Pós-Graduação em Inteligência Artificial Generativa e LLMs

PUC-Rio

Oficinas 2025.1

Notas de Aula

Vítor Wilher

OFIC251 · Eletiva

Versão 1.0

Resumo. Oficinas práticas do semestre 2025.1, com exercícios guiados e gabaritos comentados.
Palavras-chave: oficinas; prática; exercícios

Notas de estudo elaboradas a partir do material das aulas.

Índice

1 Visão Geral da Disciplina

A disciplina **Oficinas 2025.1** reúne as atividades práticas de laboratório do semestre, distribuídas em vinte e oito encontros e organizadas em seis trilhas paralelas, identificadas pelas siglas **LUI**, **RN** (Redes Neurais), **BI** (Business Intelligence), **OAG** (Otimização por Algoritmos Genéticos), **SAD** (Sistemas de Apoio à Decisão) e **DM** (Data Mining). Diferentemente de uma disciplina expositiva convencional, o formato de oficina privilegia a execução: cada encontro parte de um enunciado, de um conjunto de dados ou de um script cedido pelo professor, e o aluno constrói a solução passo a passo, usando as ferramentas do ecossistema de dados — PostgreSQL, Python (NumPy, pandas, Matplotlib), Google Colab, Power BI, Excel/Solver, Evolver e RapidMiner.

O encadeamento das oficinas segue uma progressão natural do ciclo de vida de um projeto de dados. A trilha **SAD** começa pela base de tudo: linguagem SQL (DDL, DML e DQL) para criar, alterar, popular e consultar bancos relacionais; em seguida introduz Python do zero (tipos, estruturas de controle, funções, pacotes), NumPy para computação vetorizada, pandas para manipulação de dados e Matplotlib para visualização; e termina com a construção de dashboards no Power BI. A trilha **BI** avança sobre o data warehouse — carga de tabela fato e integração com o Power BI — e, em seus encontros mais recentes, incorpora tópicos de IA generativa, como exercícios de RAG e engenharia de prompt.

A trilha **OAG** é dedicada à modelagem de problemas de otimização com algoritmos genéticos: representação cromossômica, decodificação, funções de avaliação, técnicas de normalização e windowing, operadores de crossover e mutação para representação real, e estratégias de tratamento de restrições (descarte, penalização, reparo, decodificadores e a família Genocop). Os exercícios usam Excel com Solver/Evolver, além de notebooks Python com a biblioteca DEAP.

As trilhas **DM** e **RN** fecham o ciclo com aprendizado de máquina. Em Data Mining percorre-se todo o pré-processamento (missing values, normalização, redução de dimensionalidade, balanceamento e outliers), passa-se pelos algoritmos de classificação (KNN, regressão logística, SVM, árvores, comitês e Random Forest), pelas regras de associação (Apriori, FP-Growth, Eclat) e pelo agrupamento (K-means, hierárquico e DBSCAN). Em Redes Neurais estuda-se do neurônio de McCulloch-Pitts ao Multi-Layer Perceptron treinado por backpropagation, com aplicação a bases reais de classificação e, ao final, a redes recorrentes e LSTM para séries temporais. Estas notas de aula organizam o material por oficina, enfatizando o passo a passo de cada atividade prática.

2 OFICINA - LUI

Material de slides não disponível para esta oficina.

A trilha LUI ocupa dois encontros do calendário (as aulas 25 e 28, nas datas de 23 de junho e 21 de julho), mas não há slides nem arquivos de apoio registrados no material da disciplina. Não é possível, portanto, descrever seu conteúdo sem extrapolar a fonte. Recomenda-se ao aluno recuperar as anotações e os materiais distribuídos diretamente nesses encontros.

3 OFICINA - RN

A oficina de Redes Neurais concentra-se na fundamentação conceitual da área e na sua aplicação prática a bases de classificação e a séries temporais. O material disponível cobre o encontro de introdução (Aula 01) e indica, no último encontro, uma atividade com redes recorrentes e LSTM.

3.1 Inteligência computacional e inspiração biológica

O ponto de partida é a definição de **inteligência computacional**: o desenvolvimento de paradigmas ou algoritmos que permitam às máquinas realizar tarefas cognitivas. Um sistema desse tipo deve ser capaz de fazer três coisas:

- **armazenar conhecimento**;
- **aplicar o conhecimento armazenado** para resolver problemas;
- **adquirir novo conhecimento** através da experiência.

Diversos paradigmas dessa família nascem de metáforas naturais. Os slides organizam essa correspondência de forma direta: sistemas especialistas espelham a inferência humana; lógica fuzzy, o processamento linguístico; **redes neurais**, os neurônios biológicos; **algoritmos genéticos**, a evolução biológica; *particle swarm*, o comportamento social e individual de pássaros e peixes; e sistemas híbridos combinam aspectos de vários paradigmas. Essa tabela é útil porque mostra que a oficina de RN e a oficina de OAG são ramos irmãos de uma mesma árvore.

No neurônio biológico, cada célula recebe estímulos de outros neurônios através das **sinapses**; o efeito de cada estímulo é controlado por um **peso sináptico**, que pode ser positivo ou negativo, e é o ajuste desses pesos que faz a rede aprender. O processamento é altamente paralelo.

O experimento de Watanabe e colaboradores (1995), citado nos slides, ilustra didaticamente o que se espera de uma rede neural. Pombos foram colocados em caixas de Skinner e expostos a pinturas de dois artistas, Chagall e Van Gogh, recebendo recompensa ao bicar o botão diante de um quadro de Van Gogh. Os pombos discriminaram os quadros com **95% de acurácia** nas pinturas usadas no treinamento e **85%** em pinturas nunca vistas antes. A conclusão é conceitualmente central: os pombos não memorizaram as pinturas, eles extraíram e reconheceram um padrão (o estilo) e generalizaram para casos novos — capacidade que um computador convencional não possui.

3.2 Definição e história

Redes Neurais Artificiais são sistemas inspirados nos neurônios biológicos e na estrutura massivamente paralela do cérebro, com capacidade de adquirir, armazenar e utilizar conhecimento experimental. A semelhança com o cérebro está em dois aspectos: o conhecimento é adquirido do ambiente por meio de um processo de aprendizado, e as forças de conexão entre neurônios (os pesos sinápticos) armazenam esse conhecimento. Na correspondência entre os dois mundos, o neurônio biológico vira **neurônio artificial**, a rede de neurônios vira **estrutura em camadas**, e os bilhões de neurônios do cérebro viram dezenas, centenas, milhares ou milhões de neurônios artificiais.

A linha do tempo apresentada nos slides:

- **1943 — McCulloch e Pitts**: modelo computacional do neurônio artificial, ainda sem capacidade de aprendizado.

- **1949** — **Hebb**, *The Organization of Behavior*: primeira formulação explícita de uma regra de aprendizado, a regra hebbiana.
- **1958** — **Perceptron de Rosenblatt**: método de aprendizado supervisionado.
- **1969** — **Minsky e Papert**: provam as limitações do perceptron de uma camada.
- **1982** — **Hopfield**: retomada da pesquisa na área.
- **1982** — **Kohonen**: mapas auto-organizáveis.
- **1983** — **Barto**: aprendizado por reforço e controle.
- **1986** — **Rumelhart**: **backpropagation**, o método mais popular de treinamento de perceptrons de múltiplas camadas.
- **Atualmente**: interpretações probabilísticas, redes *spiking* e *deep learning*.

3.3 O neurônio artificial passo a passo

O elemento processador recebe **entradas** (atributos independentes, normalizados, representando uma única observação), multiplica cada uma por seu **peso** e soma. Formalmente, para o neurônio `k`:

$u_k = \sum_j w_{kj} x_j$

e a saída resulta da aplicação de uma **função de ativação** sobre a soma acrescida do **bias**:

$y_k = \varphi(u_k + b_k)$

O **bias** tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação. As funções de ativação apresentadas são: **degrau (ou limiar)**, **logística**, **linear**, **tanh** e **ReLU**.

Os slides enfatizam que os **pesos** são o ponto crucial: são eles que de fato representam a rede neural, e criar uma rede neural é basicamente o processo de ajustar esses pesos. A saída pode ser contínua, binária ou categórica, e pode haver mais de uma saída. Vale notar que pesos podem assumir valor zero — a conexão continua existindo, mas não contribui para a saída daquele neurônio, o que equivale a dizer que nem todos os atributos importam para todos os neurônios.

O exemplo didático usado é a estimação do valor de um imóvel a partir de três entradas: **metragem**, **distância ao centro** e **número de quartos**, alimentando uma camada escondida e uma saída.

3.4 Topologias

- **Redes feed-forward**: uma ou mais camadas de processadores em que o fluxo de dados segue sempre em uma única direção, sem realimentação. Podem ter apenas uma camada de saída ou múltiplas camadas (uma camada escondida mais a de saída, configurando o MLP).
- **Redes recorrentes**: possuem conexões entre processadores da mesma camada e/ou com camadas anteriores, isto é, existe realimentação.

3.5 Perceptron de uma camada e o problema do OU-EXCLUSIVO

O perceptron de uma camada é a forma mais simples de rede neural, usada para classificação de padrões **linearmente separáveis**. O **teorema da convergência do Perceptron** garante que, se os padrões de treinamento forem retirados de classes linearmente separáveis, o perceptron converge e posiciona a superfície de decisão como um hiperplano entre as duas classes.

A limitação demonstrada por Minsky e Papert em 1969 é o problema do **OU-EXCLUSIVO (XOR)**:

Ponto	X1	X2	Saída
A0	0	0	0
A1	0	1	1
A2	1	0	1
A3	1	1	0

Não existe uma única reta que separe as saídas 1 das saídas 0. O perceptron de uma camada só resolve funções linearmente separáveis.

3.6 Multi-Layer Perceptron (MLP)

A solução é acrescentar uma camada intermediária. O XOR pode ser decomposto pela combinação de três neurônios que implementam **OR**, **NAND** e **AND**. Os slides mostram os dois hiperplanos envolvidos, cujas equações de fronteira são obtidas dos pesos:

- $20x_1 + 20x_2 - 10 \Rightarrow x_2 = -x_1 + 0{,}5$
- $-20x_1 - 20x_2 + 30 \Rightarrow x_2 = -x_1 + 1{,}5$

O MLP é uma extensão do Perceptron de Rosenblatt, composta de várias camadas de neurônios, sendo a arquitetura clássica mais utilizada. Contém três tipos de camadas: **camada de entrada**, **camada(s) escondida(s) ou intermediária(s)** e **camada de saída**. Qualquer neurônio de uma camada pode se interligar a qualquer neurônio da camada seguinte.

O treinamento do MLP é **supervisionado**, com o algoritmo **Backpropagation** (retropropagação do erro), baseado na regra de aprendizado por correção de erro. O desafio que ele resolve é justamente encontrar um algoritmo capaz de atualizar os pesos das camadas intermediárias: nele, a determinação do sinal de erro é um processo recursivo que se inicia nos neurônios da camada de saída e retrocede até as camadas intermediárias. Este foi um dos motivos do ressurgimento da área.

Para experimentação interativa, os slides indicam o *neural network playground* em `http://playground.tensorflow`

3.7 Estudos de caso da oficina

Os softwares indicados para o laboratório são **RapidMiner**, **R** e **Python**. Dois conjuntos de dados estruturam a prática:

1. Faixa de preço de aparelhos celulares (`mobile.csv`) — 2000 observações, 20 atributos, 4 classes correspondentes a faixas de preço. Entre os atributos: `battery_power` (energia da bateria em mAh), `blue` (tem bluetooth ou não), `clock_speed` (velocidade do microprocessador), `dual_sim`, `fc` (megapixels da câmera frontal), `four_g`, `int_memory` (memória interna em GB), `m_dep` (profundidade em cm), `mobile_wt` (peso), `n_cores` (número de núcleos), `pc` (megapixels da câmera principal), `px_height` e `px_width` (resolução), `ram` (em MB), `sc_h` e `sc_w` (altura e largura de tela em cm), `talk_time`, `three_g`, `touch_screen` e `wifi`.

2. Câncer de Mama (`BreastCancer.csv`) — base da University of Wisconsin, Clinical Sciences Center, com 30 atributos mais classe e id, e 569 instâncias (357 benignas e 212 malignas). Entre os atributos: raio (distância média do centro a pontos no perímetro do tumor), textura (desvio padrão dos valores em escala de cinza), perímetro e área.

Os notebooks de apoio (`Aula1_RN_mobile.ipynb`, `Aula1_RN_breastCancer.ipynb`) seguem o roteiro: carregar a base, normalizar as entradas, separar treino e teste, definir a arquitetura MLP, treinar por backpropagation e avaliar.

3.8 Redes recorrentes e LSTM

O último encontro da trilha traz os notebooks `RedesRecorrentesAirPassengers.ipynb` e `RedesRecorrentesAirPassengers_Enhancements.ipynb`, com arquivos `train.csv` e `test.csv`, dedicados a **LSTM** aplicada à série clássica de passageiros aéreos. *Material de slides não disponível para este encontro específico;* o conteúdo detalhado está nos notebooks e na gravação da aula.

4 OFICINA - BI

A oficina de Business Intelligence ocupa vários encontros do calendário (aulas 11, 13, 14, 15, 17, 18 e 23). O material registrado indica dois blocos: a **carga da tabela fato e a integração do Data Warehouse com o Power BI** (aula 15) e, mais adiante, dois encontros voltados a IA generativa — **exercícios de RAG** (aula 18) e uma oficina de **engenharia de prompt** com o notebook `oficina_cot_tot.ipynb` (aula 23).

