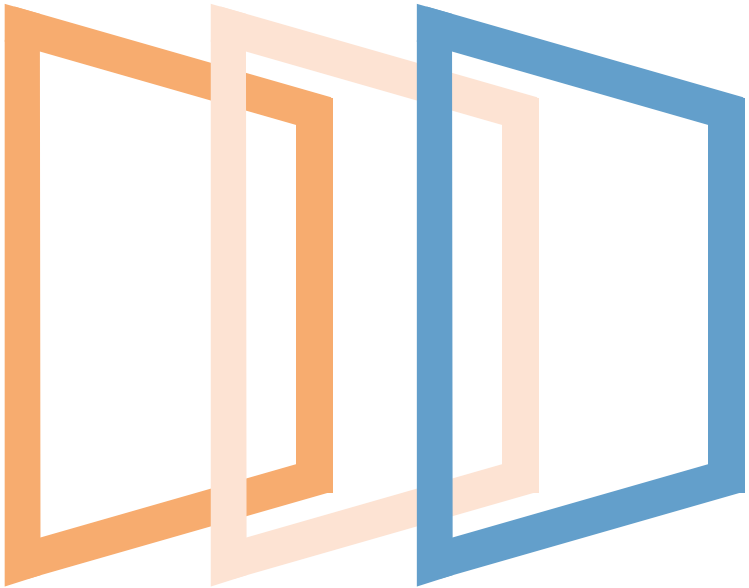
Desvendando IA e Machine Learning

Thaís Ratis

Diego Alexandre

Práticas Tecnológicas, 19.09.2023



minsait

An Indra company

Conteúdo programático

1. Perceptron
2. Rede Neural
3. Correção de erros
4. Backpropagation
5. Modos de treinamento
6. Critérios de parada
7. Métodos Ensemble

Perceptron

01

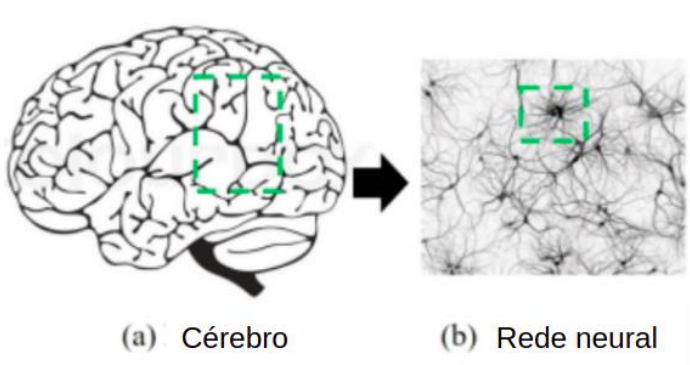
Inspiração biológica

- O cérebro é o principal órgão associado à inteligência e aprendizagem
- O cérebro é composto por uma rede complexa de aproximadamente 100 bilhões de neurônios interconectados
- Existem mais de 500 trilhões de conexões entre neurônios no cérebro humano
- Mesmo as maiores redes neurais artificiais de hoje não chegam perto do cérebro humano

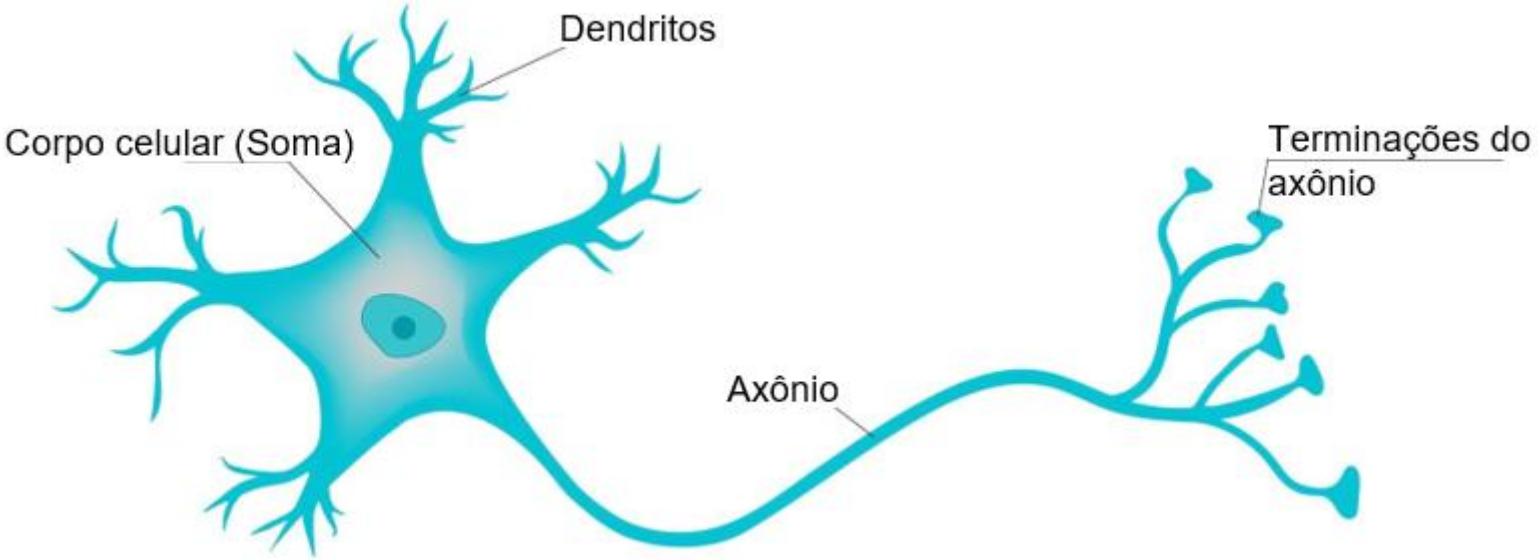
Inspiração biológica

A unidade básica de uma rede neural é o neurônio artificial

Neurônios artificiais são modelados como neurônios biológicos do cérebro, nos quais são estimulados por entradas



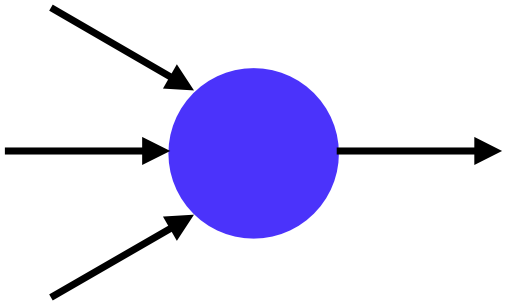
Neurônio

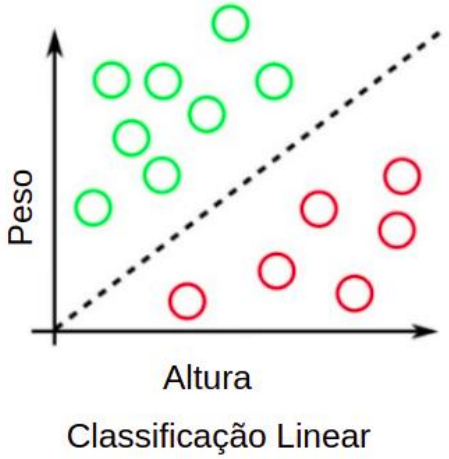

Dendritos
Corpo celular (Soma)
Terminações do axônio
Axônio

Perceptron

Em 1943, McCullock e Pitts propuseram um modelo de neurônio artificial: **perceptron**

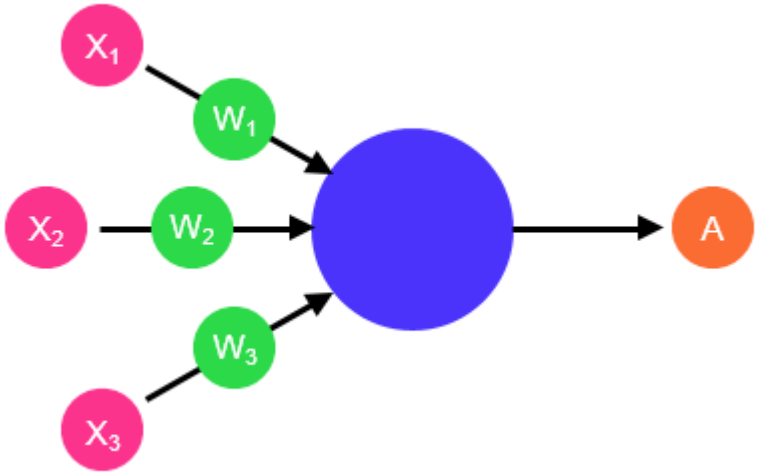
Modelo linear utilizado para classificação binária



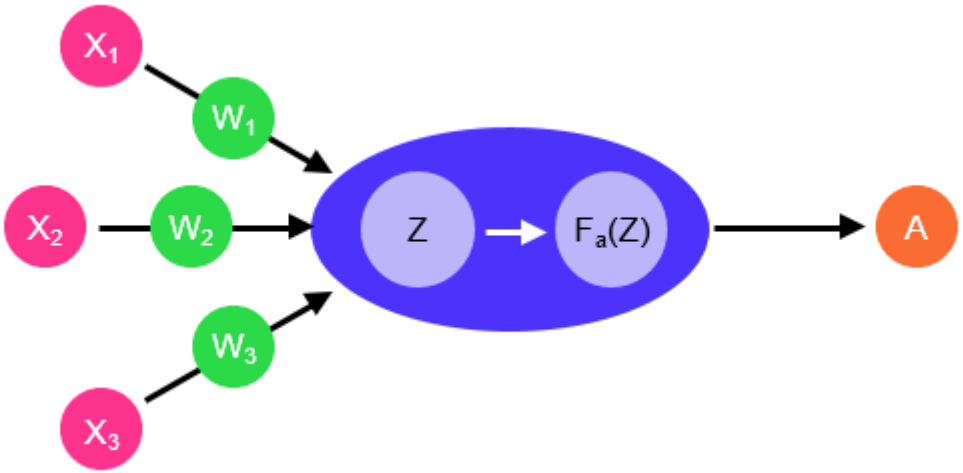


Classificação Linear

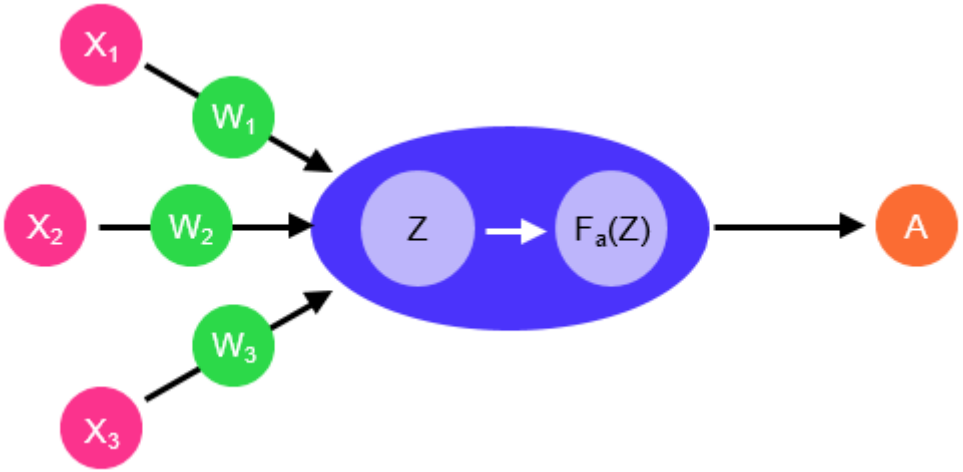
Perceptron



Perceptron

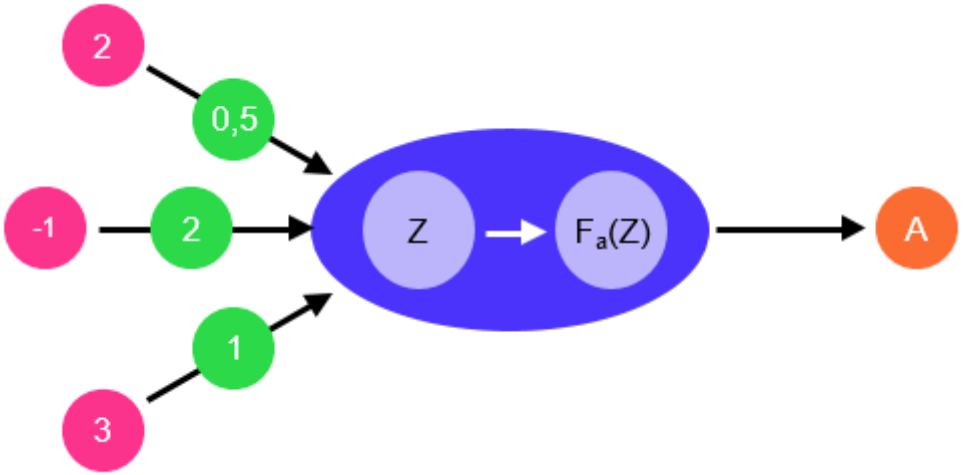


Perceptron



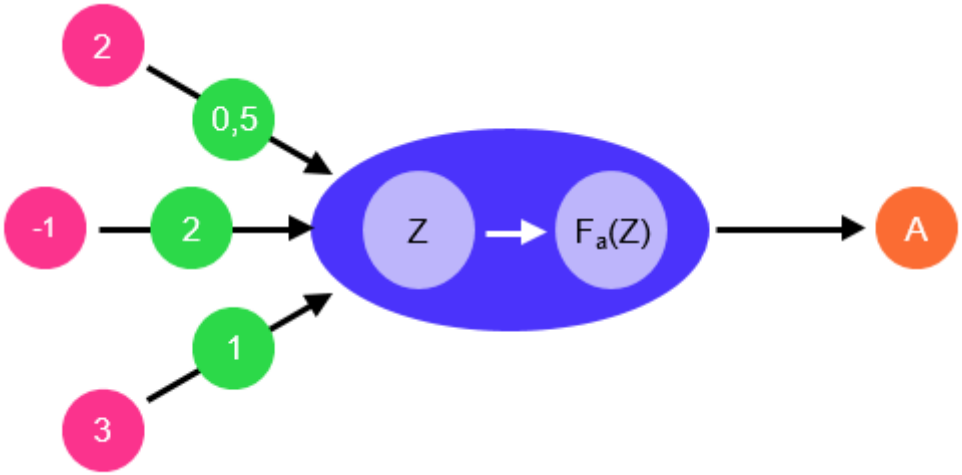
$$Z = W_1 \times X_1 + W_2 \times X_2 + W_3 \times X_3$$

