Feuille de Travaux Dirigés n° 2
Arbres de décisions : régression et classification

1 Les arbres de décisions

L'idée de de base est très simple. Nous cherchons à prédire une réponse ou un groupe d'appartenance Y à partir des données X_1, X_2, ..., X_p. Nous allons arriver à cet objectif en faisant grandir un arbre de décision binaire. À chacun des noeuds intérieurs à l'arbre, nous appliquons un test à l'une variables d'entrée X_i. En fonction du résultat du test, nous descendons l'arbre soit par la sous-branche à gauche du noeud soit par celle à droite. Après un certain nombre de descentes nous atteignons nécessairement une feuille de l'arbre où nous faisons une prédiction. Celle-ci agrège ou moyenne toutes celles réalisées à partir des données d'apprentissage qui mènent à cette feuille. La Figure 1, page 9, devrait vous permettre de clarifier cette description.

Pourquoi faire cela ? Les prédicteurs classiques comme les modèles de régression linéaire ou polynomiale sont des modèles globaux dans lesquels une seule formule de prédiction est supposée est valable pour l'intégralité des données à étudier. Lorsque les données présentes de nombreuses facettes qui interagissent entre elles de manière compliquée et non-linéaire, obtenir un modèle global s'avère très difficile et d'une complexité décourageante dans les rares cas où c'est possible. Certaines approches par lissage non-paramétrique essayent d'ajuster locallement des modèles puis de les raccorder ensemble, mais à nouveau ils peuvent être difficiles à interpréter. Les modèles additifs restent néanmoins relativement simples à appréhender mais ne sont pas satisfaisants dans toutes les situations).

Une approche alternative pour faire de la régression non-linéaire est de découper, c'est-à-dire partitionner, l'espace d'étude en des régions de taille plus petite au sein desquelles il sera plus facile de gérer les interactions entre les variables. Nous partitionnons alors à nouveau chacune de ces régions jusqu'à obtenir au final des fragments de l'espace d'étude pour lesquels il est possible d'utiliser des modèles simples pour décrire convenablement la réponse. Cette manière de procéder s'appelle le "recursive partitioning". Le modèle global est donc formé de deux parties : la première est la partition récursive de l'espace et la seconde est un modèle simple associé à chacune des cellules de cette partition.

Intéressons-nous à nouveau à la Figure 1, page 9, et à la description qui l'accompage. Les arbes de prédiction utilisent l'arbre pour représenter la partition récursive. Chacun des noeuds finaux de l'arbre, c'est-à-dire les feuilles, représente une cellule

de la partition à laquelle est attachée un modèle simple qui s'applique uniquement à cette cellule. Un point x appartient à une feuille si x appartient à la cellule correspondante de la partition. Pour déterminer dans quelle cellule un point x est situé, nous partons de la racine de l'arbre et répondons à la série de questions qui nous est posée à chaque noeud sur les caractéristiques de x. En effet, les noeuds intérieurs de l'arbre sont étiquettés avec les questions et les branches qui les relient par les réponses à ces questions. La question suivante à poser dépend ainsi des réponses que nous avons apportées aux précédentes. Dans la version classique de ces arbres, chaque question ne concerne qu'un seul attribut et a une réponse binaire par oui ou non, par exemple "Est-ce que `HSGrad > .78` ?" (Est-ce que la proportion de bacheliers dans le conté est supérieure à 0,78 ?) ou "Est-ce que `Region == Midwest`" (Le conté appartient-il au Middle-West ?). Les variables prédictives utilisées peuvent de n'importte quel type (quantitatives continues ou discrètes, qualitatives ordinales ou nominales). Il est également possible de poser des questions comportant plus de deux réponses mais il est toujours possible de se ramener au cas binaire en augmentant éventuellement la taille de l'arbre. De même poser une question concernant simultanément plusieurs variables revient à poser successivement des questions sur chacune d'entre elles.

Passons maintenant à la description de la seconde partie des arbres de prédiction, le modèle simple. Dans le cas des arbres de régression classiques, le modèle dans chaque cellule est une estimation **constante** pour Y. Plus précisement, si nous supposons que les points (x_1, y_1), (x_2, y_2), ..., (x_c, y_c) représentent toutes les qui sont associées à la feuille l, alors le modèle pour la feuille l est simplement $\widehat{y} = \frac{1}{c}\sum_{i=1}^{c} y_i$, c'est-à-dire la moyenne des valeurs observées de la réponse dans cette feuille. Il s'agit d'un modèle constant par morceaux [1]. Cette approche a plusieurs avantages :

- Il est très rapide d'obtenir une prédiction (pas de calcul complexe à réaliser, il suffit de lire la valeur de la constante reportée dans l'arbre)
- Il est facile de comprendre quelles sont les variables qui sont importantes pour faire une (bonne) prédiction (il suffit de regarder l'arbre)
- Si certaines valeurs sont manquantes, il se peut que nous ne puissions pas parcourir l'arbre jusqu'à l'une de ces feuilles, mais nous pouvons néanmoins proposer une prédiction en faisant la moyenne des valeurs associées aux feuilles auxquelles nous aurions pu parvenir à partir du noeud intérieur où nous nous sommes arrêtés.
- Le modèle fournit une réponse discontinue qui peut ainsi facilement s'ajuster à une réponse qui l'est aussi. Si la réponse est continue il sera également possibleen, utilisant un nombre suffisamment élevé de cellules, d'approcher aussi finement que nous le voulons cette réponse.
- Il existe des algorithmes rapides et faibles pour construire ces arbres.

Une dernière analogie avant de s'intéresser aux exemples. Une des méthodes non-paramétrique les plus simple à comprendre est celle des k plus proches voisins : trou-

1. Il aurait bien sûr été possible d'ajuster un modèle de régression linéaire au sein de chaque feuille ce qui aurait abouti au final à un modèle linéaire par morceaux. Toutefois, comme les feuilles sont construites pour être homogènes, il n'y a généralement que peu de différence entre ceux deux modèles.

ver les points qui sont le plus semblables à celui que nous étudions et s'intéresser à ce qu'ils font en moyenne. Toutefois cette approche a deux grands inconvenients. Le premier est que nous décidons que des points sont semblables uniquement à partir des valeurs des prédicteurs sans utiliser celle de la réponse. Le second est que nous prenons même la valeur de k pour tout le domaine d'étude et que vraisemblement certains points ont plus de proches voisins qui partagent leur comportement que d'autres. Les arbres permettent de résoudre ces deux problèmes : les feuilles correspondent aux régions de l'espace d'étude pour lesquelles à la fois les prédicteurs et la réponse sont semblables et la taille de ces régions qui contiennent ces proches voisins peut varier librement. En conclusion ; les arbres peuvent être vus comme une méthode des plus proches voisins adaptative.

