

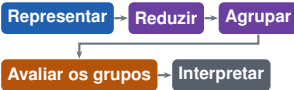
Machine Learning: folha de consulta

As 27 aulas em oito páginas. Páginas 1 a 3: o fluxo supervisionado, etapa por etapa. Página 4: sem rótulos e texto. Páginas 5 e 6: os modelos, um a um. Páginas 7 e 8: aula a aula.

Com rótulo (supervisionado)



Sem rótulo e texto (página 4)



O teste é separado no começo e aberto uma única vez, no fim.

🎯 Formular o problema

F1 · F2

- ▶ **Precisa de ML?** Sim quando as regras são desconhecidas, mudam, são numerosas demais ou pedem personalização. Regra fixa e curta (“o cartão tem 16 dígitos”): programe.
- ▶ **Pelo feedback:** supervisionado (x e y): y contínuo → **regressão**; categoria → **classificação**. Não supervisionado (só x): agrupar, reduzir, tópicos.
- ▶ **Unidade** (o que é uma linha), **target** y , **features** que existem **no momento da previsão**, métrica e meta escritas antes de treinar.
- ▶ Importa o erro em dados que o modelo **nunca viu**. Nenhum algoritmo vence sempre (*no free lunch*): compare famílias.

Baseline: a régua

regressão	DummyRegressor(): a média (a constante de menor erro quadrático); strategy='median': a mediana (de menor erro absoluto)
classificação	DummyClassifier(strategy='most_frequent'). Com 1% de positivos, acerta 99%.

🔪 Dividir

F2 · D1 · C5

treino	aprende (inclusive médias, medianas, categorias)
validação	escolhe (modelo, hiperparâmetros, corte)
teste	mede uma vez, no fim, o que já foi escolhido

```
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

- ▶ **Estratificada** (stratify=y) quando há classe rara; **temporal** quando se prevê o futuro; **por grupo** quando o mesmo paciente (cliente, loja) tem várias linhas.

🔍 Explorar (EDA): só no treino

D1

EDA: análise exploratória de dados. O produto é uma lista escrita de fatos, problemas e perguntas. Tudo o que é *estimado* vem só do treino.

- ▶ **“Uma linha não é uma casa”**: no California Housing a linha é um distrito; divida as contagens (cômodos por domicílio).
- ▶ **Sempre faça gráficos** (Anscombe: mesmas médias, r e reta). Histograma com 30 a 50 bins: o teto (500.001 em 965 distritos) só aparece assim.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2} \sqrt{\sum (y_i - \bar{y})^2}} \in [-1, 1] \quad \text{IQR} = Q_3 - Q_1$$

$\bar{x}, \bar{y}$: médias; Q_1, Q_3 : quartis; IQR: intervalo interquartil. Spearman: Pearson sobre os postos. $y = 1, 2, 3, 4, 20$ contra $x = 1.5$: $r = 0,8$, $\rho = 1$. $r \approx 0$: nenhuma associação *linear*, nunca “nenhuma relação”. Mediana e IQR resistem a extremos.

- ▶ **Simpson**: um **confundidor** pode inverter a correlação agregada. Ela é hipótese, não achado.
- ▶ **Faltantes**: conte e pergunte por quê. MCAR (ao acaso), MAR (depende de outra coluna), MNAR (do próprio valor). Descartar linhas só é inofensivo sob MCAR.
- ▶ **Outlier**: fora de $[Q_1 - 1,5 \text{ IQR}, Q_3 + 1,5 \text{ IQR}]$ é candidato; erro ou raro-mas-real, decida com o domínio.

⚠ **Armadilha**: explorar a base inteira antes de separar o teste (*data snooping*).

🔧 Preparar: fit só no treino

D2

Standard	$\frac{x - \mu}{\sigma}$	MinMax	$\frac{x - x_{\min}}{x_{\max} - x_{\min}}$	Robust	$\frac{x - \text{med.}}{\text{IQR}}$
----------	--------------------------	--------	--	--------	--------------------------------------

μ (média), σ (desvio), mín., máx., mediana e IQR vêm **do treino**. Dependem de escala: k-NN (k vizinhos mais próximos), k-means, SVM (máquina de vetores de suporte), PCA (componentes principais), modelos por gradiente, Ridge, Lasso, redes. Árvores não.

- ▶ Por coluna: **faltantes** → **forma** → **escala**. Forma: FunctionTransformer(np.log1p), PowerTransformer. Escalar não muda a forma.
- ▶ **Nominal** → OneHotEncoder(handle_unknown='ignore') (categoria nova vira zeros). **Ordinal** → OrdinalEncoder(categories=[...]) com a ordem declarada (sem ela, vale a alfabética).
- ▶ **Alta cardinalidade**: min_frequency junta as raras; TargetEncoder faz cross-fitting (cada fold com as médias dos outros).
- ▶ SimpleImputer(strategy='median', add_indicator=True); a média imputada encolhe a variância. Nunca impute o alvo.

⚠ **Armadilha**: fit_transform no teste ou em produção. O fit roda uma vez, no treino; o resto só vê transform.

🔗 Pipeline: um objeto só

D3

