative Boosting (AdaBoost)



train a weak model
and aggregate it to
the ensemble model



update the weights of
observations misclassified by
the current ensemble model

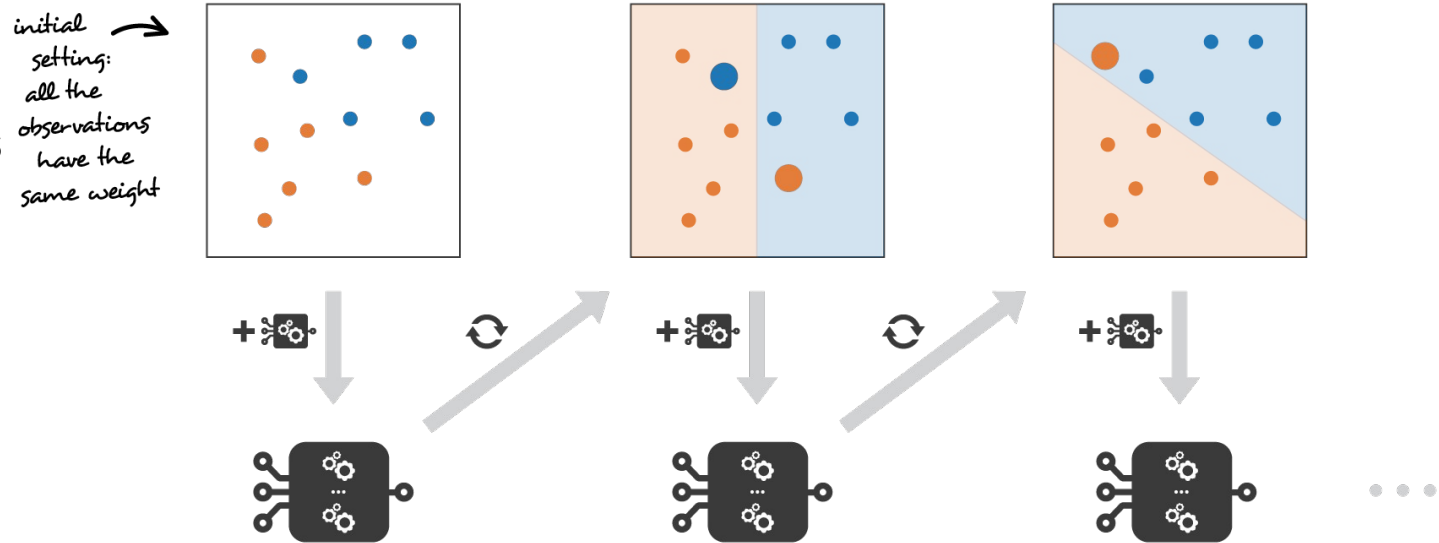


current ensemble model
predicts "orange" class



current ensemble model
predicts "blue" class

*initial
setting:
all the
observations
have the
same weight*



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

Ensemble: Boosting Methods

Adaptative Boosting (AdaBoost)



train a weak model
and aggregate it to
the ensemble model



update the weights of
observations misclassified by
the current ensemble model

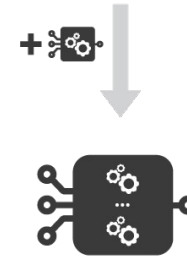
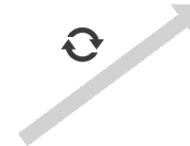
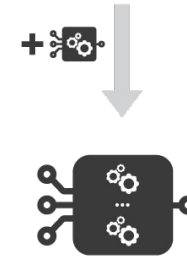
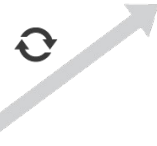
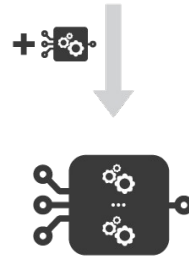
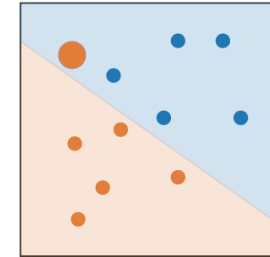
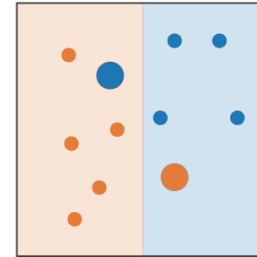
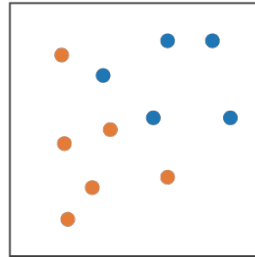


current ensemble model
predicts "orange" class



current ensemble model
predicts "blue" class

initial
setting:
all the
observations
have the
same weight



...

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

- Fit the weak model considering the observations weights.
- Evaluate the weak learner to obtain its coefficient.
- Update the strong learner adding the weak learner.
- Update the observations weights

Ensemble: Boosting Methods

Adaptative Boosting (AdaBoost)



train a weak model
and aggregate it to
the ensemble model



update the weights of
observations misclassified by
the current ensemble model

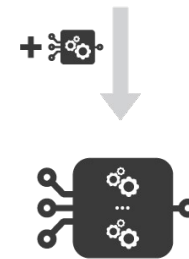
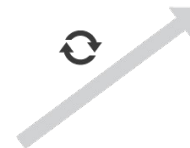
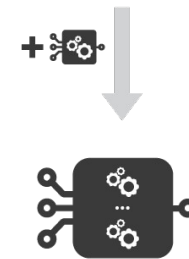
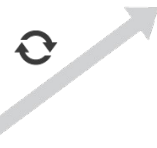
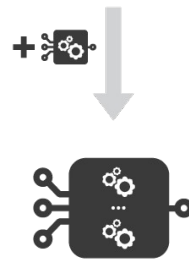
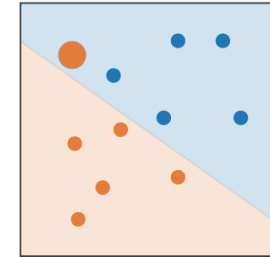
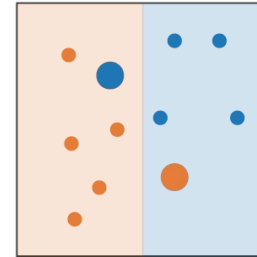
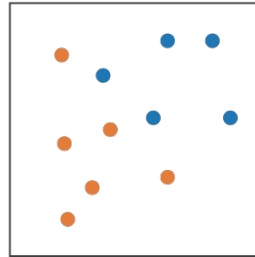


current ensemble model
predicts "orange" class



current ensemble model
predicts "blue" class

initial
setting:
all the
observations
have the
same weight



...

