# Bagging - Random Forest - Boosting

## Ensemble methods

Ricco RAKOTOMALALA

Université Lumière Lyon 2

# Principle of ensemble methods

Make cooperate several classifiers. Combine the outputs of the individual classifiers to produce the prediction.

(1) Build different types of models (e.g. tree, linear classifier, etc.) on the same learning sample.

(2) Build models of the same nature (same learning algorithm) on various versions of the learning set.

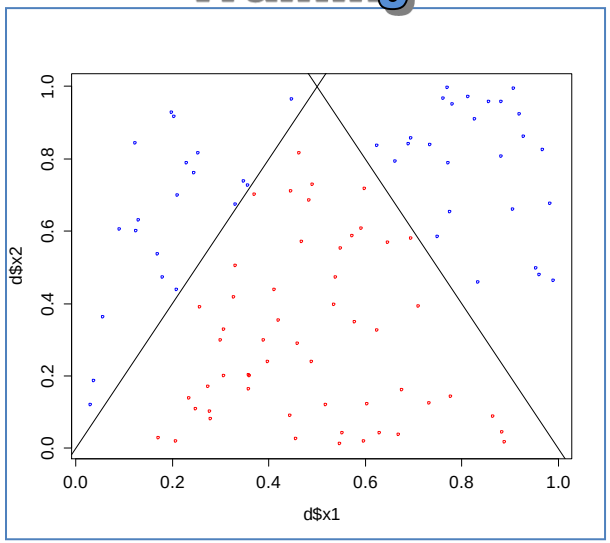
It is necessary that the models complete each other in order to benefit from combination. If they classify the instances exactly in the same way, the gain (compared to individual models) is zero!

We treat only the classification task in this course material

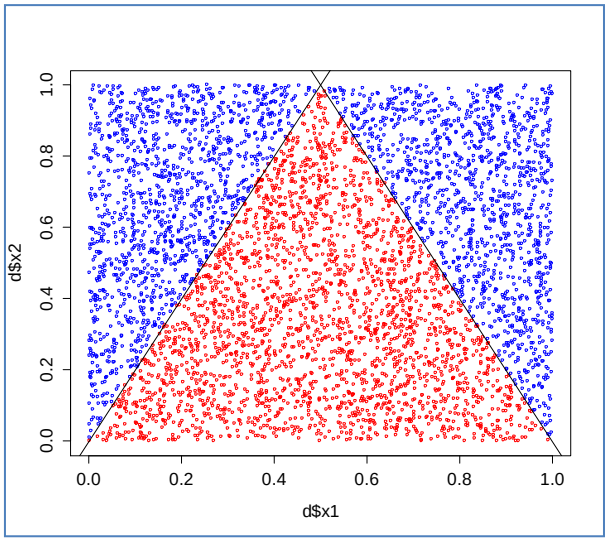
# Dataset

We use artificial dataset: $n_{app}$ = 100 training set, $n_{test}$ = 5000 test set; p = 10 predictive attributes, 2 only are relevant ($x_1$ and $x_2$); binary problem; no noise.

## Training




## Test



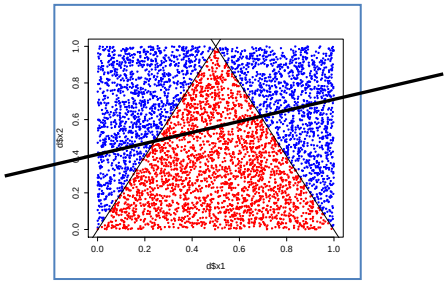

```
Y = « blue »
        if (1) x2 > 2 * x1 for x1 < 0.5
        or (2) x2 > (2 – 2 * x1) for x1 ≥ 0.5
    « red » otherwise
```

# Sources of error (1/2)

We create a model from a training set. We want it is efficient in the whole population. Two sources of error may disturb the quality of the prediction.

## Bias

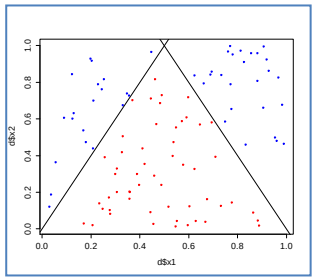
Corresponds to the inability to the representation system to highlight the relevant relation between the input attributes and the target attribute.




A linear classifier is unable to separate perfectly the classes for our problem.

## Variance

Sensitivity to small fluctuations in the training set.



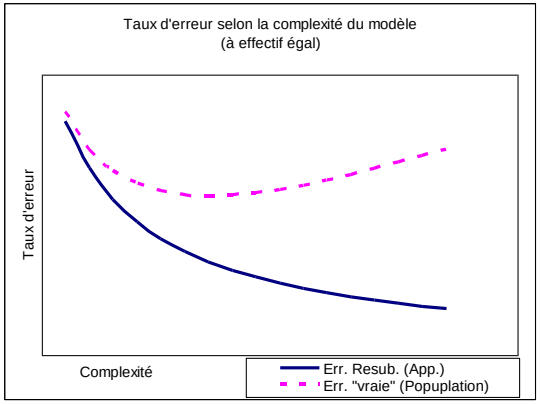

Because the small size of the training set, the learning algorithm cannot detect exactly the true frontiers between the classes.

# Sources of error (2/2)

Some important ideas about the behavior of the classifiers.

"Simple" models (e.g., linear classifier, few parameters to estimate) have a strong bias, but a low variance.

"Complex" models (e.g. many parameters to estimate) have a low bias, but a high variance




We do not forget the disturbing role that can have irrelevant variables, particularly when they are numerous (see curse of dimensionality)

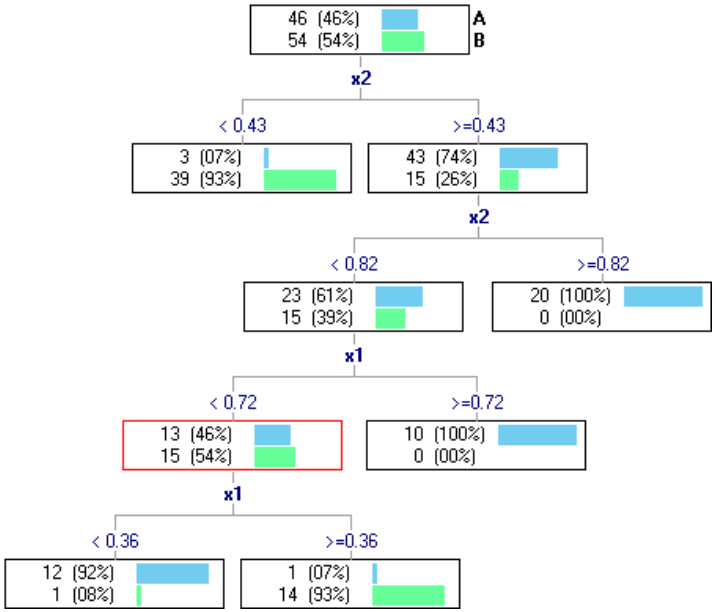
# Outline

1. Decision tree induction
2. Bagging
3. Random Forest
4. Boosting
5. Tools
6. Summary
7. References

# Decision Tree

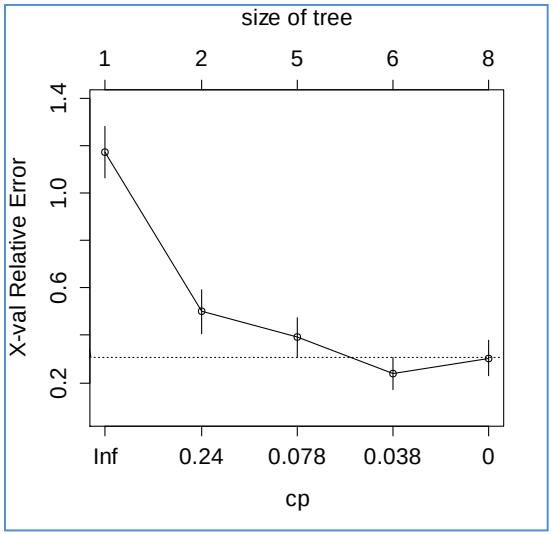
# Decision Tree Induction

Recursive algorithm for subdivision of representation space. The splits are necessarily parallel to axes. The model is piecewise linear, the whole model is non-linear.




