Boosting

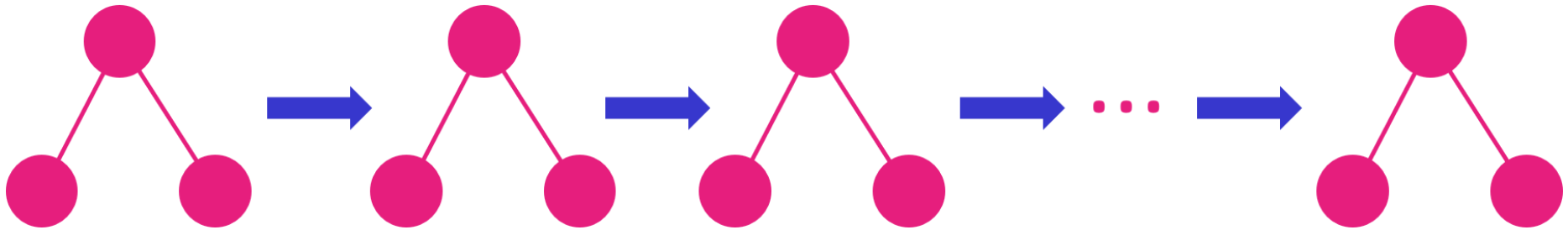
Tiago Mendonça dos Santos

tiagoms.com
tiagomendonca
tiagoms1@insper.edu.br

Introdução

Boosting

É um procedimento que combina diversos *weak classifiers* (classificadores que são apenas um pouco melhores que um preditor aleatório) para produzir um **comitê** capaz de fazer boas previsões.



Cada classificador depende do classificador anterior (note a diferença em relação à floresta aleatória).

Nessa aula trabalharemos com árvores de decisão, mas essa técnica pode ser utilizada considerando outros métodos de aprendizagem estatística.

A seguir apresentamos um dos algoritmos de *boosting* mais populares (AdaBoost.M1) desenvolvido por Yoav Freund and Robert Schapire.

Gödel Prize - 2003

Yoav Freund and Robert Shapire

The 2003 Gödel Prize for an outstanding journal article or articles in theoretical computer science is awarded to Yoav Freund and Robert Schapire for their paper "A Decision Theoretic Generalization of On-Line Learning and an Application to Boosting" Journal of Computer and System Sciences 55 (1997), pp. 119-139.

This paper introduced AdaBoost, an adaptive algorithm to improve the accuracy of hypotheses in machine learning. The algorithm demonstrated novel possibilities in analysing data and is a permanent contribution to science even beyond computer science.

Because of a combination of features, including its elegance, the simplicity of its implementation, its wide applicability, and its striking success in reducing errors in benchmark applications even while its theoretical assumptions are not known to hold, the algorithm set off an explosion of research in the fields of statistics, artificial intelligence, experimental machine learning, and data mining. The algorithm is now widely used in practice.

The paper highlights the fact that theoretical computer science continues to be a fount of powerful and entirely novel ideas with significant and direct impact even in areas, such as data analysis, that have been studies extensively by other communities.

AdaBoost.M1

Considere um problema de classificação com duas classes e a resposta codificada como $Y \in \{-1, +1\}$. Nesse caso, a taxa de erro no conjunto de treinamento é dada por:

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(\mathbf{x}_i)),$$

em que $G(\mathbf{x})$ é o classificador que resulta em um dos valores $\{-1, +1\}$.

Qual seria o valor de $\overline{\text{err}}$ na situação a seguir?

y	g(x)
-1	-1
-1	+1
+1	+1
+1	-1
-1	-1

AdaBoost.M1

Considere um problema de classificação com duas classes e a resposta codificada como $Y \in \{-1, +1\}$. Nesse caso, a taxa de erro no conjunto de treinamento é dada por:

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(\mathbf{x}_i)),$$

em que $G(\mathbf{x})$ é o classificador que resulta em um dos valores $\{-1, +1\}$.

Qual seria o valor de $\overline{\text{err}}$ na situação a seguir?

y	g(x)	indicadora
-1	-1	0
-1	+1	1
+1	+1	0
+1	-1	1
-1	-1	0

AdaBoost.M1

A ideia é combinar os resultados de inúmeros *weak classifiers* (classificadores que são apenas um pouco melhores que um preditor aleatório) para fazer a previsão a partir desse comitê de classificadores da seguinte forma:

$$G(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right),$$

em que $\alpha_i, i = 1, \ldots M$ são calculados pelo algortimo e ponderam a contribuição de cada classificador.

Exemplo

Para um dado $\mathbf{x}$ considere $\alpha_1 = 1.1$, $\alpha_2 = 1.3$, $\alpha_3 = 1.2$, $G_1(x) = +1$, $G_2(x) = +1$ e $G_3(x) = -1$. Qual seria a classe predita?

$$G(\mathbf{x}) = \text{sign}\Big((1.1 \times 1) + (1.3 \times 1) + (1.2 \times -1)\Big) = \text{sign}\big(1.2\big) = +1.$$

Como determinar a contribuição de cada classificador dada pelos valores de α?

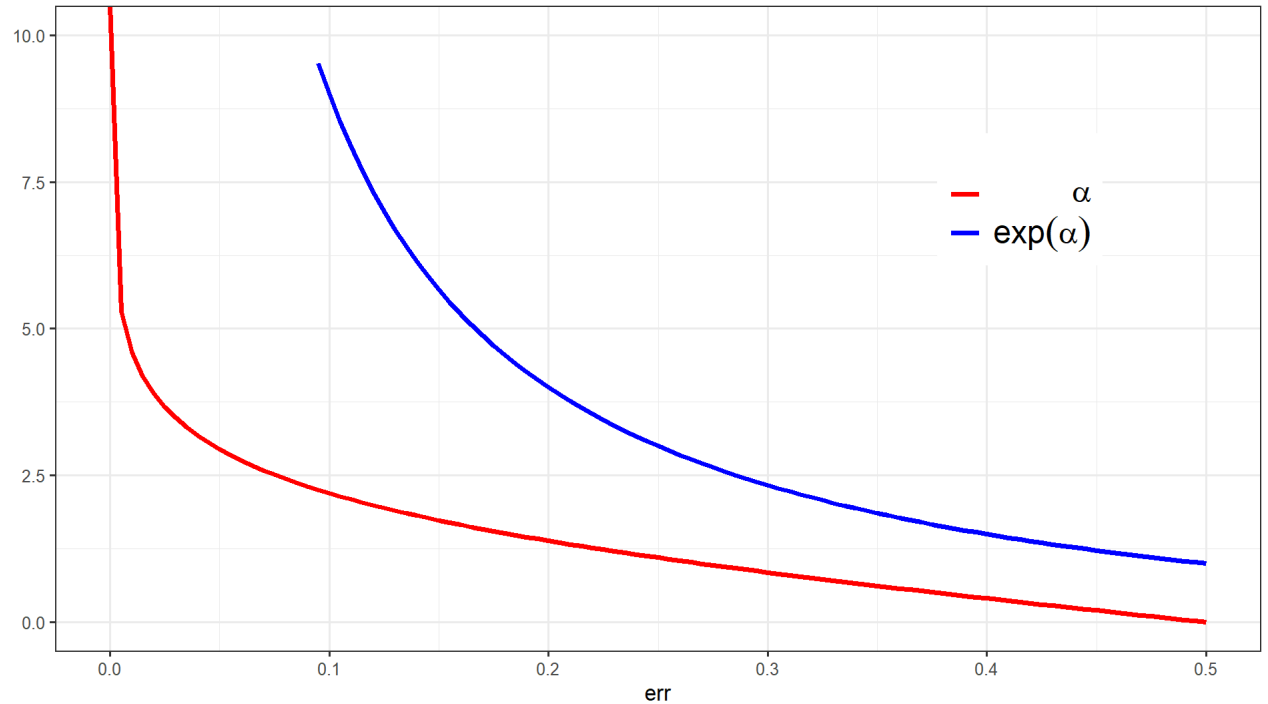
AdaBoost.M1

- Inicialize o peso das observações $w_i = 1/N$ com $i = 1, \ldots, N$.
- Para m variando de 1 a M:
  - Ajuste um classificador $G_m(\mathbf{x})$ ao conjunto de treinamento utilizando os pesos w_i.
  - Calcule $\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbf{I}(y_i \neq G_m(\mathbf{x}))}{\sum_{i=1}^{N} w_i}$
  - Calcule $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$.
  - Atualize $w_i \leftarrow w_i \cdot \exp\{\alpha_m \cdot \mathbf{I}(y_i \neq G_m(\mathbf{x}))\},\ i = 1, 2, \ldots N$
- Retorne $G(\mathbf{x}) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right]$.

descrição apresentada no livro *The Elements of Statistical Learning*.

AdaBoost.M1

$$\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right) \text{ e } w_i \leftarrow w_i \cdot \exp\{\alpha_m \cdot \mathbf{I}(y_i \neq G_m(\mathbf{x}))\}$$



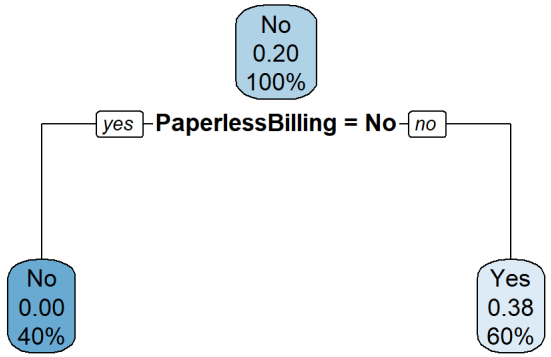
Para erro $w_i \leftarrow w_i \cdot \exp\{\alpha_m\}$ e, para acerto, $w_i \leftarrow w_i$

Churn

Vamos utilizar como exemplo os dados **Telco Customer Churn**.

id	obs	w_1
1	No	1/15
2	No	1/15
3	Yes	1/15
4	No	1/15
5	Yes	1/15
6	Yes	1/15
7	No	1/15
8	No	1/15
9	Yes	1/15
10	No	1/15
11	No	1/15
12	No	1/15
13	No	1/15
14	Yes	1/15
15	No	1/15

Churn



Churn

id	obs	w_1	pred_1
1	No	1/15	Yes
2	No	1/15	No
3	Yes	1/15	Yes
4	No	1/15	No
5	Yes	1/15	Yes
6	Yes	1/15	Yes
7	No	1/15	Yes
8	No	1/15	No
9	Yes	1/15	Yes
10	No	1/15	No
11	No	1/15	Yes
12	No	1/15	No
13	No	1/15	No
14	Yes	1/15	Yes
15	No	1/15	Yes

$$\text{err}_1 = \frac{\sum_{i=1}^{15} w_i I(y_i \neq G_1(x))}{\sum_{i=1}^{15} w_i}$$

$$\text{err}_1 = \frac{1/15+1/15+1/15+1/15}{1} = 4/15$$

Churn

id	obs	w_1	pred_1
1	No	1/15	Yes
2	No	1/15	No
3	Yes	1/15	Yes
4	No	1/15	No
5	Yes	1/15	Yes
6	Yes	1/15	Yes
7	No	1/15	Yes
8	No	1/15	No
9	Yes	1/15	Yes
10	No	1/15	No
11	No	1/15	Yes
12	No	1/15	No
13	No	1/15	No
14	Yes	1/15	Yes
15	No	1/15	Yes

$$\text{err}_1 = \frac{\sum_{i=1}^{15} w_i I(y_i \neq G_1(x))}{\sum_{i=1}^{15} w_i}$$

$$\text{err}_1 = \frac{1/15+1/15+1/15+1/15}{1} = 4/15$$

$$\alpha_1 = \log\left(\frac{1-\text{err}_1}{\text{err}_1}\right) = \log\left(\frac{11/15}{4/15}\right)$$

$$\alpha_1 = 1.011601$$

Churn

id	obs	w_1	pred_1
1	No	1/15	Yes
2	No	1/15	No
3	Yes	1/15	Yes
4	No	1/15	No
5	Yes	1/15	Yes
6	Yes	1/15	Yes
7	No	1/15	Yes
8	No	1/15	No
9	Yes	1/15	Yes
10	No	1/15	No
11	No	1/15	Yes
12	No	1/15	No
13	No	1/15	No
14	Yes	1/15	Yes
15	No	1/15	Yes

$$\text{err}_1 = \frac{\sum_{i=1}^{15} w_i I(y_i \neq G_1(x))}{\sum_{i=1}^{15} w_i}$$

$$\text{err}_1 = \frac{1/15+1/15+1/15+1/15}{1} = 4/15$$

$$\alpha_1 = \log\left(\frac{1-\text{err}_1}{\text{err}_1}\right) = \log\left(\frac{11/15}{4/15}\right)$$

$$\alpha_1 = 1.011601$$

Atualize

$$w_2 \leftarrow w_1 \cdot \exp\{\alpha_1 \cdot \mathbf{I}(y_i \neq G_1(\mathbf{x}))\}$$