Material de slides não disponível para os encontros de carga do DW e de RAG; as gravações constam como arquivos de apoio.

4.1 Engenharia de prompt: quando usar cada técnica

O material de prompt engineering é enxuto e prescritivo. A **regra prática** é: comece sempre por **zero-shot CoT** (*Chain-of-Thought* sem exemplos, isto é, pedir ao modelo que raciocine passo a passo). Só adicione técnicas mais caras, como **Self-Consistency** e **ToT** (*Tree of Thoughts*), se o problema realmente exigir mais precisão ou mais exploração do espaço de soluções.

O critério de custo é explícito: técnicas que fazem **várias chamadas** ao modelo — como Self-Consistency, que gera múltiplas cadeias de raciocínio e escolhe a resposta majoritária, e ToT, que explora ramificações alternativas — podem custar significativamente mais. A referência apontada para aprofundamento é o guia de engenharia de prompt da OpenAI (`https://developers.openai.com/api/docs/guides/prompt-engineering`).

O passo a passo implícito da oficina, refletido no nome do notebook (`oficina_cot_tot.ipynb`), é comparativo: resolver a mesma tarefa primeiro com prompt direto, depois com CoT zero-shot, depois com Self-Consistency e por fim com ToT, medindo ganho de qualidade contra número de chamadas.

5 OFICINA - OAG

A oficina de Otimização por Algoritmos Genéticos é a mais densa em conteúdo teórico-prático do material disponível. Ela se organiza em torno dos **componentes de um algoritmo genético**, enumerados nos slides na seguinte ordem: (1) Problema, (2) Representação, (3) Decodificação, (4) Avaliação, (5) Operadores, (6) Técnicas e (7) Parâmetros. Todo o restante da oficina percorre essa lista.

5.1 Conceitos fundamentais

Os **Algoritmos Genéticos (GAs)** são algoritmos baseados nos mecanismos de seleção natural e genética, inspirados no Princípio da Evolução das Espécies proposto por Darwin: *"quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes."*

São **flexíveis** e permitem incluir facilmente instruções específicas do problema de interesse. Mas há uma advertência decisiva: a qualidade dos resultados depende diretamente da qualidade da modelagem, especificamente de três elementos — a **representação cromossômica e a decodificação**, a **função de avaliação** e os **operadores genéticos**.

Antes de modelar, é preciso fazer o **estudo de contexto do problema**: conhecer regras, restrições, objetivos e procedimentos em uso. GAs são indicados em problemas difíceis de otimização, caracterizados por:

- muitos parâmetros e variáveis;
- problemas mal estruturados, com condições e restrições difíceis de modelar matematicamente;
- grandes espaços de busca onde a busca exaustiva é inviável.

5.2 Passo 1 — Representação

Representação é a maneira de traduzir a informação do problema em uma forma tratável pelo computador. Quanto mais adequada ela for ao problema, maior a qualidade dos resultados. A estrutura básica é o **cromossomo**, composto de **genes**.

Uma boa representação deve: descrever o espaço de busca relevante ao problema; codificar geneticamente a "essência" do problema; e ser compatível com os operadores de crossover e mutação.

A escolha depende do tipo de problema:

Tipo de problema	Representação
Numérico	Binário, Real, Inteiro
Ordem	Lista
Grupo	Vetor
Misto	Mista

5.2.1 Representação binária

Foi o primeiro tipo de representação usado em GAs. Um número real é codificado através de um número binário de `K` bits. A representação binária descreve um real em detalhes, sendo cada bit um gene. Exemplo elementar:

13 em binário = 1101 = $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 1$

Vantagens listadas: representa números na menor base (2); é simples de criar e manipular; produz bons resultados; tem decodificação numérica fácil (inteiro ou real); facilita a demonstração — porém nem sempre é adequada.

5.3 Passo 2 — Decodificação

A **decodificação** constrói a solução do problema a partir do cromossomo. Cromossomos *representam* soluções; a decodificação é a etapa que permite que cada indivíduo seja efetivamente avaliado. Exemplo dos slides:

Cromossomo `0011011`:

$0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 27$

5.3.1 Binário codificando real

Quando o binário codifica um número real, três aspectos importam: as **variáveis do problema** (x1, x2, …, xt), o **domínio de valores** de cada uma (mín, máx) e a **precisão** desejada de `p` casas decimais. O número de soluções distintas necessárias é (máx - mín) vezes 10^p, o que determina o número de bits `k` de cada variável pela condição de que 2^k seja pelo menos esse valor.

A fórmula de decodificação para real é:

$x_{real} = x_{bin} \cdot \frac{(mx-mn)}{2^k-1} + mn$

Assim, se todos os bits forem 0, obtém-se o mínimo; se todos forem 1, obtém-se o máximo. A precisão efetiva é $(mx - mn)/(2^k - 1)$.

5.3.2 Roteiro prático com a função F6

A **função F6** é o benchmark usado na oficina:

$F6(x,y) = 0{,}5 - \frac{(\sin\sqrt{x^2+y^2})^2 - 0{,}5}{(1{,}0+0{,}001(x^2+y^2))^2}$

Suas características: o objetivo é **maximizar**; existe uma **única solução ótima**, F6(0,0) = 1; e ela é **difícil de otimizar** por possuir vários mínimos locais.

A modelagem passo a passo:

1. **Representação**: binária codificando real, com 2 variáveis (x, y).
2. **Domínio**: x, y no intervalo [-100, +100].
3. **Precisão**: 4 a 5 casas decimais, o que leva a exigir `Ki` entre $\log_2(2 \times 10^6)$ e $\log_2(2 \times 10^7)$.
4. **Escolha**: Ki = 22 bits por variável, totalizando **44 bits** no cromossomo.

Exemplo completo de decodificação apresentado:

- Cromossomo: `00001010000110000000011000101010001110111011`
- Dividido em x e y: `0000101000011000000001` e `10001010100011110111011`
- Convertidos para base 10: 165377 e 2270139
- Multiplicados por $200/(2^{22}-1)$: 7,885791751335085 e 108,24868875710696
- Somados ao mínimo (-100): x = -92,11420824866492 e y = 8,248688757106959
- Aplicados a F6: **F6(x,y) = 0,5050708**

5.4 Passo 3 — Avaliação

A **avaliação** determina a qualidade de um indivíduo como solução do problema. A função de avaliação permite diferenciar boas de más soluções e deve embutir todo o conhecimento disponível sobre o problema e seus objetivos de qualidade.

Um cuidado importante: problemas cuja solução é do tipo "tudo ou nada" devem ter sua avaliação modificada para introduzir **gradualismo**. No exemplo das 4 rainhas, em vez de avaliar apenas se a solução é válida ou não, verifica-se **quantas rainhas satisfazem todas as restrições**.

5.4.1 Exercício resolvido de decodificação e avaliação

Dada a função de avaliação $f = x^2 - 2x$ e uma população de 10 indivíduos codificados em binário, o exercício pede (a) representação, (b) decodificação para decimal e (c) avaliação:

ID	Representação	Decodificação	Avaliação
A	1011	11	99
B	1111	15	195
C	0010	2	0
D	1101	13	143
E	1000	8	48
F	0011	3	3
G	1110	14	168
H	1100	12	120
I	1010	10	80

ID	Representação	Decodificação	Avaliação
J	0111	7	35

Exercícios análogos são propostos para outras funções, como $f(x) = x^3 + 15x$, $f(x) = x^3 + 13$, $f(x) = x^3 - 11x^2 + 3{,}2x + 1{,}9$ e $f(x) = x^2 - 8x + 4$, sempre com indivíduos codificados em sequências de bits.

5.5 Passo 4 — Seleção e os problemas da avaliação bruta

O método de seleção de pais deve simular a seleção natural: pais mais capazes geram mais filhos, ao mesmo tempo em que os menos aptos também podem gerar descendentes.

5.5.1 Método da roleta

Cria-se uma roleta na qual cada cromossomo recebe um pedaço proporcional à sua avaliação. Exemplo dos slides, com quatro indivíduos:

Indivíduo	Avaliação	Pedaço da roleta
0001	1	1,61%
0011	9	14,51%
0100	16	25,81%
0110	36	58,07%

O **método clássico** atribui como aptidão o próprio valor numérico da avaliação. Embora usado em muitos problemas, ele apresenta duas situações que precisam ser tratadas.

5.5.2 Problema 1 — Superindivíduos

Superindivíduos são indivíduos com avaliação muito superior à média, capazes de dominar o processo de seleção. Sua presença impede que o GA obtenha novas soluções, potencialmente melhores, porque a população converge prematuramente. Exemplo com $f(x) = x^2$:

Indivíduo	Avaliação	Pedaço da roleta
0001	1	0,3%
0011	9	3,2%
0100	16	5,7%
0000 (=16 em outra escala)	256	90,8%

Um único indivíduo ocupa mais de 90% da roleta.

5.5.3 Problema 2 — Competição próxima

Competição próxima ocorre quando as aptidões são numericamente muito próximas, o que dificulta distinguir a qualidade das soluções. No exemplo com a função de tipo F6, três indivíduos com avaliações 999,514, 999,066 e 999,979 recebem fatias praticamente idênticas na roleta: 33,35%, 33,32% e 33,33%. A seleção torna-se quase aleatória.

5.5.4 Solução A — Normalização

A **normalização** consiste em dar valores às avaliações dentro de um intervalo determinado. O efeito é visível: no caso dos superindivíduos, as fatias 90,8%, 5,7%, 3,2% e 0,4% passam a 45,5%, 31,8%, 18,2% e 4,5% — a pressão seletiva continua existindo, mas deixa de ser esmagadora.

As equações apresentadas em aula:

1. $V' = novo_{min} + \frac{(novo_{max} - novo_{min})}{n-1}(i-1)$ — normalização por posto (`i` é a posição do indivíduo no ranking, `n` o tamanho da população);
2. $V' = \log_{10}(v)$;
3. $V' = \frac{v-min}{max-min}$ — normalização min-max no intervalo unitário;
4. $V' = \frac{v-min}{max-min}(novo_{max} - novo_{min}) + novo_{min}$ — min-max com novo intervalo;
5. $V' = \log_2(v)$.

A tabela comparativa dos slides, para os valores originais 256, 16, 9 e 1:

Original	Eq. (1)	Eq. (2)	Eq. (3)	Eq. (4)
256	10	8,0	1,00	10,00
16	7	4,0	0,06	1,53
9	4	3,2	0,03	1,28
1	1	0,0	0,00	1,00

5.5.5 Solução B — Windowing

O **windowing** consiste em achar o valor mínimo entre as avaliações da população e designar a cada cromossomo uma avaliação igual à quantidade em que ele **excede esse mínimo**:

$V' = v - v_{min}$

Uma variante acrescenta uma **aptidão mínima de sobrevivência** (AP_{min}), para que nenhum indivíduo fique com probabilidade zero de ser selecionado:

$V' = v - v_{min} + AP_{min}$

Exemplo dos slides, com o caso de competição próxima:

Indivíduo	Avaliação	Windowing	Windowing com mínimo
x1	999,979	0,913	0,963
x2	999,066	0	0,05
x3	999,514	0,448	0,498

Sem windowing, as fatias eram praticamente iguais (34%, 33%, 33%); com windowing, tornam-se 67% e 33%, e com avaliação mínima, 64%, 3% e 33% — recuperando a capacidade de discriminação.

O exercício proposto pede exatamente isso: comparar as roletas de seleção de uma lista de indivíduos com avaliações próximas (todas em torno de 99,9x), usando ou não windowing e windowing com avaliação mínima (0,005 em um enunciado, 0,01 no exercício de breakout room).

5.6 Estudo de caso — o problema das quatro rainhas

O problema envolve dispor rainhas em um tabuleiro de xadrez 4 x 4 de forma que nenhuma seja atacada por outra: duas rainhas quaisquer não podem estar na mesma linha, coluna ou diagonal.

A pergunta central da atividade não é apontar a resposta ótima, e sim **modelar** o problema: como seria o cromossomo, quantos genes teria e o que cada gene representaria. Os slides apresentam três abordagens de codificação:

- **Abordagem 1** — um bit por casa do tabuleiro: o cromossomo tem 16 genes (4 x 4), cada gene indicando se há ou não rainha naquela casa. Exemplos: `1010101000010100 0001` e `0000010000000010`.
- **Abordagem 2** — codificar linha e coluna de cada rainha em binário: também 16 bits, mas com semântica de pares (linha, coluna). Exemplo: `0000010101110000 1 0`.
- **Abordagem 3** — codificação mais compacta, com 8 bits: `00000101`, aproveitando o fato de que cada rainha ocupa uma coluna distinta, restando codificar apenas a linha de cada uma.

A comparação entre as três abordagens é o exercício pedagógico: a abordagem 3 reduz drasticamente o espaço de busca ao embutir na representação a restrição de uma rainha por coluna.

5.7 Passo 5 — Representação real e operadores

Cromossomos podem expressar valores diretamente como **números reais** (ponto flutuante) em vez de binário. Para introduzir essa representação, os slides apresentam o conceito de **hibridização**.

5.7.1 Algoritmos Genéticos Híbridos

Consiste em construir um GA inspirado no "algoritmo de otimização em uso" no problema, se houver:

Algoritmo Híbrido = Algoritmo em Uso + Algoritmo Genético

Hibridizar significa: adotar a **representação** em uso; adaptar os **operadores**; adotar **heurísticas** de otimização.

