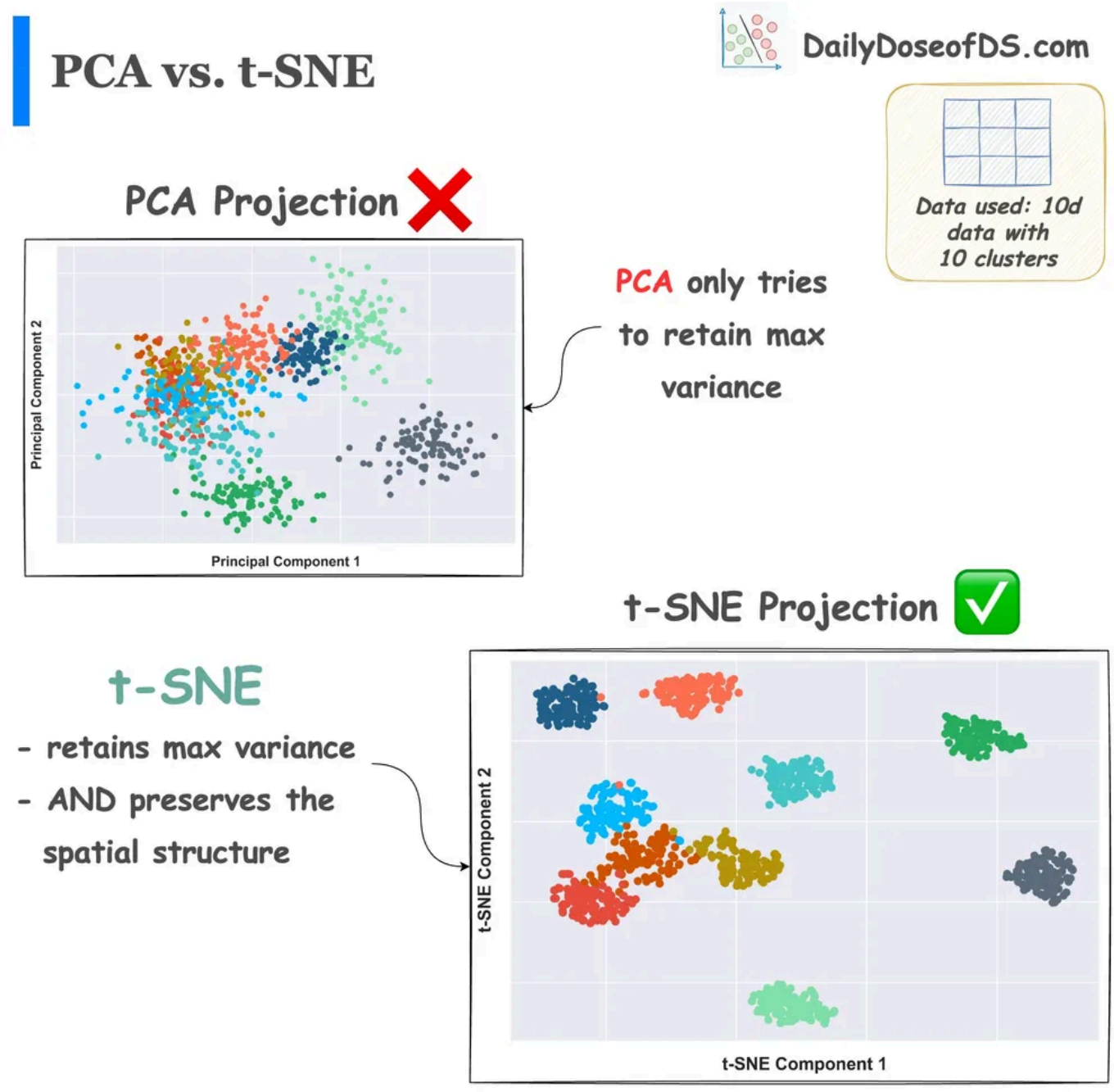
Sep 2, 2023

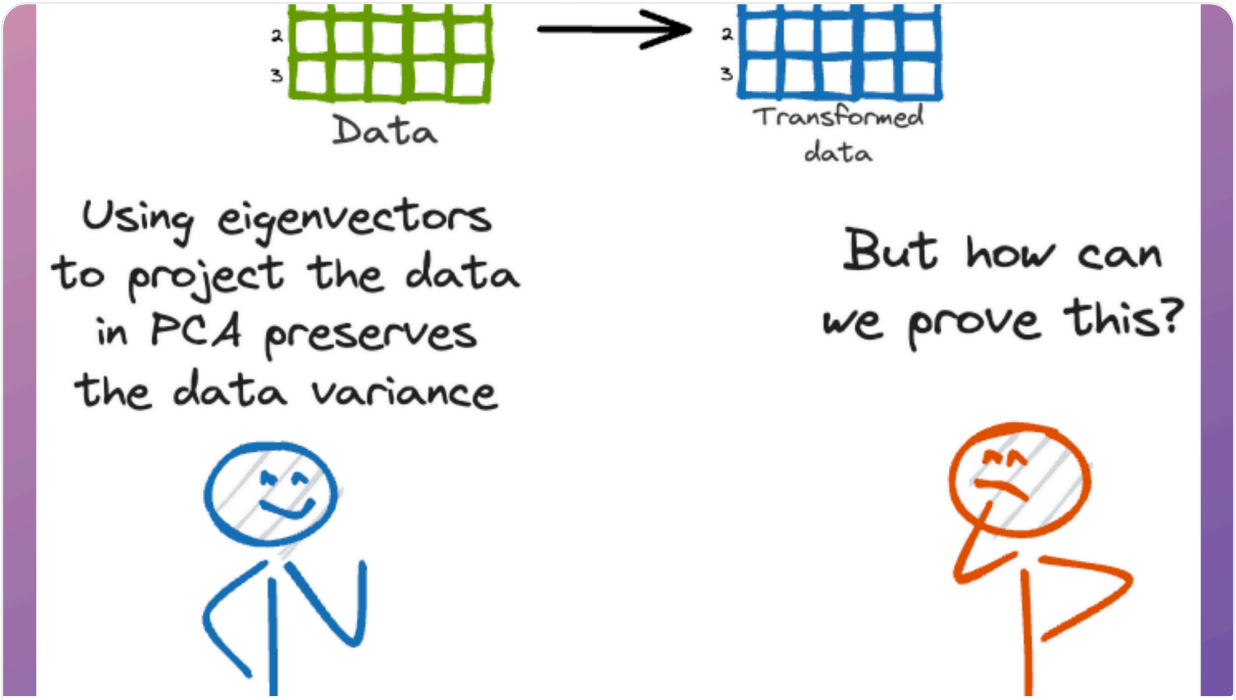
Formulating and Implementing the t-SNE Algorithm From Scratch

The most extensive visual guide to never forget how t-SNE works.

Avi Chawla



In a recent article, we devised the entire principal component analysis (PCA) algorithm from scratch.



Formulating the Principal Component Analysis (PCA) Algorithm From Scratch

Approaching PCA as an optimization problem.

Daily Dose of Data Science • Avi Chawla

We saw how projecting the data using the eigenvectors of the covariance matrix naturally emerged from the PCA optimization step.

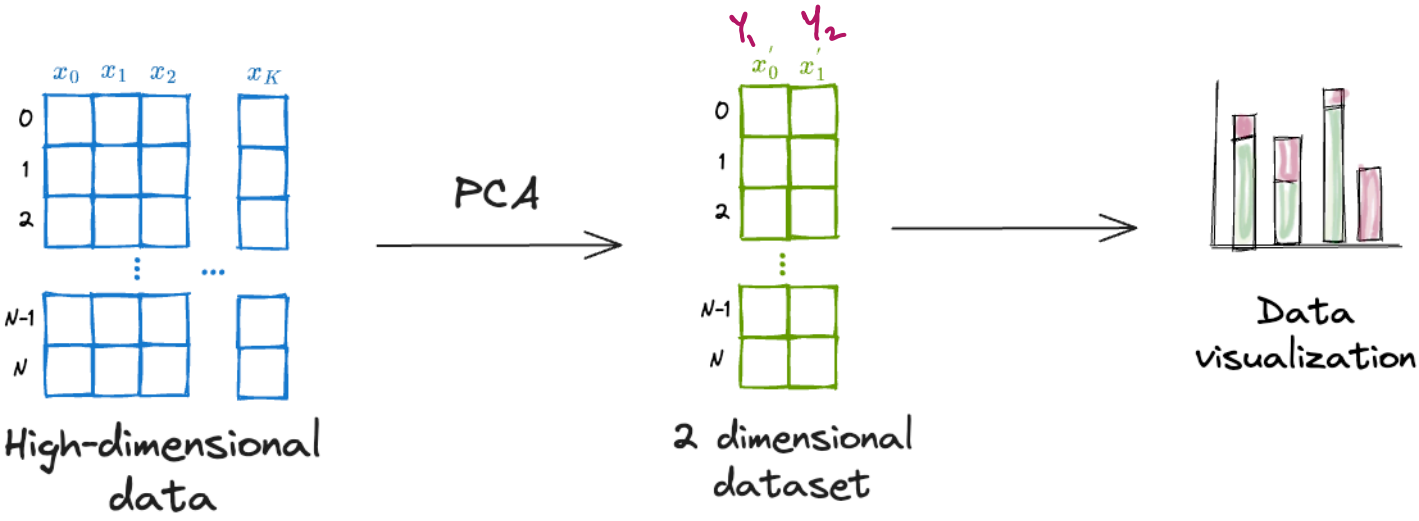
Moving on, we discussed some of its significant limitations.

Let's look at them again!

Limitations of PCA

#1) PCA for visualization

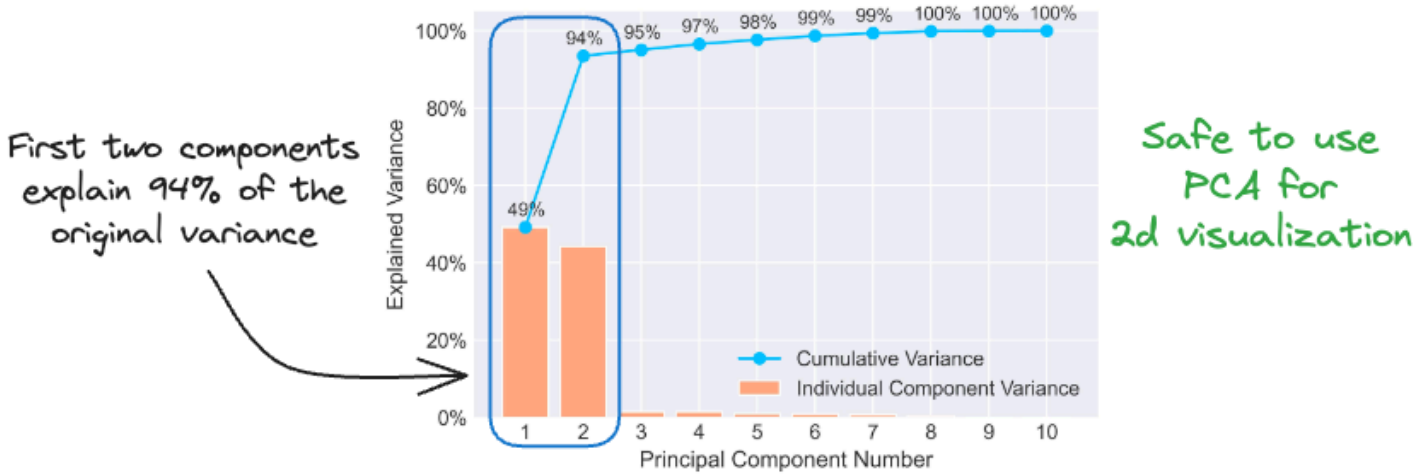
Many use PCA as a data visualization technique. This is done by projecting the given data into two dimensions and visualizing it.



While this may appear like a fair thing to do, there's a big problem here that often gets overlooked.

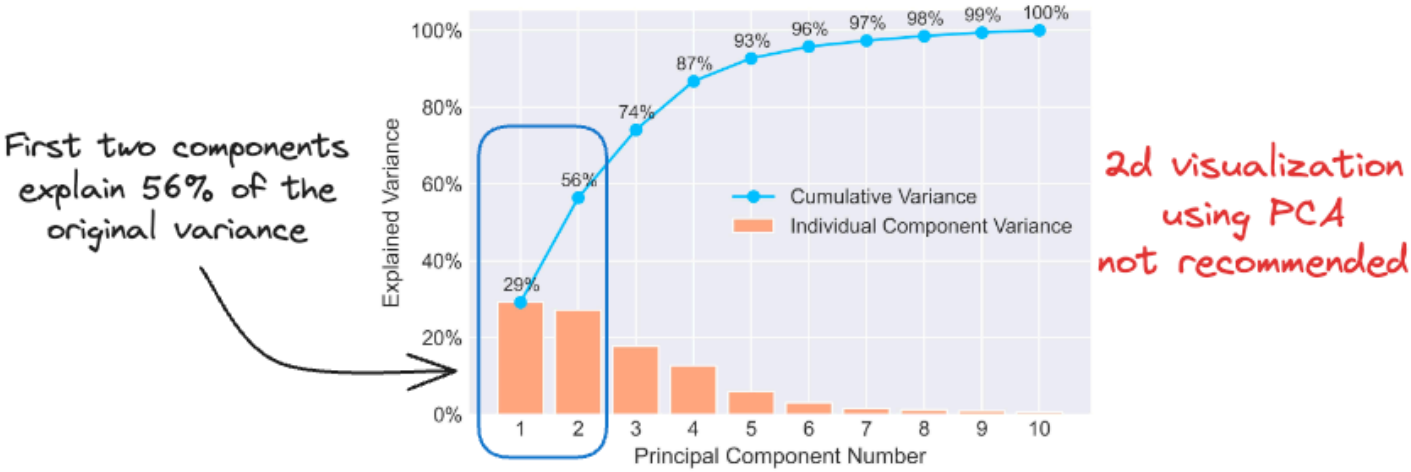
As we discussed in the PCA article, after applying PCA, each new feature captures a fraction of the original variance.

