Faculdade de Economia, Administração e Contabilidade

Machine Learning

Laboratório de Técnicas de Classificação

Árvores de Decisão com R

Prof. Antonio Geraldo da Rocha Vidal

2023

Sumário

Você deve entregar um relatório com os resultados das etapas elaboradas neste laboratório no **e-Disciplinas**, para formatá-lo siga estas orientações:

1. Crie um documento Word e identifique-o com o nome do laboratório, data de elaboração e o seu nome ou do grupo que o elaborou;
2. Crie um tópico para cada resultado que você considerar relevante (manipulação de dados ou resultado de algum processamento) identificando-o com um título e uma breve explicação. Os resultados podem ser imagens de gráficos gerados ou de listas de valores ou dados de resultados obtidos. Não devem ser incluídos os *scripts* ou instruções de processamento utilizados, inclua apenas os resultados que você considerar relevantes.
3. No final do relatório crie um último tópico denominado “Conclusões” e elabore comentários, sugestões e conclusões sobre o que você pode aprender com a elaboração deste laboratório.

Esta apostila introdutória pode conter erros, falhas ou imprecisões. Se você identificar algum problema por favor informe através do e-mail vidal@usp.br para que a correção possa ser providenciada.

Esta apostila não é autoral, tem objetivo estritamente didático como material de apoio para a disciplina. Foi desenvolvida através da compilação dos diversos textos e materiais citados na bibliografia.

 Obrigado!

Introdução

A linguagem R é uma linguagem de programação e ambiente de software aberto voltado para análise estatística e computacional de dados, geração de gráficos e de relatórios. Foi criada em 1998 por Ross Ihaka e Robert Gentleman na Universidade de Auckland, Nova Zelândia, e é atualmente desenvolvida pelo "*R Development Core Team*", além de contar com a contribuição de uma grande comunidade de cientistas de dados.

Desde 2004 a linguagem R é disponibilizada gratuitamente sob a modalidade de licença pública GNU, sendo hoje fornecida para diversos sistemas operacionais, como Linux, Windows e Mac, além de ser embarcada em várias plataformas de análise de dados.

Seu nome origina-se da primeira letra do nome de seus dois autores originais (**R**obert Gentleman and **R**oss Ihaka), e devido a uma brincadeira que ambos fizeram com o nome da linguagem S da Bell Labs.

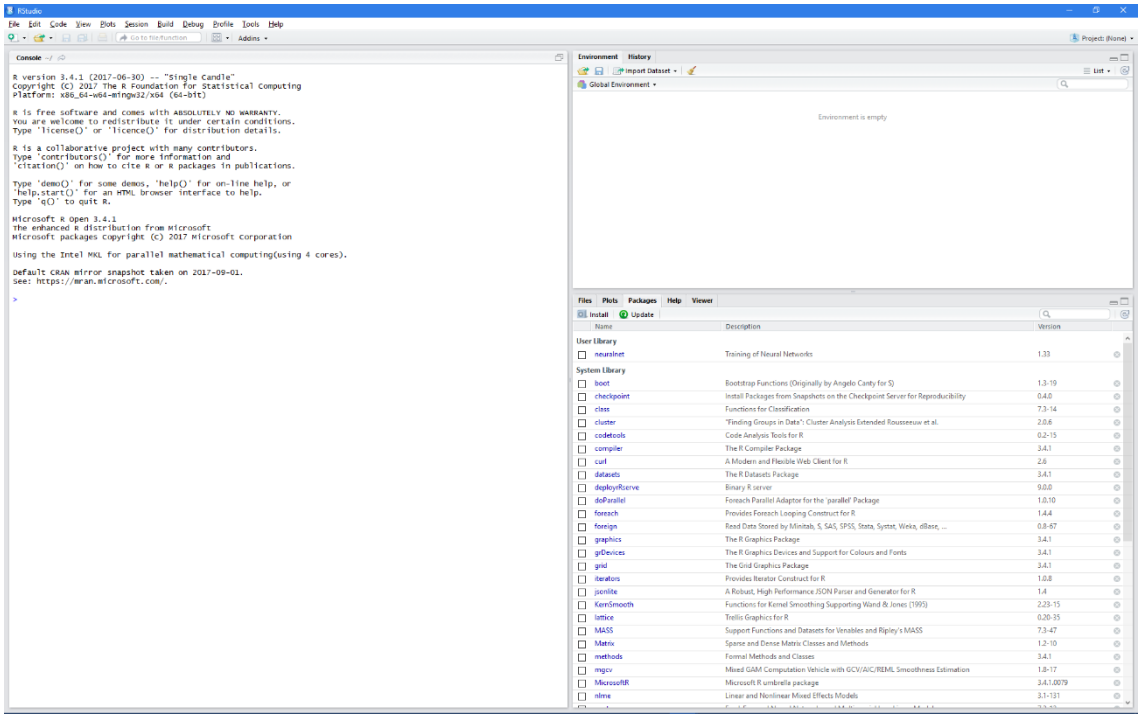
Saiba mais sobre a linguagem em https://www.r-project.org/.

O RStudio, que pode ser obtido em https://www.rstudio.com/products/rstudio/download/, é um conjunto de ferramentas integradas projetadas para ajudá-lo a ser mais produtivo com a linguagem R. Ele inclui um console, um editor inteligente de código, que suporta a execução direta de código, e uma variedade de ferramentas robustas para traçar, visualizar histórico, depurar e gerenciar seu espaço de trabalho em R. Saiba mais sobre os recursos do RStudio em https://www.rstudio.com/products/rstudio/features/ .

Após instalar o RStudio em seu computador, acione-o simplesmente localizando-o no menu iniciar do Windows e clicando o seu ícone:

Você será apresentado ao console do RStudio, conforme ilustra a figura a seguir. Inicialmente faça um passeio pelas diversas janelas do RStudio, resumidamente descritas a seguir:

- **Console**: onde você digita as instruções utilizando a sintaxe da linguagem R e recebe os resultados de sua execução;
- **Environment**: onde você poderá visualizar e consultar o seu espaço de trabalho, que apresentará os conjuntos de dados utilizados para análise, variáveis e valores de resultados das operações realizadas.
- **History**: onde você poderá consultar e visualizar todo o histórico de instruções executadas e operações realizadas.
- **Files**: onde você poderá consultar arquivos e pastas do seu espaço de trabalho. Durante a instalação do RStudio, uma pasta denominada R, é automaticamente criada no seu ambiente de usuário Windows.
- **Plots**: onde você poderá consultar os resultados gráficos das operações realizadas. Particularmente neste exercício, será onde visualizaremos a rede neural com seus neurônios.
- **Packages**: onde você poderá consultar e instalar os pacotes da linguagem R. Os pacotes constituem o verdadeiro "poder" do R, pois são gratuitos, construídos por uma comunidade mundial de cientistas de dados, avalizados pelos mantenedores da linguagem R, atendem a praticamente qualquer necessidade de análise de dados e estão em constante evolução. Os pacotes estão disponíveis através da Internet, portanto, para instalá-los em seu espaço de trabalho você precisará ter acesso à Internet.
- **Help**: a partir de onde você poderá aprender "tudo" sobre a linguagem R e começar a se tornar um verdadeiro "cientista de dados".
- **Viewer**: onde você poderá visualizar resultados e objetos, dependendo dos pacotes R que estiver utilizando.



Neste laboratório vamos estudar árvores de decisão. Este tipo de técnica de aprendizado de máquina supervisionado é utilizado para resolver uma variedade de problemas de regressão e classificação.

Vamos imaginar que você está jogando um jogo de adivinhação de **vinte perguntas**. Seu oponente escolheu secretamente um assunto, e você deve descobrir o que ele/ela escolheu. Em cada turno, você pode fazer uma pergunta sim ou não, e seu oponente deve responder com sinceridade. Como você descobre o segredo com o menor número possível de perguntas?

Deve ser óbvio que algumas perguntas são melhores do que outras. Por exemplo, perguntando "pode voar?" como sua primeira pergunta é provável que seja pouco útil; ao passo que perguntando "está vivo?" possa ser um pouco mais útil. Intuitivamente, você deseja que cada pergunta restrinja significativamente o espaço de possibilidades, eventualmente levando à sua resposta final.

Por incrível que pareça, essa é a ideia básica por trás dos algoritmos de **árvores de decisão**. Em cada ponto ou nó, você considera um conjunto de perguntas que podem particionar seu conjunto de dados. Você escolhe a pergunta que fornece a melhor divisão ou classe e novamente tenta encontrar as melhores perguntas para as partições restantes. Você para uma vez que todos os pontos que você está considerando são da mesma classe. Em seguida, a tarefa de classificação é fácil. Partindo da raiz da árvore, você pode simplesmente escolher um objeto e jogá-lo árvore abaixo. As perguntas irão guiá-lo até a folha ou classe adequada para classificá-lo.

Neste laboratório, você aprenderá sobre alguns dos diferentes tipos de árvores de decisão, suas vantagens e desvantagens, e como implementá-las utilizando a **linguagem R**.

Requisitos

A linguagem R é muito extensa e a comunidade que a desenvolve, apoia e utiliza, está continuamente agregando novos recursos e algoritmos. Neste laboratório estaremos utilizando uma pequena parte dos recursos e possibilidades analíticas da linguagem R.

