C›ONSTRUCTOR
LEARNING

Unsupervised ML: Clustering & Dimensionality Reduction

Week 9 - Machine Learning

constructor.org

Today's Plan

- Introduction to unsupervised clustering
- K-means clustering
- K-Medoids
- DBSCAN clustering
- Hierarchical Clustering
- Evaluating Clustering Models
- Dimensionality Reduction
- Projection methods - PCA
- Manifold Learning - t-SNE
- Live-Coding
- Exercises

Unsupervised Learning - Clustering

- Clustering is used in a wide variety of applications
  - customer segmentation
  - dimensionality reduction
  - anomaly detection
  - semi-supervised learning
  - search engines (group similar content)
  - image segmentation (split image into pixel subgroups)

- No universal definition of what a cluster is, depends heavily on context
  - some algorithms look for points centered around a particular (cluster) center
  - others look for continuous regions of densely packed instances
  - yet others look for clusters of clusters - hierarchical clustering

Unsupervised Learning - Clustering

How do we find structure in data when we don't already know the response/target variable?

- This is a tough task. Most of machine learning is of the supervised variety, i.e. we know the inputs and outputs and want to learn mappings between them
- Clustering falls into this task, as does feature engineering
- K-Means is a popular algorithm, Density-based clustering, Hierarchical Clustering... we'll go over a few of these algorithms

K-Means Clustering

- The idea is to group n data points into k (≤ n) sets S = {S1, S2, …, Sk}
- We choose k, and have to minimize the within-cluster sum of squares (WCSS)
- Equivalent to maximizing the sum of squared deviations between points in different clusters (between-cluster sum of squares, BCSS)
- Not KNN! ‘K’ in K-Means is the number of clusters, not number of neighbors

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2 = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

μi is the mean of points in Si, and |Si| is the number of elements in the set

- Data should be scaled before running the model!

K-Means Clustering

- There's no quick way to find the optimal means (there is no known polynomial algorithm, so that the time to find a solution grows exponentially with problem size)

How to do it? Standard algorithm:

Lloyd's algorithm:

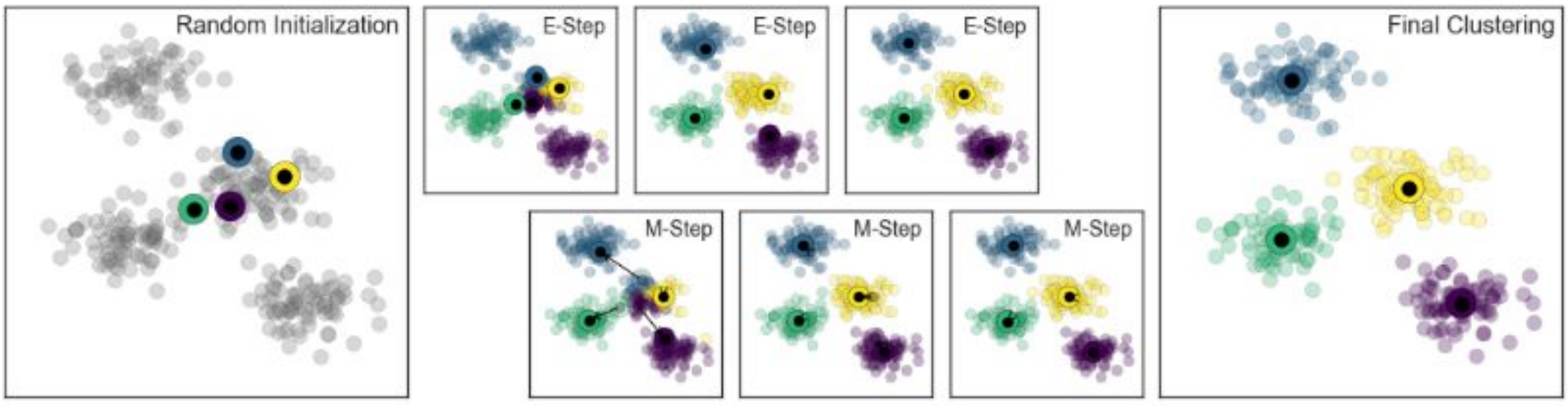
1) Initialize the model (random centroids based on k-clusters)
2) Assign each point to the mean that is the minimum distance away (**E**xpectation Step)
3) Calculate the new means (centroids) per cluster (**M**aximization Step)
4) Repeat 2-3

Initialization:

- k-means++ (push the centroids as far from one another as possible)
- Randomization (assign clusters randomly)

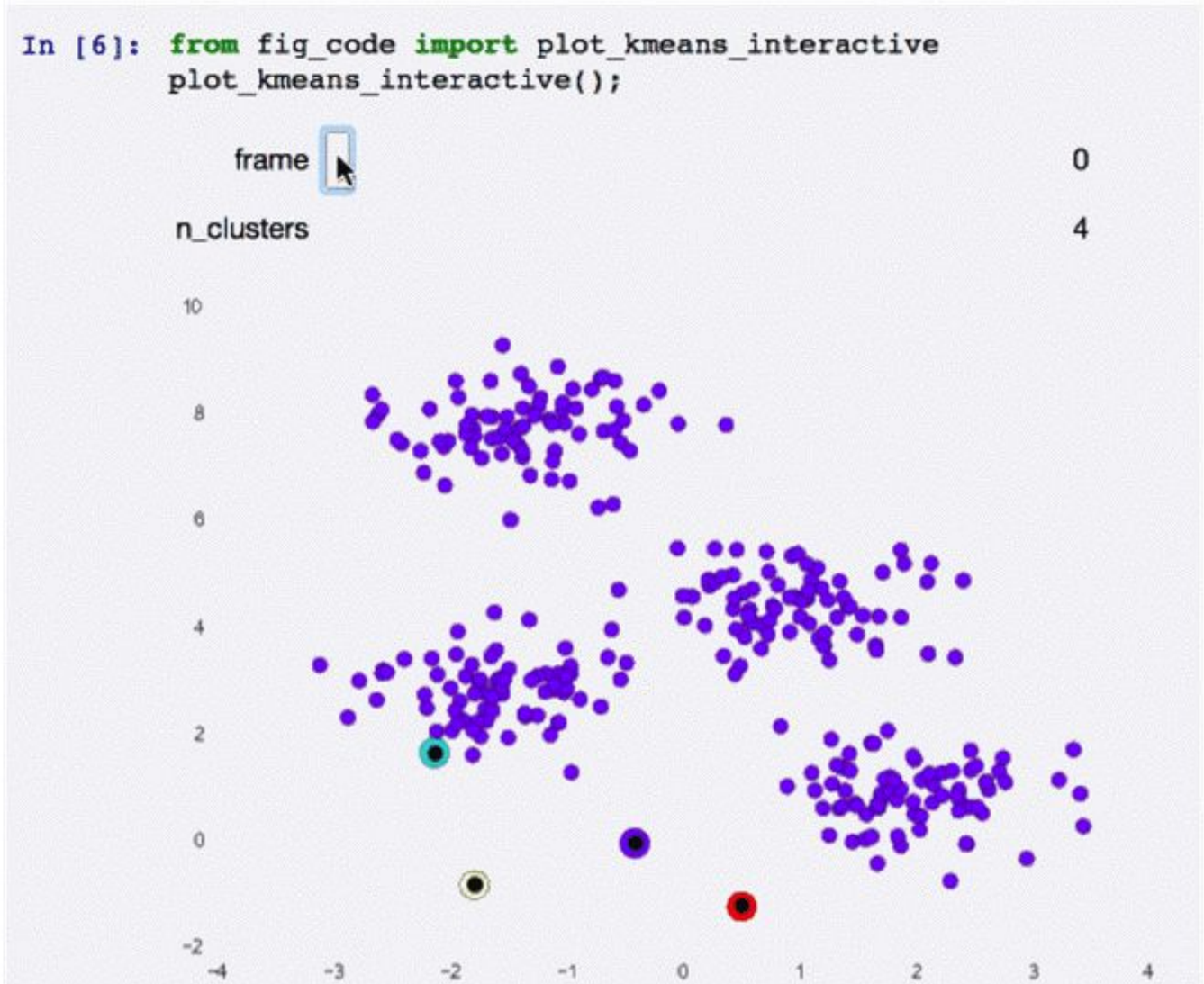
K-Means Clustering

- K initial "means" are randomly generated within the data domain
- K clusters are created by associating every observation with the nearest mean
- Previous steps are repeated until convergence has been reached



K-Means Clustering

- K initial "means" are randomly generated within the data domain
- K clusters are created by associating every observation with the nearest mean
- Previous steps are repeated until convergence has been reached



K-Means applications

- Document Classification
- Delivery Store Optimization
- Identifying Crime Localities
- Customer Segmentation
- Fantasy League Stat Analysis
- Insurance Fraud Detection
- Rideshare Data Analysis
- Cyber-Profiling Criminals
- Call Record Detail Analysis
- Automatic Clustering of IT Alerts

K-Medoids

- K-medoids is a clustering algorithm related to the k-means and the medoid shift algorithm
- Both algorithms are partitional (breaking the dataset up into groups)
- K-means attempts to minimize the total squared error, k-medoids minimizes the sum of dissimilarities between points labeled to be in a cluster and a point designated as the center of that cluster
- K-Means relies implicitly on Euclidean distance (though you could always pass a transformed feature space)
- K-Medoids instead takes actual data points as centers, similarities are less sensitive to Euclidean distance
- While a centroid, in K-Means, almost never corresponds to an actual data point

K-Medoids

Steps (Partitioning Around Medoids or PAM):

1) Randomly pick **k** points as medoids

2) Assign each point to its nearest medoids

3) Now while the loss function is decreasing for each medoid m and non-medoid point p:

   ❏ Switch **m** and **p** and see if the loss function improves overall

   ❏ If it does, keep the switch. If it doesn't, don't keep it

Pros/Cons:

- More **robust** than k-means - less sensitive to extreme outliers
- More flexible than k-means - can use any similarity measure
- More expensive than k-means