```
prep = ColumnTransformer([
    ('num', make_pipeline(SimpleImputer(strategy='median'),
                          StandardScaler()), num),
    ('cat', OneHotEncoder(handle_unknown='ignore'), cat)])
pipe = make_pipeline(prep,
                     KNeighborsRegressor(n_neighbors=10))
pipe.fit(X_train, y_train) # cada passo aprende no treino
pipe.predict(X_test)      # cada passo só reaplica
```

- ▶ fit do pipeline: fit_transform em cada passo e fit no último; predict: só transform, depois predict.
- ▶ Três bugs do fluxo manual, sem erro nenhum (MAE, erro absoluto médio, certo: US\$ 43.023): fit_transform um distrito por vez, US\$ 116.049; escalonador esquecido, US\$ 105.978; colunas trocadas, US\$ 101.517.
- ▶ DataFrame até o modelo, listas explícitas de colunas, remainder='drop'. Botão de um passo: passo__parametro. Salve o pipeline **inteiro**.

Vazamento (data leakage): os cinco padrões

1. **Passo ajustado antes da divisão**: SelectKBest antes da validação cruzada (CV), 89% contra 53% honestos; TargetEncoder fora do pipeline, MAE US\$ 14.567 contra US\$ 42.552.
2. **Atributo que conhece o futuro** (a duração da ligação, só sabida depois dela).
3. **Linhas duplicadas** nos dois lados (0,902 contra 0,515).
4. **Grupo**: o mesmo paciente no treino e na validação (KFold 93,8%, GroupKFold 78,0%).
5. **Tempo**: KFold dá $R^2 = 0,99$; TimeSeriesSplit, $-1,78$.

Antes de acreditar num número: é bom demais? Cada valor está disponível no momento da previsão? Todo passo que aprende está no pipeline? A divisão imita o uso? O teste escolheu alguma coisa?

Teste do cheiro: embaralhe o y logo no início e rode tudo de novo, pré-processamento incluído: um pipeline honesto cai ao acaso. Passar não prova nada sobre duplicatas, grupos, tempo e colunas do futuro. Quanto mais um passo usa o alvo (target encoding, seleção, sobreamostragem), maior o vazamento.

 Treinar: os modelos e os seus botões

M1 · página 5

modelo	botão: fica mais simples quando...	escala?
Ridge, Lasso	alpha sobe	sim
logística	C desce	sim
k-NN	n_neighbors sobe	sim
SVM (RBF)	C e gamma descem	sim
árvore	max_depth desce , min_samples_leaf sobe	não
floresta	max_features desce , min_samples_leaf sobe	não
boosting	learning_rate desce , menos árvores, árvores rasas	não
rede (MLP)	alpha sobe , camadas mais estreitas	sim

Em negrito, os botões que "giram ao contrário": o valor maior é o modelo mais simples. RBF: kernel gaussiano; MLP: perceptron multicamadas.

$$\mathbb{E}[(y - \hat{f}(x))^2] = \underbrace{(f(x) - \mathbb{E}[\hat{f}(x)])^2}_{\text{viés}^2} + \underbrace{\text{Var}(\hat{f}(x))}_{\text{variância}} + \underbrace{\sigma^2}_{\text{ruído}}$$

$y = f(x) + \varepsilon$, com ruído de variância σ^2 ; $\hat{f}$ muda com o conjunto de treino. Todo botão desenha o mesmo U no erro de validação.

	underfitting (viés)	overfitting (variância)
erro de treino	alto	baixo
erro de validação	alto, estável entre folds	alto e instável entre folds
o que fazer	mais capacidade, atributos melhores	regularizar, mais dados, menos capacidade

- **Curva de validação:** erro $\times$ hiperparâmetro (validation_curve).
- Curva de aprendizado:** erro $\times$ linhas de treino (learning_curve); curvas juntas e altas: mais dados não ajudam.

 Regressão: mínimos quadrados e gradiente

R1 · R2

$$\hat{y} = w_0 + \sum_j w_j x_j = Xw \quad J(w) = \|y - Xw\|^2$$
$$\nabla_w J = -2X^\top (y - Xw) = 0 \Rightarrow \hat{w} = (X^\top X)^{-1} X^\top y$$

X : n linhas, uma coluna de 1s e d atributos; w_j : quanto $\hat{y}$ muda quando x_j sobe 1, **com os outros fixos**. Na simples, $\hat{w}_1 = r_{xy} / s_x$.

Gradiente descendente $w \leftarrow w - \eta \nabla_w J(w)$
 η : taxa de aprendizado. Pequena: lenta; grande: oscila ou diverge. Batch usa todas as linhas; SGD (estocástico), uma; mini-batch, 32 a 512.

Ridge (L2) $J + \alpha \sum_j w_j^2$ **Lasso (L1)** $J + \alpha \sum_j |w_j|$
Ridge encolhe sem zerar e cura a multicolinearidade; Lasso zera (seleciona) e, entre colunas correlacionadas, fica com uma arbitrariamente. ElasticNet(alpha, l1_ratio) mistura as duas. $\alpha \rightarrow \infty$: prevê a média.

 **Armadilha:** regularizar sem escalonar (a penalidade compara coeficientes em km e em m). StandardScaler no pipeline; o intercepto não é penalizado.

 Avaliar uma regressão

R1 · M1

$$\text{MAE} = \frac{1}{n} \sum |e_i| \quad \text{RMSE} = \sqrt{\frac{1}{n} \sum e_i^2} \geq \text{MAE}$$
$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}} = 1 - \frac{\sum e_i^2}{\sum (y_i - \bar{y})^2}$$

$e_i = y_i - \hat{y}_i$ (resíduo); RMSE: raiz do erro quadrático médio; SSE, SST: somas dos quadrados de e_i e de $y_i - \bar{y}$. Erros 2, 2, 2, 2: MAE = RMSE = 2; erros 0, 0, 0, 10: MAE 2, RMSE 4,47. $R^2 < 0$ no teste: pior que prever a média. Na regressão simples, $R^2 = r^2$.

nos resíduos	significa $\rightarrow$ o que fazer
faixa sem forma	a forma linear serve
curva (U ou $\cap$)	não linearidade $\rightarrow$ termo polinomial, transformar x
funil	variância muda com $\hat{y} \rightarrow$ transformar y (log)
ponto isolado	outlier ou alavancagem $\rightarrow$ investigar, nunca apagar em silêncio

 **Armadilha:** ler o R^2 do treino: ele nunca cai quando se acrescenta uma coluna (até de ruído). Métricas só em dados separados, ao lado do DummyRegressor.

 Avaliar uma classificação

C3

	prev. +	prev. -
real +	TP	FN
real -	FP	TN

$$\text{acurácia} = (TP + TN) / n$$
$$\text{precisão} = TP / (TP + FP)$$
$$\text{revocação (recall)} = TP / (TP + FN)$$
$$\text{especificidade} = TN / (TN + FP)$$
$$F_1 = 2TP / (2TP + FP + FN)$$

TP, FP, FN, TN: verdadeiros e falsos positivos e negativos. FP é o alarme falso; FN, o escape.

```
cm = confusion_matrix(y, y_prev, labels=[1, 0])
tp, fn, fp, tn = cm.ravel() # positivo primeiro
```

$$F_\beta = \frac{(1 + \beta^2) TP}{(1 + \beta^2) TP + \beta^2 FN + FP} \quad \text{um FN pesa } \beta^2 \text{ vezes um FP}$$
$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

MCC (coeficiente de Matthews): de -1 a 1 ; 0 é acaso. Acurácia balanceada = (revocação + especificidade)/2. Multiclasse: macro (média simples, dá voz à classe rara), ponderada, micro (= acurácia).

- **Prevalência** baixa: com 1% de doentes, “sempre saudável” tem 0,990 de acurácia e revocação 0. A precisão cai com a prevalência (0,778 com 50%, 0,034 com 1%, mesmo modelo).
- **Log loss e Brier** $= \frac{1}{n} \sum (p_i - y_i)^2$ avaliam a probabilidade, sem limiar.
- A métrica pela pergunta: rastrear doença, revocação ou F_2 ; spam, precisão ou $F_{0,5}$; probabilidade como número, log loss e calibração.

 **Armadilha:** sem labels=[1, 0], TN vai para o canto de cima.

 Todos os limiares: ROC, PR e o corte

C4

$$\text{TPR} = \frac{TP}{TP + FN} \quad \text{FPR} = \frac{FP}{FP + TN}$$

Taxas de verdadeiros (TPR) e de falsos positivos (FPR). Curva ROC (receiver operating characteristic): TPR $\times$ FPR, um ponto por limiar. AUC = P (escore de um positivo $>$ escore de um negativo)
AUC: área sob a curva; 0,5 é o acaso. Só enxerga a ordem: escore² dá a mesma AUC.

$$\text{AP} = \frac{\sum_k (R_k - R_{k-1}) P_k}{\pi \text{TPR}}$$

precisão = $\frac{\pi \text{TPR}}{\pi \text{TPR} + (1 - \pi) \text{FPR}}$
Curva PR (precisão–revocação). AP (average precision): P_k e R_k no limiar k ; o acaso dá a prevalência π . Com π de 50% a 1%, a AUC fica perto de 0,92 e a AP cai de 0,92 para 0,25.

- O modelo dá um escore; **nós** escolhemos o corte t : positivo se $\hat{p} \geq t$. Baixar t sobe a revocação e os FP; AUC e AP não mudam. “Nada no 0,5 é especial.”
- Escolha **t na validação**, por um critério declarado: meta de revocação, custo, maior F_1 , J de Youden (TPR $-$ FPR).

Com escores calibrados: $t^* = \frac{c_{FP}}{c_{FP} + c_{FN}}$ FN custa 10 FP:
 $t^* = 1/11 \approx 0,09$.