Perceptron



$$Z = 0{,}5 \times 2 + 2 \times -1 + 1 \times 3 = 2$$

Perceptron



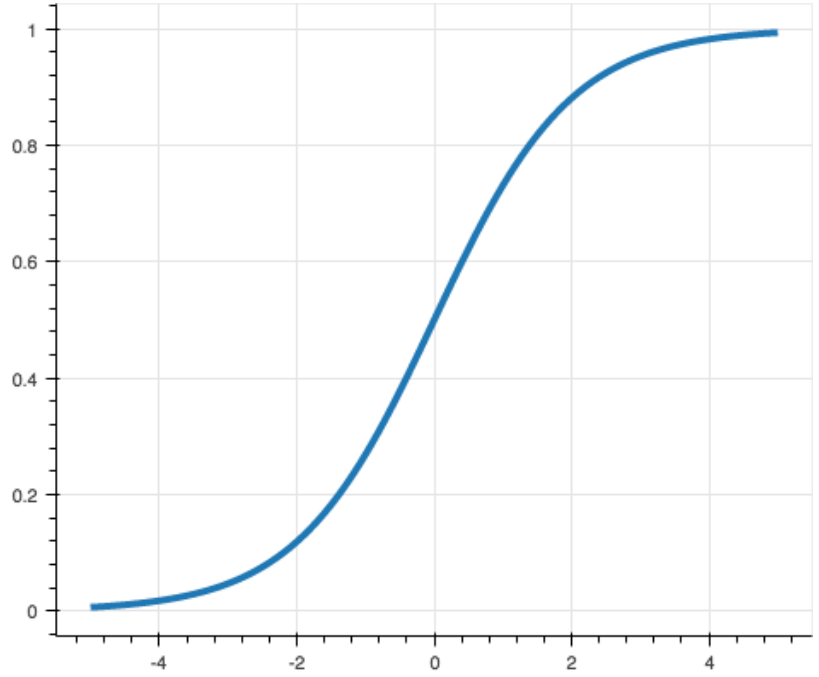
A = $F_a(2)$

Funções de ativação

Várias funções podem ser usadas como função de ativação (F_a)

Função Sigmóide:

$$f_a(z) = \frac{1}{1 + e^{-z}}$$

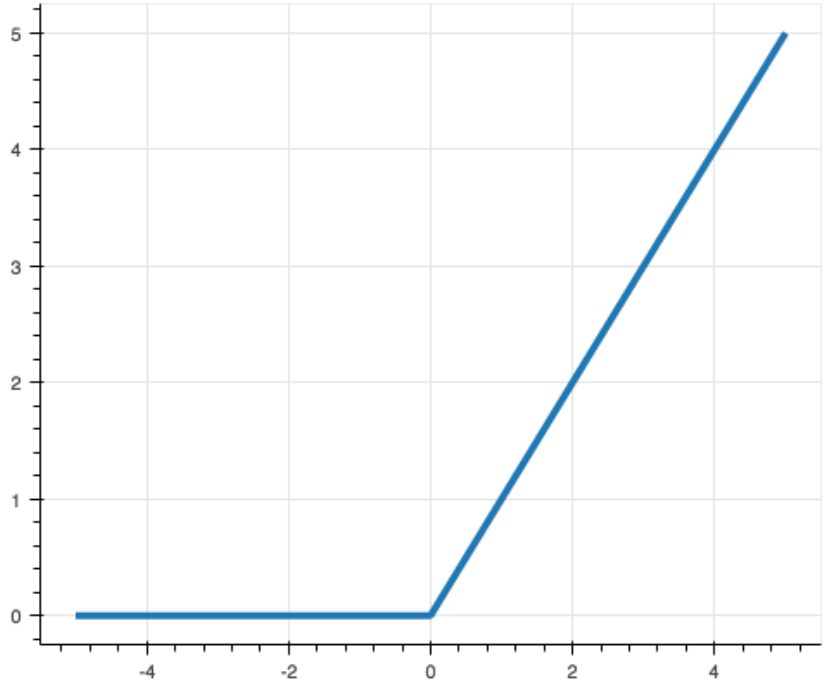


Funções de ativação

Várias funções podem ser usadas como função de ativação (F_a)

Função ReLU:

$f_a(z) = \max(0,z)$



Perceptron

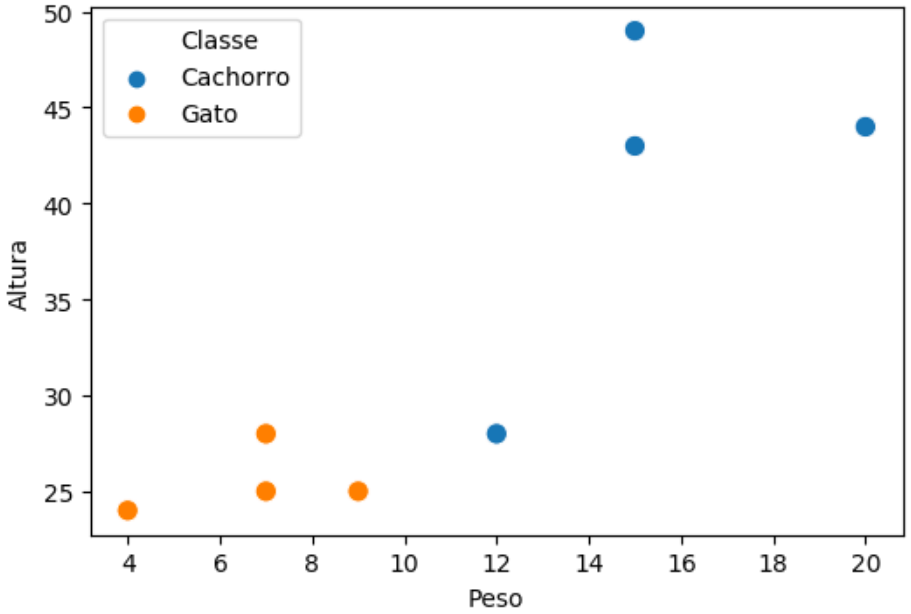
Podemos usar o neurônio artificial para diferentes problemas

Ele nos fornece uma saída de acordo com as entradas

O resultado do processamento das entradas para fornecer uma saída é determinado pela função de ativação e pelos pesos

Exemplo

Gatos e cachorros:



	Peso	Altura	Classe
0	20	44	Cachorro
1	15	43	Cachorro
2	12	28	Cachorro
3	15	49	Cachorro
4	7	28	Gato
5	7	25	Gato
6	9	25	Gato
7	4	24	Gato

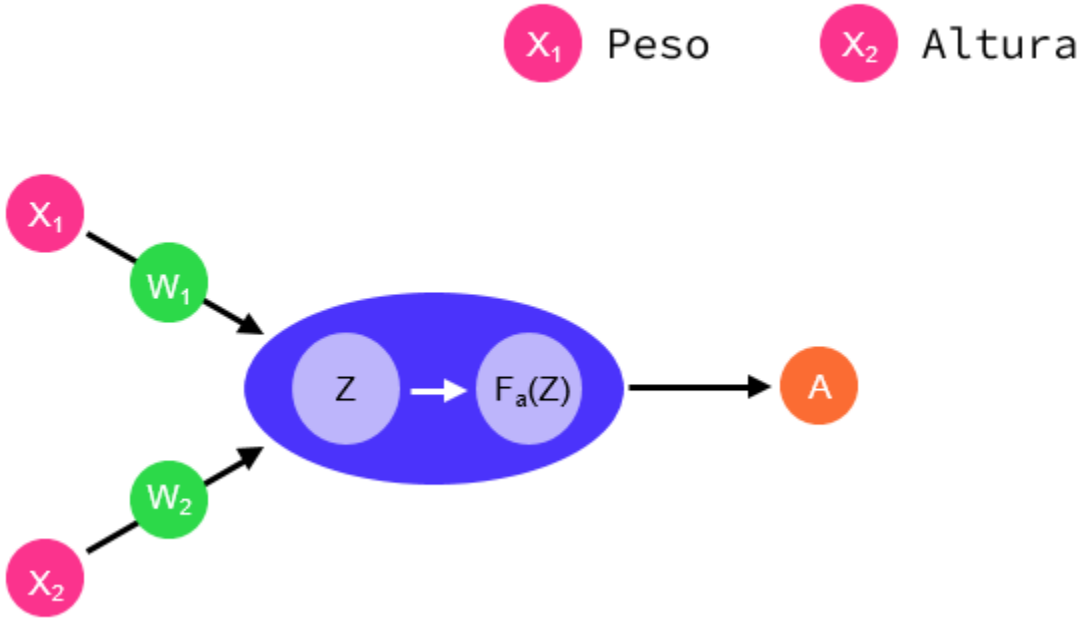
Exemplo

Dada a altura e o peso, vamos fazer o neurônio ter:

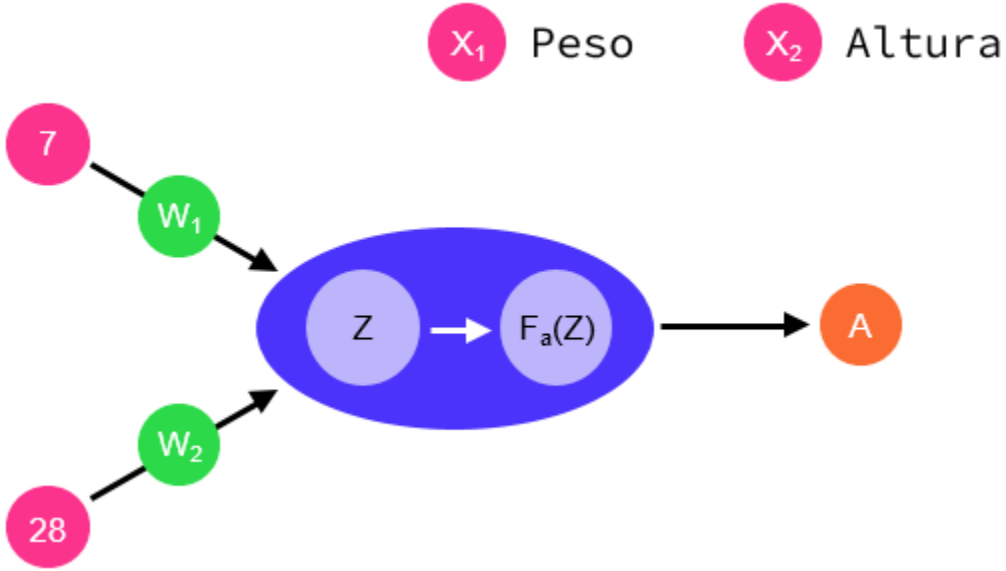
Saída **0** se for um gato

Saída **1** se for um cachorro

Exemplo



Exemplo



$$Z = W_1 \times 7 + W_2 \times 28$$

Rede Neural

02

Treinando uma rede neural

Quais são os valores de **w**?