Hierarchical Clustering

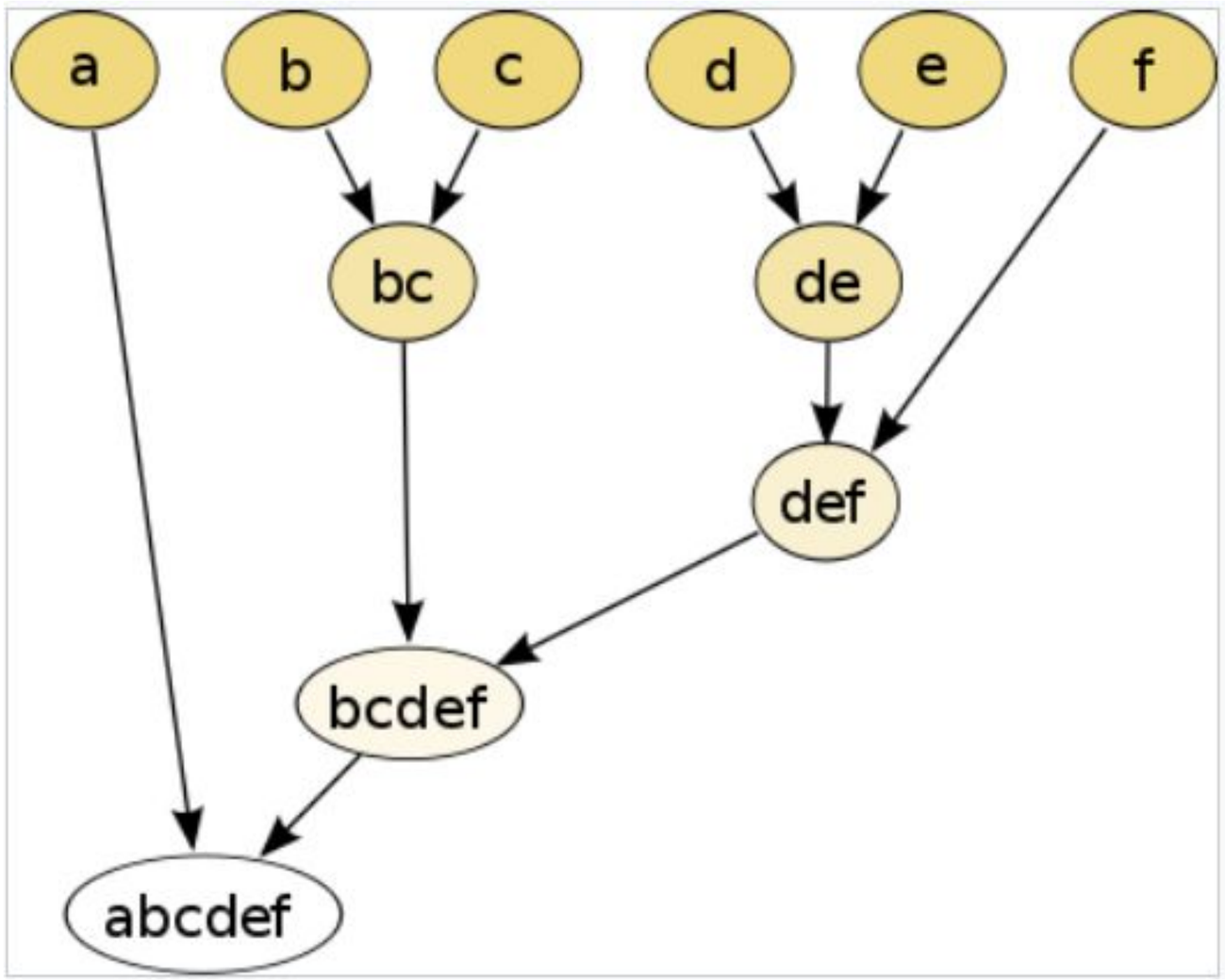
Hierarchical Clustering is when we have many layers of clusters. Think of a tree!

- You don’t know how many clusters there are, but you do have a measure of distance or similarity for each of the points

- So you can either go top-down (divisive) or bottom-up (agglomerative)

- These things are pretty computationally expensive

Hierarchical Clustering



Source: Wikipedia

The bottom-up approach:

1) Assign each data point to its own cluster

2) Merge the two most similar data points (or clusters), then recompute distances

3) Repeat 2 until you have one cluster

4) Now pick a maximum depth, and consider everything below it as clusters in terms of the number of points per cluster (below that threshold)

Hierarchical Clustering

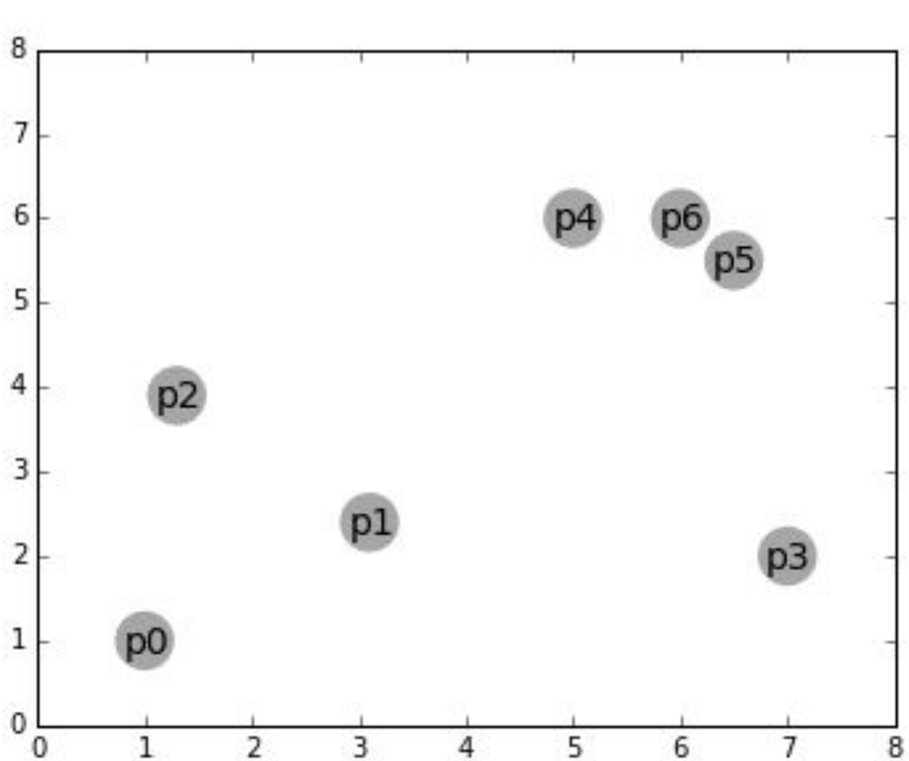
- The linkage criterion determines the distance between sets of observations (cluster) as a function of the pairwise distances between observations
- We can get wildly different results depending on the linkage choice
- Plot it! (Dendrogram)

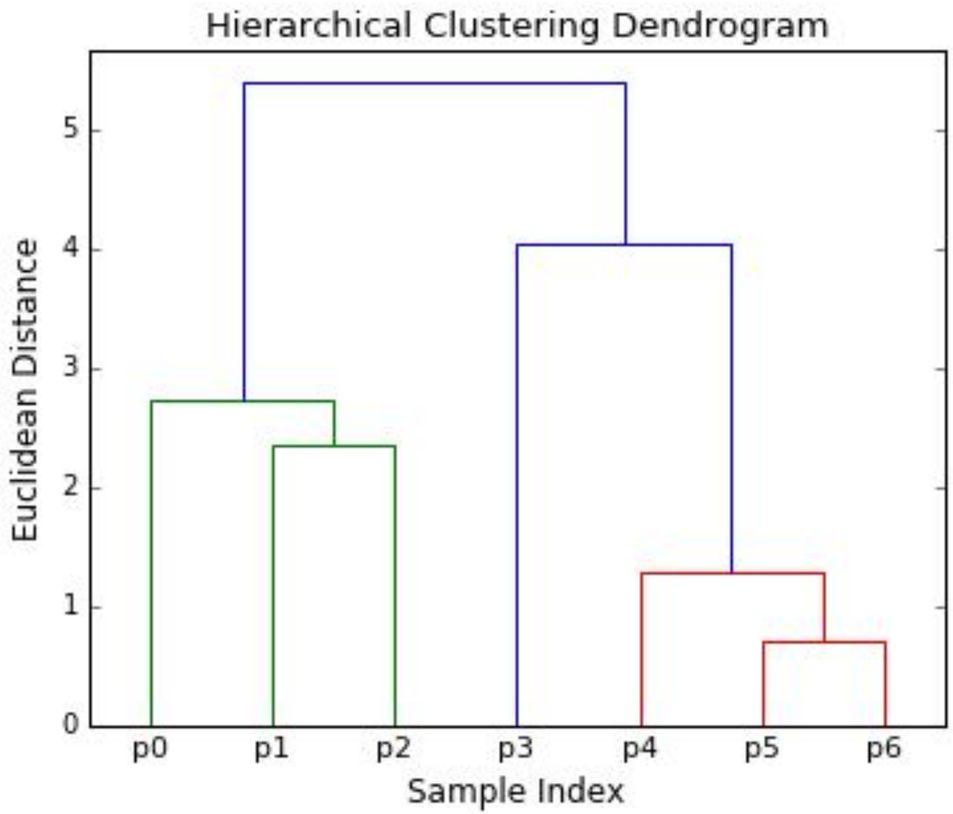
Names	Formula
Maximum or complete-linkage clustering	$\max \{\, d(a,b) : a \in A,\, b \in B \,\}.$
Minimum or single-linkage clustering	$\min \{\, d(a,b) : a \in A,\, b \in B \,\}.$
Mean or average linkage clustering, or UPGMA	$\frac{1}{\|A\|\|B\|} \sum_{a \in A} \sum_{b \in B} d(a,b).$
Centroid linkage clustering, or UPGMC	$\|c_s - c_t\|$ where c_s and c_t are the centroids of clusters *s* and *t*, respectively.
Minimum energy clustering	$\frac{2}{nm} \sum_{i,j=1}^{n,m} \|a_i - b_j\|_2 - \frac{1}{n^2} \sum_{i,j=1}^{n} \|a_i - a_j\|_2 - \frac{1}{m^2} \sum_{i,j=1}^{m} \|b_i - b_j\|_2$

where *d* is the chosen metric.

Source

Hierarchical Clustering





The bottom-up approach in action
(agglomerative)

Source

Hierarchical Clustering

Hierarchical clustering can also support categorical input data, defining distances as follows:

- If a column is *ordinal*, then the value used for clustering is the index of the ordered category, treated as if it were continuous data. These values are standardized as if they were continuous data.
- If a column is *nominal*, then the distance between two observations where the categories match is zero. If the categories differ, the distance is one. (like jaccard distance)

Hierarchical clustering enables you to choose among rules for defining distances between clusters (linkage).

