Final Notes & Review: Dimensionality Reduction for Gene Expression

Part 1: The "Big Picture" - The Curse of Dimensionality

In the last lecture, we created a **Gene Expression Matrix**. This is our starting point.

- **The Data:** A giant table where **rows are genes** (e.g., 20,000 of them) and **columns are experimental conditions** (e.g., 100 patients, or "samples").
- **The Problem:** From a machine learning perspective, each gene is a data point. The "features" describing that gene are its expression levels in each of the 100 conditions. This means each gene is a point in **100-dimensional space**.
- **Jargon: The Curse of Dimensionality:** Humans can't visualize anything beyond 3 dimensions. More importantly, algorithms for clustering and classification become extremely slow and ineffective in very high-dimensional spaces. Distances between points become less meaningful, and the amount of data needed to get a reliable result grows exponentially.

The Goal: We need to **reduce the number of dimensions** (e.g., from 100 down to 2 or 3) while **losing as little important information as possible**. This is **Dimensionality Reduction (DR)**.

Part 2: What is Dimensionality Reduction?

- **Simple Explanation:** It's about finding a "smarter" way to look at your data. It compresses the data by finding the most important underlying patterns and projecting the data onto a new, smaller set of axes that capture those patterns.
- **The Analogy:** Imagine you have a 3D model of a long, thin pencil. The pencil exists in 3D space (X, Y, Z). But is it truly 3D? The most important piece of information—its length—can be captured by a single new axis that runs along the pencil's core. By projecting all the points of the pencil onto this one new axis, you have reduced its dimensionality from 3 to 1 while preserving the most significant feature (its length).
- **Why we do it:**
    1. **Visualization:** To plot our 20,000 genes on a 2D scatter plot to see if they form visible clusters.
    2. **Data Compression:** To store the data more efficiently.
    3. **Improved Performance:** To make subsequent machine learning algorithms (like clustering or classification) work better and faster.

Part 3: The Main Algorithm - Principal Component Analysis (PCA)

PCA is the workhorse of dimensionality reduction. It's a **linear** method, meaning it finds new axes that are simple linear combinations of the old axes.

The Core Idea of PCA (Two Sides of the Same Coin):

PCA finds a new set of axes (called **Principal Components**) that satisfy two properties simultaneously:

1. **Maximize Variance (Slide 8, Left):** The first new axis (Principal Component 1, or PC1) is the line that, when you project all your data points onto it, the spread (variance) of the projected points is as large as possible. PC2 is the next axis (perpendicular to PC1) that captures the next largest variance, and so on.
2. **Minimize Reconstruction Error (Slide 8, Right):** The same line (PC1) is also the one that minimizes the average squared distance from each original data point to its projection on that line.

The "How" of PCA (The Algorithm at a Glance, Slide 9):

PCA is an algorithm that uses linear algebra. You don't need to perform the calculations by hand for the exam, but you *must* understand the concepts.

- **Jargon: Covariance Matrix:** A matrix that describes how different dimensions (our experimental conditions) vary with each other. If patients who have high expression in condition A also have high expression in condition B, their covariance will be high.
- **Jargon: Eigenvectors & Eigenvalues:** These are fundamental concepts from linear algebra. For a given matrix (like our covariance matrix), an **eigenvector** is a special direction. When you apply the matrix's transformation, vectors in this direction don't change their direction, they only get stretched or shrunk. The amount they are stretched by is the **eigenvalue**.
- **The PCA Algorithm:**
    1. Calculate the **covariance matrix** of your data.
    2. Calculate the **eigenvectors and eigenvalues** of this covariance matrix.
    3. **The Magic:** The eigenvector with the **largest eigenvalue** corresponds to the direction of maximum variance in your data. **This is Principal Component 1 (PC1)**. The eigenvector with the second-largest eigenvalue is PC2, and so on.
    4. To reduce your data to 2 dimensions, you simply take your original data and project it onto PC1 and PC2.

Part 4: Beyond PCA - The Non-Linear World (SNE & t-SNE)

PCA is powerful, but it's limited because it can only find "flat," linear patterns in the data (like our pencil). What if the data has a complex, curved structure (like a Swiss roll)?

