Universidade Federal do Paraná

Laboratório de Estatística e Geoinformação - LEG

Métodos baseados em árvores

Prof.: Eduardo Vargas Ferreira

Introdução

- Nesta seção vamos descrever os métodos baseados em **árvores** no contexto de regressão e classificação;

- Estes envolvem **estratificação** ou **segmentação** do espaço de predição em regiões simples;

- Como o conjunto de regras que regem as separação do espaço pode ser resumida em uma árvore, são conhecidas como **árvores de decisão**;

Introdução

- Nesta seção vamos descrever os métodos baseados em **árvores** no contexto de regressão e classificação;

- Estes envolvem **estratificação** ou **segmentação** do espaço de predição em regiões simples;

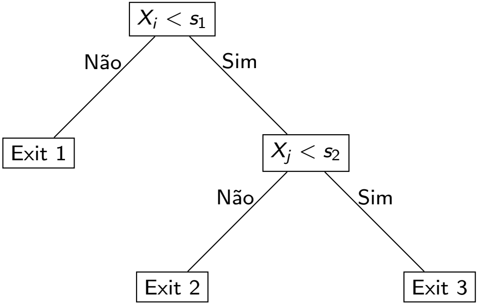
- Como o conjunto de regras que regem as separação do espaço pode ser resumida em uma árvore, são conhecidas como **árvores de decisão**;

Introdução

- Nesta seção vamos descrever os métodos baseados em **árvores** no contexto de regressão e classificação;

- Estes envolvem **estratificação** ou **segmentação** do espaço de predição em regiões simples;

- Como o conjunto de regras que regem as separação do espaço pode ser resumida em uma árvore, são conhecidas como **árvores de decisão**;



Prós e contras

✓ Podem ser aplicado em problemas de regressão e classificação;

✓ Não precisa de variáveis dummy para lidar com preditores qualitativos;

✓ São simples e úteis para interpretação. Não temos equações (a árvore é o próprio modelo), o que torna atrativo especialmente para não estatísticos;

χ São mais simples do que deveria ser, por este motivo não são competitivos com outras abordagens de aprendizado supervisionado em termos de predição;

✓ Por este motivo, discutiremos outros métodos que combinam várias árvores, como **bagging**, **random forests** e **boosting**;

Observação: ao combinar um grande número de árvores pode resultar em melhorias na predição, à custa da perda da interpretabilidade.

Prós e contras

✓ Podem ser aplicado em problemas de regressão e classificação;

✓ Não precisa de variáveis dummy para lidar com preditores qualitativos;

✓ São simples e úteis para interpretação. Não temos equações (a árvore é o próprio modelo), o que torna atrativo especialmente para não estatísticos;

χ São mais simples do que deveria ser, por este motivo não são competitivos com outras abordagens de aprendizado supervisionado em termos de predição;

✓ Por este motivo, discutiremos outros métodos que combinam várias árvores, como **bagging**, **random forests** e **boosting**;

Observação: ao combinar um grande número de árvores pode resultar em melhorias na predição, à custa da perda da interpretabilidade.

Prós e contras

✓ Podem ser aplicado em problemas de regressão e classificação;

✓ Não precisa de variáveis dummy para lidar com preditores qualitativos;

✓ São simples e úteis para interpretação. Não temos equações (a árvore é o próprio modelo), o que torna atrativo especialmente para não estatísticos;

χ São mais simples do que deveria ser, por este motivo não são competitivos com outras abordagens de aprendizado supervisionado em termos de predição;

✓ Por este motivo, discutiremos outros métodos que combinam várias árvores, como **bagging**, **random forests** e **boosting**;

Observação: ao combinar um grande número de árvores pode resultar em melhorias na predição, à custa da perda da interpretabilidade.

Prós e contras

✓ Podem ser aplicado em problemas de regressão e classificação;

✓ Não precisa de variáveis dummy para lidar com preditores qualitativos;

✓ São simples e úteis para interpretação. Não temos equações (a árvore é o próprio modelo), o que torna atrativo especialmente para não estatísticos;

χ São mais simples do que deveria ser, por este motivo não são competitivos com outras abordagens de aprendizado supervisionado em termos de predição;

✓ Por este motivo, discutiremos outros métodos que combinam várias árvores, como **bagging**, **random forests** e **boosting**;

Observação: ao combinar um grande número de árvores pode resultar em melhorias na predição, à custa da perda da interpretabilidade.

Prós e contras

✓ Podem ser aplicado em problemas de regressão e classificação;

✓ Não precisa de variáveis dummy para lidar com preditores qualitativos;

✓ São simples e úteis para interpretação. Não temos equações (a árvore é o próprio modelo), o que torna atrativo especialmente para não estatísticos;

χ São mais simples do que deveria ser, por este motivo não são competitivos com outras abordagens de aprendizado supervisionado em termos de predição;

✓ Por este motivo, discutiremos outros métodos que combinam várias árvores, como **bagging**, **random forests** e **boosting**;

Observação: ao combinar um grande número de árvores pode resultar em melhorias na predição, à custa da perda da interpretabilidade.

Prós e contras

✓ Podem ser aplicado em problemas de regressão e classificação;

✓ Não precisa de variáveis dummy para lidar com preditores qualitativos;

✓ São simples e úteis para interpretação. Não temos equações (a árvore é o próprio modelo), o que torna atrativo especialmente para não estatísticos;

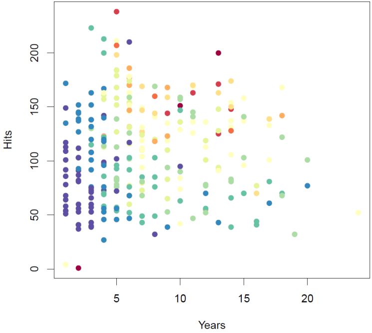
χ São mais simples do que deveria ser, por este motivo não são competitivos com outras abordagens de aprendizado supervisionado em termos de predição;

✓ Por este motivo, discutiremos outros métodos que combinam várias árvores, como **bagging**, **random forests** e **boosting**;

Observação: ao combinar um grande número de árvores pode resultar em melhorias na predição, à custa da perda da interpretabilidade.

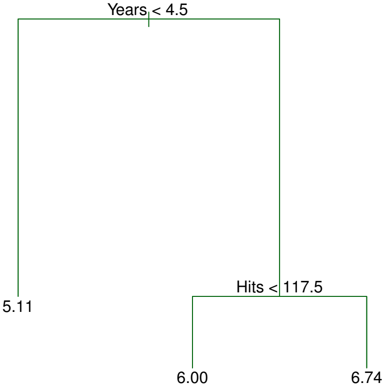
Exemplo: `Hitters` data set - Baseball salary

- Queremos prever o salário dos jogadores (`Salary`) baseado no tempo em que está na *major leagues* (`Years`) e número de acertos no ano (`Hits`);
- O salário codificado como baixo é representado pelas cores azul e verde, e alto pelas cores amarelo e vermelho;
- Como podemos estratificar estes dados?



Exemplo: `Hitters` data set - Baseball salary

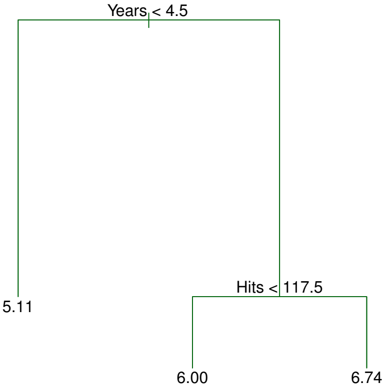
- No topo temos todos os dados;
- `Years` < 4.5 representa a primeira partição;
- O comprimento de cada ramo representa o decréscimo na SQRes;
- Por isso, temos "braços" cada vez menores;
- Ao final chega-se em três caixas: salário baixo, médio e alto;



- Note que `Hits` foi importante para determinar o salário médio de jogadores com mais de 4.5 anos na *major leagues*;

Exemplo: `Hitters` data set - Baseball salary

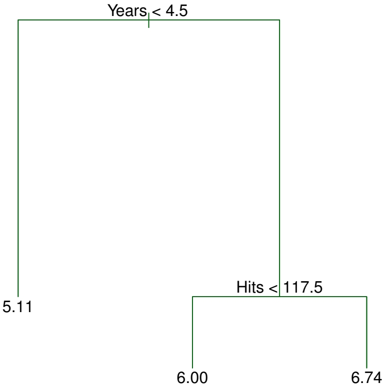
- No topo temos todos os dados;
- `Years` < 4.5 representa a primeira partição;
- O comprimento de cada ramo representa o decréscimo na SQRes;
- Por isso, temos "braços" cada vez menores;
- Ao final chega-se em três caixas: salário baixo, médio e alto;



- Note que `Hits` foi importante para determinar o salário médio de jogadores com mais de 4.5 anos na *major leagues*;

Exemplo: Hitters data set - Baseball salary

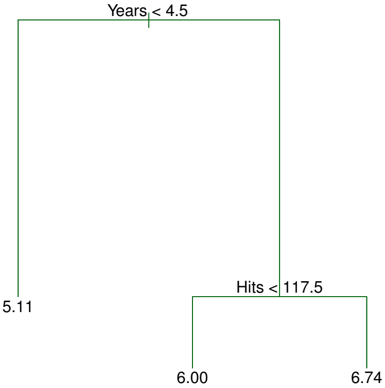
- No topo temos todos os dados;
- Years < 4.5 representa a primeira partição;
- O comprimento de cada ramo representa o decréscimo na SQRes;
- Por isso, temos "braços" cada vez menores;
- Ao final chega-se em três caixas: salário baixo, médio e alto;



- Note que Hits foi importante para determinar o salário médio de jogadores com mais de 4.5 anos na *major leagues*;

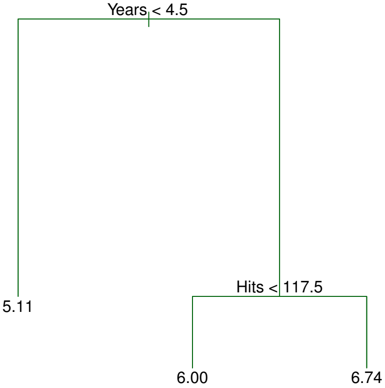
Exemplo: Hitters data set - Baseball salary

- No topo temos todos os dados;
- Years < 4.5 representa a primeira partição;
- O comprimento de cada ramo representa o decréscimo na SQRes;
- Por isso, temos "braços" cada vez menores;
- Ao final chega-se em três caixas: salário baixo, médio e alto;
- Note que Hits foi importante para determinar o salário médio de jogadores com mais de 4.5 anos na *major leagues*;



Exemplo: Hitters data set - Baseball salary

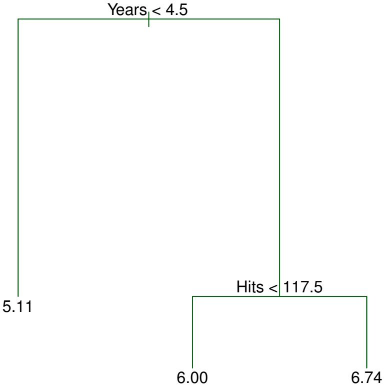
- No topo temos todos os dados;
- Years < 4.5 representa a primeira partição;
- O comprimento de cada ramo representa o decréscimo na SQRes;
- Por isso, temos "braços" cada vez menores;
- Ao final chega-se em três caixas: salário baixo, médio e alto;



