Métodos baseados em árvores

8.1 Introdução a árvores de decisões

Árvores de decisão podem ser aplicadas a ambos os problemas de regressão e classificação. Primeiramente consideraremos problemas de regressão, e então partiremos para os de classificação.

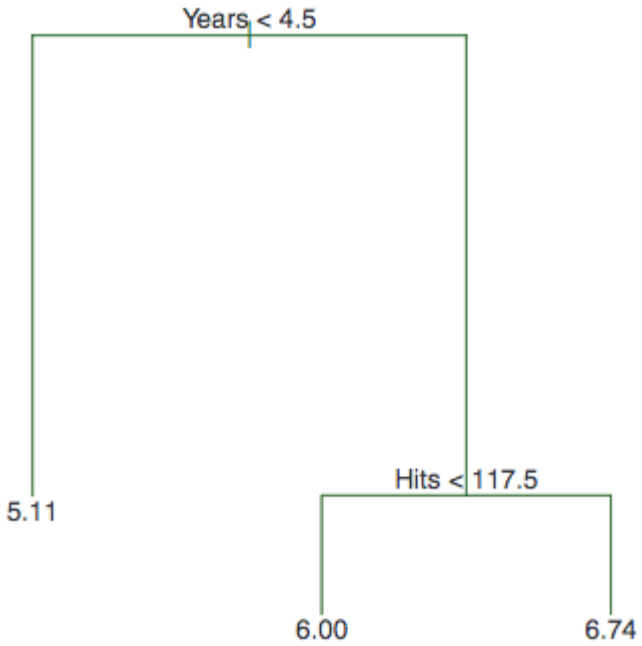


Figura 8.1.
Para os dados $Rebatedores$, uma árvore de regressão para a predição do salário registrado de um jogador de baseball, baseado no número de anos que ele jogou em ligas maiores e em seu número de acertos no ano anterior. Em um dado nó interno, a legenda (da forma X_{j} < t_{k}) indica o ramo do lado esquerdo emanando desta divisão, e o ramo direito corresponde a X_{j} $≥$ t_{k}. Por exemplo, a divisão no topo da árvore resulta em dois grandes ramos. O ramo esquerdo corresponde a $Anos$ < 4,5, e o ramo esquerdo corresponde a $Anos$ ≥ 4,5. A árvore possui dois nós internos e três nós terminais, ou folhas. O número em cada folha é a média da resposta para as observações que caem na mesma.

8.1.1. Árvores de regressão

Loading [MathJax]/extensions/Safe.js

Para estudarmos $árvores$ de $regressão$, começaremos com um simples exemplo.

Predizendo o salário de jogadores de baseball usando árvores de regressão

Usaremos o conjunto de dados $Rebatedores$ para prever o salário de um jogador de baseball baseado em $Anos$ (o número de anos em que um jogador jogou nas maiores ligas) e $Acertos$ (o número de acertos que ele fez no ano anterior). Primeiramente, removemos observações que faltam nos valores de $Salário$, e fazemos uma transformação logarítmica em $Salário$ para que sua distribuição possua uma típica forma de sino. (Relembre que $Salário$ é medido em milhares de dólares).

A figura 8.1. mostra um ajuste com árvore de regressão para estes dados, que consiste de uma série de regras de divisão, iniciando no topo da árvore. A divisão de cima atribui observações ao ramo esquerdo de $Anos$ < 4.5. O salário previsto para estes jogadores é dado pela média do valor de resposta para os jogadores no conjunto de dados com $Anos$ < 4.5. Para tais jogadores, a média do salário logarítmico é 5,107, e então temos uma predição de $e^{5,107 }$ milhares de dólares, por exemplo, 165.174 dólares, para estes jogadores. Jogadores com $Anos$ ≥ 4,5 são atribuídos ao ramo direito, e posteriormente este grupo é subdividido em $Acertos$.

Em geral, a árvore estratifica/segmenta os jogadores em três regiões do espaço preditor: jogadores que jogaram por 4 ou menos anos, jogadores que jogaram por cinco ou mais anos e jogadores que fizeram no mínimo 118 acertos no ano anterior. Estas regiões podem ser escritas como R_{1} = {X| $Anos$ < 4,5}, R_{2} = {X | $Anos$ ≥ 4,5, $Acertos$ < 117,5} e R_{3} = {X | $Anos$ ≥ 4,5, $Acertos$ ≥ 117,5}. A figura 8.2. ilustra as regiões como uma função de $Anos$ e $Acertos$. Os salários previstos para estes três grupos são 1000 $.$ $e^{5,107}$ = 165.174, 1000 $.$ $e^{5,999}$ = 402.834 e 1000 $.$ $e^{6,740}$ = 845.346, respectivamente.

Loading [MathJax]/extensions/Safe.js

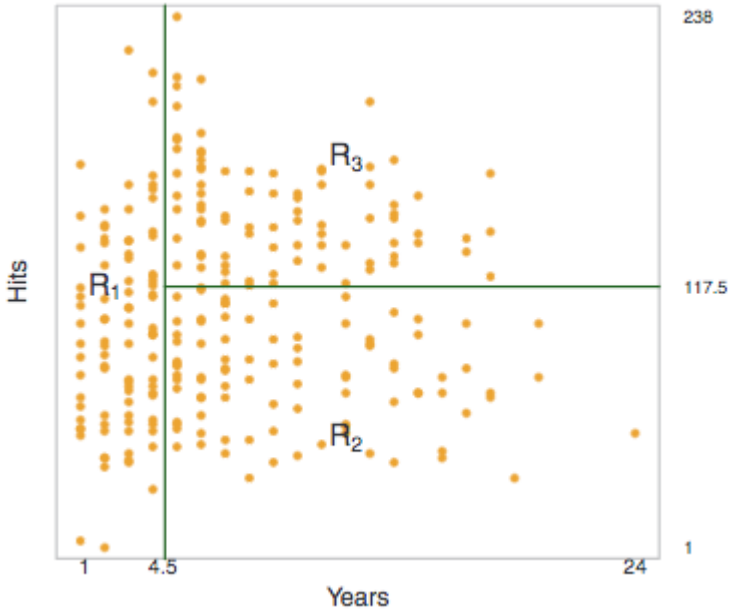


Figura 8.2.
As partições das regiões da árvore para o conjunto de dados $Rebatedores$ da árvore de regressão ilustrada na Figura 8.1.

Mantendo a analogia da $árvore$, as regiões R_{1}, R_{2} e R_{3} são conhecidas como $nós$ $terminais$ ou $folhas$ da árvore. Como é o caso para a figura 8.1, árvores de decisões são tipicamentes desenhadas de $cabeça$ $para$ $baixo$, no sentido de que as folhas estão nas partes mais inferiores da árvore. Na figura 8.1, os dois nós internos são indicados pelos textos $Anos < 4.5$ e $Acertos$ < 117,5. Nos referimos aos segmentos das árvores que conectam os nós como $ramos$.

Poderíamos interpretar a árvore de regressão mostrada na figura 8.1. da seguinte forma: $Anos$ é o fator mais importante na determinação do $Salário$, e jogadores com menos experiência ganham menos que jogadores mais experientes. Dado que um jogador é menos experiente, seu número de acertos no ano anterior parece ter um papel pequeno em seu salário. Mas, entre jogadores que estiveram jogando em ligas maiores por 5 ou mais anos, o número de acertos feito no ano anterior de fato afeta o salário, e jogadores que obtiveram mais acertos no último ano tendem a ter salários mais altos. A árvore de regressão mostrada na Figura 8.1 é provavelmente uma grande simplificação da verdadeira relação entre $Acertos$, $Anos$, e $Salário$. No entanto, ela possui vantagens sobre outros tipos de modelo de regressão: é mais facilmente interpretável e possui uma boa representação gráfica.

Predição pela estratificação do espaço de recurso

Loading [MathJax]/extensions/Safe.js

Discutiremos agora o processo de construção de uma árvore de regressão. De grosso modo, há duas etapas:

1. *Dividimos o espaço de recurso - isto é, o conjunto dos valores possíveis para X_{1},X_{2},...,X_{p} - em J regiões distintas e não-sobrepostas, R_{1}, R_{2},...,R_{J}.*

2. *Para toda observação que cai na região R_{j}, fazemos a mesma predição, a qual é simplesmente a média dos valores de resposta para as observações de treino em R_{j}.*

Por exemplo, suponha que no passo 1 obtenhamos duas regiões, R_{1} e R_{2}, e que a média da resposta das observações de treino na primeira região seja 10, enquanto a média da resposta das observações de treino na segunda região seja 20. Então, para uma dada observação X = x, se x $∈$ R_{1}, iremos predizer um valor 10, e se x $∈$ R_{2} iremos predizer um valor 20.

