

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**Dimensionalidade e Escala em
Aprendizado de Máquina**

***Uma Análise Teórica e Experimental da
Redução de Dimensionalidade***

Felipe Pereira Ramos Barboza

MONOGRAFIA FINAL

**MAC 499 — TRABALHO DE
FORMATURA SUPERVISIONADO**

Supervisora: Prof.^a Dr.^a Nina S. T. Hirata

São Paulo
2025

*O conteúdo deste trabalho é publicado sob a licença CC BY 4.0
(Creative Commons Attribution 4.0 International License)*

Resumo

Felipe Pereira Ramos Barboza. **Dimensionalidade e Escala em Aprendizado de Máquina: Uma Análise Teórica e Experimental da Redução de Dimensionalidade.**

Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2025.

O crescimento exponencial na quantidade e complexidade dos dados modernos tem trazido desafios significativos para o aprendizado de máquina e a mineração de dados, especialmente quando se trata de representações em espaços de alta dimensão. Este trabalho apresenta uma análise teórica e prática da redução de dimensionalidade, discutindo tanto os fundamentos matemáticos que asseguram sua viabilidade quanto as técnicas contemporâneas que a implementam. Primeiramente, demonstra-se, com base no Lema de Johnson–Lindenstrauss, que é possível reduzir a dimensionalidade de conjuntos de dados mantendo aproximadamente as distâncias originais entre pontos. Em seguida, investigam-se métodos modernos de redução não linear, como o *t-Distributed Stochastic Neighbor Embedding* (t-SNE) e o *Uniform Manifold Approximation and Projection* (UMAP), explorando suas propriedades e aplicações em análise de dados de alta dimensão.

Palavras-chave: Redução de Dimensionalidade. Lema de Johnson–Lindenstrauss. t-SNE. UMAP.

Listas de figuras

1.1	Grade 20×20 obtida projetando embeddings do CIFAR-10 com UMAP após extração de features por uma ResNet-50. As imagens organizam-se por similaridade visual, mesmo sem informação de classe.	2
6.1	Comparação entre t-SNE e UMAP no MNIST. O t-SNE mantém vizinhanças locais com maior fidelidade, enquanto o UMAP produz agrupamentos mais compactos e bem separados.	22
6.2	Comparação entre t-SNE e UMAP no Fashion-MNIST. O t-SNE preserva melhor pequenas variações locais, enquanto o UMAP forma agrupamentos mais definidos mesmo em classes naturalmente sobrepostas.	23
6.3	Comparação entre t-SNE e UMAP no CIFAR-10. O t-SNE revela subestruturas internas dentro das classes, enquanto o UMAP gera clusters mais compactos e separados, coerentes com os valores mais altos de DSC. . . .	24
6.4	Comparação entre t-SNE e UMAP no Breast Cancer. Como o conjunto possui apenas duas classes, ambos os métodos produzem separações claras, com diferenças visuais mais sutis do que nos datasets de imagens. . . .	24

Listas de tabelas

6.1	Tempo de execução (em segundos) para cada projeção.	21
6.2	Métricas de avaliação das projeções (valores conforme saída dos experimentos).	22

Sumário

1	Introdução	1
2	Is Dimensionality Reduction Feasible?	3
2.1	A viabilidade da redução de dimensionalidade	3
2.2	Demonstração do Lema de Johnson–Lindenstrauss via concentração gaussiana	4
3	Projeções Não Lineares e o Método t-SNE	9
3.1	Limitações das Projeções Aleatórias	9
3.2	Da SNE ao t-SNE: uma evolução motivada por limitações práticas	10
3.3	O modelo probabilístico do SNE	10
3.4	Soluções do t-SNE	11
3.5	Função de custo e otimização	12
4	O Método UMAP	13
4.1	Uma nova perspectiva sobre redução de dimensionalidade	13
4.2	Da probabilidade à topologia: o salto conceitual do t-SNE ao UMAP	13
4.3	Uma visão computacional: grafos e otimização	14
5	Metodologia Experimental	17
5.1	Conjuntos de Dados	17
5.2	Pipeline Experimental	18
5.3	Métricas de Avaliação	18
5.4	Implementação e Reprodutibilidade	19
6	Resultados	21
6.1	Visão Geral dos Resultados	21
6.2	MNIST	22
6.3	Fashion-MNIST	22

6.4	CIFAR-10	23
6.5	Breast Cancer Wisconsin	24
7	Conclusão	25

Apêndices

A	Notebook de Experimentos	27
----------	---------------------------------	-----------

Referências	37
--------------------	-----------

Capítulo 1

Introdução

Em cenários de aprendizado de máquina e mineração de dados, é comum lidar com conjuntos de dados de alta dimensionalidade. Exemplos típicos incluem bases de mercado compostas por milhares de produtos, representações textuais em que cada documento é descrito por um vetor de tamanho igual ao vocabulário, e perfis de expressão gênica com dezenas de milhares de atributos contínuos.

A alta dimensionalidade impõe desafios significativos tanto em termos computacionais, devido ao elevado custo de processamento, quanto estatísticos, em virtude do fenômeno conhecido como *maldição da dimensionalidade* (*curse of dimensionality*). À medida que o número de dimensões cresce, os dados tornam-se mais esparsos e as distâncias entre pontos perdem significado, prejudicando tarefas fundamentais como classificação, agrupamento e visualização.

Para mitigar esses problemas, recorre-se às técnicas de redução de dimensionalidade, cujo objetivo é projetar os dados em um espaço de menor dimensão preservando, tanto quanto possível, suas propriedades geométricas e estruturais. Métodos tradicionais como a Análise de Componentes Principais (PCA) e a Análise Discriminante Linear (LDA) foram amplamente estudados e aplicados nas últimas décadas, com bons resultados em contextos lineares. Entretanto, o aumento da complexidade e não linearidade dos dados modernos evidencia as limitações dessas abordagens e motiva o estudo de técnicas mais robustas.

A base teórica que fundamenta a viabilidade dessa redução é apresentada pelo Lema de Johnson–Lindenstrauss (JL), que demonstra ser possível projetar pontos de um espaço de alta dimensão em um espaço de dimensão muito menor, preservando aproximadamente as distâncias entre eles. Esse resultado fornece um respaldo matemático essencial para o uso de métodos de redução de dimensionalidade em contextos práticos, garantindo que as transformações realizadas mantenham a estrutura global dos dados de forma controlada.

Com base nesse fundamento, surgem técnicas modernas de natureza não linear, como o *t-Distributed Stochastic Neighbor Embedding* (t-SNE) [MAATEN e HINTON, 2008](#) e o *Uniform Manifold Approximation and Projection* (UMAP) [MCINNES et al., 2018](#). Ambas buscam preservar relações de vizinhança entre os dados, permitindo representar de forma mais fiel a geometria intrínseca de conjuntos complexos em espaços de duas ou três dimensões. Esses métodos são amplamente utilizados em análises exploratórias e visualizações de

dados de alta dimensão, como imagens, textos e dados biológicos.

Para dar uma ideia mais concreta do tipo de estrutura que essas técnicas conseguem revelar, a Figura 1.1 mostra um exemplo simples: projetei as features de uma ResNet-50 treinada no CIFAR-10 usando UMAP e organizei o resultado em uma grade 20×20 . Mesmo sem nenhuma informação explícita sobre classes, as imagens começam a se agrupar por similaridade visual: carros perto de carros, animais perto de animais, e assim por diante. Esse tipo de efeito dá uma boa intuição do porquê projeções não lineares se tornaram ferramentas tão úteis para explorar dados de alta dimensão.

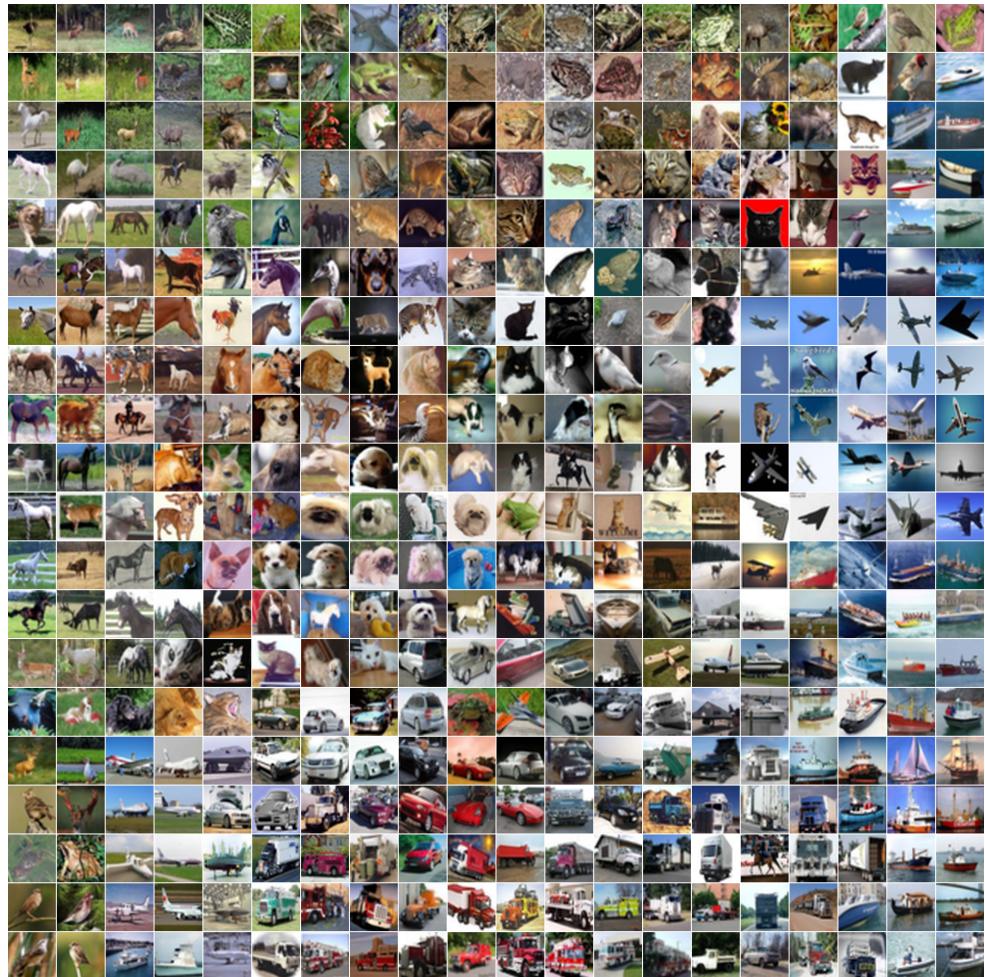


Figura 1.1: Grade 20×20 obtida projetando embeddings do CIFAR-10 com UMAP após extração de features por uma ResNet-50. As imagens organizam-se por similaridade visual, mesmo sem informação de classe.

