

MAE 5905: Introdução à Ciência de Dados

Pedro A. Morettin

Instituto de Matemática e Estatística
Universidade de São Paulo
pam@ime.usp.br
<http://www.ime.usp.br/~pam>

Aula 11

12 de maio de 2023

Sumário

- 1 Classificação por meio de árvores
- 2 Bagging
- 3 Boosting
- 4 Florestas Aleatórias
- 5 Árvores para regressão

Preliminares I

- Modelos baseados em árvores foram desenvolvidos por Leo Breiman e associados e são bastante utilizados tanto para classificação quanto para previsão.
- Esses modelos são baseados numa segmentação do espaço gerado pelas variáveis explicativas (preditoras) em algumas regiões em que ou a moda (no caso de variáveis respostas categorizadas) ou a média (no caso de variáveis contínuas) é utilizada como predição.
- A definição dessas regiões é baseada em alguma medida de erro de previsão (ou de classificação). Em geral, as árvores são construídas a partir de um conjunto de **observações de treinamento** e testadas em um conjunto de **observações de teste**
- Os modelos de árvores de decisão são conceitualmente e computacionalmente simples e bastante populares em função de sua interpretabilidade, apesar de serem menos precisos que outros modelos mais complexos.

Preliminares II

- Generalizações dos modelos originais, conhecidos como **florestas aleatórias** (*random forests*) costumam apresentar grande precisão, mesmo quando comparados com modelos lineares, porém pecam pela dificuldade de interpretação. A referência básica para esse tópico é Breiman et al. (1984). O texto de Hastie and Tibshirani (1990) também contém resultados sobre esse tópico.
- Para prever o valor de uma variável resposta Y (no caso de variáveis contínuas) ou classificar as observações em uma de suas categorias (no caso de variáveis categorizadas) a partir de um conjunto de variáveis preditoras X_1, \dots, X_p , o algoritmo usado na construção de árvores de decisão consiste essencialmente na determinação das regiões (retângulos mutuamente exclusivos) em que o espaço das variáveis preditoras é particionado. A metodologia desenvolvida em Breiman et al. (1994), e designada **CART** (de *Classification And Regression Trees*) é baseada na seguinte estratégia:
 - (a) Considere uma partição do espaço das variáveis preditoras (conjuntos dos possíveis valores de X_1, \dots, X_p) em M regiões, R_1, \dots, R_M .

Preliminares III

- (b) Para cada observação pertencente a R_j , o preditor (ou categoria) de Y (que designaremos \hat{Y}_{R_j}) será a moda (no caso discreto), a média (no caso contínuo) ou a porcentagem (no caso categorizado) dos valores de Y correspondentes àqueles com valores de X_1, \dots, X_p em R_j .
- Embora a partição do espaço das variáveis preditoras seja arbitrária, usualmente ela é composta por retângulos p -dimensionais que devem ser construídos de modo a minimizar alguma medida de erro de previsão ou de classificação (que explicitaremos posteriormente).
- Como esse procedimento geralmente não é computacionalmente factível dado o número de partições possíveis, mesmo com p moderado, usa-se uma **divisão binária recursiva** (*recursive binary splitting, RBS*) que é uma abordagem **top-down and greedy**, segundo James et al. (2013).
- Dado o vetor de variáveis preditoras $\mathbf{X} = (X_1, \dots, X_p)^\top$, o algoritmo consiste dos passos:
 - (i) Selecione uma variável preditora X_j e um limiar (ou ponto de corte) t , de modo que a divisão do espaço das variáveis preditoras nas regiões $\{\mathbf{X} : X_j < t\}$ e $\{\mathbf{X} : X_j \geq t\}$ corresponda ao menor erro de predição (ou de classificação).

Preliminares IV

- (ii) Para todos os pares (j, t) , considere as regiões

$$R_1(j, t) = \{\mathbf{X} : X_j < t\}, \quad R_2(j, t) = \{\mathbf{X} : X_j \geq t\}$$

e encontre o par (j, s) que minimize o erro de predição (ou de classificação) adotado.

- (iii) Repita o procedimento, agora dividindo uma das duas regiões encontradas, obtendo três regiões; depois divida cada uma dessas três regiões minimizando o erro de predição (ou de classificação);
 - (iv) Continue o processo até que algum critério de parada seja satisfeito.
- Um possível critério de parada consiste na obtenção de um número mínimo fixado de observações em cada região.

Árvores para Classificação

- Quando a variável resposta Y é categorizada, o objetivo é identificar a classe mais provável (**classe modal**) associada aos valores $\mathbf{X} = (X_1, \dots, X_p)^\top$ das variáveis preditoras.
- Neste caso, uma medida de erro de classificação, comumente denominada **taxa de erros de classificação** (**TEC**) é a proporção de observações do conjunto de treinamento que não pertencem à classe modal.
- Outras medidas de erro de classificação, como entropia ou índice de Gini também podem ser usadas.
- Admitamos que a variável resposta tenha K classes e que o espaço de variáveis preditoras seja particionado em M regiões. Designando por \hat{p}_{mk} , a proporção de observações de treinamento da m -ésima região, $m = 1, \dots, M$, pertencentes à k -ésima classe $k = 1, \dots, K$, a taxa de erros de classificação dos elementos pertencentes à m -ésima região é

$$TEC_m = 1 - \max_k (\hat{p}_{mk}).$$

Árvores para Classificação

- Como alternativa para as taxas de erros de classificação, pode-se usar o **índice de Gini** definido como

$$G_m = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

que, essencialmente, corresponde à soma das variâncias das proporções de classificação em cada classe. Quando o valor de \hat{p}_{mk} para um dado nó m estiver próximo de 1 para uma dada categoria k e estiver próximo de zero para as demais categorias, o índice de Gini correspondente estará próximo de zero, indicando que para esse nó, uma das K categorias concentrará uma grande proporção dos elementos do conjunto de dados; poucos deles serão classificadas nas demais $K - 1$ categorias. Quanto mais concentrados em uma categoria forem as classificações em um dado nó, tanto maior será o seu grau de **pureza**.

- Outra medida utilizada com o mesmo propósito e que tem características similares àquelas do coeficiente de Gini é a **entropia cruzada**, definida como

$$ET_m = \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk}).$$

Árvores—Exemplo 1

- Vários pacotes, [tree](#), [partykit](#), [rpart](#), [caret](#) podem ser utilizados com esse propósito. Cada um desses pacotes é regido por parâmetros que controlam diferentes aspectos da construção das árvores. Consultar os manuais correspondentes para nortear uma seleção adequada a problemas específicos.
- **Exemplo 1.** Consideremos novamente os dados analisados no Exemplo 9.2, disponíveis no arquivo [tipofacial](#), extraídos de um estudo cujo objetivo era avaliar se duas ou mais medidas ortodônticas poderiam ser utilizadas para classificar indivíduos segundo o tipo facial (braquicéfalo, mesocéfalo ou dolicocefalo).
Como no Exemplo 9.2, para efeitos didáticos consideramos apenas duas variáveis preditoras, correspondentes a duas distâncias de importância ortodôntica, nomeadamente, a altura facial ([altfac](#)) e a profundidade facial ([proffac](#)).

