Today, we’ll see the heuristics of the algorithm inside bagging techniques.

Often, bagging is associated with trees, to generate forests. But actually, it is possible using bagging for any kind of model. Recall that [bagging](https://en.wikipedia.org/wiki/Bootstrap_aggregating) means “boostrap aggregation”. So, consider a model . Let denote the estimator of obtained from sample with .

Consider now some boostrap sample, with is randomly drawn from (with replacement). Based on that sample, estimate . Then draw many samples, and consider the agregation of the estimators obtained, using either a majority rule, or using the average of probabilities (if a probabilist model was considered).

**Bagging logistic regression #1**

Consider the case of the logistic regression. To generate a bootstrap sample, it is natural to use the technique describe above. I.e. draw pairs randomly, uniformly (with probability ) with replacement. Consider here the small dataset, just to visualize. For the **b** part of **b**agging, use the following code

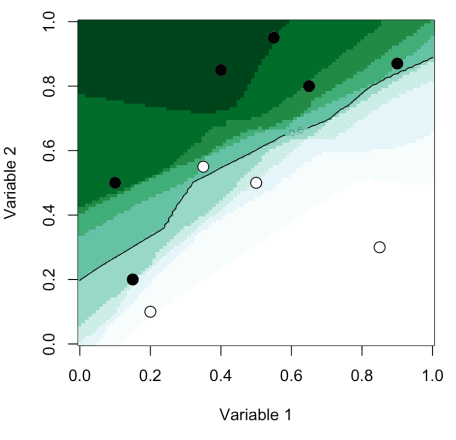
|  |  |
| --- | --- |
| 1  2  3  4  5 | L\_logit = **list**()  n = **nrow**(**df**)  **for**(s **in** 1:1000){  df\_s = **df**[**sample**(1:n,size=n,**replace**=TRUE),]  L\_logit[[s]] = **glm**(y~., df\_s, **family**=**binomial**)} |

Then we should aggregate over the 1000 models, to get the **agg** part of b**agg**ing,

|  |  |
| --- | --- |
| 1  2  3 | p = **function**(x){  nd=**data.frame**(x1=x[1], x2=x[2])  **unlist**(**lapply**(1:1000,**function**(z) **predict**(L\_logit[[z]],newdata=nd,type="response")))} |

We now have a prediction for any new observation

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | vu = **seq**(0,1,**length**=101)  vv = **outer**(vu,vu,**Vectorize**(**function**(x,y) **mean**(p(**c**(x,y)))))  **image**(vu,vu,vv,xlab="Variable 1",ylab="Variable 2",**col**=clr10,breaks=(0:10)/10)  **points**(**df**$x1,**df**$x2,pch=19,cex=1.5,**col**="white")  **points**(**df**$x1,**df**$x2,pch=**c**(1,19)[1+(**df**$y=="1")],cex=1.5)  **contour**(vu,vu,vv,**levels** = .5,add=TRUE) |



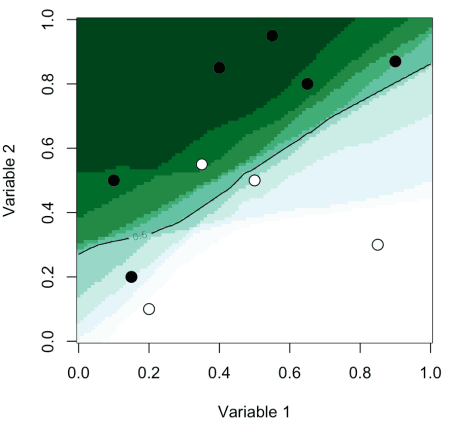
**Bagging logistic regression #2**

Another technique that can be used to generate a bootstrap sample is to keep all ‘s, but for each of them, to draw (randomly) a value for , withsinceThus, the code for the **b** part of **b**agging algorithm is now

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | L\_logit = **list**()  n = **nrow**(**df**)  reg = **glm**(y~x1+x2, **df**, **family**=**binomial**)  **for**(s **in** 1:100){  df\_s = **df**  df\_s$y = **factor**(**rbinom**(n,size=1,prob=**predict**(reg,type="response")),**labels**=0:1)  L\_logit[[s]] = **glm**(y~., df\_s, **family**=**binomial**)  } |

The **agg** part of b**agg**ing algorithm remains unchanged. Here we obtain

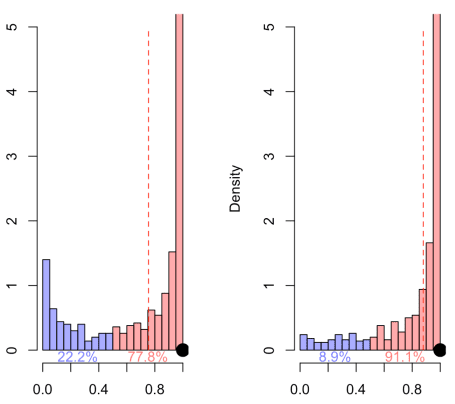
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | vu = **seq**(0,1,**length**=101)  vv = **outer**(vu,vu,**Vectorize**(**function**(x,y) **mean**(p(**c**(x,y)))))  **image**(vu,vu,vv,xlab="Variable 1",ylab="Variable 2",**col**=clr10,breaks=(0:10)/10)  **points**(**df**$x1,**df**$x2,pch=19,cex=1.5,**col**="white")  **points**(**df**$x1,**df**$x2,pch=**c**(1,19)[1+(**df**$y=="1")],cex=1.5)  **contour**(vu,vu,vv,**levels** = .5,add=TRUE) |

  
Of course, we can use that code we check the prediction obtain on the observations we have in our sample. Just to change, consider here the myocarde data. The entiere code is here

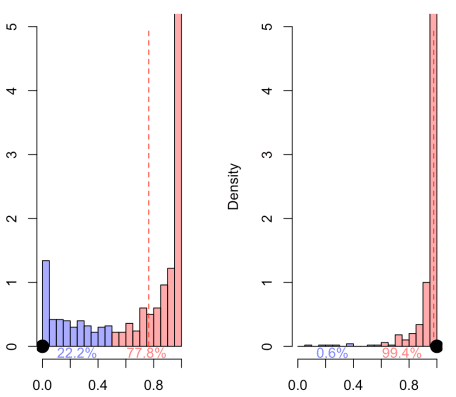
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | L\_logit = **list**()  reg = **glm**(**as.factor**(PRONO)~., myocarde, **family**=**binomial**)  **for**(s **in** 1:1000){  myocarde\_s = myocarde  myocarde\_s$PRONO = 1\***rbinom**(n,size=1,prob=**predict**(reg,type="response"))  L\_logit[[s]] = **glm**(**as.factor**(PRONO)~., myocarde\_s, **family**=**binomial**)  }  p = **function**(x){  nd=**data.frame**(FRCAR=x[1], INCAR=x[2], INSYS=x[3], PRDIA=x[4],  PAPUL=x[4], PVENT=x[5], REPUL=x[6])  **unlist**(**lapply**(1:1000,**function**(z) **predict**(L\_logit[[z]],newdata=nd,type="response")))} |

For the first observation, with our 1000 simulated datasets, and our 1000 models, we obtained the following estimation for the probability to die.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | histo = **function**(i){  x = **as.numeric**(myocarde[i,1:7])  v\_x = p(x)  **hist**(v\_x,proba=TRUE,breaks=**seq**(0,1,**by**=.05),xlab="",main="",  **col**=**rep**(**c**(**rgb**(0,0,1,.4),**rgb**(1,0,0,.4)),each=10),ylim=**c**(0,5))  **segments**(**mean**(v\_x),0,**mean**(v\_x),5,**col**="red",lty=2)  **points**(myocarde$PRONO[i],0,pch=19,cex=2)  xi = **round**(**mean**(v\_x.5)\*1000)/10  **text**(.75,-.1,**paste**(xi,"%",sep=""),**col**=**rgb**(1,0,0,.6))}  histo(1)  histo(4) |