Podemos tentar encontrá-los manualmente, porém este é um processo complexo e muitas vezes inviável

Treinando uma rede neural

Como não sabemos os pesos iniciais, vamos iniciá-los aleatoriamente

$w_1 = 0,25$

$w_2 = 0,66$

Em seguida, vamos calcular a saída para

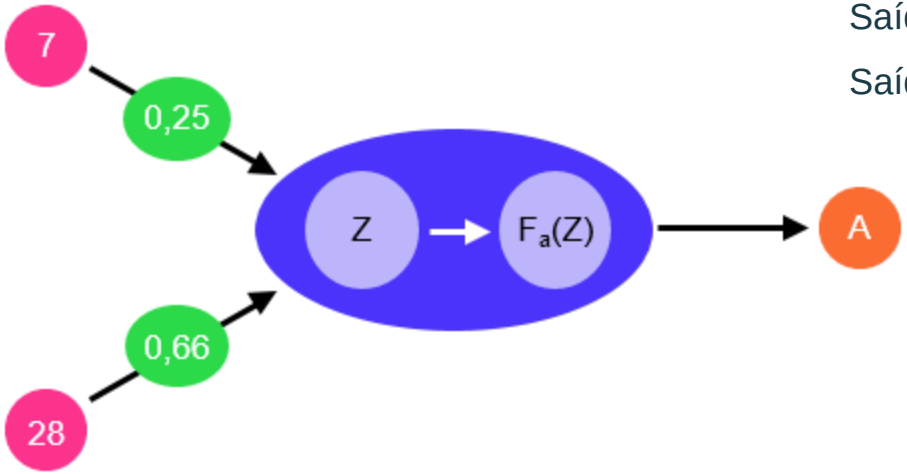
$x_1 = 7$

$x_2 = 28$

Treinando uma rede neural

Saída 0: gato

Saída 1: cachorro



$$Z = 0{,}25 \times 7 + 0{,}66 \times 28 = 20{,}23$$

$$A = F_a(20{,}23) = 0{,}99$$

Treinando uma rede neural

Esperado	Obtido	Ajuste	Esperado - Obtido
0	0	-	0
0	1	Diminuir W	-1
1	0	Aumentar W	1
1	1	-	0

Treinando uma rede neural

Para isso, eu preciso saber o erro, ou seja a diferença entre o valor esperado e o valor obtido

Quanto maior o erro, maior precisa ser o ajuste dos valores de **w**

Então, vamos atualizar $\mathbf{w_1}$ da seguinte forma:

$$w_1 \leftarrow w_1 + \eta\,(esperado - obtido)\,x_1$$

Taxa de aprendizagem: controla o tamanho da atualização

Treinando uma rede neural

Cada ajuste vai melhorando um pouco a rede neural

Vamos ajustar os pesos para cada um dos exemplos dos dados de treinamento

$$w_1 \leftarrow w_1 + \eta(esperado - obtido)x_1$$

$$w_2 \leftarrow w_2 + \eta(esperado - obtido)x_2$$

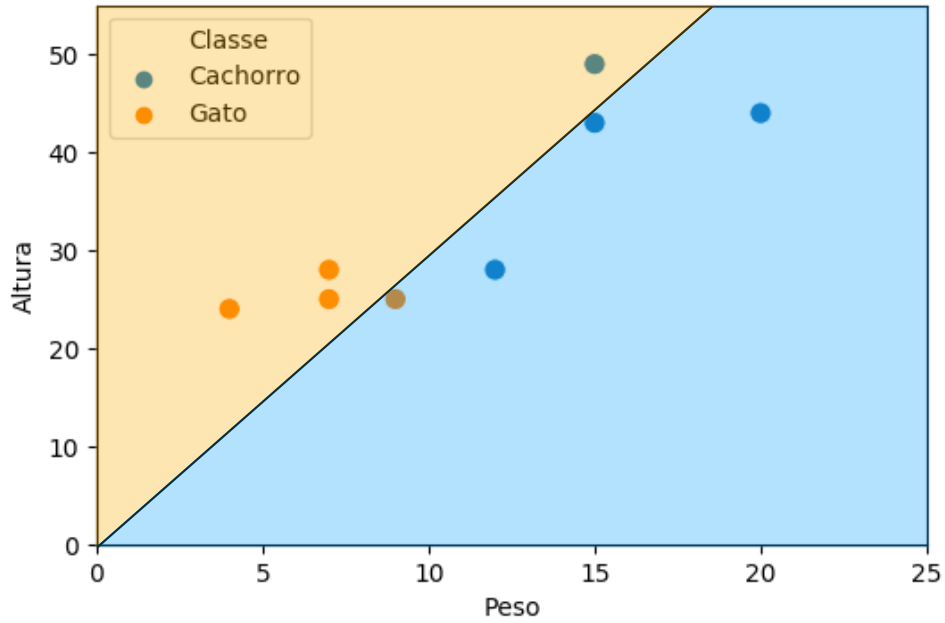
Treinando uma rede neural

Treinamento de todas as amostras do conjunto.

Esse número de vezes é chamado de **épocas**

Exemplo

Temos como resultado:



Treinando uma rede neural

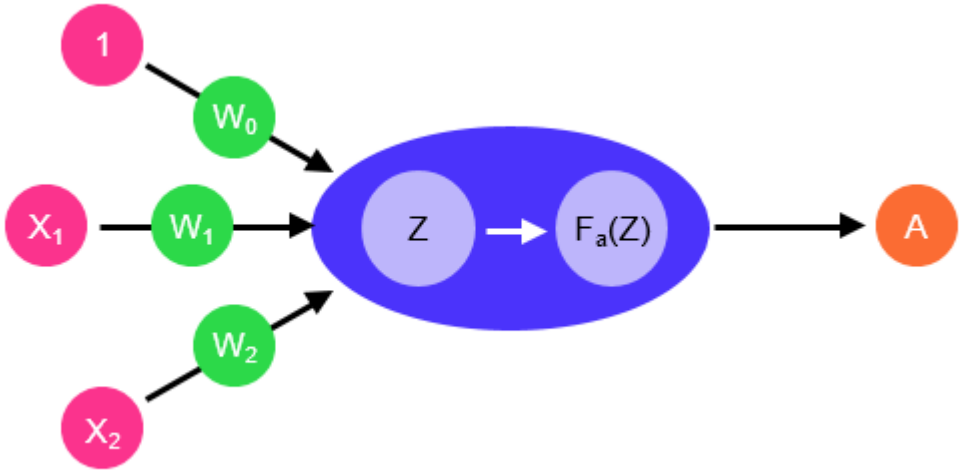
- Usando o neurônio com a estrutura que fizemos, sempre teremos uma fronteira linear partindo da origem

- Para permitir que a fronteira possa se deslocar no eixo X, vamos adicionar uma entrada extra no neurônio chamada de **viés**

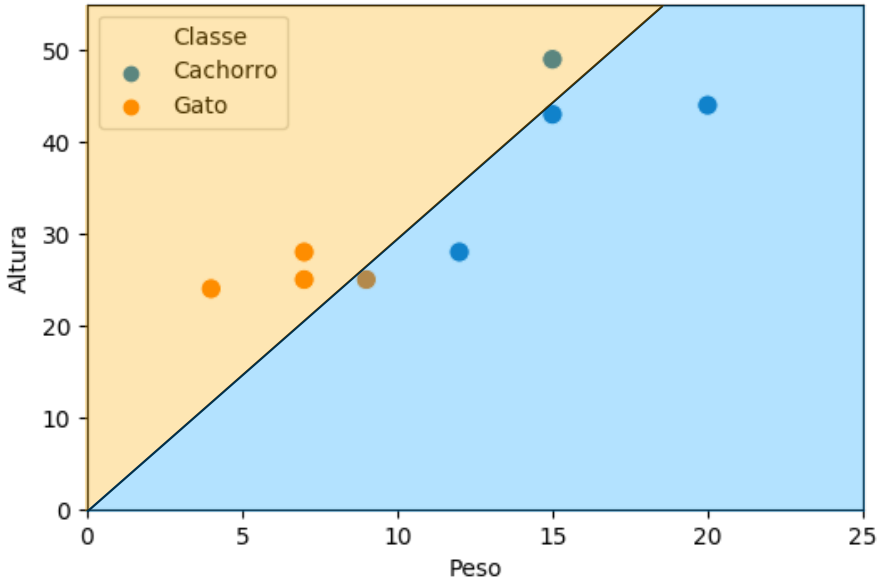
- O viés geralmente recebe 1 como entrada

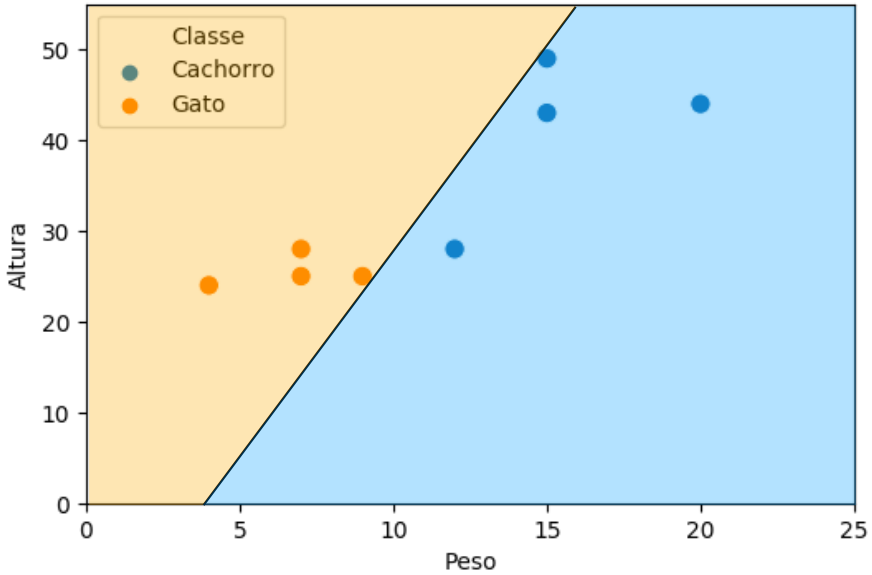
- O peso dessa entrada segue o comportamento das outras

Treinando uma rede neural



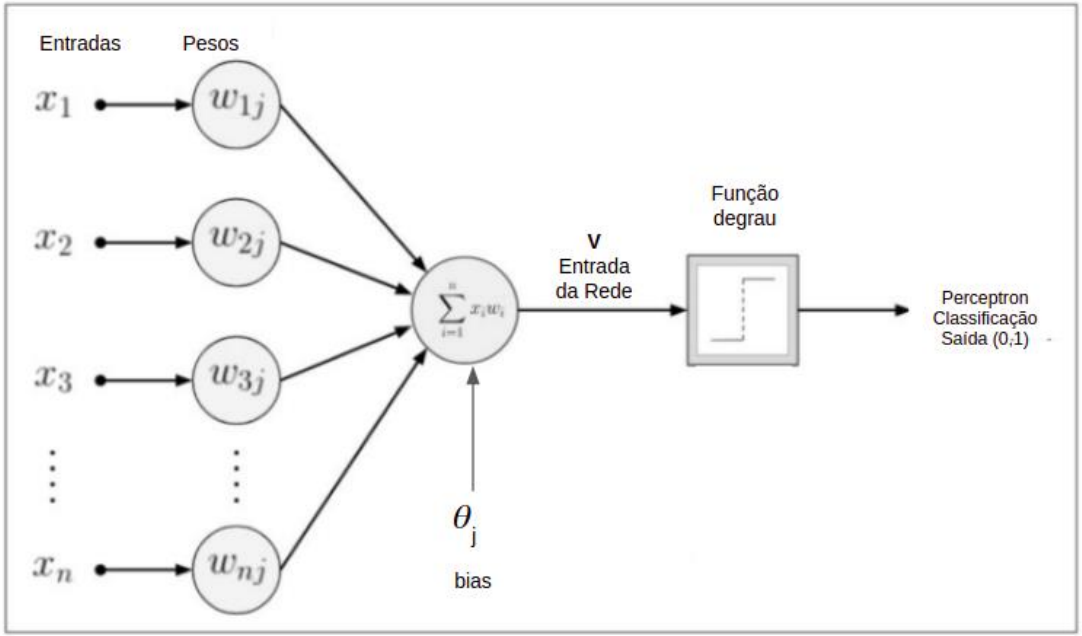
Viés (bias)


Classe
Cachorro
Gato
Altura
Peso
0
10
20
30
40
50
0
5
10
15
20
25


Classe
Cachorro
Gato
Altura
Peso
0
10
20
30
40
50
0
5
10
15
20
25

Estrutura do perceptron

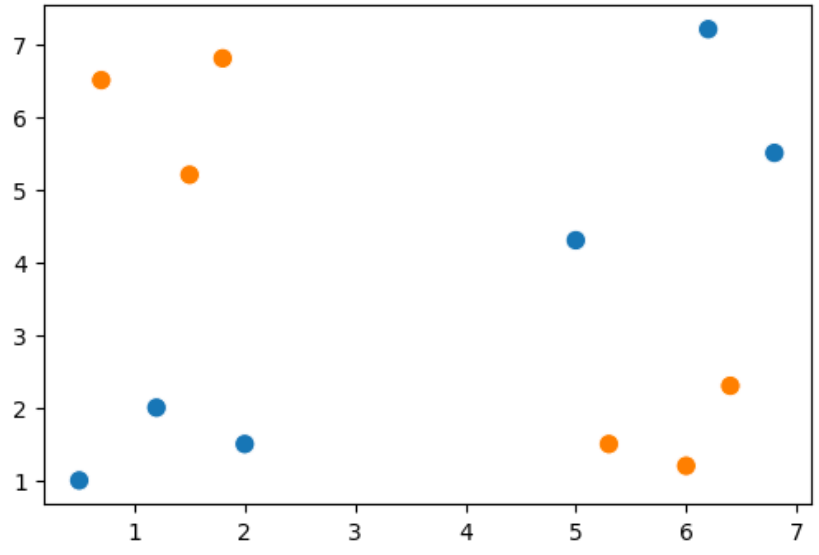
Função de ativação de Heaviside: $f(v) = \begin{cases} 0 & v < 0 \\ 1 & v >= 0 \end{cases}$



