LÉONARD DE VINCI
PÔLE UNIVERSITAIRE

Arbre de régression et de classification

(CART: Classification and Regression Tree)

Nicolas Turenne
nicolas.turenne@jouy.inra.fr

INRA

2005

- Méthode développée par
  Brieman, Friedman, Olshen, Stone (1984)

- 2 Idées clés :
  - Partitionnement récursif d'un ensemble de variables indépendantes
  - Elagage (pruning) en utilisant des données de validation

Partitionnement récursif

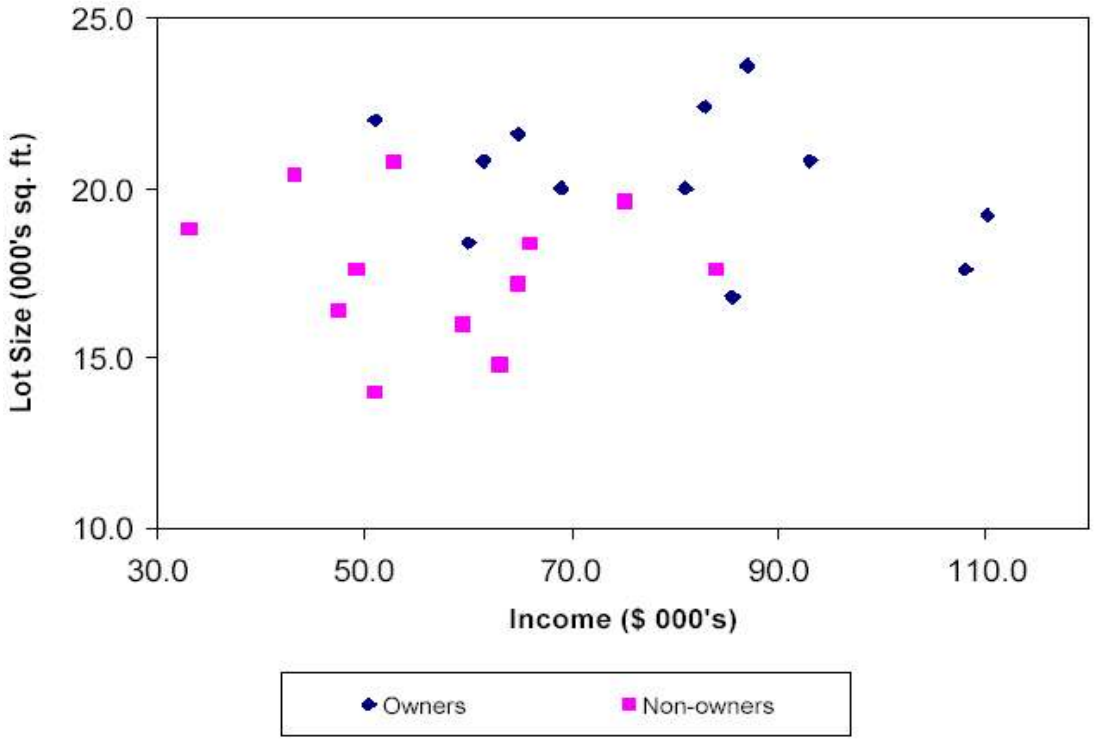
- Considérons un ensemble de variables catégorielles $x_1,...x_p$. Le partitionnement récursif divise l'espace des p variables en rectangles qui ne se chevauchent pas.

- La division est accomplie récursivement. Par exemple soit la variable x_i et une valeur s_i de cette variable. On trouve que le partitionnement où $x_i < s_i$ et $x_i > s_i$ sépare bien les données en deux ensembles disjoints. Ensuite une des deux parties est à son tour divisée par une valeur de x_i ou par la valeur d'une autre variable. On aboutit à 3 rectangles et ainsi de suite.

- L'idée est de créer n rectangles de telle sorte que l'ensemble de données contenu dans un rectangle soit homogène.

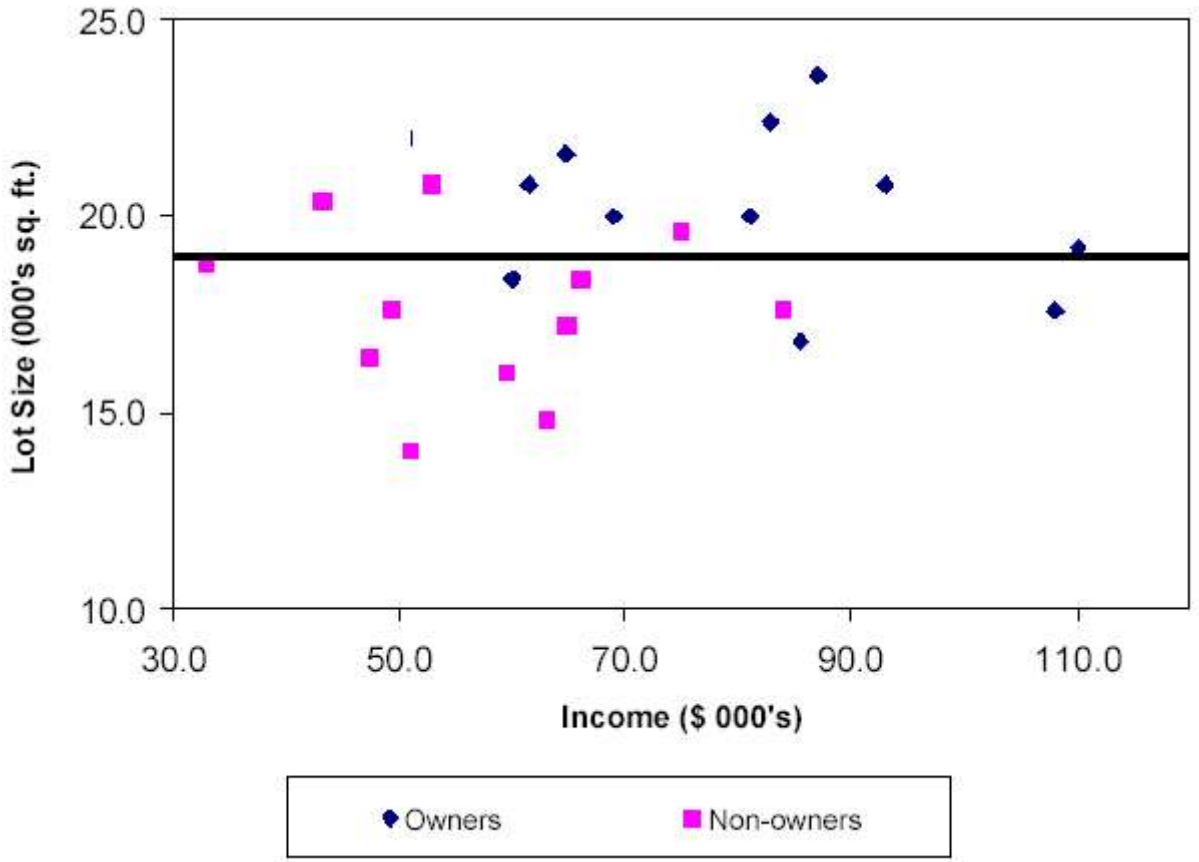
Exemple 1

- Un fabriquant de véhicules aimerait trouver un moyen de classer les familles dans une ville qui sont à même d'acheter un véhicule et celles qui ne sont pas prêtent à en acheter. Un échantillon de 12 propriétaires et 12 non-propriétaires est choisi. Les deux variables indépendantes sont : x1 (*income* ou revenus) et x2 (*lot size* ou surface en pieds par m2)