Advantages: 1) Great visualisation method

2) Provides hierarchical relations between clusters

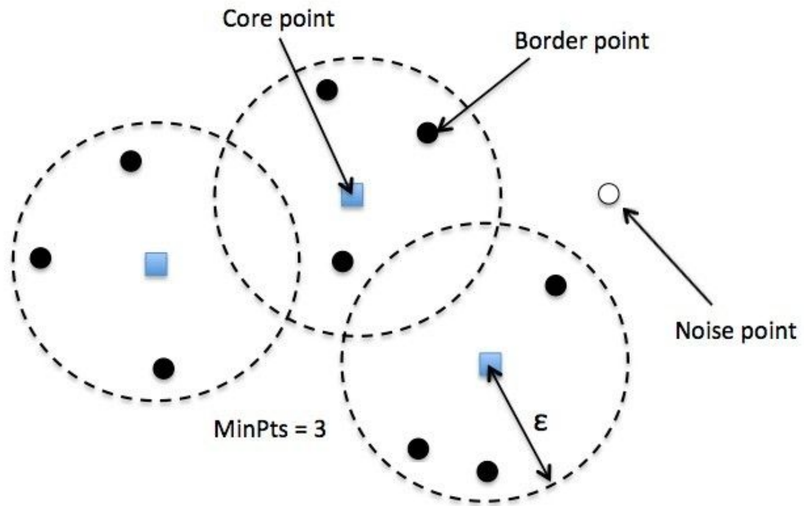
3) Are able to capture non-spherical clusters such as concentric clusters

Disadvantages: Not easy to define at which depth to cut

Density-based spatial clustering of applications with noise (DBSCAN)

DBSCAN is a **density-based** clustering algorithm

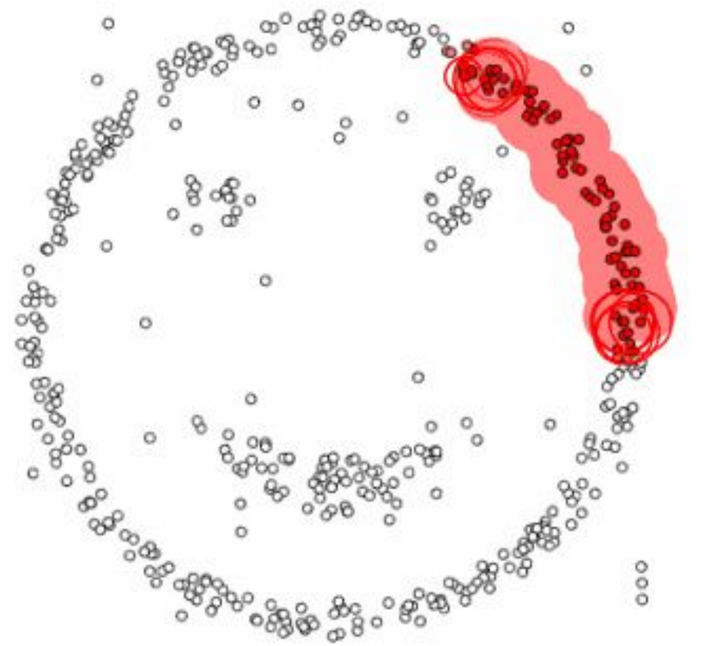
- For each sample, count how many samples are located within a small distance **ε** from it. This region is called the sample's **ε-neighbourhood**
- If the **ε-neighbourhood** has at least **min_sample**s members, then the sample is called a **core point**. Core points are located in dense regions
- All samples of the neighbourhood of a core point belong to the same cluster. This neighbourhood includes the **ε-neighbourhood**, but also other core points
- Any sample that is not a core point and does not have one a core point in its **ε-neighbourhood** is a **noise point** or an anomaly



Source

Density-based spatial clustering of applications with noise (DBSCAN)

- DBSCAN does not require us to specify the number of clusters and works well at finding arbitrarily shaped clusters while being robust to outliers
- It repeats the following process until all points have been assigned to a cluster:
  - ❏ Arbitrarily select a point P
  - ❏ Retrieve all points directly density-reachable from P with respect to **ε**
  - ❏ If P is a core point, a cluster is formed
  - ❏ Finds recursively all density connected points and assign them to the same cluster as P
  - ❏ If P is not a core point, it iterates through the remaining unvisited points in the dataset



- DBSCAN is very bad when the different clusters in your data have different densities
- Parameters must be chosen carefully.

Source

Choosing cluster counts

How we choose clusters?

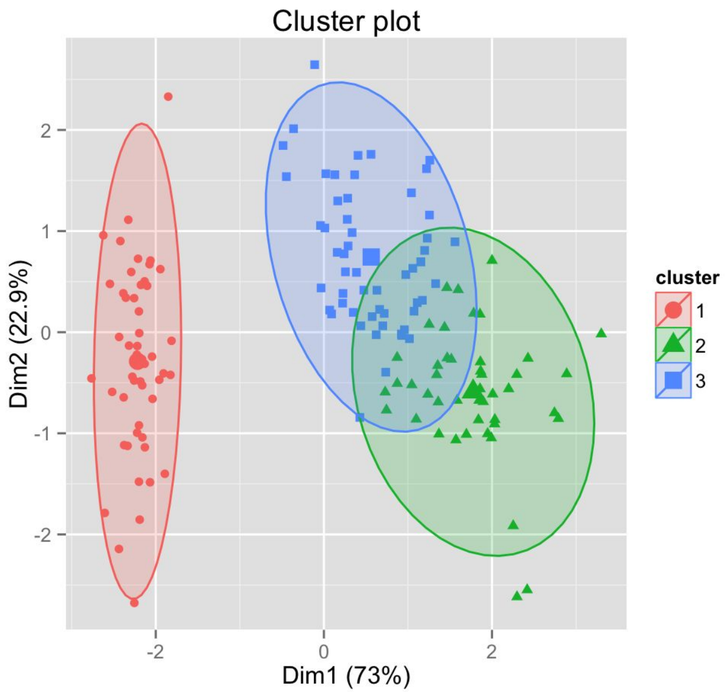
- Visualization is the key (heatmaps to visualize clusters)
- Think of each additional cluster as another parameter
- Possible approaches:
  - Elbow method
  - Silhouette method

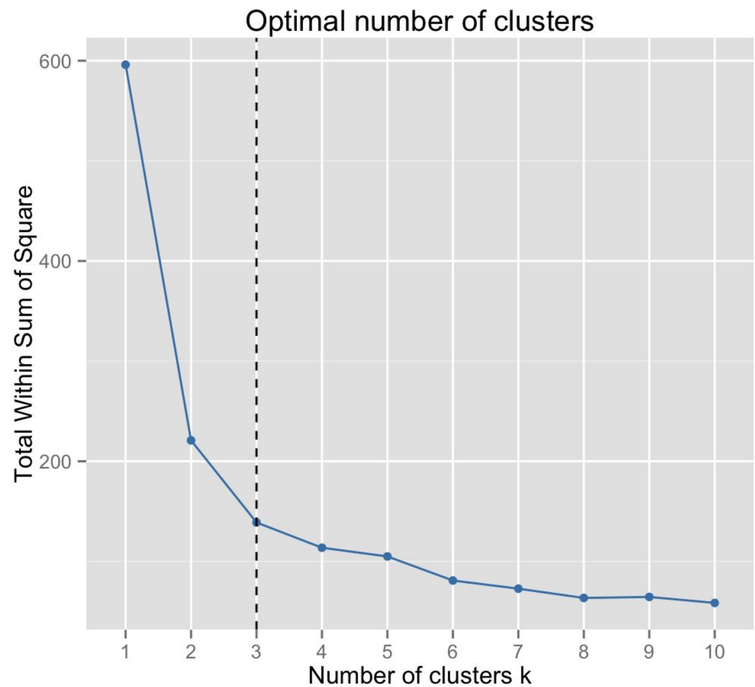
Choosing cluster counts: Elbow Method

The basic idea behind partitioning methods is to define clusters such that the total intra-cluster variation (total within-cluster sum of square, WCSS) is minimized

The *WCSS* measures the compactness of the clustering: we want it to be as small as possible!

- Compute clustering algorithm for different values of k and calculate the WCSS for each k
- Plot the curve of Total *WCSS* according to the number of clusters k
- The bending point in the plot is considered as an indicator of the appropriate number of clusters





Choosing cluster counts: Silhouette Method

- Silhouette analysis can be used to study the separation distance between the resulting clusters.
- The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

a(i) is the average distance between i and all other data within the same cluster

b(i) is the smallest average distance of i to all points in any other cluster

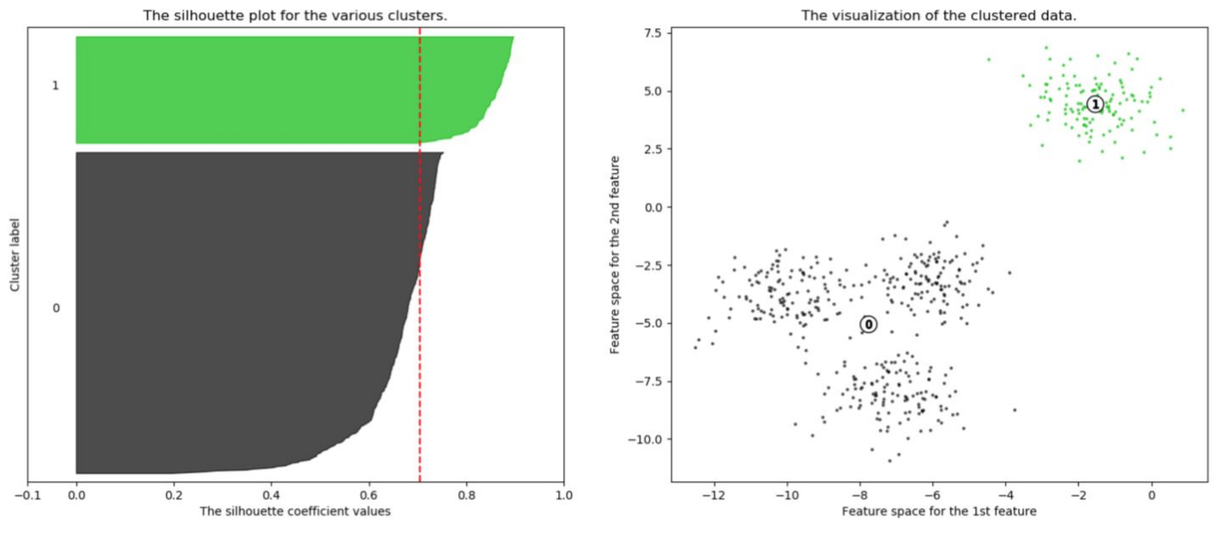
Which can be also written as:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