- Note que Hits foi importante para determinar o salário médio de jogadores com mais de 4.5 anos na *major leagues*;

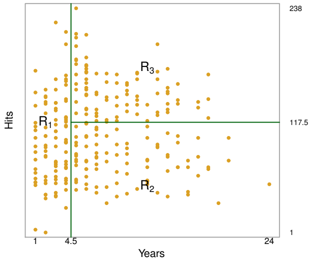
Exemplo: Hitters data set - Baseball salary

- No topo temos todos os dados;
- Years < 4.5 representa a primeira partição;
- O comprimento de cada ramo representa o decréscimo na SQRes;
- Por isso, temos "braços" cada vez menores;
- Ao final chega-se em três caixas: salário baixo, médio e alto;



- Note que Hits foi importante para determinar o salário médio de jogadores com mais de 4.5 anos na *major leagues*;

Interpretação dos resultados



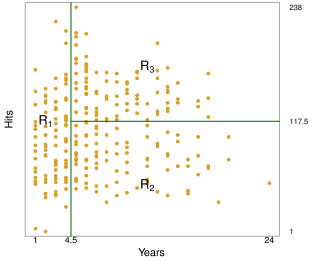
$$R_1 = \{X | \texttt{Years} < 4.5\}$$

$$R_2 = \{X | \texttt{Years} \geq 4.5, \texttt{Hits} < 117.5\}$$

$$R_3 = \{X | \texttt{Years} \geq 4.5, \texttt{Hits} \geq 117.5\}$$

- `Years` é o fator mais importante para determinar `Salary`;
- Dado que o jogador tem pouca experiência, o número de `Hits` no ano anterior não parece influenciar no salário;
- Mas, entre os jogadores que estão na *major leagues* por mais de 4.5 anos, o número de `Hits` afeta positivamente no salário;
- Certamente, esta abordagem é uma simplificação exagerada dos modelos de regressão. Todavia, é fácil de exibir, interpretar e explicar;

Interpretação dos resultados



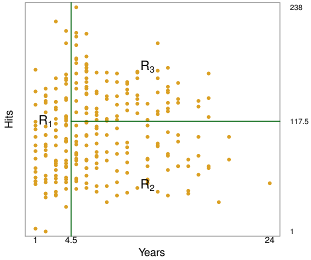
$$R_1 = \{X | \texttt{Years} < 4.5\}$$

$$R_2 = \{X | \texttt{Years} \geq 4.5, \texttt{Hits} < 117.5\}$$

$$R_3 = \{X | \texttt{Years} \geq 4.5, \texttt{Hits} \geq 117.5\}$$

- `Years` é o fator mais importante para determinar `Salary`;
- Dado que o jogador tem pouca experiência, o número de `Hits` no ano anterior não parece influenciar no salário;
- Mas, entre os jogadores que estão na *major leagues* por mais de 4.5 anos, o número de `Hits` afeta positivamente no salário;
- Certamente, esta abordagem é uma simplificação exagerada dos modelos de regressão. Todavia, é fácil de exibir, interpretar e explicar;

Interpretação dos resultados



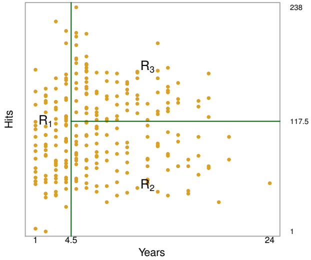
$$R_1 = \{X | \texttt{Years} < 4.5\}$$

$$R_2 = \{X | \texttt{Years} \geq 4.5, \texttt{Hits} < 117.5\}$$

$$R_3 = \{X | \texttt{Years} \geq 4.5, \texttt{Hits} \geq 117.5\}$$

- `Years` é o fator mais importante para determinar `Salary`;
- Dado que o jogador tem pouca experiência, o número de `Hits` no ano anterior não parece influenciar no salário;
- Mas, entre os jogadores que estão na *major leagues* por mais de 4.5 anos, o número de `Hits` afeta positivamente no salário;
- Certamente, esta abordagem é uma simplificação exagerada dos modelos de regressão. Todavia, é fácil de exibir, interpretar e explicar;

Interpretação dos resultados



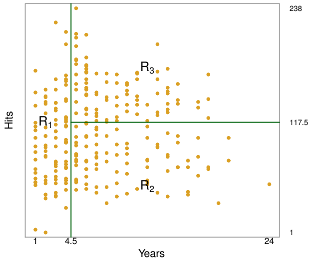
$$R_1 = \{X|\texttt{Years} < 4.5\}$$

$$R_2 = \{X|\texttt{Years} \geq 4.5, \texttt{Hits} < 117.5\}$$

$$R_3 = \{X|\texttt{Years} \geq 4.5, \texttt{Hits} \geq 117.5\}$$

- `Years` é o fator mais importante para determinar `Salary`;
- Dado que o jogador tem pouca experiência, o número de `Hits` no ano anterior não parece influenciar no salário;
- Mas, entre os jogadores que estão na *major leagues* por mais de 4.5 anos, o número de `Hits` afeta positivamente no salário;
- Certamente, esta abordagem é uma simplificação exagerada dos modelos de regressão. Todavia, é fácil de exibir, interpretar e explicar;

Interpretação dos resultados



$$R_1 = \{X | \texttt{Years} < 4.5\}$$

$$R_2 = \{X | \texttt{Years} \geq 4.5, \texttt{Hits} < 117.5\}$$

$$R_3 = \{X | \texttt{Years} \geq 4.5, \texttt{Hits} \geq 117.5\}$$

- `Years` é o fator mais importante para determinar `Salary`;
- Dado que o jogador tem pouca experiência, o número de `Hits` no ano anterior não parece influenciar no salário;
- Mas, entre os jogadores que estão na *major leagues* por mais de 4.5 anos, o número de `Hits` afeta positivamente no salário;
- Certamente, esta abordagem é uma simplificação exagerada dos modelos de regressão. Todavia, é fácil de exibir, interpretar e explicar;

Como o algoritmo funciona?

- A construção da árvore de decisão é composta por dois passos:

  1. Dividimos o espaço dos preditores $X_1, X_2, \ldots, X_p$ em J regiões distintas, $R_1, R_2, \ldots, R_J$;

  2. Para cada observação que caia na região R_j fazemos a predição, que é a resposta média para as observações de treino em R_j.

- P. ex., suponha que no Passo 1 obtemos duas regiões, R_1 e R_2, e a resposta média da região 1 é 10, enquanto da região 2 é 20;

- Assim, dada uma observação $X = x$, se $x \in R_1$ prevemos o valor como 10, caso contrário como 20;

- Mas, como construir as regiões $R_1, \ldots, R_J$?

Como o algoritmo funciona?

- A construção da árvore de decisão é composta por dois passos:

  1. Dividimos o espaço dos preditores $X_1, X_2, \ldots, X_p$ em J regiões distintas, $R_1, R_2, \ldots, R_J$;

  2. Para cada observação que caia na região R_j fazemos a predição, que é a resposta média para as observações de treino em R_j.

- P. ex., suponha que no Passo 1 obtemos duas regiões, R_1 e R_2, e a resposta média da região 1 é 10, enquanto da região 2 é 20;

- Assim, dada uma observação $X = x$, se $x \in R_1$ prevemos o valor como 10, caso contrário como 20;

- Mas, como construir as regiões $R_1, \ldots, R_J$?

Como o algoritmo funciona?

- A construção da árvore de decisão é composta por dois passos:

  1. Dividimos o espaço dos preditores $X_1, X_2, \ldots, X_p$ em J regiões distintas, $R_1, R_2, \ldots, R_J$;

  2. Para cada observação que caia na região R_j fazemos a predição, que é a resposta média para as observações de treino em R_j.

- P. ex., suponha que no Passo 1 obtemos duas regiões, R_1 e R_2, e a resposta média da região 1 é 10, enquanto da região 2 é 20;

- Assim, dada uma observação $X = x$, se $x \in R_1$ prevemos o valor como 10, caso contrário como 20;

- Mas, como construir as regiões $R_1, \ldots, R_J$?

Como o algoritmo funciona?

- A construção da árvore de decisão é composta por dois passos:

  1. Dividimos o espaço dos preditores $X_1, X_2, \ldots, X_p$ em J regiões distintas, $R_1, R_2, \ldots, R_J$;

  2. Para cada observação que caia na região R_j fazemos a predição, que é a resposta média para as observações de treino em R_j.

- P. ex., suponha que no Passo 1 obtemos duas regiões, R_1 e R_2, e a resposta média da região 1 é 10, enquanto da região 2 é 20;

- Assim, dada uma observação $X = x$, se $x \in R_1$ prevemos o valor como 10, caso contrário como 20;

- Mas, como construir as regiões $R_1, \ldots, R_J$?

Como o algoritmo funciona?

- A construção da árvore de decisão é composta por dois passos:

  1. Dividimos o espaço dos preditores $X_1, X_2, \ldots, X_p$ em J regiões distintas, $R_1, R_2, \ldots, R_J$;

  2. Para cada observação que caia na região R_j fazemos a predição, que é a resposta média para as observações de treino em R_j.

- P. ex., suponha que no Passo 1 obtemos duas regiões, R_1 e R_2, e a resposta média da região 1 é 10, enquanto da região 2 é 20;

- Assim, dada uma observação $X = x$, se $x \in R_1$ prevemos o valor como 10, caso contrário como 20;

- Mas, como construir as regiões $R_1, \ldots, R_J$?

Como o algoritmo funciona?

- A construção da árvore de decisão é composta por dois passos:

  1. Dividimos o espaço dos preditores $X_1, X_2, \ldots, X_p$ em J regiões distintas, $R_1, R_2, \ldots, R_J$;

  2. Para cada observação que caia na região R_j fazemos a predição, que é a resposta média para as observações de treino em R_j.

- P. ex., suponha que no Passo 1 obtemos duas regiões, R_1 e R_2, e a resposta média da região 1 é 10, enquanto da região 2 é 20;

- Assim, dada uma observação $X = x$, se $x \in R_1$ prevemos o valor como 10, caso contrário como 20;

- Mas, como construir as regiões $R_1, \ldots, R_J$?

Como o algoritmo funciona?

- Em teoria, as regiões podem ter qualquer forma;
- Todavia, escolhemos dividir o espaço dos preditores em retângulos (por simplicidade e facilidade de interpretação);
- O objetivo é encontrar os retângulos $R_1, \ldots, R_J$ que minimiza a *SQRes*:

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

  em que $\hat{y}_{R_j}$ é a resposta média das observações de treino dentro do j-ésimo retângulo.
- Infelizmente, é computacionalmente inviável considerar toda possível partição do espaço em J retângulos;
- Por esta razão, utilizamos a abordagem da **divisão binária recursiva** (ou *top-down approach*).

Como o algoritmo funciona?

- Em teoria, as regiões podem ter qualquer forma;
- Todavia, escolhemos dividir o espaço dos preditores em retângulos (por simplicidade e facilidade de interpretação);
- O objetivo é encontrar os retângulos $R_1, \ldots, R_J$ que minimiza a *SQRes*:

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

  em que $\hat{y}_{R_j}$ é a resposta média das observações de treino dentro do j-ésimo retângulo.
- Infelizmente, é computacionalmente inviável considerar toda possível partição do espaço em J retângulos;
- Por esta razão, utilizamos a abordagem da **divisão binária recursiva** (ou *top-down approach*).

Como o algoritmo funciona?