Hence, for the first observation, in 77.8% of the models, the predicted probability was higher than 50%, and the average probability was actually close to 75%.  
  
or, for observation 22, predictions very close to the first one (except that the first one died, while the 22nd survived)

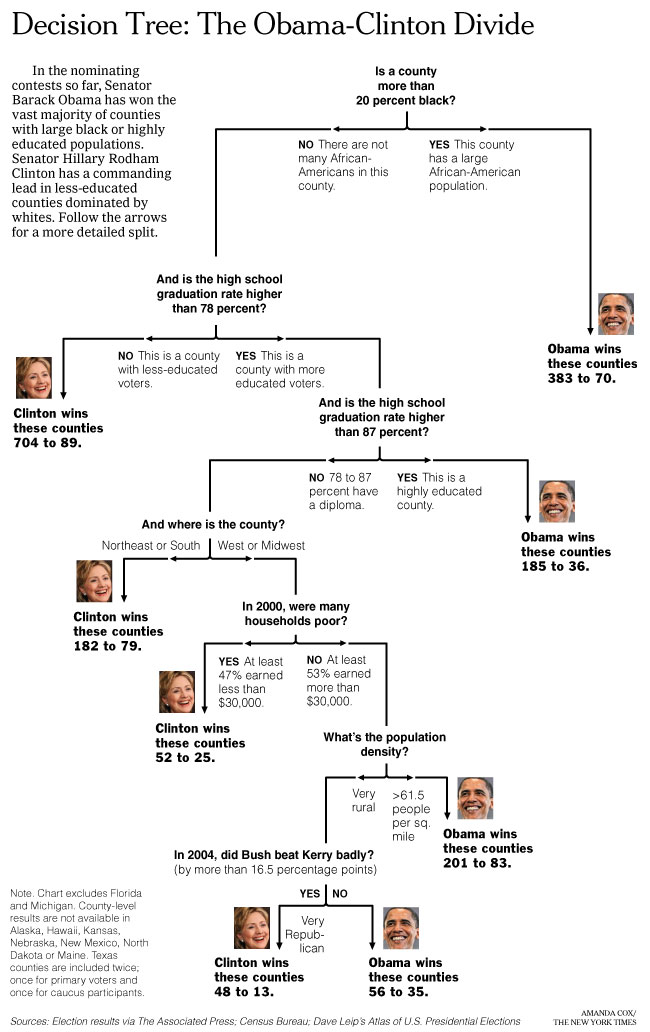
|  |  |
| --- | --- |
| 1  2 | histo(23)  histo(11) |

and, we observe here  


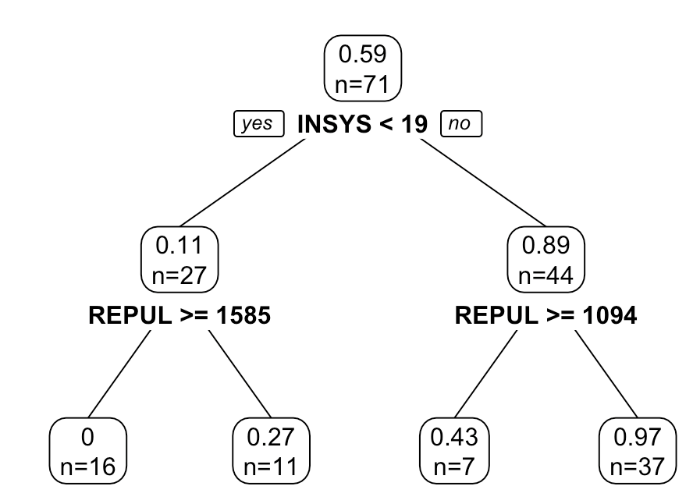
**Bagging trees**

If the first section describes the procedure, the second one introduces “Bagging Classification Trees”. Trees are nice for interpretation, but most of the time, they are rather poor predictors. The idea of bagging was to improve the accuracy of classification trees.

# Code Chunks - CLASSIFICATION FROM SCRATCH, TREES 9/8

  
We start from the top, and we go down, with a binary choice, at each stop, each node. Let us see how it works on our dataset

|  |
| --- |
| **library**(rpart)  cart = rpart(PRONO~.,**data**=myocarde)  **library**(rpart.plot)  prp(cart,type=2,extra=1) |

  
We start here with one single leaf. If we have two explanatory variable (the x*x*-axis and the y*y*-axis if we want to plot it), we will check what happens if we cut the leaf accoring to the value of the first variable (and there will be two subgroups, the one on the left and the one on the right)  
  
or if we cut according to the second one (and there will be two subgroups, the one on top and the one below).  
  
Why and where do we cut? Let us formalize a little bit. A node (a leaf) constains observations, i.e. \{y\_i,\mathbf{x})i\}){*yi*​,**x**)*i*}) for some i\in\mathcal{I}\subset\{1,\cdots,n\}*i*∈I⊂{1,⋯,*n*}. Hence, a leaf a caracterized by \mathcal{I}I. For instance, the first node in the tree is \mathcal{I}=\{1,\cdots,n\}I={1,⋯,*n*}. A (binary) split is based on one specific variable – say x\_j*xj*​ – and a cutoff, say s*s*. Then, there are two options:

* either x\_{i,j}\leq s*xi*,*j*​≤*s*, then observation i*i* goes on the left, in \mathcal{I}\_LI*L*​
* or x\_{i,j}> s*xi*,*j*​>*s*, then observation i*i* goes on the right, in \mathcal{I}\_RI*R*​

Thus, \mathcal{I}=\mathcal{I}\_L\cup\mathcal{I}\_RI=I*L*​∪I*R*​.

Now, define some impurity index, in some node. In the context of a classification tree, the most popular index used (the so-called impurity index) is [Gini](https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity) for node \mathcal{I}I is defined asG(\mathcal{I})=-\sum\_{y\in\{0,1\}}p\_y(1-p\_y)*G*(I)=−*y*∈{0,1}∑​*py*​(1−*py*​)where p\_y*py*​ is the proportion of individuals in the leaf of type y*y*. I use this notation here because it can be extended to the case of more than one class. Here, we consider only binary classification. Now, why p\_y(1-p\_y)*py*​(1−*py*​)? Because we want leaves that are extremely homogeneous. In our dataset, out of 71 individuals, 42 died, 29 survived. A perfect classification would be obtained if we can split in two, with the 29 survivors on the left, and the 42 dead on the right. In that case, leaves would be perfectly homogneous. So, when p\_0\approx1*p*0​≈1 or p\_1\approx1*p*1​≈1, we have strong homogenity. If we want an index to maximize, -p\_y(1-p\_y)−*py*​(1−*py*​) might be an interesting candidate. Further more, the worst case would be a leaf with p\_0\approx1/2*p*0​≈1/2, which is exactly what we have here. Note that we can also writeG(\mathcal{I})=-\sum\_{y\in\{0,1\}}\frac{n\_{y,\mathcal{I}}}{n\_{\mathcal{I}}}\left(1-\frac{n\_{y,\mathcal{I}}}{n\_{\mathcal{I}}}\right)*G*(I)=−*y*∈{0,1}∑​*n*I​*ny*,I​​(1−*n*I​*ny*,I​​)where n\_{y,\mathcal{I}}*ny*,I​ is the number of individuals of type y*y* in the leaf \mathcal{I}I, and n\_{\mathcal{I}}*n*I​ is the number of individuals in the leaf \mathcal{I}I.