Deep tree: low bias, high variance

Short tree: high bias, low variance

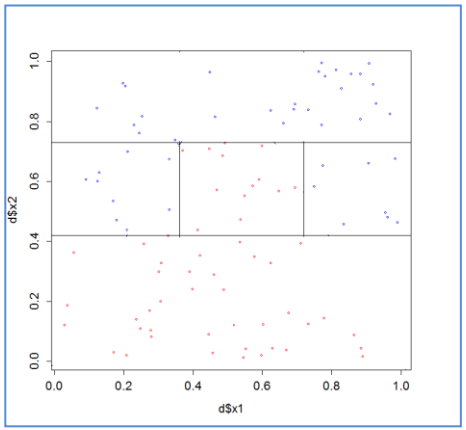



One of the main issues of the construction of the tree is to detect the "optimal" depth, corresponding to the best trade-off between bias and variance (e.g. detect the right value for the parameter **cp** in rpart [rpart package] for R)

# Frontiers defined by a decision tree

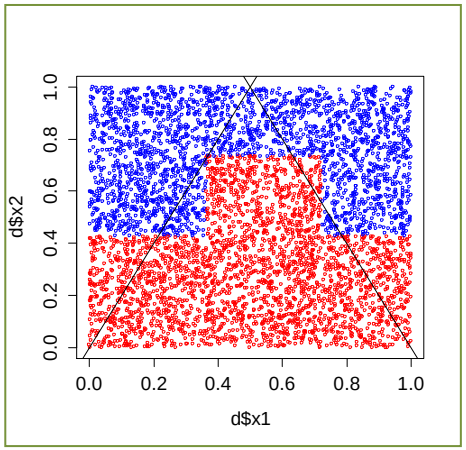
```
library(rpart)
arbre <- rpart(y ~ ., data = d.train)
print(arbre)
```

```
n= 100

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 100 46 2 (0.46000000 0.54000000)
   2) x2>=0.4271354 58 15 1 (0.74137931 0.25862069)
     4) x2>=0.733562 27  1 1 (0.96296296 0.03703704) *
     5) x2< 0.733562 31 14 1 (0.54838710 0.45161290)
      10) x1>=0.7221232 8  0 1 (1.00000000 0.00000000) *
      11) x1< 0.7221232 23  9 2 (0.39130435 0.60869565)
        22) x1< 0.3639227 10  1 1 (0.90000000 0.10000000) *
        23) x1>=0.3639227 13  0 2 (0.00000000 1.00000000) *
   3) x2< 0.4271354 42  3 2 (0.07142857 0.92857143) *
```




TRAIN

(Tree with 5 leaves = 5 areas are defined)




TEST

$\varepsilon = 0.1632$

- Modeling is constrained by the number of observations available in the training set
- If we try to force the splitting, some irrelevant variables can be inserted into the tree

```
arbre.p <- rpart(y ~ ., data = d.train,
control=list(cp=0,minsplit=2,minbucket=1))
print(arbre.p)
```

```
1) root 100 46 2 (0.46000000 0.54000000)
  2) x2>=0.4271354 58 15 1 (0.74137931 0.25862069)
    4) x2>=0.733562 27  1 1 (0.96296296 0.03703704)
      8) x4>=0.1231597 26  0 1 (1.00000000 0.00000000) *
      9) x4< 0.1231597 1  0 2 (0.00000000 1.00000000) *
    5) x2< 0.733562 31 14 1 (0.54838710 0.45161290)
     10) x1>=0.7221232 8  0 1 (1.00000000 0.00000000) *
     11) x1< 0.7221232 23  9 2 (0.39130435 0.60869565)
       22) x1< 0.3639227 10  1 1 (0.90000000 0.10000000)
         44) x4>=0.02095698 9  0 1 (1.00000000 0.00000000) *
         45) x4< 0.02095698 1  0 2 (0.00000000 1.00000000) *
       23) x1>=0.3639227 13  0 2 (0.00000000 1.00000000) *
  3) x2< 0.4271354 42  3 2 (0.07142857 0.92857143)
    6) x1< 0.1139017 3  0 1 (1.00000000 0.00000000) *
    7) x1>=0.1139017 39  0 2 (0.00000000 1.00000000) *
```

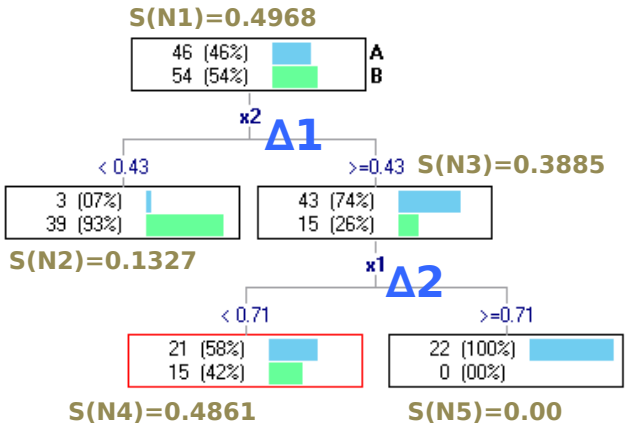
# Decision tree

# (analysis of the information gain)

## Gini index

$$S(noeud) = \sum_{k=1}^{K} p_k (1 - p_k)$$

Statistical dispersion
Impurity measure
Quadratic entropy
Variance for categorical variable




$$SCT = S(N1) = \frac{46}{100}(1 - \frac{46}{100}) + \frac{54}{100}(1 - \frac{54}{100}) = 0.4968$$

$$SCR = \frac{40}{100} \times S(N2) + \frac{36}{100} \times S(N4) + \frac{22}{100} \times S(N5)$$
$$= \frac{40}{100} \times 0.1327 + \frac{36}{100} \times 0.4861 + \frac{22}{100} \times 0.00 = 0.2307$$

Variance explained by the model

$$SCE = SCT - SCR = 0.4968 - 0.2307 = 0.2661$$

It can be decomposed according to the splitting into the internal nodes of the tree.

$$\Delta 1 = \frac{42+58}{100}\left\{S(N1) - \left[\frac{42}{100} \times S(N2) + \frac{58}{100} \times S(N3)\right]\right\} = 0.2187$$

$$\Delta 2 = \frac{36+22}{100}\left\{S(N3) - \left[\frac{36}{58} \times S(N4) + \frac{22}{58} \times S(N5)\right]\right\} = 0.0474$$

**SCE=Δ1+Δ2**

The gain Δ for each internal segmentation is a component of the overall gain (variance explained by the model SCE). The importance decreases as one moves away from the root (because the weight of the node is weak).