2 Arbres de régression

2.1 L'immobilier en Californie

```
> setwd("C:/Données/DEPULP/DU/DU_Semaine3/Sources/Arbres")
```

Installer la bibliothèque `tree` pour le langage **R**. Il existe plusieurs bibliothèque sur les arbres de régression. Celle-ci est la plus simple.

```
> install.packages("tree")
```

Charger cette bibliothèque.

```
> library("tree")
```

```
> calif = read.table("cadata.txt", header = TRUE)
> treefit = tree(log(MedianHouseValue) ~ Longitude +
+     Latitude, data = calif)
```

La dernière commande réalise une régression du logarithme népérien du prix par rapport à la longitude et à la latitude. La Figure 2, page 10, montre l'arbre en lui-même ; la Figure 3, page 11, montre la partition superposée aux valeurs réelles des prix en Californie. (Nous n'avons choisi que deux régresseurs pour pouvoir obtenir cette représentation graphique). Visuellement, nous avons l'impression que le modèle parvient à décrire honorablement l'interaction en latitude et longitude ainsi que le fait que les prix soient plus élevés à proximité des côtes et des grandes villes.
Quantitaviement l'erreur n'est pas mauvaise :

```
> summary(treefit)
```

```
Regression tree:
tree(formula = log(MedianHouseValue) ~ Longitude + Latitude,
    data = calif)
Number of terminal nodes:  12
Residual mean deviance:  0.1662 = 3429 / 20630
```

```
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-2.75900 -0.26080 -0.01359  0.00000  0.26310  1.84100
```

Dans ce cas, la déviance est simplement le carré moyen de l'erreur ; soit un RMS de l'erreur de 0,41, ce qui est honorable sachant que nous utilisons seulement 2 variables et que l'arbre ne comporte que 12 noeuds.
Une manière basique d'apprécier la flexibilité d'un arbre est de compter le nombre de feuille qu'il possède. La fonction `tree` possède plusieurs paramètre de contrôle qui servent à limiter la croissance de l'arbre : chaque noeud doit contenir un nombre minimal de points et l'ajout d'un noeud doit réduire l'erreur d'une valeur supérieure à un minimum fixé. Celui-ci, `mindev`, est fixé par défaut à 0,01 ; diminuons-le et regardons ce qui se passe.
La Figure 4, page 12, montre l'arbre. Il comporte 68 noeuds et est plutôt compliqué à lire. Toutefois il suffit de zoomer sur le graphique pour y parvenir. La Figure 5, page 13, montre la partition associée à ce nouvel arbre. Elle est évidemment plus fine que celle représentée à la Figure 3, page 11, et permet de prédire plus précisemment les prix réels des logements (RMS de l'erreur 0,32). Il est intéressant de remarquer que la partition obtenue ne correspond pas simplement à un découpage uniforme des cellules les plus grosses de la première partition : certaines nouvelles cellules sont très petites, d'autres assez grandes. Les villes deviennent beaucoup plus détaillées que le reste. Bien entendu, les coordonnées géographiques seueles ne peuvent complètement les prix des logements mais elles nous ont permis de faire des graphiques intéressants.
Considérons maintenant toutes les variables présentes dans le jeu de données pour construire notre arbre.

```
> treefit3 <- tree(log(MedianHouseValue) ~ ., data = calif)
```

Le résultat est repris dans la Figure 6, page 14,. Le modèle a 15 feuilles, à comparer aux 68 feuilles de l'arbre `treefit2`, et la RMS de l'erreur est pourtant presque aussi basse avec 0,36. Remarquons que l'arbre `treefit3` n'utilise les coordonnées géographiques que si les revenus sont suffisamment bas et que plusieurs des prédicteurs disponibles n'ont pas été retenu dans l'arbre. Ainsi nous ne disposons pas d'une partition qu'il serait possible de comparer aux précédentes. Nous représentons néanmoins les prédictions réalisées avec l'arbre `treefit3` sur la Figure 7, page 15.

2.2 Validation croisée et élagage

La bibliothèque `tree` contient les fonctions `prune.tree` et `cv.tree` pour élaguer les arbres par validation croisée.
La fonction `prune.tree` s'applique à un arbre ajusté par la fonction `tree` et évalue l'erreur de l'arbre de diverses variantes de cet arbre obtenues par élagage successif jusqu'à ce qu'il ne reste que la souche. Cette évaluation peut être réalisée à l'aide de nouvelles données de test ou du jeu de données d'apprentissage (choix par défaut). Il est possible de rechercher un arbre d'une taille précise qui est alors le meilleur arbre possible de cette taille obtenu par élagage. Si nous ne recherchons pas uniquement

le meilleur arbre, nous obtenons un objet qui montre le nombre de feuilles dans chacun des arbres et l'erreur de chacun d'entre eux. Il est possible de représenter graphiquement cet objet.

```
my.tree = tree(y ~ x1 + x2, data=my.data) # Fits tree
prune.tree(my.tree,best=5) # Returns best pruned tree with 5 leaves,
                           # evaluating error on training data
prune.tree(my.tree,best=5,newdata=test.set) # Ditto, but evaluates on
                                            # test.set
my.tree.seq = prune.tree(my.tree) # Sequence of pruned tree
                                  # sizes/errors
plot(my.tree.seq) # Plots size vs. error
my.tree.seq$dev # Vector of error rates for prunings, in order
opt.trees = which(my.tree.seq$dev == min(my.tree.seq$dev)) # Positions
  # of optimal (with respect to error) trees
min(my.tree.seq$size[opt.trees]) # Size of smallest optimal tree
```

