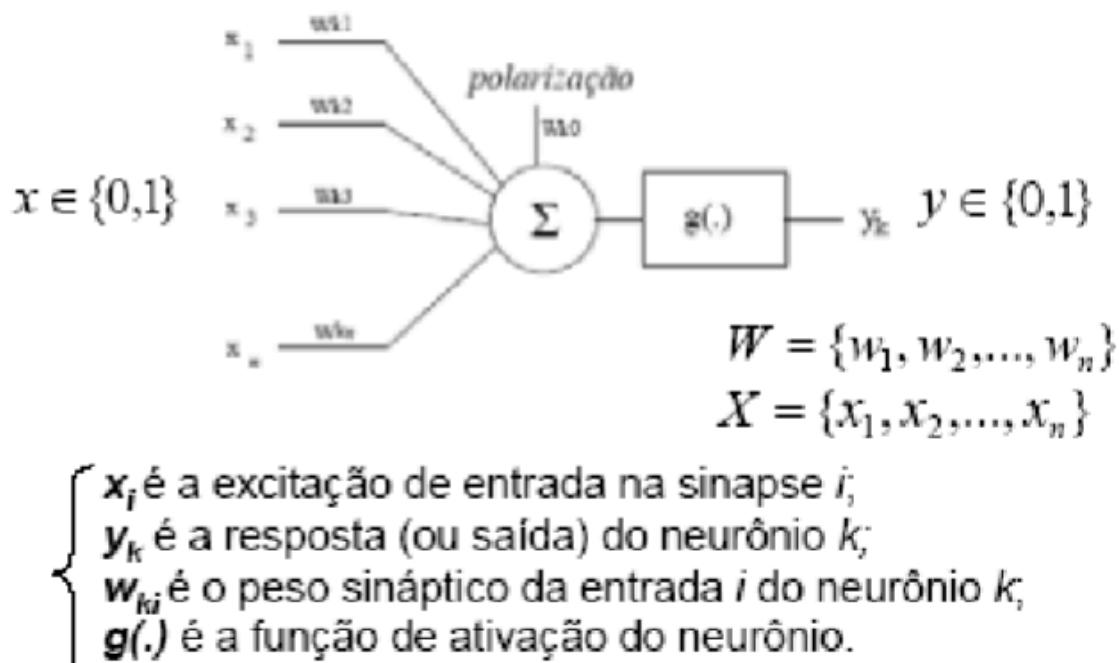


# INTELIGÊNCIA ARTIFICIAL

## Redes Neurais - 06/10

Estrutura do neurônio de McCulloch-Pitts:



A entrada é um conjunto de informações de um objeto analisado. O caso explicado na sala de aula foi de uma pessoa, então:

$x_1 = \text{Gênero . Wg}$

$x_2 = \text{Escolaridade . We}$

$x_3 = \text{Idade . Wi}$

$x_4 = \text{Nacionalidade . Wn}$

$x_{bias} = \text{Bias . Wbias}$

Peso sináptico é o peso atrelado a cada "sinapse" da entrada.

- O peso diz a importância dos pesos para o resultado final de uma análise.

O somatório do neurônio é o somatório do produtório.

- Soma o peso atrelado a entrada  $x$  com o valor da entrada de fato.

Existem uma entrada padrão chamada de polarização (BIAS) que, normalmente vai ser 1, mas o seu peso que vai fazer a diferença, ou seja:

- Se tivermos 10 entradas, teremos 11 informações → 10 entradas + 1 polarização.

A saída dessa análise é binária, logo:

- Temos uma função de ativação após o somatório do neurônio ser calculado.
- Essa função auxiliar ativa um neurônio com base em uma condição. Por exemplo:
  - Se o somatório for > que 0 a saída é 1.
  - Se o somatório for < que 0 a saída é 0.