# Summary about the decision tree

Advantages:

- "Understandable" model - Easy to interpret
- Tree = rule-based system
- Selection of the relevant attributes is embedded into the learning system
- Non parametric – No assumption about the shape of the variables distribution
- Can handle mixed predictive attributes (continuous, discrete)
- Robust against outliers, solutions for handling missing values
- Quickness and ability to process large databases

- The construction of the model can be guided by domain expert

Drawbacks:

- Instability when we deal with a small dataset
- Do not provide a good estimation of the class membership probability (e.g. for the scoring procedure)
- Often poor performance compared with other approaches (in fact, the performance of the tree heavily relies on the training set size)

# Bagging

Boostrap Aggregating (Breiman, 1996)

# Bagging - Algorithm

**Idea**: Make cooperate (vote) several models fitted to bootstrap samples. The number of models is a parameter of the algorithm.

## Learning

Input : B number of models, ALGO learning algorithm, Ω learning set with n instances, y is the target attribute with K values, X is the matrix of predictive attributes( p attributes).

MODELES = { }
For b = 1 to B Do
  Draw a sample of size n with replacement from Ω → $\Omega_b$
  Fit the model $M_b$ (based on ALGO ) to $\Omega_b$
  Add $M_b$ into MODELES
Fin Pour

## Prediction

For an instance i* to classify,
  Apply each model $M_b$ from MODELES → $\hat{y}_b(i*)$

Bagging prediction ➔ $\hat{y}_{bag}(i*) = \arg\max_k \left[ \sum_{b=1}^{B} I(\hat{y}_b(i*) = y_k) \right]$

➔ Which corresponds to a simple majority vote

# Bagging – Why it is effective?

**Interest of the cooperation.** Make cooperate models has an interest only if the models do not predict all in the same way. If the vote is always unanimous, only one model is enough.

**Bias and variance**. Bias of bagging = bias of the underlying model. Bagging reduces the variance. Therefore, the underlying models must have a low bias, capturing the complexity of the relation between y and X.

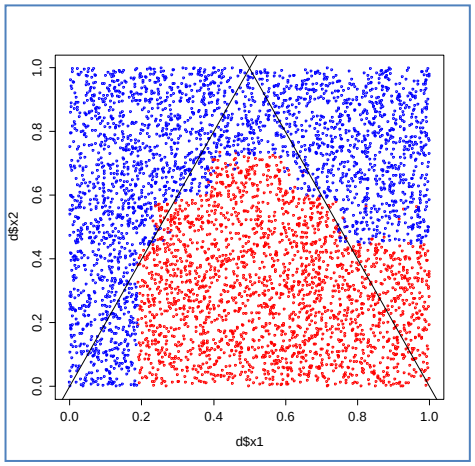
**Weak learners**. Bagging does not take advantage of weak learners (see boosting). The underlying models must be of good quality.

**Overfitting**. Increasing the number of models (B) does not lead to overfitting. In practice, a set of hundred of classifiers is sufficient, but we can adjust it to the study.

**Decision tree**. Bagging can be applied to any kind of model. But the classification trees are particularly interesting because we can reduce individual bias by creating deep trees (unpruned trees). Afterwards, Bagging can reduce the variance in order to obtain an overall good predictor.

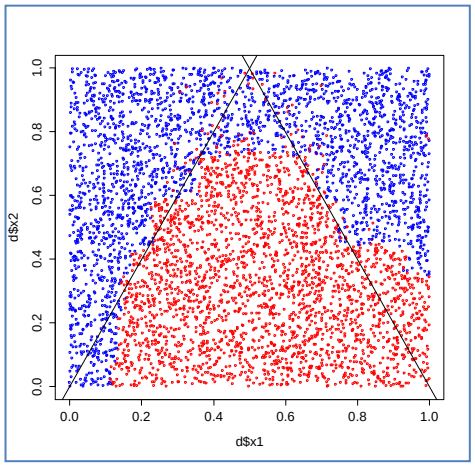
# Apply Bagging to our dataset

The « adabag » package for R proposes the bagging approach for decision trees (the underlying procedure is rpart).




```
#100 tree generated by bagging
model.bagging <- bagging(y ~ ., data = d.train,
mfinal=100)
```

$\varepsilon = 0.156$




```
#modify the parameters to obtain deeper tree
model.bagging.2 <- bagging(y ~ ., data = d.train,
mfinal=100,control=list(cp=0,minsplit=2,minbucket=1))
```

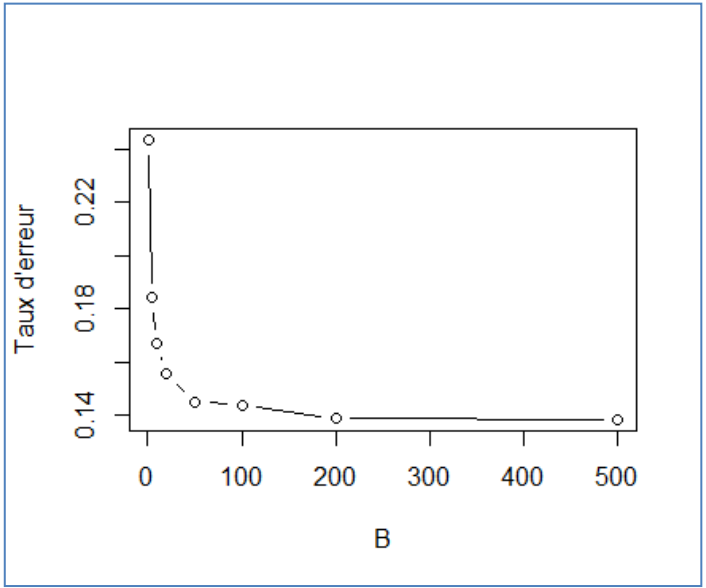
$\varepsilon = 0.1224$

By creating deeper individual trees (less biased), we improve the behavior of the bagging.

# Bagging and the number of underlying classifiers (B)

Impact of the evolution of the number of trees (B) on the quality of the model.

```
#bagging and the number of underlying models
B <- c(1,5,10,20,50,100,200,500)
#one learning and test session for a number of tree b
une_session <- function(b){
  model.bagging <- bagging(y ~ ., data = d.train, mfinal=b)
  p.bagging <- predict(model.bagging,newdata=d.test)
  return(erreur(d.test$y,p.bagging$class))
}
#measuring the error by repeating the session 20 times
errors <- replicate(20,sapply(B,une_session))
m.errors <- apply(errors,1,mean)
plot(B,m.errors,xlab="B",ylab="Taux d'erreur",type="b")
```




Beyond a certain number of trees, there is no more improvements. But the error rate remains stable (no overfitting).

# Bagging and the estimation of the class membership

In some circumstances, we need an estimate of the class membership probabilities. e.g. In a binary problem $Y \in \{+, -\}$, we want a good estimation of $P(Y = + / X)$.

## Solution 1

We can use the frequency of votes to estimate the posterior probability.

$$\hat{P}(Y = + / X) = \frac{\sum_{b=1}^{B} I(\hat{y}_b = +)}{B}$$

## Solution 2

Better in general, especially when the number of classifiers is low. This solution can be beneficially applied to the prediction.