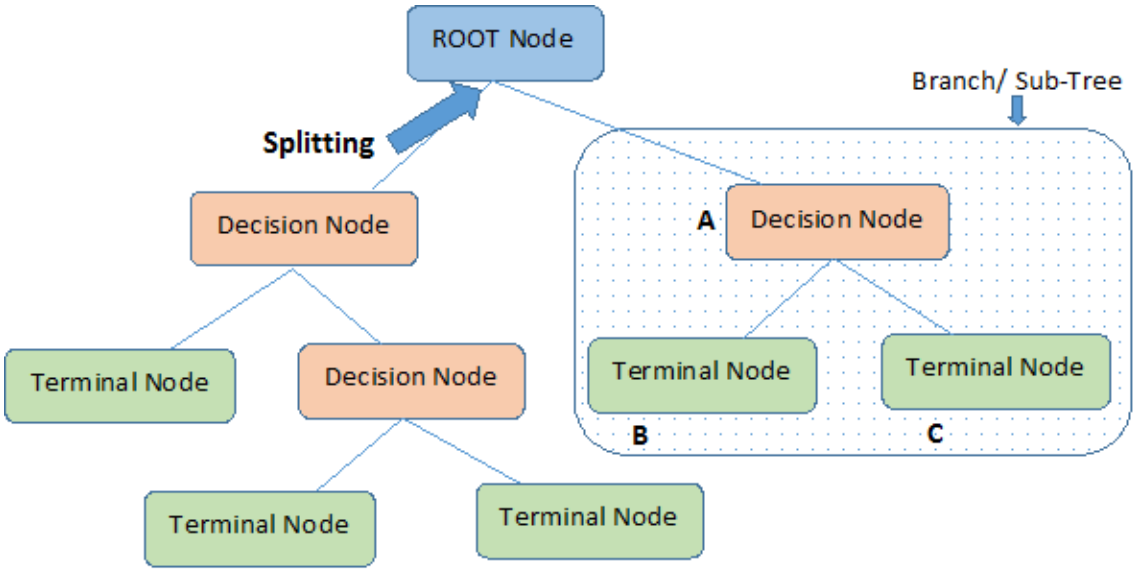
Verifique se os seguintes itens estão instalados no seu sistema:

- R versão 3.6.1 ou versão mais recente
- RStudio 1.2.5 ou versão mais recente.

Para trabalhar com a linguagem R é necessário estar conectado permanente na Internet, uma vez que todos os seus “pacotes” ou bibliotecas de recursos são acessíveis pela Web.

Árvores de Decisão

A árvore de decisão é um tipo de algoritmo de aprendizado de máquina supervisionado que pode ser usado em problemas de regressão e classificação. Ele é muito útil, pois funciona tanto para variáveis de entrada e saída contínuas (numéricas) como categóricas (não numéricas).



Note:- A is parent node of B and C.

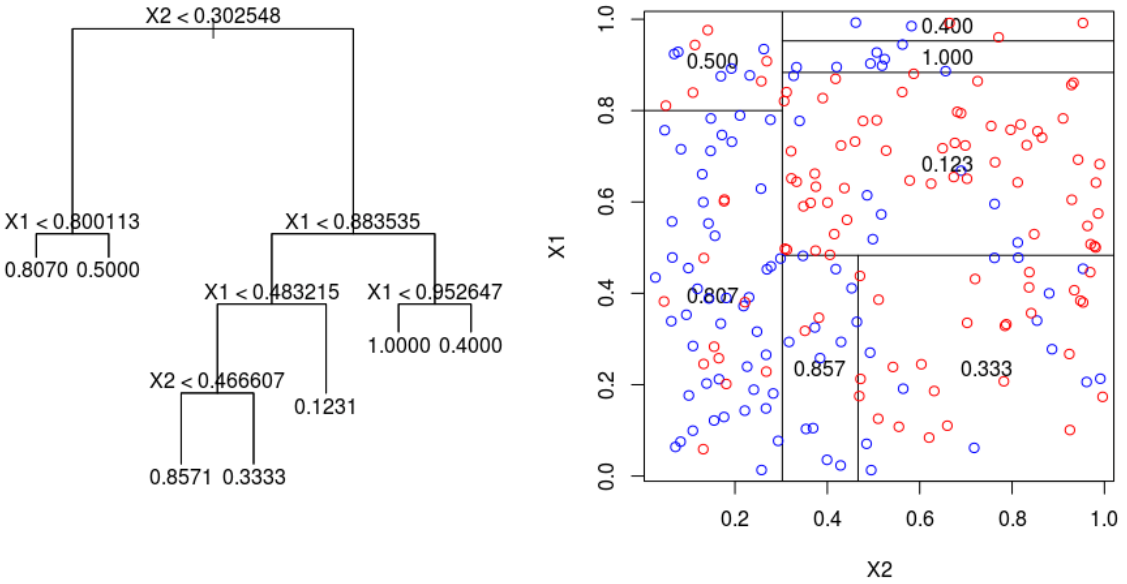
Vamos identificar terminologias importantes na árvore de decisão, olhando para a figura anterior:

- **Nó raiz** representa toda a população ou amostra de dados. Ele é dividido em dois ou mais conjuntos homogêneos.
- **Dividir** é o processo de separação de um nó em dois ou mais sub nós.
- Quando um sub nó se divide em outros sub nós, ele é chamado de **Nó de Decisão**.
- Quando um nó não pode ser mais dividido é chamado de **Nó Terminal** ou uma **Folha**.
- Quando você remove os sub nós de um nó de decisão, esse processo é chamado de **Poda**. O oposto da poda é a **Divisão**.
- Uma subseção de uma árvore inteira é chamada **Ramo**.
- Um nó, que é dividido em sub nós, é chamado um **Nó Pai** dos sub nós; os seus sub nós são denominados **Filhos** do nó pai.

Tipos de Árvores de Decisão

Árvores de Regressão

Dê uma olhada na figura a seguir. Ela ajuda a visualizar a natureza do particionamento realizado por uma **Árvore de Regressão**. Ela mostra uma árvore não podada e uma árvore de regressão ajustada a um conjunto de dados aleatório. Ambas as visualizações mostram uma série de regras de divisão, começando na parte superior da árvore. Observe que cada divisão do domínio é alinhada com um dos eixos de divisão. O conceito de divisão paralela do eixo generaliza de forma direta para dimensões maiores que dois. Para um espaço de divisão de tamanho p, um subconjunto de R_p, o espaço é dividido em M regiões R_m, cada um dos quais é um p-dimensional ou "hiperbloqueio". Não se importe com o texto se pareceu que ficou difícil, apenas olhe as figuras onde é bem mais fácil entender.



Para construir uma árvore de regressão, você usa primeiramente a *divisão binária recursiva* para gerar uma árvore grande com dados de treinamento, parando somente quando cada nó terminal tiver menos do que algum número mínimo de observações. A divisão binária recursiva é um algoritmo ganancioso, de cima para baixo, usado para minimizar a *soma residual dos quadrados* (RSS – *Residual Sum of Squares*), uma medida de erro também usada em configurações de regressão linear. O RSS, no caso de um espaço de recurso particionado com partições M é dado por:

$$\text{RSS} = \sum_{m=1}^{M} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2$$

Começando na parte superior da árvore, você a divide em 2 ramos, criando uma partição de 2 espaços. Você pode, então, realizar esta divisão em particular no topo da árvore várias vezes e escolher a divisão dos recursos que minimiza o RSS (atual).

Em seguida, você aplica *poda de complexidade de custos* para a árvore grande, a fim de obter uma sequência de melhores sub árvores, como uma função de α. A ideia básica aqui é introduzir um parâmetro de ajuste adicional, denotado por **α** que equilibra a profundidade da árvore e sua capacidade de ajuste aos dados de treinamento.

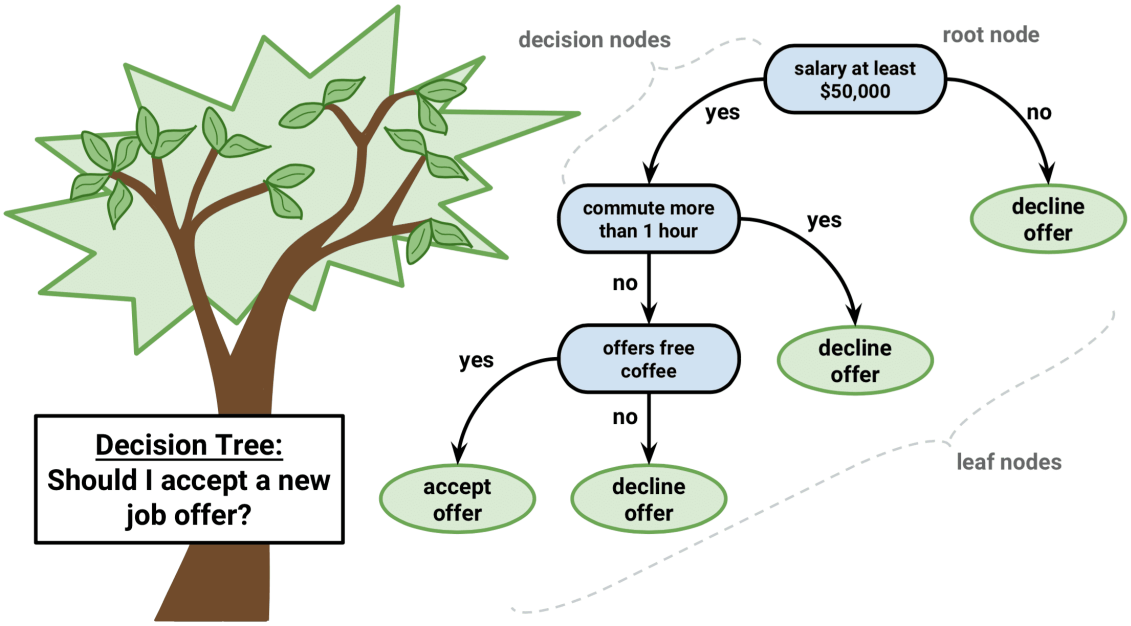
Você pode usar *K-fold cross-validation* para escolher α. Esta técnica simplesmente envolve a divisão das observações de treinamento em **K** conjuntos para estimar a taxa de erro de teste das sub árvores. Seu objetivo é selecionar aquele que leva à menor taxa de erro.

Árvores de Classificação

Uma **Árvore de Classificação** é muito semelhante a uma árvore de regressão, exceto que ela é usada para prever uma resposta qualitativa, em vez de uma resposta quantitativa como ocorre na regressão.

Em uma árvore de regressão, a resposta prevista para uma observação é dada pela resposta média das observações de treinamento que pertencem ao mesmo nó terminal. Em contrapartida, para uma árvore de classificação, você prevê que cada observação pertence à classe mais comumente ocorrendo nas observações de treinamento na região à qual ela pertence.