Observation	Income ($ 000's)	Lot Size (000's sq. ft.)	Owners=1, Non-owners=2
1	60	18.4	1
2	85.5	16.8	1
3	64.8	21.6	1
4	61.5	20.8	1
5	87	23.6	1
6	110.1	19.2	1
7	108	17.6	1
8	82.8	22.4	1
9	69	20	1
10	93	20.8	1
11	51	22	1
12	81	20	1
13	75	19.6	2
14	52.8	20.8	2
15	64.8	17.2	2
16	43.2	20.4	2
17	84	17.6	2
18	49.2	17.6	2
19	59.4	16	2
20	66	18.4	2
21	47.4	16.4	2
22	33	18.8	2
23	51	14	2
24	63	14.8	2

Si on applique l'algorithme CART sur ces données, il choisira x2 comme premier choix de division avec la valeur de division 19. L'espace (x1,x2) est maintenant divisé en 2 rectangles, l'un pour la valeur de la variable Lot Size, x2 <= 19 et l'autre avec x2> 19.



Noter comment la division en 2 rectangles a créé deux rectangles qui sont plus homogènes que le rectangle avant la division (*split*). Le rectangle supérieur contient des points qui sont davantage des propriétaires (9 propriétaires et 3 non-propriétaires) tandis que le rectangle inférieur contient davantage de non-propriétaires (9 non-propriétaires et 3 propriétaires).

Comment CART a décidé cette division particulière ?

Il a examiné chaque variable et toutes les valeurs possibles de division pour chaque variable de façon à trouver la meilleure division.

Quelles sont les meilleures valeurs de division pour une variable ?

Elles sont simplement les points-mileux entre des pairs de valeurs consécutives pour la variable. Les points de division possible pour x_1 sont {38.1, 45.3, 50.1, … , 109.5} et ceux pour x_2 sont {14.4, 15.4, 16.2, … , 23}. Ces points de division sont rangés d'après la façon dont ils réduisent l'«impureté» (hétérogénéité de composition). La réduction de l'impureté est définie par l'impureté du rectangle avant la division moins la somme des impuretés des deux rectangles qui résultent de la division.

Comment calculer l'impureté ?

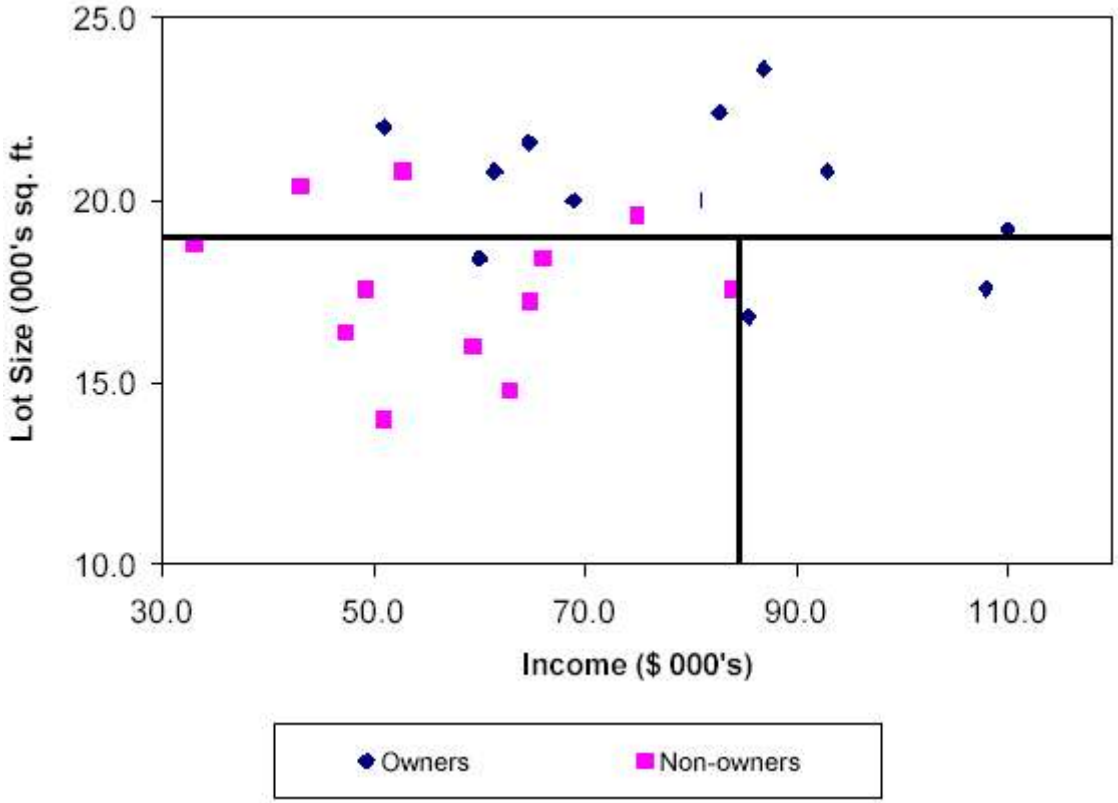
Il y a un nombre de façon important de mesurer l'impureté. On va décrire la mesure la plus populaire de mesurer cette impureté: l'indice de Gini. Si on dénote les classes par k, k=1, 2, … , C, où C est le nombre total de classes pour la variable y, l'indice d'impureté de Gini pour le rectangle A est défini par