If we do not split, we have indexG(\mathcal{I})=-\sum\_{y\in\{0,1\}}\frac{n\_{y,\mathcal{I}}}{n\_{\mathcal{I}}}\left(1-\frac{n\_{y,\mathcal{I}}}{n\_{\mathcal{I}}}\right)*G*(I)=−*y*∈{0,1}∑​*n*I​*ny*,I​​(1−*n*I​*ny*,I​​)while if we split, define indexG(\mathcal{I}\_L,\mathcal{I}\_R)=-\sum\_{x\in\{L,R\}}\frac{n\_x}{n\_{\mathcal{I}\_x}}{n\_{\mathcal{I}}}\sum\_{y\in\{0,1\}}\frac{n\_{y,\mathcal{I}\_x}}{n\_{\mathcal{I}\_x}}\left(1-\frac{n\_{y,\mathcal{I}\_x}}{n\_{\mathcal{I}\_x}}\right)*G*(I*L*​,I*R*​)=−*x*∈{*L*,*R*}∑​*n*I*x*​​*nx*​​*n*I​*y*∈{0,1}∑​*n*I*x*​​*ny*,I*x*​​​(1−*n*I*x*​​*ny*,I*x*​​​)The code to compute is would be

|  |
| --- |
| gini = **function**(y,classe){  T. = **table**(y,classe)  nx = **apply**(**T**,2,**sum**)  n. = **sum**(**T**)  pxy = **T**/**matrix**(**rep**(nx,each=2),**nrow**=2)  omega = **matrix**(**rep**(nx,each=2),**nrow**=2)/n  g. = -**sum**(omega\*pxy\*(1-pxy))  **return**(g)} |

Actually, one can consider other indices, like the entropic measureE(\mathcal{I})=-\sum\_{y\in\{0,1\}}\frac{n\_{y,\mathcal{I}}}{n\_{\mathcal{I}}}\log\left(\frac{n\_{y,\mathcal{I}}}{n\_{\mathcal{I}}}\right)*E*(I)=−*y*∈{0,1}∑​*n*I​*ny*,I​​log(*n*I​*ny*,I​​)while if we split,E(\mathcal{I}\_L,\mathcal{I}\_R)=-\sum\_{x\in\{L,R\}}\frac{n\_x}{n\_{\mathcal{I}\_x}}{n\_{\mathcal{I}}}\sum\_{y\in\{0,1\}}\frac{n\_{y,\mathcal{I}\_x}}{n\_{\mathcal{I}\_x}}\log\left(\frac{n\_{y,\mathcal{I}\_x}}{n\_{\mathcal{I}\_x}}\right)*E*(I*L*​,I*R*​)=−*x*∈{*L*,*R*}∑​*n*I*x*​​*nx*​​*n*I​*y*∈{0,1}∑​*n*I*x*​​*ny*,I*x*​​​log(*n*I*x*​​*ny*,I*x*​​​)

|  |
| --- |
| entropy = **function**(y,classe){  T. = **table**(y,classe)  nx = **apply**(**T**,2,**sum**)  n. = **sum**(**T**)  pxy = **T**/**matrix**(**rep**(nx,each=2),**nrow**=2)  omega = **matrix**(**rep**(nx,each=2),**nrow**=2)/n  g = **sum**(omega\*pxy\***log**(pxy))  **return**(g)} |

This index was used originally in [C4.5](https://en.wikipedia.org/wiki/C4.5_algorithm) algorithm.

## Dividing a leaf (or not)

For instance, consider the very first split. Assume that we want to split according to the very first variable

|  |
| --- |
| CLASSE = myocarde[,1] &lt;=100  **table**(CLASSE)  CLASSE  FALSE TRUE  13 58 |

In that case, there will be 13 invididuals on one side (the left, say), and 58 on the other side (the right).

|  |
| --- |
| gini(y=myocarde$PRONO,classe=CLASSE)  [1] -0.4640415 |

Initially, without any split, it was

|  |
| --- |
| -2\***mean**(myocarde$PRONO)\*(1-**mean**(myocarde$PRONO))  [1] -0.4832375 |

which can actually also be obtained with

|  |
| --- |
| CLASSE = myocarde[,1] gini(y=myocarde$PRONO,classe=CLASSE)  [1] -0.4832375 |

There is a net gain in spliting of

|  |
| --- |
| gini(y=myocarde$PRONO,classe=(myocarde[,1]&lt;=100))-  gini(y=myocarde$PRONO,classe=(myocarde[,1]&lt;=Inf))  [1] 0.01919591 |

Now, how do we split? Which variable and which cutoff? Well… let’s try all possible splits… Here, we have 7 variables. We can consider all possible values, using

|  |
| --- |
| **sort**(**unique**(myocarde[,1])) |

But in massive datasets, it can be very long. Here, I prefer

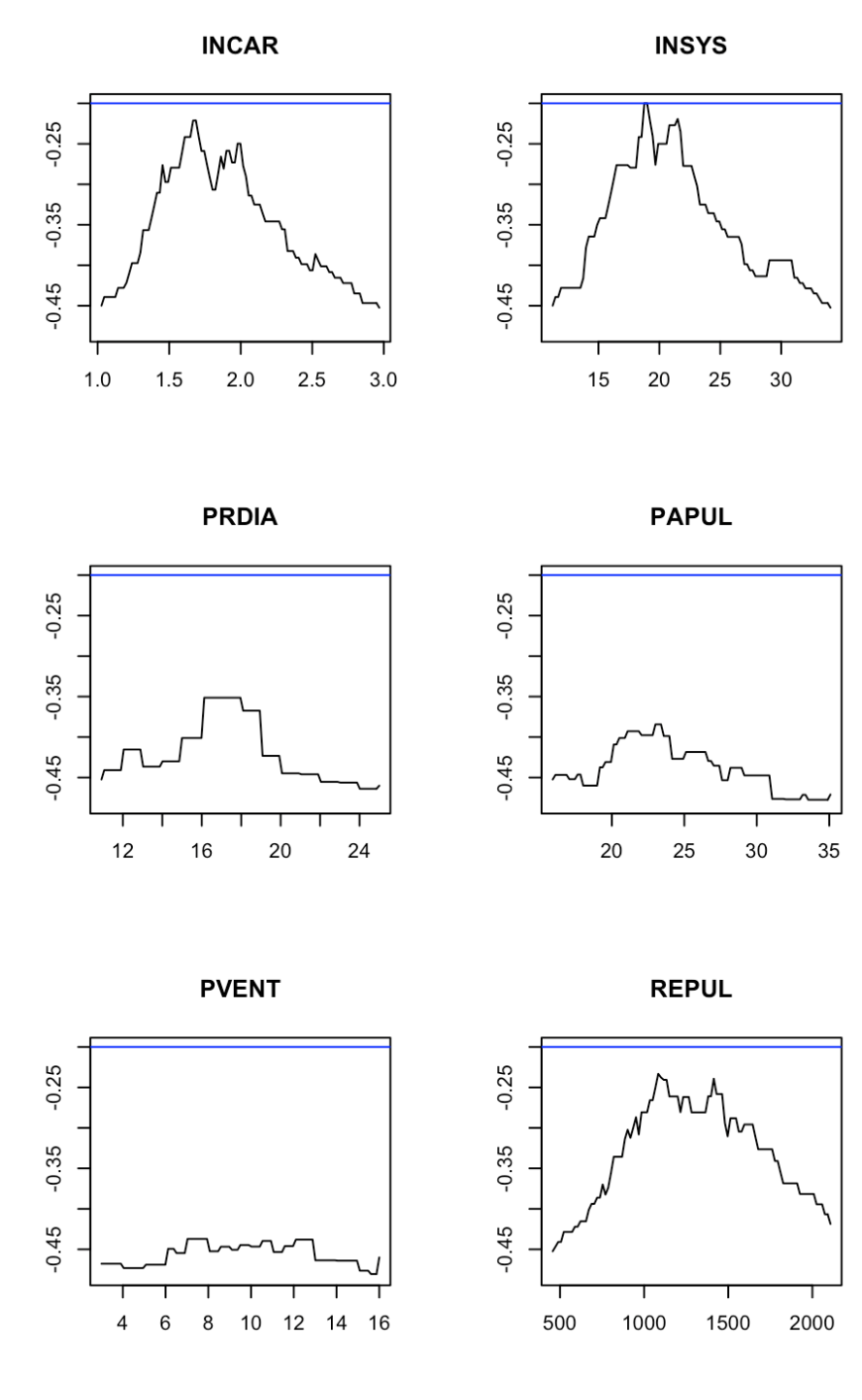
|  |
| --- |
| **seq**(**min**(myocarde[,1]),**max**(myocarde[,1]),**length**=101) |