This means that two-dimensional visualization will only be helpful if the first two principal components collectively capture most of the original data variance, as shown below:



Safe to use PCA for 2D visualization

If not, the two-dimensional visualization will be highly misleading and incorrect. This is because the first two components don't capture most of the original variance well. This is depicted below:



Using PCA for 2D visualization is not recommended

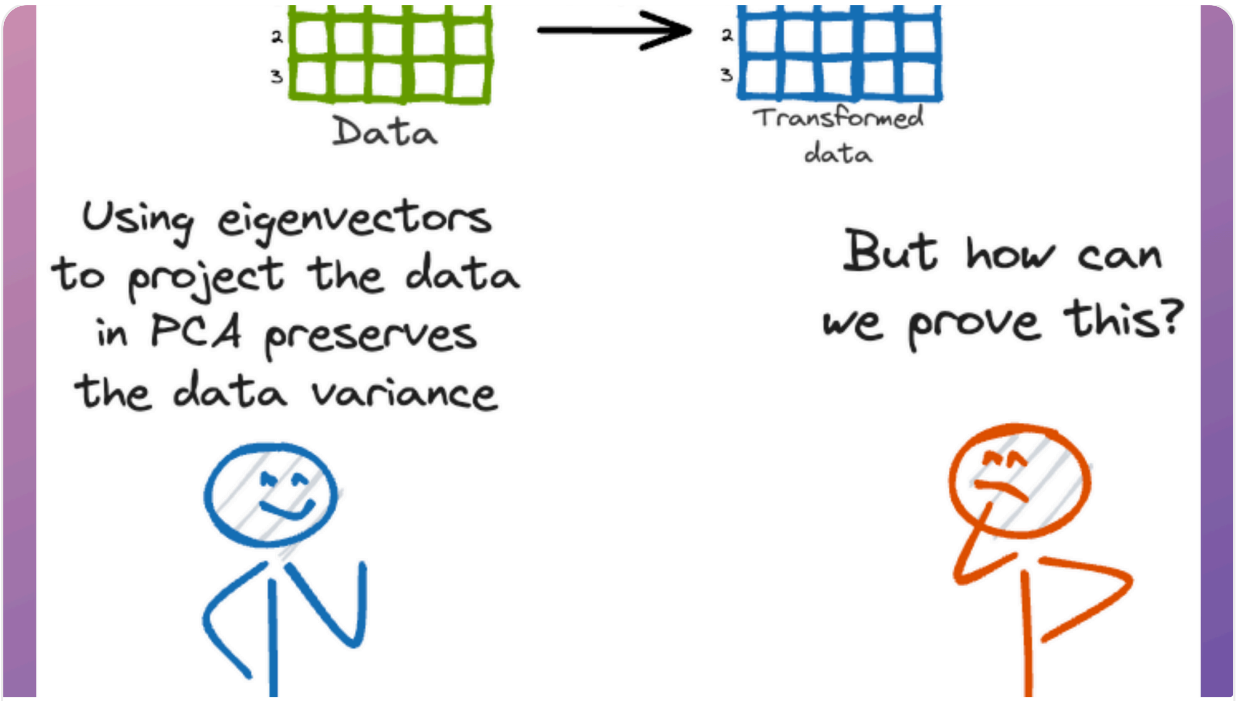
Thus, using PCA for 2D visualizations is only recommended if the cumulative explained variance plot suggests so. If not, one should refrain from using PCA for 2D visualization.

#2) PCA is a linear dimensionality reduction technique

PCA has two main steps:

- Find the eigenvectors and eigenvalues of the covariance matrix.
- Use the eigenvectors to project the data to another space.

> 👉 Why eigenvectors, you might be wondering? We discussed the origin of eigenvectors in detail in the PCA article. It is recommended to read that article before reading this article.

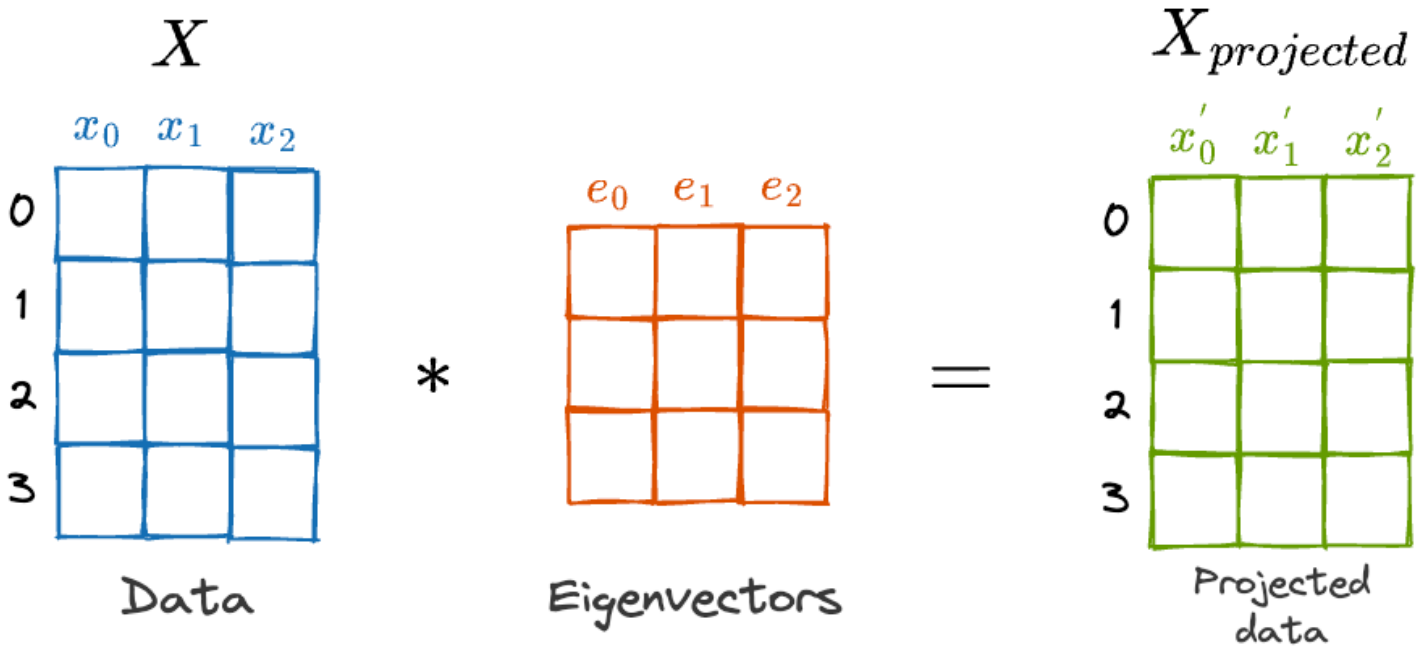


Formulating the Principal Component Analysis (PCA) Algorithm From Scratch

Approaching PCA as an optimization problem.

Daily Dose of Data Science • Avi Chawla

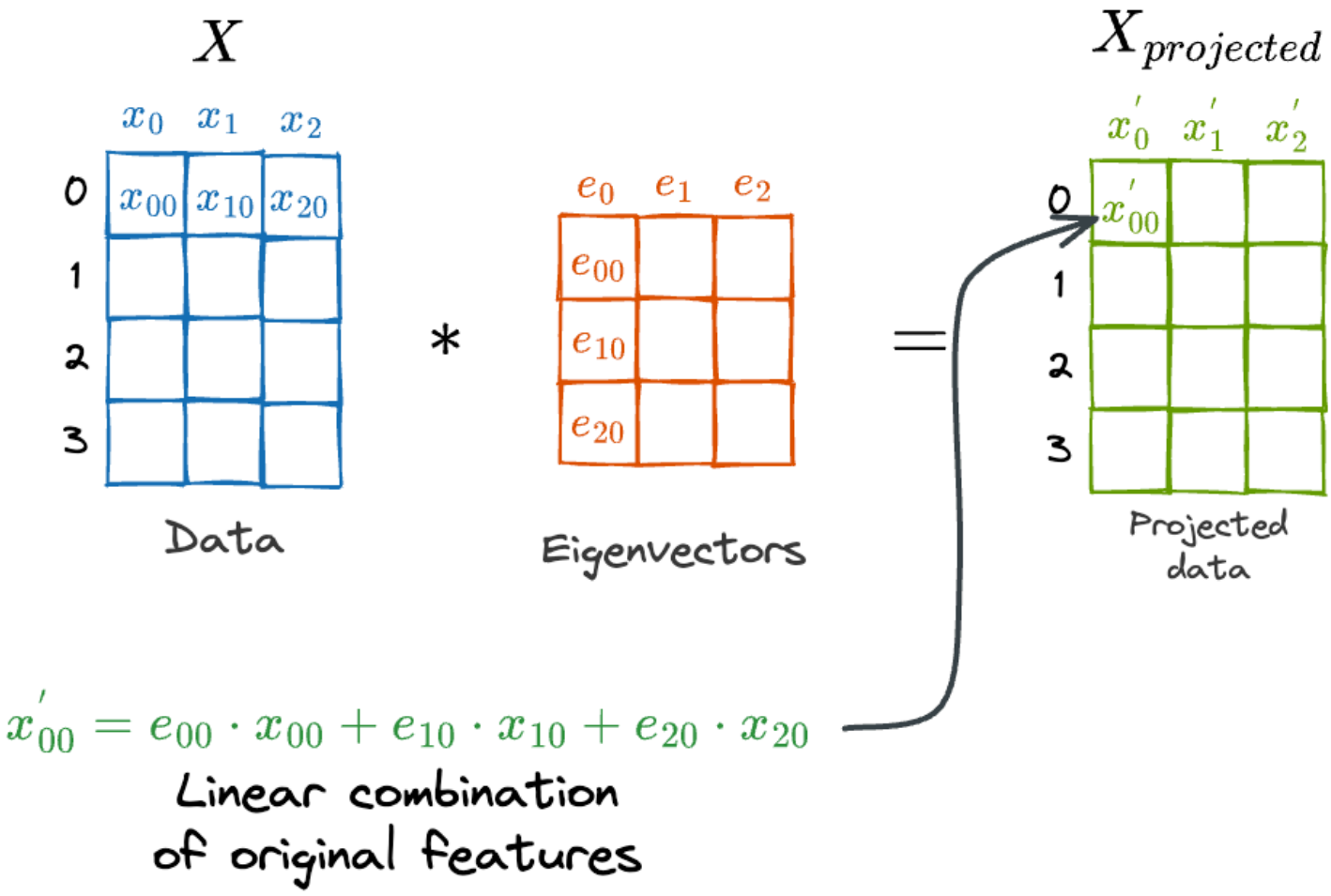
Projecting the data using eigenvectors creates uncorrelated features.



PCA projections using eigenvectors

Nonetheless, the new features created by PCA (x'_0, x'_1, x'_2) are always a linear combination of the original features (x_0, x_1, x_2).

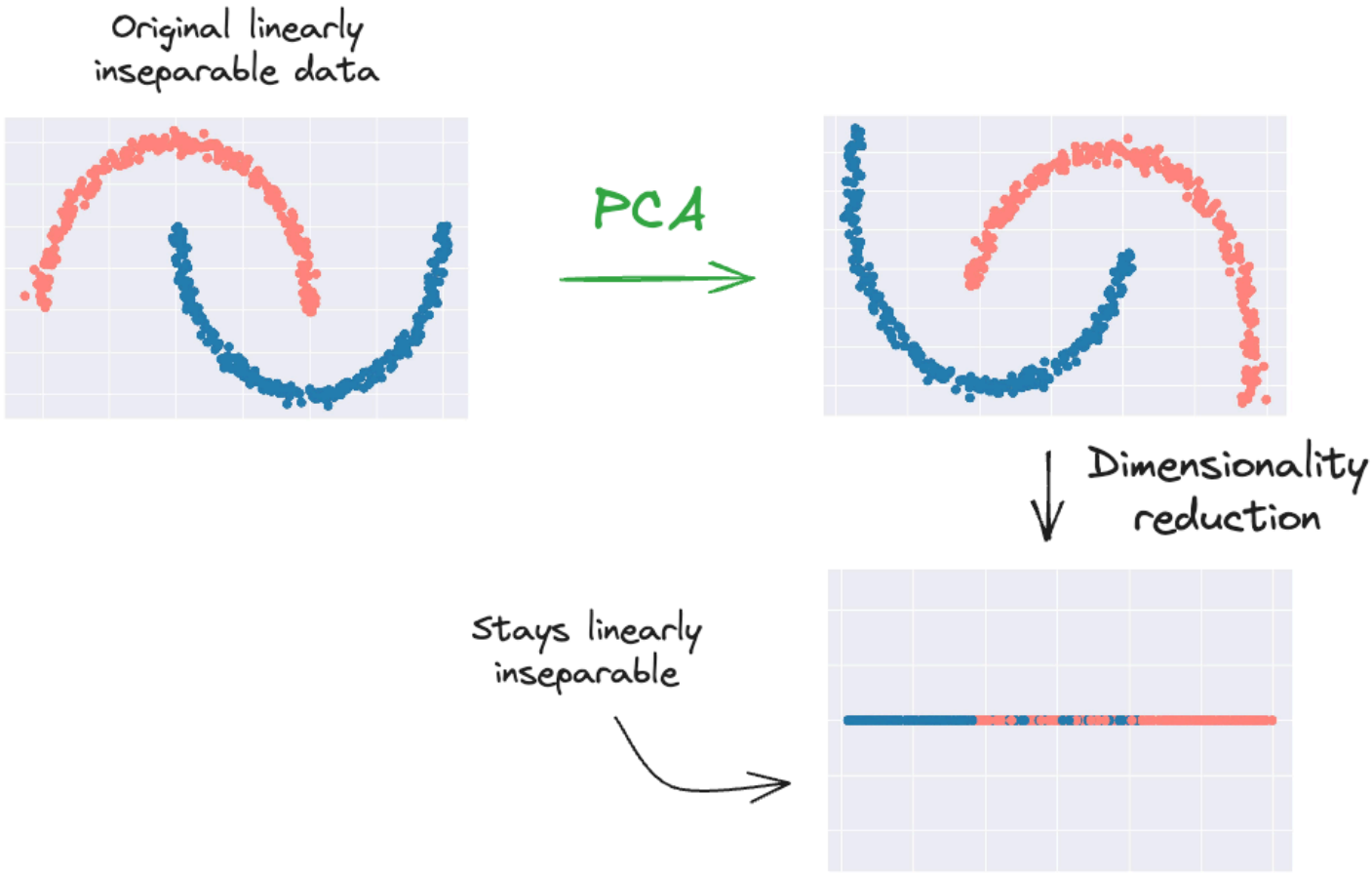
This is depicted below:



PCA creates linear features

As shown above, every new feature in $X_{projected}$ is a linear combination of the features in X.

