Arbres de régression, CART

Yannig Goude

Historique

Inventé par Léo Breiman en 84:

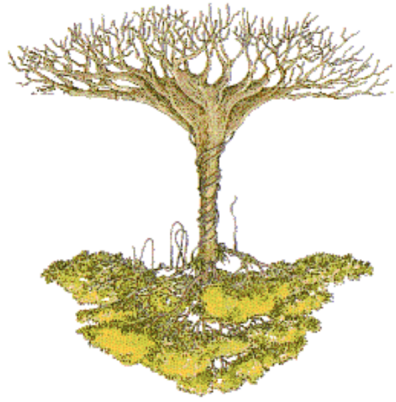
Breiman and others (1984)

Breiman's work helped to bridge the gap between statistics and computer science, particularly in the field of machine learning. His most important contributions were his work on classification and regression trees and ensembles of trees fit to bootstrap samples. Bootstrap aggregation was given the name bagging by Breiman. Another of Breiman's ensemble approaches is the random forest.

https://en.wikipedia.org/wiki/Leo_Breiman

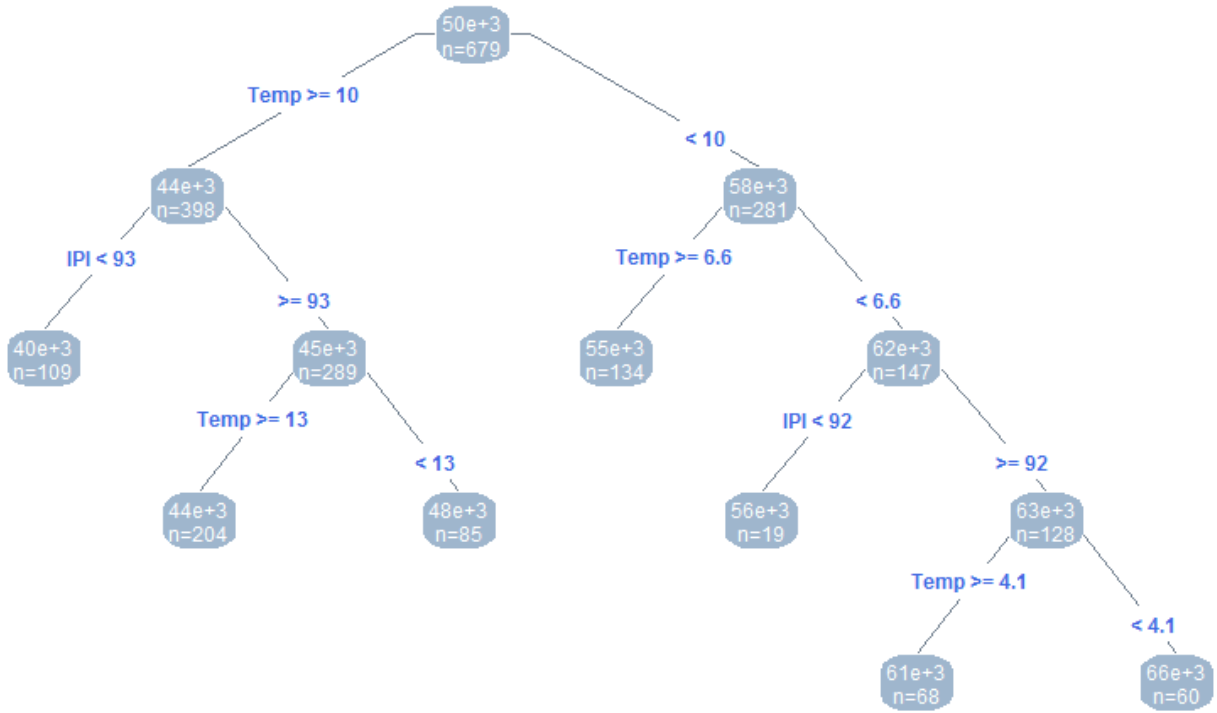
Principe

- n individus, p variables $(X_{i,k})_{1 \leq i \leq n, 1 \leq k \leq p}$
- une réponse Y_i pouvant être continue (régression) ou discrète (classification)
- CART est un algorithme de construction d'arbre binaire
- c'est un modèle **local** (contrairement à la régression linéaire par exemple)
- on effectue un partionnement récursif des données, puis on estime un modéle très simple dans chaque élément de la partition (feuilles de l'arbre)
- les variables X_j peuvent être de tous les types: continues, discrètes, discrètes et ordonnées. . . .