so that we try 101 values of possible cutoff. Overall, the number of computations is rather low, with 707 Gini indices to compute. Again, I won’t get back here on the motivations for such a technique to create partitions, I will keep that for the course in Barcelona, but it is fast.

|  |
| --- |
| mat\_gini = mat\_v=**matrix**(NA,7,101)  **for**(v **in** 1:7){  variable=myocarde[,v]  v\_seuil=**seq**(**quantile**(myocarde[,v],  6/**length**(myocarde[,v])),  **quantile**(myocarde[,v],1-6/**length**(  myocarde[,v])),**length**=101)  mat\_v[v,]=v\_seuil  **for**(i **in** 1:101){  CLASSE=variable&lt;=v\_seuil[i]  mat\_gini[v,i]=  gini(y=myocarde$PRONO,classe=CLASSE)}} |

Actually, the range of possible values is slightly different: I do not want cutoff too much on the left or on the right… having a leaf with one or two observations is not the idea, here. Not, if we plot all the functions, we get

|  |
| --- |
| **par**(mfrow=**c**(3,2))  **for**(v **in** 2:7){  **plot**(mat\_v[v,],mat\_gini[v,],type="l",  ylim=**range**(mat\_gini),xlab="",ylab="",  main=**names**(myocarde)[v])  **abline**(h=**max**(mat\_gini),**col**="blue")  } |

  
Here, the most homogenous leaves obtained using a cut in two parts is when we use variable ‘INSYS’. And the optimal cutoff variable is close to 19. So far, that’s the only information we use. Well, actually no. If the gain is sufficiently large, we go for a split. Here, the gain is

|  |
| --- |
| gini(y=myocarde$PRONO,classe=(myocarde[,3]&lt;19))-  gini(y=myocarde$PRONO,classe=(myocarde[,3]&lt;=Inf))  [1] 0.2832801 |

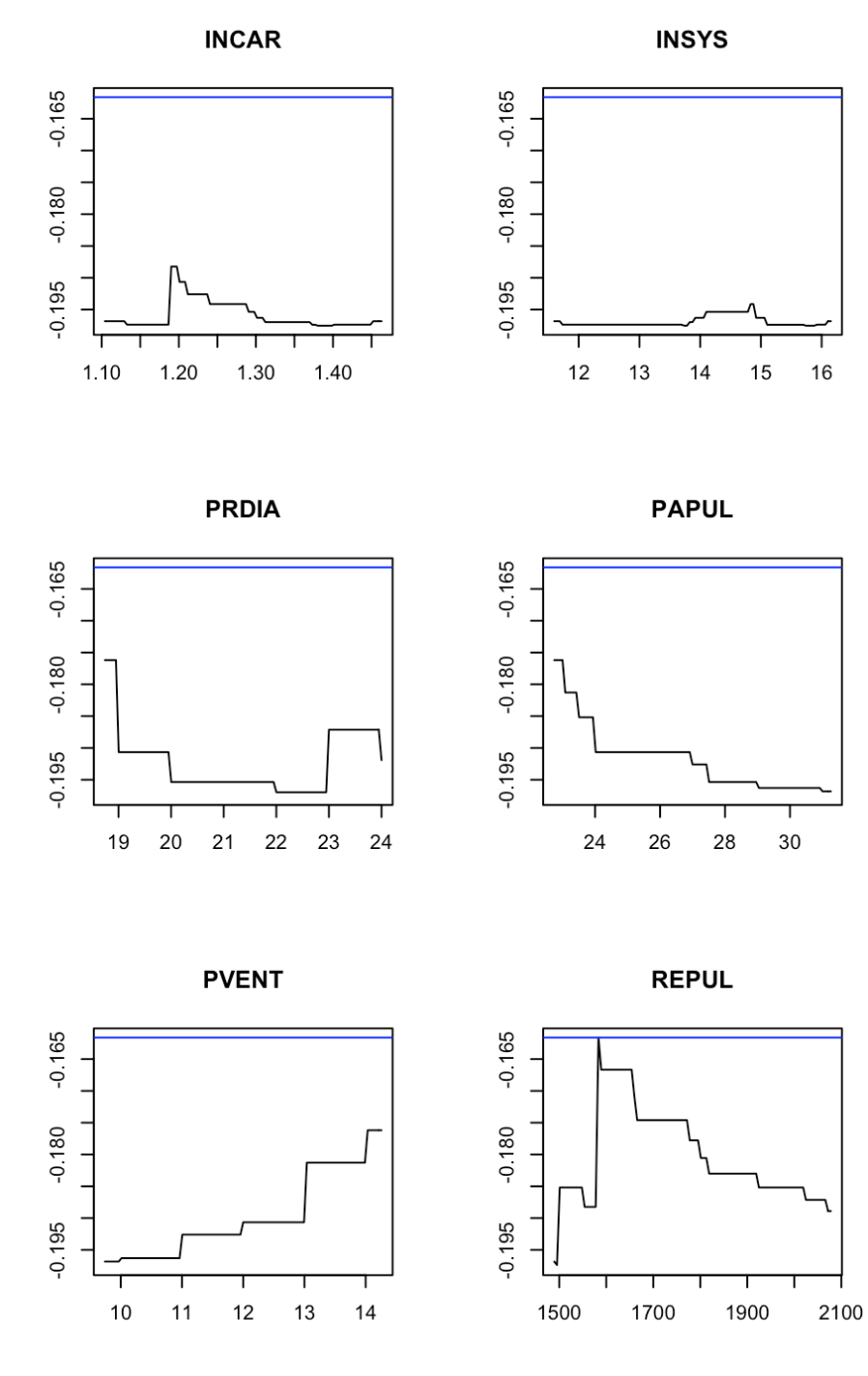
which is large. Sufficiently large to go for it, and to split in two. Actually, we look at the relative gain

|  |
| --- |
| -(gini(y=myocarde$PRONO,classe=(myocarde[,3]&lt;19))-  gini(y=myocarde$PRONO,classe=(myocarde[,3]&lt;=Inf)))/  gini(y=myocarde$PRONO,classe=(myocarde[,3]&lt;=Inf))  [1] 0.5862131 |

