Unsupervised Learning

Doesn’t need a teacher.

Introduction

- Unsupervised learning subsumes all kinds of machine learning where there is no known output, no teacher to instruct the learning algorithm
- The learning algorithm is just shown the input data, and asked to extract knowledge from this data

TYPES OF UNSUPERVISED LEARNING

Transformations of Dataset **Clustering**

Unsupervised Transformation of Dataset

- Unsupervised transformations of a dataset are algorithms that create a new representation of the data which might be easier for humans or other machine learning algorithms to understand
- A common application of unsupervised transformations is dimensionality reduction
- Which takes high-dimensional representation of the data, consisting of many features, and finding a new way to represent this data that summarizes the essential characteristics about the data with fewer features.
    - A common application for dimensionality reduction is reduction to two dimensions for visualization purposes.
- Another application for unsupervised transformations is finding the parts or components that “make up” the data.
    - An example of this is topic extraction on collections of text documents.
        - The task is to find the unknown topics that are talked about in each document, and to learn what topics appear in each document.

Clustering Algorithms

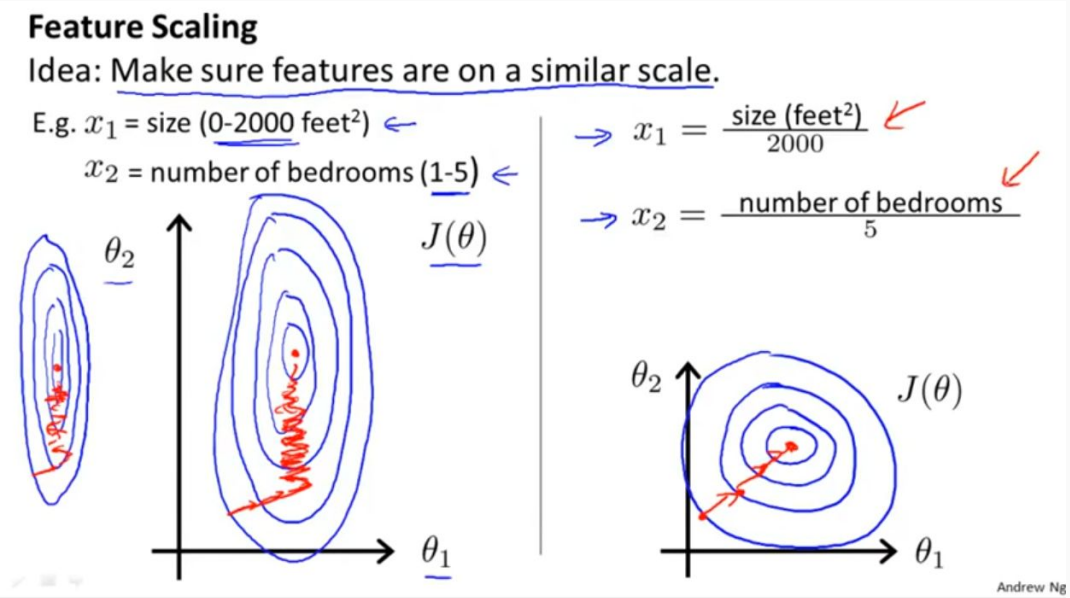
- Clustering algorithms on the other hand partition data into distinct groups of similar items.
- Consider the example of uploading photos to a social media site.
  - To allow you to organize your pictures, the site might want to group together pictures that show the same person.
  - However, the site doesn't know which pictures show whom, and it doesn't know how many different people appear in your photo collection.
  - A sensible approach would be to extract all faces, and divide them into groups of faces that look similar.
  - Hopefully, these correspond to the same person, and can be grouped together for you.

Challenges in unsupervised learning

- A major challenge in unsupervised learning is evaluating whether the algorithm learned something useful.
- Unsupervised learning algorithms are usually applied to data that does not contain any label information, so we don't know what the right output should be.
- Therefore it is very hard to say whether a model "did well".
  - Example: clustering algorithm could have grouped all face pictures that are shown in profile together, and all the face pictures that are face-forward together.
  - There is no way for us to "tell" the algorithm what we are looking for, and often the only way to evaluate the result of an unsupervised algorithm is to inspect it manually.
- As a consequence, unsupervised algorithms are used often in an exploratory settings.

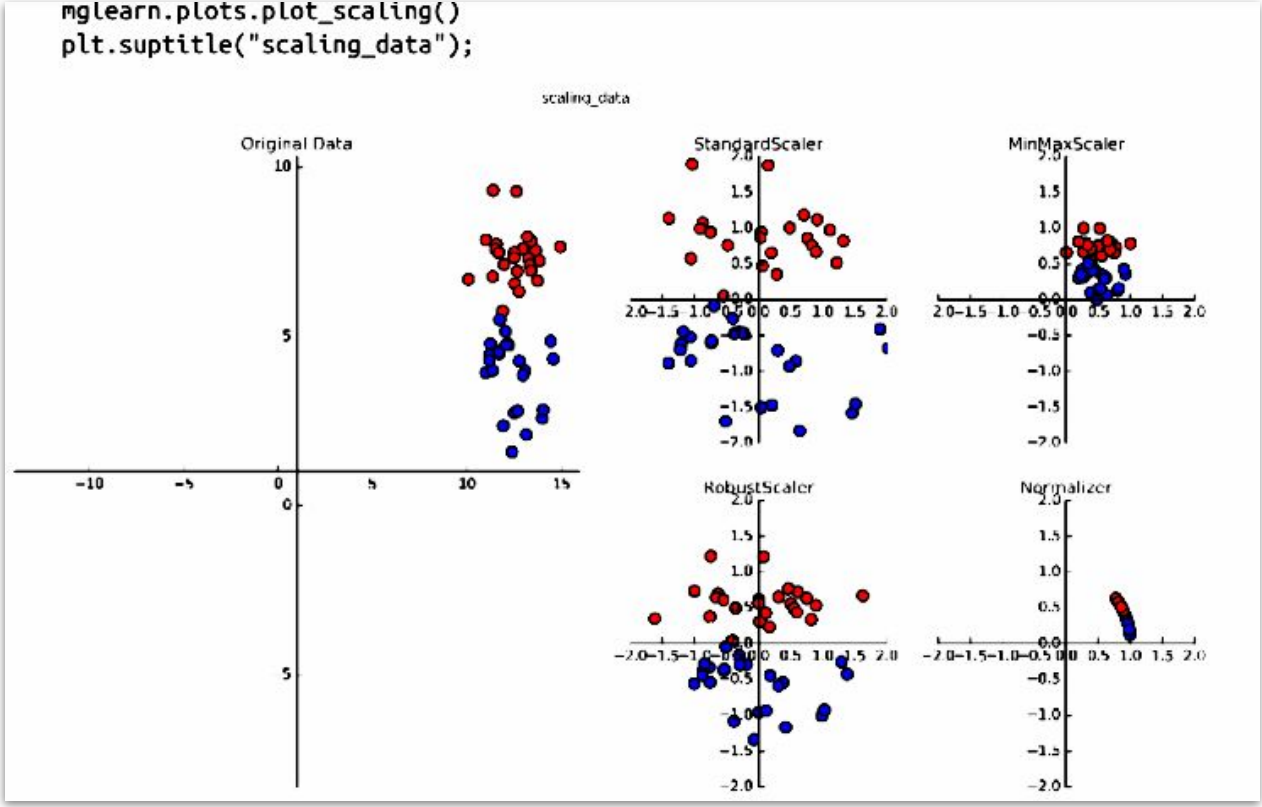
Preprocessing and Scaling

- Some algorithms, like neural networks and SVMs, are very sensitive to the scaling of the data.
- Therefore a common practice is to adjust the features so that the data representation is more suitable for these algorithms.



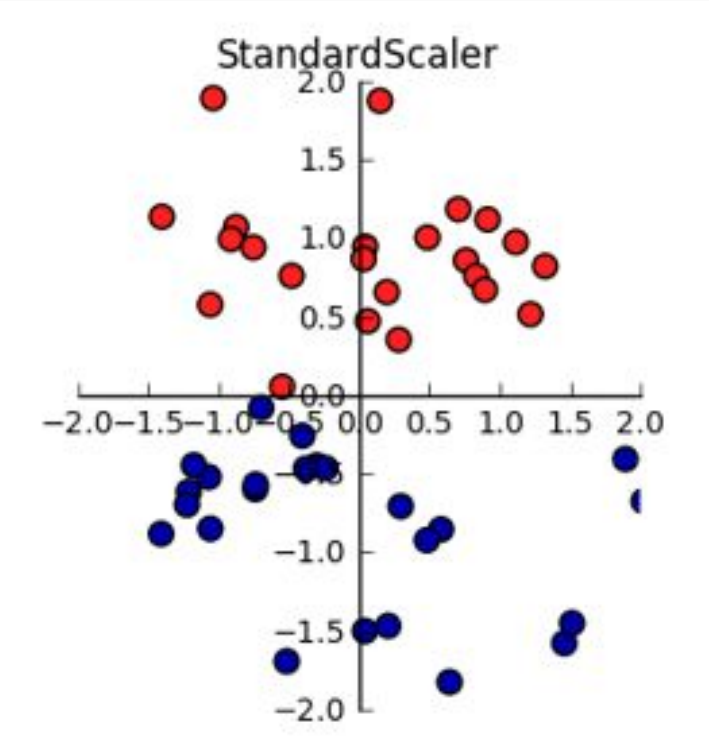
- Some examples:

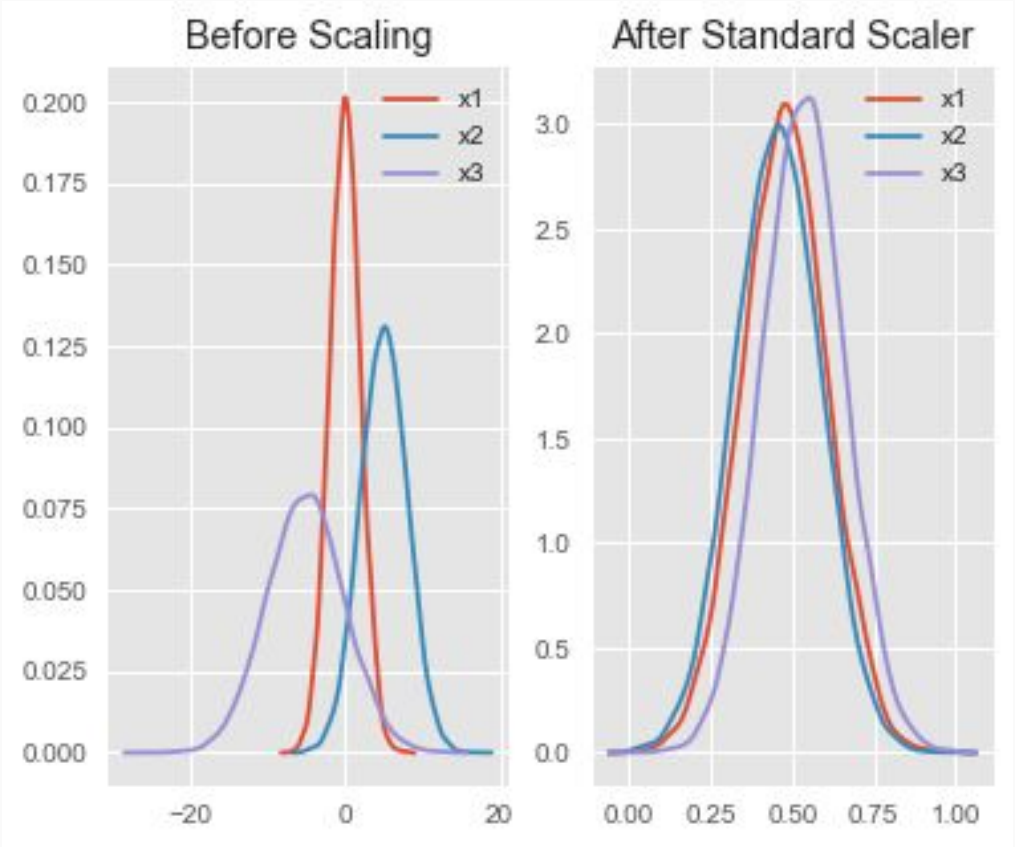
```
mglearn.plots.plot_scaling()
plt.suptitle("scaling_data");
```



