

UMAP

June 22, 2025

UMAP

ML2: AI Concepts and Algorithms (SS2025)
Faculty of Computer Science and Applied Mathematics
University of Applied Sciences Technikum Wien



Lecturer: Rosana de Oliveira Gomes
Authors: B. Knapp, M. Blaickner, S. Rezagholi, R.O. Gomes



Good [morning/afternoon], everyone.

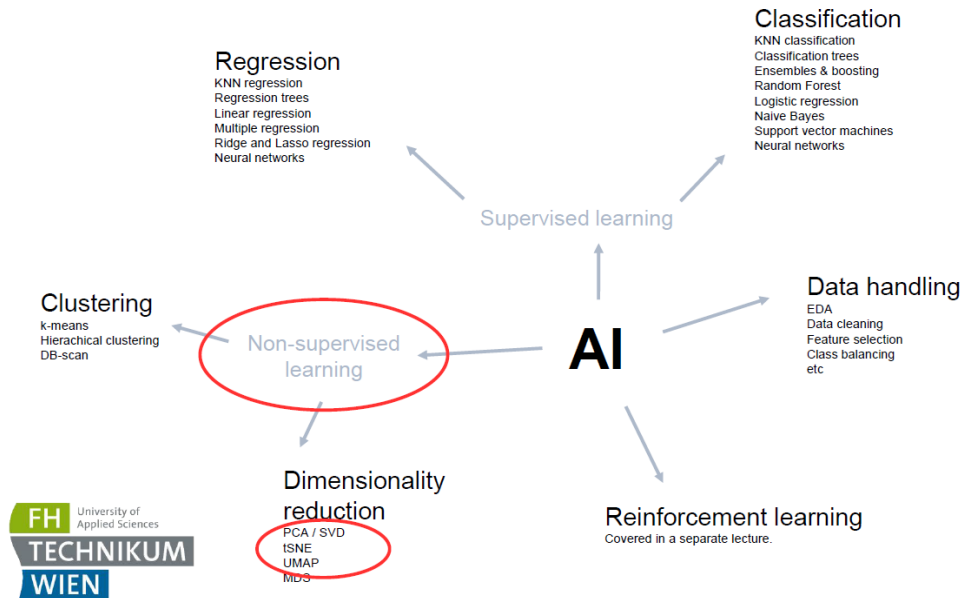
Welcome to today's presentation on **UMAP**, which stands for **Uniform Manifold Approximation and Projection**. This topic is part of the **ML2: AI Concepts and Algorithms** course for the Summer Semester 2025, offered by the **Faculty of Computer Science and Applied Mathematics** at the **University of Applied Sciences Technikum Wien**.

This presentation was prepared by:

- **B. Knapp**,
- **M. Blaickner**,
- **S. Rezagholi**, and
- **R.O. Gomes**, under the guidance of our lecturer, **Rosana de Oliveira Gomes**.

Let's now begin exploring the foundations, motivations, and applications of UMAP in the context of machine learning and dimensionality reduction.

Please go ahead and share the next slide.



Let's begin by placing UMAP within the broader context of Artificial Intelligence.

This slide shows an overview of **key areas within AI**, divided into categories like:

- **Supervised Learning**, which includes:
 - **Regression** (e.g., Linear Regression, Ridge/Lasso, Neural Networks)
 - **Classification** (e.g., Logistic Regression, SVM, Random Forest, Boosting)
- **Reinforcement Learning**, which is covered separately.
- **Data Handling**: Tasks like exploratory data analysis (EDA), cleaning, feature selection, and class balancing.

Now let's focus on the **Non-Supervised Learning** branch, which is marked in red.

It includes two main areas:

1. **Clustering** (e.g., K-means, Hierarchical Clustering, DBSCAN)
2. **Dimensionality Reduction** – and that's where **UMAP** comes in.

UMAP is listed alongside **PCA**, **SVD**, and **t-SNE** as a technique used to **reduce the dimensionality** of complex datasets. These methods help us visualize and analyze high-dimensional data by projecting it into a lower-dimensional space while preserving meaningful structure.

So in this presentation, we'll explore how UMAP achieves this goal—and why it is often preferred over other methods like PCA or t-SNE.

Let's move on to the next slide.

Dimensionality Reduction Recap

Dimensionality curse: challenges and complexities that arise when working with high-dimensional data, where the number of features or variables is significantly large

High dimensional systems: e.g. genomics, environmental science, NLP, customer segmentation.

Dimensionality reduction transforms data from high-dimensional space into a low-dimensional space while **preserving as much of the dataset's original information as possible**.



How to convert a multi-dimensional dataset into a small dimension visualization?

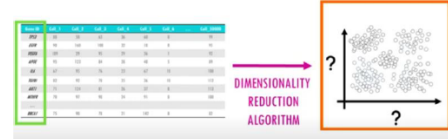


Image Source: <https://www.kdnuggets.com/2019/04/dimensionality-reduction.html>

Common algorithms

- Principle Component Analysis
- T-SNE
- UMAP

Before we dive deeper into UMAP, let's quickly recap what **Dimensionality Reduction** is all about.

0.0.1 The Problem: The Curse of Dimensionality

As we work with datasets that have **many features or variables**, we run into what's called the **curse of dimensionality**. This refers to the **increasing complexity and sparsity** of data as the number of dimensions grows—making it harder for machine learning algorithms to operate effectively.

0.0.2 High-Dimensional Data Examples:

Some real-world systems that involve high-dimensional data include:

- **Genomics** (thousands of genes),
 - **Environmental science** (multiple sensor readings),
 - **Natural Language Processing** (word embeddings in hundreds of dimensions),
 - **Customer segmentation** (many demographic and behavioral features).
-

0.0.3 What Is Dimensionality Reduction?

Dimensionality reduction is the process of:

Transforming high-dimensional data into a low-dimensional space, while trying to preserve the original structure and relationships of the data as much as possible.

This is especially useful for:

- **Visualization** (e.g., plotting in 2D/3D),
- **Noise reduction**,
- **Speeding up computations**,
- **Improving generalization**.

0.0.4 Common Algorithms:

- **PCA (Principal Component Analysis)** – a linear method that projects data along directions of maximum variance.
- **t-SNE** – a non-linear method that focuses on preserving local structure.
- **UMAP** – the focus of today’s talk, designed to preserve both local and global structure more efficiently than t-SNE.

Now that we’ve framed the problem, let’s take a closer look at **how UMAP works**. Please go ahead with the next slide.

PCA Recap: Principal Component Analysis

PCA captures the essence of the data into few **principal components** (usually 2-5 PCs).

Principal Components:

- Summarize patterns or variations in the dataset.
- Are ranked and independent from each other.
- PCs are constructed as a *linear combination* or mixture of the original variables.