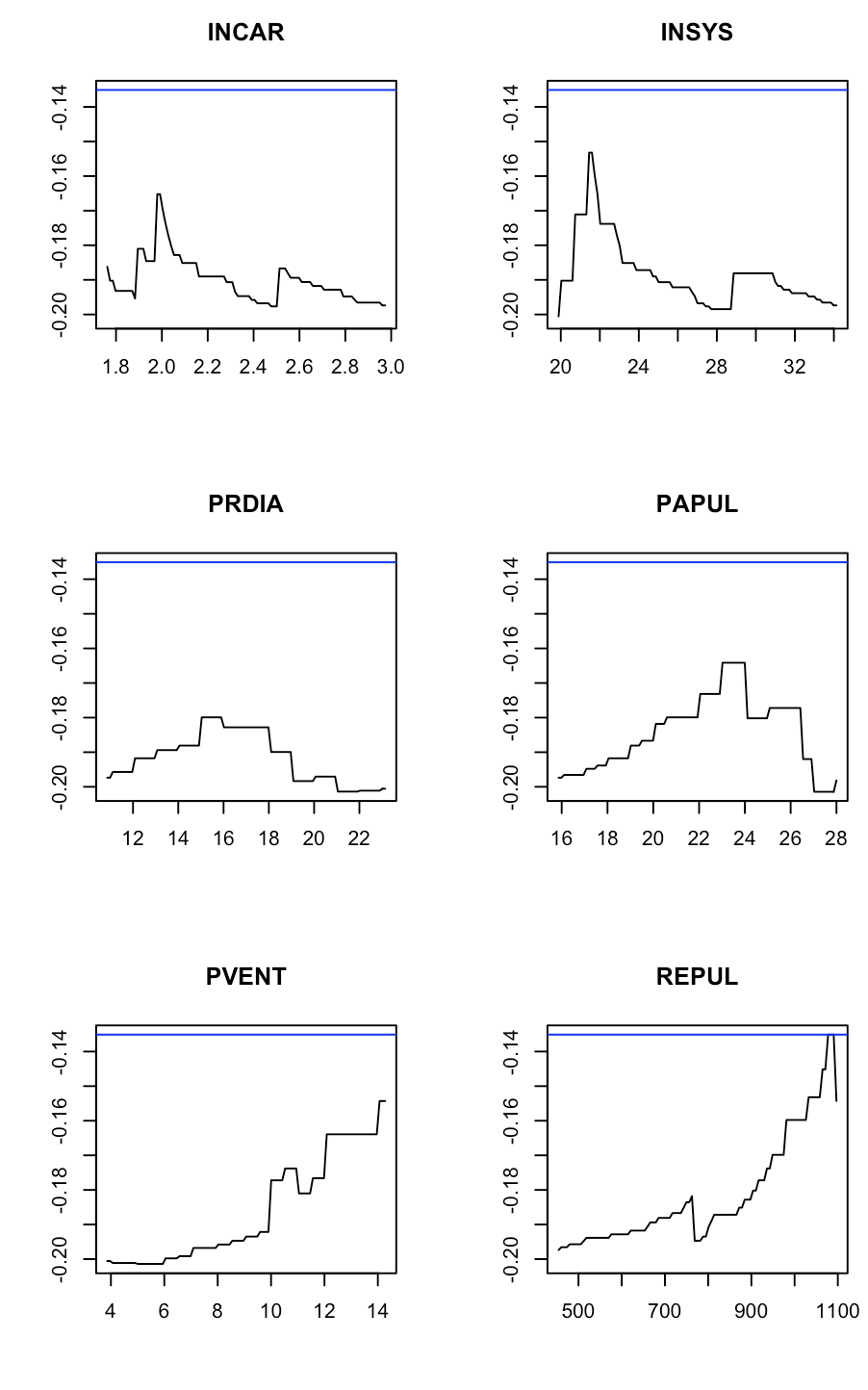
If that gain exceed 1% (the default value in R), we split in two.

Then, we do it again. Twice. First, on go on the leaf on the left, with 27 observations. And we try to see if we can split it.

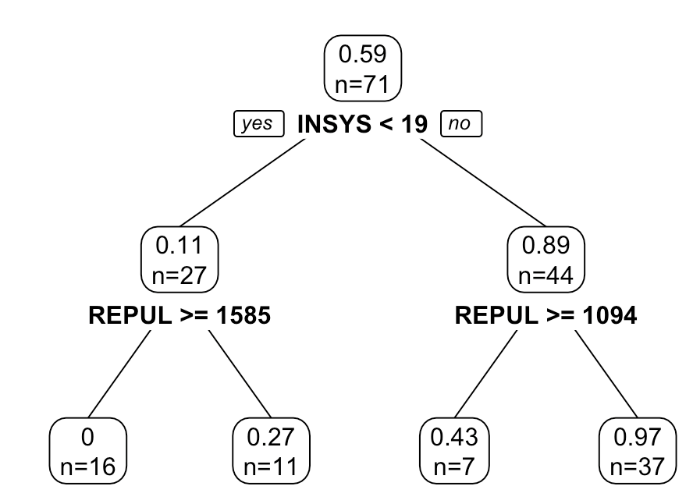
|  |
| --- |
| idx = **which**(myocarde$INSYS&lt;19)  mat\_gini = mat\_v = **matrix**(NA,7,101)  **for**(v **in** 1:7){  variable = myocarde[idx,v]  v\_seuil = **seq**(**quantile**(myocarde[idx,v],  7/**length**(myocarde[idx,v])),  **quantile**(myocarde[idx,v],1-7/**length**(  myocarde[idx,v])), **length**=101)  mat\_v[v,] = v\_seuil  **for**(i **in** 1:101){  CLASSE = variable&lt;=v\_seuil[i]  mat\_gini[v,i]=  gini(y=myocarde$PRONO[idx],classe=CLASSE)}}  **par**(mfrow=**c**(3,2))  **for**(v **in** 2:7){  **plot**(mat\_v[v,],mat\_gini[v,],type="l",  ylim=**range**(mat\_gini),xlab="",ylab="",  main=**names**(myocarde)[v])  **abline**(h=**max**(mat\_gini),**col**="blue")  } |

The graph is here the following,  
  
and observe that the best split is obtained using ‘REPUL’, with a cutoff around 1585. We check that the (relative) gain is sufficiently large, and then we go for it.  
And then, we consider the other leaf, and we run the same code

|  |
| --- |
| idx = **which**(myocarde$INSYS&gt;=19)  mat\_gini = mat\_v = **matrix**(NA,7,101)  **for**(v **in** 1:7){  variable=myocarde[idx,v]  v\_seuil=**seq**(**quantile**(myocarde[idx,v],  6/**length**(myocarde[idx,v])),  **quantile**(myocarde[idx,v],1-6/**length**(  myocarde[idx,v])), **length**=101)  mat\_v[v,]=v\_seuil  **for**(i **in** 1:101){  CLASSE=variable&lt;=v\_seuil[i]  mat\_gini[v,i]=  gini(y=myocarde$PRONO[idx],  classe=CLASSE)}}  **par**(mfrow=**c**(3,2))  **for**(v **in** 2:7){  **plot**(mat\_v[v,],mat\_gini[v,],type="l",  ylim=**range**(mat\_gini),xlab="",ylab="",  main=**names**(myocarde)[v])  **abline**(h=**max**(mat\_gini),**col**="blue")  } |

  
Here, we should split according to ‘REPUL’, and the cutoff is about 1094. Here again, we have to make sure that the split is worth it. And we cut.

Now we have four leaves. And we should run the same code, again. Actually, not on the very first one, which is homogenous. But we should do the same for the other three. If we do it, we can see that we cannot split them any further. Gains will not be sufficiently interesting.