Different kinds of preprocessing

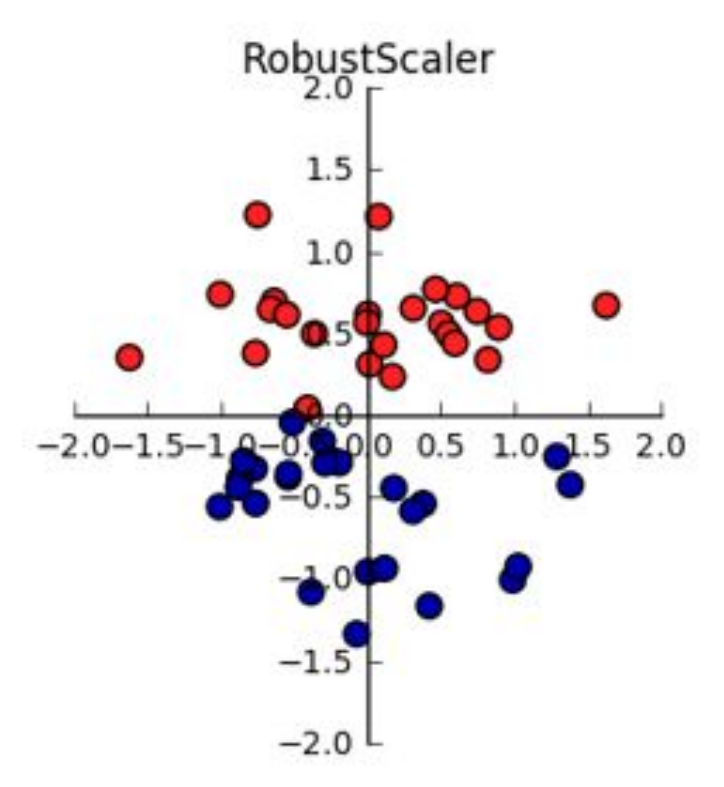
- The four plots shown in previous slides are four different ways to transform the data that yield more standard ranges.
  - The StandardScaler in scikit-learn ensures that for each feature, the mean is zero, and the variance is one, bringing all features to the same magnitude.
    - However, this scaling does not ensure any particular minimum and maximum values for the features.

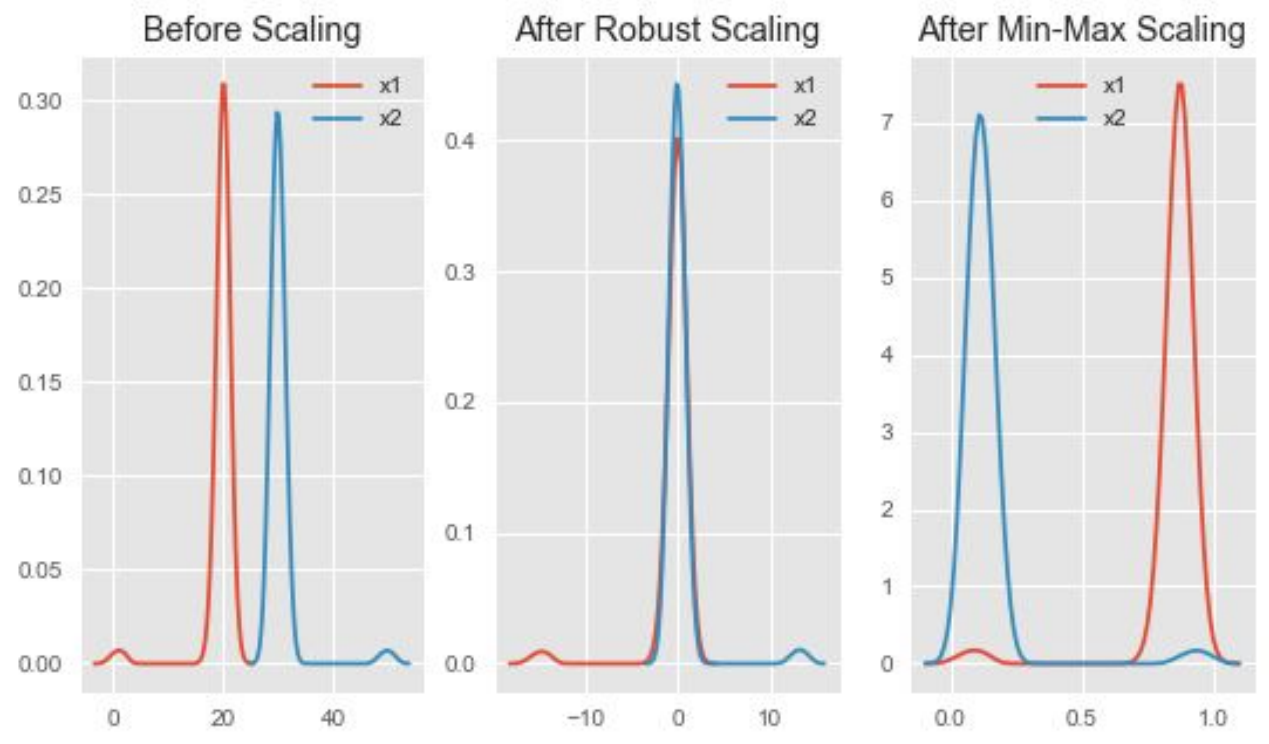



Before Scaling
After Standard Scaler
x1
x2
x3
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−20
0
20
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00

Different kinds of preprocessing

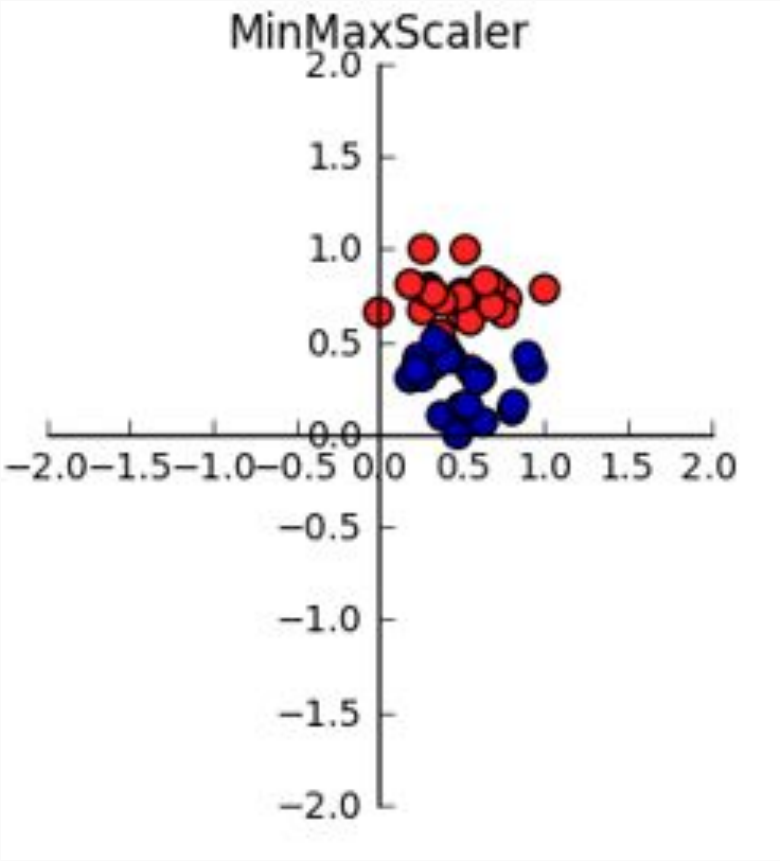
- The RobustScaler works similarly to the StandardScaler in that it ensures statistical properties for each feature that guarantee that they are on the same scale.
  - However, RobustScaler uses the median and quartiles, instead of mean and variance.
  - This makes the RobustScaler ignore outliers (like measurement errors).

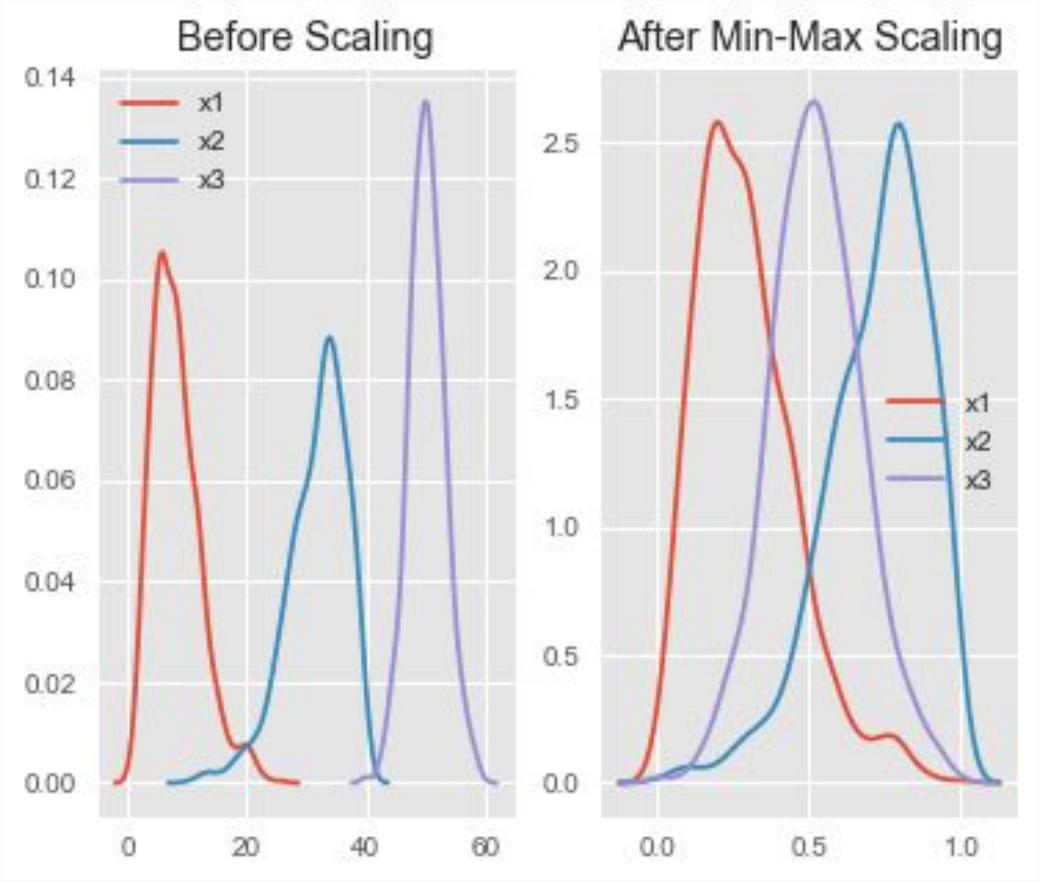



Before Scaling
x1
x2
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
After Robust Scaling
x1
x2
0.4
0.3
0.2
0.1
0.0
−10
0
10
After Min-Max Scaling
x1
x2
7
6
5
4
3
2
1
0
0.0
0.5
1.0

Different kinds of preprocessing

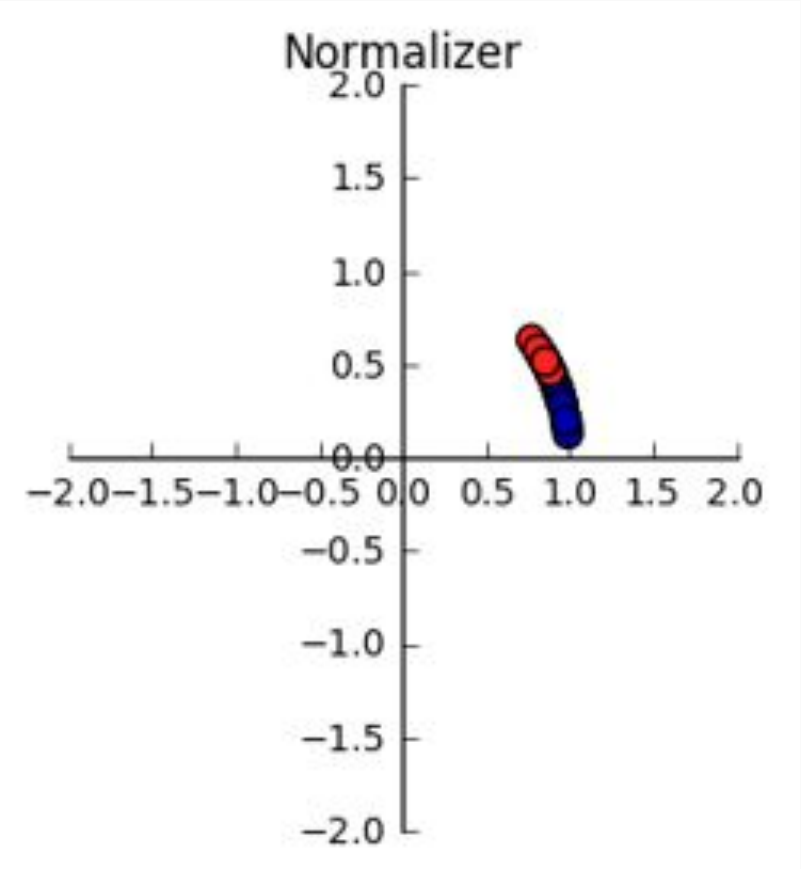
- The MinMaxScaler on the other hand shifts the data such that all features are exactly between 0 and 1.
- For the two-dimensional dataset this means all of the data is contained within the rectangle created by the x axis between 0 and 1 and the y axis between zero and one.