Capítulo 2

Is Dimensionality Reduction Feasible?

2.1 A viabilidade da redução de dimensionalidade

A alta dimensionalidade é uma característica onipresente nos dados modernos. Em aprendizado de máquina, é comum lidar com representações que envolvem milhares ou até milhões de dimensões, como vetores de palavras em modelos de linguagem, descritores visuais em imagens ou perfis genéticos em biologia computacional. Embora esses espaços sejam extremamente ricos em informação, eles impõem um custo computacional elevado e dificultam a interpretação, a visualização e o armazenamento dos dados.

Surge, então, uma questão fundamental: *até que ponto é possível reduzir a dimensionalidade dos dados sem comprometer sua estrutura geométrica essencial?* Em outras palavras, podemos projetar um conjunto de pontos de um espaço de alta dimensão para outro de dimensão muito menor, preservando, ao menos aproximadamente, as distâncias entre eles?

Antes de responder, vale lembrar um fato básico de álgebra linear: dados n pontos $x_1, x_2, \dots, x_n \in \mathbb{R}^D$, esses pontos sempre pertencem a um subespaço de dimensão no máximo $n - 1$. De modo análogo ao fato de que dois pontos definem uma reta e três pontos definem um plano, qualquer conjunto de n pontos é, no máximo, $(n - 1)$ -dimensional. Essa cota é, inclusive, *existencialmente justa*: se os vetores $x_2 - x_1, x_3 - x_1, \dots, x_n - x_1$ forem mutuamente ortogonais, não é possível representá-los em uma dimensão menor sem distorcer completamente as distâncias.

Contudo, esse limite se refere ao caso em que desejamos preservar as distâncias *exatamente*. Se aceitarmos pequenas distorções, ou seja, permitir que as distâncias sejam apenas *aproximadamente* preservadas, será que conseguimos reduzir ainda mais a dimensão?

O **Lema de Johnson-Lindenstrauss** fornece uma resposta surpreendentemente positiva e elegante a essa pergunta. Ele mostra que é possível projetar qualquer conjunto de n pontos de um espaço \mathbb{R}^D em um espaço de dimensão muito menor $k = O(\log n / \varepsilon^2)$, onde ε é o erro máximo tolerado, de forma que todas as distâncias euclidianas sejam preservadas

dentro de um fator multiplicativo de $(1 \pm \varepsilon)$. Formalmente, existe uma transformação

$$f : \mathbb{R}^D \rightarrow \mathbb{R}^k$$

tal que, para quaisquer $x, y \in \mathbb{R}^D$,

$$(1 - \varepsilon)\|x - y\|^2 \leq \|f(x) - f(y)\|^2 \leq (1 + \varepsilon)\|x - y\|^2.$$

Em outras palavras, os pontos são comprimidos em um espaço de dimensão muito menor, mas sua geometria relativa é, portanto, as relações que estruturam o conjunto de dados permanecem essencialmente inalteradas.

Para ilustrar, considere um modelo de linguagem que associa a cada palavra um vetor em \mathbb{R}^{10000} . Se o vocabulário contém 10^6 palavras, temos um conjunto de um milhão de vetores em 10 mil dimensões, um cenário computacionalmente custoso para cálculos de similaridade. O Lema de Johnson–Lindenstrauss garante, entretanto, que é possível projetar esses vetores em um espaço de dimensão $k \approx 2000$ (para $\varepsilon = 0,1$), preservando as distâncias com erro de apenas 10%.

Esse resultado é notável por duas razões principais. Primeiro, a dimensão reduzida k depende apenas do número de pontos n e do erro ε , e não da dimensão original D . Assim, mesmo dados que residem em espaços de bilhões de dimensões podem ser representados de forma compacta, mantendo sua estrutura geométrica essencial. Segundo, o lema oferece uma base teórica rigorosa para técnicas práticas de redução de dimensionalidade, como projeções aleatórias, que são amplamente empregadas em algoritmos de aprendizado de máquina, mineração de dados e compressão de informação.

Portanto, o Lema de Johnson–Lindenstrauss estabelece, de maneira formal e quantitativa, que a redução de dimensionalidade não é apenas possível: ela é garantida matematicamente dentro de limites de distorção bem controlados. A seguir, apresentaremos a demonstração do lema, baseada em propriedades de concentração de medidas gaussianas.

2.2 Demonstração do Lema de Johnson–Lindenstrauss via concentração gaussiana

A prova utiliza uma abordagem probabilística. A ideia central é mostrar que a probabilidade de existir uma transformação linear que satisfaça as desigualdades do lema é positiva.

Construção da transformação linear aleatória. Seja $G \in \mathbb{R}^{k \times D}$ uma matriz cujos elementos g_{ij} são variáveis aleatórias independentes, com distribuição normal padrão $\mathcal{N}(0, 1)$. Definimos a aplicação linear aleatória $G : \mathbb{R}^D \rightarrow \mathbb{R}^k$ dada por $y = Gx$.

Aplicando G a dois pontos $x_p, x_q \in V$, queremos demonstrar que

$$\mathbb{P}\left[\forall x_p, x_q \in \mathbb{R}^D : (1 - \varepsilon)\|x_p - x_q\| \leq \|y_p - y_q\| \leq (1 + \varepsilon)\|x_p - x_q\|\right] > 0. \quad (2.1)$$

Como cada linha de G é composta de variáveis Gaussianas independentes, o vetor $y = Gx$

é uma variável aleatória Gaussiana com média nula e variância $\sum_{i=1}^D x_i^2 = \|x\|^2$. Assim,

$$y = \|x\|z, \quad \text{onde } z = (z_1, \dots, z_k), \quad z_i \sim \mathcal{N}(0, 1).$$

Para dois vetores distintos x_p e x_q , pela linearidade de Gx , segue que

$$y_p - y_q = \|x_p - x_q\|z.$$

Logo,

$$\|Gx_p - Gx_q\| = \|x_p - x_q\|\|z\|. \quad (2.2)$$

Utilizaremos o seguinte resultado.

Definição 1 (Função Lipschitz). *Seja $F : \mathbb{R}^m \rightarrow \mathbb{R}$. Dizemos que F é Lipschitz com constante $L > 0$ se, para todos $x, y \in \mathbb{R}^m$,*

$$|F(x) - F(y)| \leq L\|x - y\|.$$

Isto é, F não pode aumentar a distância entre dois pontos por mais do que um fator L .

Lemma 1 (Concentração Gaussiana). *Seja $F : \mathbb{R}^m \rightarrow \mathbb{R}$ uma função Lipschitz com constante $L > 0$. Seja $g \sim \mathcal{N}(0, I_m)$. Então, para todo $t \geq 0$,*

$$\mathbb{P}(|F(g) - \mathbb{E}[F(g)]| \geq t) \leq 2 \exp\left(-\frac{t^2}{4L^2}\right).$$

Lemma 2. A norma euclidiana $\|\cdot\| : \mathbb{R}^m \rightarrow \mathbb{R}$ é uma função de Lipschitz com constante $L = 1$.

Demonstração. Sejam $x, y \in \mathbb{R}^m$. Pela desigualdade triangular, temos:

$$\|x\| \leq \|x - y\| + \|y\|.$$

Rearranjando os termos, obtemos:

$$\|x\| - \|y\| \leq \|x - y\|. \quad (*)$$

Aplicando novamente a desigualdade triangular, agora trocando os papéis de x e y , segue que:

$$\|y\| \leq \|y - x\| + \|x\|.$$

Reorganizando,

$$\|y\| - \|x\| \leq \|y - x\|.$$

Como $\|y - x\| = \|-(x - y)\| = |-1| \cdot \|x - y\| = \|x - y\|$, podemos reescrever:

$$-(\|x\| - \|y\|) \leq \|x - y\|. \quad (**)$$

Combinando as desigualdades (*) e (**), concluímos que:

$$\|x\| - \|y\| \leq \|x - y\|.$$

Esta é exatamente a condição de Lipschitz para a função $f(x) = \|x\|$, isto é,

$$|f(x) - f(y)| \leq L \cdot \|x - y\|,$$

com constante de Lipschitz $L = 1$. Portanto, a função norma é 1-Lipschitz. \square

A norma euclidiana $\|z\|$ é uma função Lipschitz com constante $L = 1$, portanto,

$$\mathbb{P}(\|z\| - \mathbb{E}\|z\| \geq t) \leq 2e^{-t^2/4}. \quad (2.3)$$

Aproximação da distância. Escolhendo $t = \varepsilon \mathbb{E}\|z\|$, obtemos:

$$\mathbb{P}\left[(1 - \varepsilon)\mathbb{E}\|z\| \leq \|z\| \leq (1 + \varepsilon)\mathbb{E}\|z\|\right] \geq 1 - 2e^{-\varepsilon^2(\mathbb{E}\|z\|)^2/4}.$$

Combinando com (2.2) e normalizando G por $\mathbb{E}\|z\|$, isto é, definindo $\widehat{G} = \frac{1}{\mathbb{E}\|z\|}G$, segue que

$$\mathbb{P}\left[(1 - \varepsilon)\|x_p - x_q\| \leq \|\widehat{G}x_p - \widehat{G}x_q\| \leq (1 + \varepsilon)\|x_p - x_q\|\right] \geq 1 - 2e^{-\varepsilon^2(\mathbb{E}\|z\|)^2/4}.$$

Considerando todos os pares de pontos - Union Bound Existem no máximo $\frac{N^2}{2}$ pares distintos de pontos em $V = x_1, x_2, \dots, x_n$. Usando a desigualdade da união e a lei de De Morgan, temos:

$$\mathbb{P}\left[\forall x_p, x_q \in V : (1 - \varepsilon)\|x_p - x_q\| \leq \|\widehat{G}x_p - \widehat{G}x_q\| \leq (1 + \varepsilon)\|x_p - x_q\|\right] \geq 1 - N^2 e^{-\varepsilon^2(\mathbb{E}\|z\|)^2/4}.$$

Por fim, como escolher a dimensão k ? Para garantir que a probabilidade acima seja positiva, impomos:

$$1 - N^2 e^{-\varepsilon^2(\mathbb{E}\|z\|)^2/4} > 0.$$

Sabendo que $\mathbb{E}\|z\| \geq c\sqrt{k}$ para uma constante $c > 0$, segue que a condição é satisfeita se

$$k \geq \frac{8c^2}{\varepsilon^2} \log N.$$

Portanto, se k cresce proporcionalmente a $\frac{\log N}{\varepsilon^2}$, existe, com probabilidade positiva, uma aplicação linear que satisfaz as desigualdades do lema.