Árvores-Exemplo 1

- Os comandos do pacote **party** (com os parâmetros *default*) para a construção da árvore de classificação e do gráfico correspondente seguem juntamente com os resultados

```
facetree <- ctree(grupo ~ altfac + proffac, data=face)
facetree
```

Model formula:

```
grupo ~ altfac + proffac
```

Fitted party:

```
[1] root
|   [2] altfac <= -0.1: dolico (n = 31, err = 9.7%)
|   [3] altfac > -0.1
|   |   [4] altfac <= 0.8: meso (n = 28, err = 14.3%)
|   |   [5] altfac > 0.8
|   |   |   [6] proffac <= -0.6: meso (n = 17, err = 41.2%)
|   |   |   [7] proffac > -0.6: braq (n = 25, err = 0.0%)
```

Number of inner nodes: 3

Number of terminal nodes: 4

```
> plot(facetree)
```

Árvores-Exemplo 1

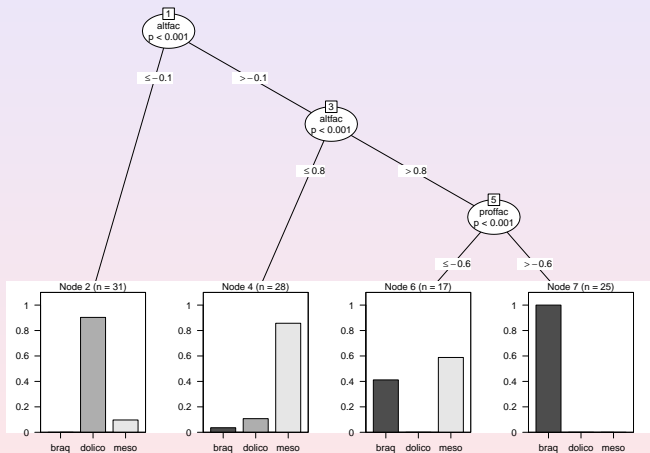


Figura 1: Árvore de decisão para os dados do Exemplo 1.

Árvores—Exemplo 1

- Na Figura 1, os símbolos ovais, que indicam divisões no espaço das variáveis preditoras são chamados de **nós internos** e os retângulos, que indicam as divisões finais são conhecidos por **nós terminais** ou **folhas** da árvore.
- Neste exemplo, temos 3 nós internos e 4 nós terminais. Os segmentos que unem os nós são os **galhos** da árvore.
- As barras em cada nó terminal indicam a frequência relativa com que as observações que satisfazem as restrições definidoras de cada galho são classificadas nas categorias da variável resposta.
- As regiões em que o espaço das variáveis preditoras foi particionado estão indicadas na Figura 2.

Árvores—Exemplo 1

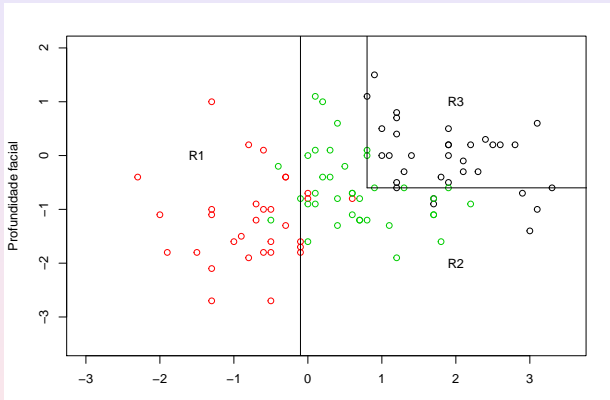


Figura 2: Regiões em que o espaço das variáveis preditoras do Exemplo 1 está particionado (círculos vermelhos = dolichocéfalos, verdes = mesocéfalos, pretos = braquicéfalos).

Árvores—Exemplo 1

- A variável preditora principal e o correspondente ponto de corte que minimiza a taxa de erros de classificação são **altfac** e $t = -0,1$, com $TEC = 9,7\%$.
- Para observações com valores $\text{altfac} \leq -0,1$ (região R_1), classificamos o indivíduo como dolicocéfalo.
- Para valores de $\text{altfac} > -0,1$, a classificação depende do valor de **proffac**. Nesse caso, se altfac estiver entre $-0,1 \leq 0,8$, (região R_2), classificamos o indivíduo como mesocéfalo com $TEC = 14,3\%$.
- Se, por outro lado, $\text{altfac} > 0,8$ e $\text{proffac} \leq -0,6$, também classificamos o indivíduo como mesocéfalo (região R_2), com $TEC = 41,2\%$.
- Agora, se $\text{altfac} > 0,8$ e $\text{proffac} > -0,6$, o indivíduo deve ser classificado como braquicéfalo (região R_3), com $TEC = 0,0\%$.

Árvores-Exemplo 1

- Uma tabela com as classificações originais e preditas por meio da árvore de classificação é obtida por meio do comando `predict`

```
table(predict(facetree), face\grupo)
```

	braq	dolico	meso
braq	25	0	0
dolico	0	28	3
meso	8	3	34

- A acurácia do classificador é de 86% e $TEC = 14\%$.

Árvores—Exemplo 2

- Um dos problemas associados à construção de árvores de decisão está relacionado com o **sobreajuste** (*overfitting*).
- Se não impusermos uma regra de parada para a construção dos nós, o processo é de tal forma flexível que o resultado final pode ter tantos nós terminais quantas forem as observações, gerando uma árvore em que cada observação é classificada perfeitamente.
- Para contornar esse problema, pode-se considerar o procedimento conhecido como **poda**, que engloba técnicas para limitar o número de nós terminais das árvores.
- A ideia que fundamenta essas técnicas está na construção de árvores com menos nós e, conseqüentemente, com menor variância e interpretabilidade. O preço a pagar é um pequeno aumento no viés.

