

Aula 01 - Apresentação da Disciplina

Profa. Gabrielly Queiroz

Ementa

- Métodos de resolução de problemas.
- Busca em espaço de estados.
- Busca sem informação.
- Uso de heurísticas.
- Satisfação de restrição.
- Representação do conhecimento.
- Representação do conhecimento incerto.
- Aprendizagem de máquina, aprendizagem de classificadores e de regras de associação; agrupamento de dados.

Data	Conteúdo
21 de julho	Apresentação da disciplina. Introdução à Ciência de Dados e IA: Definições, importância e aplicações.
25 de julho	Métodos de Resolução de Problemas e Busca em Espaço de Estados.
28 de julho	Exercícios - Métodos de Resolução de Problemas e Busca em Espaço de Estados.
01 de agosto	Busca sem Informação: BFS, DFS, Busca de Custo Uniforme.
04 de agosto	Introdução a ferramentas para manipulação de dados: Jupyter Notebook e Python. Exercícios - Busca sem Informação: BFS, DFS, Busca de Custo Uniforme.
08 de agosto	Uso de Heurísticas.
11 de agosto	Uso de Heurísticas. Exercícios.
15 de agosto	Satisfação de Restrições.
18 de agosto	Exercícios - Satisfação de Restrições.
22 de agosto	Representação do Conhecimento e Inferência Lógica.
25 de agosto	Representação do Conhecimento e Inferência Lógica - Exercícios
29 de agosto	Fundamentos de Aprendizado de Máquina.
01 de setembro	Introdução a manipulações de dados com python - Pandas
05 de setembro	Noções de estatística básica e probabilidade aplicadas à IA.
08 de setembro	Exercícios - Fundamentos de Aprendizado de Máquina e Probabilidade.
12 de setembro	Técnicas de visualização de dados: Matplotlib e Seaborn.
15 de setembro	FERIADO

19 de setembro	Algoritmos básicos: Regressão Linear e Regressão Logística.
22 de setembro	Implementação com Scikit-learn de Algoritmos básicos: Regressão Linear e Regressão Logística. Exercícios - Implementação de Regressão Linear e Logística.
26 de setembro	Algoritmos de Clusterização: K-means.
29 de setembro	Implementação com Scikit-learn de Algoritmos de Clusterização: K-means.
03 de outubro	Árvores de Decisão.
06 de outubro	Implementação com Scikit-learn de Árvores de Decisão.
10 de outubro	Ferramentas para Construção de Pipelines de IA. Ajuste e Otimização de Modelos de IA.
13 de outubro	Saco cheio
17 de outubro	Saco cheio
20 de outubro	Avaliação Teórica
24 de outubro	Random Forest e Orientações para o Projeto Final
27 de outubro	Random Forest - Exercícios
31 de outubro	Exercícios Práticos sobre Pipelines de IA e Ajuste e Otimização.
03 de novembro	Desenvolvimento do Projeto Final - Auxílio
07 de novembro	Desenvolvimento do Projeto Final - - Auxílio
10 de novembro	Apresentação do Projeto Final
14 de novembro	Apresentação do Projeto Final
17 de novembro	Apresentação do Projeto Final

Avaliação

30%

Avaliação Teórica

30%

Atividades de aula

40%

Trabalho Final

Trabalho Final

Aplicação do algoritmo de Machine Learning em um problema real.

Criar um problema. Criar o Objetivo.

Aplicar o algoritmo para solução.

INDIVIDUAL.

Aula 1 - Introdução à Ciência de Dados e IA: Definições, importância e aplicações.

Profa. Gabrielly Queiroz

Introdução

- **Ciência de Dados:** Área que analisa grandes volumes de dados para extrair informações úteis.
- **Inteligência Artificial:** Desenvolvimento de sistemas que simulam raciocínio humano e aprendem com dados.
- Ambas são fundamentais para a transformação digital.
- **Contexto atual:** Geração massiva de dados torna indispensável o uso de ferramentas analíticas.
- **Impacto:** Empresas, governos e instituições utilizam essas áreas para decisões assertivas e inovações.



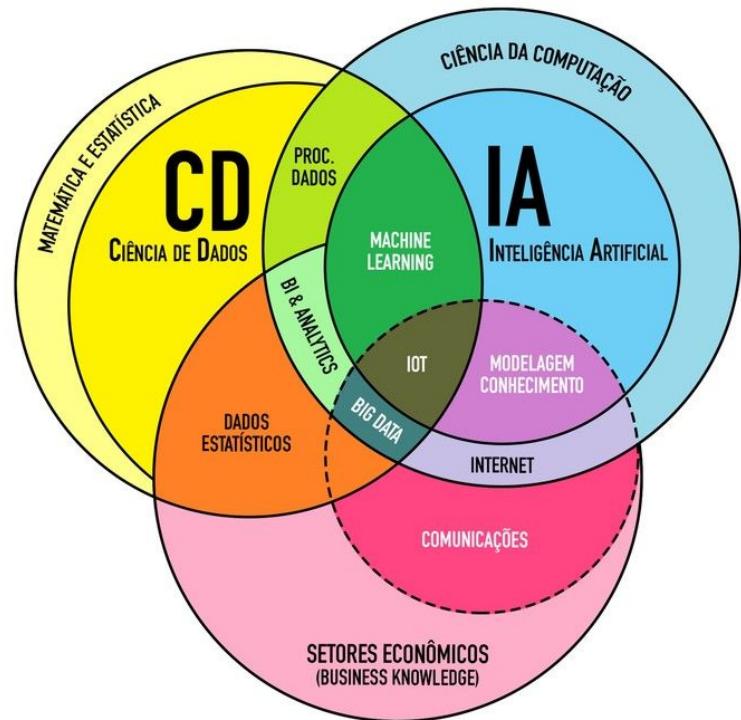
Importância e Aplicações

Os exemplos práticos reforçam a importância dessas áreas:

- **Saúde:** Algoritmos podem detectar doenças em estágios iniciais, salvar vidas e personalizar tratamentos (Hunter, Hindocha e Lee, 2022).
- **Setor financeiro:** Técnicas avançadas de análise preveem tendências de mercado e identificam fraudes (Cho, 2023).
- **Cotidiano:** Sistemas de recomendação, como aqueles usados em plataformas de streaming, personalizam experiências ao consumidor.
- **Agricultura:** Sensores e modelos preditivos ajudam a otimizar recursos, como água e fertilizantes.

Conexão entre CD e IA

- **Complementaridade:**
 - A Ciência de Dados fornece dados estruturados para algoritmos de IA.
 - A IA potencializa os resultados da CD com previsões e automatizações avançadas.
- **Big Data:**
 - Serve como base tanto para a CD quanto para a IA.
 - Utilizado para encontrar padrões e treinar modelos adaptativos.



Aula 02 - Métodos de Resolução de Problemas e Busca em Espaço de Estados.

Profa. Gabrielly Queiroz

Contextualização

- Computadores não são inteligentes, pois apenas executam as instruções que os programadores escrevem.
- Se um computador pudesse, por conta própria, **criar suas próprias regras e decidir o que fazer sem precisar de instruções pré-definidas**, ele seria considerado mais inteligente.
- Esse é um dos grandes desafios da Inteligência Artificial: dar às máquinas a capacidade de **auto programação**, ou seja, de formular suas próprias soluções para problemas.
- Uma das primeiras técnicas desenvolvidas para isso foi a **busca em espaço de estados**. Apesar de ser uma abordagem básica, ela é **fundamental para diversas outras áreas da IA**.
- O espaço de estados ensina a máquina a **explorar diferentes possibilidades e encontrar soluções**, imitando um processo de pensamento lógico.

Contextualização

Problema → Criamos um programa (Python, Java, etc.) → O computador executa → Problema resolvido.

A inteligência está no programador, não no computador.

O computador apenas segue as instruções, sem tomar decisões por conta própria.

Em vez de programar a solução diretamente, criamos um **agente** que busca a solução.

O agente interage com o ambiente e toma decisões.

Ele explora diferentes possibilidades até alcançar um **objetivo**.

A inteligência agora está no processo de busca, e não apenas no código fixo

Um problema pode ser representado por um **espaço de estados**.

O agente navega nesse espaço até encontrar a solução.

Elementos principais:

- **Estados**: Diferentes configurações do problema.
- **Ações**: Como o agente pode mudar de um estado para outro.
- **Objetivo**: O estado final desejado.

Busca em Espaço de Estados

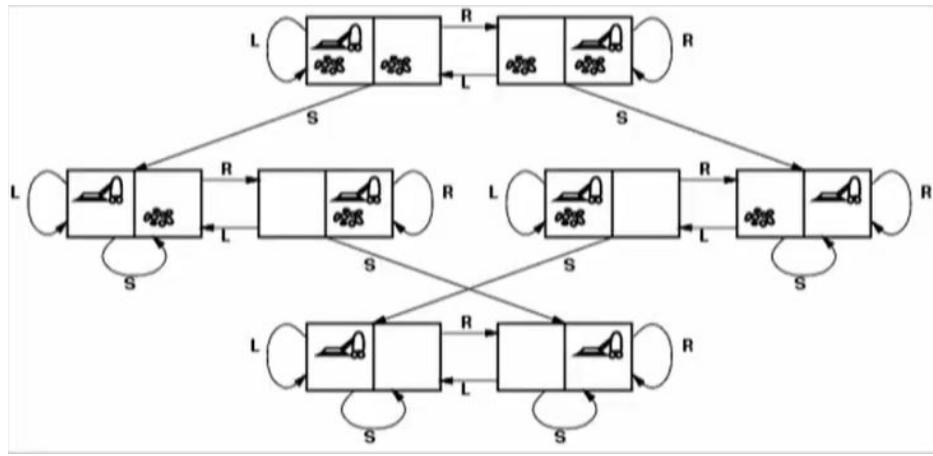
- Abordagem de Busca: Agente orientada a meta/objetivo.
- O agente está dentro do problema, tem o objetivo que quer atingir e tem o que ele percebe.
- O processo produz um programa de computador que vai resolver o problema.
- Executa o programa e resolve.

Exemplo do aspirador de pó

- Um robô aspirador de pó deve limpar uma casa com duas posições.
Operações que ele sabe executar:
 - Sugar.
 - Ir para a posição da esquerda.
 - Ir para a posição da direita.
- Como o aspirador pode montar um plano para limpar a casa se inicialmente ele está na posição direita e as duas posições tem sujeira?
- Quais os estados possíveis do mundo do aspirador e as transições?

Exemplo do aspirador de pó

- Busca: Ações e Estados.
- Ações: Sugar. Esquerda. Direita
- Estados: 2 posições. Limpo ou sujo.
- Cenários: casa suja e posição esquerda. Casa suja e na direita. Casa limpa e na direita. Casa limpa e na esquerda. Sujeira apenas na direita e robo na esquerda. Sujeira apenas na direita e robo na direita. Sujeira apenas na esquerda e robo na esquerda. Sujeira apenas na direita e robo na esquerda.
- Transformações de estados com base nas ações. Grafos.
- Estado Atual e Objetivo.
- Programação: Achar um caminho pelo grafo.



O que é busca?

- O mundo do agente é modelado por conjunto de estados possíveis (muitas vezes este conjunto é infinito)
- Existem transições entre os estados do mundo, formando um grafo
- São utilizados algoritmos para encontrar um caminho neste grafo
 - Partindo do estado inicial (atual)
 - até o estado objetivo

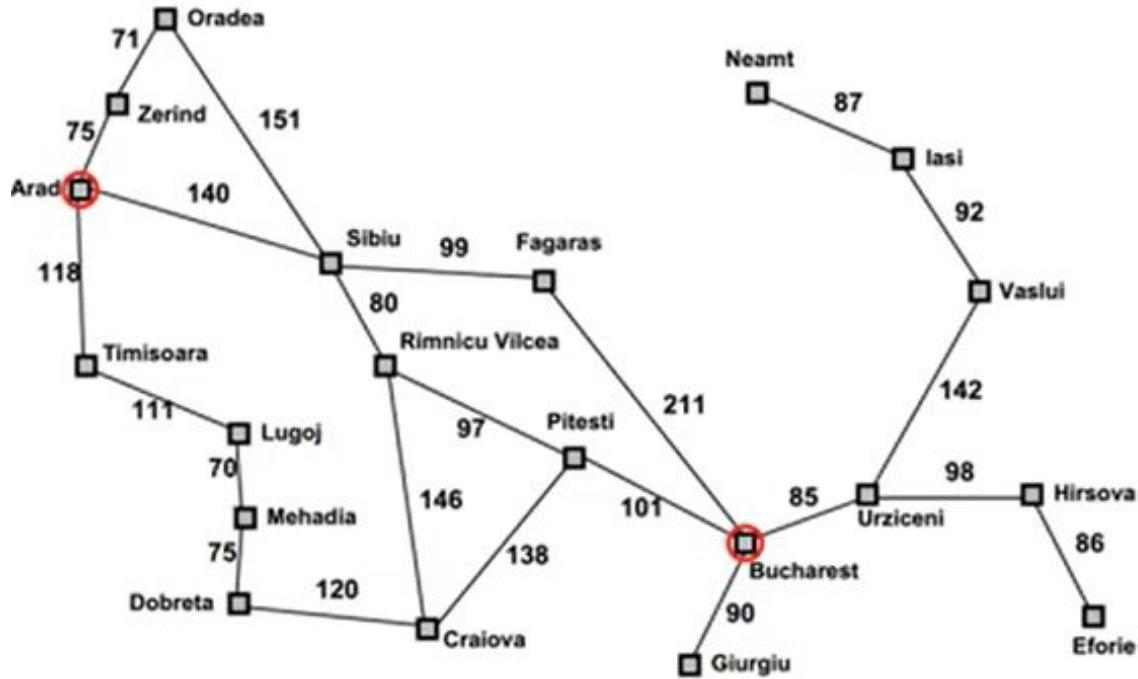
Problema de Busca

- Busca:
 - Estados - Atual e Objetivo
 - Ações
- Problema de busca: achar uma sequência de ações que leva o estado atual ao estado objetivo.
- Um agente com várias ações imediatas pode decidir o que fazer comparando diferentes sequências de ações possíveis.
- Esse processo de procurar pela melhor sequência é chamado de busca
- Formular objetivo - Buscar - Executar

Exemplo: férias na Romênia

De férias na Romênia;
atualmente em Arad.

Voo sai amanhã de Bucareste



Exemplo: férias na Romênia

- Formular objetivo: Estar em Bucareste
- Formular problema:
 - Estados: cidades
 - Ações: dirigir entre as cidades
- Encontrar solução: sequência de cidades. Ex: Arad, Sibiu, Fagaras, Bucareste.

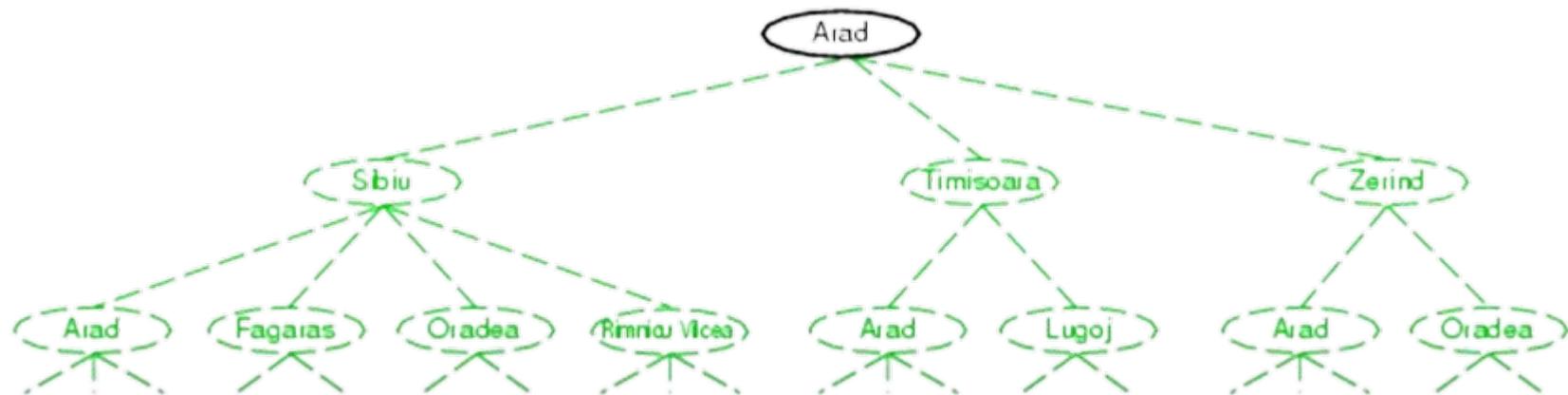
Formulação de Problemas

- Uma solução é uma sequência de ações que levam do estado inicial para o estado objetivo.
- Uma solução ótima é uma solução com menor custo de caminho.
- O conjunto de todos os estados acessíveis a partir de um estado inicial é chamado de espaço de estados.
- Os estados acessíveis são aqueles dados pela função sucessora.
- O espaço de estados pode ser interpretado como um grafo em que os nós são estados e os arcos são ações.

Busca de Soluções

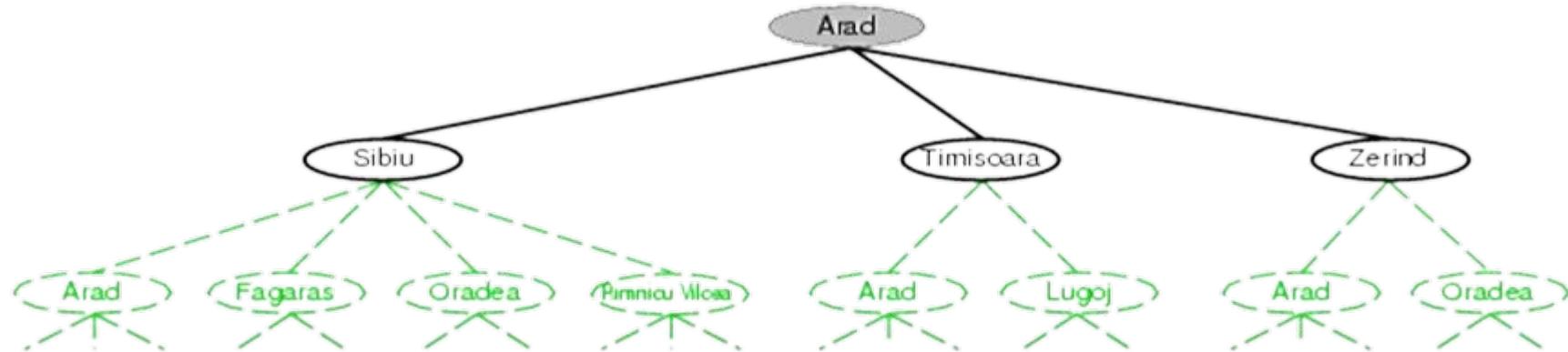
- Ideia: percorrer o espaço de estados a partir de uma árvore de busca.
- Expandir o estado atual aplicando a função sucessor, gerando novos estados.
- Busca: seguir um caminho, deixando os outros para depois.
- A estratégia de busca determina qual caminho seguir.

Exemplo de árvore de busca



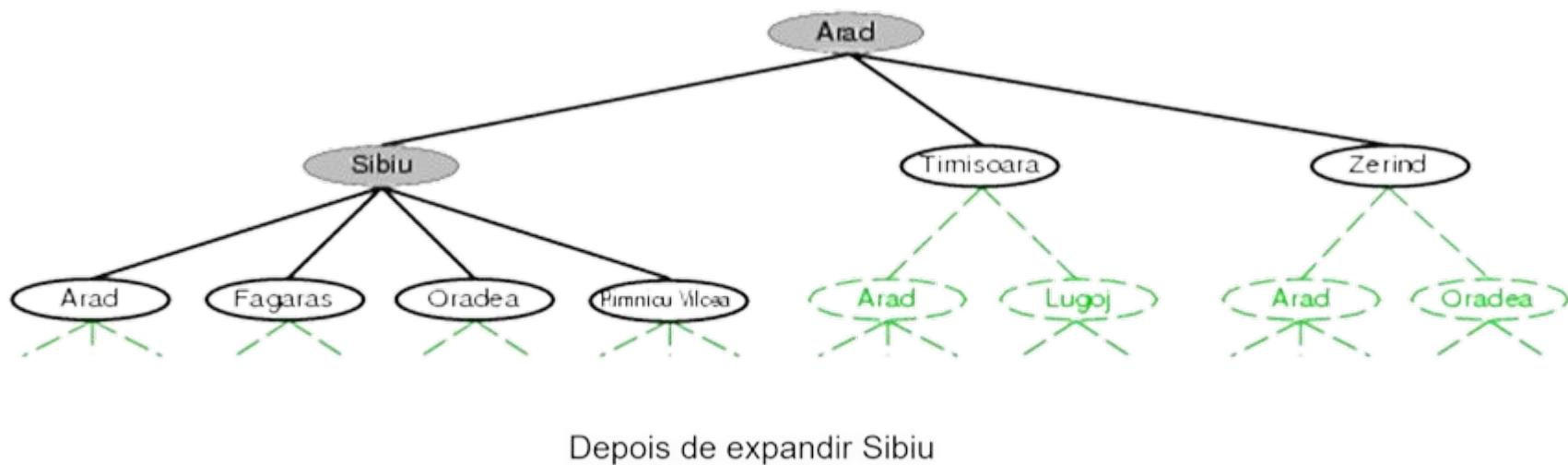
Estado inicial

Exemplo de árvore de busca



Depois de expandir Arad

Exemplo de árvore de busca



Termos importantes

Agente → Quem toma decisões e executa ações no ambiente.

Estado → Uma configuração específica do problema em um momento.

Espaço de estados → Todos os estados possíveis que o agente pode explorar.

Árvore de busca → Representação dos caminhos que o agente pode seguir para resolver o problema.

Nó de busca → Cada ponto na árvore de busca que representa um estado possível.

Objetivo/Meta → O estado final desejado que o agente precisa alcançar.

Ação → O movimento que o agente pode fazer para mudar de estado.

Função sucessor → Define quais estados novos podem ser alcançados a partir de um estado atual.

Por que espaço de estados é importante para IA?

Modela problemas → Permite representar qualquer problema como um conjunto de estados e transições.

Ajuda na tomada de decisão → O agente pode explorar diferentes possibilidades antes de agir.

