Introdução ao Machine Learning com R

Modelos de Árvores

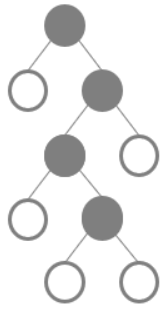
May de 2022

Conteúdo

curso-r

- Árvores de decisão
- Relação Viés-Variância
- Random Forest
- Gradient Boost
- XGboost

Árvore de Decisão
(Decision Trees)



curso-r

Floresta Aleatória
(Random Forest)

Gradient Boosting

No R

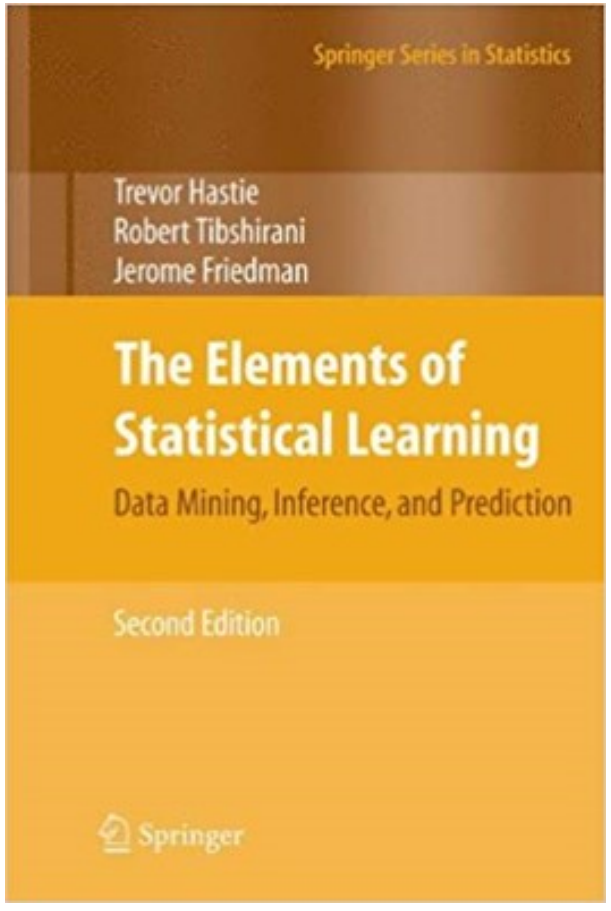
curso-r

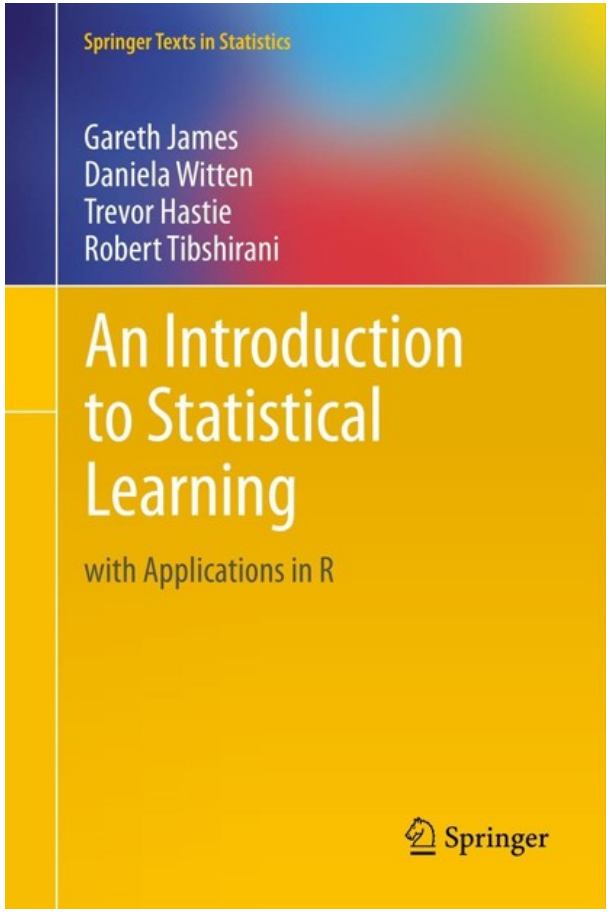
```
# árvore de decisão
modelo_tree <- decision_tree(
  min_n = tune(),
  tree_depth = tune(),
  cost_complexity = tune()
)
```

```
# Random Forest
modelo_rf <- rand_forest(
  min_n = tune(),
  mtry = tune(),
  trees = tune()
)
```

```
# XGBoost
modelo_xgb <- boost_tree(
  min_n = tune(),
  mtry = tune(),
  trees = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
  sample_size = tune()
)
```

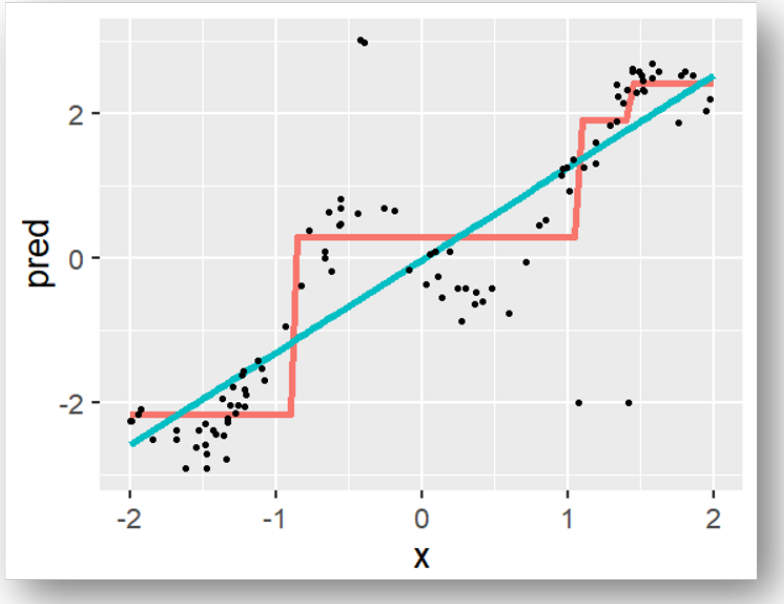
Referências



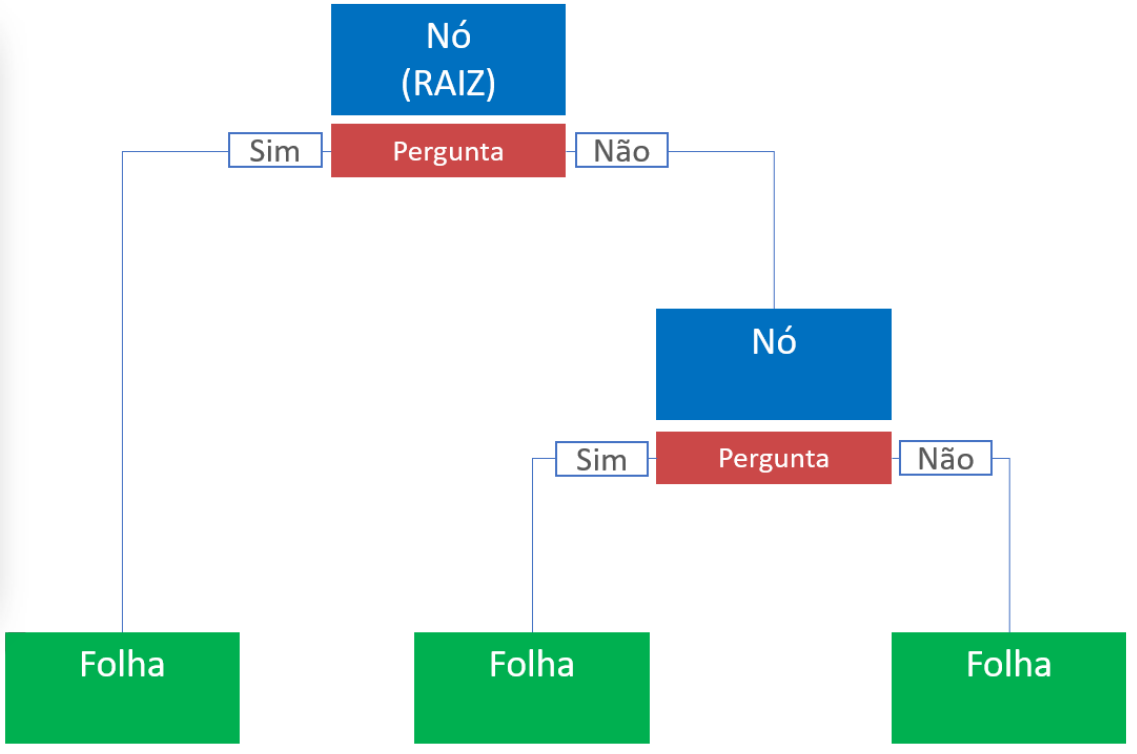


Árvore de Decisão

Árvore de Decisão



$$Y \approx f(X)$$

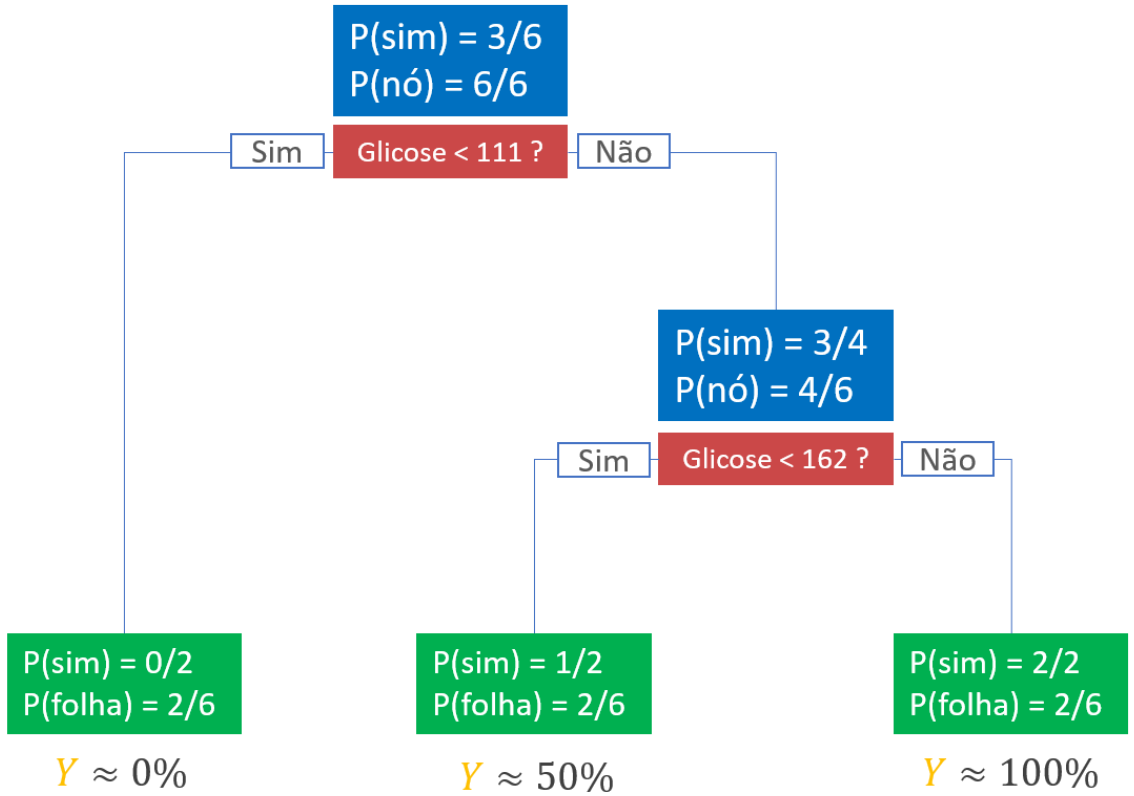


Árvore de Decisão

curso-r

Paciente	Pressão	Glicose	Diabetes
Alfredo	hipertensao	92	nao
Beatriz	normal	130	sim
Carla	normal	130	nao
Daniela	normal	55	nao
Ernesto	hipertensao	220	sim
Flavia	normal	195	sim