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

Repeat (1:L):

- Fit the weak model considering the observations weights.
- Evaluate the weak learner to obtain its coefficient.
- Update the strong learner adding the weak learner.
- Update the observations 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).

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

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

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.

Homogenous Weak Learners

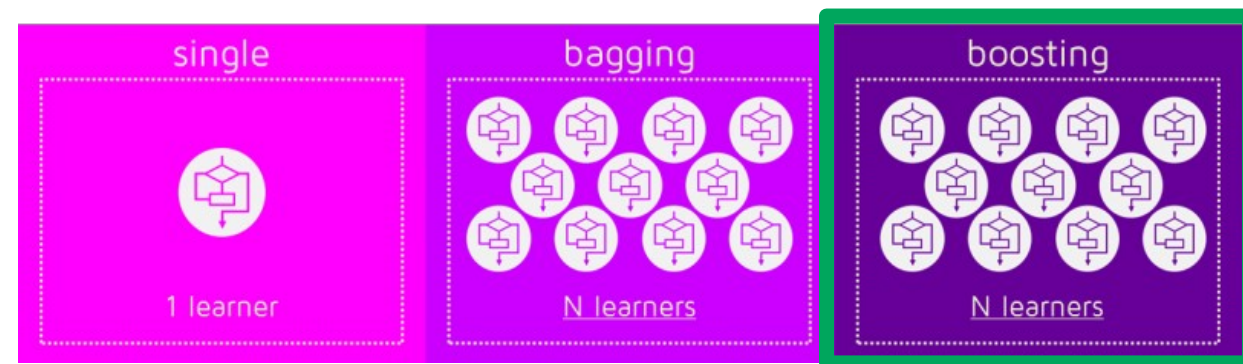
High bias and low variance



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








Gradient Descent Boosting



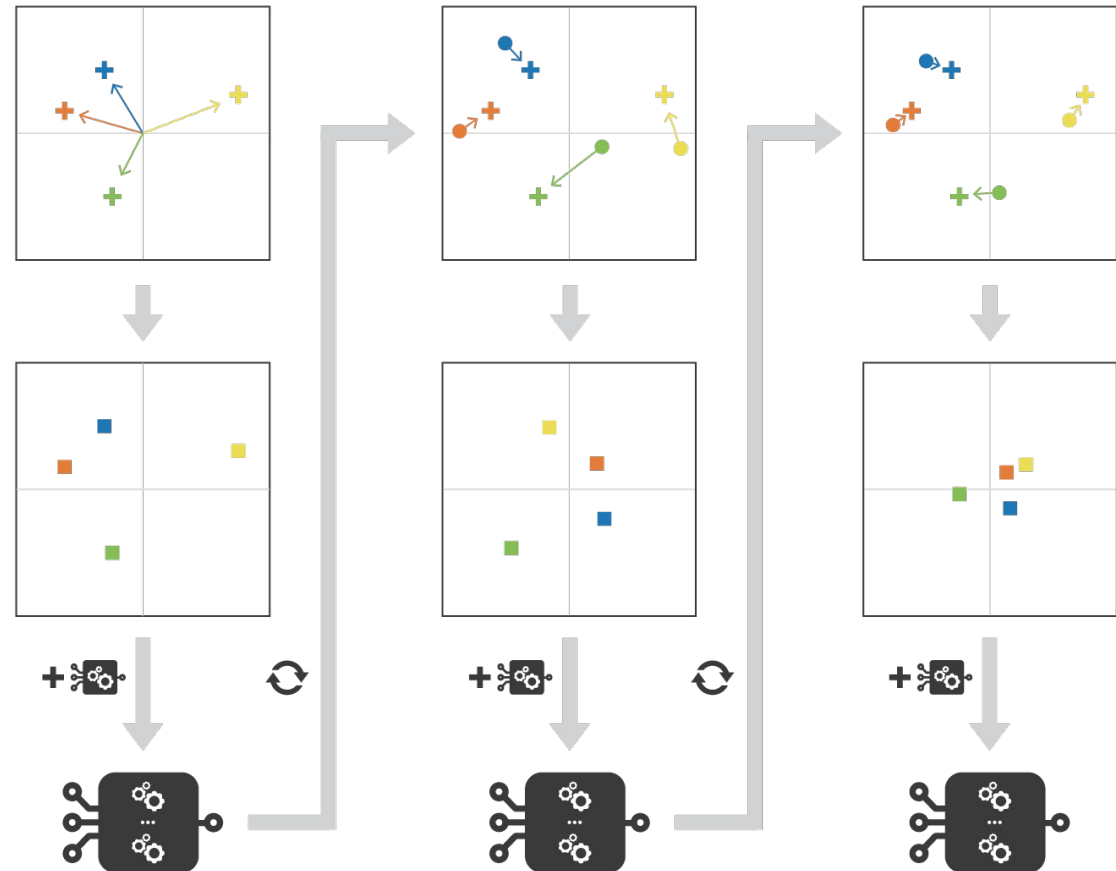
Ensemble: Boosting Methods

Gradient Boosting

-  train a weak model and aggregate it to the ensemble model
-  update the pseudo-residuals considering predictions of the current ensemble model
-  dataset values
-  predictions of the current ensemble model
-  pseudo-residuals (targets of the weak learner)

