



**Inteligência:** capacidade de resolver problemas novos através do uso do conhecimento

**Inteligência Artificial:** ciência preocupada com a construção de máquinas inteligentes, i. e., máquinas que realizam tarefas que, quando realizadas por humanos, requerem inteligência

**IA Fraca / Estreita:** IA focada numa só tarefa estreita

**IA Forte / Générica:** inteligência de uma máquina que pode realizar com sucesso qualquer tarefa intelectual que um ser humano consegue

**Agente:** Sistema Computacional, situado num dado ambiente, que tem a capacidade de percecionar o ambiente usando sensores e atuar, de forma autónoma, nesse ambiente usando os seus actuadores para cumprir uma dada função

**Sistema Multi-Agentes:** os agentes exibem comportamento autónomo e interagem com outros agentes do sistema

**Machine Learning:** área da IA que dá aos sistemas de computadores a capacidade de "aprender" (aumentar progressivamente o desempenho numa tarefa específica) a partir de dados / resultados das suas ações, sem serem explicitamente programados

**ML** { 1. Supervisionado: dirigido pelas tarefas (prever próximo valor)  
2. Não Supervisionado: dirigido pelos dados (identificar agrupamentos)  
3. Reforço: aprender com os erros

**Processamento de Linguagem Natural:** área de ciência dos computadores, IA e linguística computacional preocupada com as interações entre computadores e linguagens (naturais) humanas e, em particular, preocupada com programar computadores para processar com sucesso grandes corpos da linguagem natural

- PLN
- 1. Tradução Máquina
  - 2. Análise de Sentimentos
  - 3. Texto - Para - Fala / Fala - Para - Texto

## Agentes Inteligentes

**Agente:** Sistema Computacional, situado num dado ambiente, que tem a capacidade de percecionar o ambiente usando sensores e atuar, de forma autónoma, nesse ambiente usando os seus atuadores para cumprir uma dada função

**Agente:** qualquer coisa que pode ser vista como a percecionar o seu ambiente através de sensores e a atuar nesse ambiente através de atuadores

**Agente Facional:** aquele que faz a coisa certa  
aquela que lhe traz mais sucesso

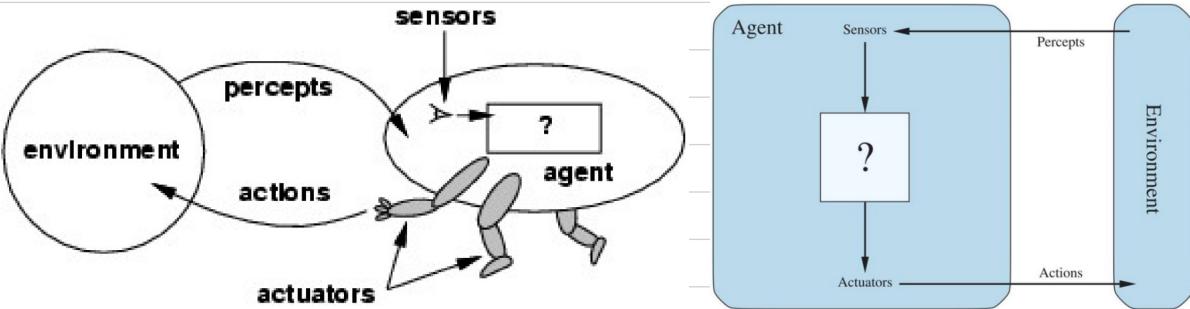
**Agente Facional Ideal:** para cada sequência de percepções, faz a ação que é expectável que maximize a sua medida de desempenho (sucesso), dado o conhecimento que tem

Função do Agente: descrição matemática abstrata que mapeia o histórico de percepções em ações -  $[f: P^* \rightarrow A]$

Programa do Agente: implementação específica que corre num sistema/arquitetura físico para produzir  $f$

$$\text{Agente} = \text{Arquitetura} + \text{Programa}$$

A tarefa da IA é desenhar o programa e a arquitetura de um agente



- Um agente mostra um comportamento - ação realizada dadas percepções
- Um agente decide o que fazer - reage a sensores e controla atuadores

Medida de Performance  
Ambiente  
Atuadores  
Sensores

- Propriedades do Ambiente**
- 1. Observável VS Inacessível
    - ↳ os sensores detectam tudo o que é relevante
  - 2. Determinista VS Não Determinista
    - ↳ o estado seguinte é determinado pelo anterior e pelas ações
  - 3. Episódico VS Não Episódico
    - ↳ dividido em episódios
  - 4. Dinâmico VS Estatico
    - ↳ muda enquanto o agente está a pensar
  - 5. Discreto VS Contínuo
    - ↳ número finito de percepções e ações
  - 6. Agente Único VS Multi-Agente
    - ↳ cooperativo ou competitivo

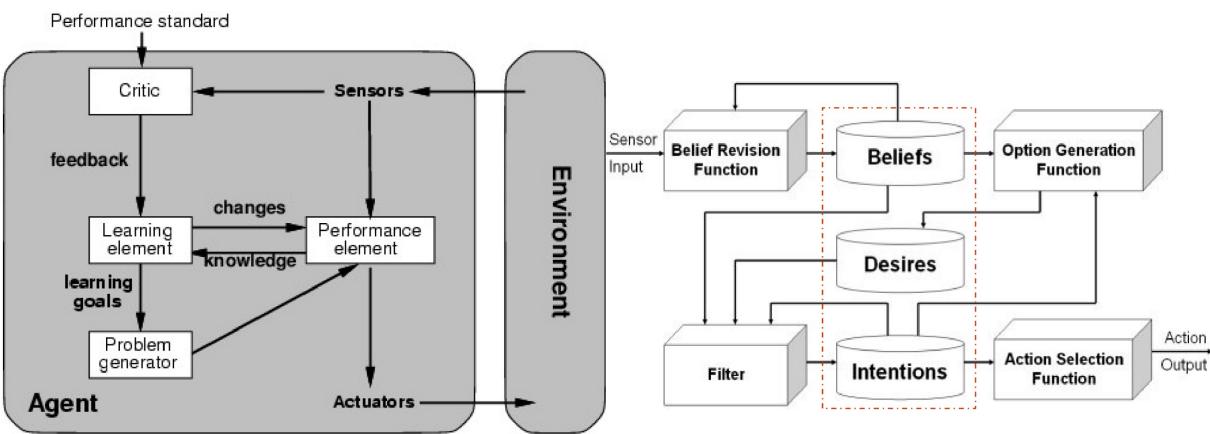
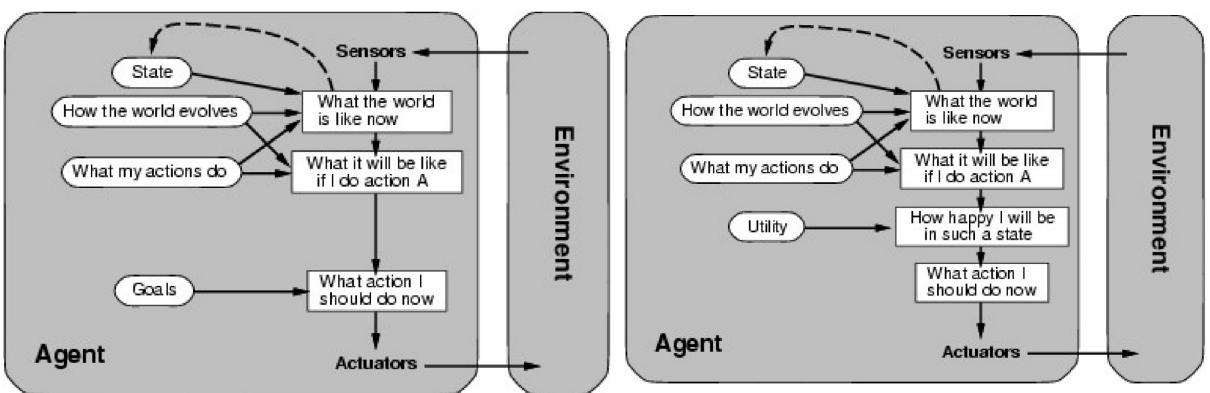
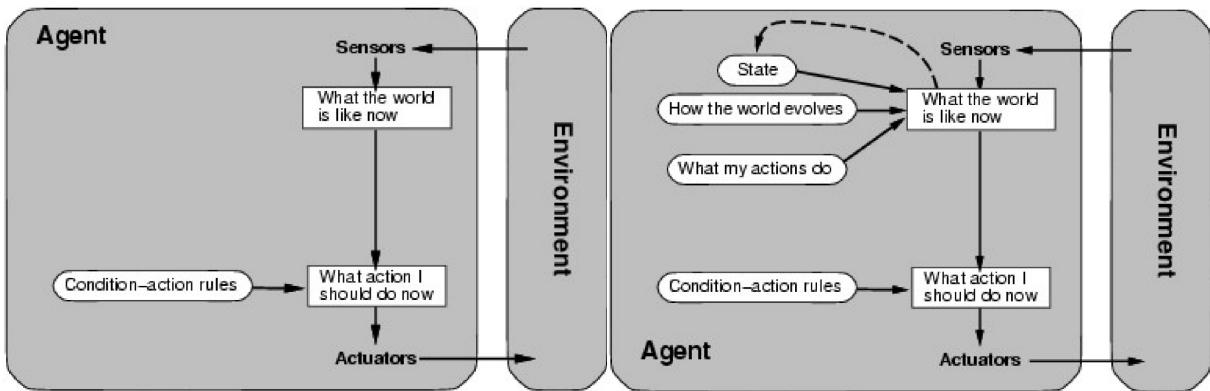
- O ambiente mais simples é totalmente observável/observável, determinista, episódico, estático, discreto e de agente único
- O ambiente mais real é parcialmente observável/inacessível, não determinista, sequencial (não episódico), dinâmico, contínuo e multi-agente

- Tipos Básicos de Agentes**
- 1. de Refleco Simples: baseados em tabelas de regras condição - ação
  - 2. com Representação do Mundo: mantêm um estado interno
  - 3. Baseados em Objetivos: com descrição do estado do mundo e do objetivo
  - 4. Baseados em Utilidade: maximizam o estado atual para um valor
  - 5. em Aprendizagem: com elementos de aprendizagem e de auto-aprendizagem
  - 6. BDI: crenças (informação), desejos (motivação), intenções (deliberação)

**B:** informação que o agente tem sobre o mundo

**D:** todos os estados possíveis que o agente pode querer alcançar

**I:** estados para os quais o agente decide trabalhar



# Pesquisa

Resolução de Problemas: Como é que um agente pode ação, definir objetivos e considerar sequências possíveis de ações para alcançar esses objetivos?

Agente de Resolução de Problemas: Tenta encontrar a sequência de ações que leva a um estado desejável SOLUÇÃO

## Formulação do Problema?

1. Quais são as ações possíveis? Quais os seus efeitos nos estados do mundo?
2. Quais são os estados possíveis? Como representá-los?
3. Como avaliar estados?