$$Y \approx f(X)$$

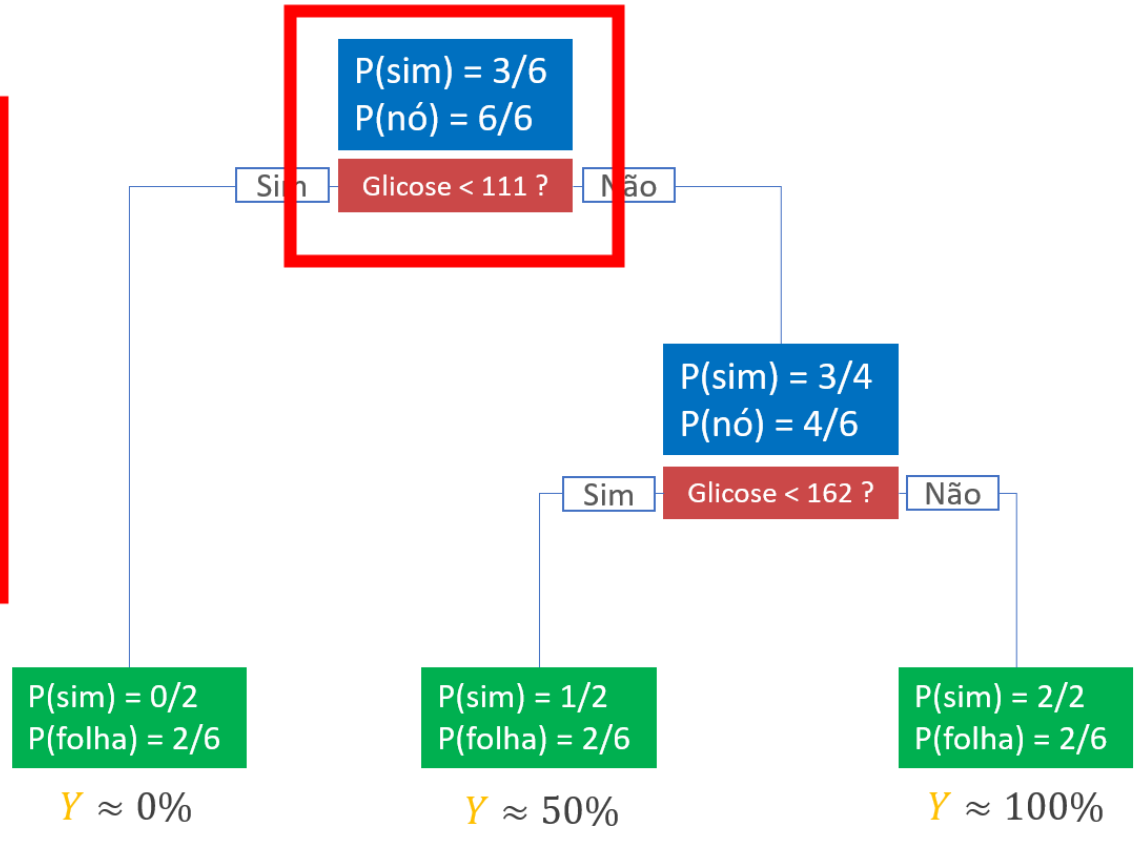


Árvore de Decisão

curso-r

Paciente	Pressão	Glicose	Diabetes
Alfredo	hipertensao	92	nao
Beatriz	normal	130	sim
Carla	normal	130	nao
Daniela	normal	55	nao
Ernesto	hipertensao	220	sim
Flavia	normal	195	sim

$$Y \approx f(X)$$

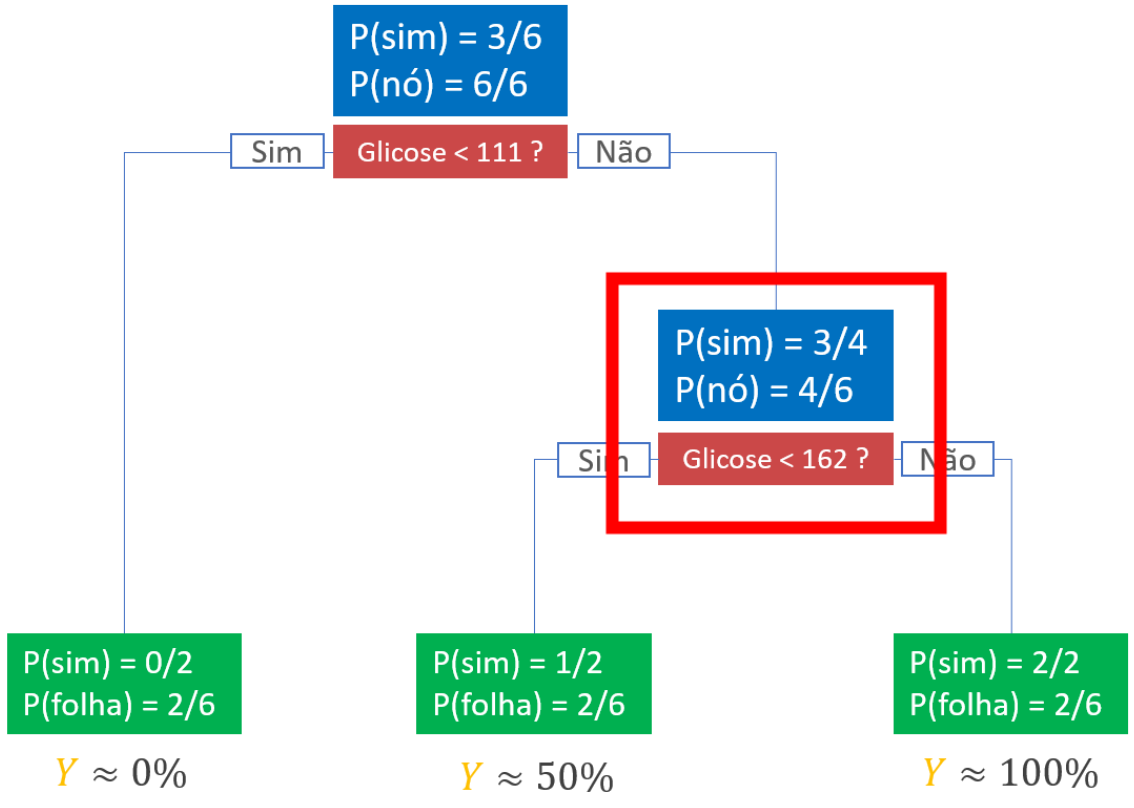


Árvore de Decisão

curso-r

Paciente	Pressão	Glicose	Diabetes
Alfredo	hipertensao	92	nao
Beatriz	normal	130	sim
Carla	normal	130	nao
Daniela	normal	55	nao
Ernesto	hipertensao	220	sim
Flavia	normal	195	sim

$$Y \approx f(X)$$

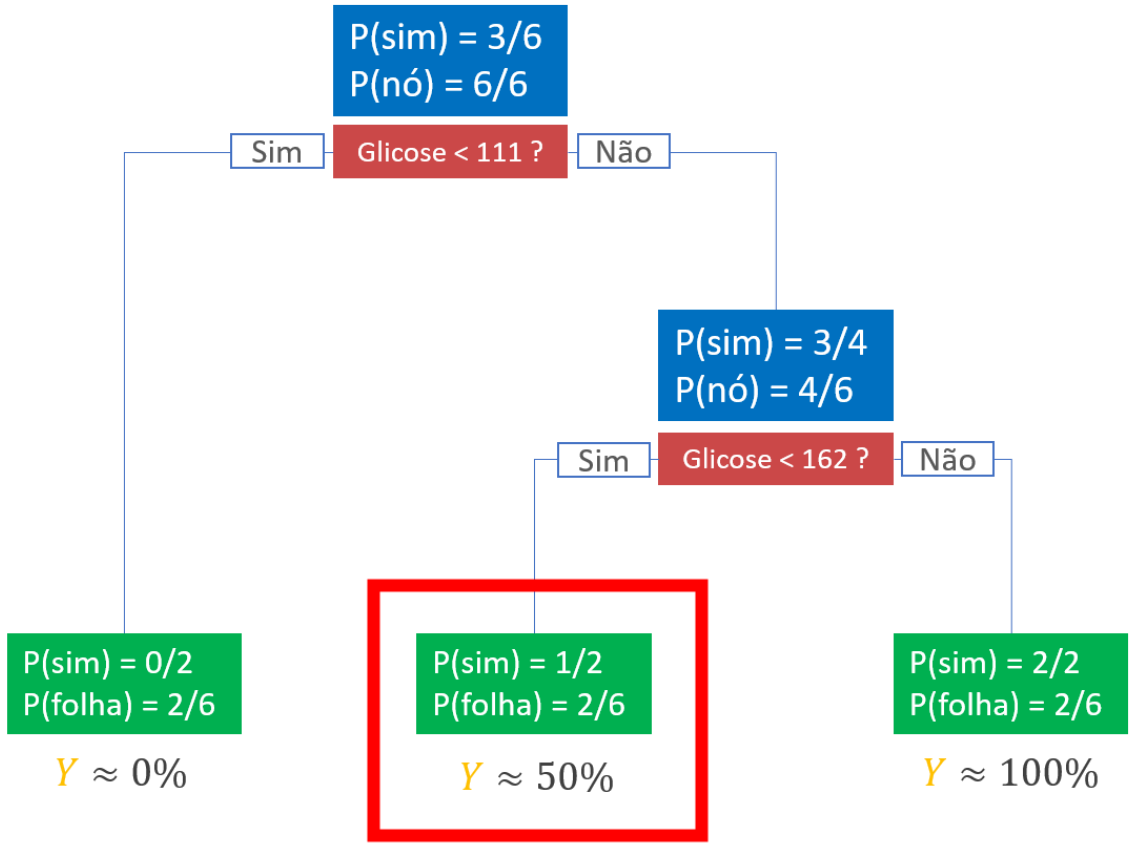


Árvore de Decisão

curso-r

Paciente	Pressão	Glicose	Diabetes
Alfredo	hipertensao	92	nao
Beatriz	normal	130	sim
Carla	normal	130	nao
Daniela	normal	55	nao
Ernesto	hipertensao	220	sim
Flavia	normal	195	sim