Vantagens: incorpora conhecimento do domínio; resulta num sistema mais familiar ao usuário; o algoritmo em uso pode fornecer "sementes" para o GA, garantindo soluções melhores. Os novos operadores devem, contudo, estar alinhados à filosofia dos GAs — **crossover** como recombinação de subpartes de indivíduos e **mutação** como variação global ou local que mantém agitada a variedade genética.

No caso da F6, o "algoritmo em uso" é uma busca aleatória de x e y reais em planilha Excel, cujo passo a passo é: (1) testa x e y aleatoriamente; (2) o Excel calcula f6 para o par (x,y); (3) salva (x,y) e f6 se f6 for maior que o anterior; (4) retorna a melhor avaliação se o tempo se esgotar; (5) retorna ao primeiro passo. A hibridização então adota: **representação** por lista de reais (x,y); **avaliação** f6(x,y) real; **inicialização** com números reais aleatórios; e **operadores** de crossover, mutação e outros inspirados no problema.

5.7.2 Operadores de crossover para reais

- **Crossover de 1 ou 2 pontos ou uniforme sobre a lista**: troca posições entre os pais segundo um padrão. Exemplo com 4 variáveis, padrão `0110`: de P1 = (x1, y1, t1, z1) e P2 = (x2, y2, t2, z2) resultam F1 = (x2, y1, t1, z2) e F2 = (x1, y2, t2, z1). Os slides observam que esse crossover é, no entanto, **pouco eficiente** em representação real.
- **Crossover de média**: cruzamento específico para o problema, baseado na ideia de que, se dois cromossomos são promissores, a média de seus valores reais pode levar a uma solução melhor. De P1 = (x1, y1) e P2 = (x2, y2) resulta F1 = ((x1+x2)/2, (y1+y2)/2).
- **Crossover aritmético**: combinação linear dos dois genitores. F1 = a · P1 + (1-a) · P2 e F2 = a · P2 + (1-a) · P1, com `a` sorteado no intervalo [0,1] no caso uniforme.

5.7.3 Operadores de mutação para reais

- **Mutação de real**: substitui cada número real do cromossomo por um número real aleatório, caso o teste de probabilidade seja verdadeiro. De (x1, y1) resulta (x1, y_rand). Tem **alto poder de dispersão**.
- **Mutação CREEP**: implementa uma **busca local**, procurando uma solução próxima através de ajustes aleatórios em ambas as direções (+ e -). De (x1, y1) resulta (x1 ± Δx, y1 ± Δy), onde Δ pode ser pequeno ou grande.

O método de ajuste 1 do CREEP é:

- $X_{t+1} = X_t + \Delta(mx - X_t)$ se o bit sorteado for 0
- $X_{t+1} = X_t - \Delta(X_t - mn)$ se o bit sorteado for 1

onde máx e mín são os limites do domínio de x, e $\Delta(s) = s \cdot rand$, com `rand` sorteado no intervalo [0, p], p menor ou igual a 1. O ajuste varia com `p`: `p` pequeno gera ajuste menor; `p` grande, ajuste maior.

5.7.4 Binária versus real — o critério de escolha

Os slides sintetizam os argumentos a favor da representação real:

- é **mais adequada** em problemas de otimização com variáveis sobre domínio contínuo;
- em grandes domínios a representação binária exige cromossomos longos: 100 variáveis no intervalo [-500, 500] com 4 casas decimais demandariam cerca de **2400 bits**;
- é **mais rápida** na execução, pois não há decodificação;
- oferece **maior precisão**, limitada apenas pelo computador;
- o desempenho pode ser melhorado com operadores específicos ao problema;

- dois pontos próximos no espaço de representação estão também próximos no espaço do problema, evitando os **Hamming Cliffs**.

A **distância de Hamming** ilustra o problema: em binário, C1 = `011111` (valor 31) e C2 = `100000` (valor 32) estão a distância 6 um do outro, embora sejam vizinhos no espaço real; em representação real, C1 = 31 e C2 = 32 estão a distância 1.

5.8 Passo 6 — Tratamento de restrições

A grande maioria dos problemas envolve restrições, de dois tipos:

- **soft**: desejáveis, mas que podem ser desobedecidas se necessário;
- **hard**: que obrigatoriamente devem ser obedecidas.

As estratégias apresentadas:

Descarte — as soluções que não respeitam as restrições são simplesmente excluídas da população.

Penalização — as soluções que violam restrições têm suas avaliações penalizadas, o que preserva as características desses indivíduos no pool genético. As funções de penalização variam quanto ao grau de violação (desvio), com constante:

- Linear: $Pen(x) = \alpha \cdot desvio$
- Quadrática: $Pen(x) = (\alpha \cdot desvio)^2$
- Logarítmica: $Pen(x) = \log_n(1 + \alpha \cdot desvio)$

Reparo da solução — soluções que violam restrições são corrigidas por um algoritmo de reparo específico.

Decodificadores — transformam os cromossomos em soluções válidas por construção.

5.8.1 A família Genocop

Genocop I auxilia o processo de otimização considerando somente **restrições lineares**. Sua estratégia é diminuir o espaço de busca, tentando encontrar uma solução inicial em regiões possíveis. A ideia operacional: para cada variável `xk` existe um intervalo possível entre `left(k)` e `right(k)` quando as demais variáveis estão fixas. No exemplo dos slides, para o ponto viável (x4, x5, x6) = (10, 8, 2), tem-se x4 no intervalo [7.25, 10.375] enquanto x5=8 e x6=2; x5 em [6, 11] enquanto x4=10 e x6=2; x6 em [1, 2.666] enquanto x4=10 e x5=8.

Genocop II estende o tratamento às **restrições não lineares**.

Genocop III incorpora **duas populações separadas**:

- **população de busca** (Ps): satisfaz as restrições lineares;
- **população de referência** (Pr): satisfaz **todas** as restrições.

O pseudocódigo apresentado:

```
# P = probabilidade de substituicao
if not isfeasible(S):
```

```
Z = a*S + (1 - a)*R          # a em [0, 1]
while not isfeasible(Z):
    Z = a*Z + (1 - a)*R      # a em [0, 1]
if evaluation(Z) > evaluation(R):
    R = Z
if rand() < P:
    S = Z                     # S e substituido por Z
else:
    evaluation(S) = evaluation(Z)
```

A intuição geométrica: um indivíduo S da população de busca que caiu na região não viável é puxado por combinação linear na direção de um indivíduo R da população de referência (que está na região viável), repetidamente, até que o ponto intermediário Z se torne viável. Se Z for melhor que R, R é atualizado; e com probabilidade P, S é substituído por Z, caso contrário S apenas herda a avaliação de Z.

5.9 Exercícios práticos da oficina

5.9.1 Exercício Telecom (com Evolver)

O enunciado pede usar o **Evolver** para encontrar a resposta ótima de cada etapa e avaliar o desempenho do algoritmo genético frente a diferentes parâmetros evolutivos.

Parte A — uma empresa de telecomunicações quer otimizar a localização de **três antenas** de transmissão, maximizando a cobertura total em número de clientes atendidos. Devem ser encontradas as coordenadas (x, y) de cada antena, sabendo que os raios de cobertura são, respectivamente, 15, 12 e 3 km. Os dados das dez cidades:

ID	Coord. X	Coord. Y	Clientes
1	18	42	7571
2	29	37	5274
3	36	28	11082
4	35	11	11879
5	28	7	9226
6	21	15	7942
7	8	26	6295
8	18	31	4286
9	6	4	8132
10	50	46	11344

Parte B — invertendo o problema: dadas as antenas já alocadas em A (22;11), B (12;33) e C (41;37), quais deveriam ser os **raios de cobertura** de cada uma, com o objetivo de minimizar custos garantindo cobertura de sinal em todas as cidades e que cada antena atenda ao menos 3 cidades. O custo é de R$ 970,00 por km de cobertura.

Parte C — versão completa, com preços diferenciados por cidade e percentuais de clientes distintos. Agora é possível escolher tanto a localização quanto o raio, sabendo que ao custo variável de R$ 970,00 por km soma-se um custo fixo de instalação por faixa de raio:

Raio (de)	Raio (até)	Preço instalação
0	5	27000
5	15	68000
15	30	115000
30	45	180000

Permanece a exigência de que cada antena atenda pelo menos 3 cidades. Os percentuais de clientes e preços médios por cidade variam de 0,12 a 0,74 e de R$ 67,03 a R$ 141,28, respectivamente.

Exercício de breakout room: a partir do resultado obtido na parte A, alterar os parâmetros do otimizador para melhorar o resultado — modificar o **tamanho da população** e a **taxa de mutação**, armazenar o resultado e os valores das variáveis encontrados, e debater os resultados com os colegas. Este é o exercício que fecha o componente (7) Parâmetros da lista inicial.

5.9.2 Exercício da Fábrica

Modelagem de um problema clássico de programação da produção. Uma empresa produz dois produtos e possui três setores independentes, com capacidade ociosa de 4, 12 e 18 horas respectivamente. O lote do produto 1 dá lucro de R$ 3.000,00; o do produto 2, R$ 5.000,00. A demanda excede a capacidade produtiva e os produtos disputam essa capacidade. As restrições:

1. o produto 1 passa pelos setores 1 e 3; o produto 2 passa pelos setores 2 e 3;
2. a produção não pode consumir mais de 4 horas no setor 1;
3. o produto 2 consome 2 horas do setor 2, e a produção não pode ultrapassar 12 horas nesse setor;
4. o produto 1 consome 3 horas do setor 3 e o produto 2 consome 2 horas, com limite de 18 horas no setor.

A pergunta: qual a melhor estratégia de produção para maximizar o lucro? A oficina resolve isso tanto em planilha (Solver) quanto em notebook Python (`Exercicio Fabrica.ipynb`), e há material de vídeo sobre a introdução ao Solver e a introdução à biblioteca **DEAP** para algoritmos genéticos em Python.

6 OFICINA - SAD

A oficina de Sistemas de Apoio à Decisão é a trilha mais longa em número de encontros e cobre três blocos práticos: **linguagem SQL**, **Python para análise de dados** e **construção de dashboards em Power BI**.

6.1 Bloco 1 — SQL: linguagem DDL e DML

O material é a Oficina 02, conduzida a partir de um diagrama de classes UML com três entidades: **Funcionário**, **Projeto** (inicialmente chamada `Proj`) e **Matrícula**, esta última representando a alocação de funcionários em projetos.

6.1.1 Passo 1 — Criar as tabelas com restrições (DDL)

O enunciado pede criar as tabelas no SGBD **PostgreSQL** como estão no modelo, com as seguintes restrições:

- a) na tabela Funcionário, o gênero deve aceitar apenas as letras M e F;
- b) o nome do funcionário não pode ser NULL;
- c) o estado civil deve aceitar apenas os valores C, S, V e D;
- d) o salário deve ser maior que o salário mínimo;
- e) na tabela Projeto, o orçamento deve ser maior que 0;
- f) na tabela Matrícula, a data de alocação deve ter como default a data do dia do cadastro (`CURRENT_DATE`).

Traduzindo o enunciado em SQL ilustrativo:

```
CREATE TABLE funcionario (
  codfunc     INTEGER PRIMARY KEY,
  nome        VARCHAR(40) NOT NULL,
  genero      CHAR(1) CHECK (genero IN ('M','F')),
  estcivil    VARCHAR(2) CHECK (estcivil IN ('C','S','V','D')),
  dtadmissao  DATE,
  cargo       VARCHAR(40),
  salario     NUMERIC(10,2) CHECK (salario > 1412.00)
);

CREATE TABLE proj (
  codproj   INTEGER PRIMARY KEY,
  nome      VARCHAR(40),
  chproj    INTEGER,
  orcamento NUMERIC(12,2) CHECK (orcamento > 0)
);

CREATE TABLE matricula (
  codfunc    INTEGER REFERENCES funcionario(codfunc),
  codproj    INTEGER REFERENCES proj(codproj),
  dtalocacao DATE DEFAULT CURRENT_DATE,
  PRIMARY KEY (codfunc, codproj)
);
```

Observe que a restrição de domínio é expressa com **CHECK**, a obrigatoriedade com **NOT NULL** e o valor padrão com **DEFAULT**. São os três mecanismos declarativos de integridade que o exercício quer fixar.

6.1.2 Passo 2 — Alterar as tabelas com ALTER

Depois de criadas, o enunciado pede nove alterações usando o comando `ALTER`:

- a) na tabela Funcionário, mudar o campo `nome` para `nomefunc`;
- b) mudar o campo nome para `VARCHAR(50)`;
- c) mudar o campo `estcivil` para `CHAR(1)`;
- d) mudar o campo `cargo` para `VARCHAR(50)`;
- e) mudar o nome da tabela `Proj` para `projeto`;
- f) na tabela Projeto, mudar o campo `nome` para `nomeproj`;

- g) inserir a restrição NOT NULL no campo `nomeproj`;
- h) apagar o campo `chproj`;
- i) na tabela Matrícula, inserir o campo `dtfim`.

```
ALTER TABLE funcionario RENAME COLUMN nome TO nomefunc;
ALTER TABLE funcionario ALTER COLUMN nomefunc TYPE VARCHAR(50);
ALTER TABLE funcionario ALTER COLUMN estcivil TYPE CHAR(1);
ALTER TABLE funcionario ALTER COLUMN cargo TYPE VARCHAR(50);

ALTER TABLE proj RENAME TO projeto;
ALTER TABLE projeto RENAME COLUMN nome TO nomeproj;
ALTER TABLE projeto ALTER COLUMN nomeproj SET NOT NULL;
ALTER TABLE projeto DROP COLUMN chproj;

ALTER TABLE matricula ADD COLUMN dtfim DATE;
```

O exercício cobre os quatro verbos essenciais do ALTER: `RENAME`, `ALTER COLUMN TYPE/SET NOT NULL`, `DROP COLUMN` e `ADD COLUMN`.