Assim, concluímos que existe uma transformação linear $\widehat{G} : \mathbb{R}^D \rightarrow \mathbb{R}^k$ tal que, para todos os $x_p, x_q \in V$,

$$(1 - \varepsilon)\|x_p - x_q\| \leq \|\widehat{G}(x_p) - \widehat{G}(x_q)\| \leq (1 + \varepsilon)\|x_p - x_q\|.$$

2.2 | DEMONSTRAÇÃO DO LEMA DE JOHNSON–LINDENSTRAUSS VIA CONCENTRAÇÃO GAUSSIANA

□

Capítulo 3

Projeções Não Lineares e o Método t-SNE

3.1 Limitações das Projeções Aleatórias

O Lema de Johnson–Lindenstrauss fornece uma base teórica sólida para a redução de dimensionalidade linear por meio de projeções aleatórias. A simplicidade e a generalidade desse resultado o tornam atraente: basta uma transformação linear gerada aleatoriamente para garantir, com alta probabilidade, que as distâncias entre pontos sejam aproximadamente preservadas. Contudo, embora eficaz em termos geométricos globais, essa abordagem ignora aspectos cruciais da estrutura interna dos dados.

Em muitas aplicações práticas, os dados de alta dimensão não estão distribuídos uniformemente no espaço \mathbb{R}^D , mas organizam-se em regiões de baixa dimensionalidade — chamadas de *variedades (manifolds)* — imersas no espaço original. Essa estrutura não linear é típica em contextos como reconhecimento de imagens, representação semântica de textos e análise de expressões gênicas. Em tais casos, preservar apenas as distâncias euclidianas globais pode distorcer significativamente as relações de vizinhança locais que definem a geometria intrínseca dos dados.

Além disso, projeções aleatórias e outros métodos lineares tratam todas as direções do espaço de forma equivalente, sem distinguir regiões de maior densidade ou agrupamentos naturais. Como consequência, embora as distâncias médias sejam preservadas, a *estrutura local*, essencial para a interpretação e visualização, é frequentemente perdida. Isso torna tais técnicas pouco adequadas para tarefas exploratórias em que se busca compreender padrões, agrupamentos e transições entre regiões do espaço de dados.

Essas limitações motivam o estudo de métodos de redução de dimensionalidade **não lineares**, capazes de preservar não apenas as distâncias globais, mas também as relações de vizinhança locais que revelam a estrutura subjacente do conjunto de dados.

3.2 Da SNE ao t-SNE: uma evolução motivada por limitações práticas

O *t-Distributed Stochastic Neighbor Embedding* (t-SNE) foi proposto por van der Maaten e Hinton em 2008 [MAATEN e HINTON, 2008](#) como uma evolução conceitual e prática do método *Stochastic Neighbor Embedding* (SNE) originalmente desenvolvido por Hinton e Roweis em 2002. Ambos os algoritmos têm como objetivo representar, em baixa dimensão, a estrutura de similaridade entre pontos de um conjunto de dados de alta dimensão, de modo que padrões e agrupamentos possam ser visualizados de forma intuitiva.

O ponto de partida é a observação de que métodos lineares, como PCA e projeções aleatórias, não são capazes de capturar estruturas não lineares presentes nos dados. Muitas vezes, as observações de um conjunto residem em uma variedade de baixa dimensão imersa em um espaço muito maior. Nesse contexto, preservar distâncias euclidianas globais – como fazem as projeções lineares – pode distorcer significativamente as relações locais entre vizinhos próximos. A motivação principal do SNE e de seu sucessor t-SNE é, portanto, preservar relações de vizinhança probabilísticas, em vez de distâncias absolutas.

3.3 O modelo probabilístico do SNE

O SNE original propõe uma formulação probabilística para medir similaridades entre pontos. Dado um conjunto de vetores de alta dimensão $x_1, \dots, x_n \in \mathbb{R}^D$, define-se a probabilidade condicional $p_{j|i}$ de que o ponto x_i escolheria x_j como seu vizinho, caso os vizinhos fossem selecionados de acordo com uma distribuição gaussiana centrada em x_i :

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)}.$$

O parâmetro σ_i é ajustado individualmente para que a entropia da distribuição local em torno de x_i corresponda a uma *perplexidade* especificada pelo usuário, que controla o número efetivo de vizinhos considerados.

De modo análogo, para os pontos em baixa dimensão $y_1, \dots, y_n \in \mathbb{R}^k$ (tipicamente com $k = 2$ ou 3), define-se uma probabilidade condicional $q_{j|i}$ baseada nas distâncias entre os pontos projetados:

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)}.$$

O objetivo do SNE é minimizar a divergência de Kullback–Leibler (KL) entre as distribuições condicionais $P_i = \{p_{j|i}\}$ e $Q_i = \{q_{j|i}\}$ para todos os pontos i :

$$C = \sum_i KL(P_i \| Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}.$$

Assim, o algoritmo busca uma configuração de pontos em baixa dimensão em que as

relações de vizinhança sejam preservadas o mais fielmente possível.

Apesar de inovador, o SNE original apresenta duas limitações principais:

1. **Assimetrias nas probabilidades:** as distribuições condicionais $p_{j|i}$ e $p_{i|j}$ não são necessariamente iguais, o que introduz inconsistências entre pares de pontos.
2. **Problema de crowding:** ao projetar um espaço de alta dimensão em duas dimensões, torna-se difícil representar corretamente todas as relações locais, levando a sobreposição de grupos.

Esses problemas motivaram o desenvolvimento de uma versão modificada, o , que introduz ajustes probabilísticos e geométricos para contornar tais limitações.

3.4 Soluções do t-SNE

Para resolver o problema da assimetria, o t-SNE substitui as probabilidades condicionais $p_{j|i}$ por probabilidades conjuntas simétricas:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}.$$

Essa reformulação garante que $p_{ij} = p_{ji}$ e que a soma de todas as probabilidades é igual a 1. No espaço de baixa dimensão, define-se de forma análoga:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}.$$

A escolha da função $(1 + \|y_i - y_j\|^2)^{-1}$ corresponde à densidade da distribuição t de Student com um grau de liberdade, introduzida para lidar com o problema de *crowding*.

O *crowding problem* decorre do fato de que, ao reduzir de alta para baixa dimensão, o volume disponível para posicionar os vizinhos próximos diminui rapidamente. Em espaços de alta dimensão, há muito “espaço” em torno de um ponto para abrigar seus vizinhos, mas em duas dimensões esse espaço é drasticamente reduzido, o que causa sobreposição de grupos e perda de separabilidade visual.

O t-SNE resolve esse problema substituindo a distribuição gaussiana usada no espaço de baixa dimensão por uma distribuição t-Student com um grau de liberdade, cuja cauda longa permite representar separações maiores entre grupos distintos. Isso evita que pontos distantes sejam projetados muito próximos, preservando melhor a estrutura global de agrupamentos.

3.5 Função de custo e otimização

O t-SNE minimiza a divergência de Kullback–Leibler entre as distribuições $P = \{p_{ij}\}$ e $Q = \{q_{ij}\}$:

$$C = KL(P\|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

Essa função mede o quanto a distribuição de vizinhança no espaço de baixa dimensão difere da original. O gradiente do custo em relação a cada ponto y_i é dado por:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}.$$

A presença do fator $(1 + \|y_i - y_j\|^2)^{-1}$ reduz a influência de pares distantes, concentrando o ajuste nas relações locais mais relevantes. O custo é minimizado por descida de gradiente, com técnicas auxiliares como *momentum* e uma fase inicial de *early exaggeration*, que amplifica temporariamente as probabilidades p_{ij} para facilitar a separação de agrupamentos no início da otimização.

Capítulo 4

O Método UMAP

4.1 Uma nova perspectiva sobre redução de dimensionalidade

O *Uniform Manifold Approximation and Projection* (UMAP), proposto em 2018 por McInnes, Healy e Melville, combina fundamentos geométricos e computacionais em um método eficiente de redução de dimensionalidade. Enquanto o t-SNE concentra-se em preservar relações locais entre pontos, o UMAP estende essa ideia ao incorporar princípios topológicos que descrevem a estrutura global dos dados.

Parte-se da hipótese de que os dados estão organizados em torno de uma variedade de baixa dimensão imersa em um espaço de dimensão maior. Reduzir a dimensionalidade, portanto, significa projetar essa variedade em um espaço plano preservando, tanto quanto possível, suas relações de vizinhança. O UMAP traduz essa formulação geométrica em um algoritmo prático, capaz de reconstruir a conectividade dos dados e projetá-la em baixa dimensão de modo coerente e eficiente.

4.2 Da probabilidade à topologia: o salto conceitual do t-SNE ao UMAP

O t-SNE parte de uma visão essencialmente estatística. Ele interpreta as distâncias entre pontos como probabilidades de vizinhança e tenta construir, em baixa dimensão, uma distribuição de similaridades que imite a do espaço original. Esse modelo funciona surpreendentemente bem para revelar agrupamentos, mas peca em dois aspectos: falta-lhe uma noção de continuidade global (os diferentes grupos aparecem isolados, sem relações entre si) e seu custo computacional cresce rapidamente com o número de pontos.