Gradient boosting casts the problem into a gradient descent one: at each iteration we fit a weak learner to the opposite of the gradient of the current fitting error with respect to the current ensemble model.

pseudo-residuals (\rightarrow) are the targets (\blacksquare) of the weak learner



...

$$s_L(.) = \sum_{l=1}^L c_l \times w_l(.)$$

where c_l 's are coefficients and w_l 's are weak learners

Ensemble: Boosting Methods

Gradient Boosting



train a weak model
and aggregate it to
the ensemble model



update the pseudo-residuals
considering predictions of
the current ensemble model

+ dataset values

● predictions of the current ensemble model

■ pseudo-residuals (targets of the weak learner)

Gradient boosting casts the problem into a gradient descent one: at each iteration we fit a weak learner to the opposite of the gradient of the current fitting error with respect to the current ensemble model.

pseudo-residuals (●) are the targets (■) of the weak learner

$$s_l(.) = s_{l-1}(.) - c_l \times \nabla_{s_{l-1}} E(s_{l-1})(.)$$

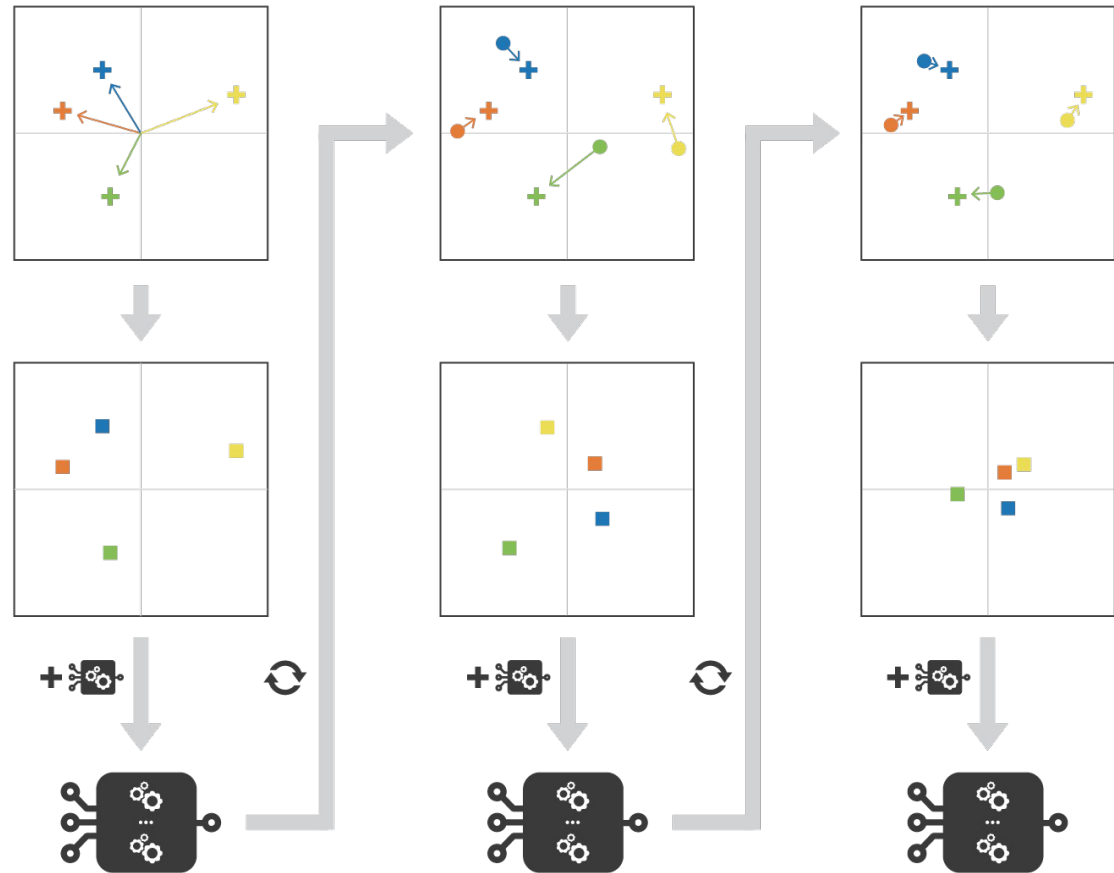
Pseudo-residuals:

$$-\nabla_{s_{l-1}} E(s_{l-1})(.)$$

Step size: how much we update the ensemble model in the direction of the new weak learner

