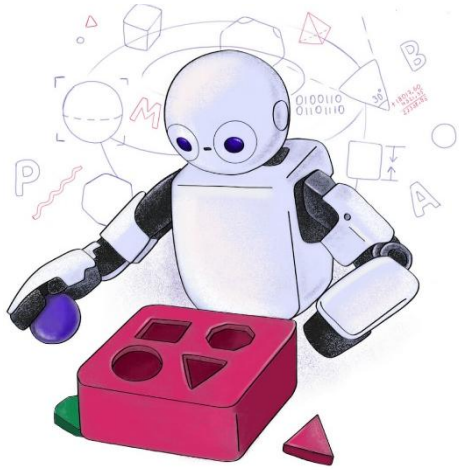
TP558 - Tópicos avançados em Machine Learning:

eXtreme Gradient Boosting (XGBoost)



Inatel

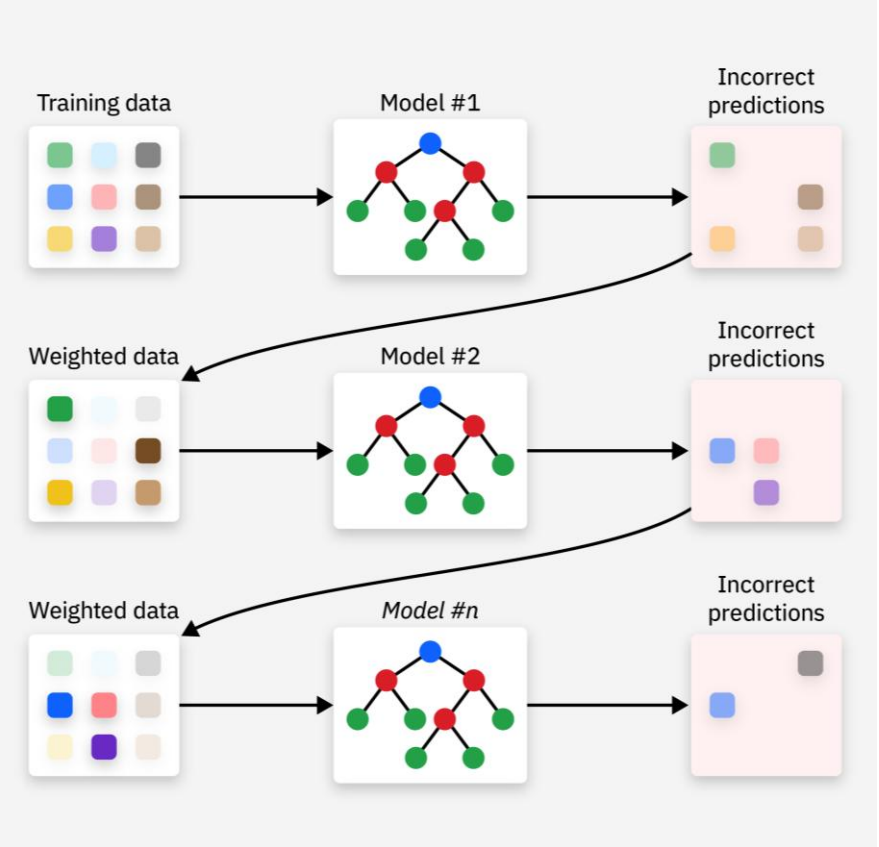
Alessandra Carolina Domiciano
alessandra.carolina@mtel.inatel.br

Introdução

- ***eXtreme Gradient Boosting*** é um algoritmo de aprendizado de máquina **supervisionado** desenvolvido por **Tianqi Chen** e **Carlos Guestrin** em 2016, amplamente utilizado em tarefas de **regressão** e **classificação**.
- Trata-se de uma implementação otimizada do ***Gradient Boosting Machine (GBM)***, projetada para ser mais rápida, eficiente e escalável, especialmente em cenários com **dados de alta dimensionalidade** e **grandes volumes**.
- Ele emprega a técnica de ***ensemble learning***, combinando múltiplas **árvores de decisão**.
- Tronou-se muito popular em competições de ciência de dados, como as promovidas pelo Kaggle.

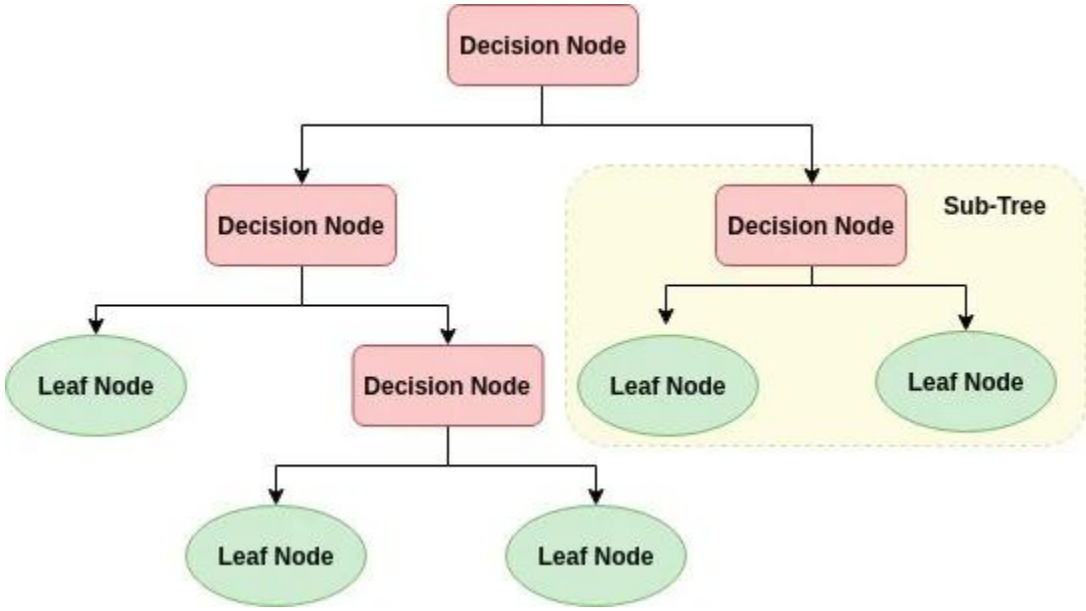
Ensemble learning

- O XGBoost implementa ***ensemble learning*** (aprendizado em conjunto) do tipo ***boosting***.
- Essa técnica combina **sequencialmente** vários modelos base (fracos) para formar um modelo final com alto desempenho.
- O objetivo é minimizar uma função de perda de forma iterativa. Assim, cada novo modelo visa **corrigir os erros** do modelo anterior.
- Através de ensemble, é possível:
  - Reduzir viés (erro sistemático).
  - Reduzir variância (sensibilidade ao ruído).
  - Melhorar a generalização do modelo.