Permite busca de soluções → A IA pode encontrar um caminho ótimo sem precisar de um código fixo para cada problema.

Base para algoritmos de busca → Métodos como BFS e DFS usam o espaço de estados para encontrar soluções.

Aplicável em diversas áreas → Usado em jogos, robótica, planejamento de rotas e sistemas de diagnóstico.

Facilita a automação → A IA pode resolver problemas sem intervenção humana direta.

Algoritmos de Busca

- **Busca Cega (Sem Inteligência)**
 - **Não usa heurísticas** → Explora todas as possibilidades sem priorizar caminhos melhores.
 - **Segue regras fixas** → Pode ser lento e ineficiente para problemas grandes.
-
- **Busca Informada (Com Inteligência)**
 - **Usa heurísticas** → Estima quais caminhos têm mais chance de levar à solução.
 - **Torna a busca mais eficiente** → Evita explorar estados desnecessários.

Aula 03 - Exercícios - Métodos de Resolução de Problemas e Busca em Espaço de Estados.

Profa. Gabrielly Queiroz

O Problema do Lobo, da Cabra e da Couve

Imagine que você precisa atravessar um rio com um **lobo**, uma **cabra** e uma **couve**. Você tem um barco que só pode carregar você e mais um item (lobo, cabra ou couve). Se o lobo ficar sozinho com a cabra, ele come a cabra. Se a cabra ficar sozinha com a couve, ela come a couve. Como você faz para atravessar todos em segurança? Você está à esquerda do rio.

1. Quais são os **estados** possíveis nesse problema? (Dica: pense na posição de cada item: margem esquerda ou direita do rio.)
2. Quais são os **estados inválidos** que devem ser evitados? (Dica: lobo e cabra sozinhos, ou cabra e couve sozinhos.)
3. Quais são as **ações** possíveis em cada estado? (Dica: o que você pode transportar no barco?)

Resolução

Ver os estados possíveis:

1. **(E, E, E, E)** - Todos na margem esquerda.
2. **(E, E, E, D)** - Pessoa, lobo e cabra na esquerda; couve na direita.
3. **(E, E, D, E)** - Pessoa, lobo e couve na esquerda; cabra na direita.
4. **(E, E, D, D)** - Pessoa e lobo na esquerda; cabra e couve na direita.
5. **(E, D, E, E)** - Pessoa, cabra e couve na esquerda; lobo na direita.
6. **(E, D, E, D)** - Pessoa e cabra na esquerda; lobo e couve na direita.
7. **(E, D, D, E)** - Pessoa e couve na esquerda; lobo e cabra na direita.
8. **(E, D, D, D)** - Pessoa na esquerda; lobo, cabra e couve na direita.
9. **(D, E, E, E)** - Pessoa na direita; lobo, cabra e couve na esquerda.
10. **(D, E, E, D)** - Pessoa e couve na direita; lobo e cabra na esquerda.
11. **(D, E, D, E)** - Pessoa e cabra na direita; lobo e couve na esquerda.
12. **(D, E, D, D)** - Pessoa, cabra e couve na direita; lobo na esquerda.
13. **(D, D, E, E)** - Pessoa e lobo na direita; cabra e couve na esquerda.
14. **(D, D, E, D)** - Pessoa, lobo e couve na direita; cabra na esquerda.
15. **(D, D, D, E)** - Pessoa, lobo e cabra na direita; couve na esquerda.
16. **(D, D, D, D)** - Todos na margem direita.

Resolução

Um estado é **inválido** se:

1. O **lobo e a cabra** ficam sozinhos em uma margem (o lobo come a cabra).
2. A **cabra e a couve** ficam sozinhos em uma margem (a cabra come a couve).

Estados Válidos:

1. **(E, E, E, E)** - Todos na esquerda (válido).
2. **(E, E, E, D)** - Lobo e cabra na esquerda; couve na direita (válido).
3. **(E, E, D, E)** - Lobo e couve na esquerda; cabra na direita (válido).
4. **(E, E, D, D)** - Lobo na esquerda; cabra e couve na direita (**inválido**: cabra e couve sozinhos).
5. **(E, D, E, E)** - Cabra e couve na esquerda; lobo na direita (**inválido**: cabra e couve sozinhos).
6. **(E, D, E, D)** - Cabra na esquerda; lobo e couve na direita (válido).
7. **(E, D, D, E)** - Couve na esquerda; lobo e cabra na direita (**inválido**: lobo e cabra sozinhos).
8. **(E, D, D, D)** - Pessoa na esquerda; lobo, cabra e couve na direita (válido).
9. **(D, E, E, E)** - Pessoa na direita; lobo, cabra e couve na esquerda (válido).
10. **(D, E, E, D)** - Lobo e cabra na esquerda; couve na direita (válido).
11. **(D, E, D, E)** - Lobo e couve na esquerda; cabra na direita (válido).
12. **(D, E, D, D)** - Lobo na esquerda; cabra e couve na direita (**inválido**: cabra e couve sozinhos).
13. **(D, D, E, E)** - Lobo na direita; cabra e couve na esquerda (**inválido**: cabra e couve sozinhos).
14. **(D, D, E, D)** - Lobo e couve na direita; cabra na esquerda (válido).
15. **(D, D, D, E)** - Lobo e cabra na direita; couve na esquerda (**inválido**: lobo e cabra sozinhos).
16. **(D, D, D, D)** - Todos na direita (válido).

Os estados válidos são:

1. **(E, E, E, E)**
2. **(E, E, E, D)**
3. **(E, E, D, E)**
4. **(E, D, E, D)**
5. **(E, D, D, D)**
6. **(D, E, E, E)**
7. **(D, E, E, D)**
8. **(D, E, D, E)**
9. **(D, D, E, D)**
10. **(D, D, D, D)**

Resolução

Ações Possíveis

A pessoa pode realizar as seguintes ações:

1. **Atravessar sozinha** (sem levar nenhum item).
2. **Levar o lobo.**
3. **Levar a cabra.**
4. **Levar a couve.**

Cada ação altera a posição da pessoa e do item transportado (se houver).

Resolução

Passo 1: Estado Inicial

- Estado: (E, E, E, E).
- Ação: Levar a **cabra** para a margem direita.
- Novo estado: (D, E, D, E).

Passo 2: Estado Atual

- Estado: (D, E, D, E).
- Ação: Voltar **sozinho(a)** para a margem esquerda.
- Novo estado: (E, E, D, E).

Passo 3: Estado Atual

- Estado: (E, E, D, E).
- Ação: Levar o **lobo** para a margem direita.
- Novo estado: (D, D, D, E).

Passo 4: Estado Atual

- Estado: (D, D, D, E).
- Ação: Voltar com a **cabra** para a margem esquerda.
- Novo estado: (E, D, E, E).

Passo 5: Estado Atual

- Estado: (E, D, E, E).
- Ação: Levar a **couve** para a margem direita.
- Novo estado: (D, D, E, D).

Passo 6: Estado Atual

- Estado: (D, D, E, D).
- Ação: Voltar **sozinho(a)** para a margem esquerda.
- Novo estado: (E, D, E, D).

Passo 7: Estado Atual

- Estado: (E, D, E, D).
- Ação: Levar a **cabra** para a margem direita.
- Novo estado: (D, D, D, D).

Resolução

1. $(E, E, E, E) \rightarrow$ Levar a cabra $\rightarrow (D, E, D, E)$.
2. $(D, E, D, E) \rightarrow$ Voltar sozinho(a) $\rightarrow (E, E, D, E)$.
3. $(E, E, D, E) \rightarrow$ Levar o lobo $\rightarrow (D, D, D, E)$.
4. $(D, D, D, E) \rightarrow$ Voltar com a cabra $\rightarrow (E, D, E, E)$.
5. $(E, D, E, E) \rightarrow$ Levar a couve $\rightarrow (D, D, E, D)$.
6. $(D, D, E, D) \rightarrow$ Voltar sozinho(a) $\rightarrow (E, D, E, D)$.
7. $(E, D, E, D) \rightarrow$ Levar a cabra $\rightarrow (D, D, D, D)$.

$(E, E, E, E) \rightarrow (D, E, D, E) \rightarrow (E, E, D, E) \rightarrow (D, D, D, E) \rightarrow (E, D, E, E) \rightarrow (D, D, E, D) \rightarrow (E, D, E, D) \rightarrow (D, D, D, D)$

Aula 04 - Busca sem Informação: BFS, DFS, Busca de Custo Uniforme.

Profa. Gabrielly Queiroz

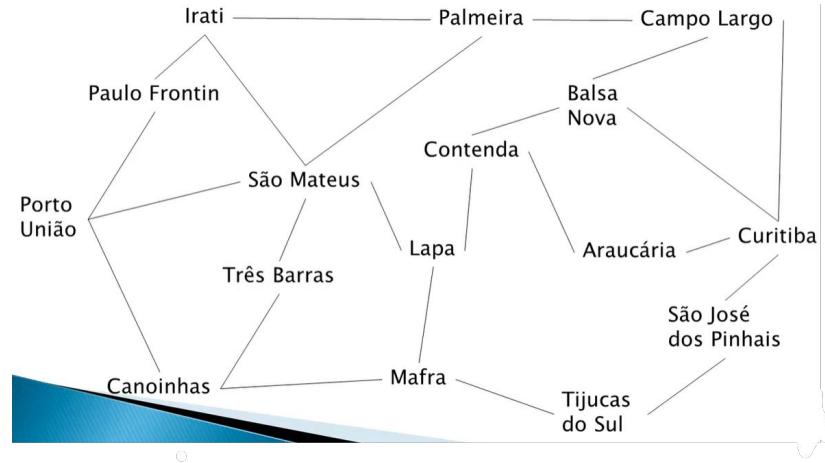
Busca sem informação

A busca sem informação, também chamada de **busca cega**, é um conjunto de estratégias de busca que não utilizam conhecimento adicional sobre o problema além da própria estrutura do grafo. Esses algoritmos exploram os estados do espaço de busca de forma sistemática, sem heurísticas.

- Não leva em conta informações sobre o problema.
- Busca cega/aleatória.

Os três principais algoritmos dessa categoria são:

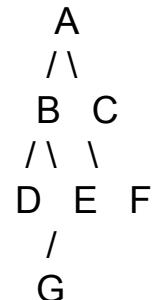
- **BFS (Busca em Largura - Breadth-First Search)**
- **DFS (Busca em Profundidade - Depth-First Search)**
- **Busca de Custo Uniforme (Uniform Cost Search - UCS)**



Representação do Problema

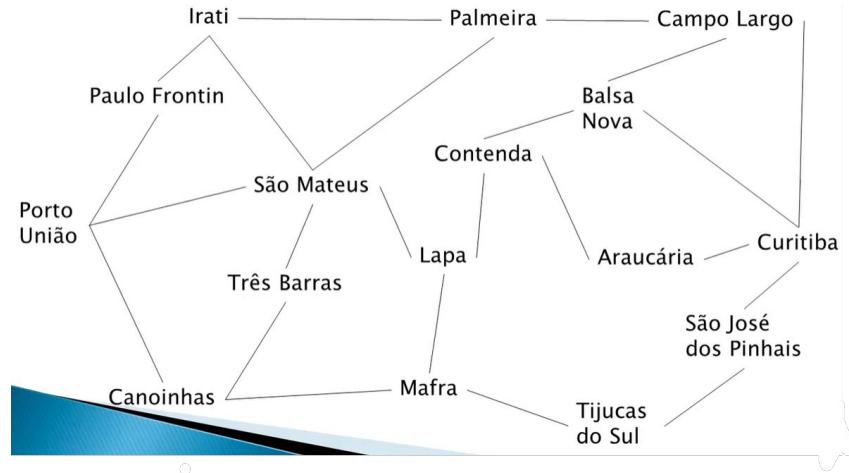
Antes de explorarmos os algoritmos, precisamos entender como representamos um problema de busca. Normalmente, um problema é modelado como um **grafo**, onde:

- Cada **nó** representa um estado do problema.
- Cada **aresta** representa uma transição entre estados.
- O **nó inicial** é o ponto de partida da busca.
- O **nó objetivo** é o estado final desejado.
- O **custo** pode ser associado às transições entre estados.



Algoritmo genérico

- Fronteira do espaço de estados
 - Algoritmo: Atribuir à fronteira (os nós a serem explorados) o estado inicial
1. Selecionar o primeiro nó da fronteira do espaço de estado
 2. Testar se o nó é um estado final (objetivo)
 - a. Caso sim, a busca termina como sucesso
 3. Gerar um novo conjunto de estados
 4. Remover o nó explorado da fronteira
 5. Inserir os nós gerados na fronteira e voltar ao passo 1



Algoritmo genérico

O que muda de um algoritmo para o outro?

A ordem em que as fronteiras são inseridas!

Remover um nó da fronteira (a ordem de remoção define o algoritmo!).

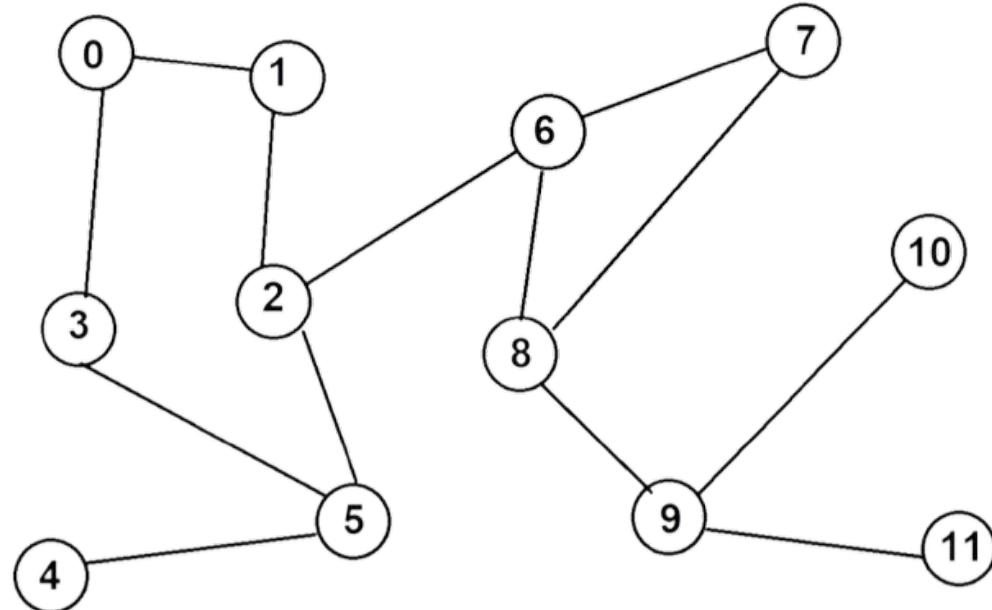
Busca em Largura (BFS - Breadth-First Search)

Explora todos os nós no mesmo nível antes de avançar para níveis mais profundos.

Utiliza uma **fila** (FIFO - First In, First Out) para armazenar os nós a serem explorados.

Garante a solução mais curta (mínimo de passos) se o custo de cada aresta for uniforme.

1. O estado inicial recebe um pai especial (0).
2. Cada nó expandido recebe como pai o nó que o descobriu primeiro (0+1).
3. Para encontrar o melhor caminho, percorremos os pais a partir do nó objetivo até o estado inicial.



Busca em Profundidade (DFS - Depth-First Search)

Explora um caminho completamente antes de retornar e tentar outro.

Utiliza uma **pilha** (**LIFO** - Last In, First Out).

Pode seguir um caminho errado por muito tempo antes de corrigir a rota.

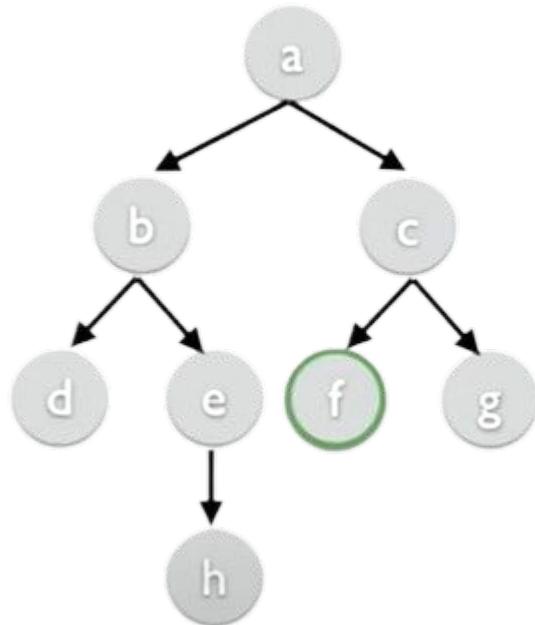
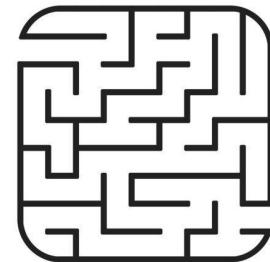
Cada nó expandido recebe como **pai** o nó que o descobriu primeiro, seguindo sempre um **caminho profundo antes de voltar atrás**.

A reconstrução do caminho segue a mesma lógica: ao encontrar o objetivo, percorremos os pais até o estado inicial.

Vantagens e Desvantagens

Usa menos memória que BFS.

Não garante o caminho mais curto.



Busca de Custo Uniforme (Uniform Cost Search - UCS)

Sempre expande o nó com menor custo acumulado.

Utiliza uma **fila de prioridade (heap)** para armazenar os nós.

Considera o custo de cada aresta, diferente do BFS.

Passo a Passo

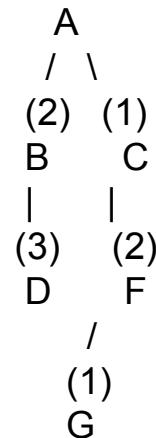
1. Insere o nó inicial na fila de prioridade com custo 0.
2. Remove o nó com menor custo e verifica se é o objetivo.
3. Se não for, adiciona seus vizinhos, somando os custos.
4. Repete o processo até encontrar a solução ou a fila ficar vazia.

Vantagens e Desvantagens

Garante o caminho de menor custo.

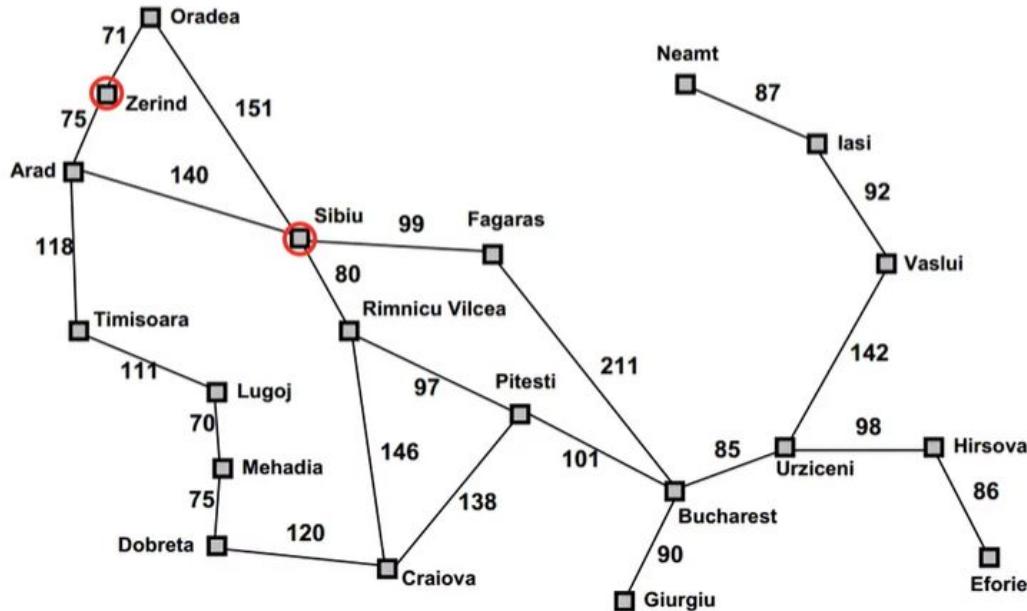
Pode ser lento se os custos forem muito altos e variados.

Exemplo - Grafo com custos

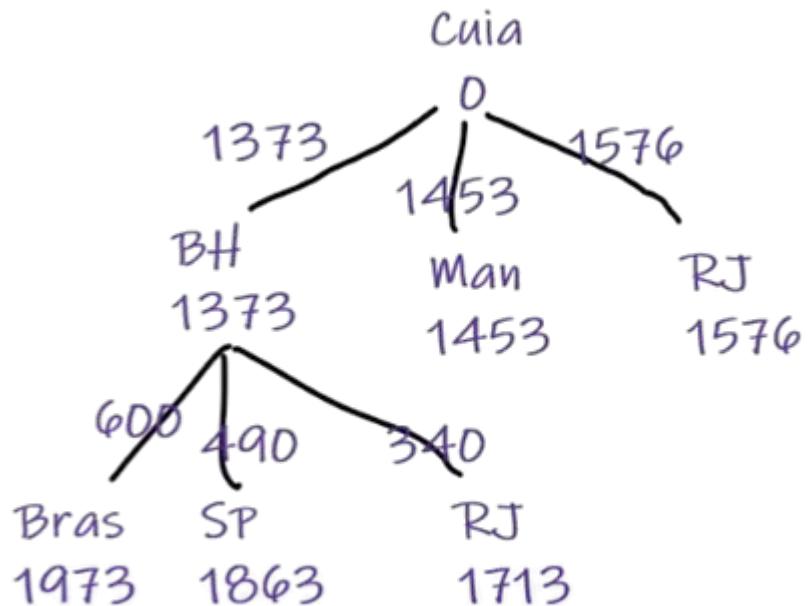


Busca de Custo Uniforme (Uniform Cost Search - UCS)

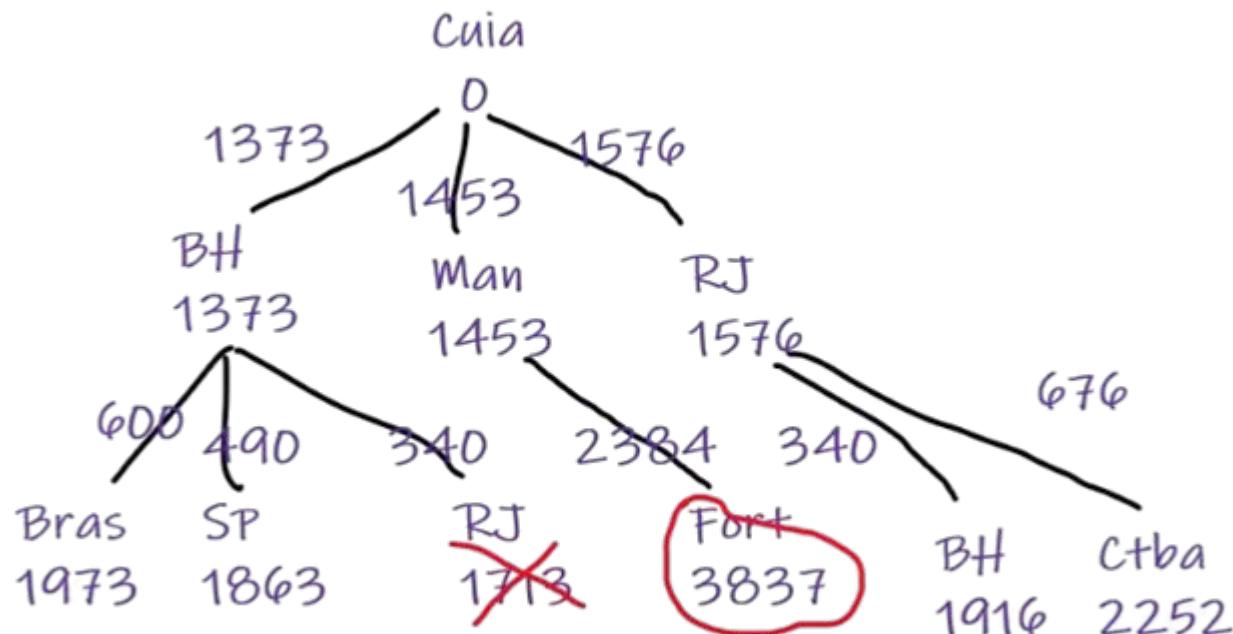
- Problema: Encontrar caminho de Zerind a Sibiu
 - Estado Inicial: Zerind.
 - Estado Objetivo: Sibiu.