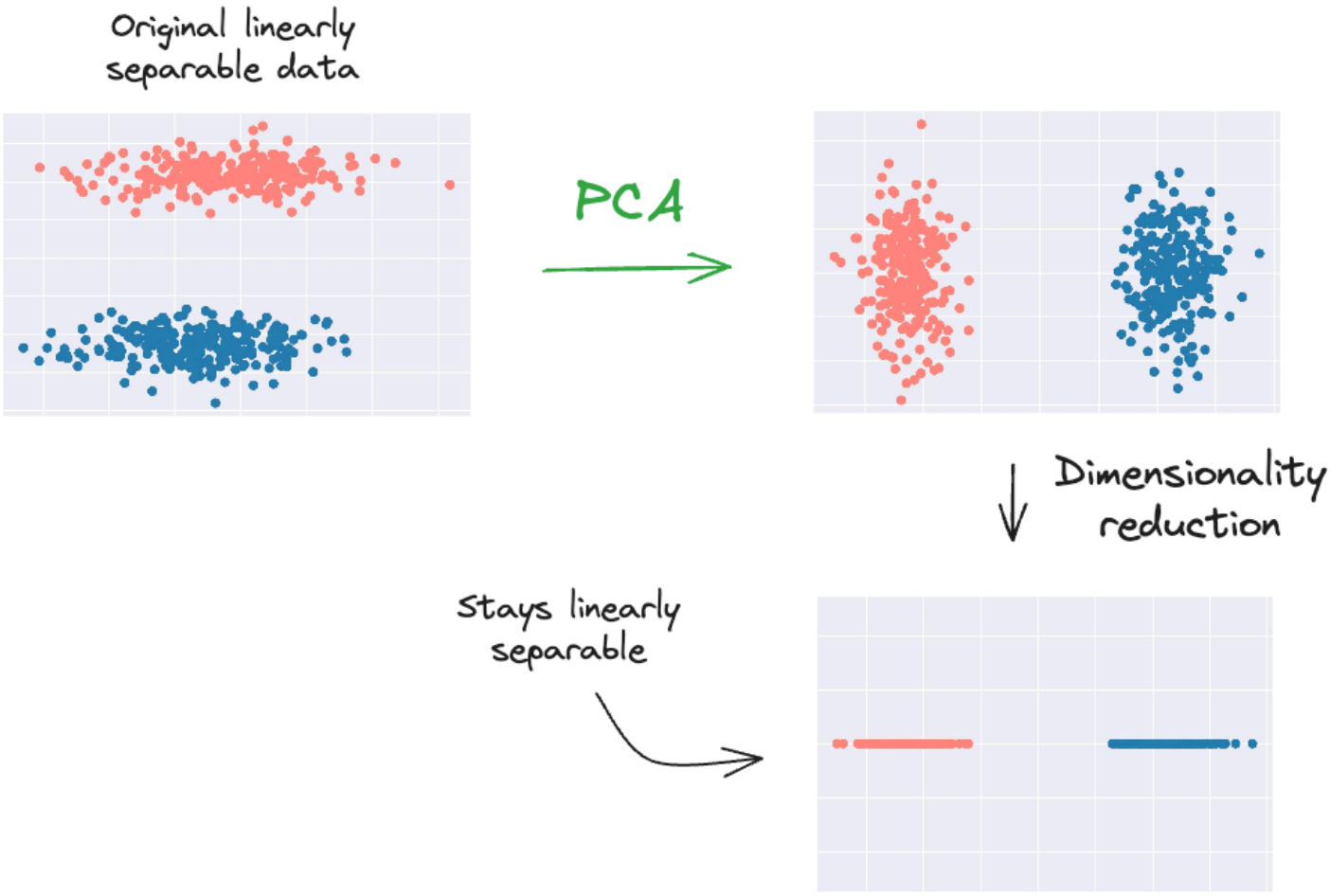
We can also prove this experimentally.



PCA is a linear technique

As depicted above, we have a linearly inseparable dataset. Next, we apply PCA and reduce the dimensionality to 1. The dataset remains linearly inseparable.

On the flip side, if we consider a linearly separable dataset, apply PCA, and reduce the dimensions, we notice that the dataset remains linearly separable:



Linearly separable dataset stays linearly separable

This proves that PCA is a linear dimensionality reduction technique.

However, not all real-world datasets are linear. In such cases, PCA will underperform.

#3) PCA only focuses on the global structure

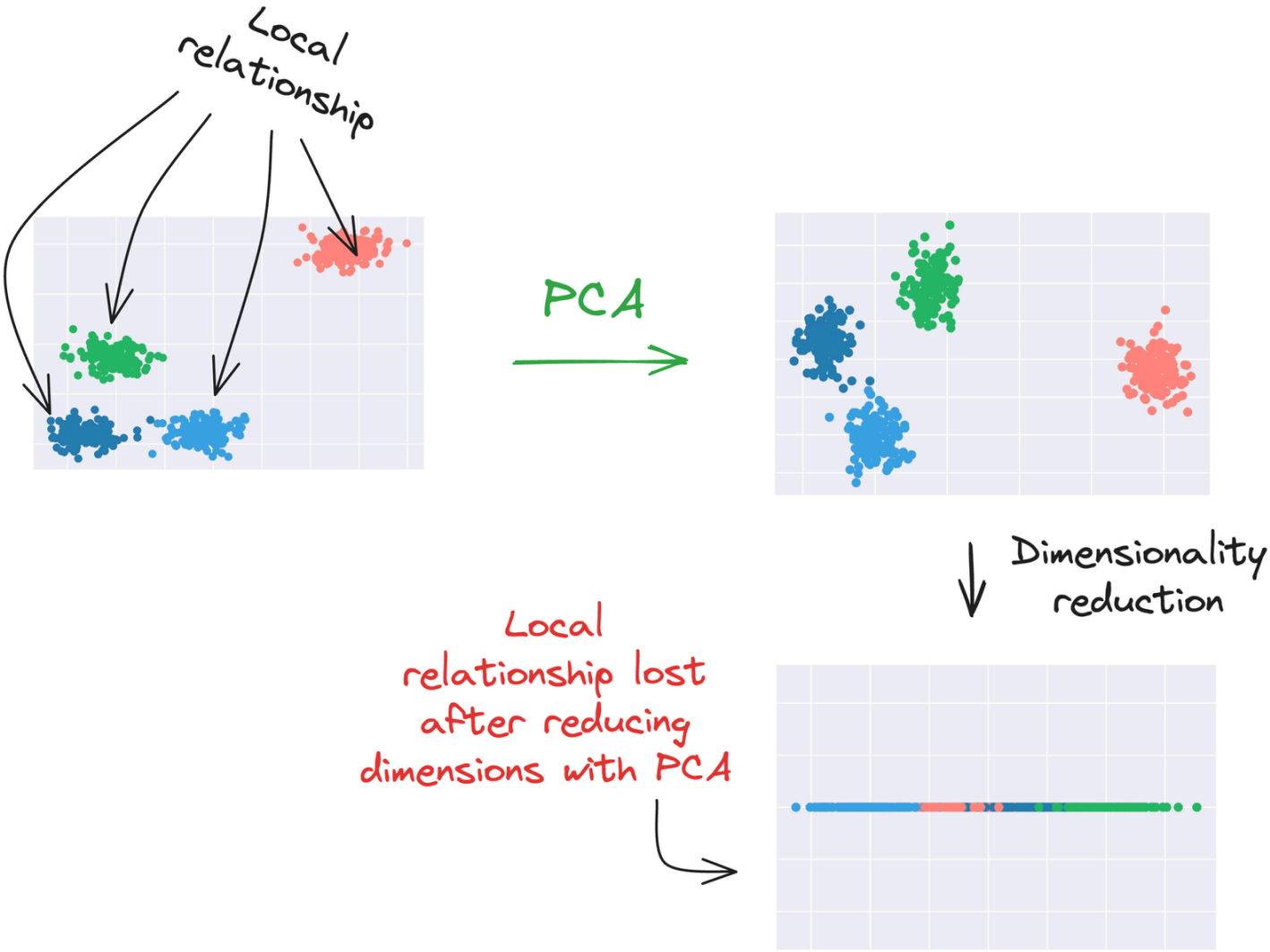
As discussed above, PCA's primary objective is to capture the overall (or global) data variance.

In other words, PCA aims to find the orthogonal axes along which the **entire dataset** exhibits the most variability.

Thus, during this process, it does not pay much attention to the local relationships between data points.

It inherently assumes that the global patterns are sufficient to represent the overall data variance.

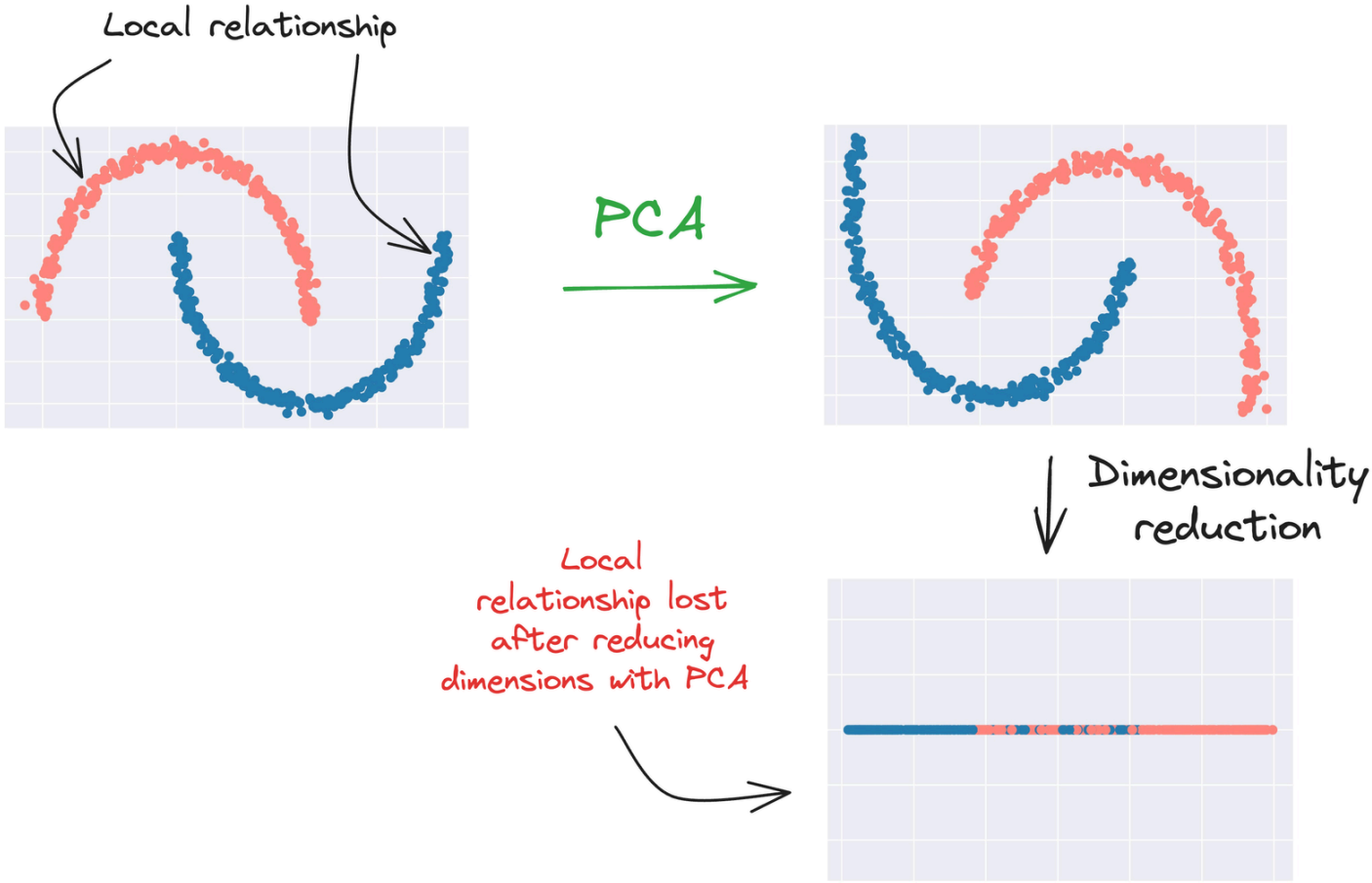
This is demonstrated below:



PCA does not focus on local structure

Because of primarily emphasizing on the global structure, PCA is not ideal for visualizing complex datasets where the underlying structure might rely on local relationships or pairwise similarities.

In cases where the data is nonlinear and contains intricate clusters or groups, PCA can fall short of preserving these finer details, as shown in another illustration below:



Local relationships are lost during PCA

As depicted above, data points from different clusters may overlap when projected onto the principal components.

This leads to a loss of information about intricate relationships within and between clusters.

So far, we understand what's specifically lacking in PCA.

While the overall approach is indeed promising, its limitations make it impractical for many real-world datasets.

What is t-SNE?

t-distributed stochastic neighbor embedding (t-SNE) is a powerful dimensionality reduction technique **mainly used to visualize high-dimensional datasets** by projecting them into a lower-dimensional space (typically 2-D).

As we will see shortly, t-SNE addresses each of the above-mentioned limitations of PCA:

- It is well-suited for visualization.
- It works well for linearly inseparable datasets.
- It focuses beyond just capturing the global data relationships.

t-SNE is an improvement to the **Stochastic Neighbor Embedding** (SNE) algorithm. It is observed that in comparison to SNE, t-SNE is much easier to optimize.

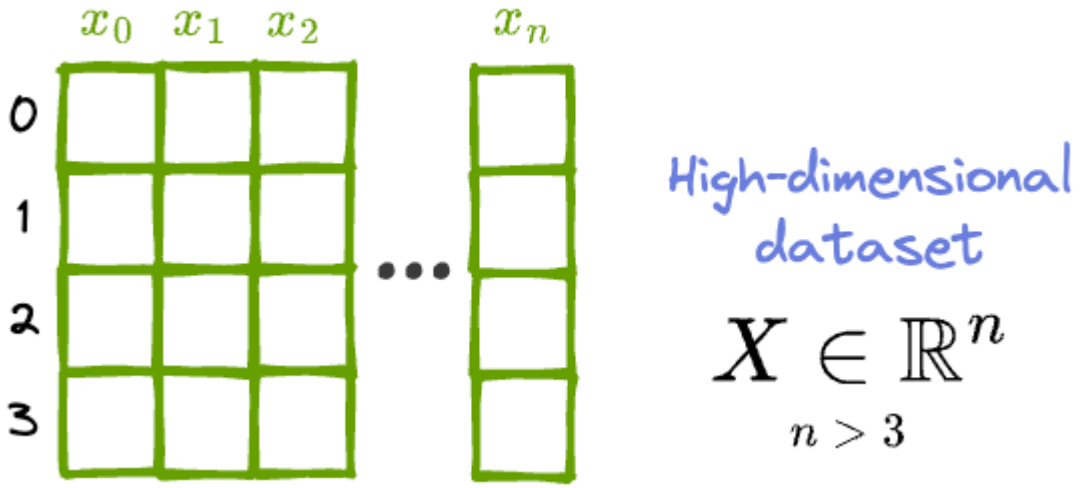
> 💡 t-SNE and SNE are two different techniques, both proposed by one common author — Geoffrey Hinton.

So, before getting into the technical details of t-SNE, let's spend some time understanding the SNE algorithm instead.