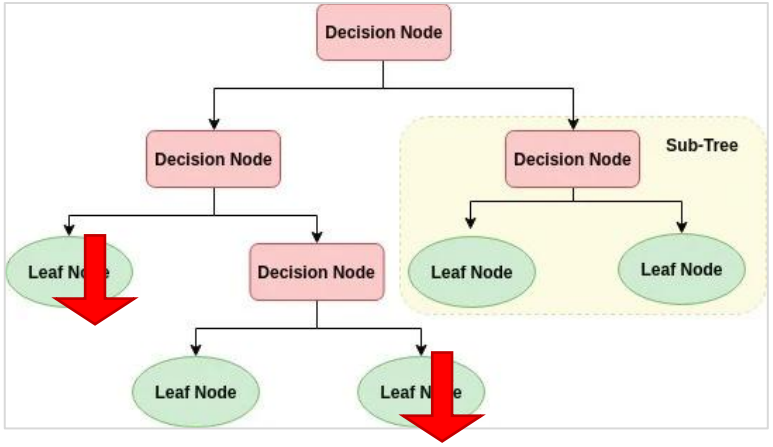
Árvores de decisão

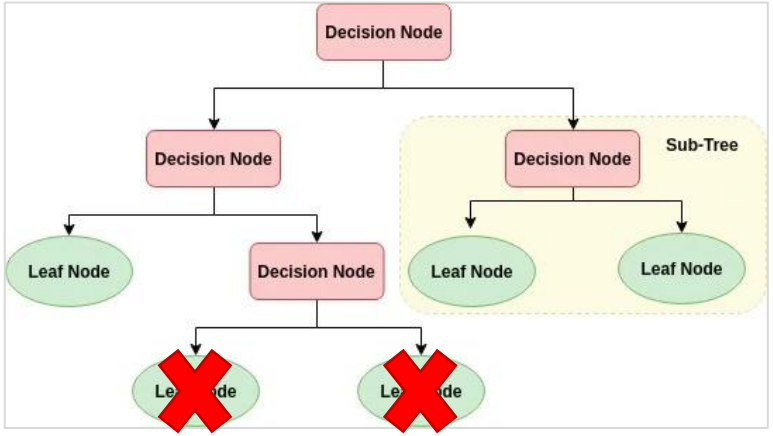
- O XGBoost utiliza árvores de decisão do tipo **CART (Classification and Regression Trees)**.
- Consiste em um tipo específico de árvore em que cada nó interno sempre se divide em **dois ramos**.
- São chamadas de **árvores de decisão binárias**.



Regularização

- A regularização é uma técnica usada para **evitar o sobreajuste** em um modelo, através da **penalização de pesos e de complexidade**.
- Penalização de pesos:
  - L1 Regularization (Lasso): Penaliza a soma dos valores **absolutos** dos pesos, reduzindo-os e podendo levar a **coeficientes iguais a zero**.
  - L2 Regularization (Ridge): Penaliza a soma dos **quadrados** dos pesos, reduzindo seus valores.
- Penalização de complexidade:
  - Poda de folhas com baixo ganho.





Visão geral da matemática do XGBoost

- O XGBoost otimiza uma **função objetivo** composta por **erro de previsão** e **termo de regularização**.

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Onde:

- $l(y_i, \hat{y}_i)$ é a função de perda (ex: MSE, logloss)
- $\Omega(f_k)$ é o termo de regularização do modelo

Visão geral da matemática do XGBoost

- Por se tratar de um algoritmo de **boosting de árvores de decisão**, a predição do modelo é feita de forma aditiva, e essa função pode ser expressa da seguinte forma:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Onde:

- $\hat{y}_i^{(t)}$ é a predição atual para a amostra x_i na iteração t
- $\hat{y}_i^{(t-1)}$ é a predição acumulada até a iteração anterior
- f_t é uma função que representa a árvore de decisão aprendida da iteração t, e retorna o peso da folha à qual a amostra x_i é associada.

- Substituindo na função objetivo: $$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{k=1}^{K} \Omega(f_k)$$

Visão geral da matemática do XGBoost

- Algumas funções de erro são menos “amigáveis” que outras, o que pode tornar a função objetivo muito complexa ao substituí-las diretamente. Para contornar essa dificuldade, aplica-se uma **expansão de Taylor de 2ª ordem** à função de perda, resultando na seguinte função objetivo:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i {f_t}^2(x_i)] + \sum_{k=1}^{K} \Omega(f_k)$$

Onde:

- g_i é o gradiente da função de perda com relação à predição anterior ${\hat{y}_i}^{(t-1)}$ (1ª derivada)
- h_i é o hessiano da função de perda com relação à predição anterior ${\hat{y}_i}^{(t-1)}$ (2ª derivada)

- Com isso, a função objetivo final passa a depender **apenas dos valores de g_i e de h_i**, e não mais da função de perda completa. Dessa forma, o XGBoost consegue lidar com diferentes funções de erro utilizando o mesmo método de otimização.

Visão geral da matemática do XGBoost

- Antes de apresentar o termo de regularização, é necessário reescrever a função f_t, que representa a árvore de decisão, em um formato mais específico, destacando os valores atribuídos às folhas.

$$f_t(x) = w_{q(x)}$$

Onde:

- w é o vetor de pesos das folhas
- q é uma função que associa cada amostra à folha correspondente

- O termo de regularização utilizado no XGBoost tem a seguinte expressão:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {w_j}^2$$

- Onde:
- T é o número de folhas na árvore
- γ é o parâmetro de regularização para poda de folhas
- w_j é o peso da folha j
- λ é o parâmetro de regularização para os pesos (L2)

Visão geral da matemática do XGBoost

- Inserindo o termo de regularização na função objetivo e reformulando-a em função dos pesos das folhas, temos:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i {f_t}^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {w_j}^2$$

- Simplificando a expressão e usando as definições $G_j = \sum_{i \in I_j} g_i$ e $H_j = \sum_{i \in I_j} h_i$:

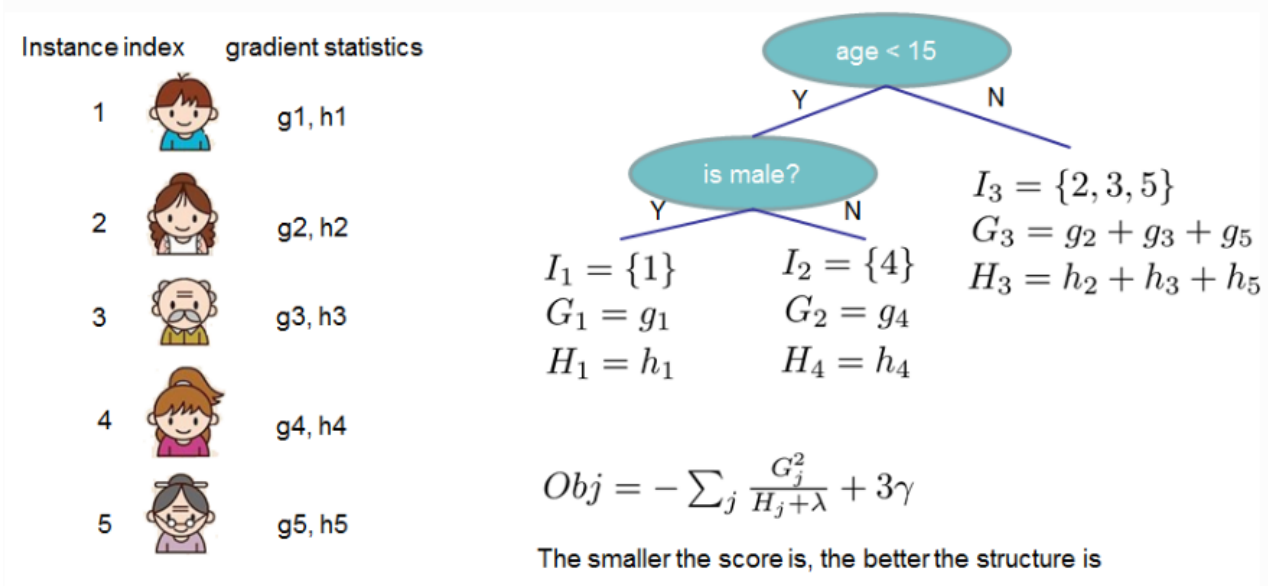
$$obj^{(t)} = \sum_{j=1}^{T} [G_j w_j + \frac{1}{2}(H_j + \lambda){w_j}^2] + \gamma T$$

Visão geral da matemática do XGBoost

- Como o objetivo é minimizar a função objetivo, devemos determinar os pesos das folhas que resultem no menor valor possível dessa função. Para isso, derivamos a função objetivo em relação aos pesos e igualamos o resultado a zero, encontrando assim os valores que minimizam a perda.
- Ao fazer isso, a expressão resultante para o valor ótimo do peso de uma folha no XGBoost e a função objetivo correspondente a ele são:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$



Visão geral da matemática do XGBoost

- Agora que já temos uma forma de medir a qualidade de uma árvore por meio da função objetivo, o ideal seria enumerar todas as estruturas possíveis e selecionar a melhor. No entanto, essa abordagem é **computacionalmente intratável**, devido ao número exponencial de combinações.
- Por isso, o XGBoost adota um método chamado ***Exact Greedy Algorithm*** que consiste em otimizar **um nível da árvore por vez**, testando todas as possibilidades de divisões naquele nível e escolhendo a configuração que tiver maior **ganho de otimização**.
- A divisão só é aceita se o ganho for positivo, o que garante que a complexidade do modelo só aumente quando houver uma melhora justificada na função objetivo.

$$Gain = \frac{1}{2}\left[\frac{G_L{}^2}{H_L+\lambda} + \frac{G_R{}^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

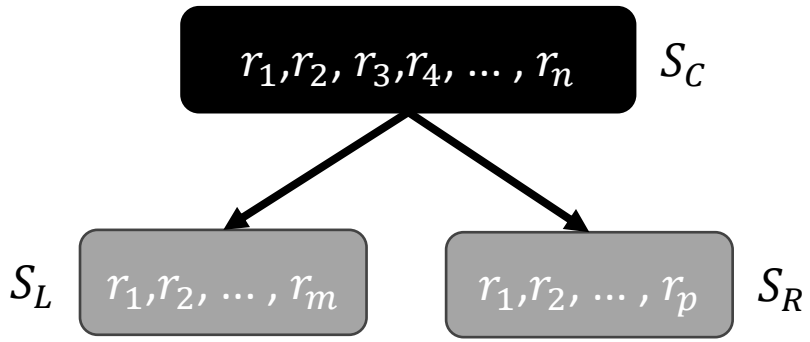
Visão geral da matemática do XGBoost

- Os termos da expressão do ganho são os fatores de similaridade de cada folha. O fator de similaridade representa a contribuição individual de uma folha para o ganho total do modelo.

$S_L = \frac{{G_L}^2}{H_L + \lambda}$ é a similaridade da folha esquerda

$S_R = \frac{{G_R}^2}{H_R + \lambda}$ é a similaridade da folha direita

S_C é a similaridade do nó pai



$$Gain = S_L + S_R - S_C - \gamma$$

XGBoost para Regressão

- As fórmulas utilizadas no processo de treinamento do XGBoost podem ser adaptadas conforme a função de perda escolhida. Em aplicações de regressão, por exemplo, é comum utilizar o erro quadrático médio (MSE). Dessa forma:
- Para $l(y_i, \hat{y}_{\boldsymbol{i}}) = \frac{1}{2}(y_i - \hat{y}_{\boldsymbol{i}})^2$:

$$g_i = \frac{d}{d\hat{y}_{\boldsymbol{i}}} l(y_i, \hat{y}_{\boldsymbol{i}}) = -(y_i - \hat{y}_{\boldsymbol{i}})$$

$$h_i = \frac{d^2}{d\hat{y}_{\boldsymbol{i}}^{\,2}} l(y_i, \hat{y}_{\boldsymbol{i}}) = 1$$

- Podemos interpretar a expressão de g_i como “resíduo negativo” e representá-la por $-r_i$.