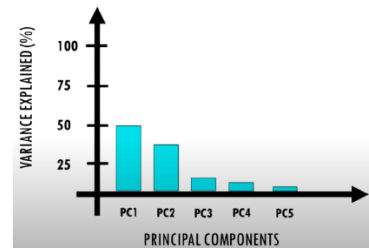



Image Source: 



	Lifespan	PC1	PC2	PC3	PC4	PC5
Person 1	82	-1	3	-1	4	4
Person 2	73	2	4	2	5	5
Person 3	95	3	2	4	2	2
Person 4	92	4	4	5	-4	-4
Person 5	87	5	5	2	2	5
Person 6	65	2	5	-4	3	2
Person 7	93	-4	-4	5	5	-4
Person 8	80	-3	-6	-6	2	5
...						
Person 20	72	8	-3	-6	-3	-6

	PC1	PC2	PC3	PC4	PC5
Height	-1	3	-1	4	4
Average heart rate	9	7	5	-4	-4
BMI	10	6.5	2	2	5
Cholesterol levels	9	5	-4	3	2
Average cigarettes/day	7	2	5	5	-4
Greasy diet	10	5	-6	2	5
Frequent exercise	-5	-6	8	1	9
Eye colour	0.1	0.3	0.1	0.3	0.3
Teeth-brushing habits	0.2	0.2	0.2	0.2	0.2

Before we fully explore UMAP, let’s briefly recap **Principal Component Analysis (PCA)**, one of the most well-established dimensionality reduction techniques.

0.0.5 What is PCA?

PCA captures the essence of the data by projecting it into a smaller set of **principal components (PCs)**—usually just **2 to 5**.

These components are:

- **Linear combinations** of the original features,

- **Orthogonal** (statistically independent),
 - And **ranked** by the amount of variance they explain in the data.
-

0.0.6 What Are Principal Components?

- **Summarize patterns:** Each PC captures a direction of maximal variance in the data.
 - **Uncorrelated:** PCs are mathematically constructed to be independent of each other.
 - **Constructed from original variables:** Each PC is a weighted sum of the original features (as seen in the bottom-right table).
-

0.0.7 On the Right Side:

- The bar chart shows that **PC1 explains the most variance**, followed by PC2, and so on. Typically, the first 2–3 PCs can already capture most of the dataset's information.
 - The tables demonstrate how **original features** are transformed into PCs (loadings) and how **individual samples** are then described using those PCs.
-

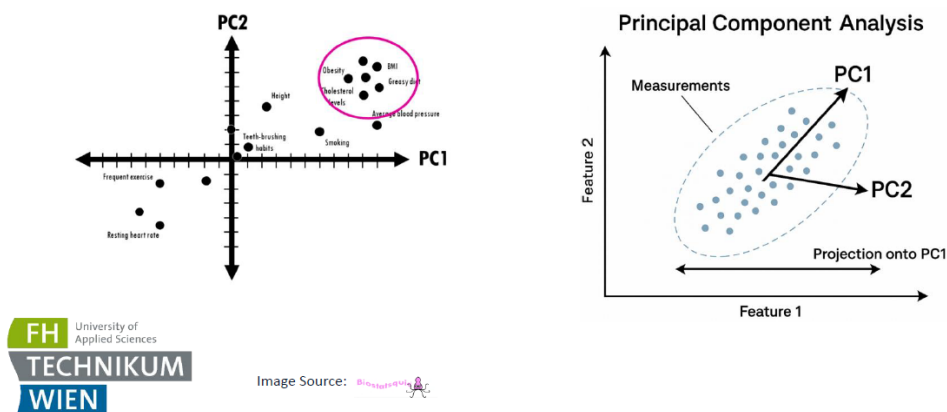
PCA is very efficient, fast, and useful when the data relationships are **linear**. But when we deal with **non-linear structures**, PCA falls short.

This limitation leads us to more powerful, **non-linear** methods—such as **t-SNE** and especially **UMAP**, which we'll be exploring next.

Let's continue with the next slide.

PCA Recap: Principal Component Analysis

The contribution of the variables to each principal component (PC) is indicated by their *loadings*.



Continuing with our recap of **Principal Component Analysis**, this slide gives us a **visual intuition** behind how PCA works.

0.0.8 What Are Loadings?

The **contribution of each variable** to the principal components is indicated by their **loadings**.

- Think of loadings as **directional arrows** that show how much a variable influences a particular principal component.
 - Variables that point in the **same direction** tend to be **positively correlated**.
 - Those pointing in **opposite directions** are **negatively correlated**.
-

0.0.9 Interpreting the Left Plot:

- This is a **loading plot** that visualizes the contributions of variables to PC1 and PC2.
 - For example, variables like **Obesity**, **BMI**, and **Greasy diet** are clustered together in the upper-right quadrant, suggesting they all load heavily on PC1 and are correlated with each other.
 - On the other hand, **Resting heart rate** and **Frequent exercise** are in the bottom-left quadrant, potentially anti-correlated with the first group.
-

0.0.10 Interpreting the Right Plot:

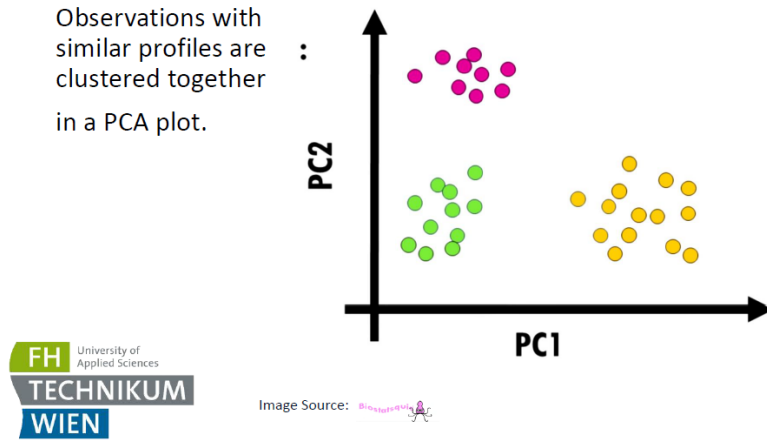
- Here we see how PCA projects the original high-dimensional data onto a **2D plane** spanned by PC1 and PC2.
 - Each dot represents a sample (or observation).
 - The **dashed ellipse** shows the spread of the data along the directions of greatest variance—PC1 and PC2.
-

So PCA not only helps reduce dimensions, but also helps **reveal structure** in the data—by identifying patterns and correlations among variables.

That said, PCA assumes **linear relationships**. If the structure in your data is non-linear, PCA may miss it. This is where non-linear methods like **UMAP** truly shine.

Let's proceed to the next slide to begin exploring UMAP in detail.