$$s_L(.) = \sum_{l=1}^L c_l \times w_l(.)$$

where c_l 's are coefficients and w_l 's are weak learners



Ensemble: Boosting Methods

Gradient Boosting



train a weak model
and aggregate it to
the ensemble model



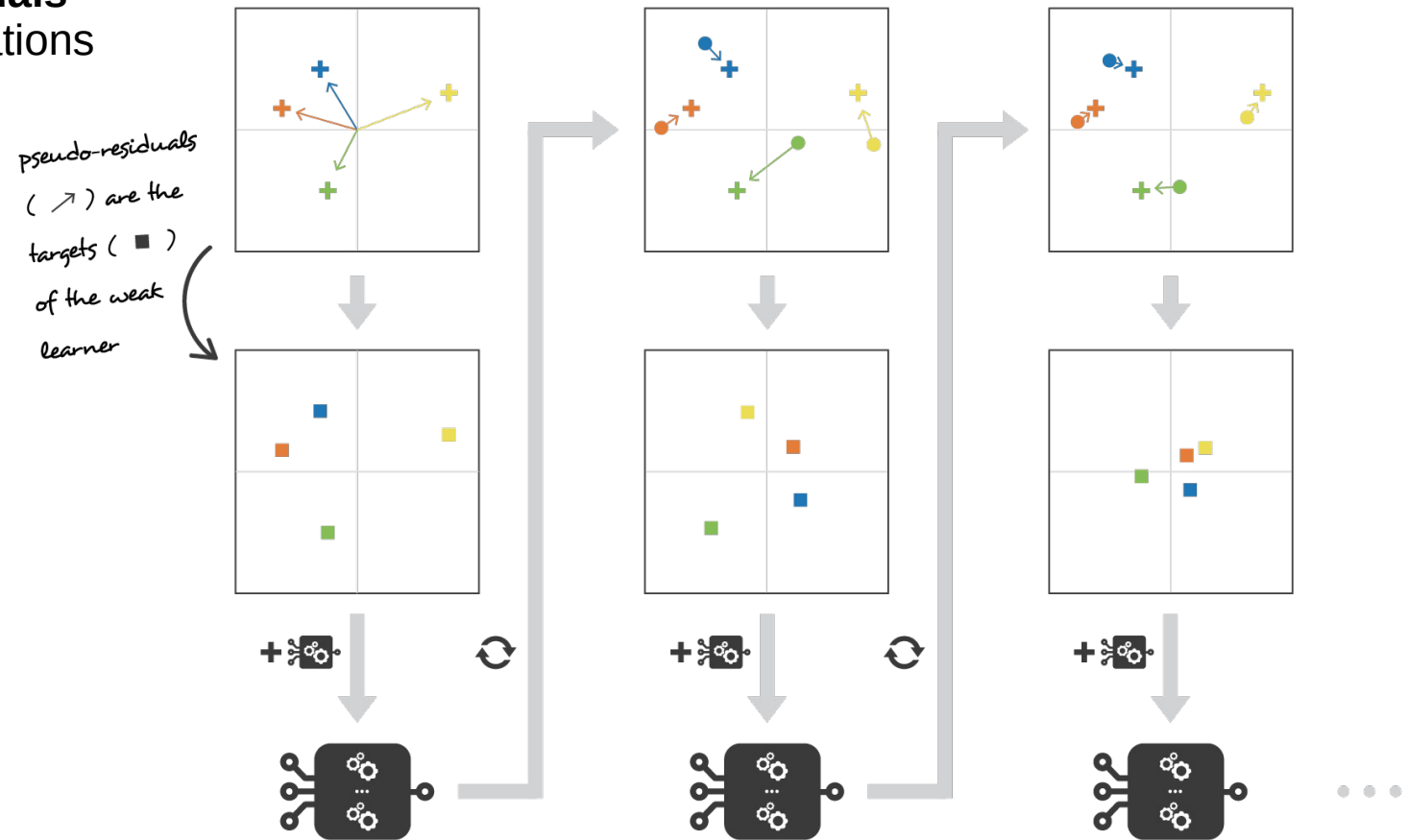
update the pseudo-residuals
considering predictions of
the current ensemble model

+ dataset values

● predictions of the current ensemble model

■ pseudo-residuals (targets of the weak learner)

Step 1: The **pseudo-residuals**
are set equal to the observations



$$s_L(\cdot) = \sum_{l=1}^L c_l \times w_l(\cdot)$$

where c_l 's are coefficients and w_l 's are weak learners

Ensemble: Boosting Methods

Gradient Boosting



train a weak model
and aggregate it to
the ensemble model



update the pseudo-residuals
considering predictions of
the current ensemble model

+ dataset values

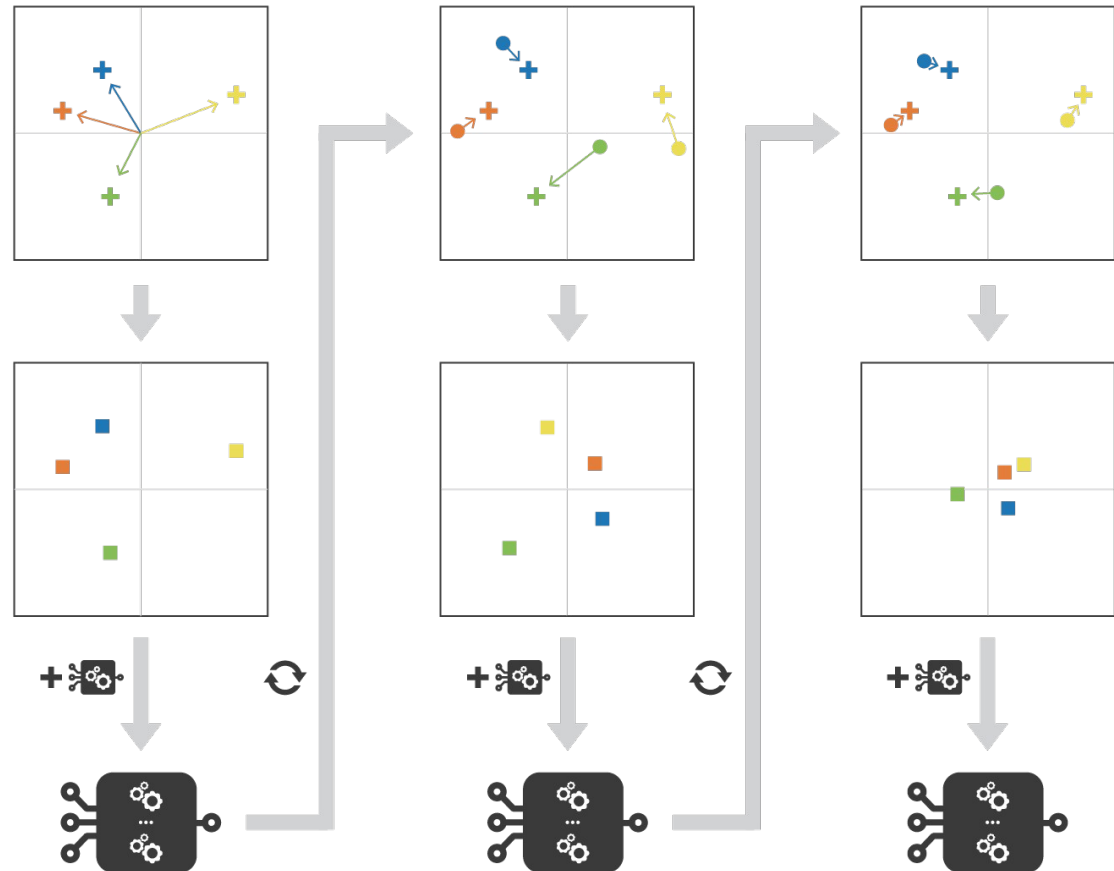
● predictions of the current ensemble model