Churn

id	obs	w_1	pred_1
1	No	1/15	Yes
2	No	1/15	No
3	Yes	1/15	Yes
4	No	1/15	No
5	Yes	1/15	Yes
6	Yes	1/15	Yes
7	No	1/15	Yes
8	No	1/15	No
9	Yes	1/15	Yes
10	No	1/15	No
11	No	1/15	Yes
12	No	1/15	No
13	No	1/15	No
14	Yes	1/15	Yes
15	No	1/15	Yes

$$\text{err}_1 = \frac{\sum_{i=1}^{15} w_i I(y_i \neq G_1(x))}{\sum_{i=1}^{15} w_i}$$

$$\text{err}_1 = \frac{1/15+1/15+1/15+1/15}{1} = 4/15$$

$$\alpha_1 = \log\left(\frac{1-\text{err}_1}{\text{err}_1}\right) = \log\left(\frac{11/15}{4/15}\right)$$

$$\alpha_1 = 1.011601$$

Atualize
$w_2 \leftarrow w_1 \cdot \exp\{\alpha_1 \cdot \mathbf{I}(y_i \neq G_1(\mathbf{x}))\}$

Para os **erros**
$w_2 \leftarrow w_1 \cdot \exp\{1.011601\} = w_1 \cdot 2.75$

Para os **acertos** $w_2 \leftarrow w_1 \cdot \exp\{0\} = w_1$

Churn

id	obs	w_1	pred_1	w_2
1	No	1/15	Yes	0.183
2	No	1/15	No	0.067
3	Yes	1/15	Yes	0.067
4	No	1/15	No	0.067
5	Yes	1/15	Yes	0.067
6	Yes	1/15	Yes	0.067
7	No	1/15	Yes	0.183
8	No	1/15	No	0.067
9	Yes	1/15	Yes	0.067
10	No	1/15	No	0.067
11	No	1/15	Yes	0.183
12	No	1/15	No	0.067
13	No	1/15	No	0.067
14	Yes	1/15	Yes	0.067
15	No	1/15	Yes	0.183

$$\text{err}_1 = \frac{\sum_{i=1}^{15} w_i I(y_i \neq G_1(x))}{\sum_{i=1}^{15} w_i}$$

$$\text{err}_1 = \frac{1/15+1/15+1/15+1/15}{1} = 4/15$$

$$\alpha_1 = \log\left(\frac{1-\text{err}_1}{\text{err}_1}\right) = \log\left(\frac{11/15}{4/15}\right)$$

$$\alpha_1 = 1.011601$$

Atualize

$$w_2 \leftarrow w_1 \cdot \exp\{\alpha_1 \cdot \mathbf{I}(y_i \neq G_1(\mathbf{x}))\}$$

Churn

id	obs	w_1	pred_1	w_2	pred_2
1	No	1/15	Yes	0.183	No
2	No	1/15	No	0.067	No
3	Yes	1/15	Yes	0.067	No
4	No	1/15	No	0.067	No
5	Yes	1/15	Yes	0.067	No
6	Yes	1/15	Yes	0.067	No
7	No	1/15	Yes	0.183	No
8	No	1/15	No	0.067	No
9	Yes	1/15	Yes	0.067	No
10	No	1/15	No	0.067	No
11	No	1/15	Yes	0.183	No
12	No	1/15	No	0.067	No
13	No	1/15	No	0.067	No
14	Yes	1/15	Yes	0.067	No
15	No	1/15	Yes	0.183	No

$$\text{err}_2 = \frac{\sum_{i=1}^{15} w_i I(y_i \neq G_2(x))}{\sum_{i=1}^{15} w_i}$$

$$\text{err}_2 = \frac{0.067+0.067+0.067+0.067+0.067}{1.469}$$

$$\text{err}_2 = 0.228$$

$$\alpha_2 = \log\left(\frac{1-\text{err}_2}{\text{err}_2}\right)$$

$$\alpha_2 = \log\left(\frac{1-0.228}{0.228}\right) = \log(3.385)$$

$$\alpha_2 = 1.219$$

Atualize

$$w_3 \leftarrow w_2 \cdot \exp\{\alpha_2 \cdot \mathbf{I}(y_i \neq G_1(\mathbf{x}))\}$$

Churn

id	obs	w_1	pred_1	w_2	pred_2
1	No	1/15	Yes	0.183	No
2	No	1/15	No	0.067	No
3	Yes	1/15	Yes	0.067	No
4	No	1/15	No	0.067	No
5	Yes	1/15	Yes	0.067	No
6	Yes	1/15	Yes	0.067	No
7	No	1/15	Yes	0.183	No
8	No	1/15	No	0.067	No
9	Yes	1/15	Yes	0.067	No
10	No	1/15	No	0.067	No
11	No	1/15	Yes	0.183	No
12	No	1/15	No	0.067	No
13	No	1/15	No	0.067	No
14	Yes	1/15	Yes	0.067	No
15	No	1/15	Yes	0.183	No

$$\text{err}_2 = \frac{\sum_{i=1}^{15} w_i I(y_i \neq G_2(x))}{\sum_{i=1}^{15} w_i}$$

$$\text{err}_2 = \frac{0.067+0.067+0.067+0.067+0.067}{1.469}$$

$$\text{err}_2 = 0.228$$

$$\alpha_2 = \log\left(\frac{1-\text{err}_2}{\text{err}_2}\right)$$

$$\alpha_2 = \log\left(\frac{1-0.228}{0.228}\right) = \log(3.385)$$

$$\alpha_2 = 1.219$$

Atualize
$w_3 \leftarrow w_2 \cdot \exp\{\alpha_2 \cdot \mathbf{I}(y_i \neq G_1(\mathbf{x}))\}$

Para os **erros**
$w_3 \leftarrow w_2 \cdot \exp\{1.219\} = w_2 \cdot 3.384$

Para os **acertos** $w_3 \leftarrow w_2 \cdot \exp\{0\} = w_2$

Churn

Insper

id	obs	w_1	pred_1	w_2	pred_2	w_3
1	No	1/15	Yes	0.183	No	0.183
2	No	1/15	No	0.067	No	0.067
3	Yes	1/15	Yes	0.067	No	0.227
4	No	1/15	No	0.067	No	0.067
5	Yes	1/15	Yes	0.067	No	0.227
6	Yes	1/15	Yes	0.067	No	0.227
7	No	1/15	Yes	0.183	No	0.183
8	No	1/15	No	0.067	No	0.067
9	Yes	1/15	Yes	0.067	No	0.227
10	No	1/15	No	0.067	No	0.067
11	No	1/15	Yes	0.183	No	0.183
12	No	1/15	No	0.067	No	0.067
13	No	1/15	No	0.067	No	0.067
14	Yes	1/15	Yes	0.067	No	0.227
15	No	1/15	Yes	0.183	No	0.183

Churn

Insper

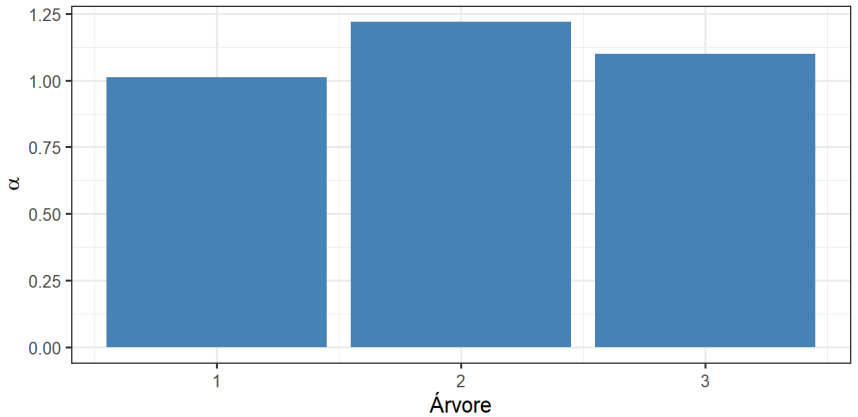
id	obs	w_1	pred_1	w_2	pred_2	w_3	pred_3
1	No	1/15	Yes	0.183	No	0.183	No
2	No	1/15	No	0.067	No	0.067	Yes
3	Yes	1/15	Yes	0.067	No	0.227	Yes
4	No	1/15	No	0.067	No	0.067	No
5	Yes	1/15	Yes	0.067	No	0.227	Yes
6	Yes	1/15	Yes	0.067	No	0.227	Yes
7	No	1/15	Yes	0.183	No	0.183	Yes
8	No	1/15	No	0.067	No	0.067	No
9	Yes	1/15	Yes	0.067	No	0.227	Yes
10	No	1/15	No	0.067	No	0.067	Yes
11	No	1/15	Yes	0.183	No	0.183	No
12	No	1/15	No	0.067	No	0.067	No
13	No	1/15	No	0.067	No	0.067	Yes
14	Yes	1/15	Yes	0.067	No	0.227	Yes
15	No	1/15	Yes	0.183	No	0.183	Yes

Pesos

A previsão final será dada por uma combinação das árvores ponderadas de acordo com o peso α.

$$G(\mathbf{x}) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right]$$

Por exemplo, para uma nova observação, os valores preditos de cada árvore foram dados por $G_1(x) = +1$, $G_2(x) = +1$ e $G_3(x) = -1$. Assim,



$$G(\mathbf{x}) = \text{sign}\Big[1.012 \cdot (+1) + 1.219 \cdot (+1) + 1.099 \cdot (-1)\Big] = \text{sign}\big[+1.131\big] = +1$$

Boosting for Regression Trees

Considere os dados de treinamento $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$

- Defina $\hat{f}(\mathbf{x}) = 0$ e $r_i = y_i$ para $i = 1, \ldots, n$
- Para b variando de 1 a B:
  - Ajuste uma árvore $\hat{f}^b$ com d divisões/splits ($d + 1$ nós terminais) utilizando os dados de treinamento $\{(\mathbf{x}_i, r_i)\}_{i=1}^n$
  - Atualize $\hat{f}$ adicionando uma versão encolhida da nova árvore: $\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \lambda \hat{f}^b(\mathbf{x})$
  - Atualize os resíduos, $r_i \leftarrow r_i - \lambda \hat{f}^b(\mathbf{x}_i)$
- Retorne a função de regressão, $\hat{f}(\mathbf{x}) = \sum_{b=1}^B \lambda \hat{f}^b(\mathbf{x})$

descrição apresentada no livro *The Elements of Statistical Learning*.