```
prec, rec, thr = precision_recall_curve(y_val, p_val)
ok = rec[:-1] >= 0.80 # a meta de revocação
i = np.argmax(np.where(ok, prec[:-1], -1))
t = thr[i] # o teste não entra aqui
```

 **Armadilha:** roc_auc_score(y, y_prev) com rótulos não dá erro: devolve a acurácia balanceada. Passe escores (predict_proba ou decision_function).

Classe rara: a ordem de ataque

1. Medir a coisa certa (AP, revocação) e estratificar.
2. Mover o limiar: de graça, sem retrainar.
3. class_weight='balanced': $w_c = n / (k n_c)$ (k classes, n_c linhas da classe c). Na logística, é quase um limiar deslocado.
4. Reamostrar (SMOTE, sobreamostragem sintética: $x_{\text{novo}} = x_i + \lambda(x_{\text{viz}} - x_i)$, $\lambda \in [0, 1]$) só no treino, dentro de um pipeline do imblearn.
5. Mais exemplos da minoria: o único remédio que melhora a própria ordenação.

```
from imblearn.pipeline import make_pipeline
m = make_pipeline(StandardScaler(),
                  SMOTE(random_state=0), LogisticRegression())
```

Pesos e reamostragem distorcem as probabilidades: a média prevista vai de 0,02 para 0,24 com taxa real de 0,02.

Validar e comparar com honestidade

C5 · M1

- Um split é uma loteria (93,0% a 100,0% em 300 sementes). **Validação cruzada** (CV): k folds, cada um validado uma vez; relate **média ± desvio**. Ela avalia uma **receita**, não um modelo.
- Um **único objeto** `cv`, com semente fixa, para todos os candidatos (o Dummy também): as notas vêm em pares.

```
cv = StratifiedKFold(n_splits=5, shuffle=True,
                    random_state=42)
s = cross_val_score(pipe, X_train, y_train, cv=cv,
                    scoring='f1')
print(f"{s.mean():.3f} ± {s.std():.3f}")
```

KFold	regressão; <code>shuffle=True</code> se o arquivo vem ordenado
StratifiedKFold	classificação: mesma proporção em cada fold
GroupKFold	linhas do mesmo paciente (cliente, loja) juntas
TimeSeriesSplit	treina no passado, valida no futuro; <code>gap</code>

Empate pareado: se a diferença média, fold a fold, é menor que o desvio das diferenças, é empate.

Regra de 1 EP: entre as configurações a menos de um erro-padrão (EP = desvio entre folds / $\sqrt{k}$) da melhor, fique com a mais simples.

- Maldição do vencedor:** em puro ruído, a melhor de 100 árvores tira 61,2% na validação e 50,9% no teste. O `best_score_` é otimista: relate o teste trancado ou a **CV aninhada** (o laço interno escolhe, o externo mede).

```
outer = StratifiedKFold(n_splits=3, shuffle=True,
                      random_state=7)
nested = cross_val_score(busca, X_dev, y_dev, cv=outer)
```

Buscar hiperparâmetros

- GridSearchCV: a grade toda; a família do modelo pode ser um hiperparâmetro (uma grade por família). Vencedor na borda: a grade era estreita.
- RandomizedSearchCV (`n_iter=...`) com `loguniform`: o baseline honesto. O que age multiplicando (`C`, `alpha`, taxas, folhas, vizinhos) vai em **escala log**.
- Optuna (TPE, estimador de Parzen em árvore): aprende onde estão as boas tentativas; `suggest_float(..., log=True)`, poda com `MedianPruner`. `HalvingRandomSearchCV`: torneio que mantém $1/\eta$ e dá η vezes mais recurso.

Armadilha: a busca otimiza o vazamento que você deu. Nenhum método de busca encontra uma coluna que conhece o futuro.

Explicar

E2

método	responde a...
MDI	“o que o treino usou?” Queda média de impureza (<code>feature_importances_</code>); premia colunas com muitos valores, até ruído
permutação	“de que a nota precisa?” Embaralha uma coluna em dados separados
PDP, ICE	“como a previsão muda com x_j ?” Dependência parcial (média) e uma curva por linha
SHAP	“por que esta previsão?” Valores de Shapley que somam a previsão
LIME	explicação local: reta em vizinhos sorteados; muda com a largura do kernel

$$PD_j(v) = \frac{1}{n} \sum_i \hat{f}(x_{i,1}, \dots, v, \dots, x_{i,d}) \quad x_j \text{ fixado em } v \text{ em todas as linhas}$$

$$\hat{f}(x) = \phi_0 + \phi_1 + \dots + \phi_d \quad \phi_0: \text{valor base (previsão média no fundo); } \phi_j: \text{parcela de } x_j$$

```
pi = permutation_importance(modelo, X_test, y_test,
                          scoring='r2', n_repeats=5, random_state=0)
```

- Colunas correlacionadas **dividem o crédito**: embaralhe o grupo junto. Um atributo domina? Pergunte quando o valor é registrado.

Armadilha: uma explicação descreve **o modelo**, não o mundo: importância não é causa.

Produção: MLOps (operações de ML)

E4

- O artefato é o **pipeline inteiro** (`joblib.dump`); um pickle é um programa: só de fonte confiável (ou `skops`). Fixe as versões.
- Um **schema** do treino (faixas, categorias) confere cada requisição: o pipeline aceita lixo (idade —3 ganha uma probabilidade).
- Versione código, dados (hash), ambiente, sementes; registre cada experimento.

covariate shift	muda $P(x)$; os monitores de entrada veem
concept drift	muda $P(y x)$; só os rótulos revelam
prior shift	muda $P(y)$; a precisão cai

$$PSI = \sum_{i=1}^B (a_i - e_i) \ln \frac{a_i}{e_i} \quad \text{índice de estabilidade populacional; } e_i,$$

a_i : fração da referência e dos dados atuais na faixa i de B . Abaixo de 0,1: estável; acima de 0,25: mudança grande. Com muitas linhas, o p-valor do teste KS (Kolmogorov–Smirnov) sempre acusa: alerte por magnitude.

- Drift não é dano:** o alerta manda olhar e talvez retrainar. Modelo novo só entra se vencer o atual em dados recentes: shadow, canary, teste A/B, rollback pronto.

Armadilha: desalinhamento treino—serving: a duração em anos, e não em meses, muda 17% das decisões sem erro nenhum.

Diagnóstico rápido

todas

sintoma	provável causa → o que fazer
nota boa demais	vazamento → os cinco padrões; embaralhar o y
treino $\gg$ validação	overfitting → regularizar, simplificar, mais dados
treino $\approx$ validação, os dois ruins	underfitting → mais capacidade, atributos melhores
acurácia alta, classe rara	é o baseline → revocação, precisão, AP
resíduos em curva ou funil	falta forma → termo novo, transformar
a nota muda com a semente	variância da divisão → CV com média ± desvio
diferença < desvio das diferenças	empate → o mais simples
coluna de ruído “importante”	viés da MDI → permutação em dados separados
k-NN ou SVM ruins	falta escala → <code>StandardScaler</code> no pipeline
<code>max_iter</code> avisa	atributos sem escala
boosting piora com mais árvores	η grande → η menor e parada antecipada
a acurácia cai em produção	drift → PSI nas entradas; métricas quando os rótulos chegam

Do problema à produção, etapa por etapa

todas

- Problema:** unidade, target, features do momento da previsão, métrica e baseline escritos antes.
- Dados:** teste separado antes de olhar; divisão que imita o uso; EDA escrita.
- Preparo:** tudo num pipeline; os cinco padrões de vazamento conferidos.
- Modelos:** baseline e candidatos no mesmo `cv`; botões por busca.
- Comparação:** média ± desvio; fold a fold; num empate, o mais simples.
- Corte:** limiar por critério declarado, na validação.
- Explicação:** permutação em dados separados; nada de causa.
- Entrega:** teste aberto uma vez; pipeline inteiro salvo com versões; limitações escritas; monitoramento.

Sem rótulos e texto, etapa por etapa

Não há gabarito: o algoritmo sempre devolve uma resposta, haja ou não estrutura. Os critérios são seus; escreva-os.

Reduzir a dimensionalidade

U1

o motivo o método
ver os dados em 2D t-SNE, UMAP
comprimir, tirar ruído PCA
 ou colinearidade
alimentar outro modelo PCA (precisa de transform)

- **Maldição da dimensionalidade:** colunas sem informação esmagam o contraste das distâncias, $(d_{\max} - d_{\min})/d_{\min}$, de 42,97 com 2 colunas para 0,11 com 1000. Pergunte quantas direções de fato variam.

PCA (análise de componentes principais)

$$C = \frac{1}{n-1} X^T X \quad w_1 = \arg \max_{\|w\|=1} w^T C w$$

$$\text{EVR}_k = \frac{\lambda_k}{\sum_j \lambda_j} \quad \text{erro de reconstrução} = \sum_{j>k} \lambda_j$$

X centrada ($n \times d$); C : covariância; w_1 : o autovetor de maior autovalor λ_1 ; EVR: fração da variância explicada. Variância máxima e reconstrução mínima são a mesma conta. Autovalores 8 e 2: o PC1 guarda 80%.