- **The Problem:** For complex biological data, PCA can fail to capture the true underlying structure.
- **The Solution: Non-Linear DR / Manifold Learning.** These are more advanced algorithms designed to find curved or complex low-dimensional structures ("manifolds") within high-dimensional data.
- **Jargon: t-distributed Stochastic Neighbor Embedding (t-SNE)** (Slides 15-21)
    - **The Core Idea:** Instead of preserving large-scale distances like PCA, t-SNE focuses on **preserving local neighborhoods**. It tries to create a 2D "map" where points that were close neighbors in the original high-dimensional space are still close neighbors in the 2D map.
    - **The Crowding Problem:** A key challenge is that it's mathematically difficult to represent all the neighbors of a point from a high-dimensional space in a low-dimensional space. There isn't enough room. This leads to the "crowding problem," where distinct clusters can get squished together on the 2D map.
    - **t-SNE's Solution:** It uses a special probability distribution (the t-distribution, which has "heavy tails") to model the neighborhoods in the 2D map. This allows points that are moderately far apart in high-D to be pushed even farther apart in 2D, which helps to separate the clusters more clearly.
    - **Key Takeaway:** t-SNE is a state-of-the-art **visualization** algorithm. It is fantastic for creating beautiful, interpretable 2D maps of high-dimensional data, but it is not typically used for pre-processing before clustering, as PCA is.

Potential Exam Questions & Key Concepts

- **Q: What is the "Curse of Dimensionality" and why is it a problem for gene expression analysis?**
    - **A:** It refers to the fact that in very high-dimensional spaces, data points become sparse and distances between them become less meaningful. This is a problem for gene expression data because we often have a small number of samples (e.g., 100 patients) but a huge number of dimensions (e.g., 20,000 genes), which makes it very difficult for clustering algorithms to find a meaningful structure.
- **Q: What are the two primary goals or interpretations of Principal Component Analysis (PCA)?**
    - **A:** 1) To find the projection directions (principal components) that **maximize the variance** of the projected data. 2) To find the projection that **minimizes the reconstruction error** (the squared distance between the original points and their projected versions).

- **Q: What is a Principal Component?**
    - **A:** A Principal Component is an eigenvector of the data's covariance matrix. The first Principal Component (PC1) is the eigenvector with the largest eigenvalue and represents the direction of the greatest variance in the data.
- **Q: You have gene expression data from 1000 cancer patients. You run PCA and find that PC1 clearly separates patients who responded to treatment from those who did not. What does this tell you?**
    - **A:** It tells you that the **single largest source of variation** in the entire gene expression dataset is related to the treatment response. PC1 has captured the dominant biological signal that distinguishes responders from non-responders.
- **Q: What is the main difference between a linear DR algorithm like PCA and a non-linear one like t-SNE?**
    - **A:** PCA is a linear method that tries to preserve the large-scale, global structure of the data by finding "flat" hyperplanes of maximum variance. t-SNE is a non-linear method that focuses on preserving the small-scale, **local neighborhood structure** of the data. This makes t-SNE particularly good for visualizing complex, non-linear datasets with well-separated clusters.

Final Deep Dive: Dimensionality Reduction - The Nitty-Gritty Details

Part 1: The Math and Complexity

Understanding the computational cost is key to knowing when to use each algorithm. Let n be the number of data points (genes) and m be the number of dimensions (conditions/patients).

- **Principal Component Analysis (PCA):**
    - **The Math:** The core of PCA is an operation from linear algebra called **Singular Value Decomposition (SVD)** on the n x m data matrix. This is a one-time calculation.
    - **Time Complexity:** The complexity of SVD is roughly $O(\min(n^2m, nm^2))$.
    - **Practical Takeaway:** The cost depends on both the number of genes and the number of conditions. If you have far more genes than conditions (e.g., 20,000 genes, 100 conditions), the complexity is dominated by n^2m, which can be slow. However, it's a single, deterministic calculation.
- **t-SNE (t-distributed Stochastic Neighbor Embedding):**
    - **The Math:** t-SNE is an iterative algorithm that uses **gradient descent** to minimize a cost function (the KL divergence between input and output neighborhood probabilities).
    - **Time Complexity:** A naive implementation is **$O(n^2)$** per iteration, which is extremely slow. Modern implementations use approximations (like the Barnes-Hut method) to speed it up to roughly **$O(n \log n)$** per iteration.
    - **Practical Takeaway:** The key here is the n. The runtime of t-SNE is highly dependent on the **number of data points (genes)**. It is manageable for thousands of points but becomes prohibitively slow for hundreds of thousands or millions of points.