## Perceptron:

Considerando a função AND:

- Ela é LINEARMENTE separável.
- Considerando que o peso de X é 0,2, de Y é -0,4 e do bias (valor 1) é 0,3:

X	Y	$X \wedge Y$	$\Sigma$	f
0	0	0	0,3	1
0	1	0		
1	0	0		
1	1	1		

$X \wedge Y \rightarrow$  É a saída esperada.

$\Sigma \rightarrow$  É o somatório do produtório - entradas vezes os pesos.

f → Saída encontrada

- Cálculos feitos:
  - Para  $\Sigma$ :  $X \cdot Wx + Y \cdot Wy + Bias \cdot Wbias$ 
    - No primeiro caso:  $0 \cdot 0,2 + 0 \cdot -0,4 + 1 \cdot 0,3 = 0,3$ .
    - No caso acima, pela função auxiliar - por ser maior que 0 - o resultado f = 1. O que está **ERRADO**.

- Após o cálculo acima, utilizamos a seguinte fórmula para ajustar esse erro:
    - $W_{t+1} = W_t + ta \cdot erro \cdot entrada$ 
      - $W_{t+1}$  → Peso novo, após o ajuste.
      - $W_t$  → Peso atual, sem o ajuste.
      - $ta$  → Taxa de aprendizado (Será bem pequena para que o ajuste aconteça devagar)
      - $erro$  → Saída esperada MENOS Saída encontrada.
      - $entrada$  → Entrada que estamos ajustando.
    - $W_{f1} \rightarrow$  primeira saída  $\rightarrow W_{f1} = 0,2 + 0,3 \cdot (0 - 1) \cdot 0$ 
      - $W_{f1} = 0,2 + 0 \rightarrow 0,2$ 
        - Atualizamos o peso de X:
      - Repetimos esse passo a passo para Y e para o BIAS.
      - Atualizamos eles, se necessário.
  - Em entradas que podem variar entre muitos valores - que são altos, por exemplo idade -, devemos fazer um "ajuste" nessas entradas para que os cálculos sejam feitos com valores na mesma escala. Isso busca evitar que uma entrada roube no resultado.
- 

## Redes Neurais - 08/10

XOR | XNOR  $\rightarrow$  Não podem ser resolvidas por um perceptron simples, pois não são lineares.

## Exercícios

Dado um perceptron simples de duas entradas e um bias, cujos pesos são  $w_1 = 0,5$ ,  $w_2 = 0,4$  e  $w_0 = -0,3$ , respectivamente, assinalar a resposta correta:

- (a) o perceptron realiza a função NOR
- (b) o perceptron realiza a função AND
- (c) o perceptron realiza a função OR
- (d) o perceptron realiza a função XOR
- (e) nenhuma das alternativas

R: (c) o perceptron realiza a função OR

$x$	$y$	$x \wedge y$	$\Sigma$	$f$	O limiar é: $> 0 = 1   < 0 = 0$
0	0	0	-0,3	0	$0 \times 0,5 + 0 \times 0,4 + 1 \times -0,3 = -0,3 \rightarrow 0$
0	1	0	0,1	1	$0 \times 0,5 + 1 \times 0,4 + 1 \times -0,3 = 0,1 \rightarrow 1$
1	0	0	0,2	1	$1 \times 0,5 + 0 \times 0,4 + 1 \times -0,3 = 0,2 \rightarrow 1$
1	1	1	0,6	1	$1 \times 0,5 + 1 \times 0,4 + 1 \times -0,3 = 0,6 \rightarrow 1$

## MLP - Multilayer Perceptron

Algoritmo Backpropagation:

- Fase 1 - Propagação (fase forward):
  - Os perceptrons recebem cada entrada e realizam o cálculo do **somatório do produtório** de suas entradas e respectivos pesos - **NÃO** podemos esquecer do BIAS.
  - A função auxiliar **NÃO É MAIS A MESMA** → Agora é utilizado uma função **NÃO** linear.
  - As saídas de cada perceptron são passadas para os próximos perceptrons e o processo se repete até a camada da saída final, onde encontramos a resposta e onde o erro é calculado.
- Fase 2 - Retropropagação (fase backward):
  - Ao encontrar um erro, precisamos estimar a contribuição de cada perceptron para esse resultado.