Enfin, `prune.tree` possède un argument optionel `method`. Sa valeur par défaut est `method=deviance` qui ajuste les arbres en minimisant le carré moyen de l'erreur (pour les réponses continues) ou l'opposé de la log-vraisemblance (pour les réponses discrètes) [2].
La fonction `cv.tree` réalise une validation croisée k-fold (avec k fixé à 10 par défaut). Il a besoin comme argument d'un arbre qui a été ajusté et d'une fonction qui prend comme arguments un arbre et de nouvelle données. Par défaut, cette fonction est la fonction `prune.tree`.

```
my.tree.cv = cv.tree(my.tree)
```

Le type de sortie de la fonction `cv.tree` est le même que celui de la fonction à la quelle elle est appliquée. Par exemple, en exécutant

```
cv.tree(my.tree,best=19)
```

nous récupérons le meilleur arbre (au sens de la validation croisée) qui comporte au plus 19 feuilles. En exécutant

```
cv.tree(my.tree)
```

nous récupérons les informations sur la validation croisée de toute la suite d'arbres élagués, par exemple en utilisant `plot(cv.tree(my.tree))`. Les arguments optionels de la fonction `cv.tree` inclut K, le nombre de groupes de validation croisée, et n'importe quelle option que pourrait utiliser la fonction à laquelle `cv.tree` est appliquée.
En guise d'exemple, revenons à l'arbre `treefit2`, qui servait à prédire le prix des logements en Californie en fonction des coordonnées géographiques mais comportait

2. Pour les réponses discrètes, il serait peut-être possible d'obtenir des meilleurs résultats en utilisant `method="misclass"` qui s'intéresse au taux de mauvais classement.

un très grand nombre de noeuds parce que nous avions modifié la valeur minimale `mindev`. La Figure 8, page 16, illustre le compromis taille/performance. Les Figures 9, page 17, et 10, page 18, montre le résultat de l'élagage à la plus petite taille compatible avec l'erreur estimée par validation croisée.

2.3 Les 2000 plus grandes entreprises du monde en 2004 : le fichier Forbes2000

Dans cet exemple nous utilisons une deuxième bibliothèque permettant de créer des arbres de décision.

```
> setwd("C:/Données/DEPULP/DU/DU_Semaine3/Sources/Arbres")
```

Installer la bibliothèque `rpart` pour le langage **R** et la bibliothèque `HSAUR` qui contient les données que nous allons analyser.

```
> install.packages("rpart")
> install.packages("HSAUR")
```

Charger cette bibliothèque, l'objet `Forbes2000` qui contient les données puis éliminer les sociétés pour lesquelles les profits ne sont pas connus.

```
> library("rpart")
> data("Forbes2000", package = "HSAUR")
> Forbes2000 <- subset(Forbes2000, !is.na(profits))
```

La fonction `rpart` peut être utilisée pour faire grandir un arbre de régression. La variable cible (ou réponse) et les variables explicatives sont définies par un modèle comme pour la fonction `lm` par exemple. Un arbre de taille importante est créé par défaut.

```
> forbes_rpart <- rpart(profits ~ assets + marketvalue +
+     sales, data = Forbes2000)
```

La fonction `print` permet de réprésenter le contenu de l'objet `forbes_rpart`.

```
> print(forbes_rpart)

n= 1995

node), split, n, deviance, yval
      * denotes terminal node

 1) root 1995 6214.95400  0.38113280
   2) marketvalue< 89.335 1962 3724.62000  0.26958210
     4) marketvalue< 32.715 1845 1666.62500  0.17708940
       8) assets>=329.025 10  404.76020 -3.36600000 *
       9) assets< 329.025 1835 1135.64500  0.19639780
```

```
      18) marketvalue< 7.895 1326  378.89680  0.07812217 *
      19) marketvalue>=7.895 509  689.87540  0.50451870 *
    5) marketvalue>=32.715 117 1793.31400  1.72812000
     10) sales>=54.845 18 1281.53800 -0.59944440 *
     11) sales< 54.845 99  396.52910  2.15131300
       22) sales< 42.935 89  262.96110  1.87247200 *
       23) sales>=42.935 10   65.06041  4.63300000 *
  3) marketvalue>=89.335 33 1014.37900  7.01333300
    6) sales< 91.92 24  529.76020  5.21083300 *
    7) sales>=91.92 9  198.70660 11.82000000 *
```

Une représentation graphique à l'aide de la fonction `print` est souvent préférée, voir la Figure 11 page 19.
La fonction `print` permet de réprésenter le contenu du tableau `cptable` de l'objet `forbes_rpart` qui nous informe si certaines branches de l'arbre devraient être élaguées. En effet la colonne `xerror` contient une estimation de l'erreur de prédiction estimée par validation croisée obtenue pour plusieurs valeurs de `nsplit`, le nombre de noeuds intérieurs de l'arbre, différentes.

```
> print(forbes_rpart$cptable)

          CP nsplit rel error    xerror      xstd
1 0.23748446      0 1.0000000 1.0012023 0.1947409
2 0.04600397      1 0.7625155 0.8519540 0.2192849
3 0.04258786      2 0.7165116 0.8521501 0.2204562
4 0.02030891      3 0.6739237 0.7992577 0.2149693
5 0.01854336      4 0.6536148 0.7979798 0.2142314
6 0.01102304      5 0.6350714 0.8108151 0.2153346
7 0.01076006      6 0.6240484 0.8324255 0.2202232
8 0.01000000      7 0.6132883 0.8337581 0.2202734

> opt = which.min(forbes_rpart$cptable[, "xerror"])
```

Nous remarquons que la valeur la plus petite de `xerror` est obtenue pour `nsplit`=5. Nous la stockons dans l'objet `opt`. Par conséquent, le meilleur arbre possède 5 noeuds intérieurs. Nous élagons donc l'abre initial `forbes_rpart` qui en comporte 7 de la manière suivante :

```
> cp <- forbes_rpart$cptable[opt, "CP"]
> forbes_prune <- prune(forbes_rpart, cp = cp)
```

Le résultat est représenté à la Figure 12 page 20. L'arbre est plus petit. Grâce aux tailles et aux boîtes à moustaches associées à chacune des feuilles et qui ont été intégrées au graphique, nous constatons que la plus grande partie des entreprises reste groupée ensemble. Toutefois, posséder un marché supérieur à 32,72 milliards de dollards US, semble être un bon indicateur de grands profits.