XGBoost para Regressão

- Substituindo g_i e h_i na expressão geral do peso ótimo das folhas $w_j{}^* = -\frac{G_j}{H_j + \lambda}$, teremos:

$$w_j{}^* = \frac{\sum r_i}{|r_i| + \lambda}$$

- Substituindo g_i e h_i na expressão do cálculo de similaridade $S = \frac{G_j{}^2}{H_j + \lambda}$, teremos:

$$S = \frac{\sum r_i{}^2}{|r_i| + \lambda}$$

Onde:

- r_i são os resíduos da folha j
- $|r_i|$ = é a quantidade de resíduos na folha j
- $\lambda =$ é o fator de regularização L2

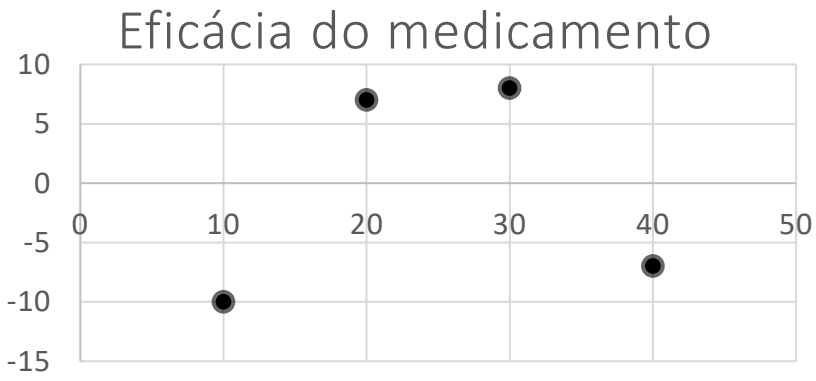
Processo de treinamento

- Inicia com uma predição única para todas as amostras (geralmente a média dos valores reais).
- Calcula os erros residuais em relação aos valores observados.
- Constrói uma nova árvore para corrigir esses erros, aplicando regularização durante o processo.
- Ajusta os pesos das folhas da árvore para otimizar a função objetivo.
- Atualiza as predições somando os valores produzidos pela nova árvore.
- Repete o processo iterativamente, adicionando árvores ao modelo, até que seja atingido um critério de parada.

Exemplo – Regressão

- Vamos considerar um cenário em que deseja-se prever a eficácia de um medicamento em função da dose ministrada, com base em observações anteriores.
- Valores positivos de eficácia significam que o efeito do medicamento foi positivo e valores negativos de eficácia significam que o medicamento fez mal ao paciente.

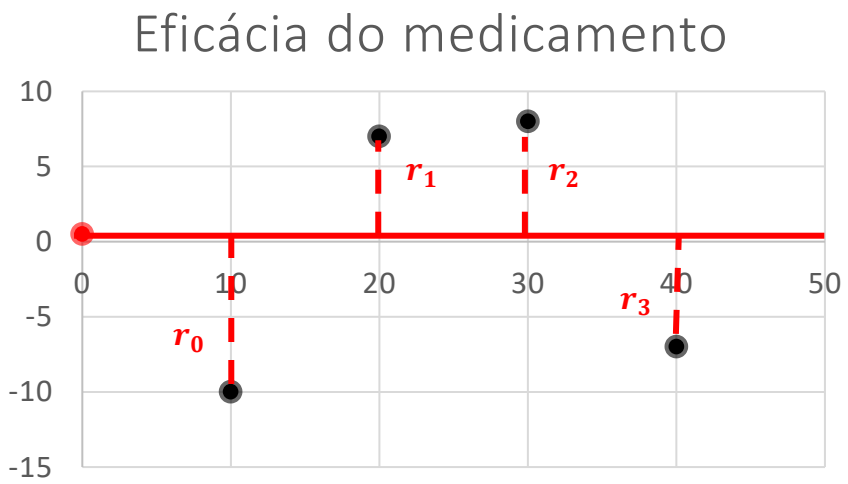
Dose (mg)	Eficácia
10	-10
20	7
30	8
40	-7



- O processo de treinamento consiste na geração de sucessivas árvores de decisão, com o objetivo de corrigir os erros das anteriores e formar um conjunto capaz de se ajustar aos dados de treinamento e generalizar bem para novos dados.

Exemplo – Regressão

x_i	y_i	$\hat{y}_i^{(0)}$	$r_i^{(1)}$
10	-10	0,5	-10,5
20	7	0,5	6,5
30	8	0,5	7,5
40	-7	0,5	-7,5



1) Faz-se uma previsão inicial para todos os exemplos, e calcula-se os resíduos em relação ao valor real:

$$\hat{y}_i^{(0)} = 0{,}5$$

$$r_i^{(1)} = y_i - \hat{y}_i^{(0)}$$

$$r_0^{(1)} = -10 - 0{,}5 = -10{,}5$$
$$r_1^{(1)} = 7 - 0{,}5 = 6{,}5$$
$$r_2^{(1)} = 8 - 0{,}5 = 7{,}5$$
$$r_3^{(1)} = -7 - 0{,}5 = -7{,}5$$

Exemplo – Regressão

2) Calcula-se o valor de similaridade para o nó raiz, que contém todos os resíduos iniciais.

$-10{,}5;\ 6{,}5;\ 7{,}5;\ -7{,}5$

$$S = \frac{\sum {r_i}^{(1)\ 2}}{|r| + \lambda}$$

λ = fator de regularização L2
|r| = quantidade de resíduos

$$S = \frac{(-10{,}5 + 6{,}5 + 7{,}5 - 7{,}5)^2}{4 + 0}$$

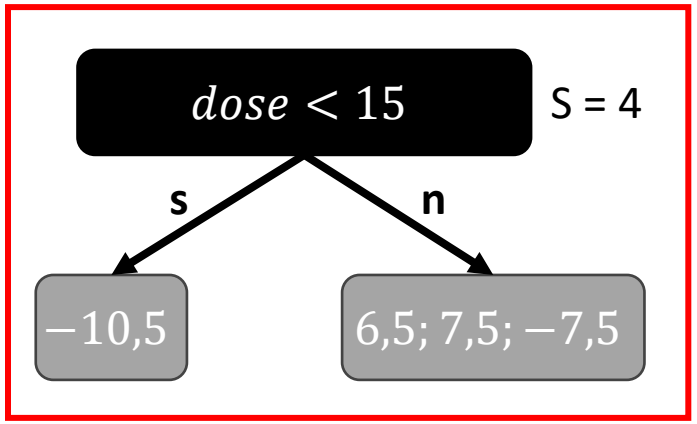
$$S = 4$$

A similaridade é uma métrica que quantifica o ganho ao se criar um nó folha com base nos dados que fazem parte dele. Ela expressa o quanto aquela folha contribui para reduzir o erro da função objetivo.