Part 2: The Ultimate Comparison Table

Feature	PCA (Principal Component Analysis)	t-SNE (and other Manifold Algs)
Core Goal	Preserves **global variance**. Finds the "most important" flat directions in the data.	Preserves **local neighborhood structure**. Tries to keep points that are close in high-D also close in low-D.
Algorithm Type	**Linear.** It can only find patterns that can be represented by straight lines (hyperplanes).	**Non-linear.** It can "unroll" complex, curved, or twisted data structures (manifolds).
Output	A **mathematical transformation**. It gives you a new set of coordinates (the principal components) for your data.	Primarily a **visualization**. The coordinates in the final plot don't have a clear mathematical meaning on their own.
Use Case	**Preprocessing & Data Compression.** Excellent for preparing high-dimensional data for use in other ML algorithms (clustering, classification). Also good for basic visualization.	**Data Exploration & Visualization.** Unbeatable for creating beautiful, intuitive 2D maps that reveal the clustering structure of complex data.
Pros	**Fast** (for moderate dimensions), **deterministic** (always gives the same output), **interpretable** (the components represent directions of variance), and can be applied to new data points easily.	Creates visually stunning, well-separated clusters. Excellent at revealing the underlying structure of complex, non-linear data.
Cons	**Fails on non-linear data** (will "crush" a Swiss roll into a flat pancake). The axes can be hard to interpret biologically.	**Very slow** on large numbers of data points (n). **Not deterministic** (different runs give slightly different plots). **Cannot be used on new data**. Distances and cluster sizes in the final plot can be misleading.

Part 3: Edge Cases & "Risky" Exam Questions

- **Q: You run PCA on your gene expression data and find that the first two principal components (PC1 and PC2) only capture 15% of the total variance. What does this tell you about your data and your PCA plot?**
  - **A:** This indicates that the variance in the data is not concentrated in just a few linear directions. The data structure is likely very complex, high-dimensional, and "blob-like." The resulting 2D PCA plot would be a **poor and misleading representation** of the real data, as it is discarding 85% of the information.
- **Q: You run t-SNE on your data and the resulting plot shows three tight, well-separated clusters. Cluster A appears much larger (takes up more space on the plot) than Cluster B. Can you conclude that Cluster A contains more genes than Cluster B?**
  - **A: No, you cannot.** This is a critical limitation of t-SNE. The algorithm's goal is to separate neighborhoods, and it will expand or contract the space between and within clusters to achieve the best possible visualization. The on-screen size of a t-SNE cluster is **not proportional** to the number of points it contains or its variance. You must color the points by a known property or count them to determine the cluster's actual size.
- **Q: You have a dataset of 1 million cells (points) from a single-cell RNA-seq experiment with 50 gene measurements (dimensions). You want to reduce the dimensionality to prepare it for a classification algorithm. Which method do you choose and why?**
  - **A: PCA**. The main reason is **scalability**. t-SNE has a time complexity that scales poorly with the number of points (n), and 1 million points would be computationally infeasible. PCA, while computationally intensive, can handle this scale. Furthermore, PCA provides a true mathematical transformation that can be applied to new data, which is necessary for a classification pipeline, whereas t-SNE is a one-time visualization tool.
- **Q: What is the "Crowding Problem" and which algorithm is specifically designed to address it?**
  - **A:** The Crowding Problem occurs when trying to map a high-dimensional neighborhood into a low-dimensional space. There simply isn't enough "room" in 2D to faithfully represent all the neighborly distances from a 100-D space. This can cause distinct clusters to get squished together in the middle of the visualization. **t-SNE** is designed to address this by using a heavy-tailed t-distribution in the output space, which allows it to push moderately separated clusters farther apart, creating clearer visualizations.
- **Q: Can you use a PCA model trained on one dataset to transform a brand new data point? Can you do the same with t-SNE?**
  - **A: Yes for PCA, no for t-SNE.** A PCA model is a mathematical transformation (a set of rotations and projections). You can save this transformation and apply it to new data points to see where they would land in the principal component space. t-SNE is an iterative optimization process that depends on the entire dataset at once. There is no simple way to add a new point to an existing t-SNE plot without re-running the entire, slow algorithm on the whole dataset.

Solutions & Explanations: Dimensionality Reduction

Question 1: Why do we need dimensionality reduction for gene expression data?