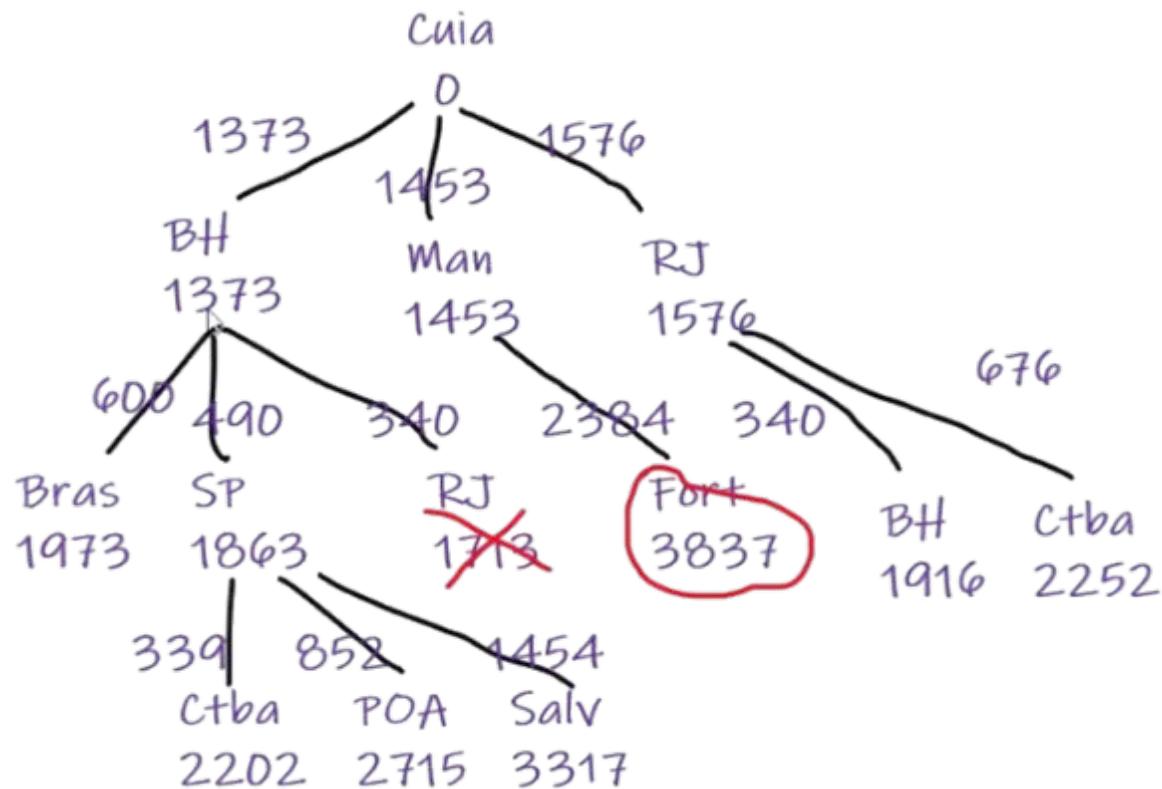
Busca de Custo Uniforme (Uniform Cost Search - UCS)



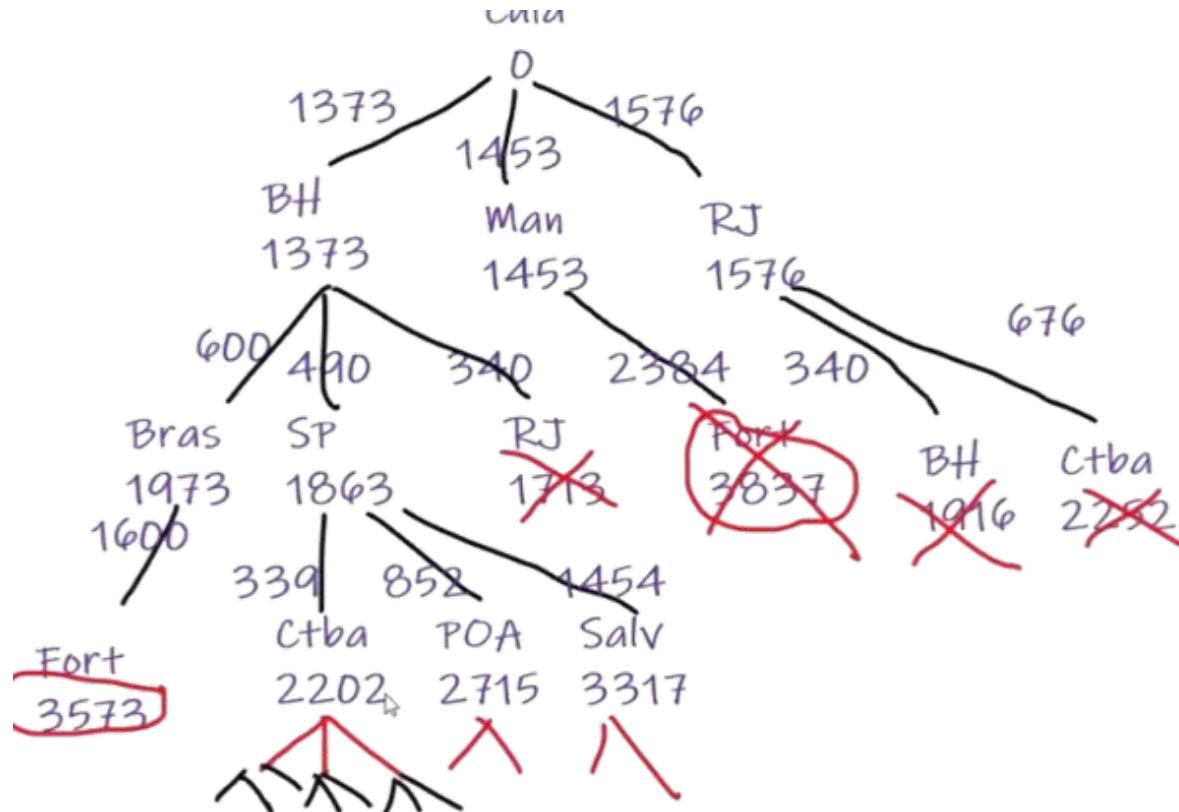
Busca de Custo Uniforme (Uniform Cost Search - UCS)



Busca de Custo Uniforme (Uniform Cost Search - UCS)



Busca de Custo Uniforme (Uniform Cost Search - UCS)



Busca de Custo Uniforme (Uniform Cost Search - UCS)

Quando parar?

Quando todos os custos forem maior que 3573

BFS X DFS X Custo Uniforme x UCS

Critério	BFS (Largura)	DFS (Profundidade)	UCS (Custo Uniforme)
Estrutura	Fila (FIFO)	Pilha (LIFO)	Fila de prioridade
Ordem de Exploração	Nível por nível	Um caminho por vez	Menor custo primeiro
Melhor Caminho?	Se custo uniforme	Não	Sempre
Memória	Alta	Baixa	Alta
Aplicação	Caminho mais curto	Jogos	Caminho de menor custo

Aula 05 - Algoritmos: Busca sem Informação: BFS, DFS, Busca de Custo Uniforme.

Profa. Gabrielly Queiroz

Introdução ao Python

Google Colab: <https://colab.research.google.com/>

Códigos da Aula

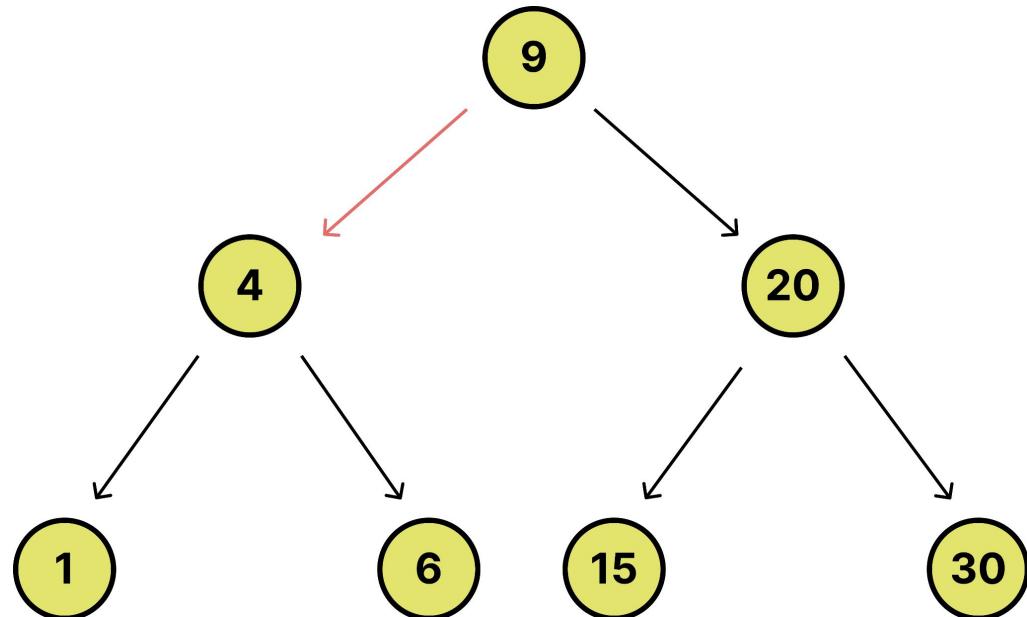
https://colab.research.google.com/drive/1TWJwjtolB1uog6_I20Ms4mCt3yUe3oMu?usp=sharing

Como ele percorre? BFS

Passo	Fila	Nó visitado
1	[1]	1
2	[2, 3, 4]	2
3	[3, 4, 5, 6]	3
4	[4, 5, 6, 7]	4
5	[5, 6, 7, 8]	5
6	[6, 7, 8, 9, 10]	6
7	[7, 8, 9, 10, 11]	7
8	[8, 9, 10, 11, 12]	8
9	[9, 10, 11, 12]	9
10	[10, 11, 12]	10
11	[11, 12]	11
12	[12]	12 (objetivo encontrado!)

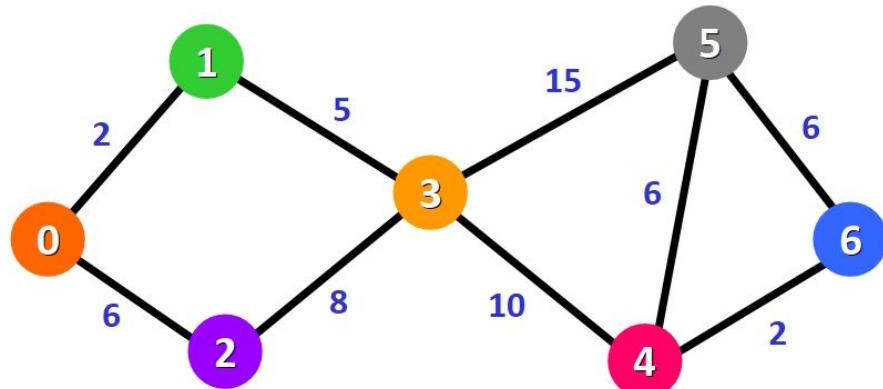
Atividade

Aplique os Algoritmo de Busca BFS e DFS para encontrar o menor caminho do grafo (1 até 30) ao lado.



Atividade

Aplique os Algoritmo de Busca em Custo Uniforme para encontrar o menor caminho do grafo (0 até 6) ao lado.



Aula 06 - Uso de Heurísticas

Profa. Gabrielly Queiroz

Introdução às Heurísticas

O que são heurísticas?

As heurísticas são estratégias usadas para encontrar soluções rapidamente quando uma busca completa ou exata seria muito demorada. Elas ajudam a reduzir o número de possibilidades analisadas, utilizando estimativas inteligentes para guiar a tomada de decisão.

Exemplos de heurísticas no dia a dia

- Escolher a fila mais curta no supermercado sem calcular exatamente o tempo de espera.
- Usar um atalho no trânsito sem saber com certeza se é a melhor opção.
- Diagnosticar uma doença baseada nos sintomas mais comuns em vez de considerar todas as possibilidades médicas.

A Importância das Heurísticas na IA

Em muitos problemas de IA, **o espaço de busca é gigantesco** e encontrar a solução ótima pode ser impraticável. Heurísticas ajudam a **reduzir o número de opções analisadas**.

As heurísticas são amplamente utilizadas em diversos domínios da IA, como:

- **Algoritmos de busca e otimização**: utilizados em jogos, navegação e problemas de roteamento.
- **Sistemas especialistas**: aplicadas para tomada de decisões baseadas em regras.
- **Aprendizado de Máquina**: influenciam escolhas de features, hiperparâmetros e estratégias de treinamento.
- **Planejamento e raciocínio**: ajudam a definir prioridades e caminhos mais prováveis para atingir um objetivo.

Vantagens e Desvantagens das Heurísticas

Vantagens:

- Encontram soluções rapidamente
- Podem ser ajustadas para diferentes problemas
- Dependem da qualidade da heurística escolhida

Desvantagens:

- Não garantem a melhor solução
- Reduzem o custo computacional
- Podem ser enviesadas ou limitadas

Principais Algoritmos

Busca Sem Informação:

- O algoritmo **não tem conhecimento prévio** sobre onde está o objetivo, então ele simplesmente segue regras fixas para explorar os nós. Explora cegamente.

Busca Com Informação:

- **Usa heurística $h(n)$** → O algoritmo usa uma função que **estima o custo restante até o objetivo** para guiar a busca de maneira mais inteligente.
 - **Mais eficiente** → Explora primeiro os caminhos mais promissores, economizando tempo e recursos.
 - **Pode encontrar a solução ótima** → Se a heurística for bem escolhida, o algoritmo pode encontrar o caminho mais curto rapidamente.
-
- ❖ **Busca Gulosa (Greedy Best-First Search)**
 - ❖ **Algoritmo A***

Função Heurística

A função heurística é uma aproximação do custo para alcançar o objetivo a partir de um determinado estado. Em outras palavras, é uma forma de **estimar o melhor caminho** sem necessariamente explorá-lo por completo.

Matematicamente, podemos definir uma função heurística como:

$h(n)$ = estimativa do custo do nó n até o nó objetivo

Ela **não precisa ser exata**, mas deve ser rápida de calcular e fornecer uma estimativa razoável. Uma boa heurística reduz o número de estados explorados, acelerando a busca.

Busca Gulosa (Greedy Best-First Search)

A **Busca Gulosa** é um algoritmo de busca informada que utiliza apenas a função heurística para decidir qual nó explorar primeiro. Ele escolhe sempre o próximo estado que **parece** estar mais próximo do objetivo, sem considerar o custo real do caminho.

A função usada na Busca Gulosa é:

$$f(n)=h(n)$$

Ou seja, o algoritmo **sempre escolhe o nó com o menor valor de $h(n)$** .

Exemplo

Imagine um problema de navegação onde um agente precisa chegar ao destino. A Busca Gulosa escolheria o próximo ponto baseado **apenas na distância ao destino**, ignorando obstáculos ou caminhos mais longos.

Vantagens da Busca Gulosa:

Simples de implementar.

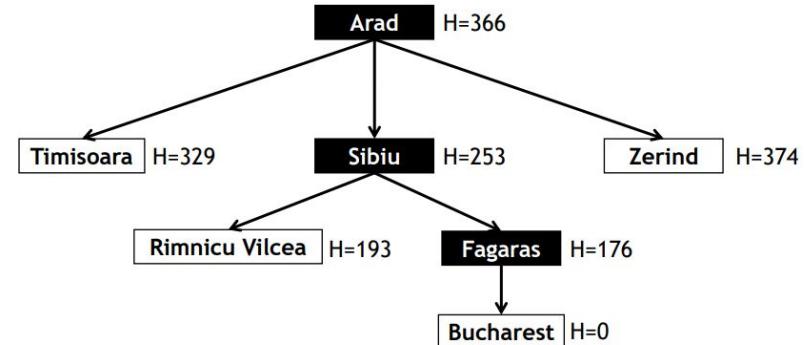
Pode encontrar soluções rapidamente.

Desvantagens da Busca Gulosa:

Pode ficar presa em mínimos locais, escolhendo um caminho que parece melhor a curto prazo, mas que na verdade é pior.

Não garante encontrar o caminho mais curto (solução ótima).

- **Busca gulosa:** expande o nó com menor valor de $h(x)$.



Algoritmo A* (A-Star)

O A* é uma melhoria da Busca Gulosa, combinando a heurística com o custo real do caminho percorrido. Isso resolve o problema da Busca Gulosa de ficar presa em mínimos locais.

A função utilizada pelo A* é:

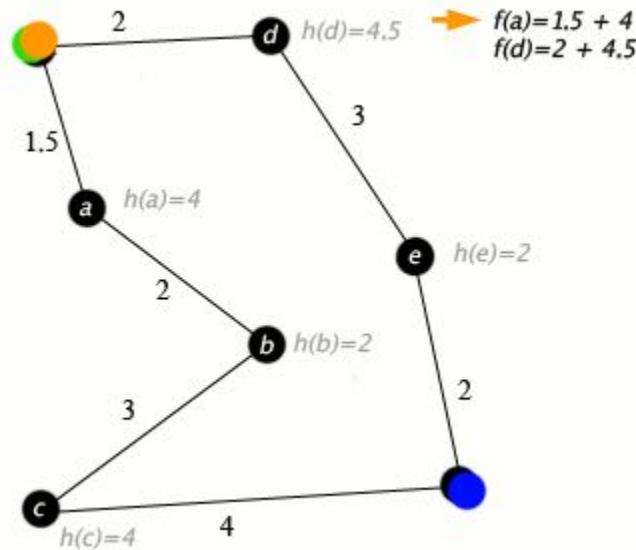
$$f(n) = g(n) + h(n)$$

Onde:

- **g(n)** → Custo real do caminho percorrido até o nó n.
- **h(n)** → Estimativa heurística do custo restante até o objetivo.

O algoritmo A* busca equilibrar **o custo percorrido** com **a previsão do custo restante**, tornando-o um dos algoritmos mais eficientes para busca.

Algoritmo A* (A-Star)



Limitações das Heurísticas

Apesar de úteis, heurísticas **não são perfeitas** e podem levar a erros. Algumas limitações incluem:

1. Podem ser enviesadas

- Se uma heurística for baseada em dados ruins, ela pode levar a decisões ruins.
- Exemplo: Se um carro autônomo aprender que uma área tem menos pedestres e, por isso, reduz a atenção, ele pode errar ao não considerar um dia atípico.

2. Podem não funcionar para todos os problemas

- Algumas heurísticas funcionam bem em um cenário, mas são ineficazes em outros.

3. Podem ser manipuladas

- Algoritmos de recomendação podem ser enganados (exemplo: sistemas de propaganda podem "forçar" recomendações irrelevantes porque seguem heurísticas simples).

Aula 07 - Uso de Heurísticas, Algoritmo Guloso e A*.

Profa. Gabrielly Queiroz

Heurística

A heurística é uma abordagem utilizada para resolver problemas de forma eficiente, ainda que não garanta uma solução ótima. Ela é empregada principalmente quando:

- O espaço de busca é muito grande, inviabilizando uma solução exaustiva.
- O problema pode ser resolvido aproximadamente, sem a necessidade de exatidão.
- Há uma necessidade de respostas rápidas, como em jogos de estratégia e navegação.

Uma heurística pode ser vista como uma função de estimativa que guia um algoritmo de busca na direção correta. Em muitos problemas, como na busca de rotas ou em jogos, uma boa heurística pode reduzir drasticamente o tempo de processamento.

Exemplo de Heurística:

Ao tentar encontrar o caminho mais curto entre dois pontos em um mapa, uma heurística comum é a distância em linha reta entre os pontos. Embora esse valor não represente necessariamente o menor caminho real, ele serve como um bom indicativo.