O fator de regularização λ controla a **regularização L2** nos pesos das folhas das árvores, o que ajuda a evitar o sobreajuste no modelo.

Exemplo – Regressão

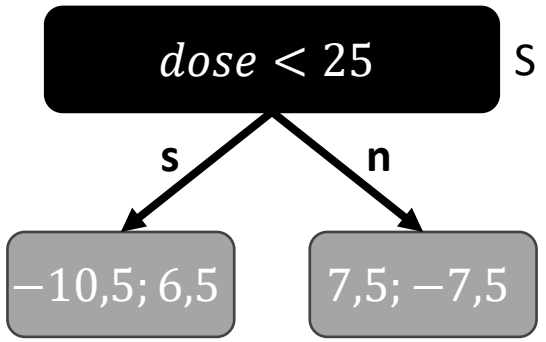
3) Avalia-se cada ponto de divisão. Para isso, divide-se os exemplos do nó em dois grupos, considerando todas as possibilidades. Então, calcula-se a similaridade para cada nó folha e a soma total de ambas as folhas. A configuração que apresentar o maior resultado será o melhor caminho.



$$S_L = \frac{(-10{,}5)^2}{1+0} = 110{,}25$$

$$S_R = \frac{(6{,}5 + 7{,}5 - 7{,}5)^2}{3+0} = 14{,}08$$

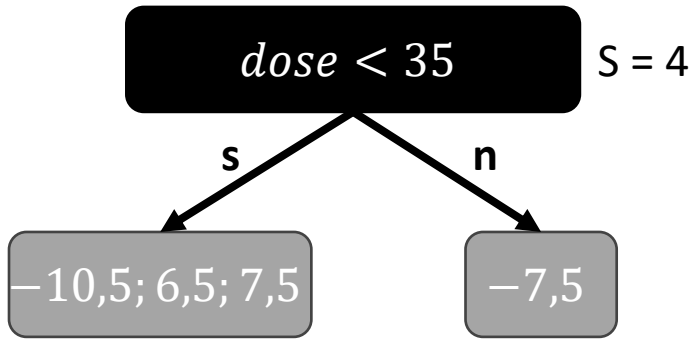
$$S_L + S_R = 110{,}25 + 14{,}08 = \mathbf{124{,}33}$$



$$S_L = \frac{(-10{,}5 + 6{,}5)^2}{2+0} = 8$$

$$S_R = \frac{(7{,}5 - 7{,}5)^2}{2+0} = 0$$

$$S_L + S_R = 8 + 0 = \mathbf{8}$$



$$S_L = \frac{(-10{,}5 + 6{,}5 + 7{,}5)^2}{3+0} = 4{,}08$$

$$S_R = \frac{(-7{,}5)^2}{1+0} = 56{,}25$$

$$S_L + S_R = 4{,}08 + 56{,}25 = \mathbf{60{,}33}$$

Exemplo – Regressão

4) Avalia-se o ganho da divisão, observando os valores de similaridade antes e depois da divisão e considerando o fator de poda (γ). Se o ganho der positivo, faz-se a divisão, se der negativo, a divisão não é realizada. Vamos considerar $\gamma = 0$.

$$G = S_L + S_R - S_C - \gamma$$

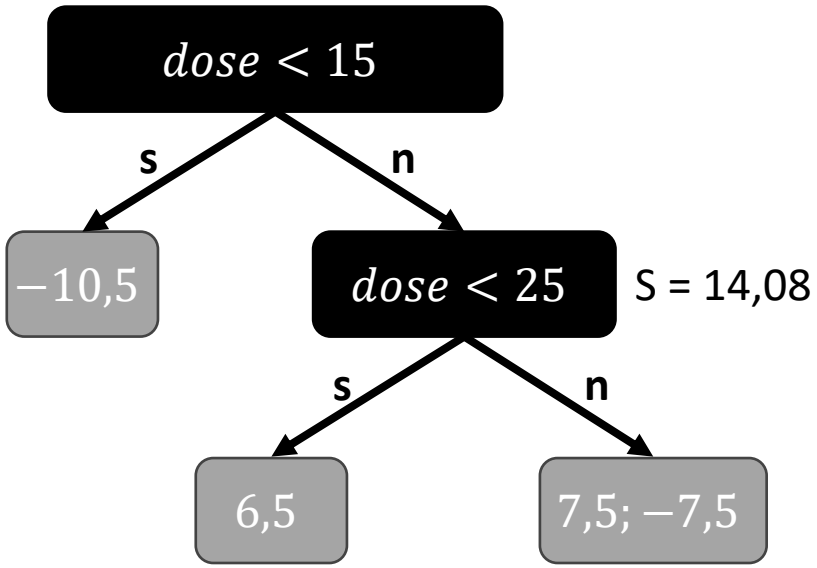
$$G = 124{,}33 - 4 - 0$$

$$G = 120{,}33$$

- Podemos prosseguir com a divisão, uma vez que 120,33 > 0.

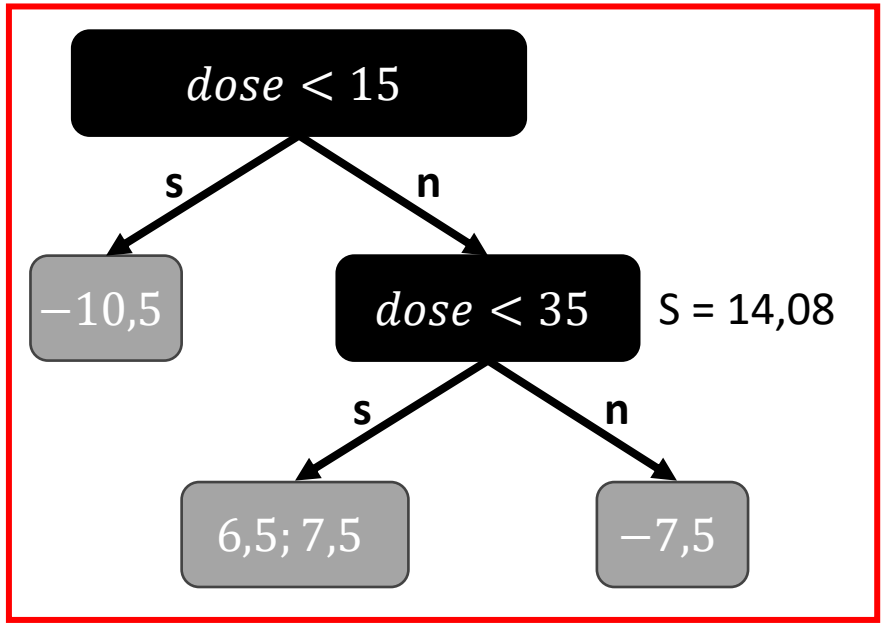
Exemplo – Regressão

5) Divide-se os nós filhos da mesma forma até uma profundidade definida. Vamos parar em 2.