Ao interpretar os resultados de uma árvore de classificação, você está frequentemente interessado não apenas na predição de classe correspondente a uma região de nó terminal específico, mas também nas proporções de classe entre as observações de treinamento que caem nessa região.



A tarefa de cultivar uma árvore de classificação é bastante semelhante à tarefa de cultivar uma árvore de regressão. Assim como na configuração de regressão, você usa a divisão binária recursiva para fazer crescer a árvore de classificação. No entanto, na configuração de classificação, a *soma residual dos quadrados* (RSS) não pode ser usada como um critério para fazer as divisões binárias. Ao invés disso, você pode usar um dos três métodos descritos a seguir:

- **Taxa de erro de classificação**: em vez de ver até que ponto uma resposta numérica está longe do valor médio, como na configuração de regressão, você pode definir a "taxa de acerto" como a fração de observações de treinamento em uma determinada região que não pertence ao mais a uma determinada Classe. O erro é dado por esta equação:

$$E = 1 - ArgMaxc(\hat{\pi}_{mc})$$

Em que $\hat{\pi}_{mc}$ representa a fração dos dados de treinamento na região R_m que pertencem à classe c.

- **Índice de Gini**: o índice Gini é uma métrica de erro alternativa que foi projetada para mostrar como uma região é "pura". "Pureza", neste caso, significa o quanto dos dados de treinamento em uma determinada região pertencem a uma única classe. Se uma região R_m contém dados que são principalmente de uma única classe *c,* então o valor do índice de Gini será pequeno:

$$G = \sum_{c=1}^{C} \hat{\pi}_{mc}(1 - \hat{\pi}_{mc})$$

- **Entropia cruzada**: uma terceira alternativa, que é semelhante ao índice de Gini, é conhecida como a entropia cruzada:

$$G = \sum_{c=1}^{C} \hat{\pi}_{mc}(1 - \hat{\pi}_{mc})$$

A entropia cruzada irá assumir um valor próximo de zero se o $\hat{\pi}_{mc}$ estão todos perto de 0 ou perto de 1. Portanto, como o índice de Gini, a entropia cruzada terá um valor pequeno se o enésimo nó é puro. De fato, verifica-se que o índice de Gini e a entropia cruzada são bastante semelhantes numericamente.

Ao criar uma árvore de classificação, o índice Gini ou a Entropia Cruzada são normalmente usados para avaliar a qualidade de uma divisão específica, uma vez que são mais sensíveis à pureza do nó do que a taxa de erro de classificação. Qualquer uma dessas três abordagens pode ser usada ao podar a árvore, mas a taxa de erro de classificação é preferível se a precisão da previsão da árvore de poda final for a meta.

Vantagens e Desvantagens das Árvores de Decisão

A principal vantagem de usar árvores de decisão é que elas são intuitivamente muito fáceis de explicar. Elas refletem de perto a tomada de decisão humana em comparação com outras abordagens de regressão e classificação. Elas podem ser exibidas graficamente, e elas podem facilmente lidar com preditores qualitativos sem a necessidade de criar variáveis fictícias.

No entanto, as árvores de decisão geralmente não têm o mesmo nível de precisão preditiva que outras abordagens, uma vez que não são muito robustas. Uma pequena alteração nos dados pode causar uma grande alteração na árvore estimada final.

Agregando muitas árvores de decisão, usando métodos como *bagging*, *random forests* e *boosting*, o desempenho preditivo das árvores de decisão pode ser substancialmente melhorado.

Métodos Baseados em Árvore

Bagging (ensacamento)

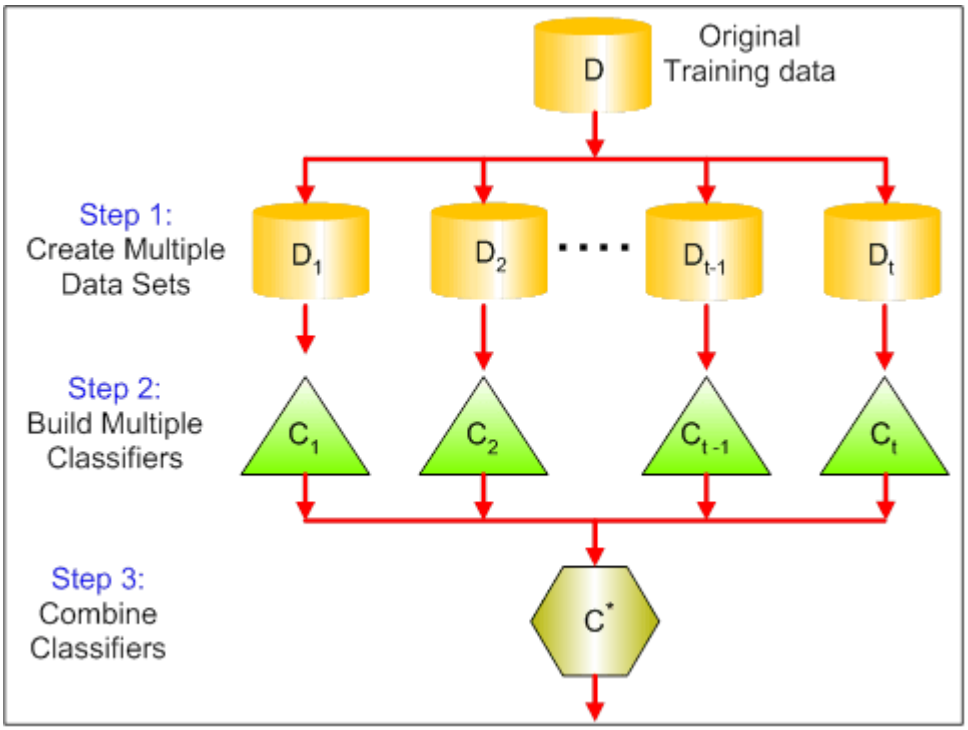
As árvores de decisão discutidas acima sofrem de *alta variância*, ou seja, se você dividir os dados de treinamento em duas partes aleatoriamente, e encaixar uma árvore de decisão em ambas as metades, os resultados que você obtém podem ser bastante diferentes. Em contrapartida, um procedimento com *baixa variância* produzirá resultados semelhantes se aplicados repetidamente a um conjunto de dados distinto.

O bagging ou **ensacamento** ou *agregação de bootstrap*, é uma técnica usada para reduzir a variância de suas previsões, combinando o resultado de vários classificadores modelados em diferentes sub amostras do mesmo conjunto de dados. A seguir é apresentada a equação para o *bagging*:

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

Nesta equação você gera **B** diferentes conjuntos de treinamento *bootstrapped*. Em seguida, você treina seu método no enésimo conjunto de treinamento *bootstrapped*, a fim de obter $\hat{f}_b(x)$ e, finalmente, a média das previsões.

A figura a seguir mostra as 3 etapas diferentes do *bagging*:



1. **Passo 1**: neste passo você substitui os dados originais por novos dados. Os novos dados geralmente têm uma fração das colunas e linhas dos dados originais, que podem ser usadas como parâmetros no modelo de bagging.
2. **Passo 2**: você cria classificadores em cada conjunto de dados. Geralmente, você pode usar o mesmo classificador para fazer modelos e previsões.
3. **Passo 3**: por fim, dependendo do problema, você usa um valor médio para combinar as previsões de todos os classificadores. Geralmente, esses valores combinados são mais robustos e precisos do que um único modelo.

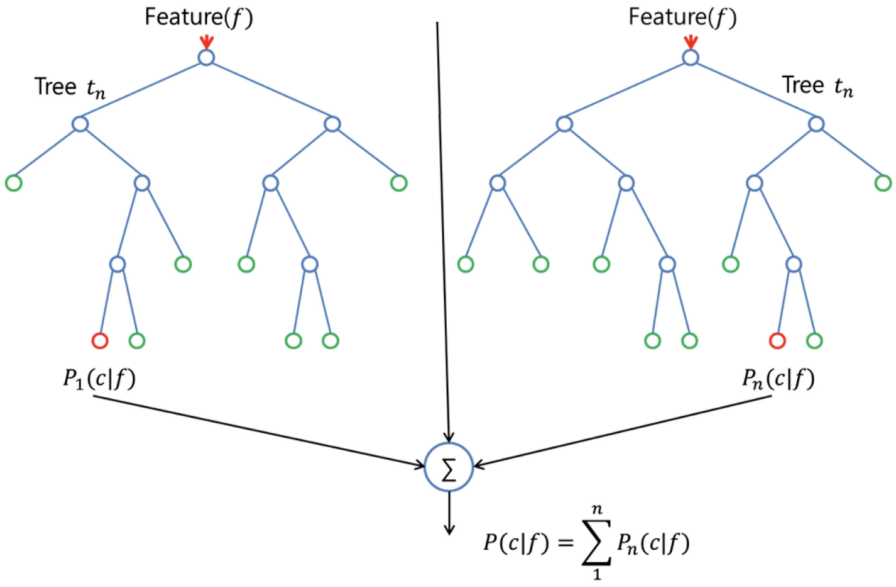
O bagging pode melhorar as previsões para muitos métodos de regressão e classificação, mas é particularmente útil para árvores de decisão. Para aplicar bagging ou ensacamento a árvores de regressão/classificação, você simplesmente constrói B árvores de regressão/classificação usando B conjuntos de treinamento, e calcula a média das previsões resultantes. Estas árvores são cultivadas profundamente, e não são podadas. Desta forma, cada árvore individual tem uma variação elevada, mas baixo viés. A média destas B árvores reduzem a variância.

Em termos gerais, o procedimento de bagging tem demonstrado obter melhorias impressionantes na exatidão do resultado, combinando centenas ou mesmo milhares de árvores em um único procedimento.

Random Forests (florestas aleatórias)

Random Forests é um método de aprendizado de máquina versátil, capaz de executar ambas as tarefas de regressão e classificação. Ele também empreende métodos de redução dimensional, trata valores ausentes, valores *outlier* e outras etapas essenciais de exploração de dados, realizando um trabalho muito bom.