Formulação → Pesquisa → Execução

## Problema:

1. conjunto  $S$  de estados
2. estado inicial  $I \in S$
3. relação de transição no espaço de estados
4. conjunto de estados finais/objetivo,  $O \subseteq S$

• O problema pode ser representado por um grafo cujos nós representam estados e os arcos/conexões os pares da relação de transição

• O problema pode ser resolvido através da pesquisa de um caminho entre o estado inicial e um estado objetivo

## Formulação do Problema:

1. Representação dos Estados
2. Estado Inicial (Atual)
3. Teste Objetivo - define os estados desejados
4. Operadores - nome, pré-condição, efeito, custo
5. Custo da Solução

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state  $\leftarrow$  UPDATE-STATE(state, percept)
    if seq is empty then do
        goal  $\leftarrow$  FORMULATE-GOAL(state)
        problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
        seq  $\leftarrow$  SEARCH(problem)
    action  $\leftarrow$  FIRST(seq)
    seq  $\leftarrow$  REST(seq)
    return action
```

Qual é o conhecimento do agente sobre o estado do mundo e as suas ações?

*Típos de Problemas* {  
1. Estado Único: ambiente determinista e acessível  
2. Estado Múltiplo: ambiente determinista e inacessível  
3. Contingência: ambiente não determinista e inacessível  
4. Exploração: espaço de estados desconhecido

## Pesquisa de Soluções:

1. Começar com o estado inicial
2. Executar o teste objetivo
3. Se a solução não foi encontrada, usar os operadores para expandir o estado atual e gerar os novos estados sucessores
4. Executar o teste objetivo
5. Se a solução não foi encontrada, escolher que estado expandir a seguir (estratégia de expansão) e expandi-lo
6. Voltar a 3

Pesquisa por Melhor: escolher um nó  $n$  com o valor mínimo de uma função de avaliação  $f(n)$

```
function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value  $node$ 
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if  $s$  is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

Nó da Árvore

<u>1. Estado</u> :	estado correspondente
<u>2. Pai</u> :	nó que lhe deu origem
<u>3. Ação / Operador</u> :	aplicado para gerar
<u>4. Custo do Caminho</u> :	deixa o nó inicial
<u>5. Profundidade</u>	

Fronteira: conjunto de nós à espera de serem expandidos

Estratégia de Pesquisa: definida pela ordem de expansão dos nós

Como avaliar uma estratégia de pesquisa?

1. Completa
2. Otimalidade
3. Complexidade Temporal
4. Complexidade Espacial

**Pesquisa em Largura:** expandir sempre um dos nós menos profundos -  $T = S = O(b^d)$

**Pesquisa de Custo Uniforme:** expandir sempre o nó com o menor custo da fronteira

**Pesquisa em Profundidade:** expandir sempre um dos nós mais profundos -  $T = O(b^m)$ ,  $S = O(b^m)$

**Pesquisa de Aprofundamento Iterativo:** realizar pesquisa em profundidade limitada, iterativamente, aumentando sempre o limite de profundidade -  $T = O(b^d)$ ,  $S = O(bd)$

É o melhor método de Pesquisa Desinformada quando o espaço de estados de pesquisa não cabe em memória e a profundidade da solução é desconhecida

**Pesquisa Bidirecional:** corre pesquisa para a frente a partir do estado inicial e pesquisa para trás através do estado final, simultaneamente

$b$ : fator máximo de ramificação da árvore de pesquisa

$d$ : profundidade da solução de menor custo

$m$ : profundidade máxima do espaço de estados

$l$ : limite da profundidade da pesquisa

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>1</sup>	Yes <sup>1,2</sup>	No	No	Yes <sup>1</sup>	Yes <sup>1,4</sup>
Optimal cost?	Yes <sup>3</sup>	Yes	No	No	Yes <sup>3</sup>	Yes <sup>3,4</sup>
Time	$O(b^d)$	$O(b^{1+[C^*/\epsilon]})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+[C^*/\epsilon]})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$

Não detectar estados repetidos pode tornar exponencial um problema linear

1. Não voltar ao estado anterior
2. Não criar ciclos
3. Não usar estados repetidos

Pesquisa Informada / Inteligente / Heurística: usa informação do problema

$h(n)$ : custo estimado do caminho mais curto do estado  $n$  até ao objetivo

$g(n)$ : custo total até ao estado  $n$

$f(n)$ : custo estimado da solução de menor custo que passa no estado  $n$

Pesquisa Ambiciosa: expande o nó que parece mais perto da solução

$$f(n) = h(n) - \text{SUB-ÓTIMO, INCOMPLETO, } T=S=O(b^m)$$

Pesquisa A\*: combina as pesquisas ambiciosa e uniforme para minimizar a soma do caminho já percorrido com o mínimo esperado até à solução

$$f(n) = g(n) + h(n) - \text{ÓTIMO, COMPLETO, } T=\text{exf., } S=n$$

Pesquisa A\* Pesada: permite usar uma heurística inadmissível (que pode sobreestimar), perdendo optimilidade, mas ganhando eficiência  $f(n) = g(n) + W h(n)$

## RESUMO

Pesquisa A\*:  $f(n) = g(n) + h(n)$ ,  $W=1$

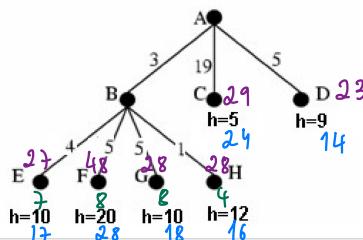
Pesquisa de Custo Uniforme:  $f(n) = g(n)$ ,  $W=0$

Pesquisa Ambiciosa:  $f(n) = h(n)$ ,  $W=\infty$

Pesquisa A\* Pesada:  $f(n) = g(n) + W h(n)$ ,  $1 < W < \infty$

Assuming the following search tree in which each arc shows the cost of the corresponding operator, indicate justifying, which node is expanded next using each of the following methods:

- a) Breadth-First Search; b) Depth-first Search; c) Uniform Cost search;  
d) Greedy Search; e) A\* Algorithm



a) [A, B] C, D, E, F, G, H

b) [A, B] E, F, G, H, C, D

c) [A, B] H, D, E, F, G, C

d) [A] C, D, E, G, H, F

e) [A] D, H, E, G, C, F

f)  $f = 2$ : [A] D, E, G, H, C, F

g)  $f = 4$ :

# Pesquisa Adversatória

jogos: perfeitos/imperfeitos; deterministas/nôtre

1. Estado Inicial: posição do tabuleiro e próximo jogador
2. Conjunto de Operadores: definem os movimentos legais
3. Teste Terminal: determina o fim do jogo
4. Função de Utilidade: atribui um valor numérico ao resultado do jogo

Algoritmo Minimax: escolher o movimento que tem o maior valor MINIMAX, i.e., o melhor que pode ser alcançado contra as melhores respostas do oponente

— ÓTIMO, COMPLETO,  $T = O(b^m)$ ,  $S = O(b^m)$

1. Gerar a árvore completa até aos estados finais
2. Aplicar a função de utilidade aos estados
3. Calcular os valores de utilidade de volta até à raiz, por camadas
4. Escolher o movimento com o maior valor

```
function MINIMAX-DECISION(state) returns an action
```

```
    v ← MAX-VALUE(state)  
    return the action in SUCCESSORS(state) with value v
```

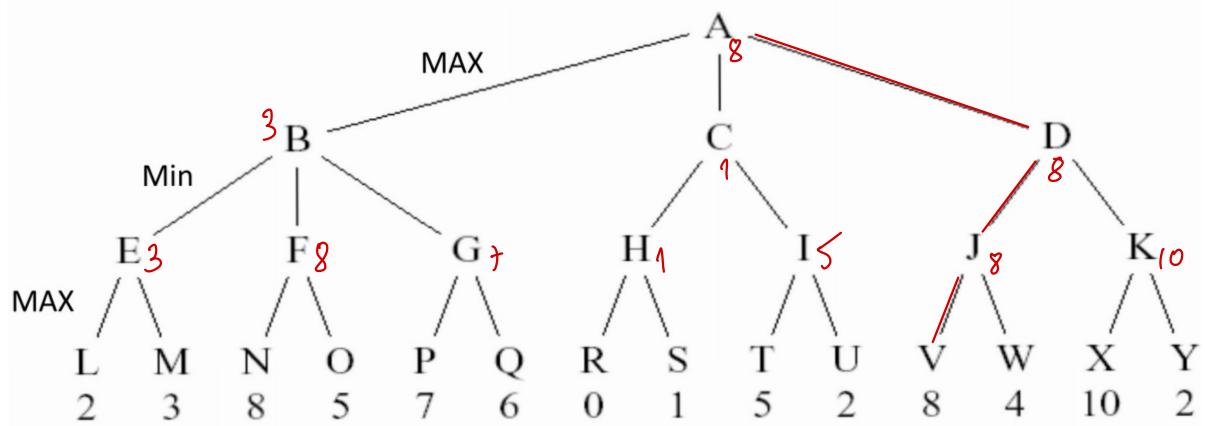
```
function MAX-VALUE(state) returns a utility value
```

```
    if TERMINAL-TEST(state) then return UTILITY(state)  
    v ← −∞  
    for a, s in SUCCESSORS(state) do  
        v ← MAX(v, MIN-VALUE(s))  
    return v
```

```
function MIN-VALUE(state) returns a utility value
```

```
    if TERMINAL-TEST(state) then return UTILITY(state)  
    v ← ∞  
    for a, s in SUCCESSORS(state) do  
        v ← MIN(v, MAX-VALUE(s))  
    return v
```

- Assuming that MAX is the first to play, apply the Minimax Algorithm to the following tree, indicating the movement selected by the algorithm and the respective estimated value



**Problema do Horizonte:** causado pela limitação da profundidade do algoritmo de pesquisa e manifestado quando algum evento negativo é inevitável, mas adiável - como só uma árvore parcial do jogo foi analisada, vai parecer ao sistema que o evento pode ser evitado quando, de facto, não pode

**Minimax Cutoff:** o teste terminal é substituído por Cutoff e a Utilidade é substituída por Avaliação (avalia a posição alcançada)

**Cuts Alpha-Beta:**  $\alpha$  é o melhor valor (para Max) encontrado até ao momento no caminho atual; se  $V$  é tão baixo que  $\alpha$ , Max deve evitá-lo e cortar o ramo (a definição de  $\beta$  é análoga) —  $T = O(b^{m/2})$

```
function ALPHA-BETA-SEARCH(state) returns an action
```

```
     $v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)$ 
```

```
    return the action in ACTIONS(state) with value  $v$ 
```

---

```
function MAX-VALUE(state,  $\alpha, \beta$ ) returns a utility value
```

```
    if TERMINAL-TEST(state) then return UTILITY(state)
```

```
     $v \leftarrow -\infty$ 
```

```
    for each  $a$  in ACTIONS(state) do
```

```
         $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
```

```
        if  $v \geq \beta$  then return  $v$ 
```

```
         $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
```

```
    return  $v$ 
```

---

```
function MIN-VALUE(state,  $\alpha, \beta$ ) returns a utility value
```

```
    if TERMINAL-TEST(state) then return UTILITY(state)
```

```
     $v \leftarrow +\infty$ 
```

```
    for each  $a$  in ACTIONS(state) do
```

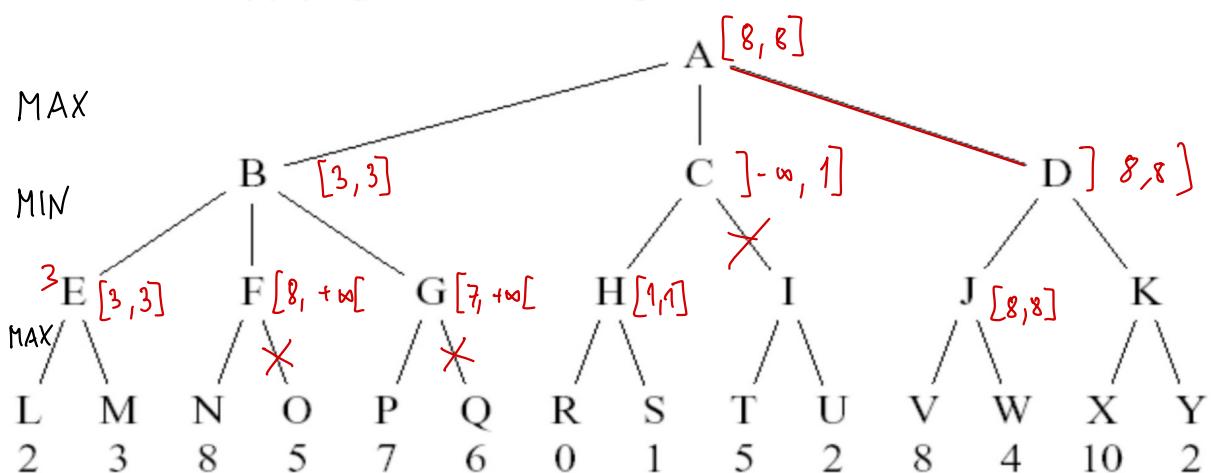
```
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
```