Exemple

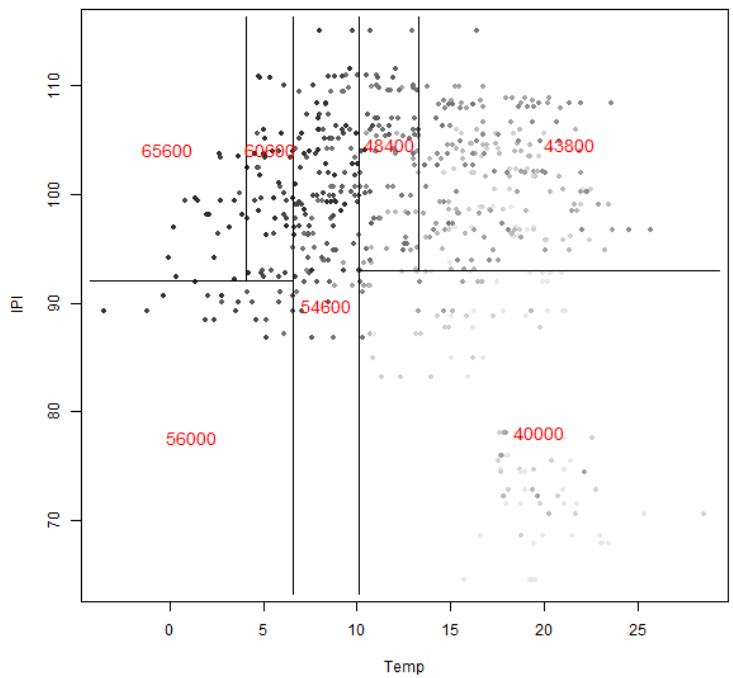
```
fit <- rpart(Load ~ NumWeek + Temp + IPI, data = data0)
```



Algorithme CART

- choisir intelligemment une variable
- couper intelligemment les données selon cette variable, la prévision $\widehat{y}_i$ est alors:
  - la moyenne des observations dans la feuille correspondant à x_i
  - la classe majoritaire dans la feuille
- recommencer sur les sous-arbres obtenus

Quand s'arrêter? Comment choisir la variable et les seuils de coupure?



Arbre de régression

Objectif: découper l'espace des variables explicatives en régions $R_1, ..., R_J$ (les feuilles de l'arbre) qui minimisent:

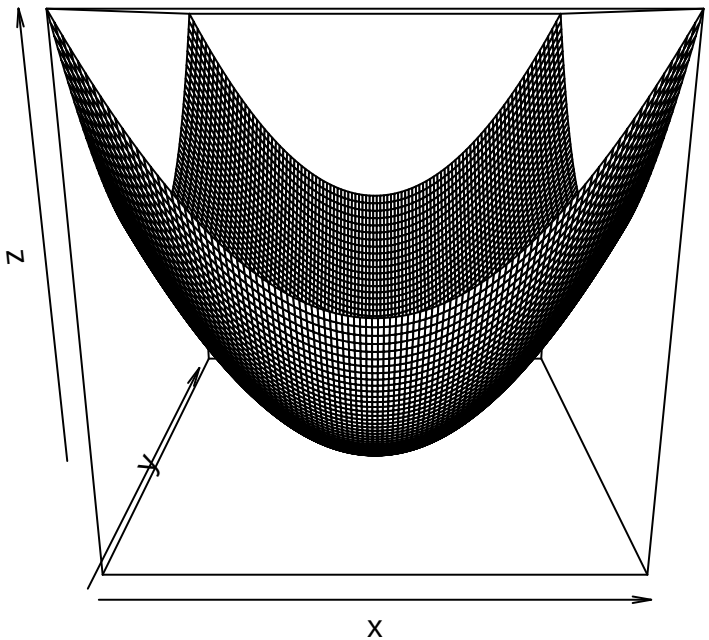
$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \widehat{y}_j)^2$$