PCA Recap: Principal Component Analysis



This final PCA recap slide helps illustrate the **practical outcome** of a PCA transformation:

0.0.11 What Does the Plot Show?

- Each **dot** represents one observation (or data point).
 - The two axes, **PC1** and **PC2**, are the first two principal components, capturing the most variance in the dataset.
 - Points that are **close together** in this 2D space have **similar profiles** across the original high-dimensional features.
-

0.0.12 Cluster Interpretation:

- PCA is often used **before clustering** (e.g., K-means, DBSCAN), or to **visualize clusters** in a simpler space.
 - Here, we see three natural groupings—marked by color—that show **how PCA can reveal structure** in the data.
 - This type of plot is often used in **bioinformatics**, **customer segmentation**, or **market analysis** to visually identify **subgroups** in a population.
-

0.0.13 Summary of PCA So Far:

- PCA is a **linear**, fast, and powerful tool.
- It can capture **global patterns** and provide **interpretable visualizations**.
- But... it assumes **linear relationships**, and can **miss non-linear manifolds** or **local structures** in the data.

That's exactly where **UMAP** comes in—offering **non-linear, topology-preserving dimensionality reduction**.

Let's now turn our attention to UMAP and see what makes it such a powerful alternative. Please proceed with the next slide.

t-SNE Recap: T-distributed Stochastic Neighbour Embedding

[Visualizing Data using t-SNE Laurens van der Maaten, Geoffrey Hinton: 9\(86\):2579–2605, 2008.](#)

Dimensionality reduction based on similarities across data points.
Allows for **non-linearity** across variables.

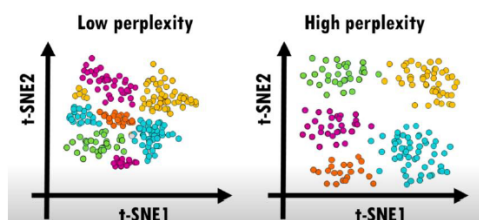

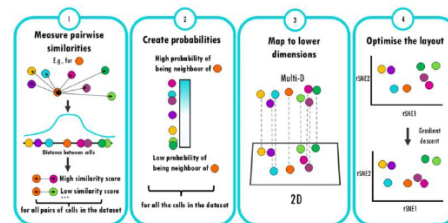


Image Source: 



Perplexity hyperparameter:
how many neighbour each data points is considering when building a similarity map



Algorithm:

1. Measure similarity score for all data-pairs in the dataset.
2. Calculate likelihood that two data points are neighbours.
3. Map the data to lower dimensions.
4. Make the distribution of similarities in the lower dimension space match the original distribution as closely as possible (with gradient descent).

Before we introduce UMAP, let's quickly recap **t-SNE** — one of the most popular non-linear dimensionality reduction techniques.

0.0.14 What is t-SNE?

t-SNE stands for **t-distributed Stochastic Neighbor Embedding**. It was introduced by **van der Maaten and Hinton (2008)** and is designed to project high-dimensional data into a **2D or 3D space** for visualization, especially when the data has **non-linear structure**.

0.0.15 How Does It Work?

1. **Measure pairwise similarities** between all points in high-dimensional space.
2. Convert those similarities into **probabilities**: how likely are two points to be neighbors?
3. Project the data to lower dimensions (e.g., 2D).
4. Use **gradient descent** to ensure the **low-dimensional similarities** match the **original high-dimensional ones**.

0.0.16 Key Parameter: Perplexity

- Think of **perplexity** as a knob that controls how many neighbors each point should consider.

- **Low perplexity** emphasizes **local structure** (tight clusters).
- **High perplexity** gives a more **global view** but may blend clusters.

This is visualized in the plots on the left:

- The left panel (low perplexity) shows tight clusters but possibly overemphasized local separation.
- The right panel (high perplexity) offers a more blended distribution.

0.0.17 Pros and Cons

Strengths:

- Captures complex, non-linear structures.
- Excellent for **visualizing clusters**.

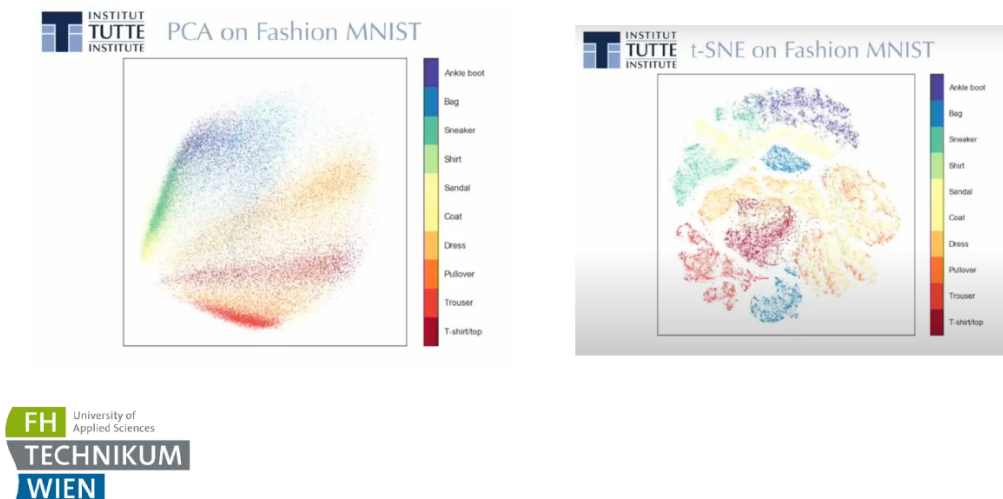
Limitations:

- Doesn't preserve global distances well.
- Results can **vary** between runs (non-deterministic).
- **Computationally expensive** on large datasets.

This sets the stage for **UMAP**, which retains the strengths of t-SNE but aims to **address these limitations**—being faster, more consistent, and more faithful to both **local and global** structure.

Let's move on to UMAP in the next slide.

PCA vs t-SNE



This side-by-side comparison shows the **visual impact of PCA vs. t-SNE** on the **Fashion MNIST dataset**, a classic benchmark in machine learning that includes grayscale images of clothing items across 10 categories.

0.0.18 PCA Plot (Left):

- The **PCA projection** is smooth and continuous.
 - Data points are spread along major directions of **global variance**.
 - However, **clusters are not well separated**—many categories blend together.
 - It reflects **linear structure** and is good for **quick overview** or **preprocessing**, but lacks fine detail.
-

0.0.19 t-SNE Plot (Right):

- The **t-SNE projection** shows **clearly defined clusters**, each representing a different clothing class (e.g., T-shirt, Pullover, Sandal).
 - t-SNE emphasizes **local neighborhood structure**, grouping similar observations closely.
 - However, the **distances between clusters** don't always reflect true global relationships.
-

0.0.20 Key Insight:

- PCA is better at preserving **global geometry**, but **fails to separate non-linear clusters**.
 - t-SNE excels at **cluster separation** and local relationships, making it ideal for **exploratory visualization**.
-

This comparison illustrates why we sometimes need **non-linear methods**. But even t-SNE has downsides:

- It's slow on large datasets.
- It lacks reproducibility across runs.
- It doesn't preserve global structure well.

