Lecture 5: Unsupervised Learning

Olexandr Isayev

Department of Chemistry, CMU

olexandr@cmu.edu

Plan for the course

Next week: Guest lecture about probabilistic methods

HW2: Clustering

Select your class final project problems:

- Upload title and one paragraph summary
- I encourage you to use your domain-specific dataset
- If not, I am happy to give you one

Machine Learning

- **Supervised:** We are given input samples (X) and output samples (y) of a function y = f(X). We would like to "learn" f, and evaluate it on new data. Types:
  - **Classification:** y is discrete (class labels).
  - **Regression:** y is continuous, e.g. linear regression.

- **Unsupervised:** Given only samples X of the data, we compute a function f such that y = f(X) is "simpler".
  - **Clustering:** y is discrete
  - Y is continuous: **Matrix factorization, Kalman filtering, unsupervised neural networks**.

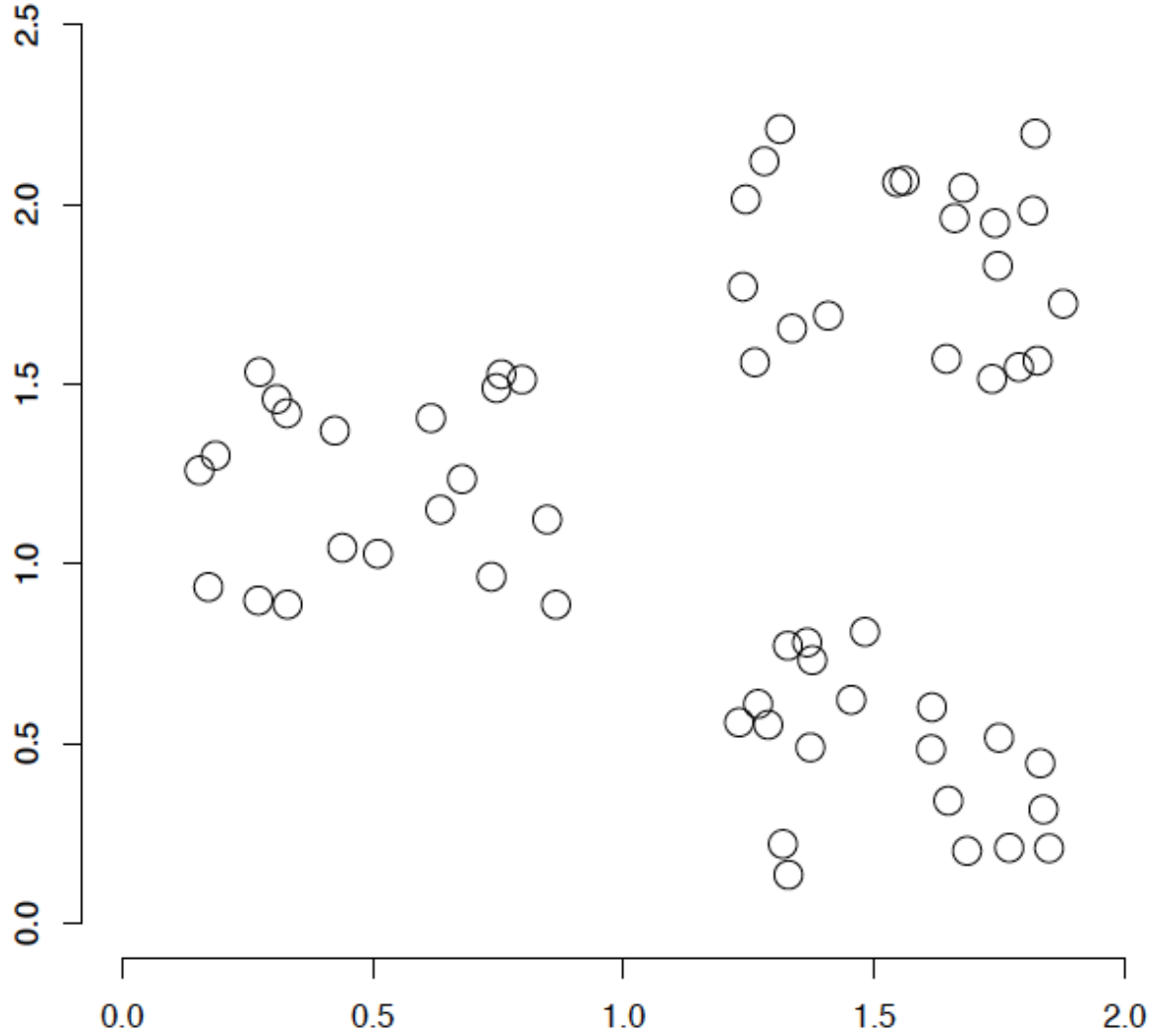
Supervised learning vs. unsupervised learning

- Supervised learning: discover patterns in the data that relate data attributes with a target (class) attribute.
  - These patterns are then utilized to predict the values of the target attribute in future data instances.
- Unsupervised learning: The data have no target attribute.
  - We want to explore the data to find some intrinsic structures in them.

Clustering

- Clustering is a technique for finding similarity groups in data, called **clusters**. I.e.,
  - it groups data instances that are similar to (near) each other in one cluster and data instances that are very different (far away) from each other into different clusters.
- Clustering is often called an **unsupervised learning** task as no class values denoting an *a priori* grouping of the data instances are given, which is the case in supervised learning.
- Due to historical reasons, clustering is often considered synonymous with unsupervised learning.
  - In fact, association rule mining is also unsupervised
- This lecture focuses on clustering.

An illustration



The data set has three natural groups of data points, i.e., 3 natural clusters.

How would you design an algorithm for finding the three clusters in this case?

Classification vs. Clustering

Classification: supervised learning

Clustering: unsupervised learning

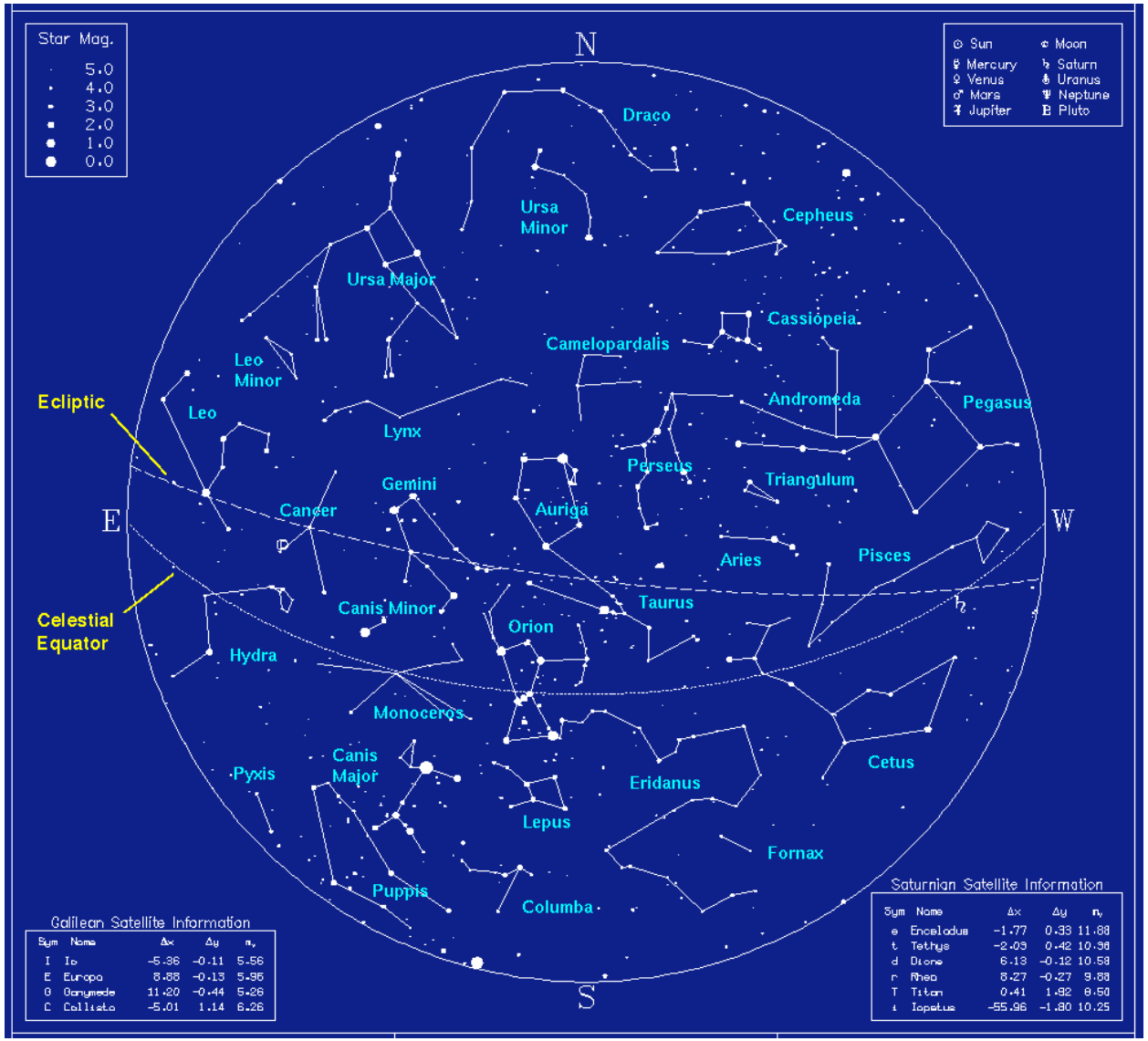
Classification: Classes are human-defined and part of the input to the learning algorithm.

Clustering: Clusters are inferred from the data without human input.

> However, there are many ways of influencing the outcome of clustering: number of clusters, similarity measure, representation of molecules, . . .

7

Cluster Bias



Aspects of clustering

- A clustering algorithm
  - Partitional clustering
  - Hierarchical clustering
  - …
- A distance (similarity, or dissimilarity) function
- Clustering quality
  - Inter-clusters distance $\Rightarrow$ maximized
  - Intra-clusters distance $\Rightarrow$ minimized
- The quality of a clustering result depends on the algorithm, the distance function, and the application.

Issues for clustering

Representation for clustering

- Data representation
  - Vector space? Normalization?
- Need a notion of similarity/distance

How many clusters?

- Fixed a priori?
- Completely data driven?
  - Avoid “trivial” clusters - too large or small
    - If a cluster's too large, then for navigation purposes you've wasted an extra time

What is Similarity?

The quality or state of being similar; likeness; resemblance; as, a similarity of features.

Webster's Dictionary

Similarity is hard to define, but…
"*We know it when we see it*"

The real meaning of similarity is a philosophical question. We will take a more pragmatic approach.

Notion of similarity/distance

- Depending on the type of data – different distance/metric is used
- Euclidean
- Cosine similarity, Manhattan Distance, Jaccard
- Tanimoto similarity

Intuitions behind desirable distance measure properties

$D(A,B) = D(B,A)$ *Symmetry*

Otherwise you could claim "Alex looks like Bob, but Bob looks nothing like Alex."

$D(A,A) = 0$ *Constancy of Self-Similarity*

Otherwise you could claim "Alex looks more like Bob, than Bob does."

$D(A,B) = 0$ If A=B *Positivity (Separation)*

Otherwise there are objects in your world that are different, but you cannot tell apart.

$D(A,B) \leq D(A,C) + D(B,C)$ *Triangular Inequality*

Otherwise you could claim "Alex is very like Bob, and Alex is very like Carl, but Bob is very unlike Carl."

Terminology

- **Hierarchical clustering:** clusters form a hierarchy. Can be computed bottom-up or top-down.
- **Flat clustering:** no inter-cluster structure.

- **Hard clustering:** items assigned to a unique cluster.
- **Soft clustering:** cluster membership is a real-valued function, distributed across several clusters.

Clustering Algorithms

- Flat algorithms
  - Usually start with a random (partial) partitioning
  - Refine it iteratively
    - *K* means clustering
    - (Model based clustering)
- Hierarchical algorithms
  - Bottom-up, agglomerative
  - (Top-down, divisive)

Hard vs. soft clustering

- Hard clustering: Each sample belongs to exactly one cluster
  - More common and easier to do

- Soft clustering: A sample can belong to more than one cluster.
  - Makes more sense for applications like creating browsable hierarchies
  - You may want to put a pair of sneakers in two clusters: (i) sports apparel and (ii) shoes
  - You can only do that with a soft clustering approach.