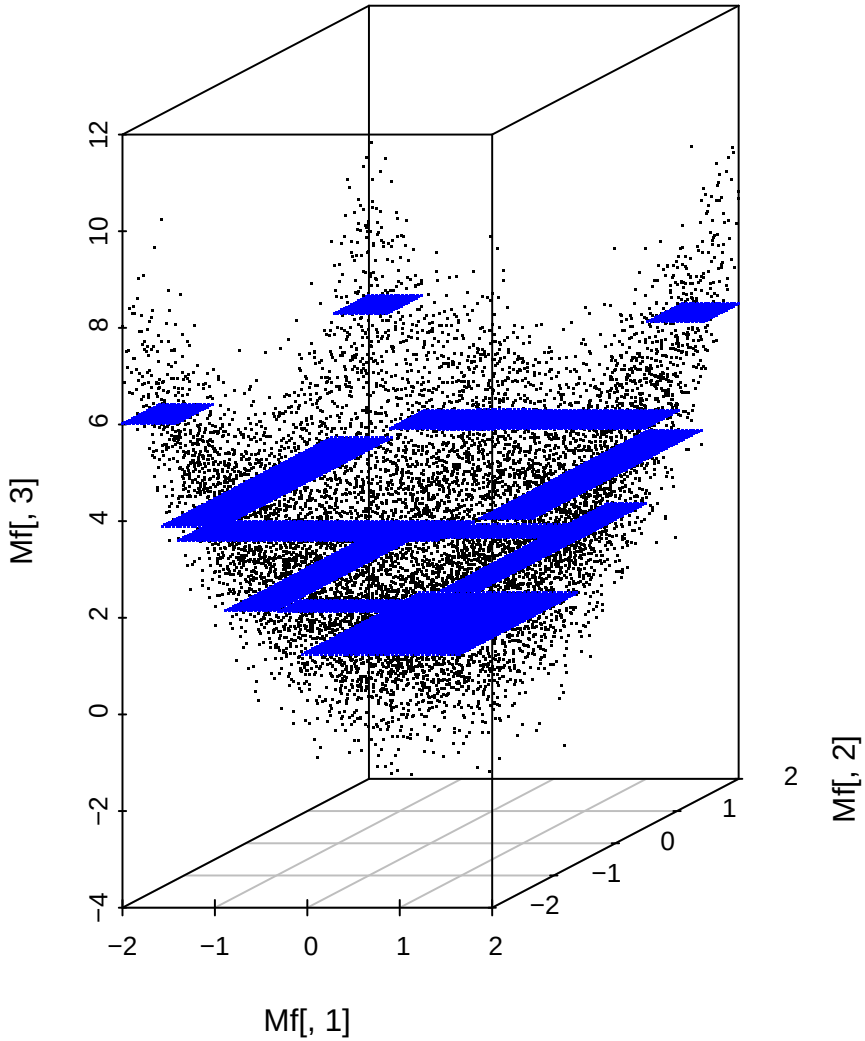
avec, dans le cas de l'algorithme CART de base $\widehat{y}_j = \frac{1}{n_j} \sum_{i \in R_j} y_i$, n_j le nombre d'obervations dans la feuille R_j. En effet, dans le CART de base on suppose que $\widehat{y} = a$ ou a est une constante à déterminer. Nous verrons qu'il existe des variantes de type $\widehat{y} = X\beta$.