O UMAP nasce como uma resposta a essas limitações. McInnes, Healy e Melville reformulam o problema não como a preservação de probabilidades, mas como a preservação de uma estrutura topológica. A proximidade entre dois pontos deixa de ser uma questão de probabilidade e passa a ser entendida como a intensidade de uma conexão em um grafo

de vizinhança, uma espécie de mapa da geometria local da variedade subjacente. Cada ponto é conectado aos seus vizinhos mais próximos, e a força dessa conexão depende da densidade local: em regiões mais povoadas, o “alcance” da conexão é menor; em regiões mais esparsas, ele se expande para compensar.

O resultado é um grafo ponderado que captura, de forma adaptativa, as relações de vizinhança entre os dados. Em vez de uma distribuição probabilística como no t-SNE, o UMAP constrói o que seus autores chamam de um *fuzzy simplicial set*: uma representação difusa da topologia da variedade, que descreve como as regiões de dados se conectam entre si.

4.3 Uma visão computacional: grafos e otimização

Do ponto de vista teórico, o UMAP foi construído sobre conceitos de topologia algébrica e teoria das categorias, mas, em sua forma computacional, ele se traduz em algo mais tangível: a construção e a manipulação de um grafo ponderado. No fim das contas, todo o aparato teórico de variedades e conjuntos simpliciais se materializa como um grafo em que cada nó representa um ponto do conjunto de dados e cada aresta representa o grau de conectividade entre pares de pontos. É por isso que o UMAP se encaixa naturalmente na família dos algoritmos de aprendizado baseados em grafos de vizinhança, ao lado do Isomap [TENENBAUM et al., 2000](#), do Laplacian Eigenmaps [BELKIN e NIYOGI, 2003](#) e, sob certo ponto de vista, também do t-SNE.

O algoritmo pode ser compreendido em duas grandes etapas conceituais. Na primeira, constrói-se um grafo que descreve a estrutura local dos dados no espaço original de alta dimensão. Na segunda, busca-se uma disposição dos mesmos vértices em um espaço de baixa dimensão tal que esse novo grafo, agora projetado, preserve as relações do grafo original tanto quanto possível. Essas duas fases, construção e layout, são comuns a toda a classe de algoritmos baseados em grafos. O que distingue o UMAP é a forma como ele define a conectividade entre pontos e como traduz essa estrutura em um problema de otimização.

Construindo o grafo de vizinhança

Comecemos pelo grafo inicial. Dado um conjunto de pontos $X = \{x_1, x_2, \dots, x_N\}$ e uma métrica $d : X \times X \rightarrow \mathbb{R}_{\geq 0}$, o UMAP procura identificar, para cada ponto x_i , os seus k vizinhos mais próximos segundo essa métrica. Em vez de apenas conectar cada ponto aos seus vizinhos, o algoritmo atribui um peso a cada aresta — um valor entre 0 e 1 que expressa o “grau de pertencimento” de x_j à vizinhança de x_i .

Esse peso é dado por uma função exponencial suavizada:

$$w_{j|i} = \exp\left(-\frac{\max(0, d(x_i, x_j) - \rho_i)}{\sigma_i}\right),$$

em que dois parâmetros locais, ρ_i e σ_i , regulam a escala da vizinhança. O valor ρ_i é definido como a menor distância positiva de x_i a outro ponto do conjunto, garantindo que cada nó do grafo esteja conectado a pelo menos um vizinho com peso máximo igual a 1. Já σ_i é determinado de modo que a soma dos pesos dentro da vizinhança de x_i corresponda aproximadamente a $\log_2(k)$; é, portanto, um fator de normalização que adapta a noção

de “escala local” à densidade dos dados. Essas escolhas asseguram que o grafo resultante respeite a conectividade local da variedade subjacente, uma propriedade crucial em espaços de alta dimensão, onde as distâncias brutas tendem a perder significado.

O resultado dessa etapa é um grafo dirigido, já que as relações de vizinhança não são simétricas em geral. Para obter uma representação coerente da conectividade, o UMAP combina as direções opostas por meio de uma operação inspirada na teoria dos conjuntos fuzzy:

$$w_{ij} = w_{i|j} + w_{j|i} - w_{i|j} w_{j|i}.$$

Se interpretarmos $w_{i|j}$ como a “probabilidade” de uma aresta dirigida existir de x_i para x_j , então w_{ij} representa a probabilidade de que ao menos uma das duas arestas, de i para j ou de j para i , esteja presente. O grafo resultante é ponderado, não-direcional, e descreve, de maneira compacta, a estrutura de conectividade do conjunto de dados. Ele é, do ponto de vista topológico, o esqueleto de um conjunto simplicial difuso que aproxima a geometria local da variedade em que os dados residem.

O layout em baixa dimensão

Uma vez construída essa representação, o objetivo é encontrar um novo conjunto de pontos $Y = \{y_1, y_2, \dots, y_N\}$ em baixa dimensão que preserve a estrutura do grafo. Em termos intuitivos, trata-se de um problema de layout de grafo: queremos posicionar os vértices em um plano de modo que arestas fortes mantenham os pontos próximos e arestas fracas os afastem. Essa é uma ideia herdada dos métodos de *force-directed graph drawing*, em que o layout emerge do equilíbrio entre forças atrativas e repulsivas.

No caso do UMAP, a força atrativa entre dois pontos y_i e y_j é proporcional ao peso da aresta w_{ij} e à derivada da função de conectividade no espaço reduzido. De forma simplificada, a relação de proximidade entre dois pontos projetados é modelada por:

$$q_{ij} = \frac{1}{1 + a \|y_i - y_j\|^{2b}},$$

onde a e b são parâmetros que controlam a forma da curva — ajustados para que ela se comporte como uma função logística invertida, com caudas longas que evitam o colapso dos clusters. O processo de otimização consiste em minimizar a divergência cruzada entre os pesos w_{ij} do grafo original e os valores q_{ij} do grafo projetado:

$$C = \sum_{i < j} \left[w_{ij} \log \frac{w_{ij}}{q_{ij}} + (1 - w_{ij}) \log \frac{1 - w_{ij}}{1 - q_{ij}} \right].$$

Essa função mede o quanto a estrutura de vizinhança do espaço original é preservada na projeção. Seu gradiente em relação às coordenadas y_i pode ser interpretado como um sistema de forças: as arestas com pesos altos puxam os pontos para mais perto, enquanto pares de pontos desconectados geram uma leve repulsão, impedindo que tudo colapse em uma única região.

Computacionalmente, o algoritmo é implementado de forma eficiente por meio de amostragem negativa — uma técnica que permite estimar as forças repulsivas sem calcular

todas as interações possíveis entre pares de pontos, reduzindo a complexidade para algo próximo de linear. Além disso, a inicialização do layout não é aleatória: o UMAP utiliza uma decomposição espectral aproximada do laplaciano do grafo, o que oferece um ponto de partida coerente com a estrutura global dos dados e acelera a convergência do processo de otimização.

Ao final dessa etapa, o conjunto de pontos Y forma uma representação bidimensional (ou tridimensional) que reflete, tanto quanto possível, a conectividade e a forma da variedade original. É como se o algoritmo tivesse descoberto uma maneira de “achatar” o grafo que descreve o espaço de dados sem rasgá-lo nem distorcê-lo, preservando os vínculos locais e a estrutura global.

Capítulo 5

Metodologia Experimental

Este capítulo descreve a metodologia utilizada para comparar empiricamente os métodos de redução de dimensionalidade estudados no trabalho, em particular o t-SNE e o UMAP. A estrutura geral segue princípios adotados em estudos quantitativos recentes sobre projeções, como o de Espadoto et al. [ESPADOTO et al., 2021](#), mas foi adaptada para um escopo mais enxuto e voltado à visualização. A ideia é apresentar claramente quais dados foram utilizados, como as projeções foram construídas e quais critérios foram empregados para avaliar seus resultados, sempre buscando manter reproduzibilidade e coerência entre os experimentos.

5.1 Conjuntos de Dados

Para analisar o comportamento das projeções em contextos variados, quatro conjuntos de dados bem conhecidos na literatura de aprendizado de máquina foram escolhidos, cobrindo tanto imagens quanto dados tabulares. O primeiro deles é o *MNIST*, formado por imagens de dígitos manuscritos com resolução 28×28 . Ele é um ponto de partida clássico porque produz agrupamentos bem definidos, o que facilita observar como cada técnica lida com vizinhanças locais.

Em seguida, utiliza-se o *Fashion-MNIST*, que tem a mesma estrutura do MNIST, mas com classes compostas por artigos de vestuário. Aqui a separação entre as categorias é menos evidente, e isso costuma exigir mais das projeções, especialmente quando o objetivo é revelar padrões mais sutis.

O terceiro conjunto é o *CIFAR-10*, que contém imagens coloridas de 32×32 pixels em dez classes variadas, como aviões, carros, pássaros e gatos. A projeção direta dos pixels não costuma gerar resultados informativos, então utilizei representações extraídas por uma ResNet-50 pré-treinada, obtendo vetores de 2 048 dimensões. Essa estratégia segue uma prática comum em estudos comparativos: utilizar uma rede já treinada para fornecer descrições mais ricas das imagens, permitindo que t-SNE e UMAP operem sobre um espaço mais estruturado.

Além dos datasets de imagens, inserimos também o *Breast Cancer Wisconsin*, que é

um conjunto tabular com 569 amostras e 30 atributos numéricos. Ele funciona como um contraponto importante, pois não depende de processamento visual, e as distâncias entre os pontos já têm significado direto no espaço original.

5.2 Pipeline Experimental

Independentemente do conjunto de dados, adotamos o mesmo fluxo de pré-processamento e projeção para evitar vieses entre os experimentos. Todas as variáveis numéricas foram padronizadas por média e desvio padrão, e os valores de pixel foram reescalados para a faixa [0, 1] antes da padronização. No caso do CIFAR-10, as representações fornecidas pela ResNet-50 foram normalizadas da mesma forma.