Random forests fornece uma melhoria sobre árvores com bagging por um pequeno ajuste *sobre* as árvores. Como no bagging, você constrói um número de árvores de decisão com amostras extraídas (*bootstrapped*) do treinamento. Mas ao construir essas árvores de decisão, cada vez que uma divisão em uma árvore é considerada, uma *amostra aleatória de preditores* ***m*** é escolhida como candidatos divididos do conjunto completo de p preditores. A divisão deve usar apenas um desses ***m*** preditores. Esta é a principal diferença entre random forests e bagging; porque como no bagging, a escolha do preditor $m=p$.



A fim de cultivar uma random forest, você deve:

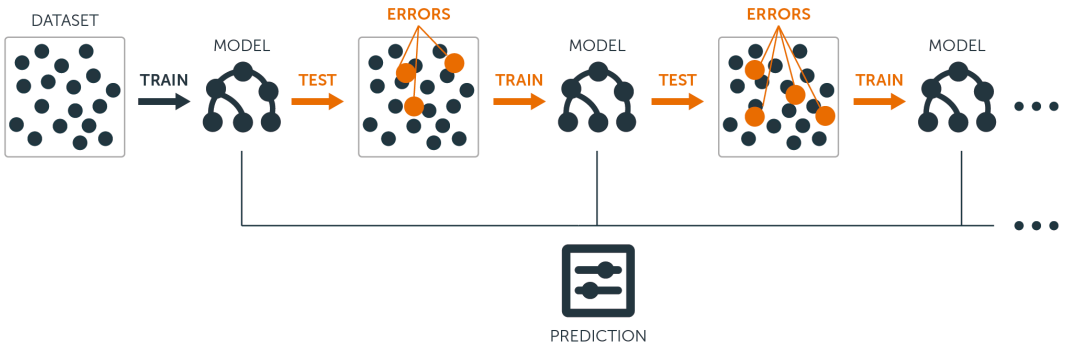
- Primeiro suponha que o número de casos no conjunto de treinamento é **K**. Em seguida, pegue uma amostra aleatória desses **K** casos e use esta amostra como o conjunto de treinamento para o cultivo da árvore.
- Se lá são ***p*** variáveis de entrada, especifique um número **m < p** que em cada nó, você pode selecionar **m** variáveis aleatórias fora do **p**. A melhor divisão destes **m** é usada para dividir o nó.
- Cada árvore é subsequentemente gerada à extensão máxima possível e nenhuma poda é necessária.
- Por fim, agregue as previsões das árvores de destino para prever novos dados.

As random forests são muito eficientes na estimativa de dados ausentes e na manutenção da precisão quando falta uma grande proporção de dados. Elas também podem equilibrar erros em conjuntos de dados, onde as classes são desequilibradas. Mais importante ainda, elas podem lidar com conjuntos de grandes volumes com grande dimensionalidade. No entanto, uma desvantagem do uso de random forests é que você pode facilmente super ajustar conjuntos de dados com muito ruído (ou de baixa qualidade), especialmente no caso de uma regressão.

Boosting

Boosting é outra abordagem para melhorar as previsões resultantes de uma árvore de decisão. Como o bagging e as random forests, é uma aproximação geral pode ser aplicada a muitos métodos de aprendizagem estatísticos para regressão ou classificação. Lembre-se de que o bagging envolve a criação de várias cópias do conjunto de dados de treinamento original usando um bootstrap, ajustando uma árvore de decisão separada para cada cópia e, em seguida, combinando todas as árvores para criar um único modelo preditivo. Notavelmente, cada árvore é construída em um conjunto de dados bootstrapped, independente das outras árvores.

Boosting opera de forma semelhante, exceto que as árvores são cultivadas s*equencialmente*: cada árvore é cultivada usando informações de árvores cultivadas anteriormente. Boosting não envolve amostragem de bootstrap; em vez disso, cada árvore é ajustada em uma versão modificada do conjunto de dados original.



Para ambas as árvores de regressão e classificação, Boosting funciona assim:

- Em vez de encaixar uma única árvore de decisão grande para os dados, o que equivale a forçar o encaixe dos dados e potencialmente gerar super ajuste (overfitting), a abordagem Boosting aprende lentamente.
- Considerando o modelo atual, você ajusta uma árvore de decisão aos resíduos do modelo. Ou seja, você se encaixa em uma árvore usando os resíduos atuais, em vez do resultado Y, como a resposta.
- Você depois adiciona esta árvore de decisão nova na função ajustada a fim atualizar os resíduos. Cada uma dessas árvores pode ser bastante pequena, com apenas alguns nós terminais, determinado pelo parâmetro **d** no algoritmo. Ao encaixar pequenas árvores para os resíduos, você lentamente melhora $\hat{f}$ em áreas onde não tem um bom desempenho.
- A escolha do parâmetro **v** retarda o processo ainda mais, permitindo que mais e diferentes formatos de árvores sejam gerados como forma de atacar os resíduos.

Boosting é muito útil quando você tem muitos dados e espera que as árvores de decisão sejam muito complexas. O Boosting tem sido usado para solucionar muitos problemas desafiadores de classificação e regressão, incluindo análise de risco, análise de sentimento, publicidade preditiva, modelagem de preços, estimativa de vendas e diagnóstico de pacientes, entre outros.

Árvores de Classificação

Instalando Pacotes

Nesta parte, você trabalhará com o conjunto de dados **Carseats**, usando um pacote de Árvore de Decisão do R. Primeiro você precisará instalar o pacote **ISLR** e os pacotes de árvore em seu ambiente do R Studio, conforme instruções a seguir.

O **Carseats** é um **conjunto de dados simulado** disponível nas bibliotecas do R contendo vendas de assentos de carro para crianças (cadeirinhas de segurança) em 400 diferentes lojas.

Vamos primeiro carregar o Carseats dataframe do pacote ISLR. Para isso, digite as instruções abaixo no R Studio.

```
install.packages("ISLR")
library(ISLR)
data(package="ISLR")
carseats<-Carseats
```

Vamos também carregar o pacote de Árvore de Decisão do R.

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

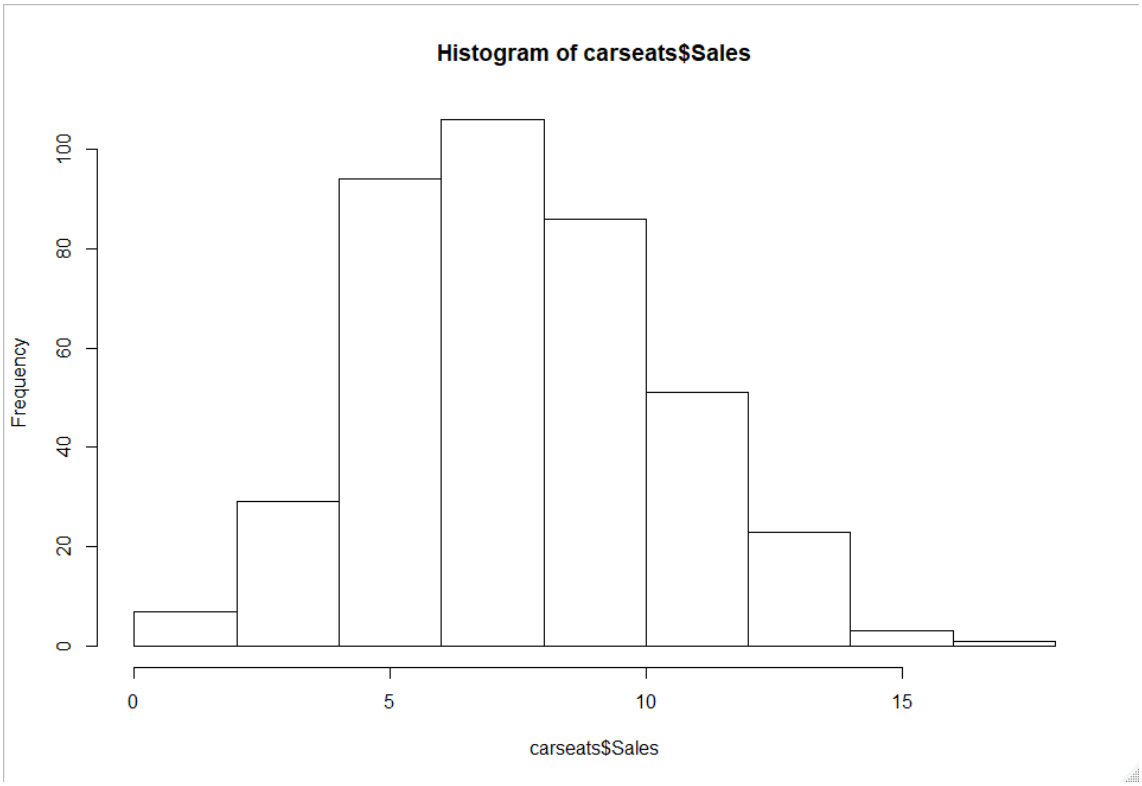
O conjunto de dados Carseats é um dataframe com 400 observações sobre as seguintes 11 variáveis:

- Sales: unidades de cadeirinhas de carro vendidas em milhares;
- CompPrice: preço cobrado pelo concorrente em cada local;
- Income: nível de renda da população em milhares de dólares;
- Advertising: orçamento do anúncio em cada local em milhares de dólares;
- Population: população regional em milhares de pessoas;
- Price: preço para assentos de carro em cada local;
- ShelveLoc: Bad, Good ou Medium, indica a qualidade da posição do produto na prateleira;
- Age: média de idade da população;
- Education: nível de educação na localidade;
- Urban: população urbana Yes/No;
- US: localidade nos Estados Unidos Yes/No

```
names(carseats)
```

Vamos dar uma olhada no histograma de vendas de assentos para carros:

```
hist(carseats $ Sales)
```