K-means

Each cluster in *K*-means is defined by a centroid.

Objective/partitioning criterion: minimize the average squared difference from the centroid

Recall definition of centroid:

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}$$

where we use ω to denote a cluster.

We try to find the minimum average squared difference by iterating two steps:

- reassignment: assign each vector to its closest centroid
- recomputation: recompute each centroid as the average of the vectors that were assigned to it in reassignment

K-means clustering

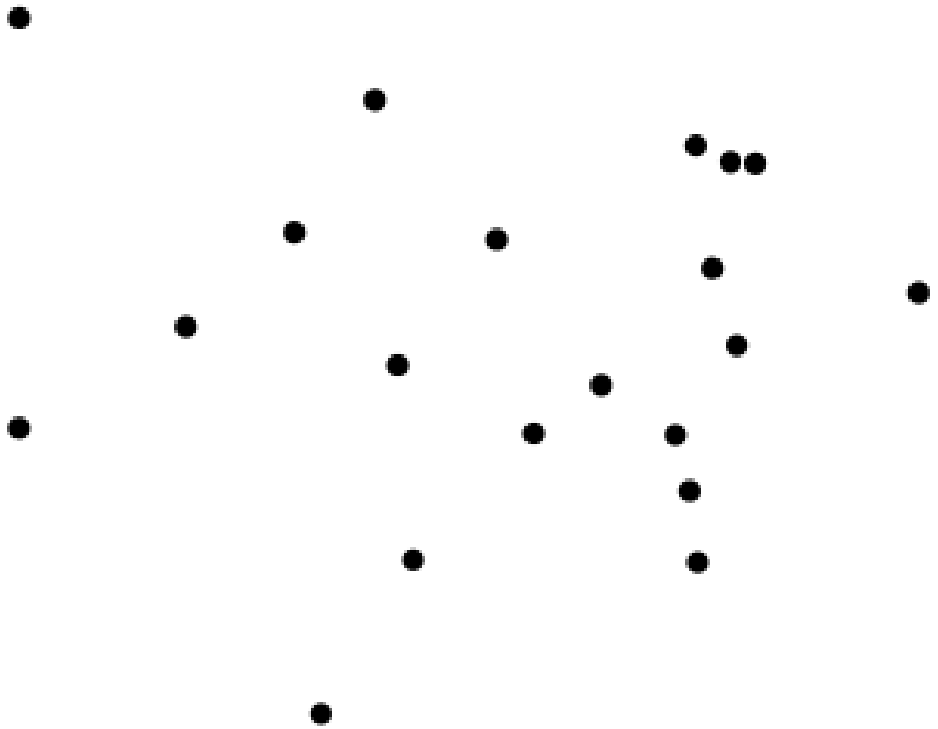
The standard k-means algorithm is based on **Euclidean distance**.

The cluster quality measure is an **intra-cluster measure only**, equivalent to the sum of item-to-centroid kernels.

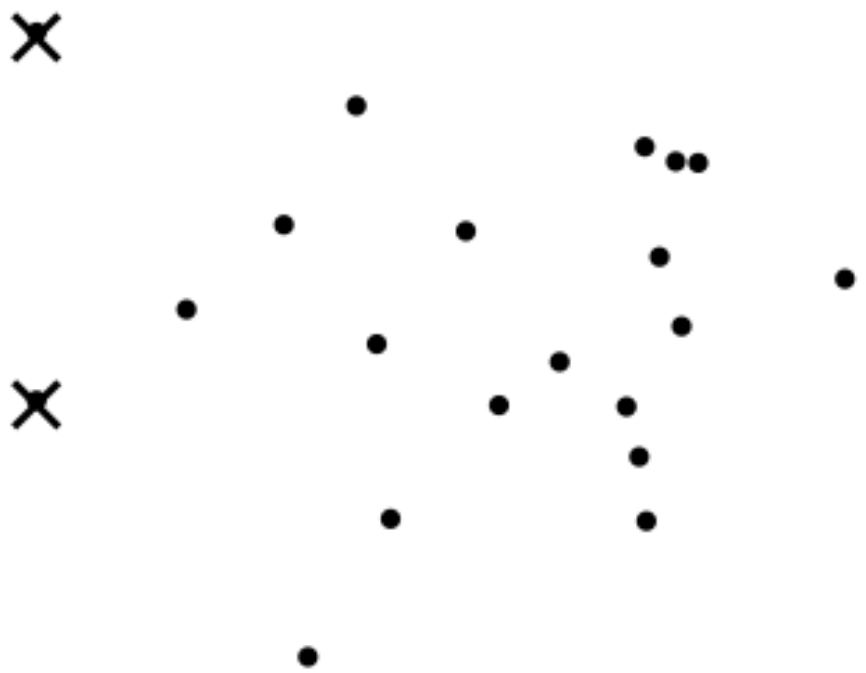
A simple greedy algorithm locally optimizes this measure (usually called Lloyd's algorithm):

- **Find the closest cluster center** for each item, and assign it to that cluster.
- **Recompute the cluster centroid** as the mean of items, for the newly-assigned items in the cluster.

Worked Example: Set of to be clustered



Worked Example: Random selection of initial centroids

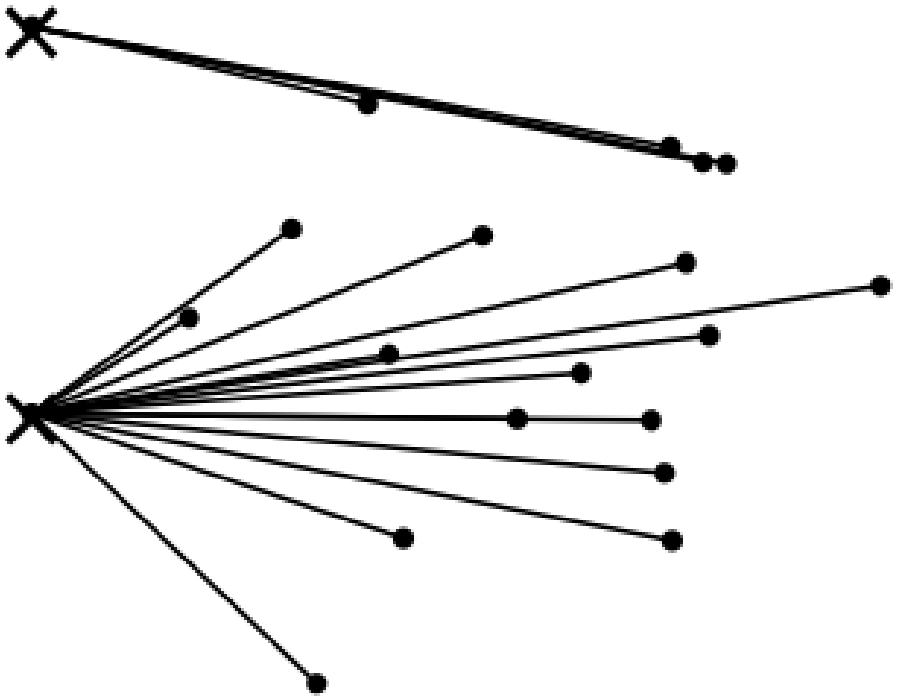


Exercise:

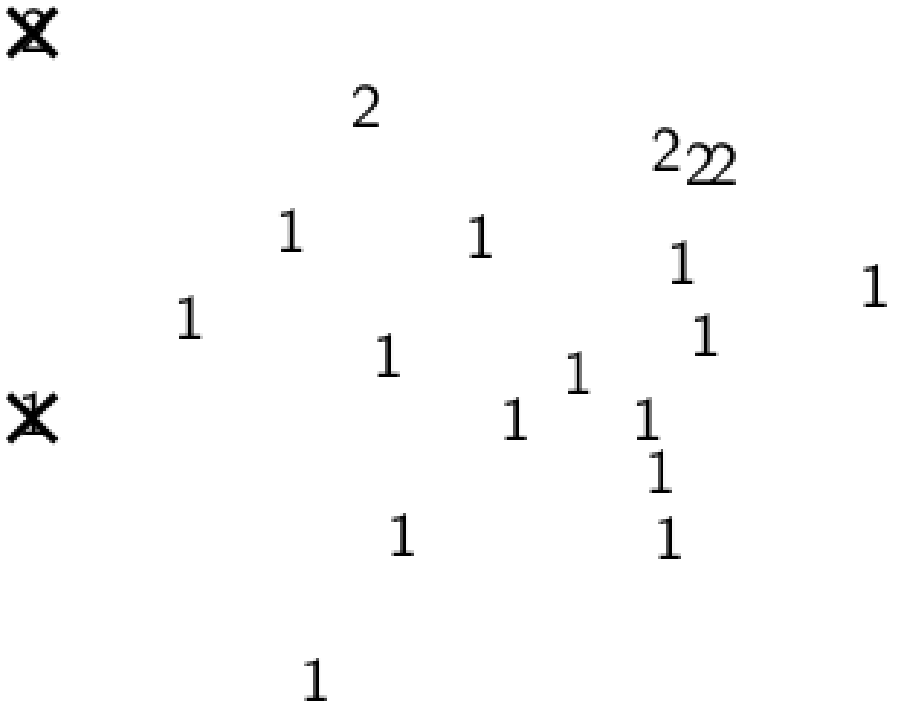
(i) Guess what the optimal clustering into two clusters is in this case;

(ii) compute the centroids of the clusters

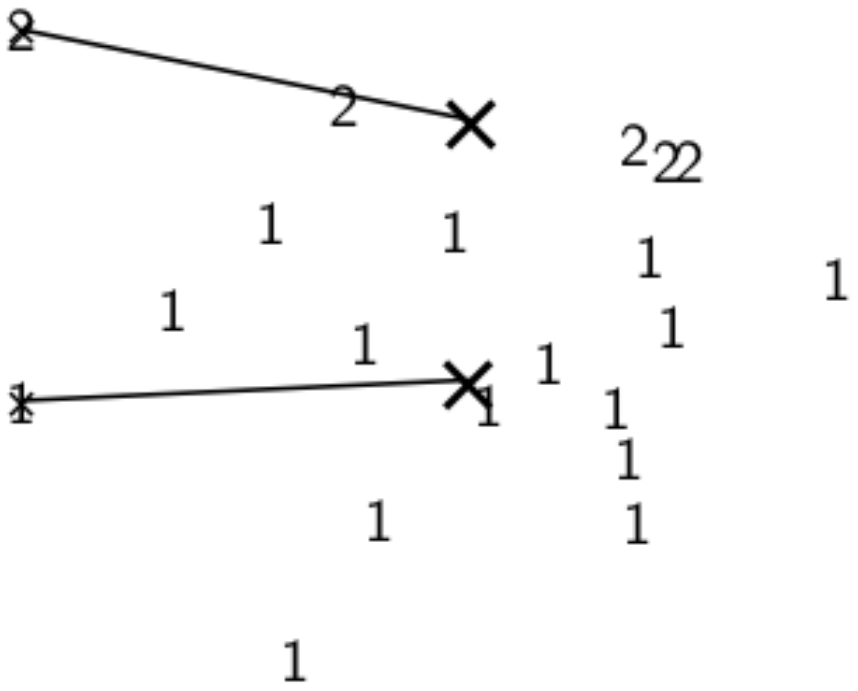
Worked Example: Assign points to closest center