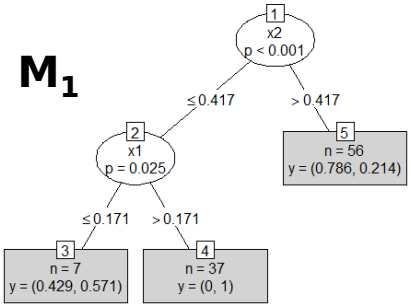
If the individual model $M_b$ can provide an estimate of the class membership probabilities $P_b$, we can add them up, then we normalize to 1.

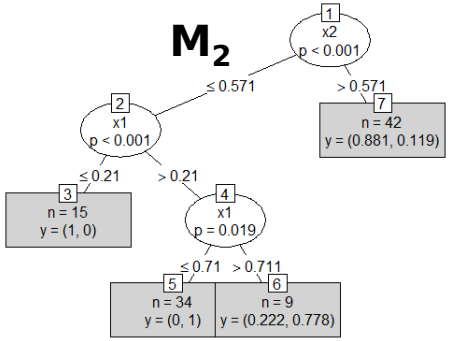
$$\hat{P}(Y = + / X) = \frac{\sum_{b=1}^{B} \hat{P}_b(Y = + / X)}{B}$$

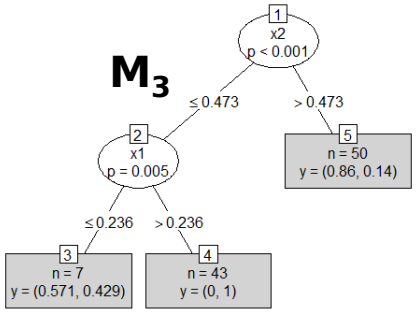
# Bagging

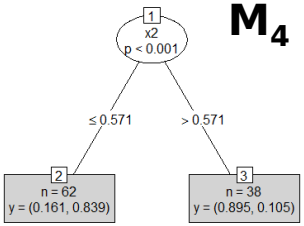
# Variable importance (1)

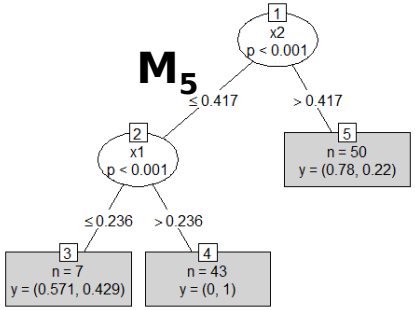
There are a multitude of trees, the overall model is not "understandable" directly. The interpretation of the predictive process is difficult.

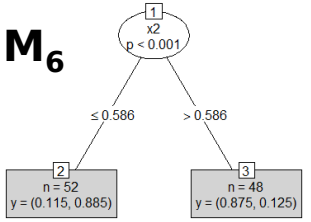



It is not possible to inspect each tree in order to distinguish variables that play a role, and how they play a role.

# Bagging

# Variable Importance (2)

Use the information gain in the tree.

See slide.

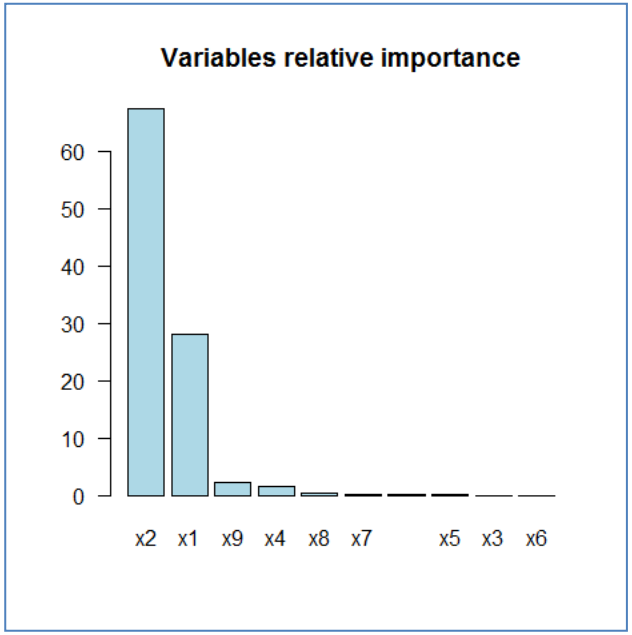
## Idea

Add up the contributions of the variables in the trees for which they appear.

**Note**: Some packages normalize to 100 the most important variable, or the sum of contributions.

## Example

We know for our data that X2 and X1 are relevant; X3 to X10 have no influence in the designation of class membership.




```
#variable importance
#library(adabag)
importanceplot(model.bagging)
```

# Bagging

# OOB Error Estimate

OOB "Out-of-bag error" estimation: Measuring the error directly during the learning process. We do not need a test sample or use additional calculations such as cross-validation.

## Idea

For the construction of the tree $M_b$, some instances are drawn several times, others are not included into the bootstrap sample (≈36.8%).
We classify these unused instances with the tree.

| i | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ | Y^ | Y | ERR |
|---|---|---|---|---|---|---|---|---|
| 1 | . | + | . | + | − | + | + | 0 |
| 2 | . | . | . | − | − | − | − | 0 |
| 3 | + | . | − | − | − | − | + | 1 |
| 4 | . | + | + | . | . | + | + | 0 |
| 5 | + | − | − | . | . | − | + | 1 |
| 6 | . | . | . | . | + | + | − | 1 |
| 7 | − | . | . | − | . | − | − | 0 |
| 8 | − | + | . | . | + | + | + | 0 |
| 9 | . | . | + | . | . | + | + | 0 |
| 10 | + | . | . | + | . | + | − | 1 |
| … | | | | | | | | |

OOB prediction by majority voting of the models (in the row).

## Process

Y={+, −}

B = 5 arbres

$ERR_{OOB}$ = Proportion(ERR)

$Err_{OOB}$ provides a reliable error estimation of the bagged model.

« . » means that the instance is used during the construction of the tree $M_b$
« + » or « − » is the prediction of the tree $M_b$ for the instance « i » which is OOB

# Bagging
# Margins concept

The margin is the difference between the proportion of voting for the correct class and the maximum of voting for another class. For a binary problem {+ , - }, this is the difference between the correct class membership probability and the complementary to 1.

For an instance "i", $Y(i)=y_{k^*}$

$$m(\omega) = \frac{\sum_{b=1}^{B} I(\hat{y}_b(i) = k^*)}{B} - \max_{k \neq k^*} \left[ \frac{\sum_{b=1}^{B} I(\hat{y}_b(i) = k)}{B} \right]$$

The margin can be negative i.e. the instance is misclassified.

Large margin leads to a decision more definite (more stable) for the prediction. More stable = less variance.

Bagging, and more generally ensemble methods, allows to enlarge the margin.

# Random Forest

(Breiman, 2001)

# Random Forest

## Initial observation

Bagging is often dominated by boosting (see later). In 2001, Breiman developed Random Forest, especially intended for underlying decision tree algorithm, which proves to be as good in general as boosting.

## Idea

The bagging is effective if:

1. Each tree is individually efficient.
2. With very low bias - Tree as deep as possible
3. And above all, very much different from each other so that the combination is efficient. Notion of "de-correlation" of the trees.

## How?

Introduce a random disturbance in the construction of the trees. The mechanism of selection of splitting variable for each node is modified.