Background: Stochastic Neighbor Embedding (SNE) algorithm

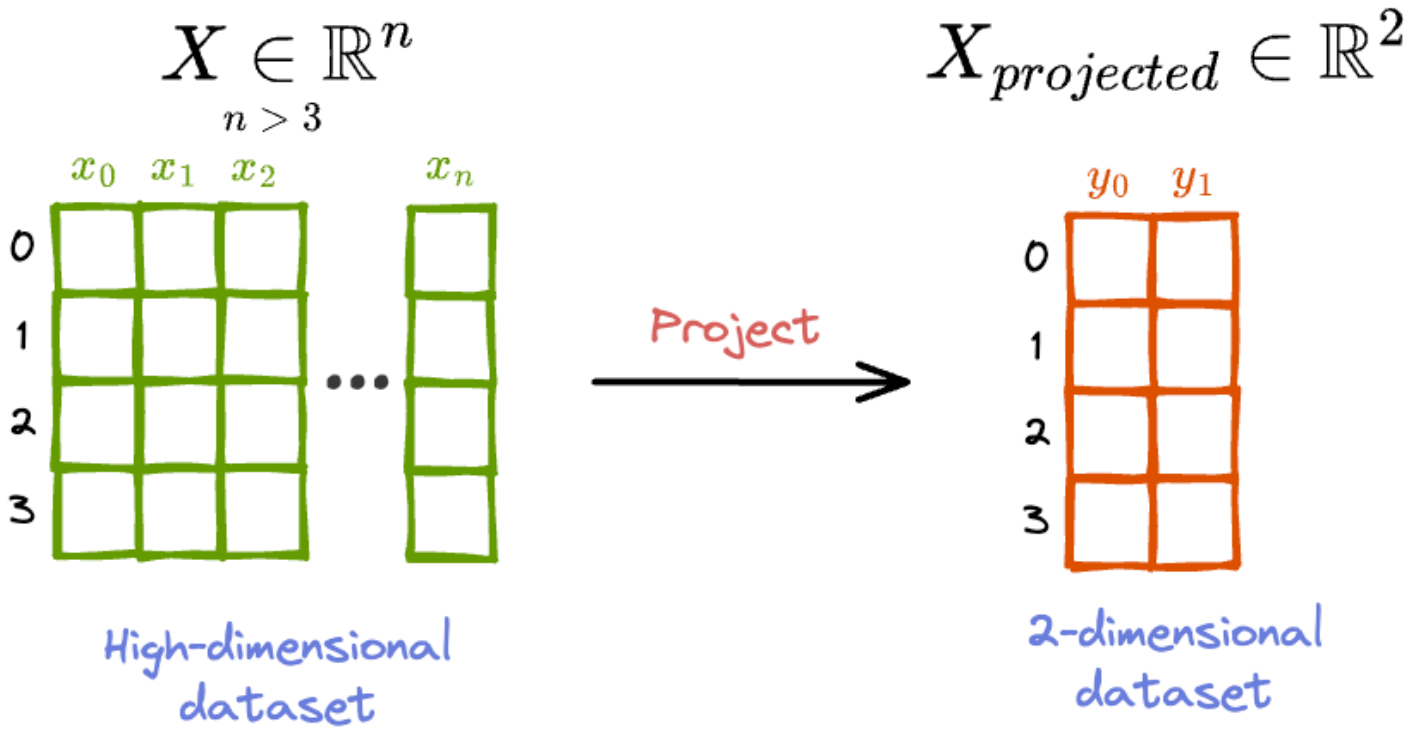
To begin, we are given a high-dimensional dataset, which is difficult to visualize.

Thus, the dimensionality is higher than 3 (typically, it's much higher than 3).



High-dimensional dataset

The objective is to project it to a lower dimension (say, 2 or 3), such that the lower-dimensional representation preserves as much of the local and global structure in the original dataset as possible.

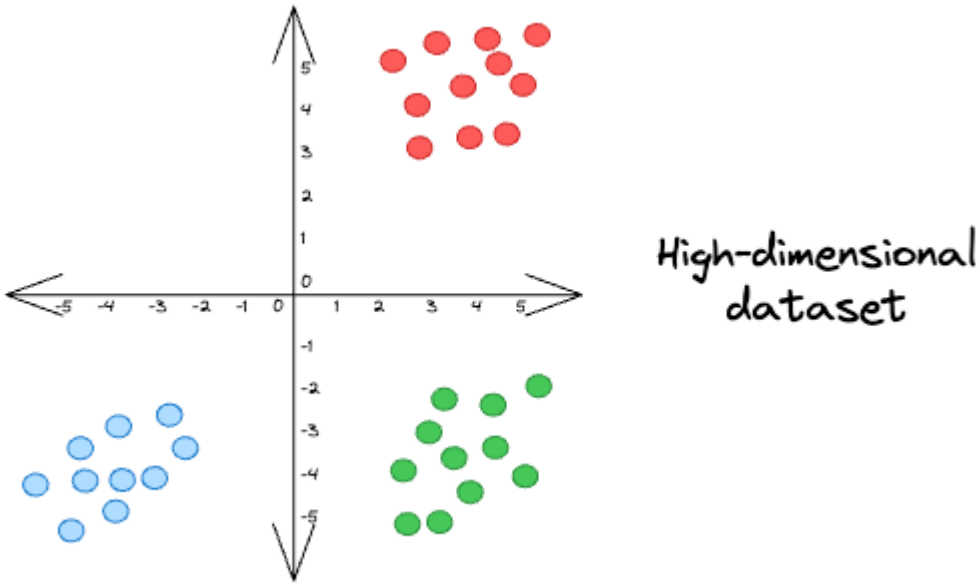


Converting a high-dimensional dataset to 2D

What do we mean by a local and global structure?

Let's understand!

Imaging this is our high-dimensional dataset:



A dummy high-dimensional dataset

💡 Of course, the above dataset is not a high-dimensional dataset. But for the sake of simplicity, let's assume that it is.

Local structure, as the name suggests, refers to the arrangement of data points that are close to each other in the high-dimensional space.

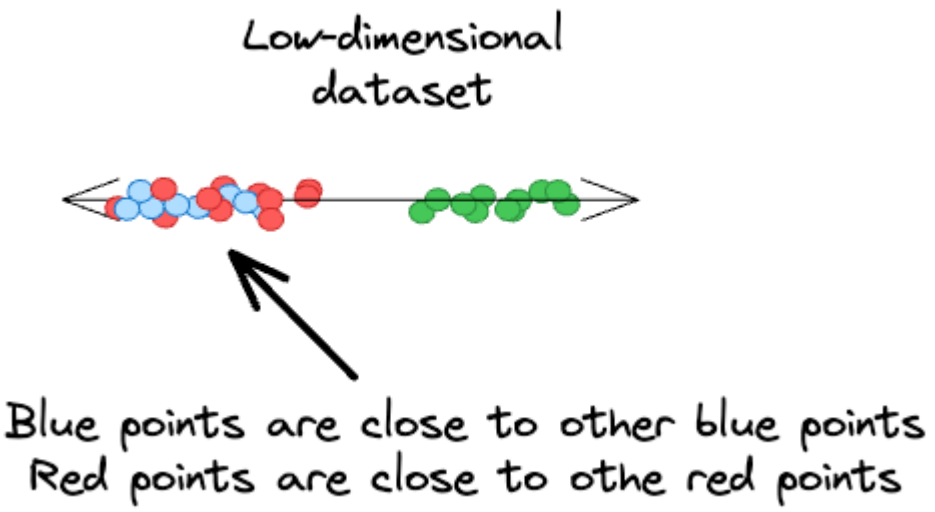
Thus, preserving the local structure would mean that:

- Red points should stay closer to other red points.
- Blue points should stay closer to other blue points.
- Green points should stay closer to other green points.

So, is preserving the local structure sufficient?

Absolutely not!

If we were to focus solely on preserving the local structure, it may lead to a situation where blue points indeed stay closer to each other, but they overlap with the red points, as shown below:



No preservation of the global structure

This is not desirable.

Instead, we also want the low-dimensional projections to capture the global structure.

Thus, preserving the global structure would mean that:

- The red cluster is well separated from the other cluster.
- The blue cluster is well separated from the other cluster.
- The green cluster is well separated from the other cluster.

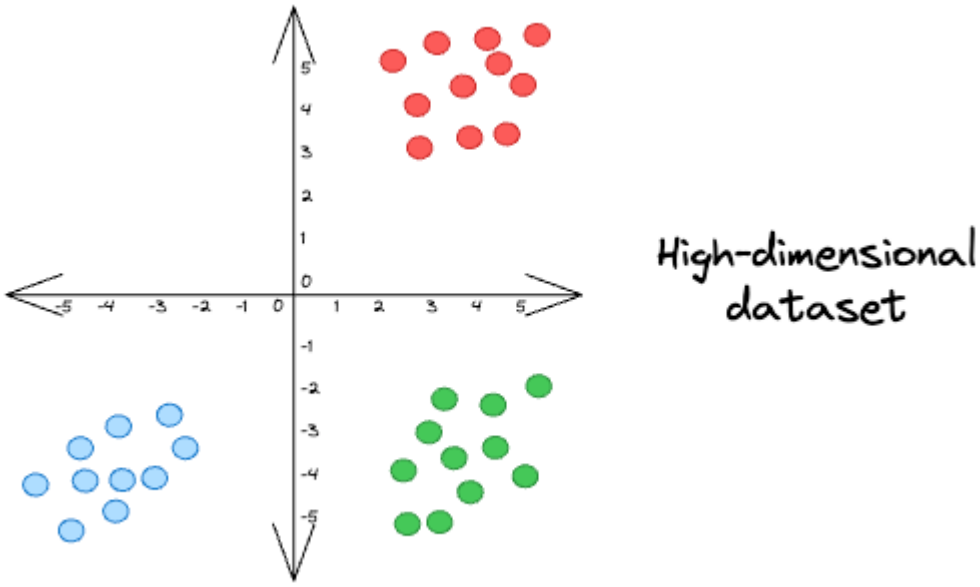
To summarize:

- Preserving the local structure means maintaining the relationships among nearby data points within each cluster.
- Preserving the global structure involves maintaining the broader trends and relationships that apply across all clusters.

How does SNE work?

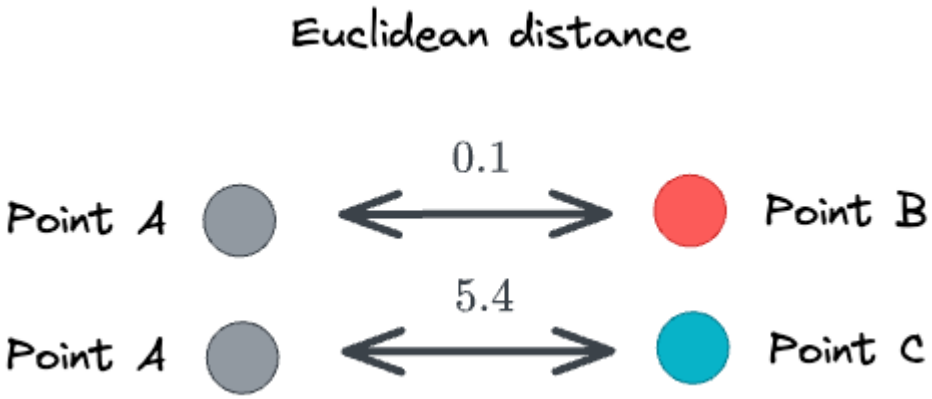
Let's stick to understanding this as visually as possible.

Consider the above high-dimensional dataset again.



A dummy high-dimensional dataset

Euclidean distance is a good measure to know if two points are close to each other or not.



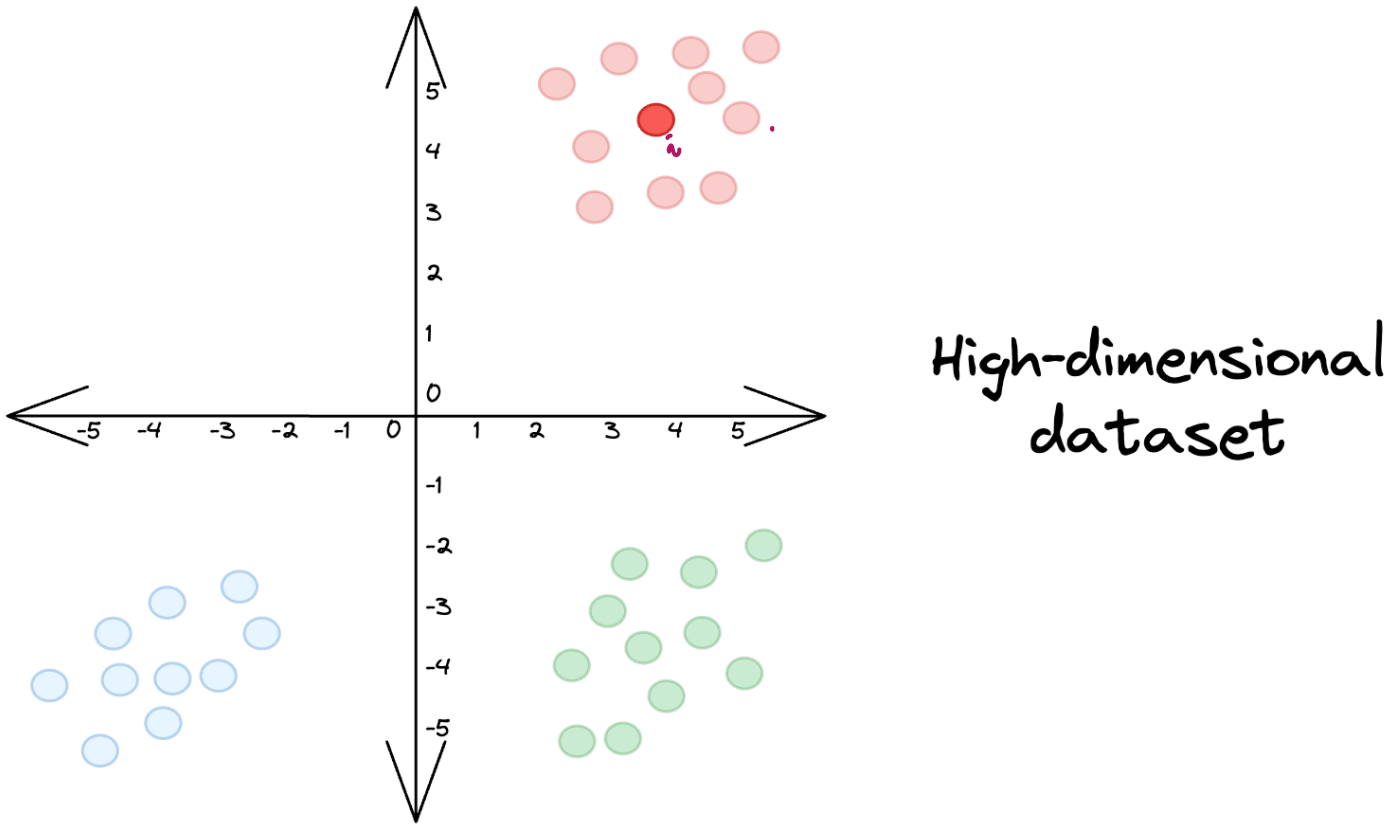
Euclidean distance between points

For instance, in the figure above, it is easy to figure out that `Point A` and `Point B` are close to each other but `Point C` is relatively much distant from `Point A`.