Agora elaboraremos o passo 1. Como nós construímos as regiões R_{1},...,R_{J}? Em teoria, as regiões poderiam ter qualquer forma. No entanto, escolhemos dividir o espaço preditor em retângulos de altas dimensões, ou $caixas$, por simplicidade e para facilitar a interpretação dos modelos preditivos resultantes. O objetivo é encontrar caixas R_{1},...,R_{j} que minimizem o RSS, dado por:

$$\sum_{j=1}^{J}\sum_{i\in R_j}(y_i-\hat{y}_{R_j})^2, \qquad (8.1)$$

onde $\hat{y}$$_{R_{j}}$ é a média da resposta para as observações de treino até a j-ésima caixa. Infelizmente, é computacionalmente inviável considerar toda partição possível do espaço de recurso em J caixas. Por esta razão, utilizamos uma abordagem mais ganaciosa e $top-down$, conhecida como $divisão$ $binária$ $recursiva$. A abordagem é $top-down$ porque começa no topo da árvore (neste ponto todas as observações pertencem a uma única região) e então sucessivamente divide o espaço de recurso; cada divisão é indicada por dois novos ramos abaixo na árvore, e é gananciosa porque, em cada etapa do processo de construção da árvore, a $melhor$ divisão é feita naquela etapa em particular, em vez de ir adiante e escolher uma divisão que levará a uma melhor árvore em uma etapa futura.

Loading [MathJax]/extensions/Safe.js

Para a realização da divisão binária recursiva, primeiro selecionamos o preditor X_{j} e o ponto de corte s tal que a divisão do espaço de recurso nas regiões {X|X_{j} < s} e {X|X_{j} ≥ s} leva a maior redução possível no RSS (*soma dos quadrados dos erros residuais*). (A notação {X|X_{j} < s} significa *a região do espaço de recurso na qual X_{j} possui um valor menor que s.*) Isto é, consideramos todos os preditores X_{1},...,X_{p} e todos os valores possíveis do ponto de corte s para cada um dos preditores, e então escolhemos o preditor e escolhemos um ponto de corte tal que a árvore resultante tenha o menor RSS. Em maiores detalhes, para quaisquer j e s, definimos o par de semi-planos

$$R_1(j,s) = \{X|X_j < s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\}, \qquad (8.2)$$

e procuramos o valor de j e s que minimize a equação

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2, \qquad (8.3)$$

onde $\hat{y}$$_{R_{1}}$ é a média da resposta para as observações de treino em R_{1}(j,s), e $\hat{y}$$_{R_{2}}$ é a média da resposta para os observações de treino em R_{2}(j,s). Pode-se encontrar os valores de j e s que minimizam (8.3) rapidamente, especialmente quando o número de recursos p não é muito grande.

Em seguida, repetimos o processo, procurando pelo melhor preditor e o melhor ponto de corte para dividir os dados posteriormente, tal que o RSS seja minimizado em cada uma das regiões resultantes. Agora, temos três regiões. Novamente, procuramos dividir uma das três regiões futuramente, tal que o RSS seja minimizado. O processo continua até que um critério de parada seja alcançado; por exemplo, podemos continuar até que nenhuma região contenha mais que cinco observações.

Uma vez que as regiões R_{1},...,R_{J} foram criadas, predizemos a resposta para uma dada observação de teste utilizando a média das observações de treino na região a qual a observação de teste pertence. Uma exemplo de região quíntupla é mostrada na figura 8.3.

Poda da árvore

Loading [MathJax]/extensions/Safe.js

O processo descrito acima pode produzir boas predições no conjunto de treino, mas provavelmente sobreajustará os dados, levando a uma performance ruim do conjunto de teste. Isto acontece porque a árvore resultante pode ser muito complexa. Uma árvore menor com um número menor de divisões (isto é, um número menor de regiões R_{1},...,R_{J}) podem resultar em uma menor variância e uma melhor interpretação a custo de um pequeno aumento na polarização. Uma alternativa possível para o processo descrito acima é construir a árvore de forma que ela cresça até que a diminuição no RSS devido à cada divisão exceda algum limite (alto). Esta estratégia resultará em árvores menores, mesmo possuindo pouca visão, já que uma divisão aparentemente sem utilidade no início da árvore pode ser seguida por uma divisão muito boa - isto é, uma divisão que leva a uma grande redução do RSS futuramente.

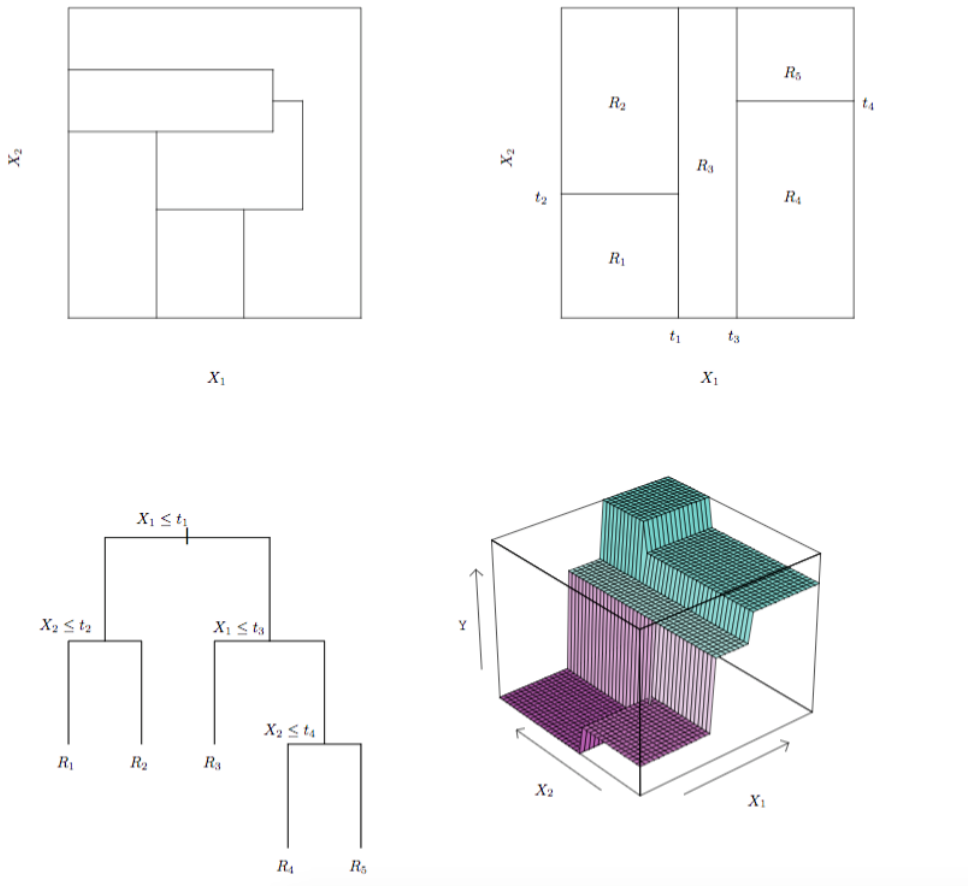


Figura 8.3.
Acima, à esquerda: *uma partição do espaço de recurso bidimensional que poderia não resultar de uma divisão binária recursiva.* Acima, à direita: *A saída da divisão binária recursiva em um exemplo bidimensional.* Abaixo, à esquerda: *Uma árvore correspondente à partição na parte de cima, à direita.* Abaixo, à direita: *um gráfico em perspectiva da superfície de predição correspondente àquela árvore.*

Loading [MathJax]/extensions/Safe.js

Portanto, uma estratégia melhor é criar uma árvore T_{0} muito grande, e então $podá-la$ de volta de forma que se obtenha uma $subárvore$. Como determinamos a melhor forma de podar a árvore? Intuitivamente, nosso objetivo é selecionar uma subárvore que resulte na menor taxa de erro de teste. Dada uma subárvore, podemos estimar seu erro de teste utilizando validação cruzada ou a abordagem de validação do conjunto. No entanto, estimar a validação cruzada para todas as subárvores possíveis seria bastante incômodo, já que existe um número extremamente grande de possíveis subárvores. Em vez disso, precismos de uma forma de selecionar um pequeno conjunto de subárvores para consideração.