■ pseudo-residuals (targets of the weak learner)

Step 1: The **pseudo-residuals** are set equal to the observations

- Fit the weak learner to pseudo-residuals.
- Compute the optimal step size of the weak learner.
- Update the strong learner adding the weak learner.
- Update the pseudo-residuals

*pseudo-residuals
(→) are the
targets (■)
of the weak
learner*



$$s_L(\cdot) = \sum_{l=1}^L c_l \times w_l(\cdot)$$

where c_l 's are coefficients and w_l 's are weak learners

Ensemble: Boosting Methods

Gradient Boosting



train a weak model
and aggregate it to
the ensemble model



update the pseudo-residuals
considering predictions of
the current ensemble model

+ dataset values

● predictions of the current ensemble model

■ pseudo-residuals (targets of the weak learner)

Step 1: The pseudo-residuals are set equal to the observations

Repeat (1:L):

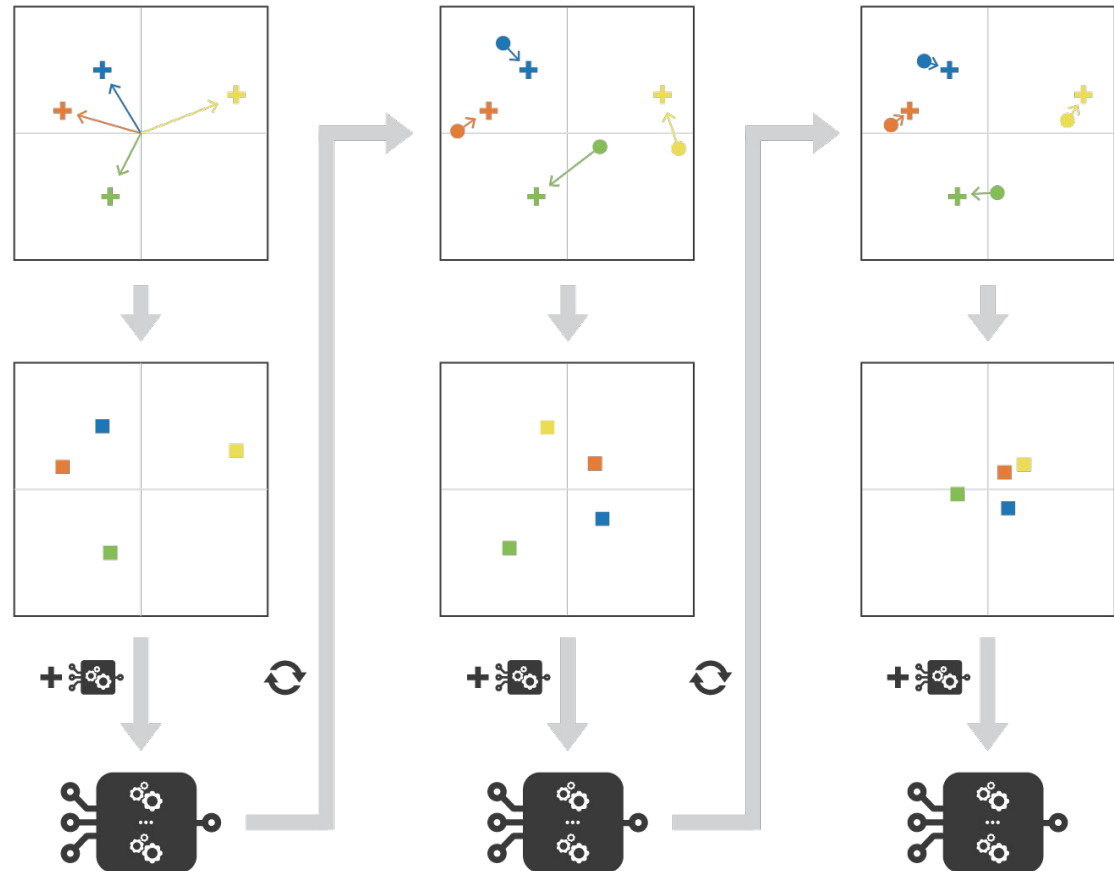
a) Fit the weak learner to pseudo-residuals.

b) Compute the optimal step size of the weak learner.

c) Update the strong learner adding the weak learner.

d) Update the pseudo-residuals

*pseudo-residuals
(→) are the
targets (■)
of the weak
learner*



<https://www.kdnuggets.com/2018/07/intuitive-ensemble-learning-guide-gradient-boosting.html>

$$s_L(\cdot) = \sum_{l=1}^L c_l \times w_l(\cdot)$$

where c_l 's are coefficients and w_l 's are weak learners

Ensemble: Boosting Methods

Adaptative vs Gradient Descent Boosting Approaches

Adaptative Boosting (AdaBoost)

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

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 observations weights

$$(c_l, w_l(.)) = \arg \min_{c, w(.)} E(s_{l-1}(.) + c \times w(.))$$

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

$$s_L(.) = \sum_{l=1}^L c_l \times w_l(.)$$

where c_l 's are coefficients and w_l 's are weak learners

Gradient Boosting

Gradient boosting casts the problem into a gradient descent one: at each iteration we fit a weak learner to the opposite of the gradient of the current fitting error with respect to the current ensemble model.

$$s_l(.) = s_{l-1}(.) - c_l \times \nabla_{s_{l-1}} E(s_{l-1})(.)$$

Pseudo-residuals: $-\nabla_{s_{l-1}} E(s_{l-1})(.)$

Step 1: The **pseudo-residuals** are set equal to the observations

Repeat (1:L):