Problème insoluble en pratique d'ou l'approche récursive (et donc mathématiquement sous-optimal) de CART

- soit les hyperplans $R_-(j, s) = \{X_j < s\}$ et $R_+(j, s) = \{X_j \geq s\}$
- à chaque étape CART choisit la variable j et le seuil s minimisant la **variance intra-groupe**

$$\frac{1}{n} \sum_{i \in R_-(j,s)} (y_i - \widehat{y}_{R_-})^2 + \frac{1}{n} \sum_{i \in R_+(j,s)} (y_i - \widehat{y}_{R_+})^2$$


z
y
x


Mf[, 3]
12
10
8
6
4
2
0
−2
−4
−2
−1
0
1
2
Mf[, 1]
2
1
0
−1
−2
Mf[, 2]

Arbre de régression

Minimiser RSS revient à minimiser:

$$RSS = \frac{1}{n} \sum_{j=1}^{J} n_j V_j$$

ou $V_j = \frac{1}{n_j} \sum_{i \in R_j} (y_i - \bar{y}_j)^2$ estl a variance d'une feuille de l'arbre, n_j le nombre d'obervation dans chaque feuilles de l'arbre.

Les critéres d'arrêts de l'algorithme récursif sont:

- un nombre d'observations q minimal dans les feuilles
- une réduction du critére d'erreur (variance inter ici) d'un seuil minimal δ

Arbre de régression

Une autre alternative: **prunning**

- ne pas s'arréter et étendre l'arbre jusqu'au bout ($q = 1$, $\delta = 0$)
- élagage de l'arbre: sur chaque couple de feuilles, regrouper les feuilles si cela fait baisser l'erreur de prévision (VC, échantillon test, minimisation d'un critère pénalisé par la taille de l'arbre...)
  - *Intuition: une variable non-sélectionnée à un étape de l'algo. récursif peut contenir de l'information pour la prévision (par exemple en interaction avec une autre), l'approche prunning a une chance de le détecter.*
  - *l'arbre maximal a une très grande variance mais un biais faible. Un arbre peu profond a une très petite variance mais un biais élevé.*
  - *de nombreuses variantes existent en fonction du type d'estimation de l'erreur de prévision, des choix de q et δ...*

Algorithme d'élagage

Notations:

- on note t un noeud de l'arbre, T_t le sous arbre dont la racine est t
- soit T un arbre, $R(T)$ l'erreur quadratique empirique associée à T, $\widetilde{T}$ l'ensemble des feuilles de l'arbre et $|\widetilde{T}|$ le nombre de feuilles de l'arbre (complexité) $R(T) = \sum_{t \in \widetilde{T}} R(t)$
- pour deux arbres T et T', on note $T \leq T'$ si T est un sous-arbre de T', ils ont la même racine et les noeuds et branches de T' sont des noeuds et branches de T
- T_{max} est l'arbre maximal, obtenu en itérant l'algorithme CART jusqu'au bout
- pour $\lambda > 0$ on note $R_\lambda(T) = R(T) + \lambda|\widetilde{T}|$

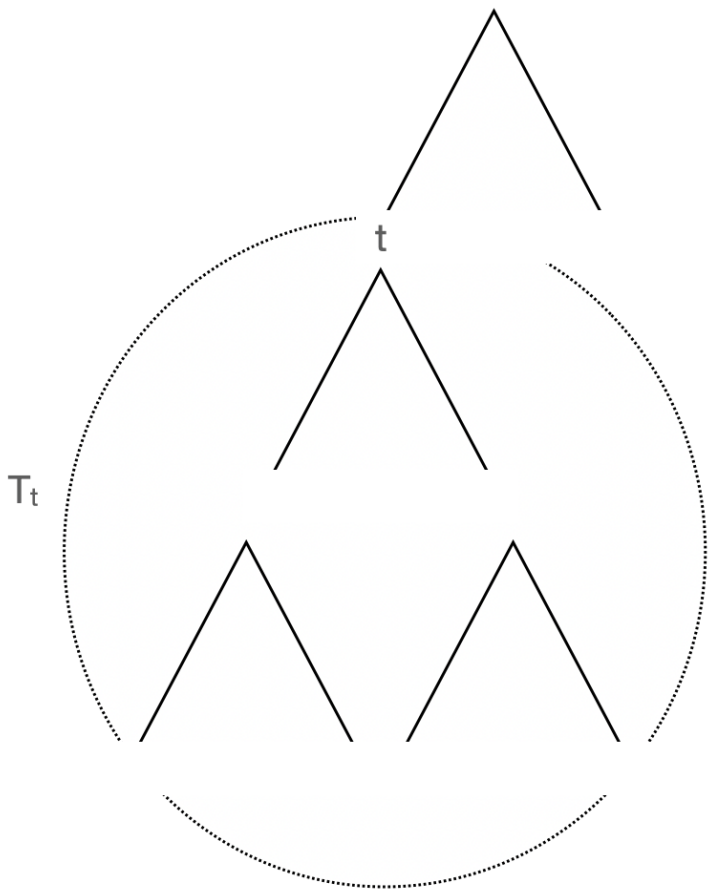
L'idée qui sous-tend l'élagage est la même que pour l'ajustement du paramètre λ en ridge. L'arbre T_{max} possède un faible biais mais obtiendra une grande erreur de généralisation du fait de sa forte variance (un léger changement des données induit un arbre T_{max} très différent). Pour diminuer cette variance il est nécessaire d'ajouter une pénalisation au critère RSS, pénalisation croissante en la complexité de l'arbre (le nombre de feuille). La méthode d'élagage est notamment, pour chaque valeur $\lambda > 0$ de construire $\widehat{T}_\lambda = \text{argmin}_T R_\lambda(T)$