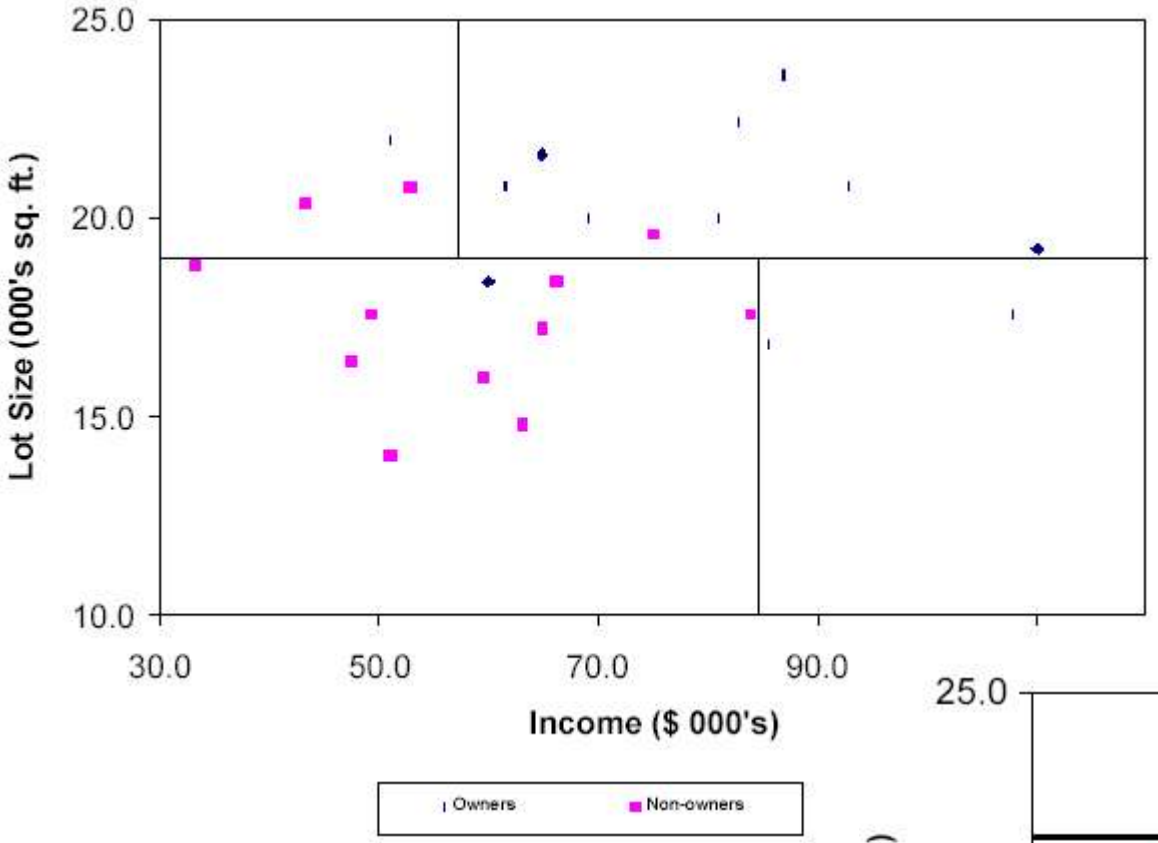
$$I(A) = 1 - \sum_{k=1}^{C} p_k^2$$

où p_k est la fraction d'observations dans le rectangle A qui appartiennent à la classe k. On note que I(A) = 0 si toutes les observations appartiennent à une classe unique et I(A) est maximisé quand toutes les classes apparaissent en proportion égales dans le rectangle A. Sa valeur maximale est (C-1)/C

La division qui suit est sur la variable Income (revenus), x_1 à la valeur 84.75.

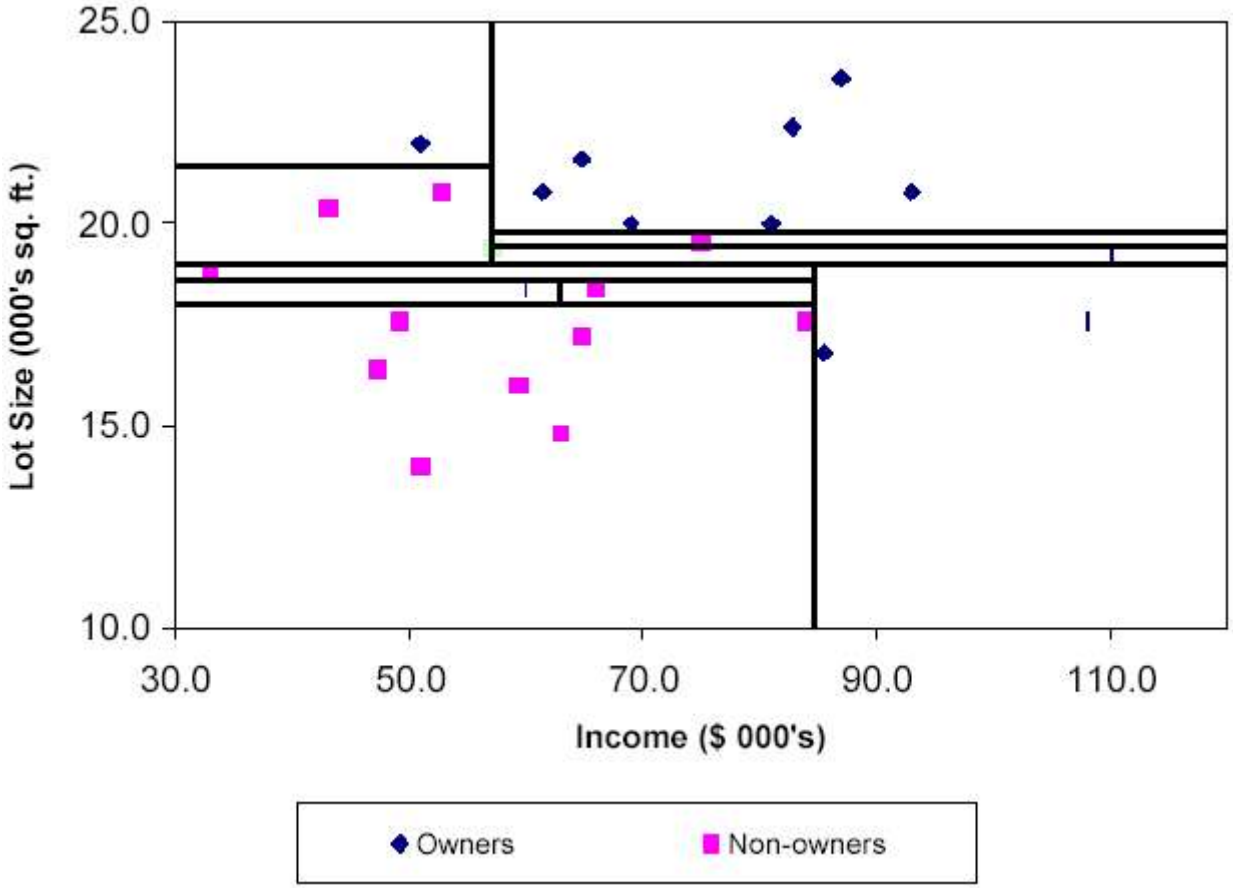
La Figure montre qu'une fois de plus la procédure CART a astucieusement choisi la division en rectangle pour améliorer la pureté des rectangles résultant Le rectangle inférieur gauche qui contient les points de données avec x_1 <84.75 et x_2 <= 19 a tous ses points sauf un qui sont non-propriétaires; tandis que le rectangle qui contient les points de données avec x1 > 84.75 and x2<= 19 sont exclusivement des propriétaires.





