

Business Club

Dimensionality Reduction

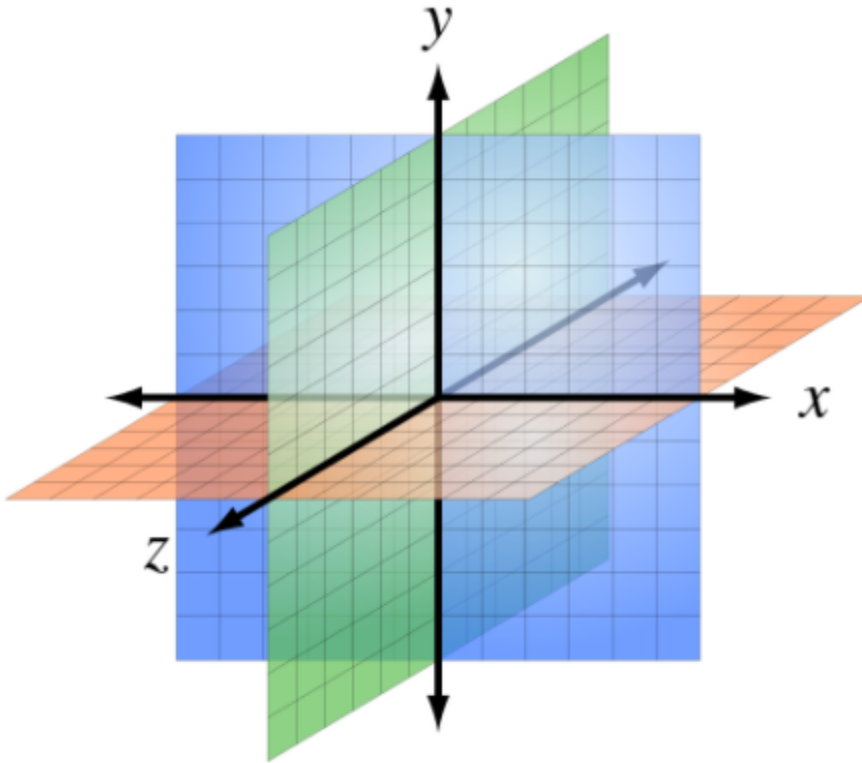
Business Club Analytics Team

June 2021

Index

1. Curse of Dimensionality Reduction
2. Introduction to Dimensionality reduction
3. Methods Dimensionality Reduction
4. PCA
5. LDA
6. t-SNE
7. UMAP
8. Conclusion

The Curse of Dimensionality

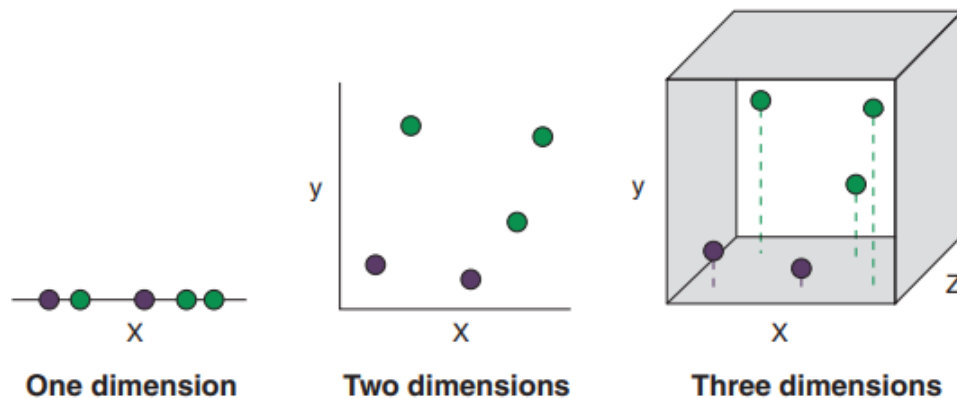


One of the first things we should do when starting an exploratory analysis is visualise our data. It's critical that we grasp the structure of our data, the relationships between variables, and how the data is dispersed intuitively. But what if we're dealing with a dataset with thousands of variables? What should we do first? Plotting each of these variables against each other isn't really an option anymore, so how can we get a feel for the overall structure in our data?

Well, we can reduce the dimensions down to a more manageable number, and plot these instead. We won't get all the information of the original dataset when doing this, but it will help us identify patterns in our data, like clusters of examples that might suggest a grouping structure in the data.

This fairly dramatic-sounding phenomena illustrates a series of difficulties we face when attempting to find patterns in a dataset with a large number of variables.

One facet of the dimensionality curse is that for a given number of examples, increasing the number of dimensions in the dataset (increasing the feature space) causes the examples to become increasingly distant. The data is considered to become sparse in this case. Many machine learning algorithms have trouble learning patterns from sparse data, and instead learn from the dataset's noise.