Algorithme d'élagage

Dans Breiman and others (1984) un théorème assure l'existance de $\widehat{T}_\lambda$ et le fait que $\widehat{T}_\lambda \leq T_{max}$. Il faut ensuite savoir comment calculer $\widehat{T}_\lambda$ et choisir λ.

Breiman propose un algorithme permettant, à partir de l'arbre maximal, de générer des arbres emboités de complexité décroissante.

Une propriété des arbres va jouer un rôle important:



$$\forall t, R(T_t) \leq R(t)$$

Algorithme d'élagage

Ainsi:

$$\exists \lambda > 0, \forall t, R_\lambda(T_t) \leq R_\lambda(t) = R(t) + \lambda$$

si il y a egalité, il faut couper la branche car la complexité de l'arbre est plus grande pour un gain de performance nul.

Ainsi, si on augmente λ progressivement, il arrive un moment ou

$$R_\lambda(T_t) = R_\lambda(t)$$

et on peut élaguer l'arbre de cette branche.

En pratique, il n'est pas nécessaire de faire varier λ continuement (ça serait d'ailleurs très couteux), et on choisit à chaque étape la plus petite valeur de λ pour laquelle il existe un noeud de l'arbre tel que $R_\lambda(T_t) = R_\lambda(t)$:

$$\lambda_0 = \min_t \frac{R(t) - R(T_t)}{|\widetilde{T}_t| - 1}$$

Algorithme d'élagage

L'algorithme est donc:

- on commence par choisir $\lambda = 0$ et élaguer l'arbre T_{max}, en otant les branches $t_1, \ldots, t_q$ telles que $R(T_t) = R(t)$. L'arbre ainsi obtenu est noté $T(0) = T_{max} - T_{t_1} - ... - T_{t_q}$.
- à chaque étape k de l'algorithme, on calcule tant que l'arbre est élagable:

$$\lambda_k = \min_{t \in T(k-1)} \frac{R(t) - R(T_t)}{|\widetilde{T}_t| - 1}$$

et les branches $T_{t_1}, ..., T_{t_{q_k}}$ telles que $R(T_{t_j}) = R(t_j)$. L'arbre élagué obtenu est ensuite:

$$T(k) = T(k-1) - T_{t_1} - ... - T_{t_{q_k}}$$

Algorithme d'élagage

Un théorème proposé par Breiman assure que ce choix de λ ainsi défini répond bien à l'objectif de minimisation de $R_\lambda(T)$:

$$\forall \lambda_{k-1} \leq \lambda < \lambda_k, \qquad \widehat{T}_\lambda = \widehat{T}_{\lambda_{k-1}}$$