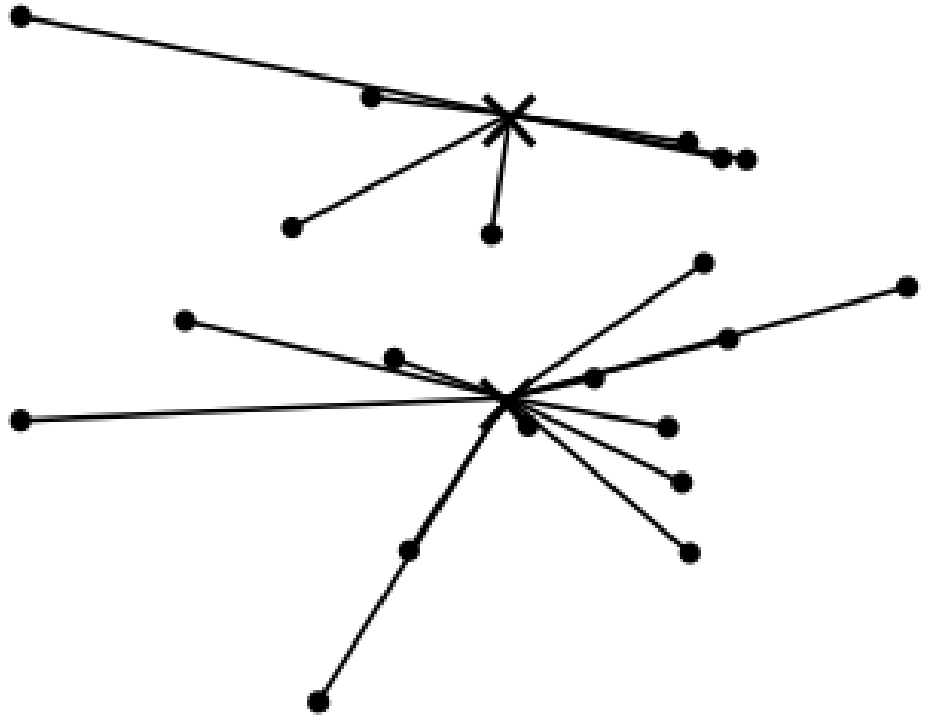
Worked Example: Assignment



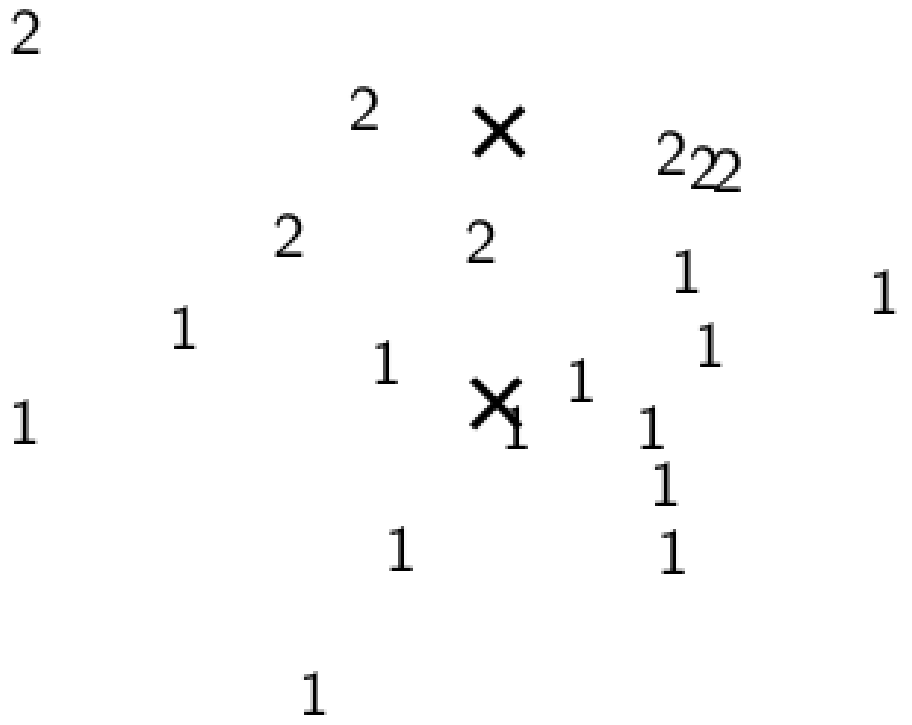
Worked Example: Recompute cluster centroids



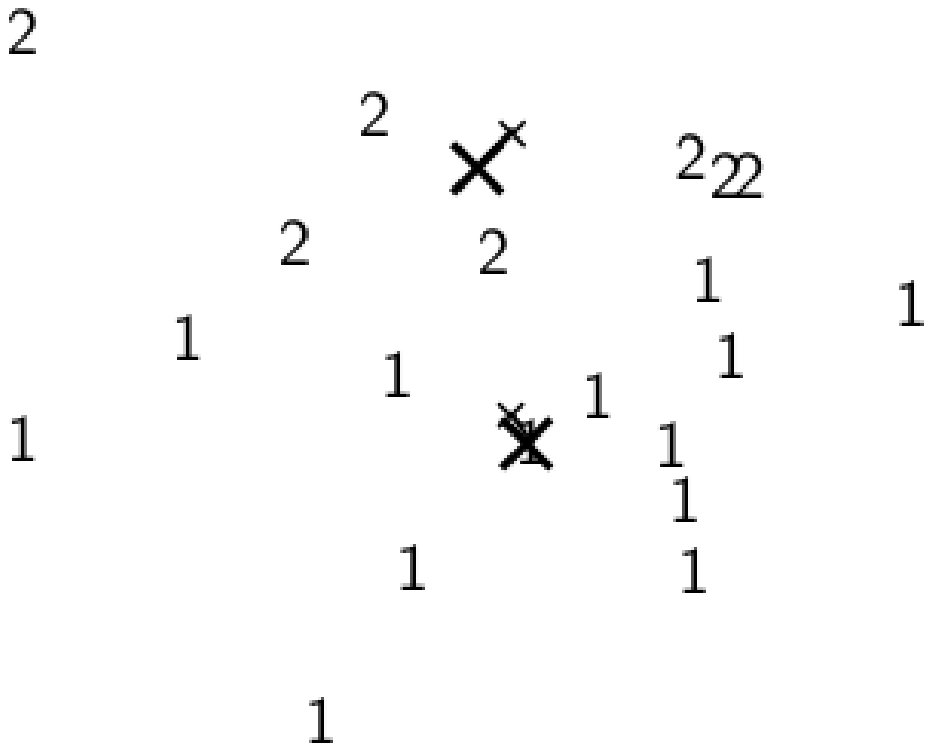
Worked Example: Assign points to closest centroid



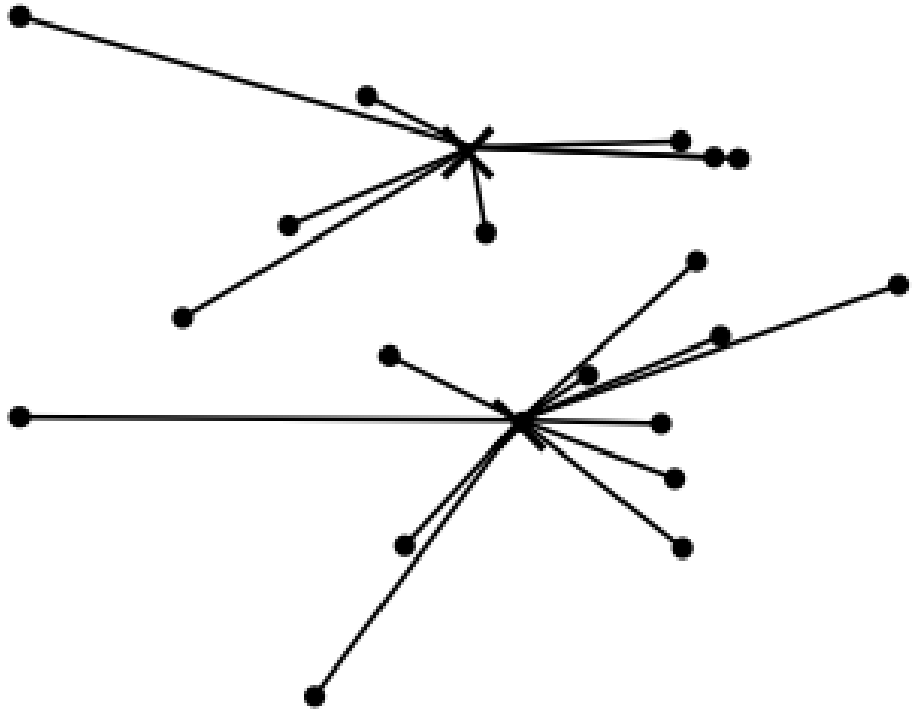
Worked Example: Assignment



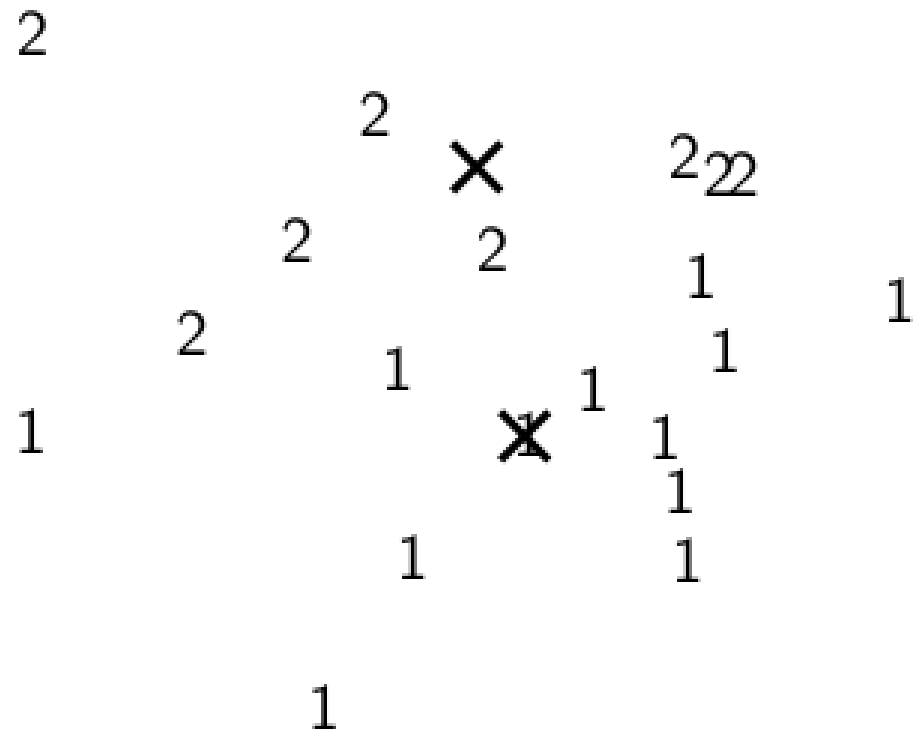
Worked Example: Recompute cluster centroids



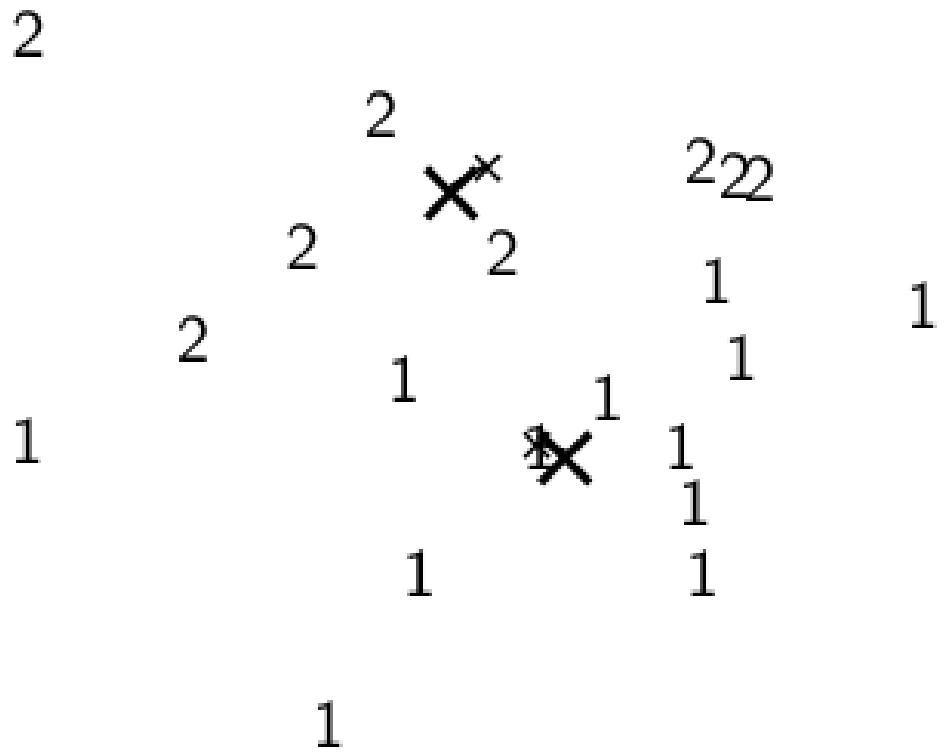
Worked Example: Assign points to closest centroid