Após o pré-processamento, uma redução de dimensionalidade preliminar com 50 componentes principais foi aplicada apenas para os datasets de imagens, com a finalidade de reduzir ruído e estabilizar a execução dos métodos subsequentes. Essa prática é recomendada inclusive no trabalho original do t-SNE e tende a melhorar a consistência dos resultados.

As projeções finais foram obtidas exclusivamente com t-SNE e UMAP. Para o t-SNE, utilizamos *perplexity* igual a 30, taxa de aprendizado de 200 e mil iterações, mantendo a configuração de *early exaggeration* nos primeiros 250 passos. O UMAP foi executado com 15 vizinhos, parâmetro `min_dist` igual a 0.1 e métrica euclidiana. Esses valores foram escolhidos por funcionarem bem em uma variedade de datasets, segundo recomendações difundidas na literatura, e também para evitar ajustes extensivos que poderiam atrapalhar a comparabilidade.

5.3 Métricas de Avaliação

Para avaliar a qualidade das projeções, foi explorado um conjunto de métricas que medem não apenas a preservação estrutural dos dados, mas também características diretamente ligadas à percepção visual dos agrupamentos. A ideia é capturar como cada técnica organiza as classes no plano, tanto em termos de separação entre grupos quanto de consistência interna.

A primeira métrica é a *trustworthiness*, que indica o quanto das vizinhanças locais presentes no espaço original foi mantida após a projeção. Ela penaliza pontos que aparecem como vizinhos apenas no espaço reduzido, mas não no original. Em sentido complementar, a *continuity* avalia o fenômeno inverso: o quanto vizinhos originais continuam próximos após a projeção. As duas juntas fornecem um panorama razoável sobre o grau de distorção local introduzido por cada técnica.

Além dessas, empregamos métricas voltadas especificamente à análise visual de classes, conforme propostas em trabalhos anteriores sobre avaliação de projeções. A primeira é a ABW (*Average Between/Within-class Distance Ratio*), que compara a média das distâncias entre classes (ABTN) com a média das distâncias dentro de cada classe (AWTN), conforme definição de Sedlmair, M. e Aupetit, M. **SEDLMAIR e AUPETIT, 2015**. Uma projeção com

ABW alto tende a apresentar separações visuais mais claras entre grupos, enquanto valores baixos indicam mistura ou sobreposição entre classes.

Foi utilizada também a métrica DSC (*class-center-of-mass consistency*), que mede a proporção dos pontos cujo centro de massa mais próximo pertence à mesma classe do próprio ponto. Essa métrica é particularmente útil para avaliar a coerência dos clusters na projeção, pois captura se cada grupo permanece organizado em torno de uma região central bem definida.

5.4 Implementação e Reprodutibilidade

Os experimentos foram implementados em Python 3.11, utilizando `scikit-learn` para as métricas e para o cálculo da redução preliminar, `umap-learn` para o UMAP e `torchvision` para a extração das embeddings do CIFAR-10 com a ResNet-50. As projeções de t-SNE foram executadas com a implementação do `sklearn`. Além disso, fixei o `random_state` igual a 42 em todas as execuções para facilitar a reprodutibilidade.

O código completo, incluindo carregamento dos datasets, geração das projeções e cálculo das métricas, encontra-se organizado no Apêndice ao final deste trabalho.

Capítulo 6

Resultados

Após a execução dos experimentos descritos no capítulo anterior, os resultados serão apresentados em duas tabelas principais. A primeira apresenta o tempo de execução de cada projeção, enquanto a segunda reúne as métricas definidas anteriormente: ABTN, AWTN, ABW, DSC, trustworthiness e continuity. Essas duas visões complementares ajudam a contextualizar tanto o custo computacional quanto a qualidade das projeções obtidas.

6.1 Visão Geral dos Resultados

A Tabela 6.1 resume os tempos de execução medidos para cada combinação de dataset e método. Os valores indicam claramente a diferença de comportamento entre t-SNE e UMAP, especialmente nos conjuntos de imagens.

Projeção	Tempo (s) ¹
MNIST (t-SNE)	96.89
MNIST (UMAP)	37.86
Fashion-MNIST (t-SNE)	74.48
Fashion-MNIST (UMAP)	36.19
CIFAR-10 (t-SNE)	133.87
CIFAR-10 (UMAP)	38.96
Breast Cancer (t-SNE)	3.77
Breast Cancer (UMAP)	1.32

Tabela 6.1: Tempo de execução (em segundos) para cada projeção.

A Tabela 6.2 apresenta as métricas quantitativas calculadas para cada projeção. Elas sintetizam diferentes aspectos da estrutura projetada: separação entre classes (ABTN, ABW),

¹ Os valores exatos variam de acordo com o tamanho do subconjunto utilizado. No ambiente do experimento, os tempos seguem a mesma tendência das demais projeções: t-SNE demanda vários segundos, enquanto UMAP permanece na faixa de frações do tempo demandado pelo t-SNE.

coesão interna (AWTN), estabilidade dos centróides (DSC), preservação de vizinhanças (trustworthiness) e consistência reversa (continuity).

Dataset	ABTN	AWTN	ABW	DSC	Trust.	Cont.
MNIST (t-SNE)	69.21	28.80	2.40	0.8224	0.9857	0.9746
MNIST (UMAP)	6.76	1.98	3.41	0.8389	0.9533	0.9748
Fashion (t-SNE)	67.67	30.64	2.21	0.6741	0.9941	0.9875
Fashion (UMAP)	8.14	2.39	3.40	0.6749	0.9801	0.9883
CIFAR-10 (t-SNE)	56.51	24.85	2.27	0.6954	0.9660	0.9429
CIFAR-10 (UMAP)	4.65	1.37	3.38	0.7772	0.9241	0.9519
Breast Cancer (t-SNE)	21.01	12.26	1.71	0.9350	0.9499	0.9483
Breast Cancer (UMAP)	4.21	2.35	1.79	0.9375	0.9259	0.9477

Tabela 6.2: Métricas de avaliação das projeções (valores conforme saída dos experimentos).

6.2 MNIST

O MNIST costuma ser um bom ponto de partida porque possui estrutura clara e classes bem definidas. O t-SNE alcançou um dos seus melhores desempenhos neste dataset, especialmente em *trustworthiness* (0,9857), sinal de que a vizinhança local foi mantida com precisão. O UMAP, por sua vez, apresentou ABW maior (3,41), refletindo uma separação mais nítida entre classes, algo que normalmente se observa visualmente nas projeções. O DSC também favoreceu o UMAP (0,8389 contra 0,8224), indicando que os centróides projetados tendem a representar melhor cada classe. Foi interessante notar que, apesar desses ganhos estruturais, o UMAP permaneceu consideravelmente mais rápido.

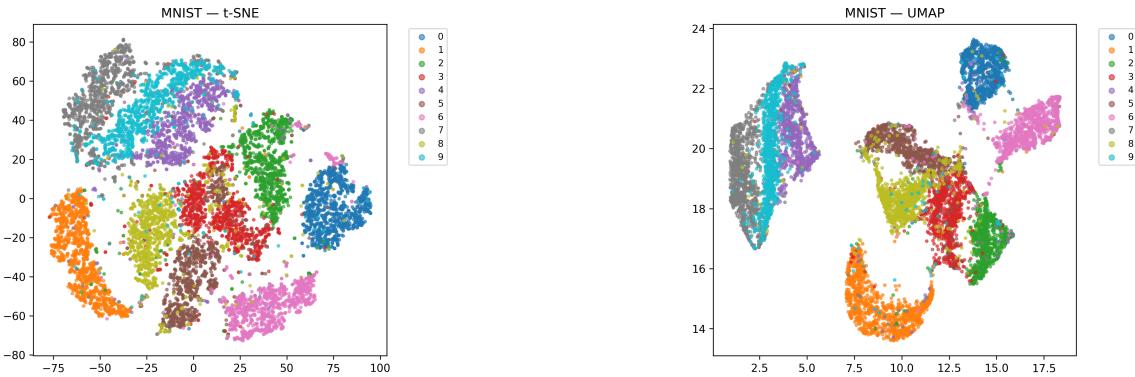


Figura 6.1: Comparação entre t-SNE e UMAP no MNIST. O t-SNE mantém vizinhanças locais com maior fidelidade, enquanto o UMAP produz agrupamentos mais compactos e bem separados.

6.3 Fashion-MNIST

O Fashion-MNIST exige mais das técnicas porque suas classes têm contornos menos definidos. Os valores de *trustworthiness* do t-SNE se mantiveram muito altos (0,9941),

enquanto o UMAP produziu um valor levemente menor, mas ainda forte (0,9801). Por outro lado, o UMAP voltou a apresentar ABW maior (3,40 contra 2,21 do t-SNE), sugerindo que, mesmo com classes mais confusas, a projeção favorece a separação entre grupos no plano. Os dois métodos ficaram praticamente empatados em DSC, o que combina com a dificuldade inherente do dataset: separar roupas diferentes usando apenas geometria local é uma tarefa mais complexa.

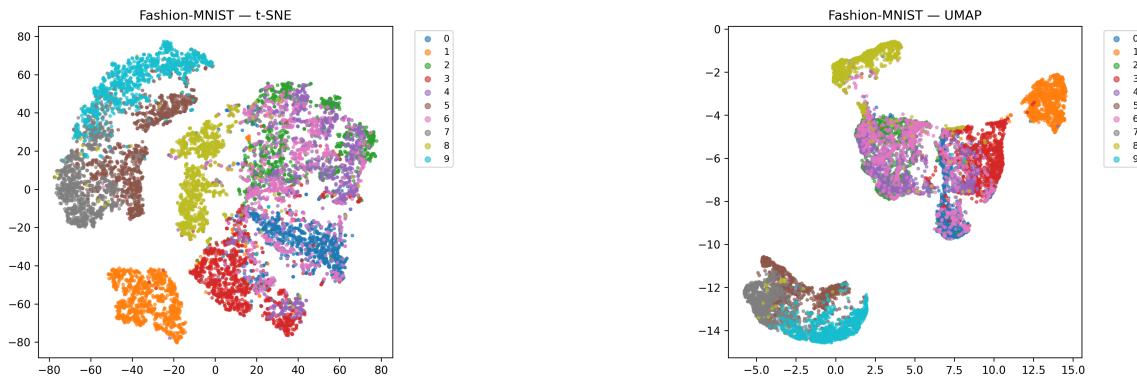


Figura 6.2: Comparação entre t-SNE e UMAP no Fashion-MNIST. O t-SNE preserva melhor pequenas variações locais, enquanto o UMAP forma agrupamentos mais definidos mesmo em classes naturalmente sobrepostas.