Árvores—Exemplo 2

- **Exemplo 2.** Consideremos agora os dados do arquivo **coronarias** provenientes de um estudo cujo objetivo era avaliar fatores prognósticos de lesão obstrutiva coronariana (L03) com categorias 1 : $\geq 50\%$ ou 0 : $< 50\%$ em 1500 pacientes.
- Embora tenham sido observadas cerca de 70 variáveis preditoras, aqui trabalharemos com SEX0 (0=fem, 1=masc), DIAB (diabetes: 0=não, 1=sim), IMC (índice de massa corpórea), IDADE1 (idade), TRIG (concentração de triglicérides) e GLIC (concentração de glicose).
- Com propósito didático eliminamos casos em que havia dados omissos em alguma dessas variáveis, de forma que 1034 pacientes foram considerados na análise.
- Os comandos do pacote **rpart** para a construção da árvore de classificação por meio de validação cruzada com os resultados correspondentes e o gráfico associado, disposto na Figura 3 seguem:

Árvores—Exemplo 2

```
lesaoobs <- rpart(formula = L03 ~ IDADE1 + SEXO + GLIC + DIAB +  
                  IMC + TRIG, method="class", data = coronarias3,  
xval = 20,minsplitlevel=10, cp=0.005)  
printcp(lesaoobs)
```

Variables actually used in tree construction:

```
[1] GLIC IDADE1 IMC SEXO TRIG
```

Root node error: 331/1034= 0.32012
n= 1034

Árvores—Exemplo 2

	CP	nsplit	rel error	xerror	xstd
1	0.0453172	0	1.00000	1.00000	0.045321
2	0.0392749	3	0.85801	0.97281	0.044986
3	0.0135952	4	0.81873	0.88218	0.043733
4	0.0090634	6	0.79154	0.87915	0.043687
5	0.0075529	7	0.78248	0.88822	0.043823
6	0.0060423	11	0.75227	0.92749	0.044386
7	0.0050000	13	0.74018	0.97885	0.045062

- A função `rpart()` tem um procedimento de VC embutido, ou seja, usa o CTr para construir a árvore e outro (CTeste) para avaliar a taxa de erros de previsão, repetindo o processo várias vezes. Cada linha da tabela representa um nível diferente da árvore. O erro de classificação obtido por validação cruzada tende a aumentar, pelo menos após o nível ótimo.
- A taxa de erro no **nó raiz** (root node error) corresponde à decisão obtida quando todas as observações do conjunto de dados são classificadas na categoria LO3 > 50%.
- O erro relativo (**rel error**) mede o erro relativo de classificação dos dados de treinamento obtido por intermédio da árvore. O termo rotulado **xerror** mede o erro relativo de classificação dos elementos do conjunto teste por intermédio da árvore construída com os dados do CTr, e o correspondente erro padrão é rotulado **xstd**.

Árvores—Exemplo 2

- O produto **root node error** \times **rel error** corresponde ao valor absoluto da taxa de erros obtida no CTr ($0,263=0,320 \times 0,740$ para a árvore com 13 subdivisões). O produto **root node error** \times **xerror** corresponde ao valor absoluto da taxa de erros obtida no CTeste e corresponde a uma medida mais objetiva da acurácia da previsão ($0,313=0,320 \times 0,979$ para a árvore com 13 subdivisões).
- A árvore gerada obtida por intermédio do comando

```
> rpart.plot(lesaoobs, clip.right.labs = TRUE, under = FALSE,  
             extra = 101, type=4)
```

está disposto na Figura 3.

Árvores-Exemplo 2

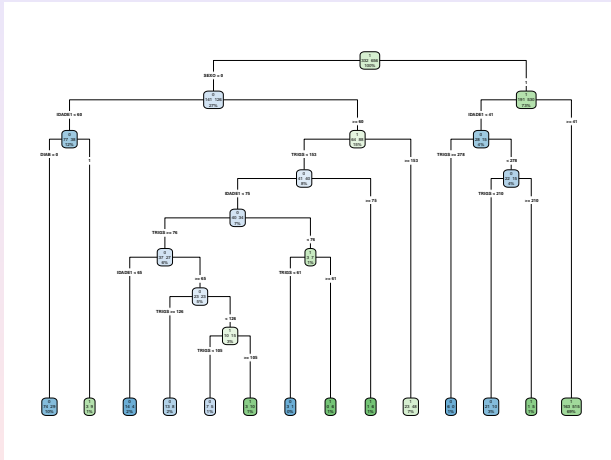


Figura 3: Árvore de decisão para os dados do Exemplo 2.

Árvores—Exemplo 2

- A tabela com as classificações originais e preditas é obtida por meio do comando

```
table(coronarias\L03, predict(lesaoobs, type="class"))  
      0      1  
0 145 186  
1  59 644
```

e indica um erro de classificação de $23,4\% = (186 + 59)/1034$.

- O parâmetro CP (**complexity parameter**) serve para controlar o tamanho da árvore e corresponde ao menor incremento no custo do modelo necessário para a adição de uma nova variável.
- Um gráfico com a variação desse parâmetro com o número de nós, obtido com o comando `plotcp()` pode ser visto na Figura 4.

Árvores-Exemplo 2

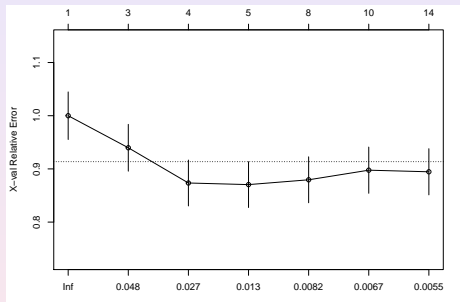


Figura 4: Gráfico CP para o ajuste da árvore aos dados do Exemplo 2.

Nesse gráfico procura-se o nível para o qual o erro relativo obtido por VC é mínimo. Para o exemplo, esse nível é 4, sugerindo que a árvore obtida no exemplo deve ser podada.

Árvores—Exemplo 3

- Vejamos, agora, ver um exemplo usando o pacote [tree](#).
- **Exemplo 3.** Vamos considerar os dados de crianças que foram submetidas a uma cirurgia da coluna para corrigir cifose congênita. Veja Chambers e Hastie (1992). Os dados contêm 81 observações, com as variáveis:

Y : cifose: uma variável qualitativa (atributo), com os valores *ausente* e *presente*, indicando se cifose estava ausente ou presente após a cirurgia;

X_1 : Age, em meses;

X_2 : Number, o número de vértebras envolvidas;

X_3 : Start, o número da primeira vértebra (a partir do topo) operada.

- O sumário do ajuste e a Figura 5 estão ilustradas a seguir.

Árvores—Exemplo 3

Classification tree:

```
tree(formula = Kyphosis ~ Age + Number + Start, data = kyphosis)
```

Number of terminal nodes: 10

Residual mean deviance: 0.5809 = 41.24 / 71

Misclassification error rate: 0.1235 = 10 / 81

- Vemos que a taxa do erro de classificação é 12,35% e o desvio (*deviance*) é definido por

$$-2 \sum_m \sum K n_{mk} \log \hat{p}_{mk},$$

onde n_{mk} é o número de observações no m -ésimo nó terminal que pertence à k -ésima class. Um desvio pequeno indica um bom ajuste aos dados de treinamento. O desvio médio da saída acima é dado pelo desvio dividido por $n - n_0$, n_0 sendo o número de nós terminais, no caso, 10.