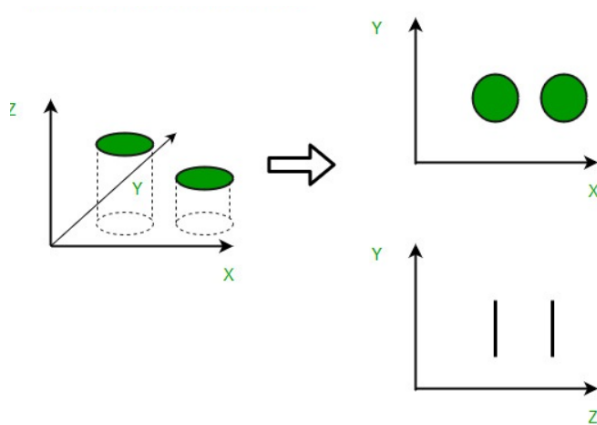
Another element of dimensionality's curse is that as the number of dimensions grows, the distances between cases begin to converge to a single value.

How can we reduce the negative effects of the curse of dimensionality on our model's prediction performance? Of course, by shrinking the dimensions! The concerns of data sparsity and near-equal distances are eliminated if we can compress most of the information from 100 variables into just 2 or 3.

Introduction to Dimensionality Reduction

Reduction of dimensionality is the method of reducing with consideration the dimensionality of the function space by obtaining a collection of principal features. Reduction of the dimensionality can be further divided into a collection of features and extraction of features.

A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 1-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.



Components of Dimensionality Reduction

There are two components of dimensionality reduction:

Feature selection: In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem.

Feature extraction: This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

Methods of Dimensionality Reduction

Depending on the method, dimensionality reduction might be linear or nonlinear. The prime linear and nonlinear methods of dimensionality reduction discussed here are:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- t-Distributed Stochastic Neighbor Embedding(t-SNE)
- Uniform manifold approximation and projection(UMAP)

Advantages of dimensionality reduction

- It cuts down on the amount of time and storage space needed.
- When data is reduced to very low dimensions, such as 2D or 3D, it becomes easier to visualise.
- It is easier to understand since it removes noise and so simplifies the explanation.
- To mitigate “curse of dimensionality”.

Principal Component Analysis

Principal Component Analysis, or PCA, is a dimensionality-reduction method for reducing the dimensionality of large data sets by transforming a large collection of variables into a smaller one that retains the majority of the information in the large set. PCA creates new variables, that are linear combinations of the original variables, that explain most of the variation in the data.

It is one of the most well-known and widely used techniques.

Why PCA?

- PCA is a versatile approach that has proven to be effective in practise.
- It's quick and easy to set up, so you can quickly evaluate performance between methods that use and don't use PCA.
- In addition, PCA has a number of modifications and extensions (e.g., kernel PCA, sparse PCA, etc.) that can be used to address specific issues.

Algorithm

STEP I - STANDARDIZATION

The importance of standardisation is due to PCA's sensitivity to the variances of the initial variables. Specifically, if the ranges of initial variables differ significantly, the variables with greater ranges will prevail over those with smaller ranges, resulting in skewed outcomes. For example, a variable with a range of 0 to 1000 will be weighted more heavily as compared to a variable with a range of 0 to 1. So, it's important to transform the variables to comparable scales.

Mathematically, subtracting the mean and dividing by the standard deviation for each value of each variable will transform all the variables to the same scale.

$$z = \frac{\text{value} - \text{mean}}{\text{standard deviation}}$$

STEP II - COMPUTING THE COVARIANCE MATRIX

The goal of this step is to figure out how the variables in the input data set differ from the mean in relation to one another, or to discover if there is a relationship between them. Because variables might be highly connected to the point where they include duplicated data. We compute the covariance matrix in order to find these associations.

The Covariance matrix is a $n \times n$ symmetric matrix (where n is the number of dimensions) containing the covariances associated with all possible pairs of the initial variables. Covariance is calculated as:

$$Cov_{xy} = \frac{\sum (x - \bar{x})(y - \bar{y})}{(n - 1)}$$

The diagonal entries are the variances of all variables whereas the other entries are covariances among variables. These covariance values tell us about the correlations between the variables. A positive covariance implies that the two variables increase or decrease together (correlated). A negative covariance implies that one variable increases when the other decreases i.e. inversely correlated.

STEP III - IDENTIFYING THE PRINCIPAL COMPONENTS

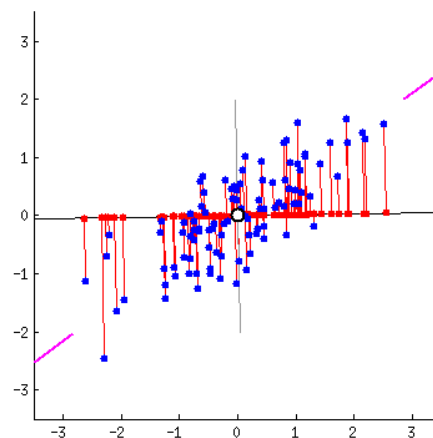
Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components.