Table des matières

Références

[1] Ripley B (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge England.

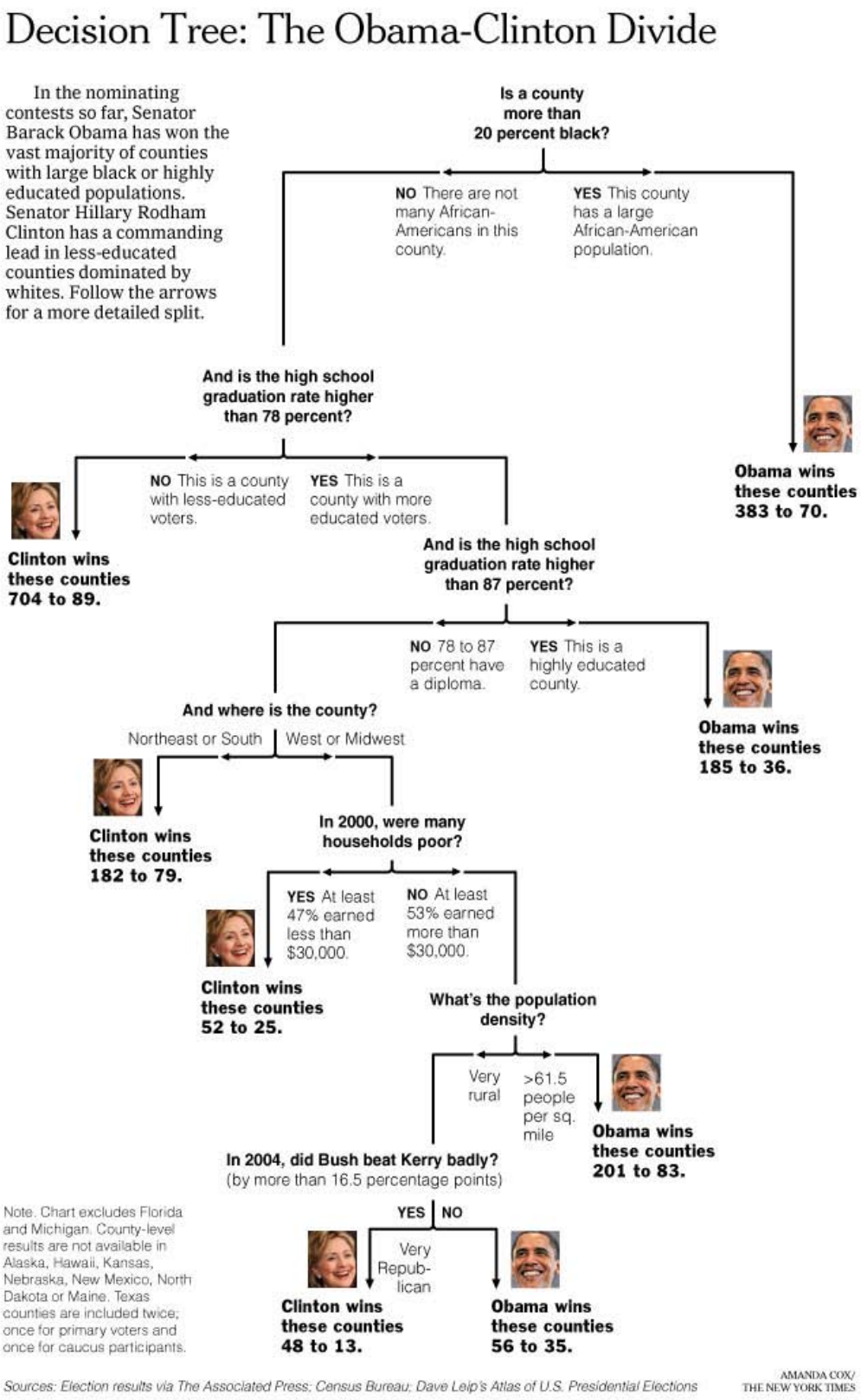
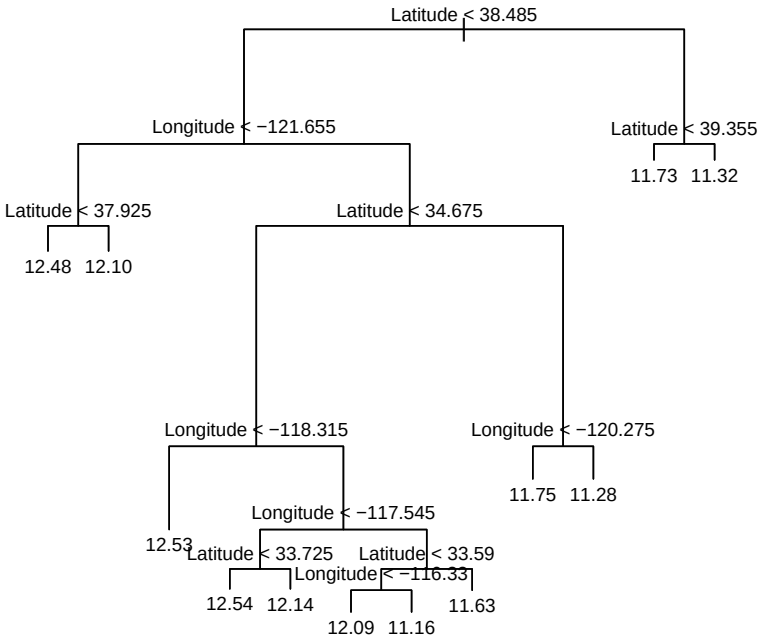
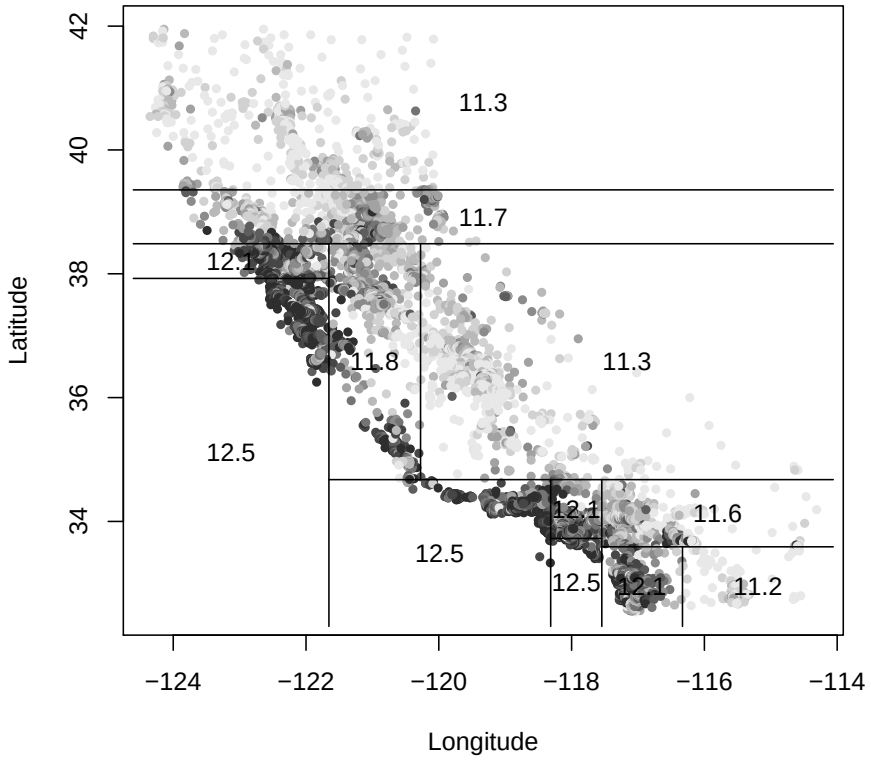