$$S_L = \frac{(6{,}5)^2}{1+0} = 42{,}25 \qquad S_R = \frac{(7{,}5-7{,}5)^2}{2+0} = 0$$

$$S_L + S_R = 42{,}25 + 0 = 42{,}25$$



$$S_L = \frac{(6{,}5+7{,}5)^2}{2+0} = 98 \qquad S_R = \frac{(-7{,}5)^2}{1+0} = 56{,}25$$

$$S_L + S_R = 98 + 56{,}25 = 154{,}25$$

Exemplo – Regressão

- Analisando o ganho da divisão:

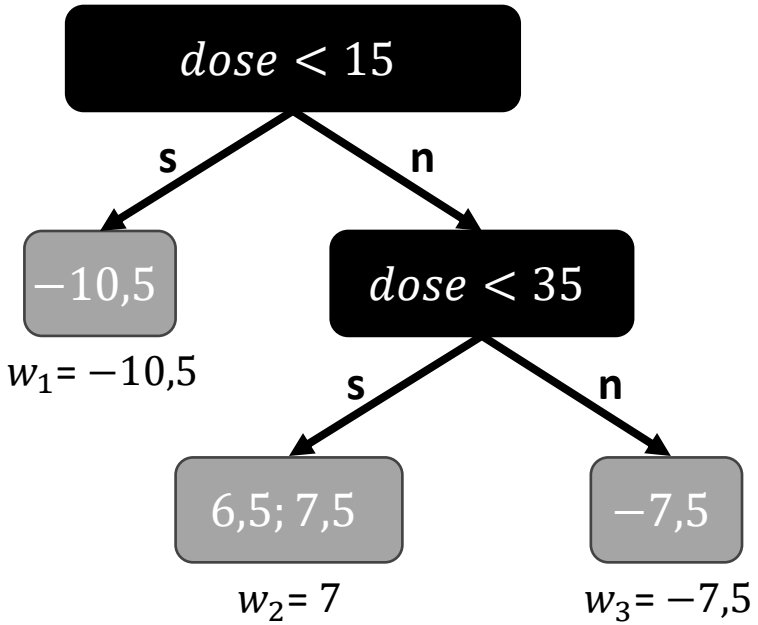
$$G = 154{,}25\ -14{,}08 - 0$$

$$G = 140{,}17$$

- Podemos manter a divisão, uma vez que 140,17 > 0.

- E se $\gamma = 150$? Nesse caso, o ganho seria negativo e nós não realizaríamos a divisão, o que significa que faríamos **uma poda** na árvore.

Exemplo – Regressão



6) Após finalizar as divisões, calcula-se os valores de saída dos nós folhas, também chamados de pesos (w_j).

$$w_j = \frac{\sum r_i^{(1)}}{|r| + \lambda}$$

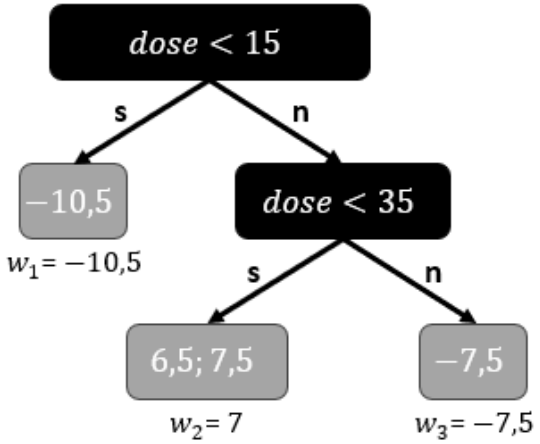
$$w_1 = \frac{-10{,}5}{1+0} = -10{,}5$$

$$w_2 = \frac{6{,}5+7{,}5}{2+0} = 7$$

$$w_3 = \frac{-7{,}5}{1+0} = -7{,}5$$

Exemplo – Regressão

x_i	y_i	$\hat{y}_i^{(0)}$	$r_i^{(1)}$	$\hat{y}_i^{(1)}$
10	-10	0,5	-10,5	-2,65
20	7	0,5	6,5	2,6
30	8	0,5	7,5	2,6
40	-7	0,5	-7,5	-1,75



7) Calcula-se novas predições usando a nova árvore gerada.

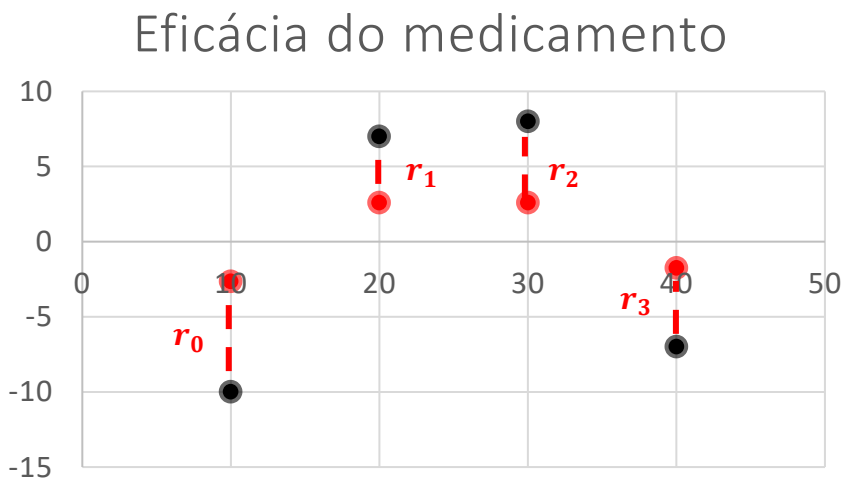
$$\hat{y}_i^{(1)} = \hat{y}_i^{(0)} + \eta w_j$$