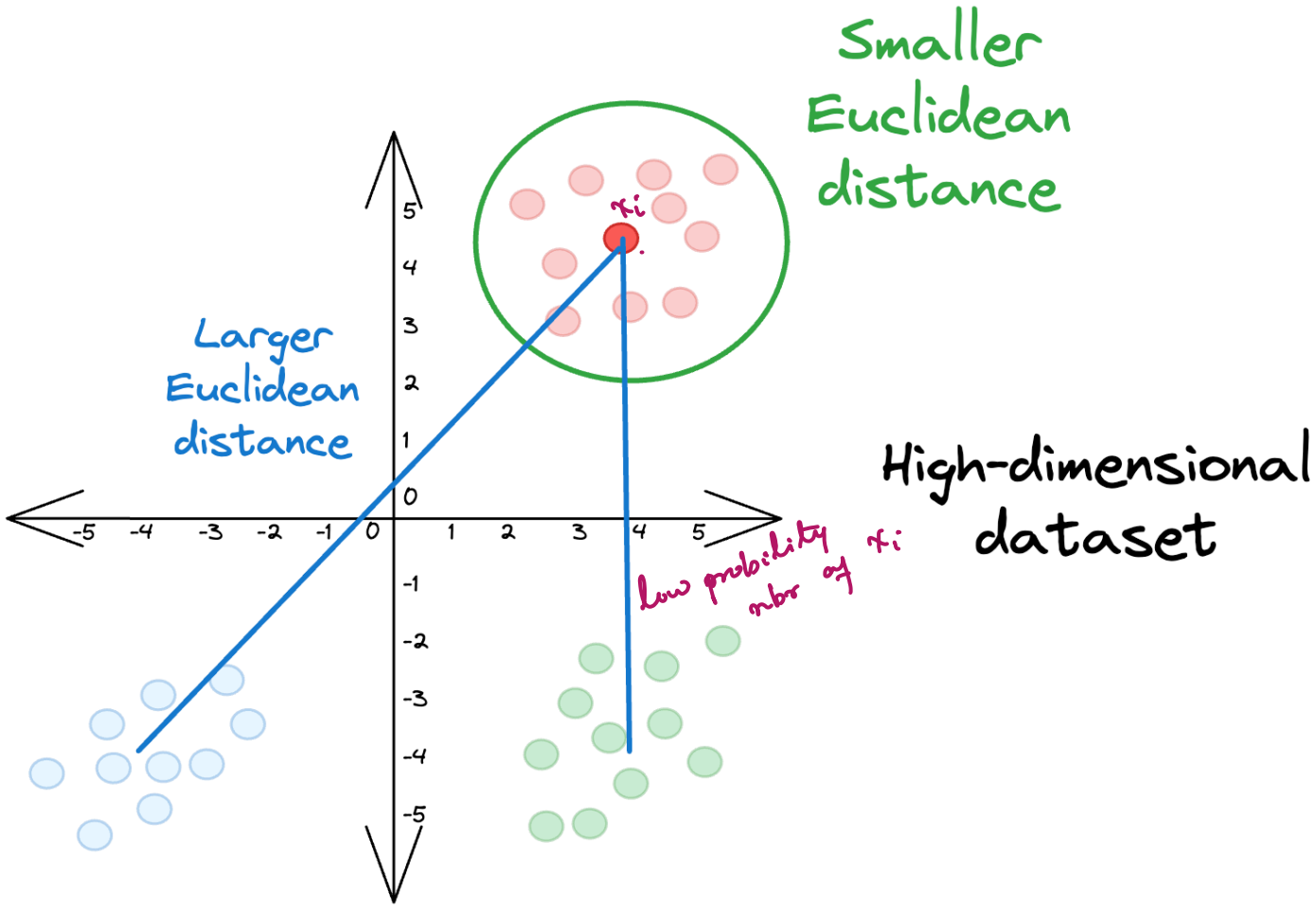
Thus, the first step of the **SNE algorithm** is to convert these high-dimensional Euclidean distances between data points into conditional probabilities that represent similarities.

Step 1: Define the conditional probabilities in high-dimensional space

For better understanding, consider a specific data point in the above dataset:



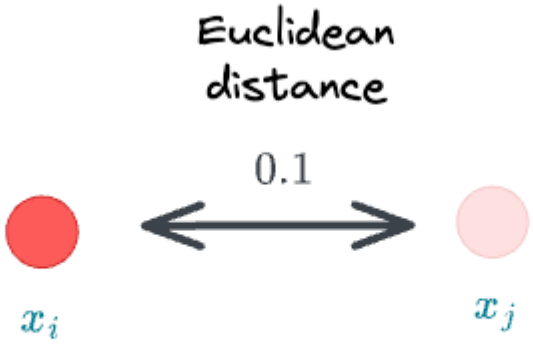
Here, points that are closer to the marked point will have smaller Euclidean distances, while other points that are far away will have larger Euclidean distances.



Euclidean distance from current points to other points

Thus, as mentioned above, for every data point (i), the **SNE algorithm** first converts these high-dimensional Euclidean distances into conditional probabilities $p_{j|i}$.

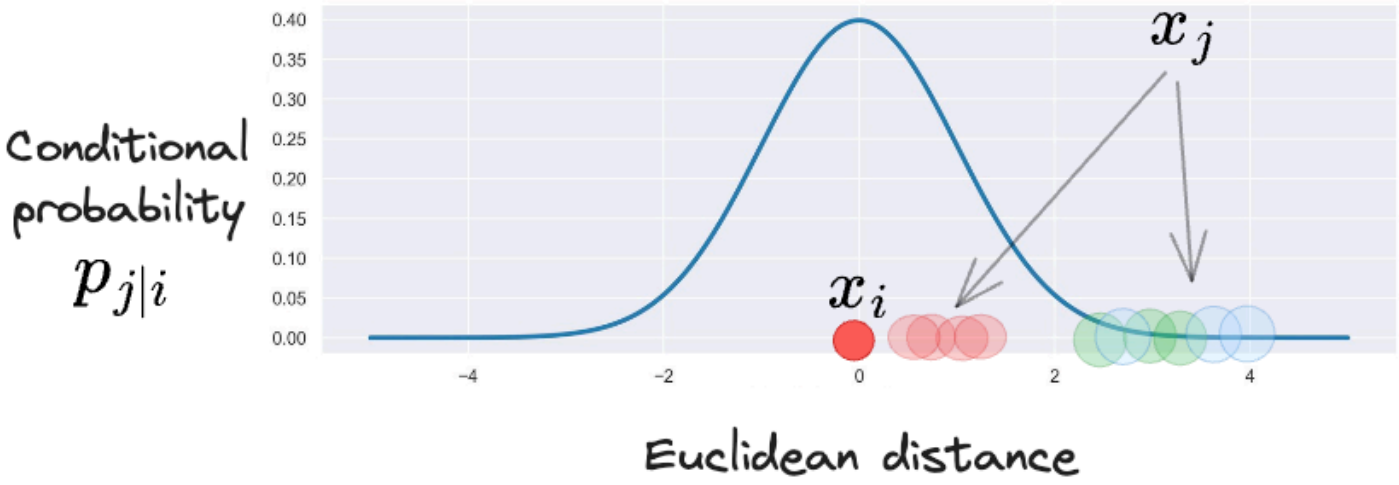
Here, $p_{j|i}$ represents the conditional probability that a point x_i will pick another point x_j as its neighbor.



$p_{j|i} \rightarrow$ point i will select point j as its neighbor.

This conditional probability is assumed to be proportional to the probability density of a Gaussian centered at x_i.

This makes intuitive sense as well. To elaborate further, consider a Gaussian centered at x_i.



It is evident from the above Gaussian distribution centered at x_i that:

- For points near x_i, $p_{j|i}$ will be relatively high.
- For points far from x_i, $p_{j|i}$ will be small.

So, to summarize, for a data point x_i, we convert its Euclidean distances to all other points x_j into conditional probabilities $p_{j|i}$.

This conditional probability is assumed to be proportional to the probability density of a Gaussian centered at x_i.

Also, as we are **only interested in modeling pairwise similarities**, we set the value of $p_{i|i} = 0$. In other words, a point cannot be its own neighbor.

> 💡 A Gaussian for a point x_i will be parameterized by two parameters: (μ_i, σ_i^2). As the x-axis of the Gaussian measures Euclidean distance, the mean (μ_i) is zero. What about σ_i^2? We'll get back to it shortly. Until then, just assume that we have somehow figured out the ideal value σ_i^2.

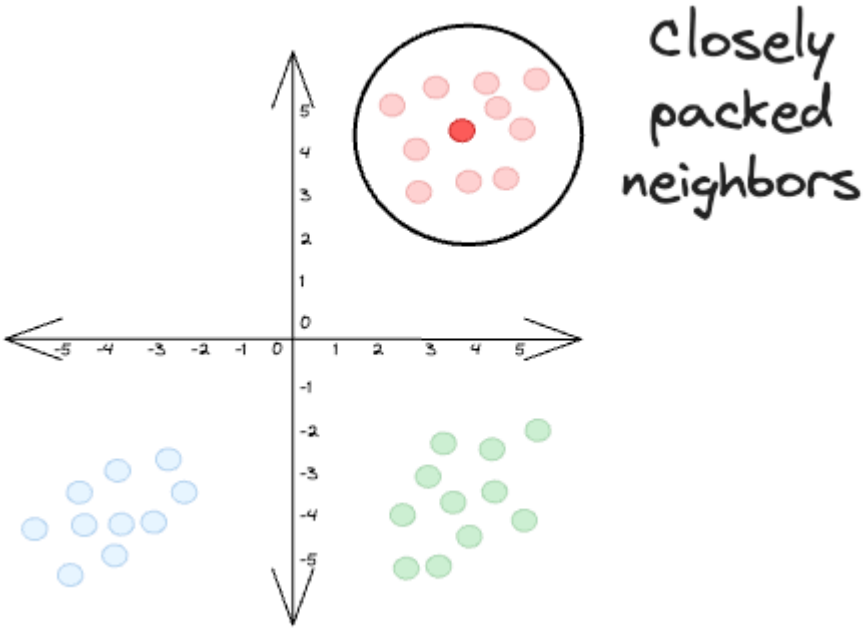
Based on what we have discussed so far, our conditional probabilities $p_{j|i}$ may be calculated using a Gaussian probability density function as follows:

$$p_{j|i} \propto exp(-\frac{||x_i - x_j||^2}{2\sigma_i^2})$$

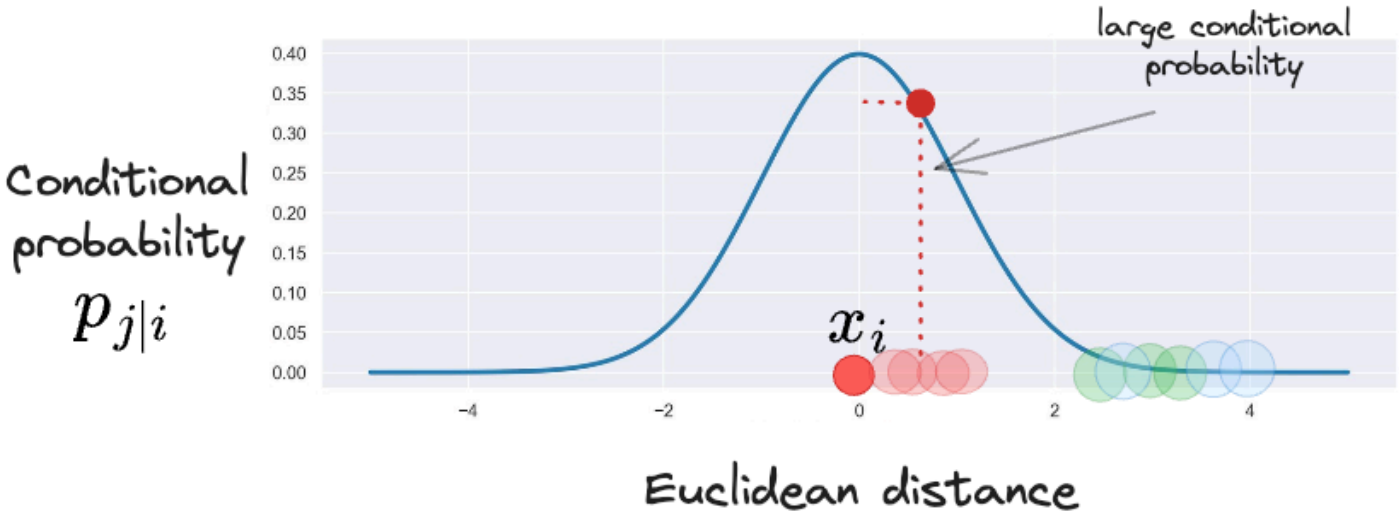
However, there's a problem here.

Yet again, let's understand this visually.

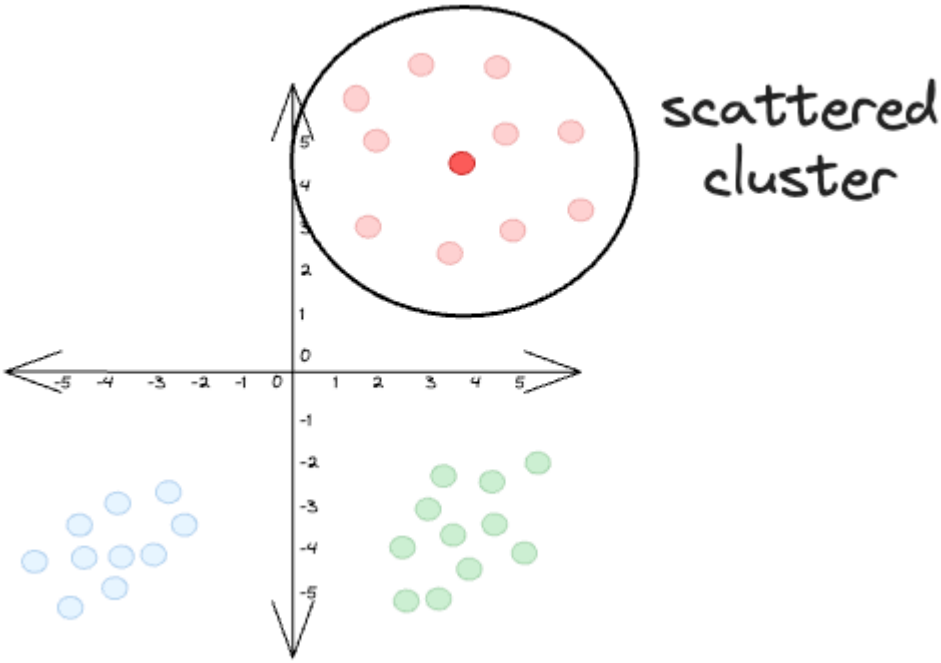
Earlier, we considered the data points in the same cluster to be closely packed.