- Em teoria, as regiões podem ter qualquer forma;
- Todavia, escolhemos dividir o espaço dos preditores em retângulos (por simplicidade e facilidade de interpretação);
- O objetivo é encontrar os retângulos $R_1, \ldots, R_J$ que minimiza a *SQRes*:

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

  em que $\hat{y}_{R_j}$ é a resposta média das observações de treino dentro do j-ésimo retângulo.
- Infelizmente, é computacionalmente inviável considerar toda possível partição do espaço em J retângulos;
- Por esta razão, utilizamos a abordagem da **divisão binária recursiva** (ou *top-down approach*).

Como o algoritmo funciona?

- Em teoria, as regiões podem ter qualquer forma;
- Todavia, escolhemos dividir o espaço dos preditores em retângulos (por simplicidade e facilidade de interpretação);
- O objetivo é encontrar os retângulos $R_1, \ldots, R_J$ que minimiza a *SQRes*:

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

  em que $\hat{y}_{R_j}$ é a resposta média das observações de treino dentro do j-ésimo retângulo.
- Infelizmente, é computacionalmente inviável considerar toda possível partição do espaço em J retângulos;
- Por esta razão, utilizamos a abordagem da **divisão binária recursiva** (ou *top-down approach*).

Como o algoritmo funciona?

- Em teoria, as regiões podem ter qualquer forma;
- Todavia, escolhemos dividir o espaço dos preditores em retângulos (por simplicidade e facilidade de interpretação);
- O objetivo é encontrar os retângulos $R_1, \ldots, R_J$ que minimiza a *SQRes*:

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

  em que $\hat{y}_{R_j}$ é a resposta média das observações de treino dentro do j-ésimo retângulo.

- Infelizmente, é computacionalmente inviável considerar toda possível partição do espaço em J retângulos;
- Por esta razão, utilizamos a abordagem da **divisão binária recursiva** (ou *top-down approach*).

Divisão binária recursiva

- Primeiro, para todos os preditores $X_1, \dots, X_p$ e todos os possíveis valores do ponto de corte, s, calculamos a $SQRes$;

- Em seguida, selecionamos X_j e o ponto de corte tal que a árvore resulte na menor $SQRes$. Ou seja, dada as regiões

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

procuramos o valor de j e s que minimize a equação

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

- Repetimos o processo agora com a partição dos dados;

- E paramos quando algum critério seja alcançado (p.ex., até que nenhuma região tenha mais de 5 observações.

Divisão binária recursiva

- Primeiro, para todos os preditores $X_1, \ldots, X_p$ e todos os possíveis valores do ponto de corte, s, calculamos a $SQRes$;

- Em seguida, selecionamos X_j e o ponto de corte tal que a árvore resulte na menor $SQRes$. Ou seja, dada as regiões

$$R_1(j, s) = \{X | X_j < s\} \quad \text{e} \quad R_2(j, s) = \{X | X_j \geq s\},$$

procuramos o valor de j e s que minimize a equação

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

- Repetimos o processo agora com a partição dos dados;

- E paramos quando algum critério seja alcançado (p.ex., até que nenhuma região tenha mais de 5 observações.

Divisão binária recursiva

- Primeiro, para todos os preditores $X_1, \ldots, X_p$ e todos os possíveis valores do ponto de corte, s, calculamos a $SQRes$;

- Em seguida, selecionamos X_j e o ponto de corte tal que a árvore resulte na menor $SQRes$. Ou seja, dada as regiões

$$R_1(j,s) = \{X | X_j < s\} \quad \text{e} \quad R_2(j,s) = \{X | X_j \geq s\},$$

  procuramos o valor de j e s que minimize a equação

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

- Repetimos o processo agora com a partição dos dados;

- E paramos quando algum critério seja alcançado (p.ex., até que nenhuma região tenha mais de 5 observações.

Divisão binária recursiva

- Primeiro, para todos os preditores $X_1, \ldots, X_p$ e todos os possíveis valores do ponto de corte, s, calculamos a $SQRes$;

- Em seguida, selecionamos X_j e o ponto de corte tal que a árvore resulte na menor $SQRes$. Ou seja, dada as regiões

$$R_1(j, s) = \{X | X_j < s\} \quad \text{e} \quad R_2(j, s) = \{X | X_j \geq s\},$$

  procuramos o valor de j e s que minimize a equação

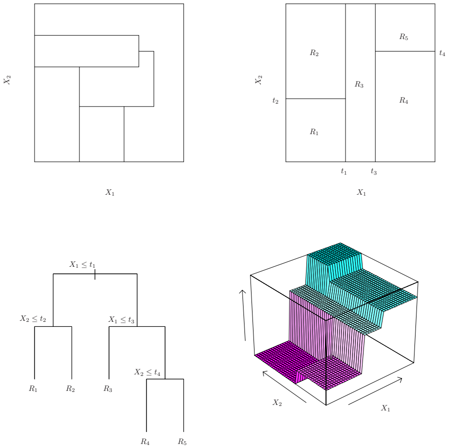
$$\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

- Repetimos o processo agora com a partição dos dados;

- E paramos quando algum critério seja alcançado (p.ex., até que nenhuma região tenha mais de 5 observações.

Exemplo simulado

- Abaixo um exemplo com cinco regiões. Note que o gráfico superior esquerdo não resulta em uma divisão binária recursiva.



Podando a árvore

- Se o modelo se adaptar muito aos dados (**overfit**), o processo anterior pode produzir boas predições no treino, e um mau desempenho no teste;
- Sendo assim, árvores menores (com menos regiões), embora apresente maior viés, conduz à menor variabilidade e melhor interpretação;
- O processo de **poda da árvore** é semelhante ao Lasso, i.e., continuamos a penalizar o modelo pela sua complexidade;
- Se antes buscávamos β's pequenos para evitar o *overfit*, agora penalizamos pelo número de regiões;
- A estratégia é a partir de uma grande árvore T_0 podá-la até obter uma **subárvore ótima**.
- Poderíamos decidir sobre a melhor através da validação cruzada. Todavia, pela quantidade de possíveis subárvores, isto é muito custoso;
- Utilizamos para isto o **Cost complexity pruning**.

Podando a árvore

- Se o modelo se adaptar muito aos dados (**overfit**), o processo anterior pode produzir boas predições no treino, e um mau desempenho no teste;
- Sendo assim, árvores menores (com menos regiões), embora apresente maior viés, conduz à menor variabilidade e melhor interpretação;
- O processo de **poda da árvore** é semelhante ao Lasso, i.e., continuamos a penalizar o modelo pela sua complexidade;
- Se antes buscávamos β's pequenos para evitar o *overfit*, agora penalizamos pelo número de regiões;
- A estratégia é a partir de uma grande árvore T_0 podá-la até obter uma **subárvore ótima**.
- Poderíamos decidir sobre a melhor através da validação cruzada. Todavia, pela quantidade de possíveis subárvores, isto é muito custoso;
- Utilizamos para isto o **Cost complexity pruning**.

Podando a árvore

- Se o modelo se adaptar muito aos dados (**overfit**), o processo anterior pode produzir boas predições no treino, e um mau desempenho no teste;
- Sendo assim, árvores menores (com menos regiões), embora apresente maior viés, conduz à menor variabilidade e melhor interpretação;
- O processo de **poda da árvore** é semelhante ao Lasso, i.e., continuamos a penalizar o modelo pela sua complexidade;
- Se antes buscávamos β's pequenos para evitar o *overfit*, agora penalizamos pelo número de regiões;
- A estratégia é a partir de uma grande árvore T_0 podá-la até obter uma **subárvore ótima**.
- Poderíamos decidir sobre a melhor através da validação cruzada. Todavia, pela quantidade de possíveis subárvores, isto é muito custoso;
- Utilizamos para isto o **Cost complexity pruning**.

Podando a árvore

- Se o modelo se adaptar muito aos dados (**overfit**), o processo anterior pode produzir boas predições no treino, e um mau desempenho no teste;
- Sendo assim, árvores menores (com menos regiões), embora apresente maior viés, conduz à menor variabilidade e melhor interpretação;
- O processo de **poda da árvore** é semelhante ao Lasso, i.e., continuamos a penalizar o modelo pela sua complexidade;
- Se antes buscávamos β's pequenos para evitar o *overfit*, agora penalizamos pelo número de regiões;
- A estratégia é a partir de uma grande árvore T_0 podá-la até obter uma **subárvore ótima**.
- Poderíamos decidir sobre a melhor através da validação cruzada. Todavia, pela quantidade de possíveis subárvores, isto é muito custoso;
- Utilizamos para isto o **Cost complexity pruning**.

Podando a árvore

- Se o modelo se adaptar muito aos dados (**overfit**), o processo anterior pode produzir boas predições no treino, e um mau desempenho no teste;
- Sendo assim, árvores menores (com menos regiões), embora apresente maior viés, conduz à menor variabilidade e melhor interpretação;
- O processo de **poda da árvore** é semelhante ao Lasso, i.e., continuamos a penalizar o modelo pela sua complexidade;
- Se antes buscávamos β's pequenos para evitar o *overfit*, agora penalizamos pelo número de regiões;
- A estratégia é a partir de uma grande árvore T_0 podá-la até obter uma **subárvore ótima**.
- Poderíamos decidir sobre a melhor através da validação cruzada. Todavia, pela quantidade de possíveis subárvores, isto é muito custoso;
- Utilizamos para isto o **Cost complexity pruning**.

Podando a árvore

- Se o modelo se adaptar muito aos dados (**overfit**), o processo anterior pode produzir boas predições no treino, e um mau desempenho no teste;

- Sendo assim, árvores menores (com menos regiões), embora apresente maior viés, conduz à menor variabilidade e melhor interpretação;

- O processo de **poda da árvore** é semelhante ao Lasso, i.e., continuamos a penalizar o modelo pela sua complexidade;

- Se antes buscávamos β's pequenos para evitar o *overfit*, agora penalizamos pelo número de regiões;

- A estratégia é a partir de uma grande árvore T_0 podá-la até obter uma **subárvore ótima**.

- Poderíamos decidir sobre a melhor através da validação cruzada. Todavia, pela quantidade de possíveis subárvores, isto é muito custoso;

- Utilizamos para isto o **Cost complexity pruning**.

Podando a árvore

- Se o modelo se adaptar muito aos dados (**overfit**), o processo anterior pode produzir boas predições no treino, e um mau desempenho no teste;
- Sendo assim, árvores menores (com menos regiões), embora apresente maior viés, conduz à menor variabilidade e melhor interpretação;
- O processo de **poda da árvore** é semelhante ao Lasso, i.e., continuamos a penalizar o modelo pela sua complexidade;
- Se antes buscávamos β's pequenos para evitar o *overfit*, agora penalizamos pelo número de regiões;
- A estratégia é a partir de uma grande árvore T_0 podá-la até obter uma **subárvore ótima**.
- Poderíamos decidir sobre a melhor através da validação cruzada. Todavia, pela quantidade de possíveis subárvores, isto é muito custoso;
- Utilizamos para isto o **Cost complexity pruning**.

Cost complexity pruning

- Considere uma sequência de árvores indexadas pelo parâmetro α;
- Para cada valor de α, temos uma subárvore $T \subset T_0$, tal que

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

seja o menor possível.

  - $|T|$ indica o número de **terminal nodes** da árvore T;
  - R_m é o retângulo correspondente ao m-ésimo *terminal node*;
  - $\hat{y}_{R_m}$ é a média das observações dos dados de treino em R_m.