- a) Fit the weak learner to pseudo-residuals.
- b) Compute the optimal step size of the weak learner.
- c) Update the strong learner adding the weak learner.
- d) Update the pseudo-residuals

Ensemble: Boosting Methods

Adaptative vs Gradient Descent Boosting Approaches

Adaptative Boosting (AdaBoost)

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

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 observations weights

$$(c_l, w_l(.)) = \arg \min E(s_{l-1}(.) + c \times w(.))$$

Notice that, while adaptative boosting tries to solve at each iteration exactly the “local” optimisation problem (find the best weak learner and its coefficient to add to the strong model), gradient boosting uses instead a gradient descent approach and can more easily be adapted to large number of loss functions. Thus, gradient boosting can be considered as a generalization of adaboost to arbitrary differentiable loss functions.

Gradient Boosting

Gradient boosting casts the problem into a gradient descent one: at each iteration we fit a weak learner to the opposite of the gradient of the current fitting error with respect to the current ensemble model.

$$s_l(.) = s_{l-1}(.) - c_l \times \nabla_{s_{l-1}} E(s_{l-1})(.)$$

Pseudo-residuals: $-\nabla_{s_{l-1}} E(s_{l-1})(.)$

Step 1: The **pseudo-residuals** are set equal to the observations

Repeat (1:L):

- a) Fit the weak learner to pseudo-residuals.
- b) Compute the optimal step size of the weak learner.

Ensemble: Boosting Methods

Adaptative vs Gradient Descent Boosting Approaches

Adaptative Boosting

boosting {adabag}

R Documentation

Applies the AdaBoost.M1 and SAMME algorithms to a data set

Description

Fits the AdaBoost.M1 (Freund and Schapire, 1996) and SAMME (Zhu et al., 2009) algorithms using classification trees as single classifiers.

Usage

```
boosting(formula, data, boos = TRUE, mfinal = 100, coeflearn = 'Breiman',
         control,...)
```

Arguments

formula	a formula, as in the <code>lm</code> function.
data	a data frame in which to interpret the variables named in formula.
boos	if TRUE (by default), a bootstrap sample of the training set is drawn using the weights for each observation on that iteration. If FALSE, every observation is used with its weights.
mfinal	an integer, the number of iterations for which boosting is run or the number of trees to use. Defaults to <code>mfinal=100</code> iterations.
coeflearn	if 'Breiman'(by default), $\alpha=1/2\ln((1-\text{err})/\text{err})$ is used. If 'Freund' $\alpha=\ln((1-\text{err})/\text{err})$ is used. In both cases the AdaBoost.M1 algorithm is used and α is the weight updating coefficient. On the other hand, if <code>coeflearn</code> is 'Zhu' the SAMME algorithm is implemented with $\alpha=\ln((1-\text{err})/\text{err})+\ln(\text{nclases}-1)$.
control	options that control details of the <code>rpart</code> algorithm. See <code>rpart.control</code> for more details.
...	further arguments passed to or from other methods.

Details

Gradient Boosting

R: Generalized Boosted Regression Modeling (GBM) ▾

Find in Topic

gbm {gbm}

R Documentation

Generalized Boosted Regression Modeling (GBM)

Description

Fits generalized boosted regression models. For technical details, see the vignette: `utils::browseVignettes("gbm")`.

Usage

```
gbm(formula = formula(data), distribution = "bernoulli",
    data = list(), weights, var.monotone = NULL, n.trees = 100,
    interaction.depth = 1, n.minobsinnode = 10, shrinkage = 0.1,
    bag.fraction = 0.5, train.fraction = 1, cv.folds = 0,
    keep.data = TRUE, verbose = FALSE, class.stratify.cv = NULL,
    n.cores = NULL)
```

Arguments

formula	A symbolic description of the model to be fit. The formula may include an offset term (e.g. $y \sim \text{offset}(n) + x$). If <code>keep.data = FALSE</code> in the initial call to <code>gbm</code> then it is the user's responsibility to resupply the offset to gbm.more .
distribution	Either a character string specifying the name of the distribution to use or a list with a component name specifying the distribution and any additional parameters needed. If not specified, <code>gbm</code> will try to guess: if the response has only 2 unique values, <code>bernoulli</code> is assumed; otherwise, if the response is a factor, <code>multinomial</code> is assumed; otherwise, if the response has class "Surv", <code>coxph</code> is assumed; otherwise, <code>gaussian</code> is assumed. Currently available options are "gaussian" (squared error), "laplace" (absolute loss), "tdist" (t-distribution loss), "bernoulli" (logistic regression for 0-1 outcomes), "huberized" (huberized hinge loss for 0-1 outcomes), "classes", "adaboost" (the AdaBoost exponential loss for 0-1 outcomes), "poisson" (count outcomes), "coxph" (right censored observations), "quantile", or "pairwise" (ranking measure using the LambdaMart algorithm). If quantile regression is specified, distribution must be a list of the form <code>list(name = "quantile", alpha = 0.25)</code> where <code>alpha</code> is the quantile to estimate. The current version's quantile regression method does not handle non-constant weights and will stop.