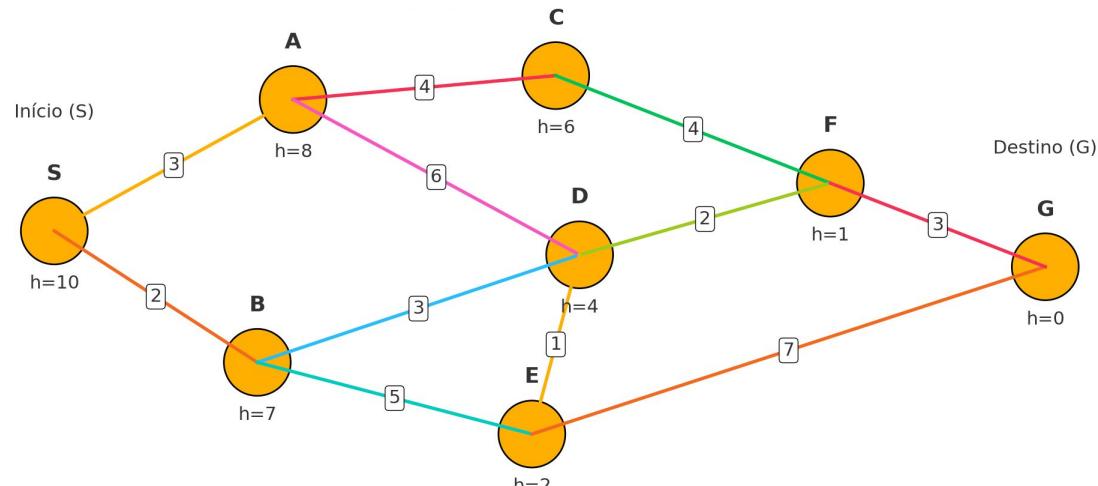
Exemplo Simples: Encontrar o caminho mais curto entre cidades

[https://colab.research.google.com/drive/18jYGeo7cLvVKc-HK3XVPKZLeAGrGHifl
?usp=sharing](https://colab.research.google.com/drive/18jYGeo7cLvVKc-HK3XVPKZLeAGrGHifl?usp=sharing)

Atividade

Implemente em Python os algoritmos de **Busca Gulosa** e **A*** para encontrar o caminho entre **S** (início) e **G** (destino) no mapa ao lado. Mostre o caminho encontrado, o custo total e compare os resultados dos dois algoritmos.

As distâncias reais estão nas arestas e a heurística (distância estimada até G) está embaixo de cada cidade.



Aula 08 - Satisfação de Restrições

Profa. Gabrielly Queiroz

Satisfação de Restrições - Constraint Satisfaction Problems - CSP

No lugar de encontrar um caminho em um espaço de busca, queremos **encontrar uma solução que satisfaça um conjunto de restrições específicas.**

CSP é um problema onde o objetivo é encontrar um estado ou conjunto de estados que satisfaça todas as restrições impostas a um conjunto de variáveis.

Estrutura de um Problema CSP

Componentes Básicos:

Em problemas de Satisfação de Restrições, temos:

- **Variáveis:** O que queremos resolver ou encontrar. Ex.: X, Y, Z.
- **Domínios:** Os valores possíveis que cada variável pode assumir. Ex.: $X \in \{1, 2, 3\}$, $Y \in \{1, 2, 3\}$, $Z \in \{1, 2, 3\}$.
- **Restrições:** Regras que determinam quais combinações de valores são aceitáveis. Ex.: $X \neq Y$, $Y < Z$.

Formalização Matemática:

Um CSP é definido como um triplo (X, D, C) , onde:

- $X = \{X_1, X_2, \dots, X_n\}$ é o conjunto de variáveis.
- $D = \{D_1, D_2, \dots, D_n\}$ é o conjunto de domínios de cada variável X_i .
- $C = \{C_1, C_2, \dots, C_m\}$ é o conjunto de restrições.

Exemplo:

Problema de Colorização de Mapas. Variáveis são regiões a serem coloridas, domínios são cores disponíveis, e restrições garantem que regiões adjacentes tenham cores diferentes.

Exemplo

Temos o Mapa ao lado, mas podemos pintar cada região com apenas três cores: verde, vermelho e azul.

As cores não podem ser a mesma cor para os seus vizinhos.



Exemplo

Variáveis:

- N,NE,SE,CO,S.

Domínios:

- {verde,vermelho,azul}

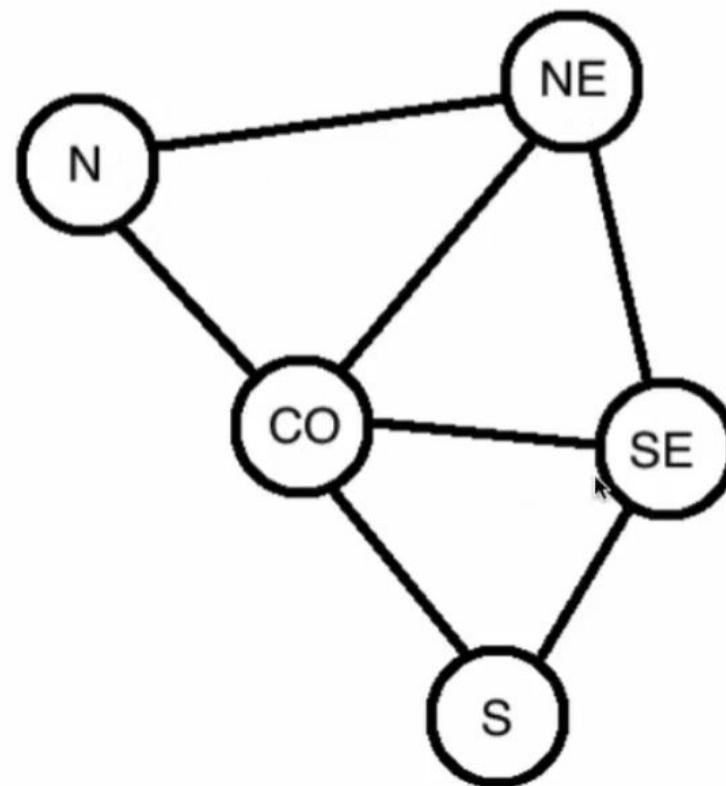
Restrições:

- Regiões adjacentes devem ter cores diferentes.
- Por exemplo, $N \neq NE$
- Relações válidas entre variáveis adjacentes (N,NE)

Soluções:

- Conjuntos de atribuições que sejam completos e consistentes, ou seja, todas as variáveis são atribuídas a cores, e todas as restrições são satisfeitas.

Exemplo



Tipos de CSPs

CSPs Discretos:

- Domínios finitos
- Domínios infinitos

CSPs Contínuos:

Variáveis que assumem valores contínuos, como problemas de engenharia.

CSPs Estruturados:

- CSPs de grande escala que podem ser divididos em subproblemas menores.

CSPs Discretos (Discrete CSPs)

Aqui, as variáveis têm um conjunto limitado (finito) ou ilimitado (infinito) de valores possíveis.

- ◆ Domínios Finitos (Comumente resolvidos por técnicas como Backtracking, Arc-Consistency, etc.)
 - Coloração de Mapas (Map Coloring):
 - Problema clássico onde queremos colorir um mapa usando o menor número possível de cores.
 - Variáveis: Regiões do mapa (ex.: A, B, C, D).
 - Domínios: {Vermelho, Verde, Azul}.
 - Restrições: Regiões adjacentes não podem ter a mesma cor.

Domínios Infinitos (Normalmente tratados por discretização ou heurísticas específicas)

- Quebra-cabeças Numéricos (Ex.: Equações Algébricas):
 - Variáveis: Variáveis numéricas (ex.: x,y,z).
 - Domínios: Todos os números inteiros ou reais possíveis.
 - Restrições: Regras matemáticas que devem ser satisfeitas (ex.: $x^2+y^2=25$).

CSPs Contínuos (Continuous CSPs)

Nestes problemas, as variáveis podem assumir **qualquer valor dentro de um intervalo contínuo**.

- ♦ **Exemplos:**

- **Otimização em Engenharia:**
 - Problemas onde queremos minimizar ou maximizar funções contínuas.
 - Variáveis: Parâmetros físicos (ex.: pressão, temperatura, corrente elétrica).
 - Domínios: Intervalos contínuos (ex.: $0 \leq x \leq 100$).
 - Restrições: Fórmulas que devem ser respeitadas (ex.: $f(x,y) \leq 10$).
- **Robótica:**
 - Planejamento de movimento de um robô.
 - Variáveis: Coordenadas contínuas do espaço (x, y, z, θ).
 - Restrições: Evitar colisões, respeitar limites físicos.
- **Aprendizado de Máquina (Otimização de Funções):**
 - Ajuste de pesos em redes neurais.
 - Variáveis: Pesos contínuos.
 - Domínios: Intervalos específicos (ex.: $[0, 1]$).
 - Restrições: Funções de custo ou perda a serem minimizadas.

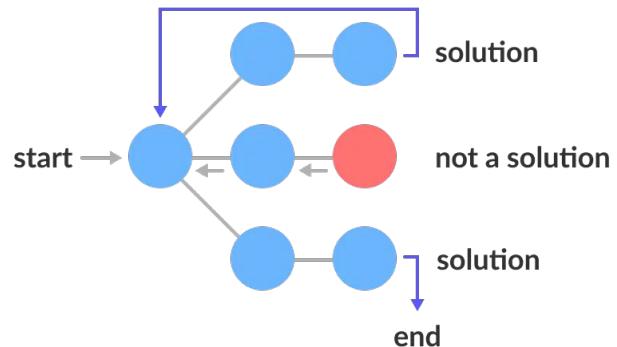
Algoritmo para resolução - Backtracking

Algoritmo Backtracking

O **Backtracking (Retrcesso)** é um algoritmo de **busca sistemática** usado para resolver problemas de decisão, otimização e satisfação de restrições, como os **Problemas de Satisfação de Restrições (CSPs)**.

Passos do Algoritmo:

1. **Escolher uma variável para atribuir.**
2. **Atribuir um valor válido à variável** do seu domínio.
3. **Verificar se a atribuição é válida:**
 - o Se for válida, continuar para a próxima variável.
 - o Se não for válida, **desfazer a atribuição (retroceder)** e tentar outro valor.
4. **Repetir até que:**
 - o Uma solução válida seja encontrada (todas as variáveis atribuídas sem violar as restrições).
 - o Todas as possibilidades sejam tentadas e nenhuma solução seja encontrada.



Exemplo - Mapa

Escolher uma variável para atribuir.

- Vamos começar pela região A.

Atribuir um valor válido à variável do seu domínio.

- Tentamos a cor **Vermelho** para A.
- Atribuição atual:
 - A = Vermelho
 - B = ?
 - C = ?
 - D = ?

Verificar se a atribuição é válida:

- Como A é a primeira variável, é válida até o momento.
- Avançamos para B.

Atribuir um valor para B:

- Tentamos a cor **Vermelho** para B.
- Como A e B são vizinhos, isso é **inválido!** (Retrocede)
- Tentamos a cor **Verde** para B (válido).
- Atribuição atual:
 - A = Vermelho
 - B = Verde
 - C = ?
 - D = ?

Continuar o processo:

- Atribuímos cores válidas para C e D.
- Testamos as cores possíveis e aplicamos **retrocesso sempre que ocorrer um conflito.**

Exercício

Cinco amigos vão organizar os livros de uma biblioteca, mas cada um só pode pegar **um tipo específico de livro**. A tarefa é descobrir **quem ficou com qual tipo de livro** baseado nas seguintes pistas.

Amigos:

- Ana, Bruno, Carlos, Daniela, Eduardo.

Tipos de Livros:

- Romance, Mistério, Ficção Científica, História, Poesia.

Pistas:

1. Ana não pegou o livro de **Ficção Científica** nem de **História**.
2. Carlos ficou com o livro de **Mistério**.
3. Bruno não ficou com o livro de **Romance**.
4. Eduardo ficou com o livro de **Ficção Científica**.
5. Daniela NÃO FICOU COM **Mistério**.

Objetivo:

Descobrir **quem ficou com qual tipo de livro**.

Resolução

Amigos (Variáveis):

- Ana, Bruno, Carlos, Daniela, Eduardo.

Tipos de Livros (Domínio):

- Romance, Mistério, Ficção Científica, História, Poesia.

Pistas (Restrições):

1. Ana não pegou o livro de Ficção Científica nem de História.
2. Carlos ficou com o livro de Mistério.
3. Bruno não ficou com o livro de Romance.
4. Eduardo ficou com o livro de Ficção Científica.
5. Daniela não gosta de Mistério.

Escolher uma variável para atribuir.

- Vamos começar atribuindo livros para **cada amigo**, um de cada vez.

Atribuir um valor válido à variável do seu domínio.

- Testamos **todos os livros possíveis** para cada amigo.

Verificar se a atribuição é válida:

- Cada atribuição é testada contra as **restrições** (pistas) dadas.

Retroceder se necessário:

- Se alguma atribuição não é válida, desfazemos e tentamos outra.

Resolução

Etapa 1: Atribuir livro para Carlos (Pista 2).

- Carlos = **Mistério** (Válido).
- Livros disponíveis: **Romance, Ficção Científica, História, Poesia.**

Etapa 2: Atribuir livro para Eduardo (Pista 4).

- Eduardo = **Ficção Científica** (Válido).
- Livros disponíveis: **Romance, História, Poesia.**

Etapa 3: Atribuir livro para Ana (Pista 1).

- Ana NÃO pode ter: **Ficção Científica, História.**
- Testar:
 - Ana = **Romance** (Válido).
- Livros disponíveis: **História, Poesia.**

Etapa 4: Atribuir livro para Bruno (Pista 3).

- Bruno NÃO pode ter: **Romance.**
- Testar:
 - Bruno = **História** (Válido).
- Livros disponíveis: **Poesia.**

Etapa 5: Atribuir livro para Daniela (Pista 5).

- Daniela NÃO pode ter: **Mistério.**
- Sobram os livros: **Poesia.**
- Daniela = **Poesia** (Válido).
 -

Resolução

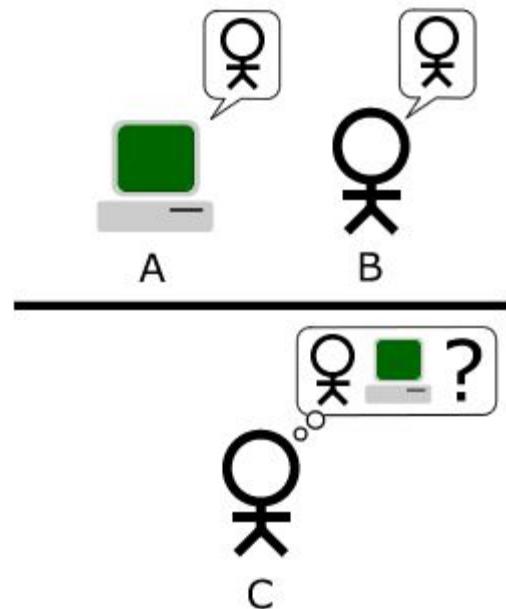
Amigo	Livro
Ana	Romance
Bruno	História
Carlos	Mistério
Daniela	Poesia
Eduardo	Ficção Científica

Aula 10 - Fundamentos de Aprendizado de Máquina

Profa. Gabrielly Queiroz

Introdução

- Aprendizado de Máquina é um ramo da Inteligência Artificial que se concentra em criar sistemas capazes de aprender e melhorar automaticamente com a experiência, sem serem explicitamente programados.



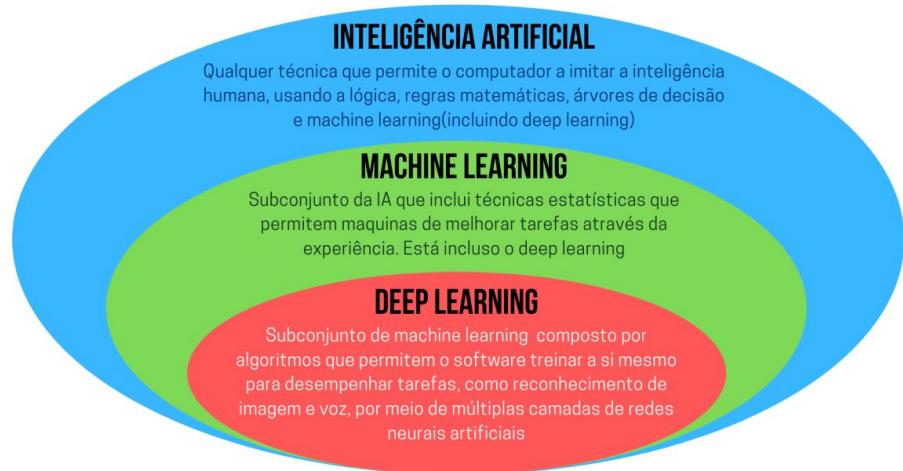
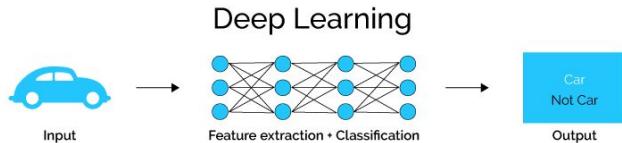
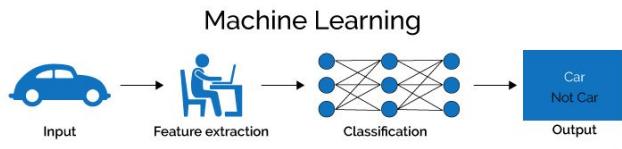
Introdução

Aprendizado de Máquina (ou *Machine Learning*, ML) é um subcampo da Inteligência Artificial que desenvolve algoritmos capazes de aprender a partir de dados e fazer previsões ou tomar decisões sem serem explicitamente programados para isso.

“A aprendizagem de máquina é o campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados” — Arthur Samuel (1959).

Diferença entre IA, ML e Deep Learning

- **Inteligência Artificial:** Ciência geral para criar sistemas inteligentes.
- **Aprendizado de Máquina:** Subárea da IA que usa dados para "ensinar" modelos.
- **Deep Learning:** Subárea do ML que utiliza redes neurais profundas para análise de dados complexos.



Tipos de Aprendizado de Máquina

Supervisionado

- Algoritmos treinados com dados rotulados.
- Frutas: Maçã e Laranja.

Não Supervisionado

- Algoritmos que identificam padrões em dados não rotulados.
- Apenas fotos de Maçã e Laranja.

Reforço

- O algoritmo aprende por tentativa e erro, recebendo recompensas ou punições com base em suas ações.
- Controle de robôs, Jogos (ex.: AlphaGo).



Tipos de Aprendizado de Máquina

Aprendizado Supervisionado

O modelo é treinado com dados rotulados (entrada + saída esperada).
Previsão de preços de casas, diagnóstico médico.

Régressão linear, árvore de decisão, SVM, kNN.

Aprendizado Não Supervisionado

O modelo encontra padrões ou grupos nos dados sem saídas definidas.
Segmentação de clientes, análise de comportamento.

K-means, PCA, algoritmos de agrupamento hierárquico.

Aprendizado por Reforço

O agente aprende por tentativa e erro, recebendo recompensas ou punições.
Robôs autônomos, jogos como xadrez ou Go, controle de irrigação inteligente.

Agente, ambiente, política, recompensa.

Etapas do ML

1. **Coleta de Dados:** Reunir informações que serão usadas para treinar o modelo.
2. **Pré-processamento de Dados:** Limpar, organizar e transformar os dados para que o modelo possa utilizá-los.
3. **Divisão dos Dados:** Separar os dados em conjuntos de treino e teste para avaliação.
4. **Escolha do Modelo:** Selecionar o algoritmo mais adequado para resolver o problema.
5. **Treinamento:** Ajustar o modelo para aprender os padrões nos dados de treino.
6. **Avaliação:** Medir o desempenho do modelo com métricas apropriadas.
7. **Predição e Uso Prático:** Usar o modelo treinado para fazer previsões em novos dados.

Exemplos de Machine Learning

Usar dados históricos de produção agrícola para prever a quantidade ideal de água para irrigação: Baseia-se em padrões climáticos e da plantação para otimizar o uso de recursos hídricos.

Usar dados de clientes para prever quais produtos serão mais vendidos em uma loja: Auxilia na reposição de estoque e campanhas de marketing.

Prever quais alunos podem ter dificuldade acadêmica com base em notas e presença: Permite oferecer suporte personalizado para melhorar o desempenho.

Recomendar músicas em plataformas como Spotify com base no histórico de escutas: Identifica padrões de preferência e sugere músicas similares.

Analizar imagens de exames médicos para detectar doenças como tumores: Usa aprendizado de padrões visuais para encontrar anomalias.

Prever a manutenção de máquinas em fábricas com base no uso e desempenho: Evita falhas e reduz custos de manutenção corretiva.

Machine Learning é ensinar computadores a aprender com dados.

Aula 11 - Introdução a manipulações de dados com python - Pandas

Profa. Gabrielly Queiroz

Introdução

Limpeza e transformação de dados são processos fundamentais para preparar dados brutos, tornando-os consistentes e adequados para análise e modelagem.

Importância:

- Melhora a qualidade dos dados.
- Reduz ruídos e inconsistências.
- Garante resultados mais precisos e confiáveis na análise.

<https://colab.research.google.com/drive/1xnzf0DIgEi5BV8PL6oGc5FWOHfpH9gbM?usp=sharing>

Pandas - Python

- O **Pandas** é uma biblioteca de código aberto do Python amplamente utilizada para análise de dados e manipulação de estruturas de dados.
- Trabalhar com dados tabulares (como planilhas) e séries temporais.

```
import pandas
```

```
import pandas as pd
```

```
import pandas  
data = pandas.DataFrame({"A": [1, 2, 3]})
```

Series		Series		DataFrame
apples	oranges	apples	oranges	
0 3	0 0	0 3	0 0	
1 2	1 3	1 2	1 3	
2 0	2 7	2 0	2 7	
3 1	3 2	3 1	3 2	

Etapas da Limpeza de Dados

Identificação de Problemas

- Exemplo de Conjunto de Dados Problemático

```
import pandas as pd

# Criando um exemplo de DataFrame
data = {
    'Nome': ['Ana', 'Bruno', 'Carlos', None, 'Ana'],
    'Idade': [23, None, 30, 25, 23],
    'Salário': [4000, 5000, None, 3500, 4000],
    'Cidade': ['São Paulo', 'São Paulo', 'Rio', 'Belo
Horizonte', 'São Paulo']
}
df = pd.DataFrame(data)
print("Dados Originais:")
print(df)
```

Tratamento de Valores Ausentes

Identificar Valores Ausentes

```
print("\nVerificando valores ausentes:")
print(df.isnull())
print("\nQuantidade de valores ausentes
por coluna:")
print(df.isnull().sum())
```

Soluções

Remover linhas/colunas com valores ausentes.