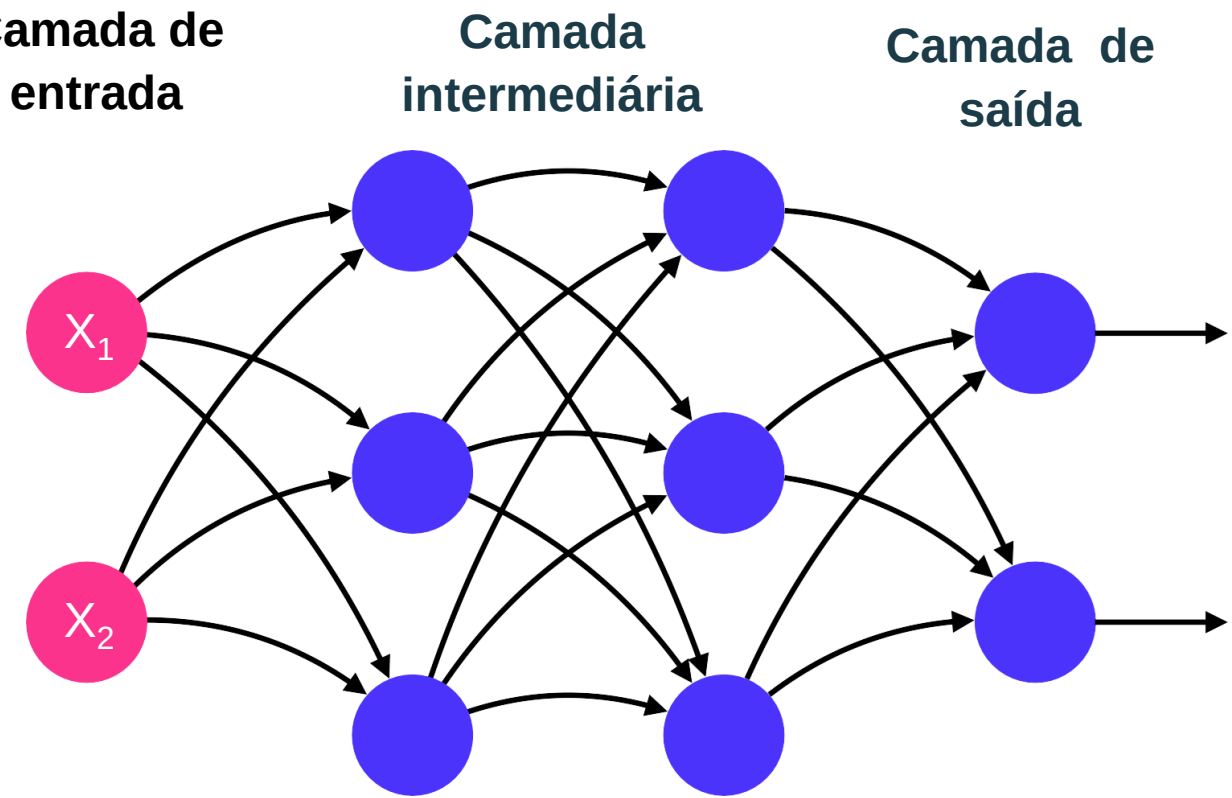
Treinando uma rede neural

Como vimos, o perceptron possui a limitação de criar apenas fronteiras lineares

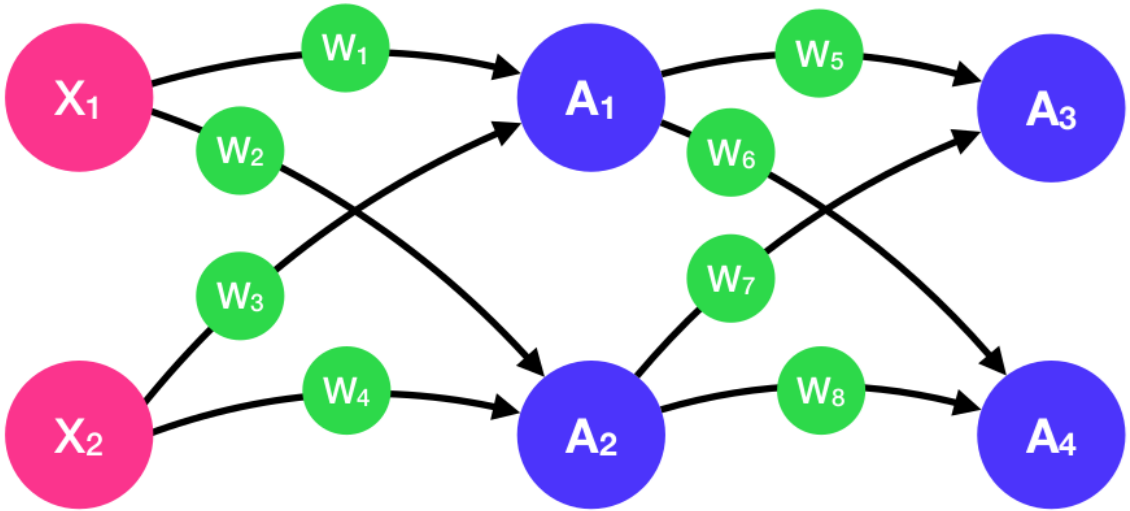
Ele não consegue classificar bem um conjunto de dados dessa forma:



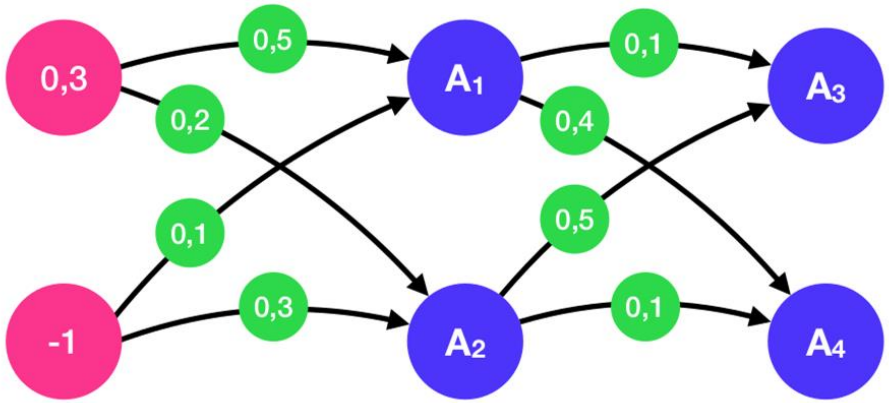
Rede neural


Camada de entrada
Camada intermediária
Camada de saída
X_1
X_2

Rede neural



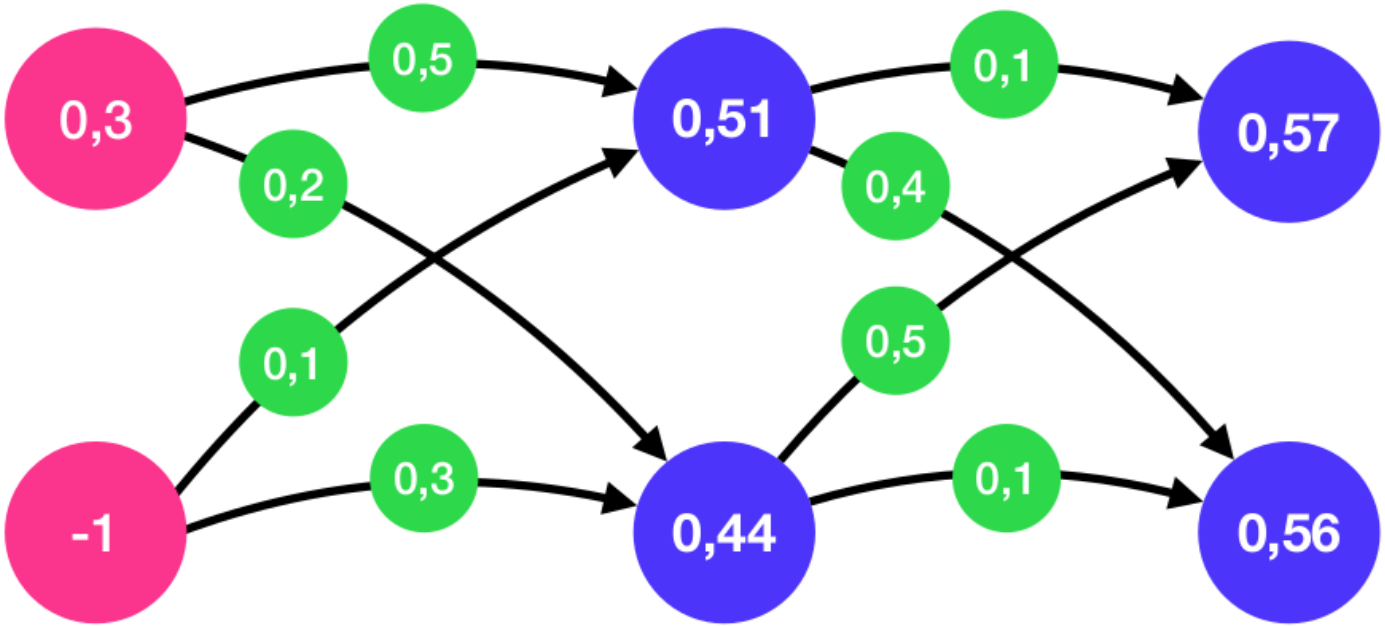
Rede neural



$$Z_1 = 0{,}5 \times 0{,}3 + 0{,}1 \times -1 = 0{,}049$$

$$A_1 = F_a(0{,}049) = 0{,}51$$

Rede neural



Treinando uma rede neural

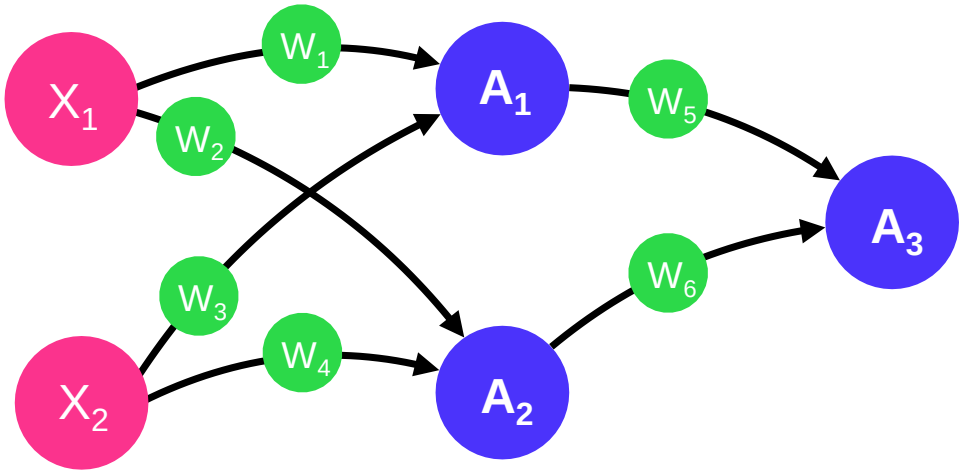
O algoritmo é semelhante ao que fizemos para apenas um neurônio

Iniciamos os pesos de forma aleatória

Entramos com os dados de treinamento e comparamos o resultado da rede com o resultado esperado

Ajustamos os pesos de acordo com o erro

Treinando uma Rede neural



O valor de A_3 é a saída da rede. Podemos compará-lo diretamente com o resultado dos dados de treinamento

E para A_1 e A_2? Quais são os valores esperados? Não temos esses dados

Treinando uma Rede neural

Para treinar uma rede com multicamadas usamos o algoritmo *backpropagation*

Nele, vamos definir uma função que mede o erro da saída da rede

Também conseguimos calcular através do gradiente da função de erro, em qual sentido devemos ajustar os pesos para que o erro diminua

Correção de Erros

03

Correção de erros

Calculamos o erro para um exemplo através da função:

$$e_k = d_k - y_k$$

Onde:

e - Sinal de erro
d - Saída desejada apresentada durante o treinamento
y - Saída real da rede