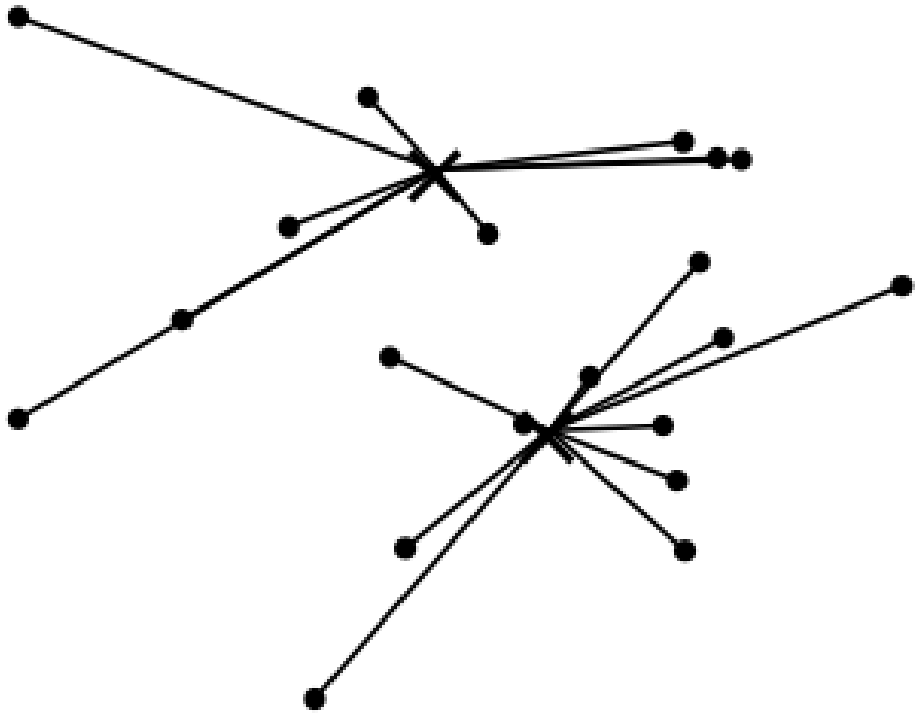
Worked Example: Assignment



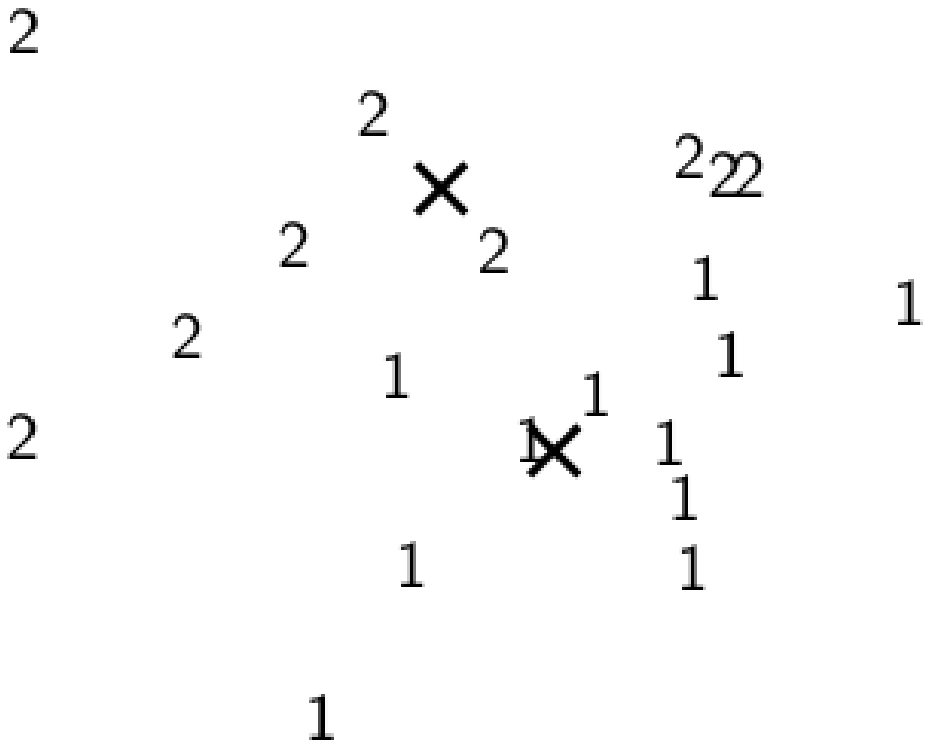
Worked Example: Recompute cluster centroids



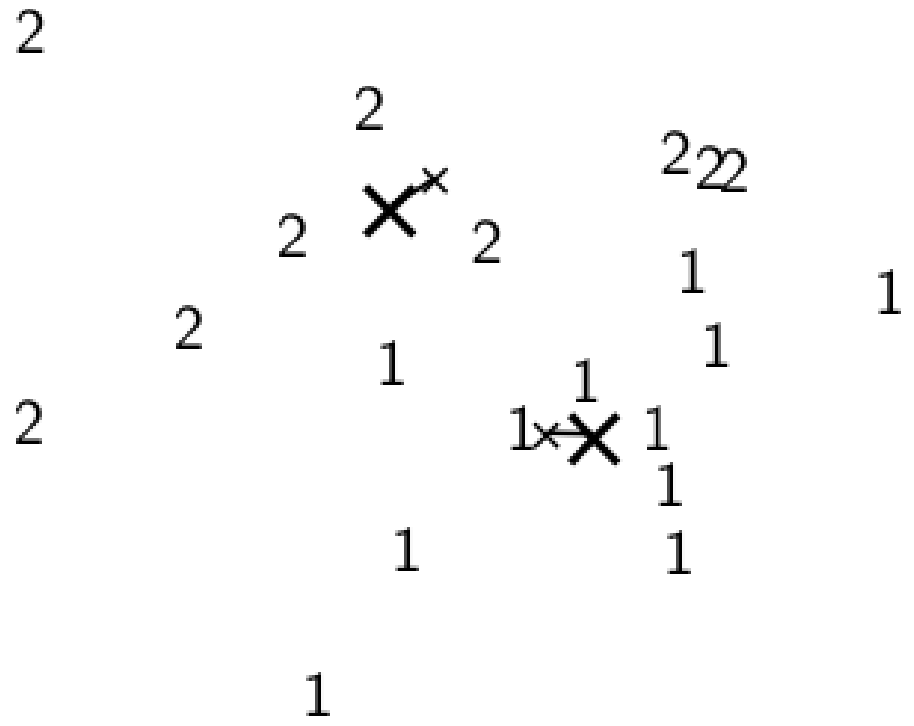
Worked Example: Assign points to closest centroid



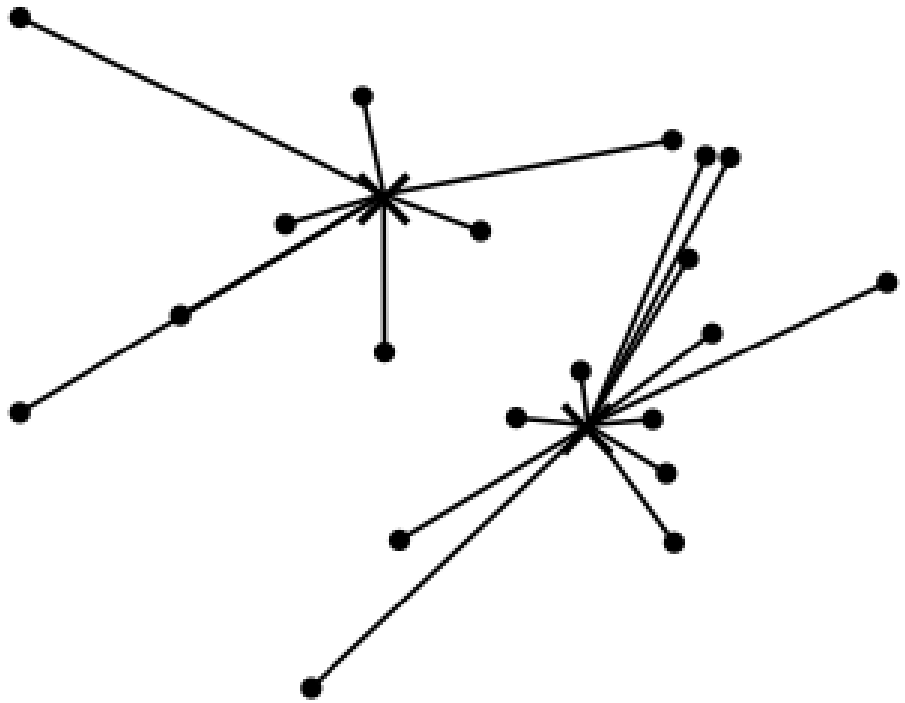
Worked Example: Assignment



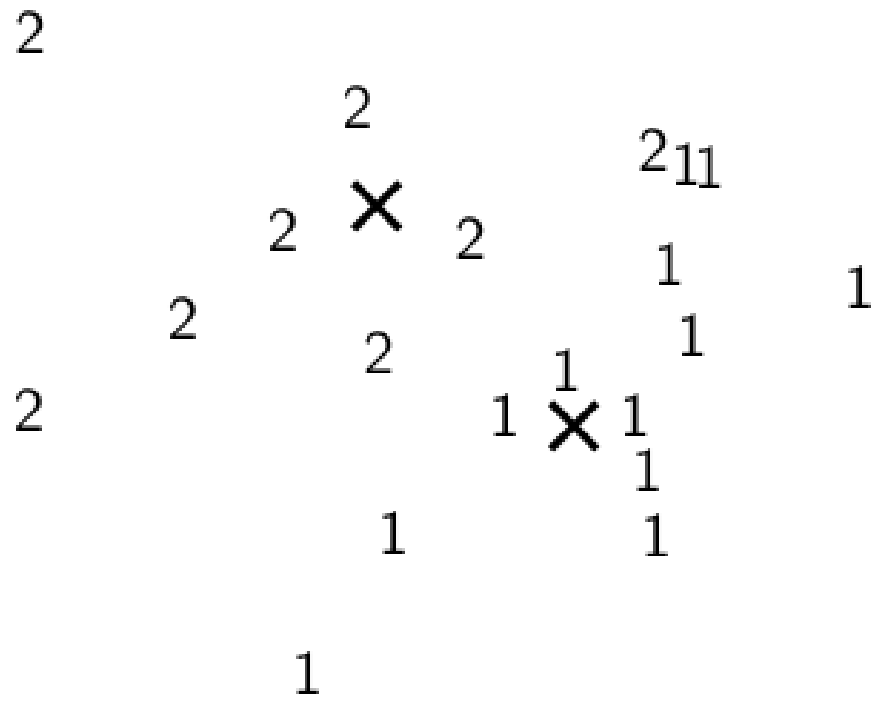
Worked Example: Recompute cluster centroids



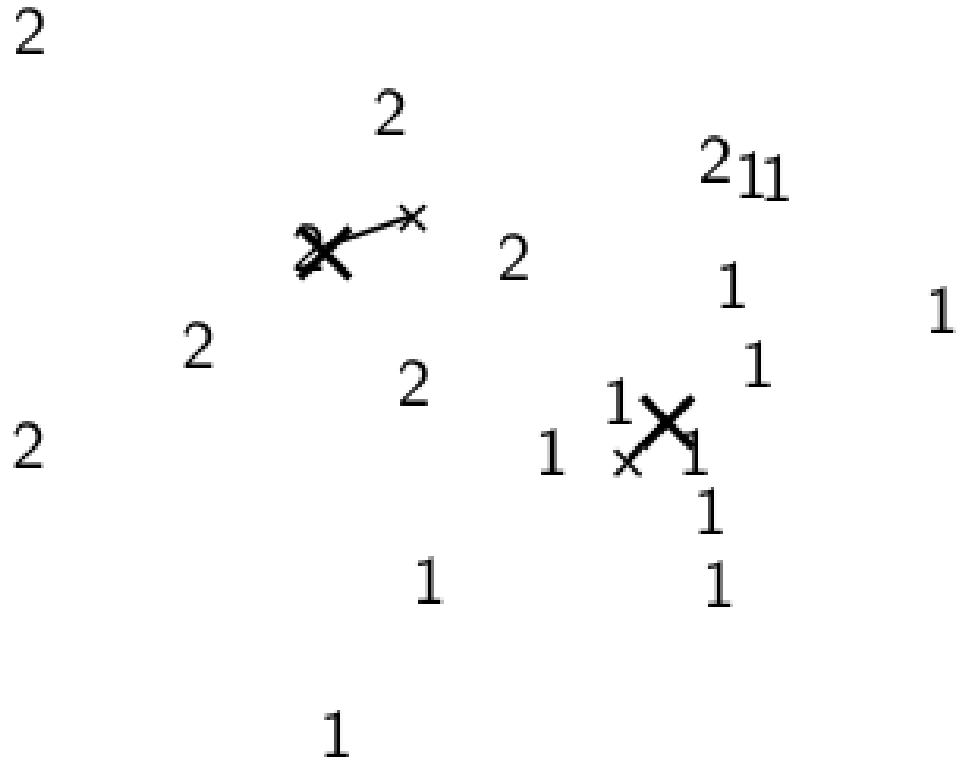
Worked Example: Assign points to closest centroid