FIGURE 1 – Arbre de classification pour prédire, au niveau des contés et au jour du 16 Avril, les résultats des primaires du Parti Démocrate en 2008, réalisé par Amanada Cox pour le New York Times.



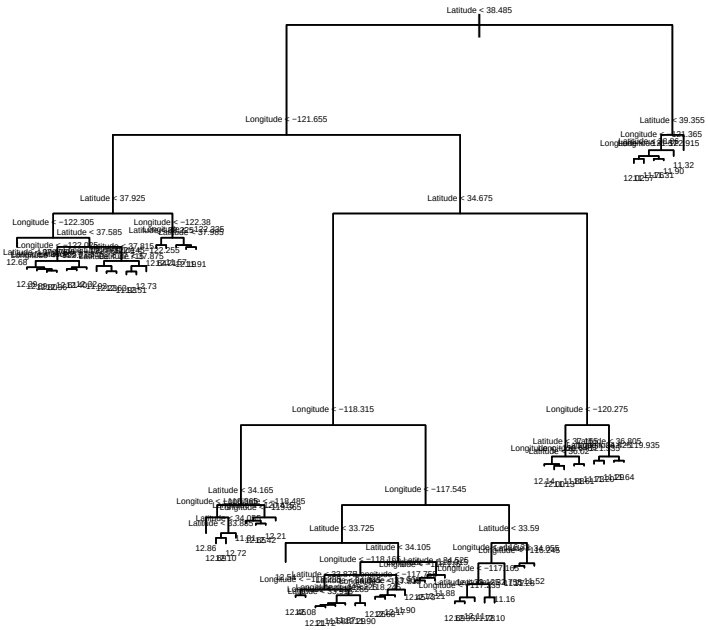

```
> plot(treefit)
> text(treefit, cex = 0.75)
```

FIGURE 2 – Arbre de régression pour prédire les prix des logements en Californie à partir de l'emplacement géographique. 'A chaque noeud intérieur, nous posons la question indiquée et descendons à gauche si la réponse est "yes", à droite si la réponse est "no". Les feuilles sont étiquettées avec le logarithme népérien du prix, la fonction utilisée pour l'affichage n'étant pas suffisamment flexible pour permettre d'appliquer une transformation aux étiquettes des feuilles.



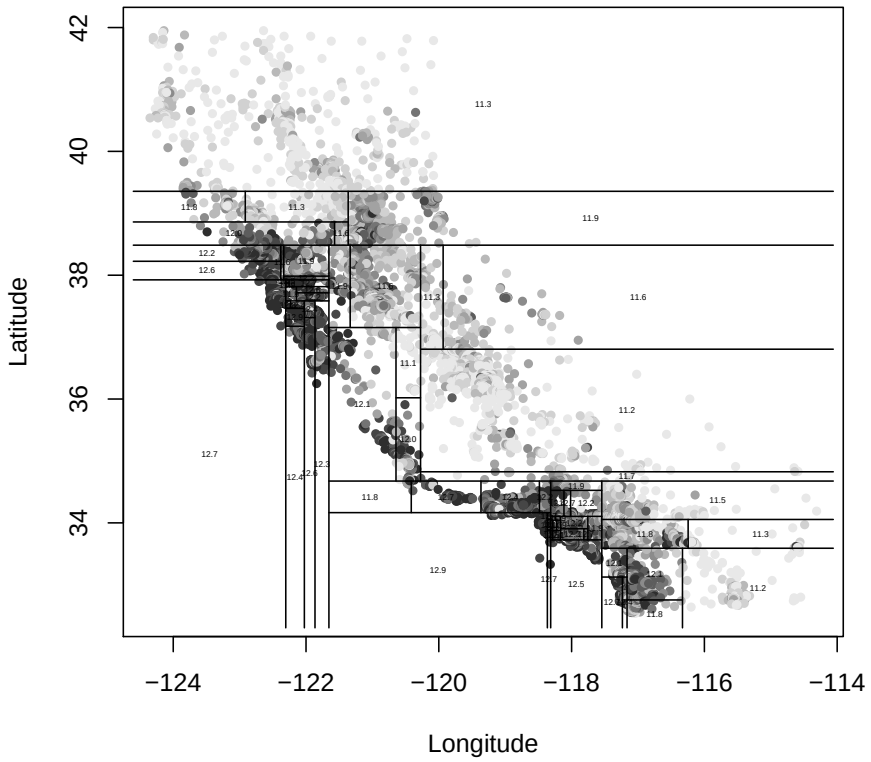

```
> price.deciles = quantile(calif$MedianHouseValue, 0:10/10)
> cut.prices = cut(calif$MedianHouseValue, price.deciles,
+     include.lowest = TRUE)
> plot(calif$Longitude, calif$Latitude, col = grey(10:2/11)[cut.prices],
+     pch = 20, xlab = "Longitude", ylab = "Latitude")
> partition.tree(treefit, ordvars = c("Longitude", "Latitude"),
+     add = TRUE)
```

FIGURE 3 – Carte des prix réels médians (les nuances de couleurs correspondent aux déciles, le plus foncé étant le plus cher) et la partition de l'arbre `treefit`.