A poda por custo de complexidade - também conhecida como a poda do elo mais fraco - nos dá uma forma de fazer isto. Em vez de considerar todas as subárvores possíveis, consideramos uma sequência de árvores indexadas por um parâmetro de afinação não-negativo $α$.

Para cada valor de $α$ há uma subárvore correspondente T $⊂$ T_{0} tal que

$$\sum_{m=1}^{|T|} \sum_{i:\ x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \qquad (8.4)$$

é o menor possível. Aqui, $|T|$ indica o número de nós terminais da árvore T, R_{m} é o retângulo (por exemplo, o subconjunto do espaço de recurso) correspondente ao m-ésimo nó terminal, e $\hat{y}$$_{R_{m}}$ é a resposta predita associada a R_{m} - isto é, a média das observações de treino em R_{m}. O parâmetro de afinação $α$ controla um trade-off entre a complexidade da subárvore e seu ajuste aos dados de treino. Quando $α$ = 0, então a subárvore T será simplesmente igual à T_{0}, porque então (8.4) apenas mede o erro de treino. No entanto, à medida que $α$ aumenta, há um preço a pagar por existir uma árvore com muitos nós terminais, e então a quantidade (8.4) tenderá a ser minimizada por uma subárvore menor.

Portanto, à medida que aumentamos $α$ partindo de 0 em (8.4), os ramos são podados da árvore de uma forma aninhada e previsível, então a obtenção da sequência inteira de subárvores como uma função de $α$ é fácil. Podemos selecionar um valor de $α$ utilizando um conjunto de validação ou utilizando validação cruzada. Então, nós retornamos para o conjunto de dados completo e obtemos a subárvore correspondente a $α$. Este processo está resumido no algoritmo 8.1.

Algoritmo 8.1 ***Construindo uma árvore de regressão***

Loading [MathJax]/extensions/Safe.js

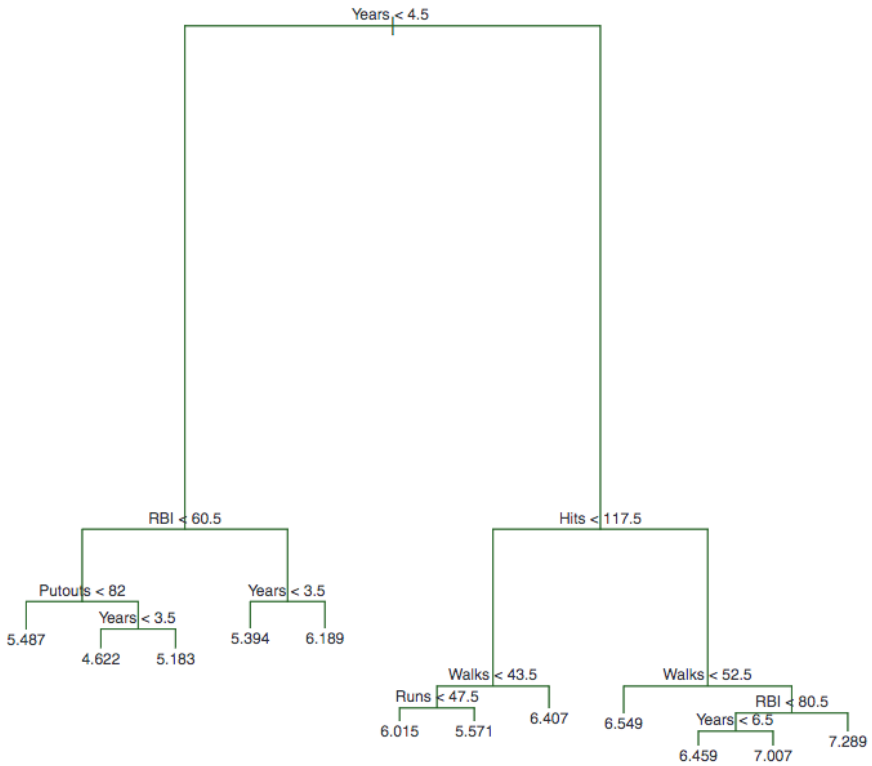
1. Utilize divisão recursiva binária para criar uma árvore grande nos dados de treino, parando apenas quando cada nó terminal possuir um número menor que algum número mínimo de observações.

2. Aplique a poda por custo de complexidade para a árvore grande para que seja obtida uma sequência das melhores subárvores, como função de $α$.

3. Utilize o método de validação cruzada K-fold para escolher $α$. Isto é, divida as observações de treino em K dobras. Para cada k = $1,...,K$:
(a) *Repita as etapas 1 e 2 em todas menos a k-ésima dobra dos dados de treino.*
(b) *Avalie o erro de predição quadrático médio nos dados da k-ésima dobra restante, como função de $α$.*

Faça a média dos resultados para cada valor de $α$, e escolha um $α$ que minimize o erro médio.

4. Retorne a subárvore da etapa 2 que corresponde ao valor escolhido de $α$.



Loading [MathJax]/extensions/Safe.js

Figura 8.4.
Análise da árvore de regressão para os dados $Rebatedores$. A árvore não-podada que resulta em uma divisão top-down gananciosa nos dados de treino é mostrada.

As figuras 8.4 e 8.5 mostram os resultados do ajuste e poda de uma árvore de regressão nos dados $Rebatedores$, utilizando nove dos recursos. Primeiramente, dividimos aleatoriamente o conjunto de dados em duas metades, produzindo 132 observações no conjunto de treino e 131 observações no conjunto de teste. Quando então construímos uma árvore de regressão grande nos dados de treino e variamos $α$ em (8.4) para que críassemos subárvores com números diferentes de nós terminais. Finalmente, realizamos validação cruzada de 6 dobras para que estimássemos o MSE de validação cruzada das árvores como uma função de $α$. (Escolhemos realizar validação cruzada de 6 dobras, pois 132 é um múltiplo exato de 6). A árvore de regressão não-podada é mostrada na figura 8.4. A curva verde na figura 8.5 mostra o erro do coeficiente de variação como uma função do número de folhas, enquanto a curva laranja indica o erro de teste. Foram mostradas também barras de erro ao redor de erros estimados. Para referência, a curva de erro de treino é mostrada em preto. O erro do coeficiente de variação é uma aproximação razoável do erro de teste: o erro do coeficiente de variação assume seu mínimo para uma árvore de três nós, enquanto o erro de teste também cai na árvore com três nós (apesar de assumir seu menor valor na árvore de 10 nós). A árvore podada contendo três nós terminais é mostrada na figura 8.1.

Loading [MathJax]/extensions/Safe.js

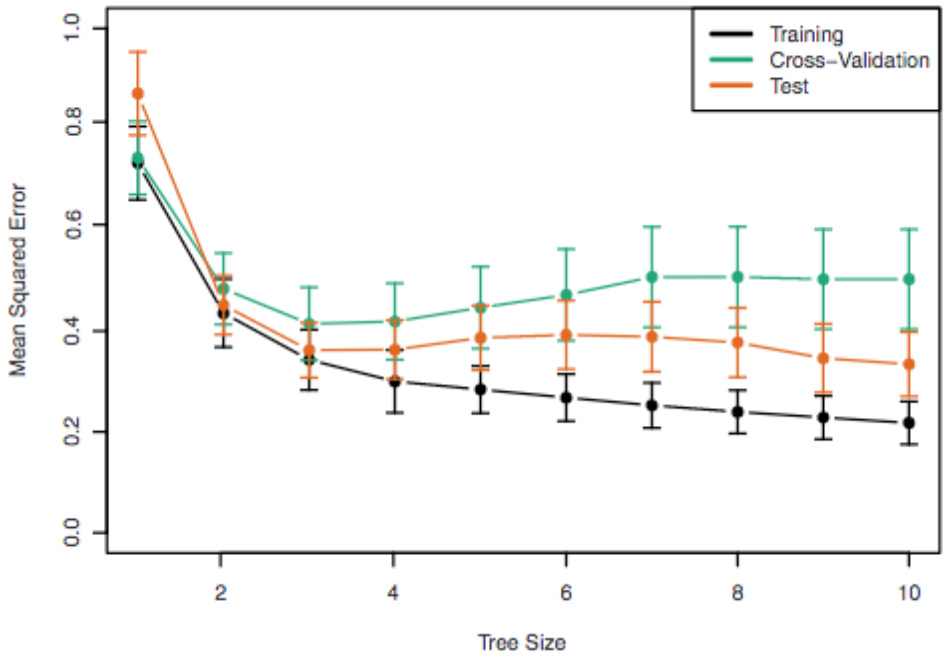


Figura 8.5.
Análise de árvore de regressão para os dados $Rebatedores$. Os MSE de treino, validação cruzada e teste são mostrados como uma função do número de nós terminais na árvore podada. As faixas de erro padrão são mostradas. O erro mínimo de validação cruzada ocorre em uma árvore de tamanho 3.