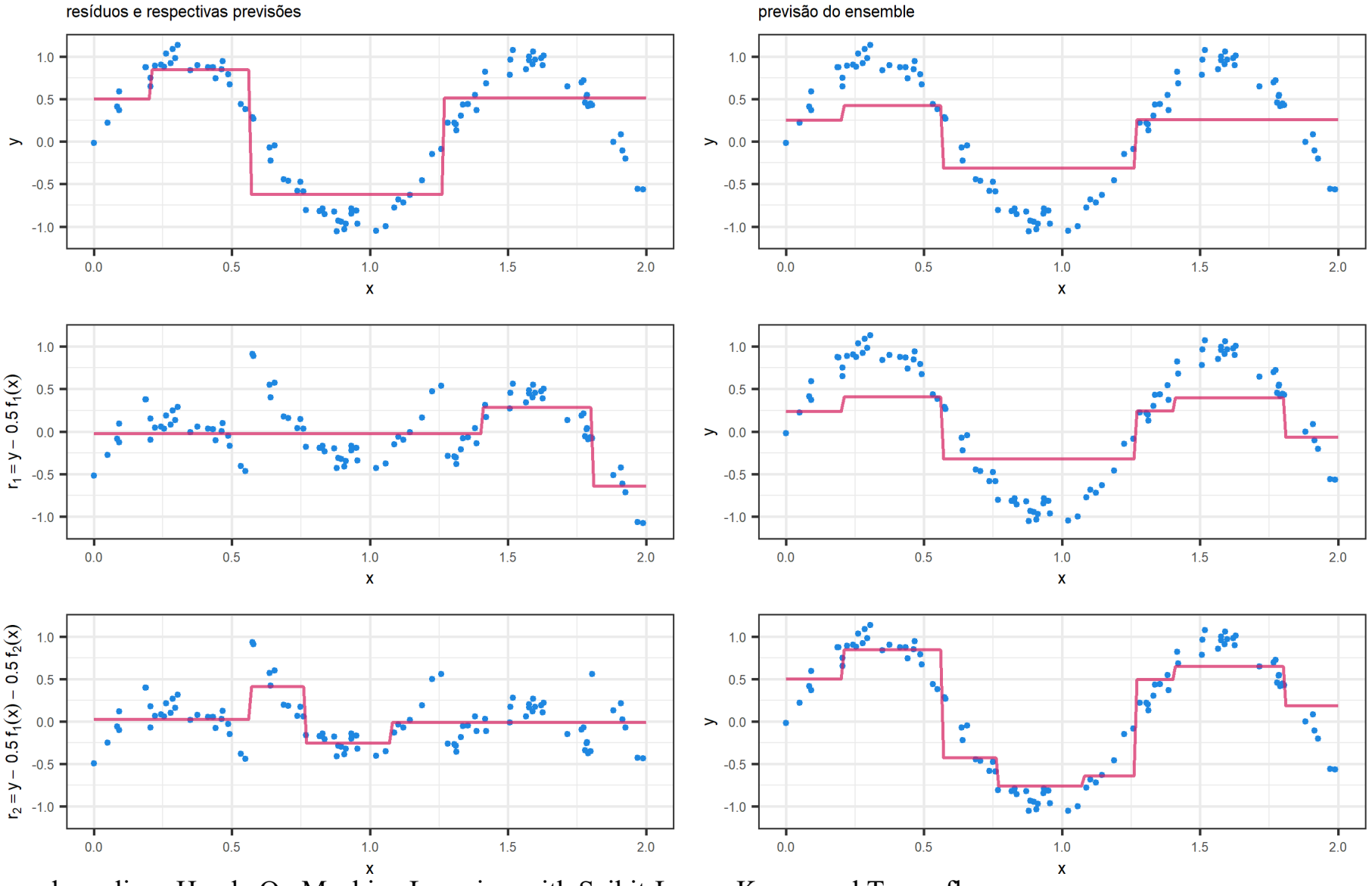


figura baseada no livro Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow

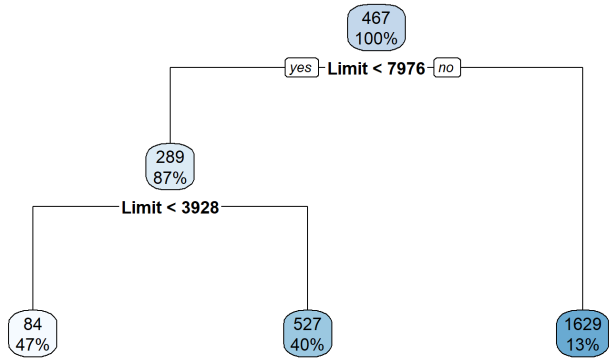
Credit

ID	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity
1	14.891	3606	283	2	34	11	Male	No	Yes	Caucasian
2	106.025	6645	483	3	82	15	Female	Yes	Yes	Asian
3	104.593	7075	514	4	71	11	Male	No	No	Asian
4	148.924	9504	681	3	36	11	Female	No	No	Asian
5	55.882	4897	357	2	68	16	Male	No	Yes	Caucasian
6	80.18	8047	569	4	77	10	Male	No	No	Caucasian
7	20.996	3388	259	2	37	12	Female	No	No	African American
8	71.408	7114	512	2	87	9	Male	No	No	Asian
9	15.125	3300	266	5	66	13	Female	No	No	Caucasian
10	71.061	6819	491	3	41	19	Female	Yes	Yes	African American

Credit

Começamos com $r_i = y_i$.

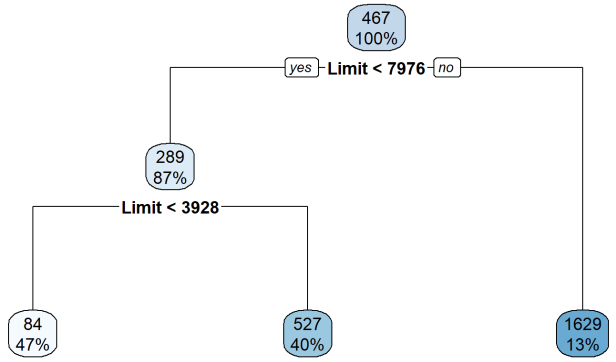
Balance
1999
0
1259
0
772
188
70
0
209
382
846
465
121
391
308



Credit

Começamos com $r_i = y_i$.

Balance	pred_1
1999	1629.0
0	84.0
1259	1629.0
0	84.0
772	527.3
188	84.0
70	84.0
0	84.0
209	84.0
382	527.3
846	527.3
465	527.3
121	84.0
391	527.3
308	527.3

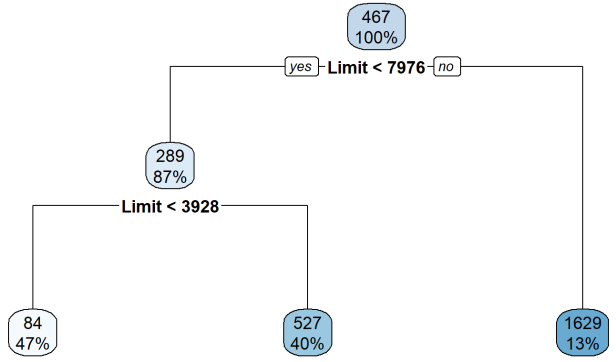


$$r_i \leftarrow r_i - \lambda \hat{f}^1(x)$$

Credit

Os resíduos são dados por $r_i - \lambda \hat{f}^b(x_i)$. Considere $\lambda = 0.1$.

Balance	pred_1	r_1
1999	1629.0	1836.1
0	84.0	-8.4
1259	1629.0	1096.1
0	84.0	-8.4
772	527.3	719.3
188	84.0	179.6
70	84.0	61.6
0	84.0	-8.4
209	84.0	200.6
382	527.3	329.3
846	527.3	793.3
465	527.3	412.3
121	84.0	112.6
391	527.3	338.3
308	527.3	255.3

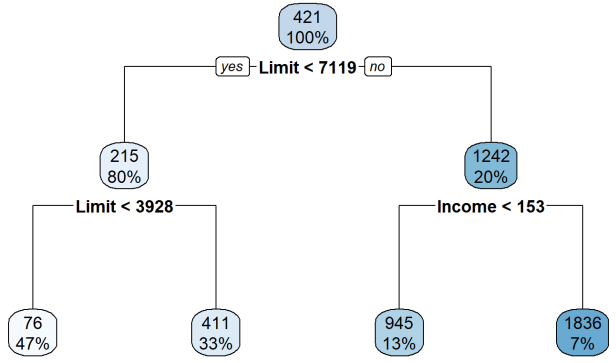


$$r_i \leftarrow r_i - \lambda \hat{f}^b(x)$$

Credit

Agora ajustamos uma nova árvore com a resposta dada pela coluna r_1

Balance	pred_1	r_1
1999	1629.0	1836.1
0	84.0	-8.4
1259	1629.0	1096.1
0	84.0	-8.4
772	527.3	719.3
188	84.0	179.6
70	84.0	61.6
0	84.0	-8.4
209	84.0	200.6
382	527.3	329.3
846	527.3	793.3
465	527.3	412.3
121	84.0	112.6
391	527.3	338.3
308	527.3	255.3

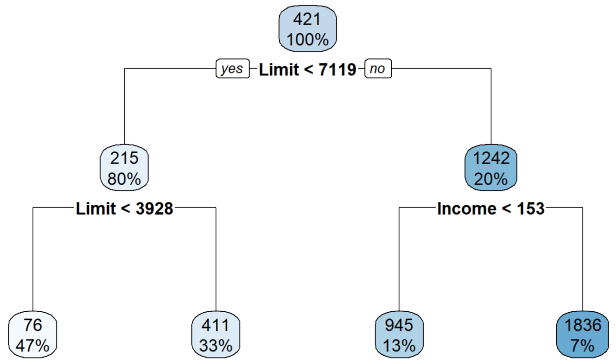


$$r_i \leftarrow r_i - \lambda \hat{f}^2(x)$$

Credit

Agora ajustamos uma nova árvore com a resposta dada pela coluna r_1

Balance	pred_1	r_1	pred_2	r_2
1999	1629.0	1836.1	1836.1	1652.5
0	84.0	-8.4	75.6	-16.0
1259	1629.0	1096.1	944.7	1001.6
0	84.0	-8.4	75.6	-16.0
772	527.3	719.3	410.9	678.2
188	84.0	179.6	75.6	172.0
70	84.0	61.6	75.6	54.0
0	84.0	-8.4	75.6	-16.0
209	84.0	200.6	75.6	193.0
382	527.3	329.3	410.9	288.2
846	527.3	793.3	944.7	698.8
465	527.3	412.3	410.9	371.2
121	84.0	112.6	75.6	105.0
391	527.3	338.3	410.9	297.2
308	527.3	255.3	410.9	214.2



$$r_i \leftarrow r_i - \lambda \hat{f}^2(x)$$

Credit

Após repetir o precesso B vezes. A previsão será dada por:

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

Ajuste de hiperparâmetros:

1. **Número de árvores B**: diferente de *bagging* e floresta aleatória, *boosting* pode sobreajustar se B for muito grande, embora o sobreajuste tenda a ocorrer vagarosamente (caso ocorra).

2. **Parâmetro de encolhimento λ:** esse parâmetro controla a taxa com a qual o método aprende. Os valores típicos são 0.01 e 0.001 e a escolha correta pode depender do problema. Valores muito pequenos de λ podem precisar de um número muito grande de árvores (B) para se alcançar uma boa performance.

3. **O número de divisões/*split*** em cada árvore: esse parâmetro controla a complexidade do método. Normalmente $d = 1$ funciona bem, nesse caso cada árvore é um *stump*, para uma árvore com apenas uma divisão/*split*

Gradient boosting

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha\text{th-quantile}\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Considere a primeira função de perda e que $f(x_i) = \gamma$.

$$L(y_i, \gamma) = \frac{1}{2}(y_i - \gamma)^2.$$

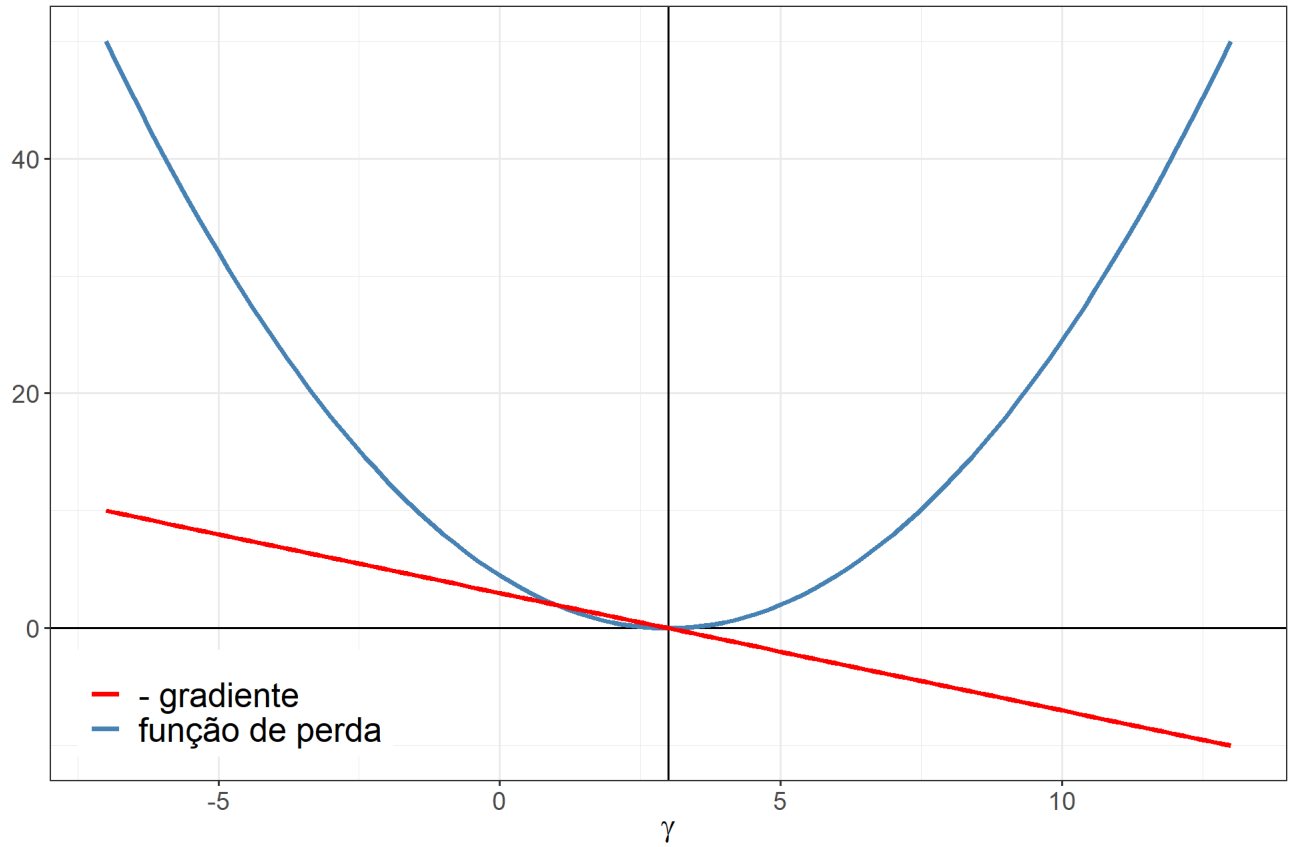
Assim,

$$-\left[\frac{\partial L(y_i, \gamma)}{\partial \gamma}\right] = -\left[\frac{2}{2}(y_i - \gamma)(-1)\right] = (y_i - \gamma)$$

tabela apresentada no livro *The Elements of Statistical Learning*.