In geometric terms, the principal components are the data directions that explain the greatest amount of variance, or the lines that encapsulate the most information in the data. The relationship between variance and information in this case is that the greater

the variance carried by a line, the greater the dispersion of data points along it, and the greater the dispersion along a line, the more information it contains. So, the principal components are new axes that provide the ideal angle for viewing and evaluating data, allowing differences between observations to be seen more clearly.

Principal components are designed in such a way that the first principal component accounts for the greatest possible variance in the data set. The second principal component is determined in the same method as the first, with the exception that it must be uncorrelated (i.e., perpendicular to) and account for the next biggest variance. This process is repeated until a total of n main components, equal to the number of variables, has been determined.

Identifying these principal components is done by computing the eigenvectors and eigenvalues of the covariance matrix. The eigenvectors of the Covariance matrix are actually the directions of the axes where there is the most variance (most information) and that we call Principal Components. And eigenvalues are simply the coefficients attached to eigenvectors, which give the amount of variance carried in each Principal Component.



By ranking the eigenvectors in order of their eigenvalues, highest to lowest, we get the principal components in order of significance.

STEP IV - FEATURE VECTOR

In the next step, we decide whether to preserve all of these components or to reject those with low eigenvalues, and then build a matrix of vectors called the Feature vector with the remaining ones.

So, the feature vector is just a matrix with the eigenvectors of the components we want to maintain as columns. If we only keep p eigenvectors (components) out of n , the final data set will only have p dimensions.

STEP V - RECASTING THE DATA ALONG THE PRINCIPAL COMPONENTS

In the last step, the aim is to reorient the data from the original axes to the ones represented by the principal components using the feature vector produced using the eigenvectors of the covariance matrix. This is accomplished by multiplying the original data set's transpose by the feature vector's transpose.

Strengths

- PCA creates new axes that are directly interpretable in terms of the original variables.
- New data can be projected onto the principal axes.
- PCA is just a mathematical transformation and so is computationally inexpensive.

Weaknesses

- Mapping from high dimensions to low dimensions cannot be nonlinear.
- It cannot handle categorical variables natively.
- The final number of principal components to retain must be decided by us for the application at hand.

Applications

1. **Neuroscience:** Spike-triggered covariance analysis is a technique in neuroscience that uses a variation of Principal Components Analysis to determine the specific qualities of a stimulus that increase a neuron's likelihood of generating an action potential. PCA is also used to find the identity of a neuron from the shape of its action potential.
2. **Finance:** There are various applications of PCA in finance including analysing the shape of the yield curve, hedging fixed income portfolios, forecasting portfolio returns, implementation of interest rate models and developing asset allocation algorithms.
3. PCA has been used on medical data to demonstrate the relationship between cholesterol and low density lipoprotein.
4. PCA has been applied to HVSR (horizontal to vertical spectral ratio) data in order to characterise earthquake-prone locations.
5. PCA has been used in the Detection and Visualization of Computer Network Attacks
6. PCA has been used in Anomaly Detection.

Linear Discriminant Analysis

Linear Discriminant Analysis(LDA) also known as Normal Discriminant Analysis is a well known feature extraction technique used in the preprocessing step of many statistical pattern recognition problems and other machine learning applications. LDA tries to reduce the dimensions so that the resulting directions have the lowest intra-cluster variance and largest inter-cluster variance. This means that the optimization seeks to keep samples from the same cluster as close to the cluster's mean as possible while also attempting to maintain cluster means as far away as possible.

The main aim of LDA is to project a dataset onto a lower-dimensional space with good class separation to avoid overfitting, i.e. the "curse of dimensionality" and lower processing costs.

Why LDA over PCA?

LDA differs from PCA because in addition to finding the component axes with LDA, we are also interested in finding the axes that maximize the separation between the multiple classes. Thus LDA seeks directions that are efficient for discriminating data whereas PCA seeks directions that are efficient for representing data.

Preparing the data for LDA

Listed below are the ways in which we can pre-process our data in order to apply LDA:

1. **Standardization:** Rescaling the input data to have a mean of 0 and a standard deviation of 1 (unit variance).
2. **Gaussian Distribution:** Each feature in the dataset is a gaussian distribution. In other words, each feature in the dataset is shaped like a bell-shaped curve.
3. **Handling Outliers:** Outliers in the data should be deleted because they introduce skewness, which affects mean and variance estimates, which, in turn, affects LDA computations.

Algorithm

The goal of an LDA is to project a feature space (a dataset of d-dimensional samples) onto a smaller subspace k (where $k \leq d-1$) while maintaining the class-discriminatory information.