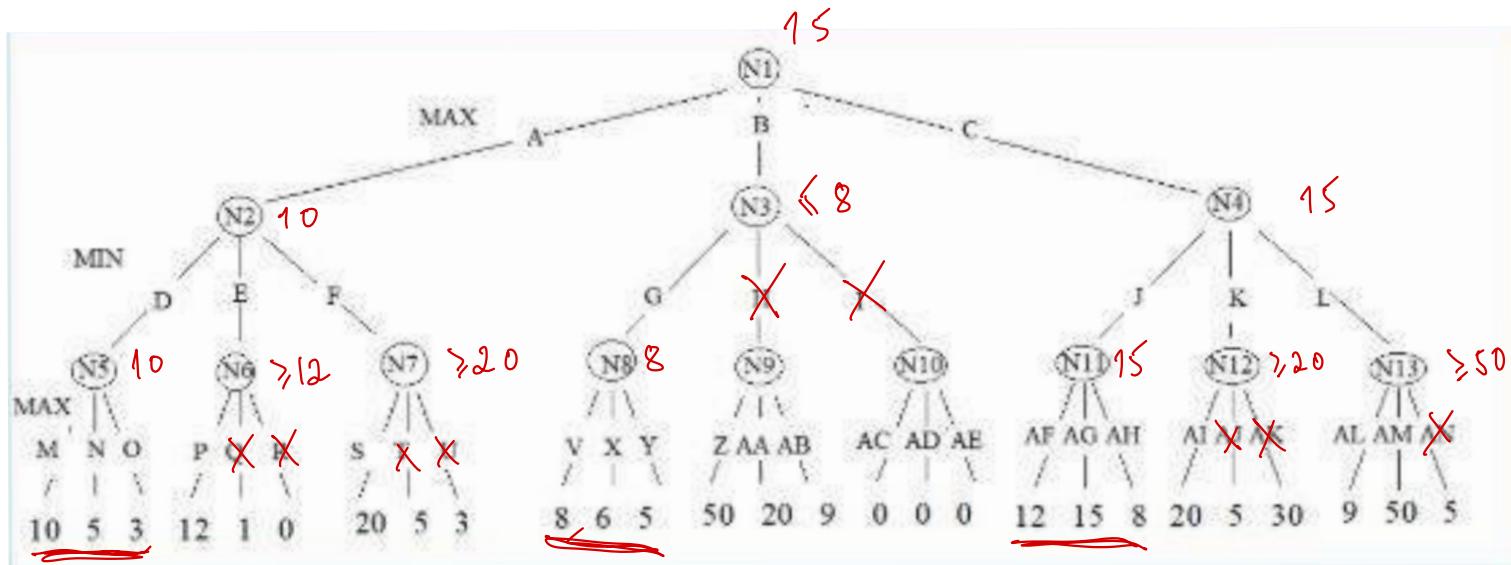
```
        if  $v \leq \alpha$  then return  $v$ 
```

```
         $\beta \leftarrow \text{MIN}(\beta, v)$ 
```

```
    return  $v$ 
```

Assuming that MAX is the first to play, apply the Minimax Algorithm with Alpha-Beta cuts to the following tree, indicating the movement selected by the algorithm. Indicate graphically and justify all the cuts made when applying the Minimax algorithm

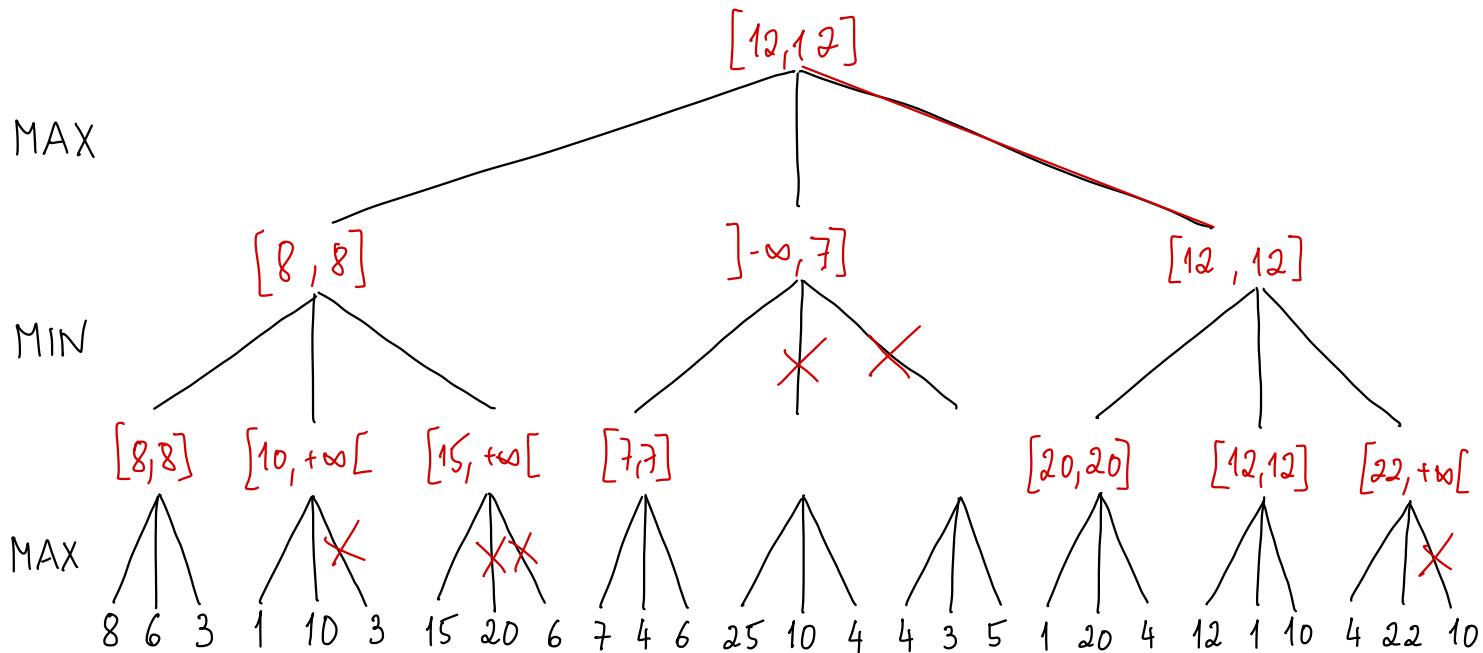




- Assuming that MAX is the first to play, apply the Minimax Algorithm with Alpha-Beta cuts to a tree with three levels, a branch factor 3 and with the following values of the evaluation function for the final line:

[ 8 6 3 1 10 3 15 20 6 7 4 6 25 10 4 4 3 5 1 20 4 12 1 10 4 22 10 ]

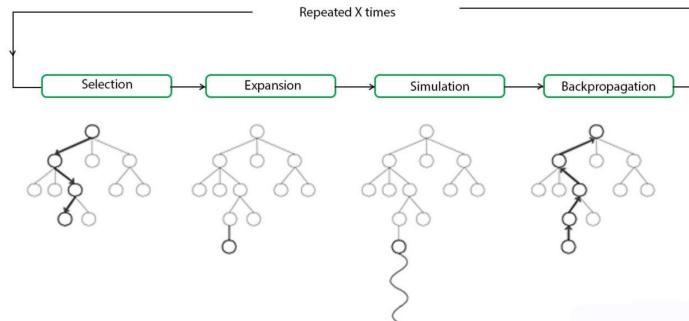
- Indicate graphically and justify all the cuts made when applying the Minimax algorithm



Negamax: como  $\text{max}(a, b) = -\min(-a, -b)$ , então o valor da posição do jogador A num jogo é a negação do valor do jogador B — o jogador A joga procura o movimento que maximiza a negação do valor resultante da jogada

Pesquisa Monte Carlo:

1. Seleção: começar pela raiz R e selecionar nós filhos até uma folha L ser alcançada — R é o estado atual do jogo e L é um nó que tem um potencial filho ainda por simular
2. Expansão: a não ser que F termine o jogo, criar um (ou mais) nós filhos e escolher um deles como C — os nós filhos são quaisquer movimentos válidos a partir da posição do jogo definida por L
3. Simulação: completar série aleatória de jogadas a partir do nó C, até ao jogo ser decidido, i.e., alcançar um resultado
4. Propagação para Trás: usar o resultado das jogadas para atualizar informação nos nós do caminho desde C até R



Expect-Maximac: a árvore de pesquisa deve incluir nós com probabilidades

# Otimização

Problema de Minimização: minimizar  $f_0(x)$  sujeito a  $g_i(x) \leq b_i$

Função Objetivo:  $f_0: \mathbb{R}^n \rightarrow \mathbb{R}$

Variáveis Controláveis:  $x = (x_1, \dots, x_n)$

Restrições:  $g_i: \mathbb{R}^n \rightarrow \mathbb{R}$

Problema de Otimização Combinatória:  $\max f(x), x \in F \subseteq S$

Solução Ótima:  $i^* \in S : f(i^*) \leq f(i), \forall i \in S$

Algoritmo (Exato): garante a solução ótima

Algoritmo Heurístico: não garante a solução ótima

Função de Vizinhança: mapeamento de uma solução para o conjunto de soluções possíveis, alcançadas por um movimento —  $N: S \rightarrow 2^S$

Vizinhança de Soluções: conjunto  $N$  para uma dada solução  $s \in S$ , tal que se  $t \in N(s)$ , então  $t$  está "perto" de  $s$

$N(s) \subseteq S$  é uma vizinhança de  $s$

Ótimo Global:  $x, f(x) \geq f(y), \forall y \in F$

Ótimo Local:  $x, f(x) \geq f(y), \forall y \in N(x)$

↳ vizinhança de  $x$

Pesquisa Local: começar com uma solução inicial e procurar iterativamente soluções melhores na vizinhança, tendo definido:

1. estratégia para aceitar solução na vizinhança
2. critério de paragem
3. o que fazer se a vizinhança não contiver uma solução melhor

Aceitar o Melhor: selecionar o melhor vizinho

Aceitar o Primeiro: selecionar o primeiro melhor vizinho encontrado

A pesquisa local é uma travessia de um grafo dirigido (grafo da vizinhança) cujos nós são os membros de  $S$

Movimento: processo de escolher uma dada solução na vizinhança da solução atual para se tornar a solução atual da próxima iteração

Problema: Como evitar que a pesquisa pare num ótimo local?