Observe que **Sales** é uma variável quantitativa. Vamos demonstrá-la usando árvores com uma resposta binária, ou seja, não quantitativa, mas categórica. Para fazer isso, você deve transformar Sales em uma variável binária, que será chamada **High** (vendas altas). Se as vendas forem iguais ou maiores que 8 mil cadeirinhas, serão consideradas altas (High = Yes). Caso contrário, serão baixas (High = No). Você pode criar esta nova variável **High** no dataframe executando as instruções a seguir.

```
High = ifelse(carseats$Sales<=8, "No", "Yes")
carseats = data.frame(carseats, High)
# Transforma a nova variável em fator
carseats$High <- as.factor(carseats$High)
```

Agora vamos criar um modelo de análise usando árvores de decisão. Você não poderá incluir a variável **Sales** pois sua variável de resposta **High** (vendas altas ou baixas) foi criada a partir de **Sales**. Assim, vamos exclui-la e criar a árvore sem ela.

```
tree.carseats = tree(High~.-Sales, data=carseats)
```

Vamos agora ver o resumo da árvore de classificação que foi criada:

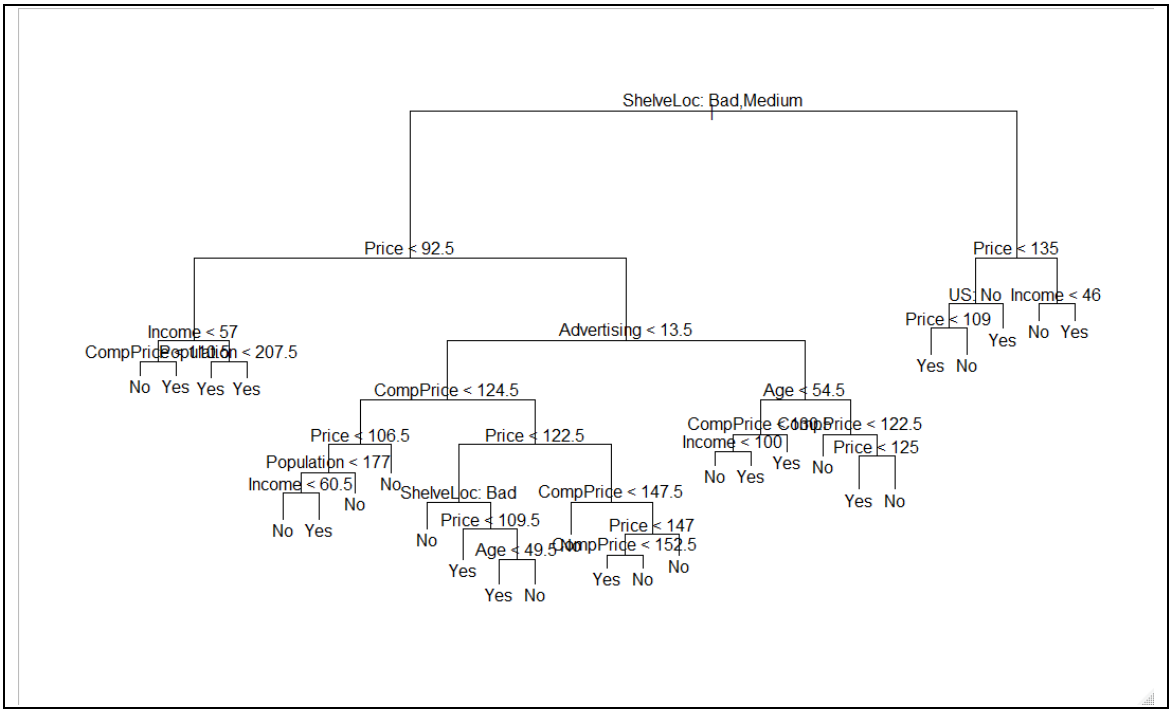
```
summary(tree.carseats)
```

Você pode ver as variáveis envolvidas, o número de nós terminais, o desvio médio residual, bem como a taxa de erro de classificação incorreta (0.09), que é bem baixa.

```
Classification tree:
tree(formula = High ~ . - Sales, data = carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"
"Population"  "Advertising"
[7] "Age"         "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
```

Para torná-la mais visual, vamos plotar a árvore e, em seguida, legendá-la usando uma função texto:

```
plot(tree.carseats)
text(tree.carseats, pretty = 0)
```



Como há muitas variáveis, torna-se um pouco complicado olhar para a árvore em si. Pelo menos, você pode ver que em cada um dos nós terminais, eles estão rotulados como **Yes** ou **No**, correspondendo a vendas Altas (Yes) ou Baixas (No). Em cada nó de divisão da árvore, as variáveis e o valor da opção de divisão são mostrados (por exemplo, Price < 92,5 ou Advertising < 13,5).

Para obter um resumo detalhado da árvore, basta imprimi-lo executando o comando a seguir. Será útil se você quiser extrair detalhes da árvore para outros fins:

```
tree.carseats
```

Como a árvore ficou um pouco complexa (muitos nós e galhos), agora vamos experimentar podá-la. Para isso, vamos criar um conjunto de dados de treinamento e um conjunto de dados de teste dividindo o dataframe **Carseats** em 250 observações para treinamento e 150 observações para amostras de teste. Primeiro, você define uma semente para tornar os resultados reprodutíveis. Em seguida, você pega uma

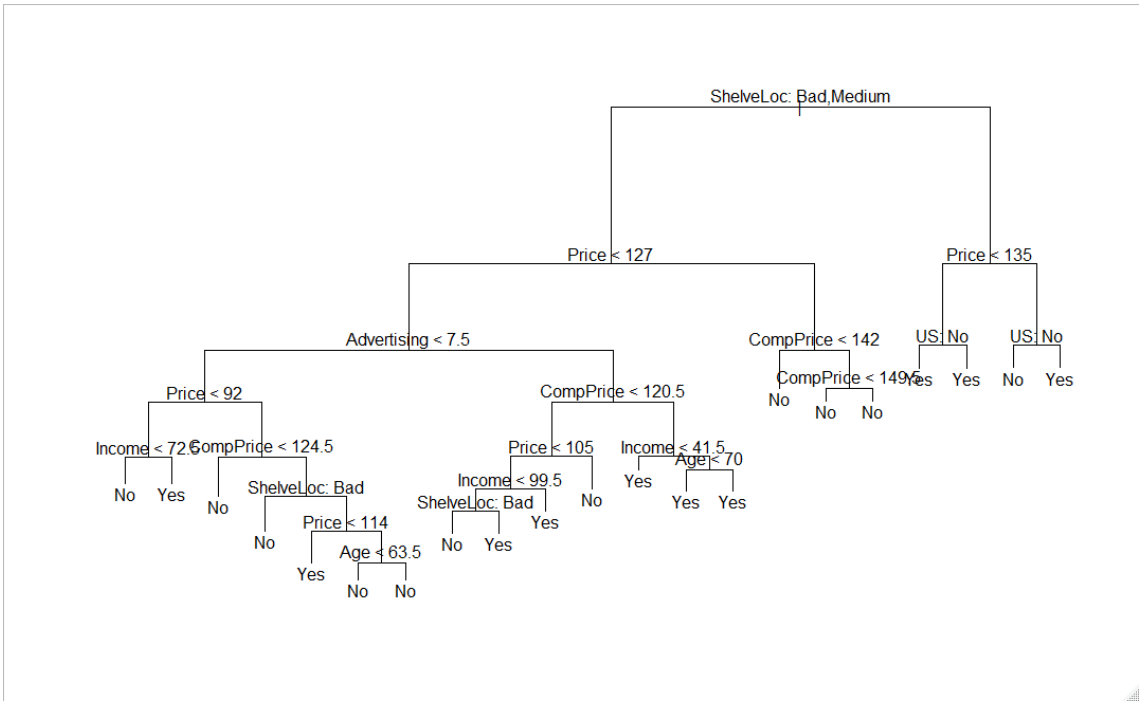
amostra aleatória dos números de ID (índice) das amostras. Especificamente aqui, você amostra o conjunto 1 para n número de linhas de assentos de carro, que é 400. Você deseja uma amostra de tamanho 250 (por padrão, a amostra será sem substituição).

```
set.seed(101)
train=sample(1:nrow(carseats), 250)
```

Agora você tem esse resultado para treinamento, que indexa 250 das 400 observações originais. Você pode reajustar o modelo da árvore, usando a mesma fórmula, exceto informando agora a árvore para usar um subconjunto de dados de treinamento. Vamos agora, então, recriar e plotar a árvore para o subconjunto de dados de treinamento:

```
tree.carseats = tree(High~.-Sales, carseats, subset=train)
plot(tree.carseats)
text(tree.carseats, pretty=0)
```

O gráfico da árvore parece um pouco diferente por causa do conjunto de dados menor (250 agora em vez de 400 antes). No entanto, a complexidade da árvore parece aproximadamente a mesma.



Você vai utilizar esta nova árvore para fazer previsões com o conjunto de dados reservado para teste, usando o método Preview para as árvores. Aqui você vai querer realmente prever a Classe ou Rótulo (Yes ou No) a que cada observação pertence.

```
tree.pred = predict(tree.carseats, carseats[-train,], type="class")
```

Em seguida, você pode avaliar o erro obtido com a classificação efetuada pela árvore usando uma tabela de classificação incorreta ou de confusão.

```
with(carseats[-train,], table(tree.pred, High))
```

Nas diagonais estão as classificações corretas, enquanto fora das diagonais estão as incorretas. Você só quer os registros corretos. Para fazer isso, você pode tomar a soma das 2 diagonais dividido pelo total (150 observações de teste).

```
         High
tree.pred No Yes
      No  72  16
      Yes 19  43
(72 + 43)/150
0,76666
```

Ok, você recebe um erro de classificação de **0,76** com esta árvore.

Ao cultivar uma grande e espessa árvore, pode ocorrer muita variância. Assim, vamos usar a validação cruzada (**cv**) para podar a árvore de forma otimizada. Usando **cv** na árvore você usará o erro de classificação incorreta como a base para fazer a poda.