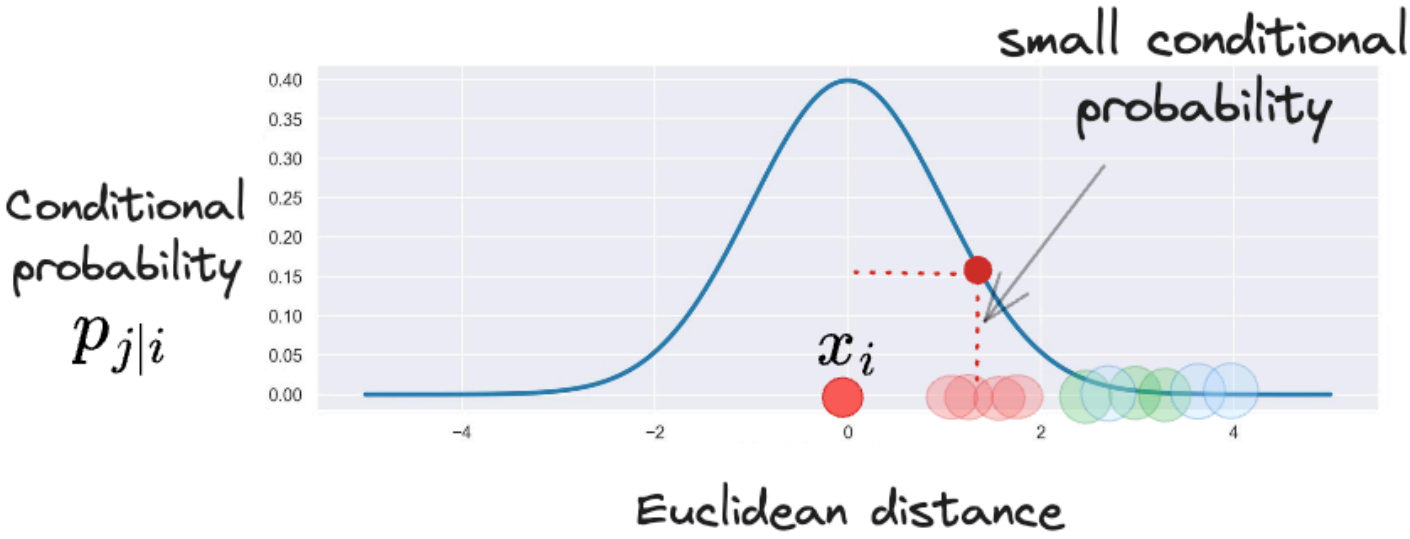
Thus, the resultant conditional probabilities were also high:



However, the data points of a cluster might be far from other clusters. Yet, they themselves can be a bit more scattered, as depicted below:



If we were to determine the resultant conditional probabilities $p_{j|i}$ for the marked data point x_i, we will get:



In this case, even though the data points belong to the same cluster, the conditional probabilities are much smaller than what we had earlier.

We need to fix this, and a common way to do this is by normalizing the individual conditional probability between $(x_i, x_j) \rightarrow p_{j|i}$ by the sum of all conditional probabilities $p_{k|i}$.

Thus, we can now estimate the final conditional probability $p_{j|i}$ as follows:

$$p_{j|i} = \frac{p_{j|i}}{\sum_{k \neq i} p_{k|i}}$$

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

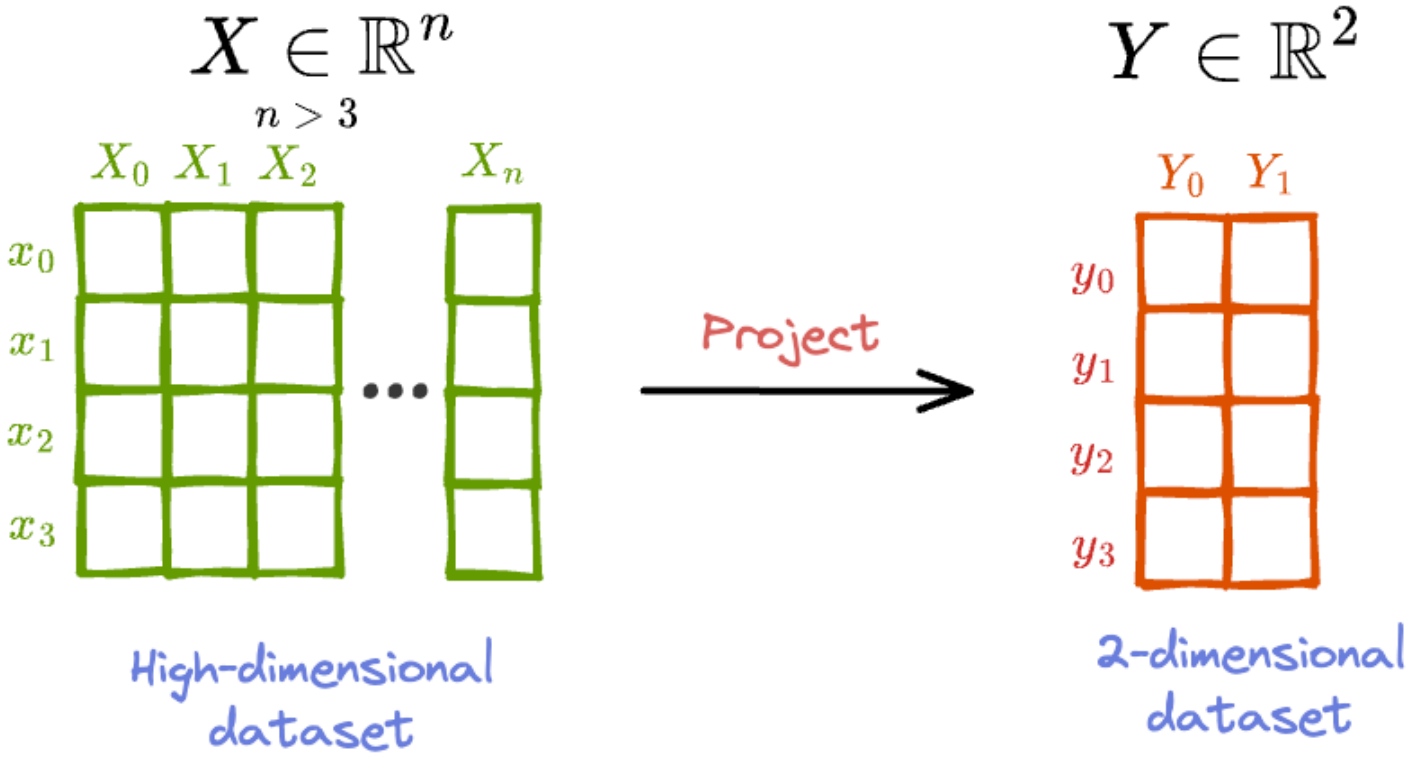
- The numerator is the conditional probability between $(x_i, x_j) \rightarrow p_{j|i}$.
- Each of the terms inside the summation in the denominator is the conditional probability between $(x_i, x_k) \rightarrow p_{k|i}$.

To reiterate, we are **only interested in modeling pairwise similarities**. Thus, we set the value of $p_{i|i} = 0$. In other words, a point cannot be its own neighbor.

Step 2: Define the low-dimensional conditional probabilities

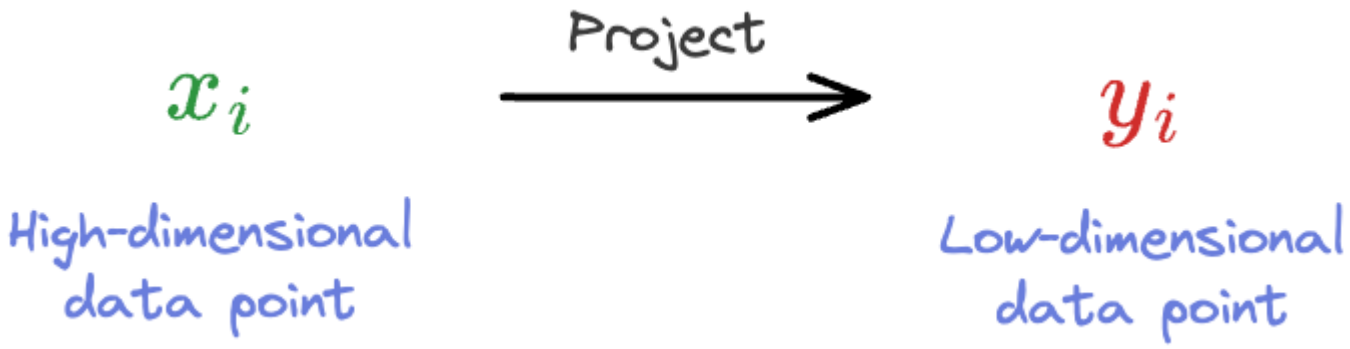
Recall the objective again.

We intend to project the given high-dimensional dataset to lower dimensions (say, 2 or 3).



Thus, for every data point $x_i \in \mathbb{R}^n$, we define it's counterpart $y_i \in \mathbb{R}^2$:

> 💡 y_i does not necessarily have to be in two dimensions. Here, we have defined $y_i \in \mathbb{R}^2$ just to emphasize that n is larger than 2.



Next, we use a similar notion to compute the pairwise conditional probability using a Gaussian, which we denote as $q_{j|i}$ in the low-dimensional space.

Furthermore, to simplify calculations, we set the variance of the conditional probabilities $q_{j|i}$ to $\frac{1}{\sqrt{2}}$.

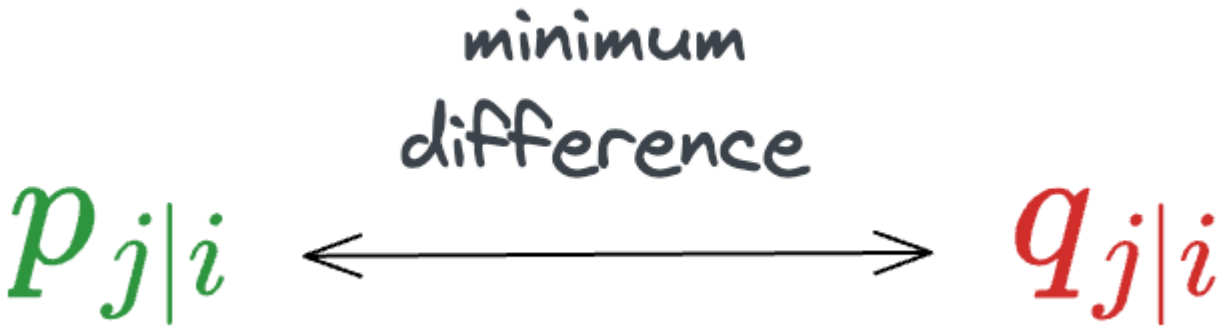
As a result, we can denote $q_{j|i}$ as follows:

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

Yet again, as we are **only interested in modeling pairwise similarities,** we set the value of $q_{i|i} = 0$.

The objective is to have conditional probabilities in the low-dimensional space $(q_{j|i})$ identical to the conditional probabilities in the high-dimensional space $(p_{j|i})$.

Thus, if the projected data points y_i and y_j will correctly model the similarity between the high-dimensional data□points x_i and x_j, the conditional probabilities $q_{j|i}$ and $p_{j|i}$ must be nearly equal.

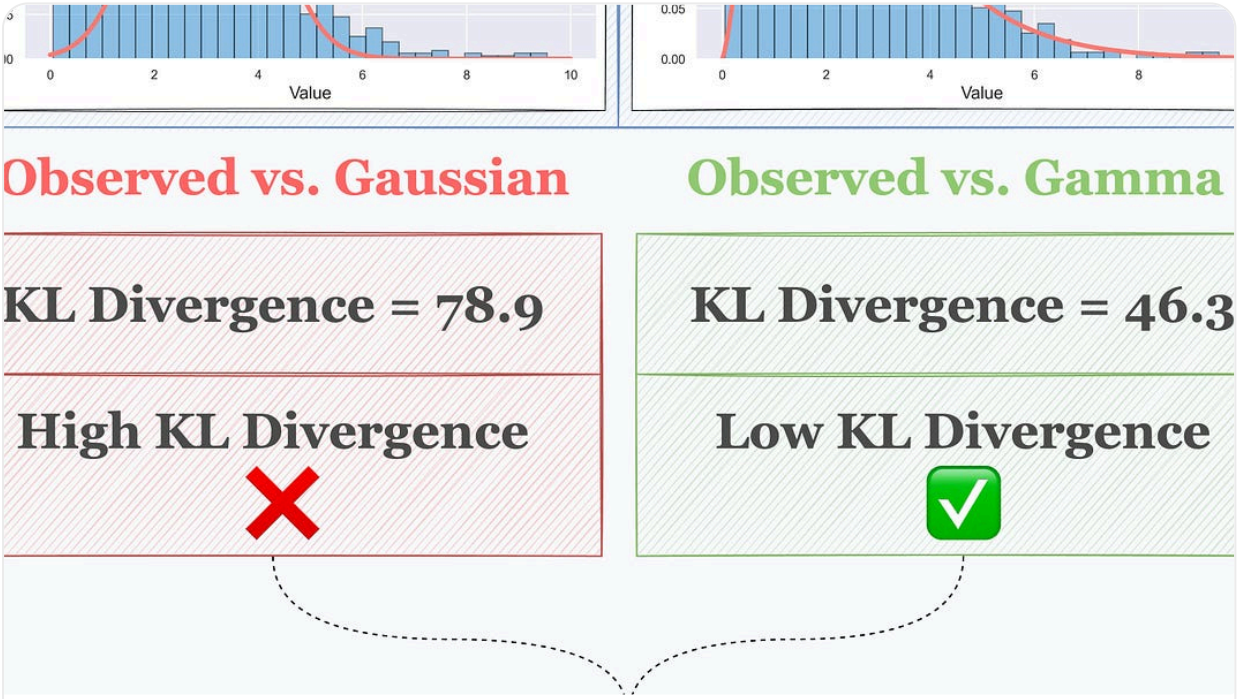


This hints that we must minimize the difference between the two conditional probabilities — $q_{j|i}$ and $p_{j|i}$.