Heurística: técnica projetada para resolver um problema que ignora se a solução pode ser provada correta, mas que normalmente produz uma boa solução

Meta-Heurística: abordagem genérica de soluções heurísticas projetada para controlar e guiar heurísticas específicas orientadas aos problemas — Tem a capacidade de escapar a óptimalidade local

x: A (memória adaptativa) OU M (sem memória)

y: N (pesquisa sistemática na vizinhança) OU S (amostragem aleatória)

z: I (uma solução atual) OU P (população de soluções)

# Meta-Heurísticas

Arranque Simulado: descida aleatória modificada no espaço de soluções  
escolher um vizinho aleatório

movimentos que melhorem não sempre aceitos  
movimentos que pioram não aceitos com uma probabilidade que depende da quantidade de deterioração (quão pior) e da temperatura (parâmetro que diminui com o tempo)

```
s = s0; T = T0
for Iter = 0 to Itermax
    T ← temp_sched(T)
    snew ← random_neighb(s)
    if ProbEval(s), Eval(snew), T) ≥ rand(0, 1)
        then s ← snew
```

Subir a Colina: gerar, um por um, os sucessores do estado atual e, se encontram um melhor do que o estado atual, selecioná-lo e aplicá-lo

Subir a Colina (Aleatório): gerar um sucessor aleatório do estado atual e, se for melhor do que o estado atual, selecioná-lo e aplicá-lo

Subida Acentuada: gerar todos os sucessores possíveis e selecionar o melhor

Quando aceitar um movimento?  $\Delta = \text{Eval}(\text{vizinho aleatório}) - \text{Eval}(\text{solução atual})$

se  $\Delta > 0$ , sempre  
não, com probabilidade  $e^{\Delta/T}$

Se um movimento não for aceito, tentar outro vizinho aleatório

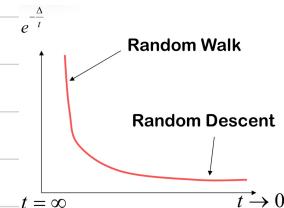
Problemas:

1.  $T_0 \rightarrow \infty \rightarrow$  perrejo aleatório
2. temperatura desce rápido  $\rightarrow$  para cedo num ótimo local
3. temperatura desce devagar  $\rightarrow$  convergência lenta

- $T_0$  deve ser alto
- devem ser feitas algumas iterações com a mesma temperatura
- $t_{i+1} = \alpha(t_i)$ , a  $\in [0,8; 1]$
- é necessário um critério de paragem (temperatura mínima)

Programação Estática: avanços específicos

Programação Dinâmica: reage a informação da pesquisa



- A programação de arrefecimento é baseada na diferença máxima no valor da função objetivo das soluções de uma vizinhança
- O número de repetições/movimentos é adaptativo: mais nas menores temperaturas (com um limite máximo)
- Não são necessárias temperaturas muito baixas

O que faz quando um algoritmo de pesquisa local para (num ótimo local ou perto de um)?

REINICIAR — repetir (iterar) o mesmo procedimento outras vezes com soluções iniciais diferentes

Pesquisa Local Iterada: escolher uma solução inicial aleatória, realizar pesquisa local e repetir, registrando o melhor ótimo local encontrado — procura nas soluções localmente ótimas, em vez de no espaço completo de soluções (como a pesquisa local)

Perturbação: movimento aleatório numa "vizinhança de ordem superior"  
a força é importante para evitar aleatoriedade e ótimos locais

Aceitação: entre descida aleatória e passo aleatório

Pesquisa em Vizinhança Variada: muda a vizinhança durante a pesquisa

Pesquisa Local Guiada: tenta contornar / superar ótimos locais removendo-os e mudando a "topografia" do estação de pesquisa — assume-se que existem funcionalidades de uma solução que se podem penalizar

Função de Avaliação Estendida = Função Objetivo Original + Penalizações

Cada funcionalidade tem um custo (constante ou variável) que representa (direta ou indiretamente) a influência da solução na função de avaliação de movimentos (estendida)

$$f^*(s) = f(s) + \lambda \sum_{i=1}^G I_i(s) \mu_i$$

$\lambda$ : influência

$I_i$ : funcionalidades

$\mu_i$ : penalizações

Num ótimo local  $s$ , deve ser aumentada a penalização para a funcionalidade com o maior valor de utilidade,  $u_i(s) = I_i(s) \times \frac{c_i}{1 + \mu_i}$

## RESUMO

SA: difícil encontrar um bom plano de arrefecimento

PLG: visita ótimos locais, mas pode encalar

PLI: perturba e faz pesquisa local

PVV: explora vizinhanças diferentes

Pesquisa Tabu: corta a pesquisa a partir de partes do estação de pesquisa (temporariamente) e guia a pesquisa em direção a outras partes da pesquisa através de penalizações e bônus

Básico: pesquisa local com estratégia "melhor melhoria"

Critério Tabu: define vizinhos que são tabu e não podem ser selecionados

Critério de Aspiração: define vizinhos tabu que podem ser selecionados se forem suficientemente bons

- É permitido mover-se para uma solução pior

- Não é permitido mover-se para soluções já visitadas (tabu)

Atributo: propriedade de uma solução ou movimento que pode ser baseado num aspecto da solução que é alterável por um movimento - não a base, para restrições tabu, usada para representar soluções visitadas recentemente

Critério Tabu: definido em atributos selecionados de um movimento (ou da solução resultante se o movimento for selecionado), evitando normalmente reversão ou repetição de movimentos e outros movimentos (mais tarde) que contêm o atributo tabu, cortando uma grande parte do espaço de pesquisa - cota algumas soluções não visitadas

Tabu Tense (11): número de iterações durante as quais o valor do atributo continua tabu - aumentar quando estagnado e diminuir quando escapa

Diversificação: estalar a pesquisa, ao priorizarativamente movimentos que dão soluções com uma nova composição de atributos

Intensificação: priorizar agressivamente atributos de uma boa solução numa nova solução

Ex: se o movimento  $s_k \rightarrow s_{k+1}$  envolver o atributo A, a restrição tabu usual não permite movimentos que revertam o estado de A

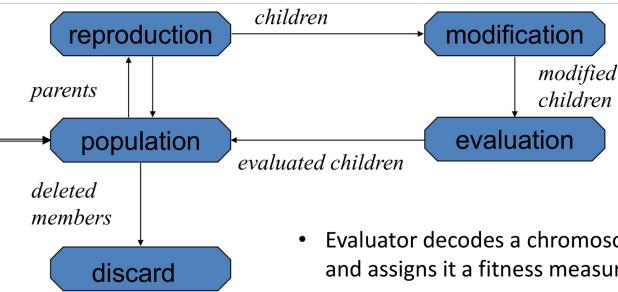
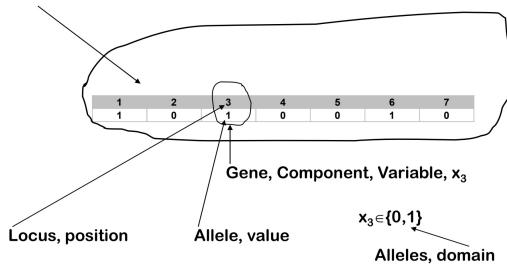
# Algoritmos Genéticos

Técnica de codificação  
procedimento de inicialização  
função de avaliação  
seleção de parentes  
operadores genéticos  
configurações de parâmetros

gene, cromosoma  
criação  
ambiente  
reprodução  
mutação, recombinação  
prática e arte

**Genótipo:** codificação de cromossomos — conjunto de strings codificados  
**Fenótipo:** expressão física — propriedades de um conjunto de soluções  
**População:** conjunto de soluções

Chromosome, component vector, vector, string, solution,  
individual  $x = (x_1, \dots, x_7)$



**Fitness:** relaciona-se com o valor da função objetivo para o problema  
↳ maximizar  $f: S \rightarrow [0, 1]$

**Operadores Genéticos:** manipulam cromossomos/soluções

**Mutação/Modificação Local:** operador unário — inversão

$$100 \rightarrow 110$$

**Crossover/Recombiнаção:** operador binário — combinação de cromossomos dos pais para gerar filhos

$$\begin{array}{|c|c|} \hline 1 & 1 \\ \hline 0 & 1 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 0 \\ \hline \end{array}$$

## Evolução:

- POA CADA PASSO
1. N gerações de populações com tamanho fixo M
  2. Seleção de indivíduos para operações genéticas
  3. Criação de novos indivíduos (reprodução)
  4. Mutação
  5. Seleção de indivíduos para sobreviver

Seleção: os pais são selecionados aleatoriamente com hipóteses de seleção enunciadas em relação às avaliações dos cromossomos

Torneio: escolha aleatória de grupos e o melhor do grupo avança para reprodução

Rouleta: seleção baseada na probabilidade dado o fitness

• A população pode ser qualquer estrutura de dados

Plano Padrão de Reprodução: tamanho fixo da população

Mutação: escolha aleatória do gene a mutar com probabilidade aleatória

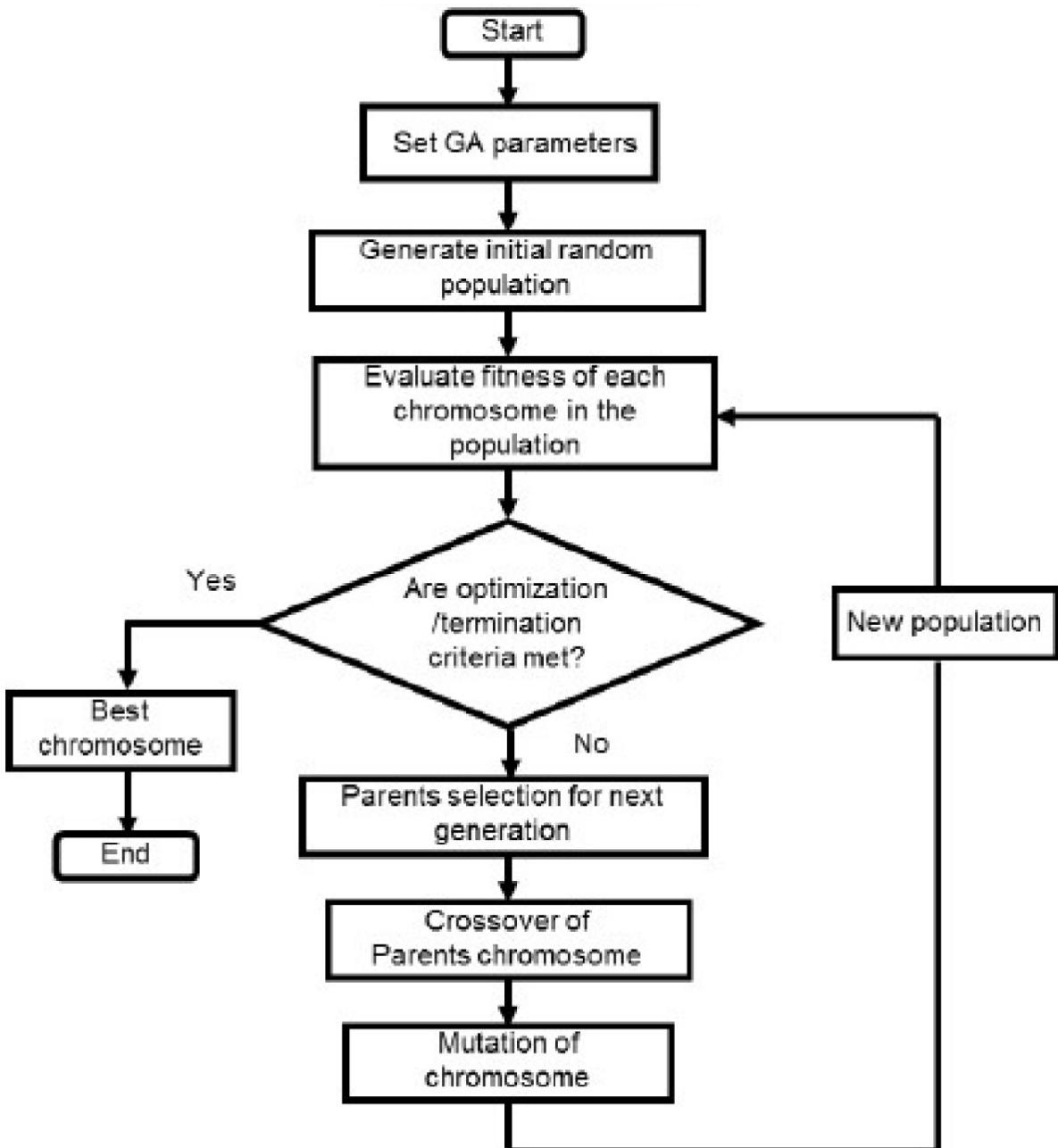
Crossover: um pai selecionado de acordo com o fitness e o outro aleatoriamente, com um ponto de crossover aleatório