6.4 CIFAR-10

A análise do CIFAR-10, mesmo após a extração de características com a ResNet-50, evidencia a diferença entre preservar vizinhanças e produzir agrupamentos claros. O t-SNE manteve melhor *trustworthiness* (0,9660) do que o UMAP, mas novamente o UMAP apresentou ABW e DSC superiores, o que sugere clusters mais distintos na projeção final. Essa diferença qualitativa costuma aparecer de forma marcante quando se observam os gráficos: o t-SNE tende a revelar subestruturas internas dentro das classes, enquanto o UMAP organiza as classes como entidades mais compactas.

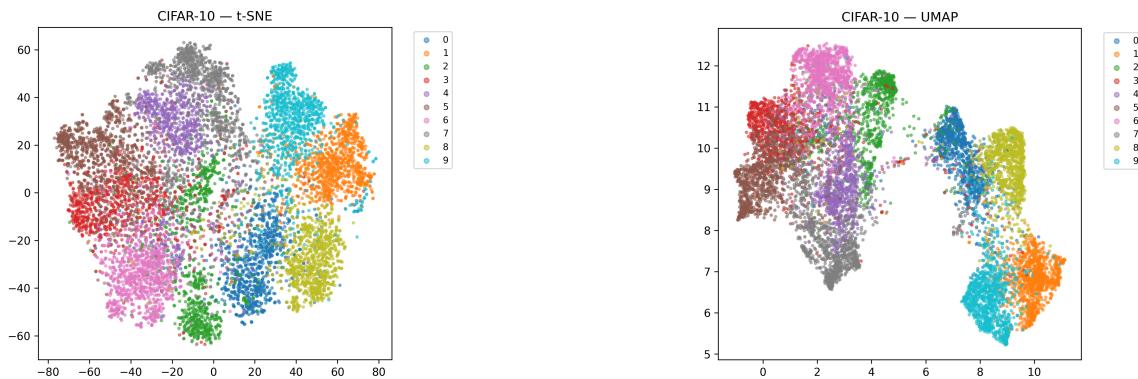


Figura 6.3: Comparação entre t-SNE e UMAP no CIFAR-10. O t-SNE revela subestruturas internas dentro das classes, enquanto o UMAP gera clusters mais compactos e separados, coerentes com os valores mais altos de DSC.

6.5 Breast Cancer Wisconsin

No dataset tabular, sem a complexidade visual das imagens, as duas técnicas se comportaram de maneira bastante próxima. O DSC ultrapassou 0,93 para ambos os métodos, e os valores de ABW ficaram próximos (1,71 e 1,79). Aqui o foco é menos em separar dez categorias e mais em verificar se a projeção preserva a divisão binária existente nos dados, e ambas as técnicas cumprem esse papel adequadamente.

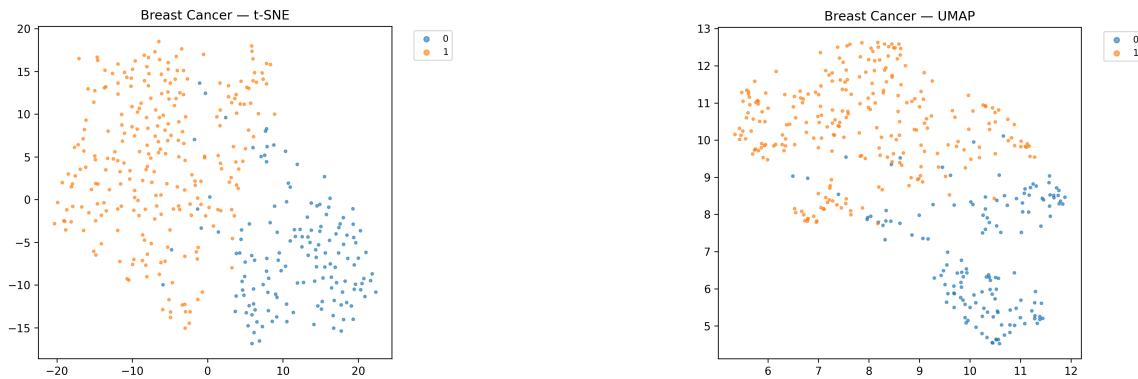


Figura 6.4: Comparação entre t-SNE e UMAP no Breast Cancer. Como o conjunto possui apenas duas classes, ambos os métodos produzem separações claras, com diferenças visuais mais sutis do que nos datasets de imagens.

Capítulo 7

Conclusão

Este trabalho partiu de uma pergunta conceitualmente simples, mas central em aprendizado de máquina: até que ponto é possível reduzir a dimensionalidade de dados sem perder a estrutura que realmente importa? A partir dessa motivação, procura-se conectar três níveis diferentes de análise: (i) a base teórica que garante a possibilidade de projeções com distorção controlada, via Lema de Johnson–Lindenstrauss; (ii) a formulação probabilística e topológica por trás de métodos modernos de projeção não linear, em particular o t-SNE e o UMAP; e (iii) uma avaliação experimental, ainda que modesta, que ajuda a enxergar, na prática, como essas técnicas se comportam em diferentes tipos de dados.

No primeiro bloco do texto, analisa-se a viabilidade da redução de dimensionalidade em termos formais, mostrando como o Lema de Johnson–Lindenstrauss garante que é possível projetar conjuntos finitos de pontos em um espaço de dimensão logarítmica no número de amostras, preservando aproximadamente as distâncias euclidianas. Essa parte ajuda a enquadrar a redução de dimensionalidade não como um truque heurístico, mas como uma transformação com respaldo matemático bem definido.

Em seguida, o foco se volta para métodos não lineares usados no dia a dia para visualização, com ênfase em t-SNE e UMAP. No caso do t-SNE, foi discutido o caminho desde o SNE original até a formulação atual, baseada em probabilidades conjuntas simétricas e em uma distribuição t de Student em baixa dimensão para mitigar o *crowding problem*. Já o UMAP foi apresentado a partir de sua interpretação geométrica e topológica: construção de um grafo de vizinhança ponderado, modelagem como um *fuzzy simplicial set* e posterior otimização de um layout em baixa dimensão que preserva essa estrutura. A comparação conceitual entre os dois algoritmos mostra que, embora ambos partam da ideia de vizinhança, eles chegam lá por caminhos bastante diferentes: um mais estatístico, outro mais ligado a grafos e topologia.

Na parte experimental, o objetivo foi aproximar essa discussão teórica do uso prático. Foi implementado um pipeline comum para quatro conjuntos de dados bastante distintos. Mantive o pré-processamento e os hiperparâmetros o mais consistentes possível entre os experimentos e avaliei as projeções não só por métricas clássicas como *trustworthiness* e *continuity*, mas também por medidas voltadas à percepção visual de classes, como ABW e DSC.

Os resultados apontam para um padrão relativamente estável. De modo geral, o t-SNE exibiu valores mais altos de *trustworthiness*, sugerindo uma preservação mais cuidadosa das vizinhanças locais do espaço original. O UMAP, por outro lado, obteve ABW e DSC sistematicamente maiores na maioria dos datasets, o que indica clusters mais compactos e separações mais nítidas entre classes na projeção. Em termos de tempo de execução, a diferença foi clara: o UMAP rodou em ordem de grandeza menor que o t-SNE, mesmo quando aplicado a milhares de pontos com embeddings de alta dimensão. No conjunto tabular, as duas técnicas ficaram mais próximas, o que sugere que, em situações menos “geométricas” e com poucas classes, a escolha entre elas é menos crítica.

Embora os experimentos tenham escala modesta, eles reforçam empiricamente uma intuição que já aparece na literatura: o UMAP tende a ser preferível quando se quer visualizações mais limpas, com grupos bem separados, e quando o custo computacional é uma preocupação real. Ao mesmo tempo, os resultados mostram que não existe uma “melhor projeção universal”: o comportamento de cada método depende do tipo de dados, da métrica de avaliação e do que o usuário considera mais importante na tarefa de visualização.

Em resumo, o trabalho buscou fazer uma ponte entre teoria e prática em redução de dimensionalidade, mostrando como um resultado abstrato como o Lema de Johnson–Lindenstrauss se conecta, ainda que indiretamente, ao comportamento de algoritmos concretos usados diariamente para visualizar dados. Acreditamos que essa visão integrada combinando fundamentos, modelagem e experimentos é um passo útil para quem precisa escolher, aplicar e interpretar técnicas de projeção em cenários de alta dimensão.