Gradient boosting

Com $y_i = 3$ considere $L(y_i, \gamma) = \frac{1}{2}(y_i - \gamma)^2$ e $-\left[\frac{\partial L(y_i,\gamma)}{\partial \gamma}\right] = (y_i - \gamma)$.



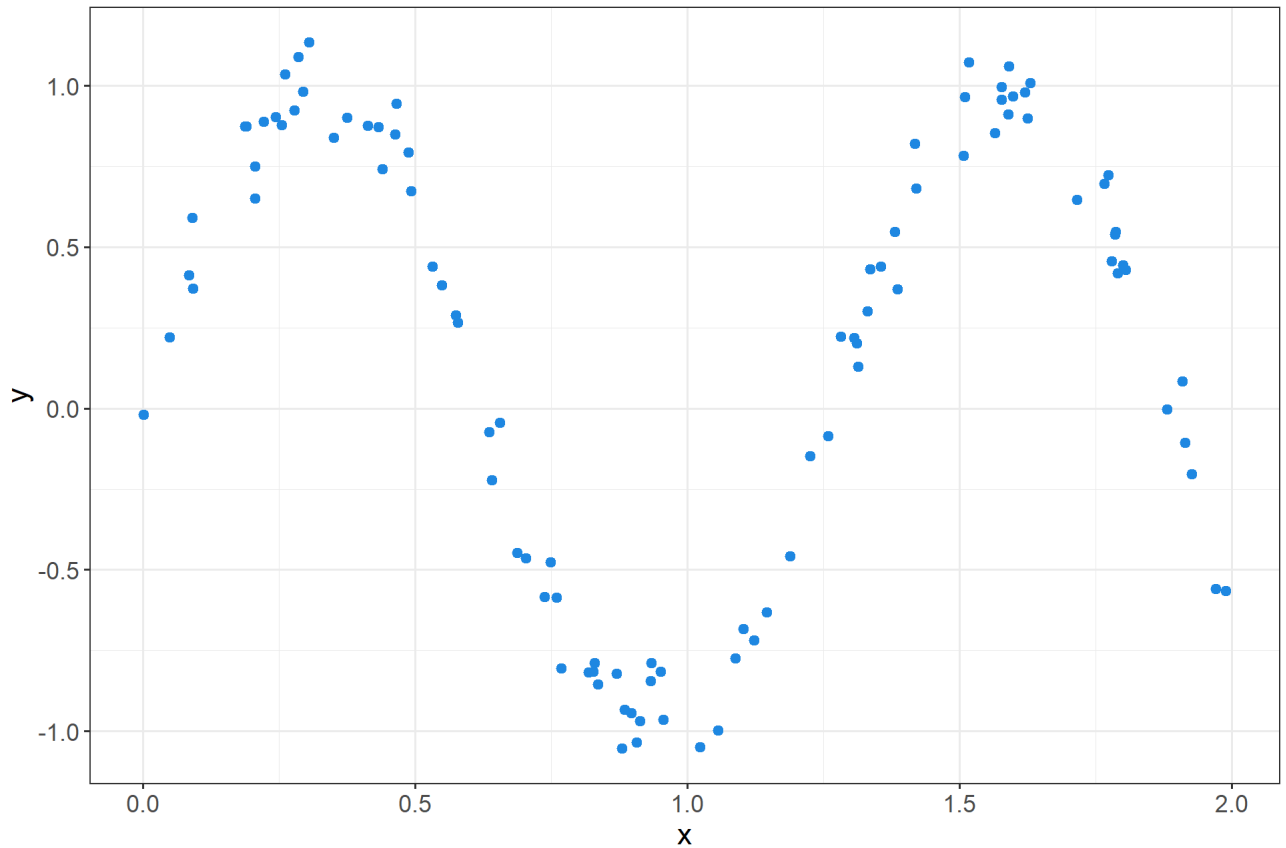
Gradient boosting

- Inicialize $f_0(\mathbf{x}) = \text{argmin} \sum_{i=1}^{N} L(y_i, \gamma)$
- Para m variando de 1 a M:
  - Para $i = 1, 2, \ldots, N$ calcule o gradiente residual utilizando $r_{im} = -\left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f=f_{m-1}}$
  - Ajuste o modelo árvore de regressão utilizando r_{im} para obter os nós terminais/folhas $R_{jm},\ j = 1, 2, \ldots, J_m$.
  - Para $j = 1, 2, \ldots, J_m$ calcule γ_{jm} que minimiza $\sum_{\mathbf{x_i} \in R_{jm}} L(y_i, f_{m-1}(\mathbf{x_i}) + \gamma_{jm})^2$
  - Atualize $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \lambda \sum_{j=1}^{J_m} \gamma_{jm} I(\mathbf{x} \in R_{jm})$
- Retorne $f(\mathbf{x}) = f_M(\mathbf{x})$.

descrição adaptado do livro *The Elements of Statistical Learning* e documentação do pacote gbm.

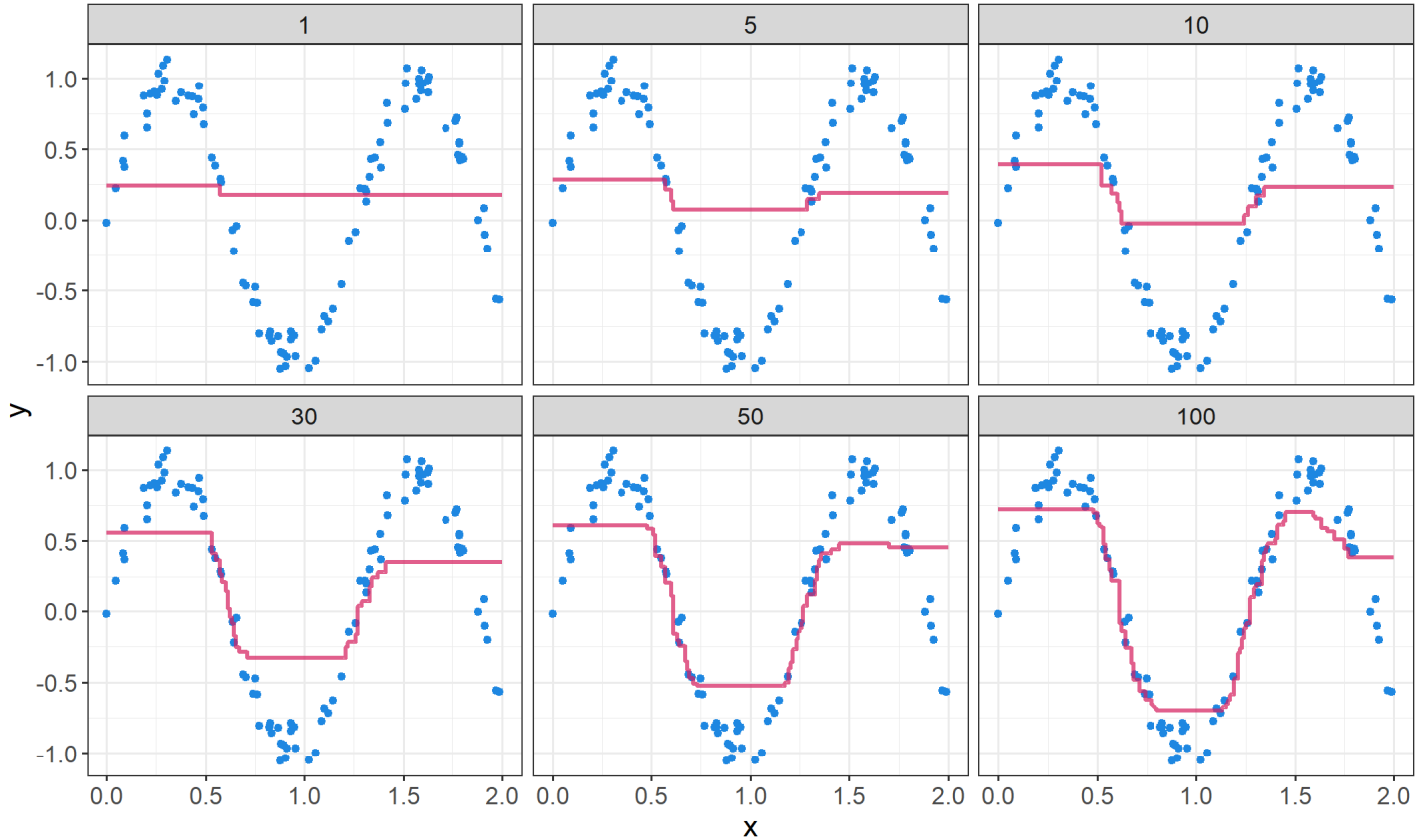
Simulação

Para desenvolver uma intuição sobre esses hiperparâmetros, vamos considerar algumas simulações. Considere os seguintes dados



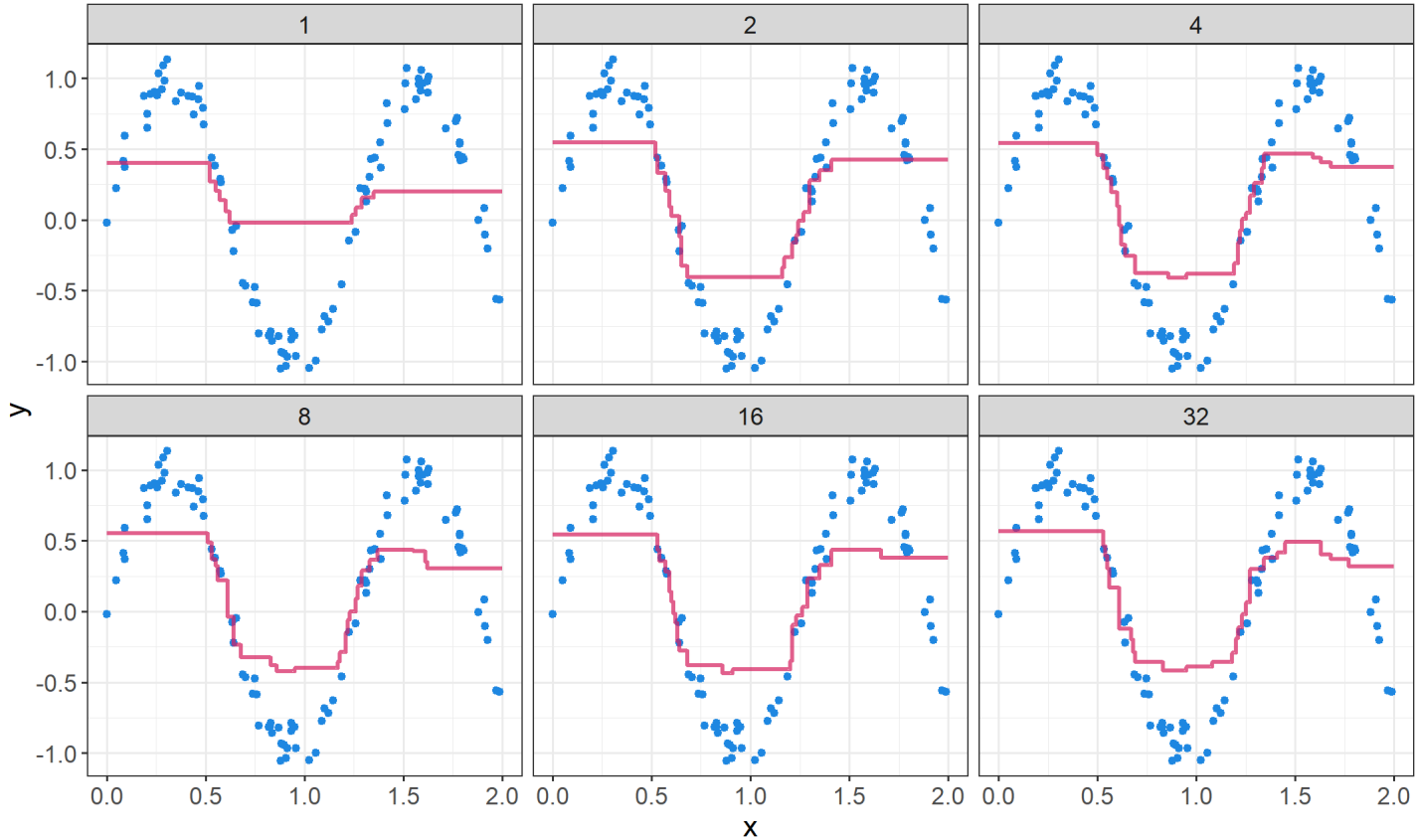
Simulação

Vamos começar variando o **número de árvores** (shrinkage = 0.1 e interaction.depth = 1).