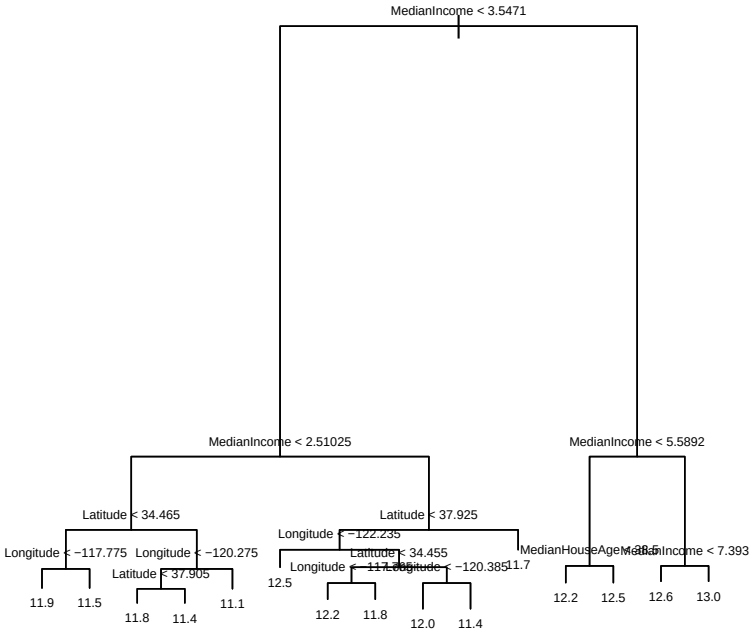

```
> treefit2 = tree(log(MedianHouseValue) ~ Longitude +
+     Latitude, data = calif, mindev = 0.001)
> plot(treefit2)
> text(treefit2, cex = 0.3)
```

FIGURE 4 – Identique à la Figure 2, page 10, en retenant des subdivisions pour une diminution plus faible de l'erreur.



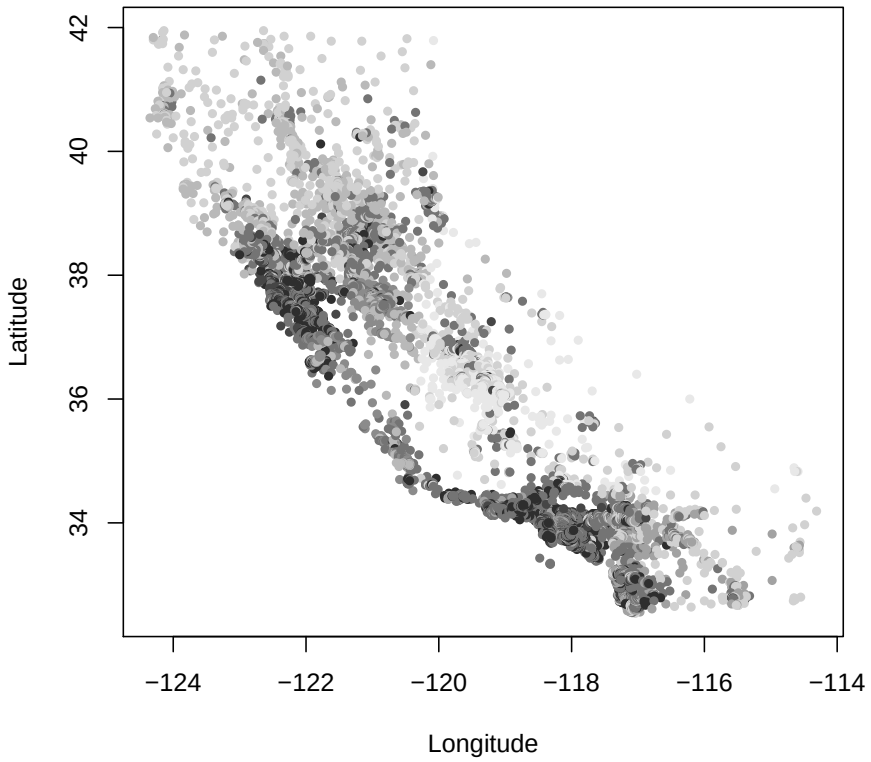

```
> plot(calif$Longitude, calif$Latitude, col = grey(10:2/11)[cut.prices],
+     pch = 20, xlab = "Longitude", ylab = "Latitude")
> partition.tree(treefit2, ordvars = c("Longitude",
+     "Latitude"), add = TRUE, cex = 0.3)
```

FIGURE 5 – Partition pour `treefit2`. Remarquer le grand niveau de détail autour des villes en comparaison avec celui moins élevé des cellules recouvrant les zones rurales où les variations de prix sont moins fortes.