8.1.2 Árvores de classificação

Loading [MathJax]/extensions/Safe.js

Uma $árvore$ de $classificação$ é bastante similar à uma árvore de regressão, com a diferença que é utilizada para predizer uma resposta qualitativa em vez de uma quantitativa. Relembre que, para uma árvore de regressão, a resposta predita para uma observação é dada pela resposta média das observações de treino que pertencem ao mesmo nó terminal. Em contraste, para uma árvore de classificação, predizemos que cada observação pertence à $classe$ $mais$ $recorrente$ de observações de treino na região a qual pertence. Ao interpretar os resultados de uma árvore de classificação, estamos frequentemente interessados não apenas na predição da classe correspondente a uma região particular de nó um terminal, mas também nas $proporções$ de $classe$ ao longo das observações de treino que caem dentro da região.

A tarefa de crescer uma árvore de classificação é bem similar à tarefa de crescer uma árvore de regressão. Como no contexto da regressão, utilizamos divisão recursiva binária para crescer uma árvore de classificação. No entanto, no contexto de classificação, o RSS não pode ser utilizado como critério para criar as divisões binárias. Uma alternativa natural para o RSS é a $taxa$ de $erro$ de $classificação$. Como planejamos atribuir uma observação em uma dada região à $classe$ $mais$ $recorrente$ de observações de treino naquela região, a taxa de erro de classificação é simplesmente a fração de observações de treino naquela região que não pertencem à classe mais comum:

$$E = 1 - \max_k(\hat{p}_{mk}). \qquad (8.5)$$

Aqui, $\hat{p}$$_{mk}$ representa a proporção de observações de treino na m-ésima região que são da k-ésima classe. No entanto, o erro de classificação não é suficientemente sensível para o crescimento de árvores, e na prática duas outras medidas são preferíveis.

O $Índice$ de $Gini$ é definido por

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \qquad (8.6)$$

uma medida da variância total ao longo das K classes. Não é difícil ver que o índice de Gini assume um valor pequeno se todos os $\hat{p}$$_{mk}$'s estão próximos a zero ou um. Por esta razão, o índice de Gini é referido como a medida da $pureza$ do nó - um valor pequeno indica que um nó contém predominantemente observações de uma única classe.

Uma alternativa ao índice de Gini é a $entropia$, dada por:

Loading [MathJax]/extensions/Safe.js

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}. \quad (8.7)$$

Dado que 0 ≤$\hat{p}$$_{mk}$≤ 1, segue que 0 ≤-$\hat{p}$$_{mk}$$log$($\hat{p}$$_{mk}$). Pode-se mostrar que a entropia assumirá um valor próximo à zero se todos os $\hat{p}$$_{mk}$'s estão próximos a zero ou a um. Portanto, como o índice de Gini, a entropia assumirá um valor pequeno se o m-ésimo nó for puro. Em fato, o índice de Gini e a entropia são bem similares numericamente.

Ao construir uma árvore de classificação, o índice de Gini ou a entropia são tipicamente utilizados para avaliar a qualidade de uma divisão em particular, já que estas duas abordagens são mais sensíveis à pureza do nó do que a taxa de erro de classificação. Qualquer uma destas três abordagens poderiam ser utilizadas ao $podar$ a árvore, mas a taxa de erro de classificação é preferível se a acurácia da predição da árvore final podada for o objetivo.

A figura 8.6. mostra um exemplo do conjunto de dados $Heart$. Estes dados contêm um resultado HD para 303 pacientes que apresentaram dor no peito. Um resultado positivo (Sim) indica a presença de doença no coração baseado em um teste angiográfico, enquanto $Não$ significa ausência de doença no coração. Há 13 preditores, inclusindo Age, Sex, $Chol$ (uma medida do colesterol), e outras medidas de função do coração e pulmão. A validação cruzada resulta em uma árvore com seis nós terminais.

Loading [MathJax]/extensions/Safe.js

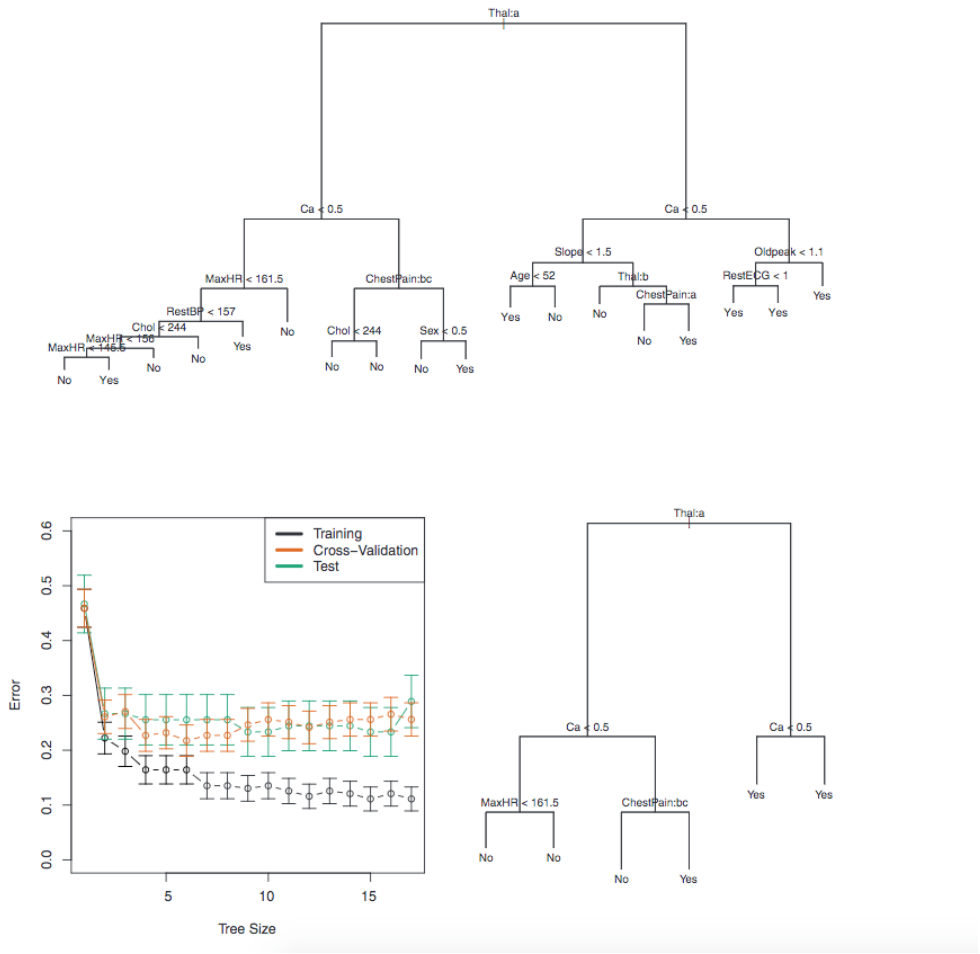


Figura 8.6.
Dados $Coração$. Acima: *A árvore não-podada*. Abaixo, à esquerda: *Erro de validação cruzada, de treino e de teste, para tamanhos diferentes da árvore podada.* Abaixo, à direita: *A árvore podada correspondente ao erro de validação cruzada mínimo.*

Loading [MathJax]/extensions/Safe.js

Em nossa discussão, até agora, assumimos que as variáveis preditivas assumem valores contínuos. No entanto, árvores de decisão podem ser construídas até na presença de variáveis preditivas qualitativas. Por exemplo, nos dados $Heart$, alguns dos preditores, tal como Sex, $Thal$ (teste de pressão de Thallium), e $ChestPain$, são qualitativos. Portanto, uma divisão em uma dessas variáveis equivale a atribuir alguns dos valores qualitativos a um ramo e atribuir os valores restantes ao outro ramo. Na figura 8.6., alguns dos nós internos correspondem à divisão de variáveis qualitativas. Por exemplo, o nó interno no topo corresponde à divisão de $Thal$. O texto $Thal:a$ indica que o ramo à esquerda saindo do nó consiste de observações com o primeiro valor da variável $Thal$ (normal), e o nó à direita consiste das observações restantes (falhas consertadas ou reversíveis). O texto $ChestPain:bc$ indica que o ramo à esquerda saindo do nó consiste das observações com os segundos e terceiros valores da variável $ChestPain$, onde os possíveis valores são angina típico, angina atípico, nor não-anginal, e assintomático.