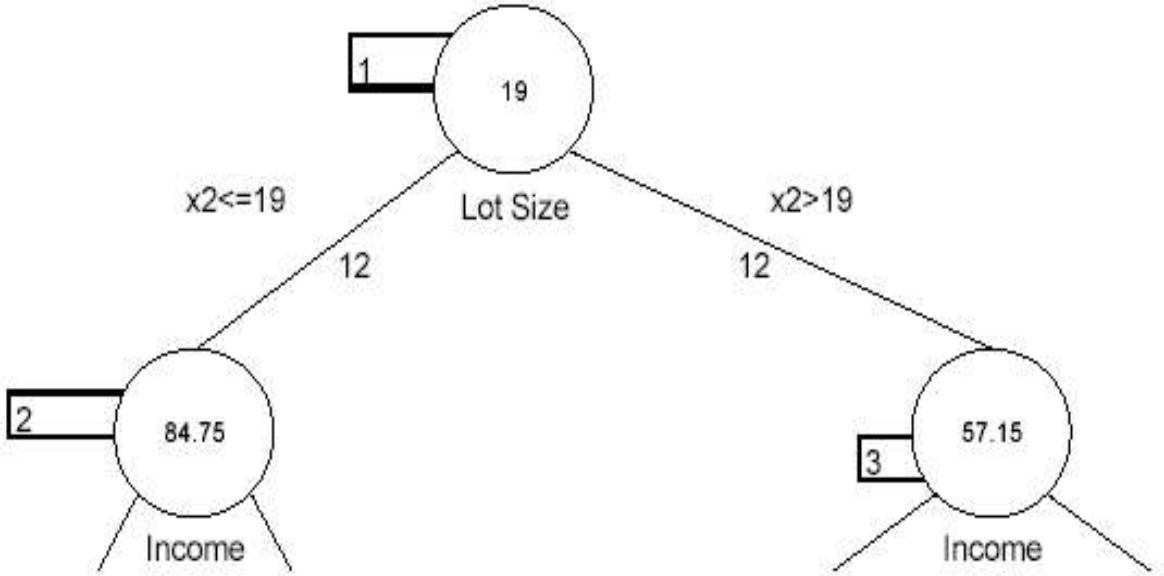
On peut voir comment le partitionnement récursif permet d'affiner l'ensemble des rectangles pour devenir plus purs de la manière dont procède l'algorihtme.

A droite on voit l'étape finale du partitionnement récursif.