- As regiões que determinam a figura são retângulos no espaço tridimensional, logo é complicado, ou não é possível, vê-las em um gráfico.

Árvores-Exemplo 3

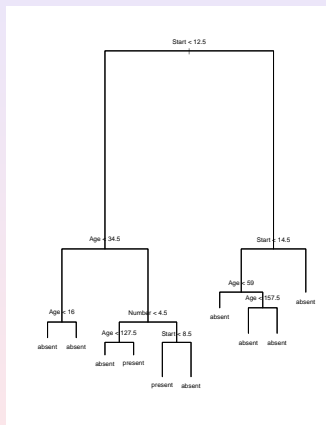


Figura 5: Árvore para o Exemplo 3, com 3 preditores.

Árvores—Exemplo 3

Para poder ver uma partição, vamos considerar somente dois preditores, Start e Age. A árvore correspondente está na Figura 6 e a partição na Figura 7.

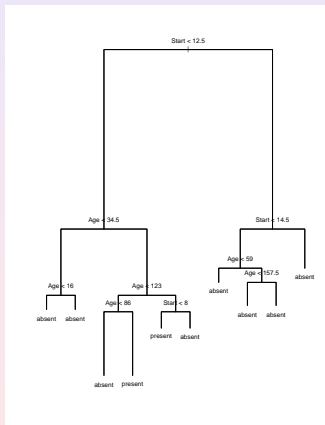


Figura 6: Árvore para o Exemplo 3, com 2 preditores, Start e Age.

Árvores–Exemplo 3

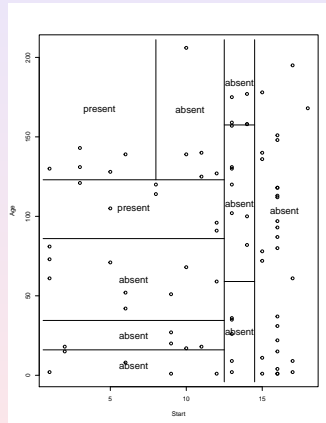


Figura 7: Regiões para o Exemplo 10.3, com 2 preditores.

Árvores—Exemplo 3

Com relação à Figura 26, algumas regiões que podem ser deduzidas são (sendo $\mathbf{X} = (X_1, X_2, X_3)'$):

$R_1 = \{\mathbf{X} : X_1 < 16, X_3 < 12, 5\}$, o valor previsto de Y é ausente;

$R_2 = \{\mathbf{X} : 16 < X_1 < 34, 5, X_3 < 12, 5\}$, o valor previsto de Y é ausente;

$R_3 = \{\mathbf{X} : 34, 5 < X_1 < 127, 5, X_2 < 4, 5, X_3 < 12, 5\}$, o valor previsto de Y é ausente etc.

Poda de uma árvore

- No Exemplo 2 (coronarias), vimos que a árvore deve ser podada, inspecionando o parâmetro de complexidade, CP, que indica valores entre 0,011 e 0,023, com nós entre 5 e 7.
- Uma regra empírica para se efetuar a poda da árvore consiste em escolher a **menor árvore** para a qual o valor de `xerror` é menor do que a soma do menor valor observado de `xerror` com seu erro padrão `xstd`. No exemplo em estudo, o menor valor de `xerror` é 0,87915 e o de seu erro padrão `xstd` é 0,043687 de maneira que a árvore a ser construída por meio de poda deverá ser a menor para a qual o valor de `xerror` seja menor que 0,922837 ($= 0,87915 + 0,043687$), ou seja a árvore com 4 subdivisões e 5 nós terminais.
- A poda juntamente com o gráfico da árvore podada e a tabela com os correspondentes valores preditos podem ser obtidos com os comandos a seguir:

Poda de uma árvore

```
lesaoobspoda <- prune(lesaoobs, cp = 0.015 , "CP", minsplit=20, xval=25)
rpart.plot(lesaoobspoda, clip.right.labs = TRUE, under = FALSE,
           extra = 101, type=4)
rpart.rules(lesaoobspoda, cover = TRUE)
```

L03	cover
0.33 when SEX0 is 0 & IDADE1 < 63 & GLIC < 134	13%
0.35 when SEX0 is 1 & IDADE1 < 41	4%
0.59 when SEX0 is 0 & IDADE1 >= 63 & GLIC < 134	10%
0.77 when SEX0 is 1 & IDADE1 >= 41	69%
0.85 when SEX0 is 0 & GLIC >= 134	4%

Poda de uma árvore

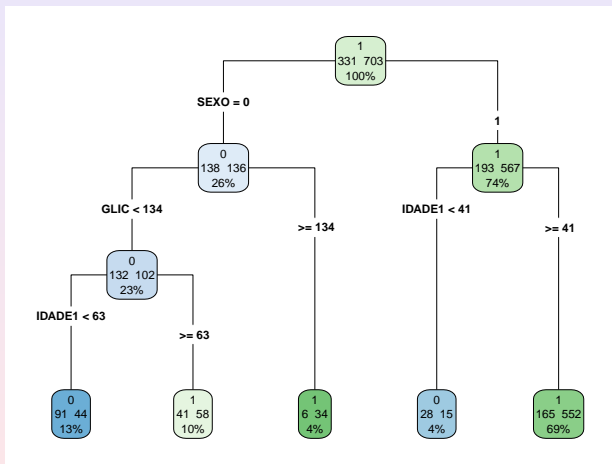


Figura 8: Árvore (podada) ajustada aos dados do Exemplo 2.

Poda de uma árvore

- O número indicado na parte superior de cada nó da Figura 8 indica a classe majoritária, na qual são classificadas os elementos do conjunto de treinamento; o valor à esquerda no centro do nó representa a frequência desses elementos pertencentes à classe $LO3 = 0$ e o valor à direita corresponde à frequência daqueles pertencentes à classe $LO3 = 1$. Na última linha aparece a porcentagem de elementos correspondentes a cada nó terminal.
- A tabela de classificação para o conjunto de dados original (treinamento + validação) obtida a partir da árvore podada é

	0	1
0	119	212
1	59	644

Nessa tabela, as linhas indicam a classificação real e as colunas informam a classificação predita pela árvore podada. O erro de classificação, 26,2%, é ligeiramente maior que o erro obtido com a árvore original, bem mais complexa.