A figura 8.6. possui uma característica surpreendente: algumas das divisões retornam dois nós terminais que possuem o mesmo valor predito. Por exemplo, considere a divisão $RestECG$ < 1 próxima ao ponto mais baixo à direita da árvore não podada. Independentemente do valor de $RestECG$, um valor de resposta de Sim é predito para estas observações. Por que, então, a divisão é realizada? A divisão é realizada porque leva ao aumento da $pureza$ do $nó$. Isto é, todas as 9 observações correspondentes à folha do lado direito possuem um valor de resposta de Sim, enquanto $^{7}/_{11}$ dos correspondente à folha do lado esquerdo possuem um valor de resposta de Sim. Porque a pureza do nó é importante? Suponha que tenhamos uma observação de teste que pertence à região dada pela folha à direita. Então, podemos estar certos de que seu valor de resposta é Sim. Em contraste, se uma observação de teste pertence à região dada pela folha à esquerda, então seu valor de resposta é provavelmente Sim, mas temos muito menos certeza. Mesmo que a divisão $RestECG<1$ não reduza o erro de classificação, o índice de Gini e a entropia são melhoradas, pois são mais sensíveis à pureza do nó.

8.1.3 Árvores versus modelos lineares

Árvores de regressão e de classificação possuem uma forma muito diferente das abordagens clássicas para regressão e classificação apresentadas anteriormente. Em particular, a regressão linear assume um modelo da forma

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j, \qquad (8.8)$$

enquanto as árvores de regressão assumem um modelo da forma

Loading [MathJax]/extensions/Safe.js

$$f(X) = \sum_{m=1}^{M} c_m \cdot 1_{(X \in R_m)} \qquad (8.9)$$

onde R_{1},...,R_{M} representam uma partição do espaço de recurso, como na figura 8.3.

Qual modelo é melhor? Isso depende do problema com que estamos lidando. Se a relação entre os recursos e as respostas é bem aproximada por um modelo linear como em (8.8), então uma abordagem como a regressão linear provavelmente trabalhará bem, e será melhor do que um método como a árvore de regressão que não explora esta estrutura linear. Se, em vez disso, houver uma relação altamente não-linear e complexa entre os recursos e as respostas como indicado pelo modelo (8.9), então árvores de decisão podem ser melhores que abordagens clássicas. Um exemplo ilustrativo é mostrado na Figura 8.7. As performances relativas de abordagem clássicas e baseadas em árvores podem ser avaliadas pela estimação do erro de teste, utilizando validação cruzada ou a abordagem do conjunto de validação.

Obviamente, outras considerações além do erro de teste podem participar da seleção de um método de aprendizado estatístico; por exemplo, em certos contextos, a predição utilizando uma árvore pode ser preferida por motivos de interpretabilidade e visualização.

Loading [MathJax]/extensions/Safe.js

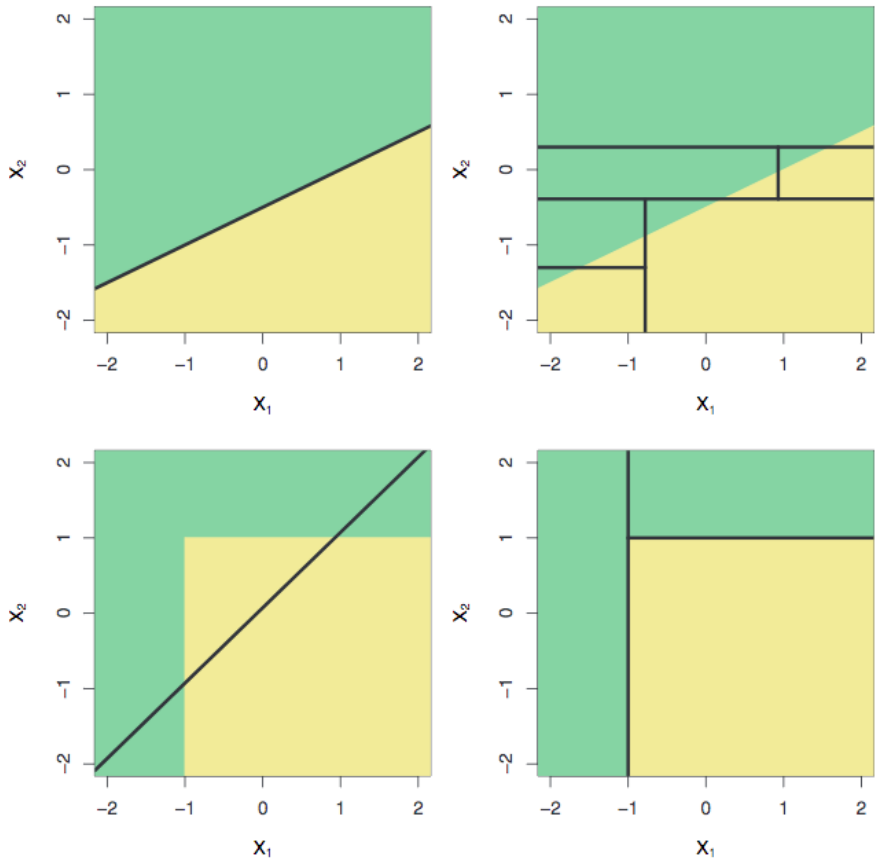


Figura 8.7.
Linha acima: *Um exemplo de classificação bidimensional na qual a verdadeira fronteira de decisão é linear, e é indicada pelas regiões sombreadas. Uma abordagem clássica que assume uma fronteira linear (à esquerda) terá uma performance melhor que uma árvore de decisão que performa divisões paralelamente aos eixos (à direita).* Linha abaixo: *Aqui, a verdadeira fronteira de decisão é não-linear. Neste caso, um modelo linear não será capaz de capturar a verdadeira fronteira de decisão (à esquerda), enquanto uma árvore de decisão é bem-sucedida (à direita).*

8.1.4 Vantagens e desvantagens das árvores

Loading [MathJax]/extensions/Safe.js

Árvores de decisão para regressão e classificação possuem vantagens sobre as abordagens mais clássicas.

VANTAGENS

1. *Árvores são muito fáceis de explicar às pessoas. De fato, elas são até mais fáceis de explicar que regressão linear!*

2. *Algumas pessoas acreditam que árvores de decisão espelham de forma mais próxima as tomadas de decisões humanas do que as abordagens de regressão e classificação vistas em capítulos anteriores.*

3. *Árvores podem ser mostradas graficamente, e são facilmente interpretadas até por não-especialistas (especificamente se elas são pequenas).*

4. *Árvores podem lidar facilmente com preditores qualitativos sem a necessidade de criar variáveis dummy.*

DESVANTAGENS

1. *Infelizmente, árvores em geral não possuem o mesmo nível de acurácia preditiva de outras abordagens de regressão e classificação.*

2. *Ademais, árvores podem ser bastante não-robustas. Em outras palavras, uma pequena mudança nos dados pode causar uma grande mudança na árvore estimada final.*

No entanto, ao agregar muitas árvores de decisão, utilizando métodos como $bagging$, $random$ $forests$, e $boosting$, a performance preditiva das árvores pode ser substancialmente melhorada.

8.2 Bagging, Random Forests, Boosting

Bagging, random forests e boosting utilizam árvores como blocos de construção para construir modelos de predição mais poderosos.

8.2.1 Bagging

Loading [MathJax]/extensions/Safe.js

O bootstrap, introduzido no capítulo 5, é uma ideia extremamente poderosa. É utilizada em muitas situações na qual é difícil ou até impossível computar diretamente o desvio padrão de uma quantidade de interesse. Nós vimos aqui que o bootstrap pode ser utilizado em um contexto completamente diferente, com o objetivo de melhorar métodos de aprendizado estatísticos tais como árvores de decisão.