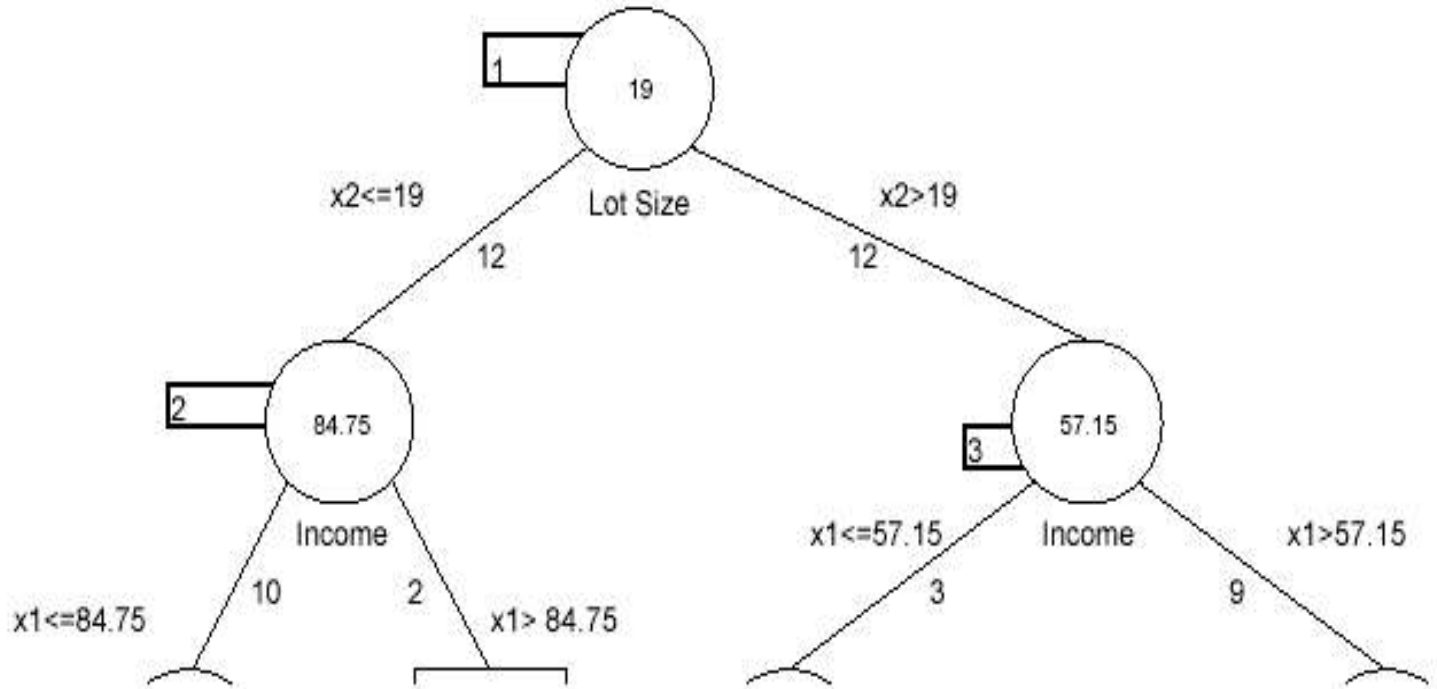


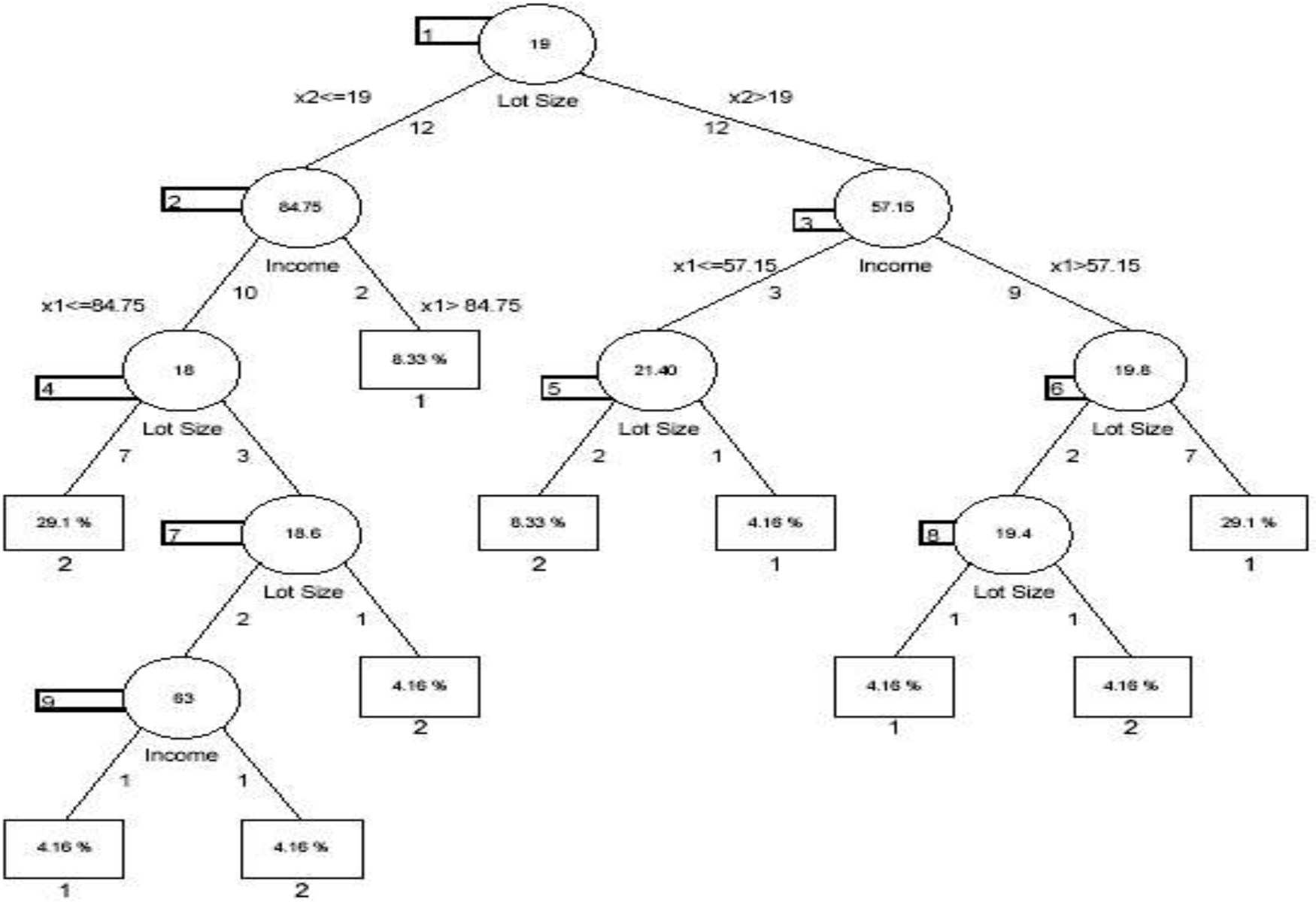
On note que chaque rectangle est pur – il contient les points de données de juste une des deux classes.

La raison pour laquelle la méthode est appelée algorithme d'arbre de classification est que chaque division peut être figurée comme la division d'un noeud en deux noeuds successeurs. La première division est montrée comme un branchement du noeud racine de l'arbre.



Voici l"arbre représentant les trois premières divisions





Voici l'arbre complet. On a représenté par des cercles les noeuds qui ont des successeurs. Les nombres à l'intérieur des cercles sont les valeurs de division et le nom de la variable choisie pour la division à ce noeud est écrit sous le noeud. Les nombres sur la fourche gauche à un noeud de décision ont des valeurs inférieures ou égales à la valeur de division tandis que le nombre de la fourche droite montre un nombre qui a une valeur plus grande.

Ils sont appelés noeuds de décision parce que si nous devions utiliser l'arbre pour classifier une nouvelle observation pour laquelle on connait seulement les valeurs des variables indépendantes, on « lacherait » l'observation en bas de l'arbre de telle sorte que la branche appropriée est prise jusqu'à ce qu'on obtienne un noeud qui n'a pas de successeurs. De tels noeuds terminaux sont appelés feuilles de l'arbre. Chaque noeud feuille correspond à des rectangles finaux dans lesquels l'espace x est partitionné et est dépeinte comme un noeud de forme rectangle.

Quand l'observation a été lachée vers tous les chemins à une feuille pouvant prédire une classe en considérant simplement un 'vote' de toutes les données d'entrainement (training), la classe avec le plus grand vote est la classe que nous aurions prédite pour un nouvelle observation.

Le nombre sous le noeud feuille est la classe avec le plus de votes dans le rectangle. La valeur % dans un noeud feuille montre le pourcentage du nombre total d'observations d'entrainement qui appartiennent à ce noeud. Il est utile de noter que le type d'arbres développés par CART (appelés arbres binaires) ont la propriété que le nombre de noeuds feuilles est exactement un de plus que le nombre de noeuds de décision.

Elagage

La seconde idée clé dans la procédure CART, est celle qui utilise des données de validation pour élaguer a posteriori l'arbre qui a grandi à partir des données d'entrainement utilisant des données de validation indépendantes. Cela a été une vraie innovation. Auparavant, les méthodes ont été développées de telle sorte qu'elles étaient basées sur l'idée d'un partitionnement récurcif mais elles ont utilisé des règles pour éviter un grossissement excessif de l'arbre et le sur-apprentissage (over-fitting) des données d'entrainement.

Par exemple, CHAID (Chi-Squared Automatic Interaction Detection) est une méthode de partitionnement récursif qui devance CART de plusieurs années et est largement utilisé dans les applications de marketing de base de données à ce jour. Il utilise un test statistique bien connu (test de chi2 pour l'indépendance) pour vérifier si la division à un noeud améliore la pureté d'une quantité statistiquement significative. Si le test ne montre pas une amélioration significative la division n'est pas réalisée. Par contraste, CART utilise une validation pour élaguer a posteriori l'arbre qui a été sur-développé délibérément en utilisant les données d'entrainement.

L'idée derrière l'élagage est de reconnaître qu'un arbre très gros est à même de provoquer un sur-apprentissage sur les données d'entrainement. Dans notre exemple, les dernières division résultant en rectangles avec très peu de points (en ce sens quatre rectangles dans l'arbre complet ont juste un point). On peut voir intuitivement que ces dernières divisions sont à même de simplement capturer du bruit dans l'ensemble d'entrainement davantage que refléter des formes qui apparaitraient dans de futures données telles que des données de validation.

L'élagage consiste à sélectionner successivement un noeud de décision et à le re-désigner comme noeud feuille (ainsi en supprimant les branches au delà du noeud de décision (ses 'sous-arbres') et ainsi réduire la taille de l'arbre). Le processus d'élagage diminue l'erreur de mauvaise classification dans l'ensemble des données de validation pour arriver à un arbre qui capture les formes mais pas le bruit dans les données d'apprentissage. Il utilise un critère appelé « coût de complexité » d'un arbre pour générer une séquence d'arbres qui sont successivement plus petits jusqu'au point d'avoir un arbre ayant juste le noeud racine.
(qu'est ce qu'une règle de classification pour un arbre ayant juste un noeud ?). On choisit comme notre meilleur arbre le seul arbre dans la séquence qui donne l'erreur de mauvaise classification la plus petite dans les données de validation.

Le critère du coût en complexité que CART utilise est simplement l'erreur de mauvaise classification d'un arbre (basé sur les données de validation) plus un facteur de pénalité pour la taille de l'arbre.