# Random Forest - Algorithm

## Learning

Input: B number of trees, ALGO decision tree induction algorithm, Ω learning sample (n instances), y target attribute, X matrix of p descriptors, m number of selected attribute for the splitting process at each node with, by default, $m = \sqrt{p}$

MODELES = { }

For b = 1 to B Do

Draw a sample of size n with replacement from Ω → $\Omega_b$

Fit the tree $M_b$ from $\Omega_b$ using ALGO, for which

For each splitting node process:

Select randomly m variables among p

Split the node with the best attribute among m

Add $M_b$ into MODELES

End For

## Prediction

For one instance **i*** to classify,

Apply each tree $M_b$ from MODELES → $\hat{y}_b(i*)$

Random Forest Prediction ➔ $\hat{y}_{rf}(i*) = \arg\max_k \left[ \sum_{b=1}^{B} I(\hat{y}_b(i*) = y_k) \right]$

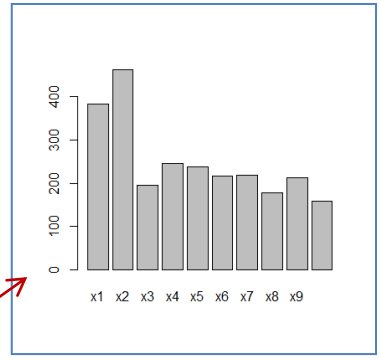
➔ Which corresponds to a simple majority vote

# Random Forest in practice

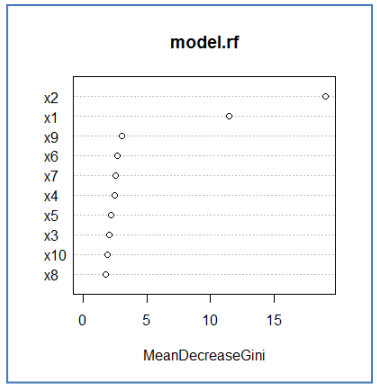
The package « randomForest » for R.

```
#random forest - B = 200 trees
library(randomForest)
model.rf <- randomForest(y ~ ., data = d.train, ntree = 200)

#list of variables used
barplot(varUsed(model.rf),names.arg=paste("x",1:10,sep=""))

#variables importance
varImpPlot(model.rf)

#OOB confusion matrix
print(mc <- model.rf$confusion)

#OOB error rate
print((mc[2,1]+mc[1,2])/sum(mc[1:2,1:2])) # (7+8)/100 = 0.15

#prediction on the test set
p.rf <- predict(model.rf,newdata = d.test,type="class")

#test error rate
print(erreur(d.test$y,p.rf)) # 0.149

#representation of frontiers into (x1,x2)
nuage(d.test,p.rf)
```

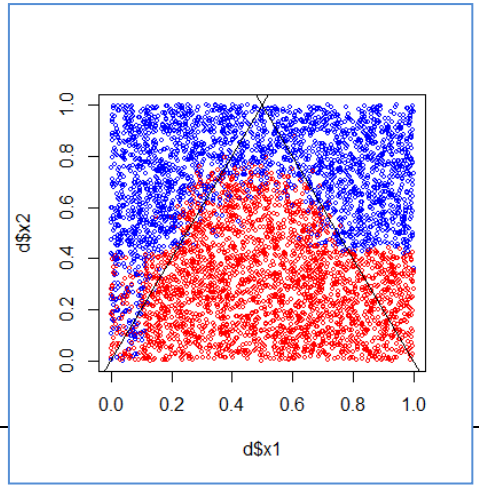
```
   1  2 class.error
1 38  8   0.1739130
2  7 47   0.1296296
```




A variable can occur more than once in a tree.




Another approach based on the OOB exists for estimating the variable importance.



# Random Forest
# Details about the trees

| | left daughter | right daughter | split var | split point | status | prediction |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 1 | 0.7333904 | 1 | 0 |
| 2 | 4 | 5 | 7 | 0.5242057 | 1 | 0 |
| 3 | 6 | 7 | 8 | 0.6437400 | 1 | 0 |
| 4 | 8 | 9 | 3 | 0.3753580 | 1 | 0 |
| 5 | 10 | 11 | 7 | 0.7430234 | 1 | 0 |
| 6 | 12 | 13 | 5 | 0.8513703 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0.0000000 | -1 | 1* |
| 8 | 14 | 15 | 1 | 0.1846170 | 1 | 0 |
| 9 | 16 | 17 | 6 | 0.4151308 | 1 | 0 |
| 10 | 0 | 0 | 0 | 0.0000000 | -1 | 2* |
| 11 | 18 | 19 | 6 | 0.4038910 | 1 | 0 |
| 12 | 0 | 0 | 0 | 0.0000000 | -1 | 1* |
| 13 | 0 | 0 | 0 | 0.0000000 | -1 | 2* |
| 14 | 0 | 0 | 0 | 0.0000000 | -1 | 1* |
| 15 | 0 | 0 | 0 | 0.0000000 | -1 | 2* |
| 16 | 20 | 21 | 10 | 0.4015341 | 1 | 0 |
| 17 | 22 | 23 | 10 | 0.1524530 | 1 | 0 |
| 18 | 24 | 25 | 2 | 0.6465817 | 1 | 0 |
| 19 | 26 | 27 | 2 | 0.6180417 | 1 | 0 |
| 20 | 28 | 29 | 3 | 0.9030539 | 1 | 0 |
| 21 | 0 | 0 | 0 | 0.0000000 | -1 | 2* |
| 22 | 0 | 0 | 0 | 0.0000000 | -1 | 2* |
| 23 | 0 | 0 | 0 | 0.0000000 | -1 | 1* |
| 24 | 0 | 0 | 0 | 0.0000000 | -1 | 2* |
| 25 | 0 | 0 | 0 | 0.0000000 | -1 | 1* |
| 26 | 0 | 0 | 0 | 0.0000000 | -1 | 2* |
| 27 | 0 | 0 | 0 | 0.0000000 | -1 | 1* |
| 28 | 0 | 0 | 0 | 0.0000000 | -1 | 1* |
| 29 | 0 | 0 | 0 | 0.0000000 | -1 | 2* |

```
#access to one tree (ex. 4th)
print(getTree(model.rf,k=4))
```

- The tree has 15 leaves.
- The splitting variable at the root is x1
- One variable can appear several times (e.g. x2, x6, etc.)
- Irrelevant variables may be incorporated into the tree (e.g. x7, x8, x3, x5, x6, etc.)

# Random Forest - Summary

## Pros

- Good performances in prediction
- Easy to configure (**B** and **m**)
- No overfitting (we can increase B)
- Variable importance measurement
- Built-in error measurement (OOB – Out-of-bag)
- Obvious solutions for parallel programming

## Cons

- Problem if number of relevant variables is very low, in the absolute and relative to the total number of variables

Because individual trees may not be efficient.

- Deploying this kind of model, especially for predictions is not easy. See PMML format PMML (example for IRIS).

Especially if we should program them ourselves in the information system.

# Boosting

Freund & Schapire  - ADABOOST (1996)

# Boosting Ideas

The aim is once again to fit models from different versions of the data. But, the learning is sequential. It is focused on individuals misclassified in the previous step.

Weak learner. A classifier which is only slightly better than the random classification.

Boosting. Combine weak learners to obtain a strong learner.