6.1.3 Passo 3 — Inserir dados (DML)

Os dados a inserir na tabela Funcionário:

codfunc	nomefunc	genero	estcivil	cargo	salario
1	Ana	F	C	Programador	5000.00
2	Bruna	F	C	Analista de Sistemas	8500.00
3	Carla	F	S	Programador	5000.00
4	Danilo	M	D	Programador	5000.00
5	Elias	M	S	Analista de Sistemas	8500.00

As datas de admissão são 2020/11/03 para os dois primeiros e 2021/05/04 para os demais. Na tabela Projeto: (1, LGPD, 95000.00), (2, Business Intelligence, 220000.00) e (3, ITIL, 150000.00). Na tabela Matrícula, seis alocações, das quais quatro com `dtfim` nula e duas encerradas em 2023/05/10.

```
INSERT INTO funcionario VALUES
 (1,'Ana','F','C','2020-11-03','Programador',5000.00),
 (2,'Bruna','F','C','2020-11-03','Analista de Sistemas',8500.00);

INSERT INTO projeto VALUES (1,'LGPD',95000.00);

INSERT INTO matricula (codfunc, codproj, dtalocacao, dtfim)
VALUES (1, 2, '2023-01-03', NULL);
```

6.1.4 Passo 4 — Testar as integridades

O último item do enunciado é pedagogicamente o mais importante: **tentar realizar inserções que firam as integridades definidas no banco**. O objetivo é ver o SGBD rejeitar, por exemplo, um gênero `'X'`, um salário abaixo do mínimo, um orçamento negativo ou um `nomefunc` nulo — confirmando que as restrições declaradas estão efetivamente ativas.

6.2 Bloco 2 — SQL: linguagem DQL, joins e funções

A Oficina 04 trabalha sobre um **script de locadora de veículos** cedido pelo professor, com entidades de clientes, funcionários, carros, marcas, modelos, acessórios e aluguéis.

6.2.1 Consultas com JOIN

O roteiro pede dez consultas, em ordem crescente de dificuldade:

1. lista com os modelos e a marca de todos os carros;
2. nome do cliente, cidade e modelo do carro alugado;
3. nome dos acessórios alugados com valor menor ou igual a 20 reais;
4. nome dos clientes que alugaram carros no dia 10/04/2023;
5. todas as informações de aluguéis feitos por funcionários de Duque de Caxias;
6. nome dos clientes que alugaram carros do modelo Mobi;
7. nome do cliente, modelo do carro e nome da marca dos carros alugados, por ordem crescente de data de aluguel;
8. nome dos modelos de carros alugados por clientes solteiros;
9. data do aluguel e nome do acessório alugado;
10. nome do funcionário e nome dos acessórios alugados, **sem repetição**.

A progressão é intencional: começa com um join de duas tabelas, avança para joins de três ou mais, introduz filtros com `WHERE`, ordenação com `ORDER BY` e, no último item, `DISTINCT`.

```sql
-- Consulta 1
SELECT mo.nome AS modelo, ma.nome AS marca
FROM carro c
JOIN modelo mo ON mo.codmodelo = c.codmodelo
JOIN marca  ma ON ma.codmarca  = mo.codmarca;

-- Consulta 10
SELECT DISTINCT f.nome, a.nome
FROM aluguel al
JOIN funcionario f ON f.codfunc = al.codfunc
JOIN aluguel_acessorio aa ON aa.codaluguel = al.codaluguel
JOIN acessorio a ON a.codacessorio = aa.codacessorio;
```

6.2.2 Consultas com funções de agregação

O segundo conjunto de dez consultas exercita funções e agrupamentos:

1. valor do acessório mais caro e do mais barato (`MAX`, `MIN`);
2. valor médio dos acessórios (`AVG`);
3. contagem de acessórios em catálogo (`COUNT`);
4. soma do valor do aluguel de todos os acessórios (`SUM`);
5. quantidade de clientes por estado (`GROUP BY`);
6. quantidade de clientes por cidade, mostrando apenas cidades com **mais de 2 clientes** (`HAVING`);
7. clientes que **nunca** alugaram carros;

8. carros que **nunca** foram alugados;
9. acessório mais alugado;
10. nome do acessório e preço do item mais caro.

```sql
-- Consulta 6: agrupamento com filtro sobre o agregado
SELECT cidade, COUNT(*) AS qtd
FROM cliente
GROUP BY cidade
HAVING COUNT(*) > 2;

-- Consulta 7: negacao com LEFT JOIN
SELECT c.nome
FROM cliente c
LEFT JOIN aluguel a ON a.codcliente = c.codcliente
WHERE a.codcliente IS NULL;
```

Os itens 7 e 8 são os mais instrutivos: exigem entender que "nunca ocorreu" se expressa por `LEFT JOIN` com teste de nulo, ou por `NOT EXISTS`/`NOT IN`. Os itens 9 e 10 exigem combinar agregação com ordenação e limite, ou subconsulta com o valor máximo.

6.2.3 Referências indicadas

O material bibliográfico da oficina de SQL inclui os livros *SQL Structured Query Language* de Luís Damas (6ª edição, LTC, 2007) e *Sistemas de Banco de Dados* de Elmasri e Navathe (7ª edição, Pearson, 2018). Entre as ferramentas de apoio: o tutorial de SQL do W3Schools, os editores de diagramas Dia, Lucidchart, Draw.io, Astah e BrModelo (versões 2.0 e 3.2), o ranking de SGBDs do db-engines, o site oficial do PostgreSQL, o SQLite Online e a documentação de tipos de dados do PostgreSQL.

6.3 Bloco 3 — Python para iniciantes

6.3.1 Lógica e algoritmos

O ponto de partida é conceitual. A **lógica** é a parte da filosofia que trata das formas do pensamento e das operações intelectuais que visam determinar o que é verdadeiro ou não. No universo da tecnologia da informação, lógica é a organização e o planejamento das instruções e assertivas em um **algoritmo**, viabilizando a implantação de um programa. Programar, nesse enquadramento, "nada mais é do que a organização coesa de uma sequência de instruções voltadas à resolução de um problema de forma lógica".

6.3.2 Por que Python

A filosofia da linguagem, segundo os slides: totalmente gratuito, usabilidade, orientado a objetos, poderoso, com ferramentas gráficas poderosas, flexível e com vasta comunidade. As características destacadas: **open-source**, **compatibilidade**, uso mundial pela academia e pela indústria, e atualização constante com novas bibliotecas de uso livre.

Sobre IDEs — Jupyter, Spyder, PyCharm, VS Code, Google Colab — a orientação é pragmática: **"a melhor IDE é aquela que você se sente mais à vontade ao utilizar"**.

6.3.3 Conceitos básicos, passo a passo

1. Comentários. Quando o programa cresce, fica difícil ler e manter; por isso é boa prática inserir documentação ou notas no código. O comentário em Python começa com `#` e continua até o fim da linha; o interpretador ignora comentários. Há comentários em linha, em bloco e docstrings para documentação de funções, módulos, pacotes e classes.

2. Indentação. Python usa indentação para delimitar blocos, em vez de chaves. Tabs e espaços são suportados.

3. Variáveis e **4. Tipos de dados**:

- **Numérico**: `int` (inteiros positivos ou negativos, sem fração ou decimal), `float` (número real com representação de ponto flutuante, especificado por ponto decimal) e `complex` (parte real mais parte imaginária j).
- **Booleano**: classe `bool`, com dois valores internos, Verdadeiro ou Falso.
- **Sequências** (coleção ordenada de tipos de dados semelhantes ou diferentes): **String** (`str`, matrizes de bytes representando caracteres, delimitadas por aspas simples, duplas ou triplas, com acesso por índice), **Lista** (`list`, mutável, ordenada, contagem definida, podendo conter tipos heterogêneos) e **Tupla** (`tuple`, ordenada como a lista, mas **imutável**).
- **Set**: coleção **não ordenada**, iterável, mutável e **sem elementos duplicados**. A principal vantagem sobre a lista é possuir um método altamente otimizado para verificar se um elemento pertence ao conjunto.
- **Dicionário**: coleção não ordenada que armazena pares **chave:valor**, funcionando como um mapa. Cada par é separado por dois pontos e os pares entre si por vírgula.

5. Operadores, **Type casting** e **interação com o usuário** completam o bloco básico.

6.3.4 Exercícios do bloco básico

Os enunciados propostos:

1. Criar uma variável com valor 6 (inteiro) e outra com valor 2 (inteiro), dividir a primeira pela segunda salvando em nova variável e verificar o **tipo** do resultado da divisão. O objetivo é descobrir que a divisão `/` produz `float` mesmo entre inteiros.
2. Dois amigos dividem o lucro de um site de consultoria em projetos de IA. O lucro mensal é de 8k; o amigo 1 tem direito a 30% e o restante pertence ao amigo 2. Calcular o lucro total do ano e o lucro de cada um.
3. Prever mentalmente o resultado de `5 + 3 * 10 / 3 == 15` e depois confirmar no Python — exercício de precedência de operadores e de aritmética de ponto flutuante.

O exercício de listas usa dados do filme *The Shining*:

```
movieName = "The Shining"
actors = ["Jack Nicholson", "Shelley Duvall", "Danny Lloyd",
          "Scatman Crothers", "Barry Nelson"]
scores = [4.5, 4.0, 5.0]
comments = ["Best Horror Film I Have Ever Seen",
            "A truly brilliant and scary film from Stanley Kubrick",
            "A masterpiece of psychological horror"]
```

Pede-se: (1) criar uma lista com os quatro elementos carregados — nome do filme, atores, avaliações e comentários — e usar essa lista nas questões seguintes; (2) imprimir a string com o nome do primeiro ator; (3) imprimir a melhor avaliação do filme, com score e comentário. As dicas dadas são `max(lista)`, que retorna o máximo, e `lista.index(valor)`, que retorna o índice do valor. A composição das duas resolve o item 3: achar o máximo em `scores`, obter seu índice e usar esse índice para recuperar o comentário correspondente.

Outro exercício pede um script que pergunte a idade do usuário, aguarde a resposta e imprima a idade, o ano de nascimento (considerando que a pessoa já fez aniversário) e o tipo do dado.

6.3.5 Estruturas condicionais e de repetição

Os exercícios de laços:

1. Imprimir uma sequência de 25 números consecutivos.
2. Imprimir a sequência acima 2 vezes (dica: `for` dentro de `for`, ou `while` dentro de `while`).
3. Na lista `[12, 23, 11, 34, 13, 56, 102, 101, 13]`, imprimir somente os números pares (dica: operador de resto `%`; por exemplo, `10 % 2 == 0`).
4. **Mega Sena**: sortear 6 números entre 1 e 60, com e sem estrutura de repetição. A dica sem laço é `np.random.randint()`. A pergunta provocativa: *estrutura de repetição e `np.random.randint` sem nenhum tratamento pode gerar algum problema?* A resposta esperada é sim — números repetidos —, e a dica para contornar é `np.random.choice()`.
5. Simular o resultado de um dado de 6 faces jogado 7 vezes, com e sem estrutura de repetição.
6. Calcular o fatorial de um número qualquer, sabendo que 5! = 5 x 4 x 3 x 2 x 1 = 120.

6.3.6 Exercício da Lei dos Grandes Números

Este é o exercício-síntese do bloco. A **lei dos grandes números** afirma que a média amostral converge para o valor esperado conforme o número de observações cresce. O exemplo de lançamento de moeda mostrado nos slides:

- 10 lançamentos: 7/3, ou seja, 70% / 30%
- 100 lançamentos: 52/48, ou seja, 52% / 48%
- 1000 lançamentos: 502/498, ou seja, 50,2% / 49,8%

A tarefa: testar a lei para N números aleatórios gerados com **distribuição normal** de média 0 e desvio padrão 1. Deve-se criar um script que conte quantos desses números caem entre -1 e 1 e divida por N. Sabe-se que E(X) = 68,2%. Verifica-se então que a média tende a E(X) conforme N aumenta.

```
import numpy as np

N = 100000
amostra = np.random.normal(size=N)
dentro = np.sum((amostra > -1) & (amostra < 1))
print(dentro / N)   # tende a 0.682
```

6.3.7 Funções

As funções são o **primeiro passo para a reutilização de código**: permitem definir um bloco reutilizável, usado repetidamente no programa. Python fornece funções internas como `print()` e `len()`, mas o usuário também define as suas. Uma função tem **parâmetros de entrada**, um corpo e um **retorno**, e deve ser declarada **antes** de sua chamada.