```
pipe = make_pipeline(StandardScaler(),
                     PCA(n_components=0.95), modelo)
```

- **Padronize** colunas de unidades diferentes: sem escala, o PC1 do wine é a prolina (99,8%) e o k-NN sobre 2 componentes cai de 0,926 para 0,722. Pixels (mesma unidade) não pedem.
- **Quantos componentes:** 90/95/99% da variância ($n_{\text{components}}=0.95$); se o PCA alimenta um modelo, o desempenho do modelo decide. O PCA nunca olha o alvo.
- **t-SNE** (*t-distributed stochastic neighbor embedding*; perplexity: vizinhos efetivos, 5 a 50) e **UMAP** (*uniform manifold approximation and projection*; $n_{\text{neighbors}}=15$, $\text{min_dist}=0.1$) guardam vizinhanças. O UMAP tem transform; o t-SNE não. Com centenas de colunas, PCA antes.
- **Trustworthiness:** fração dos vizinhos no mapa que eram vizinhos de verdade (1 é perfeito; mapa aleatório, cerca de 0,5).

Num mapa t-SNE ou UMAP, nunca diga: o tamanho do grupo é a sua variância; grupos próximos são parecidos; há 5 grupos nos dados; o eixo significa algo. Nunca rode uma vez só, use as 2 coordenadas como atributos nem ajuste antes de dividir. Um gráfico é uma hipótese; informe a perplexidade.

Agrupar e avaliar os grupos

U2 · U3

Antes do algoritmo, duas decisões: o que “parecido” quer dizer (escala, distância) e como o resultado será julgado.

$$\textbf{k-means} \quad J = \sum_{j=1}^k \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

μ_j : média do grupo C_j . Algoritmo de Lloyd: cada ponto vai ao centro mais próximo, cada centro vai à média dos seus pontos, até nada mudar. J (inércia) só cai com mais grupos: não escolha k .

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \in [-1, 1]$$

$a(i)$: distância média aos pontos do próprio grupo; $b(i)$: ao grupo vizinho mais próximo. Silhueta: média dos $s(i)$. Calinski–Harabasz: maior é melhor; Davies–Bouldin: menor. ARI (índice de Rand ajustado) compara com rótulos escondidos: serve para *conferir*, não para escolher; 0 é acaso.

- **KMeans**($n_{\text{clusters}}=k$, $n_{\text{init}}=10$, $\text{random_state}=0$): k-means++ e vários reinícios evitam mínimos locais. Supõe grupos redondos, de tamanho parecido, sem ruído.
- **Escala** se decide pelas unidades, não por um índice: nos pinguins a silhueta prefere os dados brutos (0,580 contra 0,447) e o ARI, os padronizados (0,793 contra 0,331).
- **DBSCAN**(eps , min_samples): ponto central tem min_samples vizinhos (ele incluso) a até eps ; o resto é borda ou ruído (-1). eps no joelho do gráfico de k -distâncias; um eps é uma densidade só.
- **HDBSCAN**($\text{min_cluster_size}=15$): todos os eps , cada grupo na sua densidade; nas luas e círculos, ARI 1,000 onde o k-means falha.

Quatro perguntas antes de acreditar: padronizei? reiniciei o suficiente? de onde veio o k ? comparei com os dados embaralhados? Ruído uniforme dá silhueta 0,40. Um agrupamento é uma *escolha*, não uma descoberta.

Hierárquico: a família inteira de cortes

ligação	distância entre P e Q	tende a
simples	$\min d(p, q)$	cadeias (formas finas)
completa	$\max d(p, q)$	grupos compactos
média	média das distâncias	meio-termo
Ward	aumento da inércia	redondos, como o k-means

- Ward: $\Delta = \frac{|P||Q|}{|P|+|Q|} \|\mu_P - \mu_Q\|^2$, só com distância euclidiana.
- No dendrograma leia a **altura** da fusão, nunca a ordem das folhas; corte onde a faixa de alturas é mais larga (`fcluster(Z, k, criterion='maxclust')`).
- Correlação **cofenética**: quão fiel a árvore é às distâncias, não quão bons são os grupos. Custo: $n(n-1)/2$ distâncias (100.000 linhas, 40 GB).

A Representar texto

U4

Texto → tokens → vetor → comparar. Esperso (uma coluna por palavra) ou denso (centenas de números, de um modelo treinado).

BoW (bag-of-words): x_{ij} = vezes que a palavra j aparece em i

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \left(\ln \frac{1+n}{1+\text{df}(t)} + 1 \right)$$

TF-IDF: frequência do termo $\times$ inverso da frequência nos documentos. tf: contagem de t em d ; n : documentos; df: documentos com t . Depois, cada linha é dividida pelo seu comprimento (`norm='l2'`). `sublinear_tf=True`: $\text{tf} \rightarrow 1 + \ln \text{tf}$.

$$\cos(u, v) = \frac{u \cdot v}{\|u\| \|v\|} \quad \text{o padrão para texto: o texto colado duas vezes tem cosseno 1 com o original}$$

- **Embeddings:** contextos parecidos, vetores parecidos (hipótese distribucional). Frases: `encode(textos, normalize_embeddings=True)`; norma 1 faz cosseno = produto escalar.
- TF-IDF primeiro (barato; jargão e muitos rótulos); embeddings para paráfrase, textos curtos, poucos rótulos (5 por classe: 0,557 contra 0,819). O vetorizador é uma etapa ajustada: fit só no treino.

🦊 **Armadilha:** `stop_words='english'` apaga *not*, *no* e *never*; negação e ordem enganam até embeddings.

Tópicos: do que tratam os documentos?

U5

Um tópico é uma **distribuição sobre palavras** e o documento, uma mistura (LDA, NMF); ou um **grupo de documentos** (BERTopic).

LDA (alocação latente de Dirichlet) $p(w | d) = \sum_k \theta_{d,k} \varphi_{k,w}$

NMF (fatoração não negativa) $X \approx WH$, $W, H \geq 0$

$\theta_{d,k}$: fatia do tópico k no documento d ; $\varphi_{k,w}$: probabilidade da palavra w no tópico k . NMF: X TF-IDF; linha de H é tópico, linha de W é o documento.

$$\text{c-TF-IDF}(t, c) = \frac{n_{t,c}}{\sum_{t'} n_{t',c}} \times \log \left(1 + \frac{A}{f_t} \right)$$

TF-IDF por classe. $n_{t,c}$: contagem de t no grupo c ; f_t : em todos os grupos; A : média de palavras por grupo.

- **BERTopic:** embeddings → UMAP em 5 dimensões → HDBSCAN (outliers no tópico -1) → c-TF-IDF dá as palavras de cada grupo.
- **Quantos tópicos:** a perplexidade da LDA só piora com k ; a coerência (NPMI, informação mútua pontual normalizada) é pista. Leia os tópicos em 2 ou 3 valores de k .

