Many of you might have heard of the concept “Wisdom of the Crowd”: when many people independently guess some quantity, e.g. the number of marbles in a jar glass, the average of their guesses is often pretty accurate – even though many of the guesses are totally off.

The same principle is at work in so called *ensemble methods*, like *bagging* and *boosting*. If you want to know more about boosting and how to turn pseudocode of a scientific paper into valid R code read on…

We start from an original paper of one of the authors of the first practical boosting algorithm, i.e. *AdaBoost*: [Robert E. Schapire: Explaining AdaBoost](http://rob.schapire.net/papers/explaining-adaboost.pdf). The first sentence of the introduction gives the big idea:

Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules.

The second page gives the pseudocode of Adaboost…:

Given: (x_1, y_1),...,(x_m, y_m)where x_i \in X, y_i \in \{-1,+1\}.  
Initialize: D_1(i)=1/mfor 1,...,m.  
For t=1,...,T:

* Train weak learner using distribution D_t.
* Get weak hypothesis h_t: X \rightarrow \{-1,+1\}.
* Aim: select h_twith low weighted error:

\[\epsilon_t=Pr_{i \sim D_t}[h_t(x_i] \neq y_i].\]

* Choose \alpha_t = \frac{1}{2}ln \big(\frac{1-\epsilon_t}{\epsilon_t} \big).
* Update, for i=1,...,m:

\[D_{t+1}(i)=\frac{D_t(i)exp(-\alpha_ty_ih_t(x_i))}{Z_t}\]

where Z_tis a normalization factor (chosen so that D_{t+1}will be a distribution).

Output the final hypothesis:

\[H(x)=sign \Bigg(\sum_{t=1}^{T} \alpha_th_t(x) \Bigg).\]

… with some explanation:

[…] we are given mlabeled training examples (x_1, y_1),...,(x_m, y_m)where the x_i\,'sare in some domain X, and the labels y_i \in \{−1,+1\}. On each round t = 1,...,T, a distribution D_tis computed as in the figure over the mtraining examples, and a given weak learner or weak learning algorithm is applied to find a weak hypothesis h_t: X \rightarrow \{−1,+1\}, where the aim of the weak learner is to find a weak hypothesis with low weighted error \epsilon_trelative to D_t. The final or combined hypothesis Hcomputes the sign of a weighted combination of weak hypotheses

\[F(x) = \sum_{t=1}^{T} \alpha_th_t(x).\]

This is equivalent to saying that His computed as a weighted majority vote of the weak hypotheses h_twhere each is assigned weight \alpha_t. ([…] we use the terms “hypothesis” and “classifier” interchangeably.)

So, AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous ones. But to really understand what is going on my approach has always been that you haven’t really understood something before you didn’t build it yourself…

Perhaps you might want to try to translate the pseudocode into R code before reading on… (to increase your motivation I frankly admit that I also had some errors in my first implementation… which provides a good example of how strong the R community is because I posted it on stackoverflow and got a perfect answer two hours later: [What is wrong with my implementation of AdaBoost?](https://stackoverflow.com/questions/53248179/what-is-wrong-with-my-implementation-of-adaboost)

Anyway, here is my implementation (the data can be found here: <http://freakonometrics.free.fr/myocarde.csv>):

library(rpart)

library(OneR)

maxdepth <- 1

T <- 100 # number of rounds

# Given: (x\_1, y\_1),...,(x\_m, y\_m) where x\_i element of X, y\_i element of {-1, +1}

myocarde <- read.table("data/myocarde.csv", header = TRUE, sep = ";")

y <- (myocarde[ , "PRONO"] == "SURVIE") \* 2 - 1

x <- myocarde[ , 1:7]

m <- nrow(x)

data <- data.frame(x, y)

# Initialize: D\_1(i) = 1/m for i = 1,...,m

D <- rep(1/m, m)

H <- replicate(T, list())

a <- vector(mode = "numeric", T)

set.seed(123)

# For t = 1,...,T

for(t in 1:T) {

# Train weak learner using distribution D\_t

# Get weak hypothesis h\_t: X -> {-1, +1}

H[[t]] <- rpart(y ~., data = data, weights = D, maxdepth = maxdepth, method = "class")

# Aim: select h\_t with low weighted error: e\_t = Pr\_i~D\_t[h\_t(x\_i) != y\_i]

h <- predict(H[[t]], x, type = "class")

e <- sum((h!=y) \* D)

# Choose a\_t = 0.5 \* log((1-e) / e)

a[t] <- 0.5 \* log((1-e) / e)

# Update for i = 1,...,m: D\_t+1(i) = (D\_t(i) \* exp(-a\_t \* y\_i \* h\_t(x\_i))) / Z\_t

# where Z\_t is a normalization factor (chosen so that Dt+1 will be a distribution)

D <- D \* exp(-a[t] \* y \* as.numeric(as.character(h)))

D <- D / sum(D)

}

# Output the final hypothesis: H(x) = sign(sum of a\_t \* h\_t(x) for t=1 to T)

newdata <- x

H\_x <- sapply(H, function(x) as.numeric(as.character(predict(x, newdata = newdata, type = "class"))))

H\_x <- t(a \* t(H\_x))

pred <- sign(rowSums(H\_x))

eval\_model(pred, y)

##

## Confusion matrix (absolute):

## Actual

## Prediction -1 1 Sum

## -1 29 0 29

## 1 0 42 42

## Sum 29 42 71

##

## Confusion matrix (relative):

## Actual

## Prediction -1 1 Sum

## -1 0.41 0.00 0.41

## 1 0.00 0.59 0.59

## Sum 0.41 0.59 1.00

##

## Accuracy:

## 1 (71/71)

##

## Error rate:

## 0 (0/71)

##

## Error rate reduction (vs. base rate):

## 1 (p-value < 2.2e-16)

Let’s compare this with the result from the package JOUSBoost (on CRAN):

library(JOUSBoost)

## JOUSBoost 2.1.0

boost <- adaboost(as.matrix(x), y, tree\_depth = maxdepth, n\_rounds = T)

pred <- predict(boost, x)

eval\_model(pred, y)

##

## Confusion matrix (absolute):

## Actual

## Prediction -1 1 Sum

## -1 29 0 29

## 1 0 42 42

## Sum 29 42 71

##

## Confusion matrix (relative):

## Actual

## Prediction -1 1 Sum

## -1 0.41 0.00 0.41

## 1 0.00 0.59 0.59

## Sum 0.41 0.59 1.00

##

## Accuracy:

## 1 (71/71)

##

## Error rate:

## 0 (0/71)

##

## Error rate reduction (vs. base rate):

## 1 (p-value < 2.2e-16)

As you can see: zero errors as with my implementation. Two additional remarks are in order:

An accuracy of 100% hints at one of the problems of boosting: it is prone to *overfitting* (see also [Learning Data Science: Modelling Basics](http://blog.ephorie.de/learning-data-science-modelling-basics)).

The second problem is the lack of *interpretability*: whereas decision trees are normally well interpretable ensembles of them are not. This is also known under the name *Accuracy-Interpretability Trade-Off* (another often used ensemble method is random forests, see also here: [Learning Data Science: Predicting Income Brackets](http://blog.ephorie.de/learning-data-science-predicting-income-brackets)).

Hope that this post was helpful for you to understand the widely used boosting methodology better and to see how you can get from pseudocode to valid R code. If you have any questions or feedback please let me know in the comments – Thank you and stay tuned!