As árvores de decisão discutidas na seção 8.1 sofrem de $variância$ $alta$. Isto significa que se dividirmos os dados de treino em duas partes aleatoriamente, e ajustar uma árvore de decisão a ambas as metades, os resultados poderiam ser bem diferentes. Em constraste, um procedimento com $variância$ $baixa$ irá entregar resultados similares se aplicado repetidamente a conjuntos de dados distintos; a regressão linear tende a apresentar baixa variância, se a proporção de n em relação a p for moderadamente grande. $Agregação$ de $bootstrap$, ou $bagging$, é um procedimento com propósitos gerais que possui o objetivo de reduzir a variância de um método de aprendizado estatístico; o introduzimos aqui porque é particularmente útil e é frequentemente utilizado no contexto das árvores de decisão.

Relembre que dado um conjunto de n observações independentes Z_{1},...,Z_{n}, cada um com variância σ², a variância da média $\bar{Z}$ das observações é dada por σ^2/n. Em outras palavras, *utilizar a média de um conjunto de observações reduz a variância*. Portanto, uma forma natural de reduzir a variância, e consequentemente, aumentar a acurácia da predição de um método de aprendizado estatístico seria pegar muitos conjuntos de treino da população, construir um modelo de predição separado utilizando cada conjunto de treino, e calcular a média das predições resultantes. Em outras palavras, poderíamos calcular $\hat{f}_{1}$(x)$, $\hat{f}_{2}$(x)$,...,$\hat{f}_{B}(x) utilizando B conjuntos de treino separados, e calcular a média deles para obter um modelo de aprendizado estatístico único com baixa variância, dado por

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{b}(x).$$

Obviamente, isto não é prático porque normalmente não temos acesso a múltiplos conjuntos de dados. Em vez disso, podemos utilizar bootstrap, pegando amostras repetidas do (único) conjunto de dados. Neste contexto geramos B $bootstrapped$ conjuntos de dados de treino diferentes. Então, treinamos nosso método no b-ésimo bootstrapped conjunto de treino com o objetivo de obter $\hat{f}^{*b}$(x)$, e finalmente calculamos a média de todas as predições para obter

Loading [MathJax]/extensions/Safe.js

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

Isto é denominado $bagging$.

Como o bagging pode melhorar predições para muitos métodos de regressão, é particulamente útil para árvores de decisão. Para aplicar bagging à árvores de regressão, simplesmente construímos B árvores de regressão utilizando B conjuntos de treinos $bootstrapped$, e calculamos a média das predições resultantes. Estas árvores estão profundamente crescidas, mas não estão podadas. Logo, cada árvore individual possui variância alta, mas baixa polarização. Calcular a média destas B árvores reduz a variância. A abordagem bagging causou melhoras expressivas na acurácia ao combinar centenas ou até milhares de árvores em um único procedimento.

Até então, descrevemos o procedimento de bagging no contexto de regressão para prever uma variável quantitativa de saída Y. Como o bagging pode ser extendido para um problema de classificação onde Y é qualitativo? Nesta situação, há algumas abordagens possíveis, mas a mais simples é a seguinte. Para uma dada observação de teste, podemos gravar a classe prevista por cada uma das B árvores, e fazer um $voto$ $majoritário$: a predição acima de todas é a mais recorrente entre as B predições.

A figura 8.8 mostra os resultados das árvores de bagging nos dados $Heart$. A taxa de erro de teste é mostrada como uma função de B, o número de árvores construídas utilizando conjuntos de dados de treino bootstrapped. Vemos que a taxa de erro de teste de bagging é sensivelmente menor neste caso do que no caso da taxa de teste de erro obtida de uma única árvore. O número de árvores B não é um parâmetro crítico para o bagging; utilizar um valor muito grande de B não levará à sobreajuste (overfitting). Na prática, utilizamos um valor de B suficientemente grande tal que o erro se estabilize. Utilizar B = 100 é suficiente para atingir uma boa performance neste exemplo.

Estimação do erro out-of-bag

Loading [MathJax]/extensions/Safe.js

Há uma maneira bastante direta de estimar o erro de teste de um modelo de bagging, sem a necessidade de realizar validação cruzada ou a abordagem de validação do conjunto. Relembre que a chave para o bagging é o fato das árvores se ajustarem repetidamente à subconjuntos bootstrapped das observações. Pode-se mostrar que, na média, cada árvore no bagging faz uso de aproximadamente dois terços das observações. O um terço remanescente das observações não utilizadas para ajustar uma árvore no bagging são denominadas como observações out-of-pag (OOB). Podemos predizer a resposta para a i-ésima observação utilizando cada uma das árvores na qual cada observação foi OOB. Isto resulta em aproximadamente $^{\$B\$}/_{\$3\$}$ predições para a i-ésima observação. Para obter uma única predição para a i-ésima observação, podemos calcular a média destas respostas preditas (se a regressão for o objetivo), ou podemos obter um voto majoritário (se o objetivo for a classificação). Isto leva à uma única predição OOB para a i-ésima observação. Uma predição OOB pode ser obtida desta forma para cada uma das n observações, das quais o MSE do OOB geral (para um problema de regressão) ou o erro de classificação (para um problema de classificação) pode ser computado. O erro resultante da OOB é uma estimativa válida do erro de teste para o modelo com $bagging$, já que a resposta para cada observação é predita utilizando apenas as árvores que não foram ajustadas utilizando aquela observação. A figura 8.8 mostra o erro do OOB nos dados $Heart$. Pode ser mostrado que com B suficientemente grande, o erro OOB é virtualmente equivalente ao erro leave-one-out da validação cruzada. A abordagem OOB para estimar o erro de teste é particularmente conveniente ao realizar bagging em grandes conjuntos de dados para o qual validação cruzada seria computacionalmente pesado.

Loading [MathJax]/extensions/Safe.js

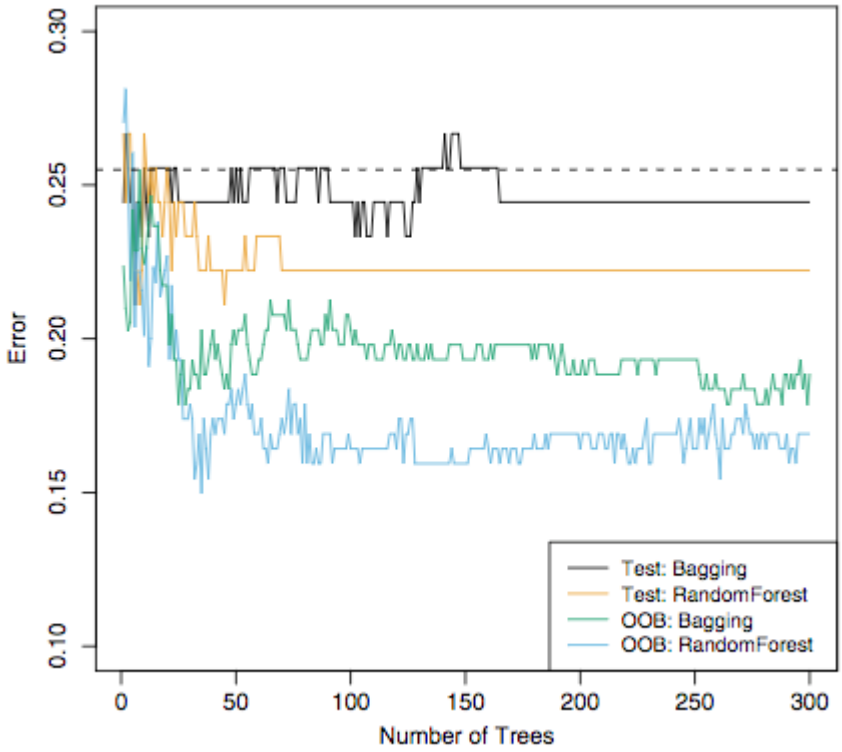


Figura 8.8
Resultados do bagging e random forest para os dados $Heart$. O erro de teste (preto e laranja) é mostrado como uma função de B, o número de conjunto de treinos bootstrapped utilizado. Random forests foram utilizados com m = √p. A linha tracejada indica o erro de teste resultante de uma única árvore de classificação. Os traços verde e azul mostram o erro OOB, o qual neste caso é consideravelmente menor.