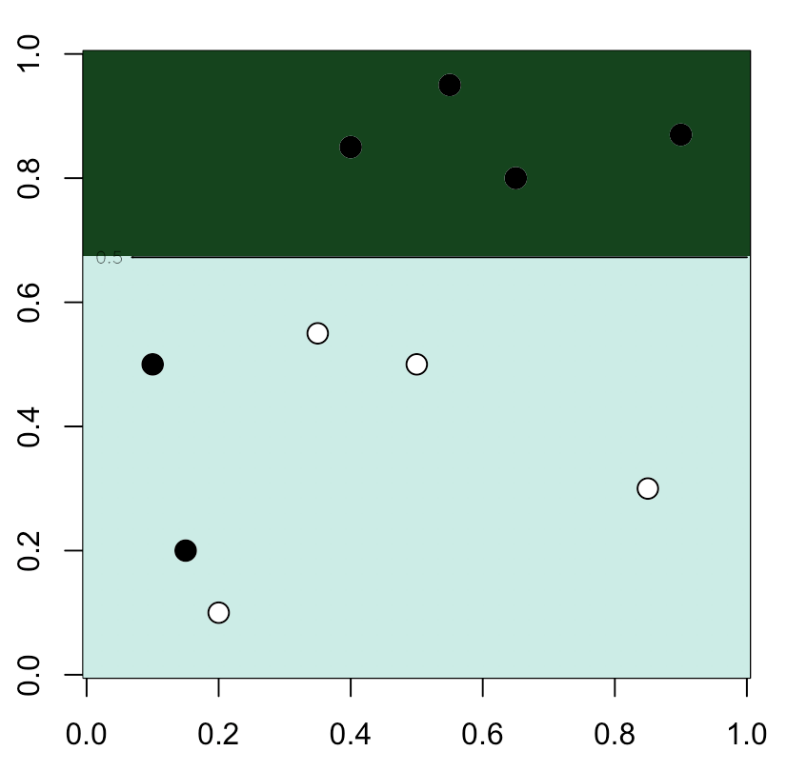
Now guess what… that’s exactly what we have obtained with our initial code  


Note that the case of categorical explanatory variables has been discussed in a [previous post](https://freakonometrics.hypotheses.org/20736), a few years ago.

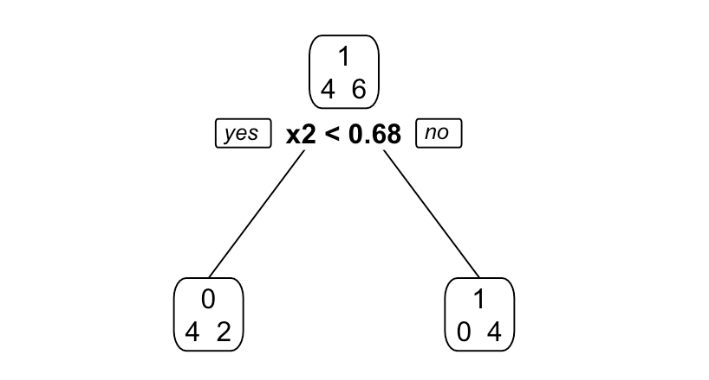
## Application on our small dataset

On our small dataset, we obtain (after changing the default values since in R, we should not have leaves with less than 10 observations… and here, the dataset is too small).

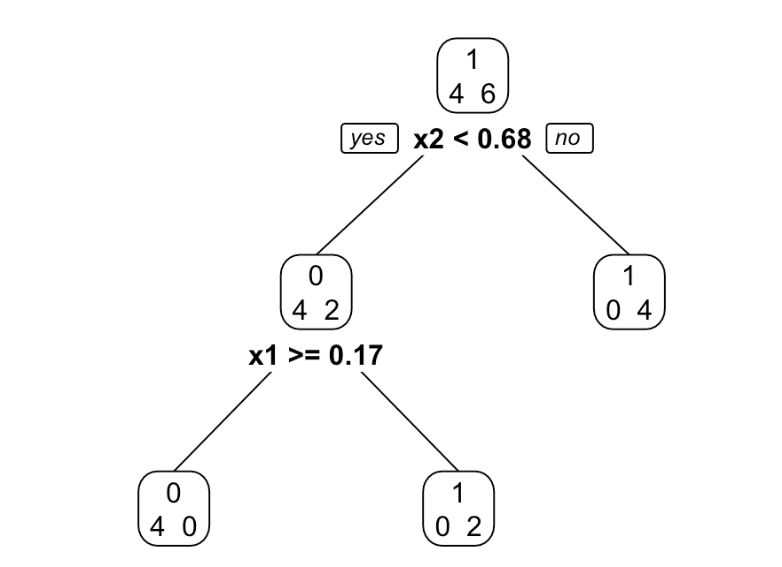
|  |
| --- |
| tree = rpart(y ~ x1+x2,**data**=**df**,  control = rpart.control(cp = 0.25,  minsplit = 7))  prp(tree,type=2,extra=1) |



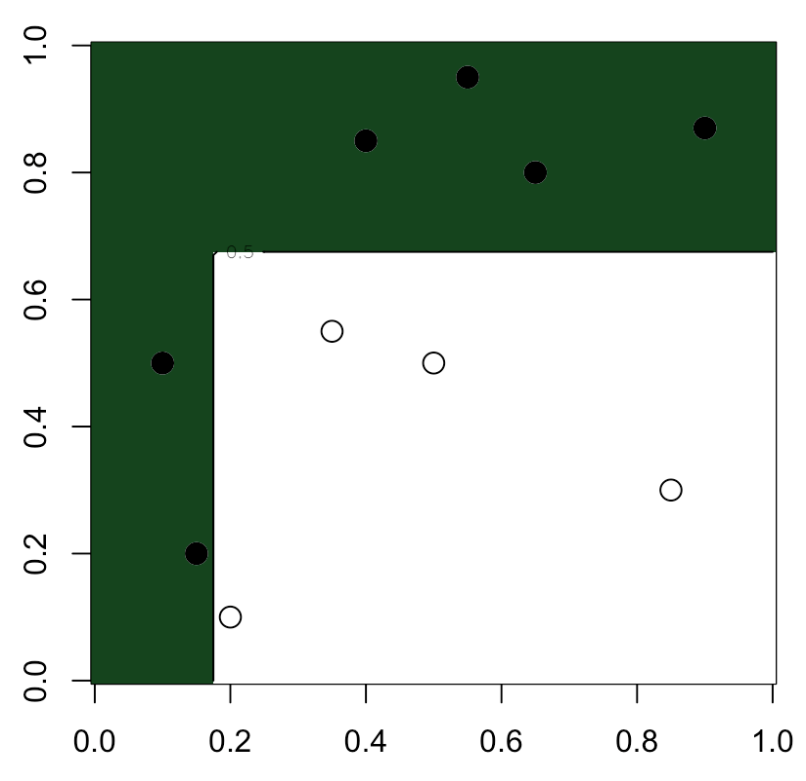
|  |
| --- |
| u = **seq**(0,1,**length**=101)  p = **function**(x,y){**predict**(tree,newdata=**data.frame**(x1=x,x2=y),type="prob")[,2]}  v = **outer**(u,u,p)  **image**(u,u,v,xlab="Variable 1",ylab="Variable 2",**col**=clr10,breaks=(0:10)/10)  **points**(**df**$x1,**df**$x2,pch=19,cex=1.5,**col**="white")  **points**(**df**$x1,**df**$x2,pch=**c**(1,19)[1+z],cex=1.5)  **contour**(u,u,v,**levels** = .5,add=TRUE) |

We have a nice and simple cut  
  
With less observations in the leaves, we can easily get a perfect model here

|  |
| --- |
| tree = rpart(y ~ x1+x2,**data**=**df**,  control = rpart.control(cp = 0.25,  minsplit = 2))  prp(tree,type=2,extra=1) |