Exercícios:

1. Criar uma função que calcule o fatorial de um número qualquer.
2. Usar a função `factorial` do pacote `math` para o mesmo cálculo.
3. Criar uma função que receba o raio de um círculo e retorne a área, usando o pacote `math` (dica: `math.pi` e o operador de potência `**`).
4. Alterar a função anterior para que retorne também o **diâmetro** e o **perímetro**; em seguida, alterar para que o raio tenha **valor padrão** de 5.
5. Criar uma função que valide a lei dos grandes números, recebendo como parâmetro o número de experimentos; depois incluir um **parâmetro opcional** de intervalo a ser validado, com default -1 e 1, e testar outros intervalos.

```
import math

def circulo(raio=5):
    area = math.pi * raio ** 2
    diametro = 2 * raio
    perimetro = 2 * math.pi * raio
    return area, diametro, perimetro

def lgn(n_exp, inf=-1, sup=1):
    x = np.random.normal(size=n_exp)
    return np.sum((x > inf) & (x < sup)) / n_exp
```

A progressão desses cinco exercícios é deliberada: função simples, função de biblioteca, função com pacote externo, **retorno múltiplo**, **argumento default** e por fim **argumento opcional aplicado a um problema estatístico** já resolvido antes sem função — evidenciando o ganho de reutilização.

6.4 Bloco 4 — Pacotes e NumPy

6.4.1 Formas de importação

```
import nome_do_pacote                    # importa todos os modulos do pacote
from nome_do_pacote import nome          # importa modulo/funcao especifica
from nome_do_pacote import *             # importa tudo, chamavel pelo nome
from nome_do_pacote import nome as apelido  # importa com nome alternativo
```

Exercício: importar a classe `pyplot` do pacote `matplotlib`, que contém a função `hist()` capaz de gerar um histograma; usar as amostras de diferentes tamanhos geradas com `np.random` e criar um histograma a partir de cada uma; avaliar o gráfico gerado. É a ponte entre a lei dos grandes números e a visualização.

6.4.2 NumPy

O **NumPy** é o pacote básico fundamental do Python para computação científica. Provê muitas das estruturas básicas e fornece ferramentas para integração e comunicação com Fortran e C, vetorização, álgebra linear e geração de números aleatórios, constituindo fundação sólida para muitas ferramentas de análise de dados.

O motivo de sua popularidade é a **eficiência** comparado às listas nativas: os slides destacam em maiúsculas que é **muito mais rápido** trabalhar com NumPy, pois o pacote permite acesso a dados de modo muito mais eficiente.

As estruturas de dados percorridas em escala crescente são **escalares, vetores, matrizes e tensores**. Um **vetor** numérico em Python é uma sequência indexada de 0 a n-1 com elementos de um mesmo tipo; uma **matriz** organiza os elementos em linhas e colunas.

6.4.3 Criação de estruturas

```
import numpy as np

t1 = np.ones((4, 3, 2))            # tensor 3D de uns
t2 = np.zeros((4, 3, 2))           # tensor 3D de zeros
t3 = np.random.random((4, 3, 2))   # tensor 3D aleatorio no intervalo 0 a 1

print(t1.shape)                    # (4, 3, 2)

X = np.array([[1, 2], [3, 4], [5, 6]])
print(X.shape)                     # (3, 2)

v = np.array([1, 2, 3, 4, 5, 6])
```

6.4.4 Operações

Um ponto conceitualmente crítico da oficina é distinguir os três tipos de multiplicação. Com A = [[1,2],[4,5]] e B = [[10,20],[30,40]]:

```
A * B      # multiplicacao element-wise: [[10, 40], [120, 200]]
A.dot(B)   # multiplicacao matricial:    [[70, 100], [190, 280]]
2 * A      # multiplicacao por escalar:  [[2, 4], [8, 10]]
```

Confundir `A * B` com `A.dot(B)` é o erro mais comum de quem vem de álgebra linear, e o slide o antecipa mostrando os três resultados lado a lado.

6.4.5 Exercício de eficiência

Operações vetorizadas são extremamente mais eficientes do que laços. O exercício pede provar isso: criar dois vetores com **100000 elementos** e fazer um produto interno (*dot product*), comparando listas nativas com NumPy. A dica é usar `time.process_time()` do pacote `time` para salvar o tempo antes e depois da operação; o tempo decorrido em segundos é a diferença entre esses valores.

```
import time

inicio = time.process_time()
resultado = np.dot(a, b)
fim = time.process_time()
print(fim - inicio)
```

6.4.6 Exercício: análise de demonstração financeira

Cenário: o aluno é um cientista de dados de uma consultoria e um colega do departamento de auditoria pede ajuda para avaliar o demonstrativo financeiro de uma organização X. Recebe dois vetores com receita e despesa mensais do ano:

```
receitas = np.array([14574.49, 7606.46, 18611.41, 19175.41, 8758.65,
                     8105.44, 11496.28, 9766.09, 10305.32, 18379.96,
                     10713.97, 15433.50])
custos   = np.array([12051.82, 5695.07, 12319.20, 12089.72, 8658.57,
                     840.20, 3285.73, 5821.12, 6976.93, 16618.61,
                     10054.37, 3803.96])
```

As métricas a calcular:

- **lucro** de cada mês;
- **lucro após imposto** de cada mês (imposto de 30% sobre o lucro);
- **margem de lucro** de cada mês (lucro após imposto dividido pela receita);
- **meses bons**: aqueles em que o lucro após taxação foi maior que o lucro médio do ano (dica: `np.where`);
- **meses ruins**: o contrário;
- o **melhor mês** e o **pior mês**, descontado o imposto.

Funções úteis indicadas: `mean()`, `max()`, `min()` e `np.where()`. O exercício é a demonstração prática da vetorização: cada uma dessas seis métricas se resolve em uma linha, sem nenhum laço.

6.5 Bloco 5 — Visualização de dados com Matplotlib

O **Matplotlib** é uma biblioteca destinada à criação de gráficos estáticos, animados e interativos. A importação canônica e o conjunto de dados de trabalho:

```
import matplotlib.pyplot as plt

years_x = [1975, 1980, 1985, 1990, 1995, 2000, 2005, 2010, 2015]
total_y = [1243, 1543, 1619, 1831, 1960, 2310, 2415, 2270, 1918]
coal_y  = [823, 1136, 1367, 1547, 1660, 1927, 1983, 1827, 1352]
gas_y   = [171, 200, 166, 175, 228, 280, 319, 399, 529]
```

A lista `total_y` representa a quantidade de carbono emitido na atmosfera e `years_x` representa os anos; `coal_y` e `gas_y` acrescentam produção de carvão e gás natural.

6.5.1 Enriquecendo o plot passo a passo

O roteiro apresenta os métodos chamados **antes** de `plt.show()`:

1. **Título**: `plt.title()`
2. **Rótulos dos eixos**: `plt.xlabel()` e `plt.ylabel()`
3. **Marcações dos eixos**: `plt.xticks()` e `plt.yticks()`, mais `plt.xlim()` e `plt.ylim()` para limites
4. **Legenda**: primeiro adicionar `label` a cada plot, depois chamar `plt.legend()`
5. **Grade**: `plt.grid(axis="y", linewidth=0.5)`
6. **Estilos**: `print(sorted(plt.style.available))` lista os disponíveis; `plt.style.use('seaborn')` aplica globalmente. Para isolar a configuração de um único plot dentro de um notebook sem afetar os demais, usa-se o gerenciador de contexto `with plt.style.context('seaborn'):`
7. **Linhas**: controle total sobre `matplotlib.lines.Line2D` via `color`, `marker` (default `None`), `linestyle` (default `-`) e `linewidth` (default 1.5)
8. **Tamanho da figura**: `plt.figure(figsize=(10,5))`

O exemplo consolidado:

```
plt.figure(figsize=(10, 5))
plt.plot(years_x, total_y, label='total', c="grey", ls=':', marker='s')
plt.plot(years_x, coal_y, label='coal')
plt.plot(years_x, gas_y, label='gas')
plt.legend()
plt.title('CO2 emissions from electricity production - US')
plt.ylim((0, 3000))
plt.ylabel('MtCO2/yr')
plt.grid(lw=0.5)
plt.show()
```

6.5.2 Axes versus Axis — a virada conceitual

A distinção é central para o Matplotlib avançado:

- **Axis** é o *eixo* do plot, aquele que recebe as marcações e o título;
- **Axes** é a *área* dentro da qual o plot aparece.

Acessa-se a instância atual de Axes com `ax = plt.gca()`. Comparando as duas formas:

```
# interface pyplot
plt.plot(years_x, total_y)
plt.ylabel('MtCO2/yr')
plt.title('CO2 emissions from electricity production - US')
plt.show()

# interface via Axes
plt.plot(years_x, total_y)
ax = plt.gca()
ax.set_title('CO2 emissions from electricity production - US')
ax.set_ylabel('MtCO2/yr')
plt.show()
```

A diferença de nomenclatura é sistemática: `matplotlib.pyplot.title` versus `matplotlib.axes.Axes.set_title`. Os motivos para acessar `ax` são três: **customização e refinamento**; **criação de múltiplos subplots**; e **integração com outras bibliotecas** como pandas.

6.5.3 Ajuste fino das spines

As *spines* são as linhas de contorno do gráfico. Pode-se removê-las ou colorí-las:

```
ax.spines['right'].set_color(None)
```

E também mudar sua posição:

```
ax.spines['bottom'].set_position(('axes', 0.5))   # metade do eixo y
ax.spines['bottom'].set_position(('data', 750))   # no valor 750 do eixo y
```

6.5.4 Figures, Subplots e Axes

No Matplotlib, **figure** significa toda a janela na interface de usuário. Dentro dela pode haver vários subgráficos, organizados e numerados em uma grade de (nlinhas, ncolunas). **Subplots** pertencem à classe Axes e podem ser considerados equivalentes em primeira ordem; a única diferença é que se pode criar um `ax` sem grade de subgráficos, posicionando-o em posição absoluta dentro da figura.

Há três formas de construir múltiplos subplots.

Interface convencional:

```
plt.figure(figsize=(10, 3))
plt.subplot(1, 2, 1)
plt.plot(years_x, coal_y, label="coal")
plt.plot(years_x, gas_y, label="gas")
plt.title('coal vs. gas')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(years_x, total_y, label="total", c='black')
plt.title("all energies")
plt.suptitle('US electricity CO2 emissions')
plt.show()
```

Interface orientada a objetos:

```
fig = plt.figure(figsize=(10, 3))
ax1 = fig.add_subplot(1, 2, 1)
ax1.plot(years_x, coal_y, label="coal")
ax1.plot(years_x, gas_y, label="gas")
ax1.set_title('coal vs. gas')
ax1.legend()
ax2 = fig.add_subplot(1, 2, 2)
ax2.plot(years_x, total_y, c='black')
ax2.set_title('all energies')
fig.suptitle('US electricity CO2 emissions')
plt.show()
```

Atribuição por desestruturação, o atalho encontrado com frequência na documentação oficial:

```
fig, axs = plt.subplots(1, 2, figsize=(10, 3))   # axs e um array (1,2)
axs[0].plot(years_x, coal_y, label="coal")
axs[0].plot(years_x, gas_y, label="gas")
axs[0].set_title('coal vs. gas')
axs[0].legend()
axs[1].plot(years_x, total_y, c='black')
axs[1].set_title('all energies')
plt.suptitle('US electricity CO2 emissions')
plt.show()
```

6.5.5 Outros tipos de gráfico

Scatter plot — utiliza coordenadas cartesianas (x, y) para exibir valores de um ou vários conjuntos de dados:

```
np.random.seed(19680801)          # fixa o estado aleatorio para reprodutibilidade
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N)) ** 2
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```

Bar plot:

```
fig, ax = plt.subplots()
fruits = ['apple', 'blueberry', 'cherry', 'orange']
counts = [40, 100, 30, 55]
bar_colors = ['tab:red', 'tab:blue', 'tab:red', 'tab:orange']
ax.bar(fruits, counts, color=bar_colors)
ax.set_ylabel('fruit supply')
ax.set_title('Fruit supply by kind and color')
plt.show()
```

Histograma — representação precisa da distribuição de dados numéricos:

```
x = np.random.normal(size=10_000)
plt.hist(x, bins=100)   # o eixo horizontal traz as frequencias de cada bin
```

6.6 Bloco 6 — pandas, estatística e publicação de bibliotecas

Os encontros mais avançados de Python trazem notebooks de apoio dedicados a: declaração de funções e importação de bibliotecas; criação de funções em Python; operações com DataFrames e criação de gráficos; pandas propriamente dito (`SAD_Oficina_pandas.ipynb`); revisão de estatística e visão geral de base de dados; e agrupamento e estatística em DataFrames. As bases usadas incluem `results_modif.csv`, `biostats.txt`, `movies.sqlite` e `FuelEfficiency.csv`. *Material de slides não disponível para esses notebooks; o conteúdo está nos próprios arquivos e na gravação da aula.*

6.6.1 Publicação de biblioteca Python no GitHub

Um material extra trata da publicação de bibliotecas, respondendo a quatro perguntas: por que usar biblioteca, como é o processo de importação, como publicar e como usar a biblioteca publicada.

O motivo da publicação é enunciado como um problema concreto: **e quando o código deve estar disponível para várias máquinas?** A solução é torná-lo acessível online. O passo a passo:

1. **Primeiro passo**: gerar o arquivo `setup.py`.
2. **Segundo passo**: publicar o código-fonte (no GitHub).
3. **Instalação**: instalar a biblioteca publicada, a partir do repositório ou do índice de pacotes.

6.7 Bloco 7 — Dashboards com Power BI

A Oficina 01 de visualização de dados usa a planilha `vendaCarros.xlsx` cedida pelo professor. O enunciado é integralmente prescritivo e serve de checklist para a construção do painel.

6.7.1 Passo 1 — Gráficos

Criar um dashboard que exiba:

- **percentual de vendas por estado**;
- **custo por fabricante**;
- **cores mais vendidas**;
- **vendas por ano**.

6.7.2 Passo 2 — Filtros

Incluir segmentações por:

- **Fabricante**;
- **Estado**;
- **Modelo**;
- **Cliente**.

6.7.3 Passo 3 — Cartões

Incluir cartões (indicadores numéricos) com:

- **total de vendas por ano**, considerando a coluna `ValorVenda`;
- **total de descontos aplicados**, considerando a coluna `TotalDesconto`.

6.7.4 Passo 4 — Layout do dashboard

- elaborar **apenas 1 painel**;
- criar um **template com cabeçalho e título**.

A restrição de um único painel é intencional: força decisões de hierarquia visual e de uso do espaço, em vez de espalhar visuais por várias páginas. Os arquivos de apoio incluem `logocarro.jpg` para o cabeçalho e o arquivo `.pbix` de referência.

7 OFICINA - DM

A oficina de Data Mining é organizada em torno de um programa amplo, recapitulado no início de cada encontro, que abrange: **análise exploratória**; **pré-processamento** (balanceamento, outliers, missing values, normalização e seleção de atributos por filtros, wrappers e PCA); **classificação** (regressão logística, SVM, árvores de decisão, Random Forest, redes neurais e KNN); **regressão** (linear e não linear, simples e múltipla); **agrupamento** (particionamento com K-means e K-medoids, hierárquico com DIANA e AGNES, densidade com DBSCAN); **associação** (Apriori, FP-Growth, Eclat); e **séries temporais** (Naive, média móvel, amortecimento exponencial, ARIMA e auto-regressivo não linear).

Data Mining é apresentado como campo **interdisciplinar**, com **5 classes de problemas** e um esquema básico de projeto que estrutura todas as oficinas.

7.1 Encontro 1 — Tratamento de dados

7.1.1 Análise exploratória: a base Mushroom

A base **Mushroom** (disponível no Kaggle) inclui descrições de amostras hipotéticas correspondentes a **23 espécies de cogumelos**. Cada espécie é identificada como definitivamente comestível, definitivamente venenosa, ou de consumo desconhecido e não recomendado — esta última classe combinada com a venenosa. O detalhe que torna a base interessante para mineração está no próprio guia de origem, que afirma claramente **não existir uma regra simples** para determinar a comestibilidade de um cogumelo. É exatamente o tipo de problema em que um algoritmo pode descobrir regras que o especialista não formulou.

7.1.2 Missing values

Valores faltantes são muito comuns no mundo real. As causas apontadas: atributos novos que surgem com o tempo e com a necessidade de novas informações por parte das empresas; e atributos não preenchidos por falta de obrigatoriedade. A observação prática: verificar se o número de registros com dados faltantes para um atributo específico não justifica, por si só, a **remoção do atributo**.

O quadro geral de tratamento:

- **Deletar**: o registro ou o atributo;
- **Imputar valor**: por **estatística** (valor único ou sequência) ou por **machine learning**.

Os slides mostram três estratégias de imputação com o mesmo exemplo, uma tabela de cinco registros com Idade, Estado Civil, Nota e Atrito:

1. Substituição pela média. Para a nota faltante, calcula-se a média das notas presentes: (9 + 7 + 10 + 8) / 4 = **8,5**, valor que substitui o ausente.

2. Substituição pela média baseada em outro atributo. Aqui a média é condicionada. Restringindo aos registros com Atrito = "Sim", a média das notas é (9 + 7) / 2 = **8**, valor que substitui o ausente. A imputação condicional é mais informativa que a média global porque preserva a relação entre o atributo faltante e o atributo condicionante.

3. Substituição pelo valor mais frequente. Para o Estado Civil faltante, adota-se a moda: "Casado", que aparece três vezes contra uma de "Solteiro". É a estratégia natural para atributos categóricos, para os quais média não faz sentido.

7.1.3 Normalização

Os dois objetivos da normalização são explícitos: **dar aos atributos pesos iguais** e **diminuir o tempo de convergência dos algoritmos**. O primeiro evita que um atributo em escala de milhares domine outro em escala unitária apenas por causa da unidade de medida; o segundo é um efeito numérico do treinamento.

O bloco também menciona a **conversão de atributos**, necessária quando o algoritmo exige entrada numérica e a base traz categorias.

7.1.4 Redução de dimensionalidade

A **maldição da dimensionalidade** (*curse of dimensionality*) é o termo que se refere a vários fenômenos que surgem na análise de dados em espaços com muitas dimensões — muitas vezes centenas ou milhares. A consequência prática: **adicionar características não significa sempre melhora no desempenho de um classificador**. De modo geral, o desempenho tende a se **degradar** a partir de um determinado número de atributos, mesmo que eles sejam atributos úteis.

Os objetivos da redução: diminuir o custo do aprendizado; aumentar a precisão do algoritmo; e gerar modelos compactos, mais fáceis de interpretar. Em geral espera-se que todos os atributos sejam relevantes, mas nem sempre isso é garantido, e alguns são **redundantes**. O objetivo é definir um conjunto de atributos relevantes e não redundantes.

Há duas grandes abordagens:

- **Seleção de atributos**: escolha de um subconjunto de atributos relevantes dentre os disponíveis (por exemplo, filtros e wrappers);
- **Agregação de atributos**: criação de novos atributos a partir da combinação dos existentes (por exemplo, PCA).

7.1.5 Seleção — filtros

Os métodos de seleção por **filtro** aplicam uma medida estatística para atribuir uma pontuação a cada atributo. Os atributos são ranqueados por essa pontuação e então selecionados para serem mantidos ou removidos.

O exemplo trabalhado é o **ganho de informação**, cujo passo a passo é:

1. Calcular a **entropia para a classe** (0 significa dados homogêneos; 1 significa dados igualmente distribuídos).
2. Dividir a base nos diferentes atributos e calcular a entropia para cada um deles; o **ganho de informação** é a entropia para a classe menos a entropia para o atributo.

O ganho de informação é, portanto, baseado na diminuição da entropia depois que a base é subdividida por um atributo. Quanto maior a queda de entropia, mais informativo o atributo.

7.1.6 Seleção — wrappers

Os métodos **wrapper** consideram a seleção de um conjunto de atributos como um **problema de busca**, no qual diferentes combinações são preparadas, avaliadas e comparadas. Um modelo preditivo é usado para avaliar cada combinação e atribuir uma pontuação baseada na precisão do modelo.

O exemplo dos slides conecta diretamente esta oficina com a de algoritmos genéticos. Suponha uma base com 5 atributos, A1 a A5. Constrói-se um **cromossomo** em que cada gene assume 0 ou 1: **0 significa sem o respectivo atributo, 1 significa com o respectivo atributo**. Cada indivíduo criado durante a evolução do GA é apresentado ao algoritmo classificador; toda a base, restrita aos atributos escolhidos, é usada para treinar o classificador, e a função de avaliação pode ser, por exemplo, a **acurácia de treinamento**.

Os slides mostram a evolução: um indivíduo inicial `01101` (atributos A2, A3 e A5) obtém acurácia de **30%**; o algoritmo genético evolui até chegar ao indivíduo `11001`, com acurácia de **93%**.

7.1.7 Seleção — embarcados

Nos métodos **embedded**, o processo de seleção faz parte do próprio algoritmo de aprendizado. O exemplo dado é a **árvore de decisão**, que ao escolher os atributos de cada nó já está, implicitamente, selecionando atributos.

7.1.8 Agregação e PCA

O exemplo introdutório de agregação é aritmético: dois atributos, “massa” e “volume”, podem ser agregados em um único atributo, “densidade” = massa / volume. Nesse caso **não há perda de informação**.

A **PCA (Análise de Componentes Principais)** consiste em transformar um conjunto de variáveis originais em outro conjunto de variáveis de mesma dimensão, denominadas **componentes principais**, com propriedades importantes:

- cada componente principal é uma **combinação linear** de todas as variáveis originais;
- todos os componentes são **ortogonais** entre si, portanto não há informações redundantes;
- são estimados com o propósito de reter, em ordem de estimação, o **máximo de informação** em termos da variação total contida nos dados.

O resultado é a redução da massa de dados com a menor perda possível de informação. A PCA é completamente **reversível**, o que a torna versátil para redução e compressão de dados. O exemplo numérico dado: com limiar de variância de **0,95**, obtêm-se **31 componentes principais** a partir de um total de **49 atributos**.

7.1.9 Balanceamento de dados

A grande maioria das bases não tem o mesmo número de instâncias em cada classe; quando a diferença é pequena, não há problema. Em algumas bases o desbalanceamento é esperado — por exemplo, em bases de transações fraudulentas, a maioria será "Normal" e uma pequena minoria será "Fraudulenta".

O problema, quando a classe 1 representa 95% e a classe 2 apenas 5%: os classificadores ficam **"preguiçosos"**, já que se consegue alta acurácia classificando tudo como classe 1 — no exemplo, 95% de acurácia sem ter aprendido nada. Isso é o **paradoxo da acurácia**.

As seis estratégias apresentadas:

1. Coletar mais dados.

2. Utilizar diferentes métricas de performance: matriz de confusão, precisão, recall, Kappa e F1 score. Nenhuma delas se deixa enganar por um classificador trivial da forma como a acurácia se deixa.

3. Reamostrar o conjunto de dados: **over-sampling** aleatório (replicar exemplos da classe minoritária) ou **under-sampling** aleatório (descartar exemplos da classe majoritária).

4. Gerar amostras sintéticas, como o **SMOTE** (*Synthetic Minority Over-Sampling Technique*): método de over-sampling que gera amostras **sintéticas** da classe rara — em vez de criar cópias — perturbando um atributo por vez por um valor dentro da diferença entre os exemplos vizinhos. É a diferença essencial em relação ao over-sampling aleatório: o SMOTE cria pontos novos no espaço de atributos, não duplicatas. Os slides mencionam ainda **GANs** como alternativa de geração.

5. Testar diferentes algoritmos: a orientação é explícita — não se deve ter um algoritmo favorito nesse caso, deve-se tentar diferentes algoritmos.

6. Penalização: custo adicional para a classificação errada da classe rara. No exemplo, com razão entre classes de 5:1, classificar erroneamente como classe 0 custa 1, enquanto classificar erroneamente como classe 1 custa 5.

7.1.10 Outliers

Algoritmos de aprendizado de máquina são sensíveis à distribuição e ao intervalo dos dados. Outliers podem enviesar e induzir ao erro, resultando em **maior tempo de treinamento**, **menor acurácia** e **modelos piores**. As três abordagens de detecção:

1. **análise de valores extremos**;
2. **métodos de proximidade** (por exemplo, k-means);
3. **métodos de projeção** (por exemplo, Kohonen).

7.1.11 Estudos de caso do encontro

SECOM — um processo moderno de fabricação de semicondutores é monitorado por sinais e variáveis coletadas de sensores e pontos de medição. Nem todos esses sinais são igualmente valiosos: eles contêm uma combinação de informações úteis, informações irrelevantes e ruído. Engenheiros tipicamente têm muito mais sinais do que o necessário. Considerando cada tipo de sinal como uma característica, a seleção de atributos pode identificar os sinais mais relevantes, que os engenheiros de processo então usam para determinar os fatores-chave da produção — trazendo redução de tempo e diminuição de custos. A base tem **1567 exemplos e 591 atributos**, uma proporção que ilustra bem a maldição da dimensionalidade.