- Selecionamos o valor ótimo $\hat{\alpha}$ através de validação. Em seguida, obtemos a subárvore utilizando $\hat{\alpha}$;
- Quando $\alpha = 0$, a subárvore T é simplemente T_0 (a equação mede somente o erro de treinamento).

Cost complexity pruning

- Considere uma sequência de árvores indexadas pelo parâmetro α;
- Para cada valor de α, temos uma subárvore $T \subset T_0$, tal que

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

  seja o menor possível.

  - $|T|$ indica o número de **terminal nodes** da árvore T;
  - R_m é o retângulo correspondente ao m-ésimo *terminal node*;
  - $\hat{y}_{R_m}$ é a média das observações dos dados de treino em R_m.

- Selecionamos o valor ótimo $\hat{\alpha}$ através de validação. Em seguida, obtemos a subárvore utilizando $\hat{\alpha}$;
- Quando $\alpha = 0$, a subárvore T é simplemente T_0 (a equação mede somente o erro de treinamento).

Cost complexity pruning

- Considere uma sequência de árvores indexadas pelo parâmetro α;
- Para cada valor de α, temos uma subárvore $T \subset T_0$, tal que

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

  seja o menor possível.

  - ⋆ $|T|$ indica o número de **terminal nodes** da árvore T;
  - ⋆ R_m é o retângulo correspondente ao m-ésimo *terminal node*;
  - ⋆ $\hat{y}_{R_m}$ é a média das observações dos dados de treino em R_m.

- Selecionamos o valor ótimo $\hat{\alpha}$ através de validação. Em seguida, obtemos a subárvore utilizando $\hat{\alpha}$;
- Quando $\alpha = 0$, a subárvore T é simplemente T_0 (a equação mede somente o erro de treinamento).

Cost complexity pruning

- Considere uma sequência de árvores indexadas pelo parâmetro α;
- Para cada valor de α, temos uma subárvore $T \subset T_0$, tal que

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

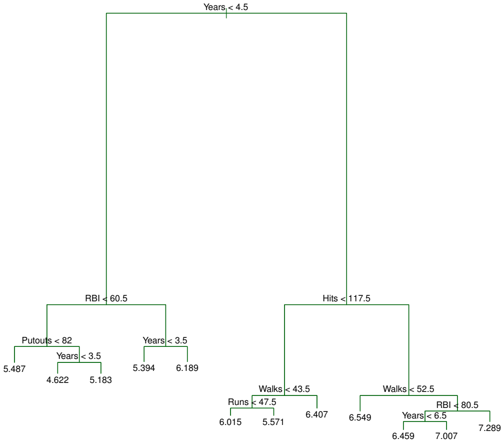
  seja o menor possível.

  - ⋆ $|T|$ indica o número de **terminal nodes** da árvore T;
  - ⋆ R_m é o retângulo correspondente ao m-ésimo *terminal node*;
  - ⋆ $\hat{y}_{R_m}$ é a média das observações dos dados de treino em R_m.

- Selecionamos o valor ótimo $\hat{\alpha}$ através de validação. Em seguida, obtemos a subárvore utilizando $\hat{\alpha}$;
- Quando $\alpha = 0$, a subárvore T é simplemente T_0 (a equação mede somente o erro de treinamento).

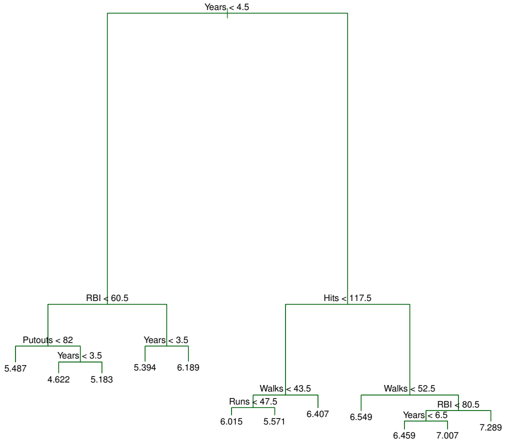
Continuação do exemplo do Baseball

- Dividimos as 263 observações em treinamento (132) e teste (131);
- Construímos uma grande árvore nos dados de treino;
- A figura ao lado refere-se à árvore sem poda;
- Variando α temos diferentes subárvores;
- E com a validação cruzada ($6-$ *fold*) encontramos $\hat{\alpha}$;
- Escolhemos $6-$ *fold* por ser múltiplo de 132;



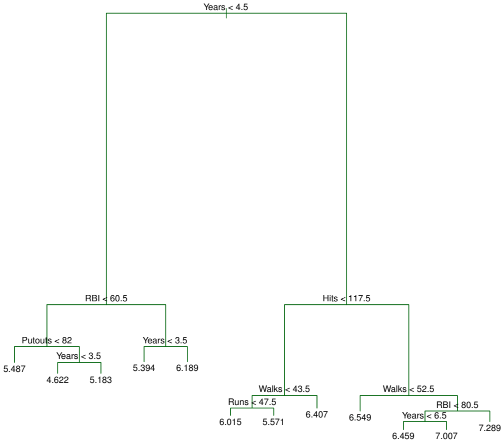
Continuação do exemplo do Baseball

- Dividimos as 263 observações em treinamento (132) e teste (131);
- Construímos uma grande árvore nos dados de treino;
- A figura ao lado refere-se à árvore sem poda;
- Variando α temos diferentes subárvores;
- E com a validação cruzada ($6 - \textit{fold}$) encontramos $\hat{\alpha}$;
- Escolhemos $6 - \textit{fold}$ por ser múltiplo de 132;



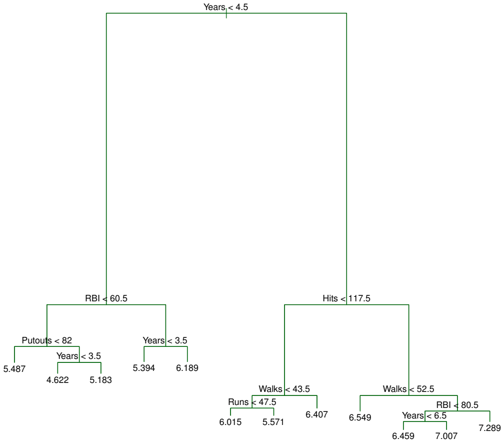
Continuação do exemplo do Baseball

- Dividimos as 263 observações em treinamento (132) e teste (131);
- Construímos uma grande árvore nos dados de treino;
- A figura ao lado refere-se à árvore sem poda;
- Variando α temos diferentes subárvores;
- E com a validação cruzada ($6 - \textit{fold}$) encontramos $\hat{\alpha}$;
- Escolhemos $6 - \textit{fold}$ por ser múltiplo de 132;



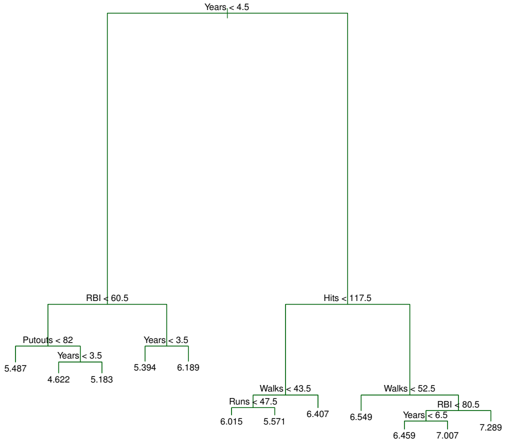
Continuação do exemplo do Baseball

- Dividimos as 263 observações em treinamento (132) e teste (131);
- Construímos uma grande árvore nos dados de treino;
- A figura ao lado refere-se à árvore sem poda;
- Variando α temos diferentes subárvores;
- E com a validação cruzada ($6-$*fold*) encontramos $\hat{\alpha}$;
- Escolhemos $6-$*fold* por ser múltiplo de 132;



Continuação do exemplo do Baseball

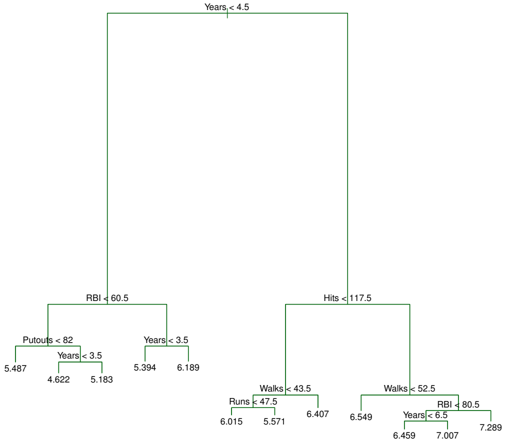
- Dividimos as 263 observações em treinamento (132) e teste (131);
- Construímos uma grande árvore nos dados de treino;
- A figura ao lado refere-se à árvore sem poda;
- Variando α temos diferentes subárvores;
- E com a validação cruzada $(6 - \textit{fold})$ encontramos $\hat{\alpha}$;
- Escolhemos $6 - \textit{fold}$ por ser múltiplo de 132;



Continuação do exemplo do Baseball

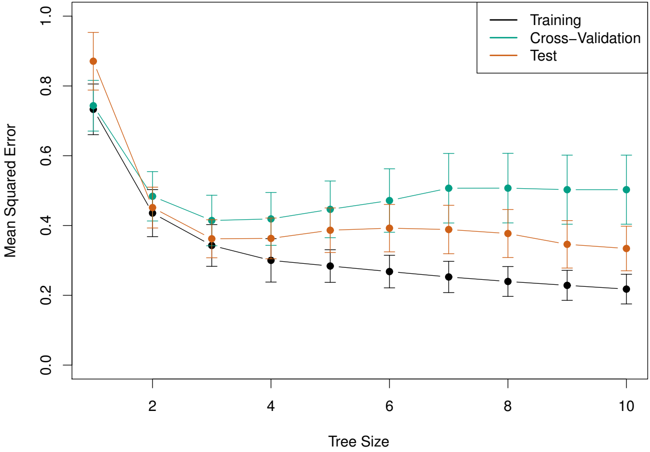
- Dividimos as 263 observações em treinamento (132) e teste (131);

- Construímos uma grande árvore nos dados de treino;
- A figura ao lado refere-se à árvore sem poda;
- Variando α temos diferentes subárvores;
- E com a validação cruzada ($6 - $*fold*) encontramos $\hat{\alpha}$;
- Escolhemos $6 - $*fold* por ser múltiplo de 132;



Continuação do exemplo do Baseball

- O erro mínimo na validação cruzada ocorre na árvore de tamanho 3.