The five general steps for performing a linear discriminant analysis are listed below:

1. Calculate the d-dimensional mean vectors for all the different classes of the dataset.
2. Calculate the in-between-class (S_b) matrix and within-class scatter matrix (S_w).

Suppose we have a 2-D dataset C1 and C2. So to calculate S_w for the 2-D dataset, the formula of S_w is:

$$S_w = S_1 + S_2$$

Now, the formula of covariance matrix S_1 :

$$S_1 = \sum (x - u_1).(x - u_1)^T$$

Where u_1 is the mean of class C1. Similarly, we can calculate S_2 and S_2 .

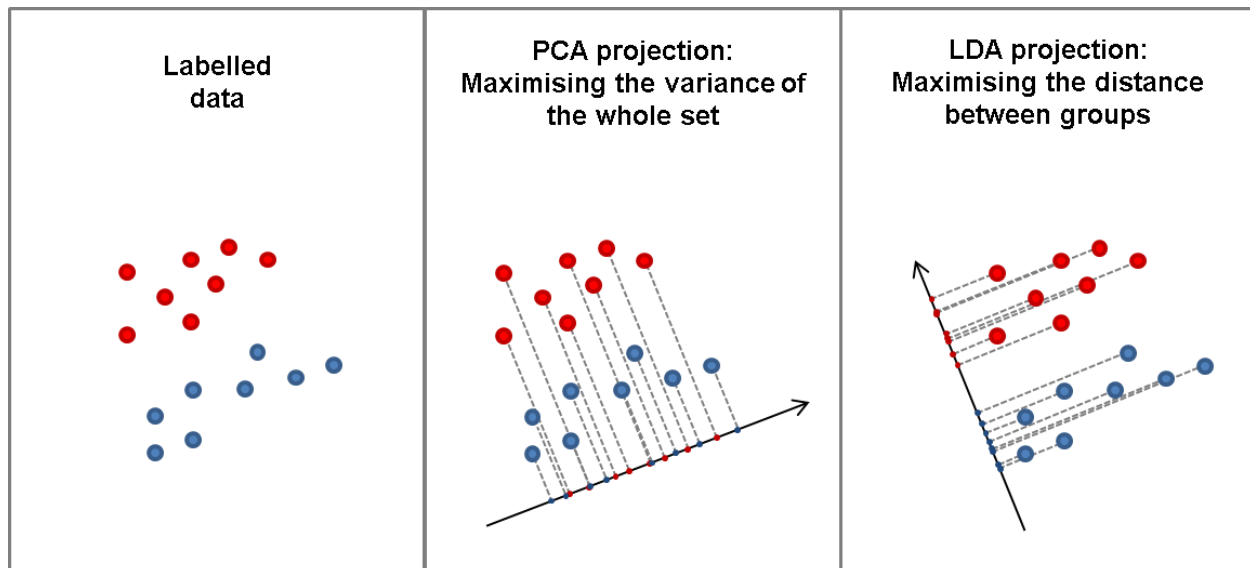
S_1 is the covariance matrix for the class C1 and S_2 is the covariance matrix for the class for C2.

Compute between class Scatter Matrix (S_b)

The formula for calculating between class Scatter Matrix (S_b) is-

$$S_b = (u_1 - u_2).(u_1 - u_2)^T$$

3. Calculate the eigenvectors ($e_1, e_2, e_3, \dots, e_d$) and corresponding eigenvalues ($\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_d$) for the scatter matrices.
4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ dimensional matrix W (where every column represents an eigenvector). We select eigenvectors with the highest eigenvalues since they contain more information about our data distribution.
5. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: $Y = X \times W$ (where X is an $m \times d$ -dimensional matrix representing the m samples, and y are the transformed $m \times k$ -dimensional samples in the new subspace).



Strengths

LDA is a supervised algorithm that can increase the extracted features' expected performance. In addition, LDA provides modifications to tackle specific roadblocks like when the final decision boundary between two classes will be quadratic, we use Quadratic Discriminant Analysis(QDA).



Weaknesses

When the mean of the distributions is shared, Linear Discriminant Analysis fails because it is impossible for LDA to discover a new axis that renders both classes linearly separable. Non-linear discriminant analysis is used in these situations.

The new features, like PCA, are difficult to understand, and you must still manually set or tweak the amount of components to keep. LDA also requires labeled data, which makes it more situational.