Bagging

- Usualmente, árvores de decisão produzem resultados com grande variância, ou seja, dependendo de como o conjunto de dados é subdividido em conjuntos de treinamento e de teste, as árvores produzidas podem ser diferentes. As técnicas que descreveremos a seguir têm a finalidade de reduzir essa variância.
- A técnica de **agregação bootstrap** (*bootstrap aggregating*) ou, simplesmente **bagging**, é um método para gerar múltiplas versões de um previsor (ou classificador) a partir de vários conjuntos de treinamento e, com base nessas versões, construir um previsor (ou classificador) agregado.
- O ideal seria ter vários conjuntos de treinamento, construir previsores e tomar uma média (no caso de regressão) dos previsores. Todavia, na prática, isso não acontece. Uma maneira de prosseguir, é usar réplicas bootstrap do conjunto de treinamento.

Bagging

- Suponha que tenhamos o conjunto de treinamento $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ e considere o caso de y quantitativa (regressão). Considere B réplicas bootstrap desse conjunto e para cada réplica b , obtemos o previsor de y , $\hat{f}^{*b}(\mathbf{x})$ e construímos o previsor

$$\hat{f}_{\text{bag}} = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(\mathbf{x}). \quad (1)$$

- No caso de classificação, para uma observação teste, considere a classe prevista para cada uma das B árvores e, então, tome como previsor a classe de maior ocorrência comum entre os B previsores (voto majoritário). Especificamente,

$$\hat{c}_{\text{bag}}(\mathbf{x}) = \operatorname{argmax}_k [\#\{(b | \hat{c}^b(\mathbf{x}) = k)\}]$$

em que $\#\{A\}$ denota a cardinalidade do conjunto A .

- O número de réplicas bootstrap sugerido por Breiman (1996) é cerca de 25. Para detalhes sobre bootstrap veja Efron e Tibshirani (1993) e Notas de Capítulo.

Bagging-Exemplo 4

- **Exemplo 4.** A técnica bagging pode ser aplicada aos dados do Exemplo 2 por meio dos comandos

```
> set.seed(054)
>
> # train bagged model

> lesaoobsbag <- bagging(
+   formula = L03 ~ SEX0 + IDADE1+ GLIC,
+   data = coronarias3,
+   nbagg = 200,
+   coob = TRUE,
+   control = rpart.control(minsplit = 20, cp = 0.015)
+ )
>
> lesaoobspred <- predict(lesaoobsbag, coronarias3)
> table(coronarias3$L03, predict(lesaoobsbag, type="class"))
```

```
      0      1
0 117 214
1  78 625
```

- Variando a semente do processo aleatório, as taxas de erros de classificação giram em torno de 27% a 28%.

Bagging-Exemplo 4

Quatro das 200 árvores obtidas em cada réplica bootstrap podem ser obtidas por meio dos comandos a seguir e estão representadas na Figura 9.

```
as.data.frame(coronarias4)
clr12 = c("#8dd3c7", "#ffffb3", "#bebada", "#fb8072")
n = nrow(coronarias4)
par(mfrow=c(2,2))
sed=c(1,10,22,345)
for(i in 1:4){
  set.seed(sed[i])
  idx = sample(1:100, size=n, replace=TRUE)
  cart = rpart(L03 ~ DIAB + IDADE1 + SEX0, data=coronarias4[idx,], mod
  prp(cart, type=1, extra=1, box.col=clr12[i])
}
```

Bagging-Exemplo 4

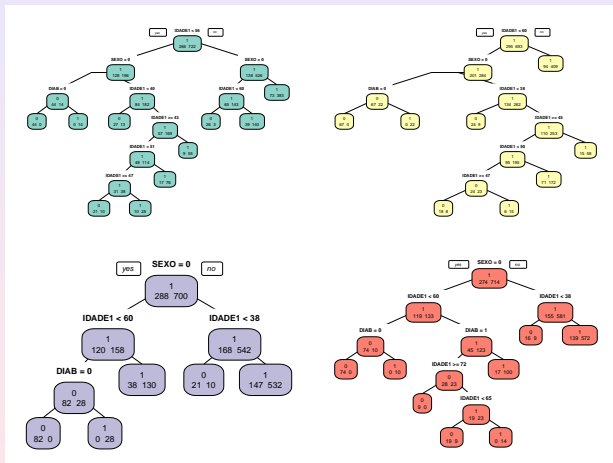


Figura 9: Exemplos de Árvores obtidas por bagging para os dados do Exemplo 2.

Boosting I

- 1 O objetivo do procedimento **boosting** é reduzir o viés e a variância em modelos utilizados para aprendizado supervisionado.
- 2 A ideia básica é considerar um conjunto de **previsores (classificadores) fracos** de modo a obter um **previsor (classificador) forte**.
- 3 Classificadores fracos têm taxas de erro de classificação altas. No caso binário, por exemplo, isso corresponde a uma taxa próxima de 0,50, que seria obtida com uma decisão baseada num lançamento de moeda. Um classificador forte, por outro lado, tem uma taxa de erro de classificação baixa.
- 4 Diferentemente da técnica bagging, em que B árvores são geradas independentemente por meio de bootstrap, com cada observação tendo a mesma probabilidade de ser selecionada em cada um dos conjuntos de treinamento, no procedimento boosting, as B árvores são geradas **sequencialmente** a partir de um único conjunto de treinamento, com probabilidades de seleção (pesos) diferentes atribuídos às observações.

Boosting II

- 5 Observações mal classificadas em uma árvore recebem pesos maiores para seleção na árvore subsequente (obtida do mesmo conjunto de treinamento), com a finalidade de dirigir a atenção aos casos em que a classificação é mais difícil.
- 6 Em ambos os casos, o classificador final é obtido por meio da aplicação dos B classificadores fracos gerados com as diferentes árvores, por meio do voto majoritário. Além dos pesos atribuídos às observações no processo de geração dos classificadores fracos, o procedimento boosting atribui pesos a cada um deles, em função das taxas de erros de classificação de cada um.
- 7 Essencialmente, o classificador forte pode ser expresso como

$$\hat{c}_{\text{boost}}(\mathbf{x}) = \sum_{b=1}^B \hat{c}^b(\mathbf{x}) w(b) \quad (2)$$

em que $w(b)$ é o peso atribuído ao classificador $\hat{c}^b(\mathbf{x})$.

Boosting III

- 8 Se por um lado, o procedimento bagging raramente reduz o viés quando comparado com aquele obtido com uma única árvore de decisão, por outro, ele tem a característica de evitar o sobreajuste. Essas características são invertidas com o procedimento boosting.
- 9 Existem vários algoritmos para a implementação de boosting. O mais usado é o algoritmo conhecido como **AdaBoost** (de *adaptive boosting*), desenvolvido por Freund e Schapire (1997). Dada a dificuldade do processo de otimização de (2), esse algoritmo considera um processo iterativo de otimização que produz bons resultados embora não sejam ótimos.
- 10 A ideia é adicionar o melhor classificador fraco numa determinada iteração ao classificador fraco obtido na iteração anterior, ou seja, considerar o processo

$$\hat{c}_{boost}^b(\mathbf{x}) = \hat{c}_{boost}^{b-1}(\mathbf{x}) + \hat{c}^b(\mathbf{x})w(b)$$

em que $\hat{c}_{boost}^b(\mathbf{x})$ é o classificador com o melhor incremento no desempenho relativamente ao classificador $\hat{c}_{boost}^{b-1}(\mathbf{x})$.