Árvores de classificação

- O processo é similar à árvore de regressão, exceto pela resposta que é qualitativa;
- Agora, tentamos prever a pertinência das observações nas classes;
- Tal como árvore de regressão, utilizamos divisões binárias para crescer nossa árvore;
- Mas, o critério não é mais a soma de quadrados dos resíduos e sim o **Gini index** que mede a variância total entre as classes

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

- E o **cross-entropy**, dado por

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

- Buscamos valores baixos destas quantidades;

Árvores de classificação

- O processo é similar à árvore de regressão, exceto pela resposta que é qualitativa;
- Agora, tentamos prever a pertinência das observações nas classes;
- Tal como árvore de regressão, utilizamos divisões binárias para crescer nossa árvore;
- Mas, o critério não é mais a soma de quadrados dos resíduos e sim o **Gini index** que mede a variância total entre as classes

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

- E o **cross-entropy**, dado por

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

- Buscamos valores baixos destas quantidades;

Árvores de classificação

- O processo é similar à árvore de regressão, exceto pela resposta que é qualitativa;

- Agora, tentamos prever a pertinência das observações nas classes;

- Tal como árvore de regressão, utilizamos divisões binárias para crescer nossa árvore;

- Mas, o critério não é mais a soma de quadrados dos resíduos e sim o **Gini index** que mede a variância total entre as classes

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

- E o **cross-entropy**, dado por

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

- Buscamos valores baixos destas quantidades;

Árvores de classificação

- O processo é similar à árvore de regressão, exceto pela resposta que é qualitativa;
- Agora, tentamos prever a pertinência das observações nas classes;
- Tal como árvore de regressão, utilizamos divisões binárias para crescer nossa árvore;
- Mas, o critério não é mais a soma de quadrados dos resíduos e sim o **Gini index** que mede a variância total entre as classes

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

- E o **cross-entropy**, dado por

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

- Buscamos valores baixos destas quantidades;

Árvores de classificação

- O processo é similar à árvore de regressão, exceto pela resposta que é qualitativa;
- Agora, tentamos prever a pertinência das observações nas classes;
- Tal como árvore de regressão, utilizamos divisões binárias para crescer nossa árvore;
- Mas, o critério não é mais a soma de quadrados dos resíduos e sim o **Gini index** que mede a variância total entre as classes

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

- E o **cross-entropy**, dado por

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

- Buscamos valores baixos destas quantidades;

Árvores de classificação

- O processo é similar à árvore de regressão, exceto pela resposta que é qualitativa;
- Agora, tentamos prever a pertinência das observações nas classes;
- Tal como árvore de regressão, utilizamos divisões binárias para crescer nossa árvore;
- Mas, o critério não é mais a soma de quadrados dos resíduos e sim o **Gini index** que mede a variância total entre as classes

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

- E o **cross-entropy**, dado por

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

- Buscamos valores baixos destas quantidades;

Exemplo: heart disease - HD

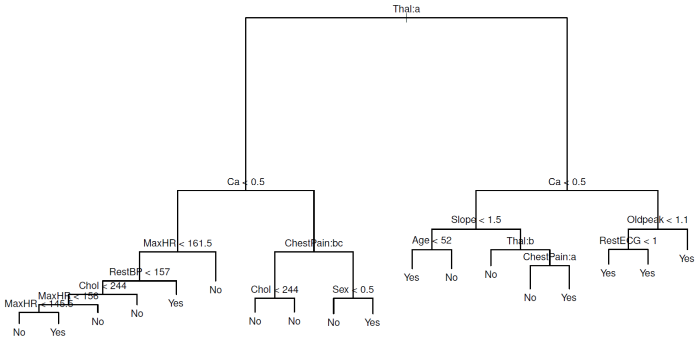
- Os dados contêm o diagnóstico de 303 pacientes com dores no peito:
  - Yes: indica a presença de doença cardíaca;
  - No: indica ausência de doença cardíaca;
- Os dados apresentam 13 preditores incluindo Age, Sex, Chol (medida de colesterol), e outras medidas de funções cardíacas e pulmonar;

Exemplo: heart disease - `HD`

- Os dados contêm o diagnóstico de 303 pacientes com dores no peito:
  - ⋆ `Yes`: indica a presença de doença cardíaca;
  - ⋆ `No`: indica ausência de doença cardíaca;
- Os dados apresentam 13 preditores incluindo `Age`, `Sex`, `Chol` (medida de colesterol), e outras medidas de funções cardíacas e pulmonar;

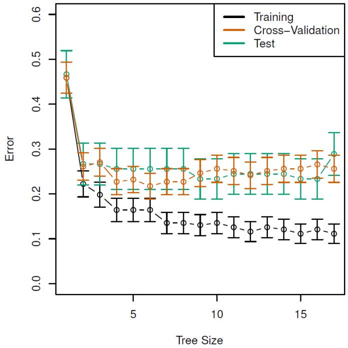
Exemplo: heart disease - HD

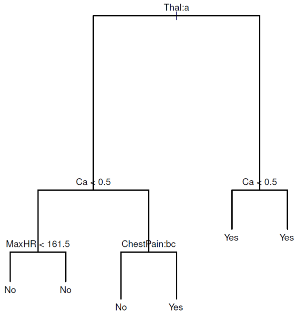
- Os dados contêm o diagnóstico de 303 pacientes com dores no peito:
  - ⋆ Yes: indica a presença de doença cardíaca;
  - ⋆ No: indica ausência de doença cardíaca;
- Os dados apresentam 13 preditores incluindo Age, Sex, Chol (medida de colesterol), e outras medidas de funções cardíacas e pulmonar;



Exemplo: heart disease - HD

- Após validação cruzada chegamos na árvore com seis *terminal nodes*;



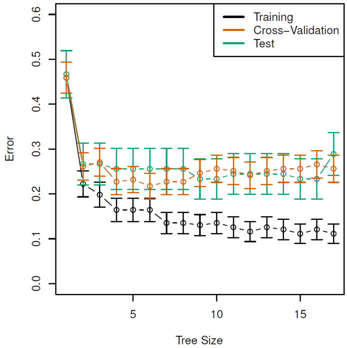


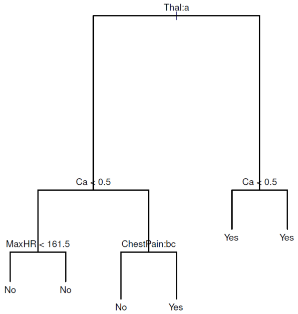
- Note que, em MaxHR temos duas respostas No. Isto se deve a um dos nós ser "puro" e o outro ser majoritariamente No.

- Por que "nó puro" é importante? Se tivermos uma observação no teste que pertença a este grupo, estaremos certo da resposta;

Exemplo: heart disease - HD

- Após validação cruzada chegamos na árvore com seis *terminal nodes*;



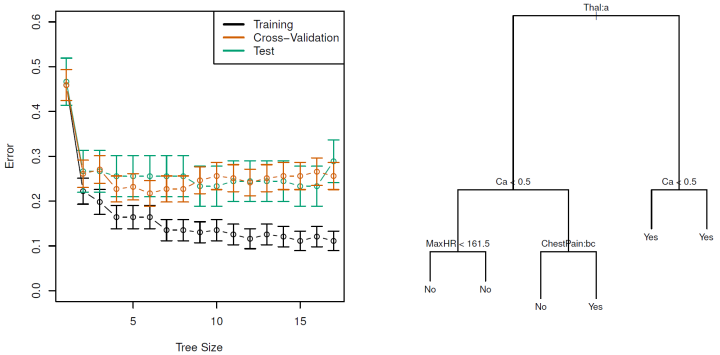


- Note que, em MaxHR temos duas respostas No. Isto se deve a um dos nós ser "puro" e o outro ser majoritariamente No.

- Por que "nó puro" é importante? Se tivermos uma observação no teste que pertença a este grupo, estaremos certo da resposta;

Exemplo: heart disease - HD

- Após validação cruzada chegamos na árvore com seis *terminal nodes*;



- Note que, em MaxHR temos duas respostas No. Isto se deve a um dos nós ser "puro" e o outro ser majoritariamente No.

- Por que "nó puro" é importante? Se tivermos uma observação no teste que pertença a este grupo, estaremos certo da resposta;

Árvores versus modelos lineares

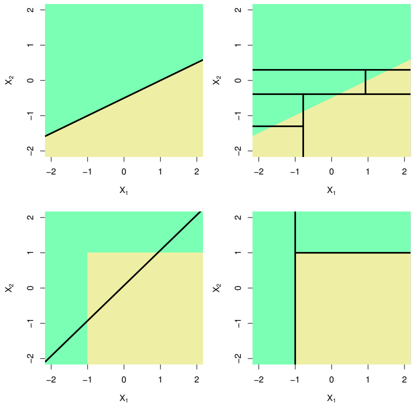
- Se a relação entre as características e resposta é bem aproximada por um modelo linear, $f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$ fará um bom trabalho;
- Se em vez disso, temos uma relação não linear e complexa métodos baseado em árvores se sairá melhor;
- No exemplo, as cores representam a verdadeira relação das respostas;

Árvores versus modelos lineares

- Se a relação entre as características e resposta é bem aproximada por um modelo linear, $f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$ fará um bom trabalho;
- Se em vez disso, temos uma relação não linear e complexa métodos baseado em árvores se sairá melhor;
- No exemplo, as cores representam a verdadeira relação das respostas;

Árvores versus modelos lineares

- Se a relação entre as características e resposta é bem aproximada por um modelo linear, $f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$ fará um bom trabalho;
- Se em vez disso, temos uma relação não linear e complexa métodos baseado em árvores se sairá melhor;
- No exemplo, as cores representam a verdadeira relação das respostas;



Ensembles

Bagging

- Como já vimos, o método de **bootstrap** é uma poderosa ideia para reamostragem;

- Ele é usado em muitas situações nas quais é difícil (ou mesmo impossível) calcular diretamente o desvio-padrão da quantidade de interesse;

- Agora, vamos estudar sua utilidade em um contexto completamente diferente: para aprimorar os métodos de árvores de decisão;

- Relembre que um conjunto de n observações independentes $Z_1, \ldots, Z_n$, cada com variância σ^2, a variância de $\bar{Z}$ é dada por σ^2/n;

- I.e., tomar a média das observações reduz a variância. Evidentemente, isto não é prático, pois não temos acesso a múltiplos dados de treino;

- Mas, podemos gerar repetidas amostras de treino utilizando o bootstrap!

Bagging

- Como já vimos, o método de **bootstrap** é uma poderosa ideia para reamostragem;

- Ele é usado em muitas situações nas quais é difícil (ou mesmo impossível) calcular diretamente o desvio-padrão da quantidade de interesse;

- Agora, vamos estudar sua utilidade em um contexto completamente diferente: para aprimorar os métodos de árvores de decisão;

- Relembre que um conjunto de n observações independentes $Z_1, \ldots, Z_n$, cada com variância σ^2, a variância de $\bar{Z}$ é dada por σ^2/n;

- I.e., tomar a média das observações reduz a variância. Evidentemente, isto não é prático, pois não temos acesso a múltiplos dados de treino;

- Mas, podemos gerar repetidas amostras de treino utilizando o bootstrap!

Bagging

- Como já vimos, o método de **bootstrap** é uma poderosa ideia para reamostragem;

- Ele é usado em muitas situações nas quais é difícil (ou mesmo impossível) calcular diretamente o desvio-padrão da quantidade de interesse;

- Agora, vamos estudar sua utilidade em um contexto completamente diferente: para aprimorar os métodos de árvores de decisão;

- Relembre que um conjunto de n observações independentes $Z_1, \ldots, Z_n$, cada com variância σ^2, a variância de $\bar{Z}$ é dada por σ^2/n;

- I.e., tomar a média das observações reduz a variância. Evidentemente, isto não é prático, pois não temos acesso a múltiplos dados de treino;

- Mas, podemos gerar repetidas amostras de treino utilizando o bootstrap!