$$Y \approx f(X)$$

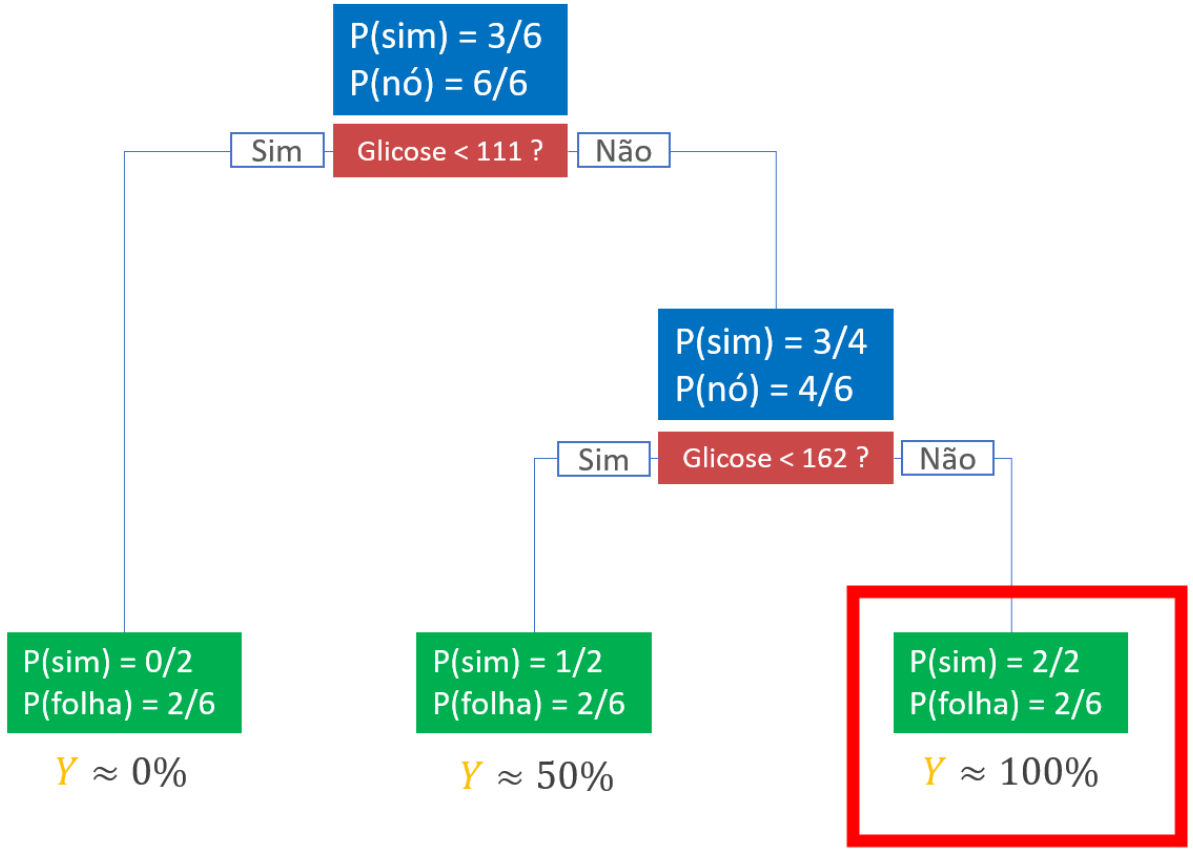


Árvore de Decisão

curso-r

Paciente	Pressão	Glicose	Diabetes
Alfredo	hipertensao	92	nao
Beatriz	normal	130	sim
Carla	normal	130	nao
Daniela	normal	55	nao
Ernesto	hipertensao	220	sim
Flavia	normal	195	sim

$$Y \approx f(X)$$

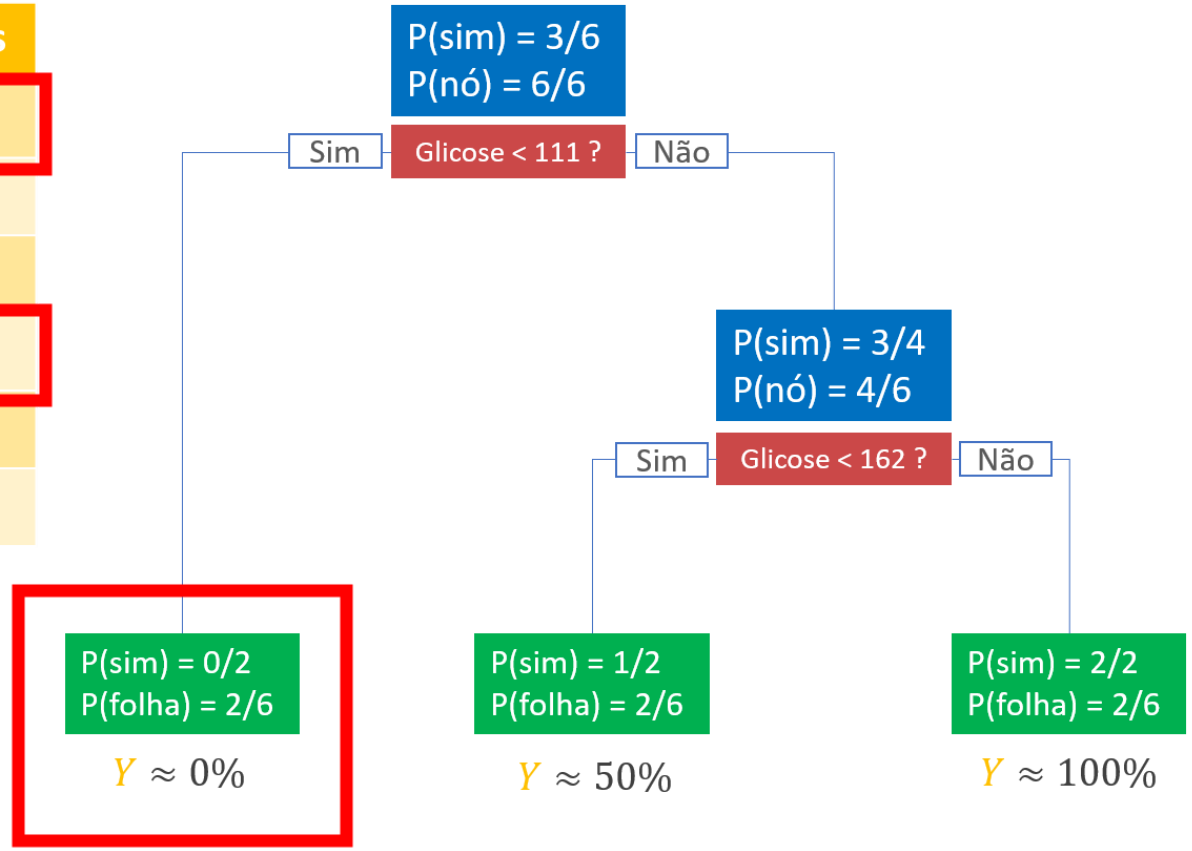


Árvore de Decisão

curso-r

Paciente	Pressão	Glicose	Diabetes
Alfredo	hipertensao	92	nao
Beatriz	normal	130	sim
Carla	normal	130	nao
Daniela	normal	55	nao
Ernesto	hipertensao	220	sim
Flavia	normal	195	sim

$$Y \approx f(X)$$

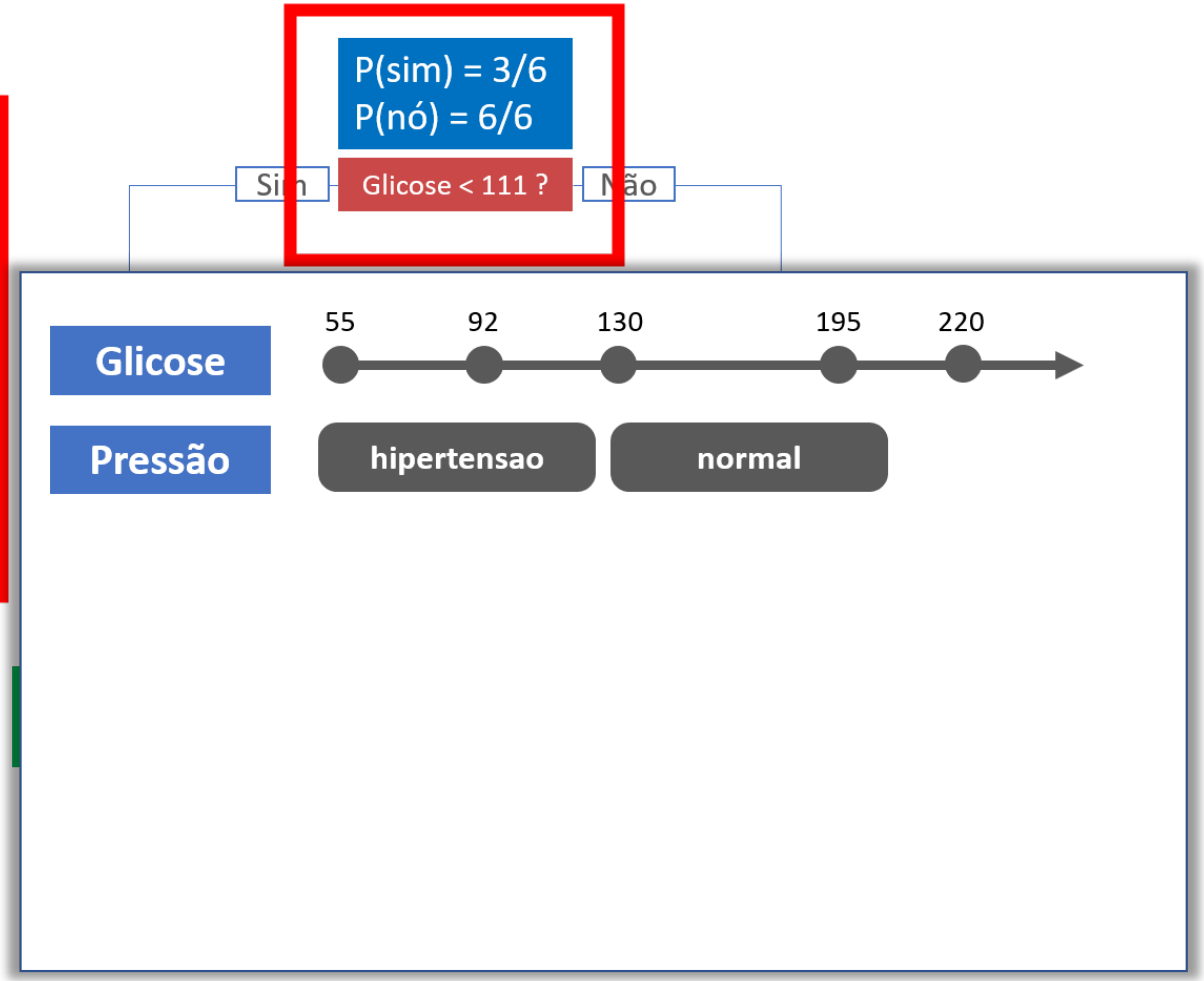


Árvore de Decisão - Perguntas

curso-r

Paciente	Pressão	Glicose	Diabetes
Alfredo	hipertensao	92	nao
Beatriz	normal	130	sim
Carla	normal	130	nao
Daniela	normal	55	nao
Ernesto	hipertensao	220	sim
Flavia	normal	195	sim