AG Geracional: populações inteiras substituídas em cada iteração

AG de Estado Estável: poucos membros substituídos em cada geração

Esquema: subconjuntos semelhantes de cromossomos - os mesmos alelos em certas localizações - um dado cromossoma pode estar em vários esquemas

Rádio de Fitness: relação entre o fitness médio de um esquema e o fitness médio da população



# Representação de Conhecimento

Lógica: classe geral de representações para suportarem agentes baseados no conhecimento  
→ adaptam-se às mudanças no ambiente ao atualizarem o conhecimento

Base de Conhecimento (KB): conjunto de "frases" em que cada uma representa alguma assertão sobre o mundo - expressa numa linguagem de representação de conhecimento e a partir de conhecimento de fundo

TELL: adiciona novas frases (assertões) à base de conhecimento  
ASK: interroga o que é conhecido

Inferência: derivação de novas frases a partir das já existentes

Nível de Conhecimento: aquilo que o agente sabe e quais os seus objetivos  
Nível de Implementação: estruturas de dados dentro da KB e algoritmos que funcionam sobre elas

Mundo de Wumpus: parcialmente observável, determinista, episódico/sequencial, estático, discreto, de agente único

Sintaxe: especifica as frases que são bem formadas

Semântica: atribui significado às frases, determinando o seu valor de verdade em relação a cada mundo/modelo possível

$M(\alpha)$ : o conjunto dos modelos de  $\alpha$

↳ a frase  $\alpha$  é verdade no modelo  $m$  ↳  $m$  satisfaz / é modelo de  $\alpha$

**Vínculo:**  $\alpha \models \beta$

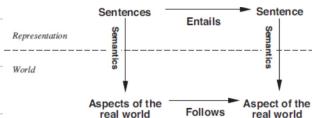
→  $\alpha$  vincula  $\beta$  →  $\beta$  segue logicamente de  $\alpha$   
se e só se  $M(\alpha) \subseteq M(\beta)$  q  $\beta$   
 $\alpha$  é uma assertão mais forte do que

• KB  $\models \alpha$  adiciona conhecimento à base de conhecimento

**Inferência Lógica:** aplicação do vínculo para derivar conclusões  
• KB  $\models_i \alpha$  algoritmo de inferência i deriva  $\alpha$  de KB

**Preservação da Verdade:** só devia frases vinculadas

**Completeness:** devia qualquer frase vinculada



• Se KB é verdade no mundo real, então qualquer frase  $\alpha$  derivada de KB por um procedimento de inferência sólido também é verdade no mundo real

**Frases Logicamente Equivalentes:** não verdade no mesmo conjunto de modelos,  $M(\alpha) = M(\beta)$  —  $\alpha \equiv \beta \leftrightarrow \alpha \models \beta \wedge \beta \models \alpha$

**Frase Válida:** verdade em todos os modelos

**Frase Satisfazível:** verdade em algum modelo

**Tautologia:** frase necessariamente dedutível

**Dedução:**  $\alpha \models \beta$  se e só se  $\alpha \Rightarrow \beta$  é válido

•  $\alpha$  é válido  $\leftrightarrow \neg \alpha$  é não satisfazível

•  $\alpha$  é satisfazível  $\leftrightarrow \neg \alpha$  é não válido

•  $\alpha \models \beta \leftrightarrow \alpha \wedge \neg \beta$  é insatisfazível

Regras de Inferência: permitem inferir nem construir tabelas de verdade

1. Modus Ponens:  $\alpha \Rightarrow \beta, \alpha \vdash \beta$
2. Eliminação do E:  $\alpha_1 \wedge \dots \wedge \alpha_n \vdash \alpha_i$
3. Introdução do E:  $\alpha_1, \dots, \alpha_n \vdash \alpha_1 \wedge \dots \wedge \alpha_n$
4. Introdução do OU:  $\alpha_i \vdash \alpha_1 \vee \dots \vee \alpha_n$
5. Eliminação da Dupla Negação:  $\neg\neg\alpha \vdash \alpha$
6. Resolução da Unidade:  $\alpha \vee \beta, \neg\beta \vdash \alpha$
7. Resolução:  $\alpha \vee \beta, \neg\beta \vee \gamma \text{ OU } \neg\alpha \Rightarrow \neg\beta, \beta \Rightarrow \gamma$   
 $\alpha \vee \gamma \quad \neg\alpha \Rightarrow \gamma$  !  
CNF

Cláusula de Horn:  $P_1 \wedge \dots \wedge P_n \Rightarrow Q$

Encadeamento para a Frente: dispor an qualquer regra cujas premissas não satisfazem a KB e adicionar as suas conclusões à KB - razão dirigida pelos dados: começando pelos dados conhecidos, deriva conclusões das permissões resultantes, sem uma interrogação específica em mente **PROGNÓSTICO**

Encadeamento para Trás: a partir de uma interrogação  $Q$ , encontrar implicações na KB cuja conclusão é  $Q$  - razão dirigida pelo objetivo: começando pela interrogação, deriva respostas aos objetivos específicos **DIAGNÓSTICO**

Quantificador Universal ( $\forall$ ): expressa propriedades de coleções de objetos  
Quantificador Existencial ( $\exists$ ): afirma algo sobre algum objeto, nem o nomeam

Prova por Contradição: para provar  $P$ , assumir que  $P$  é falso, adicionando  $\neg P$  à base de conhecimento

# Engenharia do Conhecimento

## Intelligent Systems

### Knowledge Based Systems

Exhibit intelligent behavior

Use explicit domain knowledge, stored separately

### Expert Systems

Use expert knowledge to solve difficult real-world problems, replacing the human expert

#### Vantagens:

1. Disponibilidade: experiência disponível de forma rápida e permanente
2. Fiabilidade: um computador vai dar sempre a mesma resposta solução
3. Explicabilidade: processo de raciocínio que pode verificar a correta da

#### Tarefas:

1. Aquisição de Conhecimento: adquirir conhecimento (especializado) sobre resolver um problema num dado domínio
2. Representação de Conhecimento: representar o conhecimento numa linguagem de representação computável
3. Controlo da Racionalização/Explicação

# Incerteza

Teoria da Decisão = Teoria das Probabilidades + Teoria da Utilidade

- Um agente racional escolhe a ação que retorna a maior utilidade esperada, pesados todos os resultados possíveis
- Attribuir-se um grau de crença (entre 0 e 1) aos eventos que não podem ser obtidos/determinados de forma precisa

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

**Probabilidade Condicionada:** calculada com base na presença de outros eventos independentes

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

**Teorema de Bayes:**  $P(b|a) = \frac{P(a|b) P(b)}{P(a)}$

O conhecimento diagnóstico é normalmente mais fraco do que o conhecimento causal

**Forma Geral:**  $P(d | s_1 \wedge \dots \wedge s_n) = \frac{P(s_1 \wedge \dots \wedge s_n | d) P(d)}{P(s_1 \wedge \dots \wedge s_n)}$

**Bayes Ingênuo:**  $P(d | s_1 \wedge \dots \wedge s_n) = P(d) \prod_i P(s_i | d)$

# Machine Learning

**Machine Learning:** área da IA que dá aos sistemas de computadores a capacidade de "aprender" (aumentar progressivamente o desempenho numa tarefa específica) a partir de dados / resultados das suas ações, sem serem explicitamente programados

**ML** { 1. Supervisionado: dirigido pelas tarefas (prever próximo valor)  
2. Não Supervisionado: dirigido pelos dados (identificar agrupamentos)  
3. Reforço: aprender com os erros

**ML** { 1. Supervisionado: não fornecidos exemplos de inputs e outputs desejados  
2. Não Supervisionado: não são dadas etiquetas nem outputs  
3. Reforço: dados só são dados como feedback das ações do agente

**Supervisionado** { Classificação: a variável de output é uma categoria - discreto  
Regressão: a variável de output é um valor real - contínuo

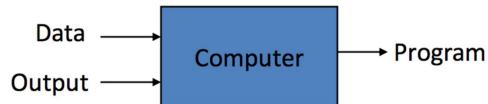
**Não Supervisionado** { Agrupamento  
Associação

**Reforço**: um agente inteligente interage com o ambiente e aprende a ação <sup>rela</sup>

Traditional Programming



Machine Learning



# Pandas

import pandas as pd

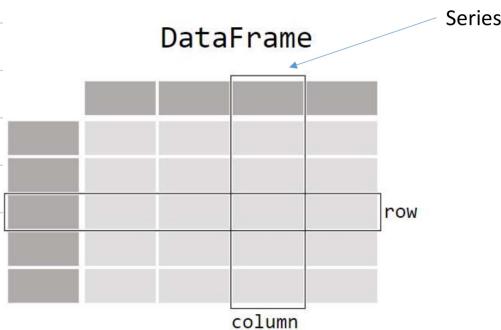
DataFrame: array bi-dimensional de valores com índices de linha e coluna

Series: array uni-dimensional de valores com um índice - uma coluna

pd.DataFrame(data = [[...]], columns = [...])  
pd.Series(data = [...])

- auto.head()
- auto.tail()
- auto.shape
- auto.info()
- pd.read\_\*
- pd.to\_\*

1. Criar novas colunas derivadas de colunas existentes
2. Estatísticas de Resumo
3. Combinar dados de múltiplas tabelas
4. Lidar com dados de séries temporais
5. Manipular dados textuais



# Dados

**Mineração de Dados:** extração não-trivial de informação implícita, anteriormente desconhecida e potencialmente útil a partir dos dados - exploração e análise, por meios automáticos ou semi-automáticos, de grandes quantidades de dados de maneira a descobrir padrões significativos

**Métodos Preditivos:** usar algumas variáveis para prever valores desconhecidos ou futuros de outras variáveis

**Métodos Descritivos:** encontrar padrões interpretáveis por humanos que descrevem os dados

**Classificação:** encontrar um modelo para um atributo de uma classe como uma função dos valores de outros atributos

**Regressão:** prever um valor de uma dada variável contínua com base nos valores de outras variáveis, assumindo um modelo de dependência linear ou não linear

**Agrupamento:** encontrar grupos de objetos tais que os objetos num grupo vao ser similares (ou relacionados) uns com os outros e diferentes (ou não relacionados) de/a objetos nouhos grupos - minimizar distâncias intra-grupos e maximizar distâncias inter-grupos

**Deteção de Alterações/Anomalias/Derrios:** detectar desvios significativos do comportamento normal

## Métodos de Extração de Conhecimento:

1. KDD: Knowledge Discovery in Databases
2. SEMMA: Sample, Explore, Modify, Model and Assess
3. CRISP-DM: Cross-Industry Standard for Data-Mining

**Dados**: coleção de objetos de dados e os seus atributos

**Atributo**: propriedade ou característica de um objeto - variável/campo/dimensão  
**Objeto**: coleção de atributos - registo/ ponto/ amostra/ entidade/ instância

**Valores de Atributos**: números ou símbolos atribuídos a um atributo para um objeto particular

**Propriedades de Atributos**  
1. Distinção:  $\neq$   
2. Ordem:  $<$   $>$   
3. Diferenças têm Significado: + -  
4. Rácios têm Significado: \* /