η = passo de aprendizagem

$$\hat{y}_1^{(1)} = \hat{y}_1^{(0)} + \eta w_1 = 0{,}5 + 0{,}3(-10{,}5) = -2{,}65$$
$$\hat{y}_2^{(1)} = \hat{y}_2^{(0)} + \eta w_2 = 0{,}5 + 0{,}3(7) = 2{,}6$$
$$\hat{y}_3^{(1)} = \hat{y}_3^{(0)} + \eta w_2 = 0{,}5 + 0{,}3(7) = 2{,}6$$
$$\hat{y}_4^{(1)} = \hat{y}_4^{(0)} + \eta w_3 = 0{,}5 + 0{,}3(-7{,}5) = -1{,}75$$

Exemplo – Regressão

x_i	y_i	$\hat{y}_i^{(0)}$	$r_i^{(1)}$	$\hat{y}_i^{(1)}$	$r_i^{(2)}$ ↓
10	-10	0,5	-10,5	-2,65	$-7,35$
20	7	0,5	6,5	2,6	4,4
30	8	0,5	7,5	2,6	5,4
40	-7	0,5	-7,5	-1,75	$-5,25$



8) Calcula-se os novos resíduos usando as novas predições.

$$r_i^{(2)} = y_i - \hat{y}_i^{(1)}$$

$$r_1^{(2)} = y_1 - \hat{y}_1^{(1)} = -10 - (-2,65) = -7,35$$
$$r_2^{(2)} = y_2 - \hat{y}_2^{(1)} = 7 - 2,6 = 4,4$$
$$r_3^{(2)} = y_3 - \hat{y}_3^{(1)} = 8 - 2,6 = 5,4$$
$$r_4^{(2)} = y_4 - \hat{y}_4^{(1)} = -7 - (-1,75) = -5,25$$

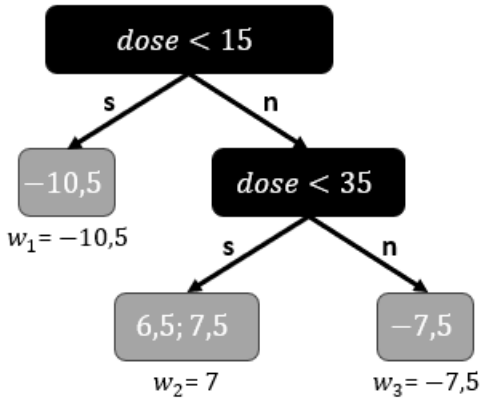
Podemos perceber que as novas predições têm resíduos menores que as iniciais, o que significa que foi dado um pequeno passo na direção correta.

Exemplo – Regressão

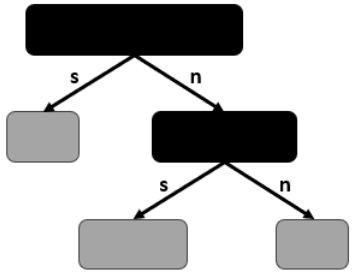
- Agora, geramos uma nova árvore com base nos novos resíduos, que nos dará resíduos ainda menores. E, então, geramos outra árvore com base nos novos resíduos, que nos dará resíduos ainda menores.
- E assim sucessivamente até que o resíduo seja minimizado ou até atingirmos o número máximo de árvores.

Predição inicial: **0,5**

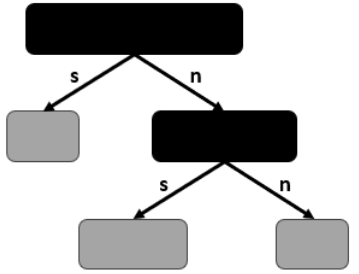
+ η X



+ η X



+ η X



...

Analisando o λ

- O λ controla a **regularização L2** dos pesos das folhas das árvores, o que ajuda a evitar o sobreajuste no modelo.
- Ele atua reduzindo a sensibilidade das predições a amostras individuais, pois sua influência é maior quanto menor a quantidade de elementos em uma folha.

$$w_j = \frac{\sum r_i^{(1)}}{|r_i| + \lambda}$$

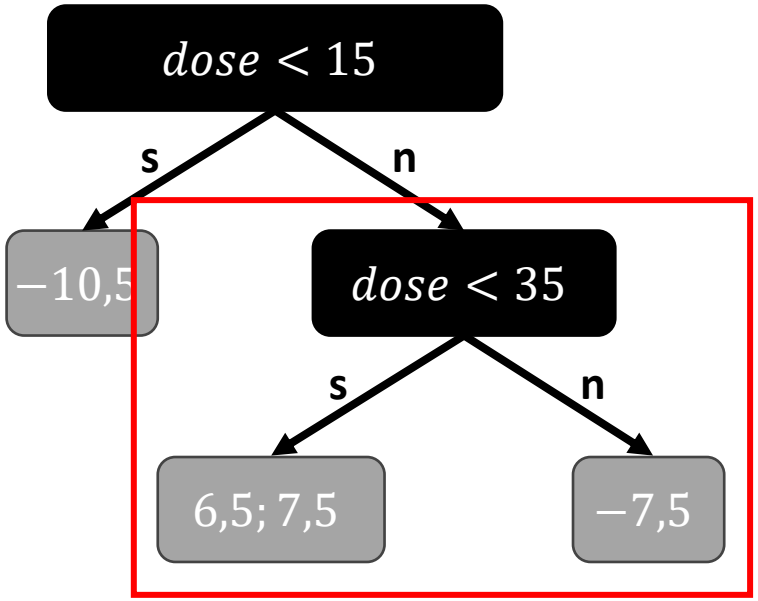
$\lambda = 0$		
Peso da folha (saída)	Valor predito	resíduo
$w_1 = \frac{-10,5}{1+0} = -10,5$ $w_2 = \frac{6,5+7,5}{2+0} = 7$ $w_3 = \frac{-7,5}{1+0} = -7,5$	$\hat{y}_1^{(1)} = -2,65$ $\hat{y}_2^{(1)} = 2,6$ $\hat{y}_3^{(1)} = 2,6$ $\hat{y}_4^{(1)} = -1,75$	$r_1^{(2)} = -7,35$ $r_2^{(2)} = 4,4$ $r_3^{(2)} = 5,4$ $r_4^{(2)} = -5,25$