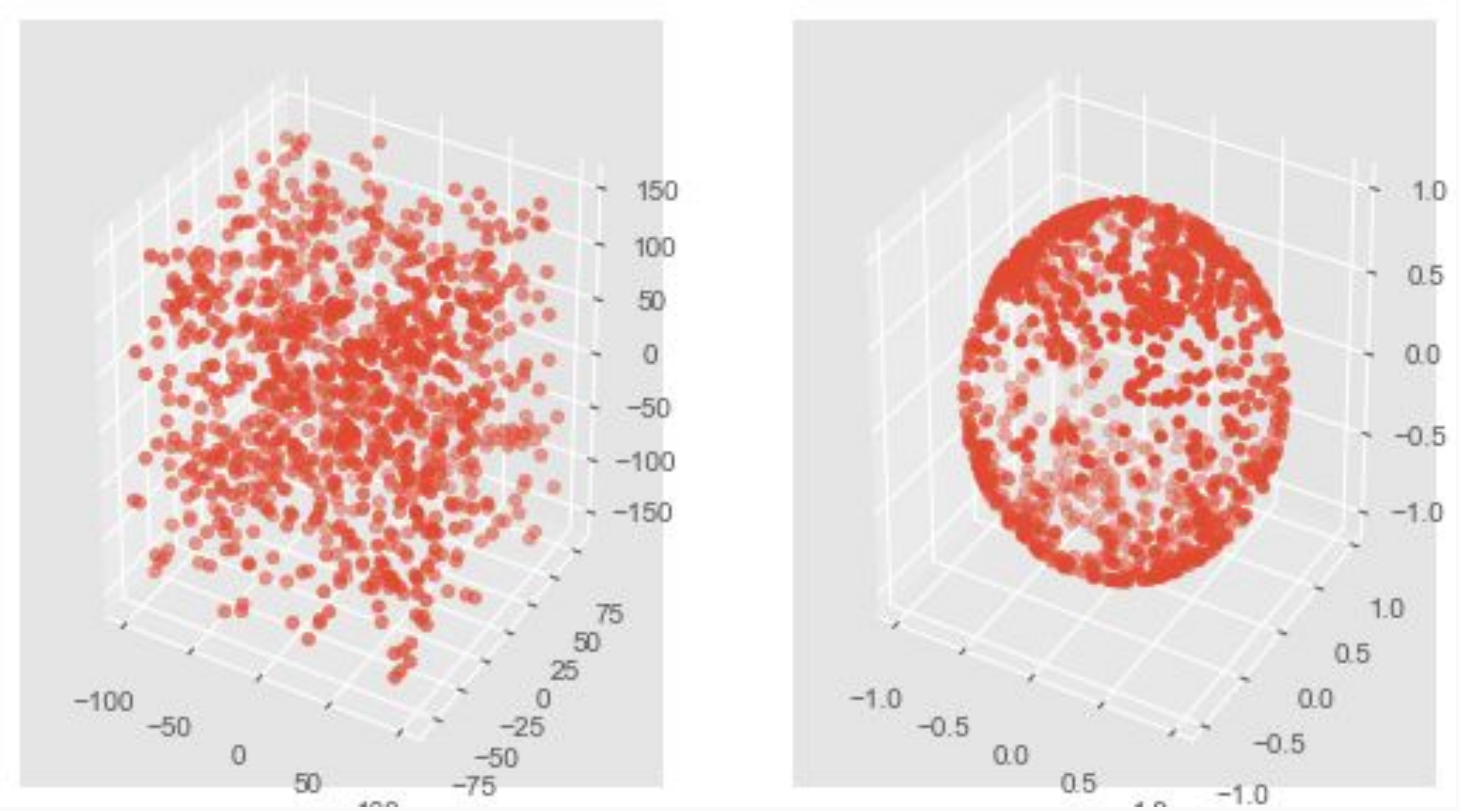



Before Scaling
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
x1
x2
x3
0
20
40
60
After Min-Max Scaling
2.5
2.0
1.5
1.0
0.5
0.0
x1
x2
x3
0.0
0.5
1.0

Different kinds of preprocessing

- Finally, the Normalizer does a very different kind of rescaling.
- It scales each data point such that the feature vector has a euclidean length of one.
- In other words, it projects a data point on the circle (or sphere in the case of higher dimensions) with a radius of 1.
- This means every data point is scaled by a different number (by the inverse of it's length).



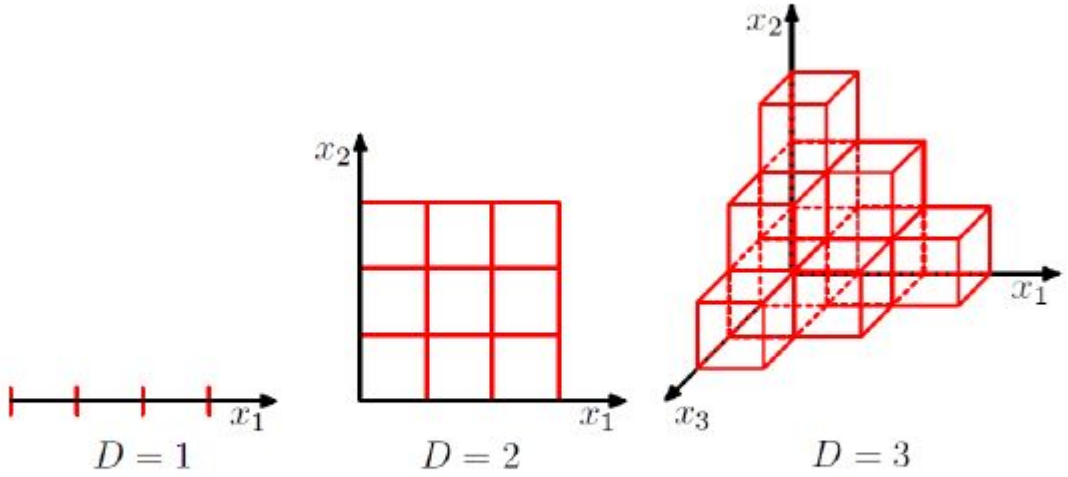

150
100
50
0
−50
−100
−150
−100
−50
0
50
75
50
25
0
−25
−50
−75
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
0.5
0.0
−0.5
−1.0

Dimensionality Reduction, Feature Extraction and Manifold Learning

- The most common motivations for transforming data are visualization, compressing the data, and finding a representation that is more informative for further processing.
- One of the simplest and most widely used algorithms for all of these is **Principal Component Analysis.**

Curse of Dimensionality

Exponentially large # of examples required to "fill up" high-dim spaces



Fewer dimensions ⇒ Less chances of overfitting ⇒ Better generalization

Dimensionality reductions is a way to beat the curse of dimensionality

Principal Component Analysis (PCA)

- As the dimensions of data increases, the difficulty to visualize it and perform computations on it also increases.
  - Remove the redundant dimensions
  - Only keep the most important dimensions

Variance : It is a measure of the variability or it simply measures how spread the data set is. Mathematically, it is the average squared deviation from the mean score. We use the following formula to compute variance *var(x)*.

$$var(x) = \frac{\sum (x_i - \bar{x})^2}{N}$$

$$cov(x, y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{N}$$

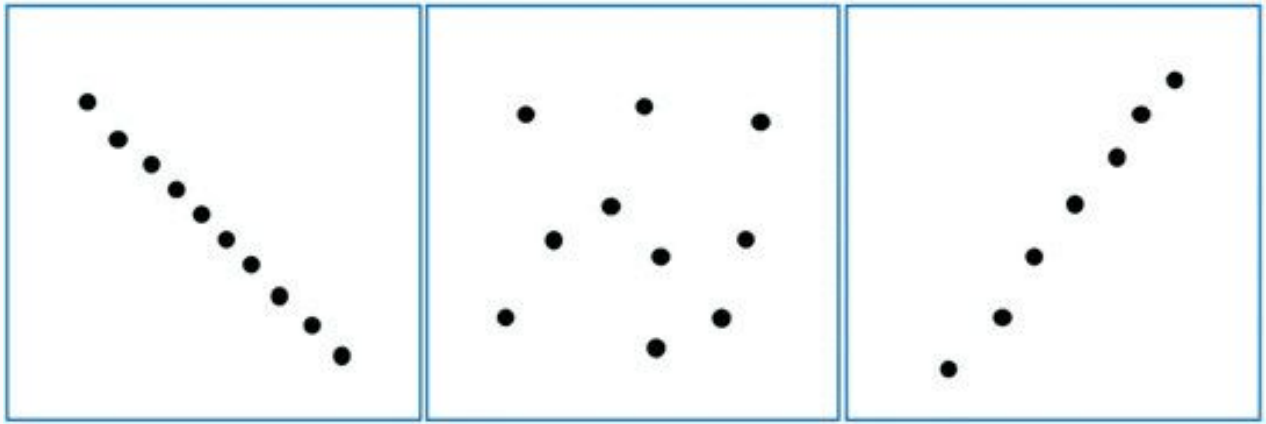
PCA

Covariance : It is a measure of the extent to which corresponding elements from two sets of ordered data move in the same direction. Formula is shown above denoted by *cov(x,y)* as the covariance of *x* and *y*.

Here, *xi* is the value of x in *ith* dimension.

x bar and *y bar* denote the corresponding mean values.

One way to observe the covariance is how interrelated two data sets are.



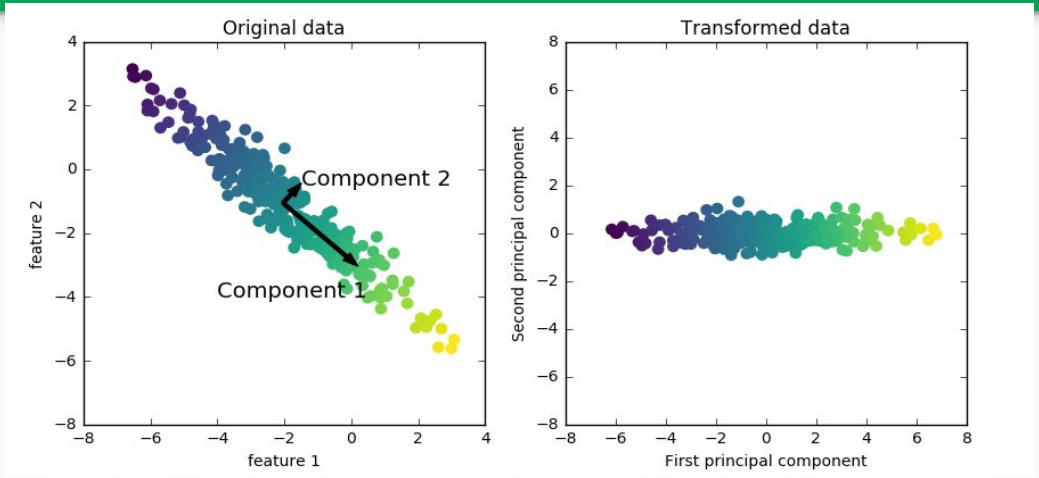
PCA

- Now let's think about the requirement of data analysis.
  - We want the data to be spread out across each dimension.
  - Also, we want the dimensions to be independent.
  - If data has high covariance when represented in some *n* number of dimensions then we replace those dimensions with *linear combination* of those n dimensions. Now that data will only be dependent on linear combination of those related n dimensions. *(related = have high covariance)*

What does Principal Component Analysis (PCA) do?