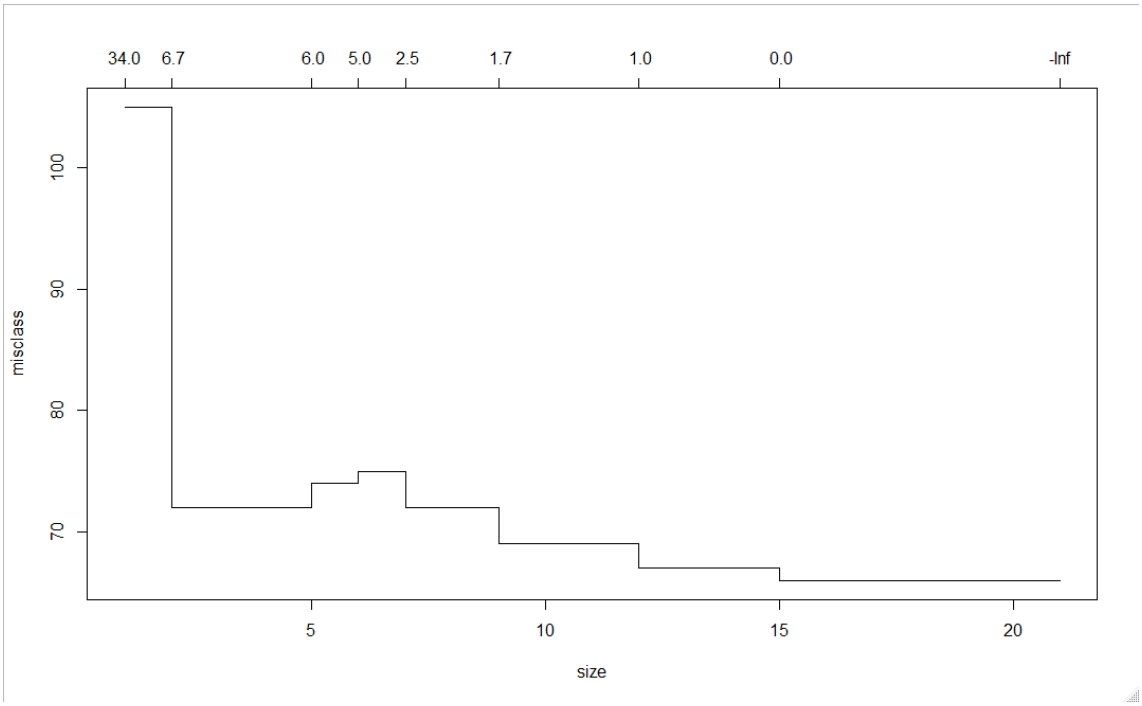
```
cv.carseats = cv.tree(tree.carseats, FUN = prune.misclass)
cv.carseats
```

Ao imprimir os resultados são apresentados os detalhes do caminho da validação cruzada. Você pode ver os tamanhos das árvores como elas foram podadas para trás, os desvios como a poda prosseguiu, bem como o parâmetro de complexidade de custo usado no processo.

```
$size
[1] 21 15 12  9  7  6  5  2  1
$dev
[1]  66  66  67  69  72  75  74  72 105
$k
[1]      -Inf  0.000000  1.000000  1.666667  2.500000  5.000000
6.000000  6.666667 34.000000
$method
[1] "misclass"
attr(,"class")
[1] "prune"         "tree.sequence"
```

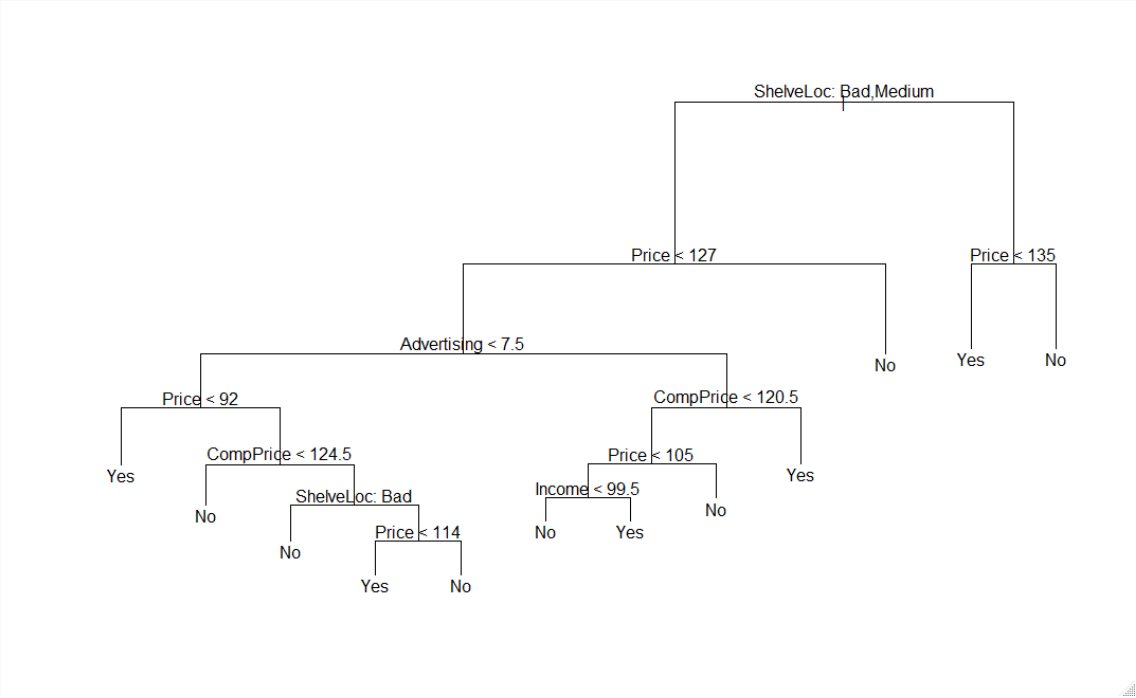
Vamos colocar isso num gráfico:

```
plot(cv.carseats)
```



Olhando para o gráfico, você vê uma espécie de espiral descendente por causa do erro de classificação incorreto em 250 pontos de validação cruzada. Vamos escolher um valor nas etapas para baixo, por exemplo 12. Vamos, então, podar a árvore para um tamanho de 12 para identificar essa árvore. Por fim, vamos plotar e anotar essa árvore para verificar o resultado.

```
prune.carseats = prune.misclass(tree.carseats, best = 12)
plot(prune.carseats)
text(prune.carseats, pretty=0)
```



O resultado é um pouco mais raso do que as árvores anteriores, e agora você pode realmente ler os rótulos. Vamos avaliá-lo no conjunto de dados de teste novamente.

```
Tree.pred = predict(prune.carseats, carseats[-train,], type="class")
with(carseats[-train,], table(tree.pred, High))
         High
tree.pred No Yes
      No  74  20
      Yes 17  39
(74 + 39)/150
```

Parece que as classificações corretas caíram um pouco. Porém, esta nova árvore fez aproximadamente o mesmo trabalho que a árvore original mais complexa, assim a poda não comprometeu muito os erros da classificação, e resultou em uma árvore mais simples.

Muitas vezes, as árvores de decisão não fornecem previsões muito boas, gerando erros de classificação relativamente altos.

No próximo tópico vamos aplicar para um problema de regressão a técnica das florestas aleatórias (Random Forests), que tendem a superar as árvores de decisão, tanto quanto para previsão de valores (regressão) como para a classificação.

Árvores de Regressão

Random Forests

Para elaborar esta parte do laboratório, você usará os dados sobre **bairros em Boston** para explorar a técnica de florestas aleatórias em um problema de regressão, isto é, em vez de utilizar a árvore de decisão para classificar, você vai utilizá-la para estimar ou prever um determinado valor. O conjunto de dados está localizado no pacote **MASS**. Ele fornece valores de habitação e outras estatísticas em cada um dos 506 subúrbios ou bairros da cidade de Boston (EUA) com base em um censo de 1970.

```
library(MASS)
data(package="MASS")
boston<-Boston
dim(boston)
names(boston)
View(boston)
```

Este conjunto de dados contém as seguintes colunas:

- crim: taxa de criminalidade per capita por bairro.
- zn: proporção de terrenos residenciais zoneados para lotes acima de 25.000 sqft.
- indus: proporção de acres comerciais e não comerciais por bairro.
- chas: variável fictícia Charles River (=1 se o trecho é próximo ao rio; =0 em caso contrário).
- nox: concentração de óxidos de nitrogênio (partes por 10 milhões).
- rm: número médio de quartos por habitação.
- age: proporção de idade das unidades ocupadas pelo proprietário construídas antes de 1940.
- dis: média ponderada de distâncias para cinco centros de emprego de Boston.
- rad: índice de acessibilidade às rodovias radiais do bairro.
- tax: imposto propriedade de valor total taxa de imposto por $10000.
- ptratio: relação aluno-professor por bairro.
- black: 1000 $(bk - 0,63)^2$ onde bk é a proporção de negros por bairro.
- lstat: população de baixo status (porcentagem).
- **medv:** valor mediano de casas ocupadas pelo proprietário em \$1000s.

Vamos também carregar o pacote **randomForest**.

```
install.packages("randomForest")
require(randomForest)
```

Para preparar dados para aplicar a técnica de floresta aleatória, vamos definir uma semente e criar um conjunto de treinamento com uma amostra de 300 observações.

```
set.seed(101)
train = sample(1:nrow(boston), 300)
```

Neste conjunto de dados, há 506 bairros de Boston. Para cada bairro, você tem variáveis como crime per capita, média de n º de quartos por habitação, proporção média de idade das casas etc. Vamos usar o dado **medv** - o valor médio das casas ocupadas pelo proprietário para cada um destes bairros, como a variável de resposta, ou seja, para a regressão.