- Começamos a estimar na camada de saída.
  - Se for trivial fazemos: **derivada(função de ativação) \* (desejado - real)**.
  - Se não for trivial, utilizamos a solução do próximo item.
- Essa é a fórmula mais usual que utilizamos para encontrar esse erro:

Ao invés de usar a diferença absoluta, pode-se utilizar também o **erro médio quadrático (Mean Square Error – MSE) ou Root Mean Square Error – RMSE**

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \quad \text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (S_i - O_i)^2}$$

## Técnicas de Agrupamento - 27/10

Aprendizado Indutivo → Cria uma função e com base nela classifica os testes.

- Descritivo:
  - Associação.
  - Agrupamento.
- Preditivo:
  - Classificação.
  - Regressão.
  - O Random Forest, Backpropagation e Cart resolvem esse problema.

Tipos de agrupamento → Agrupamentos são formados com base em similaridades

- Particional:
  - K-Means → Vamos ver o que cada indivíduo está em um único grupo.
    - Muito bom quando temos uma clara divisão de grupos.

- Complexidade:  $O(n*k*I*d)$ 
  - N = Número de instâncias.
  - K = Número de clusters (grupos).
  - I = Número de iterações.
  - D = Número de atributos.
- Existe o **nebuloso**, onde um indivíduo do teste pode participar de diversos grupos.
- Distâncias intra-clusters são minimizadas, já as entre clusters são maximizadas.
  - Distância de Manhattan.
- 1a interação:
  - Chuta os Ks (instâncias).
  - Calcula a distância entre cada ponto com os Ks escolhidos (centróides).
- 2a interação:
  - Calcula a média entre as distâncias das instâncias do grupo para cada atributo. Os centróides antigos participam do cálculo dessa média.
  - Marca os novos centróides como + em cada grupo.
  - Calcula a distância entre todas as instâncias aos novos centróides, criando um **NOVO** grupo.
- 3a interação e seguintes:
  - Refaz o passo 2.
  - O critério de parada é quando X% que estão mudando de grupo. "Até que somente 1% dos objetos mudam de clusters", onde X = 1.
- Vamos pensar que utilizamos o K-Means e encontramos 3 grupos:
  - De fato essa base tem 3 grupos?
  - Não! Afinal, se ele encontrou 3, é porque foi definido K = 3.

- Uma solução é utilizar esse algoritmo diversas vezes com diversos Ks, calculando a qualidade de cada uma dessas “respostas” e utilizando a melhor.
- Como posso dar um nome ao grupo?
  - Roda uma árvore, gerando regras que ajudam a dar uma ideia sobre o nome do grupo.
- Hierárquico:
- Por densidade:
  - DB Scan.
- Baseado em modelo:
  - SOM.

## 12/11

**Dados inconsistentes** → São aqueles dados que possuem valores conflitantes em seus atributos.

- Duas instâncias com todos atributos iguais, mas com o resultado diferente.
  - Exemplo: Dois pacientes com todos os seus sintomas iguais, mas com doenças diagnosticadas diferentes.
- Quanto mais reduzimos a quantidade de atributos, maior a chance de termos dados inconsistentes.
- Retiramos todos os dados inconsistentes. Se temos duas instâncias em conflito, retiramos as duas.

**Dados redundantes** → São aqueles dados que possuem valores semelhantes em seus atributos.

- Uma instância é redundante quando ela é muito semelhante a uma outra instância do mesmo conjunto de dados.
- Podemos se referir a **instâncias** ou **atributos**.
  - Dados redundantes em relação ao atributo X.
  - Dados redundantes em relação a instâncias completas.