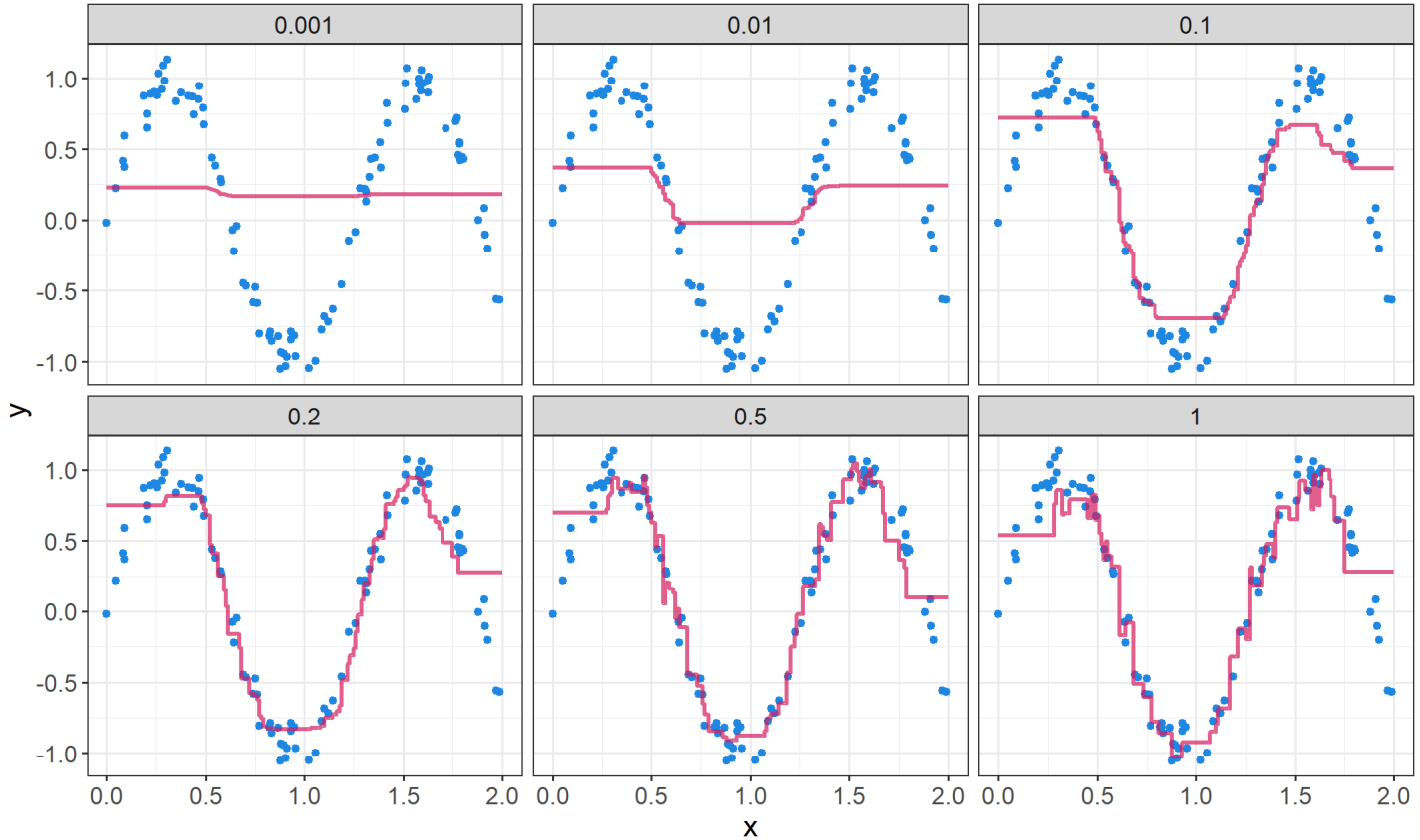
Simulação

Nesse cenário variamos **interaction.depth** (shrinkage = 0.1 e n.trees = 10).



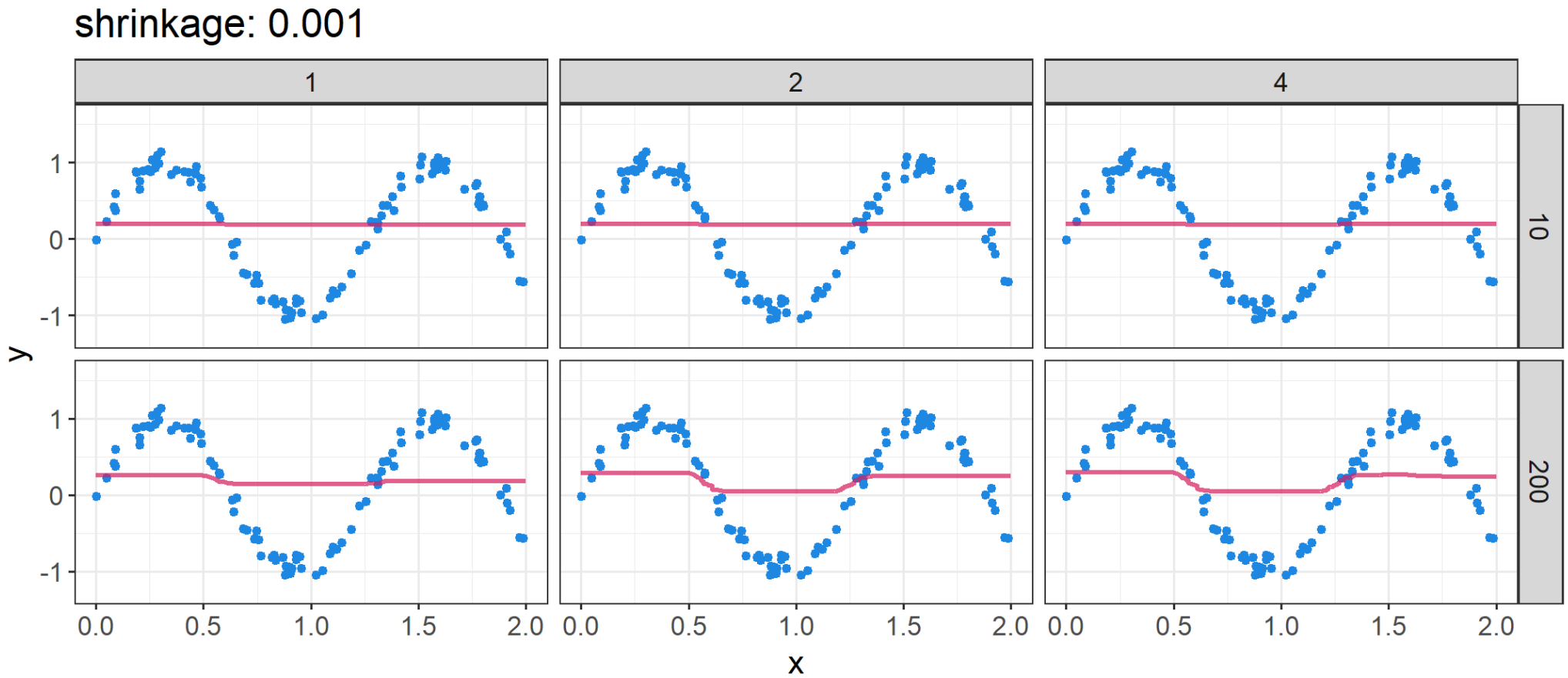
Simulação

Nesse cenário variamos **shrinkage** (interaction.depth = 1 e n.trees = 100).



Simulação

Nesse cenário vamos variar **interaction.depth** (1, 2 e 4) e **n.trees** (10, 200). Para cada uma dessas combinações, vamos variar o hiperparâmetro **shrinkage**.



Boston

```
library(MASS)
library(gbm)
library(rsample)

data(Boston)

set.seed(123)

split <- initial_split(Boston, prop = .8)

treinamento <- training(split)
teste <- testing(split)

(fit_bst <- gbm(medv ~ ., distribution = "gaussian",  n.trees = 5000,
                interaction.depth = 1, shrinkage = 0.1, data = treinamento))
```

```
## gbm(formula = medv ~ ., distribution = "gaussian", data = treinamento,
##     n.trees = 5000, interaction.depth = 1, shrinkage = 0.1)
## A gradient boosted model with gaussian loss function.
## 5000 iterations were performed.
## There were 13 predictors of which 13 had non-zero influence.
```

Importância

Para uma árvore de decisão T, temos a seguinte medida de relevância para cada preditora X_l:

$$I_l^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 \mathbf{I}(v(t) = l),$$

a soma é relativa a todos os $J-1$ nós internos da árvore. Para cada nó t, uma das variáveis $X_{v(t)}$ é utilizada para particionar a região em duas subregiões e em cada subregião considera uma constante para predição. A variável selecionada é a que apresenta o melhor aumento estimado $\hat{i}_t^2$ no *squared error risk* relativo ao ajuste de uma constante para a região inteira. A importância relativa quadrática de X_l é a soma de todos os melhoramentos quadráticos para todos os nós para os quais a variável foi escolhida como variável de divisão/*split*.

Importância Relativa

Essa medida de importância é facilmente generalizada para a utilização de diversas árvores da seguinte forma:

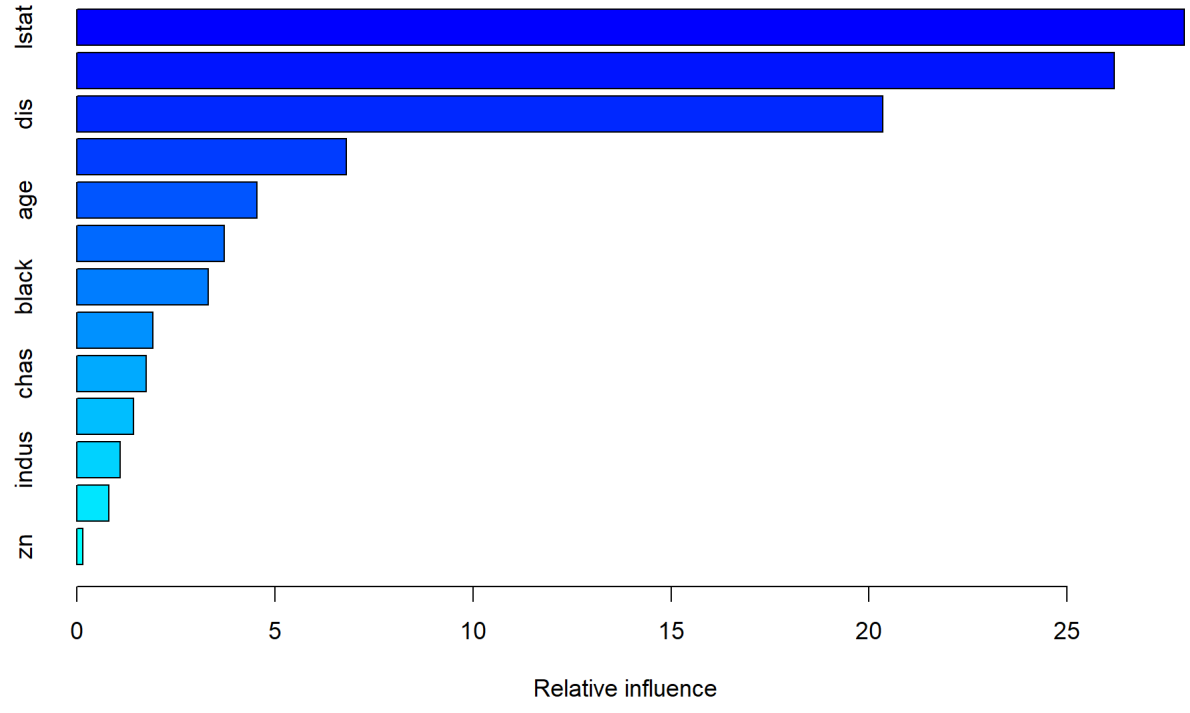
$$I_l^2 = \frac{1}{M} \sum_{m=1}^{M} I_l^2(T_m).$$

Note que, como essas medidas são relativas, é comum utilizar o maior valor como 100 e escalar os demais valores de acordo com essa quantidade.