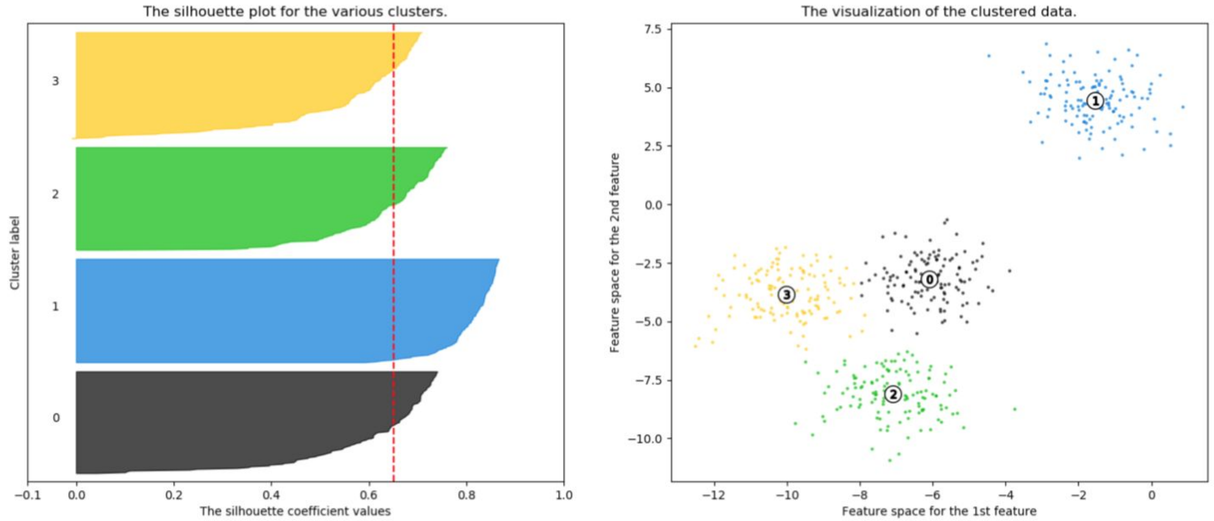
$$-1 \leq s(i) \leq 1$$

+1: sample is far away from the neighboring clusters.
0: on or very close to the decision boundary between two neighboring clusters.
Negative: samples might have been assigned to the wrong cluster.

Choosing cluster counts: Silhouette Method



Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



- Plot the Silhouette score of each observation (x-axis) vs its index (y-axis)
- Inspect for any irregularities, e.g. negative Silhouette scores

- Compare average Silhouette scores for different values of k

- Even though k=2 has a higher average Silhouette score than k=4, we may still prefer k=4, because it gives us more evenly-sized clusters

Live Coding - I

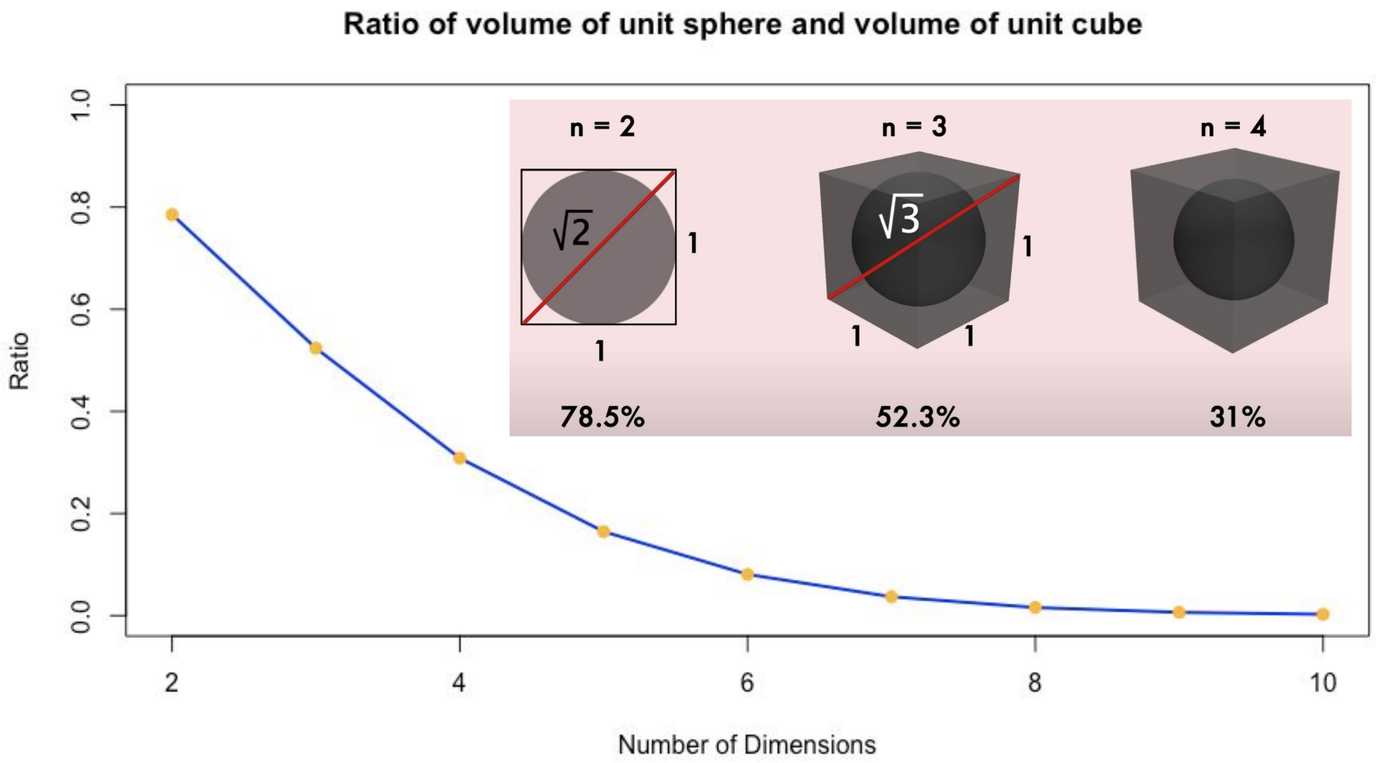
Training and Evaluating K-means & Hierarchical Clustering Models

- Get the notebook from:
  https://colab.research.google.com/drive/1bJn5dAYtk1kwZRxJMm4n8S4UexzR7oUs

- Run it on Colab by copying to your drive or local system & follow along with the instructor

- Recommended to run on Google Colab

The curse of dimensionality



- volume of space grows exponentially
- in high dimensions most points are located close to the border
- Euclidean distances lose their meaning (expected maximum and minimum distances between two random points become the same)

- not enough data in high dimensions
- clustering algorithms based on nearest neighbours become worse
- danger of overfitting

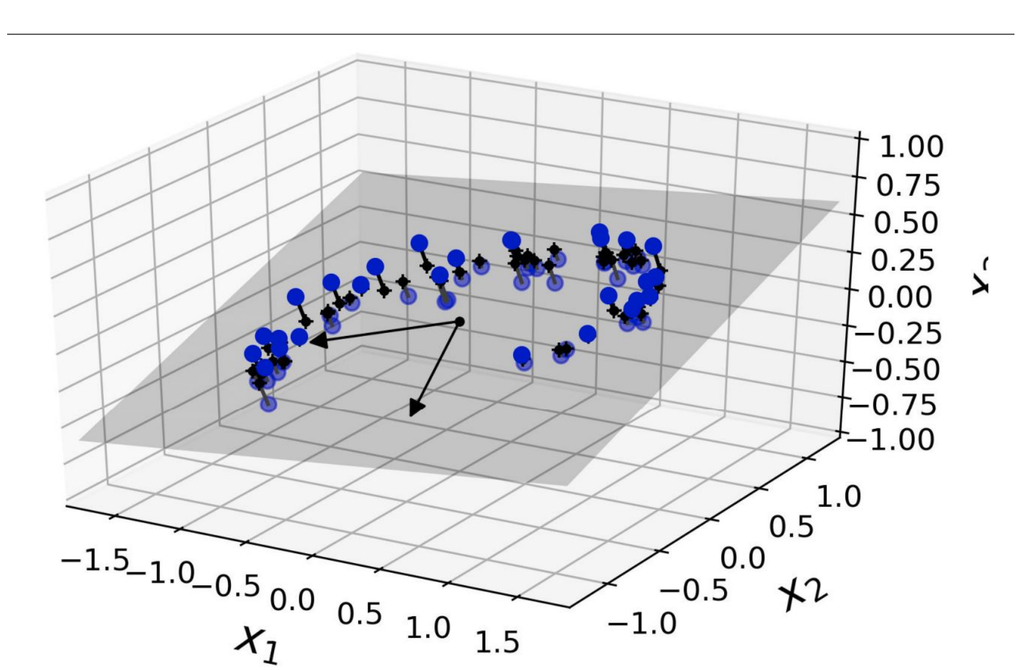
Note: Curse of dimensionality only if dimensions are independent

The blessing of dimensionality

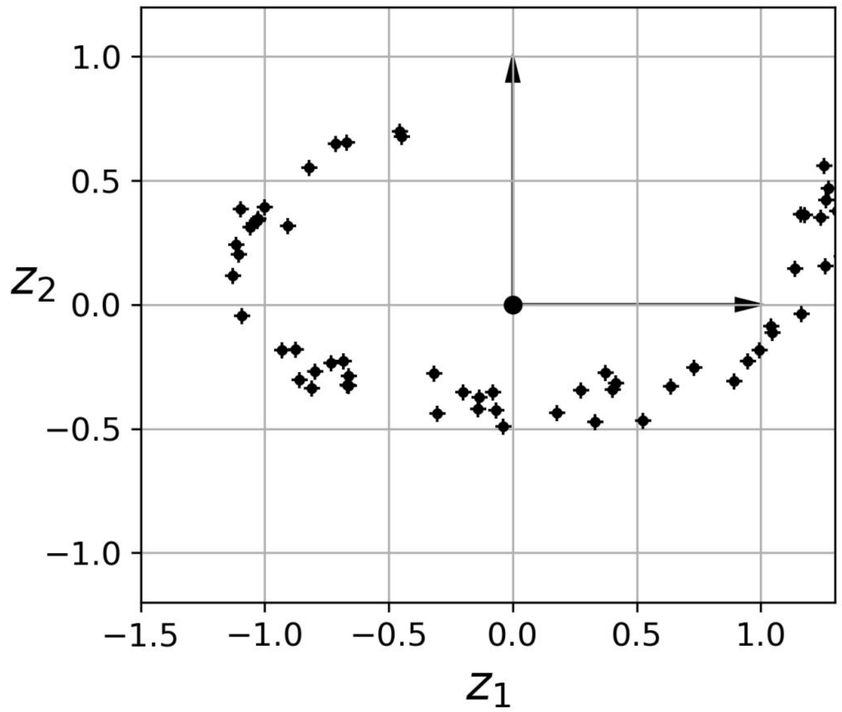
- Typical formalizations of curse of dimensionality affect i.i.d. data
- What really matters is the **signal-to-noise** ratio in the dataset
- Every dimension that adds information makes separation easier
- Moreover dimensions are correlated in real-world datasets
- Real-world datasets are not i.i.d. points in all dimensions -> there is a lot of structure to the observations. Taking advantage of that structure can actually yield better estimates
- Do not automatically assume that increasing dimensionality without corresponding increase in the number of samples is bad and should be avoided at all costs