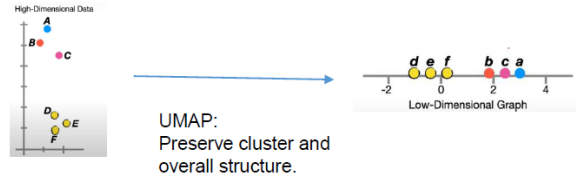
That's why we now turn to **UMAP**, a newer method that offers the **best of both worlds**: → speed, reproducibility, local + global structure.

Let's move to the next slide to explore UMAP.

UMAP: Uniform Manifold Approximation and Projection

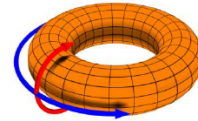
[UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction](#)
Leland McInnes, John Healy, James Melville (2018)

Motivation: Visualization of complex, high dimensional data in fewer dimensions applying topology concepts.



Assumptions

High-dimensional data lies on a lower dimensional manifold (curved surface).
Riemannian manifold: the data is uniformly distributed.



We now arrive at the centerpiece of this presentation: **UMAP**, which stands for **Uniform Manifold Approximation and Projection**.

0.0.21 What Is UMAP?

UMAP is a **non-linear dimensionality reduction technique** introduced by **Leland McInnes, John Healy, and James Melville** in 2018.

Its **core motivation** is:

To enable the **visualization** of high-dimensional data in 2D or 3D while preserving both **local clusters** and **global structure**, using concepts from **topology and manifold theory**.

0.0.22 The Main Idea

UMAP assumes that:

1. High-dimensional data **lies on a lower-dimensional manifold**—a curved surface embedded in higher space.
2. This manifold is **Riemannian**—meaning the data is **uniformly distributed** along it.

These assumptions allow UMAP to build a **graph-based representation** of the data that captures **topological relationships**, and then **optimize** a lower-dimensional layout that preserves those relationships.

0.0.23 Visual Example

- On the **left**, we have points A–F in high-dimensional space.
 - On the **right**, UMAP rearranges them into a 1D embedding.
 - The goal is to **preserve cluster integrity** (e.g., a/b/c grouped, d/e/f grouped) and also reflect their **relative distances**.
-

0.0.24 The Manifold Perspective

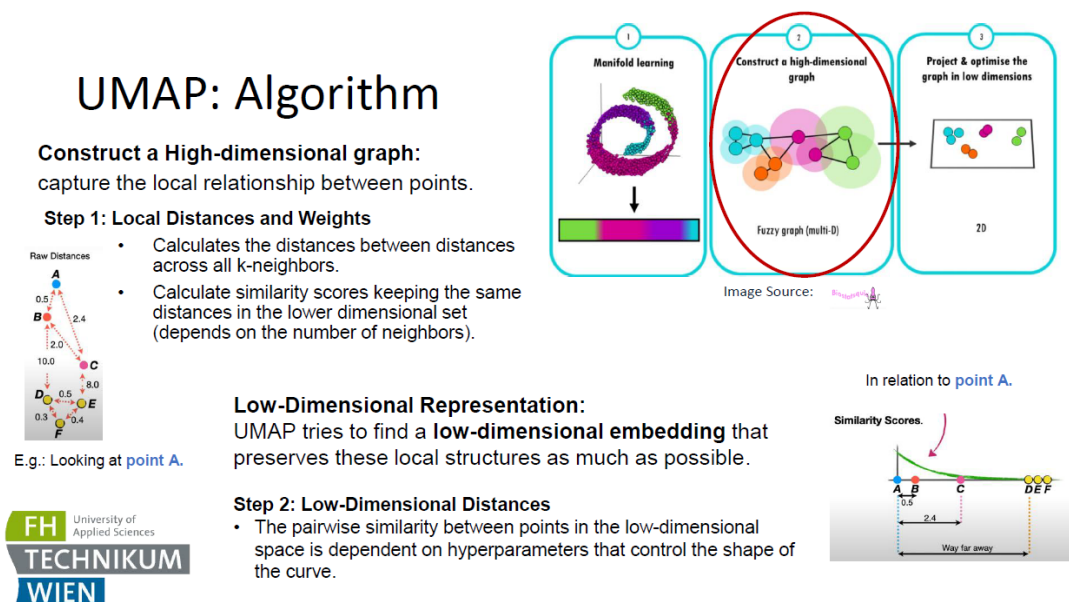
- The **orange torus** shown at the bottom right represents a **manifold**—a curved but mathematically structured surface.
 - UMAP’s core idea is that data often lives on such surfaces, and if we can “unwrap” them carefully, we can represent the data faithfully in fewer dimensions.
-

0.0.25 Why UMAP?

Compared to t-SNE, UMAP:

- Preserves **both local and global structure** better,
- Is **much faster**,
- Scales to **large datasets**,
- And offers **reproducible** results (if the seed is fixed).

Let’s now dive into **how** UMAP achieves this—on the next slide.



Now let’s break down the **UMAP** algorithm step by step.

0.0.26 Step 1: Construct a High-Dimensional Graph

UMAP begins by building a **graph in high-dimensional space**. Each node is a data point, and edges connect **nearest neighbors**, weighted by **distance-based similarity**.

For example:

- Looking at **point A**, we compute distances to its neighbors: B (0.5), C (2.4), D (10.0), etc.
 - The closer the neighbor, the stronger the edge in the graph.
-

0.0.27 How Are Weights Assigned?

UMAP uses a technique similar to t-SNE, but more refined:

- It calculates **local distances and weights** for each data point to its k-nearest neighbors.
- These are used to build a **fuzzy simplicial complex**—essentially, a soft-edged graph capturing **local topology**.

This process is shown in the center image at the top:

- Nodes are connected based on how close they are.
 - Overlapping neighborhoods indicate **local structure**.
-

0.0.28 Step 2: Low-Dimensional Embedding

UMAP then **projects this high-dimensional graph into 2D or 3D**, trying to:

Preserve local structure — that is, keep close neighbors in the high-dimensional space close in the lower-dimensional one.

As shown in the image on the bottom right:

- A remains close to B (0.5),
- But distant points like E and F get pushed far away.

This step involves:

- A **cost function** that encourages **similar points to stay close**,
 - And **dissimilar points to repel**, much like t-SNE—but using **cross-entropy loss** instead of KL divergence.
-

0.0.29 Hyperparameters Matter

- The shape of the final curve and neighborhood structure depends on hyperparameters like:
 - **n_neighbors** (how many neighbors to consider),
 - **min_dist** (how tightly points are packed),
 - and others that we'll discuss shortly.
-

0.0.30 In Summary

UMAP:

1. Learns the data's shape by constructing a **topological graph**,
2. Optimizes a **low-dimensional embedding** that preserves that shape.

It's **fast**, **scalable**, and produces **reproducible** results—perfect for exploratory data analysis and embedding tasks.

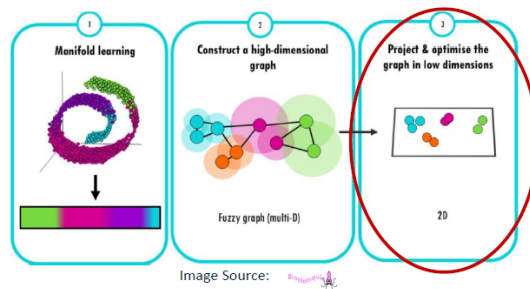
Let's continue to the next slide to explore UMAP's hyperparameters and how they affect the output.