Applications of LDA

1. **Customer Identification:** Let's say we want to figure out what kind of clients are most likely to buy a specific product in a shopping centre. We may acquire all of the characteristics of the clients by conducting a simple question-and-answer survey. Linear discriminant analysis will assist us in identifying and selecting attributes that describe the characteristics of the group of customers most likely to purchase that particular product in the shopping mall.
2. **Face Recognition:** Face recognition is a common application in the field of computer vision, in which each face is represented by a huge number of pixel values. Before the classification procedure, linear discriminant analysis (LDA) is performed to reduce the number of features to a more manageable quantity. Each of the new dimensions created is a template made up of a linear combination of pixel values.

t-Distributed Stochastic Neighbor Embedding

The t-SNE algorithm (t-Distributed Stochastic Neighbor Embedding) is a non-linear dimensionality reduction algorithm used for exploring high-dimensional data. High-dimensional features are projected onto two- or three-dimensional space using t-SNE, which minimises the difference between similar points in high- and low-dimensional space. As a result, high-dimensional feature vectors that are close to each other in the two-dimensional embedding are very likely to be close to each other.

Why t-SNE?

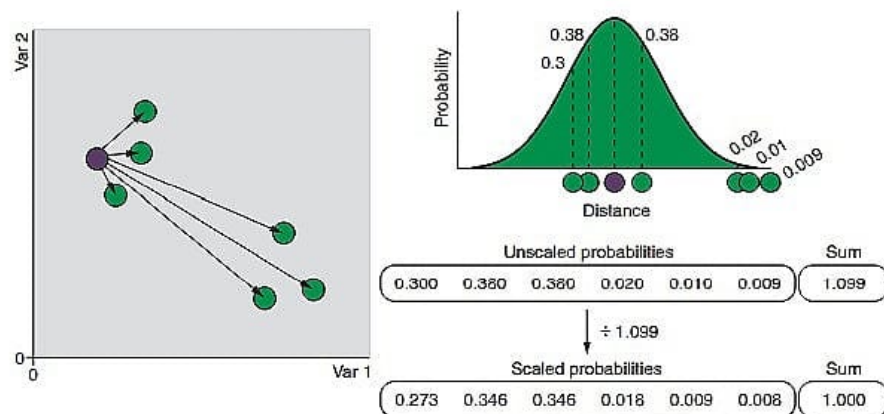
It is a non-linear Dimensionality reduction technique. It can handle outliers unlike the linear Dimensionality reduction techniques like PCA and LDA. The main benefit of this approach is that t-SNE will almost always do a better job than PCA or LDA of highlighting patterns in the data (such as clusters).

Algorithm:

The first step in the t-SNE algorithm is to compute the distance between each example and every other example in the dataset. By default, this distance is the Euclidean distance, which is the straight-line distance between any two points in the feature space. These distances are then converted into probabilities.

The distance between a particular example in the dataset and all other examples is calculated. The distances are then converted into probabilities by mapping them onto the probability density of the normal distribution, which is then centred on this example.

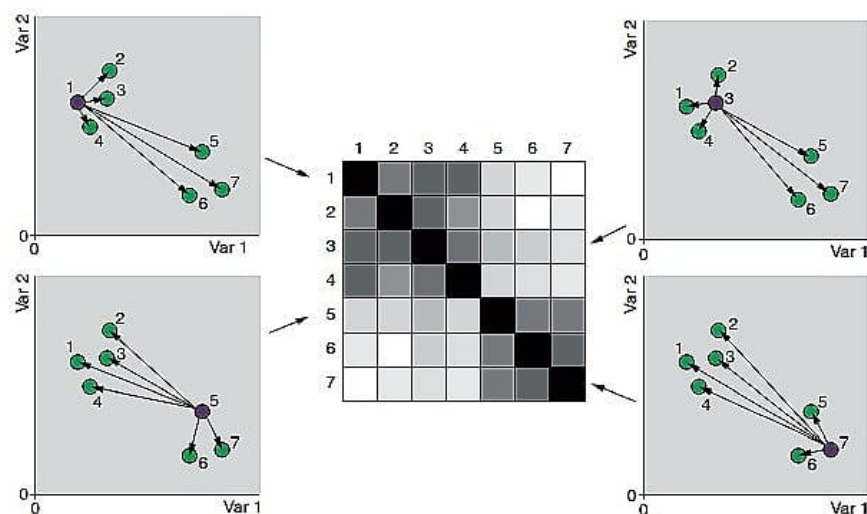
After converting the distances to probabilities, the probabilities for each example are scaled by dividing them by their sum. This makes the probabilities sum to 1 for every example in the dataset.



t-SNE measures the distance from each case to every other case, converted into a probability by fitting a normal distribution over the current case. These probabilities are scaled by dividing them by their sum, so that they add to 1.

t-SNE will expand dense clusters and compress sparse ones by using different standard deviations for different densities and then normalising the probabilities to 1 for each scenario. This allows them to be visualized more easily together. The relation between data density and the normal distribution's standard deviation depends on a hyperparameter known as **perplexity**.