PCA finds a new set of dimensions (or a set of basis of views) such that all the dimensions are orthogonal (and hence linearly independent) and ranked according to the variance of data along them. It means more important principle axis occurs first. (more important = more variance/more spread out data)

Principal Component Analysis (PCA)



- **Component 1** This is the direction in the data that contains most of the information, or in other words, the direction along which the features are most correlated with each other.
- Then, the algorithm finds the direction that contains the most information while being orthogonal (is at a right angle) to the first direction.
- In two dimensions, there is only one possible orientation that is at a right angle, but in higher dimensional spaces there would be (infinitely) many orthogonal directions.

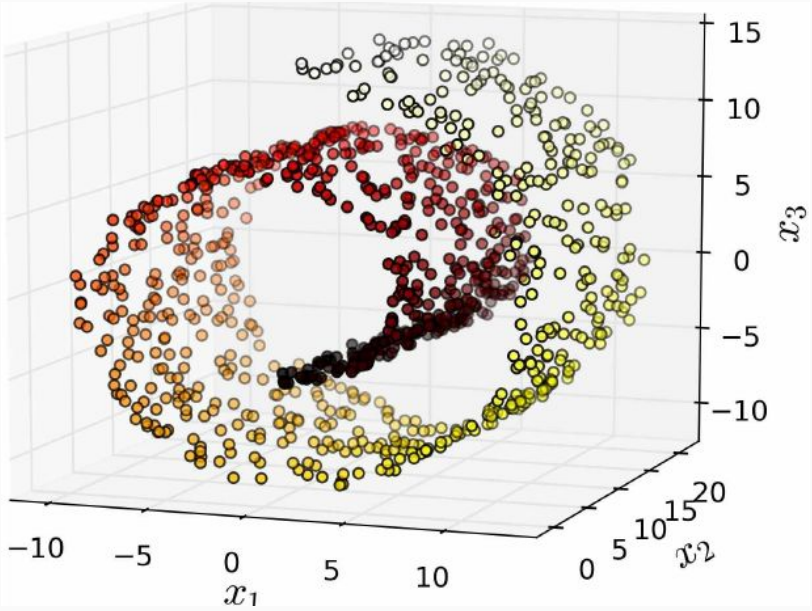
PCA

- The directions found using this process are called principal components, as they are the main directions of variance in the data.
- In general, there are as many principal components as original features.c

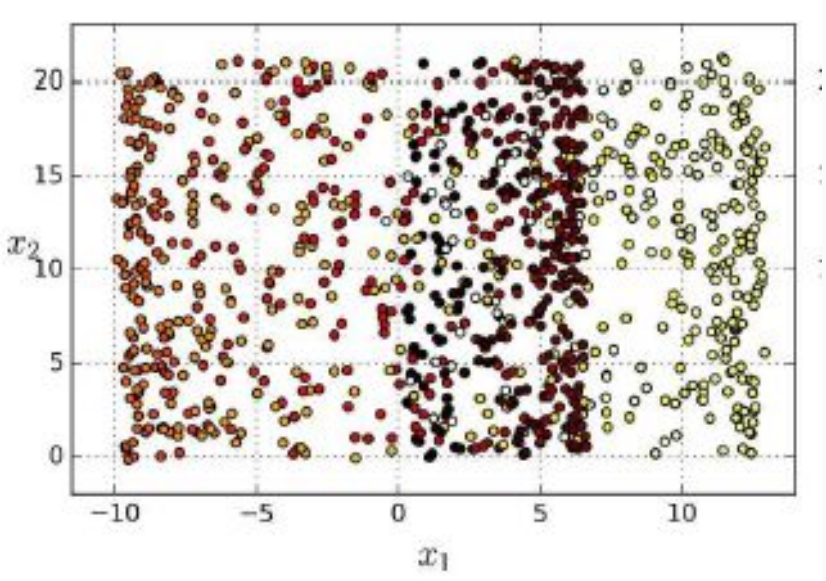
http://setosa.io/ev/principal-component-analysis/

Is PCA always a good preprocessing step?

Swiss Roll dataset



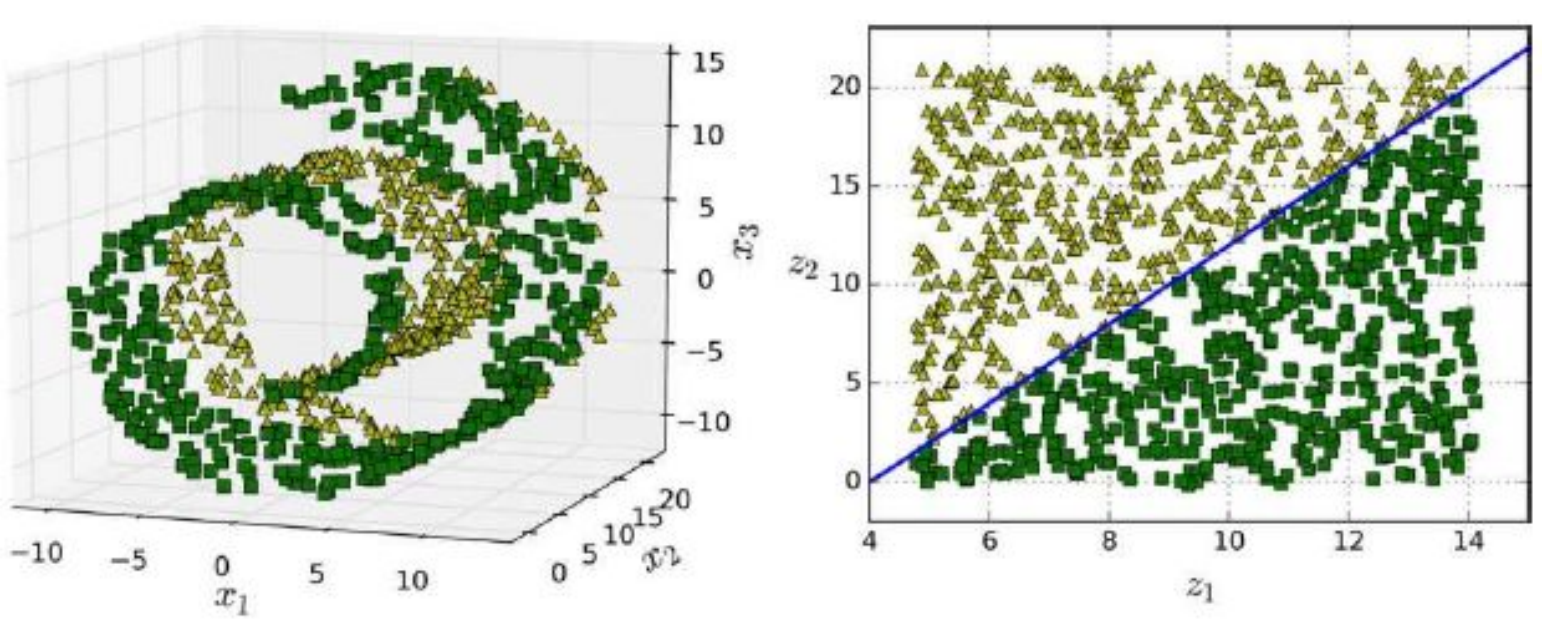
PCA



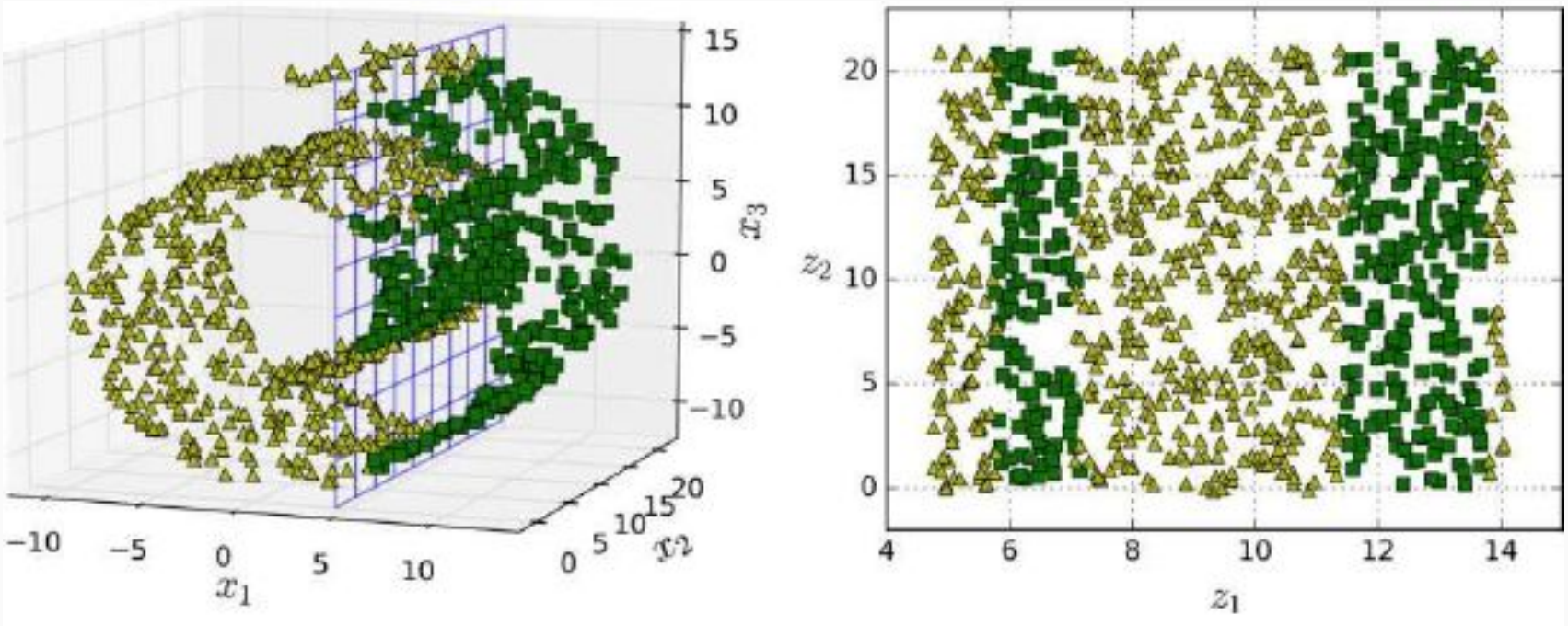
Manifold Learning

- The Swiss roll is an example of a 2D manifold.
- 2D manifold is a 2D shape that can be bent and twisted in a higher-dimensional space.
- More generally, a d-dimensional manifold is a part of an n-dimensional space (where d < n) that locally resembles a d-dimensional hyperplane.
- Swiss roll, d = 2 and n = 3: it locally resembles a 2D plane, but it is rolled in the third dimension.
- Many dimensionality reduction algorithms work by modeling the manifold on which the training instances lie; this is called **Manifold Learning**
- It relies on the manifold assumption, also called the manifold hypothesis, which holds that **most real-world high-dimensional datasets lie close to a much lower dimensional manifold.** This assumption is very often empirically observed.

- The manifold assumption is often accompanied by another implicit assumption: that the task at hand (e.g. classification or regression) will be simpler if expressed in the lower-dimensional space of the manifold.



- Not always true.