Medidas de importância de variável

Como discutimos, o bagging tipicamente resulta em uma melhora da acurácia sobre a predição utilizando uma única árvore. Infelizmente, no entanto, pode ser difícil interpretar o modelo resultante. Relembre que uma das vantagens das árvores de decisão são os diagramas resultantes atrativos e facilmente interpretados, tal como o mostrado na figura 8.1. No entanto, quando fazemos bagging em um grande número de árvores, não é mais possível representar o procedimento de aprendizado estatístico resultante utilizando uma única árvore, e não é mais claro quais variáveis são mais importantes para o procedimento. Portanto, o bagging melhora a acurácia da predição ao custo da interpretabilidade.

Loading [MathJax]/extensions/Safe.js

Apesar da coleção de árvores nas quais foram utilizadas bagging ser muito mais difícil de interpretar que uma única árvore, se pode obter um resumo geral da importância de cada preditor utilizando o RSS (para árvores de regressão de bagging) ou o índice de Gini (para árvores de classificação de bagging). No caso de árvores de regressão de bagging, podemos gravar a quantidade total na qual o RSS (8.1) decresce devido à divisão sobre um dado preditor, feita a média sobre todas as B árvores. Um valor grande indica um preditor importante. Similarmente, no contexto das árvores de classificação de bagging, podemos adicionar o valor total no qual o índice de Gini (8.6) é diminuído ao se dividir sobre um dado preditor, feita a média sobre todas as B árvores.

Uma representação gráfica das $importâncias$ de $variável$ nos dados $Heart$ é mostrada na figura (8.9). É observada a diminuição da média no índice de Gini para cada variável, relativa à maior. As variáveis com maior diminuição da média no índice de Gini são $Thal$, Ca, e $ChestPain$.

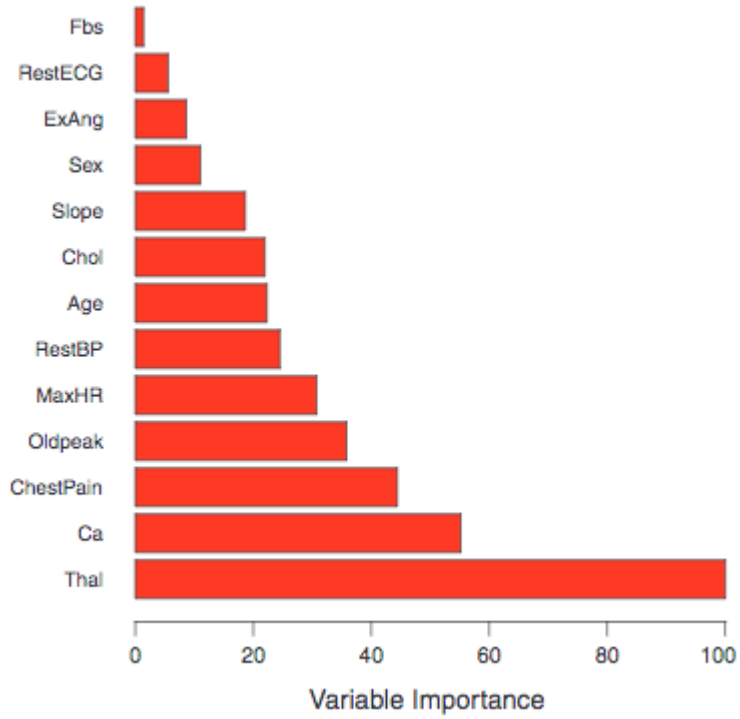


Figura 8.9
Um gráfico de importância de variável para os dados $Heart$. A importância de variável é computada utilizando a diminuição da média no índice de Gini, e expressa relativo ao máximo.

8.2.2 Random Forests

Loading [MathJax]/extensions/Safe.js

Random forests fornecem uma melhora em relação a árvores com bagging por meio de um pequeno "puxão" que $descorrelaciona$ as árvores. Como no bagging, construímos um número de árvores de decisão em amostras de treino com bootstrap. Mas, ao construir estas árvores de decisão, a cada vez em que uma divisão em uma árvore é considerada, uma $amostra$ $aleatória$ de m $preditores$ é escolhida como candidatos para divisão do conjunto completo de p preditores. A divisão é permitida para usar apenas um dos m preditores. Uma amostra fresca de m preditores é utilizada em cada divisão, e tipicamente escolhemos m = $≅$ √p $-$ isto é, o número de preditores considerado em cada divisão é aproximadamente igual à raiz quadrada do número total de preditores (4 dos 13 para os dados $Heart$).

Em outras palavras, ao construir uma random forest, em cada divisão na árvore, o algoritmo não é *permitido a considerar* a maioria dos preditores disponíveis. Pode soar estranho, mas é um raciocínio esperto. Suponha que há um preditor muito forte no conjunto de dados, juntamente com um número de outros preditores moderadamente fortes. Então, na coleção de árvores nas quais foram utilizadas o bagging, a maioria ou todas as árvores utilizarão este forte preditor na divisão do topo. Consequentemente, todas as árvores com bagging serão similares umas às outras. Logo, as predições destas árvores serão altamente correlacionadas. Infelizmente, calcular a média de quantidades altamente correlacionadas não leva a uma redução tão grande na variância. Em particular, isto significa que o bagging não levará a uma redução substancial na variância sobre uma única árvore neste contexto.

Loading [MathJax]/extensions/Safe.js

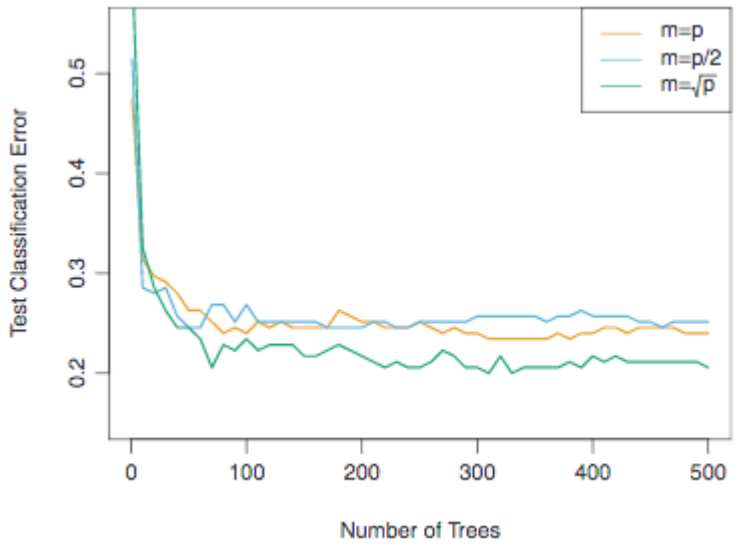


Figura 8.10.
Resultados de random forests do conjunto de dados da expressão de gene de 15 classes com p = 500 preditores. O erro de teste é mostrado como uma função do número de árvores. Cada linha colorida corresponde a um valor diferente de m, o número de preditores disponível para divisão em cada nó interior da árvore. Random forests (m < p) levam a uma sensível melhora em relação ao bagging (m = p). Uma única árvore de classificação possui uma taxa de erro de 45,7%.

Random forests superam este problema ao forçar cada divisão a considerar apenas um subconjunto dos preditores. Portanto, na média, $(p-m)$/p das divisões não considerarão o preditor forte, e então os outros preditores terão mais chance. Podemos pensar nesse processo como $descorrelacionamento$ das árvores, assim fazendo a média das árvores resultante serem menos variáveis e portanto mais confiáveis.

A principal diferença entre $bagging$ e $random$ $forests$ é a escolha do tamanho do subconjunto preditor m. Por exemplo, se um random forest é construído utilizando m = p, então isto equivale simplesmente ao $bagging$. Nos dados $Heart$, random forests utilizando m = √p leva a uma redução em ambos o erro de teste e o erro OOB, se os compararmos com os do bagging. (Figura 8.8).