Main Approaches for Dimensionality Reduction

- **Projection** - data is not always spread uniformly in all dimensions, but instead lies within a much smaller subspace

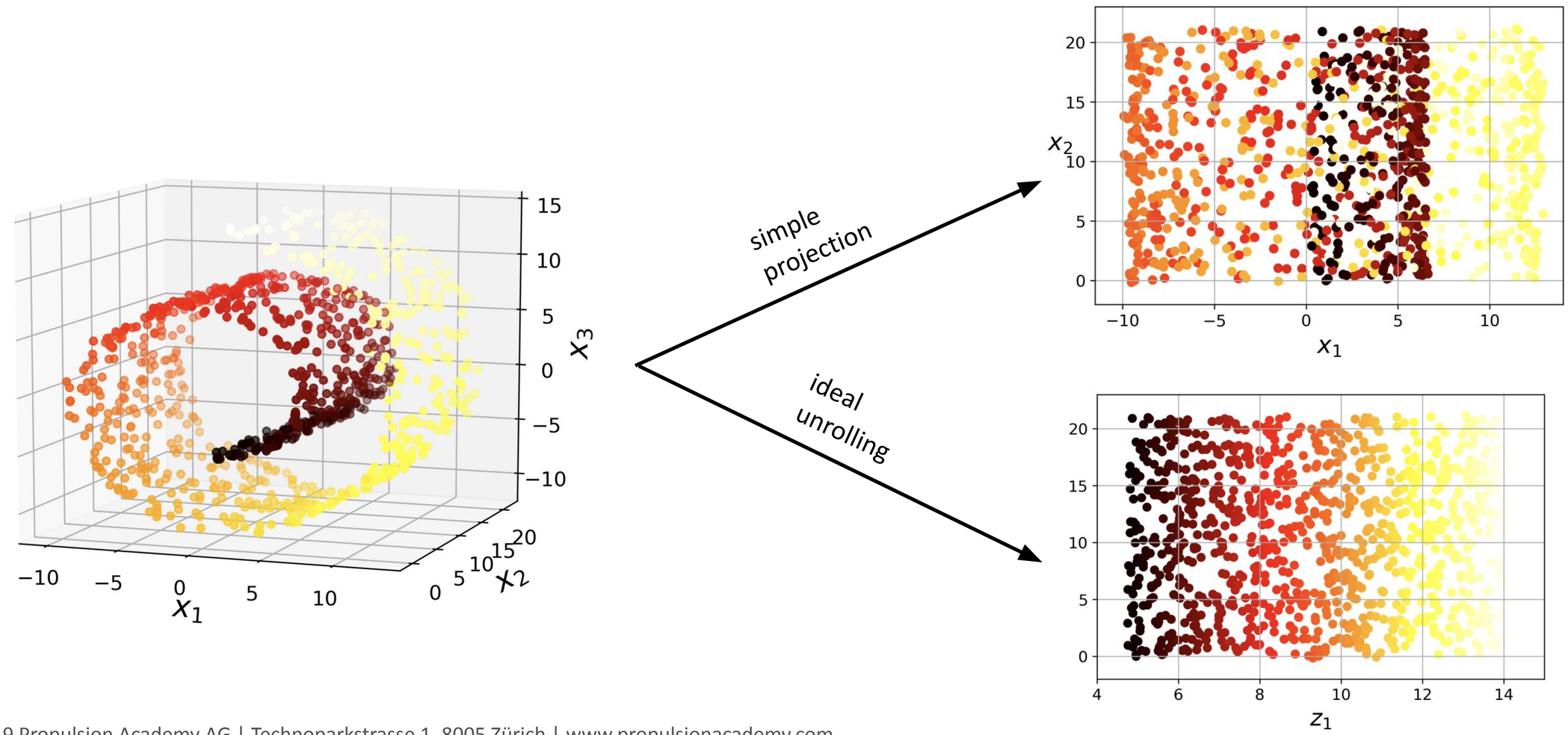


drop extra dimension



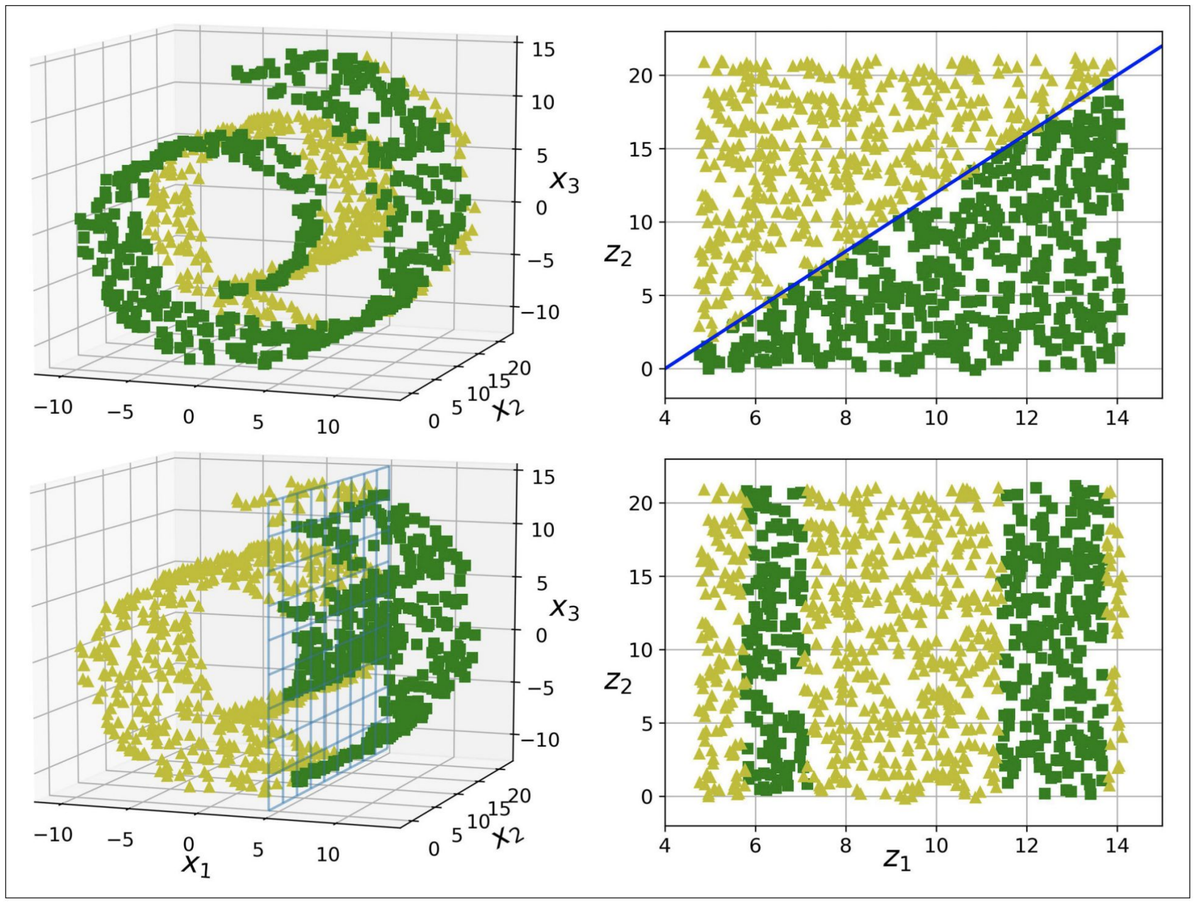
Main Approaches for Dimensionality Reduction

- Projection does not always works if subspace is twisted and turned.



Main Approaches for Dimensionality Reduction

- **Manifold Learning** - “learn” what the subspace is by modelling it
  - most real-world datasets lie close to a much lower-dimensional manifold (manifold hypothesis)

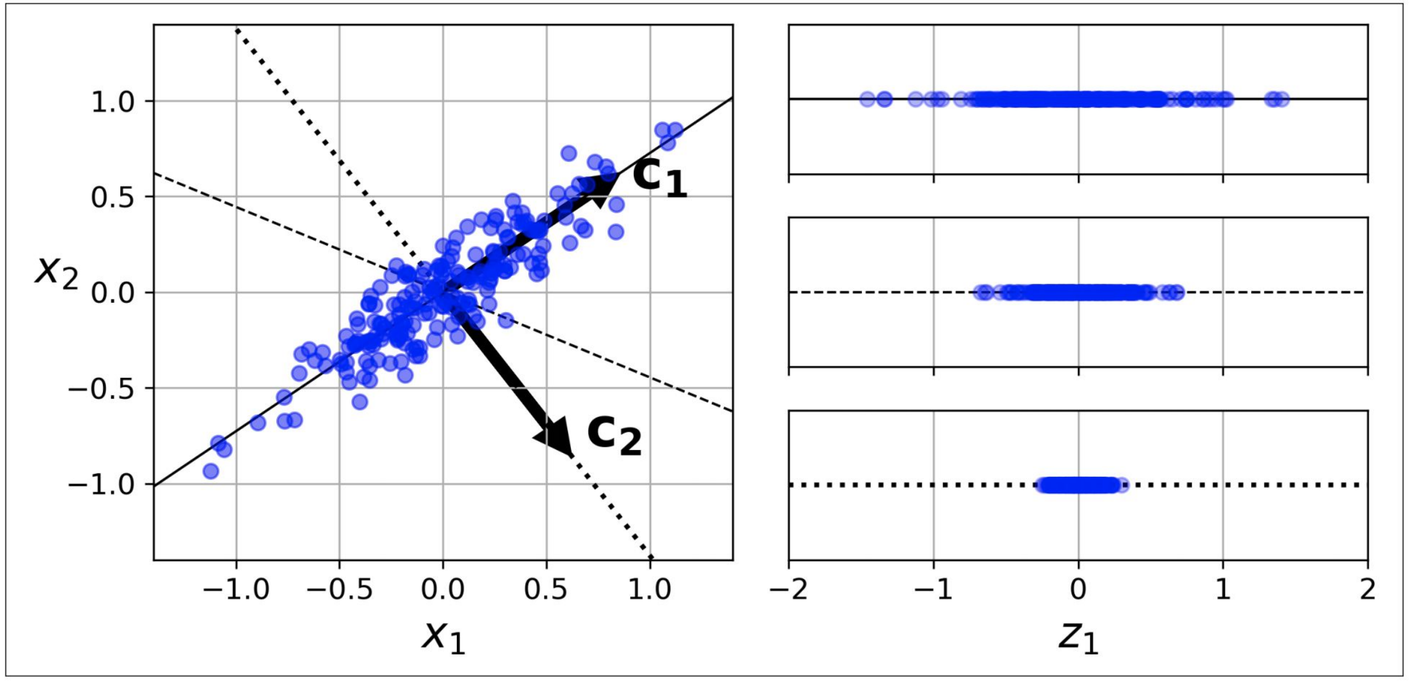


- In addition to the manifold hypothesis manifold learning makes the implicit assumption that the decision boundary in the lower-dimensional space is simpler than in high dimensions (top plots)