Aprendizado por Correção de erros

- O processo de aprendizado por correção de erros utiliza algoritmos para caminhar sobre a curva de erros, com o intuito de alcançar o menor valor de erro possível, o mínimo global
- Muitas vezes o algoritmo não alcança este mínimo global, atingindo o que chamamos de mínimo local. Caso este erro alcançado seja desfavorável, é necessário recomeçar o processo de aprendizagem

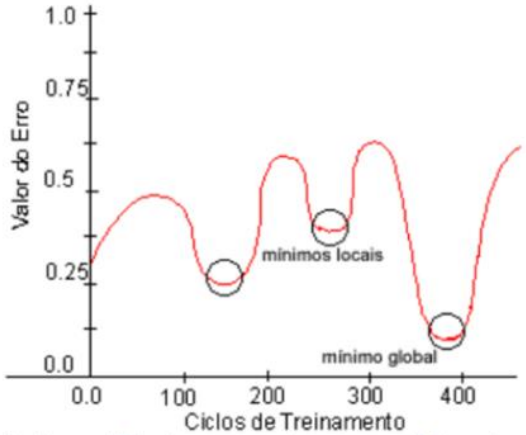


Gráfico de uma possível superfície de erro mostrando os mínimos locais e o mínimo global

Aprendizado por Correção de erros

Para correção do erro, os pesos devem ser ajustados de forma a aproximar a saída real à desejada

$$\Delta wi(n) = \eta\, e(n)\, xi(n)$$

Onde:

△wi(n) - Valor de ajuste a ser acrescido ao peso wi;

n - Taxa de aprendizagem (constante positiva);

e(n) - Valor do erro;

xi - Valor da entrada

Correção de erros

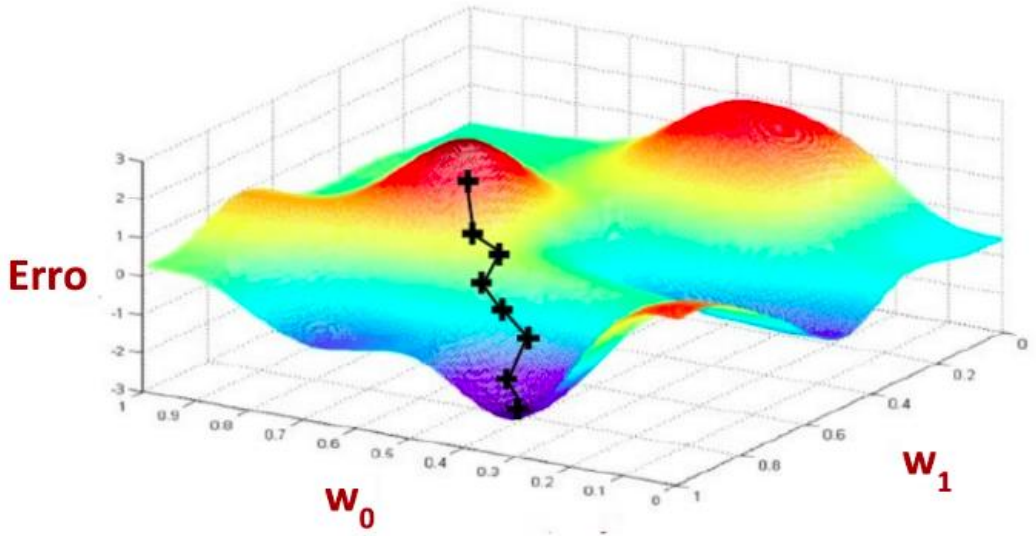
Encontrar o vetor (matriz) de pesos sinápticos (w*) que minimizem o erro (e) entre a saída da rede neural e a saída desejada

Backpropagation

04

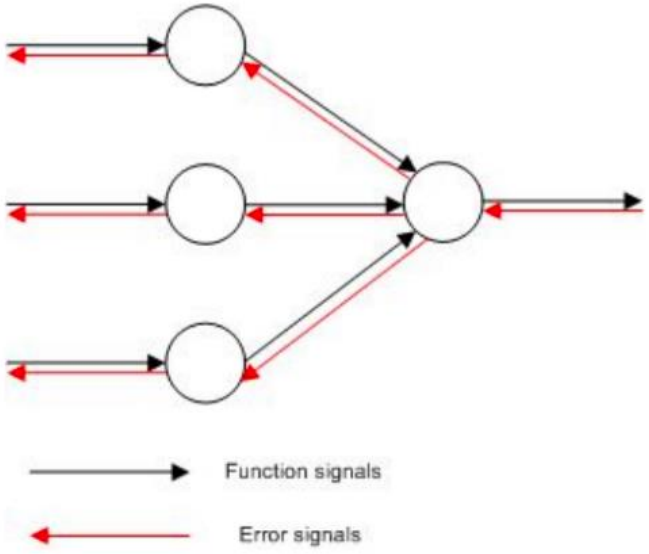
Método do Gradiente

O método do gradiente (ou método do máximo declive) é um método numérico usado em otimização. Para encontrar um mínimo (local) de uma função usa-se um esquema iterativo, onde em cada passo se toma a direção (negativa) do gradiente, que corresponde à direção de declive máximo.



Backpropagation

Sinal de erro se origina em um neurônio de saída e se propaga para trás (camada por camada) através da rede



Correção dos pesos - Regra delta

$$\Delta w_{ji}(n) = \eta * \delta j(n) * y_i(n)$$

- $\Delta w_{ji}(n)$: Correção de peso
- η: Taxa de aprendizagem
- $\delta j(n)$: Gradiente local
- $y_i(n)$: Sinal de entrada do neurônio *j*

Gradiente Local

Camada de saída

$$\delta_j(n) = e_j * \varphi'(v_j(n))$$

- Gradiente Local
- Sinal de erro na saída de *j*
- Derivada da função de ativação no neurônio j

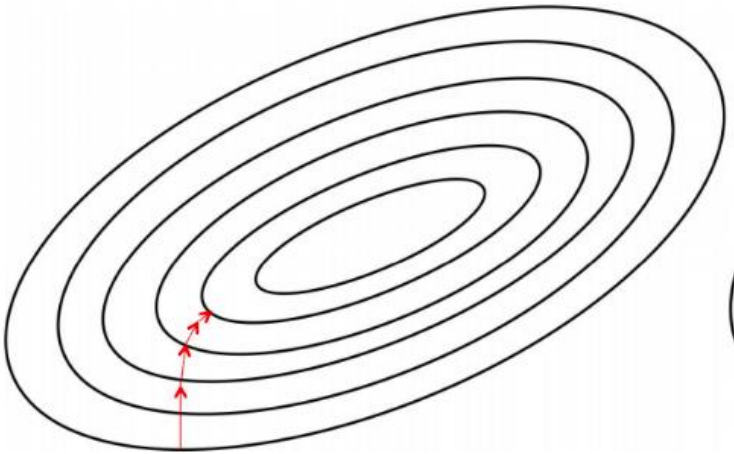
Camada oculta

$$\delta_j(n) = \varphi'\left(v_j(n)\right) \sum_k \delta_k(n) w_{kj}(n)$$

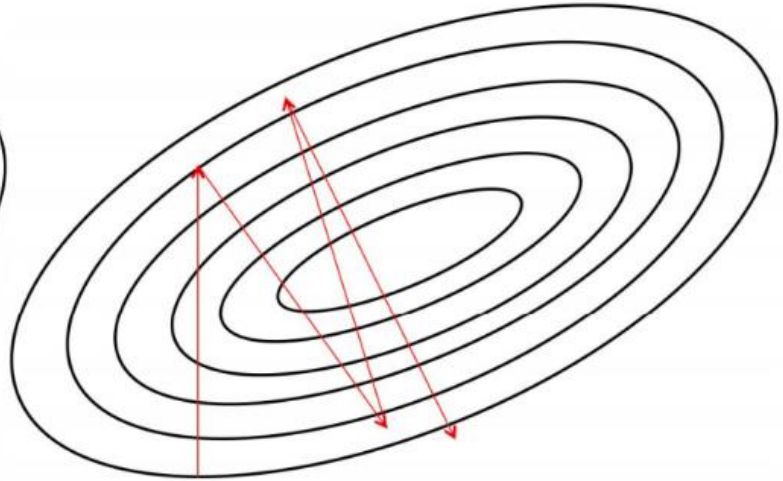
- Gradiente Local
- Derivada associada ao neurônio j
- Soma ponderada dos gradientes locais da camada seguinte k

Taxa de aprendizagem

η muito pequeno
Aprendizagem e convergência lenta



η muito grande
Rede instável



Modos de treinamento: Estocástico, por lote e mini-lote

05

Modos de treinamento

Baseado na forma de atualização dos pesos, temos 3 modos de treinamento:

- Estocástico - Ajuste de pesos é realizado após a apresentação de cada exemplo
- Por lote (batch) - Ajuste de pesos é realizado após a apresentação de todo os exemplos à rede (fim da época)
- Mini-lote (mini-batch) - Ajuste de pesos é realizado após a apresentação de um subconjunto de exemplos

Estocástico

Vantagens:

- Ajuda a escapar de mínimos locais
- Mais simples de implementar
- Pode tirar vantagens de dados (exemplos) redundantes

Desvantagens:

- Estimar o erro baseado em um único exemplo não é uma boa aproximação do erro real
- Treinamento muito lento
- Mais difícil provar teoricamente que o algoritmo converge

Por lote (batch)

Vantagens:

- Estimativa precisa do gradiente
- Convergência mais rápida sob condições simples
- Mais fácil de paralelizar

Desvantagens:

- Pode ficar preso em mínimos locais

Por mini-lote (mini-batch)

Bom balanço entre o modo estocástico e o modo por lote:

- Convergência mais rápida (modo por lote)
- Evita mínimos locais

Critérios de parada

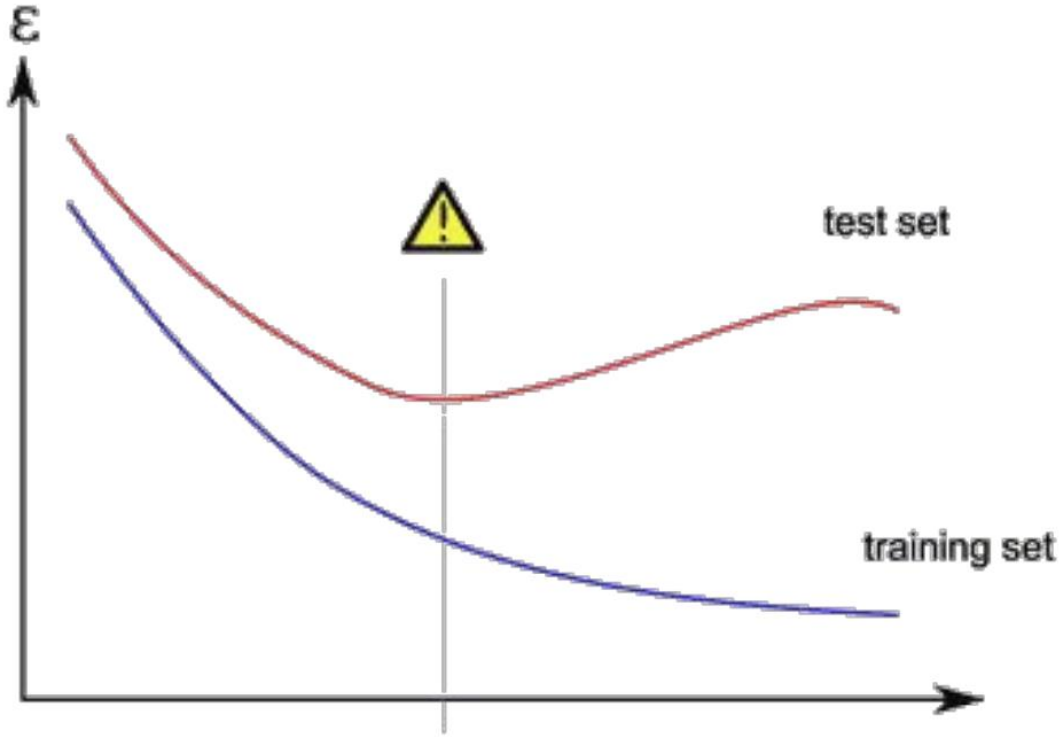
06

Critérios de parada

Existem alguns critérios razoáveis para a convergência:

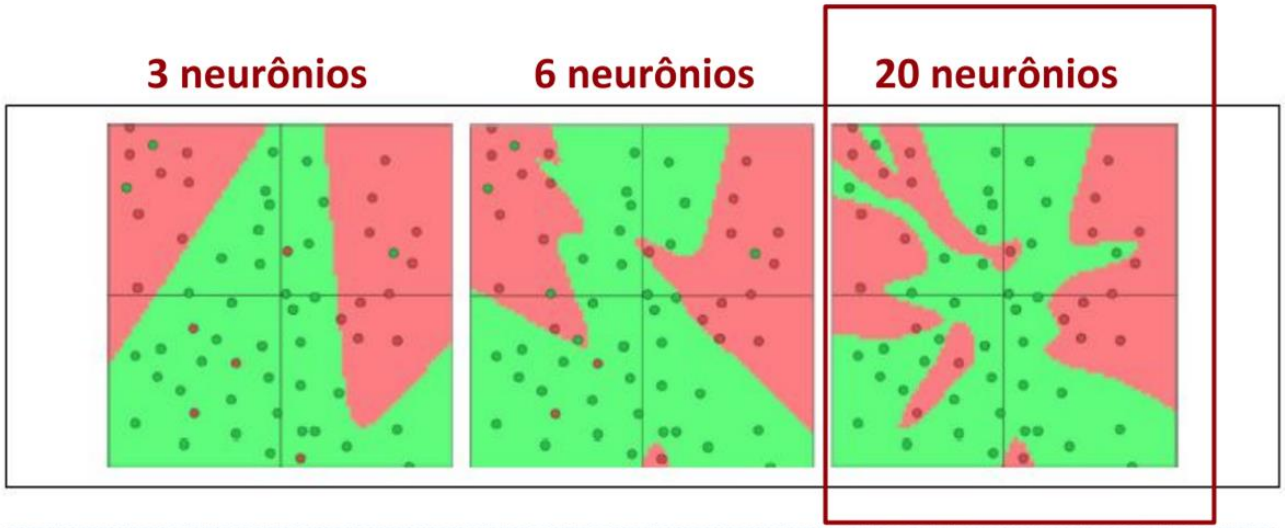
- Vetor gradiente alcançar um limiar suficiente pequeno
- Taxa de variação do erro muito pequena entre as épocas (ex: menor que 1%)
- Rede apresenta um bom desempenho de generalização, ou seja, funciona bem com um outro conjunto de exemplos (conjunto de validação)

Overfitting



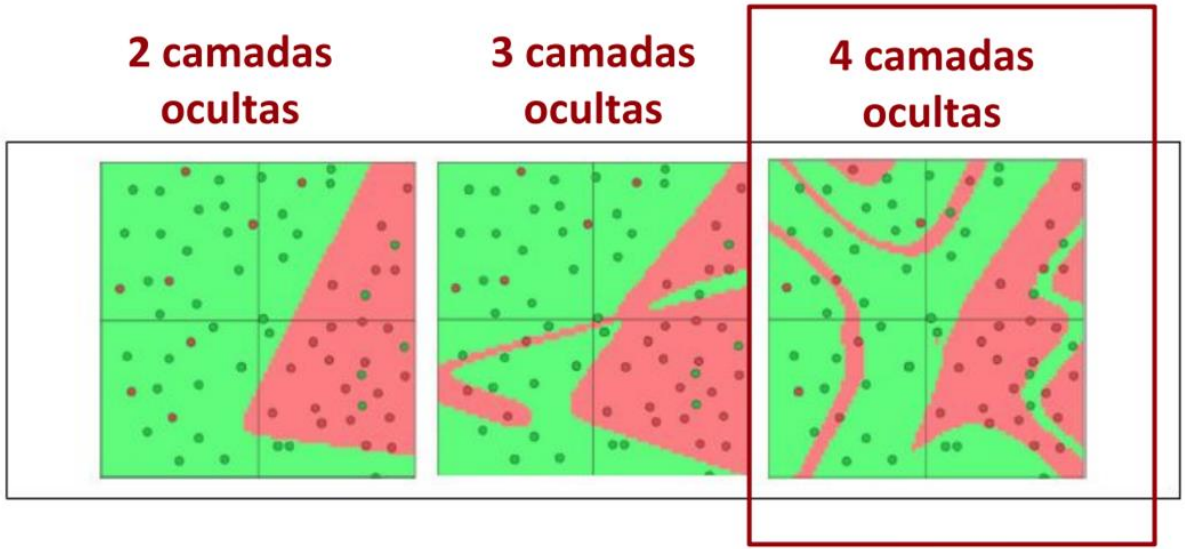
Overfitting

- Aumento no número de conexões implica em maior propensão a **overfitting.**
- Ex: RNA formada por 1 camada oculta contendo:



Overfitting

- Aumento no número de camadas também aumenta a propensão a **Overfitting**.



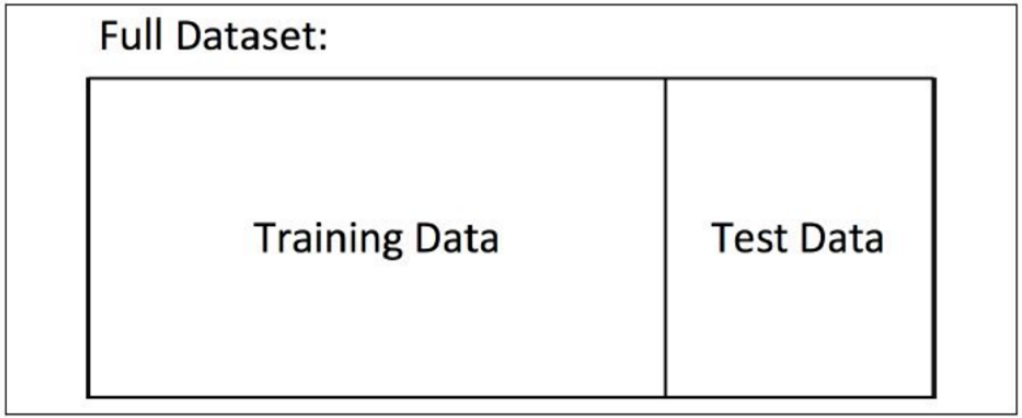
Overfitting

- Overfitting é um problema comum em DL.

- DL procura resolver **problemas muito complexos** com **modelos complexos**
  - Necessário tomar medidas para prevenir o **overfitting**.
  - Veremos algumas dessas medidas

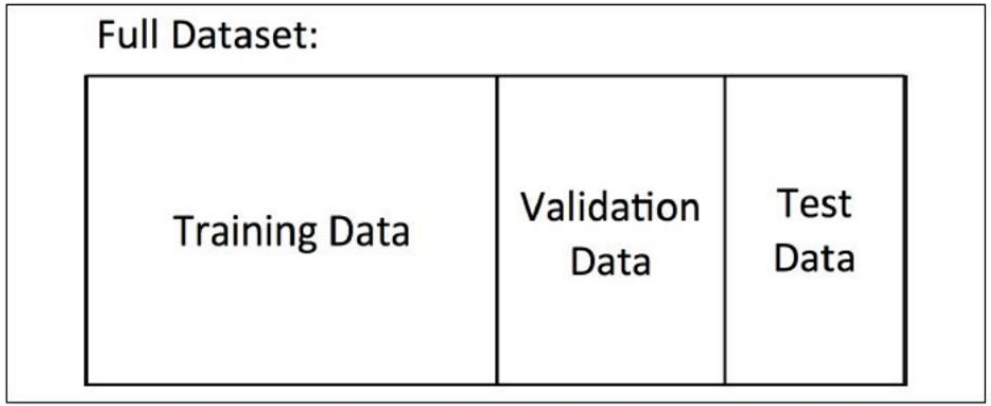
Prevenção de Overfitting

- **Medida 1: Não avalie seu modelo com os dados que você usou para treinar.**
  - Divida seu conjunto em conjunto de treinamento e conjunto de testes.



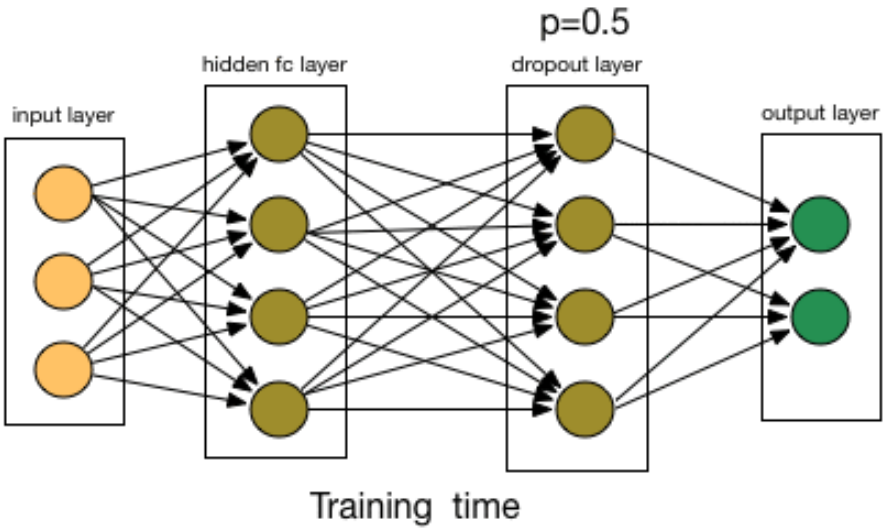
Prevenção de Overfitting

- **Medida 2: Crie também um conjunto de validação.**
  - Use-o para avaliar o treinamento de época em época;
  - Se o desempenho continuar **melhorando no treinamento** mas **piorando no conjunto de validação**, é hora de parar.
    - **Você está em OVERFITTING.**



Dropouts

- Durante o treinamento, mantém um neurônio ativo com uma probabilidade *p*.



Dropout

- **Uma das estratégias preferidas, em DL, para prevenir overfitting.**
- Força a rede a ter uma boa acurácia mesmo na ausência de certas informações.
- Evita que a rede se torne muito dependente de um (ou um conjunto de) neurônios.

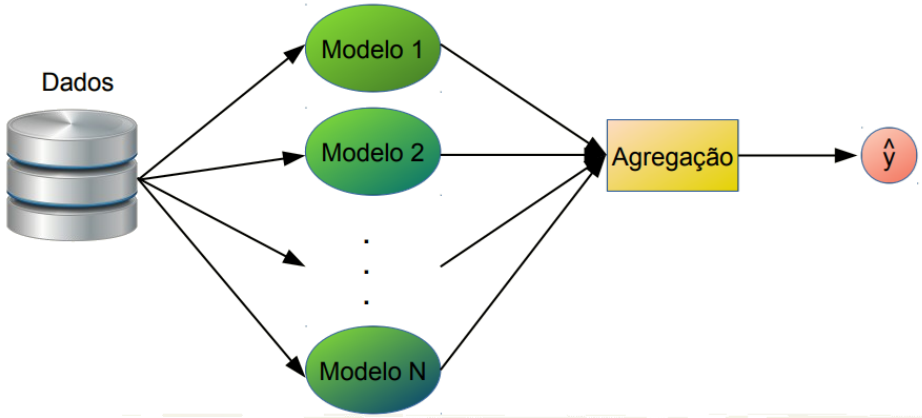
Métodos Ensemble

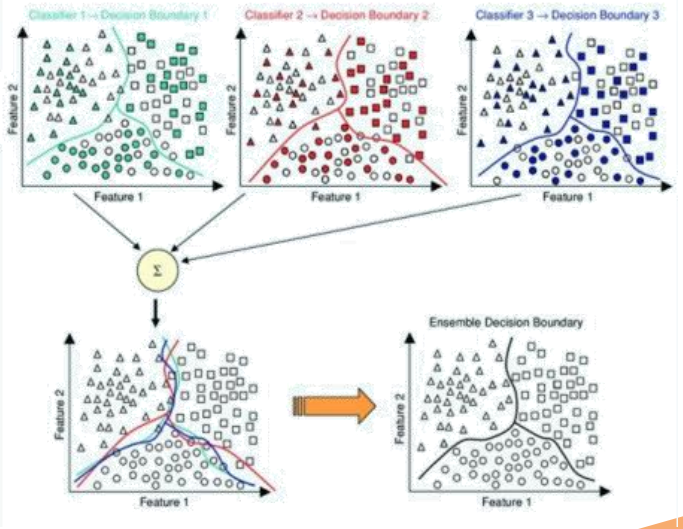
07

Métodos *Ensemble*

Estes métodos constroem vários modelos de *machine learning*, utilizando o resultado de cada modelo na definição de um único resultado, obtendo-se assim um valor final único.

Isso significa que a resposta agregada de todos esses modelos é que será dada como o resultado final para cada dado que se está testando. Aqui estamos falando de algoritmos mais robustos e complexos, que envolvem mais operações, com um custo computacional um pouco maior.