🦊 **Armadilha:** sem stop words, “the, of, to” e assinaturas de e-mail viram tópicos.

Os modelos supervisionados, um a um

Como cada um prevê, o que aprende no fit, os botões (com o nome no scikit-learn), a escala, quando brilha e onde engana.

Baselines Dummy · R1 · C3	
Prevê	uma constante: a média (<code>DummyRegressor()</code>) ou a mediana (<code>strategy='median'</code>); a classe mais frequente (<code>'most_frequent'</code>) ou a proporção (<code>'prior'</code>).
Lê-se	acurácia = $1 - \text{prevalência}$; ROC AUC 0,5; AP = prevalência; $R^2 \leq 0$.
Para quê	é a régua: todo modelo é medido contra ela, no mesmo cv.

Regressão linear LinearRegression · R1 · R2	
Prevê	$\hat{y} = w_0 + w_1x_1 + \dots + w_dx_d$.
Aprende	<code>coef_</code> e <code>intercept_</code> por mínimos quadrados (forma fechada) ou por gradiente (<code>SGDRegressor</code>).
Botão	nenhum; os termos que entram (<code>PolynomialFeatures(degree=3)</code>).
Escala	a forma fechada não liga; o gradiente liga muito; padronize para comparar coeficientes.
Lê-se	w_j : quanto $\hat{y}$ muda por unidade de x_j , com os outros fixos. Coeficiente não é importância.
Engana	multicolinearidade (coeficientes trocam de sinal, a soma fica); outliers e alavancagem; não segue curvas.

Ridge, Lasso, Elastic Net Ridge · Lasso · ElasticNet · R2	
Custo	$\ y - Xw\ ^2 + \alpha \sum w_j^2$ (L2) ou $\alpha \sum w_j $ (L1).
Botão	<code>alpha</code> : maior, mais simples (no limite, a média); <code>l1_ratio</code> na <code>ElasticNet</code> . Escolha com <code>RidgeCV</code> , <code>LassoCV</code> ou busca.
Escala	obrigatória; o intercepto não é penalizado.
Diferença	Ridge encolhe e divide o peso entre colineares; Lasso zera (seleciona) e fica com um colinear ao acaso.

Regressão logística LogisticRegression · C1	
Prevê	$z = w^T x + b$, $\hat{p} = 1/(1 + e^{-z})$; <code>predict</code> é $\hat{p} \geq 0,5$, ou seja, $z \geq 0$.
Aprende	os pesos, minimizando a log loss (convexa): $-\frac{1}{n} \sum [y \ln \hat{p} + (1 - y) \ln(1 - \hat{p})]$.
Botões	C (penalidade L2 de fábrica; menor, mais forte; <code>C=np.inf</code> desliga); <code>max_iter</code> . No 1.9, <code>penalty</code> está depreciado: use <code>l1_ratio</code> .
Escala	<code>StandardScaler</code> no pipeline; o aviso de <code>max_iter</code> costuma ser falta dela.
Fronteira	uma reta (hiperplano); curvas só com atributos novos (x_1^2 , polinômios).
Lê-se	e^{w_j} multiplica a chance $p/(1 - p)$, não a probabilidade (mulher no Titanic: 14,66 vezes a chance).

k-NN KNeighborsClassifier · C2	
Prevê	voto dos k vizinhos mais próximos; $\hat{p}$ = fração deles na classe.
Aprende	nada: guarda o treino (aprendiz preguiçoso).
Botões	<code>n_neighbors</code> ($k = 1$ decora; k enorme vira a classe majoritária; ímpar com 2 classes); <code>weights='distance'</code> ; <code>metric/p</code> .
Escala	obrigatória: wine 0,722 sem escala, 0,963 com.
Brilha	fronteira curva com poucas features que importam (círculos: 0,993 contra 0,493 da logística).
Engana	features irrelevantes (maldição da dimensionalidade); previsão cara, $O(nd)$; $\hat{p}$ com só $k + 1$ valores.

SVM SVC · LinearSVC · C6	
Prevê	sinal de $f(x) = \sum_{i \in \text{VS}} \alpha_i y_i K(x_i, x) + b$, somando nos vetores de suporte (VS), com K o kernel, $\alpha_i \geq 0$ o peso de cada um e $y \in \{-1, +1\}$.
Aprende	a rua mais larga ($2/\ w\ $) com perda hinge $\max(0, 1 - yf(x))$; só os VS contam.
Botões	C (menor: rua larga, muitos VS); <code>kernel='rbf'</code> com <code>gamma</code> (maior: cada VS vira uma ilha); <code>degree</code> e <code>coef0</code> no polinomial. Busque C e <code>gamma</code> juntos, em escala log.
Escala	obrigatória, dentro do pipeline; <code>gamma='scale'</code> não resolve.
Brilha	n pequeno ou médio, fronteira não linear, muitos atributos.
Engana	treino de n^2 a n^3 ; <code>decision_function</code> não é probabilidade (<code>probability=True</code> está depreciado no 1.9: use <code>CalibratedClassifierCV</code>).

Árvore de decisão DecisionTreeClassifier · M2	
Prevê	segue cortes "coluna $\leq$ limiar" até uma folha; $\hat{p}$ = proporção da folha.
Aprende	cortes gulosos: o que mais reduz a impureza (Gini $1 - \sum_c p_c^2$ ou entropia).
Botões	<code>max_depth</code> , <code>min_samples_leaf</code> , <code>max_leaf_nodes</code> , <code>ccp_alpha</code> (poda). Num platô, a menor.
Escala	não precisa (padronizar dá as mesmas previsões).
Brilha	regras legíveis; interações sozinha.
Engana	sem limite, treino 1,000 até com rótulos aleatórios; instável; não extrapola.

Random forest RandomForestClassifier · M3	
Prevê	o voto (ou a média) de muitas árvores profundas.
Aprende	cada árvore numa amostra bootstrap (deixa de fora $1/e \approx 37\%$), sorteando <code>max_features</code> colunas por corte.
Botões	<code>max_features</code> (o botão próprio), <code>min_samples_leaf</code> ; <code>n_estimators</code> é orçamento: mais nunca piora. <code>oob_score=True</code> : validação de graça nas linhas que cada árvore não viu (OOB, <i>out-of-bag</i>).
Escala	não precisa.
Brilha	forte sem ajuste; variância $\rho\sigma^2 + (1 - \rho)\sigma^2/B$ (correlação ρ , B árvores).
Engana	não corta o viés; não extrapola; MDI enviesada; pesada (14 MB contra 0,04 MB).

Gradient boosting HistGradientBoostingClassifier · M4	
Prevê	uma soma: $F_m = F_{m-1} + \eta h_m$ (h_m : a árvore m).
Aprende	árvores pequenas em sequência, cada uma no resíduo (o gradiente negativo da perda).
Botões	<code>learning_rate</code> (η), <code>max_iter</code> , <code>max_leaf_nodes</code> / <code>max_depth</code> , <code>early_stopping</code> . η e o nº de árvores são um botão só.
Escala	não precisa; faltantes e categorias nativos.
Brilha	tabelas: corta o viés; muitas vezes o melhor.
Engana	mais árvores podem piorar ($\eta = 1$); a parada antecipada mantém as <code>n_iter_no_change</code> árvores feitas depois da melhor.