Boosting IV

- 11 Nesse contexto, a escolha de $[\hat{c}^b(\mathbf{x}), w(b)]$ deve satisfazer

$$[\hat{c}^b(\mathbf{x}), w(b)] = \operatorname{argmin}_{[c^b(\mathbf{x}), w(b)]} \{E[\hat{c}_{boost}^{b-1}(\mathbf{x}) + \hat{c}^b(\mathbf{x})w(b)]\}$$

em que $E[c^b]$ denota o erro de classificação do classificador c^b .

- 12 Consideremos, por exemplo, um problema de classificação binária baseado num conjunto de treinamento com N observações. No algoritmo **AdaBoost**, o classificador sempre parte de um único nó (conhecido como **stump**), em que cada observação tem peso $1/N$.
- 13 O ajuste por meio desse algoritmo é realizado por meio dos seguintes passos:
- 1) Ajuste o melhor classificador (fraco) com os pesos atuais e repita os passos seguintes para os $B - 1$ classificadores subsequentes.
 - 2) Calcule o valor do peso a ser atribuído ao classificador fraco corrente a partir de alguma métrica que indique quanto esse classificador contribui para o classificador forte corrente.
 - 3) Atualize o classificador forte adicionando o classificador fraco multiplicado pelo peso calculado no passo anterior.

Boosting V

- 4) Com esse classificador forte, calcule os pesos atribuídos às observações de forma a indicar quais devem ser o foco da próxima iteração (pesos atribuídos às observações mal classificadas devem ser maiores que pesos atribuídos a observações bem classificadas).

14 Os algoritmos para implementação de boosting têm 3 parâmetros:

(i) o número de árvores, B , que pode ser determinado por validação cruzada;

(ii) **parâmetro de encolhimento** (*shrinkage*), $\lambda > 0$, pequeno, da ordem de 0,01 ou 0,001, que controla a velocidade do aprendizado;

(iii) o número de divisões em cada árvore, d ; como vimos, o AdaBoost, usa $d = 1$ e, em geral, esse valor funciona bem.

15 O pacote **gbm** implementa o boosting e o pacote **adabag** implementa o algoritmo AdaBoost.

Boosting-Exemplo 5

Exemplo 5. Os dados do CD-esteira contêm informações de 16 variáveis, sendo 4 qualitativas (Etiologia, Sexo, Classe funcional NYHA e Classe funcional WEBER) além de 13 variáveis quantitativas [Idade, Altura, Peso, Superfície corporal, Índice de massa corpórea (IMC), Carga, Frequência cardíaca (FC), VO2 RER, VO2/FC, VE/VO2, VE/VCO2] em diversas fases (REP, LAN, PCR e PICO) de avaliação de um teste de esforço realizado em esteira ergométrica. A descrição completa das variáveis está disponível no arquivo dos dados. Neste exemplo vamos usar as variáveis Carga, Idade, IMC e Peso na fase LAN (limiar anaeróbico) como preditores e VO2 (consumo de oxigênio, como resposta. Para efeito do exemplo, essas variáveis, com as respectivas unidades de medida, serão denotadas como:

Y : consumo de oxigênio (VO2), em $mL/kg/min$;

X_1 : carga na esteira, em W (Watts);

X_2 : índice de massa corpórea (IMC), em kg/m^2 .

X_3 : Idade, em anos;

X_4 : Peso, em kg .

Boosting-Exemplo 5

Vamos usar o pacote gbm. Com o comando `summary()` obtemos a influência relativa dos preditores e a `figure` respectiva, Figura 10.

```
summary(boost.esteira)
var      rel.inf
CARGA    43.46858
IMC      24.22825
Idade    16.84917
Peso     15.45401
```

Vemos que os preditores CARGA e IMC são os mais importantes. Podemos produzir gráficos parciais de dependência para essas duas variáveis (Figura 11). Vemos que o consumo de oxigênio cresce com a CARGA e diminui com o IMC.

Boosting-Exemplo 5

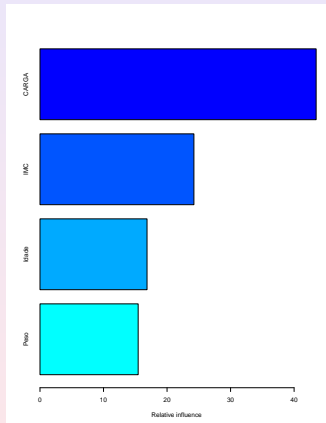


Figura 10: Influência relativa para os preditores do Exemplo 5.

Boosting-Exemplo 5

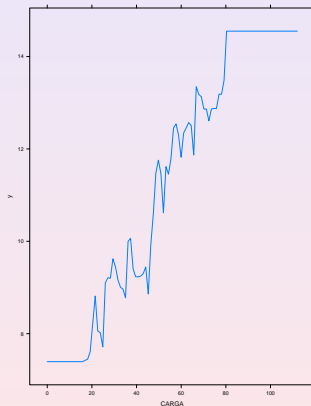


Figura 11: Consumo de oxigênio versus CARGA .

Boosting-Exemplo 5

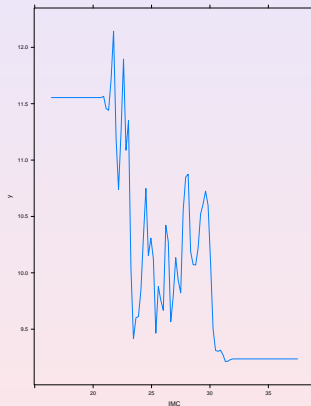


Figura 12: Consumo de oxigênio versus IMC .

Boosting-Exemplo 5

Agora usamos o modelo via *boosting* para prever VO2 para o conjunto teste.

```
yhat.boost=predict(boost.esteira,newdata=esteira2[-train,],n.trees=5000  
mean((yhat.boost-boston.test)^2)  
[1] 2.063152e-05
```

Vemos que o EQM obtido via *boosting* é consideravelmente menor do que aquele obtido via *bagging*.