Kernel PCA

- Remember, kernel trick, a mathematical technique that implicitly maps instances into a very high-dimensional space (called the feature space), enabling nonlinear classification and regression with Support Vector Machines.
- Linear decision boundary in the high-dimensional feature space corresponds to a complex nonlinear decision boundary in the original space.
- It turns out that the same trick can be applied to PCA, making it possible to perform complex nonlinear projections for dimensionality reduction. This is called **Kernel PCA (kPCA).**
- It is often good at preserving clusters of instances after projection, or sometimes even unrolling datasets that lie close to a twisted manifold.

- Below figure shows the Swiss roll, reduced to two dimensions using a linear kernel (equivalent to simply using the PCA class), an RBF kernel, and a sigmoid kernel (Logistic).

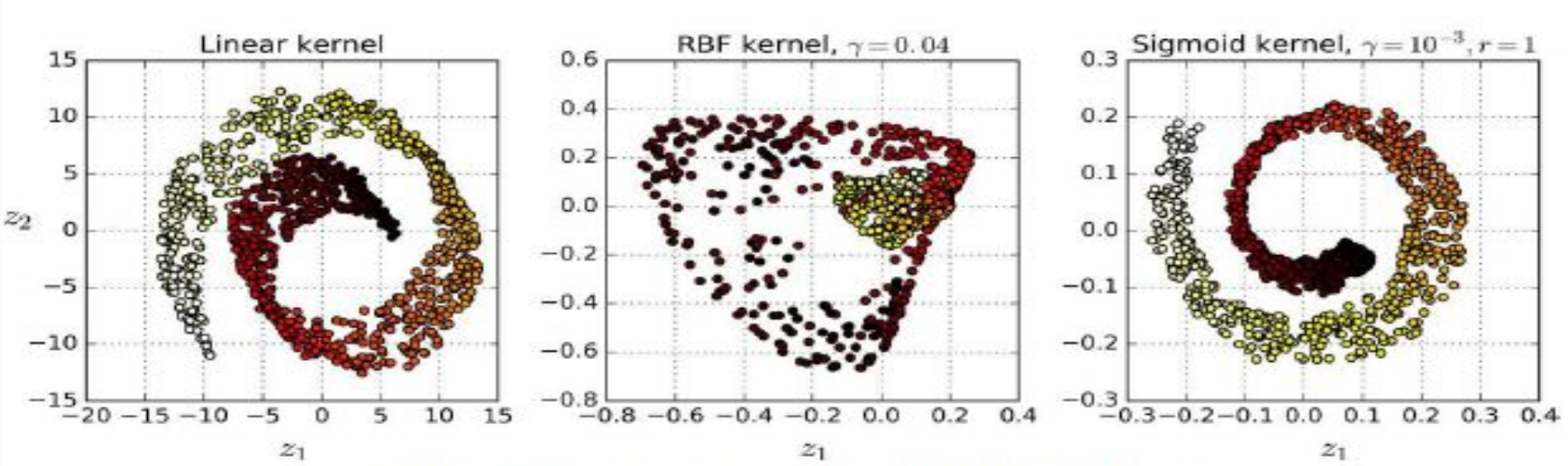
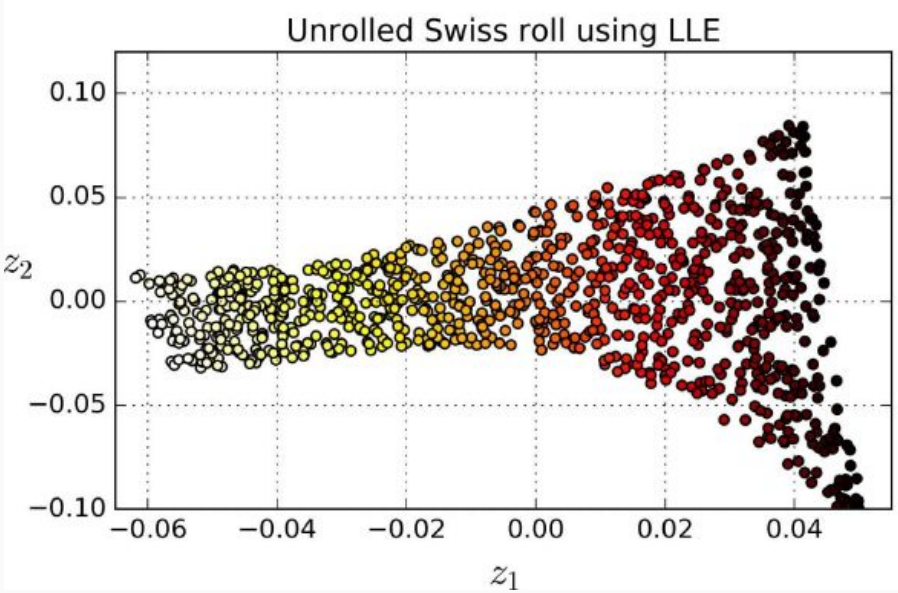


Figure 8-10. Swiss roll reduced to 2D using kPCA with various kernels

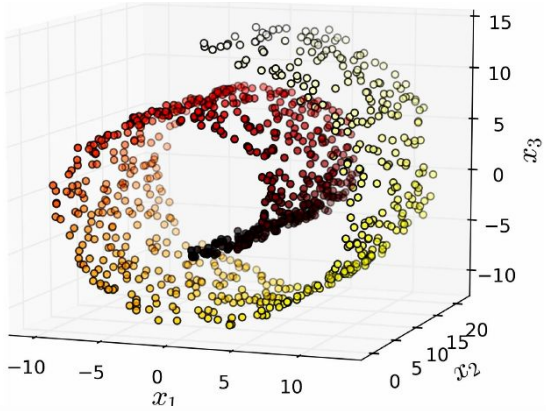
Locally Linear Embedding

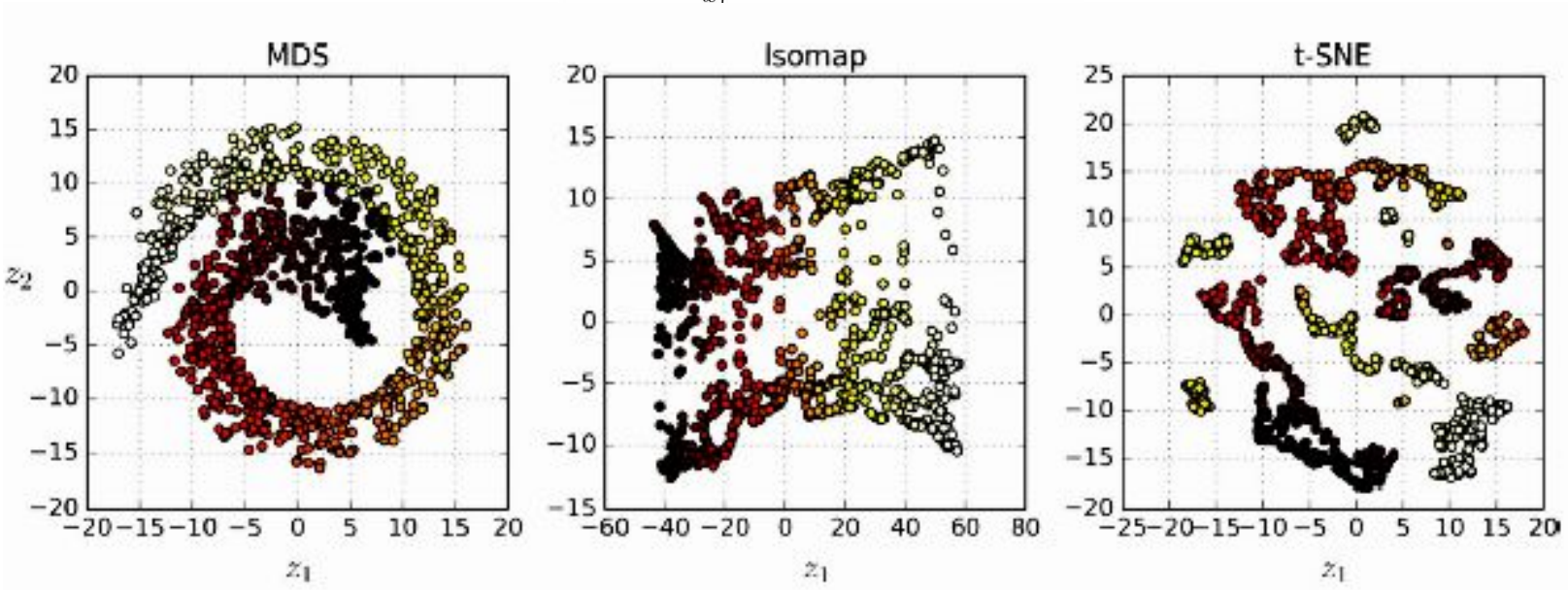
- Locally Linear Embedding (LLE) is another very powerful nonlinear dimensionality reduction (NLDR) technique.
- It is a Manifold Learning technique that does not rely on projections like the previous algorithms.
- In a nutshell, LLE works by first measuring how each training instance linearly relates to its closest neighbors, and then looking for a low-dimensional representation of the training set where these local relationships are best preserved.
- This makes it particularly good at unrolling twisted manifolds, especially when there is not too much noise.



Other Dimensionality Reduction Techniques

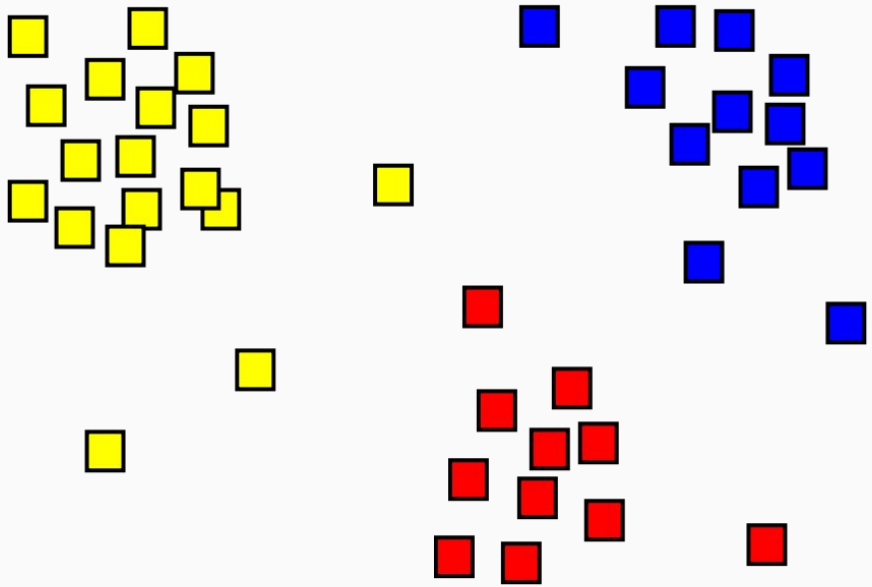
- **Multidimensional Scaling (MDS)** reduces dimensionality while trying to preserve the distances between the instances
- **Isomap** creates a graph by connecting each instance to its nearest neighbors, then reduces dimensionality while trying to preserve the geodesic distances between the instances.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE)** reduces dimensionality while trying to keep similar instances close and dissimilar instances apart. It is mostly used for visualization, in particular to visualize clusters of instances in high-dimensional space.
- **Linear Discriminant Analysis (LDA)** is actually a classification algorithm, but during training it learns the most discriminative axes between the classes, and these axes can then be used to define a hyperplane onto which to project the data. The benefit is that the projection will keep classes as far apart as possible, so LDA is a good technique to reduce dimensionality before running another classification algorithm such as an SVM classifier.


15
10
5
0
−5
−10
x_3
−10
−5
0
5
10
x_1
0
5
10
15
20
x_2


MDS
Isomap
t-SNE
z_2
z_1
z_1
z_1

CLUSTERING

UNSUPERVISED LEARNING - 02

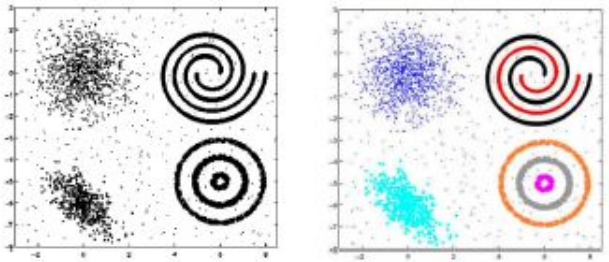


Clustering

Usually an unsupervised learning problem

Given: N unlabeled examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; no. of desired partitions K

Goal: Group the examples into K "homogeneous" partitions



(a) Input data (b) Desired clustering

Picture courtesy: "Data Clustering: 50 Years Beyond K-Means", A.K. Jain (2008)

Loosely speaking, it is classification without ground truth labels

A good clustering is one that achieves:

- High within-cluster similarity
- Low inter-cluster similarity

Clustering: Some Examples

Document/Image/Webpage Clustering

Image Segmentation (clustering pixels)

Clustering web-search results

Clustering (people) nodes in (social) networks/graphs

.. and many more..