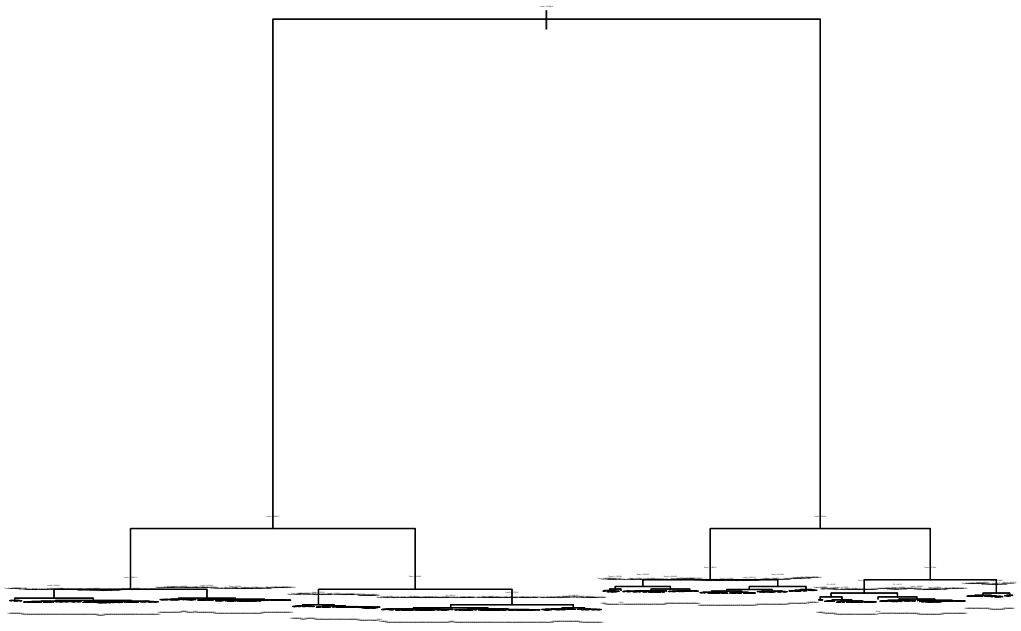
Les arbres ainsi générés forment une famille d'arbre décroissante [1] dont chaque élément ecorrespond à λ_k.

[1] pour deux arbres T et T', on note $T \leq T'$ si T est un sous-arbre de T', ils ont la même racine et les noeuds et branches de T' sont des noeuds et branches de T

Exemple

Construction d'un arbre maximal (package `tree`)

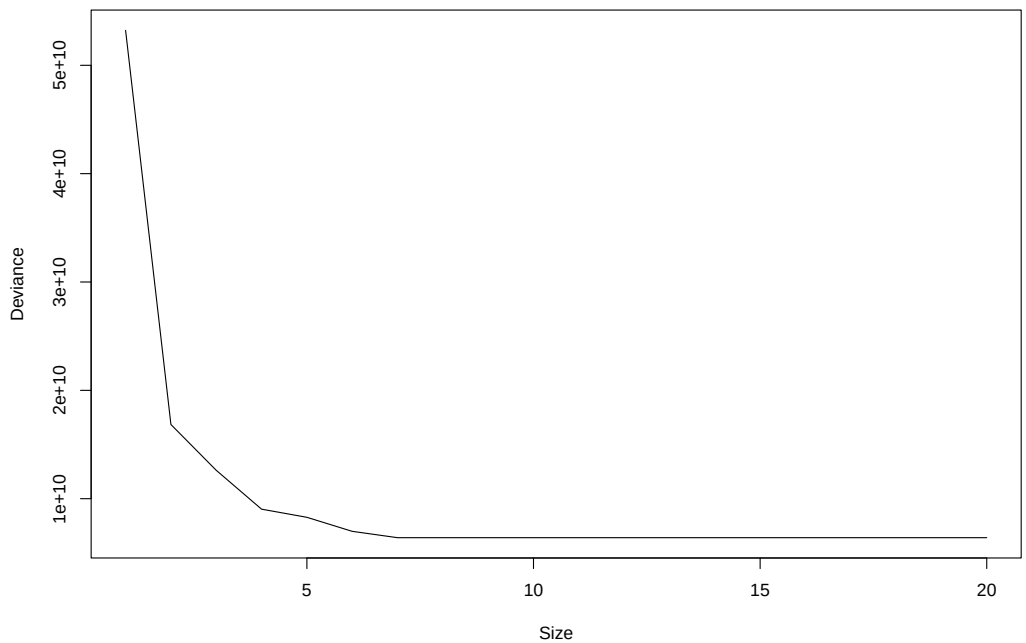
```
n<-nrow(data0)
treefit<-tree(Load ~Temp+IPI+Load1+NumWeek+Temp1 ,data = data0,
              control =  tree.control(nobs=n,minsize=1, mindev=0))
plot(treefit)
text(treefit,cex=.1)
```