Weighting of instances. At step (b+1), the idea is to set higher weight to the individuals which are misclassified in the previous step (classifier $M_b$). The constructions of the successive models is sequential by nature.

Weighting of models. The models are weighted according to their performance in the voting process.

Bias and variance. By guiding the learning at each step, boosting reduces the bias; by combining them, it reduces also the variance.

Overfitting. Increasing B does not lead to overfitting (see summary).

Decision tree. Boosting can be applied to any kind of model. But trees are advantageous because we can modulate the properties of the model (more or less in depth - e.g. decision stump)

# Adaboost.M1

## Adaptative boosting

The algorithm is defined first for binary problems Y = {+, -} but it can be applied to multiclass problems under certain adjustments.

Learning

Input: B number of models, ALGO learning algorithm, Ω training set, with size = n, y target attribute, X matrix with p predictive attributes.

MODELES = { }

All the instances have the same weight $\omega^1_i = 1/n$

For b = 1 to B Do

Fit the model $M_b$ from $\Omega(\omega^b)$ using ALGO ($\omega^b$ weighting system at the step b)

Add $M_b$ into MODELES

Calculate the weighted error rate for $M_b$ : $\varepsilon_b = \sum_{i=1}^{n} \omega_i^b \times I(y_i \neq \hat{y}_i)$

If $\varepsilon_b > 0.5$ or $\varepsilon_b = 0$, STOP the process

Else

Calculate $\alpha_b = \ln \frac{1-\varepsilon_b}{\varepsilon_b}$

The weights are updated $\omega_i^{b+1} = \omega_i^b \times \exp[\alpha_b . I(y_i \neq \hat{y}_i)]$

And normalized so that the sum is equal to 1

End For

**Note**: If ($\varepsilon_b > 0.5$), the weights become negative, the process is stopped. This natural condition in a binary problem (K = 2) is very restrictive in multiclass context (K > 2). The learner must do much better than "weak".

Using a weighted voting system during the classification phase.

## Prediction

For one instance i* to classify,

Apply each model $M_b$ from MODELES → $\hat{y}_b(i*)$

Boosting prediction ➔ $$\hat{y}_{M1}(i*) = \arg\max_k \left[ \sum_{b=1}^{B} \alpha_b . I(\hat{y}_b(i*) = y_k) \right]$$

➔ This corresponds to a weighted voting system

**Note**: More the model is effective ($\varepsilon_b$ low), more its weight is high ($\alpha_b$ high), more it will be influential in the decision process.

# Boosting

## Multiclass problem

The condition ($\varepsilon_b < 0.5$) is too restrictive when ($K > 2$).
How to go beyond this?

**1.**

Break down the problem to a set of binary problems modeling (Wikipedia):

• One vs. rest, there are **K** training process to do
→ prediction, applying all classifiers and predicting the label “k” for which the corresponding classifier reports the highest confidence score

• One vs. one, there are **K(K-1)/2** training process to do
→ prediction, all classifiers are applied and the class that got the highest number of "+1" predictions gets predicted by the combined classifier

**2.**

SAMME
(Zhu & al., 2009)

Modify the calculation of α

$$\alpha_b = \ln \frac{1-\varepsilon_b}{\varepsilon_b} + \ln(K-1)$$

What has the effect to alleviate the constraint

$$\varepsilon_b < 1 - \frac{1}{K}$$

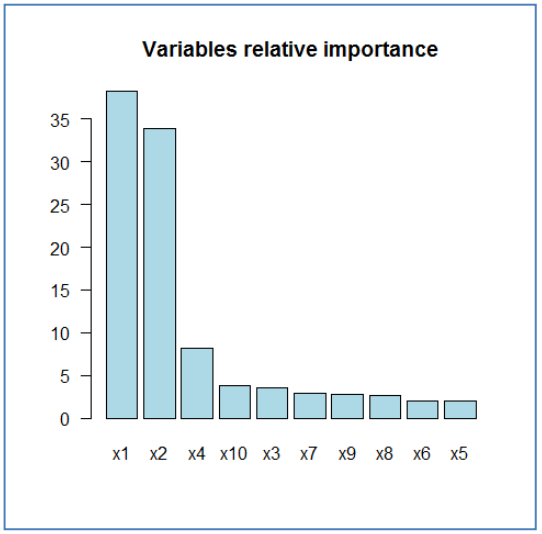
True generalization.
When K = 2, we have the usual Adaboost.M1.

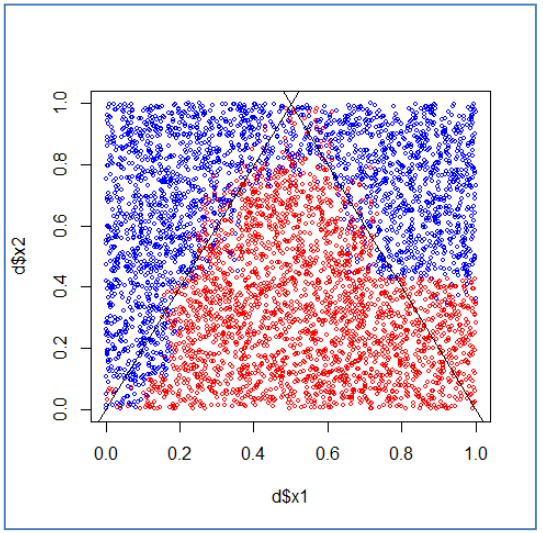
Other approaches (multiclass boosting) exist, but they are not implemented in popular tools.

# Boosting in practice

The package « adabag » for R.

```
library(adabag)

#boosting with100 trees
model.boosting <- boosting(y ~ ., data = d.train, mfinal=100)

#variable importance
importanceplot(model.boosting)

#prediction  on the test sample
p.boosting <- predict(model.boosting,newdata=d.test)

#test error rate
print(erreur(d.test$y,p.boosting$class)) # 0.1242

#frontiers into (x1, x2)
nuage(d.test,factor(p.boosting$class))
```

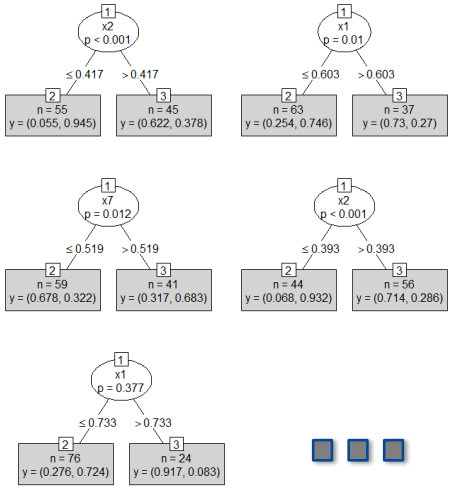


# Boosting
# Decision Stump

Boosting reduces the bias. We can use a very simple models such as the "decision stump" (a one-level decision tree) who have a high bias but a very low variance.