Aspects of Clustering

- A clustering algorithm such as
  - Partitional Clustering e.g k-Means
  - Hierarchical Clustering e.g. AHC
  - Mixture of Gaussians
- A distance or similarity function
  - Such as euclidean, Minkowski, cosine
- Clustering Quality
  - Inter-cluster distance → Maximized
  - Intra-cluster distance → Minimized

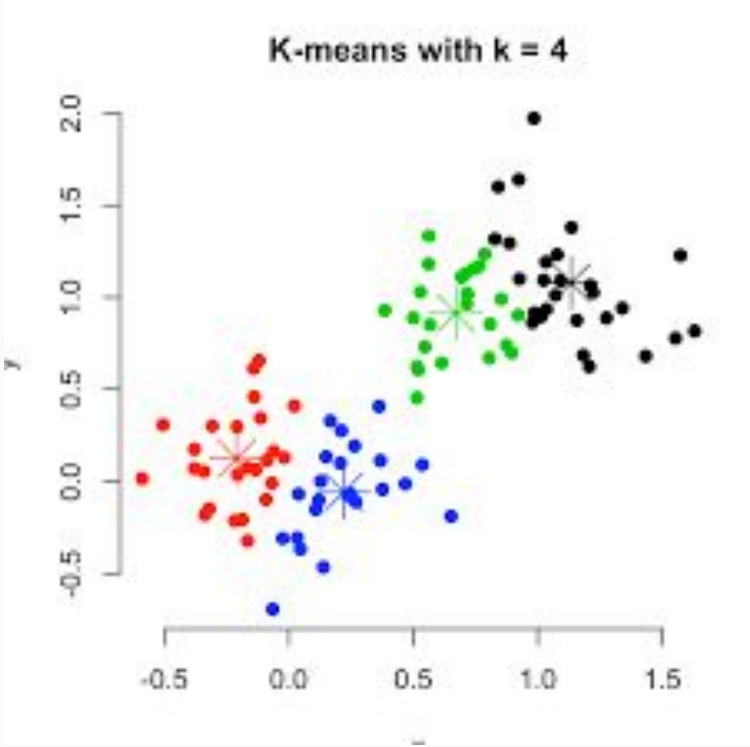
Quality of clustering result depends upon the algorithm, distance function and the application

Major Clustering Approaches

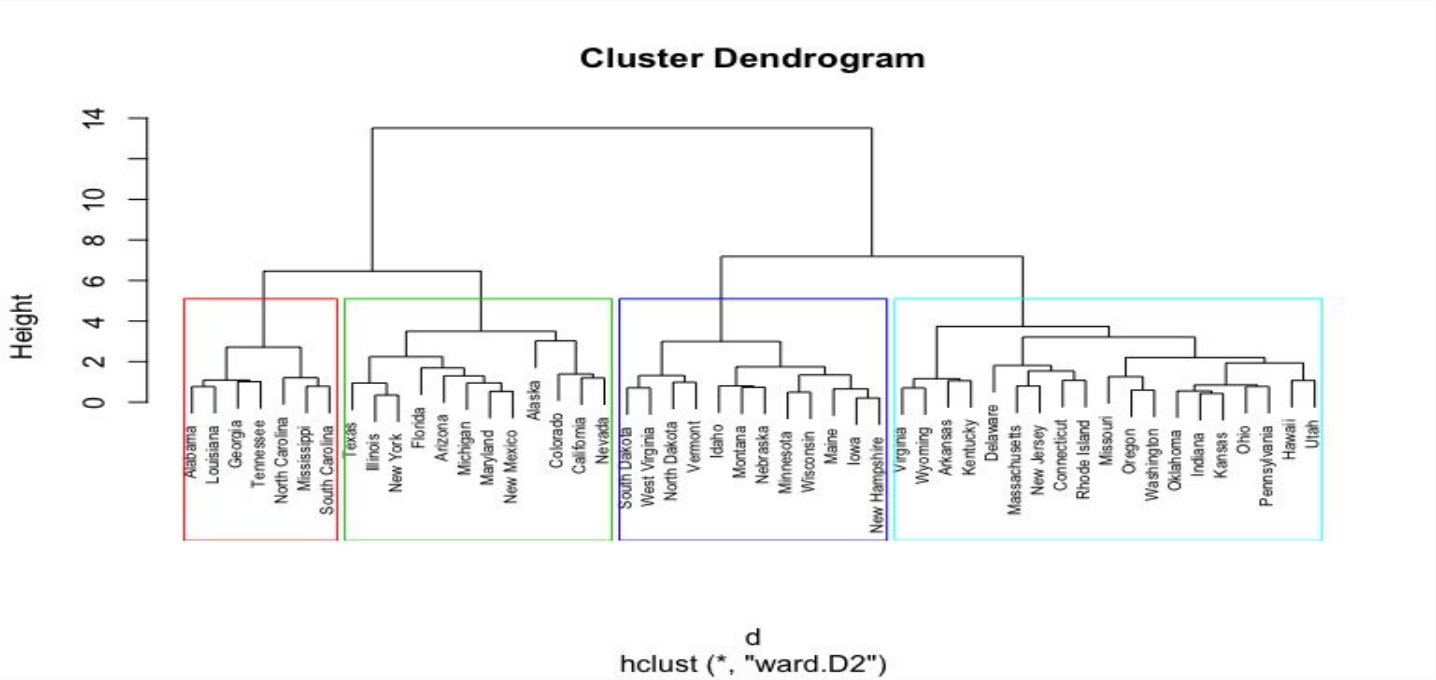
- **Partitioning:** Construct various partitions and then evaluate them by some criterion.
- **Hierarchical:** Create a hierarchical decomposition of the set of objects using some criterion.
- **Model-Based:** Hypothesize a model for each cluster and find best fit of models to data.
- **Density Based:** Guided by connectivity and density functions.
- **Graph Theoretic** Clustering

Partitioning Algorithm

- **Partitioning:** Construct a partition of a database D of m objects into a set of k clusters.
- Given a k, find a partition of k clusters that optimizes the chosen partitioning criterion.
  - Global Optimal : exhaustively enumerate all partitions
  - Heuristic Method : k-Means (MacQueen, 1967)



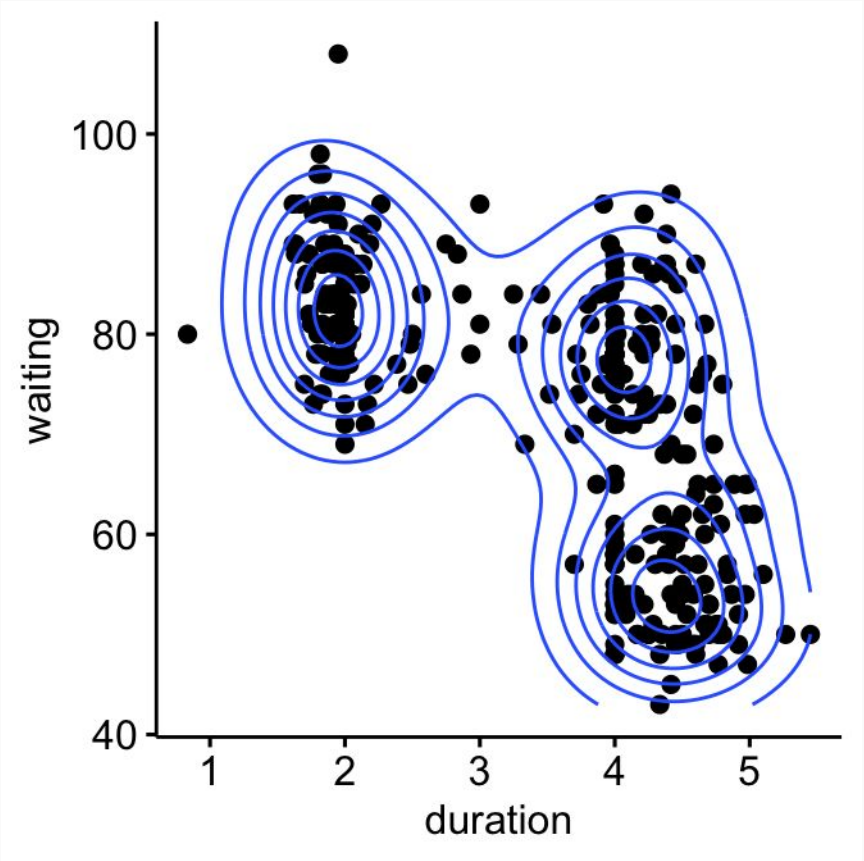
Hierarchical Clustering



- Produces a nested sequence of clusters.
- One approach: recursive application of a partition algorithm.

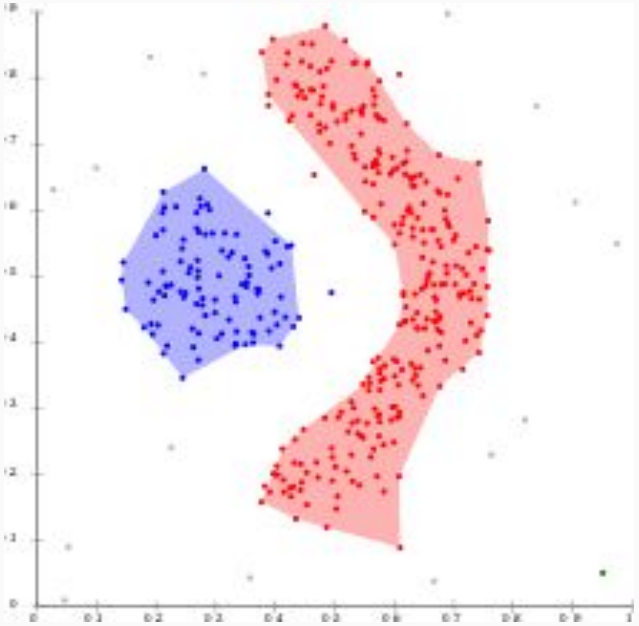
Model Based Clustering

- A model is hypothesized
- Assume data generated by a mixture of underlying probability distributions
- Fit the data to model



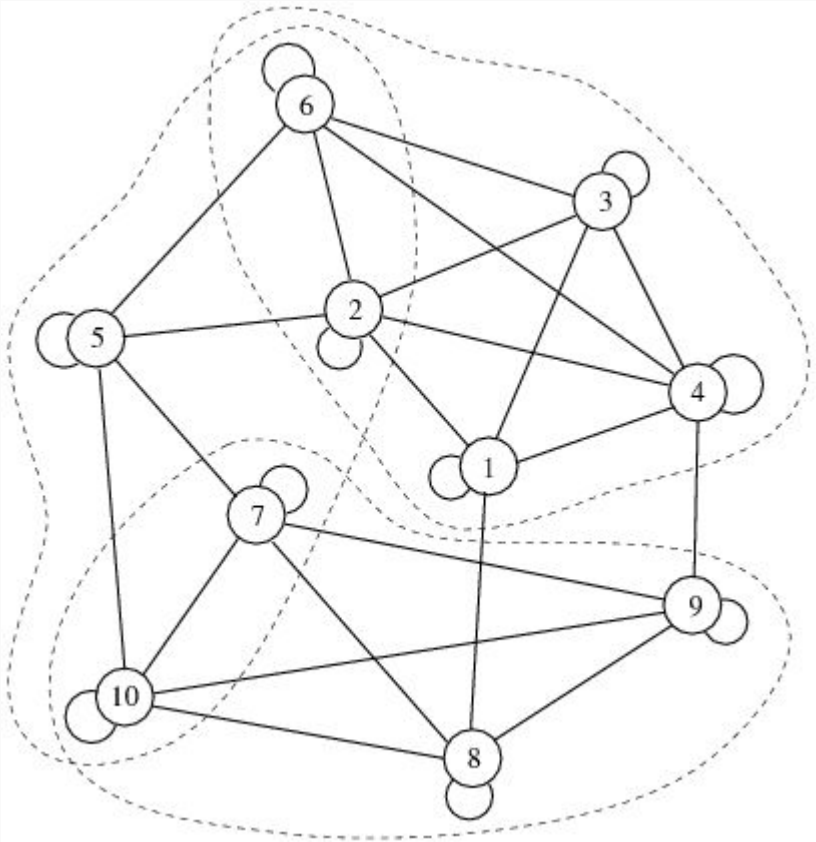
Density Based Clustering

- Based on density of connected points
- Locates regions of high density separated by regions of low density
- E.g. DBSCAN