- Exemplo: Pacientes com todos ou quase todos sintomas iguais, com diagnósticos iguais.
- Retiramos todos os dados redundantes, exceto uma instância deles.
- A redundância de um atributo está relacionada à sua **correlação** com um ou mais atributos do conjunto de dados.
  - Dois atributos estão correlacionados quando apresentam um perfil de variação semelhante.
    - Exemplo: Alunos que tem mais **horas de estudo** possuem maiores **notas**. (Atributos de entrada!)
    - Se a correlação ocorrer entre um **atributo de entrada** e um **atributo de saída**, o atributo de entrada terá uma **grande** influência na predição do valor do atributo de classe (saída).
      - Exemplo: Uma instância com o atributo de entrada “quebrou regra do tipo” está **muito** correlacionada ao atributo de saída “quebrou regra?”. Afinal, ter um valor positivo no de entrada, diz 100% sobre o valor do de saída.
- **Dados ausentes** → O ideal é **NÃO** ter dado ausente, mas na vida real **quase sempre** temos eles.
  - **Próxima aula.**

## 17/11

### O que fazer com dados ausentes?

- Podemos eliminar as instâncias com dados ausentes, mas somente quando realmente é possível removê-la. (**Não é prática e não acontece muito no mundo real**)
  - Ao remover, podemos ficar com um conjunto muito pequeno, o que torna ruim essa opção.
- Podemos definir manualmente em alguns casos específicos. (**Não é prática e não acontece muito no mundo real**)
  - Não acontece muito, pois nem sempre é possível preencher os dados de um conjunto. “Forjá-los” nem sempre manterá a base interessante para trabalho.

- Podemos utilizar algum método ou heurística para automaticamente definir valores para os campos ausentes, mas isso pode deixar a nossa base distorcida. (**Tem problemas quanto a distorção da variação da base, pois, ao utilizar a média e mediana, a variável é “achatada”, colocando instâncias em lugares indevidos**)
- Utilizar algoritmos de AM (Aprendizado de Máquina) que lidam internamente com dados ausentes.
  - Missforest
  - KNNimputer
- **Balanceamento** → É necessário balancer base antes de utilizá-las, pois pode acontecer de que uma classe seja completamente entendida, mas uma segunda classe tenha 100% de erro.
  - Se essa segunda classe tiver poucas instâncias comparadas a primeira classe, a acurácia geral vai ficar muito boa, mas esse modelo não vai ser útil pra nada.
- **Como resolver?**
  - Redefinir o tamanho do conjunto de dados.
    - Adicionar instâncias à classe minoritária. (Métodos **oversampling**)
      - Existe o risco de adicionar instâncias representem situações que nunca ocorrerão, induzindo um modelo inadequado.
      - Pode ocorrer o problema de **overfitting**, em que o modelo é superajustado em relação aos dados de treinamento.
      - Método **SMOTE**.
    - Remover instâncias da classe majoritária. (Métodos **undersampling**)
      - Ao remover essas instâncias da base da classe majoritária, é possível que dados de grande importância sejam perdidos, gerando um modelo errado.
      - Pode ocorrer o problema de **underfitting**, em que o modelo é superajustado em relação aos dados de treinamento.
      - Método **RandomUnderSampler**.

- **Under-over** → Reduzimos da majoritária e acrescentamos na minoritária, encontrando as duas classes no “meio do caminho”.
- Utilizar diferentes custos de classificação para as diferentes classes.
- Induzir um modelo para uma classe.