UMAP: Algorithm

Steps 1 and 2:

- **similarity measures** used to capture the relationships between points in the **high-dimensional** and **low-dimensional** spaces.

Goal: UMAP tries to **match** these two graphs as closely as possible, meaning that if two points are close in the high-dimensional space (high similarity), they should also be close in the low-dimensional space (high similarity).



Optimization:

Step 3: UMAP minimizes the cross-entropy loss between the graphs:



$$\text{Loss} = \sum_{i \neq j} f_{ij} \log \left(\frac{f_{ij}}{g_{ij}} \right) + (1 - f_{ij}) \log \left(\frac{1 - f_{ij}}{1 - g_{ij}} \right)$$

→ Similarity in high-dimension (for f_{ij})
→ Similarity in low-dimension (for g_{ij})

Let's now look at the **final step of the UMAP algorithm**—how it performs **optimization** to align high-dimensional and low-dimensional structures.

0.0.31 Goal Recap:

UMAP has now constructed:

- A **graph in high-dimensional space**, using similarity scores f ,
- And a **graph in low-dimensional space**, using similarity scores g .

The key goal:

Make these graphs match: Points that are close in high-dim space should stay close in low-dim space.

0.0.32 Step 3: Optimization

UMAP minimizes a **cross-entropy loss function** between the two graphs.

The Loss Function:

$$\text{Loss} = \sum_{i \neq j} f_{ij} \log \left(\frac{f_{ij}}{g_{ij}} \right) + (1 - f_{ij}) \log \left(\frac{1 - f_{ij}}{1 - g_{ij}} \right)$$

- f_{ij} : similarity between point i and j in the **high-dimensional** space
- g_{ij} : similarity between point i and j in the **low-dimensional** space

This formula ensures:

- If two points are **similar in high dimensions**, they are **pulled closer** in the embedding.
 - If they are **not similar**, they are **pushed apart**.
-

0.0.33 Why Cross-Entropy?

- Cross-entropy is widely used for **matching probability distributions**.
 - UMAP treats the two graphs as **fuzzy probability distributions** of neighborhoods, and tries to minimize the **divergence** between them.
-

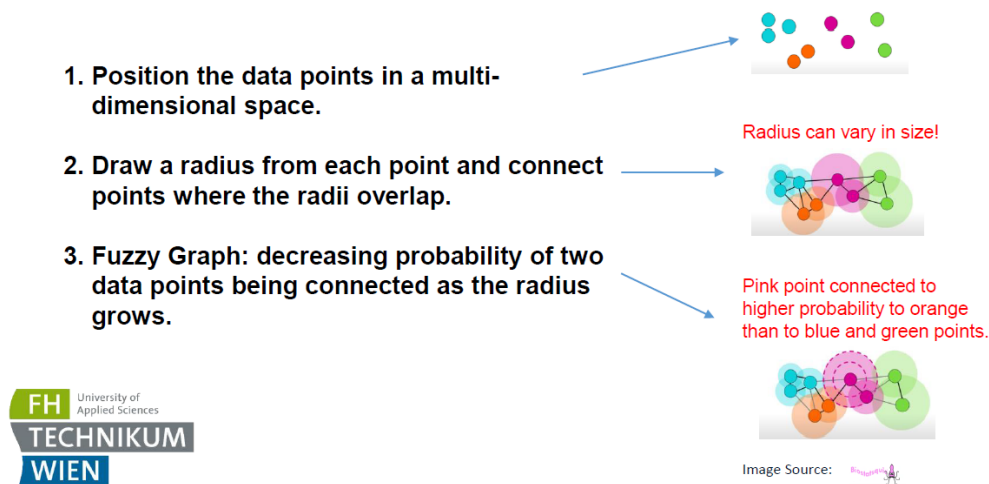
0.0.34 Summary

- **Step 1:** Build a graph based on neighborhood similarities in high dimensions.
- **Step 2:** Initialize a layout in low dimensions.
- **Step 3:** Optimize the layout to minimize loss using **stochastic gradient descent**, aligning the two graphs.

With this, UMAP produces a **compact, faithful, and topology-preserving low-dimensional embedding**.

Let's move to the next slide to explore the key **hyperparameters** that give us control over UMAP's behavior.

UMAP: High-dimensional Fuzzy Graph Construction



To understand how UMAP builds its internal representation, this slide explains how it constructs a **fuzzy graph in high-dimensional space**—a crucial first step in the algorithm.

0.0.35 Step-by-Step: High-Dimensional Fuzzy Graph Construction

1. Position the Data Points The input data already exists in a **multi-dimensional space** (e.g., 50D from word embeddings, or 784D from flattened images).

Each data point is initially just a vector in this space.

2. Define Radii Around Each Point UMAP draws a **radius** around each point—this defines a **local neighborhood**.

- But unlike fixed-radius methods, UMAP adapts the radius **per point**, depending on **local density**.
- This makes UMAP more robust to **non-uniformly distributed data**.

Two points are connected **if their radii overlap**, forming the **edges of the graph**.

Note: Radii are **not fixed**; they vary to capture **adaptive neighborhood sizes**.

3. Build the Fuzzy Graph This graph is “fuzzy” because:

- Edges are **weighted with a probability** (not binary connections).
- That weight reflects how **strong the relationship** is between two points.

- As the distance between points increases, the **probability of connection decreases**.

In the lower image:

- The **pink point** has stronger probability connections with **orange** points (closer),
- And weaker connections to **blue** or **green** points (further away).

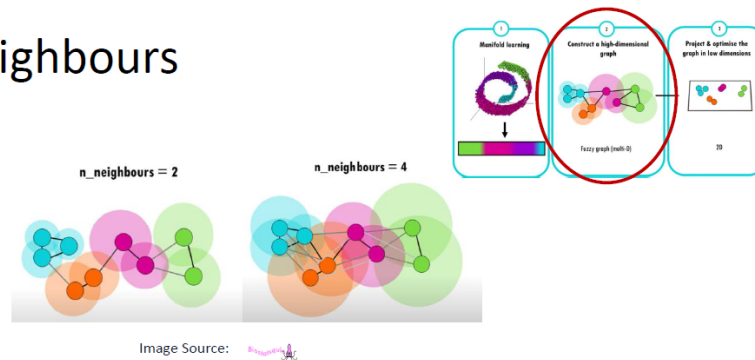
This step results in a **fuzzy simplicial complex**, capturing the local topological structure of the data.

0.0.36 Why This Matters

This fuzzy graph is the **foundation** that UMAP then projects into lower dimensions. Preserving the **structure and probabilities** of this graph is the goal of the embedding optimization.

Next, we'll explore the **hyperparameters** that let us control this process. Please continue with the next slide.