Bagging

- Como já vimos, o método de **bootstrap** é uma poderosa ideia para reamostragem;

- Ele é usado em muitas situações nas quais é difícil (ou mesmo impossível) calcular diretamente o desvio-padrão da quantidade de interesse;

- Agora, vamos estudar sua utilidade em um contexto completamente diferente: para aprimorar os métodos de árvores de decisão;

- Relembre que um conjunto de n observações independentes $Z_1, \ldots, Z_n$, cada com variância σ^2, a variância de $\bar{Z}$ é dada por σ^2/n;

- I.e., tomar a média das observações reduz a variância. Evidentemente, isto não é prático, pois não temos acesso a múltiplos dados de treino;

- Mas, podemos gerar repetidas amostras de treino utilizando o bootstrap!

Bagging

- Como já vimos, o método de **bootstrap** é uma poderosa ideia para reamostragem;

- Ele é usado em muitas situações nas quais é difícil (ou mesmo impossível) calcular diretamente o desvio-padrão da quantidade de interesse;

- Agora, vamos estudar sua utilidade em um contexto completamente diferente: para aprimorar os métodos de árvores de decisão;

- Relembre que um conjunto de n observações independentes $Z_1, \ldots, Z_n$, cada com variância σ^2, a variância de $\bar{Z}$ é dada por σ^2/n;

- I.e., tomar a média das observações reduz a variância. Evidentemente, isto não é prático, pois não temos acesso a múltiplos dados de treino;

- Mas, podemos gerar repetidas amostras de treino utilizando o bootstrap!

Bagging

- Como já vimos, o método de **bootstrap** é uma poderosa ideia para reamostragem;

- Ele é usado em muitas situações nas quais é difícil (ou mesmo impossível) calcular diretamente o desvio-padrão da quantidade de interesse;

- Agora, vamos estudar sua utilidade em um contexto completamente diferente: para aprimorar os métodos de árvores de decisão;

- Relembre que um conjunto de n observações independentes $Z_1, \ldots, Z_n$, cada com variância σ^2, a variância de $\bar{Z}$ é dada por σ^2/n;

- I.e., tomar a média das observações reduz a variância. Evidentemente, isto não é prático, pois não temos acesso a múltiplos dados de treino;

- Mas, podemos gerar repetidas amostras de treino utilizando o bootstrap!

Bagging

- Nesta abordagem, geramos B diferentes dados (*bootstrapped*) de treino. Treinamos nosso modelo a fim de obter a predição no ponto x, $\hat{f}^{*b}(x)$;

- Em seguida, calculamos a média de todas as predições (chamamos de **bagging**):

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

- Esta abordagem se aplica à árvore de regressão;

- Para classificação, em cada dado de teste, registramos a classe predita pelas B árvores, e escolhemos pela maioria dos votos;

- Note que podar a árvore reduz a variância, **mas aumenta o viés**;

- Então, a ideia é não podá-la e aumentar o tamanho da amostra.

Bagging

- Nesta abordagem, geramos B diferentes dados (*bootstrapped*) de treino. Treinamos nosso modelo a fim de obter a predição no ponto x, $\hat{f}^{*b}(x)$;

- Em seguida, calculamos a média de todas as predições (chamamos de **bagging**):

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

- Esta abordagem se aplica à árvore de regressão;

- Para classificação, em cada dado de teste, registramos a classe predita pelas B árvores, e escolhemos pela maioria dos votos;

- Note que podar a árvore reduz a variância, **mas aumenta o viés**;

- Então, a ideia é não podá-la e aumentar o tamanho da amostra.

Bagging

- Nesta abordagem, geramos B diferentes dados (*bootstrapped*) de treino. Treinamos nosso modelo a fim de obter a predição no ponto x, $\hat{f}^{*b}(x)$;

- Em seguida, calculamos a média de todas as predições (chamamos de **bagging**):

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

- Esta abordagem se aplica à árvore de regressão;

- Para classificação, em cada dado de teste, registramos a classe predita pelas B árvores, e escolhemos pela maioria dos votos;

- Note que podar a árvore reduz a variância, **mas aumenta o viés**;

- Então, a ideia é não podá-la e aumentar o tamanho da amostra.

Bagging

- Nesta abordagem, geramos B diferentes dados (*bootstrapped*) de treino. Treinamos nosso modelo a fim de obter a predição no ponto x, $\hat{f}^{*b}(x)$;

- Em seguida, calculamos a média de todas as predições (chamamos de **bagging**):

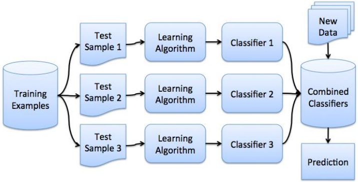
$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x).$$

- Esta abordagem se aplica à árvore de regressão;

- Para classificação, em cada dado de teste, registramos a classe predita pelas B árvores, e escolhemos pela maioria dos votos;

- Note que podar a árvore reduz a variância, **mas aumenta o viés**;

- Então, a ideia é não podá-la e aumentar o tamanho da amostra.



Bagging

- Nesta abordagem, geramos B diferentes dados (*bootstrapped*) de treino. Treinamos nosso modelo a fim de obter a predição no ponto x, $\hat{f}^{*b}(x)$;

- Em seguida, calculamos a média de todas as predições (chamamos de **bagging**):

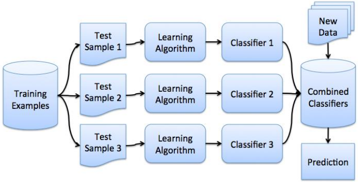
$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

- Esta abordagem se aplica à árvore de regressão;

- Para classificação, em cada dado de teste, registramos a classe predita pelas B árvores, e escolhemos pela maioria dos votos;

- Note que podar a árvore reduz a variância, **mas aumenta o viés**;

- Então, a ideia é não podá-la e aumentar o tamanho da amostra.



Bagging

- Nesta abordagem, geramos B diferentes dados (*bootstrapped*) de treino. Treinamos nosso modelo a fim de obter a predição no ponto x, $\hat{f}^{*b}(x)$;

- Em seguida, calculamos a média de todas as predições (chamamos de **bagging**):

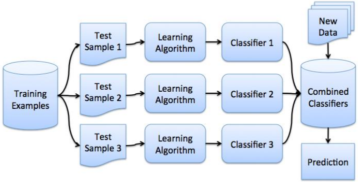
$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

- Esta abordagem se aplica à árvore de regressão;

- Para classificação, em cada dado de teste, registramos a classe predita pelas B árvores, e escolhemos pela maioria dos votos;

- Note que podar a árvore reduz a variância, **mas aumenta o viés**;

- Então, a ideia é não podá-la e aumentar o tamanho da amostra.



Out-of-Bag Error Estimation

- Utilizando o modelo *bagged*, podemos estimar o erro do teste de uma forma bastante simples;
- Sabemos que árvores são ajustadas para todos os subconjuntos *bootstrapped*. E cada uma utiliza cerca de 2/3 das observações;
- Sendo assim, temos 1/3 de observações (em média) que não participa do ajuste do b-ésimo modelo. Estas são as **observações out-of-bag (OOB)**;
- Podemos prever a i-ésima observação utilizando as árvores nas quais a observação é OOB. Isso resultará em torno de $B/3$ previsões;
- Esta abordagem é essencialmente a validação cruzada leave-one-out quando B é grande.

Out-of-Bag Error Estimation

- Utilizando o modelo *bagged*, podemos estimar o erro do teste de uma forma bastante simples;

- Sabemos que árvores são ajustadas para todos os subconjuntos *bootstrapped*. E cada uma utiliza cerca de $2/3$ das observações;

- Sendo assim, temos $1/3$ de observações (em média) que não participa do ajuste do b-ésimo modelo. Estas são as **observações out-of-bag (OOB)**;

- Podemos prever a i-ésima observação utilizando as árvores nas quais a observação é OOB. Isso resultará em torno de $B/3$ previsões;

- Esta abordagem é essencialmente a validação cruzada leave-one-out quando B é grande.

Out-of-Bag Error Estimation

- Utilizando o modelo *bagged*, podemos estimar o erro do teste de uma forma bastante simples;
- Sabemos que árvores são ajustadas para todos os subconjuntos *bootstrapped*. E cada uma utiliza cerca de $2/3$ das observações;
- Sendo assim, temos $1/3$ de observações (em média) que não participa do ajuste do b-ésimo modelo. Estas são as **observações out-of-bag (OOB)**;
- Podemos prever a i-ésima observação utilizando as árvores nas quais a observação é OOB. Isso resultará em torno de $B/3$ previsões;
- Esta abordagem é essencialmente a validação cruzada leave-one-out quando B é grande.

Out-of-Bag Error Estimation

- Utilizando o modelo *bagged*, podemos estimar o erro do teste de uma forma bastante simples;

- Sabemos que árvores são ajustadas para todos os subconjuntos *bootstrapped*. E cada uma utiliza cerca de 2/3 das observações;

- Sendo assim, temos 1/3 de observações (em média) que não participa do ajuste do b-ésimo modelo. Estas são as **observações out-of-bag (OOB)**;

- Podemos prever a i-ésima observação utilizando as árvores nas quais a observação é OOB. Isso resultará em torno de $B/3$ previsões;

- Esta abordagem é essencialmente a validação cruzada leave-one-out quando B é grande.

Out-of-Bag Error Estimation

- Utilizando o modelo *bagged*, podemos estimar o erro do teste de uma forma bastante simples;

- Sabemos que árvores são ajustadas para todos os subconjuntos *bootstrapped*. E cada uma utiliza cerca de 2/3 das observações;

- Sendo assim, temos 1/3 de observações (em média) que não participa do ajuste do b-ésimo modelo. Estas são as **observações out-of-bag (OOB)**;

- Podemos prever a i-ésima observação utilizando as árvores nas quais a observação é OOB. Isso resultará em torno de $B/3$ previsões;

- Esta abordagem é essencialmente a validação cruzada leave-one-out quando B é grande.

Random Forests

- Como já vimos, as amostras *bootstrap* são muito correlacionadas, e o *bagging* herda esta deficiência (árvores correlacionadas);
- **Random forests** fornece uma melhora em relação a este fato:
  - ⋆ Como em *Bagging*, construímos as regras de decisão baseadas nas amostras *bootstrapped*;
  - ⋆ Entretanto, para cada **partição** da árvore, uma **seleção aleatória de m preditores** é escolhido de um total de p;
  - ⋆ Na divisão é permitido utilizar somente um desse m preditores.
- Em outras palavras, em cada divisão da árvore, não é permitido para o algoritmo sequer considerar a maioria dos preditores;
- Tipicamente, utilizamos $m \approx \sqrt{p}$;

Random Forests

- Como já vimos, as amostras *bootstrap* são muito correlacionadas, e o *bagging* herda esta deficiência (árvores correlacionadas);

- **Random forests** fornece uma melhora em relação a este fato:

    - Como em *Bagging*, construímos as regras de decisão baseadas nas amostras *bootstrapped*;

    - Entretanto, para cada **partição** da árvore, uma **seleção aleatória de m preditores** é escolhido de um total de p;

    - Na divisão é permitido utilizar somente um desse m preditores.

- Em outras palavras, em cada divisão da árvore, não é permitido para o algoritmo sequer considerar a maioria dos preditores;

- Tipicamente, utilizamos $m \approx \sqrt{p}$;

Random Forests

- Como já vimos, as amostras *bootstrap* são muito correlacionadas, e o *bagging* herda esta deficiência (árvores correlacionadas);

- **Random forests** fornece uma melhora em relação a este fato:

  - Como em *Bagging*, construímos as regras de decisão baseadas nas amostras *bootstrapped*;

  - Entretanto, para cada **partição** da árvore, uma **seleção aleatória de** m **preditores** é escolhido de um total de p;

  - Na divisão é permitido utilizar somente um desse m preditores.