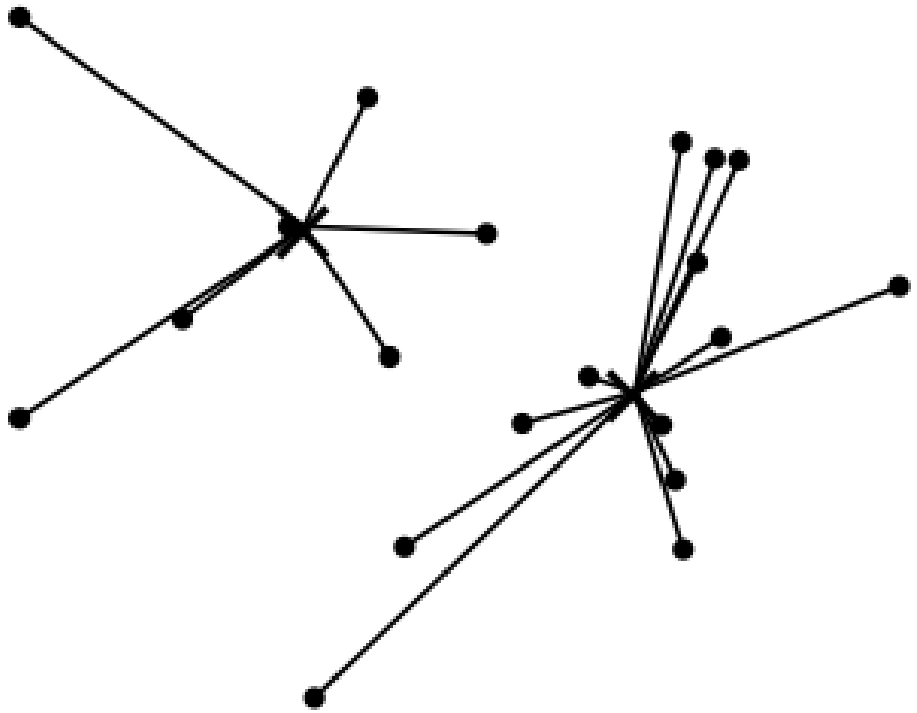
Worked Example: Assignment



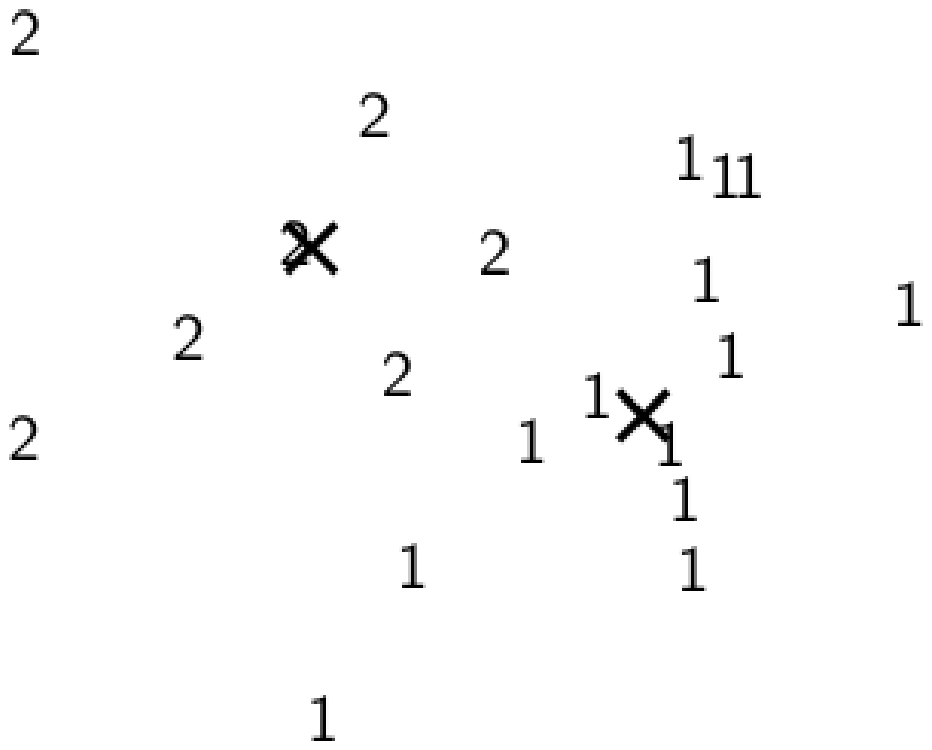
Worked Example: Recompute cluster centroids



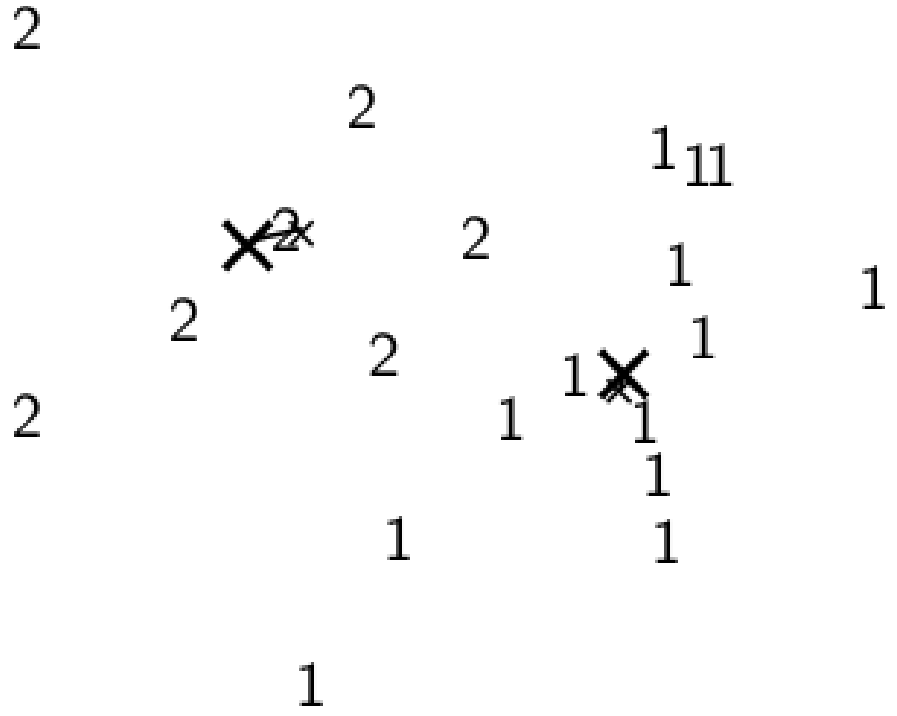
Worked Example: Assign points to closest centroid



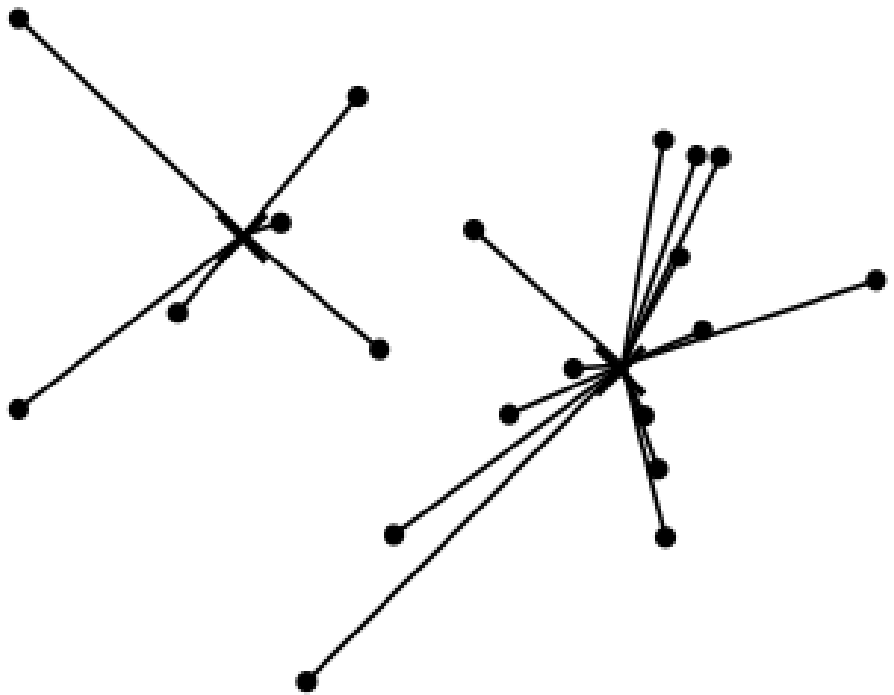
Worked Example: Assignment



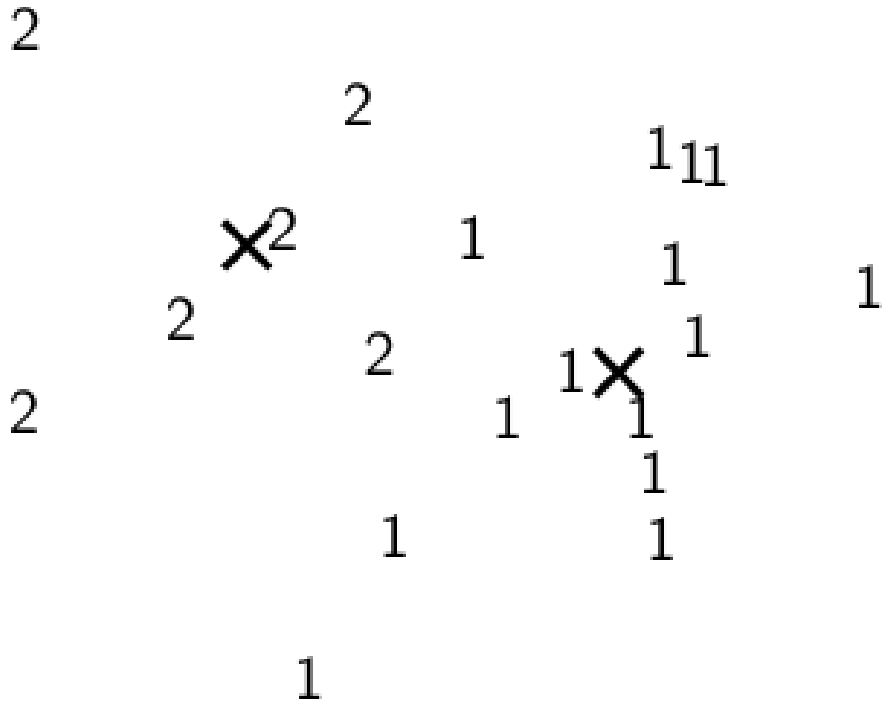
Worked Example: Recompute cluster centroids