Florestas

- Bagging e florestas aleatórias têm o mesmo objetivo: diminuir a variância e o viés de procedimentos baseados em árvores de decisão.
- Enquanto os dois primeiros enfoques são baseados em um conjunto de B árvores utilizando o mesmo conjunto de p variáveis preditoras em cada um deles, o enfoque conhecido por **florestas aleatórias** utiliza diferentes conjuntos das variáveis preditoras na construção de cada árvore.
- Na construção de um novo nó, em vez de escolher a melhor variável dentre as p disponíveis no conjunto de treinamento, o algoritmo de florestas aleatórias seleciona a melhor delas dentre um conjunto de $m < P$ selecionadas ao acaso. Usualmente, escolhe-se $m \approx \sqrt{p}$.

Florestas

- Formalmente, para cada árvore j , um vetor aleatório θ_j é gerado, independentemente de vetores prévios $\theta_1, \dots, \theta_{j-1}$, mas com a mesma distribuição. A árvore usando o conjunto de treinamento e θ_j resulta num classificador $\hat{f}_j(\mathbf{x}, \theta_j)$. Cada árvore vota na classe mais popular para o vetor \mathbf{x} .
- A acurácia das árvores aleatórias é tão boa quanto a do *AdaBoost* e, às vezes, melhor. O resultado obtido por intermédio do algoritmo de árvores aleatórias em geral é mais robusto com relação a valores atípicos e ruído além de ser mais rápido do que bagging e boosting.
- O pacote [randomForest](#) do R implementa florestas. Nesse caso, o número de preditores tem que ser menor do que p .

Florestas-Exemplo 6

- Exemplo 6. Vamos usar o conjunto de dados [esteira2](#) e as variáveis CARGA e IMC para crescer uma floresta. Obtemos o sumário abaixo:

Call:

```
randomForest(formula = VO2 ~ Idade + CARGA + IMC + Peso,  
              data = esteira2, mtry = 2, importance = TRUE,  
              subset = train)
```

Type of random forest: regression

Number of trees: 500

No. of variables tried at each split: 2

Mean of squared residuals: 4.164505

Percentage Var explained: 52.23

- Obtendo-se previsões para o conjunto teste, temos o sumário abaixo:

```
yhat.rf=predict(rf.esteira,newdata=esteira2[-train,])  
mean((yhat.rf-esteira.test)^2)  
[1] 3.713565
```

- O EQM de previsão via floresta é menor do que aquele via *bagging*, mas maior do que o via *boosting*.

Florestas-Exemplo 6

Um sumário da importância de cada preditor é dado abaixo:

```
summary(boost.esteira)
var   rel.inf
CARGA 43.46858
IMC    24.22825
Idade  16.84917
Peso   15.45401
```

e um gráfico está na Figura 13. Novamente, as variáveis CARGA e IMC são as mais importantes.

Concluindo, sobre os três algoritmos, *bagging*, floresta e *boosting*, o último é o que apresentou o maior poder preditivo.

Florestas-Exemplo 6

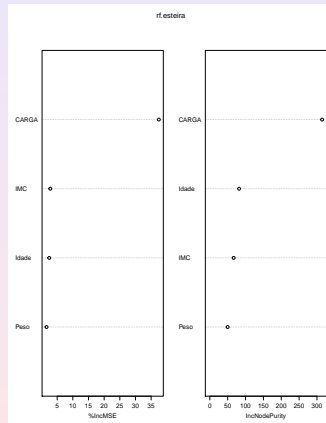


Figura 13: Importância relativa os preditores do Exemplo 6.

Árvores para regressão

- Consideremos uma situação com variáveis preditoras X_1, \dots, X_p e uma variável resposta quantitativa Y . Quando a relação entre variáveis resposta e preditoras for compatível com um modelo linear (no caso de uma relação polinomial, por exemplo), o uso de regressão linear é conveniente e obtemos modelos com interpretabilidade e poder preditivo satisfatórios.
- Quando essa condição não for satisfeita (no caso de modelos não lineares, por exemplo), o uso de árvores pode ser mais apropriado. Além disso, algumas das variáveis preditoras podem ser qualitativas e nesse caso não é necessário transformá-las em **variáveis fictícias** (*dummy variables*) como no caso de modelos de regressão usuais.
- A ideia subjacente é similar àquela empregada em modelos de classificação: subdividir o espaço gerado pelas variáveis explicativas em várias regiões e adotar como previsores, as respostas médias em cada região. As regiões são selecionadas de forma a produzir o menor **erro quadrático médio** ou o menor **coeficiente de determinação**.
- A construção de árvores de regressão pode ser concretizada por meio dos pacotes `tree` e `rpart` entre outros. Ilustraremos a construção de uma árvore para regressão por meio de um exemplo.

Árvores para regressão - Exemplo 7

- Os dados do arquivo **antracose** foram extraídos de um estudo cuja finalidade era avaliar o efeito da idade (*idade*), tempo vivendo em São Paulo (*tmunic*), horas diárias em trânsito (*htransp*), carga tabágica (*cargatabag*), classificação sócio-econômica (*ses*), densidade de tráfego na região onde o indivíduo morou (*densid*) e distância mínima entre a residência a vias com alta intensidade de tráfego (*distmin*) num índice de antracose (*antracose*) que é uma medida de fuligem (*black carbon*) depositada no pulmão.
- Como esse índice varia entre 0 e 1, consideramos

$$\logrc = \log[\text{índice de antracose}/(1 - \text{índice de antracose})]$$

como variável resposta.

- Inicialmente, construímos uma árvore de regressão para os dados por meio de validação cruzada. Os comandos do pacote *rpart* para gerar a árvore apresentada na Figura 14 são dados a seguir.

Árvores para regressão - Exemplo

```
pulmaotree2 <- rpart(formula = logrc ~ idade + tmunic + htransp +
  cargatabag + ses + densid + distmin, data=pulmao,
  method="anova", xval = 30, cp=0.010)
rpart.plot(pulmaotree2, clip.right.labs = TRUE, under = FALSE,
  extra = 101, type=4)
```

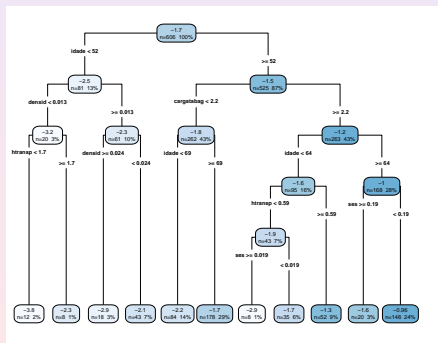


Figura 14: Árvore de decisão para os dados do Exemplo 7.