- Em outras palavras, em cada divisão da árvore, não é permitido para o algoritmo sequer considerar a maioria dos preditores;

- Tipicamente, utilizamos $m \approx \sqrt{p}$;

Random Forests

- Como já vimos, as amostras *bootstrap* são muito correlacionadas, e o *bagging* herda esta deficiência (árvores correlacionadas);

- **Random forests** fornece uma melhora em relação a este fato:

  - Como em *Bagging*, construímos as regras de decisão baseadas nas amostras *bootstrapped*;

  - Entretanto, para cada **partição** da árvore, uma **seleção aleatória de m preditores** é escolhido de um total de p;

  - Na divisão é permitido utilizar somente um desse m preditores.

- Em outras palavras, em cada divisão da árvore, não é permitido para o algoritmo sequer considerar a maioria dos preditores;

- Tipicamente, utilizamos $m \approx \sqrt{p}$;

Random Forests

- Como já vimos, as amostras *bootstrap* são muito correlacionadas, e o *bagging* herda esta deficiência (árvores correlacionadas);

- **Random forests** fornece uma melhora em relação a este fato:

  - ⋆ Como em *Bagging*, construímos as regras de decisão baseadas nas amostras *bootstrapped*;

  - ⋆ Entretanto, para cada **partição** da árvore, uma **seleção aleatória de m preditores** é escolhido de um total de p;

  - ⋆ Na divisão é permitido utilizar somente um desse m preditores.

- Em outras palavras, em cada divisão da árvore, não é permitido para o algoritmo sequer considerar a maioria dos preditores;

- Tipicamente, utilizamos $m \approx \sqrt{p}$;

Random Forests

- Como já vimos, as amostras *bootstrap* são muito correlacionadas, e o *bagging* herda esta deficiência (árvores correlacionadas);

- **Random forests** fornece uma melhora em relação a este fato:

  - ⋆ Como em *Bagging*, construímos as regras de decisão baseadas nas amostras *bootstrapped*;

  - ⋆ Entretanto, para cada **partição** da árvore, uma **seleção aleatória de m preditores** é escolhido de um total de p;

  - ⋆ Na divisão é permitido utilizar somente um desse m preditores.

- Em outras palavras, em cada divisão da árvore, não é permitido para o algoritmo sequer considerar a maioria dos preditores;

- Tipicamente, utilizamos $m \approx \sqrt{p}$;

Random Forests

- Como já vimos, as amostras *bootstrap* são muito correlacionadas, e o *bagging* herda esta deficiência (árvores correlacionadas);

- **Random forests** fornece uma melhora em relação a este fato:

  - Como em *Bagging*, construímos as regras de decisão baseadas nas amostras *bootstrapped*;

  - Entretanto, para cada **partição** da árvore, uma **seleção aleatória de m preditores** é escolhido de um total de p;

  - Na divisão é permitido utilizar somente um desse m preditores.

- Em outras palavras, em cada divisão da árvore, não é permitido para o algoritmo sequer considerar a maioria dos preditores;

- Tipicamente, utilizamos $m \approx \sqrt{p}$;

Random Forests

- Suponha que existe um preditor (ou um conjunto) muito forte nos dados de treinamento, juntamente com outros moderadamente fortes;

- Assim, na coleção de *bagged trees*, a maioria (ou todas) as árvores utilizarão os preditores fortes;

- Consequentemente, todas serão muito semelhantes entre si;

- *Random forests* força com que diferentes preditores sejam escolhidos (*decorrelating the trees*). Se $m = p$, estaremos no método *Bagging*;

Random Forests

- Suponha que exista um preditor (ou um conjunto) muito forte nos dados de treinamento, juntamente com outros moderadamente fortes;

- Assim, na coleção de *bagged trees*, a maioria (ou todas) as árvores utilizarão os preditores fortes;

- Consequentemente, todas serão muito semelhantes entre si;

- *Random forests* força com que diferentes preditores sejam escolhidos (*decorrelating the trees*). Se $m = p$, estaremos no método *Bagging*;

Random Forests

- Suponha que exista um preditor (ou um conjunto) muito forte nos dados de treinamento, juntamente com outros moderadamente fortes;

- Assim, na coleção de *bagged trees*, a maioria (ou todas) as árvores utilizarão os preditores fortes;

- Consequentemente, todas serão muito semelhantes entre si;

- *Random forests* força com que diferentes preditores sejam escolhidos (*decorrelating the trees*). Se $m = p$, estaremos no método *Bagging*;

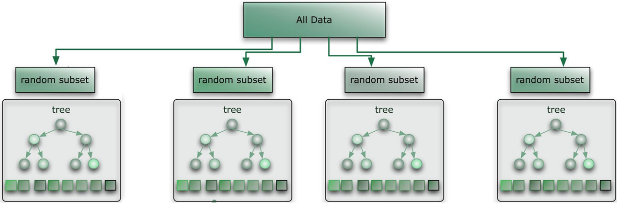
Random Forests

- Suponha que exista um preditor (ou um conjunto) muito forte nos dados de treinamento, juntamente com outros moderadamente fortes;

- Assim, na coleção de *bagged trees*, a maioria (ou todas) as árvores utilizarão os preditores fortes;

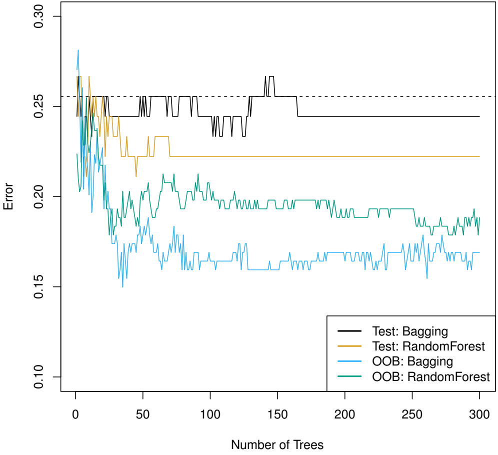
- Consequentemente, todas serão muito semelhantes entre si;

- *Random forests* força com que diferentes preditores sejam escolhidos (*decorrelating the trees*). Se $m = p$, estaremos no método *Bagging*;



Exemplo 1: Heart data set

- Abaixo, o erro do teste como função de B. A linha tracejada representa o erro utilizando uma árvore somente;



Exemplo 2: Gene expression data

- Os dados consistem na medida de expressão de 4.718 genes;
- Foram amostrados em tecidos de 349 pacientes;
- Cada paciente possui um marcador qualitativo (de 15 níveis):
  - Normal;
  - Ou 14 tipos de câncer;
- Utilizamos *Random forests* para prever o tipo de câncer baseado nos 500 genes de maior variabilidade nos dados de treino, variando m;
- A taxa de erro considerando única árvore foi de 45,7%;

Exemplo 2: Gene expression data

- Os dados consistem na medida de expressão de 4.718 genes;

- Foram amostrados em tecidos de 349 pacientes;

- Cada paciente possui um marcador qualitativo (de 15 níveis):

  - Normal;

  - Ou 14 tipos de câncer;

- Utilizamos *Random forests* para prever o tipo de câncer baseado nos 500 genes de maior variabilidade nos dados de treino, variando *m*;

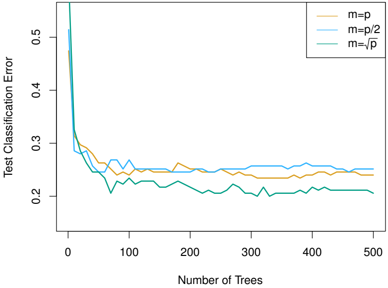
- A taxa de erro considerando única árvore foi de 45,7%;

Exemplo 2: Gene expression data

- Os dados consistem na medida de expressão de 4.718 genes;
- Foram amostrados em tecidos de 349 pacientes;
- Cada paciente possui um marcador qualitativo (de 15 níveis):
  - Normal;
  - Ou 14 tipos de câncer;
- Utilizamos *Random forests* para prever o tipo de câncer baseado nos 500 genes de maior variabilidade nos dados de treino, variando m;
- A taxa de erro considerando única árvore foi de 45,7%;

Exemplo 2: Gene expression data

- Os dados consistem na medida de expressão de 4.718 genes;
- Foram amostrados em tecidos de 349 pacientes;
- Cada paciente possui um marcador qualitativo (de 15 níveis):
  - ⋆ Normal;
  - ⋆ Ou 14 tipos de câncer;



- Utilizamos *Random forests* para prever o tipo de câncer baseado nos 500 genes de maior variabilidade nos dados de treino, variando m;
- A taxa de erro considerando única árvore foi de 45,7%;

Boosting

- **Boosting** é uma abordagem bastante geral, e pode ser aplicada em vários métodos de aprendizagem estatística para regressão e classificação.;

- Vimos em aulas anteriores o *gradient boosting*, nesta seção vamos aplicar o método em árvores de decisão;

- Relembre que em *bagging*:

    - Criamos múltiplas cópias dos dados de treino originais (utilizando *bootstrap*); Ajustamos diferentes árvores de decisão;

    - Combinando todas chegamos em um modelo preditivo.

- I.e., cada árvore é construída independente das outras. *Boosting* funciona de modo similar, exceto pelo fato delas crescerem **sequencialmente**;

- A árvore seguinte se baseará nos erros da árvore anterior.

Boosting

- **Boosting** é uma abordagem bastante geral, e pode ser aplicada em vários métodos de aprendizagem estatística para regressão e classificação.;
- Vimos em aulas anteriores o *gradient boosting*, nesta seção vamos aplicar o método em árvores de decisão;
- Relembre que em *bagging*:
  - ⋆ Criamos múltiplas cópias dos dados de treino originais (utilizando *bootstrap*); Ajustamos diferentes árvores de decisão;
  - ⋆ Combinando todas chegamos em um modelo preditivo.
- I.e., cada árvore é construída independente das outras. *Boosting* funciona de modo similar, exceto pelo fato delas crescerem **sequencialmente**;
- A árvore seguinte se baseará nos erros da árvore anterior.

Boosting

- **Boosting** é uma abordagem bastante geral, e pode ser aplicada em vários métodos de aprendizagem estatística para regressão e classificação.;
- Vimos em aulas anteriores o *gradient boosting*, nesta seção vamos aplicar o método em árvores de decisão;
- Relembre que em *bagging*:
  - ⋆ Criamos múltiplas cópias dos dados de treino originais (utilizando *bootstrap*); Ajustamos diferentes árvores de decisão;
  - ⋆ Combinando todas chegamos em um modelo preditivo.
- I.e., cada árvore é construída independente das outras. *Boosting* funciona de modo similar, exceto pelo fato delas crescerem **sequencialmente**;
- A árvore seguinte se baseará nos erros da árvore anterior.

Boosting

- **Boosting** é uma abordagem bastante geral, e pode ser aplicada em vários métodos de aprendizagem estatística para regressão e classificação.;
- Vimos em aulas anteriores o *gradient boosting*, nesta seção vamos aplicar o método em árvores de decisão;
- Relembre que em *bagging*:
  - ⋆ Criamos múltiplas cópias dos dados de treino originais (utilizando *bootstrap*); Ajustamos diferentes árvores de decisão;
  - ⋆ Combinando todas chegamos em um modelo preditivo.
- I.e., cada árvore é construída independente das outras. *Boosting* funciona de modo similar, exceto pelo fato delas crescerem **sequencialmente**;
- A árvore seguinte se baseará nos erros da árvore anterior.