$$Y \approx f(X)$$

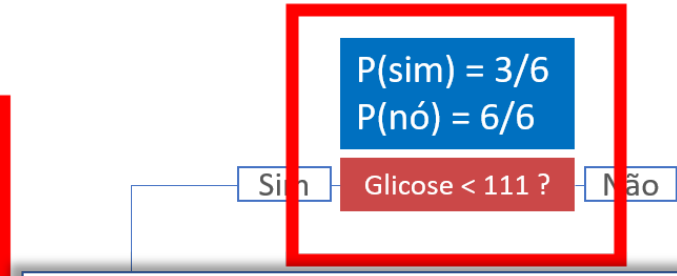


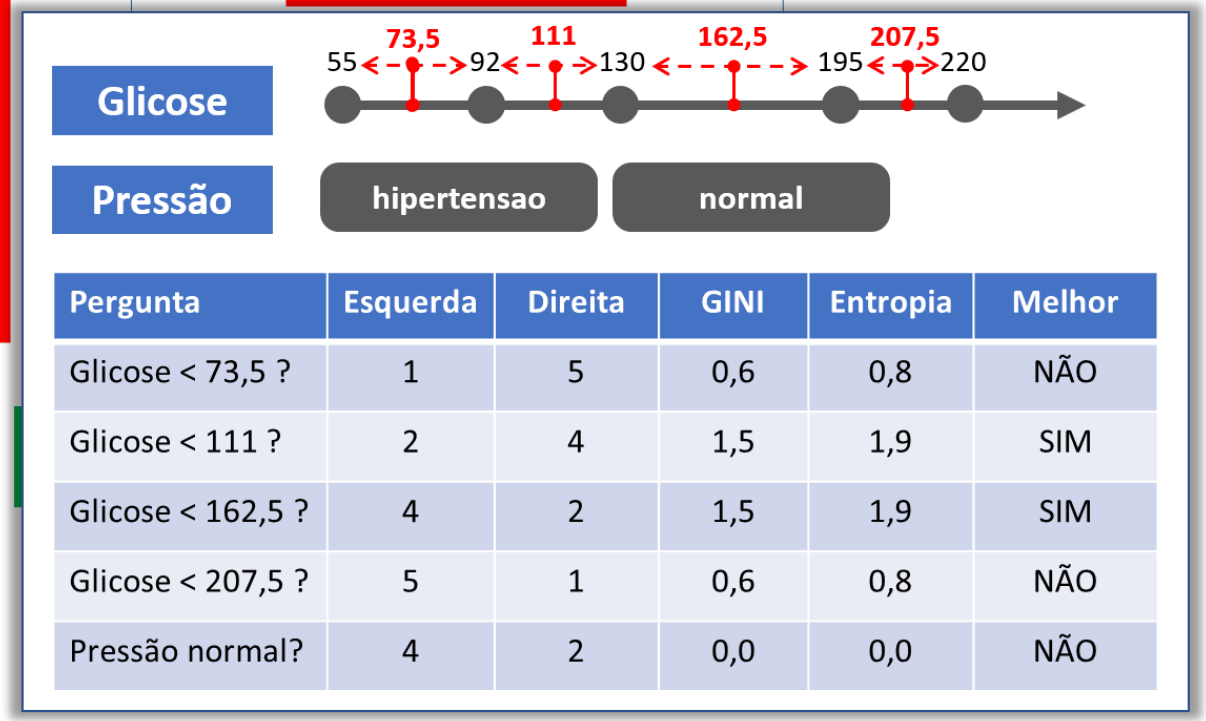
Árvore de Decisão - Perguntas

curso-r

Paciente	Pressão	Glicose	Diabetes
Alfredo	hipertensao	92	nao
Beatriz	normal	130	sim
Carla	normal	130	nao
Daniela	normal	55	nao
Ernesto	hipertensao	220	sim
Flavia	normal	195	sim

$$Y \approx f(X)$$





Pergunta	Esquerda	Direita	GINI	Entropia	Melhor
Glicose < 73,5 ?	1	5	0,6	0,8	NÃO
Glicose < 111 ?	2	4	1,5	1,9	SIM
Glicose < 162,5 ?	4	2	1,5	1,9	SIM
Glicose < 207,5 ?	5	1	0,6	0,8	NÃO
Pressão normal?	4	2	0,0	0,0	NÃO

Árvore de Decisão - Impureza e Ganho de Informação

curso-r

Ganho de Informação (information gain)

$$\text{GI} = N.Imp(\text{nó}) - N(esq).Imp(esq) - N(dir).Imp(dir)$$

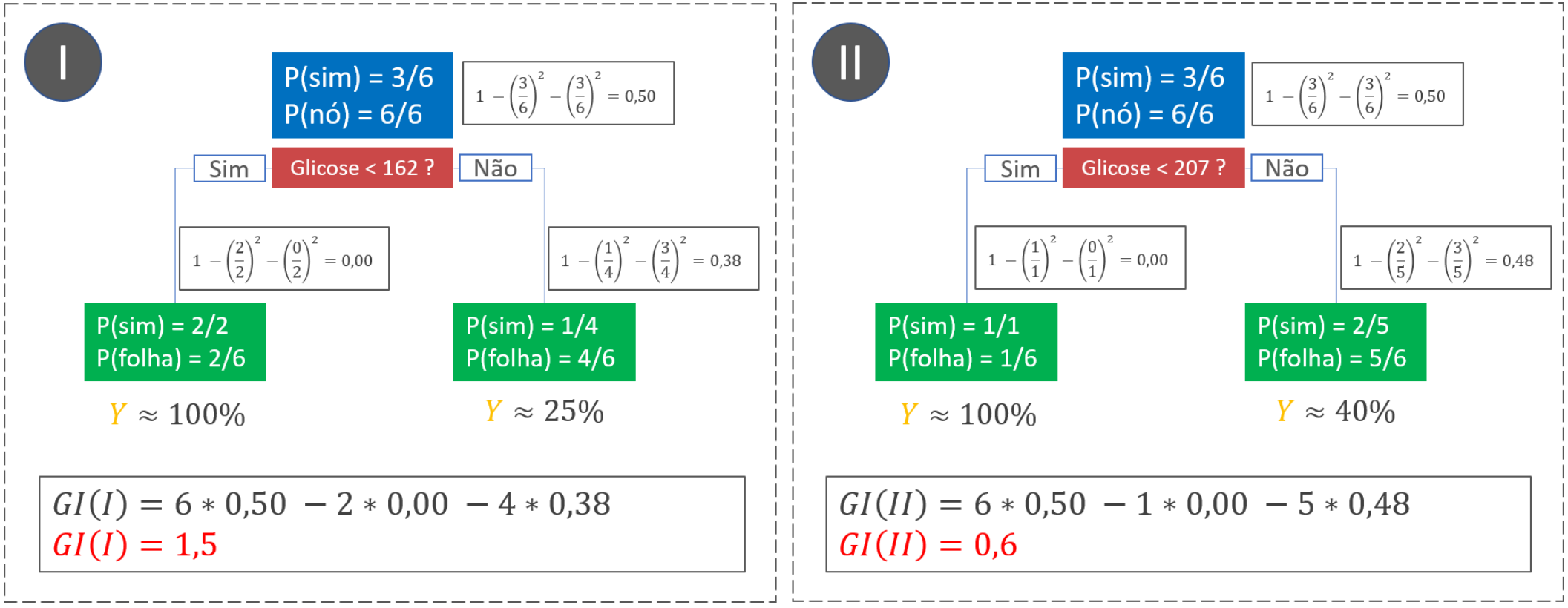
Medidas de Impureza mais comuns

Impureza	Tarefa	Fórmula	Descrição
GINI	Classificação	$1 - \sum p_i^2$	p_i é a proporção do rótulo i, $i = 1, \ldots, C$.
Entropia	Classificação	$-\sum p_i \log(p_i)$	p_i é a proporção do rótulo i, $i = 1, \ldots, C$.
Variância	Regressão	$\frac{1}{N}\sum(y_k - \hat{y}_k)^2$	y_ké o observado e $\hat{y}_k$ é a média da folha.

Fonte: spark.apache.org/docs/1.3.0/mllib-decision-tree.html

Árvore de Decisão - Impureza e Ganho de Informação

Exemplo usando o GINI



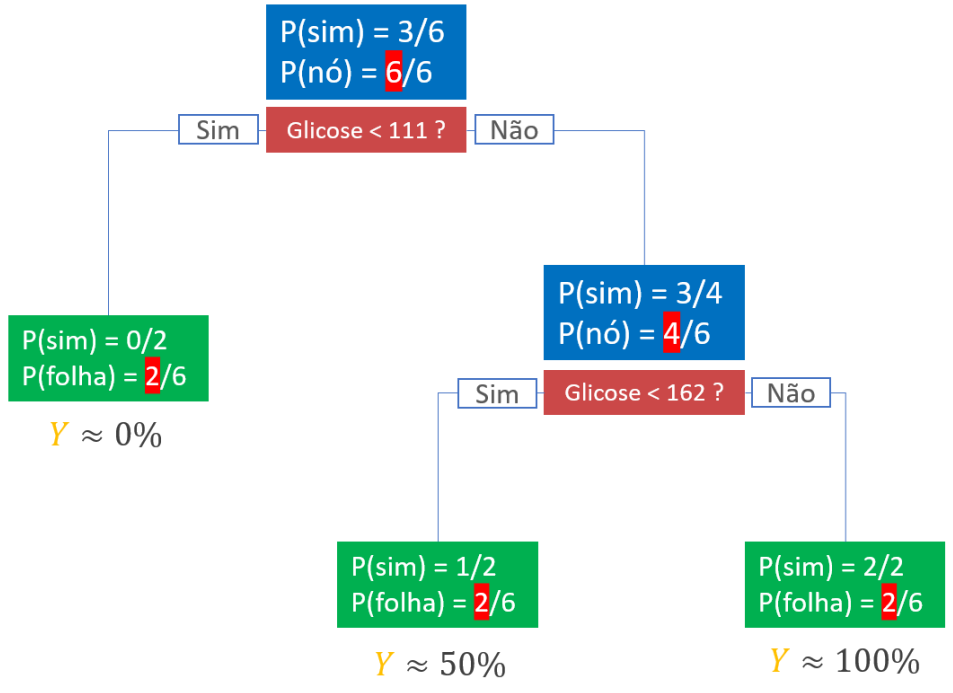
Árvore de Decisão - Hiperparâmetros e Overfitting

curso-r

min_n - Quantidade mínima de observações dentro de um nó para se considerar dividir em duas folhas novas. Quanto menor, maior risco de overfitting.

tree_depth - Profundidade: quanto mais profunda a árvore for, maior risco de overfitting.

cost_complexity - Parâmetro de complexidade: limite mínimo de ganho de informação que a divisão tem que fornecer para concretizar a criação das folhas.



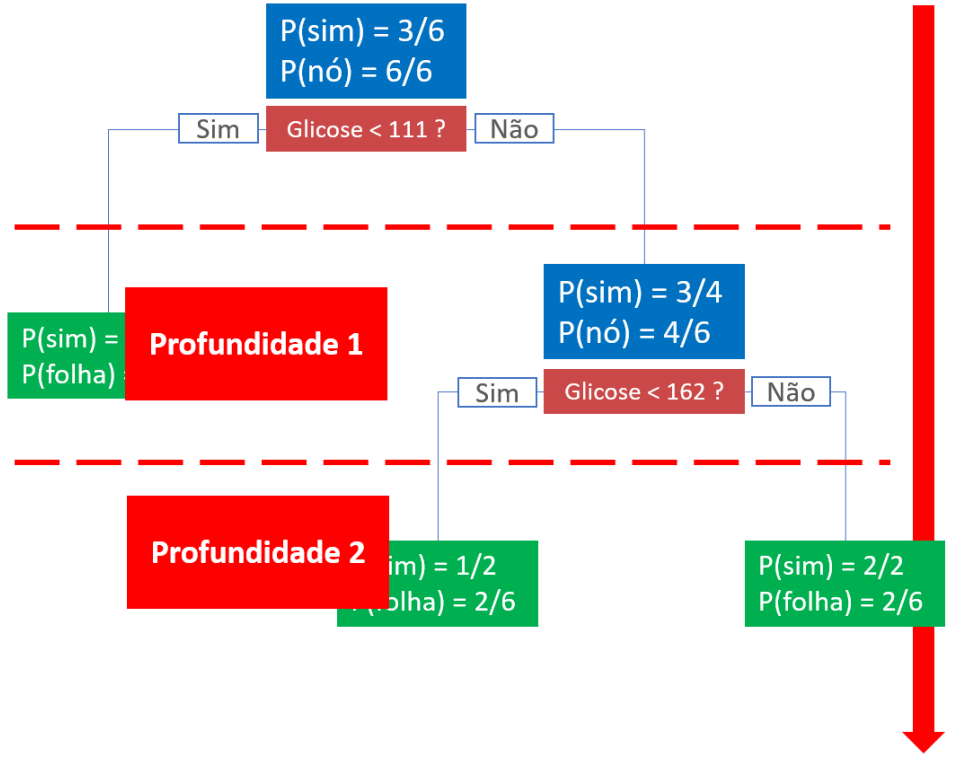
Árvore de Decisão - Hiperparâmetros e Overfitting

curso-r

min_n - Quantidade mínima de observações dentro de um nó para se considerar dividir em duas folhas novas. Quanto menor, maior risco de overfitting.

tree_depth - Profundidade: quanto mais profunda a árvore for, maior risco de overfitting.

cost_complexity - Parâmetro de complexidade: limite mínimo de ganho de informação que a divisão tem que fornecer para concretizar a criação das folhas.