Vamos criar uma floresta aleatória e ver o quão bem ela executa esta regressão. Como mencionado, você usará a resposta **medv**, o valor médio da habitação (em milhares de dólares), e o conjunto de amostra de treinamento.

```
rf.boston = randomForest(medv~., data = boston, subset = train)
rf.boston
```

O resultado é apresentado a seguir (não se preocupe, o resultado pode variar de caso para caso):

```
Call:
 randomForest(formula = medv ~ ., data = boston, subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 12.34243
                    % Var explained: 85.09
```

A impressão da floresta aleatória fornece seu resumo: o n º de árvores (500 foram cultivadas), os resíduos médios quadrados (MSR) e a porcentagem de variância explicada. O MSR e a variância% explicada

baseiam-se em "*out-of-bag estimates*", um dispositivo muito inteligente em florestas aleatórias para obter estimativas honestas de erro.

O único parâmetro de ajuste em uma floresta aleatória é o argumento chamado **mtry**, que é o número de variáveis que são selecionadas em cada divisão de cada árvore quando você faz uma divisão. Como visto aqui, o **mtry** foi de 4 das 13 variáveis exploratórias (excluindo **medv**) nos dados dos bairros de Boston - significando que cada vez que a árvore vem dividir um nó, 4 variáveis seriam selecionadas aleatoriamente, a seguir a separação seria restringida a 1 daquelas 4 variáveis. É assim que o algoritmo de Random Forests correlaciona as árvores.

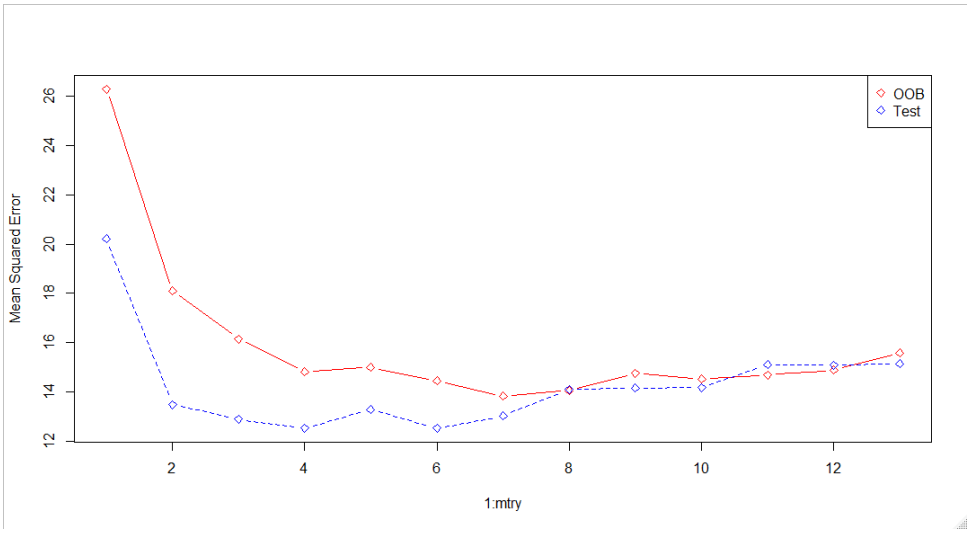
Vamos ajustar uma série de florestas aleatórias. Como há 13 variáveis, vamos ter o **mtry** variando de 1 a 13, conforme segue:

- A fim de gravar os erros obtidos, você configurará 2 variáveis **oob.err** e **test. err**.
- Em um loop de **mtry** de 1 a 13, você primeiro encaixa o randomForest com esse valor de **mtry** no conjunto de dados de treinamento, restringindo o número de árvores a 350.
- Em seguida, extrai o erro médio-quadrado do objeto (*out-of-bag error*).
- Depois você faz a previsão utilizando o conjunto de dados de teste (Boston [-train]) usando **fit** (o ajuste de randomForest).
- Por fim, é calculado o erro de teste: erro médio-quadrado, igual: mean((medv - pred) ^ 2).

```
oob.err = double(13)
test.err = double(13)
for(mtry in 1:13){
  fit = randomForest(medv~., data = boston, subset=train, mtry=mtry,
ntree = 350)
  oob.err[mtry] = fit$mse[350]
  pred = predict(fit, boston[-train,])
  test.err[mtry] = with(boston[-train,], mean( (medv-pred)^2 ))
}
```

Demorou um pouquinho porque você acabou de gerar e cultivar 4.550 árvores (13 vezes 350). Agora vamos fazer um gráfico usando o comando **matplot**. O erro de teste e o erro *out-of-bag* foram estimados juntos para fazer uma matriz de 2 colunas. Há alguns outros argumentos na matriz, incluindo os valores de caracteres de plotagem (PCH = 23 significa diamante preenchido), cores (vermelho e azul), tipo é igual a ambos (plotando ambos os pontos e conectando-os com as linhas) e nome do eixo y (erro quadrático médio). Você também pode colocar uma legenda no canto superior direito do gráfico.

```
matplot(1:mtry, cbind(test.err, oob.err), pch = 23, col = c("red",
"blue"), type = "b", ylab="Mean Squared Error")
legend("topright", legend = c("OOB", "Test"), pch = 23, col = c("red",
"blue"))
```



Idealmente, estas duas curvas deveriam se alinhar, mas parece que o erro de teste é um pouco menor. No entanto, há muita variabilidade nessas estimativas de erro de teste. Como a estimativa de erro fora do bag/saco foi calculada em um conjunto de dados e a estimativa de erro de teste foi calculada em outro conjunto de dados, essas diferenças estão muito bem dentro dos erros esperados.

Observe que a curva vermelha está suavemente acima da curva azul. Estas estimativas de erro são muito correlacionadas, porque o randomForest com **mtry = 4** é muito semelhante ao que tem mtry = 6. É por isso que cada uma das curvas é bastante suave. O que você vê é que o mtry em torno de 4 parece ser a melhor escolha, pelo menos para o erro de teste. Este valor de o mtry para o erro fora do bag/saco é igual a 9.

Assim, com muitos poucos níveis, você tem um modelo de previsão muito poderoso usando florestas aleatórias. Como assim? O lado esquerdo mostra o desempenho de uma única árvore. A média de erro quadrado em fora-do-bag/saco é 26, e você caiu para cerca de 15 (apenas um pouco acima da metade). Isso significa que você reduziu o erro pela metade. Da mesma forma para o erro de teste, você reduziu o erro de 20 para 12.

Boosting

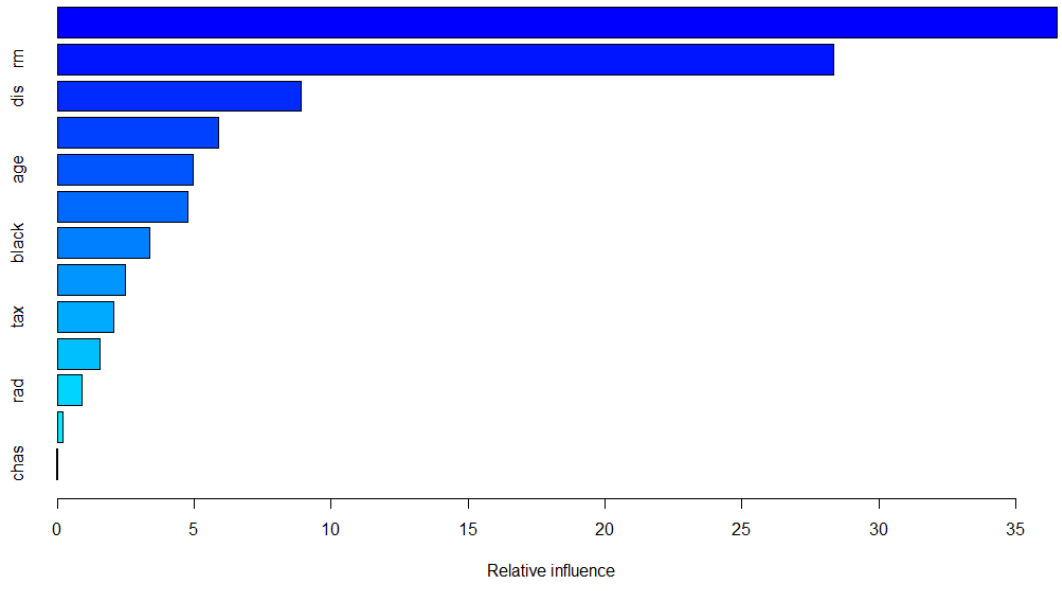
Comparado às florestas aleatórias, a técnica de **boosting** gera árvores menores e mais resistentes ao viés. Você usará o pacote **gbm** (*Gradient Boosted Modeling*), em R.

```
install.packages("gbm")
require(gbm)
library(gbm)
```

O pacote **gbm** utiliza uma distribuição, que é Gaussiana, porque você estará reduzindo a perda de erro ao quadrado. Você vai pedir ao **gbm** para criar 10.000 árvores, o que soa como muito, mas estas árvores vão ser rasas. Profundidade de interação é o número de divisões, então você quer 4 divisões em cada árvore. Encolhimento é 0,01, que é o quanto você vai encolher o passo da árvore.