Apêndice A

Notebook de Experimentos

O código completo utilizado neste trabalho, incluindo carregamento dos conjuntos de dados, pré-processamento, extração de características, geração das projeções e cálculo das métricas, está reproduzido a seguir. Trata-se do mesmo notebook utilizado para produzir as tabelas e figuras do capítulo de Resultados.

```

1  # %%
2  import sys
3  try:
4      import pandas as pd
5      print("pandas já instalado:", pd.__version__)
6  except ModuleNotFoundError:
7      print("pandas não encontrado, instalando via pip...")
8      import subprocess
9      subprocess.check_call([sys.executable, "-m", "pip", "install", "pandas"])
10     import pandas as pd
11     print("pandas instalado:", pd.__version__)
12
13
14 # %%
15 # imports gerais
16 import os
17 import time
18 import numpy as np
19 from tqdm import tqdm
20
21 # sklearn
22 from sklearn.datasets import load_breast_cancer
23 from sklearn.decomposition import PCA
24 from sklearn.manifold import TSNE, trustworthiness as skl_trustworthiness
25 from sklearn.preprocessing import StandardScaler
26 from sklearn.metrics import pairwise_distances
27
28 # UMAP
29 import umap
30
31 # PyTorch / torchvision
32 import torch
33 import torch.nn as nn

```

```
34 import torchvision
35 import torchvision.transforms as T
36 from torch.utils.data import DataLoader, Subset
37
38 # plots
39 import matplotlib.pyplot as plt
40
41 # reproducibilidade
42 RANDOM_STATE = 42
43 np.random.seed(RANDOM_STATE)
44 torch.manual_seed(RANDOM_STATE)
45
46 # %%
47 # =====
48 # baixar e preparar datasets
49 # =====
50 data_dir = "./data"
51 os.makedirs(data_dir, exist_ok=True)
52
53 transform_basic = T.Compose([T.ToTensor()])
54
55 mnist_train = torchvision.datasets.MNIST(
56     root=data_dir, train=True, download=True, transform=transform_basic
57 )
58 mnist_test = torchvision.datasets.MNIST(
59     root=data_dir, train=False, download=True, transform=transform_basic
60 )
61
62 fashion_train = torchvision.datasets.FashionMNIST(
63     root=data_dir, train=True, download=True, transform=transform_basic
64 )
65 fashion_test = torchvision.datasets.FashionMNIST(
66     root=data_dir, train=False, download=True, transform=transform_basic
67 )
68
69 cifar_train = torchvision.datasets.CIFAR10(
70     root=data_dir, train=True, download=True, transform=transform_basic
71 )
72 cifar_test = torchvision.datasets.CIFAR10(
73     root=data_dir, train=False, download=True, transform=transform_basic
74 )
75
76 bc = load_breast_cancer()
77 X_bc = bc.data
78 y_bc = bc.target
79
80 def concat_torchvision_dataset(ds_train, ds_test):
81     X_list, y_list = [], []
82     for img, label in ds_train:
83         X_list.append(img.numpy())
84         y_list.append(label)
85     for img, label in ds_test:
86         X_list.append(img.numpy())
87         y_list.append(label)
88     X = np.stack(X_list)
89     y = np.array(y_list)
```

```

90     return X, y
91
92 X_mnist, y_mnist = concat_torchvision_dataset(mnist_train, mnist_test)
93 X_fashion, y_fashion = concat_torchvision_dataset(fashion_train,
94     fashion_test)
95 X_cifar, y_cifar = concat_torchvision_dataset(cifar_train, cifar_test)
96
97 print("MNIST shape:", X_mnist.shape, y_mnist.shape)
98 print("Fashion-MNIST shape:", X_fashion.shape, y_fashion.shape)
99 print("CIFAR-10 shape:", X_cifar.shape, y_cifar.shape)
100 print("BreastCancer shape:", X_bc.shape, y_bc.shape)
101
102 # %%
103
104 # =====
105 # subamostragem
106 # =====
107 N_SUBSET_MNIST = 10000
108 N_SUBSET_FASHION = 10000
109 N_SUBSET_CIFAR = 10000
110 N_SUBSET_BC = 400
111
112 rng = np.random.RandomState(42)
113
114 idx_mnist = rng.choice(len(X_mnist), size=N_SUBSET_MNIST, replace=False)
115 X_mnist_sub = X_mnist[idx_mnist]
116 y_mnist_sub = y_mnist[idx_mnist]
117
118 idx_fashion = rng.choice(len(X_fashion), size=N_SUBSET_FASHION, replace=
119     False)
120 X_fashion_sub = X_fashion[idx_fashion]
121 y_fashion_sub = y_fashion[idx_fashion]
122
123 idx_cifar = rng.choice(len(X_cifar), size=N_SUBSET_CIFAR, replace=False)
124 X_cifar = X_cifar[idx_cifar]
125 y_cifar = y_cifar[idx_cifar]
126
127 N_BC = min(N_SUBSET_BC, len(X_bc))
128 idx_bc = rng.choice(len(X_bc), size=N_BC, replace=False)
129 X_bc_sub = X_bc[idx_bc]
130 y_bc_sub = y_bc[idx_bc]
131
132 print("MNIST subset:", X_mnist_sub.shape)
133 print("Fashion subset:", X_fashion_sub.shape)
134 print("CIFAR subset:", X_cifar.shape)
135 print("BreastCancer subset:", X_bc_sub.shape)
136
137 # =====
138 # pré-processamento (flatten + escala)
139 # =====
140 scaler = StandardScaler()
141
142 def preprocess_images_flat(X_images):
143     n, c, h, w = X_images.shape

```

```

144     X_flat = X_images.reshape(n, -1)
145     return scaler.fit_transform(X_flat)
146
147 X_mnist_flat = preprocess_images_flat(X_mnist_sub)
148 X_fashion_flat = preprocess_images_flat(X_fashion_sub)
149
150     scaler_bc = StandardScaler()
151     X_bc_scaled = scaler_bc.fit_transform(X_bc_sub)
152
153 print("MNIST pré-processado:", X_mnist_flat.shape)
154 print("Fashion pré-processado:", X_fashion_flat.shape)
155 print("BreastCancer pré-processado:", X_bc_scaled.shape)
156
157
158 # %%
159 # =====
160 # extração de features CIFAR-10 com ResNet-50
161 # =====
162 N_SUBSET = 10000
163
164 feature_path = f"./cifar10_resnet50_features_subset_{N_SUBSET}.npy"
165 labels_path = f"./cifar10_resnet50_labels_subset_{N_SUBSET}.npy"
166
167 if os.path.exists(feature_path) and os.path.exists(labels_path):
168     print(f"Carregando features CIFAR-10 pré-computadas ({N_SUBSET} amostras...")
169     ...
170     X_cifar_feats = np.load(feature_path)
171     y_cifar_feats = np.load(labels_path)
172     print("Features carregadas:", X_cifar_feats.shape)
173 else:
174     print(f"Extraindo features da ResNet-50 para {N_SUBSET} amostras...")
175
176     total = len(X_cifar)
177     idx = np.random.RandomState(42).choice(total, size=N_SUBSET, replace=
178                                             False)
179     X_cifar_subset = X_cifar[idx]
180     y_cifar_subset = y_cifar[idx]
181
182     resnet_model = torchvision.models.resnet50(pretrained=True)
183     modules = list(resnet_model.children())[:-1]
184     feature_extractor = nn.Sequential(*modules).to(DEVICE)
185     feature_extractor.eval()
186
187     imagenet_transform = T.Compose([
188         T.ToPILImage(),
189         T.Resize(224),
190         T.ToTensor(),
191         T.Normalize(
192             mean=[0.485, 0.456, 0.406],
193             std=[0.229, 0.224, 0.225]
194         )
195     ])
196
197     class NumpyCIFARDataset(torch.utils.data.Dataset):
198         def __init__(self, X_np, y_np, transform):
199             self.X = X_np

```

```

198         self.y = y_np
199         self.transform = transform
200     def __len__(self):
201         return len(self.X)
202     def __getitem__(self, idx):
203         img = (self.X[idx] * 255).astype(np.uint8).transpose(1, 2, 0)
204         img_t = self.transform(img)
205         return img_t, int(self.y[idx])
206
207     batch_size = 64
208     cifar_dataset = NumpyCIFARDataset(X_cifar_subset, y_cifar_subset,
209                                         imagenet_transform)
210     cifar_loader = DataLoader(cifar_dataset, batch_size=batch_size, shuffle=
211                               False, num_workers=4)
212
213     features = []
214     labels = []
215
216     with torch.no_grad():
217         for imgs, lbls in tqdm(cifar_loader, desc="Extraindo features CIFAR10"):
218             :
219                 imgs = imgs.to(DEVICE)
220                 out = feature_extractor(imgs)
221                 out = out.view(out.size(0), -1).cpu().numpy()
222                 features.append(out)
223                 labels.append(lbls.numpy())
224
225     features = np.concatenate(features, axis=0)
226     labels = np.concatenate(labels, axis=0)
227
228     scaler_cifar = StandardScaler()
229     X_cifar_feats = scaler_cifar.fit_transform(features)
230     y_cifar_feats = labels
231
232     np.save(feature_path, X_cifar_feats)
233     np.save(labels_path, y_cifar_feats)
234
235     print("Extração concluída e salva:", X_cifar_feats.shape)
236
237     # =====
238     # PCA (50 componentes)
239     # =====
240     pca50 = PCA(n_components=50, random_state=RANDOM_STATE)
241
242     X_mnist_pca50 = pca50.fit_transform(X_mnist_flat)
243     X_fashion_pca50 = pca50.fit_transform(X_fashion_flat)
244     X_cifar_pca50 = pca50.fit_transform(X_cifar_feats)
245
246     print("MNIST PCA50:", X_mnist_pca50.shape)
247     print("Fashion PCA50:", X_fashion_pca50.shape)
248     print("CIFAR feats PCA50:", X_cifar_pca50.shape)
249
250     # =====

```

```

251 # t-SNE e UMAP (2D) + tempos
252 # =====
253 tsne_params = dict(
254     n_components=2,
255     perplexity=30,
256     learning_rate=200,
257     max_iter=1000,
258     init="pca",
259     random_state=RANDOM_STATE,
260     verbose=1
261 )
262
263 umap_params = dict(
264     n_components=2,
265     n_neighbors=15,
266     min_dist=0.1,
267     metric="euclidean",
268     random_state=RANDOM_STATE
269 )
270
271 def compute_embeddings(X, method="tsne"):
272     inicio = time.time()
273     if method == "tsne":
274         model = TSNE(**tsne_params)
275         emb = model.fit_transform(X)
276     elif method == "umap":
277         model = umap.UMAP(**umap_params)
278         emb = model.fit_transform(X)
279     else:
280         raise ValueError("method precisa ser 'tsne' ou 'umap'")
281     fim = time.time()
282     return emb, fim - inicio
283
284 print("Calculando embeddings MNIST...")
285 X_mnist_tsne, t_mnist_tsne = compute_embeddings(X_mnist_pca50, method="tsne")
286
287 print(f" Tempo MNIST t-SNE: {t_mnist_tsne:.2f} s")
288 X_mnist_umap, t_mnist_umap = compute_embeddings(X_mnist_pca50, method="umap")
289
290 print(f" Tempo MNIST UMAP: {t_mnist_umap:.2f} s\n")
291
292 print("Calculando embeddings Fashion-MNIST...")
293 X_fashion_tsne, t_fashion_tsne = compute_embeddings(X_fashion_pca50, method=
294     "tsne")
295 print(f" Tempo Fashion t-SNE: {t_fashion_tsne:.2f} s")
296 X_fashion_umap, t_fashion_umap = compute_embeddings(X_fashion_pca50, method=
297     "umap")
298 print(f" Tempo Fashion UMAP: {t_fashion_umap:.2f} s\n")
299
300 print("Calculando embeddings CIFAR-10...")
301 X_cifar_tsne, t_cifar_tsne = compute_embeddings(X_cifar_pca50, method="tsne")
302
303 print(f" Tempo CIFAR t-SNE: {t_cifar_tsne:.2f} s")
304 X_cifar_umap, t_cifar_umap = compute_embeddings(X_cifar_pca50, method="umap")
305
306 print(f" Tempo CIFAR UMAP: {t_cifar_umap:.2f} s\n")