Boosting

- **Boosting** é uma abordagem bastante geral, e pode ser aplicada em vários métodos de aprendizagem estatística para regressão e classificação.;
- Vimos em aulas anteriores o *gradient boosting*, nesta seção vamos aplicar o método em árvores de decisão;
- Relembre que em *bagging*:
  - ⋆ Criamos múltiplas cópias dos dados de treino originais (utilizando *bootstrap*); Ajustamos diferentes árvores de decisão;
  - ⋆ Combinando todas chegamos em um modelo preditivo.
- I.e., cada árvore é construída independente das outras. *Boosting* funciona de modo similar, exceto pelo fato delas crescerem **sequencialmente**;
- A árvore seguinte se baseará nos erros da árvore anterior.

Boosting para árvore de regressão

- Inicie com $\hat{f}(x) = 0$ e $r_i = y_i$, para todo i dos dados de treino;
- Para $b = 1, 2, \ldots, B$, repita:
  - ⋆ Ajuste a árvore $\hat{f}^b$ com d divisões ($d + 1$ *terminal nodes*) para os dados de treino (X, r);
  - ⋆ Atualize $\hat{f}$ adicionando uma versão da nova árvore:

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

  - ⋆ Atualize os resíduos,

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

- O modelo de saída fica então,

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

- O pacote `gbm` (*gradient boosted models*) lida com uma variedade de problemas de regressão e classificação.

Boosting para árvore de regressão

- Inicie com $\hat{f}(x) = 0$ e $r_i = y_i$, para todo i dos dados de treino;
- Para $b = 1, 2, \ldots, B$, repita:
  - ⋆ Ajuste a árvore $\hat{f}^b$ com d divisões ($d + 1$ *terminal nodes*) para os dados de treino (X, r);
  - ⋆ Atualize $\hat{f}$ adicionando uma versão da nova árvore:
$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
  - ⋆ Atualize os resíduos,
$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$
- O modelo de saída fica então,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x);$$
- O pacote `gbm` (*gradient boosted models*) lida com uma variedade de problemas de regressão e classificação.

Boosting para árvore de regressão

- Inicie com $\hat{f}(x) = 0$ e $r_i = y_i$, para todo i dos dados de treino;
- Para $b = 1, 2, \ldots, B$, repita:
  - ⋆ Ajuste a árvore $\hat{f}^b$ com d divisões ($d + 1$ *terminal nodes*) para os dados de treino (X, r);
  - ⋆ Atualize $\hat{f}$ adicionando uma versão da nova árvore:
  $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
  - ⋆ Atualize os resíduos,
  $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$
- O modelo de saída fica então,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x);$$
- O pacote `gbm` (*gradient boosted models*) lida com uma variedade de problemas de regressão e classificação.

Boosting para árvore de regressão

- Inicie com $\hat{f}(x) = 0$ e $r_i = y_i$, para todo i dos dados de treino;
- Para $b = 1, 2, \ldots, B$, repita:
  - ⋆ Ajuste a árvore $\hat{f}^b$ com d divisões ($d + 1$ *terminal nodes*) para os dados de treino (X, r);
  - ⋆ Atualize $\hat{f}$ adicionando uma versão da nova árvore:
  $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
  - ⋆ Atualize os resíduos,
  $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$
- O modelo de saída fica então,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x);$$
- O pacote `gbm` (*gradient boosted models*) lida com uma variedade de problemas de regressão e classificação.

Boosting para árvore de regressão

- Inicie com $\hat{f}(x) = 0$ e $r_i = y_i$, para todo i dos dados de treino;
- Para $b = 1, 2, \ldots, B$, repita:
  - ⋆ Ajuste a árvore $\hat{f}^b$ com d divisões ($d + 1$ *terminal nodes*) para os dados de treino (X, r);
  - ⋆ Atualize $\hat{f}$ adicionando uma versão da nova árvore:
  $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
  - ⋆ Atualize os resíduos,
  $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$
- O modelo de saída fica então,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x);$$
- O pacote `gbm` (*gradient boosted models*) lida com uma variedade de problemas de regressão e classificação.

Boosting para árvore de regressão

- Inicie com $\hat{f}(x) = 0$ e $r_i = y_i$, para todo i dos dados de treino;
- Para $b = 1, 2, \ldots, B$, repita:
  - ⋆ Ajuste a árvore $\hat{f}^b$ com d divisões ($d + 1$ *terminal nodes*) para os dados de treino (X, r);
  - ⋆ Atualize $\hat{f}$ adicionando uma versão da nova árvore:

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

  - ⋆ Atualize os resíduos,

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

- O modelo de saída fica então,

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

- O pacote `gbm` (*gradient boosted models*) lida com uma variedade de problemas de regressão e classificação.

Boosting para árvore de regressão

- Inicie com $\hat{f}(x) = 0$ e $r_i = y_i$, para todo i dos dados de treino;
- Para $b = 1, 2, \ldots, B$, repita:
  - ⋆ Ajuste a árvore $\hat{f}^b$ com d divisões ($d + 1$ *terminal nodes*) para os dados de treino (X, r);
  - ⋆ Atualize $\hat{f}$ adicionando uma versão da nova árvore:
  $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
  - ⋆ Atualize os resíduos,
  $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$
- O modelo de saída fica então,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x);$$
- O pacote `gbm` (*gradient boosted models*) lida com uma variedade de problemas de regressão e classificação.

Boosting para árvore de regressão

- O parâmetro λ controla a taxa com que o algoritmo aprende (tipicamente são 0,01 ou 0,001).
- Quando muito pequenos exigem que o número de árvores, B, seja grande;
- Diferentemente de *bagging* e *random forests*, *boosting* pode *overfit* se B é muito grande. Utilizamos CV para selecionar esta quantidade;
- Muito embora *random forests* e *boosting* apresentem excelentes predições, seus resultados podem ser difíceis de se interpretar;
- A seguir, voltaremos ao *Gene expression data*, a fim de prever pacientes com **câncer** versus **normal**;
- Para os dois modelos *boosted* utilizamos $\lambda = 0,01$. E a taxa de erro para única árvore é de 24%;

Boosting para árvore de regressão

- O parâmetro λ controla a taxa com que o algoritmo aprende (tipicamente são 0,01 ou 0,001).

- Quando muito pequenos exigem que o número de árvores, B, seja grande;

- Diferentemente de *bagging* e *random forests*, *boosting* pode *overfit* se B é muito grande. Utilizamos CV para selecionar esta quantidade;

- Muito embora *random forests* e *boosting* apresentem excelentes predições, seus resultados podem ser difíceis de se interpretar;

- A seguir, voltaremos ao *Gene expression data*, a fim de prever pacientes com **câncer** versus **normal**;

- Para os dois modelos *boosted* utilizamos $\lambda = 0,01$. E a taxa de erro para única árvore é de 24%;

Boosting para árvore de regressão

- O parâmetro λ controla a taxa com que o algoritmo aprende (tipicamente são 0,01 ou 0,001).

- Quando muito pequenos exigem que o número de árvores, B, seja grande;

- Diferentemente de *bagging* e *random forests*, *boosting* pode *overfit* se B é muito grande. Utilizamos CV para selecionar esta quantidade;

- Muito embora *random forests* e *boosting* apresentem excelentes predições, seus resultados podem ser difíceis de se interpretar;

- A seguir, voltaremos ao *Gene expression data*, a fim de prever pacientes com **câncer** versus **normal**;

- Para os dois modelos *boosted* utilizamos $\lambda = 0,01$. E a taxa de erro para única árvore é de 24%;

Boosting para árvore de regressão

- O parâmetro λ controla a taxa com que o algoritmo aprende (tipicamente são 0,01 ou 0,001).
- Quando muito pequenos exigem que o número de árvores, B, seja grande;
- Diferentemente de *bagging* e *random forests*, *boosting* pode *overfit* se B é muito grande. Utilizamos CV para selecionar esta quantidade;
- Muito embora *random forests* e *boosting* apresentem excelentes predições, seus resultados podem ser difíceis de se interpretar;
- A seguir, voltaremos ao *Gene expression data*, a fim de prever pacientes com **câncer** versus **normal**;
- Para os dois modelos *boosted* utilizamos $\lambda = 0,01$. E a taxa de erro para única árvore é de 24%;

Boosting para árvore de regressão

- O parâmetro λ controla a taxa com que o algoritmo aprende (tipicamente são 0,01 ou 0,001).
- Quando muito pequenos exigem que o número de árvores, B, seja grande;
- Diferentemente de *bagging* e *random forests*, *boosting* pode *overfit* se B é muito grande. Utilizamos CV para selecionar esta quantidade;
- Muito embora *random forests* e *boosting* apresentem excelentes predições, seus resultados podem ser difíceis de se interpretar;
- A seguir, voltaremos ao *Gene expression data*, a fim de prever pacientes com **câncer** versus **normal**;
- Para os dois modelos *boosted* utilizamos $\lambda = 0,01$. E a taxa de erro para única árvore é de 24%;

Boosting para árvore de regressão

- O parâmetro λ controla a taxa com que o algoritmo aprende (tipicamente são 0,01 ou 0,001).

- Quando muito pequenos exigem que o número de árvores, B, seja grande;

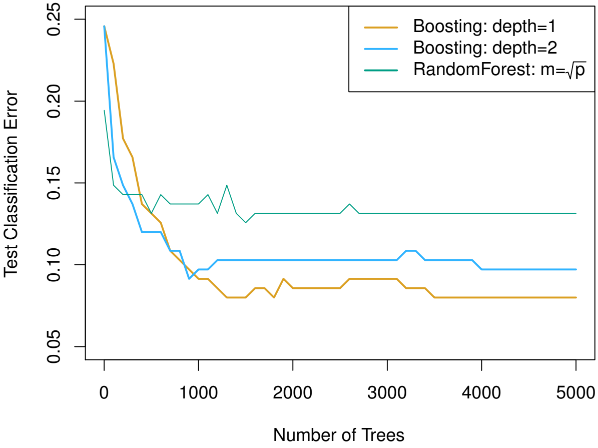
- Diferentemente de *bagging* e *random forests*, *boosting* pode *overfit* se B é muito grande. Utilizamos CV para selecionar esta quantidade;

- Muito embora *random forests* e *boosting* apresentem excelentes predições, seus resultados podem ser difíceis de se interpretar;

- A seguir, voltaremos ao *Gene expression data*, a fim de prever pacientes com **câncer** versus **normal**;

- Para os dois modelos *boosted* utilizamos $\lambda = 0,01$. E a taxa de erro para única árvore é de 24%;

Exemplo: Gene expression data



Boosting para árvore de classificação

- Não vamos entrar em detalhes sobre esta abordagem. O aluno interessado pode encontrar em **Elements of Statistical Learning, capítulo 10**.

- A ideia é ponderar os erros para que nas próximas árvores eles tenham mais importância. Em seguida, combinar os classificadores.

Boosting para árvore de classificação

- Não vamos entrar em detalhes sobre esta abordagem. O aluno interessado pode encontrar em **Elements of Statistical Learning, capítulo 10**.

- A ideia é ponderar os erros para que nas próximas árvores eles tenham mais importância. Em seguida, combinar os classificadores.