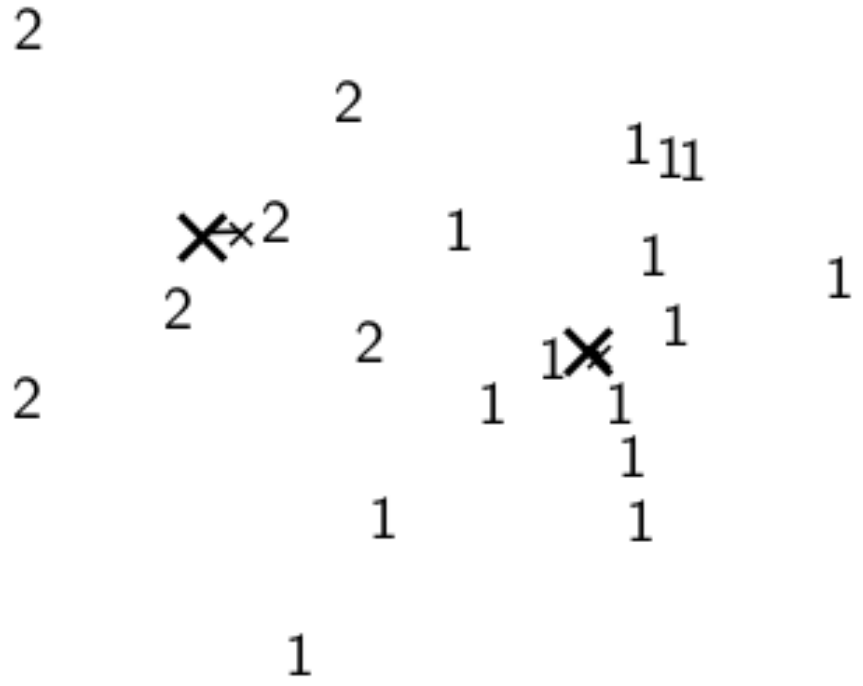
Worked Example: Assign points to closest centroid



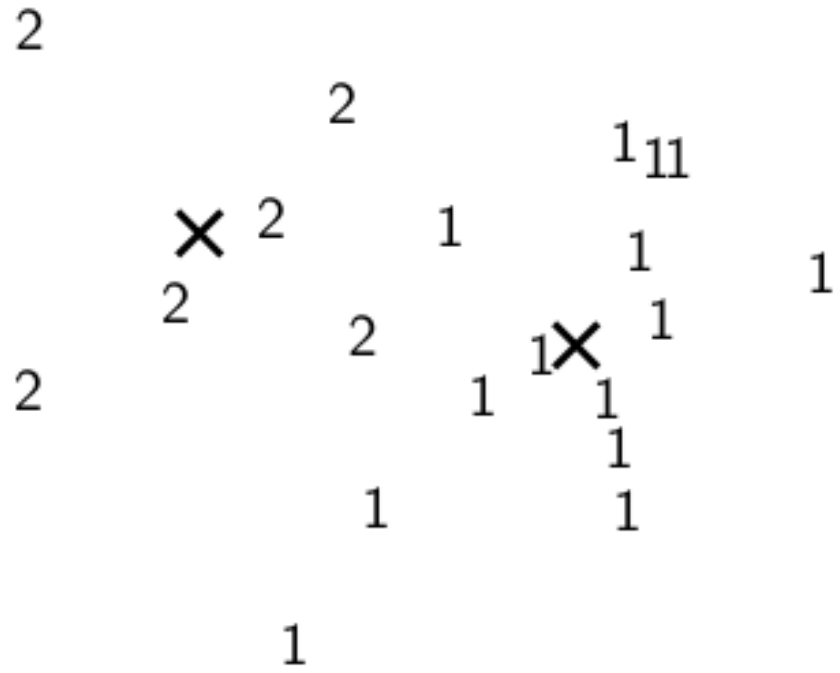
Worked Example: Assignment



Worked Example: Recompute cluster centroids



Worked Ex.: Centroids and assignments after convergence



K-means algorithm

K-MEANS$(\{\vec{x}_1, \ldots, \vec{x}_N\}, K)$
1 $(\vec{s}_1, \vec{s}_2, \ldots, \vec{s}_K) \leftarrow$ SELECTRANDOMSEEDS$(\{\vec{x}_1, \ldots, \vec{x}_N\}, K)$
2 **for** $k \leftarrow 1$ **to** K
3 **do** $\vec{\mu}_k \leftarrow \vec{s}_k$
4 **while** stopping criterion has not been met
5 **do for** $k \leftarrow 1$ **to** K
6 **do** $\omega_k \leftarrow \{\}$
7 **for** $n \leftarrow 1$ **to** N
8 **do** $j \leftarrow \arg\min_{j'} |\vec{\mu}_{j'} - \vec{x}_n|$
9 $\omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$ *(reassignment of vectors)*
10 **for** $k \leftarrow 1$ **to** K
11 **do** $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$ *(recomputation of centroids)*
12 **return** $\{\vec{\mu}_1, \ldots, \vec{\mu}_K\}$

K-means is guaranteed to converge

- RSS = sum of all squared distances between document vector and closest centroid
- RSS decreases during each reassignment step.
  - because each vector is moved to a closer centroid
- RSS decreases during each recomputation step.
  - see next slide
- There is only a finite number of clusterings.
- Thus: We must reach a fixed point.
- Assumption: Ties are broken consistently.

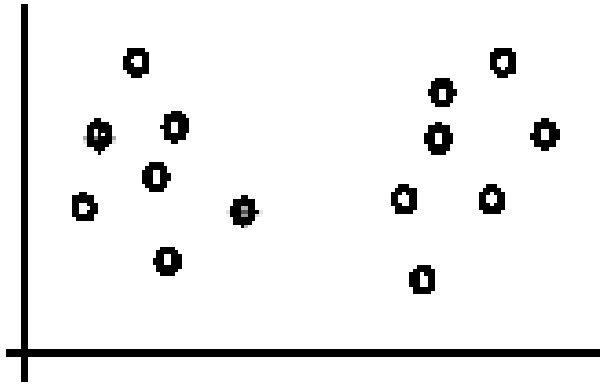
K-means is guaranteed to converge

But we don't know how long convergence will take!

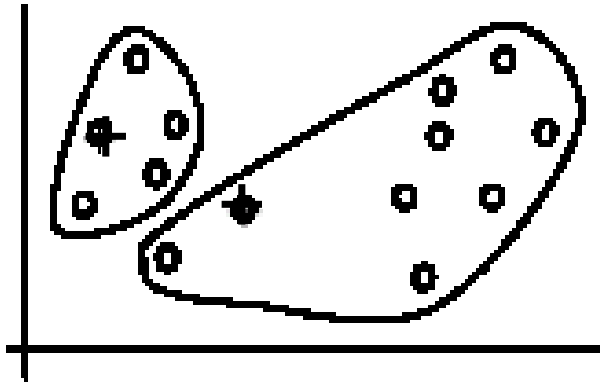
If we don't care about a few points switching back and forth, then convergence is usually fast (< 10-20 iterations).

However, complete convergence can take many more iterations.

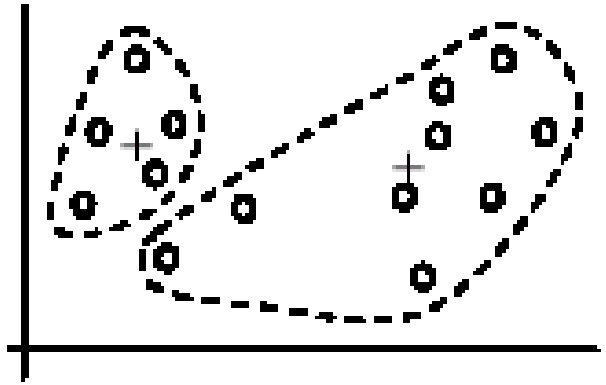
An example



(A). Random selection of k centers

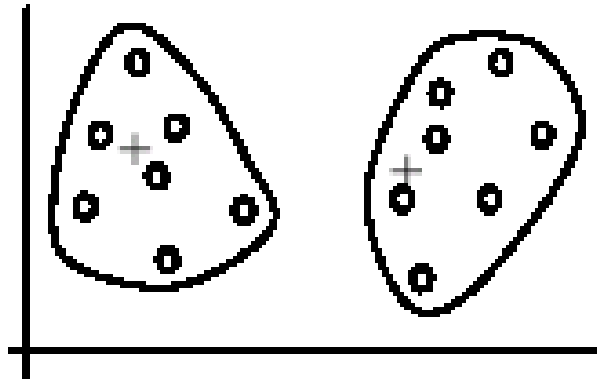


Iteration 1: (B). Cluster assignment

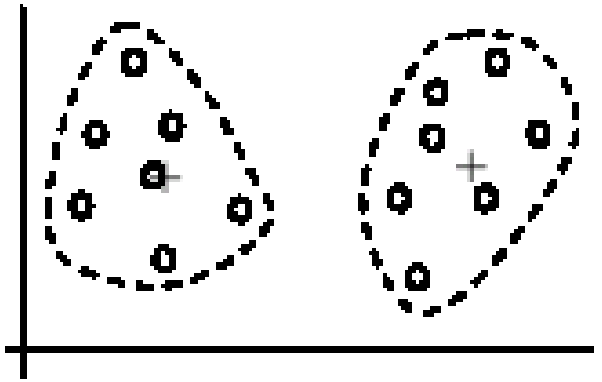


(C). Re-compute centroids

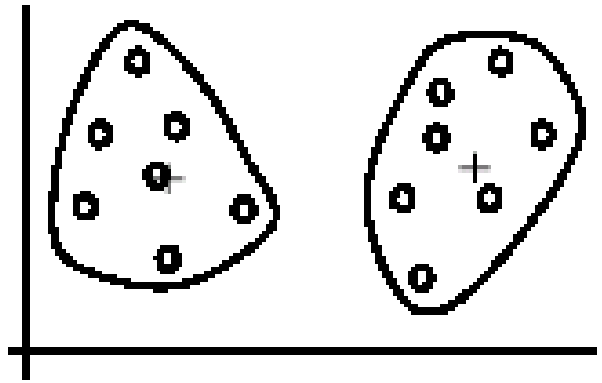
An example (cont …)



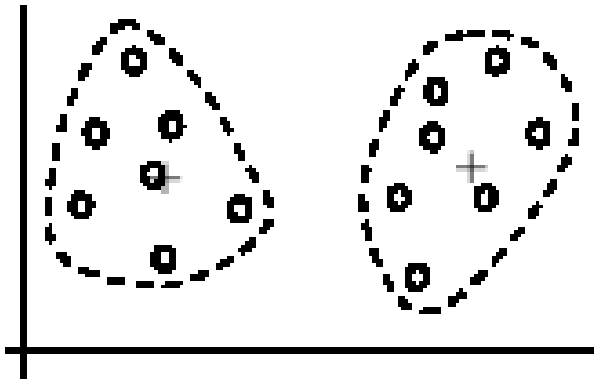
Iteration 2: (D). Cluster assignment



(E). Re-compute centroids



Iteration 3: (F). Cluster assignment



(G). Re-compute centroids

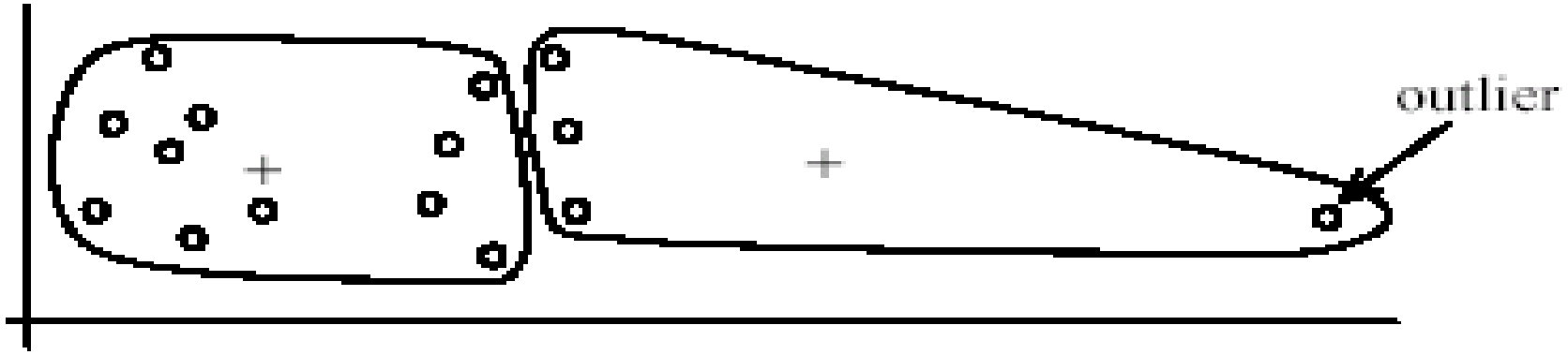
Strengths of k-means

- Strengths:
  - Simple: easy to understand and to implement
  - Efficient: Time complexity: $O(tkn)$,
    where n is the number of data points,
    k is the number of clusters, and
    t is the number of iterations.
  - Since both k and t are small. k-means is considered a linear algorithm.
- K-means is the most popular clustering algorithm.
- Note that: it terminates at a local optimum if RSS is used. The global optimum is hard to find due to complexity.