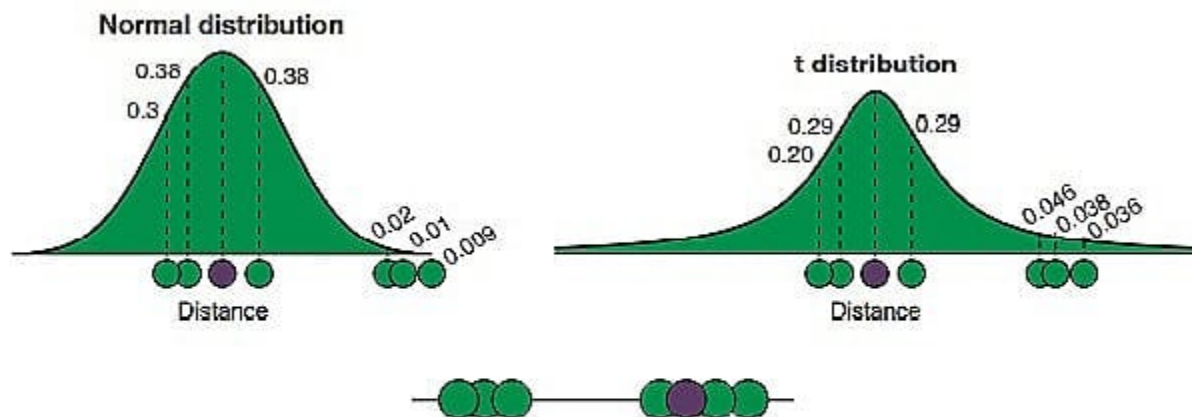
Once the scaled probabilities have been calculated for each example in the dataset, we have a matrix of probabilities that describes how similar each example is to each of the other examples.



The scaled probabilities for each case are stored as a matrix of values. This is visualized here as a heatmap: the closer two cases are, the darker the box is that represents their distance in the heatmap.

Our matrix of probabilities now serves as a reference for how the data values in the original, high-dimensional space relate to one another. The t-SNE algorithm then randomises the examples along (usually) two new axes (thus the “stochastic” part of the name).

In this new, randomised, low-dimensional space, t-SNE calculates the distances between the examples and translates them into probabilities, just like previously. The main difference is that it now employs Student's t distribution instead of the normal distribution. The t distribution resembles a normal distribution in appearance, except that its midsection isn't quite as tall, and its tails are flatter and stretch further out.



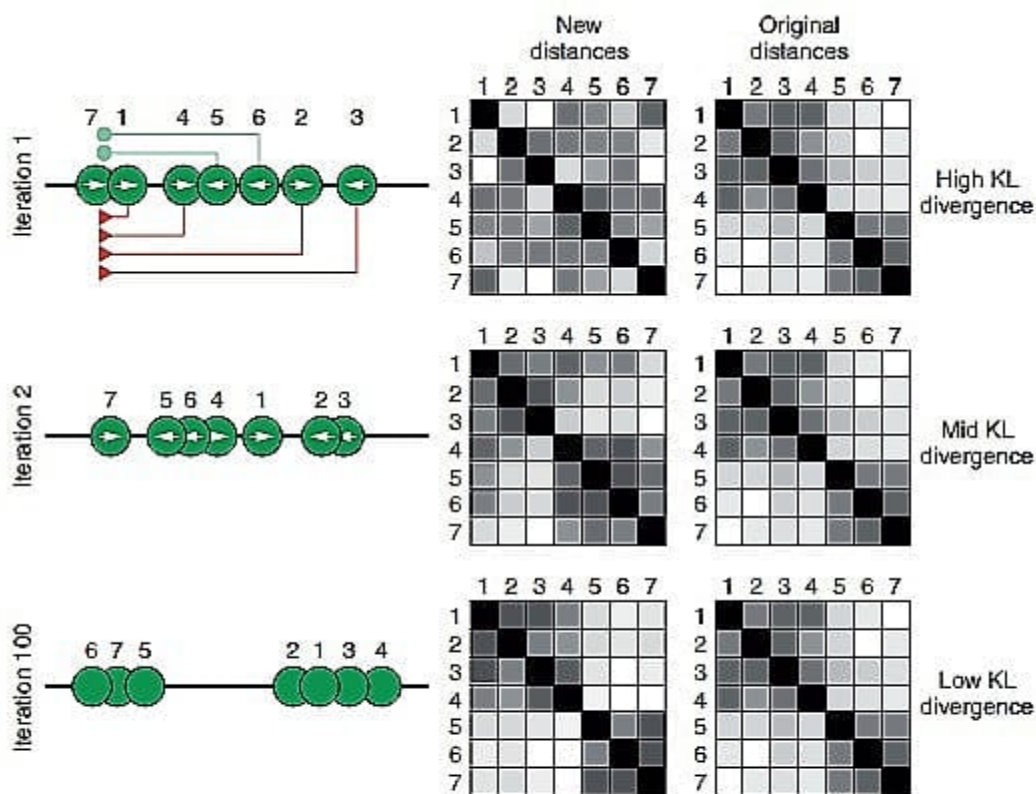
When converting distances in the lower-dimensional representation into probabilities, t-SNE fits a Student's t distribution over the current case instead of a normal distribution. The Student's t distribution has longer tails, meaning dissimilar cases are pushed further away to achieve the same probability as in the high-dimensional representation.