Preencher valores ausentes com média, mediana ou valores específicos.

```
# Removendo linhas com valores ausentes
```

```
df_cleaned = df.dropna()
```

```
print("\nDados após remover linhas com valores ausentes:")
```

```
print(df_cleaned)
```

```
# Preenchendo valores ausentes na coluna 'Idade' com a média
```

```
df['Idade'] = df['Idade'].fillna(df['Idade'].mean())
```

```
print("\nDados após preencher valores ausentes em 'Idade':")
```

```
print(df)
```

Remoção de Duplicatas

```
# Verificando duplicatas
```

```
print("\nLinhas duplicadas:")
```

```
print(df.duplicated())
```

```
# Removendo duplicatas
```

```
df = df.drop_duplicates()
```

```
print("\nDados após remover duplicatas:")
```

```
print(df)
```

Correção de Inconsistências

```
# Padronizando os nomes das cidades  
  
# Converte todos os nomes para letras minúsculas  
(str.lower())  
  
df['Cidade'] = df['Cidade'].str.lower()  
  
# Exibindo os dados padronizados  
  
print("\nDados com padronização de cidades:")  
  
print(df)
```

str.strip(): Remove espaços no início e no final da string.

str.lower(): Converte todas as letras para minúsculas.

str.upper(): Converte todas as letras para maiúsculas.

str.title(): Converte a primeira letra de cada palavra para maiúscula.

str.replace(old, new): Substitui ocorrências de um valor antigo (**old**) por um novo (**new**).

str.startswith(prefix): Verifica se a string começa com um prefixo específico.

str.endswith(suffix): Verifica se a string termina com um sufixo específico.

str.split(delimiter): Divide a string em partes com base em um delimitador.

str.normalize('NFKD'): Remove acentos e normaliza caracteres Unicode.

Abrir Arquivo Externo com Pandas

```
import pandas as pd  
  
# Ler o arquivo Excel (substitua 'arquivo.xlsx' pelo nome do arquivo)  
  
df2 = pd.read_excel('arquivo.xlsx')  
  
# Exibir os primeiros dados  
  
print(df2.head())
```

Transformação de Dados

```
# **1. Criando uma nova coluna com base em cálculos**  
  
# Criar uma coluna "Salário Anual" multiplicando o salário mensal por 12  
  
df['Salário Anual'] = df['Salário'] * 12  
  
# **2. Filtragem de dados**  
  
# Filtrar as linhas onde a idade é maior que 25  
  
df_filtrado = df[df['Idade'] > 25]  
  
# **3. Reordenando colunas**  
  
# Alterar a ordem das colunas para: Nome, Cidade, Idade, Salário, Salário Anual  
  
df = df[['Nome', 'Cidade', 'Idade', 'Salário', 'Salário Anual']]  
  
# Exibindo os resultados  
  
print("\nDataFrame após criar 'Salário Anual':")  
  
print(df)  
  
print("\nDataFrame filtrado (Idade > 25):")  
  
print(df_filtrado)
```

Exercício

Você recebeu os dados de funcionários de uma empresa em formato de dicionário. Seu objetivo é usar o **Pandas** para realizar limpeza, transformação e análise desses dados.

Crie um DataFrame com os dados dos funcionários.

Calcule o salário anual:

- Adicione uma nova coluna chamada Salário Anual, que será o valor da coluna Salário multiplicado por 12.

Filtre os funcionários:

- Crie um novo DataFrame contendo apenas os funcionários com idade maior que 30.

Reorganize as colunas:

- Reordene o DataFrame para que as colunas fiquem na seguinte ordem: Nome, Departamento, Idade, Salário, Salário Anual.

Obtenha estatísticas dos salários:

- Exiba a média, o maior e o menor salário mensal.

Exiba os resultados:

- Mostre o DataFrame original com a nova coluna Salário Anual.
- Mostre o DataFrame filtrado.

Aula 12 - Noções de estatística básica e probabilidade aplicadas à IA.

Profa. Gabrielly Queiroz

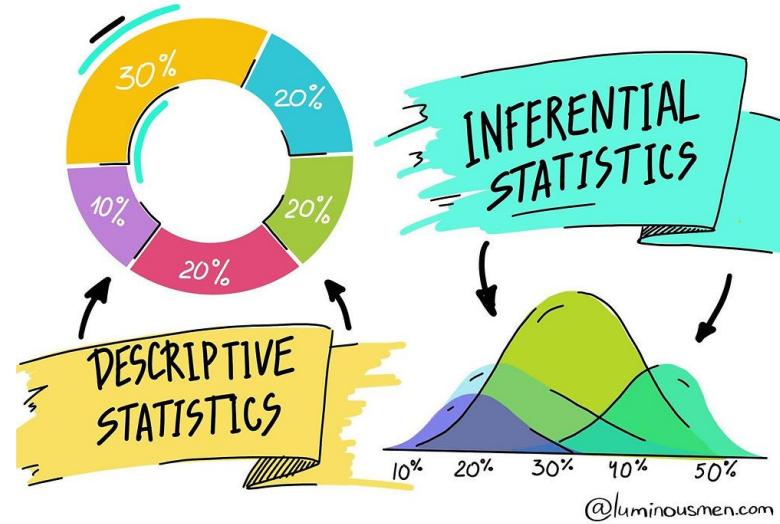
Noções de Estatística Básica e Probabilidade

- Essenciais para Ciência de Dados e Inteligência Artificial.
- Fundamentos matemáticos para análise, identificação de padrões e previsões.
- Estatística: coleta, representação e análise de dados.
- Probabilidade: estudo de eventos aleatórios e incertos.

Divisão da Estatística

Estatística Básica

- **Estatística Descritiva**
 - Organiza, resume e descreve dados.
 - Principais medidas:
 - Média: Valor médio.
 - Mediana: Valor central.
 - Moda: Valor mais frequente.
 - Medidas de dispersão: Desvio padrão e variância.
- **Estatística Inferencial**
 - Conclusões sobre uma população com base em uma amostra.
 - Técnicas: testes de hipóteses, intervalos de confiança, análise de regressão.



@luminousmen.com

Probabilidade

- Mede a chance de eventos ocorrerem (valor entre 0 e 1).
 - Exemplo: Probabilidade de tirar um número par em um dado justo = 0,5.
- Essencial para modelagem preditiva e análise de incertezas.

Distribuições de Probabilidade

- Representam como valores de uma variável aleatória estão distribuídos.
- Principais distribuições:
 - Normal
 - Binomial
 - Poisson

Distribuição Normal

Características

- Modelo de dados contínuos.
- Exemplo: Altura de pessoas.
- Formato de sino: valores próximos à média são mais comuns.

Gráfico

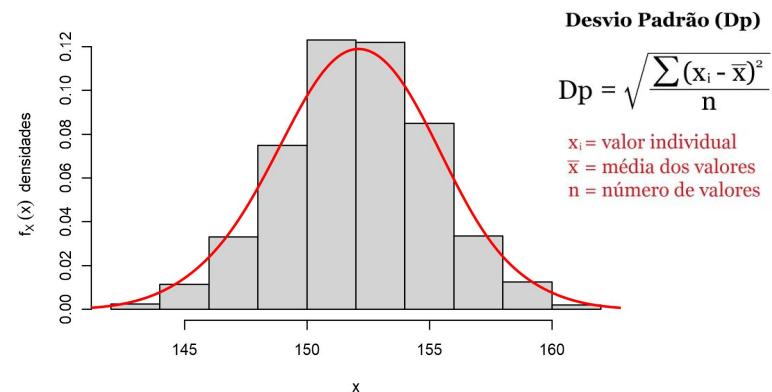
- Larga no meio (média).
- Estreita nas extremidades (valores raros).

Z-score: quão próximo da média.

x : O valor observado (dado que você quer analisar).

μ : A média da distribuição.

σ : O desvio padrão da distribuição.



$$Z = \frac{(x - \mu)}{\sigma}$$

Média (μ) de uma turma em um teste: 70 pontos.

Desvio padrão (σ): 10 pontos.

Aluno tirou $x = 85$.

$$Z = \frac{85 - 70}{10} = 1,5$$

O Z-score é 1,5, ou seja, o aluno está 1,5 desvios padrão acima da média.

Distribuição Binomial

$$P(X = k) = \binom{n}{k} \cdot p^k \cdot (1 - p)^{n-k}$$

Características

- Experimentos com dois resultados possíveis (sucesso ou falha).
- Exemplo: Jogar uma moeda 10 vezes.
 - Resultados: "cara" ou "coroa".
- Aplicações: Aprovação em provas, jogos, etc.

$P(X = k)$: Probabilidade de obter exatamente k sucessos.

$\binom{n}{k}$: Número de combinações possíveis de k sucessos em n tentativas. Calculado como:

$$\binom{n}{k} = \frac{n!}{k!(n - k)!}$$

p : Probabilidade de sucesso em uma única tentativa.

$1 - p$: Probabilidade de falha.

k : Número de sucessos desejados.

n : Número total de tentativas.

$$P(X = 3) = \binom{5}{3} \cdot (0,5)^3 \cdot (1 - 0,5)^{5-3}$$

$$\binom{5}{3} = \frac{5!}{3!(5 - 3)!} = \frac{5 \cdot 4}{2 \cdot 1} = 10$$

$$P(X = 3) = 10 \cdot (0,5)^3 \cdot (0,5)^2 = 10 \cdot 0,125 \cdot 0,25 = 0,3125$$

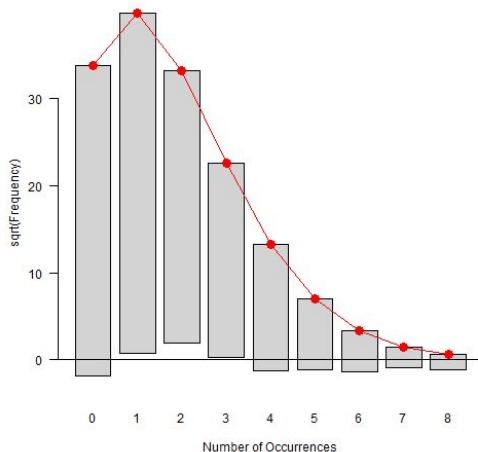
A probabilidade de obter exatamente 3 "caras" é **31,25%**.

Distribuição de Poisson

$$P(X = k) = \frac{\lambda^k \cdot e^{-\lambda}}{k!}$$

Características

- Modela eventos raros em intervalos de tempo ou espaço.
- Exemplo: Quantas pessoas entram em uma loja por hora.
- Gráfico:
 - Poucos eventos ocorrem com frequência.
 - Valores muito altos ou baixos são raros.



$P(X = k)$: Probabilidade de ocorrer exatamente k eventos no intervalo.

λ : Média esperada de eventos por intervalo (valor esperado).

k : Número de eventos reais que você quer calcular a probabilidade.

e : Número de Euler (aproximadamente 2,718).

$k!$: Fatorial de k ($k! = k \cdot (k - 1) \cdot (k - 2) \cdot \dots \cdot 1$).

Uma pizzaria recebe, em média, **3 pedidos por hora** ($\lambda=3$). Qual é a probabilidade de receber exatamente **5 pedidos** em uma hora ($k=5$)?

$$P(X = 5) = \frac{3^5 \cdot e^{-3}}{5!}$$

A probabilidade de receber exatamente **5 pedidos** em uma hora é **10,09%**.

Importância para Ciências de Dados e IA

Estatística e Probabilidade na Prática

- Transformam dados brutos em conhecimento útil.
- Estatística Básica:
 - Organização e resumo de dados.
 - Identificação de padrões e variabilidade.
 - Exemplo: Análise de tendências de consumo.
- Estatística Inferencial:
 - Conclusões baseadas em amostras.
 - Validação de modelos.

Probabilidade:

- Previsão de eventos futuros.
- Aplicações:
 - Algoritmos de aprendizado de máquina.
 - Classificação e tomada de decisões.

Importância para Ciências de Dados e IA

Distribuição normal: para dados contínuos que variam em torno de uma média, como altura ou erro de previsão.

Distribuição binomial: para contar quantos acertos ou erros acontecem em várias tentativas, como prever se uma IA acerta ou erra.

Distribuição de Poisson: para contar quantas vezes algo raro acontece em certo tempo, como quantas falhas ocorrem por hora.

Atividade

Um hospital de pronto-atendimento observa que, em média, 4 pacientes chegam para atendimento de emergência por hora durante o período da noite (das 18h às 6h).

- a) Suponha que, em determinada hora, chegam 12 pessoas ao hospital, mas nem todas são emergências. Cada pessoa tem 1/3 de chance de ser classificada como emergência. Qual é a probabilidade de que exatamente 6 dessas 12 pessoas sejam casos de emergência?
- b) Usando a distribuição de Poisson, qual a probabilidade de exatamente 6 pacientes de emergência chegarem ao hospital em uma hora qualquer durante o período da noite? (Use $\lambda=4$)

Aula 13 - Estatística e Probabilidade - Python

Profa. Gabrielly Queiroz

Numpy e Scipy

NumPy (Numerical Python) é uma biblioteca de código aberto para Python que suporta operações matemáticas e computação científica.

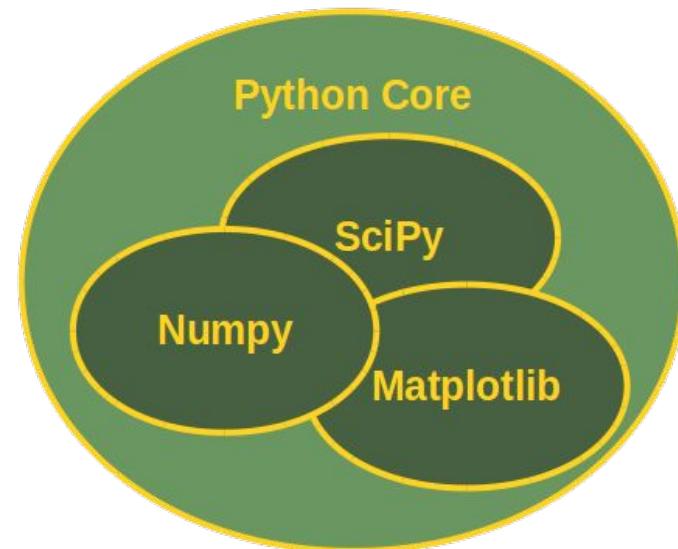
Scipy - computação científica.

Fundamentais para análise de dados e machine learning.

```
import numpy as np  
from scipy import stats
```

<https://numpy.org/>

<https://docs.scipy.org/doc/scipy/tutorial/stats.html>



Scipy

O SciPy (Scientific Python) é outra biblioteca importante no ecossistema da IA e Ciência de Dados. Enquanto o NumPy foca em arrays e álgebra linear, o SciPy expande isso com módulos avançados de:

- Estatística
- Otimização
- Integração
- Álgebra linear
- Interpolação
- Transformadas
- Probabilidade (usado bastante em ML)

Em Inteligência Artificial, o módulo mais usado do SciPy é `scipy.stats`, que oferece ferramentas para distribuições de probabilidade, testes estatísticos, análise de hipóteses, entre outros.

Numpy

`np.array()` - **Cria vetores e matrizes (arrays)**, que são a base para cálculos matemáticos em IA.

```
import numpy as np  
  
x = np.array([1, 2, 3])      # vetor 1D  
  
m = np.array([[1, 2], [3, 4]]) # matriz 2D
```

<https://colab.research.google.com/drive/1wOGA1pQVdl104q7q5OJN1-rRxtlHZyfc?usp=sharing>

Numpy

`np.mean()`, `np.median()`, `np.std()`, `np.var()` - **Estatística descritiva**, usada para explorar o desempenho de modelos e dados de entrada.

```
media = np.mean([10, 20, 30])
```

```
mediana = np.median([1, 2, 3, 4, 5])
```

```
variância = np.var([10, 12, 14])
```

```
desvio = np.std([10, 12, 14])
```

Numpy

np.dot() - **Produto escalar de vetores ou multiplicação de matrizes**, usado em redes neurais e regressão linear

```
w = np.array([0.2, 0.5])
```

```
x = np.array([1, 2])
```

```
resultado = np.dot(w, x) # 0.2*1 + 0.5*2 = 1.2
```

Numpy

`np.random` - **Geração de dados aleatórios**, ideal para criar datasets simulados para treinar modelos.

`np.random.rand()` – números aleatórios entre 0 e 1

`np.random.randint()` – inteiros aleatórios

`np.random.normal()` – dados com distribuição normal

```
# Geração de vetores aleatórios
x = np.random.rand(5)
y = np.random.normal(loc=0, scale=1, size=1000)
```

Numpy

np.where() - Criação de decisões condicionais – semelhante a um if vetorial, muito útil para rótulos em classificação.

```
valores = np.array([0.7, 0.85, 0.95])
classificacao = np.where(valores > 0.9, 'ótimo', 'regular')
print(classificacao)
```

Numpy

`np.argmax()` e `np.argmin()` - **Índice do maior ou menor valor**, útil para identificar a classe com maior probabilidade em modelos de classificação.

```
probs = np.array([0.1, 0.3, 0.6])
```

```
classe_predita = np.argmax(probs) # retorna 2
```

Numpy

`np.linspace()` e `np.arange()` - **Criação de sequências numéricas**, comuns em visualização, testes ou algoritmos genéticos.

```
valores = np.linspace(0, 1, 5) # [0. , 0.25, 0.5, 0.75, 1. ]
```

`np.sum()`, `np.max()`, `np.min()` - **Operações agregadas** essenciais para cálculo de perdas, métricas e análise de desempenho.

```
dados = np.array([[1, 2], [3, 4]])
```

```
print(np.sum(dados)) # soma total: 10
```

```
print(np.max(dados)) # maior valor: 4
```

Scipy

Módulo	Função	Aplicação
scipy.stats	norm, binom, poisson	Distribuições probabilísticas
scipy.stats	mode, ttest_ind	Estatísticas e testes
scipy.optimize	minimize	Minimização de funções
scipy.linalg	inv, eig, solve	Álgebra linear
scipy.integrate	quad, dblquad	Integração numérica
scipy.cluster	kmeans, vq	Agrupamento de dados
scipy.spatial	distance.euclidean	Distância vetorial

Scipy

`scipy.stats.mode()` – Moda - Determinar o valor mais frequente (usado para rótulos, predição mais comum).

```
from scipy import stats
```

```
valores = [1, 2, 2, 3, 3, 3, 4]
```

```
moda = stats.mode(valores)
```

```
print(moda.mode[0], moda.count[0])
```

Distribuição Normal

```
norm.cdf(x, loc=media, scale=desvio_padrao)
```

x é Valor que você quer saber a probabilidade acumulada até ele. Ex: altura < x

```
from scipy.stats import norm
```

```
prob = norm.cdf(valor, loc=media, scale=desvio_padrao)
```

```
from scipy.stats import norm
```

```
media = 0.90
```

```
desvio = 0.02
```

```
# Qual a probabilidade de um modelo ter precisão menor que  
0.88?
```

```
prob = norm.cdf(0.88, loc=media, scale=desvio)
```

```
print(f"Probabilidade de precisão < 0.88: {prob:.4f}")
```

Distribuição Binomial

```
from scipy.stats import binom
```

```
# Probabilidade de acertar exatamente 7 de 10 previsões com 80% de chance de  
acerto
```

```
prob = binom.pmf(k=7, n=10, p=0.8)
```

```
print(f"P(X = 7): {prob:.4f}")
```

n # número de transações
p # probabilidade de acerto
k # número de acertos desejados

```
prob_binomial = binom.pmf(k, n, p)
```

Distribuição de Poisson

```
from scipy.stats import poisson
```

mu # taxa média de falhas
k # número de falhas que queremos

```
prob_poisson = poisson.pmf(k, mu)
```

Qual a chance de ocorrerem exatamente 3 erros, sabendo que a média é 2 por hora?

```
prob = poisson.pmf(k=3, mu=2)  
  
print(f"P(X = 3): {prob:.4f}")
```

Atividade

Faça o download do seguinte conjunto de dados em formato .csv:

https://people.sc.fsu.edu/~jburkardt/data/csv/hw_200.csv

Esse arquivo contém informações sobre **altura (em polegadas)** e **peso (em libras)** de 200 indivíduos. Com base nesses dados, utilize Python (com Pandas, NumPy e SciPy) para realizar uma análise estatística.

Calcule a média e o desvio padrão das colunas Altura e peso. Em seguida, **utilize a função `scipy.stats.norm.cdf()` para calcular a probabilidade de um indivíduo ter altura menor que 65 polegadas e peso menor que 120 libras**, assumindo que os dados seguem uma distribuição normal. Crie uma classificação entre altura baixa, média e alta.

Aula 14 - Técnicas de visualização de dados: Matplotlib e Seaborn.

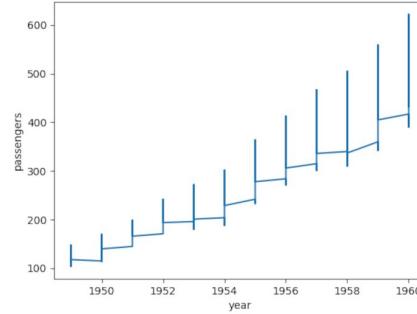
Profa. Gabrielly Queiroz

Matplotlib e Seaborn

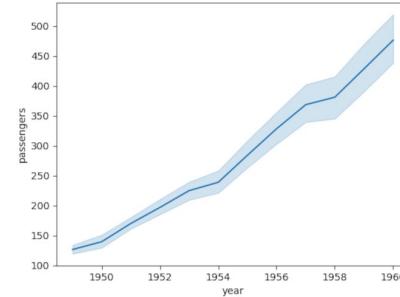
Matplotlib: Biblioteca básica para criar gráficos em Python. É altamente personalizável, mas pode ser um pouco "manual".

Seaborn: Biblioteca mais avançada, construída sobre o Matplotlib. Facilita a criação de gráficos bonitos e com análises estatísticas.

```
● ● ●  
import matplotlib.pyplot as plt  
import pandas as pd  
  
flights = pd.read_csv('flights.csv')  
  
plt.plot(flights['month'], flights['passengers'])  
plt.xlabel('year')  
plt.ylabel('passengers')  
plt.show()
```



```
● ● ●  
import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
  
flights = pd.read_csv('flights.csv')  
  
sns.lineplot(x='year', y='passengers', data=flights)  
plt.show()
```



Instalar as bibliotecas

```
pip install matplotlib seaborn
```

Matplotlib

Quando você precisa de controle completo sobre todos os elementos do gráfico.