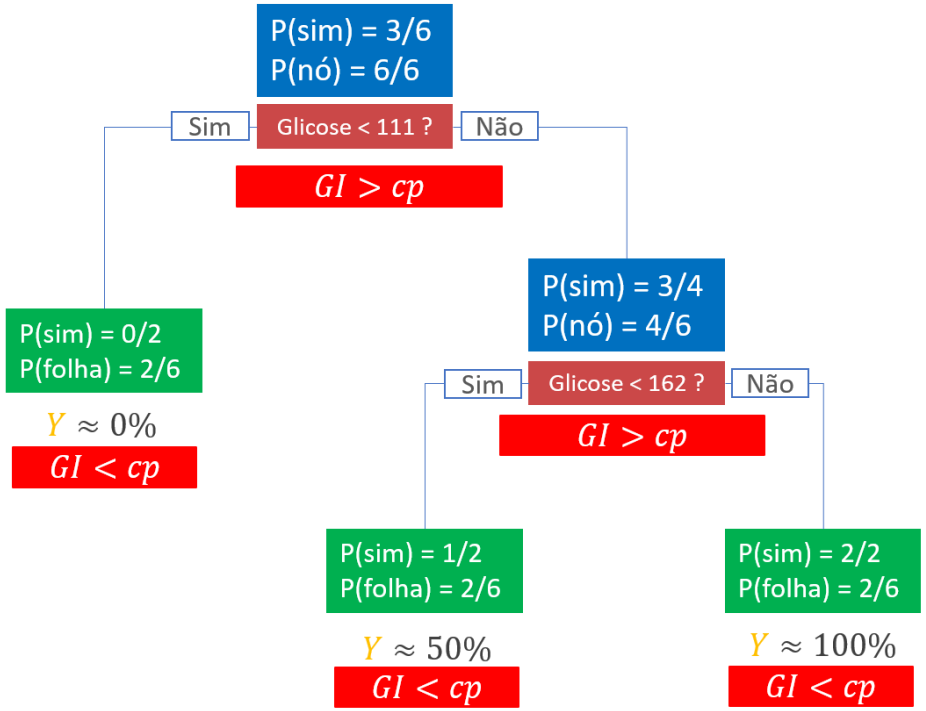
Árvore de Decisão - Hiperparâmetros e Overfitting

curso-r

min_n - Quantidade mínima de observações dentro de um nó para se considerar dividir em duas folhas novas. Quanto menor, maior risco de overfitting.

tree_depth - Profundidade: quanto mais profunda a árvore for, maior risco de overfitting.

cost_complexity - Parâmetro de complexidade: limite mínimo de ganho de informação que a divisão tem que fornecer para concretizar a criação das folhas.



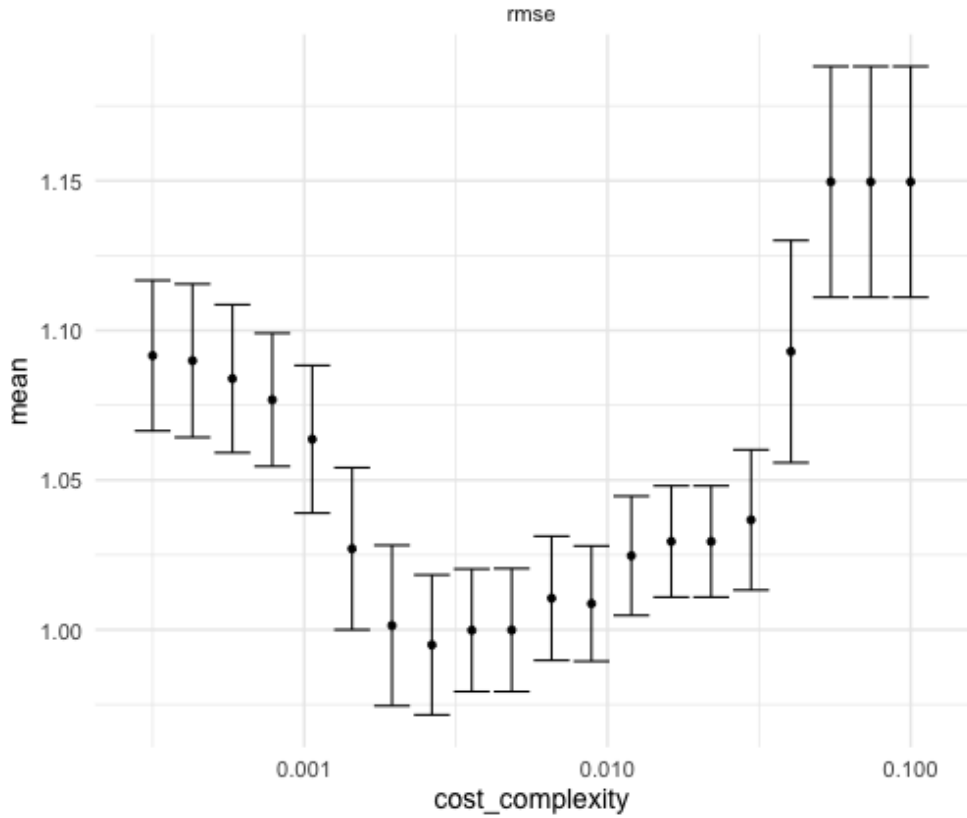
Árvore de Decisão - Cost Complexity

$$R_{cp} = R(T) + cp * |T|$$

- Quanto maior o CP, menos quebras a árvore vai ter.
- Selecionamos o tamanho de árvore ideal variando o CP (por meio de cross-validation).
- Sugere-se progressão geométrica da grade de valores.

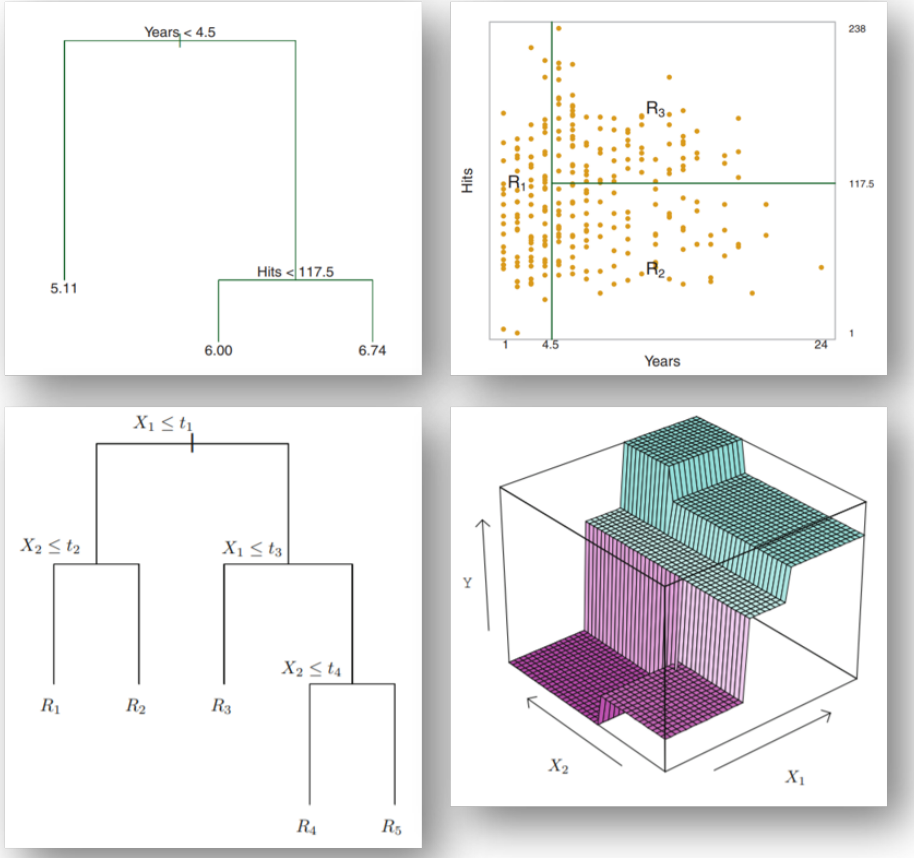
  Exemplo: $10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$

  (o `tune()` já está programado para isso).



Árvore de Decisão

- O exemplo foi dado com variável resposta (diabetes) de apenas duas classes, SIM e NÃO, mas poderia ter três ou mais.
- A variável explicativa hipertensão apresentava apenas duas classes também, mas poderia apresentar mais. Nesse caso, os algoritmos de árvores têm de decidir como fazer as PERGUNTAS. Esse link da Freakonometrics apresenta a heurística mais utilizada nesse caso.
- As figuras são representações diferentes para um mesmo modelo de árvore. As regiões $R_1, R_2, \ldots$ correspondem às folhas da árvore.



Ver ISL página 305 (Tree-based Methods).