- This is, however, not always true (bottom plots)

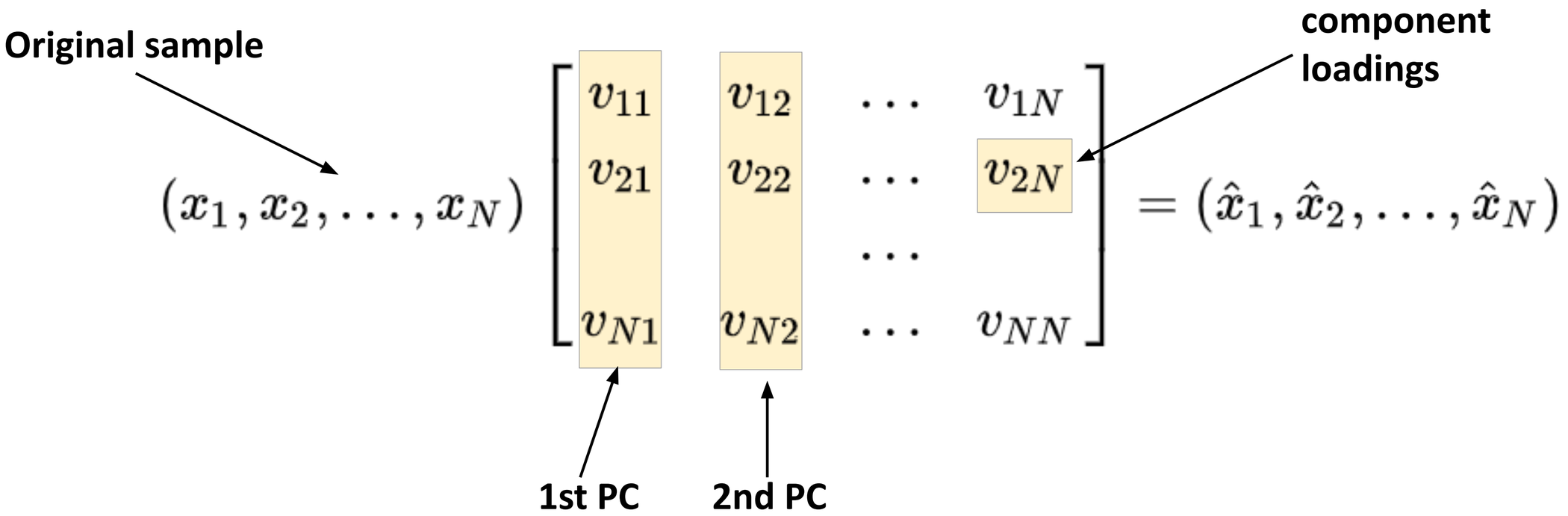
PCA - Principal Component Analysis

- PCA is a projection method to identify the best lower-dimensional hyperplane on which to project the data
- PCA identifies **orthogonal** axes that account for the largest amount of **variance** in the data:
  - the first axis accounts for maximum variance, second axis - maximum of the remaining variance and so on
- The ***i***th axis in the projection is called the ***i***th principal component of the data



PCA - Nothing more than simple matrix multiplication

- The projection of a vector **x** on another vector **k** is the **dot product** between **x** and **k**



- Even though PCA-transformed data results in features hard to interpret, component loadings can still give an indication as to the importance of the original features

PCA - How to compute

- How do we get the principal components?
- **Method 1: Singular Value Decomposition** of the data matrix X
  - Generalization of the eigendecomposition of a square matrix to any mxn matrix

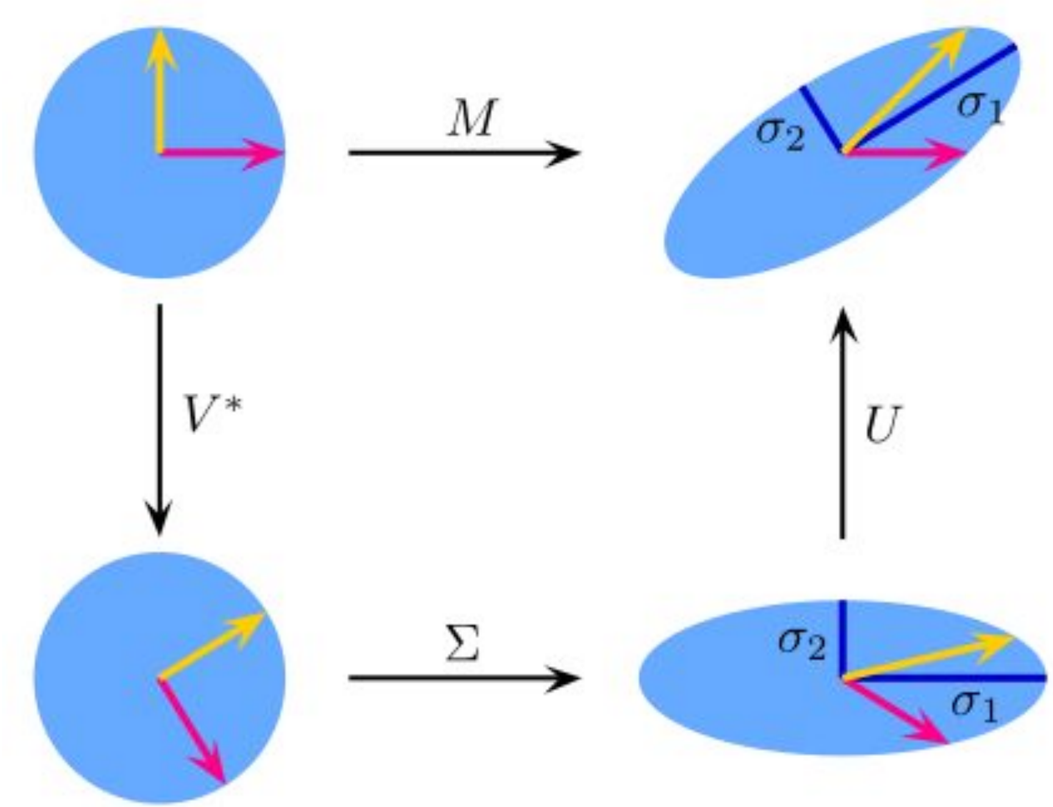
$$\mathbf{X} = \underbrace{\mathbf{U}}_{\text{Unitary Matrix}} \times \underbrace{\mathbf{\Sigma}}_{\text{Diagonal Matrix}} \times \underbrace{\mathbf{V}^T}_{\text{Matrix of principal components}}$$

- **Method 2: Eigendecomposition** of the **symmetric** Covariance Matrix of the data X

$$\mathbf{Q} \propto \mathbf{X}^T\mathbf{X} = \underbrace{\mathbf{W}}_{\text{Matrix of Principal Components, column-wise}} \times \underbrace{\Lambda}_{\text{Diagonal Matrix of eigenvalues of Q}} \times \mathbf{W^T}$$

- Method 1 is more stable, faster and thus the default computation method

Singular Value Decomposition

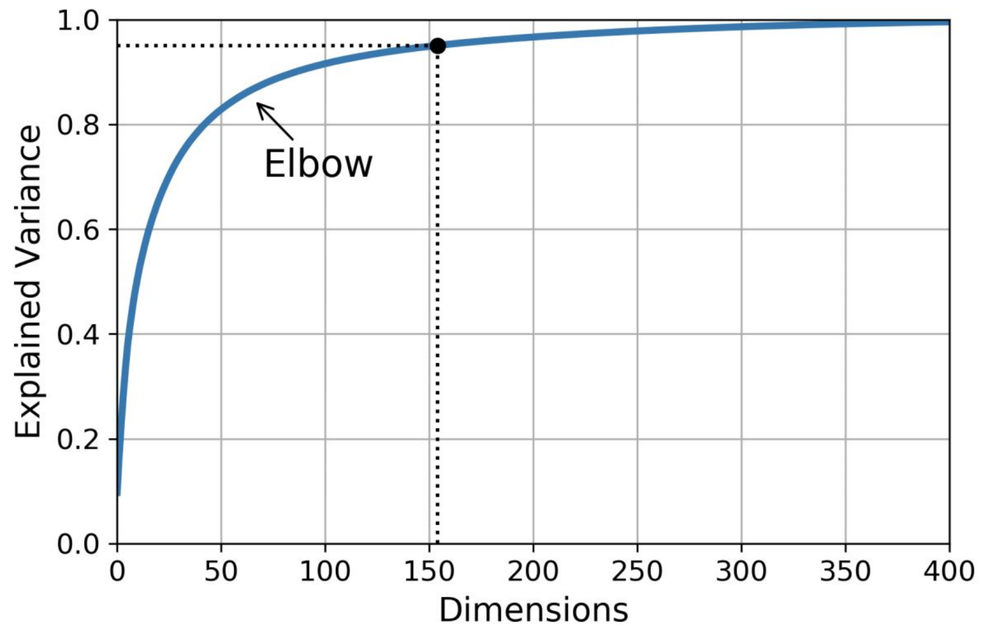


$$M = U \cdot \Sigma \cdot V^*$$

- **Top**: The action of **M**, indicated by its effect on the unit disc D and the two canonical unit vectors e1 and e_2.
- **Left**: The action of **V***, a rotation, on D, e1, and e2.
- **Bottom**: The action of **Σ**, a scaling by the singular values σ1 horizontally and σ2 vertically.
- **Right**: The action of **U**, another rotation.