Exemplo de gráficos disponíveis:

- `plot()`: Gráficos de linha.
- `scatter()`: Gráficos de dispersão.
- `bar()`: Gráficos de barras.
- `hist()`: Histogramas.
- `boxplot()`: Gráficos de caixa (distribuição).

```
import matplotlib.pyplot as plt
```

```
# Criar dados
```

```
x = [1, 2, 3, 4, 5]
```

```
y = [2, 4, 6, 8, 10]
```

```
# Criar gráfico de linha
```

```
plt.plot(x, y) # Plota os dados
```

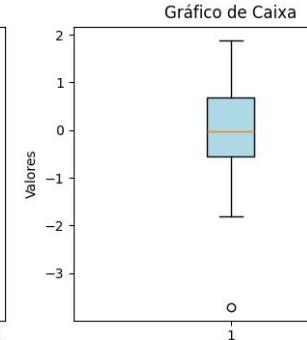
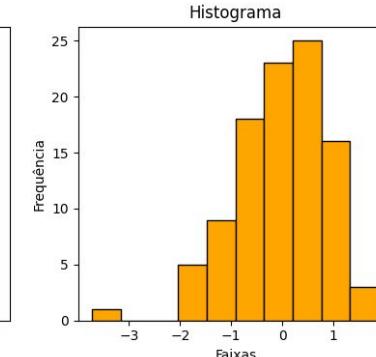
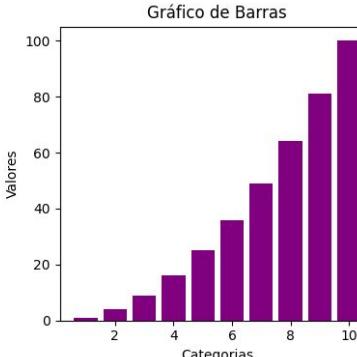
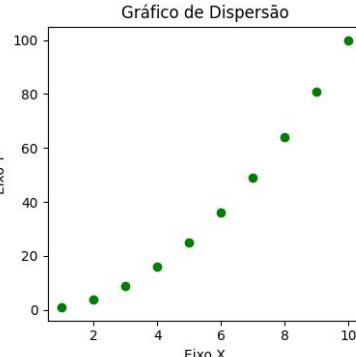
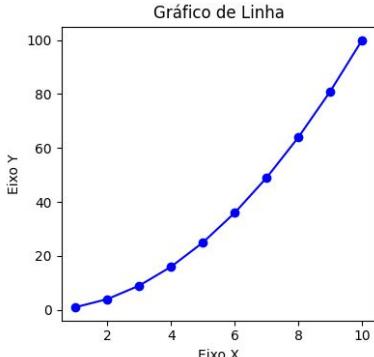
```
plt.title("Meu Gráfico") # Adiciona um título
```

```
plt.xlabel("Eixo X") # Adiciona rótulo ao eixo X
```

```
plt.ylabel("Eixo Y") # Adiciona rótulo ao eixo Y
```

```
plt.grid(True) # Adiciona uma grade
```

```
plt.show() # Exibe o gráfico
```



Quando utilizar cada tipo de gráfico

- `plot()`: Gráficos de linha. Ideal para **visualizar tendências ou variações ao longo do tempo** ou de uma sequência, como crescimento de vendas mensal ou variação de temperatura diária.
- `scatter()`: Gráficos de dispersão. Útil para **mostrar a relação ou correlação entre duas variáveis**. Por exemplo, altura versus peso ou horas de estudo versus desempenho. Perfeito para **comparar categorias ou grupos distintos**. Exemplo: vendas por região, população por cidade ou desempenho de diferentes times.
- `bar()`: Gráficos de barras. Perfeito para **comparar categorias ou grupos distintos**. Exemplo: vendas por região, população por cidade ou desempenho de diferentes times.
- `hist()`: Histogramas. Melhor para **analisar a distribuição de uma variável contínua**. Por exemplo, frequência de idades em uma amostra ou distribuição de notas de alunos.
- `boxplot()`: Gráficos de caixa (distribuição). Indicado para **resumir e comparar a distribuição de dados** com foco em mediana, quartis e valores atípicos (outliers). Exemplo: comparar salários entre diferentes setores.

Seaborn

```
import seaborn as sns  
  
import matplotlib.pyplot as plt
```

Configurando estilo

```
sns.set(style="darkgrid") # Estilos:  
white, dark, whitegrid, darkgrid, ticks
```

scatterplot: Dispersão (relação entre duas variáveis).

lineplot: Linhas (tendências ao longo de uma variável, como tempo).

barplot: Barras (valores agregados, como médias, por categoria).

histplot: Histogramas (distribuição de uma variável contínua).

kdeplot: Densidade (curva para distribuição de probabilidade).

boxplot: Boxplot (distribuição e outliers de uma variável).

violinplot: Violino (distribuição detalhada com densidade e quartis).

heatmap: Mapa de calor (relações em matrizes, como correlações).

pairplot: Múltiplos gráficos (relações entre todas as variáveis numéricas).

Seaborn

Use um countplot quando você quiser **visualizar quantas ocorrências existem de cada categoria** em uma coluna específica. Por exemplo:

```
import seaborn as sns  
  
import matplotlib.pyplot as plt  
  
# Contar quantos alunos são de cada situação (Aprovado ou Em Risco)  
sns.countplot(x='situacao', data=df, palette='Set2')  
  
plt.title("Quantidade de Alunos por Situação")  
  
plt.xlabel("Situação Acadêmica")  
  
plt.ylabel("Número de Alunos")  
  
plt.show()
```

Documentação Matplotlib e Seaborn

Matplotlib: <https://matplotlib.org/stable/index.html>

Seaborn: <https://seaborn.pydata.org/>

https://colab.research.google.com/drive/1BjGYWf26PU8_NSYCu7A-4hZ5gbIsK2-M?usp=sharing

Atividade

Utilizando o conjunto de dados disponível no link

<https://archive.ics.uci.edu/dataset/320/student%2Bperformance>, baixe os dois arquivos

CSV fornecidos: student-mat.csv (dados de Matemática) e student-por.csv (dados de Português). Carregue os dados usando o pandas e selecione colunas relevantes para visualizar a relação entre o tempo de estudo semanal (studytime) e a nota final (G3). Para isso, produza um gráfico de dispersão com Matplotlib para visualizar essa relação, mapa de calor e relação entre as variáveis. Em seguida, use a biblioteca Seaborn para criar um gráfico de boxplot que mostre a distribuição da nota final (G3) de acordo com a frequência de consumo de álcool durante a semana (Dalc).

Interprete visualmente se o consumo de álcool parece afetar o desempenho.

Aula 15 - Algoritmos básicos: Regressão Linear e Regressão Logística.

Profa. Gabrielly Queiroz

Estrutura de um Algoritmo de ML

1. Coleta de Dados
2. Pré-processamento (limpeza, normalização, seleção de atributos)
3. Divisão em treino e teste
4. Treinamento do modelo
5. Avaliação (métricas)
6. Aplicação prática

Regressão Linear

A regressão linear é um método de aprendizado supervisionado usado para prever valores contínuos. Ela modela a relação entre uma variável dependente (y) e uma ou mais variáveis independentes (x), assumindo uma relação linear entre elas.

$$y = \beta_0 + \beta_1 x + \varepsilon$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

$$\beta_1 = \frac{\sum((x - \bar{x})(y - \bar{y}))}{\sum((x - \bar{x})^2)}$$

- y: variável dependente (o valor que queremos prever).
- x: variável independente (o valor usado para prever).
- β_0, β_1 : coeficientes a serem ajustados.
- β_0 : **intercepto** (bias), o valor de y quando x=0.
- β_1 : **coeficiente angular**, que representa a inclinação da linha e a variação de y para cada unidade de mudança em x.

Objetivo: Minimizar o erro (soma dos resíduos ao quadrado).

Aplicações: Previsão de vendas, crescimento populacional, etc.

Vantagens e Desvantagens:

- Simplicidade e interpretabilidade.
- Limitação em problemas não lineares.

Em Machine Learning

O modelo recebe um conjunto de dados com pares de entrada (x) e saída (y).

Ele tenta encontrar os coeficientes β_0 (intercepto) e β_1 (inclinação) que minimizam o **erro** entre os valores previstos e os valores reais.

$$\text{Erro Total} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

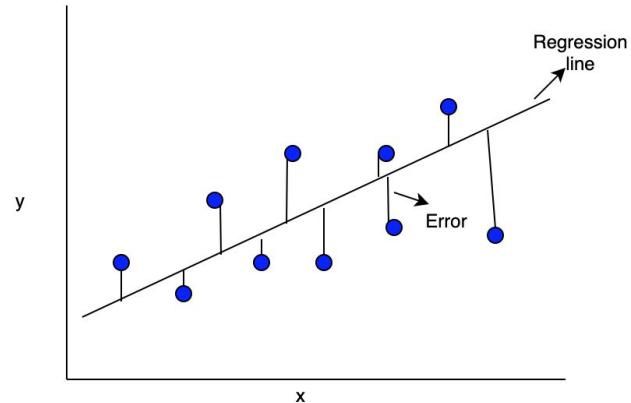
y_i é o valor real,
 $\hat{y}_i = \beta_0 + \beta_1 x_i$ é o valor previsto pelo modelo.

Após ajustar a equação (ou seja, encontrar β_0 e β_1), o modelo é testado com novos dados (xteste).

Ele usa a equação ajustada para prever y_{teste} e compara com os valores reais para verificar a precisão.

Erro Quadrático Médio (MSE):

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 x_i))^2$$



Exemplo

Prever o preço de uma casa com base na área (em m²).

Área (m ²)	Preço (R\$)
50	150.000
60	180.000
70	200.000
80	230.000
90	250.000

Exemplo

Prever o preço de uma casa com base na área (em m²).

Área (m ²)	Preço (R\$)
50	150.000
60	180.000
70	200.000
80	230.000
90	250.000

$$\bar{x} = \frac{\text{soma dos valores de área}}{\text{número de entradas}} = \frac{50 + 60 + 70 + 80 + 90}{5} = 70$$
$$\bar{y} = \frac{\text{soma dos valores de preço}}{\text{número de entradas}} = \frac{150000 + 180000 + 200000 + 230000 + 250000}{5} = 202000$$

$$\beta_1 = \frac{(50 - 70)(150000 - 202000) + \dots}{(50 - 70)^2 + \dots} = 4000$$

$$\beta_0 = 202000 - (4000)(70) = -80000$$

$$y = -80000 + 4000x$$

Prever o preço para x=75

$$y = -80000 + 4000(75) = 220000$$

Regressão Logística

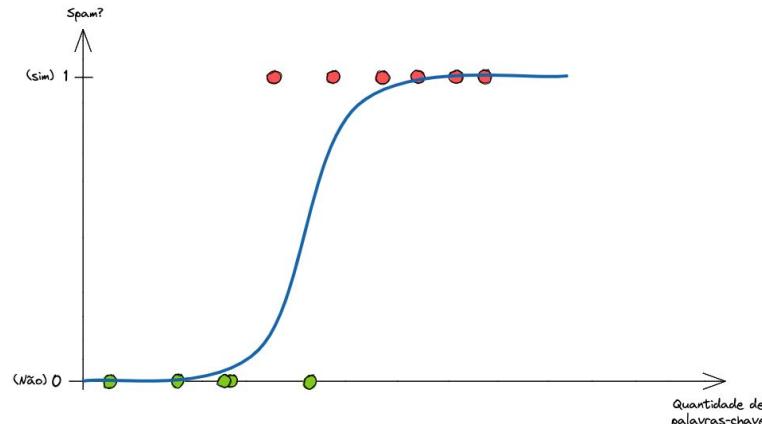
A regressão logística é usada para problemas de classificação, onde a saída é categórica (ex.: sim/não). Diferentemente da regressão linear, que gera valores contínuos, a regressão logística usa a função sigmóide para limitar os valores preditos entre 0 e 1, representando a probabilidade de pertencer a uma classe.

Fórmula:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

$P(y=1|x)$: probabilidade da classe positiva.

Se a probabilidade for maior que 0.5, y é classificado como 1; caso contrário, como 0.



Regressão Logística

Na regressão logística, **os parâmetros (β_0 e β_1) são encontrados usando um método numérico de otimização**, chamado **Máxima Verossimilhança**.

Em regressão logística: você **ajusta** os parâmetros buscando maximizar a chance dos dados observados acontecerem (**não dá pra calcular direto por fórmula básica**).

Esse processo é feito com **algoritmos iterativos**, como o **Gradiente Descendente**. Esses algoritmos funcionam assim:

1. Começam com valores aleatórios ou nulos para β_0 e β_1 ;
2. Calculam o quanto esses valores erram nas previsões (usando uma função chamada *log-verossimilhança*);
3. Ajustam os valores de β_0 e β_1 **um pouquinho por vez**, sempre tentando **reduzir o erro**;
4. Repetem isso centenas ou milhares de vezes, **até encontrar os melhores valores possíveis**.

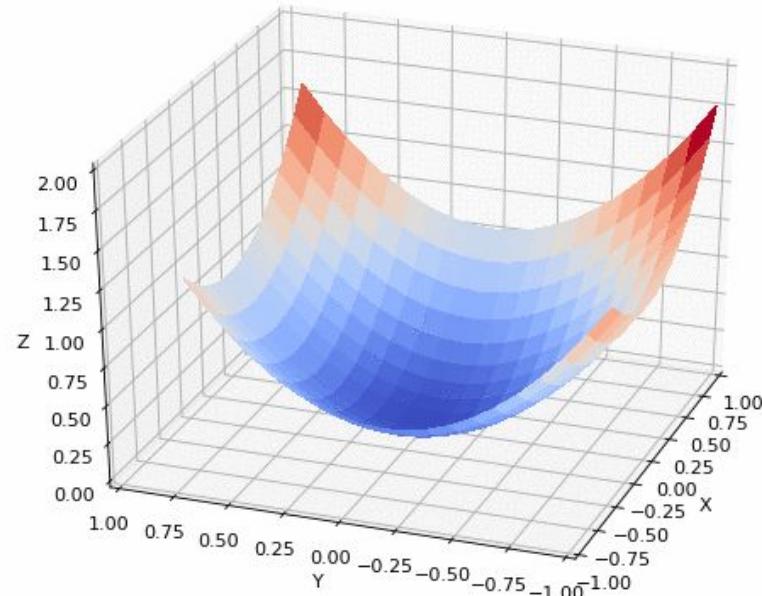
Gradiente Descendente

A função de erro (ou função de custo) usada na regressão logística é a **log-verossimilhança**:

$$J(\beta) = - \sum_{i=1}^n [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Onde $p_i = \frac{1}{1+e^{-(\beta_0+\beta_1x_i)}}$ é a **probabilidade prevista**.

O gradiente descendente testa diferentes combinações de β_0 e β_1 , calcula o erro para cada uma delas, e vai ajustando os valores até encontrar o ponto onde o erro é o menor possível.



Exemplo

Classificar pacientes como "doente" ou "saudável" com base na idade.

	Idade	Diagnóstico (0 = saudável, 1 = doente)
	25	0
	30	0
	35	1
	40	1
	45	1

Idade

Diagnóstico (0 = saudável, 1 = doente)

Exemplo

Classificar pacientes como "doente" ou "saudável" com base na idade.

25	0
30	0
35	1
40	1
45	1

Previsão para idade $x=28$

$$P(y = 1|x = 28) = \frac{1}{1 + e^{-(-10 + 0.25(28))}}$$

$$\beta_0 = -10, \beta_1 = 0.25$$

$$P(y = 1|x = 28) = \frac{1}{1 + e^{-3}} \approx 0.952$$

Se $P > 0.5$, $y = 1$ (doente). Caso contrário, $y = 0$ (saudável).

Para $x = 28$, o paciente é classificado como **doente**.

Atividade

Uma pesquisadora está estudando a relação entre o número de horas de estudo por semana e a nota final dos alunos em uma disciplina. A tabela a seguir mostra os dados coletados:

Horas de Estudo	Nota Final
5	55
10	60
15	65
20	75
25	85

Com base nesses dados, monte a equação da reta de regressão linear que melhor representa a relação entre as horas de estudo e a nota final. Depois, utilize essa equação para prever a nota de um aluno que estuda 18 horas por semana.

Aula 16 - Implementação com Scikit-learn de Algoritmos básicos: Regressão Linear e Regressão Logística.

Profa. Gabrielly Queiroz

O que é Scikit-learn?

- Biblioteca de Machine Learning em Python baseada em NumPy, SciPy e Matplotlib.
- Possui ferramentas para classificação, regressão, clustering, pré-processamento de dados e muito mais.
- Simplicidade e interface consistente.
- Ampla gama de algoritmos prontos para uso.
- Documentação rica e comunidade ativa.

<https://scikit-learn.org/stable/>

pip install scikit-learn



Colab

<https://colab.research.google.com/drive/176rORCVORzBeHOGSxb7KlpmHRzqHEkzS?usp=sharing>

Instanciar x Fit

Instanciar (model = LinearRegression()): Configura o algoritmo, mas ele ainda não sabe nada sobre seus dados.

Ajustar com fit (model.fit(X, y)): O modelo analisa os dados, aprende com eles e fica pronto para fazer previsões ou análises.

```
print(f"Intercepto (b0): {model.intercept_[0]}") # b0: termo independente  
print(f"Coeficientes (b1, b2, ..., bn): {model.coef_[0]}") # b1, b2, ..., um para cada variável de entrada
```

Atividades

Crie um conjunto de dados com pelo menos 6 exemplos, onde exista uma relação entre uma variável independente (entrada) e uma variável dependente (saída).

Em seguida:

- Treine um modelo de regressão linear usando esses dados.
- Faça a previsão para um novo valor de entrada que você escolher.
- Mostre o valor previsto.

Atividades

Crie um conjunto de dados com pelo menos 6 exemplos, onde exista uma variável de entrada (número) e uma variável de saída binária (classe 0 ou 1).

Em seguida:

- Treine um modelo de regressão logística usando esses dados.
- Faça a previsão para um novo valor de entrada que você escolher.
- Mostre a classe prevista (0 ou 1).

Aula 17 - Algoritmos de Clusterização: K-means.

Profa. Gabrielly Queiroz

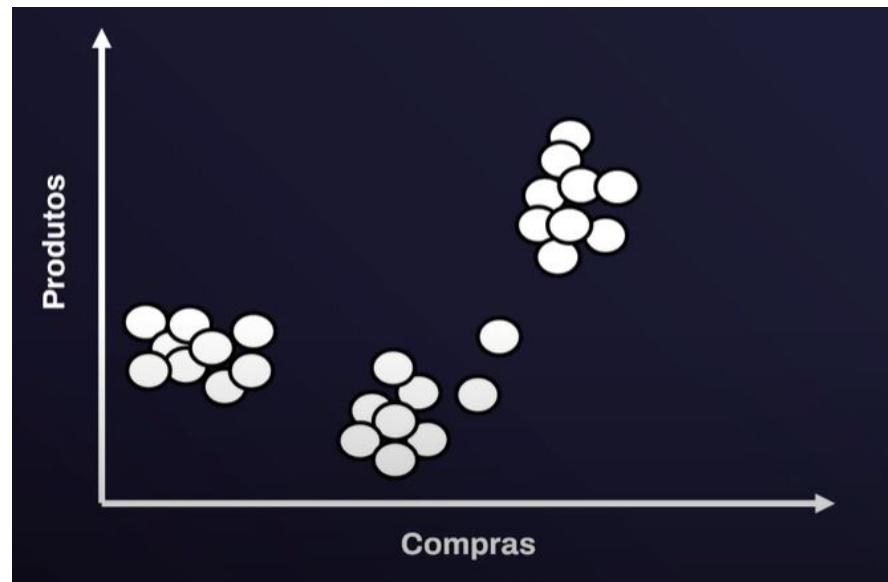
K-Means (clusterização).

- K-means é um algoritmo de aprendizado não supervisionado usado para agrupar dados em k clusters. Ele funciona calculando os centróides dos grupos e associando cada ponto ao centróide mais próximo. O processo é repetido até que os centróides se estabilizem.
- **k**: é o número de clusters que você deseja formar. Esse valor é definido antes de executar o algoritmo.
- **Centróides**: são os "centros" de cada cluster, representando o ponto médio de todos os pontos do grupo.

Etapas principais:

1. Escolher o número de clusters (k).

$$k = 3$$

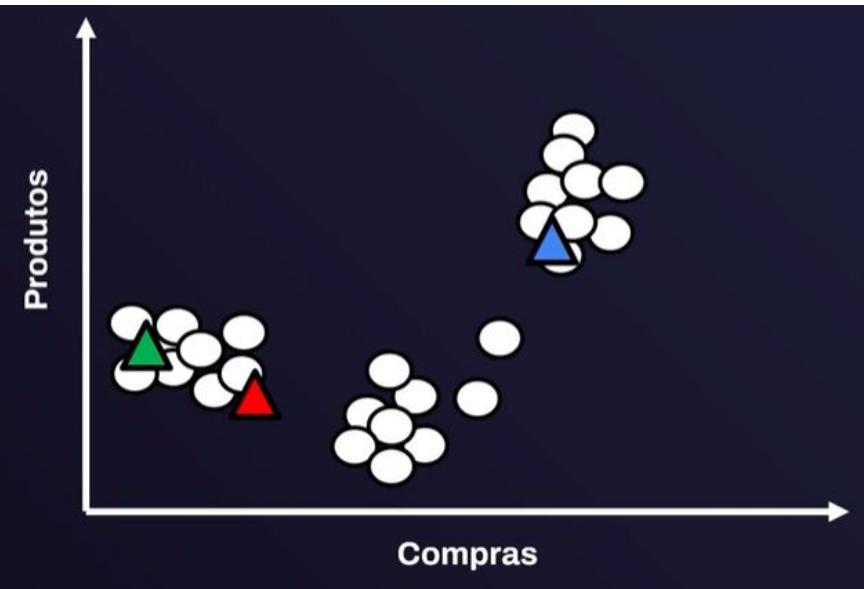


K-Means (clusterização).