Random Forest

Relação Viés-Variância (Bias-variance tradeoff)

curso-r

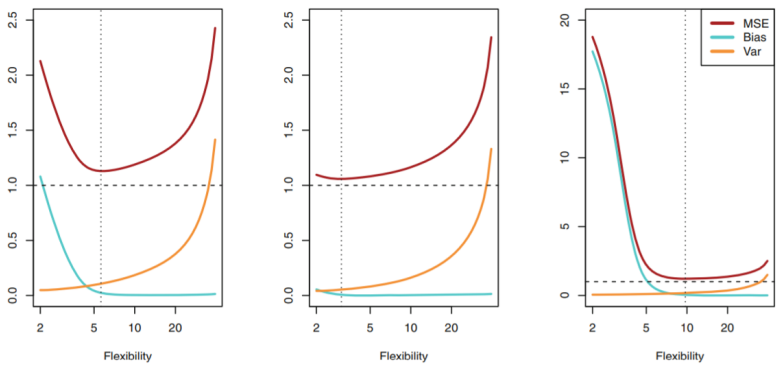
Erro de Predição Esperado

$$\mathrm{E}[(Y - \hat{f}(x_o))^2] =$$

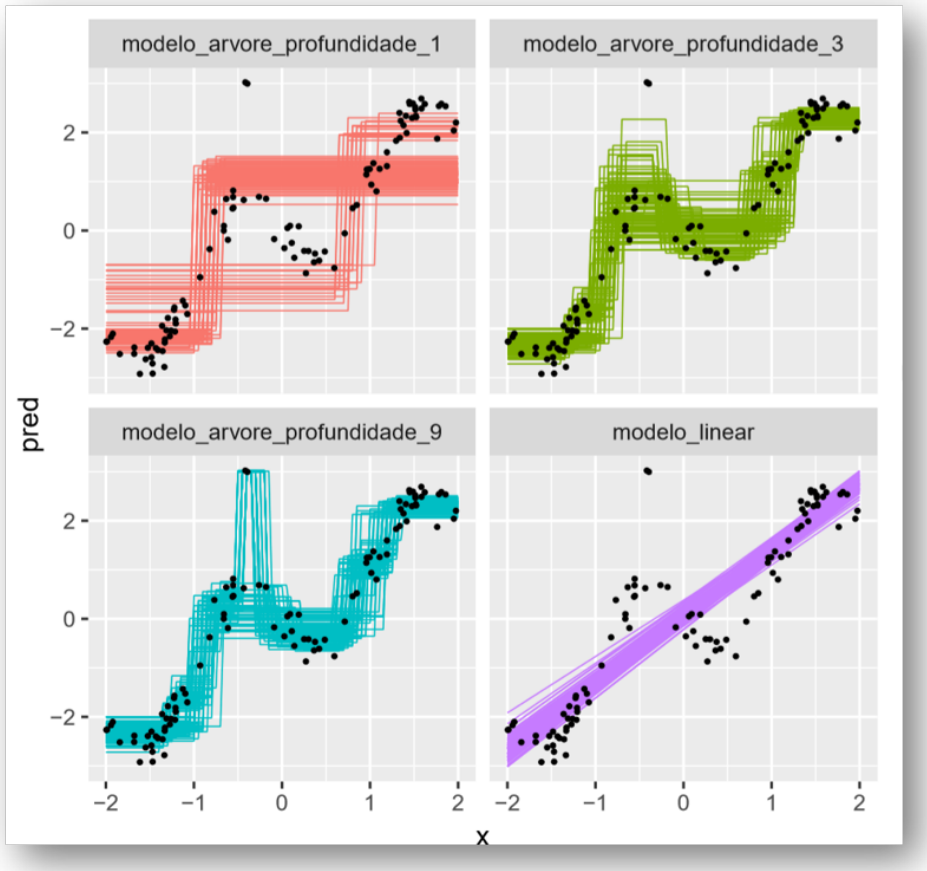
$$\mathrm{E}[(f(x_o) + \epsilon - \hat{f}(x_o))^2] =$$

$$(\mathrm{E}\hat{f}(x_o) - f(x_o))^2 + \mathrm{E}[(\hat{f}(x_o) - \mathrm{E}\hat{f}(x_o))^2] + Var(\epsilon) =$$

$$Viés^2 + Variância + Erro\ Irredutível$$



Ver ISL página 33 (The Bias-Variance Trade-Off).



Relação Viés-Variância (Bias-variance tradeoff)

curso-r

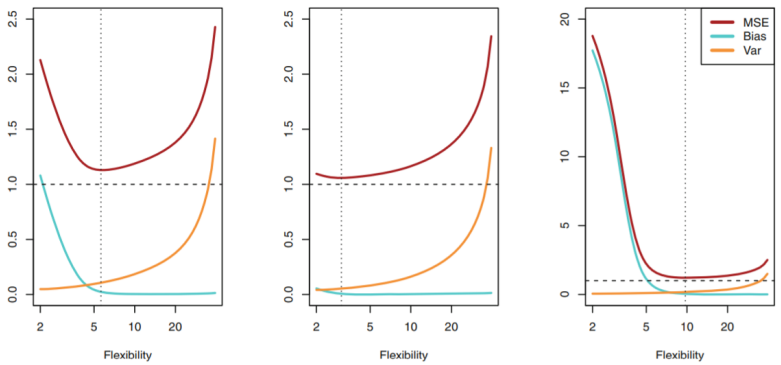
Erro de Predição Esperado

$$\mathrm{E}[(Y - \hat{f}(x_o))^2] =$$

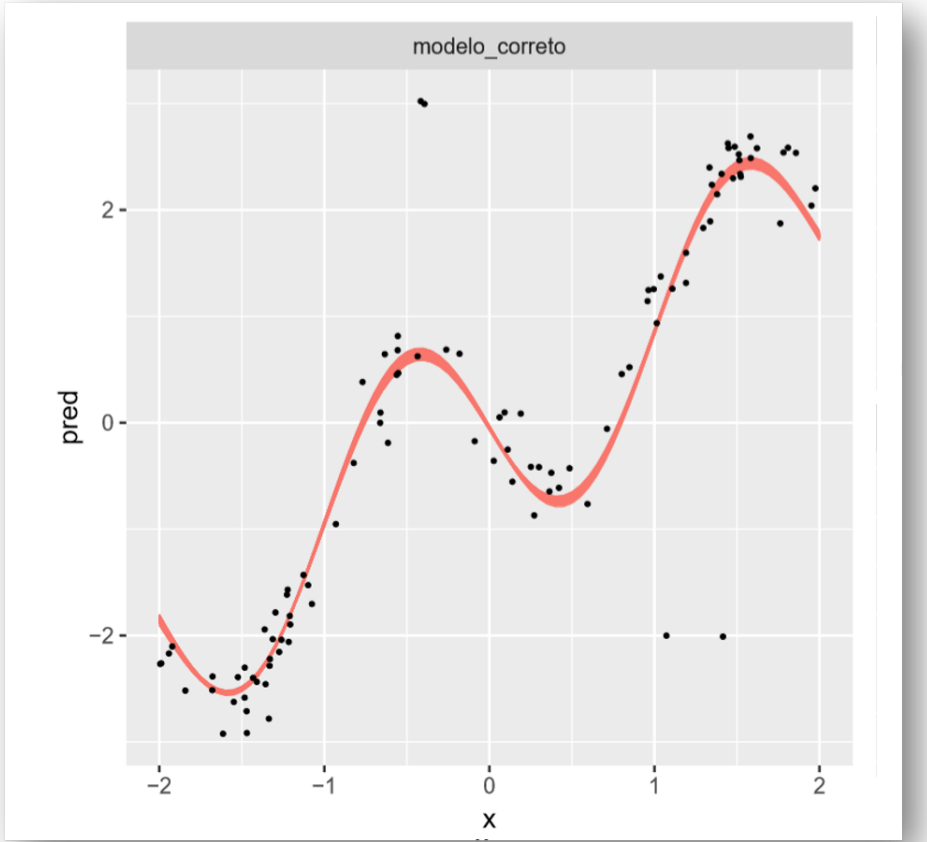
$$\mathrm{E}[(f(x_o) + \epsilon - \hat{f}(x_o))^2] =$$

$$(\mathrm{E}\hat{f}(x_o) - f(x_o))^2 + \mathrm{E}[(\hat{f}(x_o) - \mathrm{E}\hat{f}(x_o))^2] + Var(\epsilon) =$$

$$Viés^2 + Variância + Erro\ Irredutível$$



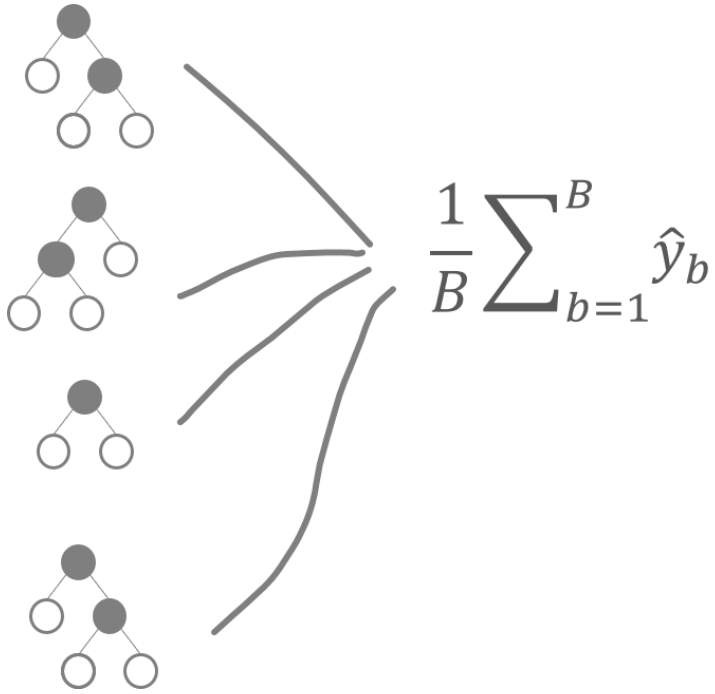
Ver ISL página 33 (The Bias-Variance Trade-Off).



Random Forest

curso-r

- **Random Forest** é a combinação de “palpites” de um monte de árvores de decisão. É um algoritmo de uma classe especial de ENSEMBLE: BAGGING.
- **ENSEMBLE**: mistura de 2 ou mais modelos. (ESL p 605)
- **BAGGING**: Bootstrap AGGregation. (ESL p 282)
- Diferença para os **BAGGINGs**: Sorteia as colunas também.



Algoritmo

1. Sorteie **B** conjuntos de observações da base **D**
2. Para cada conjunto b de **B**, sorteie m variáveis de **D**
3. Para cada uma das **B** sub-bases geradas por (b, m) construa uma árvore de decisão
4. Para previsão final, agregue as previsões individuais de cada uma das **B** árvore.

Random Forest - Hiperparâmetros e Overfitting

min_n – Qtd mínima de observações no nó para poder dividir.

mtry – Quantidade de variáveis (colunas) sorteadas por árvore. Tem que testar via cross-validation, pois é afetado pela razão entre variáveis boas e ruído.

trees – Número de árvores (amostras bootstrap) para treinar. Não afeta muito o overfitting.

PS: random forest não usa CP. Ele permite que as árvores cresçam indeterminadamente, condicionadas apenas pelo **min_n**.

Ver ISL página 319 (Random Forest).

Gradient Boosting

Gradient Boosting

- *Boosting* também é a combinação de “palpites” de um monte de árvores de decisão.
- Porém, não existe amostras *bootstrap* dentro do algoritmo, as árvores são construídas sequencialmente (cada árvore é construída usando informação da árvore passada).