$\lambda = 1$		
Peso da folha (saída)	Valor predito	resíduo
$w_1 = \frac{-10,5}{1+1} = -5,25$ $w_2 = \frac{6,5+7,5}{2+1} = 4,67$ $w_3 = \frac{-7,5}{1+1} = -3,75$	$\hat{y}_1^{(1)} = -1,075$ $\hat{y}_2^{(1)} = 1,9$ $\hat{y}_3^{(1)} = 1,9$ $\hat{y}_4^{(1)} = -0,625$	$r_1^{(2)} = -8,925$ $r_2^{(2)} = 5,1$ $r_3^{(2)} = 6,1$ $r_4^{(2)} = -6,375$

Analisando o λ

- Ele também atua indiretamente favorecendo o processo de poda da árvore, pois influencia inversamente o ganho de uma divisão.

$$S = \frac{\sum r_i^{(1)\,2}}{|r_i| + \lambda} \qquad G = S_L + S_R - S_C - \gamma$$



$\lambda = 0; \gamma = 85;$	$\lambda = 1; \gamma = 85;$
$S_R = \frac{(6{,}5 + 7{,}5 + (-7{,}5))^2}{3 + 0}$	$S_R = \frac{(6{,}5 + 7{,}5 + (-7{,}5))^2}{3 + 1}$
$S_R = 14{,}08$	$S_R = 10{,}56$
$S_E = \frac{(6{,}5 + 7{,}5)^2}{2 + 0} = 98$	$S_E = \frac{(6{,}5 + 7{,}5)^2}{2 + 1} = 65{,}33$
$S_D = \frac{(-7{,}5)^2}{1 + 0} = 56{,}25$	$S_D = \frac{(-7{,}5)^2}{1 + 1} = 28{,}13$
$G = 98 + 56{,}25 - 14{,}08 - 85$	$G = 65{,}33 + 28{,}13 - 10{,}56 - 85$
$G = 55{,}17$ **Faz divisão.**	$G = -2{,}1$ **Não faz divisão.**

Outros algoritmos de divisão de nós

- O algoritmo apresentado para divisão de nós ***(Exact Greedy Algorithm)*** é considerado robusto, pois enumera todos os pontos de divisão possíveis em cada nível da árvore.
- Mas, ele se torna **inviável em grades volumes da dados**, uma vez que exige manter toda a matriz de dados em memória e realizar múltiplas varreduras por atributo. Isso pode acarretar em **alto consumo de tempo e recursos computacionais**.
- Por isso, existem variações que podem ser selecionadas conforme o contexto da aplicação:

- ***Approximate Greedy Algorithm***: Agrupa as amostras aplicando uma técnica de **quantização baseada na distribuição dos dados ponderados**, utilizando o algoritmo **Weighted Quantile Sketch**. Os pontos de divisão refletem melhor a importância relativa das amostras (com base nos gradientes). É **mais preciso**, porém é **mais lento** do que o método baseado em histogramas simples.
- **Histogram-based**: Agrupa as amostras aplicando uma técnica de **quantização simples**, dividindo os valores dos atributos em **intervalos fixos (bins)**. É **mais rápido** e **mais eficiente**, porém **ligeiramente menos preciso** que o anterior. Atualmente, é o **padrão da biblioteca XGBoost**.

Outros algoritmos de divisão de nós

- ***Sparsity-aware Split Finding***: O XGBoost implementa uma técnica para encontrar divisões mesmo quando as amostras contêm **valores ausentes**, sem descartar ou forçar imputações nesses dados.
- Basicamente, em cada divisão, o algoritmo testa duas possíveis direções para os valores ausentes (esquerda ou direita) e calcula o ganho de otimização em cada caso. Então, escolhe a direção que maximiza o ganho e essa se torna a **direção padrão** para os valores ausentes nesse nó da árvore.

Biblioteca xgboost

- A implementação do XGBoost em Python é feita através da biblioteca **xgboost**.
- Ela possui uma API compatível com **scikit-learn**, permitindo o uso das classes **XGBClassifier** e **XGBRegressor**. Isso também facilita a integração com recursos como Pipeline, GridSearchCV, cross_val_score, entre outros.
- A biblioteca também oferece uma API de baixo nível, baseada nas funções train() e na estrutura DMatrix, que permite maior controle e flexibilidade na configuração e no treinamento do modelo.

Biblioteca xgboost – parâmetros importantes

Parâmetro	Descrição	Padrão
booster	Modelo base	'gbtree'
n_estimators	Número máximo de árvores (iterações)	100
learning_rate (eta)	Passo de aprendizagem	0.3
gamma	Ganho mínimo necessário para realizar um split	0
lambda	Regularização L2	1
alpha	Regularização L1	0
max_depth	Profundidade máxima da árvore	6
min_child_weight	Mínimo somatório dos pesos de observações em um nó filho	1
max_delta_step	Limite no valor máximo de atualização do peso de uma árvore	0
subsample	Proporção de amostras usadas por árvore	1
colsample_bytree	Proporção de features usadas por árvore	1

Exemplos de aplicação

Saúde e Medicina

- Predição de doenças e diagnósticos automatizados.

Finanças e Score de Crédito

- Análise de risco de crédito, detecção de fraude, precificação de seguros.

Marketing e Vendas

- Previsão de saída de clientes, recomendação de produtos, segmentação de clientes.

Logística e Cadeia de Suprimentos

- Previsão de demanda, otimização de rotas, prevenção de falhas em equipamentos via manutenção preditiva.

Perguntas?

Link para o quiz

- https://forms.gle/t3EFAqLpP3tBNwGh8

Referências

- CHEN, Tianqi; GUESTRIN, Carlos. XGBoost: a scalable tree boosting system. In: **SIGKDD Conference on Knowledge Discovery and Data Mining**, 22., 2016, San Francisco. New York: ACM, 2016. p. 785–794. Disponível em: https://doi.org/10.1145/2939672.2939785. Acesso em: 29 ago. 2025.
- **XGBOOST DEVELOPERS.** *XGBoost Documentation*. Disponível em: https://xgboost.readthedocs.io/en/stable/index.html. Acesso em: 29 ago. 2025.
- **DMLC.** *xgboost : Scalable, Portable and Distributed Gradient Boosting (GBDT, GBRT or GBM) Library*. Disponível em: https://github.com/dmlc/xgboost. Acesso em: 29 ago. 2025.
- **STATQUEST WITH JOSH STARMER.** *XGBoost Part 1 (of 4): Regression* [vídeo online]. YouTube, 16 dez. 2019. Disponível em: https://www.youtube.com/watch?v=OtD8wVaFm6E. Acesso em: 29 ago. 2025.

Obrigado!