**Tipos de Atributos**  
1. Nominal - propriedade (1)  
2. Ordinal - propriedades (1) e (2)  
3. Intervalo - propriedades (1), (2) e (3)  
4. Rácio - propriedades (1), (2), (3) e (4)

} CATEGÓRICOS  
} NUMÉRICOS

**Atributos Discretos**: o conjunto de valores é finito ou infinito numerável

**Atributos Contínuos**: os valores são números reais

**Características Importantes**: dimensionalidade (número de atributos); esparcida; resolução, tamanho

## Tipos de Conjuntos de Dados:

1. Registo: coleção de registos, cada um com um conjunto fixo de atributos
  - a. Matrix: uma linha para cada objecto e uma coluna para cada atributo
  - b. Documento: cada termo é uma componente (atributo) de um vetor
  - c. Transação: cada transação envolve um conjunto de itens
2. Gráfico
3. Ordenados

## Problemas de Qualidade dos Dados:

1. Ruido: objectos estranhos ou modificação dos valores originais dos atributos
2. Outliers: objectos com características consideravelmente diferentes dos outros
3. Valores em Falta ~~remove~~ eliminar, estimar ou ignorar
4. Dados Duplicados

Medida de Semelhança: quão semelhantes são dois objectos [0,1] PROXIMIDADE

↓ PROPRIEDADES

1.  $s(x, y) = 1 \Leftrightarrow x = y$
2.  $s(x, y) = s(y, x)$

Attribute Type	Dissimilarity	Similarity
Nominal	$d = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$	$s = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$
Ordinal	$d =  x - y  / (n - 1)$ (values mapped to integers 0 to $n - 1$ , where $n$ is the number of values)	$s = 1 - d$
Interval or Ratio	$d =  x - y $	$s = -d, s = \frac{1}{1+d}, s = e^{-d}, s = 1 - \frac{d - \min_d}{\max_d - \min_d}$

Medida de Diferença: quão diferentes são dois objectos [0, +∞[ DISTÂNCIA

↓ PROPRIEDADES

1.  $d(x, y) = 0 \Leftrightarrow x = y$
2.  $d(x, y) = d(y, x)$
3.  $d(x, z) \leq d(x, y) + d(y, z)$

} DISTÂNCIA MÉTRICA

Distância de Minkowski:  $d(x, y) = \left( \sum_{k=1}^n |x_k - y_k|^p \right)^{1/p}$

p = 1: Manhattan  
p = 2: Euclides  
p → ∞: suprema

# PRÉ-PROCESSAMENTO

**Agregação:** combinar dois ou mais atributos/objetos num só atributo/objeto

**Amostragem:** escolher uma amostra representativa (com aproximadamente as mesmas propriedades de interesse) do conjunto original de dados

**1. Aleatória:** todos os itens têm a mesma probabilidade de serem escolhidos

a. sem Reposição: os itens selecionados não removidos da população

b. com Reposição: os itens selecionados não são removidos da população

**2. Estratificada:** particionar os dados e amostrar aleatoriamente cada partição

**Discretização:** converter um atributo contínuo num atributo ordinal

**Binariação:** mapear um atributo contínuo ou categórico em variáveis binárias

**Transformação:** mapear o conjunto de valores de um dado atributo num novo conjunto de valores, de modo que cada valor antigo possa ser identificado com um dos novos valores

**Normalização:** ajustar as diferenças entre atributos

**Padrãozização:** subtrair a média e dividir pelo desvio padrão

• Quando a dimensionalidade aumenta, os dados tornam-se progressivamente mais esparsos no espaço que ocupam

**Análise dos Componentes Principais:** encontrar uma projeção que captura a maior quantidade de variação nos dados

**Seleção de Subconjuntos de Funcionalidades:** ignorar redundantes ou irrelevantes

**Criação de Funcionalidades:** criar novos atributos que podem capturar informação importante

# Classificação

Classificação: dada uma coleção de registos (conjunto de treino) em que cada registo é caracterizado por um tuplo  $(x, y)$ , sendo  $x$  o conjunto de atributos (variável independente ou input) e sendo  $y$  a etiqueta da classe (variável dependente ou output), a tarefa é aprender um modelo que associe cada conjunto de atributos  $x$  numa das etiquetas de classes predefinidas  $y$ .

Árvore de Decisão: classificador básico - pode haver mais do que uma árvore que encaixe nos mesmos dados

Algoritmo de Hunt: seja  $D_t$  o conjunto de registos de treino que chegam ao nó  $t$

1. Se  $D_t$  contém registos que pertencem à mesma classe  $y_t$ , então  $t$  é um nó folha etiquetado como  $y_t$
2. Se  $D_t$  contém registos que pertencem a mais do que uma classe, usar um atributo de teste para dividir os dados em subconjuntos mais pequenos
3. Aplicar recursivamente o procedimento a cada subconjunto

Como dividir? Dependendo dos tipos dos atributos

Atributos Contínuos: discretização (categorias) ou decisão binária ( $A < v$  ou  $A \geq v$ )

Quando parar? Quando tudo com a mesma classe ou os mesmos valores

Divisão Multi-Via: usar várias partções como valores distintos

Divisão Binária: dividir valores em dois subconjuntos, preservando a ordem

Como determinar a melhor divisão?

1. Comptar a medida de impureza antes da divisão - P
2. Comptar a medida de impureza depois da divisão (impureza gerada dos nós filhos) - M
3. Escolher a condição que produz o maior ganho =  $P - M$

Sendo  $f_i(t)$  a frequência da classe  $i$  no nó  $t$  e  $c$  o número de classes, existem três medidas de impureza:

$$\text{Índice } G_{in} = 1 - \sum_{i=0}^{c-1} f_i(t)^2$$

$$\text{Entropia} = - \sum_{i=0}^{c-1} f_i(t) \log_2 f_i(t)$$

$$\text{Erro (de Classificação)} = 1 - \max_i [f_i(t)]$$

A impureza mínima é 0

Seja o nó pai,  $P$ , dividido em  $K$  partções/filhos e  $n_i$  o número de registos no nó filho  $i$ :

$$\text{Gain}_{\text{SPLIT}} = \text{Entropia}(P) - \sum_{i=1}^K \frac{n_i}{n} \text{Entropia}(i)$$

$$\text{Split}_{\text{INFO}} = - \sum_{i=1}^K \frac{n_i}{n} \log_2 \left( \frac{n_i}{n} \right)$$

$$\text{Gain Ratio} = \text{Gain}_{\text{SPLIT}} / \text{Split}_{\text{INFO}}$$

+: barato; rápido; fácil de interpretar; robusto ao ruído; lida bem  
-: não funciona bem com atributos interativos; só um atributo por decisão

Atributos Interativos: distinguem entre classes em conjunto, mas não roçinhos

Erros de Treino: erros cometidos no conjunto de treino

Erros de Teste: erros cometidos no conjunto de teste

Erro de Generalização: erro esperado de um modelo numa seleção aleatória de registos da mesma distribuição

Underfitting: quando o modelo é demasiado simples, os erros de treino e os erros de teste são grandes

Overfitting: quando o modelo é demasiado complexo, o erro de treino é pequeno mas o erro de teste é grande

↓  
· Aumentar o tamanho dos dados de treino reduz a diferença entre erros de treino e de teste do modelo até um determinado tamanho — o erro de treino não dá uma boa estimativa de quanto bem uma árvore vai decidir sobre novos registos

Seleção do Modelo: para evitar que o modelo seja demasiado complexo (overfitting), dividem-se os dados em treino em dois conjuntos:

1. Conjunto de Treino: para construir o modelo

2. Conjunto de Teste: para estimar o erro de generalização

Avaliação do Modelo: para estimar o desempenho do classificador em registos anteriormente não vistos, através de um de dois métodos:

1. Holdout: reservar  $k\%$  para treino e  $(100-k)\%$  para teste

2. Bootstrap: amostragem com reposição (para conjuntos de treino pequenos)

3. Cross Validation: particionar os dados em  $k$  subconjuntos disjuntos, treinando em  $k-1$  partições e testando na restante

↓  
a. Repetida: para estimar a variância do erro de generalização

b. Estatística: quando as classes não desequilibradas e a amostra é pequena, para garantir a mesma percentagem de classes nos conjuntos de treino e teste

## MATRIZ DE CONFUSÃO

		PREDICTED CLASS	
		Class=Yes	Class>No
ACTUAL CLASS	Class=Yes	a VP	b FN
	Class>No	c FP	d VN

- a: verdadeiro positivo VP  
 b: falso negativo FN  
 c: falso positivo FP  
 d: verdadeiro negativo VN

$$\text{Exatidão} = \frac{a + d}{a + b + c + d} = \frac{VP + VN}{VP + FN + FP + VN} = \frac{\text{VERDADEIROS}}{\text{TOTAL}}$$

		PREDICTED CLASS		
		C(i j)	Class=Yes	Class>No
ACTUAL CLASS	Class=Yes	C(Yes Yes)	C(No Yes)	
	Class>No	C(Yes No)	C(No No)	

## MATRIZ DE CUSTO

$C(i|j)$ : custo de classificar um exemplo da classe  $j$  como classe  $i$

$$\text{Exatidão Perda} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

$$\text{Precisão} = \frac{a}{a + c} = \frac{VP}{VP + FP} \longrightarrow \text{enviada para } C(Y|Y) \text{ & } C(Y|N)$$

$$\text{Lembrança} = \frac{a}{a + b} = \frac{VP}{VP + FN} \longrightarrow \text{enviada para } C(Y|Y) \text{ & } C(N|Y)$$

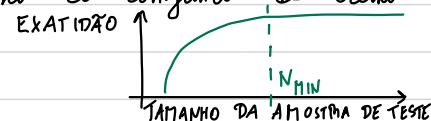
$$\text{Medida F} = \frac{2VP}{2VP + FP + FN} \longrightarrow \text{enviada para tudo exceto } C(N|N)$$

**Problema do Desequilíbrio de Classes:** se a classe interessante é dominada rara, então o classificador vai ignorá-la

↓ **COMO RESOLVER/EQUILIBRAR?**

1. Amostrar da classe maior de modo que o tamanho de ambas seja o mesmo **OU**
2. Replicar os dados da classe de interesse de modo que as classes sejam equilibradas

**Curva de Aprendizagem:** mostra como a exatidão muda com a variação do tamanho da amostra —  $n_{\min}$  representa o tamanho do conjunto de treino em que a curva converge para um plateau



**"Receiver Operating Characteristic" (ROC):** gráfico VPR contra FPR

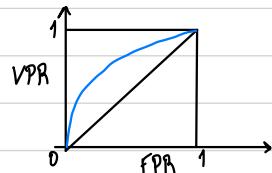
**VPR:** rácio de instâncias positivas previstas corretamente —  $VPR = \frac{VP}{VP + FN}$

**FPR:** rácio de instâncias negativas previstas incorretamente —  $FPR = \frac{FP}{FP + VN}$

↓ **O desempenho do classificador é representado como um ponto na curva ROC**

↓ **Num conjunto de dados uni-dimensional que contenha duas classes (positivo e negativo), quaisquer pontos localizados em  $x > t$  são classificados como positivos**

$(VPR, FPR) = \begin{cases} (0,0) & \text{: declara tudo como classe negativa} \\ (1,1) & \text{: declara tudo como classe positiva} \\ (1,0) & \text{: ideal} \end{cases}$



↓ **Na linha diagonal, a previsão é aleatória**

↓ **Abaixo da linha diagonal, a previsão é oposta à classe verdadeira**

**NEAREST NEIGHBOR:** usar o voto dos  $K$  vizinhos mais próximos e usar a maioria -  $w = 1/d^2$

Requisitos:

1. Um conjunto de registros etiquetados
2. Métrica de proximidade para computar distância/similaridade entre um par de registros
3. Número de vizinhos mais próximos a usar -  $K$
4. Método para usar as etiquetas das classes dos  $K$  vizinhos mais próximos para determinar a etiqueta da classe de um registro desconhecido

• Normalmente, é necessário pré-processar os dados

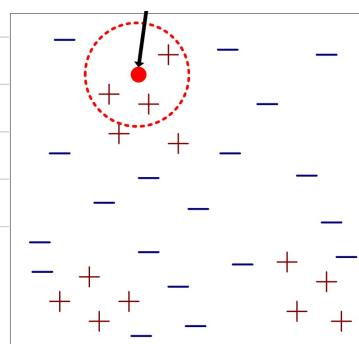
• Se  $K$  é demaisado pequeno, é suscível a pontos de ruído

• Se  $K$  é demaisado grande, a vizinhança pode incluir pontos de outras classes

**Valores em Falta:** normalmente, é necessária a presença de todos os atributos, mas pode usar-se um subconjunto dos atributos presentes em duas instâncias

**Atributos Irrelevantes:** adicionam ruído à medida de proximidade

**Atributos Redundantes:** encorajam a medida de proximidade em direção a outros atributos



**SUPPORT VECTOR MACHINES:** encontrar um hiper-plano linear (fronteira de decisão) que separe os dados — encontrar o hiper-plano que maximiza a margem

Modelo Linear:  $f(\vec{x}) = \begin{cases} 1, & \text{se } \vec{w} \cdot \vec{x} + b \geq 1 \\ -1, & \text{se } \vec{w} \cdot \vec{x} + b < -1 \end{cases}$

Oobjetivo: maximizar a margem  $= \frac{2}{\|\vec{w}\|}$   $\Leftrightarrow$  minimizar  $L(\vec{w}) = \frac{\|\vec{w}\|^2}{2}$

- Aprender o modelo é equivalente a determinar os valores de  $\vec{w}$  e  $b$
- A fronteira de decisão não depende dos vetores de suporte

E se o problema não for linearmente separável? introduzir variáveis — minimizar  $L(\vec{w}) = \frac{\|\vec{w}\|^2}{2} + C \left( \sum_{i=1}^N \xi_i \right)$  sujeito a  $y_i = \begin{cases} 1, & \text{se } \vec{w} \cdot \vec{x}_i + b \geq 1 - \xi_i \\ -1, & \text{se } \vec{w} \cdot \vec{x}_i + b \leq -1 + \xi_i \end{cases}$   
 → encontrar o hiper-plano que otimiza ambos os fatores

E se a fronteira de decisão não for linear? Transformar os dados num espaço com mais dimensões — a fronteira de decisão é  $\vec{w} \cdot \phi(\vec{x}) + b = 0$

Modelo Não Linear: minimizar  $\frac{\|\vec{w}\|^2}{2}$  sujeito a  $y_i (\vec{w} \cdot \phi(\vec{x}_i) + b) \geq 1, \forall (x_i, y_i)$

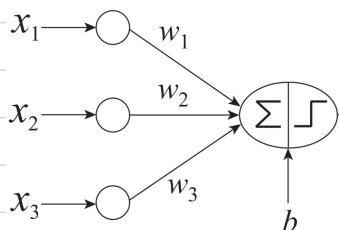
Truque de Kernel:  $\phi(x_i) \cdot \phi(x_j) = k(x_i, x_j)$  — função Kernel expressa em termos das coordenadas no espaço original

Características: robusto ao ruído; lida com overfitting ao maximizar a margem da fronteira de decisão; lida bem com atributos irrelevantes e redundantes; lida mal com valores em falta

**RÉDES NEURONIAIS ARTIFICIAIS:** coleção de unidades de processamento simples (nós) conectadas por ligações dirigidas (arestas) — cada nó recebe sinais das arestas que chegam, realiza computações e transmite sinais às arestas que saem (o peso de uma aresta determina o peso da conexão entre os nós)

Ideia: uma função não-linear complexa pode ser aprendida como uma composição de unidades de processamento simples

Perceção: um único neurônio — aprende fronteiras de decisão lineares



$$y = \begin{cases} 1, & \text{se } w^T x + b > 0 \\ -1, & \text{caso contrário} \end{cases}$$

$$\tilde{w} = (w^T b)^T$$

$$\hat{y} = \text{sign}(\tilde{w}^T \tilde{x})$$

↑ F.A.

$$\tilde{x} = (x^T 1)^T$$

Regra de Aprendizagem:

1. Inicializar os pesos ( $w_0, \dots, w_d$ )

2. Para cada exemplo de treino ( $x_i, y_i$ ), computar  $\hat{y}_i$  e atualizar os pesos:

$$w_j^{k+1} = w_j^k + \lambda (y_i - \hat{y}_i) x_{ij}$$

3. Só parar na condição de paragem — iterações ( $k$ ) ou rácio de aprendizagem ( $\lambda$ )

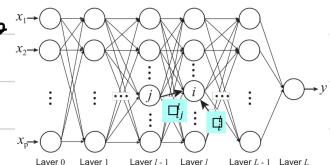
ERRO:  $e = y - \hat{y}$

Se  $y$  for uma combinação linear das variáveis de input, a fronteira de decisão é linear

Para problemas não linearmente separáveis, o algoritmo de aprendizagem do perceptron vai falhar porque nenhum hiper-plano linear consegue separar os dados perfeitamente

Rede Neuronal Multi-Camada: mais do que uma camada escondida de nós de computação — cada nó numa camada escondida opera em ativações a partir da camada precedente e transmite ativações para os nós da camada seguinte

- Com pelo menos uma camada escondida, podem resolver qualquer tipo de tarefa de classificação que envolva superfícies de decisão não-lineares
- Cada camada escondida representa um nível de abstração



Profundidade: número de camadas

$$\text{Função de Ativação: } a_i^l = f(z_i^l) = f\left(\sum_j w_{ij}^l a_{j-1}^l + b_j^l\right)$$

• Aproximar o erro nos nós escondidos pelo erro nos nós de output é perigoso porque não é claro como é que o ajustamento nos nós escondidos afeta o erro global, pelo que não garante convergência para o ótimo

Função Perda: para medir erros entre todos os pontos de treino

$$E(w, b) = \sum_{k=1}^n \text{Loss}(y_k, \hat{y}_k)$$

$$\text{Loss}(y_k, \hat{y}_k) = (y_k - \hat{y}_k)^2$$

Descida de Gradiente: atualizar parâmetros na direção da "descida máxima" da função perda entre todos os pontos —  $x_{ij}^l \leftarrow x_{ij}^l - \lambda \frac{\partial E}{\partial x_{ij}^l}, \forall i, j, l$

Estocástica: atualizar o peso para cada instância

Algoritmo de Propagação Para Trás: os gradientes na camada  $i$  podem ser computados usando os gradientes na camada  $i+1$  — começar na camada  $L$  e propagar os gradientes para trás para todas as camadas anteriores

Nós de Input: um por atributo contínuo/limiarizado,  $K$  ou  $\log_2 K$  por categoria

Nós de Output: um para classificação binária,  $K$  ou  $\log_2 K$  para  $K$  classes

Características: treino intensivo, teste rápido; lida bem com atributos redundantes e irrelevantes; lida mal com valores em falta; suscetível ao ruído no treino

# Redes Neuronais

- Uma rede neuronal profunda com complexidade arbitrariamente alta pode aproximar qualquer função com precisão arbitrariamente alta

Simples:  $x \in \mathbb{R}^1, y \in \mathbb{R}^1 \rightarrow y = wx + b$  | INPUT → ESCONDIDAS → OUTPUT

- Uma rede é basicamente muitas matrizes de pesos que são multiplicadas por vetores de nós

- As camadas têm funções de ativação - a última camada pode não ter

$$\begin{cases} h = g(\sum(w_k) + b) \\ y = \sum(w'h + b') \end{cases}$$

FUNÇÃO DE ATIVAÇÃO

Propagação para Trás: minimizar a função de perda  $L$  (diferença absoluta ou quadrada entre os outputs de treino e da rede) até 0 ou perto, usando derivadas parciais para obter a contribuição de cada peso no erro  
 $\rightarrow L(y, y') = \sum(|y' - y|)$        $\rightarrow L(y, y') = \sum((y' - y)^2)$

Pipeline: para a frente e para trás, batch por batch

Camadas:

- Totalmente Ligadas: ligam todos os nós da camada  $i$  à camada  $i+1$
  - Convolucionais: ter um conjunto de pesos que processa muitas partes pequenas da camada anterior
  - Recorrentes: a camada dá o output anterior a outras camadas
  - Memória Longa a Curto-Prazo: a camada guarda informação de passos para a frente para dar a futuras passos para a frente
- Se as coisas não funcionam, ninguém sabe porque ...

# Aprendizagem Reforçada

Aprendizagem Reforçada: aprendizagem dirigida pelos objetivos a partir da interação — aprender o que fazer (como mapear situações em ações) para maximizar uminal numérico de recompensa

- Não é dito ao agente que ações tomar: ele deve descobrir quais ações dão a melhor recompensa ao tentá-las
- Normalmente, as ações podem afetar não só a recompensa imediata mas também a próxima situação e recompensas subsequentes

Objetivo: selecionar ações que maximizam a recompensa cumulativa esperada

Política  $\pi$ : mapeamento (formalmente estocástico) de estados percepções para ações — "como é que o agente se deve comportar ao longo do tempo?"

Sinal de Recompensa  $r$ : define o objetivo do problema — em cada unidade de tempo, o ambiente envia uma recompensa para o agente

Função de Valor  $v$ : especifica o que é bom a longo prazo

Valor do Estado: quantidade total de recompensa que um agente pode esperar acumular nesse estado para a frente

Modelo do Ambiente: permite inferências sobre como o ambiente se vai comportar

• Procuram-se ações que trazem estados de maior valor e não maior recompensa, porque essas ações obtêm a maior quantidade de recompensa a longo prazo

Estado: função do histórico de observações, ações e recompensas  
Estado do Agente: representação interna do que o agente já viu

Ambiente Totalmente Observável: o agente observa diretamente o estado do ambiente  
Ambiente Parcialmente Observável: o agente pode ter ciúmes em estados possíveis do ambiente

"Exploitation": preferir ações conhecidas/estimadas como efetivas (a produzir recompensa) - maior recompensa a curto-prazo **GREEDY**  
"Exploration": tentar ações não selecionadas anteriormente (melhorar estimativas nos valores das ações) - menor recompensa a curto-prazo, maior recompensa a longo-prazo **NÃO GREEDY**

Problema do Bandido: uma configuração simples com um único estado  
• Existem  $K$  ações diferentes  
• Depois de cada ação, uma recompensa numérica é recebida a partir de uma distribuição estacionária de probabilidade  
• Cada ação tem um valor (recompensa média/esperada) não conhecido pelo agente -  $q_*(a) = E[R_t | A_t = a]$   
• O agente estima, em cada tempo  $t$ , o valor de uma ação  $a$  -  $\hat{Q}_t(a)$

$\hat{Q}_t(a) = \frac{\text{soma das recompensas quando } a \text{ foi tomado antes de } t}{\text{número de vezes em que } a \text{ foi tomado antes de } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}$

Regra de Atualização:  $\hat{Q}_{n+1} = \hat{Q}_n + \frac{1}{n} (R_n - \hat{Q}_n)$

$\alpha$  constante, peso mais recompensas recentes  $\in (0, 1]$  direção desejável para se mover