Forward Stagewise Algorithm (coração do gradient boost)

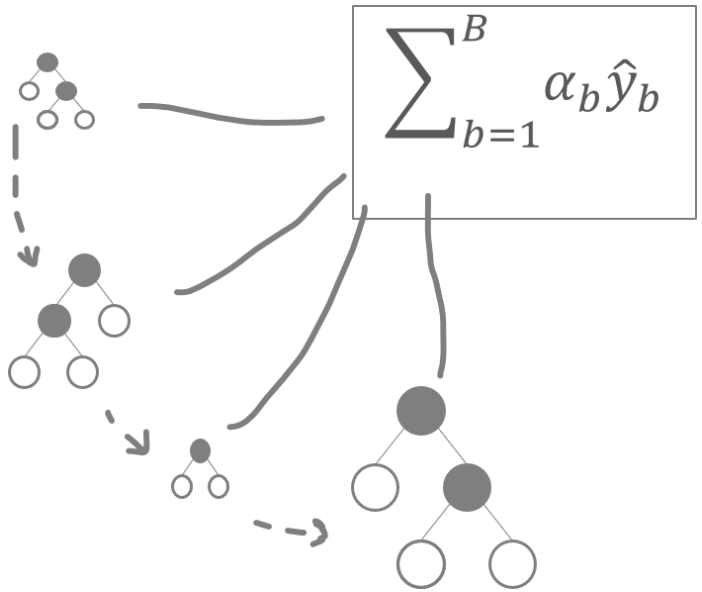
1) Inicialize $f_0(x) = 0$
2) Para $m = 1$ *a* M:
   a) Calcule:

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\big(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)\big)$$

   b) Atribua $f_m(x) = f_{m-1}(x) + \beta_m b(x_i;\gamma_m)$

EXEMPLO DE LOSS FUNCTION: $L(y_i, f(x)) = \big(y_i - f(x)\big)^2$

$$\begin{aligned} L(y_i, f(x)) &= \big(y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma)\big)^2 \\ &= \big(r_i - \beta b(x_i;\gamma)\big)^2 \end{aligned}$$



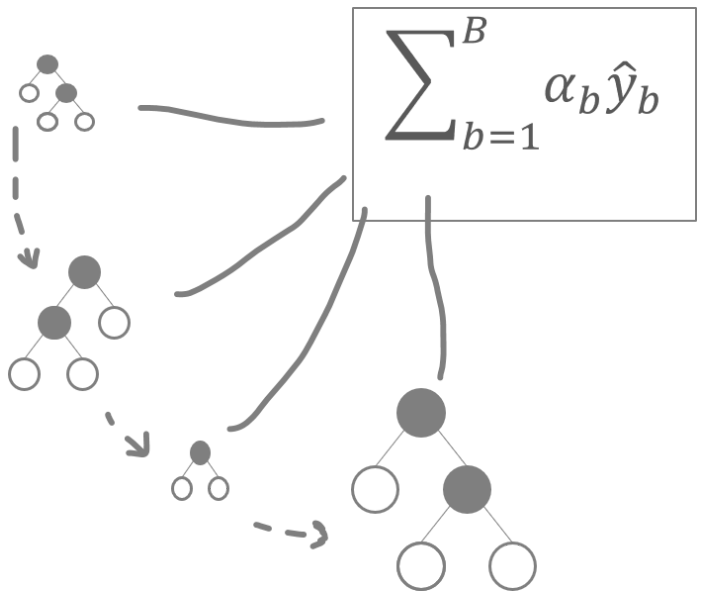
Ver ESL página 341 (Gradient Boosting).

Gradient Boosting

- *Boosting* também é a combinação de “palpites” de um monte de árvores de decisão.
- Porém, não existe amostras *bootstrap* dentro do algoritmo, as árvores são construídas sequencialmente (cada árvore é construída usando informação da árvore passada).

Adaboost (versão para classificação binária)

1) Codifique $Y \in \{-1, 1\}$.
2) Inicializa-se pesos $w_i = \frac{1}{N}, i = 1, 2, \dots, N$
3) Para $m = 1$ *a* M:
   a) Ajuste uma árvore $T_m(x)$ usando os pesos w_i.
   b) Calcule o erro de m: $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq T_m(x_i))}{\sum_{i=1}^{N} w_i}$
   c) Compute $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$
   d) Atualize os pesos: $w_i \leftarrow w_i . \exp\left(\alpha_m . I\left(y_i \neq T_m(x_i)\right)\right), i = 1, \dots, N$
4) Previsão: $sinal[\sum_{m=1}^{M} \alpha_m T(x)]$



Ver ESL página 341 (Gradient Boosting).

XGBoost

XGBoost

curso-r

- XGBoost é uma implementação melhorada do Gradient Boost.
- O XGBoost traz de volta reamostragem e hiperparâmetros de regularização.
- Top 2 de Algoritmos que mais ganharam Kaggle.

$$\text{obj}^{(t)} \approx \sum_{i=1}^{n} [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

min_n – Qtd mínima de observações no nó para poder dividir.

mtry – Quantidade de variáveis sorteadas por árvore. Tem que testar via CV, pois é afetado pela razão entre variáveis boas e ruído.

trees – Número de árvores (de passos).

Ver Introduction to Boosted Trees do XGboost.io.

tree_depth – Profundidade máxima da árvore.

learn_rate – Tamanho do "passo". Quanto menor, mais devagar. PS: Aumentar o número de árvores junto!

loss_reduction – Parâmetro regularizador. Análogo ao CP do `rpart`.

sample_size – Proporção de linhas para sortear por árvore.

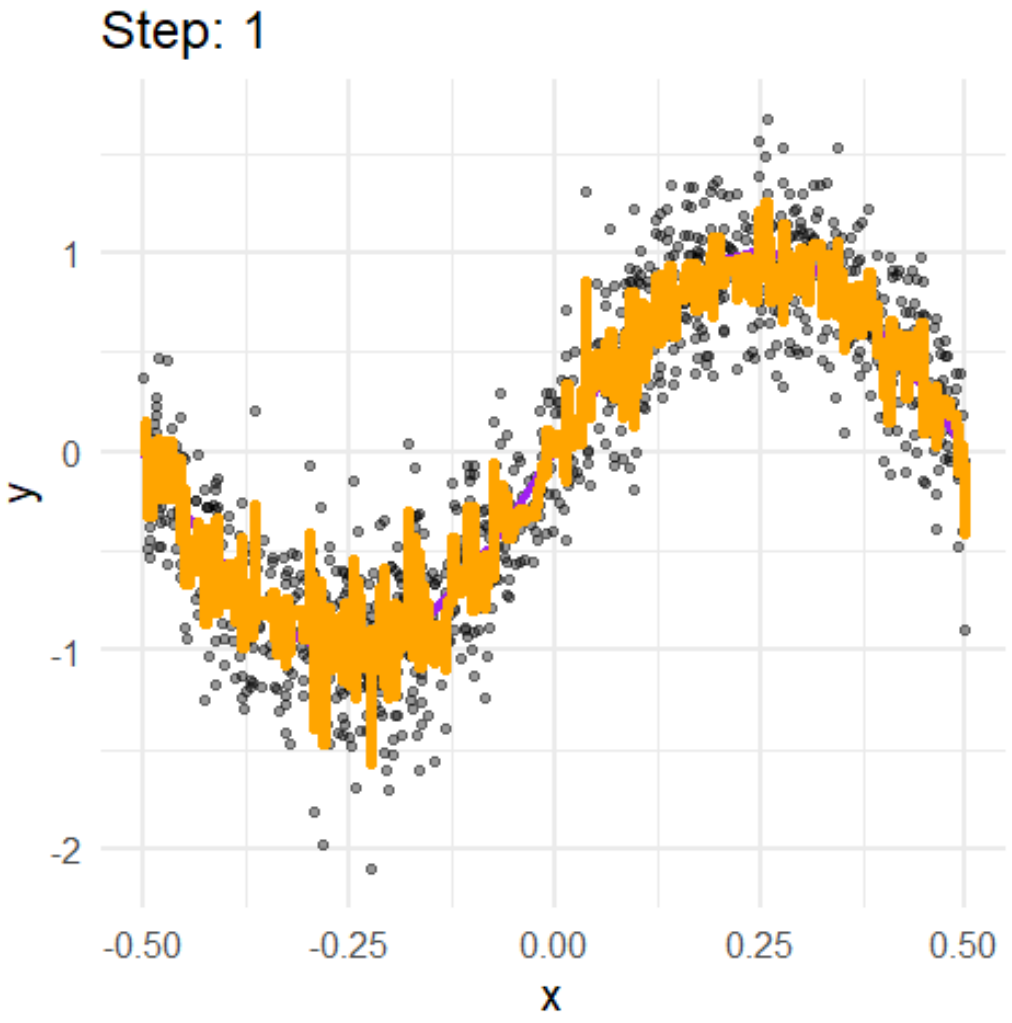
Sobre os problemas nos dados

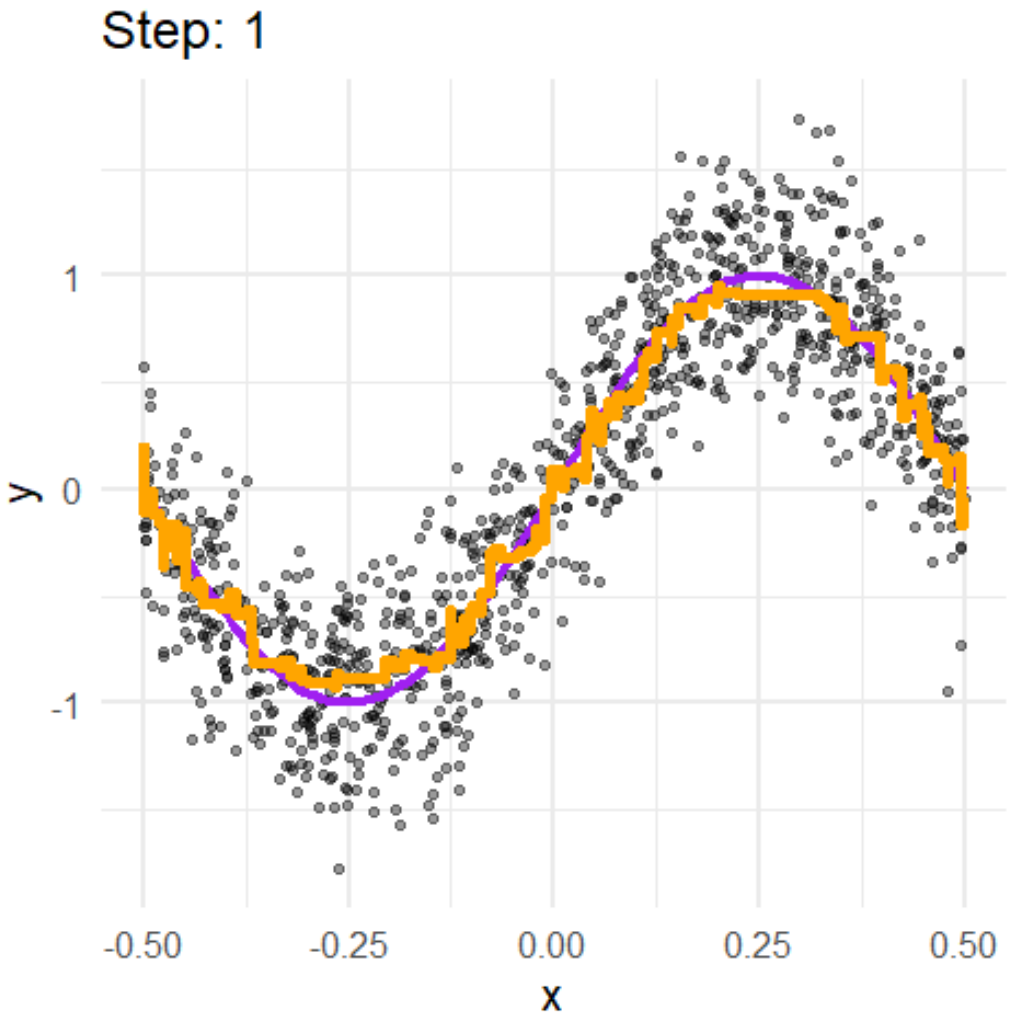
curso-r

- XGBoost trata missing automaticamente dentro dele, não precisa tratar. Porém, sempre vale técnicas de imputação para tentar aprimorar o modelo!
- Multicolinearidade não é um problema grave para modelos de árvore. Mas é sempre bom filtrar variáveis explicativas muito correlacionadas. Ler esse post para exemplo.
- Variável resposta precisa ir como factor. Não pode ser character nem 0/1.
- As variáveis categóricas precisam ser "dummyficadas" antes.
- A escala das variáveis explicativas não atrapalham modelos de árvores.
- A assimetria das variáveis explicativas não atrapalham modelos de árvores.