UMAP: n_neighbours



n_neighbours is proportional to the radii around the points.

- Size of radii is based on the distance of the point's nearest end neighbour.
- Points connected to the same number of neighboring points.
- The probability of the connections (in shades of gray) decreases as points farther away get connected.



Now let's look at the **first key hyperparameter** in UMAP: **n_neighbours**.

0.0.37 n_neighbours: Controls Local Connectivity

This parameter determines **how many neighbors** each point considers when building the high-dimensional graph.

Effects on Radius and Graph:

- A **larger n_neighbours** leads to **larger radii** around each point (shown on the right).
- More neighbors means **more connections** in the graph, capturing **more global structure**.

- A **smaller value** (left side) keeps neighborhoods **tight and local**, preserving fine structure and dense clusters.
-

0.0.38 Visual Explanation

- With `n_neighbors = 2`, the graph is **sparser**—points are only connected to a few others. Clusters are more **segmented**.
 - With `n_neighbors = 4`, the **radii are larger**, and there's more **overlap** between local neighborhoods. This blends local structures with more **global continuity**.
-

0.0.39 Practical Guidelines

- **Smaller values** (e.g., 5–15): good for **fine cluster detail** (e.g., cell types in biology).
 - **Larger values** (e.g., 30–100): good for **capturing macro patterns** (e.g., broad topic clusters in text).
-

0.0.40 Summary

- `n_neighbors` shapes the **locality vs. globality** trade-off in UMAP.
- It's directly linked to the **graph construction step**—the very first phase of the algorithm.
- It also affects **runtime and output structure**, so tuning it carefully is important for good embeddings.

Let's now move to the next slide to explore the second major hyperparameter: `min_dist`.

UMAP: `n_neighbours`

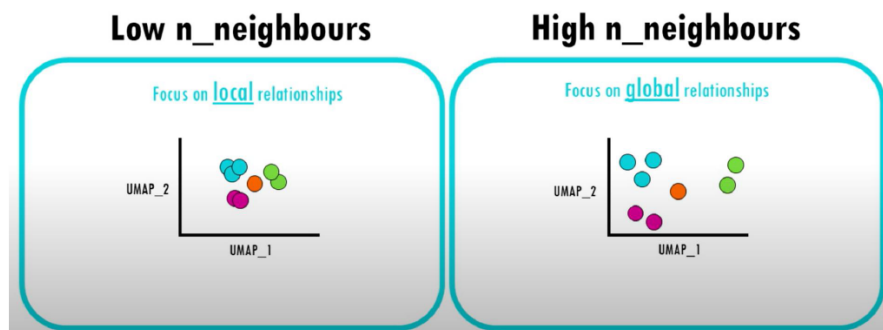


Image Source: [https://umap-learn.org/](#)

This slide gives a **visual comparison** of how different values for `n_neighbors` affect the **structure of the UMAP embedding**.

0.0.41 Left: Low `n_neighbors`

- Focus is on **local relationships**.
 - Small, **tight clusters** are formed.
 - Data points group closely based on **fine-grained similarities**.
 - This setting is ideal when you want to detect **micro-structures** or subpopulations.
-

0.0.42 Right: High `n_neighbors`

- Focus shifts to **global relationships**.
 - Clusters may be more **spread out** and **globally ordered**.
 - It better captures **macro patterns** and **overall geometry** of the dataset.
 - Useful when you want a **holistic view** of how groups relate to each other.
-

0.0.43 Key Insight

`n_neighbors` acts as a **scale controller**:

- **Low values** → local fidelity, more granular
- **High values** → global structure, less cluster separation

And remember:

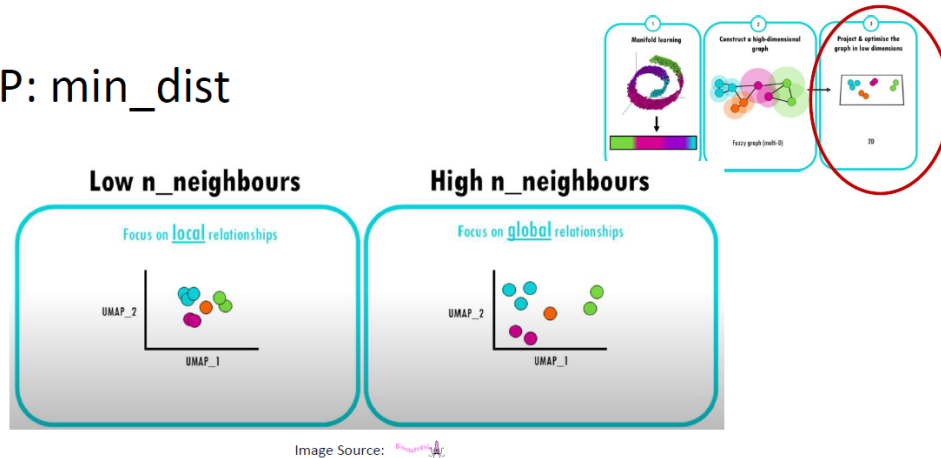
- The point itself is **included** in its neighborhood count.
 - The **default** in many UMAP implementations is `n_neighbors = 15`.
-

With this, we've completed our deep dive on the `n_neighbors` hyperparameter.

Next, we'll look at another powerful hyperparameter in UMAP: `min_dist`, which controls how **tightly points are packed** in the embedding.

Please continue with the next slide.

UMAP: min_dist



min_dist is the minimum distance between points in the low-dimensional space.



Related to visualization of relationships which were already computed.

Let's now focus on another **crucial UMAP hyperparameter**: **min_dist**.

0.0.44 What is min_dist?

min_dist controls the **minimum spacing allowed** between points in the **low-dimensional space** (e.g., the 2D embedding).

Think of it as a **compression limit**:

- **Low min_dist** → points can pack tightly together
- **High min_dist** → points are kept more spread apart

0.0.45 Why is it important?

This parameter directly affects the **visual appearance of clusters** in your embedding:

- A **lower min_dist** emphasizes **local density** and makes **tight, well-separated clusters**.
- A **higher min_dist** creates more **space between points**, helping to highlight **larger global trends** or relationships.

0.0.46 Illustration

In this slide:

- On the **left**, with **low min_dist**, points are **clustered tightly**, emphasizing **fine detail**.
- On the **right**, with **higher min_dist**, points are **more spread out**, showing a **broader structure**.

This step is part of the **projection & optimization phase** (highlighted top-right in red), where UMAP places points in the 2D plane.

0.0.47 Summary

- `min_dist` affects the **final layout**, not the neighborhood graph.
 - Use **low values** (e.g., **0.1**) when you want **compact clusters**.
 - Use **higher values** (e.g., **0.5** or **0.8**) when you want to **preserve inter-cluster geometry**.
-

Together with `n_neighbors`, `min_dist` gives you powerful control over how **local vs. global** and **dense vs. sparse** your final UMAP plot will look.