## 19/11

- Passo a passo para o pré-processamento:
  1. **Visualização** de dados (Seleção da base e compreensão da base):
    - a. Nominal Normal (Até 10 “opções” nos atributos) → Barras
    - b. Nominal de Alta Cardinalidade (Cidades, Países) → Mapa
    - c. Numérica → Histograma
  2. **Seleção** dos atributos:
    - a. Observamos a **Correlação** de atributos → Muitos atributos correlacionados não são necessários, como Data de Nascimento e Idade. Podemos remover alguns para simplificar o processo.
    - b. Removemos aqueles atributos que não são relevantes para o objetivo, seja ele classificação, agrupamento, etc.
    - c. Seleccionamos aqueles atributos que são relevantes, seja de forma empírica (a partir de algum estudo na área, conhecimento de especialistas, a própria visualização dos dados, etc) ou utilizando algum método computacional, como a entropia.
  3. **Codificação:**
    - a. Após codificar, é bom “olhar pra trás” para a etapa de visualização, pois é possível ter prejudicado a qualidade da nossa base.
    - b. Label encoder.
    - c. One hot encoder.
    - d. Podemos imputar a discretização nesse momento.
  4. Eliminação de **Inconsistências e Redundâncias**.
  5. Eliminação de **Outlier**:
    - a. Boxplot ou outros métodos.

- b. Atenção para os outliers extremos quando existem muitos outliers na base.

## 6. TRAIN TEST SPLIT:

- a. Divide a base para obter o conjunto de treino e o conjunto de teste.
- b. X% para teste.

## 7. Imputação de dado ausente → Fazemos no conjunto de treino e no teste:

- a. Utilizamos machine learning.

## 8. Normalização/Padronização → Fazemos no conjunto de treino e no teste.

## 9. Balanceamento → Fazemos apenas no conjunto de treino:

- a. Sem balanceamento.
- b. Com over.
- c. Com under.
- d. Com over-under.

# 24/11

## Métodos de Busca:

- Busca Cega (Não informada):
  - Não usam nenhuma função de avaliação.
  - Largura.
  - Profundidade.
  - Dijkstra
- Busca Informada (Busca heurística):
  - A\*.
  - Gulosa.

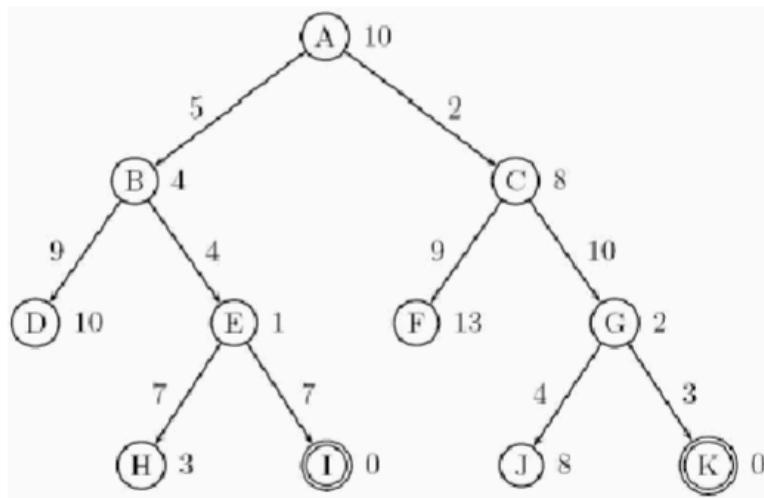
Observações: **PESQUISAR MELHOR!!!**

- Método completo = se tiver uma solução, ele garante encontrá-la.
  - Largura: só é completo se o fator de ramificação for finito.

- Profundidade: só é completo se não existir ciclo (Não volta em estados já visitados).
- Dijkstra: só é completo se o fator de ramificação for finito.  $f(n) = g(n)$
- Busca Gulosa: não é completo.  $f(n) = h(n)$
- A\*:  $f(n) = g(n) + h(n)$
- Método ótimo = encontra a melhor solução.
  - Largura: só é ótimo para grafos com pesos de arestas iguais.
  - Profundidade: **não** garante solução ótima de jeito algum.
  - Dijkstra: só é ótimo para grafos sem peso negativo.
  - Busca Gulosa: não garante ser ótimo.
  - A\*: só é ótimo se a heurística for admissível (ela é menor que o valor real).  $f(n) = g(n) + h(n)$