XGboost - Intuição dos hiperparâmetros

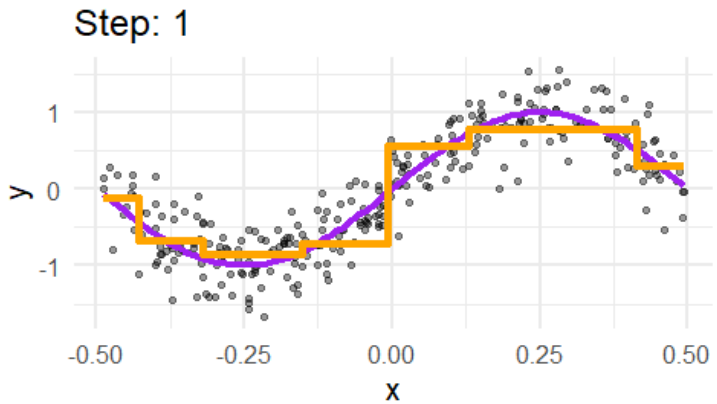
curso-r



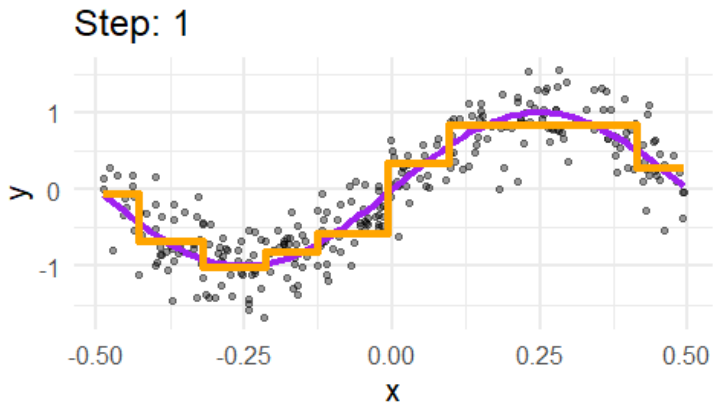


XGboost - Intuição dos hiperparâmetros

```
modelo <- boost_tree(
  mtry = 1,
  trees = 100,
  min_n = 1,
  tree_depth = 1,
  learn_rate = 1,
  sample_size = 1,
  loss_reduction = 1
)
```



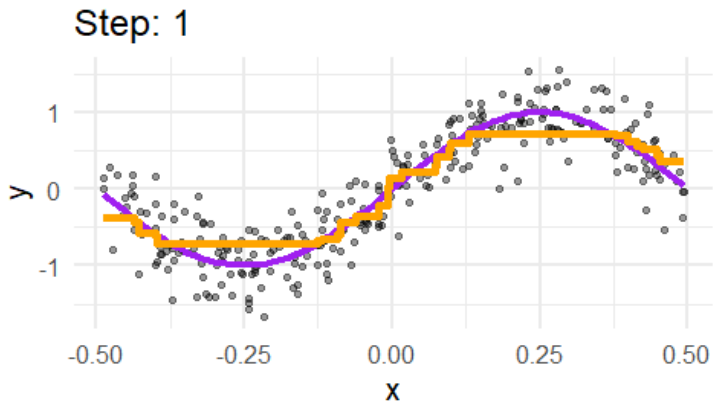

```
modelo <- boost_tree(
  mtry = 1,
  trees = 100,
  min_n = 1,
  tree_depth = 2,
  learn_rate = 1,
  sample_size = 1,
  loss_reduction = 1
)
```



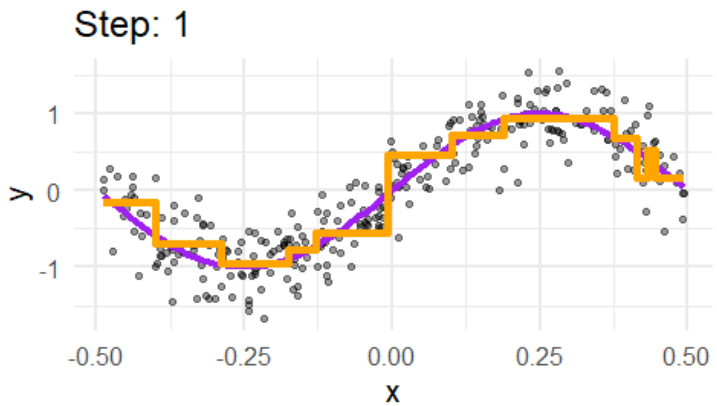
XGboost - Intuição dos hiperparâmetros

curso-r

```
modelo <- boost_tree(
  mtry = 1,
  trees = 100,
  min_n = 1,
  tree_depth = 1,
  learn_rate = 0.1,
  sample_size = 1,
  loss_reduction = 1
)
```

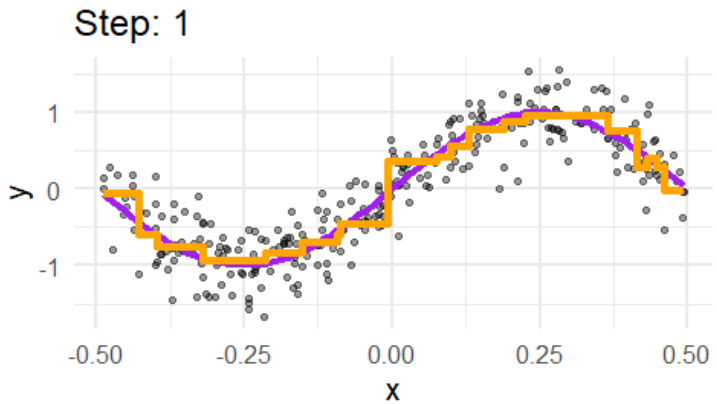



```
modelo <- boost_tree(
  mtry = 1,
  trees = 100,
  min_n = 1,
  tree_depth = 1,
  learn_rate = 1,
  sample_size = 0.5,
  loss_reduction = 1
)
```

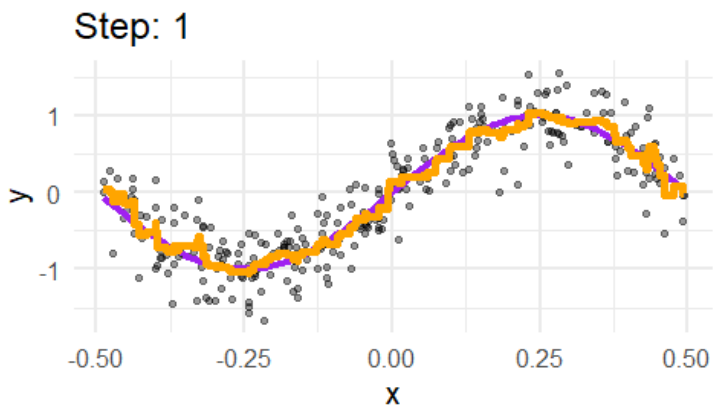


XGboost - Intuição dos hiperparâmetros

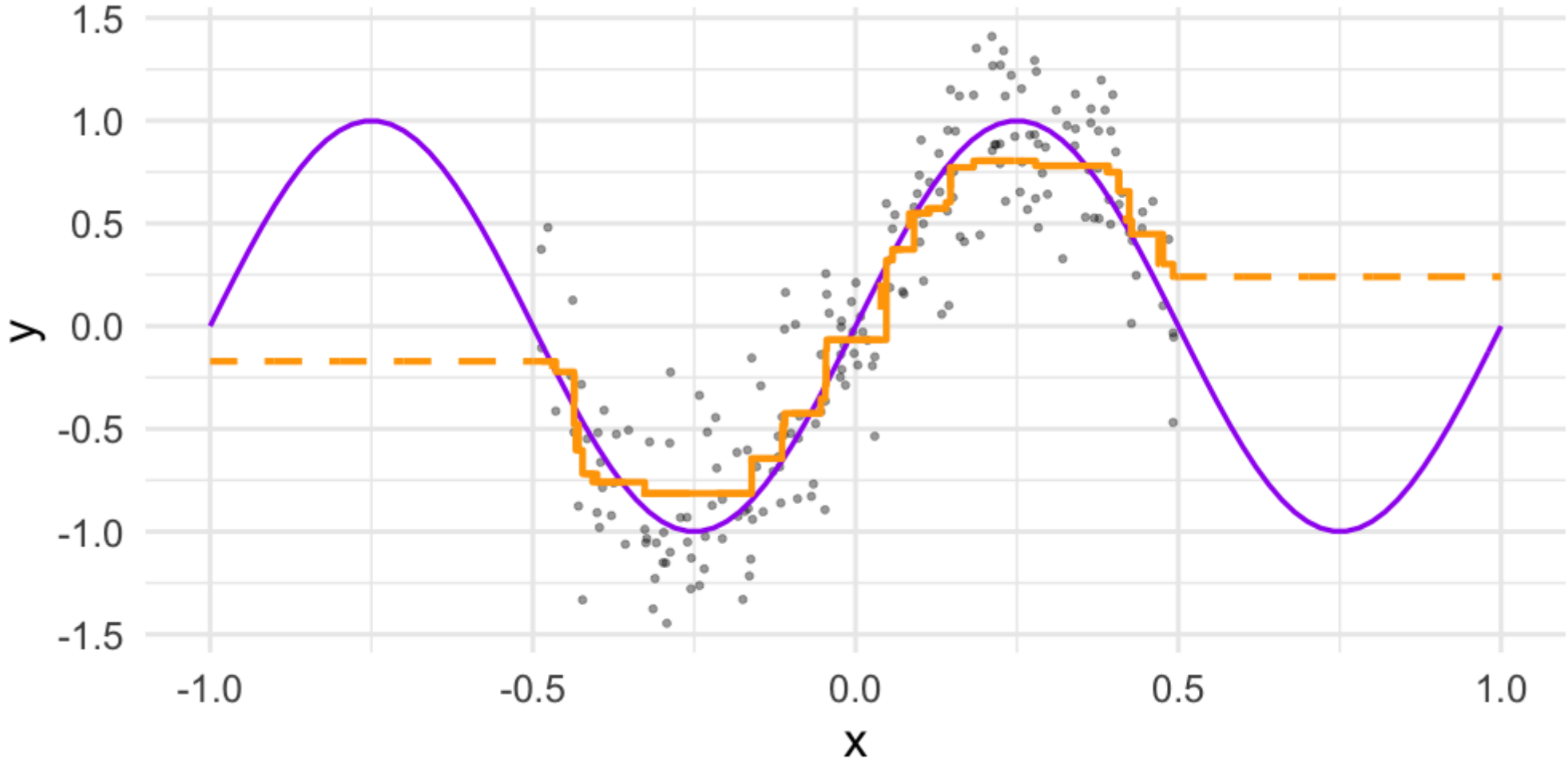
```
modelo <- boost_tree(
  mtry = 1,
  trees = 100,
  min_n = 1,
  tree_depth = 1,
  learn_rate = 1,
  sample_size = 1,
  loss_reduction = 0.1
)
```




```
modelo <- boost_tree(
  mtry = 1,
  trees = 100,
  min_n = 1,
  tree_depth = 2,
  learn_rate = 0.1,
  sample_size = 0.5,
  loss_reduction = 0.1
)
```



Extrapolação dos modelos de árvores



No R

curso-r

```
# árvore de decisão
modelo_tree <- decision_tree(
  min_n = tune(),
  tree_depth = tune(),
  cost_complexity = tune()
)
```

```
# Random Forest
modelo_rf <- rand_forest(
  min_n = tune(),
  mtry = tune(),
  trees = tune()
)
```

```
# XGBoost
modelo_xgb <- boost_tree(
  min_n = tune(),
  mtry = tune(),
  trees = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
  sample_size = tune()
)
```