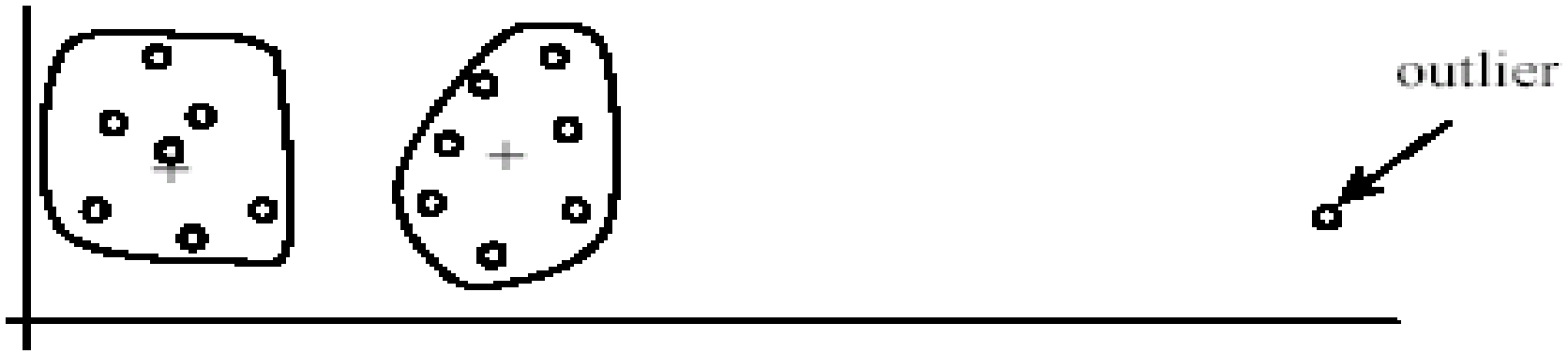
Weaknesses of k-means

- The algorithm is only applicable if the mean is defined.
  - For categorical data, *k*-mode - the centroid is represented by most frequent values.
- The user needs to specify *k*.
- The algorithm is sensitive to **outliers**
  - Outliers are data points that are very far away from other data points.
  - Outliers could be errors in the data recording or some special data points with very different values.

Weaknesses of k-means: Problems with outliers



(A): Undesirable clusters



(B): Ideal clusters

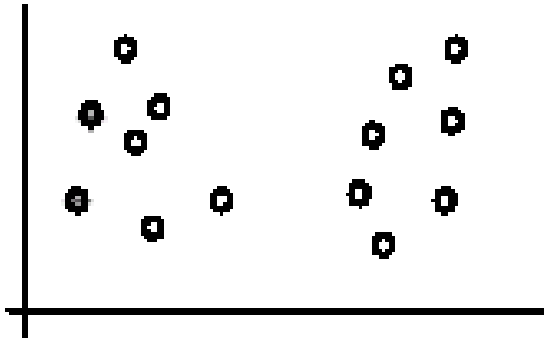
Weaknesses of k-means: To deal with outliers

- One method is to remove some data points in the clustering process that are much further away from the centroids than other data points.
  - To be safe, we may want to monitor these possible outliers over a few iterations and then decide to remove them.

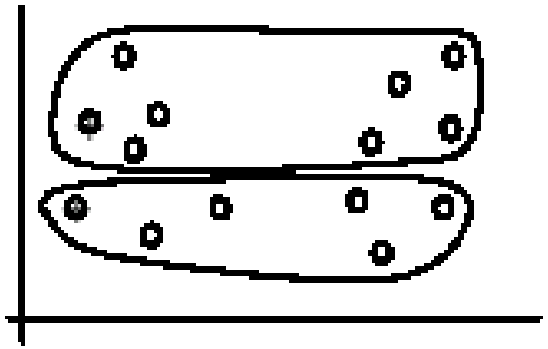
- Another method is to perform random sampling. Since in sampling we only choose a small subset of the data points, the chance of selecting an outlier is very small.
  - Assign the rest of the data points to the clusters by distance or similarity comparison, or classification

Weaknesses of k-means (cont …)

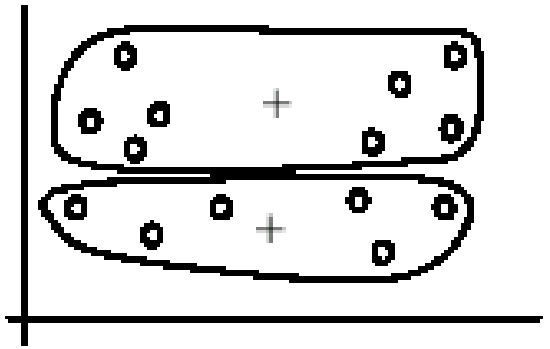
- The algorithm is sensitive to initial seeds.



(A). Random selection of seeds (centroids)



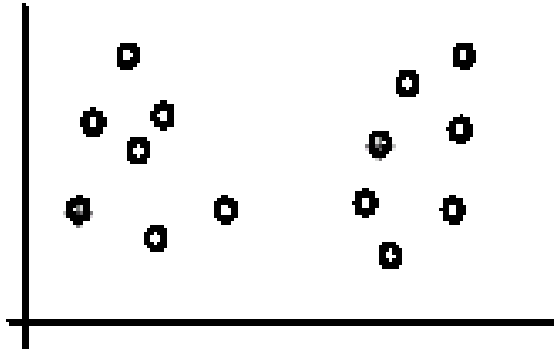
(B). Iteration 1



(C). Iteration 2

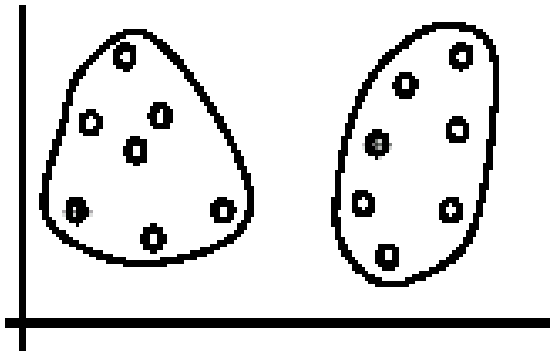
Weaknesses of k-means (cont …)

- If we use different seeds: good results

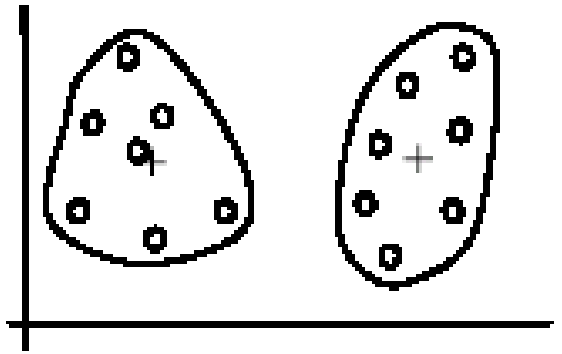


(A). Random selection of k seeds (centroids)

There are some methods to help choose good seeds



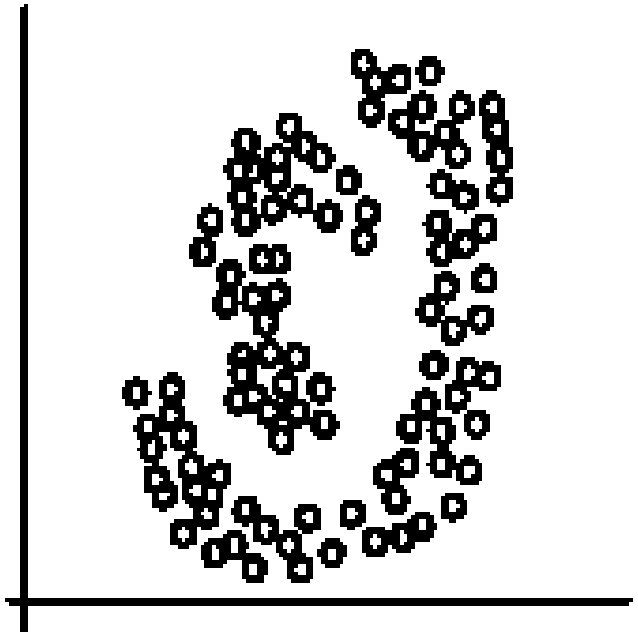
(B). Iteration 1



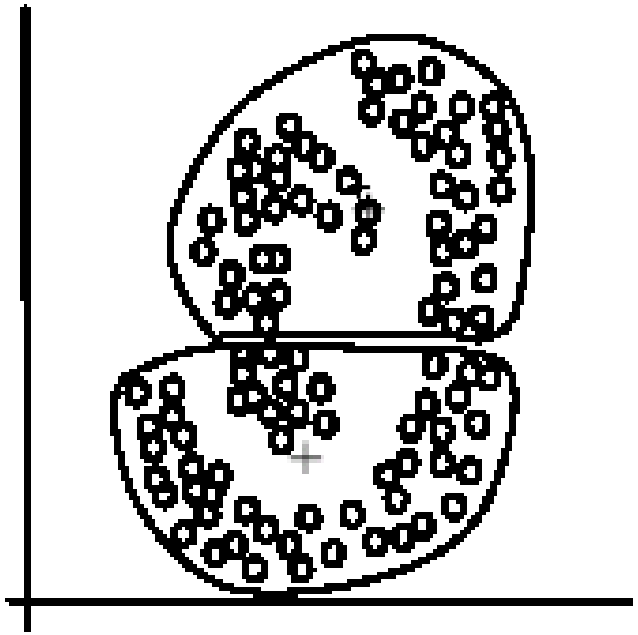
(C). Iteration 2

Weaknesses of k-means (cont …)

- The *k*-means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).



(A): Two natural clusters



(B): *k*-means clusters

K-means Initialization

We need to pick some points for the first round of the algorithm:

- **Random sample:** Pick a random subset of k points from the dataset.
- **K-Means++:** Iteratively construct a random sample with good spacing across the dataset.

Note: Finding an exactly-optimal k-Means clustering is NP-hard. Randomization helps avoid bad configurations.