Step 3: Define the loss function

One of the most common and popular ways to quantify the difference between two probability distributions is KL Divergence.

As we have covered this previously, we won't get into much detail:



A Visual and Intuitive Guide to KL Divergence

Support visual inspection with quantitative measures.

Daily Dose of Data Science • Avi Chawla

...but here's a quick overview of what it does.

The core idea behind KL divergence is to assess how much information is lost when one distribution is used to approximate another.

Thus, the more information is lost, the more the KL divergence. As a result, the more the dissimilarity.

KL divergence between two probability distributions $P(x)$ and $Q(x)$ is calculated as follows:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x} P(x) \cdot \log \left(\frac{P(x)}{Q(x)} \right)$$

The formula for KL divergence can be read as follows:

The KL divergence $D_{KL}\left(P||Q\right)$ between two probability distributions P and Q is calculated by summing the above quantity over all possible outcomes x. Here:

- $P(x)$ represents the probability of outcome x occurring according to distribution P.
- $Q(x)$ represents the probability of the same outcome occurring according to distribution Q.

It measures how much information is lost when using distribution Q to approximate distribution P.

Imagine this. Say P and Q were identical. This should result in zero loss of information. Let's verify this from the formula above.

If the probability distributions P and Q are identical, it means that for every x, $P(x) = Q(x)$. Thus,

$$\frac{P(x)}{Q(x))} = 1$$

or $$log(\frac{P(x)}{Q(x)}) = 0$$

Simplifying, we get:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x} P(x) \cdot \log \left(\frac{P(x)}{Q(x)} \right) \quad \text{(0)}$$

$$= \sum_{x} P(x) \cdot 0$$

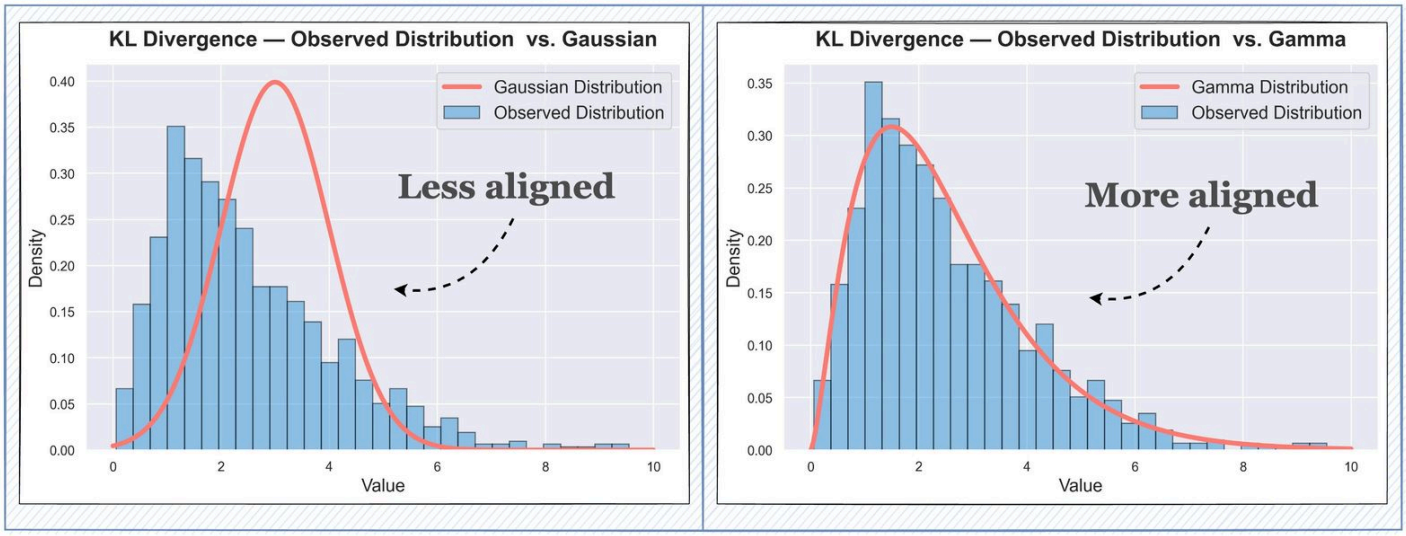
$$= 0$$

no loss of information

Hence proved.

Let's look at another illustration below:

KL Divergence



KL Divergence = 78.9 ❌

KL Divergence = 46.3 ✅

Here, we have observed distribution (the histogram in blue). The objective is to quantify whether it is more like a Gaussian distribution (left) or a Gamma distribution (right).

By measuring the KL divergence, we notice that the observed distribution resembles a Gamma distribution (right).

This is because the KL divergence between:

- The observed distribution and Gamma distribution is low.
- The observed distribution and Gaussian distribution is high.

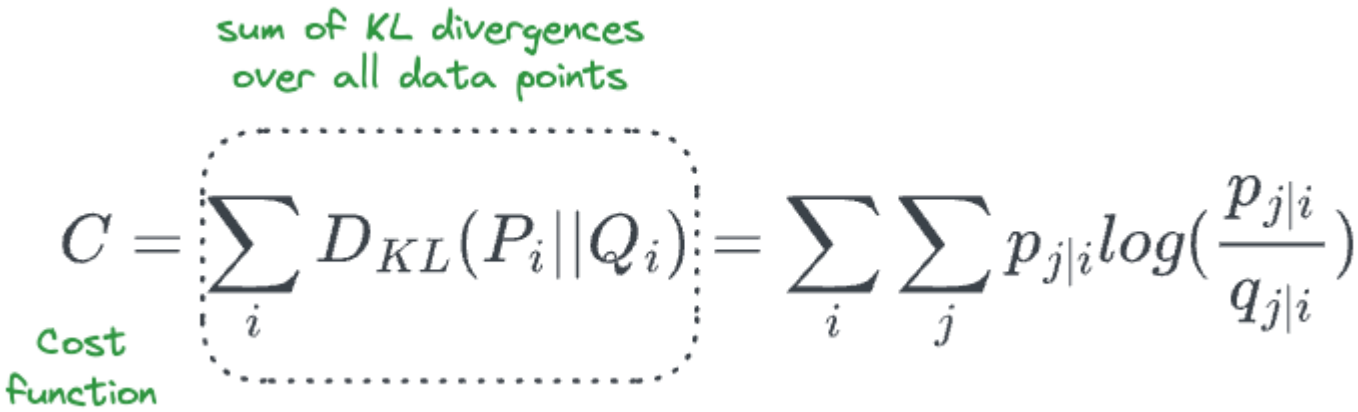
Based on this notion, KL divergence between two conditional probability distributions – $q_{j|i}$ and $p_{j|i}$ can potentially be a loss function in our problem.

The goal will be to find the most optimal projected points y_i (these are like model parameters), such that the KL divergence between the two conditional probability distributions – $q_{j|i}$ and $p_{j|i}$ is minimum.

Thus, once we have formulated a KL divergence-based loss function, we can minimize it with respect to the points y_i. Further, we can use gradient descent to update the points y_i.

Let's do it.

The cost function C of the **SNE algorithm** is given by:



Here:

- P_i denotes the conditional probability distribution over all other data points given data□point x_i.
- Q_i denotes the conditional probability distribution over all other map points given map point y_i.

For every pair of points (i, j), we have the following notations:

- $(x_i, x_j) \rightarrow$ the high-dimensional data points (x_i, x_j).
- $(y_i, y_j) \rightarrow$ the low-dimensional mapping of (x_i, x_j).
- $p_{j|i} \rightarrow$ the probability density of the Euclidean distance to x_j when x_i is the center point.
- $q_{j|i} \rightarrow$ the probability density of the Euclidean distance to y_j when y_i is the center point.

In a gist, the rightmost formulation denotes the pairwise similarities we intend to capture. That is why we get two summations.

Step 4: Compute the gradient and update the weights

We can further simplify the above cost function C as follows:

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

$$= \sum_i \sum_j [\ p_{j|i} log(p_{j|i}) - p_{j|i} log(q_{j|i})\ ]$$

constant
(independent of Q)

The first term will always be a constant. For a given dataset X, the pairwise probability densities can never change.

Also, this term is independent of $q_{j|i}$ (or $y_i \rightarrow$ the model parameters). Thus, we can safely ignore the first term.

Now, the loss function looks more like a cross-entropy loss function with an additive constant a.

$$C = a - \sum_i \sum_j p_{j|i} log(q_{j|i})$$

cross-entropy

Finding the gradient with respect to y_i, we get the following:

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$$

> 👉 Although the above gradient expression is quite simple, it is a bit complicated to derive. For those who are interested, I have provided a derivation at the end of the article.

Assuming γ to be the set of all mapped points in the low-dimensional space:

$$\gamma = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

set of all mapped points

We can update the parameters as follows:

$$\gamma^{(t)} = \gamma^{(t-1)} - \alpha \cdot \frac{\delta C}{\delta \gamma}$$

learning rate

updated parameter

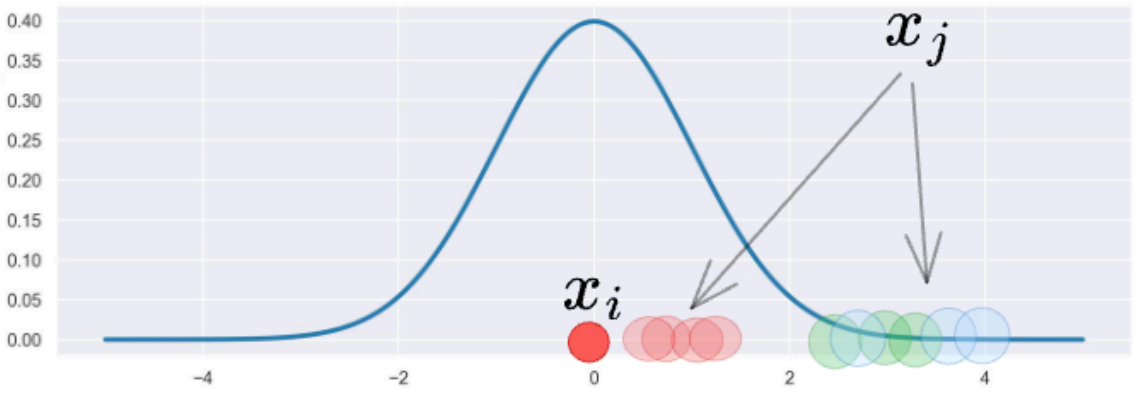
previous parameter

Gradient

Computing the variance

We are yet to discuss the variance computation of the probability density function $p_{j|i}$ we defined for the high-dimensional data points:

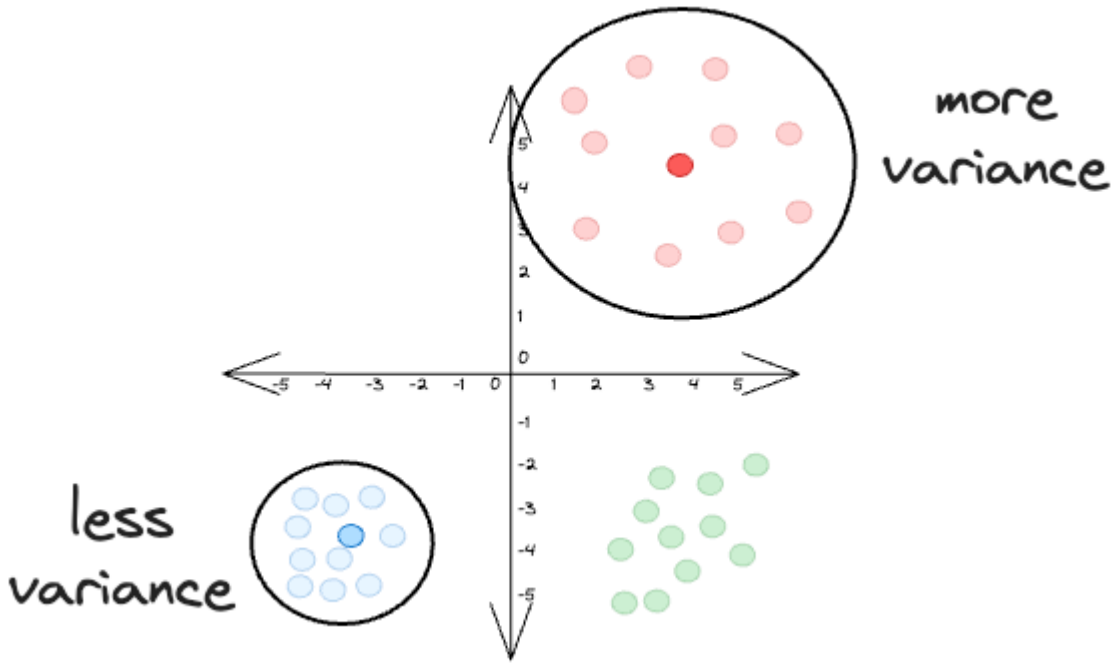
Conditional probability $p_{j|i}$



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

σ_i^2 denotes the variance of the Gaussian centered over each high-dimensional datapoint x_i.