PCA - Explained Variance Ratio

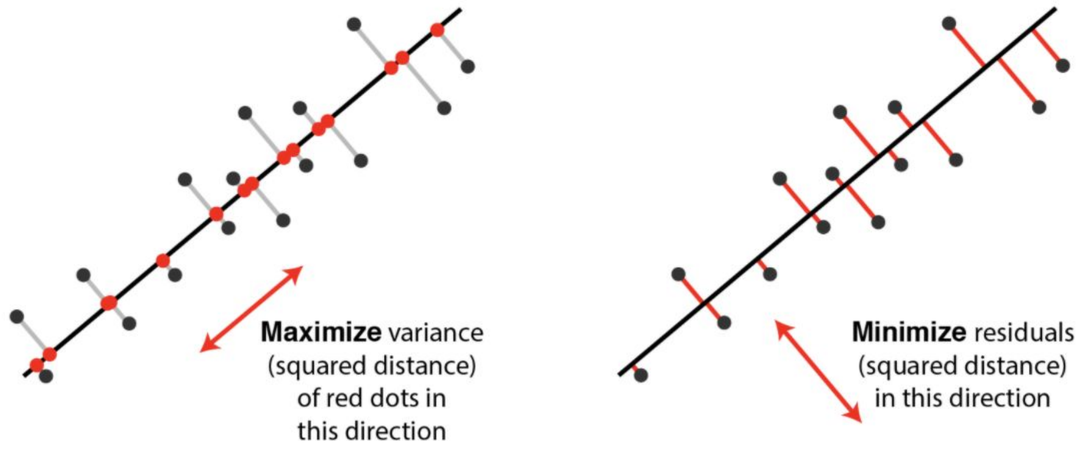
- A common method to choose the right number of dimensions is to plot the principal components vs the percentage of the variance that they explain and then cut at a threshold (e.g. 95%)



- If you use PCA for visualization only it is common to choose only the top 2 or 3 PCs.

PCA - Why do we care about the variance in the first place?

- Maximizing variance is equivalent to minimizing squared distances between original points and their projections



- Maximizing variance aims at **preserving the distances between the original data points** as much as possible
  - underlying assumption in many dimensionality reduction techniques is that distances between points are **informative**

PCA - Summary

- PCA assumes that features have std. dev. of 1
- One can take output as an input to regression (principal component regression)
- PCA assumes that the principle components are linear combinations of the original features
- PCA can fail if:
  - non-linear patterns -> try KernelPCA
  - distances in the original space (i.e. variance) are not the informative criteria for separating the data
  - sometimes the top PC is not the most informative if the variance in that direction is not informative

Live Coding - II

Understanding how PCA works

- Get the notebook from:
  https://colab.research.google.com/drive/1HV4i8pa_ijKLiquFZj2LG1dFiCkYnLuD

- Run it on Colab by copying to your drive or local system & follow along with the instructor

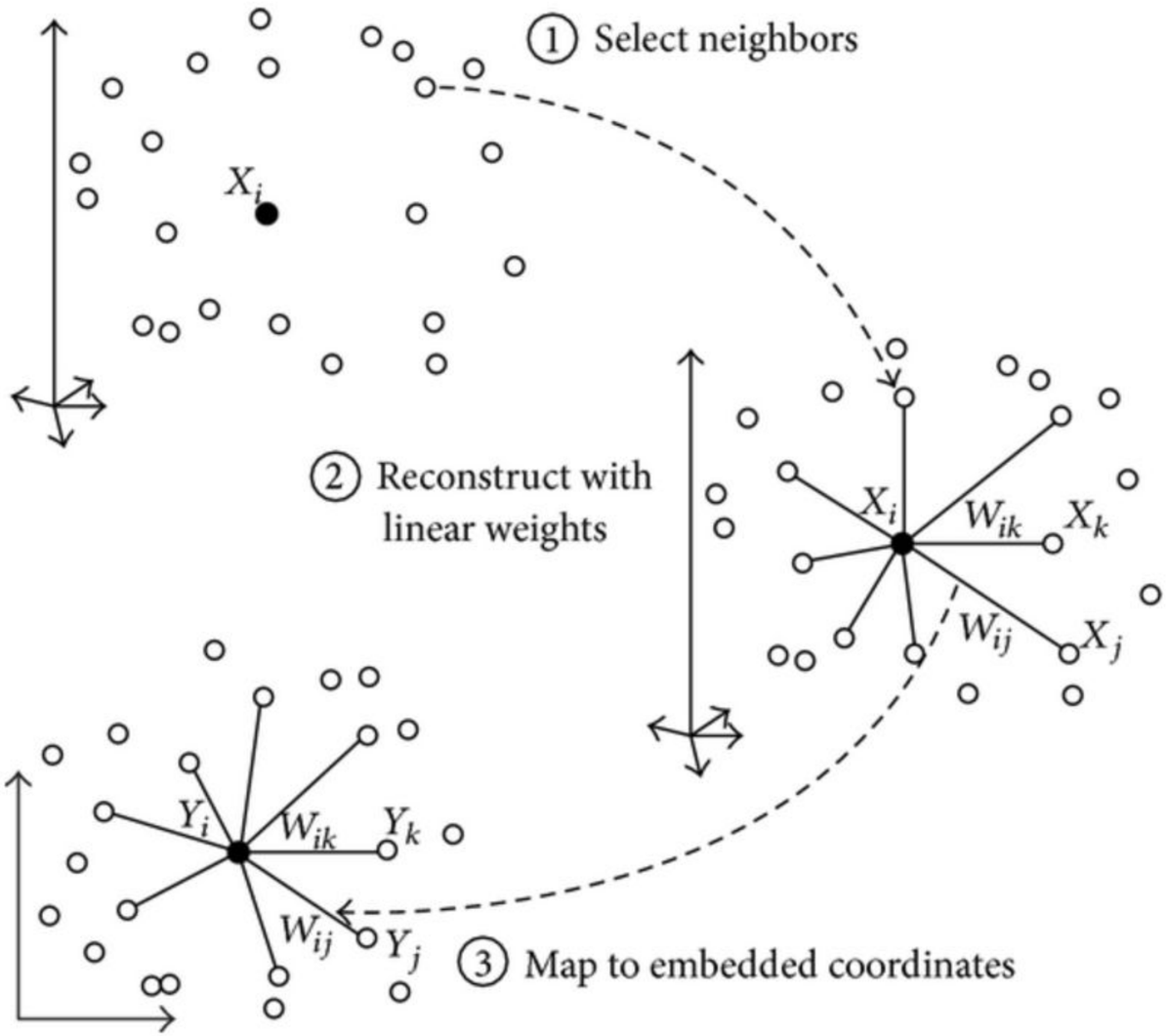
- Recommended to run on Google Colab

Manifold Learning - Locally Linear Embedding (LLE)

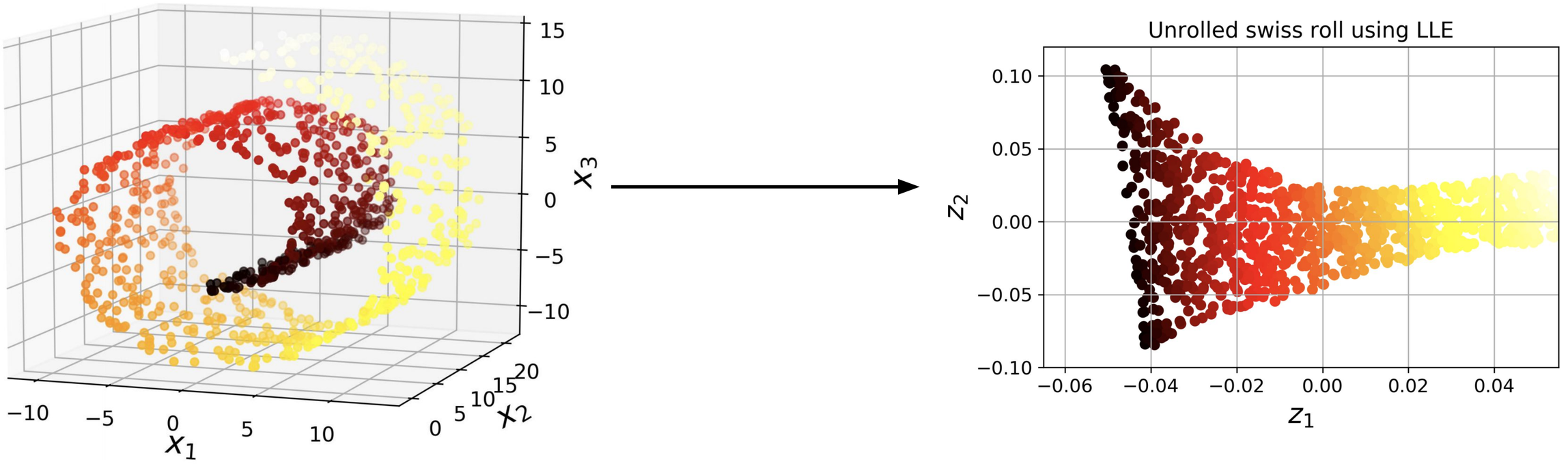
- Measure how each training instance relates to its neighbours and then find a low-dimensional representation where these local relations are best preserved
- Best when dataset has local structure (e.g. Swiss roll dataset)

1. For each training instance **x_i** identify the **k** nearest neighbours
2. Reconstruct **x_i** as a linear combination of these neighbours
   a. find weights **w_ij**, such that $\vec{x}_i = \sum_{j=1}^{k} w_{ij}\vec{x}_j$ and $\sum_{j=1}^{k} w_{ij} = 1, \forall i$
   b. do a) by minimizing the function $\sum_{i=1}^{N} \left(\vec{\mathbf{x}}_\mathbf{i} - \sum_j w_{ij}\vec{\mathbf{x}}_\mathbf{j}\right)^2$
3. Map training instances from **n-dimensional** space to **d-dimensional** space (d < n) preserving the weights **w_ij** as much as possible. The cost function to minimize is: $\sum_{i=1}^{N} \left(\vec{z}_i - \sum_j \mathbf{w}_\mathbf{ij}\vec{z}_j\right)^2$
   a. Note, here **w_ij** are constant and we change the projected points **z**

Manifold Learning - Locally Linear Embedding (LLE)



Manifold Learning - Locally Linear Embedding (LLE)



Manifold Learning - t-Distributed Stochastic Neighbor Embedding (t-SNE)

- t-SNE calculates a similarity measure between pairs of instances in the high dimensional space and in the low dimensional space.
- Then it tries to optimize these two similarity measures using a cost function.