Rede neural (MLP) MLPClassifier · E1	
Prevê	camadas $h = \varphi(Wx + b)$, com φ a ativação; saída sigmoide (2 classes) ou softmax (mais classes).
Aprende	pesos por gradiente com retropropagação; perda não convexa (a semente importa).
Botões	<code>hidden_layer_sizes</code> , <code>alpha</code> (L2), <code>activation='relu'</code> , <code>solver('adam', 'lbfgs')</code> em dados pequenos), <code>early_stopping</code> .
Escala	obrigatória.
Engana	sem ativação não linear, é uma reta disfarçada; em tabelas, o boosting costuma vencer.

Qual escolher? M1 · E5	
►	Baseline e um modelo simples (linear, logística) primeiro: a régua forte e legível.
►	Tabela heterogênea: gradient boosting ou floresta. Poucas linhas: linear, k-NN, SVM.
►	Regras para pessoas lerem: árvore rasa.
►	Mesmo cv, fold a fold; num empate, o mais simples. Não há almoço grátis.

Os modelos sem rótulo e de texto, um a um

Em todos: fit só no treino quando o resultado alimenta outro modelo, a escala decidida pelas unidades e uma comparação com os dados embaralhados.

PCA	PCA · U1
Faz	projeta nas direções de maior variância (autovetores da covariância).
Botão	n_components (inteiro ou fração da variância, como 0.95).
Escala	sim: padronize colunas de unidades diferentes.
Lê-se	explained_variance_ratio_; components_ com sinal arbitrário.
Engana	é linear; não olha o alvo; one-hot apaga as categorias raras.

t-SNE e UMAP	TSNE · umap.UMAP · U1
Faz	um mapa 2D que guarda as vizinhanças locais.
Botões	t-SNE: perplexity (5 a 50; baixa estilização). UMAP: n_neighbors (15; maior, mais global), min_dist (cosmético).
Escala	sim; PCA antes em dimensão alta.
Engana	tamanhos e distâncias entre grupos não se leem; muda com a semente; t-SNE sem transform.

k-means	KMeans · U2
Faz	k centros; minimiza a inércia (Lloyd).
Botões	n_clusters, init='k-means++', n_init=10.
Escala	sim.
Brilha	grupos redondos; tem predict; escala a milhões (MiniBatchKMeans).
Engana	luas, círculos, densidades diferentes; o cotovelo é ambíguo.

DBSCAN e HDBSCAN	DBSCAN · HDBSCAN · U2
Faz	grupos são regiões densas; o resto é ruído (−1).
Botões	DBSCAN: eps, min_samples. HDBSCAN: min_cluster_size (o menor grupo aceito).
Escala	sim.
Brilha	qualquer forma; o HDBSCAN lida com densidades diferentes.
Engana	sem predict; sofre em muitas dimensões.

Hierárquico	AgglomerativeClustering · linkage · U3
Faz	funde o par mais próximo, n − 1 vezes; a árvore traz todos os cortes.
Botões	linkage ('ward' é o padrão), metric, n_clusters ou distance_threshold.
Escala	sim.
Engana	memória $O(n^2)$; sem predict; ligação simples encadeia.

Bag-of-words	CountVectorizer · U4.1
Faz	conta cada palavra do vocabulário aprendido no fit; matriz esparsa (CSR: só os não nulos).
Botões	stop_words, min_df, max_df, ngram_range, binary, analyzer='char_wb'.
Engana	perde a ordem; palavra nova é descartada; "the, to, of" dominam a contagem.

TF-IDF	TfidfVectorizer · U4.2
Faz	contagem × idf; linhas com norma 1.
Botões	os do BoW, mais sublinear_tf, smooth_idf, norm.
Brilha	termos raros e exatos (códigos de erro, jargão); barato.
Engana	sinônimos dão cosseno 0; não sabe nada de significado.

Word embeddings	Word2Vec · GloVe · fastText · U4.3
Faz	um vetor denso por palavra, aprendido do contexto.
Botões	(gensim) vector_size, window, sg=1 (skip-gram), negative, min_count.
Engana	antônimos ficam perto (hot/cold 0,80); um vetor por palavra mistura sentidos; herda vieses.

Sentence embeddings	SentenceTransformer · U4.4
Faz	um vetor por frase (média dos tokens de um Transformer, ajustada em pares).
Uso	encode(..., normalize_embeddings=True); busca = vizinhos por cosseno.
Brilha	paráfrases, textos curtos, várias línguas, poucos rótulos.
Engana	negação e ordem; corta o texto longo em silêncio; modelo de outra língua.

Em código	U4.2 · U4.4 · U5.3
<pre>vec = TfidfVectorizer(stop_words='english', min_df=2).fit(docs_tr) X_tr, X_te = vec.transform(docs_tr), vec.transform(docs_te) E = SentenceTransformer('all-MiniLM-L6-v2').encode(docs, normalize_embeddings=True) nn = NearestNeighbors(n_neighbors=5, metric='cosine').fit(E) topic_model = BERTopic(umap_model=UMAP(n_components=5, min_dist=0.0, metric='cosine', random_state=42), hdbscan_model=HDBSCAN(min_cluster_size=10), vectorizer_model=CountVectorizer(stop_words='english')) topics, _ = topic_model.fit_transform(docs, embeddings=E)</pre>	

LDA	LatentDirichletAllocation · U5.1
Faz	tópico = distribuição de palavras; documento = mistura; usa contagens.
Botões	n_components, doc_topic_prior(α), topic_word_prior, random_state.
Brilha	documento como mistura de vários tópicos.
Engana	pede muitos documentos longos; depende da semente; ordem dos tópicos arbitrária.

NMF	NMF · U5.2
Faz	$X \approx WH$ com $W, H \geq 0$ sobre TF-IDF: tópicos "por partes".
Botões	n_components, init='nndsvda' (estável), max_iter.
Brilha	o mais barato; estável entre sementes (ARI 0,736).
Engana	pesos não são probabilidades; documentos curtos.

BERTopic	BERTopic · U5.3 · U5.4
Faz	embeddings → UMAP → HDBSCAN → c-TF-IDF.
Botões	embedding_model, umap_model (com random_state), hdbscan_model (min_cluster_size), vectorizer_model (stop words), nr_topics.
Brilha	documentos curtos; número de tópicos encontrado; outliers explícitos.
Engana	o nº de tópicos muda em degraus e com a semente; 20% de outliers é comum.

Avaliar sem rótulo	U1 · U2 · U5
Grupos	silhueta (maior), Calinski–Harabasz (maior), Davies–Bouldin (menor); a inércia só cai com k.
Mapas	trustworthiness em mais de um número de vizinhos.
Tópicos	coerência NPMI = $\log \frac{P(w_i, w_j)}{P(w_i)P(w_j)} / (-\log P(w_i, w_j)) \in [-1, 1]$, com P = fração de documentos; diversidade = fração de palavras distintas no topo.
Conferir	ARI ou homogeneidade contra um rótulo escondido; outras sementes; os dados embaralhados.

Qual escolher?	U1 · U2 · U4 · U5