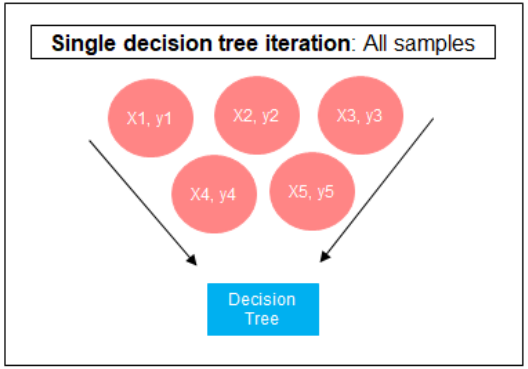
Métodos Ensemble - Funcionamento

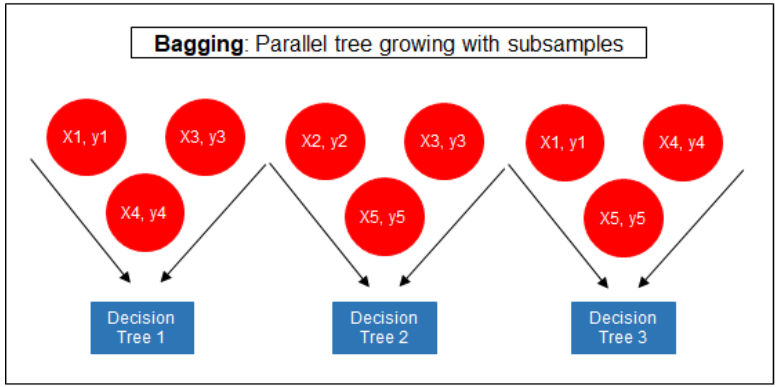
Vamos imaginar que estamos trabalhando com algoritmos de árvores de decisão: ao invés de criar apenas uma árvore que classifique os dados, e depois utilizar um novo conjunto de dados para receber sua classificação para ver como essa árvore de decisão os classificará, vamos construir várias árvores de decisão, e agregar os resultados de todas elas num resultado final.

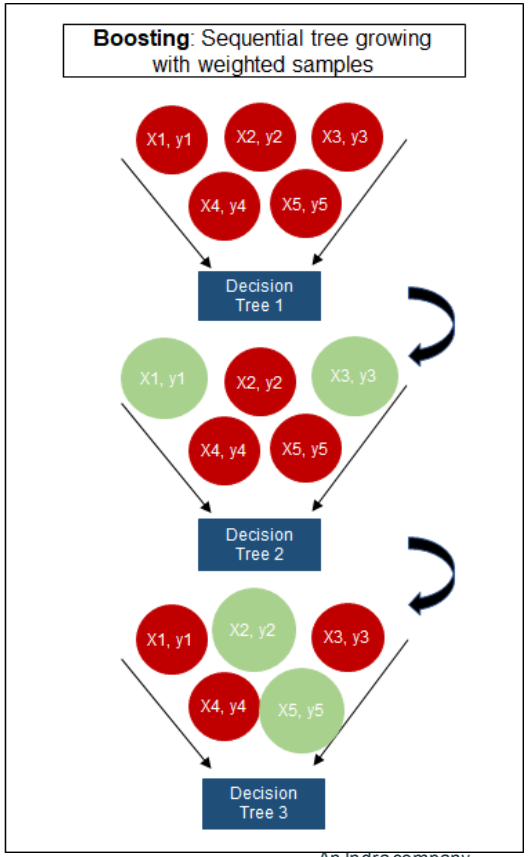
1. Maior quantidade de votos;

2. Usar os resultados de um algoritmo de uma árvore de decisão e, a partir dele, construir uma nova árvore que vai aprender com os erros da árvore anterior.

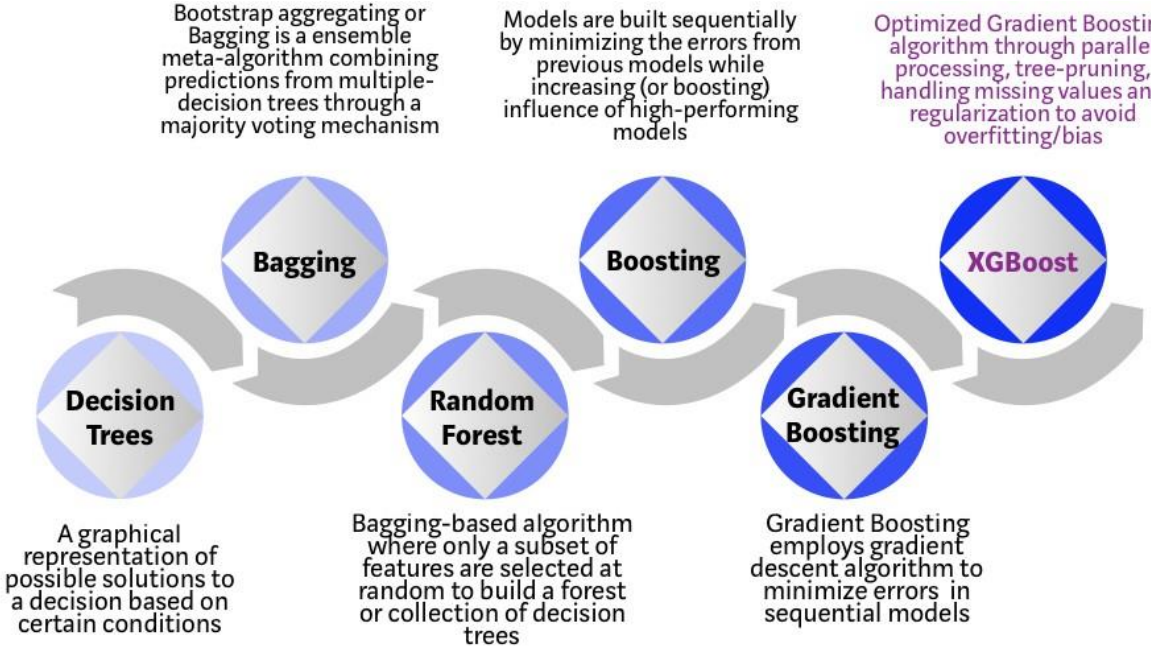
Métodos Ensemble - Funcionamento



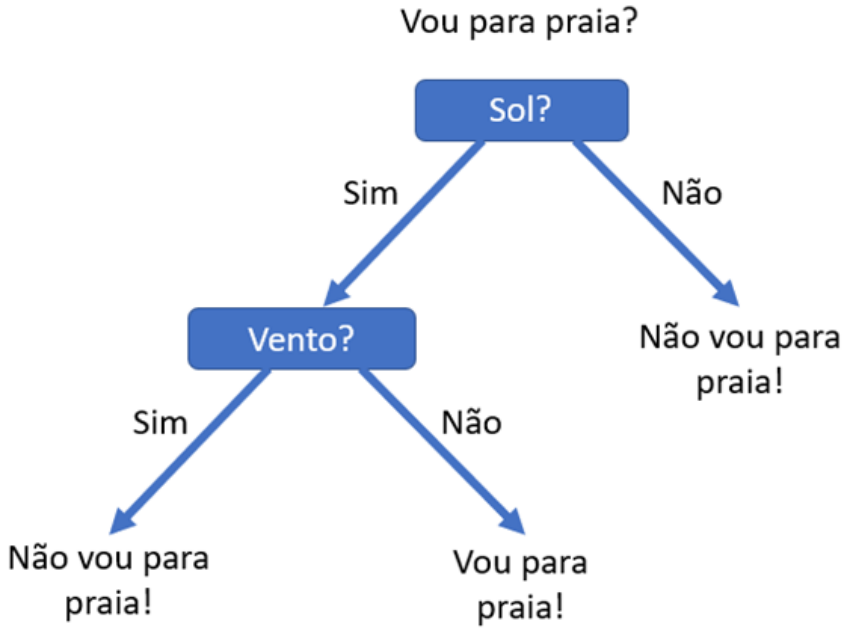




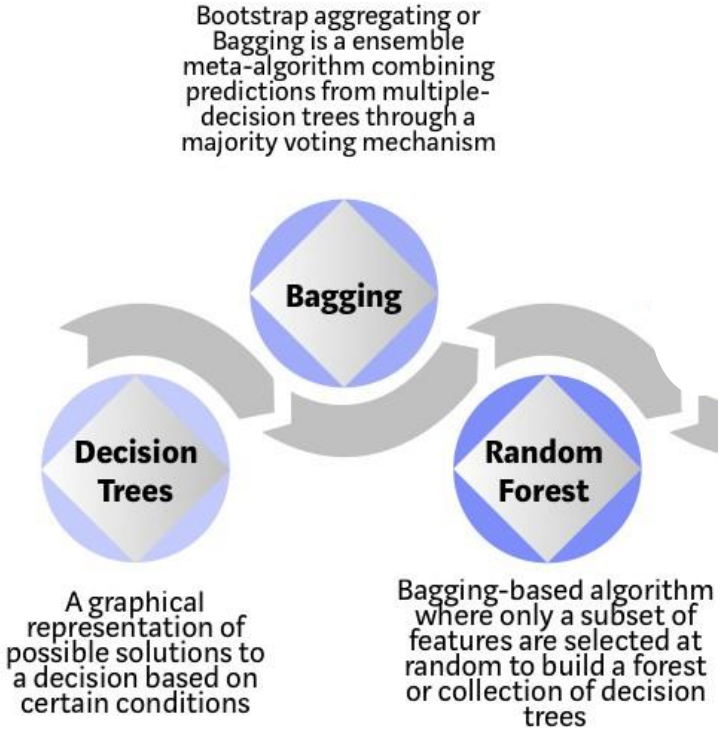
Métodos Ensemble - Evolução dos algoritmos de árvore



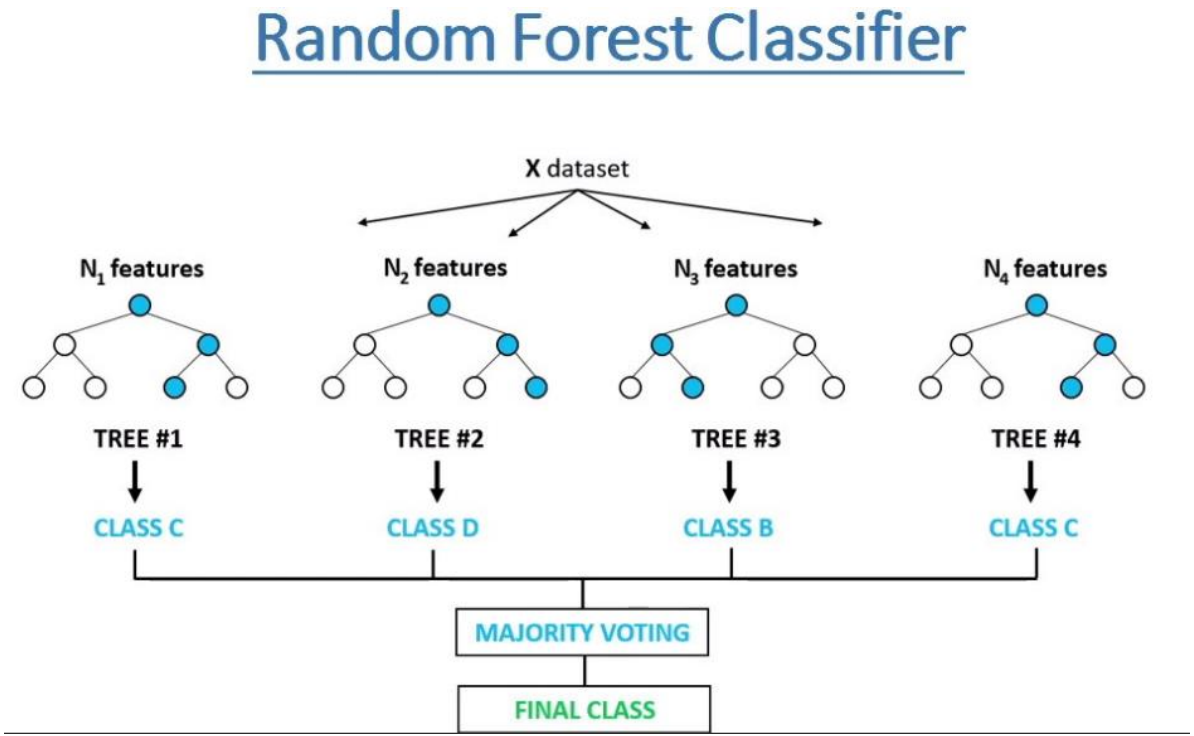
Árvore de Decisão



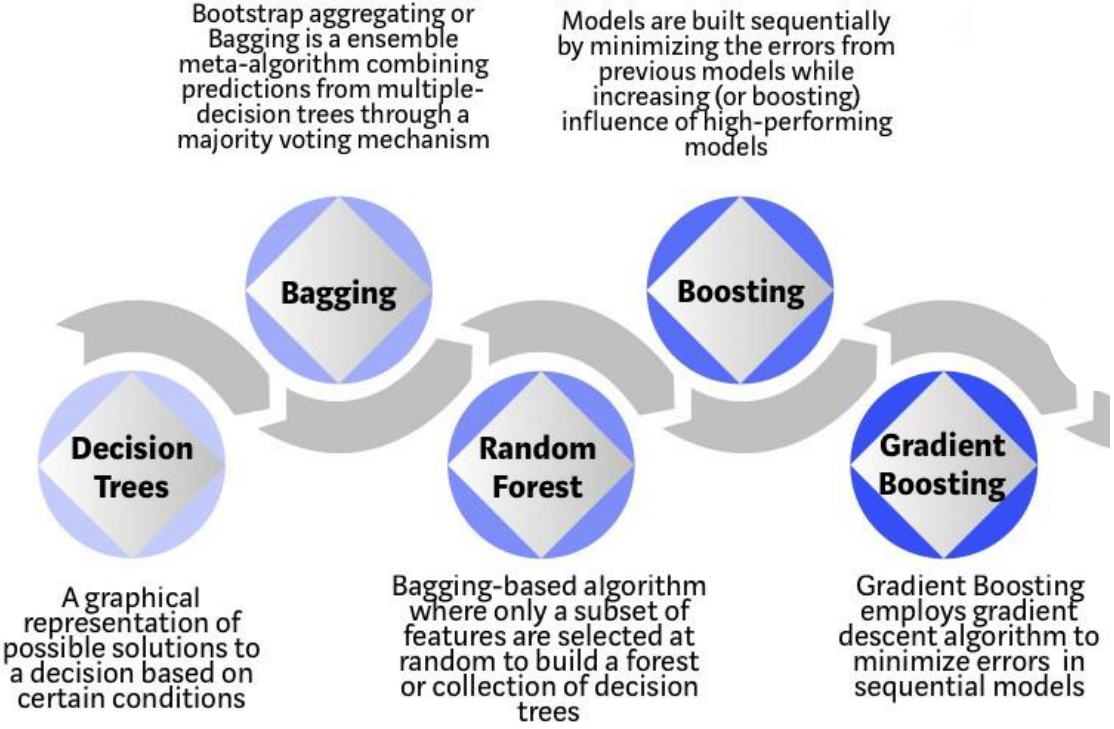
Métodos Ensemble - Evolução dos algoritmos de árvore