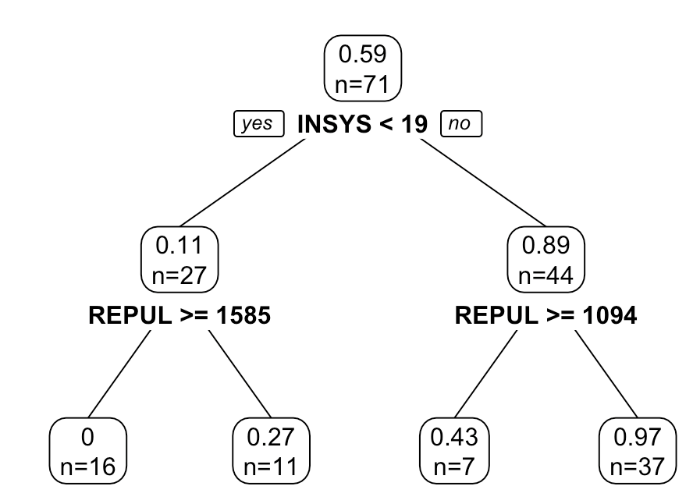
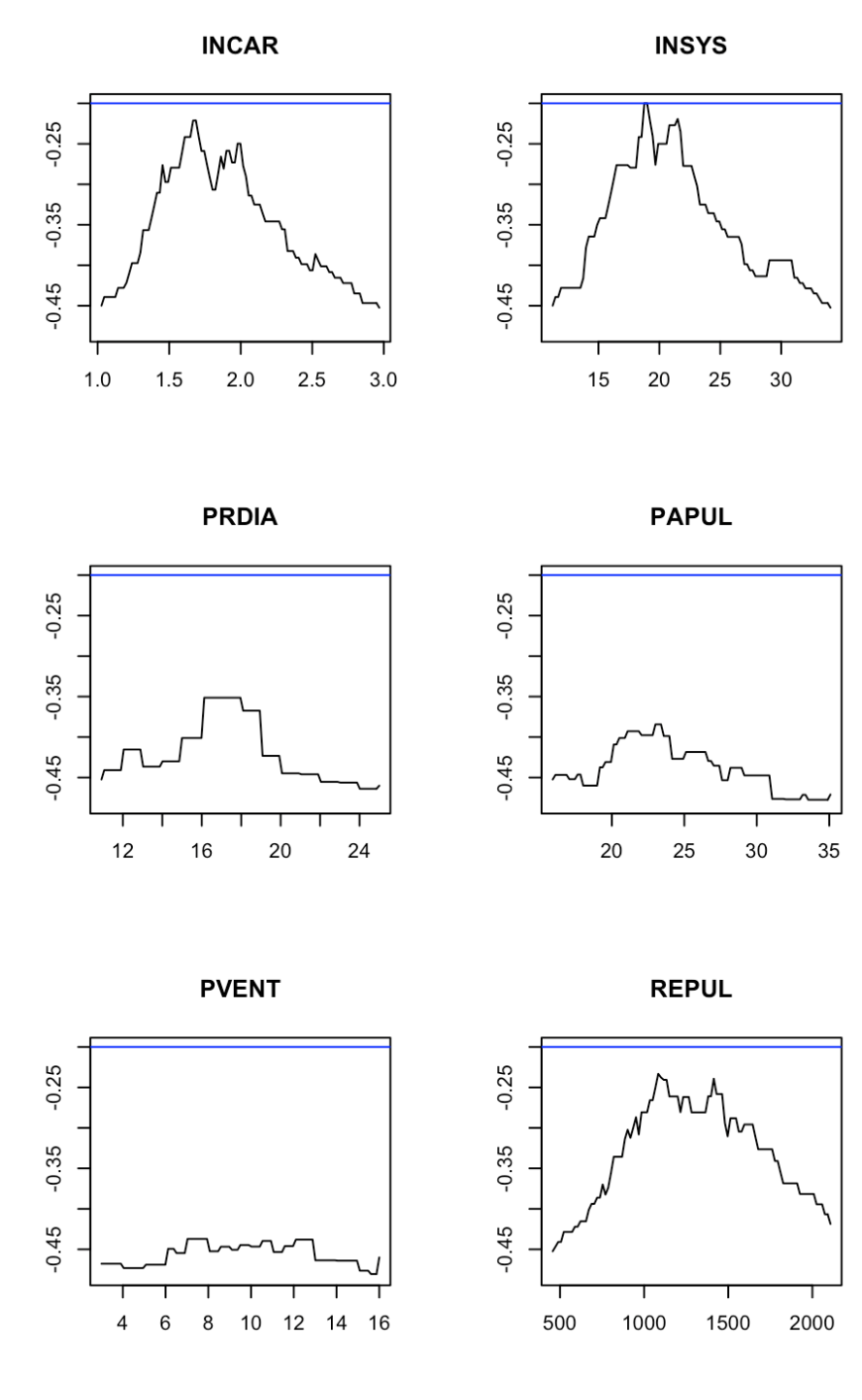
|  |
| --- |
| u = **seq**(0,1,**length**=101)  p = **function**(x,y){**predict**(tree,newdata=**data.frame**(x1=x,x2=y),type="prob")[,2]}  v = **outer**(u,u,p)  **image**(u,u,v,xlab="Variable 1",ylab="Variable 2",**col**=clr10,breaks=(0:10)/10)  **points**(**df**$x1,**df**$x2,pch=19,cex=1.5,**col**="white")  **points**(**df**$x1,**df**$x2,pch=**c**(1,19)[1+z],cex=1.5)  **contour**(u,u,v,**levels** = .5,add=TRUE) |

  
Nice, isn’t it? Now, just two little additional comments before growing some more trees…

## Pruning

Because there are two possible strategies when growing trees. Either we keep spliting, until we obtain only homogeneous leaves. Once we have a big, deep tree, we go for pruning. Or we use the stategy mentionned here : at each step, we check if the split is worth it. If not, we stop.

## Variable Importance

The heuristic idea is that if we use variable ‘INSYS’ to split, it is an important variable. And its importance is related to the gain in Gini index. If we get back to the visualization of the tree, it seems that two variables are interesting here: ‘INSYS’ and ‘REPUL’. And we should get back to previous computation to quantify how important both are.  
  
This will be used in our next post, on [random forests](https://en.wikipedia.org/wiki/Random_forest). But actually it is not the case here, with one single tree. Let us get back to the graph on the initial node.  
  
Indeed, ‘INSYS’ is important, since we decided to use it. But what about ‘INCAR’ or ‘REPUL’? They were very close… And actually, in R, those surrogate splits are considered in the computation, as briefly explained in the [vignette](https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf). Let us look more carefully at the output of the R function

|  |
| --- |
| cart = rpart(PRONO~., myocarde)  **split** = **summary**(cart)$splits |

If we look at the first part of that object, we get

|  |
| --- |
| **split**  count ncat improve index adj  INSYS 71 -1 0.58621312 18.850 0.0000000  REPUL 71 1 0.55440034 1094.500 0.0000000  INCAR 71 -1 0.54257020 1.690 0.0000000  PRDIA 71 1 0.27284114 17.000 0.0000000  PAPUL 71 1 0.20466714 23.250 0.0000000 |

So indeed, ‘INSYS’ was the most important variable, but surrogate splits can also be considered, and ‘INCAR’ and ‘REPUL’ are indeed very important. The gain was 58% (as we obtained) using ‘INSYS’ but there were gains of 55% (nothing to be ashamed of). So it would be unfair to claim that they have no importance, at all. And it is the same for the other leaves that we split,