The job for t-SNE now is to “shuffle” the data points around these new axes, step by step, to make the matrix of probabilities in the lower-dimensional space look as close as possible to the matrix of probabilities in the original, high-dimensional space.

Each example must shift closer to examples that were close to it in the original data and away from examples that were far away to make the probability matrix in low-dimensional space appear like the one in high-dimensional space. As a result, examples that should be close by will attract their neighbours, while examples that should be far away will repel non-neighbors.

The balance of these attractive and repulsive forces causes each example in the dataset to move in a direction that makes the two probability matrices a little more similar. The low-dimensional probability matrix is calculated again in this new position, and the examples move again, making the low- and high-dimensional matrices look more similar.

This method is repeated until the divergence (difference) between the matrices reaches a predetermined number of iterations or the divergence (difference) between the matrices stops improving.



Cases are randomly initialized over the new axes (one axis is shown here). The probability matrix is computed for this axis, and the cases are shuffled around to make this matrix resemble the original, high-dimensional matrix by minimizing the Kullback-Leibler (KL) divergence. During shuffling, cases are attracted toward cases that are similar to them (lines with circles) and repulsed away from cases that are dissimilar (lines with triangles).

Strengths

- Works well for Nonlinear data: It is able to interpret the complex relationship between features and represent similar data points in high dimension to close together in low dimension.
- Preserves Local and Global Structure: t-SNE is capable of preserving the local and global structure of the data. This means, points that are close to one another in the high-dimensional dataset, will tend to be close to one another in the low dimension.


Weaknesses

- With the number of examples in the dataset, the computing time grows exponentially. t-SNE could take hours to run for exceptionally large datasets.
- It is impossible to project additional data onto the embedding Because the initial placement of the data onto the new axis is random. Rerunning t-SNE on the same dataset will yield somewhat different results.
- Distances between clusters are frequently meaningless. Assume we have three data clusters in our final t-SNE representation: two are close together, while the third is far apart. We can't argue that the first two clusters are more comparable to each other than the third cluster since t-SNE focuses on local, not global, structure (Calculations that are conducted over the entire data set are referred to as global, and calculations that are conducted locally to a point or a partition are referred to as local).

Applications of t-SNE:

Genomic research, computer security research, music analysis, cancer research, bioinformatics, and biological signal processing are just a few of the applications where t-SNE has been employed for visualisation.

We explore the applicability of t-SNE to human genetic data and make these observations:

- 
- (i) t-SNE can distinguish samples from different continents;
 - (ii) t-SNE is more resilient in the presence of outliers; and
 - (iii) t-SNE can show both continental and sub-continental patterns in a single plot.

The potential of t-SNE to identify population stratification at various scales, we conclude, could be valuable for human genetic association research.

Uniform manifold approximation and projection

UMAP is another nonlinear dimension-reduction algorithm that overcomes some of the limitations of t-SNE. It works similarly to t-SNE (finds distances in a feature space with many variables and then tries to reproduce these distances in low-dimensional space), but differs in the way it measures distances.

Why UMAP?

- UMAP is considerably faster than the rest of the methods.
- It is a deterministic algorithm i.e. given the same input, it will always give the same output.
- It preserves both local and global structure.

Algorithm

UMAP is a graph layout method, similar to t-SNE, but with a few crucial theoretical underpinnings that give the algorithm a stronger foundation.

In its simplest sense, the UMAP algorithm consists of two steps:

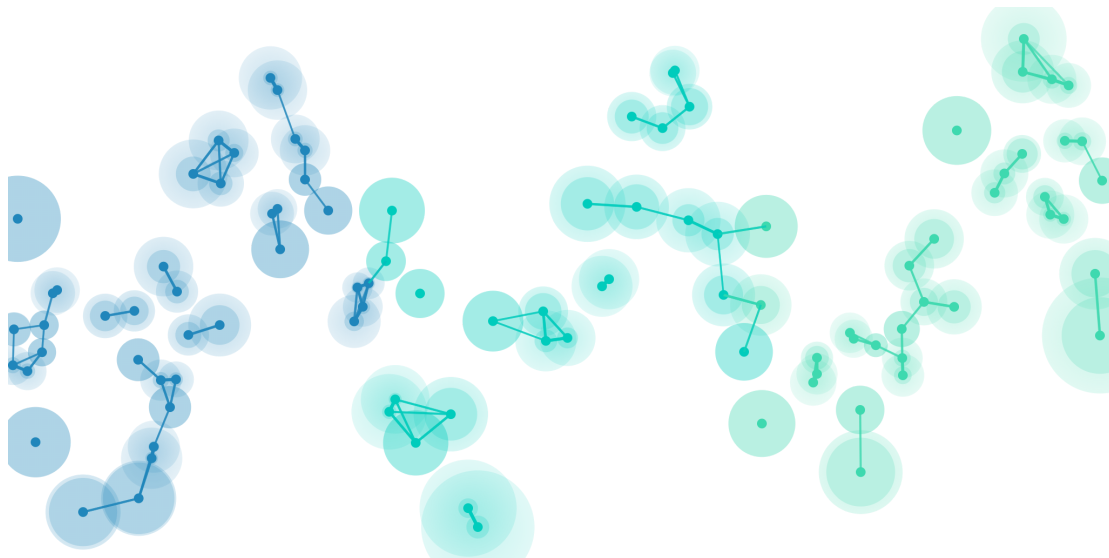
1. Construction of a graph in high dimensions
2. An optimization step finding the most similar graph in lesser dimensions.