Let's move forward to see some examples or further tuning strategies. Please share the next slide.

UMAP: hyperparameters

Image adapted from McInnes et al. (2018): UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction

Check out: [Understanding UMAP](#) to play around with parameters

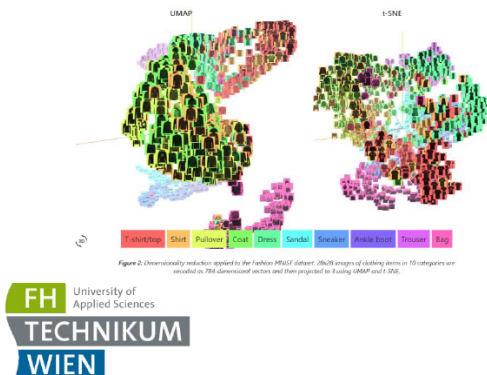
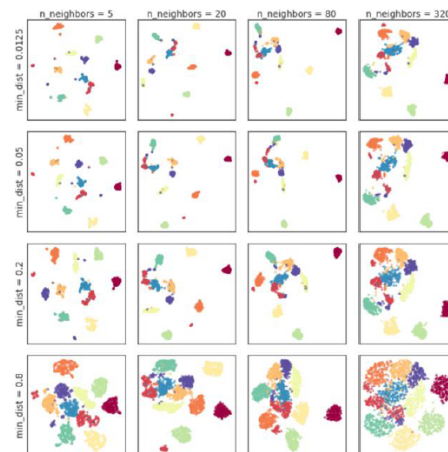


Figure 2: Dimensionality reduction applied to the Fashion MNIST dataset. 10000 images of clothing items are recorded as 784-dimensional vectors and then projected to 2 using UMAP and t-SNE.



Variation of UMAP hyperparameters `n` and `min-dist` result in different embeddings. The data is the PenDigits dataset, where each point is an 8x8 grayscale image of a hand-written digit.

This slide summarizes the **combined impact** of UMAP's two most important hyperparameters: `n_neighbors` and `min_dist`.

0.0.48 Left Side: Fashion MNIST Visualization

Here we see a side-by-side comparison of:

- **UMAP** (left) and
- **t-SNE** (right)

Both were applied to the **Fashion MNIST** dataset (images of clothing). UMAP clearly preserves both **clusters and relative positions**, while t-SNE provides a **tight clustering** but less interpretability in global structure.

0.0.49 Right Side: Hyperparameter Grid (PenDigits dataset)

This matrix shows how UMAP embeddings **change across different parameter settings**.

n_neighbors: Left → Right

- Increases from **5** to **320**
- Low: fine-grained clusters
- High: smoother, more blended structure

min_dist: Top → Bottom

- Increases from **0.0125** to **0.8**
 - Low: tight clusters (points packed)
 - High: loose clusters (points spread out)
-

0.0.50 Observations

- **Top-left (low n, low min_dist):** Very tight clusters, highly local detail.
 - **Bottom-right (high n, high min_dist):** Very global structure, but detail is smoothed out.
 - **Middle zone (n 20–80, min_dist 0.05–0.2):** Often gives a **good trade-off** between preserving detail and seeing global structure.
-

0.0.51 Practical Tip

When using UMAP:

- Tune **n_neighbors** to control the **scale of relationships** (local vs. global).
- Adjust **min_dist** to control the **tightness of clusters**.

There's no one-size-fits-all setting—UMAP gives you flexibility depending on your **data** and **analysis goal**.

We've now covered the **theory, mechanics, and tuning** of UMAP. Let me know if you'd like to proceed to applications, examples, or wrap-up slides.

UMAP: Properties

- As a nonlinear dimensionality reduction: lacks the strong interpretability of PCA.
- Preservation of Local and Global structures: UMAP tries to keep clusters of data together, while respecting the overall organization of the data.
- Similarly to t-SNE, UMAP assumes that retaining local geometry is more important than retaining large-scale geometry (global).
- It is known that UMAP is almost entirely driven by the k-neighborhood graph!
- UMAP is not robust for small dataset.



	t-SNE	UMAP
COIL20	20 seconds	7 seconds
MNIST	22 minutes	98 seconds
Fashion MNIST	15 minutes	78 seconds
GoogleNews	4.5 hours	14 minutes

UMAP is much faster than t-SNE!

17

Let's wrap up our technical overview by summarizing the **key properties of UMAP**.

0.0.52 Interpretability

- UMAP is a **nonlinear dimensionality reduction method**, so unlike PCA, it **lacks straightforward interpretability**.
- In PCA, each principal component is a linear combination of features—but UMAP embeddings **do not carry such semantic meaning**.

0.0.53 Local and Global Structure

- A major strength of UMAP is its **ability to preserve both local and global structure**:
 - **Local**: Points that are close in high-dimensional space remain close.
 - **Global**: The overall geometry and relationships between clusters are reasonably well maintained.

0.0.54 Local Geometry Focus

- Similar to t-SNE, UMAP emphasizes **local geometry** over large-scale geometry.
- This means it's particularly good at:
 - Identifying **clusters**
 - Revealing **fine structure**
 - Organizing data based on neighborhood graphs

0.0.55 Driven by k-Nearest Neighbors

- UMAP's behavior is **strongly influenced by the k-nearest-neighbor graph** built during the initial phase.
 - This makes it sensitive to the choice of `n_neighbors`.
-

0.0.56 Limitations

- **Not robust on small datasets.**
 - It performs best when there is enough data to build meaningful neighborhood relationships.
 - For very small datasets, the graph construction may not be reliable, and results can be noisy or unstable.
-

0.0.57 Performance Comparison

The **table** highlights one of UMAP's biggest practical advantages:

Dataset	t-SNE Time	UMAP Time
COIL20	20 seconds	7 seconds
MNIST	22 minutes	98 seconds
Fashion MNIST	15 minutes	78 seconds
GoogleNews	4.5 hours	14 minutes

UMAP is dramatically faster than t-SNE, especially on large datasets.

So in conclusion:

- UMAP is a **powerful and efficient tool** for high-dimensional data visualization and analysis.
- While it's not perfect for small datasets or direct feature interpretation, its **speed**, **scalability**, and **topology-preserving embeddings** make it a preferred choice for many modern machine learning tasks.

Let me know if you'd like to proceed with an application, demo, or final summary slide.

Summary: Dimensionality Reduction Methods

	PCA	t-SNE	UMAP
Type	Linear	Non-linear	Non-linear
Focus	Maximize variance (global structure)	Preserving local structure	Preserving local and global relationships
Preserves	Spread of data (variance)	Similarity across neighbors	Shape and clusters (local and global)
Speed	Very Fast	Slow (especially for	Faster than t-SNE
Dimensionality	any	2-3	any
Stochasticity	Deterministic	Stochastic	Stochastic
Interpretability	Easy to interpret	Hard to interpret	More interpretable than t-SNE, but less than PCA
Outputs	Orthogonal axes	Non-linear clusters	Dense, non-linear manifold



Let's conclude with this clear **summary table**, comparing the three main dimensionality reduction methods: **PCA**, **t-SNE**, and **UMAP**.