## 26/11

Lista 10 →



Verificar a ordem de visitação dos seguintes algoritmos:

**Observação:** Paramos no I ou no K, mas esses caminhos passam nos dois:

- Ov(largura): A B C D E F G H I J K
- Ov(profundidade): A B D E H I C F G J K

- Ov(dijkstra ou custo uniforme): A C B E F G D **K**
- Ov(gulosa): A B E **I** H C G **K**
- Ov(A<sup>\*</sup>): A B C E G **K I**

Julgue os itens a seguir, relativos a métodos de busca com informação (busca heurística) e sem informação (busca cega), aplicados a problemas em que todas as ações têm o mesmo custo, o grafo de busca tem fator de ramificação finito e as ações não retornam a estados já visitados.

- A primeira solução encontrada pela estratégia de busca em largura é a solução ótima.
- A primeira solução encontrada pela estratégia de busca em profundidade é a solução ótima.
- As estratégias de busca com informação usam funções heurísticas que, quando bem definidas, permitem melhorar a eficiência da busca.
- A estratégia de busca gulosa é eficiente porque expande apenas os nós que estão no caminho da solução.

Estão certos apenas os itens

- I e II.
- I e III.
- I e IV.
- II e IV.
- III e IV.

I - Está certa, pois o custo é o mesmo. Então, independente do caminho que a busca por largura fizer, será uma solução ótima.

II - Não está certa, pois profundidade **não** garante solução ótima.

III - Está certa, pois buscas com informação carregam a heurística (previsões) para melhorar a eficiência da busca.

IV - Falso, pois a busca gulosa pode **SIM** abrir nós fora do caminho da solução.

[TE] Considerando que  $h(n)$  é o custo estimado do nó  $n$  até o objetivo, em relação à busca informada, pode-se afirmar que

- a busca *gulosa* minimiza  $h(n)$ .
- a busca  $A^*$  minimiza  $h(n)$ .
- a busca de custo uniforme minimiza  $h(n)$ .
- a busca *gulosa* minimiza  $h(n)$  somente se a heurística for admissível.
- a busca  $A^*$  minimiza  $h(n)$  somente se a heurística for admissível.

A - Verdadeiro.

B - Falso, pois A\* usa  $h(n) + g(n)$

C - Falso, o Dijkstra nem usa  $h(n)$ .

D - Falso, pois ela minimiza o  $h(n)$  e ponto. Ela não se importa se a heurística é admissível.

E - Falso, pois ela não minimiza o  $h(n)$  e sim o  $h(n) + g(n)$ . Outro erro é que, independente se a heurística é admissível ou não, o A\* continua rodando. A heurística admissível apenas garante que a solução será ótima, mas se não for o A\* ainda roda.

Considere  $h(x)$  como uma função heurística que define a distância de  $x$  até a meta; considere ainda  $h^r(x)$  como a distância real de  $x$  até a meta.  $h(x)$  é dita admissível se e somente se:

- (a)  $\exists n \ h(n) \leq h^r(n)$ .
- (b)  $\forall n \ h(n) \leq h^r(n)$ .
- (c)  $\forall n \ h(n) > h^r(n)$ .
- (d)  $\exists n \ h(n) > h^r(n)$ .
- (e)  $\exists n \ h(n) < h^r(n)$ .

Heurística admissível  $\rightarrow$  Para todo nó o valor da heurística deve ser menor ou igual ao custo real.

Utilizamos o símbolo de **para todos** para representar isso.

A resposta é a letra **B**.

Até aqui só tivemos problemas sem interação com outro agente.

Para casos com dois agentes, temos métodos de busca para casos com outro agente. O minimax é um deles.

O max sempre começa jogando primeiro, em seguida o min.

Jogo da velha pode ser resolvido com esse algoritmo.

Nesse algoritmo a melhor opção é sempre escolhida.

A função heurística é a estimativa do max - a estimativa do min.