IBM Analytics Employee Attrition & Performance — conjunto de dados fictício criado por cientistas de dados da IBM, para descobrir os fatores que levam ao desgaste de funcionários e explorar questões como a influência da distância de casa em função do cargo e do atrito, ou a comparação da renda média mensal por educação e atrito. São **34 atributos** e **2 classes** (Attrition: Yes / No). Vários atributos são escalas ordinais codificadas: `Education` (1 'Below College' a 5 'Doctor'), `EnvironmentSatisfaction`, `JobInvolvement`, `JobSatisfaction` e `RelationshipSatisfaction` (1 'Low' a 4 'Very High'), `PerformanceRating` (1 'Low' a 4 'Outstanding') e `WorkLifeBalance` (1 'Bad' a 4 'Best'), além de `Gender`, `MaritalStatus`, `MonthlyIncome`, `Age`, `DistanceFromHome`, `OverTime`, `JobRole` e `YearsWithCurrentManager`.

7.2 Encontro 2 — Classificação

7.2.1 O mapa do aprendizado de máquina

O encontro começa situando a classificação no mapa geral. **Machine Learning** divide-se em:

- **Supervisionado**: os dados têm atributos e **rótulo**. Abrange **classificação**, **regressão** e **previsão de séries temporais**.
- **Não supervisionado**: os dados têm apenas atributos. Abrange **agrupamento** e **associação**.
- **Reforço**: aprendizado através da interação de agentes com um ambiente.

Dentro do supervisionado, o modelo é um **aproximador**, uma função que mapeia entradas e saída. A distinção entre as três tarefas é o tipo de rótulo:

Tarefa	Rótulo	Exemplo
Classificação	Categórico	Aprovado / Reprovado a partir de dados do estudante
Regressão	Contínuo	Nota a partir de dados do estudante
Previsão de séries	Contínuo e dependente do tempo	Previsão a partir de dados históricos

No não supervisionado, **agrupamento** é a descoberta de semelhanças e grupos entre registros, enquanto **associação** é a descoberta de relações entre variáveis.

7.2.2 SVM

A ideia geral do **SVM não linear**: o espaço de atributos original pode — com alta probabilidade, pelo **Teorema de Cover** — ser mapeado para um espaço de atributos maior, no qual os dados podem ser separados. Formalmente, aplica-se uma transformação $\Phi : x \to \varphi(x)$. Um dos encontros da trilha DM é dedicado inteiramente a SVM, com o notebook `cls_svm.ipynb` sobre a base `Sleep_health_and_lifestyle_dataset.csv`.

7.2.3 Árvores de decisão e comitês

As árvores de decisão são apresentadas pelo algoritmo **ID3**, que produz **regras de decisão** legíveis.

Os **comitês** (*ensembles*) agregam múltiplos modelos treinados com o objetivo de melhorar a acurácia do modelo conjunto. A intuição: simula o que fazemos quando combinamos o conhecimento de vários especialistas em um processo de tomada de decisão. As técnicas discutidas são **bagging**, **boosting**, **stacking** e o **Random Subspace Method (RMS)**.

O **RMS** é similar ao bagging, mas com aleatorização sobre os **atributos**: os classificadores-base aprendem em subespaços S de mesma dimensão e a decisão final é por votação.

7.2.4 Random Forest — workflow

O passo a passo apresentado:

1. Cada árvore é construída usando uma inicialização (**bootstrap**) diferente do dataset original.
2. Aproximadamente **1/3 dos casos** ficam de fora da amostra de inicialização e não são usados na construção daquela árvore.
3. Cada caso deixado de fora é apresentado a cada uma das árvores da floresta, e cada árvore retorna uma classificação.
4. Para cada caso apresentado à floresta, verifica-se a classe que obteve o **maior número de votos**.
5. A proporção de votos diferentes da classe alvo em relação ao total de votos é o **erro OOB** (*Out-Of-Bag estimate*).

A elegância do OOB é ser uma estimativa de erro obtida sem conjunto de validação separado: os próprios casos deixados de fora do bootstrap fazem esse papel.

7.2.5 Estudo de caso: Netflix Prize

O **Netflix Prize** ofereceu **1 milhão de dólares** por uma melhora de **10%** na acurácia do sistema de recomendação de filmes da Netflix. A tarefa era de aprendizado supervisionado: os dados de treinamento eram formados por um conjunto de usuários e suas avaliações de filmes (1 a 5 estrelas), e o objetivo era construir um classificador que, dado um usuário e um filme não avaliado, classificasse corretamente aquele filme como 1, 2, 3, 4 ou 5 estrelas. A lição registrada nos slides: **os melhores times combinaram diversos modelos e algoritmos em um comitê** — validação empírica de peso para o conteúdo sobre ensembles.

7.2.6 KNN passo a passo

O **K-Nearest Neighbors** é apresentado em cinco passos explícitos:

1. **Determinar o valor de K**, ou número de vizinhos (no exemplo, K = 5).
2. **Calcular a distância** entre cada par de registros.
3. **Determinar quais são os K registros (vizinhos) mais próximos** do novo registro, por distância euclidiana.
4. **Contar**, dentre esses K vizinhos, o número de vizinhos em cada classe (no exemplo, 3 de uma classe e 2 de outra).
5. **Atribuir ao novo registro a classe majoritária** entre os vizinhos mais próximos.

Em resumo: a classe do novo padrão é igual à da maioria entre os K mais próximos. Restam três pendências, que são as decisões de projeto do algoritmo:

Qual tipo de distância usar? e **Qual valor de K?** — ambas resolvidas por **escolha experimental**.

Como desempatar? Três estratégias, ilustradas com K = 4:

- **Escolha aleatória**: "jogue uma moeda honesta" — se sair cara, escolha a classe vermelha; se sair coroa, a azul.
- **Escolha aleatória ponderada**: "jogue uma moeda desonesta" — dê mais chance à classe que possui mais padrões.
- **Classe mais próxima**: selecione a classe cuja distância é menor.

7.2.7 Regressão logística

A construção conceitual parte da **regressão linear simples**, $y = b_0 + b_1 x_1$, e observa que, quando se quer modelar **probabilidades**, os valores previstos pela reta deixam de fazer sentido (podem ultrapassar 0 e 1). A solução é aplicar a função sigmoidal:

$f(a) = \frac{1}{1+e^{-a}}$

o que produz a regressão logística:

$y' = f(y) = \frac{1}{1+e^{-(b_0+b_1 x_1)}}$

O exemplo dos slides relaciona idade e probabilidade: para idades 20, 30, 40 e 50, obtêm-se probabilidades de **0,7%**, **23%**, **85%** e **99,4%** — a curva em S característica. Para a **inferência**, adota-se um limiar, tipicamente **0,5**, acima do qual se classifica em uma classe e abaixo na outra.

7.2.8 Estudo de caso: câncer de mama

Base da **University of Wisconsin, Clinical Sciences Center**, com **30 atributos** mais classe e id, e **569 instâncias**, sendo **357 benignas** e **212 malignas**. Entre os atributos: raio (distância média do centro a pontos no perímetro do tumor), textura (desvio padrão dos valores em escala de cinza), perímetro e área.

O roteiro da atividade tem três passos:

1. Criar um modelo **KNN** para classificar os tumores em maligno ou benigno, na ferramenta de preferência (Python e/ou RapidMiner), usando o arquivo `breastCancer.csv`.
2. Fazer um **pós-processamento conservador**: somente inferências com **80% de probabilidade** serão classificadas como benigno. Este passo é conceitualmente importante — em um contexto clínico, o custo de um falso negativo (classificar como benigno um tumor maligno) é muito maior que o de um falso positivo, e ajustar o limiar de decisão é a forma direta de refletir essa assimetria.
3. Repetir para a base `breastCancer_3classes.csv`, que tem uma classe adicional: a classe **'Suspeito'**.

Os notebooks de apoio incluem versões com KNN, com regressão logística e uma versão com **Optuna** para otimização de hiperparâmetros (`Cancer_Mama_KNN_optuna.ipynb`).

7.3 Encontro 3 — Regras de associação

7.3.1 Conceitos e métricas

O problema de associação pertence ao aprendizado não supervisionado: descobrir **relações entre variáveis**, expressas como **regras de associação** da forma "se A então B".

As duas métricas fundamentais são **suporte** e **confiança**, com o **lift** como terceira medida crítica.

Para uma regra $A \rightarrow B$:

$Confiana(A \rightarrow B) = \frac{Suporte(A \cap B)}{Suporte(A)}$

$LIFT(A \rightarrow B) = \frac{Confiana(A \rightarrow B)}{Suporte(B)} = \frac{Suporte(A \cap B)}{Suporte(A) \cdot Suporte(B)}$

A forma final do lift revela uma propriedade importante: **ele independe da ordem** — o lift de A para B é igual ao de B para A. O que muda entre as duas direções é o suporte da premissa e a confiança da regra, mas o lift é simétrico.

7.3.2 Entendendo o lift — o exemplo Netflix

Este é o exemplo mais didático do material. Considere 100 usuários da Netflix, dos quais **10 gostaram de 'Breaking Bad'** e **40 gostaram de 'Dexter'**. Destes, **7 pessoas que gostaram de 'Breaking Bad' também gostaram de 'Dexter'**. A hipótese: é provável que quem viu 'Dexter' goste também de 'Breaking Bad'.

Para a regra **Se Dexter então Breaking Bad**:

- Se a Netflix recomendasse 'Breaking Bad' aleatoriamente, a probabilidade da pessoa gostar seria de **10%** (esse é o suporte do consequente);
- Usando o conhecimento a priori (quem gostou de 'Dexter' gostará de 'Breaking Bad'), a probabilidade sobe para **17,5%**;
- **Lift**: melhora de **75%** utilizando o conhecimento a priori.

Para a regra inversa, **Se Breaking Bad então Dexter**:

- Recomendação aleatória de 'Dexter': probabilidade de **40%**, ou seja, Suporte = 0,4;
- Com conhecimento a priori: 7/10 = **70%**, ou seja, Confiança = 0,7;
- **Lift** = 0,7 / 0,4 = **1,75**, isto é, os mesmos 75% de melhora.

A simetria fica demonstrada numericamente.

7.3.3 Preparação da base

A base deve ser transformada em uma **matriz esparsa** para ser apresentada ao algoritmo de associação. O exemplo dos slides parte de seis transações:

id	Itens
1	Ovo, Leite, Manteiga
2	Manteiga
3	Chocolate
4	Água
5	Refrigerante, Água
6	Leite, Maçã

e produz a matriz binária com uma coluna por produto (Ovo, Leite, Manteiga, Chocolate, Água, Refrigerante, Maçã), preenchida com 1 quando o item pertence à transação e 0 caso contrário. A transação 1 vira `1 1 1 0 0 0 0`; a transação 5 vira `0 0 0 0 1 1 0`; e assim por diante.

7.3.4 Como escolher os parâmetros

O material dá regras práticas concretas, ancoradas na base do estudo de caso (7501 transações em uma semana):

SUPPORT — para otimizar a venda de produtos comprados pelo menos 5 vezes ao dia, o cálculo semanal é 5 x 7 = 35 ocorrências, logo o suporte mínimo é 35/7501, aproximadamente **0,005**.

CONFIDENCE — o dilema é enunciado nos dois extremos: confiança **baixa** produz regras que não fazem sentido; confiança **alta** produz regras óbvias. A recomendação é começar com o valor default de **0,8** e ir diminuindo por tentativa e erro. Um valor de 0,8 significa que todas as regras geradas devem estar corretas em 80% das transações.

LIFT — ordenar as regras pelo lift em ordem **decrescente**. O lift é a medida de melhora na recomendação considerando o conhecimento a priori, e é ele que separa a regra útil da regra trivial.

7.3.5 Algoritmo Apriori

O Apriori opera por níveis, gerando conjuntos de itens candidatos e podando os que não atingem o suporte mínimo, para então derivar as regras. O estudo de caso é a base de **transações de um supermercado francês**: cada linha é uma transação, cada transação tem de 1 a N itens, existem **119 produtos diferentes** no mercado e a base tem **7501 transações** feitas ao longo de uma semana.

O exercício prático inclui uma provocação instrutiva: colocar $\mathbf{T = 0}$ no operador 'Apriori' e ordenar por *confidence*, para ver o que acontece; e depois voltar o parâmetro para $\mathbf{T = 1}$, ordenando por lift. A dica dada é que **'mineral water', 'eggs' e 'spaghetti'** são os 3 itens mais comprados. Com T = 0 e ordenação por confiança, as regras que emergem tendem a apontar justamente para esses itens mais frequentes — regras de alta confiança mas de lift baixo, porque o consequente já é provável por si só. É a demonstração empírica de por que se ordena por lift.

Os objetivos de negócio listados: perceber associações e implementar estratégias de oferta desses produtos; perceber associações **nada óbvias**; automatizar sistema de recomendação; aumentar vendas; e oferecer produtos condizentes com o perfil do cliente.

7.3.6 Algoritmo FP-Growth

O **FP-Growth** (*Frequent Pattern Growth*) é um dos algoritmos mais populares para cálculo de termos frequentes. Usa uma representação eficiente da base na forma de estrutura em árvore, a **FP-tree**. Faz **dois scans** no banco: o primeiro para contar itens frequentes, o segundo para construir a FP-tree. Uma vez construída a árvore, usa uma abordagem recursiva **divide-and-conquer** para obter os conjuntos de itens frequentes.

Passo 1 — ordenar os itens por prioridade. A partir das 10 transações do exemplo, conta-se a frequência de cada item e atribui-se prioridade:

Item	Frequência	Prioridade
A	7	1
B	8	2
C	7	3
D	4	4
E	3	5

Cada transação é então reescrita com seus itens nessa ordem: `{B,A}` vira `{A,B}`, `{D,C,B}` vira `{B,C,D}`, `{E,D,C,A}` vira `{A,C,D,E}`, e assim por diante.

Passo 2 — construir a árvore incrementalmente. Lê-se transação por transação, criando ou incrementando nós a partir da raiz `null`. Depois de ler TID=1 (`{A,B}`), tem-se o ramo A:1 seguido de B:1. Depois de TID=2 (`{B,C,D}`), acrescenta-se um novo ramo B:1, C:1, D:1 diretamente da raiz. Depois de TID=3 (`{A,C,D,E}`), o nó A passa a A:2 e ganha um ramo C:1, D:1, E:1. Ao final das 10 transações, a árvore tem A:7 e B:3 como filhos da raiz, com B:5, C:3, C:1, D:1, E:1 e demais nós nos níveis inferiores. Apontadores (a *header table*) ligam todas as ocorrências de um mesmo item, facilitando a geração dos termos frequentes.

Passo 3 — minerar por conditional pattern bases, em **ordem inversa de frequência**, ou seja, começando pelo item menos frequente.