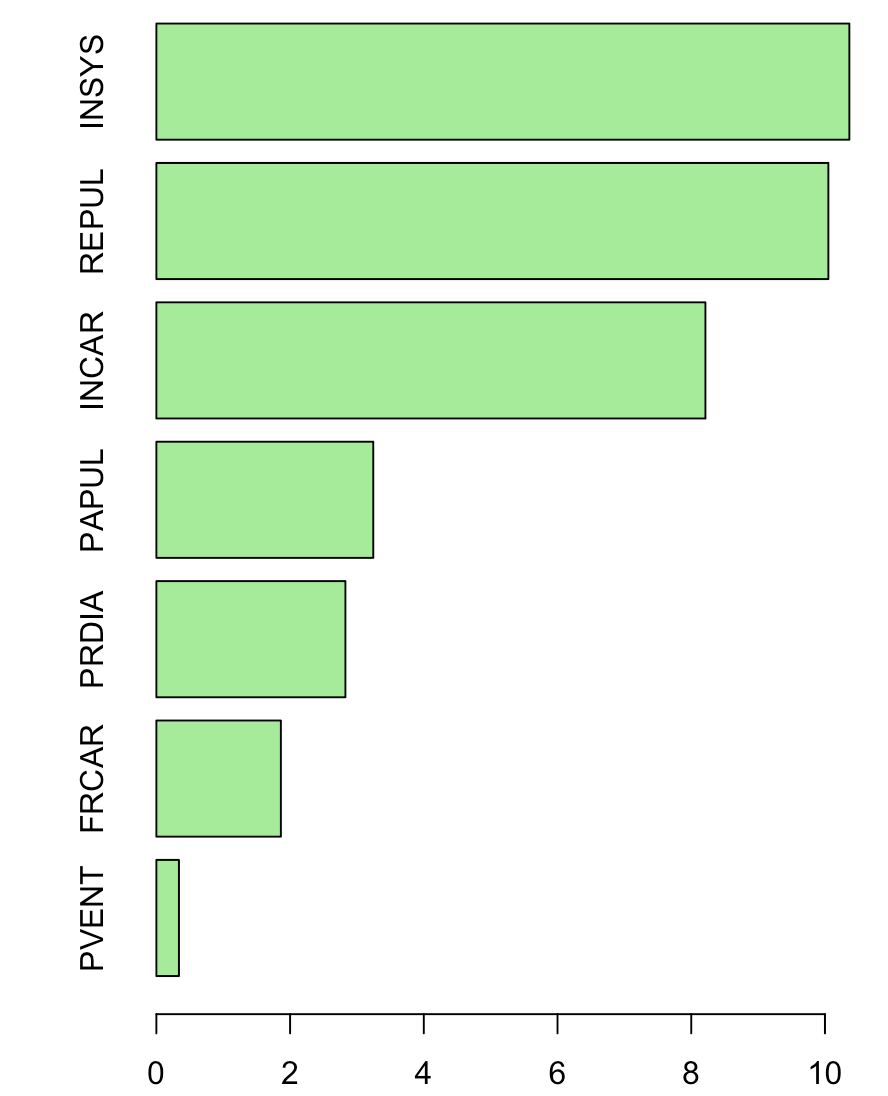
|  |
| --- |
| REPUL 27 1 0.18181818 1585.000 0.0000000  PVENT 27 -1 0.10803571 14.500 0.0000000  PRDIA 27 1 0.10803571 18.500 0.0000000  PAPUL 27 1 0.10803571 22.500 0.0000000  INCAR 27 1 0.04705882 1.195 0.0000000 |

On the left, we did use ‘REPUL’ (with 18% gain), but ‘PVENT’, ‘PRDIA’ and ‘PAPUL’ were not that bad, with (almost) 11% gain… We can obtain variable importance by summing all those values, and we have

|  |
| --- |
| cart$variable.importance  INSYS REPUL INCAR PAPUL PRDIA FRCAR PVENT  10.3649847 10.0510872 8.2121267 3.2441501 2.8276121 1.8623046 0.3373771 |

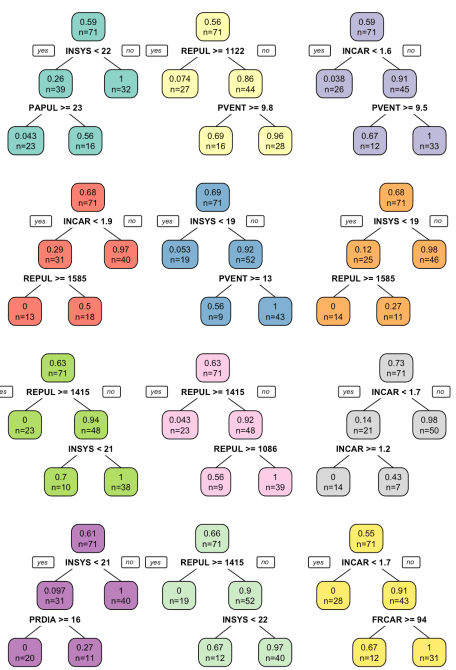
that we can visualize using

|  |
| --- |
| **barplot**(**t**(cart$variable.importance),horiz=TRUE) |



The idea of **b**agging to to generate a lot of trees

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | clr12 = **c**("#8dd3c7","#ffffb3","#bebada","#fb8072","#80b1d3","#fdb462","#b3de69","#fccde5","#d9d9d9","#bc80bd","#ccebc5","#ffed6f")  n = **nrow**(myocarde)  **par**(mfrow=**c**(4,3))  sed=**c**(1,2,4,5,6,10,11,21,22,24,27,28,30)  **for**(i **in** 1:12){  **set.seed**(sed[i])  idx = **sample**(1:n, size=n, **replace**=TRUE)  cart = rpart(PRONO~., myocarde[idx,])  prp(cart,type=2,extra=1,box.col=clr12[i])} |

  
The strategie is actually the same as before. For the **b**ootstrap part, store the tree in a list

|  |  |
| --- | --- |
| 1  2  3  4  5 | L\_tree = **list**()  **for**(s **in** 1:1000){  idx = **sample**(1:n, size=n, **replace**=TRUE)  L\_tree[[s]] = rpart(**as.factor**(PRONO)~., myocarde[idx,])  } |

and for the **agg**regation part, just take the average of predicted probabilities

|  |  |
| --- | --- |
| 1  2  3  4 | p = **function**(x){  nd=**data.frame**(FRCAR=x[1], INCAR=x[2], INSYS=x[3], PRDIA=x[4],  PAPUL=x[4], PVENT=x[5], REPUL=x[6])  **unlist**(**lapply**(1:1000,**function**(z) **predict**(L\_tree[[z]],newdata=nd,type="prob")[,2]))} |

Because with this example, we cannot visualize predictions, let us run the same code on the smaller dataset

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | L\_tree = **list**()  n = **nrow**(**df**)  **for**(s **in** 1:1000){  idx = **sample**(1:n, size=n, **replace**=TRUE)  L\_tree[[s]] = rpart(y~x1+x2, **df**[idx,],control = rpart.control(cp = 0.25,  minsplit = 2))  }  p = **function**(x){  nd=**data.frame**(x1=x[1], x2=x[2])  **unlist**(**lapply**(1:1000,**function**(z) **predict**(L\_tree[[z]],newdata=nd,type="prob")[,2]))}  vu=**seq**(0,1,**length**=101)  vv=**outer**(vu,vu,**Vectorize**(**function**(x,y) **mean**(p(**c**(x,y)))))  **image**(vu,vu,vv,xlab="Variable 1",ylab="Variable 2",**col**=clr10,breaks=(0:10)/10)  **points**(**df**$x1,**df**$x2,pch=19,cex=1.5,**col**="white")  **points**(**df**$x1,**df**$x2,pch=**c**(1,19)[1+(**df**$y=="1")],cex=1.5)  **contour**(vu,vu,vv,**levels** = .5,add=TRUE) |