Graph Theoretic Clustering

- Weight of edges between items based on similarity
- E.g. look for minimum-cut in a graph



Dis(similarity) Measure

(Dis)similarity measures

- Distance metric (scale-dependent)
  - Minkowski family of distance measures

$$d(x_i, x_j) = \left(\sum_{s=1}^{m} |x_{is} - x_{js}|^p \right)^{1/p}$$

Manhattan (p=1), Euclidean (p=2)

  - Cosine distance

$$\text{cosine}(x_i, x_j) = \frac{x_i . x_j}{\|x_i\| . \|x_i\|}$$

Dis(similarity) Measure

(Dis)similarity measures

- Correlation coefficients (scale-invariant)
- Mahalanobis distance

$$d(x_i, x_i) = \sqrt{(x_i - x_j)\Sigma^{-1}(x_i - x_j)}$$

- Pearson correlation

$$r(x_i, x_j) = \frac{Cov(x_i, x_j)}{\sigma_{x_i}\sigma_{x_j}}$$

Quality of Clustering

- Internal evaluation:
  - assign the best score to the algorithm that produces clusters with high similarity within a cluster and low similarity between clusters, e.g., Davies-Bouldin index

$$DB = \frac{1}{n}\sum_{i=1}^{k} \max_{j \neq i} \frac{\sigma_i + \sigma_j}{d(c_i, c_j)}$$

- External evaluation:
  - evaluated based on data such as known class labels and external benchmarks, eg, Rand Index, Jaccard Index, f-measure

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP + FP + FN}$$

K-Means Clustering

Input: N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; $\boldsymbol{x}_n \in \mathbb{R}^D$; the number of partitions K

Initialize: K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, each $\boldsymbol{\mu}_k \in \mathbb{R}^D$

- Usually initialized randomly, but good initialization is crucial; many smarter initialization heuristics exist (e.g., K-means++, Arthur & Vassilvitskii, 2007)

Iterate:

- (Re)-Assign each example $\boldsymbol{x}_n$ to its closest cluster center (based on the smallest Euclidean distance)

$$\mathcal{C}_k = \{n : \quad k = \arg\min_k ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2\}$$

  ($\mathcal{C}_k$ is the set of examples assigned to cluster k with center $\boldsymbol{\mu}_k$)

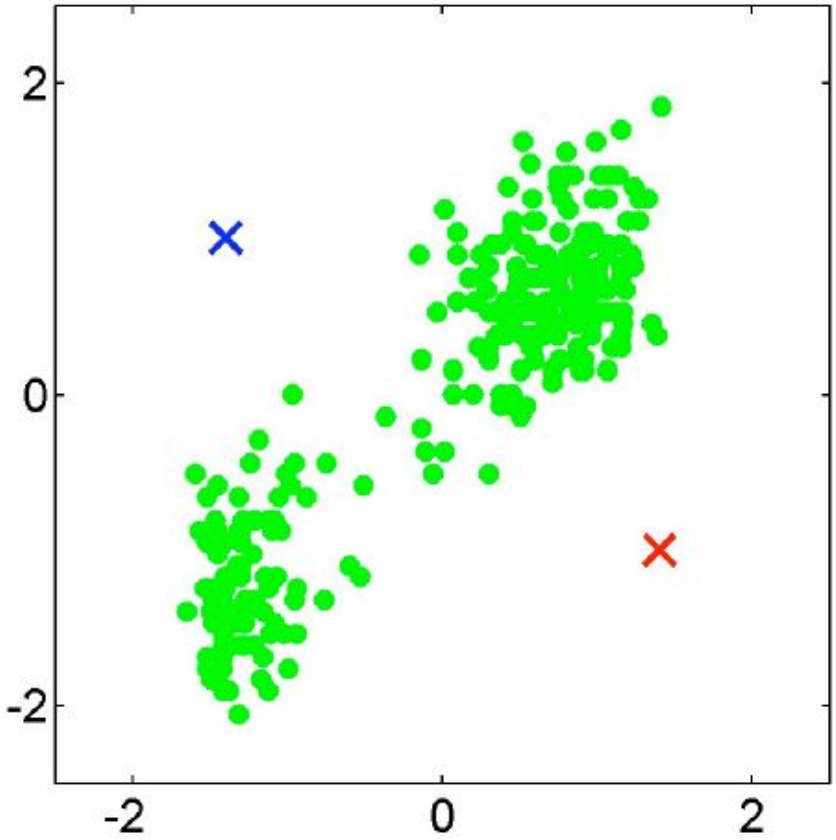
- Update the cluster means

$$\boldsymbol{\mu}_k = \text{mean}(\mathcal{C}_k) = \frac{1}{|\mathcal{C}_k|} \sum_{n \in \mathcal{C}_k} \boldsymbol{x}_n$$

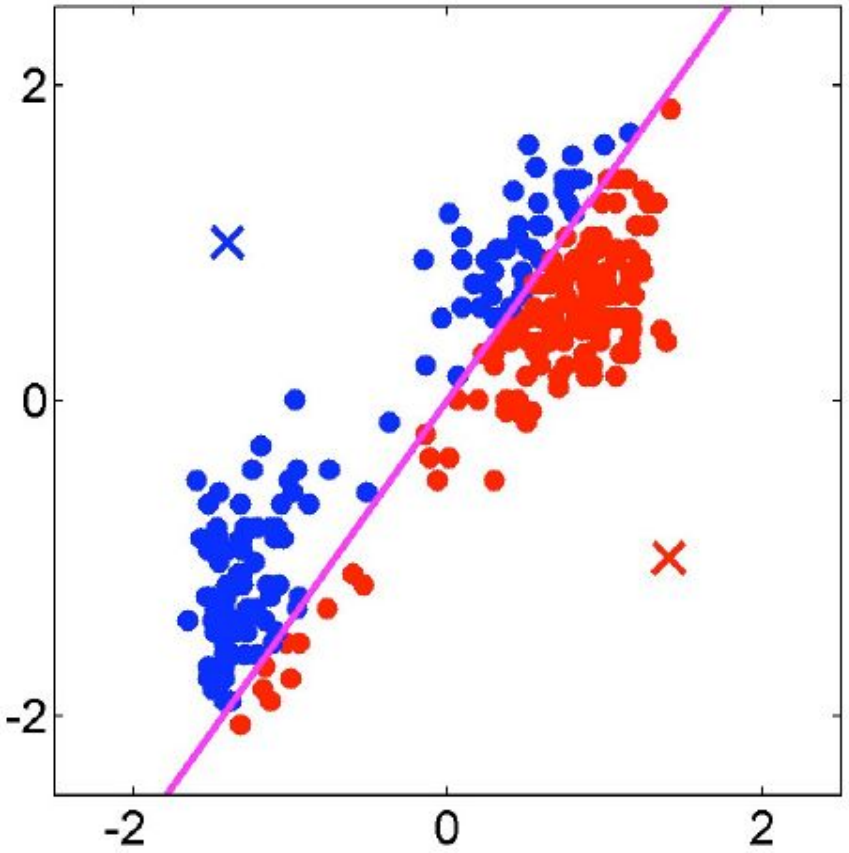
- Repeat while not converged

Stop when cluster means or the "loss" does not change by much

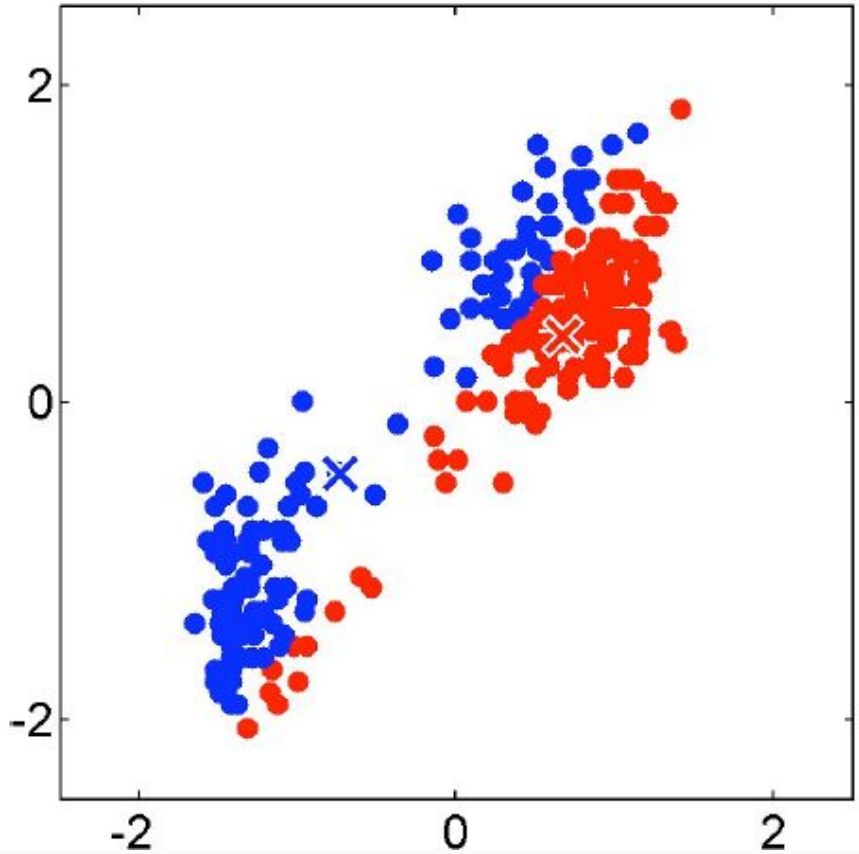
K-means: Initialization (assume K = 2)