Boston

Insper

```
summary(fit_bst)
```




```
##             var    rel.inf
## lstat     lstat 27.9562043
```

Average / Partial Dependence Plots

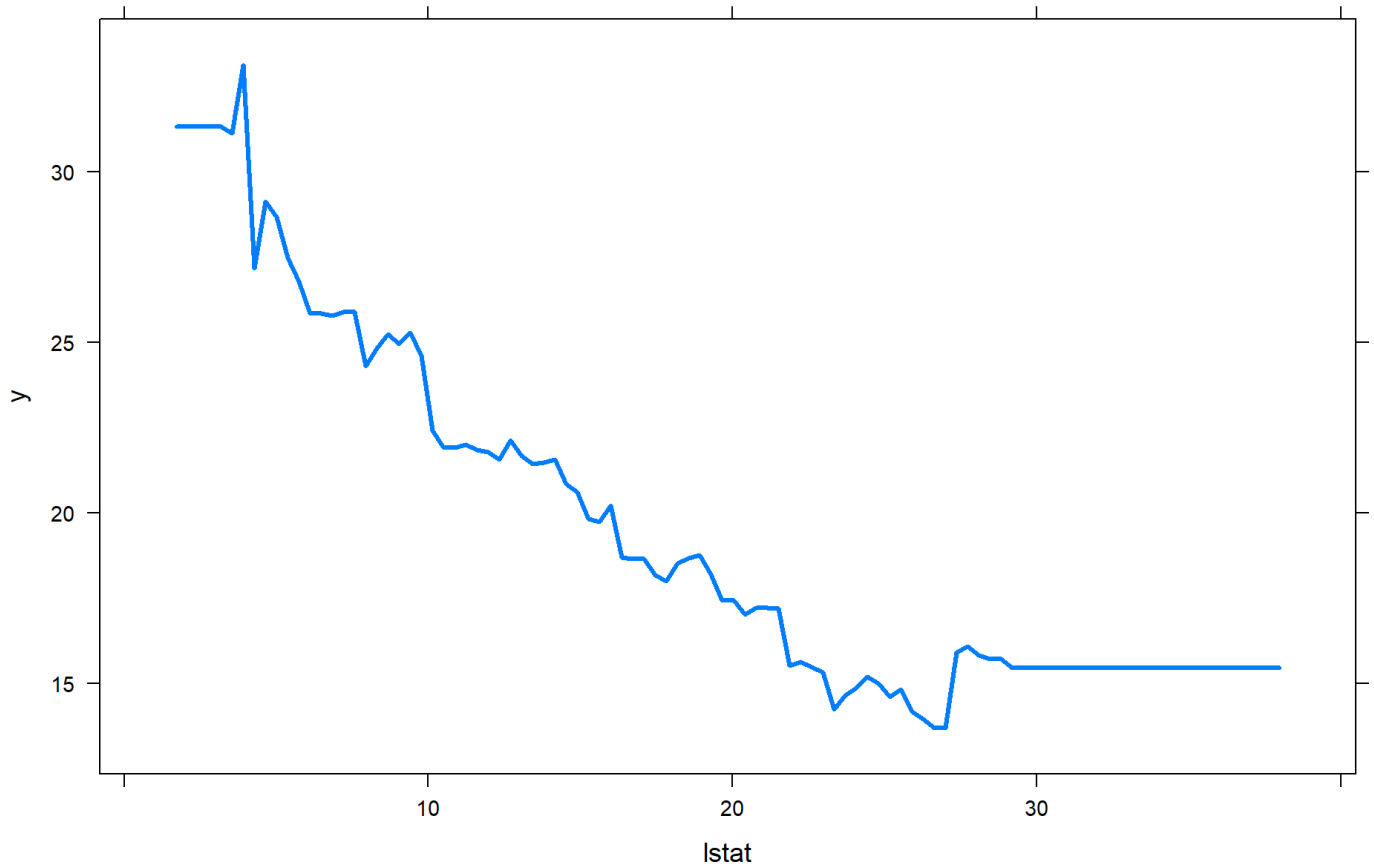
Funções de dependência parcial podem ser utilizadas para interpretar os resultados de alguns modelos de aprendizados tidos como *black box*. Essas funções podem ser estimadas por:

$$\bar{f}_S(X_S) = \frac{1}{N}\sum_{i=1}^{N} f(X_S, \mathbf{x}_{iC}),$$

em que $\{\mathbf{x}_{1C}, \ldots, \mathbf{x}_{NC}\}$ são os valores de X_C no conjunto de treinamento.

Boston

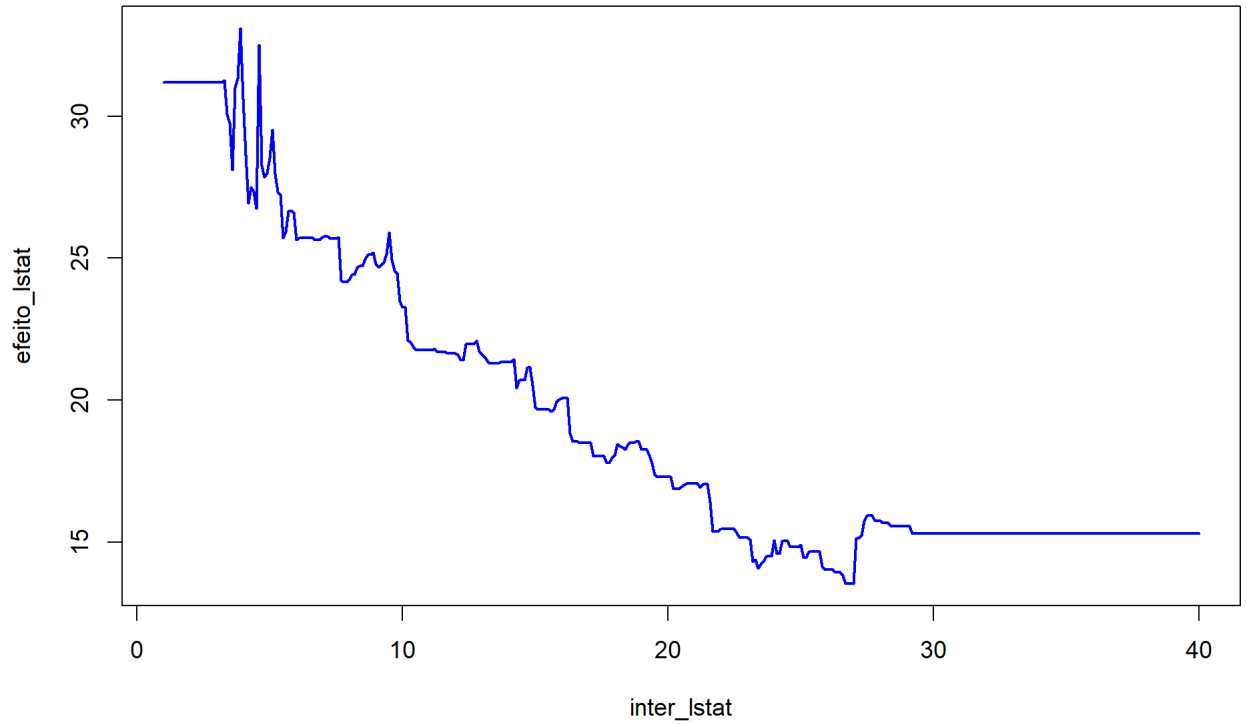
```
plot(fit_bst, i = "lstat", lwd = 3)
```



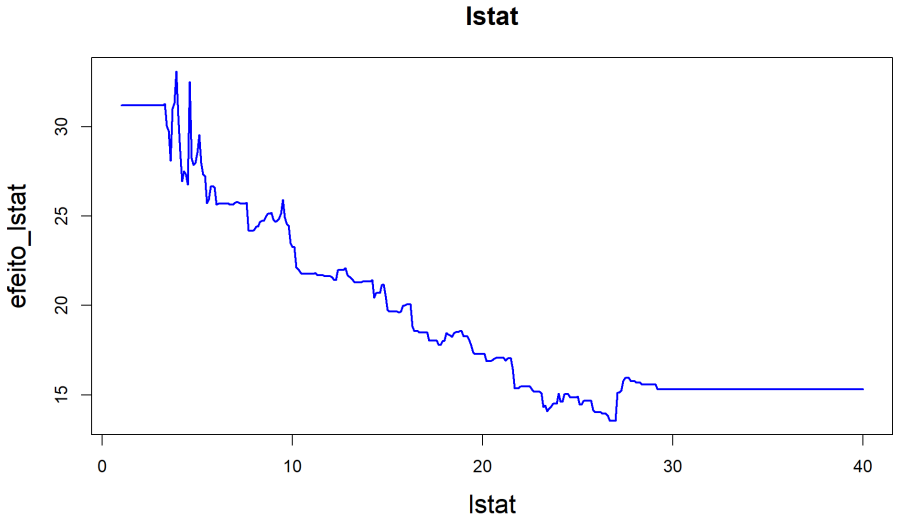
Boston

```
inter_lstat  <- seq(1, 40, 0.1)

efeito_lstat <- vector("numeric", length(inter_lstat))

tr2 <- treinamento

for (i in 1:length(inter_lstat)) {

  tr2$lstat <- inter_lstat[i]

  efeito_lstat[i] <- mean(predict(fit_bst, tr2, n.trees = 5000))

}

plot(inter_lstat, efeito_lstat, type = "l", col = "blue", lwd = 2)
```