▶ Alimentar um modelo: PCA. Só ver: UMAP ou t-SNE. ▶ Grupos redondos: k-means. Formas e ruído: HDBSCAN. Vários níveis: hierárquico. ▶ Texto: TF-IDF primeiro; embeddings quando o sentido importa. ▶ Tópicos: NMF barato e estável; BERTopic para textos curtos.	

Aula a aula

A pergunta de cada aula, o que levar dela e a armadilha mais comum. Os números são os das páginas das aulas.

F1 · Introdução e história

fundamentos

O que é uma máquina aprender e quando vale a pena?

- ▶ Mitchell: desempenho em T , medido por P , melhora com E . Entram dados e respostas, sai um modelo.
- ▶ Regras à mão falham quando são desconhecidas, mudam, são numerosas ou pedem personalização.
- ▶ Por que agora: dados, computação e software. Os clássicos ficam (mínimos quadrados, 1805).

🚩 **Armadilha:** usar ML para uma regra fixa e curta, que se programa.

F2 · O panorama de ML

fundamentos

Que tipos de aprendizado existem e o que é generalizar?

- ▶ Supervisionado (regressão, classificação), não supervisionado, por reforço; semi e autossupervisionado.
- ▶ Ciclo: formular, explorar, preparar, treinar, avaliar, implantar e monitorar. O trabalho está nos dados.
- ▶ 99,8% no treino e 62% em dados novos: overfitting. Justiça, privacidade, transparência.

🚩 **Armadilha:** avaliar com os dados de treino; “o modelo disse” como justificativa.

D1 · Análise exploratória

dados

Os 20.640 distritos do California Housing merecem confiança?

- ▶ Dívida antes; uma linha é um distrito, não uma casa.
- ▶ Tetos: valor em 500.001 (965 distritos), idade em 52; 207 total_bedrooms faltando.
- ▶ Pearson e Spearman; Simpson: bedrooms_per_room tem $r = -0,256$ no geral e positivo em quatro dos cinco quintis de renda.

🚩 **Armadilha:** ler $r \approx 0$ como “sem relação”; apagar outliers sem investigar.

D2 · Pré-processamento

dados

Como transformar colunas brutas em números que não enganem o algoritmo?

- ▶ Sem escala, a tarifa é 92,9% da distância; k-NN no Titanic: 0,676 sem escala, 0,821 padronizado.
- ▶ One-hot nos nominais (C = 0, Q = 1, S = 2 inventa uma ordem).
- ▶ Target encoding ingênuo “explica” ruído (treino 0,775, teste 0,505); a média imputada encolhe a variância.

🚩 **Armadilha:** fit_transform no teste.

D3 · Pipelines

dados

Como impedir, por construção, que o preparo veja o teste?

- ▶ MAE manual US\$ 43.023 (baseline 92.894); três bugs silenciosos chegam a 116.049.
- ▶ Pipeline e ColumnTransformer: o mesmo MAE, categoria nova vira zeros, colunas trocadas são recusadas.
- ▶ Target encoder fora do pipeline: 14.567 (falso); dentro: 42.552.

🚩 **Armadilha:** escolher uma configuração olhando a nota do teste.

U1 · Redução de dimensionalidade

sem rótulo

Quantas direções de fato variam? E para que reduzir?

- ▶ O motivo decide: ver (t-SNE, UMAP) ou comprimir e alimentar outro modelo (PCA).
- ▶ PCA padronizado no wine: k-NN de 0,722 para 0,926; nos dígitos, 95% da variância em 29 de 64.
- ▶ O t-SNE iguala tamanhos e encolhe distâncias; o UMAP tem transform.

🚩 **Armadilha:** ler tamanho ou distância entre grupos num mapa t-SNE.

U2 · Agrupamento

sem rótulo

Sem gabarito, como saber se os grupos existem?

- ▶ k-means, inércia, n_init; silhueta, Calinski–Harabasz, Davies–Bouldin; ARI só para conferir.
- ▶ Pinguins: os índices internos preferem $k = 2$; o ARI é máximo em $k = 3$ (0,793).
- ▶ Luas e círculos: o HDBSCAN acerta (ARI 1,000) onde o k-means falha.

🚩 **Armadilha:** silhueta alta como evidência: ruído uniforme dá 0,40.

U3 · Agrupamento hierárquico

sem rótulo

E se, em vez de k grupos, quisermos todos os cortes de uma vez?

- ▶ Ligação simples encadeia; completa e média compactam; Ward imita o k-means.
- ▶ Leia a altura, corte na faixa larga, relate níveis aninhados (Zoo: $k = 7$ dá ARI 0,950 contra 0,641 do k-means).
- ▶ $O(n^2)$ de memória: amostragem e atribua (ARI 0,993).

🚩 **Armadilha:** ler semelhança na ordem das folhas do dendrograma.

U4 · Representação de texto

texto · U4.1 a U4.4

Como transformar strings em vetores comparáveis?

- ▶ **BoW:** contagens, 99,90% de zeros (CSR); stop_words leva a concordância de vizinhos de 0,490 a 0,793.
- ▶ **TF-IDF:** o idf rebaixa as comuns; melhor resultado, 0,865, com sublinear_tf.
- ▶ **Word embeddings:** PMI, word2vec, GloVe; analogias aproximadas, vieses herdados.
- ▶ **Sentence embeddings:** no ASSIN 2, Spearman 0,715 (multilíngue) contra 0,547 (TF-IDF).

🚩 **Armadilha:** ajustar o vetorizador com o teste; stop words que apagam negações.

U5 · Modelagem de tópicos

texto · U5.1 a U5.4

Dados milhares de documentos, do que eles tratam?

- ▶ **LDA:** documento é mistura de tópicos; precisa de muitos documentos.
- ▶ **NMF:** $X \approx WH$ no TF-IDF; nos posts, ARI 0,767 contra 0,456 da LDA.
- ▶ **BERTopic:** embeddings, UMAP, HDBSCAN, c-TF-IDF; forte em textos curtos.
- ▶ **Comparação:** ARI entre sementes, LDA 0,164, NMF 0,736, BERTopic 0,324.

🚩 **Armadilha:** achar que min_cluster_size é um botão suave: 10 dá 24 tópicos, 15 dá 5.

R1 · Regressão linear

regressão

Como ajustar uma reta e o que os erros dela dizem?

- ▶ Mínimos quadrados e equação normal; Galton: inclinação 0,64, regressão à média não é uma força.
- ▶ MAE, RMSE, R^2 (5 pontos à mão: $R^2 = 0,727$).
- ▶ Resíduos: curva, funil, alavancagem; Anscombe.

🚩 **Armadilha:** ler coeficiente como importância; confiar no R^2 do treino.

R2 · Gradiente e regularização

regressão

Como treinar sem forma fechada e como frear o overfitting?

- ▶ $w \leftarrow w - \eta \nabla J$; η grande oscila ou diverge.
- ▶ Grau 30: RMSE de treino 2,46 e de validação 10,43 (desvio 13,46); erra de forma imprevisível.
- ▶ Ridge encolhe, Lasso zera; alpha por CV.

🚩 **Armadilha:** regularizar sem escalonar; escolher pelo erro de treino.

C1 · Regressão logística	classificação
<i>Como transformar um escore em probabilidade e depois em decisão?</i>	