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}[\text{Target} - \text{OldEstimate}]$$

Seleção de Ações Greedy: "exploit" -  $A_t = \operatorname{argmax}_a Q_t(a)$

Seleção de Ações E-Greedy: comportar-se de forma greedy na maior parte do tempo, mas com uma pequena probabilidade  $\epsilon$  de selecionar aleatoriamente entre todas as ações -  $Q_t(a)$  vai convergir para  $q_*(a)$  se a for selecionada com frequência suficiente

Seleção de Ações Softmax:  $P(A_t = a) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^K e^{Q_t(b)/\tau}}$  TEMPERATURA

$\tau = \begin{cases} \text{alto: ações tendem a ser equiprováveis} \\ \text{baixo: valores das ações importam mais} \\ 0: \text{seleção de ações greedy} \end{cases}$

#### A simple bandit algorithm

Initialize, for  $a = 1 \dots k$ :

$$Q(a) \leftarrow 0$$

$$N(a) \leftarrow 0$$

Loop forever:

$$A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{with probability } 1 - \epsilon \\ \text{a random action} & \text{with probability } \epsilon \end{cases} \quad (\text{breaking ties randomly})$$

$$R \leftarrow \text{bandit}(A)$$

$$N(A) \leftarrow N(A) + 1$$

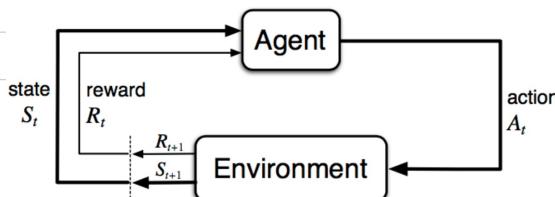
$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)} [R - Q(A)]$$

Processos de Decisão de Markov: formalização de tomada sequencial de decisões em que o ambiente é totalmente observável e as ações não influenciam só recompensas imediatas, mas também situações (estados) subsequentes e por isso recompensas futuras - estima-se o valor  $q_*(s, a)$

$$\cdot P(s', r | s, a) = P(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$$

• A probabilidade de cada valor formado para  $s'$  e  $r$  depende só do estado imediatamente precedente  $s$  e ação  $a$

Propriedade de Markov: o estado deve incluir toda a informação relevante sobre a interação agente-ambiente passada



**Sinal de Recompensa:** usado para definir o objetivo do agente — forma de comunicar ao agente o que alcançar e não como alcançar, pelo que não deve codificar conhecimento anterior (é parte do ambiente, não do agente)

As recompensas devem ser fornecidas de modo que ao maximizá-las o agente consiga alcançar o objetivo — maximizar recompensa cumulativa

O agente quer maximizar o retorno esperado:  $G_t = R_{t+1} + R_{t+2} + \dots + R_T$

↓ ADICIONAR DESCONTO

O agente quer maximizar o retorno descontado esperado

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$$

$\gamma$ : rácio de desconto — valor presente de futuras recompensas,  $0 < \gamma \leq 1$

O valor de receber a recompensa  $R$  depois de  $k+1$  passos é  $\gamma^k R$

O agente valoriza recompensa imediata em vez de recompensa atrasada

$\gamma = \begin{cases} 0: & \text{o agente é miópico, só interessa a recompensa imediata} \\ \rightarrow 1: & \text{o agente vê para a frente, considera fortemente recompensas futuras} \end{cases}$

$G_t$  torna-se finito, desde que  $\gamma < 1$ , existindo-se retornos infinitos

**Equação de Bellman:** a função de valor pode ser recursivamente decomposta em duas partes — a recompensa imediata  $R_{t+1}$  e o valor descontado do estado sucessor  $\gamma v(S_{t+1})$

$$v(s) = E [G_t | S_t = s]$$

- Recompensas futuras dependem da escolha de ações
- Funções de Valor não definidas em relação a políticas (formas de agir)

**Política:** mapeamento de estados para probabilidades de selecionar cada ação possível -  $\pi(a|s) = P(A_t = a | S_t = s)$

**Função Estado-Valor:** retorno esperado ao começar em  $s$  e seguir  $\pi$  depois  
 $\rightarrow v_\pi(s) = E_\pi[G_t | S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s]$

**Bellman:** ver para a frente desde um estado para os seus possíveis estados sucessores

**Função Ação-Valor:** retorno esperado ao tomar a ação  $a$  no estado  $s$  e seguir  $\pi$  depois

$$\rightarrow q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a] = E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a]$$

**Bellman:** ver para a frente desde um par estado-ação para os seus possíveis estados sucessores

**Função Estado-Valor Ótima:**  $v_*(s) = \max_\pi v_\pi(s), \forall s \in S$

**Função Ação-Valor Ótima:**  $q_*(s, a) = \max_\pi q_\pi(s, a), \forall s \in S, \forall a \in A$

**MAXIMIZAR**

• Conhecidas  $v_*$  ou  $q_*$ , a política ótima  $\pi_*$  é greedy - o retorno esperado é melhor do que qualquer outra política

**Avaliação da Política:** computar a função estado-valor  $v_\pi$  para uma política arbitrária  $\pi$  - f.e., programação dinâmica

#### Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input  $\pi$ , the policy to be evaluated

Algorithm parameter: a small threshold  $\theta > 0$  determining accuracy of estimation  
 Initialize  $V(s)$ , for all  $s \in S^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop:

$$\Delta \leftarrow 0$$

Loop for each  $s \in S$ :

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until  $\Delta < \theta$

• Resolver a equação de optimidade de Bellman é equivalente a pesquisa exaustiva e, por isso, impraticável para grandes espaços de estados

### ↓ APROXIMAÇÃO

• Colocar mais esforço em aprender a tomar decisões para estados encontrados frequentemente

**Aprendizagem Reforçada Livre de Modelo:** não é dado MDP, i.e., não se assume conhecimento completo do ambiente — aprende-se diretamente por experiência real ao interagir com o ambiente

**Aprendizagem Monte-Carlo:** faz a média dos retornos de sequências amostrais completas de estados, ações e recompensas — MDPs episódicos      AMOSTRAGEM  
 $\rightarrow V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$

**Aprendizagem Diferença Temporal:** atualiza estimativas com base em outras estimativas aprendidas, sem esperar por um resultado final — atualiza estimativas com base em recompensa imediatamente observada e estado BOOTSTRAP  
 $\rightarrow V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

**TD(λ):** combina a amostragem de Monte Carlo com o bootstrap da Diferença Temporal

**Sarsa:**  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$  — converge para a política e função ação-valor ótimas se todos os pares estado-ação são visitados infinitamente e a política converge para greedy ( $\epsilon = 1/t$ )

**Q-Learning:**  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$  — a política always  $\pi$  é greedy em relação a  $Q(S, A)$  e a função ação-valor aprendida  $Q$  aproxima diretamente  $q_*$ , independentemente da política a seguir

# Processamento de Linguagem Natural

Teste de Turing: "um computador merece ser chamado inteligente se conseguir enganar um humano em acreditar que o computador é um humano"

Processamento de Linguagem Natural: área de ciência dos computadores, IA e linguística computacional preocupada com as interações entre computadores e linguagens (naturais) humanas e, em particular, preocupada com programar computadores para processar com sucesso grandes corpos da linguagem natural

Tokeneração: dividir uma frase em tokens (palavras)

Partir em Frases: dividir um texto em frases

Etiquetagem de Partes do Discurso: determinar a categoria de cada palavra

Parsing Sintático: determinar a árvore de parse (análise gramatical) de uma frase

Desambiguação do Sentido de Palavras: selecionar o significado de palavras num contexto

Reconhecimento de Entidades Nomeadas: determinar quais itens num texto se referem a entidades

Resolução de Co-Referências: determinar quais palavras (menções) se referem aos mesmos objetos (entidades)

Funcionalidades Linguísticas

1. Lexical
2. Sintática
3. Gramatical
4. Semântica
5. Estrutural

Aplicações:

1. Análise de Sentimentos e Mineração de Opiniões
2. Sumarização de Textos
3. Tradução Máquina
4. Fala - Para - Texto / Texto - Para - Fala
5. Extração de Informação
6. Resposta a Perguntas
7. Verificação de Factos e Detecção de Notícias Falsas
8. Mineração de Argumentos e Portais de Debate
9. Geração de Linguagem Natural
10. IA Conversacional (Cháteus)

## PROCESSAMENTO

Expressão Regular: sequência de caracteres que define um padrão de pesquisa

Normalização de Palavras: por palavras/tokens num formato padrão

Lematização: determinar a raiz da palavra - PALAVRA = TRONCO + AFIXO  
significado principal

"Stemming": cortar afixos finais das palavras  
significado adicional

## CLASSIFICAÇÃO

Input: um documento  $d$  e um conjunto fixo de classes  $C = \{c_1, c_2, \dots, c_m\}$

Output: classe prevista  $c \in C$  para o documento  $d$

→ Regras Codificadas Manualmente

→ Aprendizagem Supervisionada: usar conjuntos de dados anotados - desenvolver um classificador ( $D \rightarrow C$ ) para fornecer como input

Saco de Palavras: conjunto desordenado de palavras, mantendo apenas a sua frequência no documento - assume que a posição não importa

Náïve Bayes: faz uma aproximação simplificada sobre como as funcionalidades interagem

Regra de Bayes:  $P(c|d) = \frac{P(d|c) P(c)}{P(d)}$

Classe Mais Provável:  $\hat{c} = \operatorname{argmax}_{c \in C} P(c|d) = \operatorname{argmax}_{c \in C} \frac{P(d|c) P(c)}{P(d)} = \operatorname{argmax}_{c \in C} P(d|c) P(c)$

$= \operatorname{argmax}_{c \in C} \underbrace{P(f_1, f_2, \dots, f_n | c)}_{\text{LIKELIHOOD}} \underbrace{P(c)}_{\text{PRIOR}}$

Assumindo independência condicional,  $P(f_1, f_2, \dots, f_n | c) = P(f_1 | c) \times \dots \times P(f_n | c)$

Classificador:  $c_{NB} = \operatorname{argmax}_{c \in C} \prod_{f \in F} P(f | c) = \operatorname{argmax}_{c \in C} \prod_{i \in \text{for.}} P(w_i | c)$

$$\hat{P}(c) = \frac{N_c}{N_{\text{doc}}}$$

$$\hat{P}(w_i | c) = \frac{\operatorname{count}(w_i, c)}{\sum_{w \in V} \operatorname{count}(w, c)} \approx \frac{\operatorname{count}(w_i, c) + 1}{\sum_{w \in V} (\operatorname{count}(w, c) + 1)} = \boxed{\frac{\operatorname{count}(w_i, c) + 1}{\sum_{w \in V} \operatorname{count}(w, c) + |V|}}$$

Como lidar com negação?

- ou
1. Adicionar NOT no início das palavras afetadas
  2. Usar bigramas em vez de palavras únicas

Léxico: listas de palavras pré-anotadas - conhecimento externo

Como construir outras funcionalidades?

1. Predefinir conjuntos prováveis de palavras ou frases
2. Funcionalidades paralingüísticas e extra-lingüísticas
3. N-gramas

Modelo Gênerativo: usa um termo de probabilidade - como gerar as funcionalidades de um documento se se souber que ele era de classe c?

Modelo Discriminativo: tenta aprender a distinguir as classes e tenta computar diretamente  $P(c|d)$  - melhor quando há muitas funcionalidades correlacionadas

Incorporações de Palavras: representar o significado de uma palavra através de vetores de valores reais, baseados no contexto em que as palavras ocorrem - a similaridade semântica computa-se usando operadores matemáticos

Analogia: capacidade de capturar significados relacionais - calcular diferenças (offsets) entre vetores de incorporações