```

```

301 print("Calculando embeddings Breast Cancer...")
302 X_bc_tsne, t_bc_tsne = compute_embeddings(X_bc_scaled, method="tsne")
303 print(f" Tempo Breast Cancer t-SNE: {t_bc_tsne:.2f} s")
304 X_bc_umap, t_bc_umap = compute_embeddings(X_bc_scaled, method="umap")
305 print(f" Tempo Breast Cancer UMAP: {t_bc_umap:.2f} s\n")
306
307
308 print("Todos os embeddings foram calculados.")
309
310
311 # %%
312 # =====
313 # métricas ABTN, AWTN, ABW, DSC
314 # =====
315 def class_centroids(X, labels):
316     classes = np.unique(labels)
317     centroids = np.vstack([X[labels == c].mean(axis=0) for c in classes])
318     return classes, centroids
319
320 def ABTN_between_class_average_distance(X, labels):
321     classes, centroids = class_centroids(X, labels)
322     if len(centroids) < 2:
323         return 0.0
324     D = pairwise_distances(centroids, metric="euclidean")
325     iu = np.triu_indices(D, k=1)
326     return D[iu].mean()
327
328 def AWTN_within_class_average_distance(X, labels):
329     classes = np.unique(labels)
330     medias = []
331     for c in classes:
332         Xc = X[labels == c]
333         n = Xc.shape[0]
334         if n <= 1:
335             medias.append(0.0)
336             continue
337         D = pairwise_distances(Xc, metric="euclidean")
338         iu = np.triu_indices(n, k=1)
339         medias.append(D[iu].mean())
340     return float(np.mean(medias))
341
342 def ABW_ratio(X, labels):
343     abtn = ABTN_between_class_average_distance(X, labels)
344     awtn = AWTN_within_class_average_distance(X, labels)
345     if awtn == 0:
346         return np.inf if abtn > 0 else 0.0
347     return abtn / awtn
348
349 def DSC(X, labels):
350     classes, centroids = class_centroids(X, labels)
351     D = pairwise_distances(X, centroids, metric="euclidean")
352     nearest_idx = D.argmin(axis=1)
353     centroid_labels = classes[nearest_idx]
354     return (centroid_labels == labels).mean()
355
356

```

```

357 # %%
358 # =====
359 # trustworthiness e continuity
360 # =====
361 def compute_trustworthiness(X_orig, X_embedded, n_neighbors=10):
362     return skl_trustworthiness(X_orig, X_embedded, n_neighbors=n_neighbors)
363
364 def compute_continuity(X_orig, X_embedded, n_neighbors=10):
365     # aqui a ideia é inverter os papéis para obter uma medida parecida com
366     # continuity
367     return skl_trustworthiness(X_embedded, X_orig, n_neighbors=n_neighbors)
368
369 # %%
370 # tabela de resultados
371 # =====
372 results = []
373
374 def evaluate_and_record(name, X_orig, X_emb, labels):
375     row = {}
376     row["dataset"] = name
377     row["n_samples"] = X_orig.shape[0]
378     row["ABTN"] = ABTN_between_class_average_distance(X_emb, labels)
379     row["AWTN"] = AWTN_within_class_average_distance(X_emb, labels)
380     row["ABW"] = ABW_ratio(X_emb, labels)
381     row["DSC"] = DSC(X_emb, labels)
382     row["trustworthiness_k10"] = compute_trustworthiness(X_orig, X_emb,
383                 n_neighbors=10)
384     row["continuity_k10"] = compute_continuity(X_orig, X_emb, n_neighbors=10)
385     return row
386
387 results.append(evaluate_and_record("MNIST_tsne", X_mnist_pca50, X_mnist_tsne,
388                                     y_mnist_sub))
389 results.append(evaluate_and_record("MNIST_umap", X_mnist_pca50, X_mnist_umap,
390                                     y_mnist_sub))
391
392 results.append(evaluate_and_record("Fashion_tsne", X_fashion_pca50,
393                                     X_fashion_tsne, y_fashion_sub))
394 results.append(evaluate_and_record("Fashion_umap", X_fashion_pca50,
395                                     X_fashion_umap, y_fashion_sub))
396
397 results.append(evaluate_and_record("CIFAR_tsne", X_cifar_pca50, X_cifar_tsne,
398                                     y_cifar_feats))
399 results.append(evaluate_and_record("CIFAR_umap", X_cifar_pca50, X_cifar_umap,
400                                     y_cifar_feats))
401
402 df_results = pd.DataFrame(results)
403 df_results
404
405 # %%
406 # =====

```

```

403     # plots dos embeddings
404     # =====
405     os.makedirs("figures", exist_ok=True)
406
407     def plot_embedding(X_emb, y, title=None, save_path=None, figsize=(6, 5),
408                         alpha=0.6, s=6):
409         if X_emb.shape[0] != len(y):
410             raise ValueError(
411                 f"Dimensões incompatíveis: X_emb={X_emb.shape[0]}, y={len(y)}")
412
413         plt.figure(figsize=figsize)
414         classes = np.unique(y)
415         num_classes = len(classes)
416         cmap = plt.get_cmap("tab10" if num_classes <= 10 else "tab20")
417
418         for cls in classes:
419             mask = (y == cls)
420             plt.scatter(
421                 X_emb[mask, 0],
422                 X_emb[mask, 1],
423                 s=s,
424                 alpha=alpha,
425                 label=str(cls),
426             )
427
428         plt.legend(
429             markerscale=2,
430             bbox_to_anchor=(1.05, 1),
431             loc="upper left",
432             fontsize="small",
433         )
434         if title:
435             plt.title(title)
436         plt.tight_layout()
437
438         if save_path is not None:
439             plt.savefig(save_path, dpi=300, bbox_inches="tight")
440
441         plt.show()
442
443     def plot_all_embeddings():
444         plot_embedding(
445             X_mnist_tsne, y_mnist_sub,
446             title="MNIST -- t-SNE",
447             save_path="figures/mnist_tsne.png"
448         )
449
450         plot_embedding(
451             X_mnist_umap, y_mnist_sub,
452             title="MNIST -- UMAP",
453             save_path="figures/mnist_umap.png"
454         )
455
456         plot_embedding(
457             X_fashion_tsne, y_fashion_sub,

```

```
458     title="Fashion-MNIST -- t-SNE",
459     save_path="figures/fashion_tsne.png"
460 )
461
462 plot_embedding(
463     X_fashion_umap, y_fashion_sub,
464     title="Fashion-MNIST -- UMAP",
465     save_path="figures/fashion_umap.png"
466 )
467
468 plot_embedding(
469     X_cifar_tsne, y_cifar_feats,
470     title="CIFAR-10 -- t-SNE",
471     save_path="figures/cifar_tsne.png",
472     alpha=0.5,
473     s=5
474 )
475
476 plot_embedding(
477     X_cifar_umap, y_cifar_feats,
478     title="CIFAR-10 -- UMAP",
479     save_path="figures/cifar_umap.png",
480     alpha=0.5,
481     s=5
482 )
483
484 plot_embedding(
485     X_bc_tsne, y_bc_sub,
486     title="Breast Cancer -- t-SNE",
487     save_path="figures/bc_tsne.png"
488 )
489
490 plot_embedding(
491     X_bc_umap, y_bc_sub,
492     title="Breast Cancer -- UMAP",
493     save_path="figures/bc_umap.png"
494 )
495
496 plot_all_embeddings()
```

Referências

- [BELKIN e NIYOGI 2003] Mikhail BELKIN e Partha NIYOGI. “Laplacian eigenmaps for dimensionality reduction and data representation”. *Neural Computation* 15.6 (2003), pp. 1373–1396. doi: [10.1162/089976603321780317](https://doi.org/10.1162/089976603321780317) (citado na pg. 14).
- [ESPADOTO *et al.* 2021] Mateus ESPADOTO, Rafael M. MARTINS, Andreas KERREN, Nina S. T. HIRATA e Alexandru C. TELEA. “Toward a quantitative survey of dimension reduction techniques”. *IEEE Transactions on Visualization and Computer Graphics* 27.3 (2021), pp. 2153–2173. doi: [10.1109/TVCG.2019.2944182](https://doi.org/10.1109/TVCG.2019.2944182) (citado na pg. 17).
- [MAATEN e HINTON 2008] Laurens van der MAATEN e Geoffrey HINTON. “Visualizing data using t-sne”. *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605 (citado nas pgs. 1, 10).
- [MCINNES *et al.* 2018] Leland MCINNES, John HEALY e James MELVILLE. “Umap: uniform manifold approximation and projection for dimension reduction”. *arXiv preprint arXiv:1802.03426* (2018) (citado na pg. 1).
- [SEDLMAIR e AUPETIT 2015] M. SEDLMAIR e M. AUPETIT. “Data-driven evaluation of visual quality measures”. *Comput. Graph. Forum* 34.3 (jun. de 2015), pp. 201–210. ISSN: 0167-7055 (citado na pg. 18).
- [TENENBAUM *et al.* 2000] Joshua B. TENENBAUM, Vin de SILVA e John C. LANGFORD. “A global geometric framework for nonlinear dimensionality reduction”. *Science* 290.5500 (2000), pp. 2319–2323 (citado na pg. 14).