▶ A reta dá $-0,23$ e $1,18$; a sigmoide fica em $(0, 1)$; fronteira em $z = 0$. ▶ Log loss, não o erro quadrático; $e^{w \cdot}$: a chance $\times 13,727$ por hora de estudo. ▶ Titanic: 0,804 no teste contra 0,615 do baseline.	
⚠ Armadilha: ler razão de chances como razão de probabilidades.	
C2 · k-NN	classificação
<i>Como o k-NN decide e o que ele precisa para decidir bem?</i>	
▶ Os vizinhos votam; o fit só guarda. Escala: wine de 0,722 para 0,963. ▶ Luas: $k = 1$ tem treino 1,000 e validação 0,850; $k = 73$, validação 0,910. ▶ 1000 colunas de ruído: de 0,963 para 0,556.	
⚠ Armadilha: escolher k pela acurácia de treino.	
C3 · Métricas de classificação	classificação
<i>Quais erros o modelo comete e como medir cada um?</i>	
▶ Matriz $[[TP, FN], [FP, TN]]$; precisão, revocação, especificidade, F_1. ▶ Prevalência de 1%: “sempre saudável” tem 0,990 de acurácia e revocação 0. ▶ F_β, MCC, macro; log loss e Brier. Pima: corte 0,3 por F_2, revocação 0,796.	
⚠ Armadilha: matriz sem <code>labels=[1, 0]</code> ; acurácia sem baseline.	
C4 · ROC, PR e desbalanceados	classificação
<i>Como avaliar um modelo em todos os limiares e escolher um?</i>	
▶ AUC: 86 de 100 pares na ordem certa, 0,86. AP contra a prevalência (0,653 contra 0,023). ▶ Revocação $\geq 0,8$: corte 0,0632; no teste, revocação 0,827. ▶ Pesos e reamostragem movem o ponto de operação; o corte faz o mesmo, de graça.	
⚠ Armadilha: <code>roc_auc_score</code> com rótulos; reamostrar antes de dividir.	
C5 · Validação e vazamento	classificação
<i>Como medir uma receita sem depender da sorte de uma divisão?</i>	
▶ Um split vai de 93,0% a 100,0%; CV com média $\pm$ desvio. ▶ Cinco padrões de vazamento; <code>GroupKFold</code>, <code>TimeSeriesSplit</code>. ▶ Fold a fold, 1 EP; CV aninhada 0,702 contra <code>best_score_</code> 0,712.	
⚠ Armadilha: pré-processar antes do <code>cross_val_score</code> .	

C6 · SVM	classificação
<i>Entre infinitas retas que separam os dados, qual escolher? E por quê?</i>	
▶ A rua mais larga; só os vetores de suporte contam; perda hinge. ▶ C e γ juntos; o kernel dá fronteiras curvas. ▶ Ionosphere: RBF $0,954 \pm 0,019$, vence logística e k-NN em todos os folds.	
⚠ Armadilha: não escalonar; ajustar C com γ fixo.	
M1 · Seleção de modelos	modelos
<i>O modelo erra por ser rígido demais ou por ouvir demais a amostra?</i>	
▶ Viés² + variância + ruído: grau 3 dá 0,113, com piso de 0,09. ▶ Mais dados não curam viés; curvas de validação e de aprendizado. ▶ California: Ridge com grau 3, 57.473 na CV e 56.790 no teste.	
⚠ Armadilha: relatar a nota que escolheu o vencedor.	
M2 · Árvores de decisão	modelos
<i>Como uma árvore escolhe as perguntas e por que precisa ser parada?</i>	
▶ Gini ou entropia, corte guloso; poda com <code>ccp_alpha</code>. ▶ Spambase: árvore completa, treino 1,000 e validação 0,906; 40 folhas empatam. ▶ Instável: 50 árvores votando dão 0,950.	
⚠ Armadilha: confiar no treino 1,000, que sai até com rótulos aleatórios.	
M3 · Random forest	modelos
<i>Como uma média de modelos ruins pode ser um modelo bom?</i>	
▶ Média de B árvores de variância σ^2 e correlação ρ: $\rho\sigma^2 + (1 - \rho)\sigma^2 / B$. Árvores boas e pouco correlacionadas. ▶ <code>max_features</code> descorrelaciona (digits: erro de 0,053 para 0,023). ▶ OOB 0,975 contra CV 0,977; mais árvores nunca pioram.	
⚠ Armadilha: MDI: uma coluna de ruído em 2º lugar.	
M4 · Gradient boosting	modelos
<i>E se somarmos árvores pequenas, cada uma no erro das anteriores?</i>	
▶ A floresta combate a variância; o boosting, o viés (tocos: 0,280 contra 0,572). ▶ Resíduo = gradiente; η menor pede mais árvores; parada antecipada. ▶ Bank Marketing: tirar o vazamento mudou a AP em 0,18; ajustar, em 0,007.	
⚠ Armadilha: <code>duration</code> , que só existe depois da ligação.	

E1 · Redes neurais	extensões
<i>O que uma rede calcula e como ela aprende?</i>	
▶ Um neurônio sigmoide é a logística; o XOR pede camada oculta. ▶ Sem não linearidade: 0,510; com <code>relu</code>: 0,983. Backprop é a regra da cadeia. ▶ Spambase: boosting 0,956, MLP 0,948.	
⚠ Armadilha: escalonar fora do pipeline; ignorar a semente.	
E2 · Explicabilidade	extensões
<i>Em que o modelo se apoia? E por que fez esta previsão?</i>	
▶ Interpretável primeiro; permutação no teste; MDI premia ruído. ▶ SHAP soma: $207,39 + 222,35 = 429,74$; PDP e ICE. ▶ Hospital: 93,4% no A, 53,8% no B (atalho).	
⚠ Armadilha: ler explicação como causa.	
E3 · AutoML	extensões
<i>Como gastar o orçamento da busca?</i>	
▶ Escala log; a aleatória vence a grade; TPE, halving, Hyperband. ▶ Phoneme: a busca ganha 0,008 no teste; o espaço importa mais que o método. ▶ CV aninhada 0,9472 contra <code>best_score_</code> 0,9499.	
⚠ Armadilha: a busca otimiza o vazamento (0,916 contra 0,792).	
E4 · MLOps	extensões
<i>O que mantém um modelo confiável depois de entregue?</i>	
▶ Pipeline inteiro, versões fixas, schema, sementes e hash dos dados; um pickle executa código ao carregar. ▶ Covariate, concept e prior shift; PSI acima de 0,25 alerta. ▶ Estático: acurácia de 0,801 a 0,273 em 12 meses; retreinado acompanha.	
⚠ Armadilha: achar que drift é dano; desalinhamento treino–serving.	
E5 · A fronteira	extensões
<i>Onde esta caixa de ferramentas fica entre os modelos de fundação?</i>	
▶ Transfer learning: extrair atributos ou fine-tuning (LoRA, adaptação de baixo posto: 0,4% dos pesos). ▶ O clássico vence em tabelas, latência, decisões reguladas, dados pequenos. ▶ RAG (geração aumentada por recuperação) é k-NN sobre embeddings; roteamento: linguagem para os modelos de fundação, tabelas para o clássico.	
⚠ Armadilha: dispensar o teste rotulado porque “o prompt funciona”.	

Adaptada da folha de consulta de Machine Learning da disciplina Inteligência Artificial, de Fabio Roberto de Miranda, Diego Pavan Soler e Humberto Sandmann.