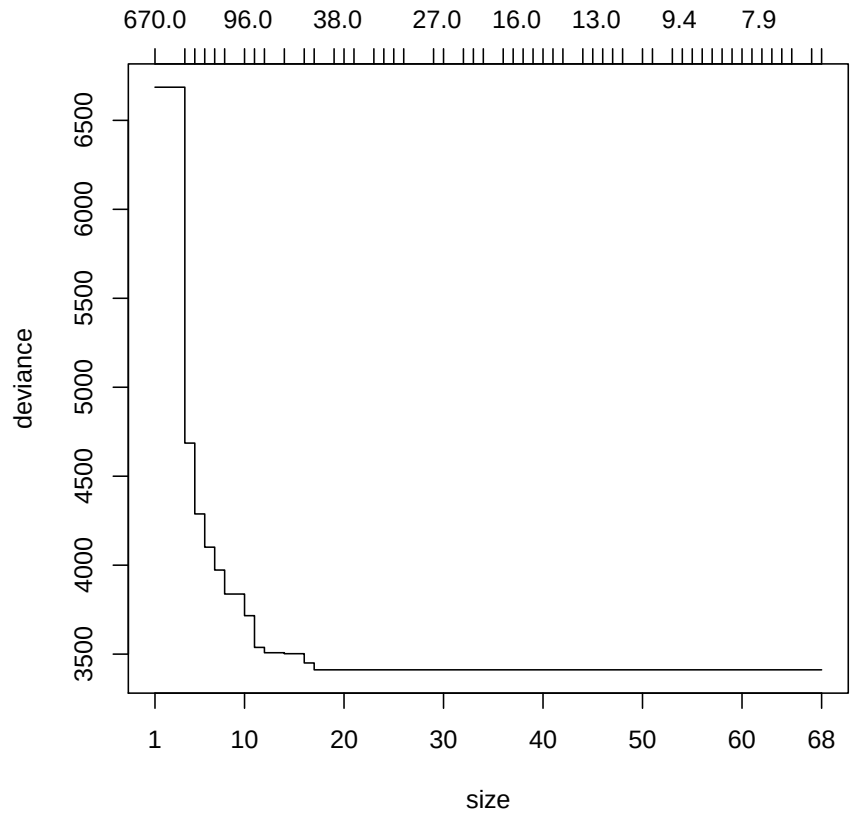

```
> plot(treefit3)
> text(treefit3, cex = 0.5, digits = 3)
```

FIGURE 6 – Arbre de régression `treefit3` construit à partir de tous les prédicteurs (potentiels) du logarithme népérien du prix. Remarquer que beaucoup des prédicteurs potentiels ne sont pas retenus au final dans l'arbre.



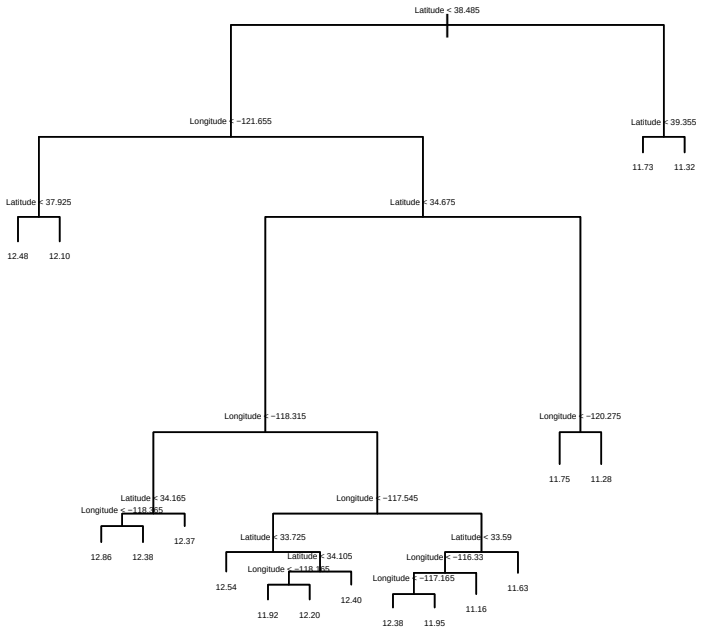

```
> cut.predictions = cut(predict(treefit3), log(price.deciles),
+     include.lowest = TRUE)
> plot(calif$Longitude, calif$Latitude, col = grey(10:2/11)[cut.predictions],
+     pch = 20, xlab = "Longitude", ylab = "Latitude")
```

FIGURE 7 – Valeurs prédites pour l'arbre `treefit3`. L'échelle de couleurs utilisée est la même que pour les autres graphiques où les points représentaient les valeurs réelles des prix.



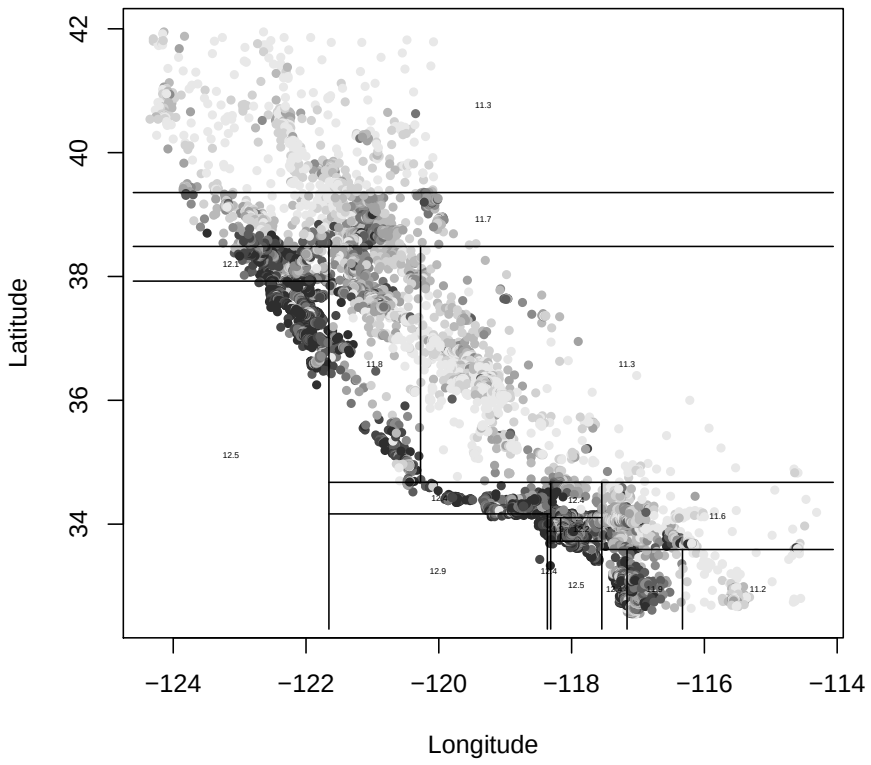

```
> treefit2.cv <- cv.tree(treefit2)
> plot(treefit2.cv)
```

FIGURE 8 – Somme des carrés des erreurs estimée par validation croisée (axe vertical) en fonction de la taille de l'arbre (axe horizontal) pour des élagages successifs de l'arbre `treefit2`. (L'échelle supérieure sur l'axe horizontal représente la pénalité "coût/complexité". L'idée est que l'élagage minimise (l'erreur totale)$+\lambda$(complexité) pour une certaine valeur de λ qui est ce qui est représenté sur cette échelle. Ici la complexité est une fonction du nombre de feuilles, voir [1] ou plus simplement ignorer l'échelle supérieure).