Le facteur de pénalité est basé sur un paramètre, que nous appelerons α, qui est la pénalité par noeud. Le critère du coût en complexité pour un arbre est $Err(T) + \alpha |L(T)|$, où Err(T) est la fraction des données d'observation de validation qui sont mal classés par l'arbre T, L(T) est le nombre de feuilles dans l'arbre T et α le coût de pénalité par noeud variant de zéro à l'infini. Quand $\alpha = 0$ il n'y a pas de pénalité pour avoir trop de noeuds dans l'arbre et le meilleur arbre retenu est l'arbre complet non élagué. Quand on augmente α vers de grandes valeurs le coût de la pénalité du composant submerge l'erreur de mauvaise classification du critère et le meilleur arbre est simplement l'arbre avec le moins de feuilles, et précisément l'arbre avec un noeud. En augmentant la valeur de α à partir de zéro jusqu'à une certaine valeur on se trouvera dans la situation où pour un certain arbre T1 formé par coupure d'un sous-arbre à un noeud de décision on aura la balance entre l' extra coût de l'augmentation de l'erreur de mauvaise classification contre le coût de pénalité d'avoir très peu de feuilles. On élague l'arbre complet à ce noeud de décision par coupure de son sous-arbre et en re-définissant le noeud de décision comme noeud feuille. Appelons cet arbre T1. On repète la logique que nous avons appliqué précédemment pour l'arbre complet et en augmentant α . En continuant de cette manière on génère une succession d'arbres en diminuant le nombre de noeuds jusqu'à l'arbre trivial composé d'un seul noeud.

A partir de cet ensemble d'arbres il semble naturel de choisir celui qui donne l'erreur de classification minimale sur le jeu de données de validation. On appelle cet arbre: **Arbre d'Erreur Minimum**.

Jetons un oeil sur les données du Boston Housing pour illustrer. Les fichiers sont générés par XLMiner. A droite le taux d'erreur en fonction du nombre de noeuds. Les données sont réparties en 3 classes avec 304 observations. L'arbre complet ne comporte aucune erreur (phase de grossissement de l'arbre)

Training Misclassification Summary

Classification Confusion Matrix			
	Predicted Class		
Actual Class	1	2	3
1	59	0	0
2	0	194	0
3	0	0	51

Error Report			
Class	# Cases	# Errors	% Error
1	59	0	0.00
2	194	0	0.00
3	51	0	0.00
Overall	304	0	0.00

Training Log

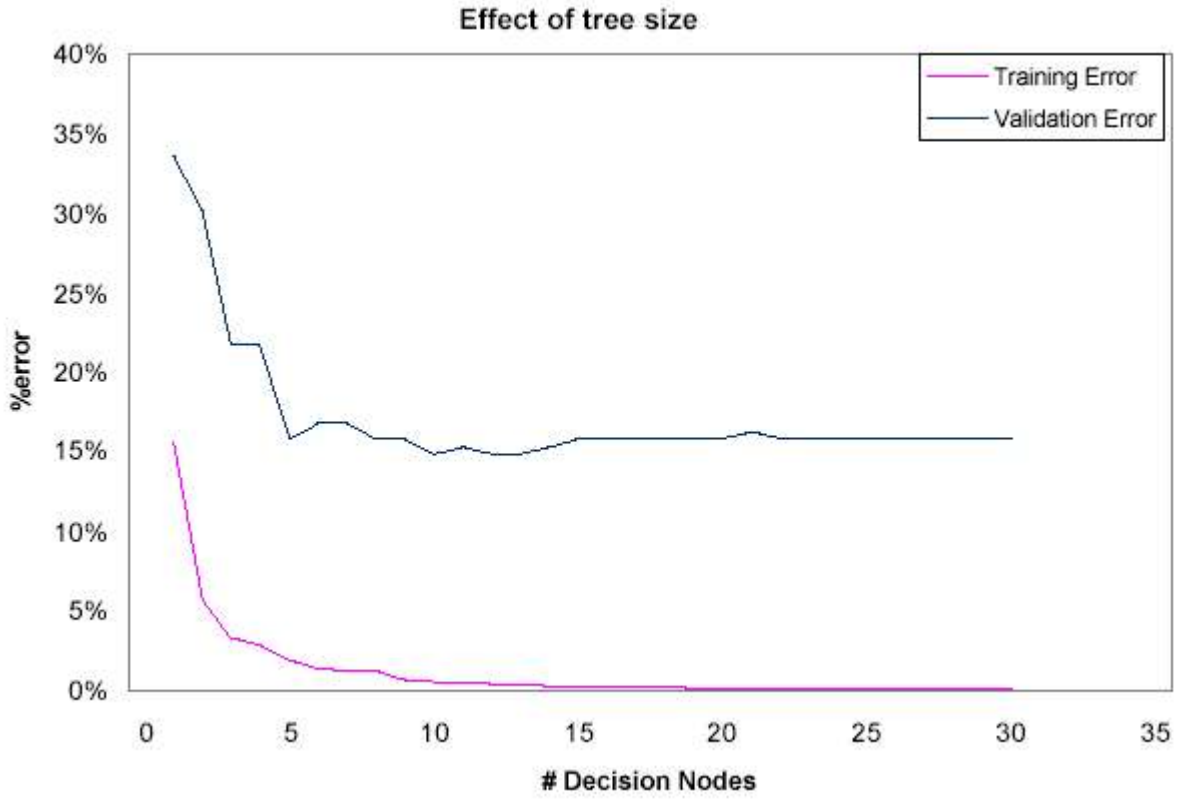
Growing the Tree	
#Nodes	Error
0	36.18
1	15.64
2	5.75
3	3.29
4	2.94
5	1.88
6	1.42
7	1.26
8	1.2
9	0.63
10	0.59
11	0.49
12	0.42
13	0.35
14	0.34
15	0.32
16	0.25
17	0.22
18	0.21
19	0.15
20	0.09
21	0.09
22	0.09
23	0.08
24	0.05
25	0.03
26	0.03
27	0.02
28	0.01
29	0
30	0

La table des logs montre pour chaque ligne le nombre de noeuds de décision dans l'arbre à chaque étape et l'erreur de mauvaise classification correspondante (pourcentage) pour les données d'entrainement appliquant une règle de vote à chaque feuille. On voit que l'erreur decroit fortement au fur et à mesure que le nombre de noeuds augmente à partir de zéro (où l'arbre consiste juste en un noeud racine) jusqu'à 30. L'erreur chute rapidement au début passant de 36% à 3% avec variation de noeuds de décision passant de 0 à 3. Ensuite l'amélioration est lente au fur et à mesure de l'accroissement de la taille de l'arbre. Finalement on arrête pour un arbre complet de 30 noeuds de décision (ou en équivalent à 31 feuilles) avec aucune erreur dans les données d'entrainement, comme cela est indiqu é dans la table de confusion et le report d'erreur par classe.