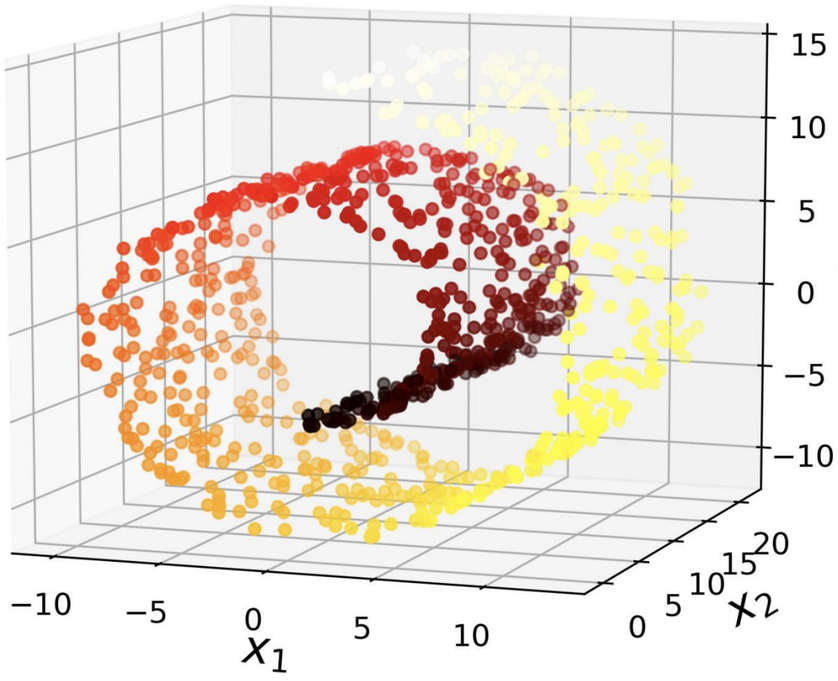
t-SNE in three steps:

1. Measures similarities between points in the **high dimensional space**.
   - For each data point (**xi**) we center a Gaussian distribution over that point.
   - Then we measure the density of all points (**xj**) under that Gaussian distribution.
   - This gives us a set of probabilities (**Pij**) for all points.
   - The Gaussian distribution or circle can be manipulated using what's called perplexity, which influences the variance of the distribution (circle size) and essentially the number of nearest neighbors considered

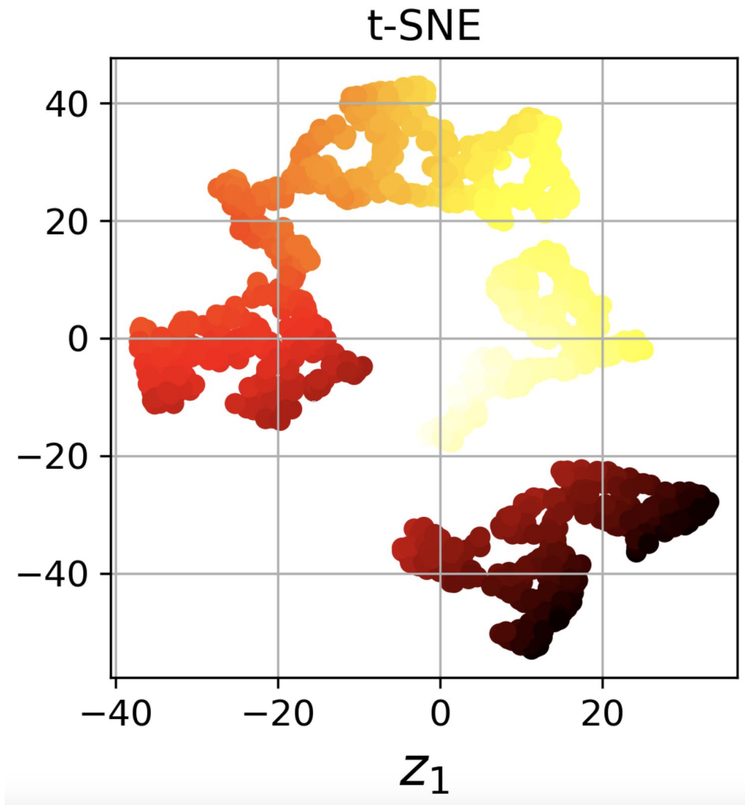
Manifold Learning - t-Distributed Stochastic Neighbor Embedding (t-SNE)

2. Similar to step 1, it does the same steps in the **lower-dimensional space** using a Student t-distribution with one degree of freedom, which is also known as the Cauchy distribution.
    - This gives us a second set of probabilities (**Qij**) in the low dimensional space.
3. In the final step we want these set of probabilities from the low-dimensional space (Qij) to reflect those of the high dimensional space (Pij) as best as possible.
    - We measure the difference between the probability distributions of the two-dimensional spaces using Kullback-Liebler divergence (KL).
    - We use gradient descent to minimize our KL cost function.

Manifold Learning - t-SNE on the Swiss roll dataset



`sklearn.manifold.TSNE`



Manifold Learning - Uniform Manifold Approximation and Projection (UMAP)

- At its core very similar to t-SNE: both arrange data in lower dimensional space, such that the resulting graph is as structurally similar to the higher dimensional space as possible
- UMAP builds a high-dimensional graph in which nodes represent data points and weighted edges represent probabilities that points are connected
  - UMAP extends a radius outwards from each point, connecting points when those radii overlap
  - Radius is chosen locally based on the distance to each point's ***n***th nearest neighbours

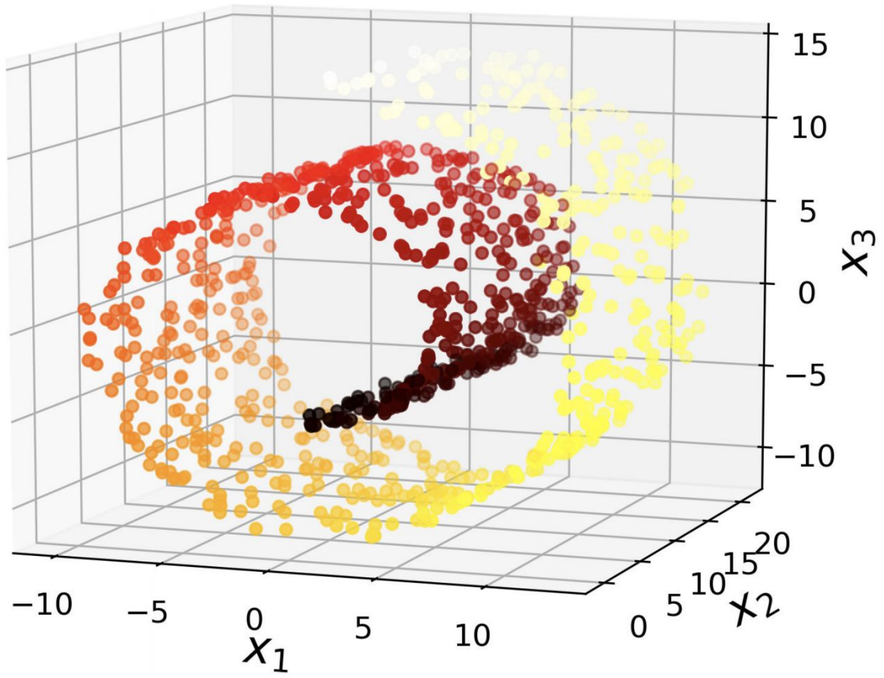
[Interactive Visualization to show the building of the graph (Figure 3)]

Manifold Learning - Uniform Manifold Approximation and Projection (UMAP)

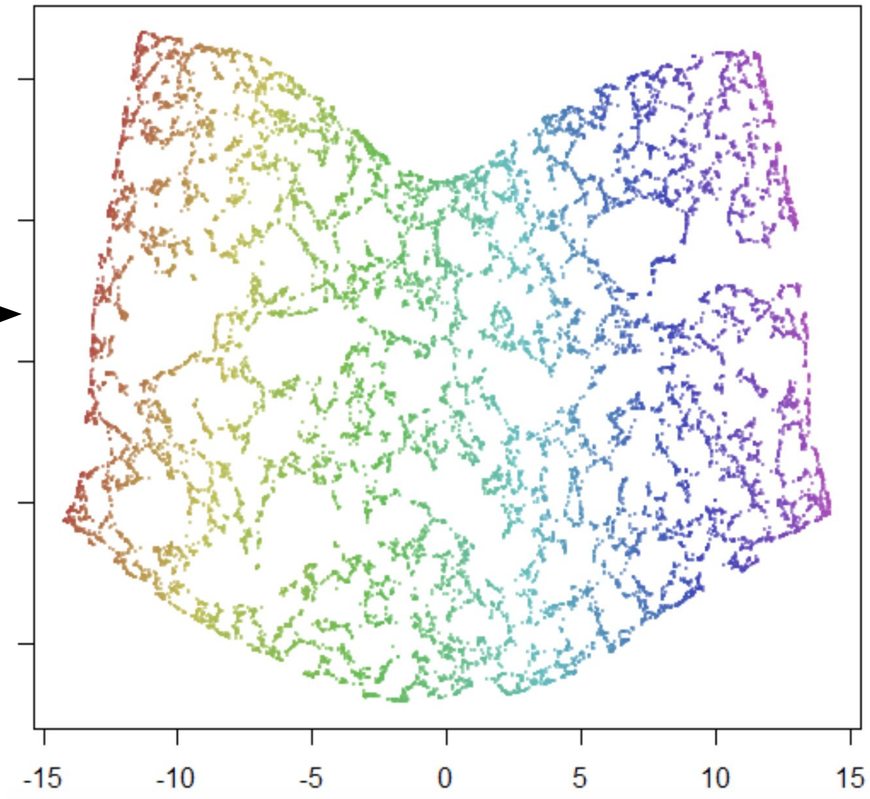
- Once the high dimensional graph is constructed, UMAP optimizes the layout of the low-dimensional representation to be as similar as possible. Similar to t-SNE
- There are 2 important parameters in constructing the high-dimensional graph
  - **number of neighbours** - the number of nearest neighbours considered. High values wash out the local structure in favour of global connectedness
  - **minimum distance** - the minimum distance between any 2 connected points. High values will spread the points more, i.e. focus on global structure,. Low values will pack points closely together

- The biggest advantage of UMAP over t-SNE is the more optimal balance between local and global structure and the computational efficiency
- Package **`umap`** in Python

Manifold Learning - Uniform Manifold Approximation and Projection (UMAP)



note: colors differ



Other dimensionality reduction techniques

- Random projections
  - projects data into lower dimensions using a random linear projection. Works surprisingly well (see **`sklearn.random_projections`** and associated docs)
- Multidimensional scaling (MDS)
  - preserves (Euclidean) distances between points from high-dimensional space into lower dimensional space (**`sklearn.manifold.MDS`**)
- Isomap
  - extends MDS by trying to preserve geodesic distances (**`sklearn.manifold.Isomap`**)
- Linear Discriminant Analysis (LDA)
  - classification algorithm that during training learns the most discriminant axes between classes and these axes are then used to project the data onto (**`sklearn.discriminant_analysis.LinearDiscriminantAnalysis`**)

Live Coding - III

Dimensionality Reduction Techniques

- Get the notebook from:
  https://colab.research.google.com/drive/13UNmp5dvAalz0gwoRnDZ6_D47uO1UdcF

- Run it on Colab by copying to your drive or local system & follow along with the instructor

- Recommended to run on Google Colab

Live Coding - IV

Case-Study: Customer Segmentation

- Get the notebook from:
  https://colab.research.google.com/drive/1Q1U08qBq7UT0tad3b27DugMLBRXeECFT

- Run it on Colab by copying to your drive or local system & follow along with the instructor

- Recommended to run on Google Colab