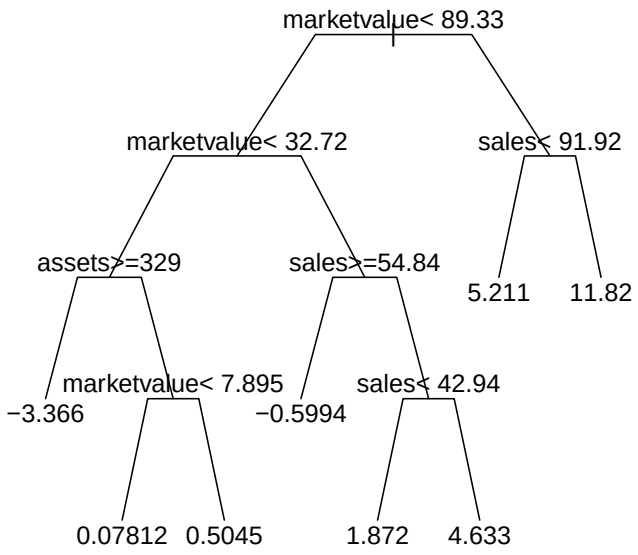

```
> opt.trees = which(treefit2.cv$dev == min(treefit2.cv$dev))
> best.leaves = min(treefit2.cv$size[opt.trees])
> treefit2.pruned = prune.tree(treefit2, best = best.leaves)
> plot(treefit2.pruned)
> text(treefit2.pruned, cex = 0.3)
```

FIGURE 9 – `treefit2`, après avoir été élagué par validation croisée 10-fold.



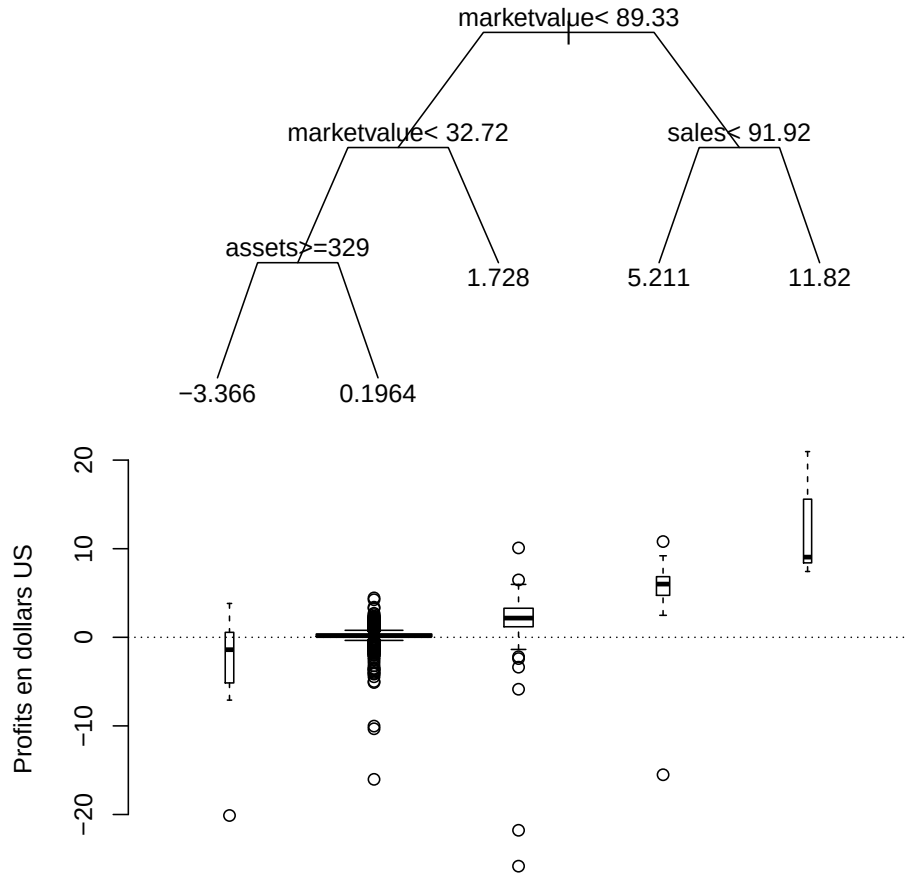

```
> plot(calif$Longitude, calif$Latitude, col = grey(10:2/11)[cut.prices],
+     pch = 20, xlab = "Longitude", ylab = "Latitude")
> partition.tree(treefit2.pruned, ordvars = c("Longitude",
+     "Latitude"), add = TRUE, cex = 0.3)
```

FIGURE 10 – La partition de la Californie associée à l'arbre `treefit2.pruned`. À comparer avec la Figure 5, page 13.




```
> plot(forbes_rpart, uniform = TRUE, margin = 0.1, branch = 0.5,
+     compress = TRUE)
> text(forbes_rpart)
```

FIGURE 11 – Arbre initial, `forbes_rpart`, pour les données Forbes 2000.




```
> layout(matrix(1:2, nc = 1))
> op <- par(mar = c(0, 4, 0, 0) + 0.1)
> plot(forbes_prune, uniform = TRUE, margin = 0.1, branch = 0.5,
+     compress = TRUE)
> text(forbes_prune)
> rn <- rownames(forbes_prune$frame)
> lev <- rn[sort(unique(forbes_prune$where))]
> where <- factor(rn[forbes_prune$where], levels = lev)
> n <- tapply(Forbes2000$profits, where, length)
> boxplot(Forbes2000$profits ~ where, varwidth = TRUE,
+     ylim = range(Forbes2000$profit), pars = list(axes = FALSE),
+     ylab = "Profits en dollars US")
> abline(h = 0, lty = 3)
> axis(2)
> text(1:length(n), max(Forbes2000$profit) * 1.2, paste("n = ",
+     n))
> par(op)
> layout(1)
```

FIGURE 12 – Arbre élagué, `forbes_prune`, pour les données Forbes 2000.