- K-means é um algoritmo de aprendizado não supervisionado usado para agrupar dados em k clusters. Ele funciona calculando os centróides dos grupos e associando cada ponto ao centróide mais próximo. O processo é repetido até que os centróides se estabilizem.
- **k**: é o número de clusters que você deseja formar. Esse valor é definido antes de executar o algoritmo.
- **Centróides**: são os "centros" de cada cluster, representando o ponto médio de todos os pontos do grupo.

Etapas principais:

1. Escolher o número de clusters (k).
2. Inicializar os centróides aleatoriamente.
3. Calcular a Distância para todos os pontos para cada centróide.

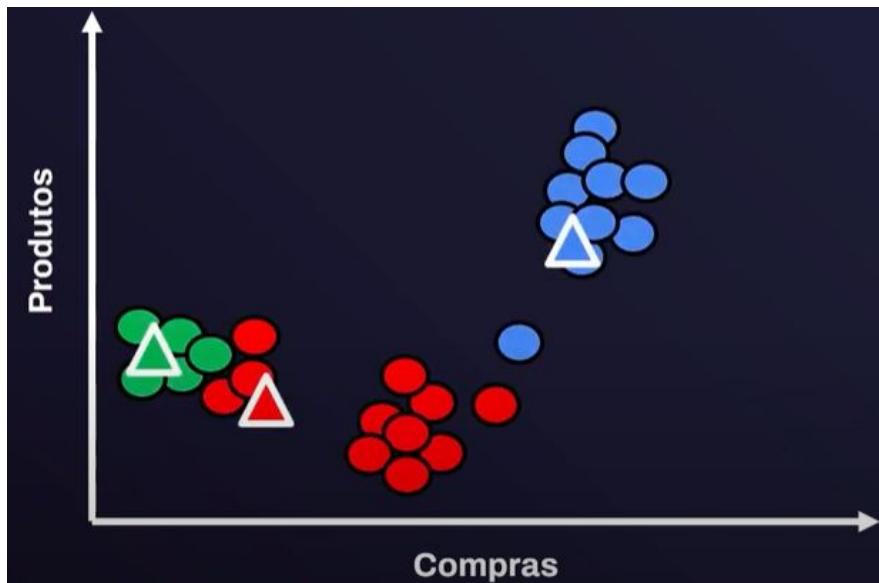


K-Means (clusterização).

- K-means é um algoritmo de aprendizado não supervisionado usado para agrupar dados em k clusters. Ele funciona calculando os centróides dos grupos e associando cada ponto ao centróide mais próximo. O processo é repetido até que os centróides se estabilizem.
- **k:** é o número de clusters que você deseja formar. Esse valor é definido antes de executar o algoritmo.
- **Centróides:** são os "centros" de cada cluster, representando o ponto médio de todos os pontos do grupo.

Etapas principais:

1. Escolher o número de clusters (k).
2. Inicializar os centróides aleatoriamente.
3. Calcular a Distância para todos os pontos para cada centróide.
4. Usar as distâncias mínimas para atribuir uma classe.

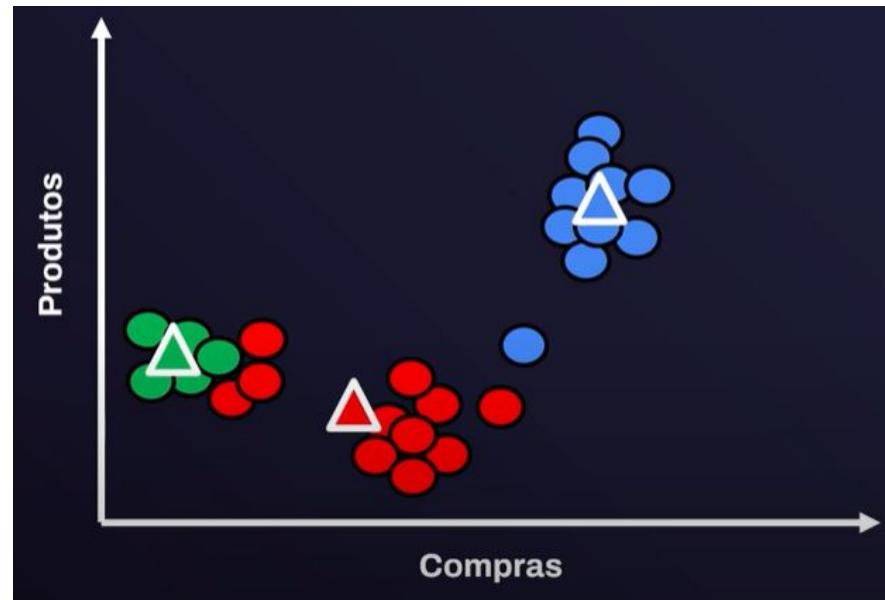


K-Means (clusterização).

- K-means é um algoritmo de aprendizado não supervisionado usado para agrupar dados em k clusters. Ele funciona calculando os centróides dos grupos e associando cada ponto ao centróide mais próximo. O processo é repetido até que os centróides se estabilizem.
- **k:** é o número de clusters que você deseja formar. Esse valor é definido antes de executar o algoritmo.
- **Centróides:** são os "centros" de cada cluster, representando o ponto médio de todos os pontos do grupo.

Etapas principais:

1. Escolher o número de clusters (k).
2. Inicializar os centróides aleatoriamente.
3. Calcular a Distância para todos os pontos para cada centróide.
4. Usar as distâncias mínimas para atribuir uma classe.
5. Recalcula o Centróide. Média das distâncias de cada grupo e coloca o centróide neste local.

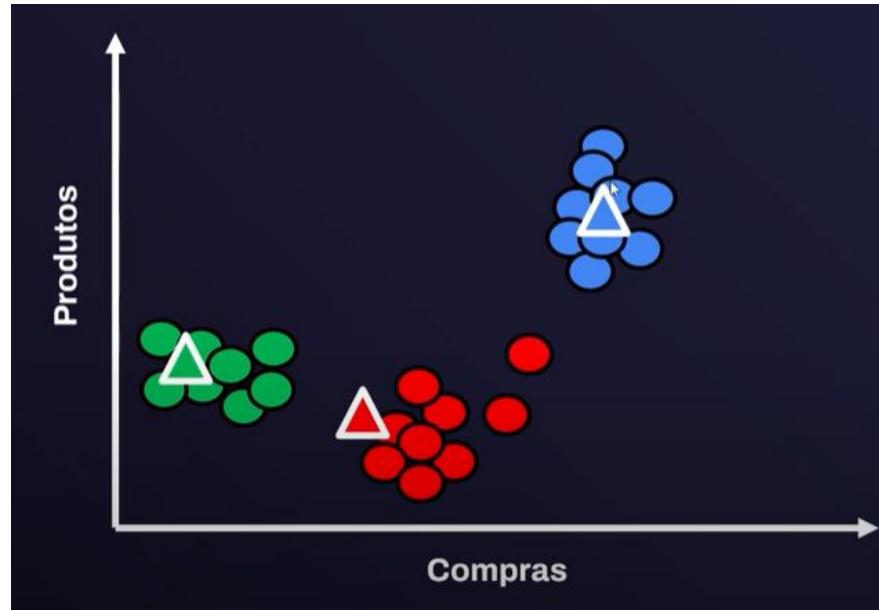


K-Means (clusterização).

- K-means é um algoritmo de aprendizado não supervisionado usado para agrupar dados em k clusters. Ele funciona calculando os centróides dos grupos e associando cada ponto ao centróide mais próximo. O processo é repetido até que os centróides se estabilizem.
- **k**: é o número de clusters que você deseja formar. Esse valor é definido antes de executar o algoritmo.
- **Centróides**: são os "centros" de cada cluster, representando o ponto médio de todos os pontos do grupo.

Etapas principais:

1. Escolher o número de clusters (k).
2. Inicializar os centróides aleatoriamente.
3. Calcular a Distância para todos os pontos para cada centróide.
4. Usar as distâncias mínimas para atribuir uma classe.
5. Recalcula o Centróide. Média das distâncias de cada grupo e coloca o centróide neste local.

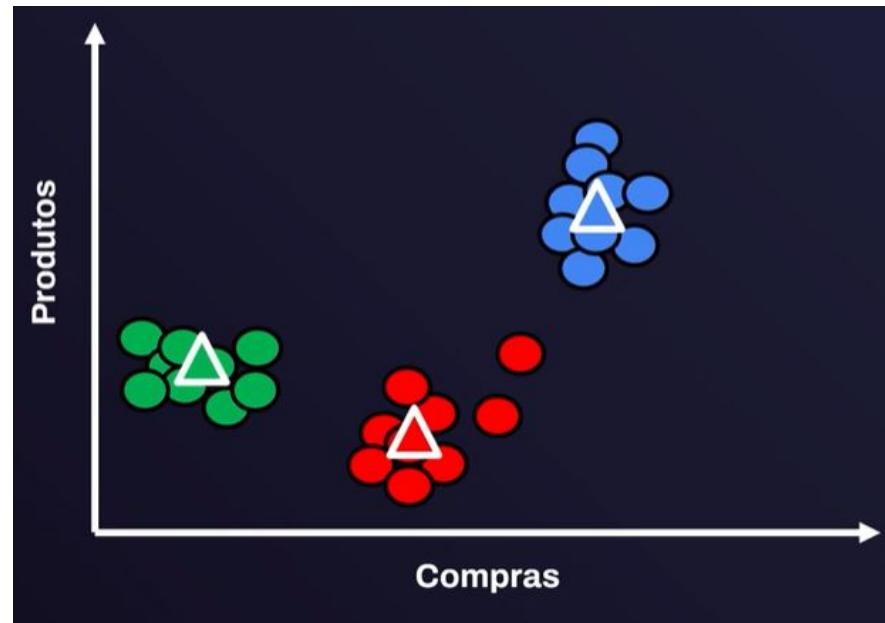


K-Means (clusterização).

- K-means é um algoritmo de aprendizado não supervisionado usado para agrupar dados em k clusters. Ele funciona calculando os centróides dos grupos e associando cada ponto ao centróide mais próximo. O processo é repetido até que os centróides se estabilizem.
- **k:** é o número de clusters que você deseja formar. Esse valor é definido antes de executar o algoritmo.
- **Centróides:** são os "centros" de cada cluster, representando o ponto médio de todos os pontos do grupo.

Etapas principais:

1. Escolher o número de clusters (k).
2. Inicializar os centróides aleatoriamente.
3. Calcular a Distância para todos os pontos para cada centróide.
4. Usar as distâncias mínimas para atribuir uma classe.
5. Recalcula o Centróide. Média das distâncias de cada grupo e coloca o centróide neste local.
6. Recalcula novamente até não ter troca de cluster.



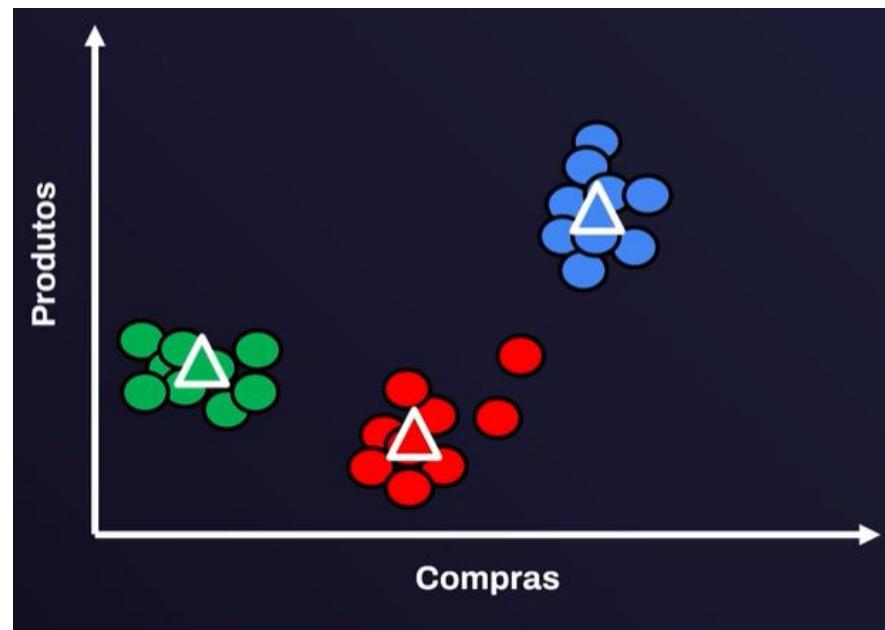
K-Means (clusterização).

Segmentação de cliente

Verde: clientes que compram poucos produtos e com pouca frequência.

Vermelho: clientes que compram poucos produtos e com alta frequência.

Azul: clientes que compram muitos produtos e com alta frequência.



K-Means - Resumo

- Declarar número de clusters K.
 - Inicia aleatoriamente os centróides no espaço.
 - Calcula a distância do centróide para todos os pontos.
 - Atribui aos pontos a classe do centróide mais próximo.
 - Reposiciona o centróide no centro médio dos clusters e refaz os cálculos.
-
- Simples de entender e aplicar.
 - Computacionalmente eficiente, eficiente para quantidade de dados.

Exemplo

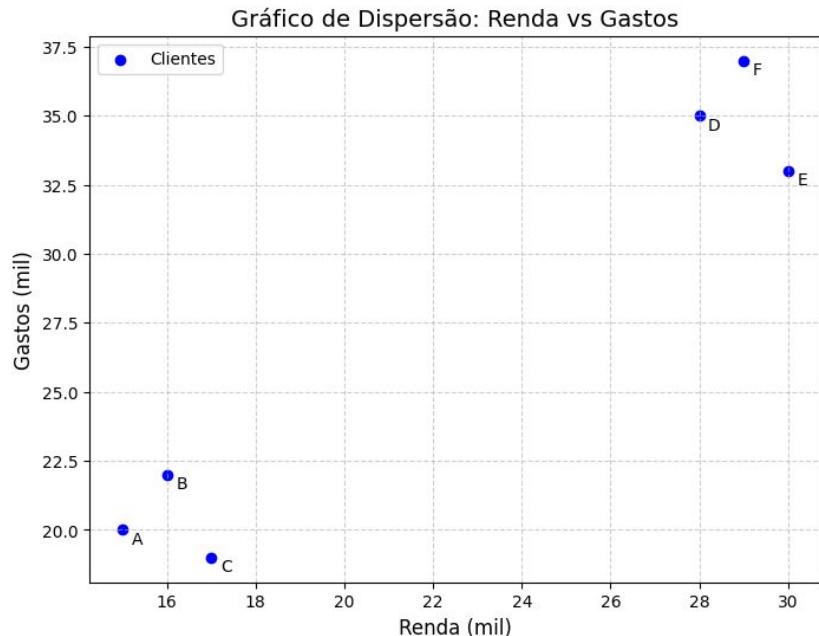
Agrupar clientes com base em renda e gastos.

Cliente	Renda (mil)	Gastos (mil)
A	15	20
B	16	22
C	17	19
D	28	35
E	30	33
F	29	37

Exemplo

- No K-means, **Renda e Gastos** são as dimensões (ou variáveis) que definem as características dos dados.
- O k representa o número de **grupos de observações semelhantes** que queremos formar a partir dessas dimensões.
- Visualizar os Dados (gráficos como dispersão auxiliam).
- Técnicas em algoritmos.

$k = 2$ (renda baixa e gasto baixo; renda alta e gasto alto).



Exemplo

Escolhemos k=2, o que significa que teremos **2 centróides iniciais**. Eles podem ser selecionados aleatoriamente ou com base em pontos do dataset.

Digamos que inicialmente escolhemos os centróides como:

- $C1=(15,20)$
- $C2=(30,33)$

Calculamos a distância até cada centróide. Exemplo para o ponto $A = (15,20)$:

Distância até $C1 = (15, 20)$:

$$d_{A,C1} = \sqrt{(15 - 15)^2 + (20 - 20)^2} = 0$$

Distância até $C2 = (30, 33)$:

$$d_{A,C2} = \sqrt{(15 - 30)^2 + (20 - 33)^2} = \sqrt{225 + 169} = \sqrt{394} \approx 19.85$$

Ponto	Distância para $C1$	Distância para $C2$
A	0	19.85
B	2.24	17.80
C	2.24	19.13
D	19.85	2.24
E	20.81	0
F	22.02	2.24

Exemplo

Cada ponto é atribuído ao cluster cujo centróide está mais próximo:

- A,B,C: mais próximos de C1 (Cluster 1).
- D,E,F: mais próximos de C2 (Cluster 2).

Para cada cluster, recalculamos o centróide como a **média dos pontos do cluster**.

Para $C1$ (média de A, B, C):

$$C1 = \left(\frac{15 + 16 + 17}{3}, \frac{20 + 22 + 19}{3} \right) = (16, 20.33)$$

Para $C2$ (média de D, E, F):

$$C2 = \left(\frac{28 + 30 + 29}{3}, \frac{35 + 33 + 37}{3} \right) = (29, 35)$$

Com os novos centróides, repetimos os passos. O processo continua até que os centróides não mudem mais (convergência).

Exemplo

Calculamos novamente as distâncias de cada ponto aos novos centróides.

Cliente	$d(C1)$	$d(C2)$	Cluster
A	1.05	18.44	1
B	1.67	17.11	1
C	1.36	18.72	1
D	17.44	1.00	2
E	19.44	0.57	2
F	21.56	1.41	2

Nenhuma mudança nos clusters!

- Como os centróides não mudaram na segunda iteração, o algoritmo **convergiu**. Isso significa que os clusters finais foram encontrados.

Atividade

Agrupar cidades com base em temperatura média e umidade relativa.

Cidade	Temperatura (°C)	Umidade (%)
A	22.0	70
B	25.5	65
C	18.0	80
D	27.0	60

Aula 18 - Implementação com Scikit-learn de Algoritmos de Clusterização: K-means.

Profa. Gabrielly Queiroz

Colab

[https://colab.research.google.com/drive/1d7LLKMUUu00oI60VKLiWLqn3qvm01Zx
p?usp=sharing](https://colab.research.google.com/drive/1d7LLKMUUu00oI60VKLiWLqn3qvm01Zxp?usp=sharing)

Atividade

Utilize o algoritmo K-Means para realizar o agrupamento de clientes com base nas colunas de idade, renda anual e pontuação de consumo. Leia os dados do seguinte arquivo CSV:

https://raw.githubusercontent.com/tirthajyoti/Machine-Learning-with-Python/master/Datasets/Mall_Customers.csv

Selecione apenas as colunas numéricas necessárias para o agrupamento: Age, Annual Income (k\$) e Spending Score (1-100).

Aplique o K-Means com o melhor valor de k, determinado com base no método do cotovelo e no Silhouette Score.

Adicione ao DataFrame a coluna com o número do cluster correspondente a cada cliente.

Visualize os agrupamentos utilizando um gráfico de dispersão colorido por cluster.

Por fim, exiba a inércia do modelo e utilize groupby() para mostrar as médias das variáveis por grupo e descrever os perfis de clientes encontrados.

Aula 19 - Árvore de Decisão - Decision Tree

Profa. Gabrielly Queiroz

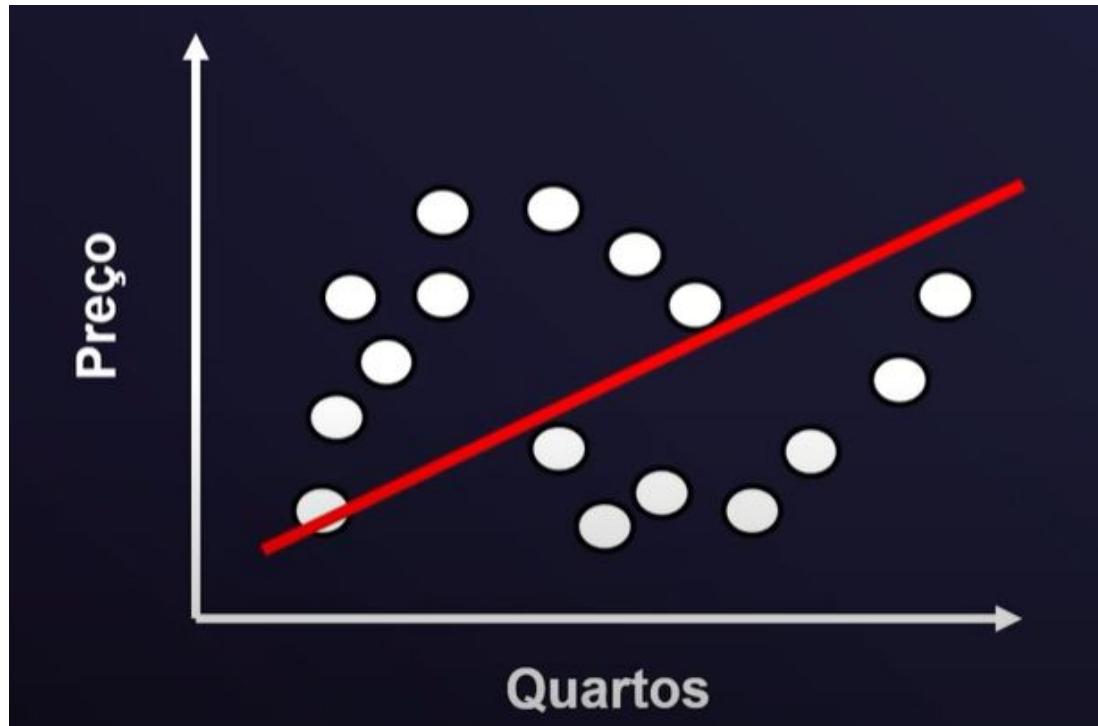
Árvores de Decisão

- As árvores de decisão foram criadas para resolver problemas onde é necessário tomar decisões claras e objetivas com base em dados. Elas simulam o processo humano de tomada de decisão, como responder a uma série de perguntas **sim/não** até chegar a uma conclusão.
- "Devo aprovar ou não o crédito de um cliente?" Renda do cliente. Histórico de pagamento. Dívida atual.
- Antes das árvores de decisão, a abordagem seria usar **um único modelo matemático**, como a regressão linear ou logística. Esses modelos são bons, mas podem sofrer com o **problema de viés e variância**, que afeta sua precisão.
- **Viés:** simplicidade excessiva. Um modelo com alto viés pode ser **muito rígido** e não captar a complexidade dos dados. Exemplo: Usar uma regra simples como "Se a renda for maior que X, aprove o crédito" pode ignorar outros fatores importantes.
- **Variância: excesso de ajuste** do modelo. Um modelo com alta variância tenta **memorizar os dados** em vez de generalizar, ou seja, ele funciona bem no treinamento, mas falha com novos dados. Exemplo: Criar uma regra extremamente específica para cada cliente nos dados históricos pode levar a decisões confusas para novos clientes.