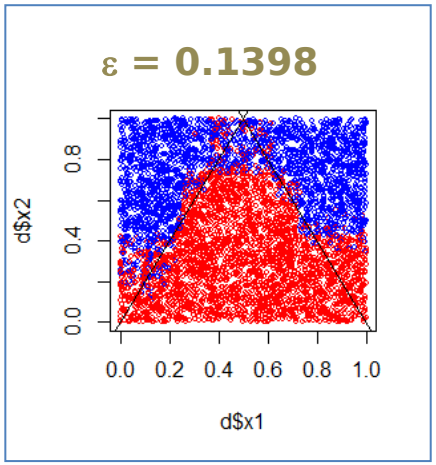
```
library(adabag)

#settings for the decision tree induction
parametres = list(cp=0,maxdepth=1,minbucket=1)

#boosting of 100 decision stumps
stump.boosting <- boosting(y ~ ., data = d.train,
          mfinal=100, control=parametres)
```




(1) We do not take account of interactions in the base model. Yet boosting improves performance because various variables appear round in round in different trees, and that with different splitting thresholds. The model induces a kind of "fuzzy" decision rules.

(2) A two-levels tree would reflect the interactions of order 2 between the variables. Etc. But a lesser bias may increase the variance.

(3) A boosted decision stump leads to a linear classifier. It is especially obvious when we deal with binary predictors (0/1).



# Boosting - Summary

## Pros

- Good performances in prediction
- Easy to configure (**B**)
- Variable importance measurement
- Operate on bias and the variance
- Do not need large tree (quickness)
- We can modulate the depth of the tree in order to take into account the interactions between the variables

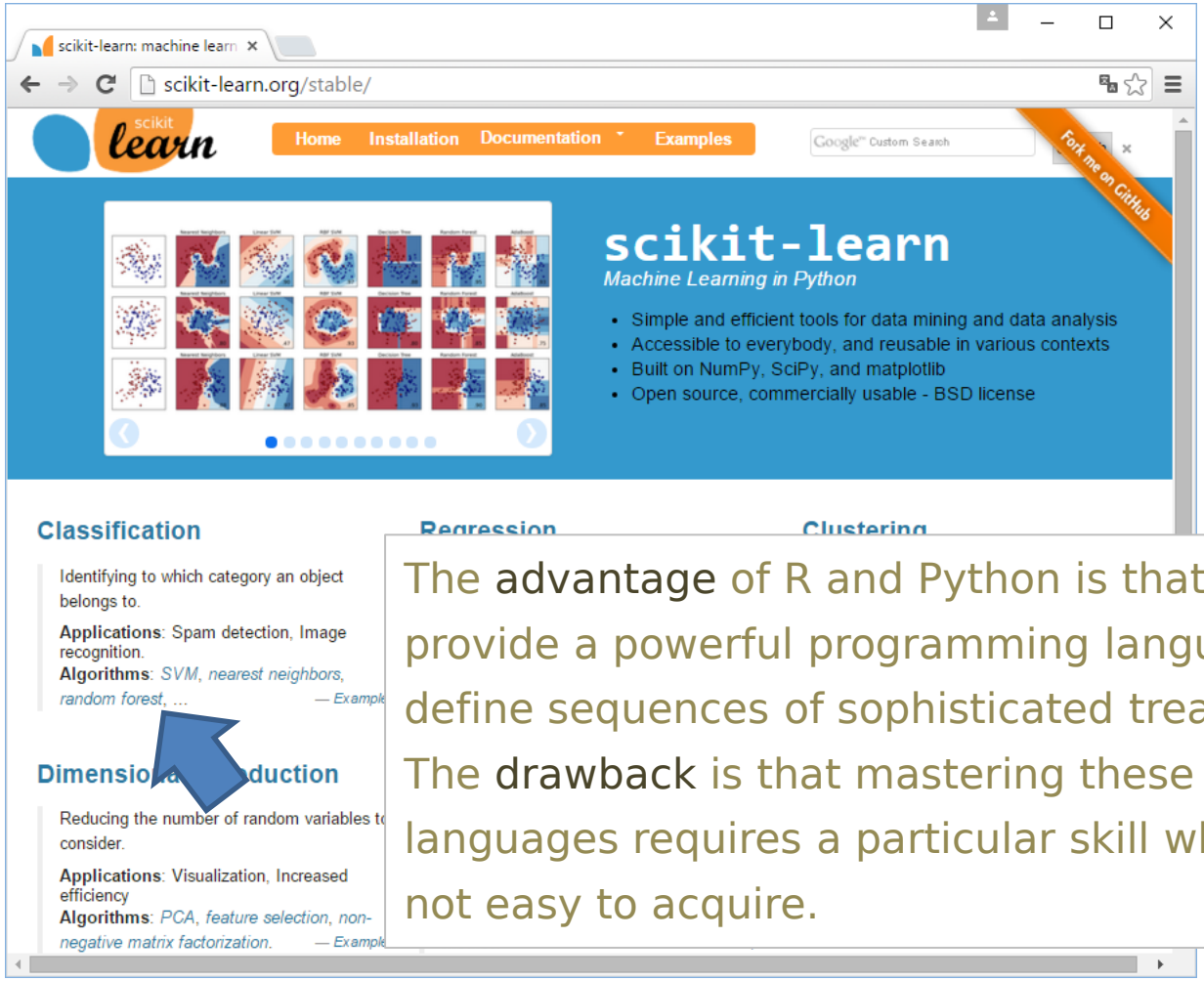
## Cons

- No (obvious) parallelisation of the computations
- If $M_b$ too weak: underfitting
- If $M_b$ too complex (overfit): overfitting
- Not robust against outliers or noise on the class attribute, excessive weights
- Deploying this kind of model for prediction is not easy

# Tools

# R and Python

R and Python incorporate high-performance packages for the ensemble methods. E.g. R with “randomForest”, “adabag”; Python with the module “scikit-learn”.