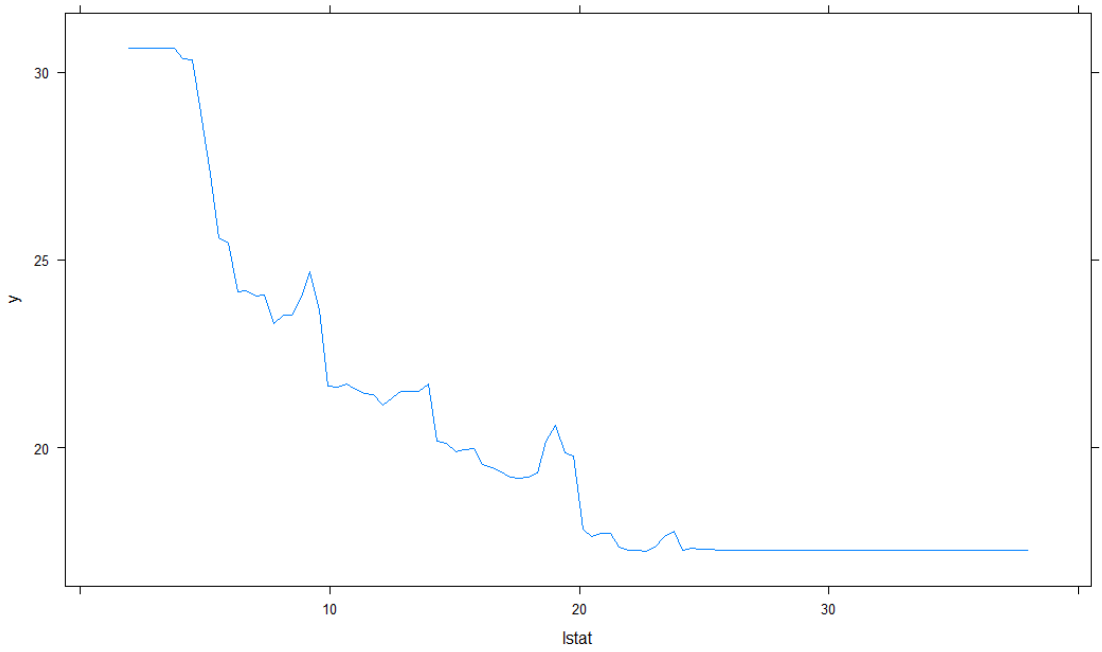
```
boost.boston = gbm(medv~., data = boston[train,], distribution =
"gaussian", n.trees = 10000, shrinkage = 0.01, interaction.depth = 4)
summary(boost.boston)
```

A função *Summary* gera um gráfico de importância por variável. Pelo resultado, parece que existem duas variáveis que têm alta importância relativa: **rm** (número de quartos) e **lstat** (percentagem de pessoas com status econômico inferior no bairro).

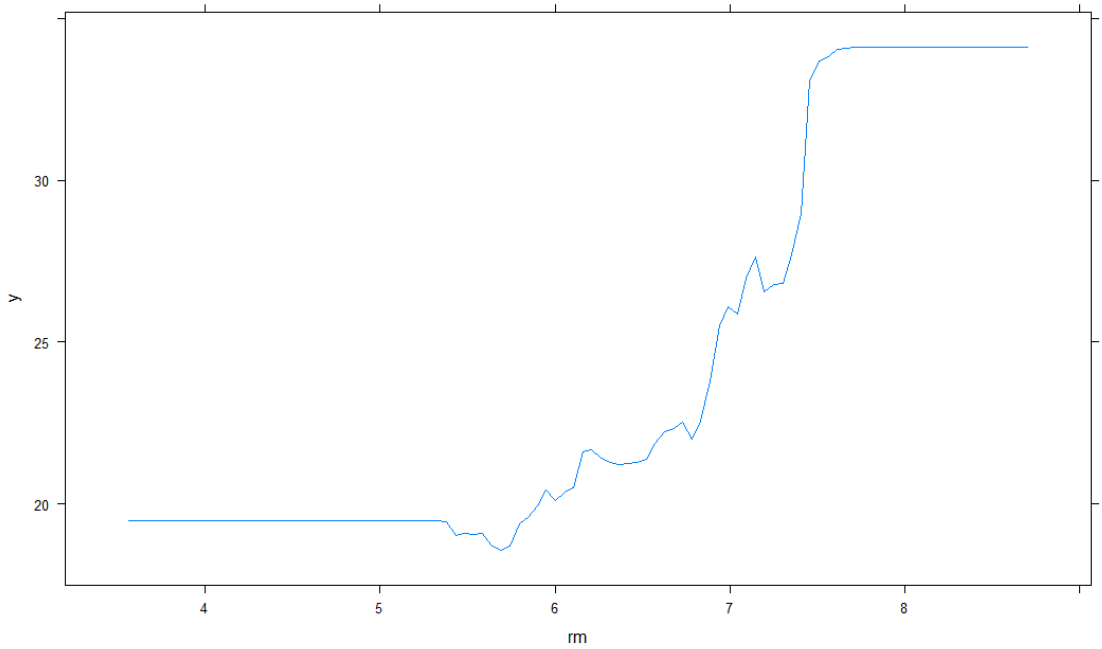


Vamos traçar essas duas variáveis:

```
plot(boost.boston,i="lstat")
plot(boost.boston,i="rm")
```



O gráfico anterior mostra que quanto maior a proporção de pessoas de status inferior no bairro, menor o valor dos preços das habitações. O gráfico a seguir mostra a relação inversa com o número de quartos, ou seja, o número médio de quartos na casa aumenta à medida que o preço aumenta.



Agora já é hora de prever um modelo aprimorado no conjunto de dados de teste. Vamos examinar o desempenho do teste como uma função do número de árvores:

- Primeiro, você faz uma grade com o número de árvores em etapas de 100 de 100 a 10.000.
- Em seguida, você executa a função Preview no modelo boosted. Ele utiliza o número de árvores **n.trees** como um argumento e produz uma matriz de previsões sobre os dados de teste.

- As dimensões da matriz são 206 observações de teste e 100 diferentes vetores de previsão em 100 diferentes valores de árvore.

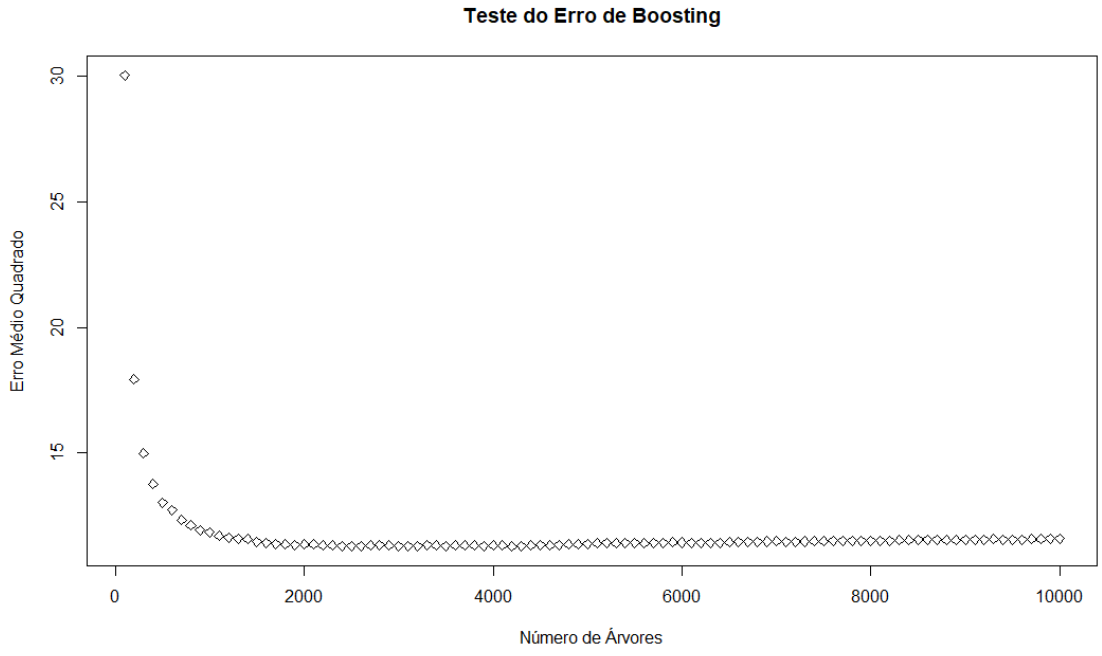
```
n.trees = seq(from = 100, to = 10000, by = 100)
predmat = predict(boost.boston, newdata = boston[-train,], n.trees =
n.trees)
dim(predmat)
```

Agora vamos calcular o erro de teste para cada um dos vetores de previsão:

- predmat é uma matriz, medv é um vetor, assim (predmat - medv) é uma matriz de diferenças. Você pode usar a função **apply** para as colunas dessas diferenças quadradas (a média). Isso calcula o erro quadrado médio de cada coluna para os vetores de previsão.
- Em seguida, você faz um gráfico usando parâmetros semelhantes àqueles usado para Random Forest. Ele mostrará o erro de boosting.

```
boost.err = with(boston[-train,], apply( (predmat - medv)^2, 2, mean))
plot(n.trees, boost.err, pch = 23, ylab = "Erro Médio Quadrado", xlab
= "Número de Árvores", main = "Teste do Erro de Boosting")
```

O erro de boosting cai consideravelmente quando o número de árvores aumenta. Esta é uma evidência que mostra que boosting pode gerar um super ajuste (overfit).



Etapa Final: Relatório de Elaboração do Laboratório

Você deve entregar um relatório com os resultados das etapas elaboradas neste laboratório no e-Disciplinas, para formatá-lo siga estas orientações:

1. Crie um documento Word e identifique-o com o nome do laboratório, data de elaboração e o seu nome ou da dupla que o elaborou;
2. Crie um tópico para cada resultado que você considerar relevante (manipulação de dados ou resultado de algum processamento) identificando-o com um título e uma breve explicação. Os resultados podem ser imagens de gráficos gerados ou de listas de valores ou dados de resultados obtidos. Não devem ser incluídos os scripts ou instruções de processamento utilizados, inclua apenas os resultados que você considerar relevantes.
3. No final do relatório crie um último tópico denominado “Conclusões” e elabore comentários, sugestões e conclusões sobre o que você pode aprender com a elaboração deste laboratório.

Conclusão

Parabéns! Você concluiu com sucesso o Laboratório R sobre a sobre a construção de modelos de árvore de decisão: árvores de classificação, florestas aleatórias e boosted trees. Os últimos dois são métodos poderosos que você pode usar a qualquer momento conforme necessário. Geralmente boosted supera RandomForest, mas RandomForest é mais fácil de implementar. Em RandomForest, o único parâmetro de ajuste é o número de árvores; enquanto no boosted, mais parâmetros de ajuste são necessários além do número de árvores, incluindo o encolhimento e a profundidade de interação.

Referências

- https://www.datacamp.com/community/tutorials/decision-trees-R