0.0.58 Method Overview

Property	PCA	t-SNE	UMAP
Type	Linear	Non-linear	Non-linear
Focus	Maximize variance (global)	Preserve local structure	Preserve local and global relationships
Preserves	Spread of data (variance)	Neighborhood similarities	Cluster shape + manifold geometry
Speed	Very fast	Slow (esp. for large datasets)	Much faster than t-SNE
Dimensionality	Any	Usually 2–3	Any
Stochasticity	Deterministic	Stochastic	Stochastic
Interpretability	Easy to interpret	Hard to interpret	Mid: better than t-SNE, worse than PCA
Outputs	Orthogonal axes	Non-linear clusters	Dense manifold with clusters and structure

0.0.59 Key Takeaways

- **PCA** is ideal for **interpretability** and speed, but only captures **linear structure**.
- **t-SNE** is great for visualizing **tight clusters**, but is **slow**, **non-deterministic**, and doesn't preserve global structure.

- **UMAP** offers a **balanced, fast, and expressive** alternative that works well for **visualization, clustering, and exploration**—especially on large, complex datasets.
-

Thank you for your attention! Let me know if you'd like to add a **Q&A slide, references, or applications** to complete the presentation.

When to use what?

PCA works well for linear relationships (preserving variance).

E.g.: Stock price analysis where the relationships between financial metrics are often linear.

t-SNE for clear, detailed cluster separation (local structures).

E.g.: Identifying subtypes in medical diagnostics (e.g., cancer cell types) where clear separation can improve treatment outcomes.

UMAP for large, complex, nonlinear data (typically tens of thousands to millions of samples)

E.g.: Analyzing high-dimensional gene expression or natural language embeddings, where patterns are inherently nonlinear.

Consider the **trade-off** between speed, interpretability, and structure preservation.



Multiple dimensionality reduction methods can be used in combination.

This final slide provides **practical guidance** on **when to use PCA, t-SNE, or UMAP**, depending on your data and goals.

0.0.60 PCA

- Best for **linear** relationships and **variance preservation**.
 - Very fast and easy to interpret.
 - **Example**: Stock price analysis, where features often exhibit **linear dependencies**.
-

0.0.61 t-SNE

- Ideal for **fine-grained cluster separation**—especially when visualizing distinct **subpopulations**.
 - Great for **medical diagnostics**, such as identifying cancer subtypes, where local structure reveals **biological differences**.
 - Works well with **small to medium** datasets.
-

0.0.62 UMAP

- Designed for **large, high-dimensional, non-linear datasets**.
 - Excels when:
 - Working with **embeddings** (e.g., NLP),
 - Or analyzing **biological data** (e.g., single-cell RNA-seq).
 - Handles **millions of data points**, offering **speed, scalability**, and **balanced structure preservation**.
-

0.0.63 Final Thought

Always **consider the trade-off** between:

- **Speed**
- **Interpretability**
- **Structure preservation**

You may even combine methods:

- Use PCA to reduce from 1000D to 50D,
 - Then apply UMAP or t-SNE for 2D visualization.
-

Thank you for following this presentation on **UMAP and dimensionality reduction**. Let me know if you'd like a conclusion slide, references, or Q&A wrap-up.

Takeaway

PCA (Principal Component Analysis): Captures **linear** relationships among variables while maximizing **variance**. It is fast and interpretable, but limited to linear patterns.

t-SNE (t-Distributed Stochastic Neighbor Embedding): Focuses on **local** structures and is great for **cluster visualization**. It captures nonlinear relationships, but is computationally intensive and less interpretable.

UMAP (Uniform Manifold Approximation and Projection): Balances **local** and **global** structures. It is very fast, flexible, and effective for large, nonlinear data. UMAP maintains more global structure than t-SNE.

Next up: MDS



Let's close with the key **takeaways** from this session on **dimensionality reduction**:

0.0.64 PCA – Principal Component Analysis

- **Strength:** Captures **linear** relationships and **maximizes variance**.
 - **Use it when** you need speed, interpretability, and your data is **linearly structured**.
 - *Think stock market data or financial indicators.*
-

0.0.65 t-SNE – t-Distributed Stochastic Neighbor Embedding

- **Strength:** Reveals **local structure** and forms **clear, detailed clusters**.
 - **Use it when** you want high-resolution **cluster visualization**.
 - Computationally heavy and hard to interpret globally.
 - *Think cancer cell subtype analysis or digit recognition.*
-

0.0.66 UMAP – Uniform Manifold Approximation and Projection

- **Strength:** Balances **local and global** structure.
 - **Use it when** you're working with **large, nonlinear datasets** (tens of thousands to millions of points).
 - *Faster than t-SNE, retains more global continuity, very flexible.*
 - *Think gene expression, NLP embeddings, recommender systems.*
-

0.0.67 Final Thought

Choosing between PCA, t-SNE, and UMAP is a **trade-off** between speed, interpretability, and structure preservation.

0.0.68 *Next Topic:* MDS (Multidimensional Scaling)

In the next session, we'll explore **MDS**, an older but still useful technique for preserving **pairwise distances**.

Thank you for your attention! Let me know if you'd like speaker notes, a quiz, or a summary handout.

References


UMAP Paper (see Moodle)

[UMAP Uniform Manifold Approximation and Projection for Dimension Reduction | SciPy 2018](#)

[BioTuring Webinar: A Practical Guide to UMAP by its author John Healy](#)



```
pip install umap-learn
conda install -c conda-forge umap-learn
```

The UMAP logo, consisting of a stylized blue triangle with a yellow 'A' inside, followed by the text 'UNIFORM MANIFOLD UMAP APPROXIMATION & PROJECTION' in a sans-serif font.
<https://umap-learn.readthedocs.io> (Also includes many examples!)

22

This final slide provides useful **references and resources** for further exploration and application of UMAP.

0.0.69 Core Paper

UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction
Link (2018, McInnes et al.): [SciPy 2018 UMAP Paper](#) Also available via **Moodle**.

This paper introduces the theory behind UMAP, including its topological foundations, fuzzy simplicial sets, and optimization framework.

0.0.70 Video Resource

BioTuring Webinar: A Practical Guide to UMAP By co-author **John Healy**, this is an excellent hands-on explanation of the algorithm, use cases, and tuning parameters. [Watch the webinar](#)

0.0.71 Installation Instructions

To install the official umap-learn package:

```
pip install umap-learn
# or
conda install -c conda-forge umap-learn
```

0.0.72 Documentation and Examples

Official docs: <https://umap-learn.readthedocs.io> Includes:

- Example notebooks,
 - Parameter explanations,
 - FAQs and common applications.
-

With that, your UMAP presentation is now complete. Let me know if you'd like:

- a compact speaker script,
- a notebook template for the assignment,
- or practice quiz questions to reinforce learning.