Ensemble: Boosting Methods

Extensions of Gradient Descent Boosting Approach

dmlc
XGBoost

Extreme Gradient Boosting

XGBoost: A Scalable Tree Boosting System

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

Carlos Guestrin
University of Washington
guestrin@cs.washington.edu

<https://arxiv.org/pdf/1603.02754.pdf>

<https://github.com/dmlc/xgboost>

<https://xgboost.readthedocs.io/en/latest/parameter.html>

Gradient Boosting

Gradient boosting casts the problem into a gradient descent one: at each iteration we fit a weak learner to the opposite of the gradient of the current fitting error with respect to the current ensemble model.

$$s_l(.) = s_{l-1}(.) - c_l \times \nabla_{s_{l-1}} E(s_{l-1})(.)$$

Pseudo-residuals: $-\nabla_{s_{l-1}} E(s_{l-1})(.)$

Step 1: The **pseudo-residuals** are set equal to the observations

Repeat (1:L):

- Fit the weak learner to pseudo-residuals.
- Compute the optimal step size of the weak learner.
- Update the strong learner adding the weak learner.
- Update the pseudo-residuals

$$s_L(.) = \sum_{l=1}^L c_l \times w_l(.)$$

where c_l 's are coefficients and w_l 's are weak learners

Ensemble: Boosting Methods

Extensions of Gradient Descent Boosting Approach

TensorFlow > Learn > TensorFlow Core > Tutorials

Boosted trees using Estimators

Contenido

Load the titanic dataset

Explore the data

Create feature columns and input functions

Train and evaluate the model

https://www.tensorflow.org/tutorials/estimator/boosted_trees

<https://arxiv.org/pdf/1710.11555.pdf>

TF Boosted Trees: A scalable TensorFlow based framework for gradient boosting

Natalia Ponomareva, Soroush Radpour, Gilbert Hendry, Salem Haykal,
Thomas Colthurst, Petr Mitrichev, Alexander Grushetsky

Google, Inc.
tfbt-public@google.com

Abstract. TF Boosted Trees (TFBT) is a new open-sourced framework for the distributed training of gradient boosted trees. It is based on TensorFlow, and its distinguishing features include a novel architecture, automatic loss differentiation, layer-by-layer boosting that results in smaller ensembles and faster prediction, principled multi-class handling, and a number of regularization techniques to prevent overfitting.

Gradient Boosting

Gradient boosting casts the problem into a gradient descent one: at each iteration we fit a weak learner to the opposite of the gradient of the current fitting error with respect to the current ensemble model.

$$s_l(.) = s_{l-1}(.) - c_l \times \nabla_{s_{l-1}} E(s_{l-1})(.)$$

Pseudo-residuals: $-\nabla_{s_{l-1}} E(s_{l-1})(.)$

Step 1: The **pseudo-residuals** are set equal to the observations

Repeat (1:L):

- Fit the weak learner to pseudo-residuals.
- Compute the optimal step size of the weak learner.
- Update the strong learner adding the weak learner.
- Update the pseudo-residuals

Ensemble Methods in R

Classification problem (iris)

R-Packages:

```
rm(list = ls())
install.packages("tree")
install.packages("randomForest")
install.packages("adabag")
install.packages("gbm")
```

Loading R-Packages:

```
library(tree)
library(randomForest)
library(adabag)
library(gbm)
library(caret)
library(MASS)
```

Accuracy (Trees):

```
print(sum(diag(table(pred.t.test, iris$Species[indtest])) / length(indtest))
print(sum(diag(table(pred.t.train, iris$Species[indtrain])) / length(indtrain))
```

Train/Test partition:

```
set.seed(23)
n <- nrow(iris)
indtrain <- sample(1:n, round(0.75*n)) # indices for train
indtest <- setdiff(1:n, indtrain) # indices for test
```

Single Tree:

```
t <- tree(Species ~., iris, subset = indtrain, control = tree.control(length(indtrain),
mincut = 1, minsize = 2, mindev = 0))
```

Prediction for test

```
pred.t.test <- predict(t, iris[indtest, ], type = "class")
```

Prediction for train

```
pred.t.train <- predict(t, iris[indtrain, ], type = "class")
```

Ensemble Methods in R

Classification problem (iris)

R-Packages:

```
rm(list = ls())  
install.packages("tree")  
install.packages("randomForest")  
install.packages("adabag")  
install.packages("gbm")
```

Loading R-Packages:

```
library(tree)  
library(randomForest)  
library(adabag)  
library(gbm)  
library(caret)  
library(MASS)
```

Train/Test partition:

```
set.seed(23)  
n <- nrow(iris)  
indtrain <- sample(1:n, round(0.75*n)) # indices for train  
indtest <- setdiff(1:n, indtrain) # indices for test
```

Bagging: Random Forests

```
rf <- randomForest(Species ~., iris, subset = indtrain, ntree = 500)
```

Prediction for test

```
pred.rf.test <- predict(rf, iris[indtest, ])
```

Prediction for train

```
pred.rf.train <- predict(rf, iris[indtrain, ])
```

Accuracy (Trees):

```
print(sum(diag(table(pred.t.test, iris$Species[indtest]))) / length(indtest))  
print(sum(diag(table(pred.t.train, iris$Species[indtrain]))) / length(indtrain))
```

Accuracy (Random Forests):

```
print(sum(diag(table(pred.rf.test, iris$Species[indtest]))) / length(indtest))  
print(sum(diag(table(pred.rf.train, iris$Species[indtrain]))) / length(indtrain))
```


Ensemble Methods in R

Classification problem (iris)

R-Packages:

```
rm(list = ls())
install.packages("tree")
install.packages("randomForest")
install.packages("adabag")
install.packages("gbm")
```

Loading R-Packages:

```
library(tree)
library(randomForest)
library(adabag)
library(gbm)
library(caret)
library(MASS)
```

Train/Test partition:

```
set.seed(23)
n <- nrow(iris)
indtrain <- sample(1:n, round(0.75*n)) # indices for train
indtest <- setdiff(1:n, indtrain) # indices for test
```

Boosting: Adaptive Boosting (AdaBoost)

```
ab <- boosting(Species ~., iris[indtrain, ], mfinal = 20,
               control=rpart.control(minsplit = 2, minbucket = 1, cp = 0.01))
```

Prediction for test

```
pred.ab.test <- predict(ab, iris[indtest, ])
```

Prediction for train

```
pred.ab.train <- predict(ab, iris[indtrain, ])
```

Accuracy (Trees):

```
print(sum(diag(table(pred.t.test, iris$Species[indtest])))) / length(indtest)
print(sum(diag(table(pred.t.train, iris$Species[indtrain])))) / length(indtrain))
```