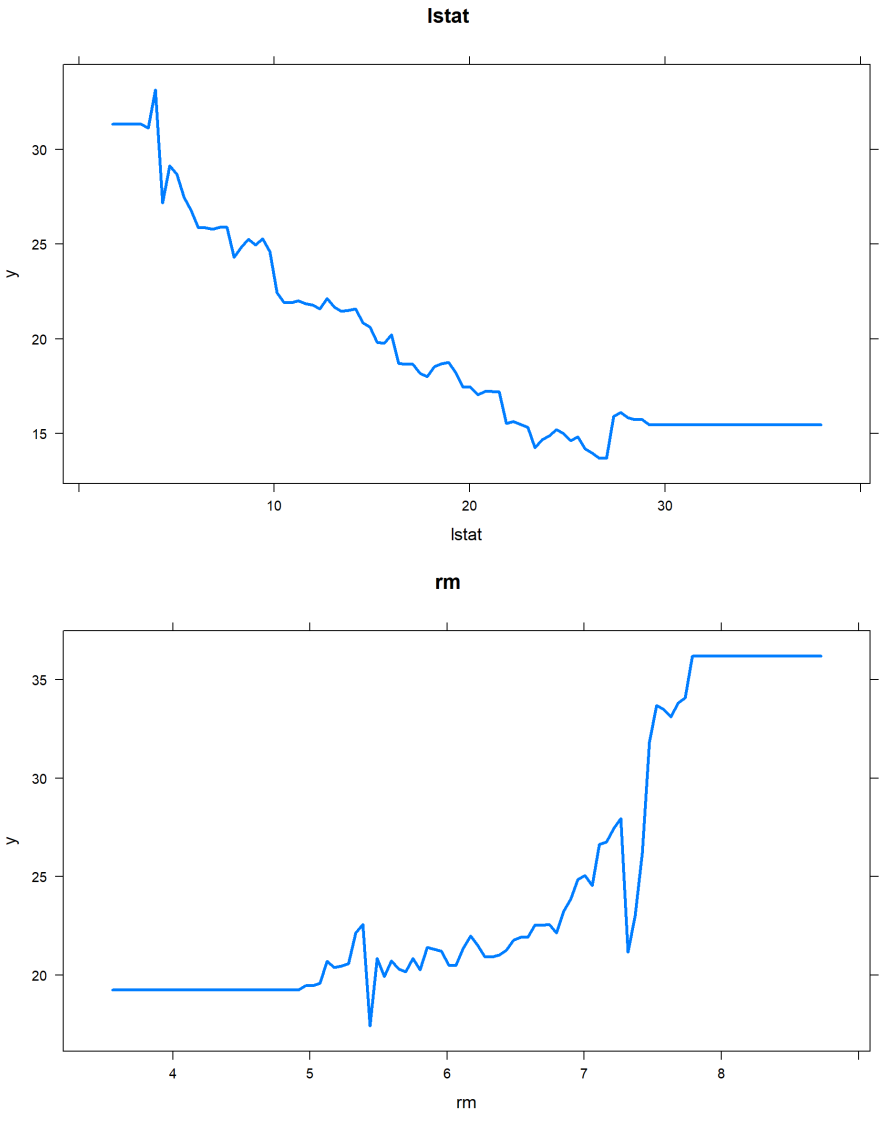
Boston



Programação

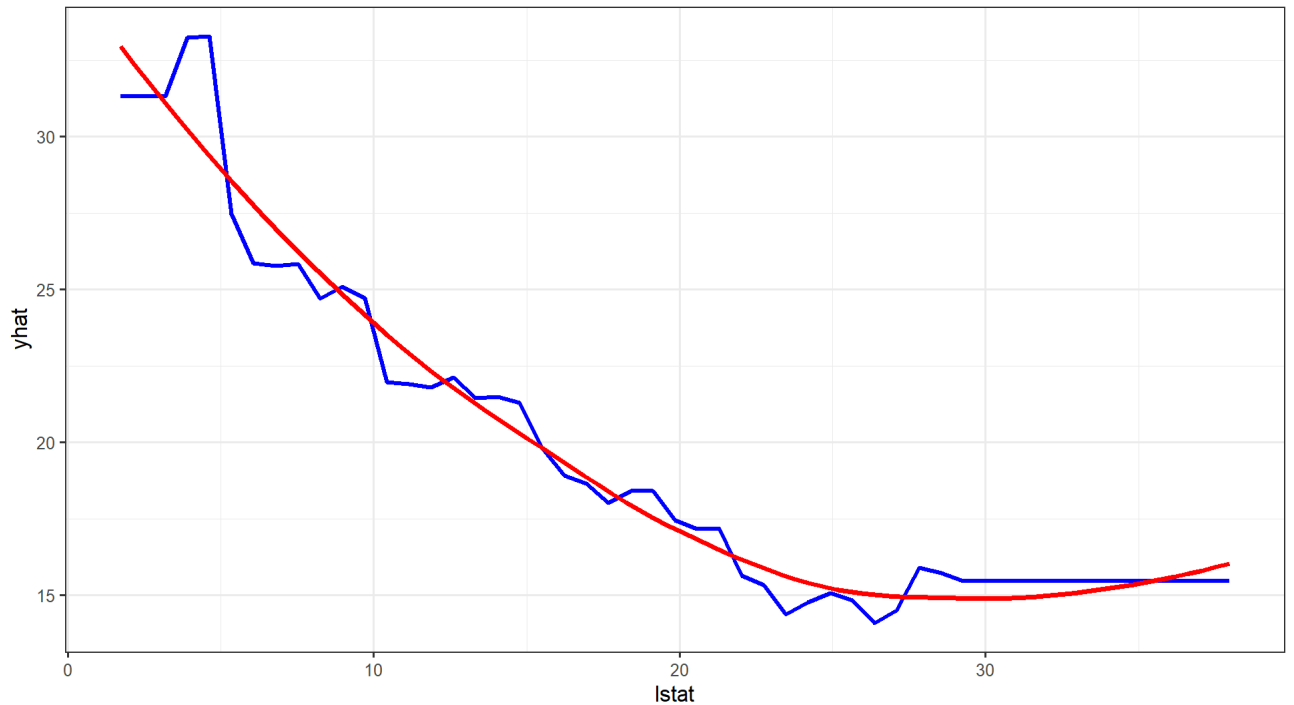


Função pronta



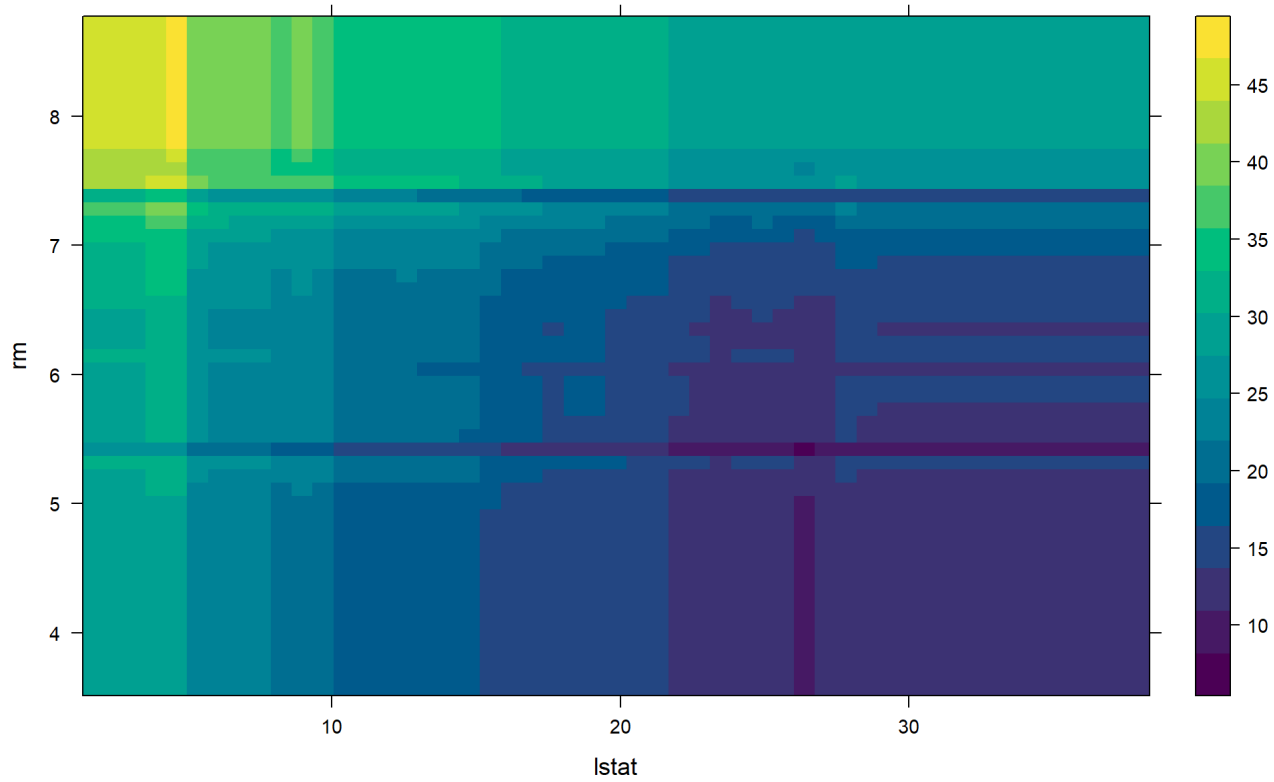
Boston

```
pdp::partial(fit_bst, pred.var = "lstat", n.trees = 5000) %>%
  ggplot(aes(lstat, yhat)) +
    geom_line(color = "blue", linewidth = 1) +
    geom_smooth(color = "red", linewidth = 1.2, se = FALSE) +
    theme_bw()
```



Boston

```
pdp::partial(fit_bst, pred.var = c("lstat", "rm"), n.trees = 5000) %>%
  pdp::plotPartial()
```



pacote *pdp*.

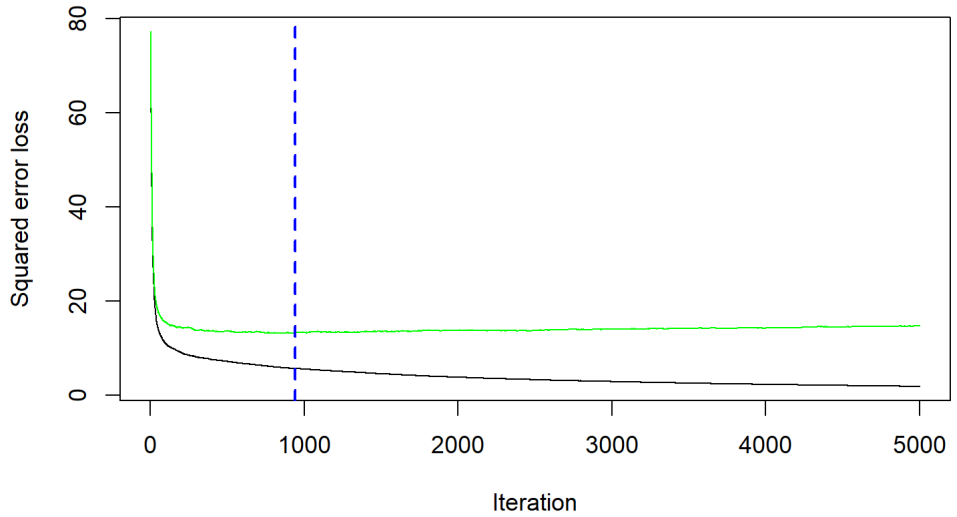
Boston

É possível estimar o número ótimo de iterações do algoritmo

```
boston_cv <- gbm(medv ~ ., data = treinamento, cv.folds = 5, n.trees = 5000,
                 interaction.depth = 1, shrinkage = 0.1)
```

```
## Distribution not specified, assuming gaussian ...
```

```
gbm.perf(boston_cv, method = "cv")
```



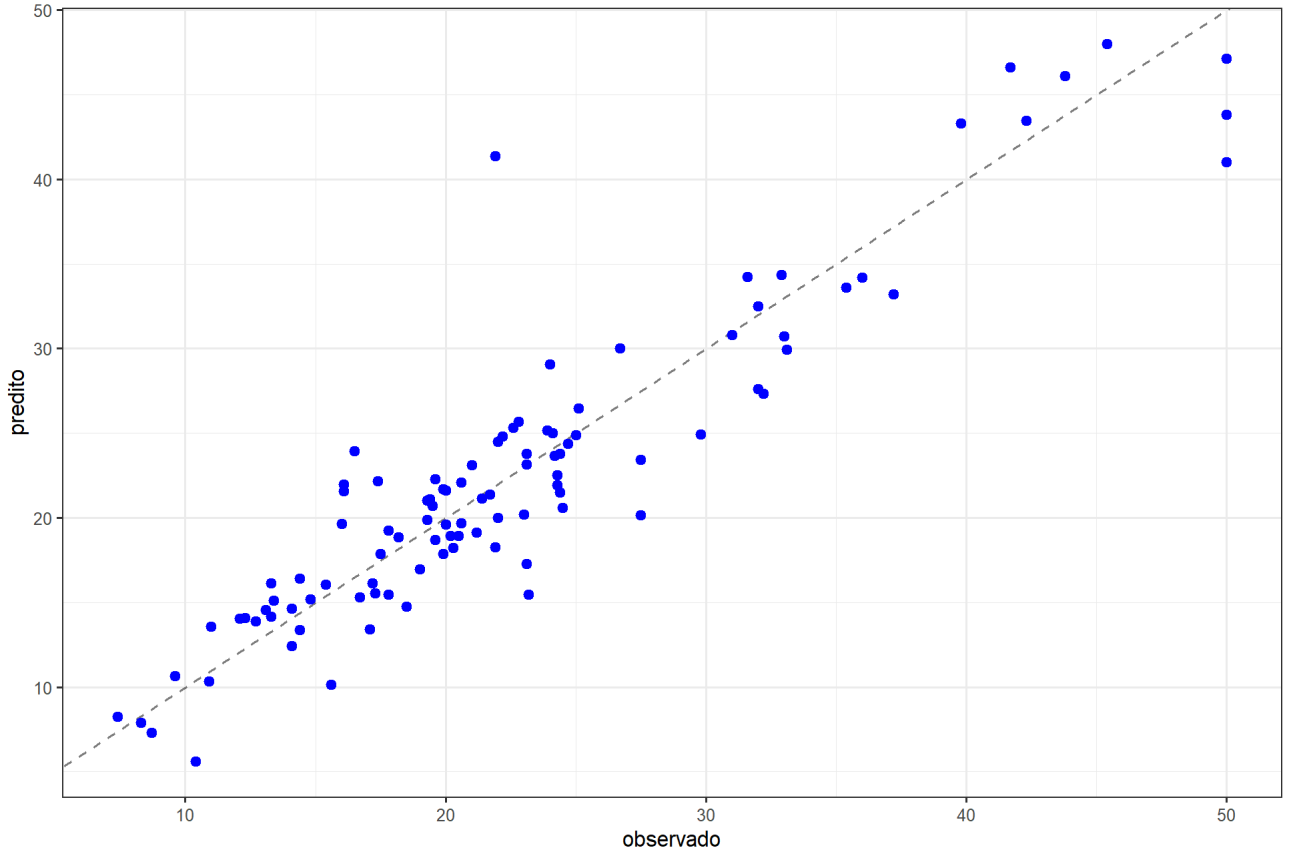
Boston

Avaliação da previsão dos dados Boston.

```
predito_bst <- predict(fit_bst, newdata = teste, n.trees = 766)

(erro_bst <- mean((teste$medv - predito_bst)^2))

resultados <- tibble(observado = teste$medv,
                     predito = predito_bst)

ggplot(resultados, aes(observado, predito)) +
  geom_abline(intercept = 0, slope = 1, color = "black", alpha = .5, linetype = "dashed") +
  geom_point(color = "blue", size = 2) +
  theme_bw()
```