Loading [MathJax]/extensions/Safe.js

Utilizar um pequeno valor de m ao construir uma random forest irá ser tipicamente útil quando tivermos um grande número de predições correlacionadas. Aplicamos random forests para um conjunto de dados biólogico altamente dimensional consistindo de medidas de expressão de 4718 genes medidos em amostras de tecidos de 49 pacientes. Há cerca de 20000 genes em humanos, e genes individuais possuem níveis diferentes de atividade, ou expressão, em células particulares, tecidos, e condições biológicas. Neste conjunto de dados, cada uma das amostras dos pacientes possui uma legenda quantitativa com 15 níveis diferentes: normal ou 1 dos 14 tipos diferentes de câncer. Nosso objetivo foi utilizar random forests para predizer o tipo de câncer baseado em 500 genes que possuem a maior variância no conjunto de dados. Nós dividimos aleatoriamente as observações em conjuntos de treino e de teste, e aplicamos random forests ao conjunto de treino para três valores diferentes do número de variáveis de divisão m. Os resultados são mostrados na Figura 8.10. A taxa de erro de uma única árvore é 45,7% e a taxa nula é de 75,4%. Nós vimos que utilizar 400 árvores é suficiente para dar uma boa performance, e que a escolha m = √p melhorou sensivelmente o erro de teste em relação ao bagging (m = p) neste exemplo. Como no bagging, random forests não terão sobreajuste se aumentarmos B, então na prática utilizamos um valor de B suficientemente grande para que a taxa de erro seja estabilizada.

8.2.3 Boosting

Agora discutiremos o $boosting$, outra abordagem para melhorar as predições resultantes de uma árvore de decisão. Como o bagging, o boosting é uma abordagem geral que pode ser aplicada a muitos métodos de aprendizado estatístico para regressão ou classificação. Aqui, restrigimos nossa discussão do boosting para o contexto de árvores de decisão.

Relembre que o bagging envolve a criação de cópias múltiplas do conjunto de dados original utilizando o bootstrap, ajustando uma árvore de decisão separada para cada cópia, e então combinando todas as árvores para criar um único modelo preditivo. Notavelmente, cada árvore é construída em um conjunto de dados com bootstrap, independentemente das outras árvores. O boosting funciona de uma fórmula similar, exceto que as árvores são crescidas $sequencialmente$: cada árvore é crescida utilizando informação de árvores crescidas anteriormente. O boosting não involve amostragem do bootstrap; em vez disso, cada árvore é ajustada em uma versão modificada do conjunto de dados original.

Considere primeiramente o contexto de regressão. Como o bagging, o boosting envolve a combinação de um grande número de árvores de decisão, $\hat{f}^{1}$,...,$\hat{f}^{B}$. O boosting é descrito no Algoritmo 8.2.

Loading [MathJax]/extensions/Safe.js

Qual a ideia por trás deste procedimento? Ao contrário do ajuste de uma única árvore de decisão grande para os dados, o qual equivale a potecialmente um sobreajuste, a abordagem do boosting $aprende$ $vagarosamente$. Dado o modelo atual, ajustamos a árvore de decisão para os resíduos do modelo. Isto é, ajustamos uma árvore utilizando os resíduos atuais, em vez da saída Y, como a resposta. Então, adicionamos esta nova árvore de decisão na função ajustada com o objetivo de atualizar os resíduos. Cada uma destas árvores pode ser pequena, com apenas alguns nós terminais, determinados pelo parâmetro d no algoritmo. Ao ajustar árvores pequenas aos resíduos, nós vagarosamente melhoramos $\hat{f}$ nas áreas em que a performance não é boa. O parâmetro de encurtamento λ atrasa o processo mais ainda, aceitando mais e árvores diferentes para atacar os resíduos. Em geral, abordagens de aprendizado estatístico que $aprendem$ $vagarosamente$ tendem a ter uma boa performance. Note que no boosting, ao contrário do bagging, a construção de cada árvore depende fortemente nas árvores que já foram crescidas.

Acabamos de descrever o processo de boosting de árvores de regressão. As árvores de classificação de boosting trabalham em uma forma similar, mas de uma forma um pouco mais complexas, e os detalhes estão omitidos aqui.

O Boosting possui três parâmetros de afinação:

1. O número de árvores B. Ao contrário de bagging e random forests, o boosting pode sobreajustar se B é muito grande, apesar deste sobreajuste tender a ocorrer vagarosamente. Utilizamos validação cruzada para selecionar B.

2. O parâmetro de encurtamento λ, um pequeno número positivo. Isto controla a taxa na qual o boosting aprende. Típicos valores são 0,01 ou 0,001, e a escolha certa pode depender do problema. Um λ muito pequeno pode pedir um valor bastante grande B para alcançar uma boa performance.

3. O número d de divisões em cada árvore, o qual controla a complexidade do conjunto no qual foi utilizado o boosting. Frequentemente, $d=1$ funciona bem, e no caso cada árvore é um $pedaço$, consistindo de uma única divisão. Neste caso, o conjunto "em que foi realizado boosting" está ajustando um modelo aditivo, já que cada termo envolve apenas uma única variável. Generalizando, d é a $profundidade$ da $interação$, e controla a ordem de interação do modelo em que foi realizado boosting, já que d divisões podem envolver no máximo d variáveis.

Algoritmo 8.2 ***Boosting para árvores de regressão***

Loading [MathJax]/extensions/Safe.js

1. Estabeleça $\hat{f}(x)=0$ e r_{i} = y_{i} para todo i no conjunto de treino.

2. Para $b=1,2,...,B$, repita:

(a) Ajuste uma árvore $\hat{f}^{b}$ com d divisões (d + 1 nós terminais) aos dados de treino (X,r).

(b) Atualize $\hat{f}$ ao adicionar uma versão encolhida da nova árvore:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^{b}(x). \qquad (8.10)$$

(c) Atualize os resíduos,

$$r_i \leftarrow r_i - \lambda \hat{f}^{b}(x_i). \qquad (8.11)$$

3 Retorne o modelo com boosting,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^{b}(x). \qquad (8.12)$$

Loading [MathJax]/extensions/Safe.js

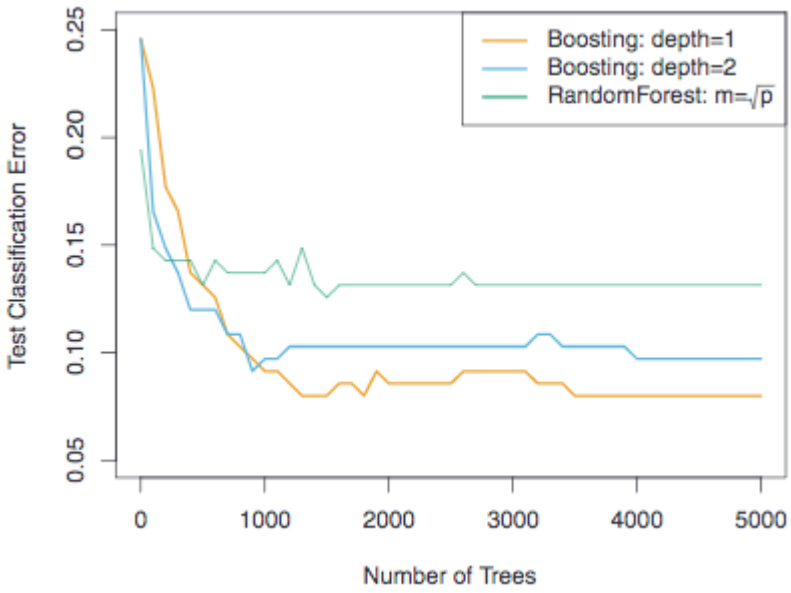


Figura 8.11.
Resultados da performance de boosting e random forests no conjuntos de dados da expressão de um gene de 15 classes com o objetivo de predizer cancer *x* normal. *O erro de teste é mostrado como uma função do número de árvores. Para os dois modelos em que foi realizado boosting, λ = 0,01. Árvores com profundidade 1 possuem performance melhor que as com profundidade 2, e ambas possuem melhor performance que o random forest, apesar dos erros padrões serem em torno de 0,02, fazendo estas diferenças serem insignificantes. A taxa de erro de teste para uma única árvore é de 24%.*

Na Figura 8.11, aplicamos o boosting ao conjunto de dados da expressão de um gene de 15 classes, com o objetivo de desenvolver um classificador que pudesse distinguir a classe normal das 14 classes de câncer. Apresentamos o erro de teste como uma função do número total de árvores e da profundidade de interação d. Vemos que pedaços simples com uma profundidade de interação 1 possuem boa performance se há uma quantidade suficiente deles inlcusos. Este modelo possui performance melhor que o de profundidade 2, e ambos possuem melhor performance que um random forest. Isso realça uma diferença entre boosting e random forests: no boosting, devido ao fato do crescimento de uma árvore particular levar em consideração as outras árvores que já foram crescidas, árvores menores são tipicamentes suficientes. Utilizar árvores menores podem melhorar a interpretabilidade; por exemplo, utilizar pedaços leva a um modelo aditivo.

Loading [MathJax]/extensions/Safe.js