We need dimensionality reduction for gene expression data primarily to overcome the **"Curse of Dimensionality."** A typical gene expression matrix might have 20,000 genes (rows) but only 100 samples/patients (columns). This means each gene is a data point in a 100-dimensional space. This creates several major problems:

1. **Visualization:** Humans cannot visualize data beyond 3 dimensions. To see if our genes form natural clusters or patterns, we must reduce the data to 2D or 3D for plotting.
2. **Noise Reduction:** Many of the dimensions (conditions) might be noisy or irrelevant to the biological question at hand. Dimensionality reduction methods, especially PCA, are excellent at identifying the main axes of variation (the "signal") and discarding the less important axes (the "noise").
3. **Computational Efficiency:** Algorithms like clustering run much slower and less effectively in high-dimensional spaces. Reducing the dimensions first makes subsequent analyses (like clustering or classification) faster and often more accurate.
4. **Meaningful Distances:** In very high-dimensional space, the distance between every point and every other point tends to become almost the same. This makes distance-based algorithms like k-Means less meaningful. DR projects the data into a lower-dimensional space where the distances between points are more significant.

Question 2: Among PCA, MDS and SNE, which do you prefer and why? Explain pros and cons of each method. What do each of these methods try to minimize?

There is no single "best" method; the preference depends entirely on the goal of the analysis.

Method	**What it Minimizes**	**Pros**	**Cons**
PCA (Principal Componen t Analysis)	**Reconstruction Error.** (Maximizes Variance)	**Fast, deterministic, interpretable.** The components are real mathematical transformations that can be applied to new data. Excellent for pre-processing.	**Linear.** It fails to capture complex, non-linear (curved) structures in the data. Can be misleading for visualization if the main variance is not in the first few components.
MDS (Multi-Di mensional Scaling)	**"Stress":** The difference between the pairwise distances in the original space and the pairwise distances in the low-dimensional space.	**Intuitive.** Its goal is easy to understand: create a "map" that preserves the original distances as faithfully as possible. Can handle non-linear data well if the distance metric is meaningful.	**Slow ($O(n^2)$).** Requires calculating a full n x n distance matrix, which is infeasible for very large datasets. The output can be sensitive to the initial random placement.
SNE / t-SNE	**Kullback-Leibler (KL) Divergence** between the probability distributions of neighborhoods in the high-D and low-D spaces.	**Excellent for visualization.** Unbeatable at revealing the underlying cluster structure in complex, non-linear data by separating clusters very clearly.	**Very slow ($O(n \log n)$).** Can't be applied to new data points. The output plot is **only for visualization**—the distances and sizes of clusters on the plot can be misleading and do not have a direct mathematical meaning.

Which do I prefer and why?

- **For Pre-processing Data for another ML algorithm:** I prefer **PCA**. It is fast, deterministic, and provides a true transformation of the data that is suitable for feeding into a classifier or clustering algorithm.
- **For Visual Data Exploration:** I prefer **t-SNE**. It produces the most intuitive and easy-to-interpret visualizations of complex data, making it the best tool for understanding the potential clustering structure of a dataset.
- **For when Pairwise Distances are Themselves Important:** I would use **MDS**. If my primary goal is to see how well the known distances between items (e.g., evolutionary distances between species) can be represented on a 2D map, MDS is designed for exactly that.

Question 3 & 4: Explain KL-divergence and simulate SNE, MDS, and PCA.

What is KL-Divergence?
KL-Divergence is a measure of how one probability distribution differs from a second, reference probability distribution. It's often called "relative entropy."

- **Analogy:** It measures the "surprise" of seeing data from distribution P when you were expecting data from distribution Q. If P and Q are identical, the surprise is zero (KL-divergence = 0). The more different they are, the higher the KL-divergence.

Simulation with Data: p1=(2,1), p2=(3,4), p3=(0,3)

How SNE Works (Example):

1. **Input Space (2D):** SNE's goal is to preserve neighborhoods. First, it calculates the pairwise Euclidean distances between the points.
2. **Input Probabilities (p_ij):** It converts these distances into conditional probabilities. For p1, it asks: "If I were to pick a neighbor for p1, what is the probability I would pick p2? What about p3?" Closer points get a higher probability. This is done for all points, creating a set of neighborhood probabilities.
3. **Output Space (1D):** It randomly places the three points on a 1D line, e.g., y1=1.5, y2=-0.5, y3=0.2.
4. **Output Probabilities (q_ij):** It calculates the neighborhood probabilities for these new 1D points.
5. **Minimize KL-Divergence:** The algorithm then iteratively moves the points y1, y2, y3 on the line. At each step, it nudges them in a direction that makes the output probabilities q look more like the input probabilities p. This process of nudging is called **gradient descent**, and it continues until the KL-divergence between the p and q distributions is minimized.