Exemple

Prunning par validation croisée K-fold:

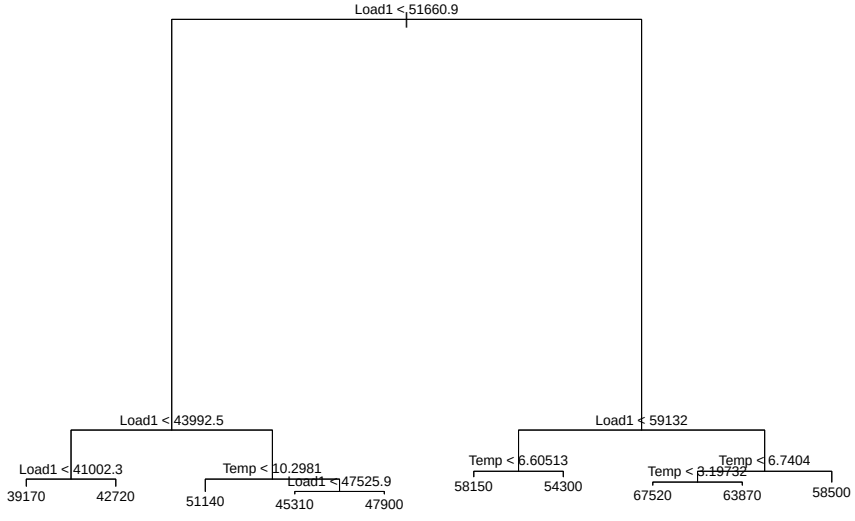
```
treefit.cv <- cv.tree(treefit, FUN = prune.tree, K=10)
plot(tail(treefit.cv$size,20), tail(treefit.cv$dev,20),
     type='l', xlab="Size", ylab="Deviance")
```



Exemple

Arbre élagué par VC:

```
size.opt<-10
treefit.prune<-prune.tree(treefit,best=size.opt)
plot(treefit.prune)
text(treefit.prune,cex=.8)
```



Remarques sur les arbres de régressions

- implémentation aisée et efficace, pouvant passer é l'échelle
- pas d'hypothèses sur les distributions des variables
- adapté au cas ou le nombre de variable est élevé, selection des variables "automatique", prise en compte des interactions
- interprétation/compréhension des interactions simple: il suffit de parcourir l'arbre!
- souvent très efficace pour débroussailler un pb, trouver les bonnes transformations de variables
- robuste aux outliers, travaille sur les rangs
- peut modéliser des phénoménes discontinue, non-fonctionnels
- approxime raisonnablement des phénomènes continus malgré tout (fonctions constantes par morceau), attention toutefois à l'extrapolation
- analogies avec les plus proches voisins
  - les plus proches voisins prévois de la méme maniéres des données prochent en X
  - les arbres intégrent en plus une proximité en Y, on peut les voir comme des plus proches voisins adaptatifs

Implémentation en R

- CART: package `rpart` , `tree`
- ctree (conditional inference trees): package `party`
- représentation graphique améliorée: package `rpart.plot` (fonction `prp` notamment), `plotmo`

Bibliographie

- L.Breiman and others (1984), Classification and Regression Trees, new edition, New York: Chapman & Hall, 1984, p. 358. ISBN: 0-412-04841-8.
- Tibshirani, R., & Friedman, J. (2001). The elements of statistical learning: data mining, inference, and prediction. Heidelberg: Springer.