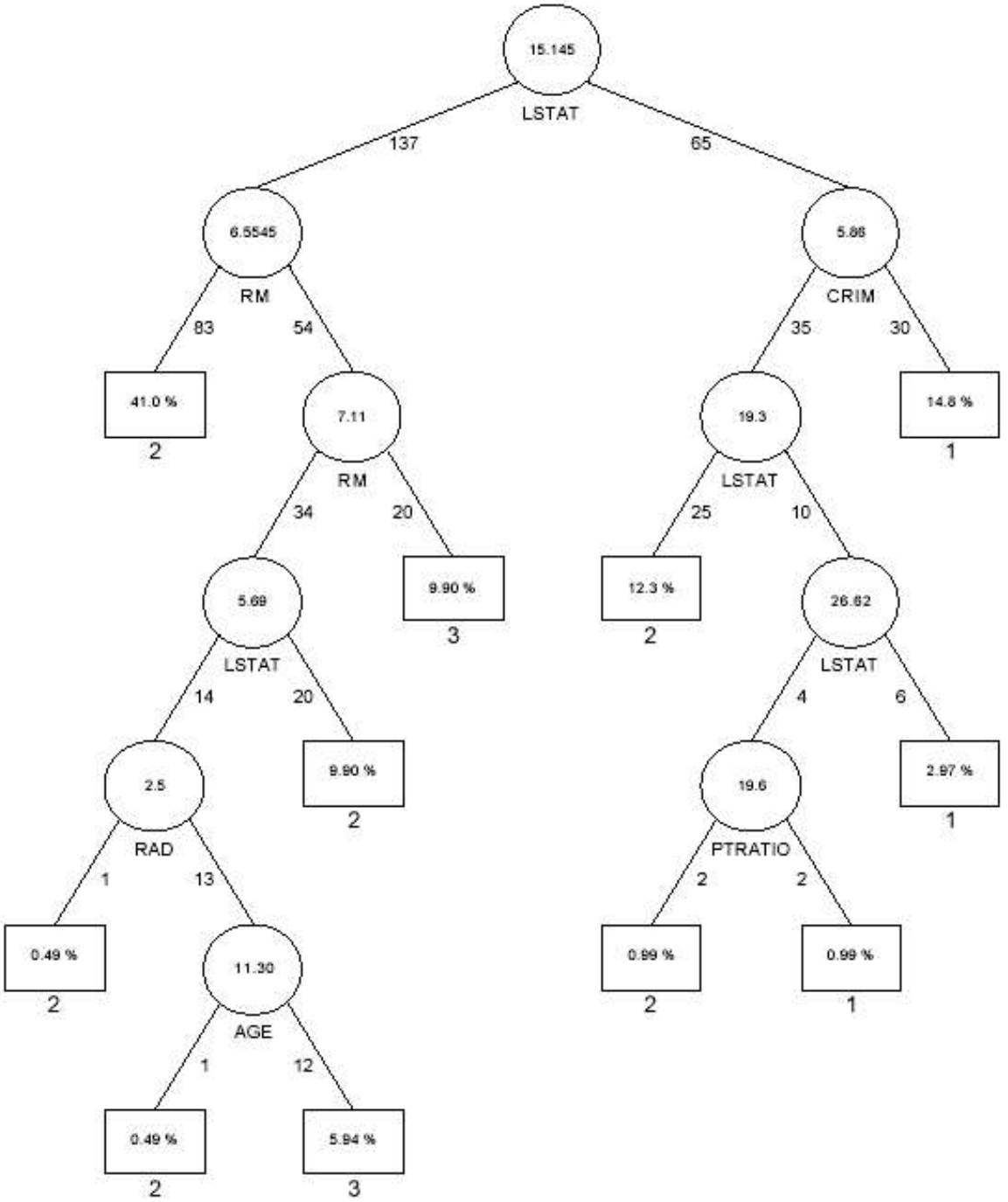
La sortie générée par XLMiner durant la phase d'élagage.

# Decision Nodes	Training Error	Validation Error			
30	0.00%	15.84%			
29	0.00%	15.84%			
28	0.01%	15.84%			
27	0.02%	15.84%			
26	0.03%	15.84%			
25	0.03%	15.84%			
24	0.05%	15.84%			
23	0.08%	15.84%			
22	0.09%	15.84%			
21	0.09%	16.34%			
20	0.09%	15.84%			
19	0.15%	15.84%			
18	0.21%	15.84%			
17	0.22%	15.84%			
16	0.25%	15.84%			
15	0.32%	15.84%			
14	0.34%	15.35%			
13	0.35%	14.85%			
12	0.42%	14.85%			
11	0.49%	15.35%			
10	0.59%	14.85%	<-- Minimum Error Prune	Std. Err.	0.02501957
9	0.63%	15.84%			
8	1.20%	15.84%			
7	1.26%	16.83%			
6	1.42%	16.83%			
5	1.88%	15.84%	<-- Best Prune		
4	2.94%	21.78%			
3	3.29%	21.78%			
2	5.75%	30.20%			
1	15.64%	33.66%			

On note maintenant que lorsque le nombre de noeuds de décision décroit l'erreur avec les données de validation a une tendance à décroitre lentement (avec une certaine fluctuation) jusqu'à un taux d'erreur de 14.85% pour un arbre à 10 noeuds. Voir le graphe ci-dessous. Ensuite l'erreur augmente, montant brusquement lorsque l'arbre devient petit. L'arbre d'erreur minimum est sélectionné comme étant celui à 10 noeuds de décision (pourquoi pas celui à 13 noeuds?)



Voici l'Arbre d'Erreur Minimum



On notera que la sortie de XLMiner dans la phase d'élagage met en valeur un autre arbre en plus de l'arbre d'erreur minimum. C'est le meilleur arbre élagué, un arbre avec 5 noeuds de décision. La raison pour laquelle cet arbre est important est qu'il est le plus petit de la séquence d'élagage qui a une erreur proche de l'erreur standard de l'arbre d'erreur minimum. L'estimation de l'erreur que nous obtenons à partir des données de validation est juste cela:

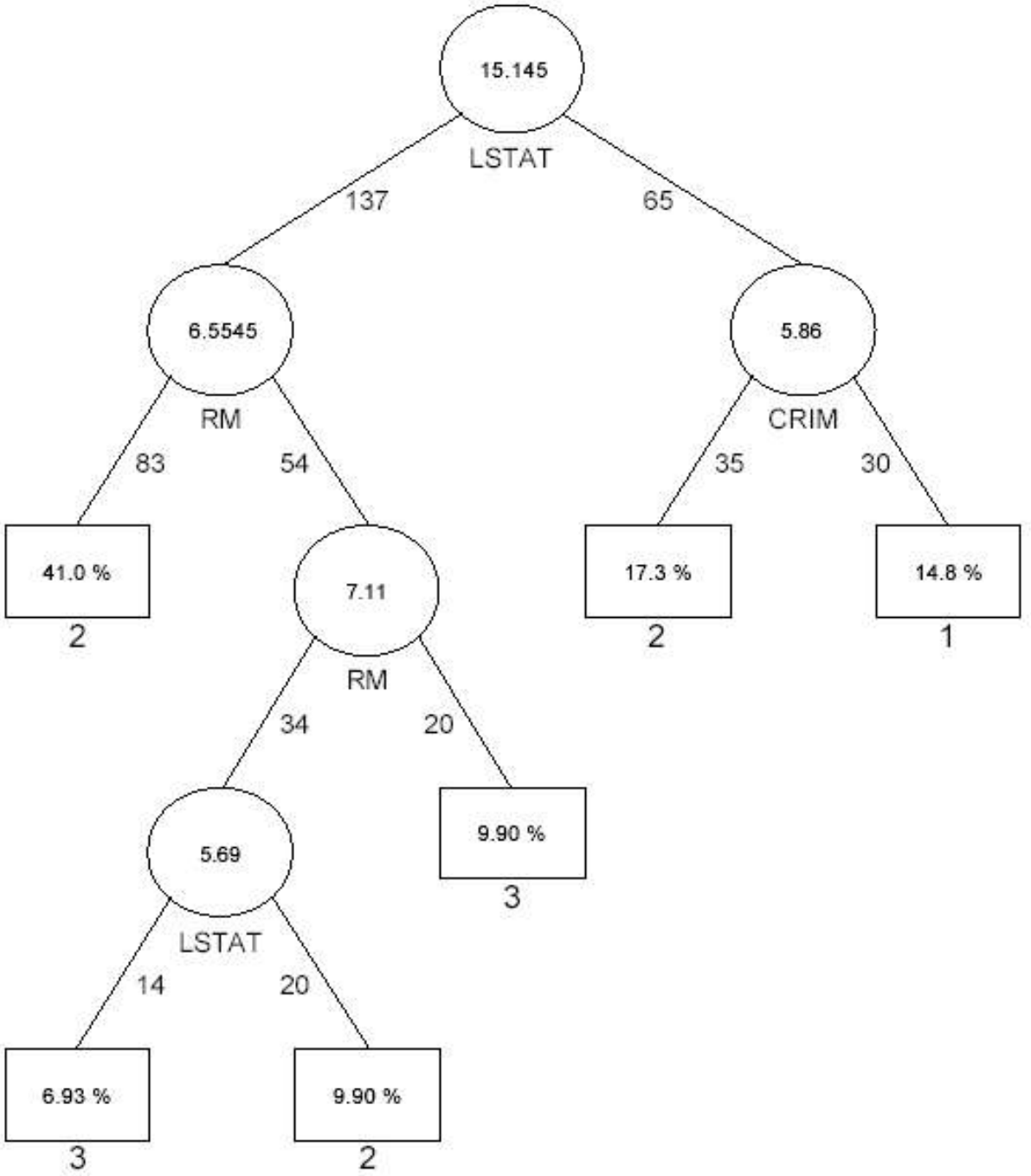
c'est une estimation. Si on avait eu un autre ensemble de données de validation l'erreur minimum aurait été différente. Le taux d'erreur minimum que nous avons calculé peut être envisagé comme la valeur observée d'une variable aléatoire avec une erreur standard (écart type estimé) égale à :

$$\sqrt{E_{\min}(1-E_{\min})/N_{val}}$$

où Emin est le taux d'erreur (comme fraction) pour l'arbre d'erreur minimum et Nval est le nombre d'observations

dans l'ensemble des données de validation. Pour notre exemple $E\text{min} = 0.1485$
et $Nval = 202$, ainsi l'erreur standar est 0.025.

Voici le meilleur arbre élagué



On montre dans ce tableau la table de confusion et un résumé des erreurs de classification pour le meilleur arbre élagué.

Validation Misclassification Summary

Classification Confusion Matrix			
	Predicted Class		
Actual Class	1	2	3
1	25	10	0
2	5	120	9
3	0	8	25

Error Report			
Class	# Cases	# Errors	% Error
1	35	10	28.57
2	134	14	10.45
3	33	8	24.24
Overall	202	32	15.84

Il est important de noter que du fait de l'utilisation de données de validation pour la classification, strictement parlant cela n'est pas correct de comparer le taux d'erreur ci-dessus directement avec d'autres procédures de classification qui utilisent seulement les données d'entrainement pour construire les règles de classification. Une comparaison juste serait de partitionner en données d'entrainement (TDautre) pour les autres procédures en un partition comprenant les données d'entrainement (TDarbre) et les données de validation (VDarbre) pour les arbres de classification. Le taux d'erreur pour l'arbre de classification créé en utilisant TDarbre pour faire pousser l'arbre et ainsi VDarbre pour l'élaguer peut maintenant être comparé en utilisant les données de validation utilisées par d'autres classifieurs (VDautre) en tant que nouvelles données.

Règles de classification à partir des arbres

Une des raisons pour lesquelles les arbres de classification sont très populaires est qu'ils fournissent facilement des règles de classification compréhensibles (au moins si les arbres ne sont pas trop grands). Chaque feuille est équivalente à une règle de classification. Par exemple, la feuille supérieure gauche du meilleur arbre élagué précédent, nous donne la règle:

IF(LSTAT <= 15.145) AND (ROOM <= 6.5545) THEN CLASS = 2

De telles règles sont facilement explicables aux managers et aux directions opérationnelles comparées aux sorties d'autres classifieurs tels que les fonctions discriminantes. Leur logique est certainement de loin plus transparente que les poids d'un réseau de neurones !

La méthode de l'arbre est un bon choix de classifieur « sur étagère »

Nous avons dit au début de ce chapitre que les arbes de classification nécessitent un relatif petit effort pour les développeurs. Donnons nos raisons pour ceci. Les arbres n'ont pas besoin de paramètres de tuning. Il n'y a pas besoin de transformation de variables (n'importe quelle transformation monotone des variables donnera les mêmes arbres). La sélection d'un sous-ensemble de variables est automatique puisque c'est une partie de la sélection de la division (split); dans notre exemple on note que le meilleur arbre élagué a automatiquement sélectionné juste 3 variables (LSTAT, RM et CRIM) parmi un ensemble de 13 variables au départ. Les arbres sont aussi intrinsèquement robustes aux points extrèmes comme le choix d'une division dépend de l'ordre des valeurs d'observation et non des magnitudes absolues de ces valeurs.

Finalement la procédure CART peut être lisiblement modifiée pour supporter des valeurs manquantes sans avoir à modifier des valeurs ou supprimer des observations avec des valeurs manquantes. La méthode peut aussi être étendue pour incorporer un classement (ranking) d'importance pour les variables en terme de leur impact sur la qualité de la classification.

En plus de CHAID, une autre méthode d'arbre de classification populaire est ID3 (induction decision tree) (et son successeur C4.5). Cette méthode a été développée par Quinlan un chercheur leader en apprentissage, et est populaire parmi les développeurs de classifieurs qui ont un background en apprentissage.