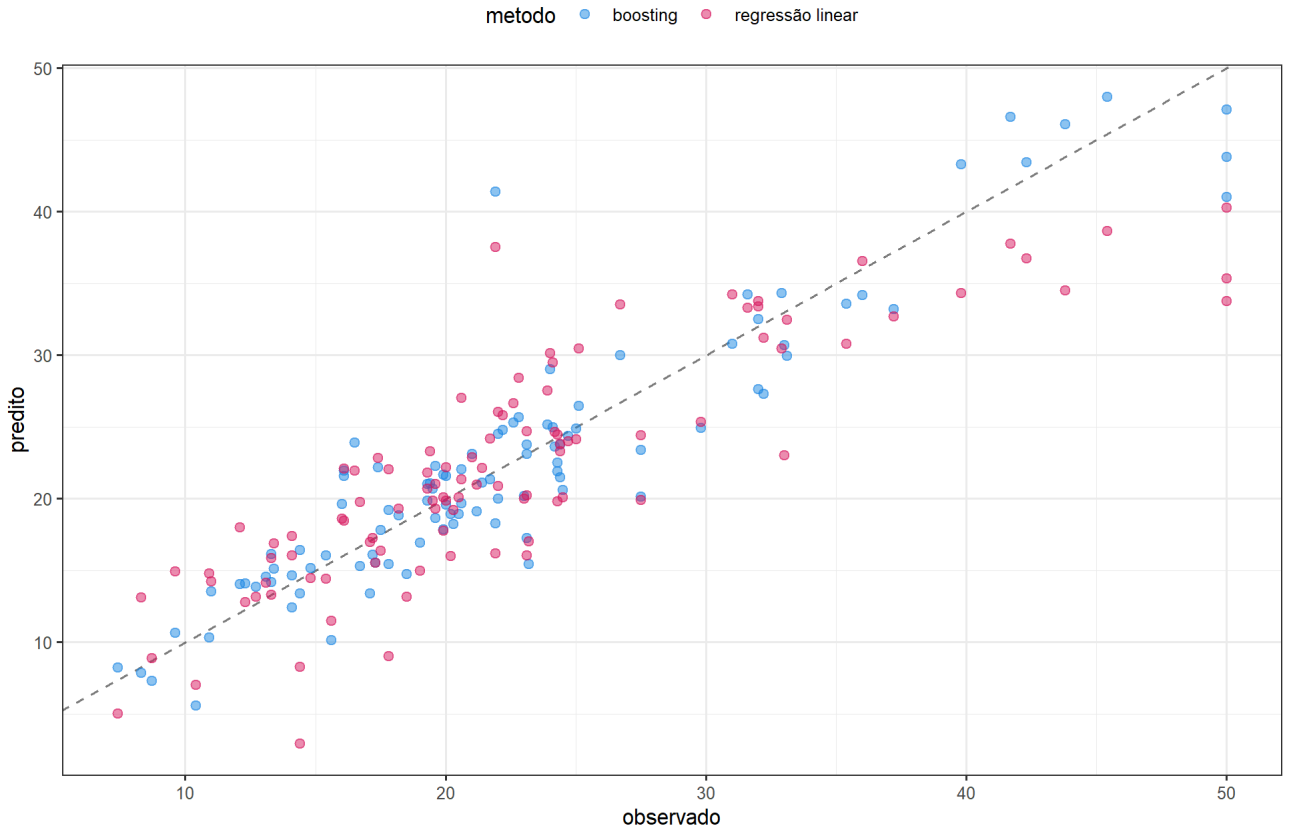
Boston

```
## [1] 12.79732
```



Boston

Comparação com modelo linear ($EQM_{bst} = 12.8$) e ($EQM_{LM} = 23.68$).



Boosting

Atualmente contamos com diversas implementações e versões desse método. Além das citadas anteriormente, podemos considerar

- XGBoost
- LightGBM
- CatBoost

A seguir faremos uma aplicação com a biblioteca xgboost.

XGBoost

XGBoost (*eXtreme Gradient Boosting*) é um pacote otimizado para *boosting* com algumas alterações e novas funcionalidades. Com esse pacote é possível considerar processamento em paralelo e atualizar um modelo com base na última iteração caso um novo conjunto de dados esteja disponível.

É possível ajustar diversos hiperparametros:

- **eta**: control the learning rate. Used to prevent overfitting by making the boosting process more conservative. Lower value for eta implies larger value for nrounds: low eta value means model more robust to overfitting but slower to compute. Default: 0.3

- **subsample**: subsample ratio of the training instance. Setting it to 0.5 means that xgboost randomly collected half of the data instances to grow trees and this will prevent overfitting. It makes computation shorter (because less data to analyse). It is advised to use this parameter with eta and increase nrounds. Default: 1

- **colsample_bytree**: subsample ratio of columns when constructing each tree. Default: 1

XGBoost

Insper

```
library(xgboost)
library(rsample)
library(forcats)

dados <- AmesHousing::make_ames() %>%
  mutate_if(is.factor, ~ fct_lump(.x, prop = .1))

X <- model.matrix(Sale_Price ~ . - 1, dados)

dados <- data.frame(X) %>%
  mutate(Sale_Price = dados$Sale_Price)

set.seed(321)

splits <- initial_split(dados, prop = 0.7)

treino <- training(splits)
teste <- testing(splits)
```

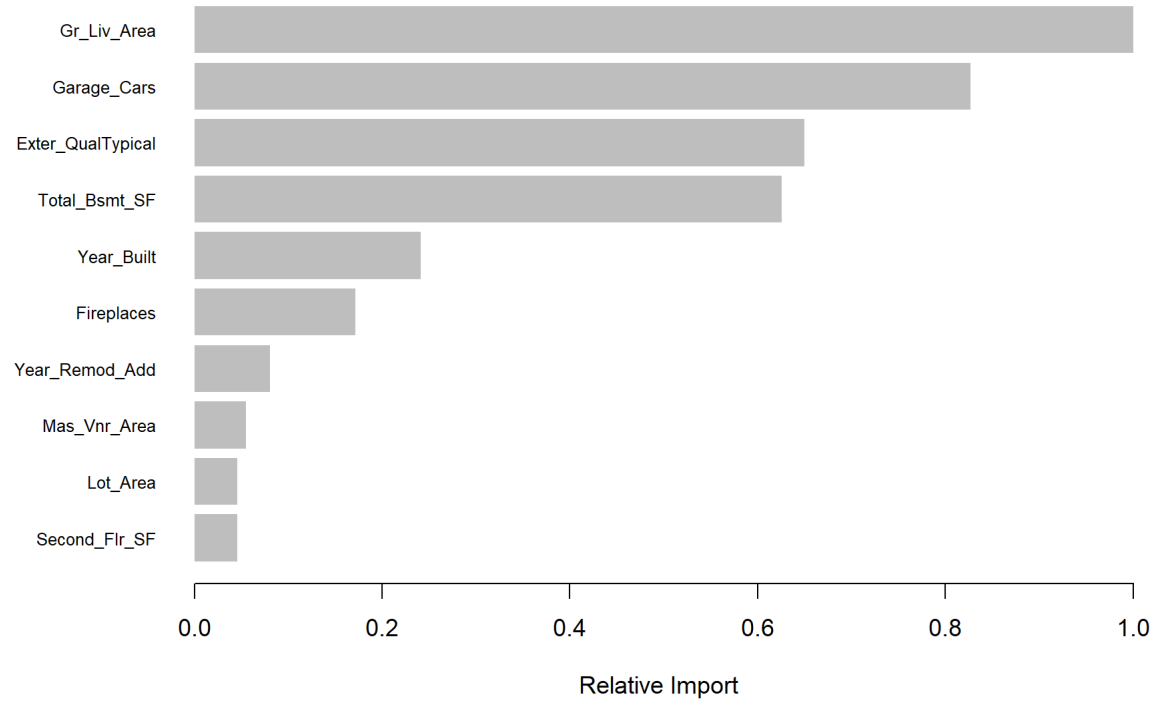
XGBoost

```
d_treino <- xgb.DMatrix(label = treino$Sale_Price,
                        data = as.matrix(select(treino, -Sale_Price)))

d_teste  <- xgb.DMatrix(label = teste$Sale_Price,
                        data = as.matrix(select(teste, -Sale_Price)))

(fit_xgb <- xgb.train(data = d_treino, nrounds = 100, max_depth = 1, eta = 0.3,
                      nthread = 3, verbose = FALSE, objective = "reg:squarederror"))
```

```
## ##### xgb.Booster
## raw: 78 Kb
## call:
##   xgb.train(data = d_treino, nrounds = 100, verbose = FALSE, max_depth = 1,
##     eta = 0.3, nthread = 3, objective = "reg:squarederror")
## params (as set within xgb.train):
##   max_depth = "1", eta = "0.3", nthread = "3", objective = "reg:squarederror", validate_parameters = "TRUE"
## xgb.attributes:
##   niter
## # of features: 117
## niter: 100
## nfeatures : 117
```

XGBoost

```
importancia <- xgb.importance(model = fit_xgb)

xgb.plot.importance(importancia, rel_to_first = TRUE, top_n = 10, xlab = "Relative Import")
```



XGBoost

Para avaliar o desempenho preditivo, considere:

```
pred_xgb <- predict(fit_xgb, d_teste)

sqrt(mean((pred_xgb - teste$Sale_Price)^2))
```

```
## [1] 28157.91
```

XGBoost

```
ajusta_bst <- function(splits, eta, nrounds, max_depth) {

  tr <- training(splits)
  teste <- testing(splits)

  d_treino <- xgb.DMatrix(label = tr$Sale_Price,
                          data = as.matrix(select(tr, -Sale_Price)))

  d_teste  <- xgb.DMatrix(label = teste$Sale_Price,
                          data = as.matrix(select(teste, -Sale_Price)))

  fit <- xgb.train(data = d_treino, nrounds = nrounds, max_depth = max_depth, eta = eta,
                   nthread = 3, verbose = FALSE, objective = "reg:squarederror")

  eqm <- mean((teste$Sale_Price - predict(fit, as.matrix(select(teste, -Sale_Price))))^2)

  return(sqrt(eqm))

}
```

XGBoost

```
set.seed(123)

hiperparametros <- crossing(eta = c(.01, .1),
                            nrounds = c(250, 750),
                            max_depth = c(1, 4))

resultados <- rsample::vfold_cv(treino, 5) %>%
  crossing(hiperparametros) %>%
  mutate(reqm = pmap_dbl(list(splits, eta, nrounds, max_depth), ajusta_bst))

resultados %>%
  group_by(eta, nrounds, max_depth) %>%
  summarise(reqm = mean(reqm)) %>%
  arrange(reqm)
```

XGBoost

```
## # A tibble: 8 × 4
## # Groups:   eta, nrounds [4]
##     eta nrounds max_depth   reqm
##   <dbl>   <dbl>     <dbl>  <dbl>
## 1  0.1      750         4 23875.
## 2  0.1      250         4 23960.
## 3  0.01     750         4 25286.
## 4  0.1      750         1 27581.
## 5  0.1      250         1 29354.
## 6  0.01     250         4 33009.
## 7  0.01     750         1 35378.
## 8  0.01     250         1 48415.
```

XGBoost

```
fit_xgb <- xgb.train(data = d_treino, nrounds = 750, max_depth = 4, eta = 0.1,
                     nthread = 3, verbose = FALSE, objective = "reg:squarederror")

pred_xgb <- predict(fit_xgb, d_teste)

sqrt(mean((teste$Sale_Price - pred_xgb)^2))
```

```
## [1] 23275.71
```

```
fit_lm <- lm(Sale_Price ~ ., treino)

pred_lm <- predict(fit_lm, teste)

sqrt(mean((teste$Sale_Price - pred_lm)^2))
```

```
## [1] 32010.77
```

Obrigado!

tiagoms.com

tiagomendonca

tiagoms1@insper.edu.br