To achieve this purpose, the technique draws on a number of insights from [algebraic topology](#) and [Riemannian geometry](#).


Despite the daunting mathematics, the key ideas' intuitions are actually rather straightforward: UMAP creates a weighted graph from the high-dimensional data, with edge strength denoting how "near" one point is to another, and then projects it down to a lower dimensionality.

To create the initial high-dimensional graph, UMAP relies on constructing what's known as a [Čech complex](#), which is a way of representing a topology combinatorially. To get there, we'll use a simplex, which is a basic building block. A simplex is a n -dimensional object constructed by joining $n + 1$ points in geometric terms; a 0-simplex is a point, a 1-simplex is a line, and a 2-simplex is a triangle, for example. We consider the data as a set of simplices and combine them in a specific way to form a Čech complex.

To begin, we think of each data point as a sample from a continuous, high-dimensional shape (our topology). Each point can be thought of as a 0-simplex. We can create sets of 1-, 2-, and higher-dimensional simplices by extending a radius r out from each point and linking points that overlap. This simplicial complex does a reasonable job of approximating the fundamental topology of the dataset, explained by the [Nerve Theorem](#).



Unfortunately, real-world high-dimensional data poses a hurdle for UMAP to solve: determining the proper radius size. If the radius is too tiny, we'll wind up with isolated, localised groupings of dots. When a space is too big, everything starts to connect. The



curse of dimensionality, which causes distances between points to become increasingly identical in higher dimensions, worsens the problem. UMAP solves this problem in a clever way: Rather than using a fixed radius, UMAP uses a variable radius determined for each point based on the distance to its k th nearest neighbor. Within this narrow radius, connectedness is then "fuzzy" by assigning a probability to each link, with distant points becoming less likely to be related. This method produces a weighted graph, with edge weights denoting the likelihood that two points in our high-dimensional manifold are "linked".

After constructing the final, fuzzy simplicial complex, UMAP uses a force-directed graph layout technique to project the data into lower dimensions. This optimization stage is quite similar to t-SNE, however UMAP is able to speed up the optimization and maintain far more global structure than t-SNE by jumping through the theoretical hoops while generating our initial simplicial complex.

Strengths


- They can learn nonlinear patterns in the data.
- They tend to separate clusters of cases better than PCA.
- UMAP can make predictions on new data.
- UMAP is computationally inexpensive.
- UMAP preserves both local and global distances.

Weaknesses

- The new axes of UMAP are not directly interpretable in terms of the original variables.
- It cannot handle categorical variables natively.

Applications

1. **Population Genetics:** Uniform manifold approximation and projection (UMAP) has been rapidly adopted by the population genetics community to study



population structure. It has become common in visualizing the ancestral composition of human genetic datasets, as well as searching for unique clusters of data, and for identifying geographic patterns.

2. **Bioinformatics:** UMAP has been proved useful in high-dimensional cytometry and single-cell RNA sequencing, meaningful organization of cell clusters and preservation of continuums.

Conclusion

The module describes what dimensionality reduction is and why it is useful. We then discuss in detail four dimensionality reduction techniques, PCA, LDA, t-SNE, UMAP. For each technique, we discuss why it is used, a detailed description of the algorithm, its strengths and weaknesses and its applications.

A short comparative review of all the algorithms discussed:

	PCA	LDA	t-SNE	UMAP
Type of transformation	Linear	Linear	Non-linear	Non-linear
Supervised vs Unsupervised	Unsupervised	Supervised	Unsupervised	Semi-supervised
Objective	Preserving Variance	Classification Performance	Neighbourhood Preservation	Manifold Learning
Parametric vs Nonparametric	Parametric	Parametric	Non parametric	Non parametric
Deterministic Property	Non deterministic	Deterministic	Non deterministic	Deterministic