Árvores de Decisão

Régressão Linear:

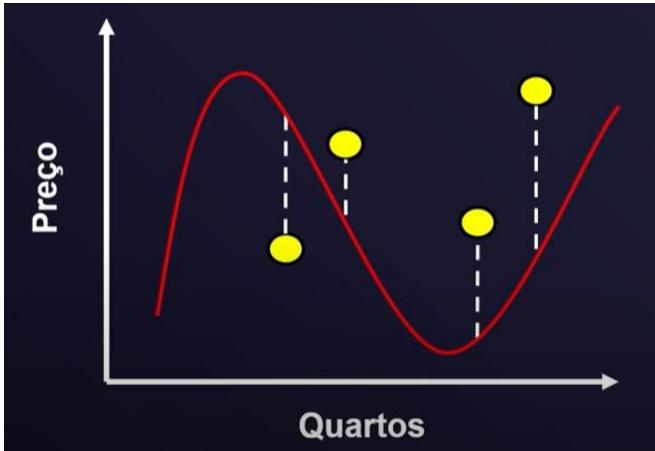
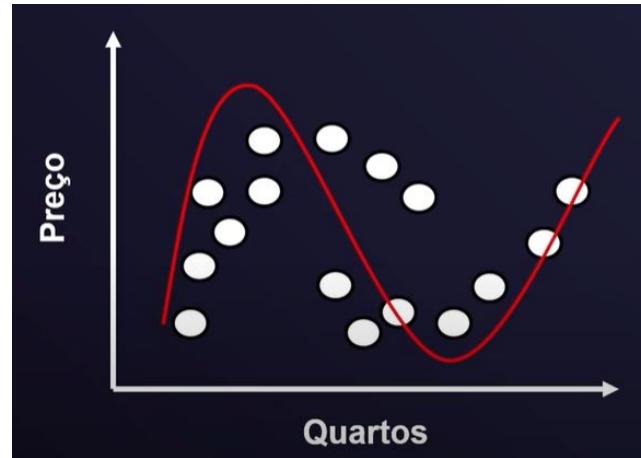
- Alto Viés e Baixa variância



Árvores de Decisão

Regressão Polinomial:

- Baixo Viés e Alta Variância



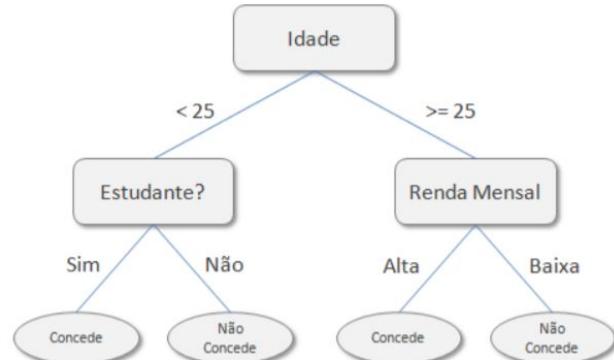
Árvores de Decisão

Uma árvore de decisão é um modelo de aprendizado supervisionado que divide os dados em subgrupos com base em perguntas ou condições

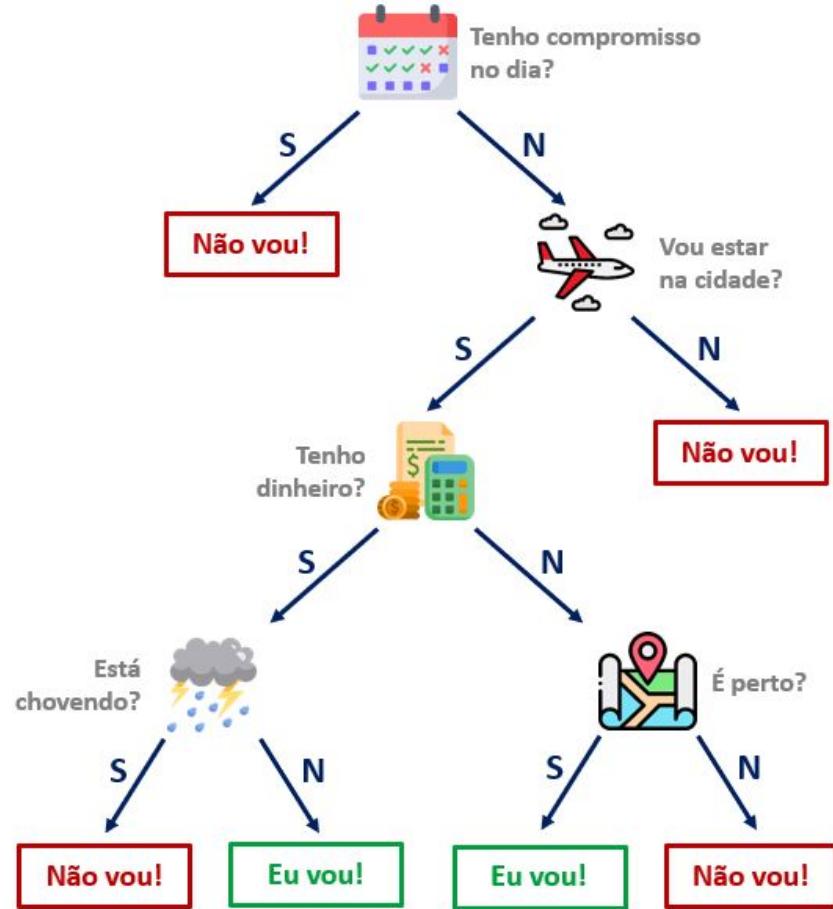
As árvores de decisão foram criadas para encontrar um equilíbrio entre **viés** e **variância**:

1. **Regras gerais no topo:** A árvore começa simples, com uma divisão ampla, como "**Idade < 25?**", que captura padrões gerais e reduz o **viés**.
2. **Regras específicas em níveis inferiores:** As divisões se tornam mais refinadas, como "**Estudante?**" ou "**Renda Alta?**", garantindo que as decisões considerem detalhes relevantes sem serem excessivamente complexas, reduzindo a **variância**.

Esse processo hierárquico evita decisões simplistas (alto viés) ou específicas demais (alta variância), criando um modelo equilibrado e eficaz.



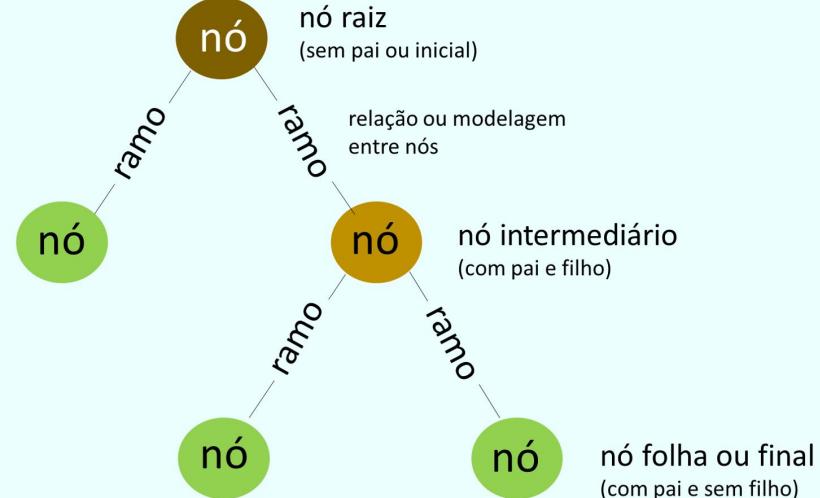
Árvores de Decisão



Árvores de Decisão

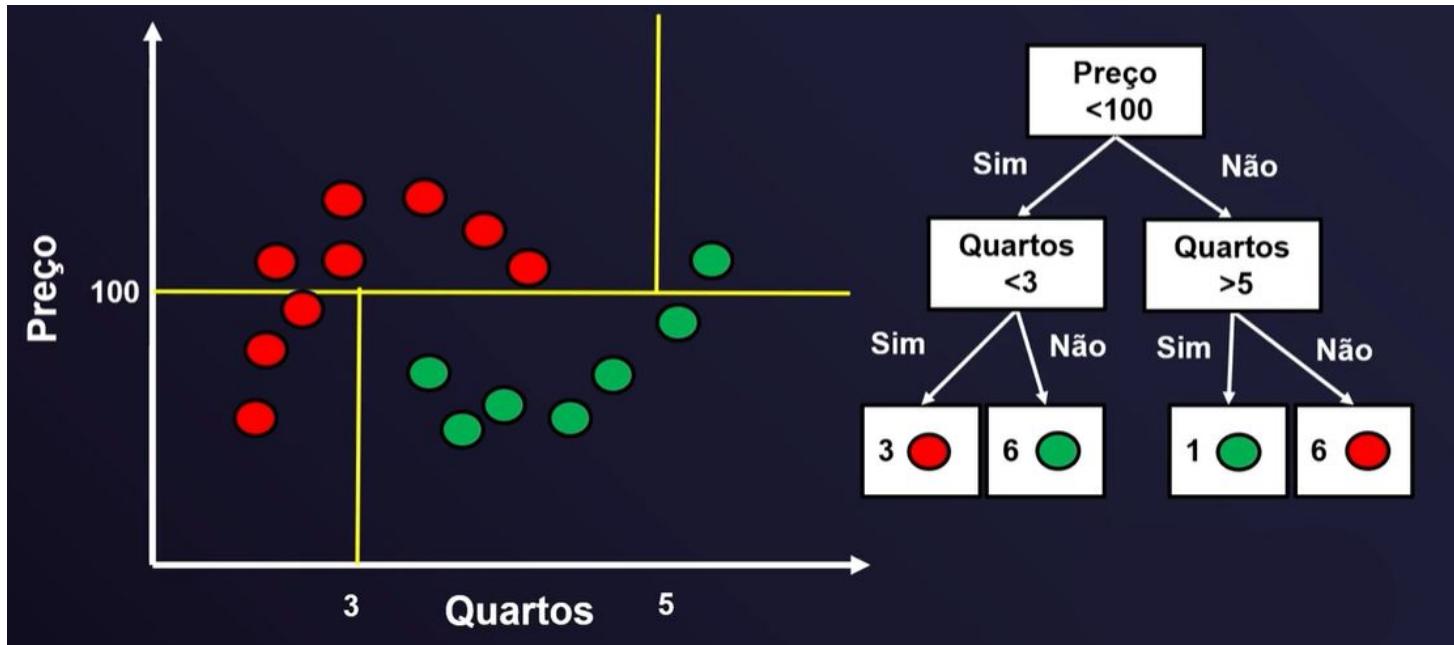
Uma **árvore de decisão** é uma **estrutura hierárquica** usada para tomar decisões com base em regras. Ela é composta por:

- **Nó Raiz (Root)**: primeiro ponto de decisão.
- **Nós Internos**: representam testes lógicos sobre atributos.
- **Folhas (Leaf Nodes)**: representam decisões finais (classes ou valores).



Estrutura em Árvore

Árvores de Decisão

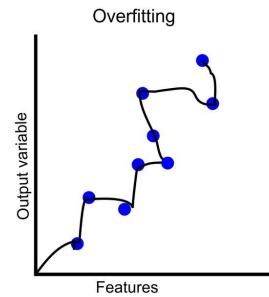
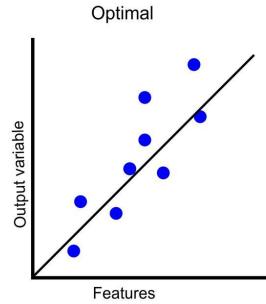


Árvores de Decisão

Desvantagens:

Tamanho excessivo - Árvore pode crescer muito e levar ao overfitting.

Overfitting é quando o modelo **decora os dados de treino**, mas não consegue prever bem em situações novas.



Árvores de Decisão

A árvore de decisão por si só é apenas uma estrutura de representação (uma forma de organizar regras).

O que caracteriza o **Aprendizado de Máquina (Machine Learning)** é o processo automático de escolher as melhores regras para construir essa árvore com base em dados.

Árvore de Decisão

Algoritmo	Tipo	Critério de Divisão	Pontos-Chave
ID3	Classificação	Ganho de Informação (Entropia)	Simples, sem poda, não lida com valores contínuos. Base teórica para outros.
C4.5	Classificação	Razão de Ganho	Evolução do ID3. Suporta valores contínuos e faltantes. Faz poda.
CART	Classificação e Regressão	Gini (classificação), MSE (regressão)	Usa apenas divisões binárias. Muito eficiente. Base do Scikit-learn.
CHAID	Classificação	Teste Qui-quadrado	Usa estatística para ramificação múltipla. Comum em pesquisas e análises sociais.

Critérios de Divisão de Nós

Índice de Gini

Gini = 0 → Conjunto **puro**: todos os exemplos pertencem à mesma classe.

Gini próximo de 1 → Conjunto **impuro**: os exemplos estão bem distribuídos entre várias classes.

$$\text{Gini}(S) = 1 - \sum_{i=1}^n p_i^2$$

n é o número de classes possíveis,

p_i é a proporção de elementos da classe i no conjunto S .

Índice de Gini

Suponha um conjunto S com 10 exemplos:

- 6 da classe **Sim**

- 4 da classe **Não**

Cálculo:

$$p_{\text{Sim}} = \frac{6}{10} = 0.6, \quad p_{\text{Não}} = \frac{4}{10} = 0.4$$

$$\text{Gini}(S) = 1 - (0.6^2 + 0.4^2) = 1 - (0.36 + 0.16) = 1 - 0.52 = 0.48$$

Esse valor indica que o conjunto ainda está um pouco misturado (não puro).

Como funciona uma Árvore de Decisão

Seleciona-se o **atributo mais informativo** para dividir os dados.

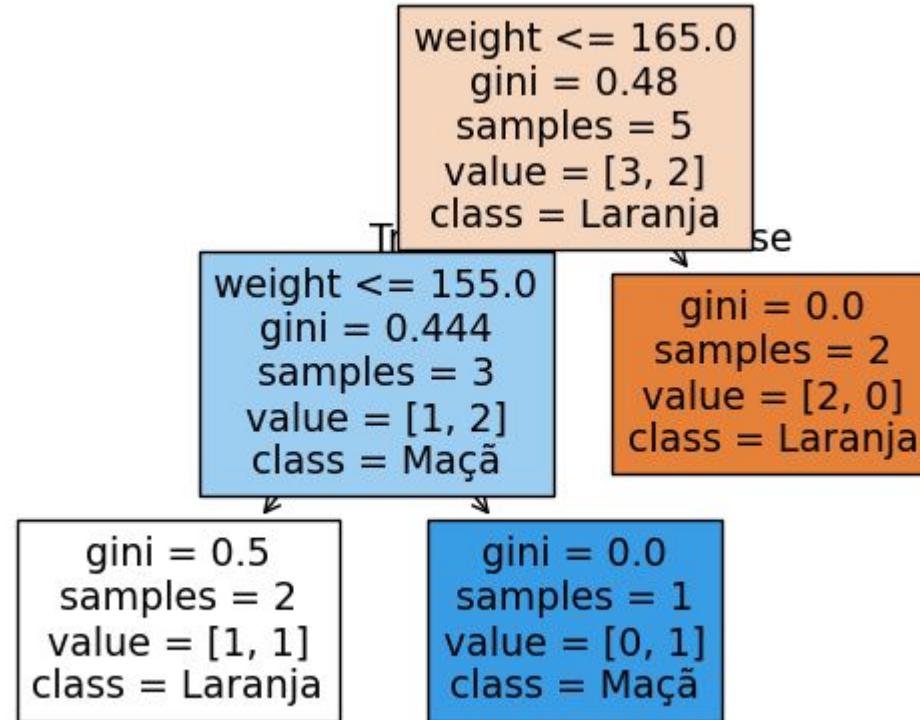
Os dados são separados em **subconjuntos** com base nessa divisão.

O processo se repete **recursivamente** para cada subconjunto.

Quando não há mais ganhos significativos ou os dados estão puros, o nó vira **folha**.

CART

Peso das frutas em
gramas e Textura.



Resumo

Algoritmo	Tipo	Uso Principal	Vantagens	Desvantagens
Regressão Linear	Supervisado (Regressão)	Predizer valores contínuos (ex.: preço, peso).	Simples, fácil de interpretar; bom para relações lineares.	Fraco para relações não lineares; sensível a outliers.
Regressão Logística	Supervisado (Classificação)	Classificar em duas ou mais categorias (ex.: spam/não spam).	Interpretação probabilística; funciona bem para problemas lineares.	Fraco para relações complexas; sensível a dados desbalanceados.
K-means (Clusterização)	Não Supervisado	Agrupar dados com características semelhantes (ex.: segmentação de clientes).	Simples, rápido para grandes dados; útil para clusters bem definidos.	Sensível à escolha de k; pode convergir para soluções locais.
Árvores de Decisão	Supervisado (Classificação/Regressão)	Tomar decisões com base em perguntas hierárquicas (ex.: aprovar crédito).	Fácil de interpretar; trabalha bem com dados categóricos e contínuos.	Propensa ao overfitting; pode criar árvores muito complexas sem poda.

Colab

[https://colab.research.google.com/drive/1oAgC8NnWSBkscDbDPjeN0hJ7yUN1m
yDk?usp=sharing](https://colab.research.google.com/drive/1oAgC8NnWSBkscDbDPjeN0hJ7yUN1myDk?usp=sharing)

Atividade

Escolha uma base de dados (do Scikit-learn, do Kaggle ou um arquivo CSV). Faça o carregamento dos dados, selecione colunas que sirvam como atributos de entrada (X) e uma coluna como rótulo (y). Em seguida, divida os dados em treino e teste, treine um modelo de árvore de decisão com o Scikit-learn, avalie o desempenho do modelo (acurácia) e faça pelo menos três previsões para novos dados simulados. Mostre também a visualização da árvore.

Aula 20 - Ferramentas para Construção de Pipelines de IA. Ajuste e Otimização de Modelos de IA. Balanceamento e Métricas.

Profa. Gabrielly Queiroz

Pipelines

Um pipeline é uma sequência automática de etapas que transforma os dados e treina o modelo, como uma linha de montagem.

https://colab.research.google.com/drive/1xSKRIG3xpRkARodOMd4S9IEF7XRRK_KIM?usp=sharing

Matriz Confusão

CLASSIFICAÇÃO DO MODELO

REAL

		acertos
		erros
SITUAÇÃO REAL	VERDADEIRO POSITIVO	VP 70
	VERDADEIRO NEGATIVO	VN 50
VERDADEIRO POSITIVO	FALSO POSITIVO	FP 30
	FALSO NEGATIVO	FN 10

REAL	Situação real	Previsão do modelo	Res.	Nome técnico
VERDADEIRO POSITIVO	Pessoa tem a doença	Modelo diz que tem	A	Verdadeiro Positivo (TP)
	Pessoa não tem a doença	Modelo diz que não tem	A	Verdadeiro Negativo (TN)
VERDADEIRO NEGATIVO	Pessoa não tem a doença	Modelo diz que tem	E	Falso Positivo (FP)
	Pessoa tem a doença	Modelo diz que não tem	E	Falso Negativo (FN)

acertos

erros

VP - Verdadeiros Positivos

VN - Verdadeiros Negativos

FP - Falsos Positivos

FN - Falsos Negativos

Métricas

Precision (Precisão):

- Mede **quantas das previsões positivas estavam corretas**.

Alta precisão → poucos falsos positivos.

Recall (Revocação ou Sensibilidade):

- Mede **quantas das amostras positivas foram detectadas**.

Alto recall → poucos falsos negativos.

F1-Score:

- Média harmônica entre *precision* e *recall*.
- Equilibra os dois indicadores.

Ideal quando há **dados desbalanceados**.

Accuracy (Acurácia):

- Percentual de acertos totais (todas as classes).

Pode ser **enganosa em bases desbalanceadas**.

$$\text{Precision} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Positivos}}$$

$$\text{Recall} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Negativos}}$$

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

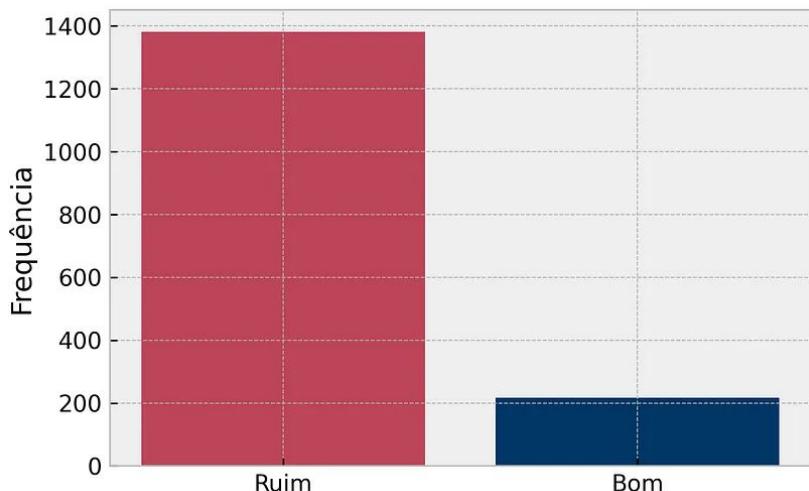
$$\text{Accuracy} = \frac{\text{Acertos Totais}}{\text{Total de Amostras}}$$

Balanceamento dos Dados

O modelo aprende a **favorecer a classe majoritária**, resultando em alta acurácia, mas **baixa detecção da classe rara**.

Problemas comuns:

- Falsos negativos altos (ignora eventos importantes).
- Acurácia enganosa.
- Métricas tradicionais (accuracy) se tornam pouco informativas.



Balanceamento dos Dados

SMOTE — Synthetic Minority Over-sampling Technique:

- Cria **novas amostras sintéticas** da classe minoritária, em vez de apenas duplicar exemplos existentes.
- Gera pontos *intermediários* entre exemplos reais próximos.

`class_weight='balanced'` (do Scikit-learn)

- **Não altera os dados.**
- Ajusta **os pesos de cada classe** dentro do cálculo do erro.
- O modelo **pune mais** os erros da classe minoritária.