How MDS Works (Example):

1. **Input Distances:** MDS's goal is to preserve distances. It calculates the actual Euclidean distances: d(p1,p2)≈3.16, d(p1,p3)≈2.83, d(p2,p3)≈3.16.
2. **Output Space:** It randomly places the points on a 1D line, e.g., y1=1.5, y2=-0.5, y3=0.2.
3. **Output Distances:** It calculates the new distances: d(y1,y2)=2.0, d(y1,y3)=1.3, d(y2,y3)=0.7.
4. **Minimize Stress:** It compares the original distances to the new distances. The difference is the "stress." The algorithm iteratively moves y1, y2, y3 on the line to make the new distances as close to the original distances as possible, thereby minimizing the stress.

How PCA Works (Example):

1. **Center the Data:** First, find the mean (centroid) of the points: mean = ((2+3+0)/3, (1+4+3)/3) = (5/3, 8/3) ≈ (1.67, 2.67). Subtract this mean from each point.
2. **Find Principal Component 1 (PC1):** PCA finds the direction (a line) through the centered data that **maximizes the variance** (the spread) of the points when they are projected onto it. For these three points, this will be the line that best fits the triangle they form.
3. **Project:** It takes the three original points and projects them onto this PC1 line. The position of each point's projection along this line becomes its new single coordinate. This projection **minimizes the reconstruction error**.

Question 5 & 6: SNE and Soft Neighborhoods

Why is a distance-based metric not suitable for SNE?
Because SNE is not trying to preserve distance; it's trying to preserve **neighborhood probabilities**.

- **Analogy:** MDS tries to create a map where if two cities were 100 miles apart, they are now 10 cm apart on the map. SNE tries to create a map where if City B was the *closest neighbor* to City A, it remains the *closest neighbor* on the map, even if their scaled distance isn't perfect. It cares about the *ranking* of neighbors, not their absolute distances. KL-divergence is the perfect tool for comparing these probability rankings, whereas a simple distance metric is not.

Explain Soft Neighborhood. Why is the squared distance used in the formula?

- **Soft Neighborhood:** This is a probabilistic definition of a neighborhood. Instead of a "hard" rule (a point is either in or out), every point has a **probability** of being a neighbor to every other point. This probability is very high for close points and drops off smoothly (but never reaches zero) for faraway points.
- **Why Squared Distance?** The formula $\exp(-d^2)$ is a **Gaussian (Normal) distribution**. Using the squared distance in the exponent makes the probability fall off **very rapidly** as distance increases. This is a deliberate choice: it forces the SNE algorithm to focus almost exclusively on preserving the structure of the **very local neighbors** while caring much less about the relative positions of faraway points.

Question 7 & 8: Practical PCA Questions

How do you select the output dimensionality k for PCA?
There are two primary methods:

1. **The Scree Plot (Elbow Method):** Plot the variance captured by each principal component, in descending order. The plot will typically be steep at first and then level off. The "elbow" of this curve—the point where adding another component gives very little additional information—is a good choice for k.
2. **Cumulative Variance Threshold:** Decide on a threshold for the total variance you want to preserve (e.g., 90%). Add the variance captured by PC1, PC2, PC3, etc., until the cumulative sum crosses your threshold. The number of components you needed is your chosen k.

Is it possible to know which input dimensions or genes are important using PCA?
Yes, absolutely. This is a key advantage of PCA's interpretability.

- **How:** Each principal component (e.g., PC1) is a linear combination of the original variables (genes). The coefficients of this combination are called **loadings**. A gene with a **high absolute loading value** for a given principal component is a gene that contributes heavily to that component.
- **Example:** If you run PCA and find that PC1 separates your "healthy" patients from your "cancer" patients, you would then look at the loadings for PC1. The genes with the highest loadings are the ones that are most responsible for driving the difference between the two groups, making them excellent candidates for further biological investigation.

–Fahad Nadim Ziad, 24341216