Árvores para regressão - Exemplo 7

- Os valores apresentados na parte superior de cada nó são as previsões associadas às regiões correspondentes. Na parte inferior encontram-se o número e a porcentagem de elementos incluídos em cada região.
- Para evitar um possível sobreajuste da árvore proposta, convém avaliar o efeito de seu tamanho segundo algum critério. No pacote `rpart` esse critério é baseado no parâmetro de complexidade (CP) que está relacionado com o número de nós terminais e na relação $1 - R^2$ em que R^2 tem a mesma interpretação do coeficiente de determinação utilizado em modelos lineares.
- Com essa finalidade, podemos utilizar o comando `rsq.rpart(pulmaotree2)`, que gera a tabela com os valores de CP e os gráficos apresentados na Figura 15.

Árvores para regressão - Exemplo 7

Variables actually used in tree construction:

[1] cargatabag densid htransp idade ses

Root node error: 765.87/606 = 1.2638

n= 606

	CP	nsplit	rel error	xerror	xstd
1	0.087230	0	1.00000	1.00279	0.077419
2	0.062020	1	0.91277	0.96678	0.076624
3	0.025220	2	0.85075	0.91078	0.072406
4	0.024106	3	0.82553	0.91508	0.073489
5	0.015890	4	0.80142	0.85814	0.069326
6	0.014569	5	0.78553	0.87882	0.070312
7	0.011698	6	0.77097	0.89676	0.070309
8	0.011667	7	0.75927	0.90730	0.069025
9	0.011347	8	0.74760	0.91268	0.069066
10	0.010289	9	0.73625	0.91536	0.069189
11	0.010000	10	0.72596	0.91250	0.069202

Árvores para regressão - Exemplo 7

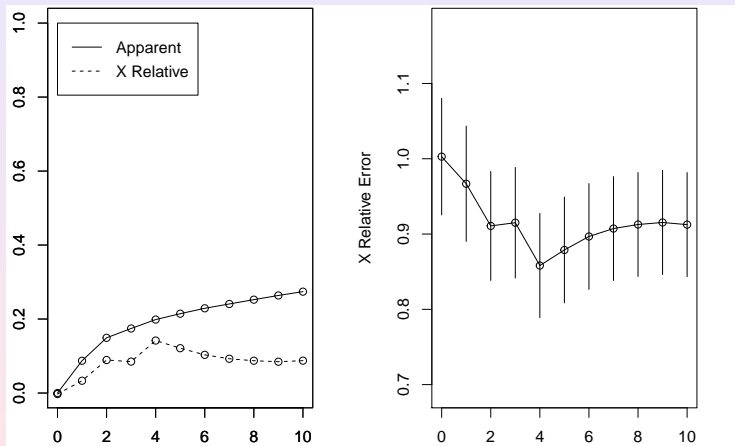


Figura 15: Efeito do número de divisões para a árvore ajustada aos dados do Exemplo 7.

Árvores para regressão - Exemplo 7

Utilizando o mesmo critério aplicado no caso de classificação, a sugestão é podar a árvore, fixando o número de subdivisões em 4, correspondendo a 5 nós terminais. O gráfico da árvore podada (disposto na Figura 16), além da regras de partição do espaço das variáveis explicativas podem ser obtidos com os comandos

```
cpmin <- pulmaotree2$cptable[which.min(pulmaotree2$cptable[, "xerror"]),
                                "CP"]
pulmaotree2poda <- prune(pulmaotree2, cp = cpmin)
rpart.plot(pulmaotree2poda, clip.right.labs = TRUE, under = FALSE,
           extra = 101, type=4)
pulmaotree2poda$cptable
```

	CP	nsplit	rel error	xerror	xstd
1	0.08722980	0	1.0000000	1.0027880	0.07741860
2	0.06201953	1	0.9127702	0.9667786	0.07662398
3	0.02521977	2	0.8507507	0.9107847	0.07240566
4	0.02410635	3	0.8255309	0.9150809	0.07348898
5	0.01588985	4	0.8014246	0.8581417	0.06932582

```
> rpart.rules(pulmaotree2poda, cover = TRUE)
logrc
```

	cover
-2.5 when idade < 52	13%
-2.2 when idade is 52 to 69 & cargatabag < 2.2	14%
-1.7 when idade >= 69 & cargatabag < 2.2	29%
-1.6 when idade is 52 to 64 & cargatabag >= 2.2	16%
-1.0 when idade >= 64 & cargatabag >= 2.2	28%

Árvores para regressão - Exemplo 7

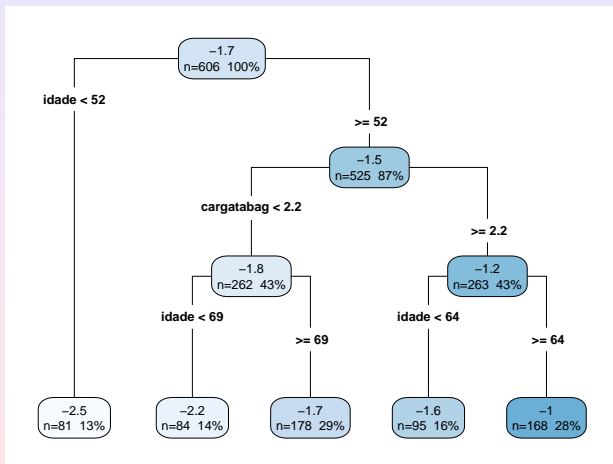


Figura 16: Árvore podada ajustada aos dados do Exemplo.

Árvores para regressão - Exemplo 7

Valores preditos para o conjunto de dados com o respectivo *RMSE* são obtidos por meio dos comandos

```
rmse = function(actual, predicted) {  
  sqrt(mean((actual - predicted)^2))  
}  
rmse(predpulmatree2poda, pulmao$logrc)  
[1] 1.006402
```

Convém notar que embora a utilização de árvores de decisão gere um modelo bem mais simples do que aquele obtido por meio de análise regressão, os objetivos são bem diferentes.

Nesse contexto, o modelo baseado em árvores deve ser utilizado apenas quando o objetivo é fazer previsões, pois pode deixar de incluir variáveis importantes para propósitos inferenciais. No Exemplo, uma das variáveis mais importantes para entender o processo de deposição de fuligem no pulmão é o número de horas gastas em trânsito. Essa variável foi incluída significativamente no ajuste do modelo de regressão mas não o foi no modelo baseado em árvores.

Há um algoritmo **boosting** aplicável na construção de árvores para regressão. Veja Hastie et al. (2017, cap. 10) para detalhes.

Referências

Breiman, L. (2001). Random forests. *Machine Learning*, **45**, 5–32.

Chambers, J. M and Hastie, T. J. (1992:). *Statistical Models in S*. California: Wadsworth and Brooks/Cole.

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized Additive Models*. London: Chapman and Hall.

James, G., Witten, D., Hastie, T. and Tibshirani, R. (2017). *Introduction to Statistical Learning*. Springer.

Morettin, P. A. e Singer, J. M. (2022). *Estatística e Ciência de Dados*. LTC, Rio de Janeiro.