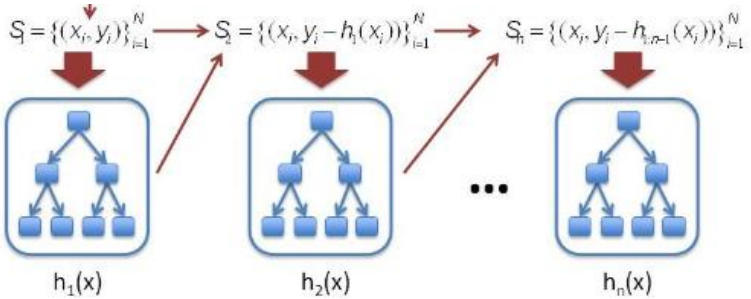
Random Forest

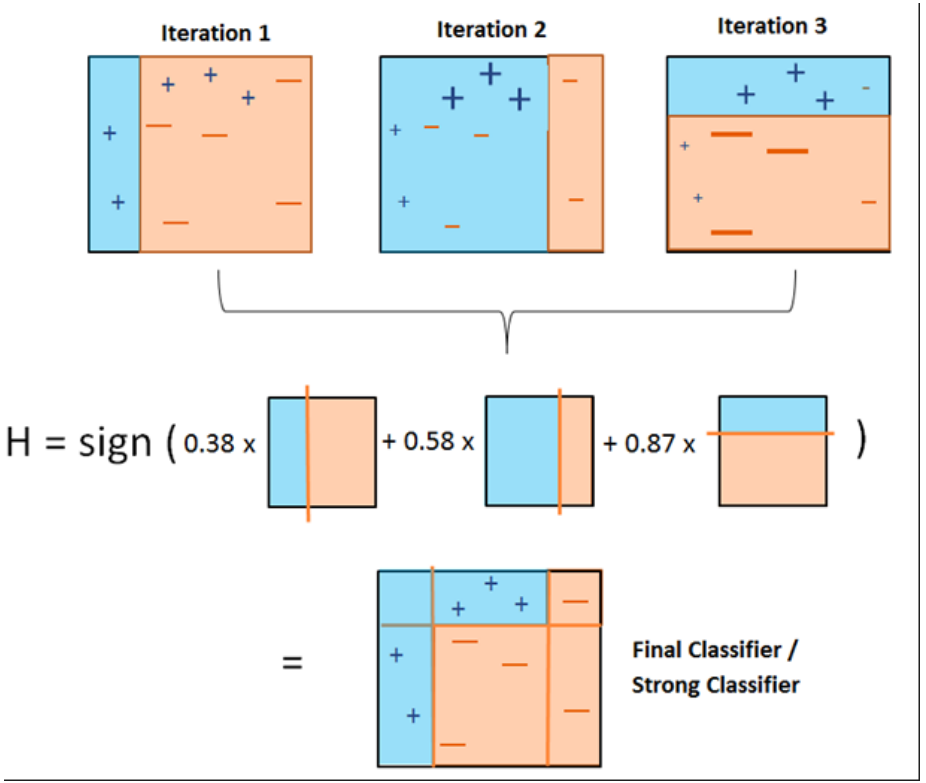


Métodos Ensemble - Evolução dos algoritmos de árvore

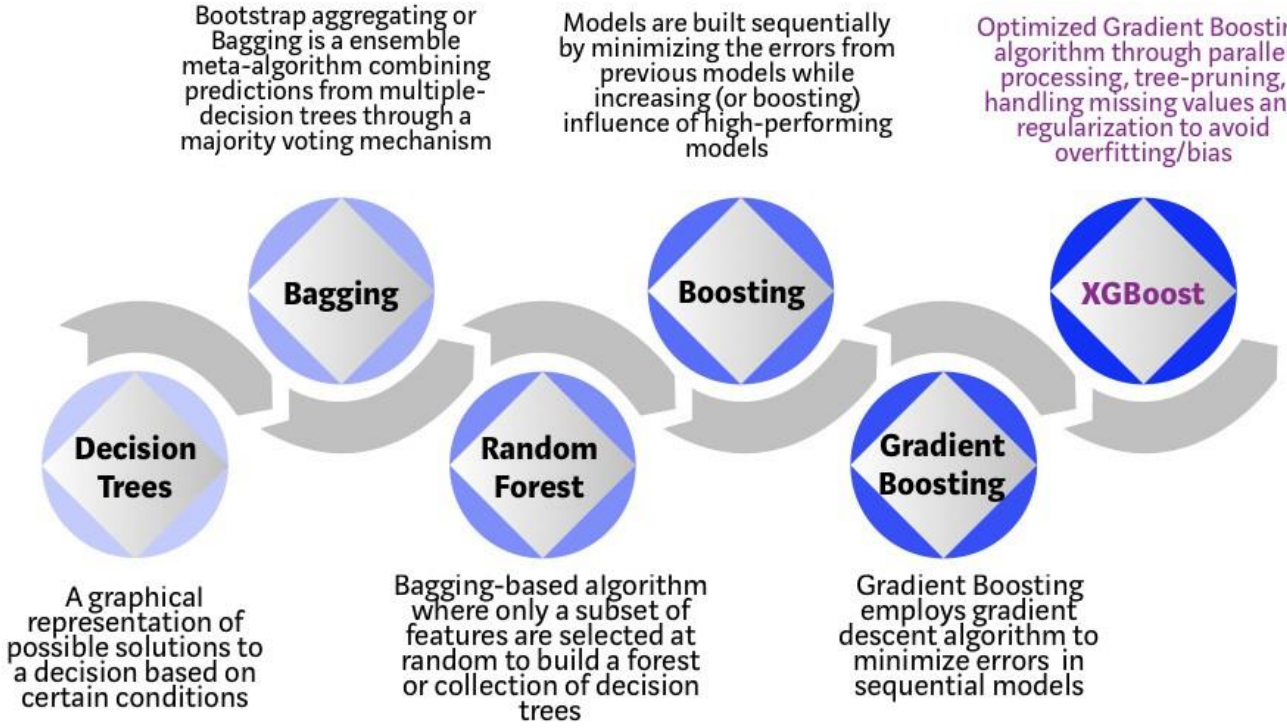


Gradient Boost

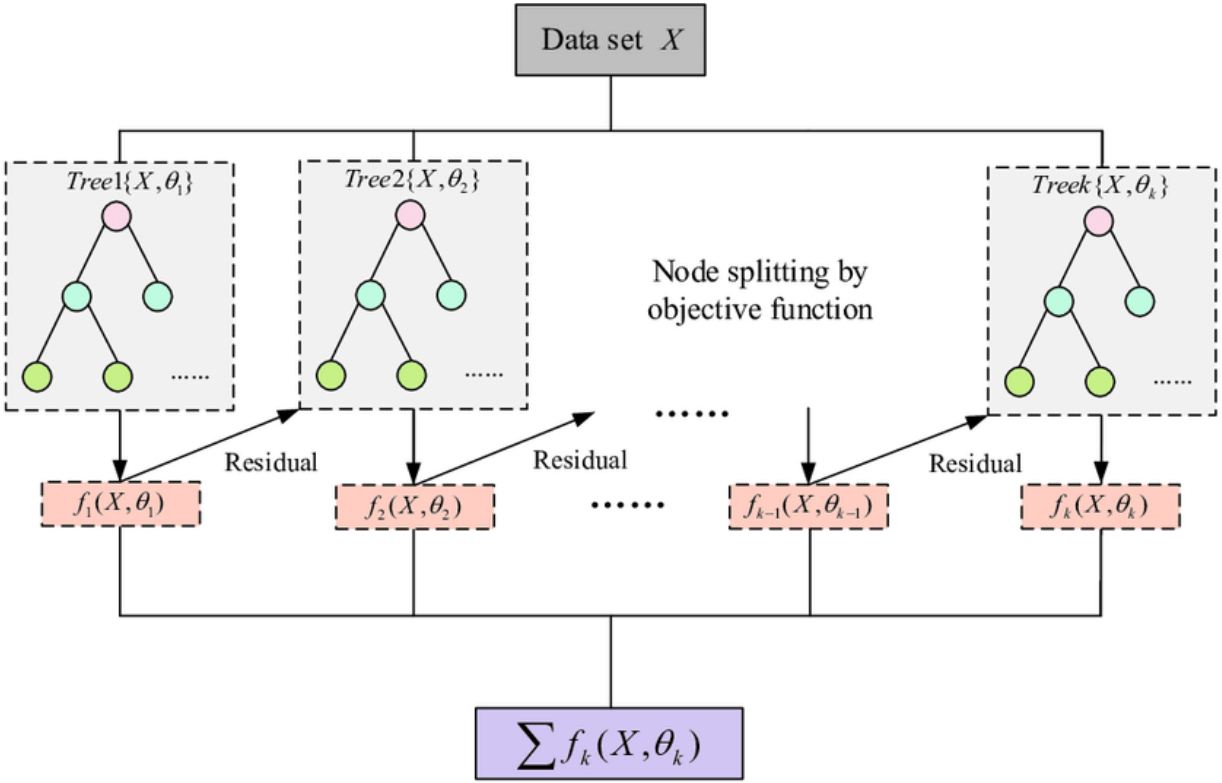




Métodos Ensemble - Evolução dos algoritmos de árvore



XGBoost


Data set X
$Tree1\{X,\theta_1\}$
$Tree2\{X,\theta_2\}$
$Treek\{X,\theta_k\}$
Node splitting by objective function
Residual
Residual
Residual
$f_1(X,\theta_1)$
$f_2(X,\theta_2)$
$f_{k-1}(X,\theta_{k-1})$
$f_k(X,\theta_k)$
$\sum f_k(X,\theta_k)$

Referências

Inteligência Artificial - Aulas de Inteligência Artificial (google.com)

ARIA - YouTube

O que são Métodos Ensemble e como eles funcionam? (didatica.tech)

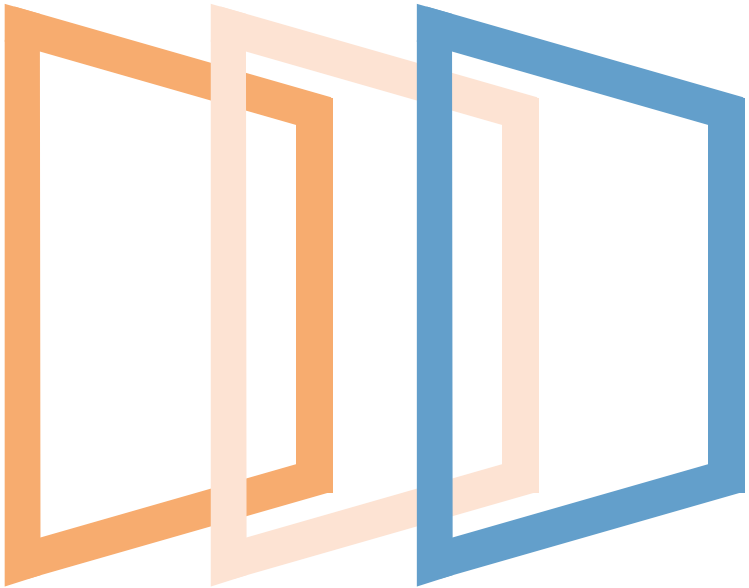
Aula-Ensemble-Learning.pdf (cefet-rj.br)

Desvendando IA e Machine Learning

Thaís Ratis

Diego Alexandre

Práticas Tecnológicas, 19.09.2023



minsait

An Indra company