It is unlikely that we can specify a single value of σ_i^2 for all data points. This is evident from the figure below:



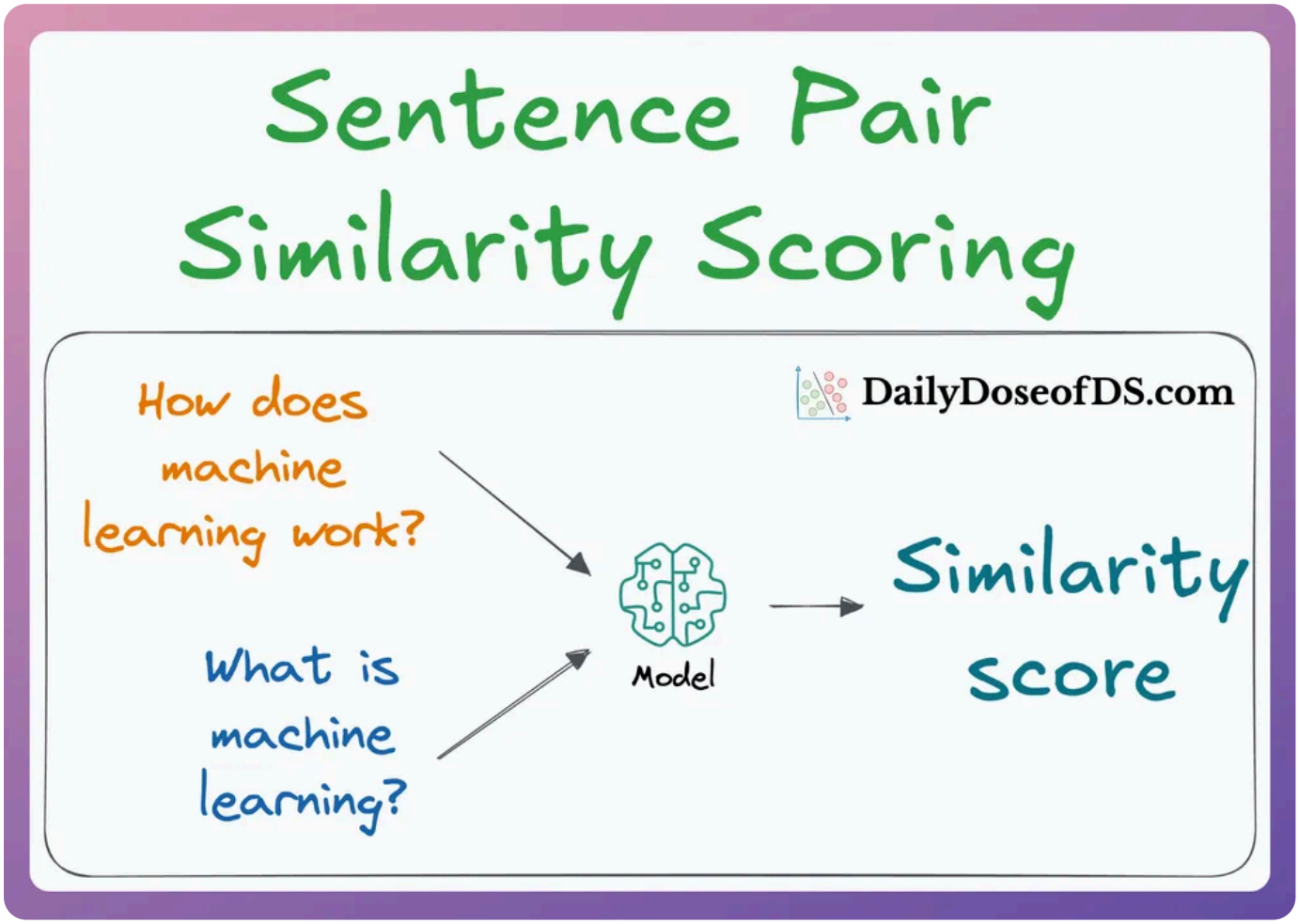
The density of the data will vary across all data points.

Thus, in dense regions, we need a small value of σ_i^2, but in sparse regions, we need a larger value of σ_i^2.

While finding the absolute best value of σ_i^2 for every data point x_i appears to be a bit difficult, we can still try.

This is where we introduce a **user-specified hyperparameter** called `perplexity`.

Read next

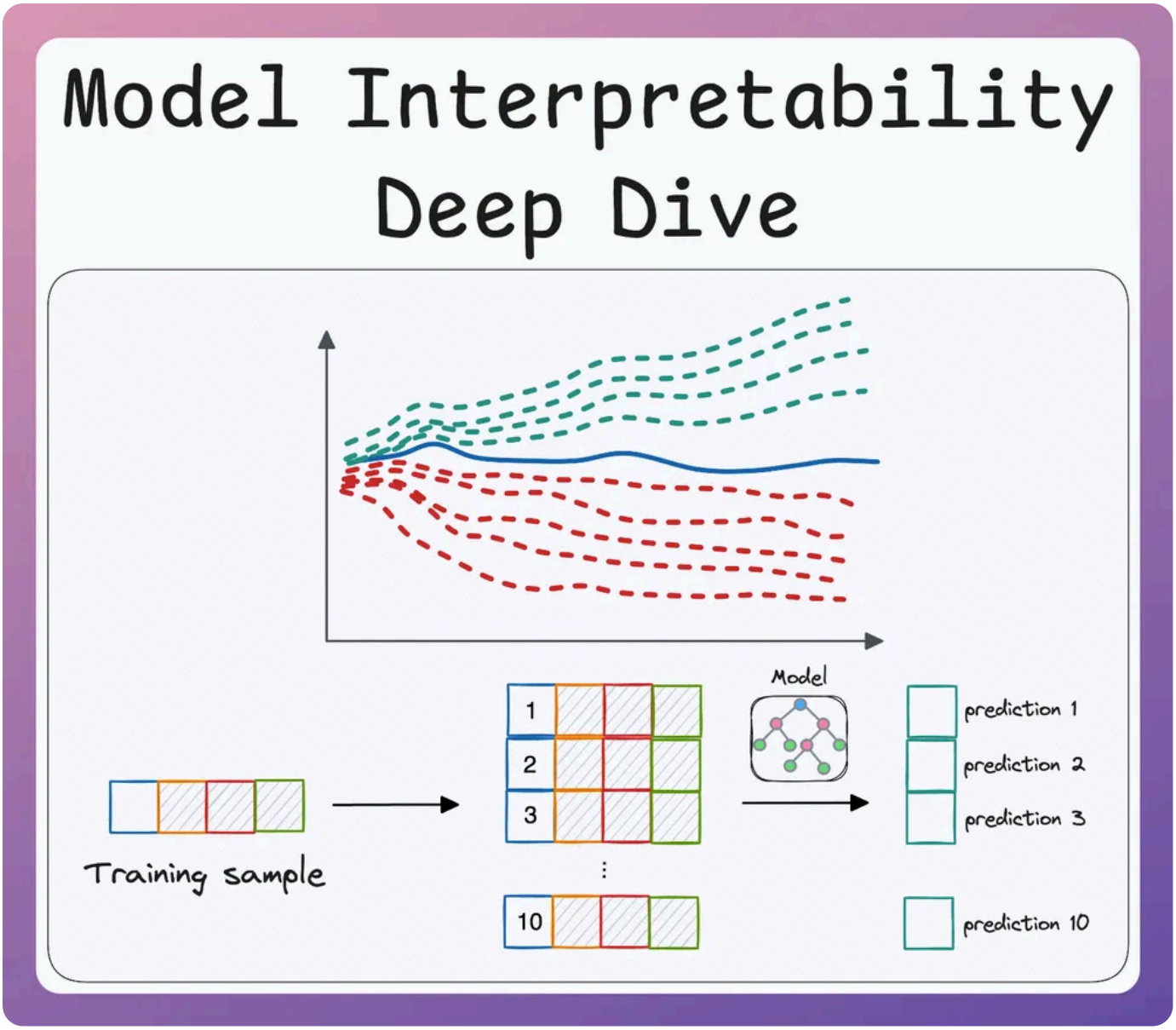


Natural Language Processing Oct 13, 2024 • 20 min read

Bi-encoders and Cross-encoders for Sentence Pair Similarity Scoring – Part 1

A deep dive into why BERT isn't effective for sentence similarity and advancements that shaped this task forever.

Avi Chawla

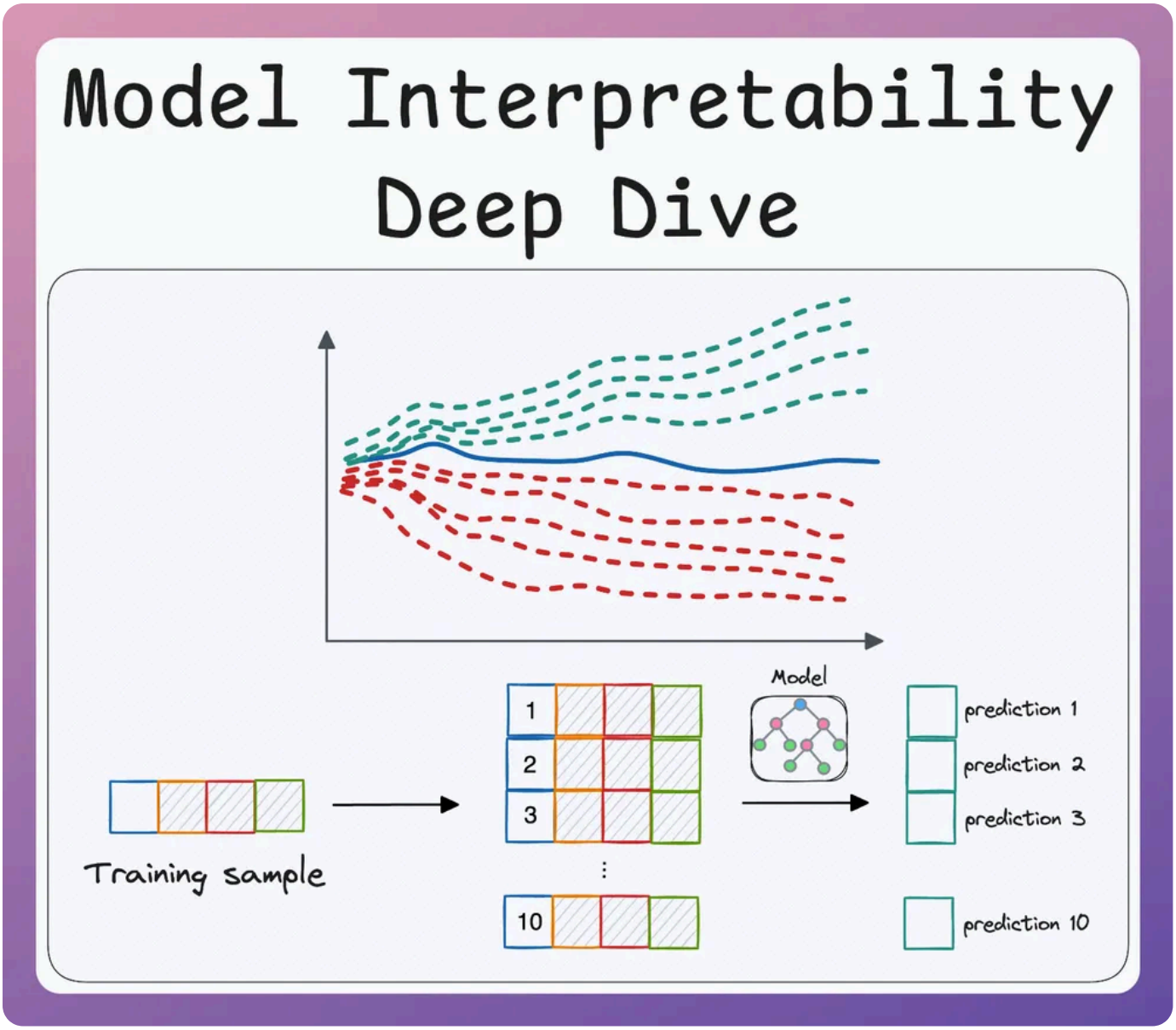


Model Interpretability Oct 6, 2024 • 26 min read

A Crash Course on Model Interpretability – Part 3

A deep dive into interpretability methods, why they matter, along with their intuition, considerations, how to avoid being misled, and code.

Avi Chawla



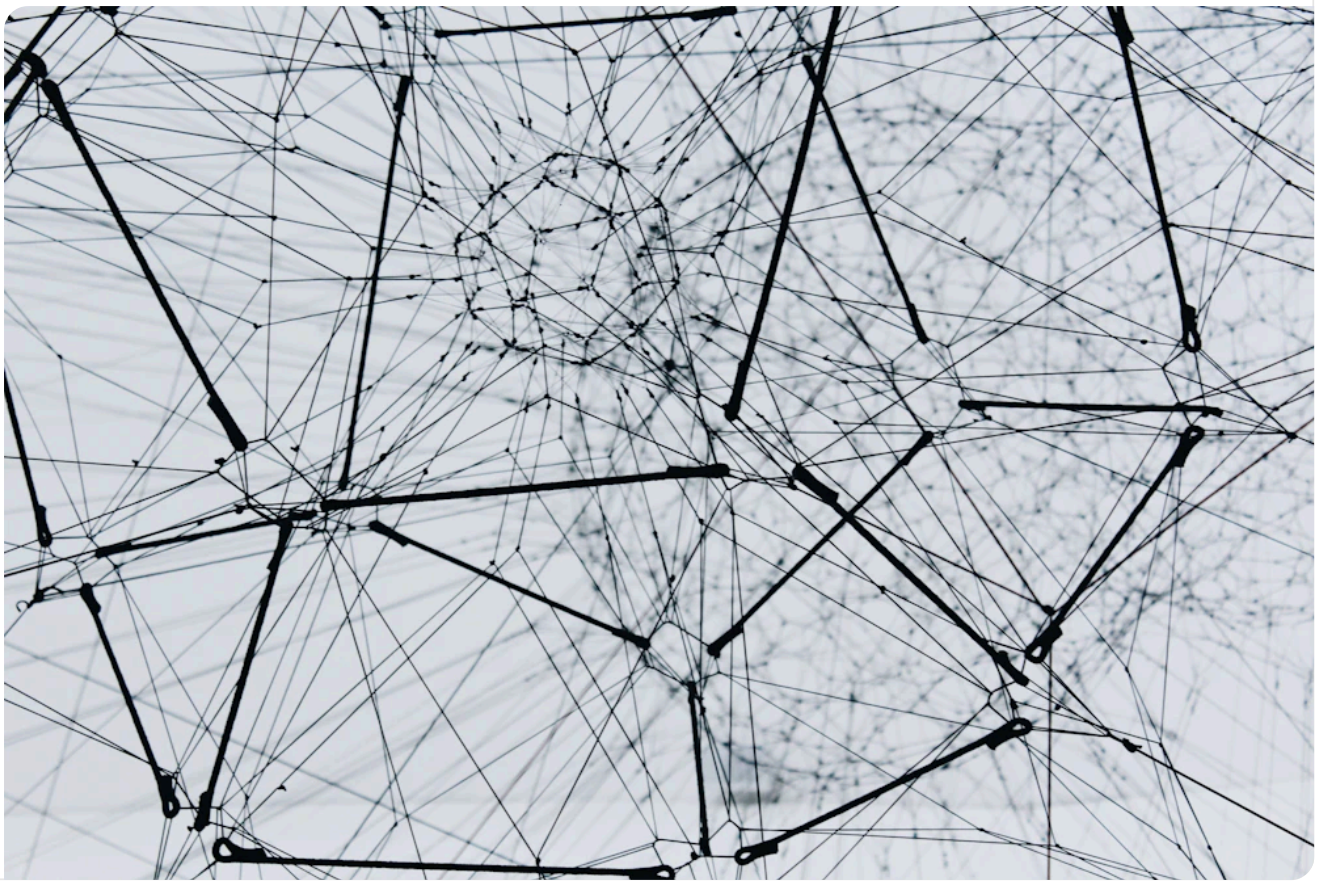
Model Interpretability Sep 29, 2024 • 19 min read

A Crash Course on Model Interpretability – Part 2

A deep dive into interpretability methods, why they matter, along with their intuition, considerations, how to avoid being misled, and code.

Avi Chawla

Get Started!



A daily column with insights, observations, tutorials and best practices on python and data science. Read by industry professionals at big tech, startups, and engineering students, across:

Navigation

Sponsor
Newsletter
Blogs
More
Contact
FAQs
About
Search 🔎
Sign in
Get Started