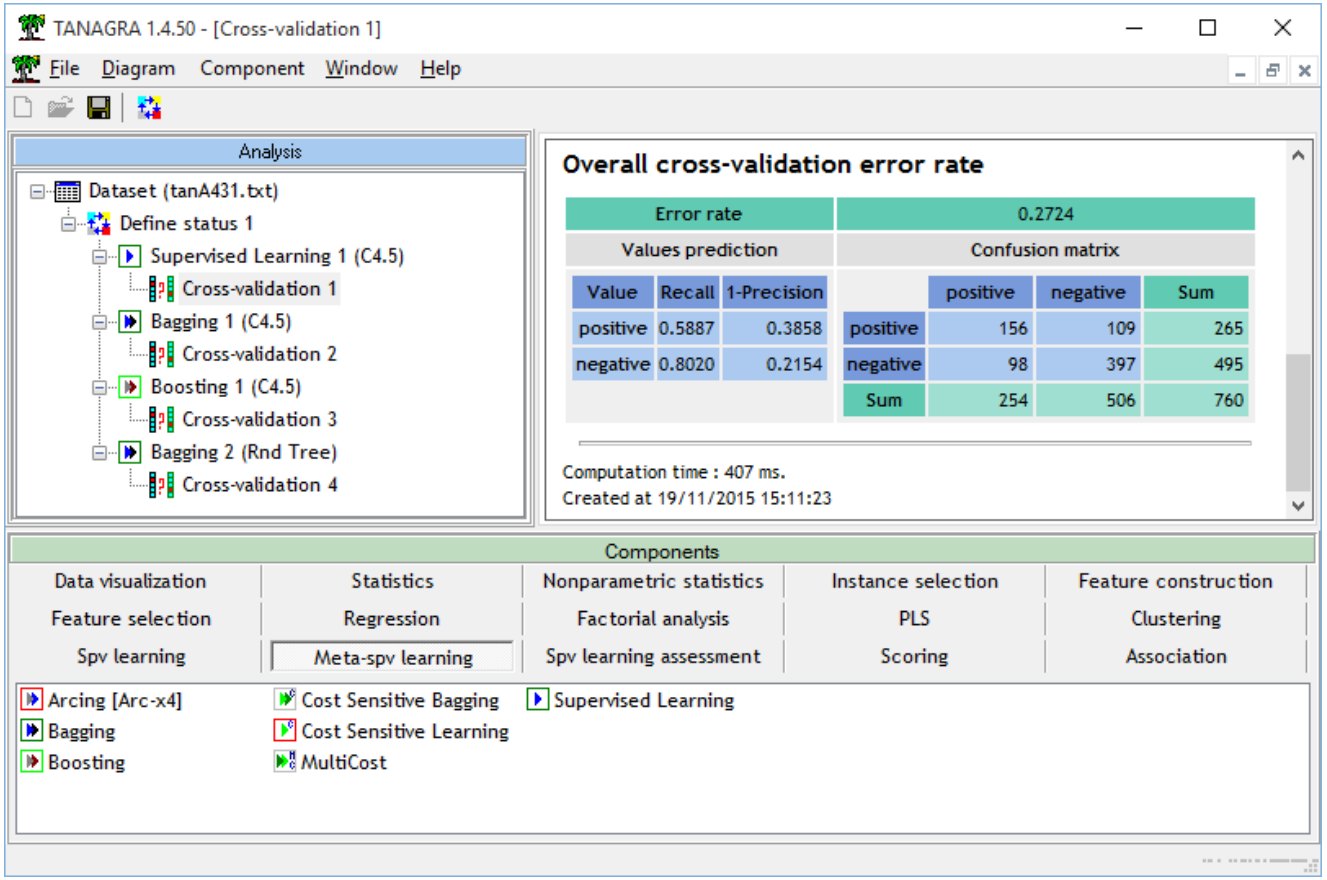



The advantage of R and Python is that they provide a powerful programming language to define sequences of sophisticated treatments. The drawback is that mastering these languages requires a particular skill which is not easy to acquire.

# Tanagra

With Tanagra, any underlying supervised learning algorithm can be combined into an ensemble methods (in the screenshot at right, we use C4.5).

The duo formed by BAGGING + RND TREE leads to the RANDOM FOREST approach.

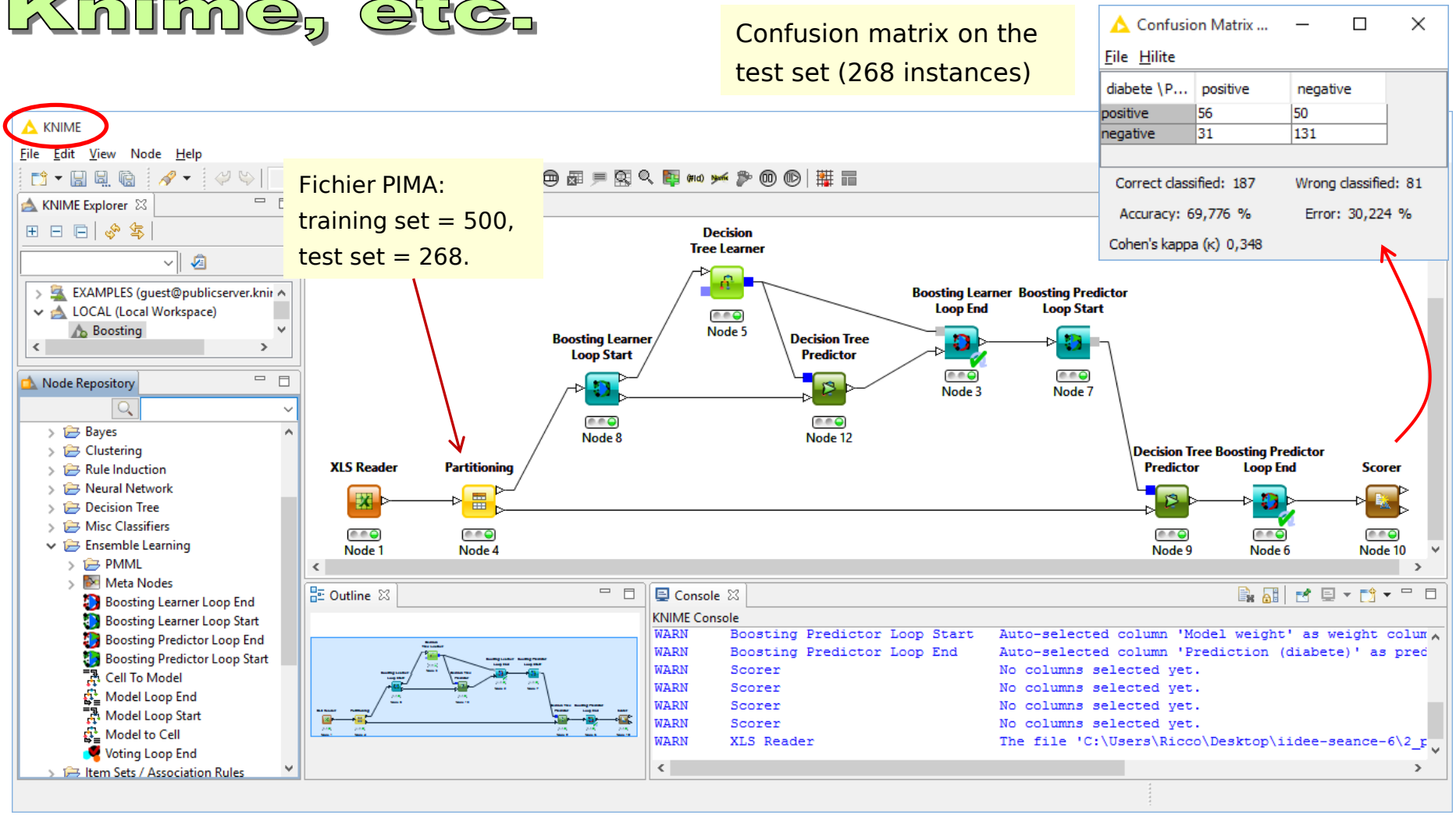



Tanagra, "Random Forest", november 2008.

Tanagra, "PLS Discriminant Analysis – A comparative study", july 2011.

# Knime, etc.

Confusion matrix on the test set (268 instances)




The diagram seems complex, but we identify quite precisely the main steps of the process.

# Summary

# Bagging, Random Forest, Boosting

The ensemble methods that repeatedly apply a learning algorithm on different versions of the data (by resampling or re-weighting) are now well known. These approaches are effective and are well compared with other machine learning algorithms.

The only real drawback is the lack of interpretability of the meta-model, despite the indicator "variable importance". It prevents a sophisticated interpretation of the causal relationship between the input attributes and the target attribute. This is fundamental in some domains.

# References

# State-of-the-art papers

Breiman L., « Bagging Predictors », Machine Learning, 26, p. 123-140, 1996.

Breiman L., « Random Forests », Machine Learning, 45, p. 5-32, 2001.

Freund Y., Schapire R., « Experiments with the new boosting algorithm », International Conference on Machine Learning, p. 148-156, 1996.

Zhu J., Zou H., Rosset S., Hastie T., « Multi-class AdaBoost », Statistics and Its Interface, 2, p. 349-360, 2009.