Accuracy (Random Forests):

```
print(sum(diag(table(pred.rf.test, iris$Species[indtest])))) / length(indtest)
print(sum(diag(table(pred.rf.train, iris$Species[indtrain])))) / length(indtrain))
```

Accuracy (AdaBoost):

```
print(sum(diag(table(pred.ab.test$class, iris$Species[indtest])))/length(indtest))
print(sum(diag(table(pred.ab.train$class, iris$Species[indtrain])))/length(indtrain))
```

Ensemble Methods in R

Classification problem (iris)

R-Packages:

```
rm(list = ls())
install.packages("tree")
install.packages("randomForest")
install.packages("adabag")
install.packages("gbm")
```

Loading R-Packages:

```
library(tree)
library(randomForest)
library(adabag)
library(gbm)
library(caret)
library(MASS)
```

Accuracy (Trees):

```
print(sum(diag(table(pred.t.test, iris$Species[indtest])))) / length(indtest))
print(sum(diag(table(pred.t.train, iris$Species[indtrain])))) / length(indtrain))
```

Accuracy (Random Forests):

```
print(sum(diag(table(pred.rf.test, iris$Species[indtest])))) / length(indtest))
print(sum(diag(table(pred.rf.train, iris$Species[indtrain])))) / length(indtrain))
```

Accuracy (AdaBoost):

```
print(sum(diag(table(pred.ab.test$class, iris$Species[indtest])))/length(indtest))
print(sum(diag(table(pred.ab.train$class, iris$Species[indtrain])))/length(indtrain))
```

Accuracy (Gradient Boosting):

```
print(sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.test,
FUN = which.max, MARGIN = 1)], iris$Species[indtest])))) / length(indtest))
print(sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.train,
FUN = which.max, MARGIN = 1)], iris$Species[indtrain])))) / length(indtrain))
```

Boosting: Gradient Boosting

```
gb <- gbm(Species~., data=iris[indtrain, ], n.trees=1000,
          interaction.depth=20, shrinkage = 0.01)
```

Prediction for test

```
pred.gb.test <- predict(object = gb, newdata = iris[indtest, ], n.trees = 1000, type = "response")
```

Prediction for train

```
pred.gb.train <- predict(object = gb, newdata = iris[indtrain, ], n.trees = 1000, type = "response")
```

Ensemble Methods in R

Classification problem (iris)

R-Packages:

```
rm(list = ls())
install.packages("tree")
install.packages("randomForest")
install.packages("adabag")
install.packages("gbm")
```

Loading R-Packages:

```
library(tree)
library(randomForest)
library(adabag)
library(gbm)
library(caret)
library(MASS)
```

Accuracy (Trees):

```
print(sum(diag(table(pred.t.test, iris$Species[indtest])))) / length(indtest))
print(sum(diag(table(pred.t.train, iris$Species[indtrain])))) / length(indtrain))
```

Accuracy (Random Forests):

```
print(sum(diag(table(pred.rf.test, iris$Species[indtest])))) / length(indtest))
print(sum(diag(table(pred.rf.train, iris$Species[indtrain])))) / length(indtrain))
```

Accuracy (AdaBoost):

```
print(sum(diag(table(pred.ab.test$class, iris$Species[indtest])))/length(indtest))
print(sum(diag(table(pred.ab.train$class, iris$Species[indtrain])))/length(indtrain))
```

Accuracy (Gradient Boosting):

```
print(sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.test,
FUN = which.max, MARGIN = 1)], iris$Species[indtest])))) / length(indtest))
print(sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.train,
FUN = which.max, MARGIN = 1)], iris$Species[indtrain])))) / length(indtrain))
```

Boosting: Gradient Boosting – Tuning the parameters

```
gb.cv <- gbm(Species~., data=iris[indtrain, ], n.trees=1000, interaction.depth=20,
             shrinkage = 0.01, cv.folds = 4)
ntree_opt_cv <- gbm.perf(gb.cv, method = "cv")
ntree_opt_oob <- gbm.perf(gb.cv, method = "OOB")
print(c(ntree_opt_cv, ntree_opt_oob))
```

Ensemble Methods in R

Classification problem (iris)

R-Packages:

```
rm(list = ls())
install.packages("tree")
install.packages("randomForest")
install.packages("adabag")
install.packages("gbm")
```

Loading R-Packages:

```
library(tree)
library(randomForest)
library(adabag)
library(gbm)
library(caret)
library(MASS)
```

Accuracy (Trees):

```
print(sum(diag(table(pred.t.test, iris$Species[indtest])))) / length(indtest))
print(sum(diag(table(pred.t.train, iris$Species[indtrain])))) / length(indtrain))
```

Accuracy (Random Forests):

```
print(sum(diag(table(pred.rf.test, iris$Species[indtest])))) / length(indtest))
print(sum(diag(table(pred.rf.train, iris$Species[indtrain])))) / length(indtrain))
```

Accuracy (AdaBoost):

```
print(sum(diag(table(pred.ab.test$class, iris$Species[indtest])))/length(indtest))
print(sum(diag(table(pred.ab.train$class, iris$Species[indtrain])))/length(indtrain))
```

Accuracy (Gradient Boosting):

```
print(sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.test,
FUN = which.max, MARGIN = 1)], iris$Species[indtest])))) / length(indtest))
print(sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.train,
FUN = which.max, MARGIN = 1)], iris$Species[indtrain])))) / length(indtrain))
```

Boosting: Gradient Boosting

```
gb <- gbm(Species~., data=iris[indtrain, ], n.trees=ntree_opt_cv,
          interaction.depth=20, shrinkage = 0.01)
```

```
print(gb)
summary(gb)
```

Prediction for test

```
pred.gb.test <- predict(object = gb, newdata = iris[indtest, ], n.trees = ntree_opt_cv, type = "response")
```

Prediction for train

```
pred.gb.train <- predict(object = gb, newdata = iris[indtrain, ], n.trees = ntree_opt_cv, type = "response")
```