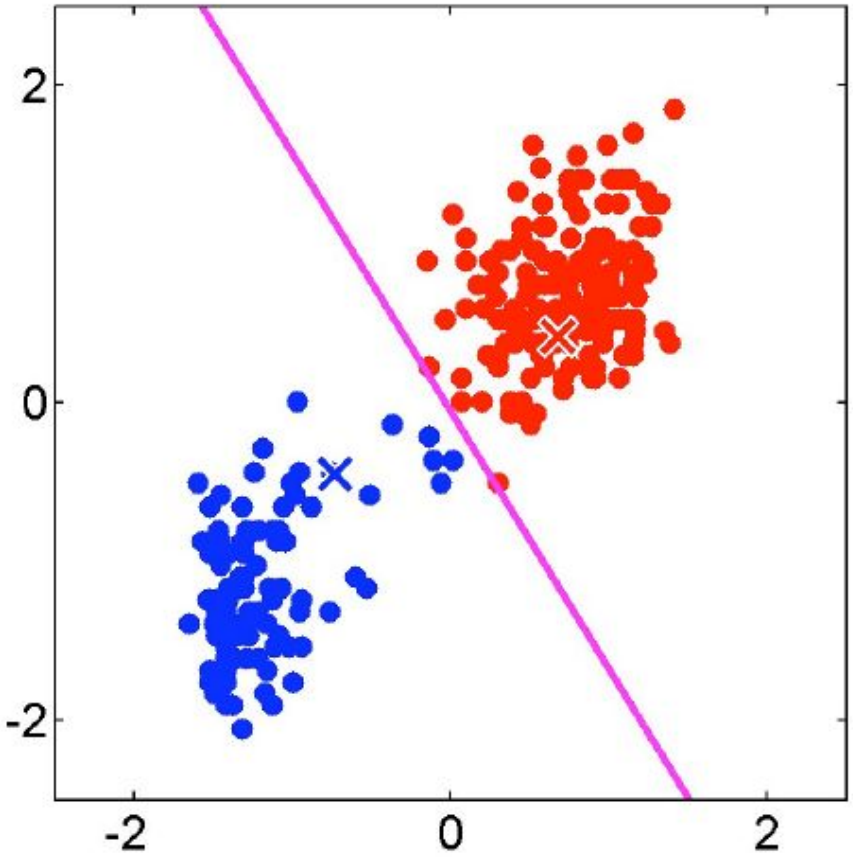
K-means iteration 1: Assigning points



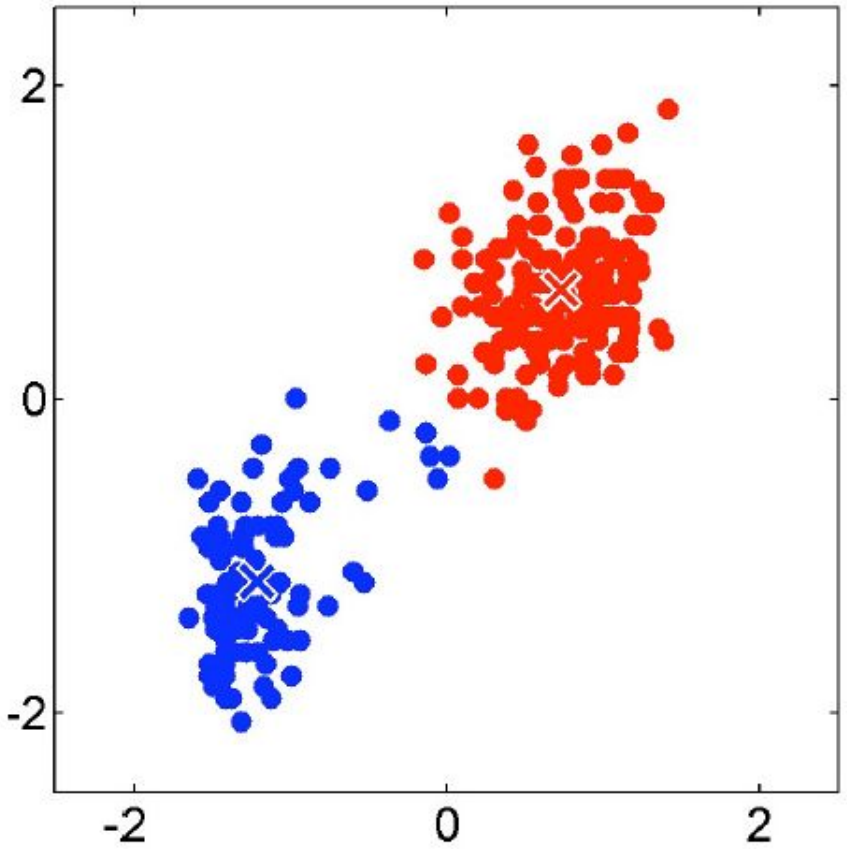
K-means iteration 1: Recomputing the centers



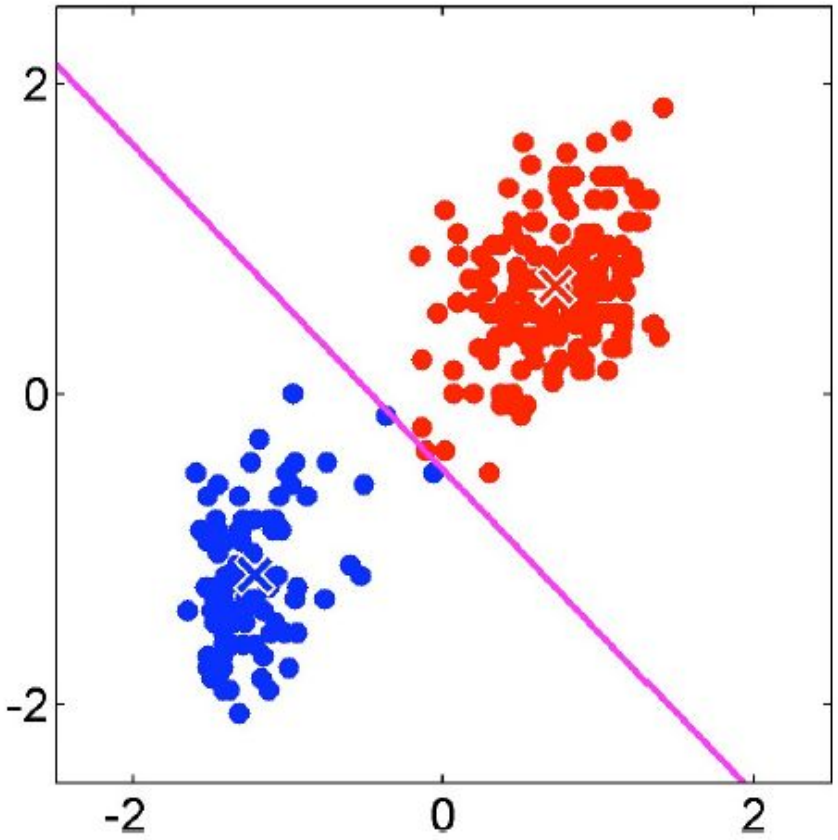
K-means iteration 2: Assigning points



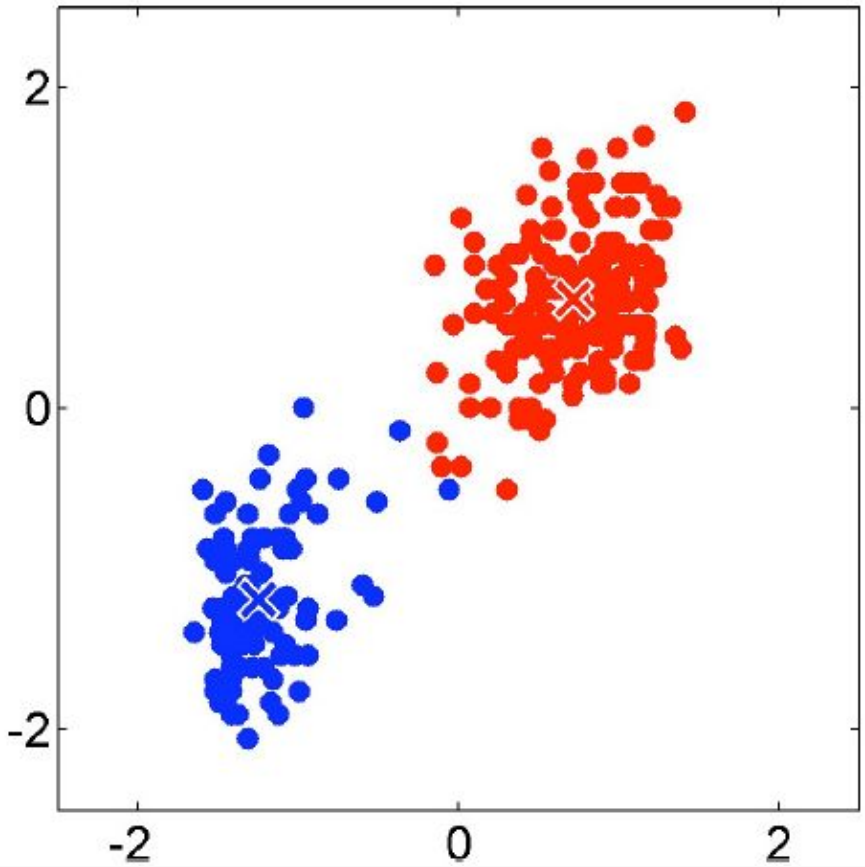
K-means iteration 2: Recomputing the centers



K-means iteration 3: Assigning points



K-means iteration 3: Recomputing the centers



What Loss Function is K-means Optimizing?

Let $\mu_1, \ldots, \mu_K$ be the K cluster centroids (means)

Let $z_{nk} \in \{0, 1\}$ be s.t. $z_{nk} = 1$ if $\boldsymbol{x}_n$ belongs to cluster k, and 0 otherwise

- Note: $\boldsymbol{z}_n = [z_{n1}\ z_{n2}\ \ldots\ z_{nK}]$ represents a length K one-hot encoding of $\boldsymbol{x}_n$

Define the distortion or "loss" for the cluster assignment of $\boldsymbol{x}_n$

$$\ell(\boldsymbol{\mu}, \boldsymbol{x}_n, \boldsymbol{z}_n) = \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

Total distortion over all points defines the K-means "loss function"

$$L(\boldsymbol{\mu}, \mathbf{X}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 = ||\mathbf{X} - \mathbf{Z}\boldsymbol{\mu}||^2$$

where $\mathbf{Z}$ is $N \times K$ (row n is $\boldsymbol{z}_n$) and $\boldsymbol{\mu}$ is $K \times D$ (row k is $\boldsymbol{\mu}_k$)

The K-means **problem** is to minimize this objective w.r.t. $\boldsymbol{\mu}$ and $\mathbf{Z}$

- Note that the objective only minimizes within-cluster distortions

K-means Objective

Consider the K-means objective function

$$L(\boldsymbol{\mu}, \mathbf{X}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \mu_k||^2$$

It is a non-convex objective function

- Many local minima possible

Also NP-hard to minimize in general (note that $\mathbf{Z}$ is discrete)

The K-means algorithm we saw is a heuristic to optimize this function

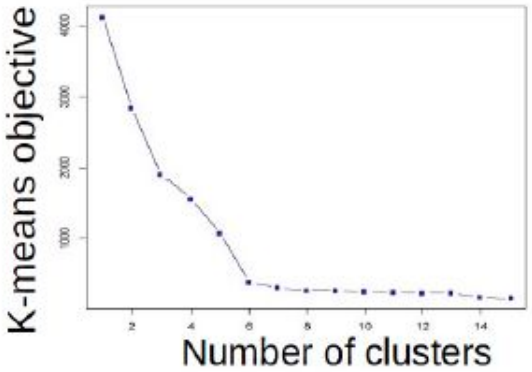
K-means algorithm alternated between the following two steps

- Fix $\boldsymbol{\mu}$, minimize w.r.t. $\mathbf{Z}$ (assign points to closest centers)
- Fix $\mathbf{Z}$, minimize w.r.t. $\boldsymbol{\mu}$ (recompute the center means)

Note: The algorithm usually converges to a local minima (though may not always, and it may jus convergence "somewhere"). Multiple runs with different initializations can be tried to find a good solution.

K-means: Choosing K

One way to select K for the K-means algorithm is to try different values of K, plot the K-means objective versus K, and look at the "elbow-point"



For the above plot, $K = 6$ is the elbow point

Can also information criterion such as AIC (Akaike Information Criterion)

$$AIC = 2L(\hat{\mu}, \mathbf{X}, \hat{\mathbf{Z}}) + K \log D$$

.. and choose the K that has the smallest AIC (discourages large K)

K-means: Some Limitations

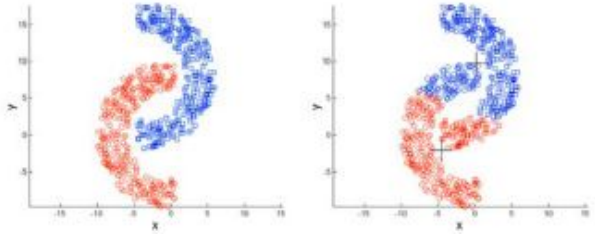
Makes hard assignments of points to clusters

- A point either completely belongs to a cluster or doesn't belong at all
- No notion of a soft assignment (i.e., probability of being assigned to each cluster: say $K = 3$ and for some point $\boldsymbol{x}_n$, $p_1 = 0.7, p_2 = 0.2, p_3 = 0.1$)

Works well only is the clusters are roughly of equal sizes

Probabilistic clustering methods such as Gaussian mixture models can handle both these issues (model each cluster using a Gaussian distribution)

K-means also works well only when the clusters are round-shaped and does badly if the clusters have non-convex shapes



Kernel K-means or Spectral clustering can handle non-convex