Para **E**, com suporte mínimo = 2, a *conditional pattern base* é P = {(A,C,D:1), (A,D:1), (B,C:1)}. A *conditional FP-tree* resultante é {(A:2, C:1, D:2), (B:1, C:1)}, e os padrões frequentes extraídos são **{(A,E:2), (D,E:2), (A,D,E:2)}**.

Em seguida vem **D**. A *conditional pattern base* é P = {(A,B,C:1), (A,B:1), (A,C:1), (A:1), (B,C:1)}; a *conditional FP-tree* é {(A:4, B:2, C:2), (B:1, C:1)}; e os padrões frequentes são **{(A,D:4), (B,D:2), (C,D:2), (A,B,D:2), (A,C,D:2), (B,C,D:2), (A,B,C,D:2)}**.

O mesmo procedimento é repetido para todos os itens e todas as *conditional trees*, sempre em ordem decrescente de frequência.

7.3.7 Algoritmo Eclat

A observação registrada nos slides é a diferença de saída: **o algoritmo Eclat não fornece regras, mas sim a lista dos itens mais frequentemente comprados juntos**. É uma distinção prática relevante — se o objetivo é gerar recomendações no formato "se A então B", Eclat sozinho não basta.

Há scripts Python de apoio para Apriori e FP-Growth, e as bases `Groceries.csv`, `Market_Basket_Optimisation.c` e `Iris.csv`.

7.4 Encontro 4 — Agrupamento

7.4.1 Conceitos

Um **cluster** é uma coleção de objetos **similares aos objetos do mesmo cluster** e **dissimilares aos objetos de outros clusters**. **Clusterização** é o agrupamento de conjuntos de dados em clusters, e constitui uma **classificação não supervisionada**: sem classes predefinidas.

Um aviso importante do material: **a noção de cluster pode ser ambígua**. O mesmo conjunto de pontos admite mais de um agrupamento razoável, e a escolha depende do critério adotado.

Não se conhece o padrão nem o número total de grupos a serem encontrados. O conjunto de dados é particionado em grupos, baseados em características específicas, de modo que os pontos dentro de um cluster sejam mais similares do que os pontos de outros grupos.

O que é uma boa clusterização? Aquela que produz clusters com **alta similaridade dentro das classes** e **baixa similaridade entre as classes**. A qualidade dos resultados depende da **medida de similaridade** usada e do **método e sua implementação**.

7.4.2 Aplicações

- **Marketing**: identifica grupos distintos de clientes, útil para desenvolver programas de marketing (CHIANG, 2003).
- **Uso da terra**: identifica a possibilidade de alocação de uso da terra para fins agrários e/ou urbanos em base de dados de observação via satélite (LEVIA JR, 2000).
- **Seguro**: identifica grupos de clientes que fazem comunicação de sinistro com alta frequência (YEOH, 2001).
- **Planejamento urbano**: identifica grupos de casas de acordo com tipo, valor e localização geográfica.

7.4.3 Métodos

- **Particionamento**: constrói várias partições e as avalia usando algum critério.
- **Hierárquico**: cria uma decomposição hierárquica dos objetos usando algum critério.
- **Baseado em densidade**: fundamenta-se em funções de conectividade e de densidade.

Nos métodos de particionamento, dado um valor de `k`, busca-se encontrar `k` clusters que otimizem um critério escolhido. Os principais algoritmos são o **K-means** (MacQueen, 1967), em que cada cluster é representado pelo **centroide**, e o **K-medoids** ou **PAM** (*Partition Around Medoids*, Kaufman e Rousseeuw, 1987), em que cada cluster é representado por **um dos objetos** do cluster.

7.4.4 K-means passo a passo

O algoritmo é apresentado em cinco passos, com iteração:

1. **Escolher o número de clusters** (no exemplo, K = 2).
2. **Selecionar arbitrariamente K pontos como centroides iniciais** — e os slides destacam: não necessariamente pontos da base de dados.
3. **Associar cada objeto ao cluster (centroide) mais próximo**, ou seja, de maior similaridade, formando K clusters.
4. **Calcular e realocar o novo centroide de cada cluster** (por exemplo, a média para cada atributo).
5. **Associar cada objeto ao cluster mais próximo. Voltar ao passo 4 se algum objeto foi movido de cluster; terminar caso contrário.**

A condição de parada é, portanto, a **estabilidade das atribuições**: quando nenhuma observação muda de cluster, o modelo final foi alcançado. Os slides percorrem quatro ciclos completos de passos 4 e 5 até a convergência.

7.4.5 Como determinar o número de clusters — WCSS e Elbow Method

A métrica usada é o **WCSS** (*Within Cluster Sum of Squares*), a soma, sobre todos os clusters, das distâncias quadráticas de cada ponto ao centroide do seu cluster:

$wcss = \sum_j \sum_{P_i \in cluster_j} dist(P_i, C_j)^2$

O raciocínio é construído incrementalmente. Com um único cluster, o WCSS é muito grande, pois a distância de cada ponto ao centroide é grande. Com dois clusters, o WCSS diminui, já que as distâncias encolhem. Com três, diminui mais ainda. E aqui está a armadilha: **o número de clusters pode chegar ao número de registros da base**, caso em que cada ponto é um cluster, seu centroide coincide com ele e **WCSS = 0**. Obviamente essa não é uma boa abordagem — minimizar WCSS sem restrição é degenerado.

A solução é o **Elbow Method**. Plota-se o WCSS contra o número de clusters e procura-se o "cotovelo". Os valores tabelados nos slides:

K	WCSS
2	908.329
3	531.742
4	368.399
5	222.242
6	186.169
7	151.367
8	125.059
9	113.953
10	98.218

Usualmente, o número de clusters é definido pela **inclinação da reta**: escolhe-se o ponto em que a melhora obtida ao aumentar o número de clusters já não é tão significativa quando comparada à melhora imediatamente anterior. Na tabela, as quedas de K=2 para K=5 são acentuadas (de 908 para 222), enquanto a partir de K=6 os ganhos se achatam.

7.4.6 Variações do K-means

Algumas versões diferem em: **seleção dos pontos iniciais**; **cálculo da similaridade** entre pontos; e **estratégias para calcular os centroides**. Para atributos **nominais**, existe o **K-modes** (Huang, 1998), que substitui as médias dos clusters por **modas**, usa medidas de similaridade apropriadas a atributos nominais e usa um método baseado em frequências para atualizar as modas.

7.4.7 Clusterização hierárquica

Há dois sentidos de construção:

- **Métodos divisivos**: todos os registros formam um "grande cluster", que é dividido em dois ou mais clusters menores até que cada cluster tenha somente registros semelhantes (*top-down*).
- **Métodos aglomerativos**: cada registro é um cluster; a cada passo, combinam-se clusters com alguma característica comum, até chegar a um "grande cluster" (*bottom-up*).

Os slides percorrem onze iterações de um agrupamento aglomerativo, mostrando a formação progressiva das junções e a construção do **dendrograma**.

AGNES decompõe objetos em vários níveis de particionamento aninhados, formando uma árvore de clusters conhecida como **dendrograma**. Uma clusterização dos objetos é obtida **particionando-se o dendrograma em um nível desejado**: cada componente conectado forma um cluster. O

mesmo dendrograma, cortado em alturas diferentes, produz 2, 3 ou 4 clusters — e é isso que dá ao método hierárquico sua flexibilidade.

DIANA (*Divisive Analysis*) faz o procedimento inverso de AGNES; eventualmente cada nó forma um cluster.

7.4.8 K-means versus hierárquico

Modelo	Prós	Contras
K-means	Simples de entender; trabalha bem em bases pequenas e grandes; rápido e eficiente	É preciso passar como parâmetro o número de clusters
Hierárquico	O número ótimo de clusters pode ser obtido pelo próprio modelo; visualização prática através de dendrograma	Não é apropriado para bases muito grandes

O trade-off é claro: o hierárquico dispensa a escolha prévia de K, mas paga por isso em escalabilidade.

7.4.9 Estudo de caso: clientes de um shopping

O exemplo aplica K-means a clientes de um shopping, com dois eixos: **ganho anual** e **traço de gastos**. A interpretação de negócio dos cinco clusters encontrados é o produto final da análise:

- **Cluster 1** — ganho alto e baixo gasto: **clientes cuidadosos**;
- **Cluster 2** — ganho médio e médio gasto: **clientes padrão**;
- **Cluster 3** — ganho alto e alto gasto: **clientes alvo**, sobre os quais se deve entender melhor os produtos comprados e direcionar melhor as campanhas de marketing;
- **Cluster 4** — ganho baixo e baixo gasto: **clientes sensíveis**;
- **Cluster 5** — ganho baixo e alto gasto: **clientes pouco cuidadosos**.

Este é o passo que fecha o ciclo de um projeto de mineração: o algoritmo entrega grupos, mas é a leitura de negócio que os transforma em ação.

7.4.10 Clusterização baseada em densidade — DBSCAN

O **DBSCAN** define **densidade** como o número de pontos dentro de um raio específico (**Eps**). A partir disso, classifica cada ponto em três tipos:

- **core point**: tem um número mínimo de pontos especificado pelo usuário (**MinPts**) dentro do raio Eps;
- **border point**: fica localizado na vizinhança de um core point, mas não tem vizinhos suficientes para ser core;
- **noise point**: qualquer ponto que não se classifica como core point nem como border point.

A ideia geral: **um cluster é definido como um conjunto máximo de pontos densamente conectados**.

O algoritmo passo a passo:

1. Arbitrariamente, seleciona um ponto p.
2. Identifica todos os pontos densamente conectados a p com relação aos parâmetros Eps e MinPts.
3. Se p é um **core point**, um cluster é formado.
4. Se p é um **border point** e não há pontos densamente conectados a p, o DBSCAN visita o próximo ponto do conjunto de dados.
5. Continua o processo até que todos os pontos tenham sido analisados.

A vantagem estrutural do DBSCAN, discutida sob o título "Quando DBSCAN funciona bem?", é que ele não exige a definição prévia do número de clusters, encontra clusters de **forma arbitrária** (não apenas esféricos, como o K-means) e identifica explicitamente **ruído**, em vez de forçar todo ponto a pertencer a algum grupo.

8 Síntese da Disciplina

As Oficinas 2025.1 constituem, em conjunto, um percurso completo pelo ciclo de vida de um projeto de dados e inteligência artificial, do armazenamento à decisão. O que dá unidade às seis trilhas é a insistência no **fazer**: cada tópico chega ao aluno na forma de um enunciado a resolver, uma base a tratar, um parâmetro a ajustar.

O primeiro fio condutor é a **cadeia de ferramentas**. A trilha SAD estabelece a fundação em duas linguagens: SQL, para criar, restringir, alterar, popular e consultar dados relacionais; e Python, do comentário e da indentação até funções com argumentos default, passando por NumPy para vetorização e Matplotlib para visualização. Sobre essa fundação, o Power BI acrescenta a camada de entrega — dashboards com gráficos, filtros e cartões — e a trilha BI conecta essa camada ao data warehouse pela carga da tabela fato.

O segundo fio é a **modelagem**. A oficina de OAG mostra, com clareza rara, que a qualidade de um resultado de otimização depende menos do algoritmo e mais da modelagem: como representar (binário, real, lista), como decodificar, como avaliar — e como corrigir os patologias da avaliação bruta, que são o superindivíduo e a competição próxima, tratados por normalização e windowing. O problema das quatro rainhas, resolvido em três codificações diferentes, e a função F6, com seus 44 bits e sua superfície cheia de mínimos locais, são os dois exemplos que fixam a lição. O tratamento de restrições — descarte, penalização, reparo, decodificadores e a família Genocop — completa o repertório.

O terceiro fio é o **aprendizado a partir de dados**. A trilha DM percorre a sequência canônica: primeiro o pré-processamento, que consome a maior parte do esforço real de qualquer projeto (missing values, normalização, redução de dimensionalidade, balanceamento, outliers); depois os modelos supervisionados (KNN em cinco passos, regressão logística construída sobre a sigmoide, SVM pelo Teorema de Cover, árvores, comitês e Random Forest com seu erro OOB); depois os não supervisionados (Apriori, FP-Growth e Eclat para associação; K-means com Elbow Method, hierárquico com dendrograma e DBSCAN com core, border e noise points para agrupamento). A trilha RN chega ao mesmo território por outra porta, partindo do neurônio de McCulloch-Pitts, passando

pela limitação do perceptron de uma camada diante do XOR e chegando ao MLP treinado por backpropagation, para terminar em redes recorrentes e LSTM.

As conexões entre as trilhas são explícitas e vale destacá-las. O método **wrapper** de seleção de atributos, ensinado em Data Mining, é literalmente um algoritmo genético cujo cromossomo é um vetor binário de presença/ausência de atributos — a oficina de OAG é pré-requisito conceitual da oficina de DM. As **redes neurais** aparecem tanto na lista de classificadores de DM quanto como trilha própria. A **normalização** aparece em três contextos distintos: como técnica de pré-processamento em DM, como entrada normalizada em RN e como transformação de função de avaliação em OAG. E a base de **câncer de mama** da University of Wisconsin atravessa as trilhas DM e RN, permitindo comparar KNN, regressão logística e MLP sobre o mesmo problema.

Por fim, o material mais recente incorpora **IA generativa** ao repertório da oficina de BI, com exercícios de RAG e uma regra prática de engenharia de prompt que sintetiza bem o espírito de toda a disciplina: comece pelo mais simples — zero-shot CoT — e só adote técnicas mais caras, como Self-Consistency e Tree of Thoughts, quando o problema realmente exigir. É a mesma disciplina de custo e benefício que aparece na escolha entre representação binária e real em GAs, entre K-means e hierárquico em agrupamento, e entre acurácia e métricas robustas em bases desbalanceadas.