K-means++

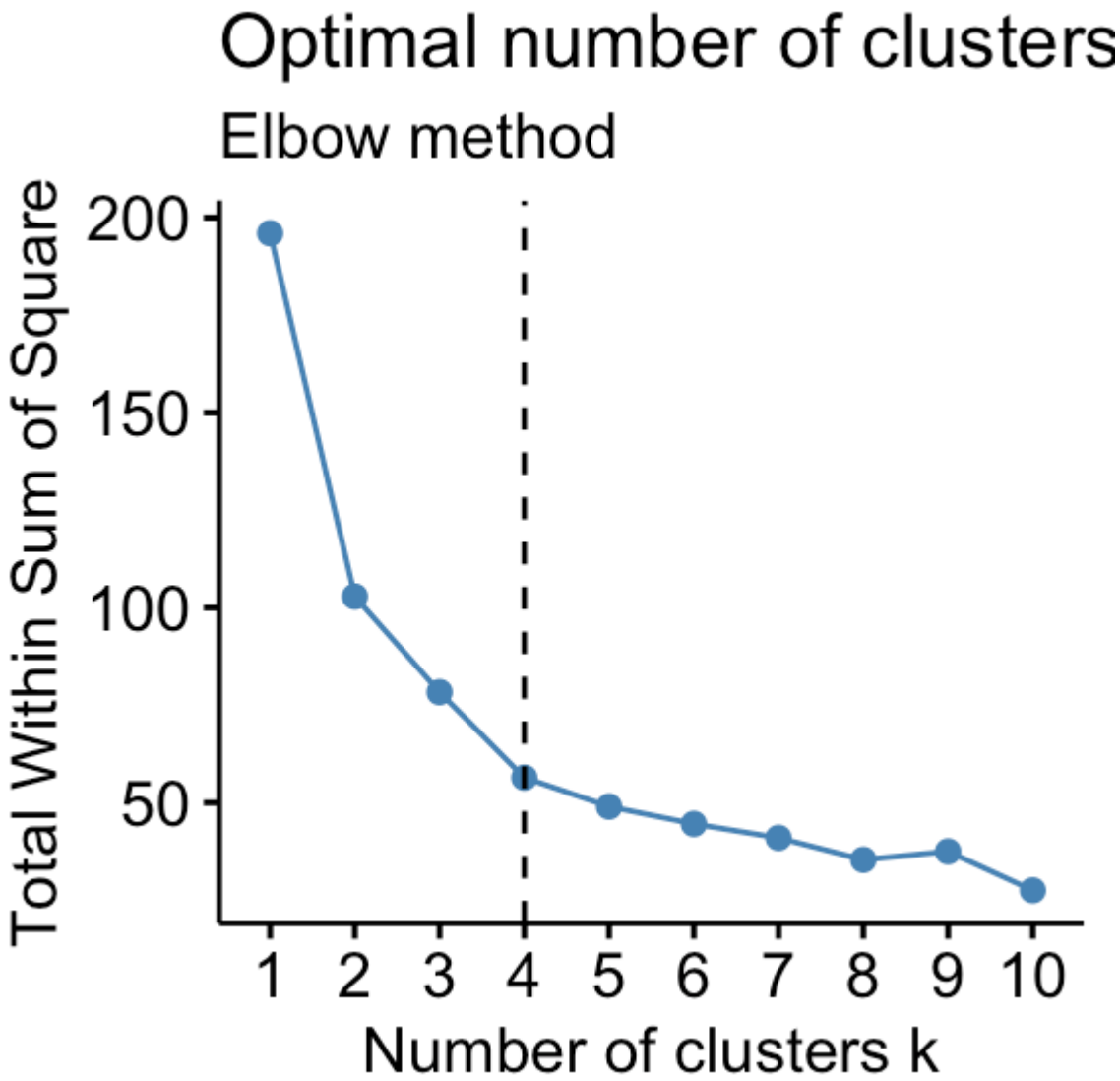
Start:

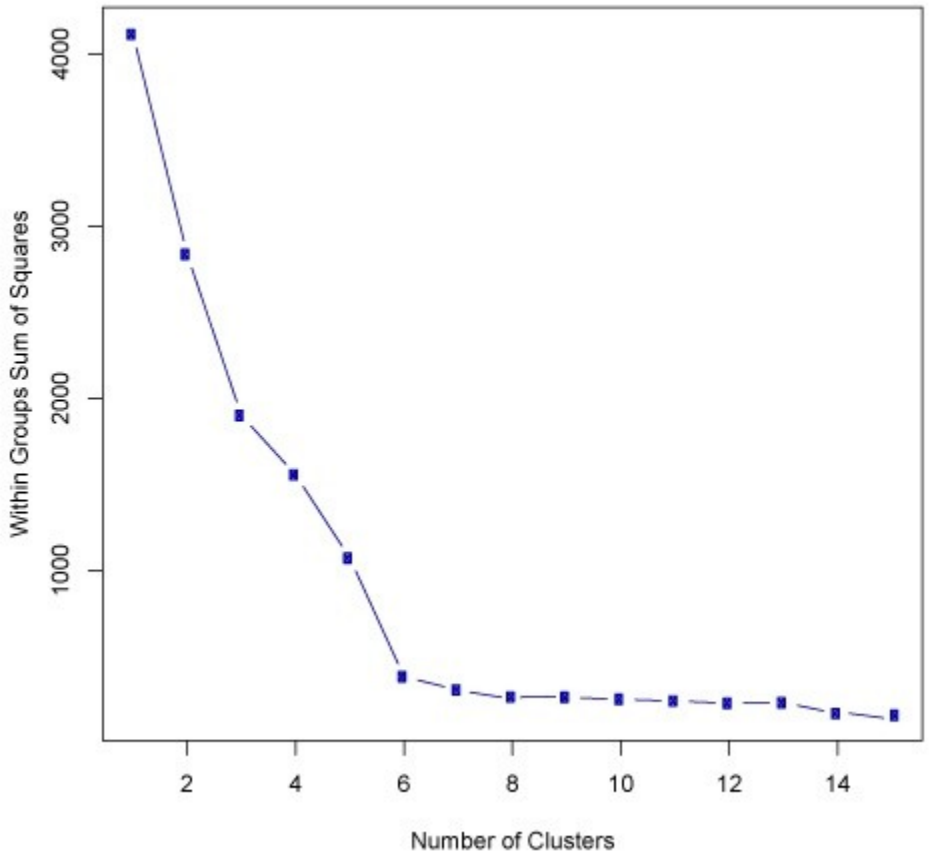
- Choose first cluster center at random from the data points

Iterate:

- For every remaining data point x, compute D(x) the distance from x to the closest cluster center.
- Choose a remaining point x randomly with probability proportional to $D(x)^2$, and make it a new cluster center.

Intuitively, this finds a sample of widely-spaced points avoiding "collapsing" of the clustering into a few internal centers.


Optimal number of clusters
Elbow method
Total Within Sum of Square
200
150
100
50
1 2 3 4 5 6 7 8 9 10
Number of clusters k

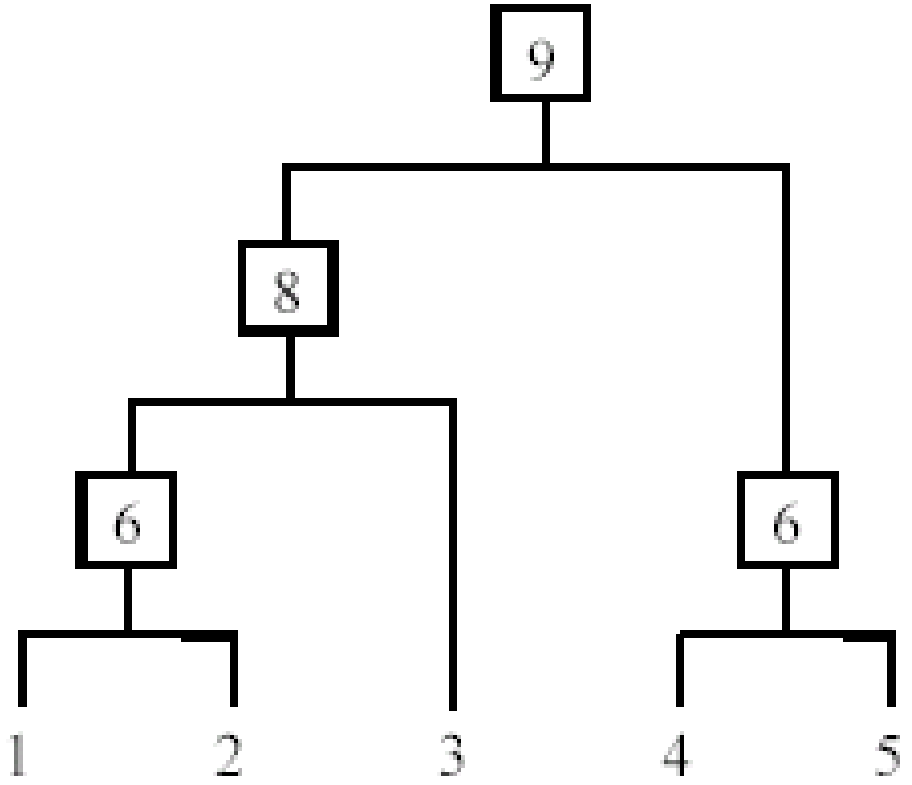

Within Groups Sum of Squares
4000
3000
2000
1000
2 4 6 8 10 12 14
Number of Clusters

K-means properties

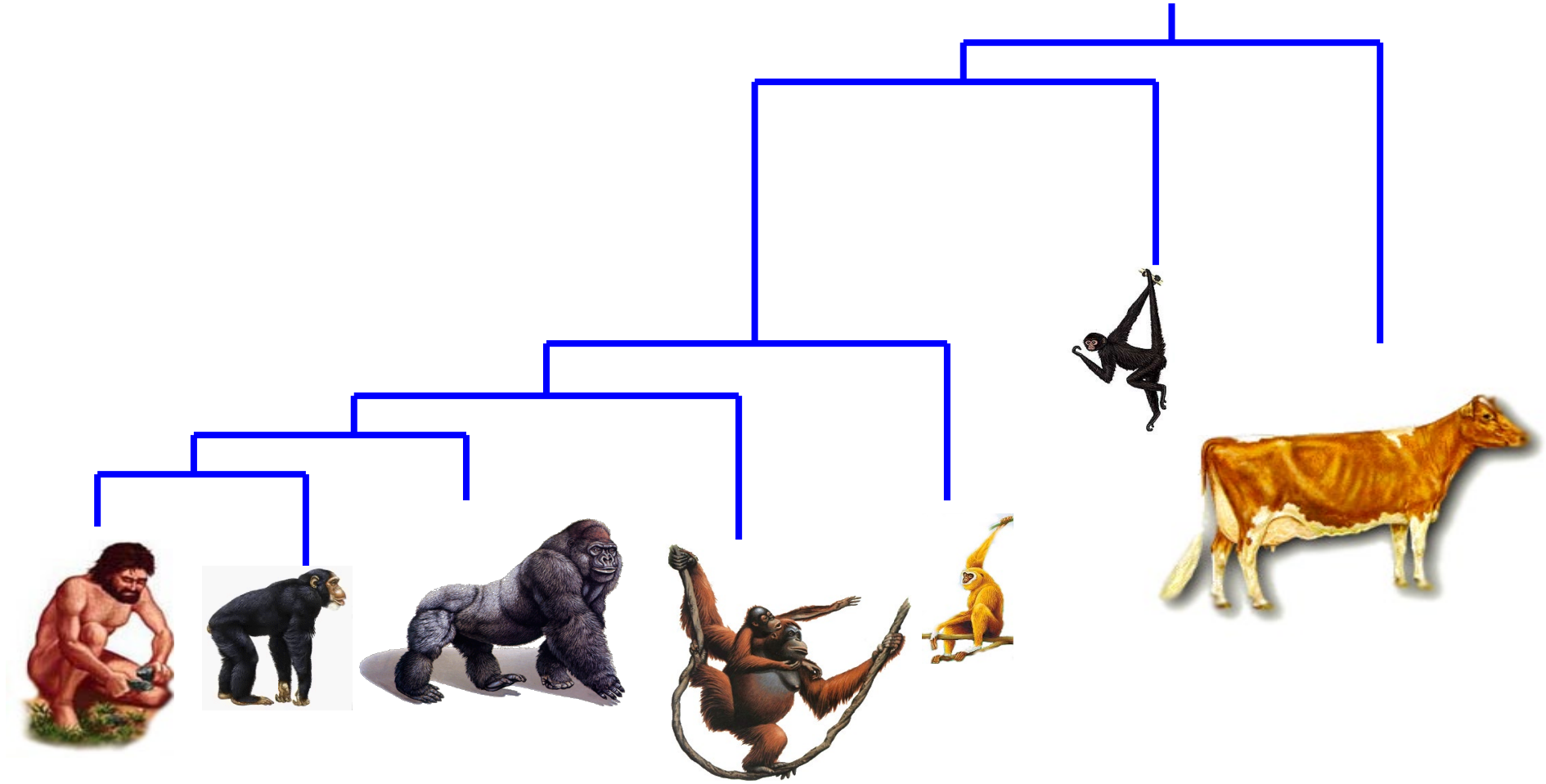
- It's a greedy algorithm with random setup – **solution isn't optimal** and varies significantly with different initial points.
- Very simple convergence proofs.
- **Performance is O(nk) per iteration**, not bad and can be heuristically improved.
  n = total features in the dataset, k = number clusters
- Many generalizations, e.g.
  - Fixed-size clusters
  - Simple generalization to m-best soft clustering
- As a "local" clustering method, it works well for data condensation/compression.

Hierarchical Clustering

- Produce a nested sequence of clusters, a tree, also called Dendrogram.



There is only one dataset that can be perfectly clustered using a hierarchy…



(Bovine:0.69395, (Spider Monkey 0.390, (Gibbon:0.36079,(Orang:0.33636,(Gorilla:0.17147,(Chimp:0.19268, Human:0.11927):0.08386):0.06124):0.15057):0.54939);

Types of hierarchical clustering

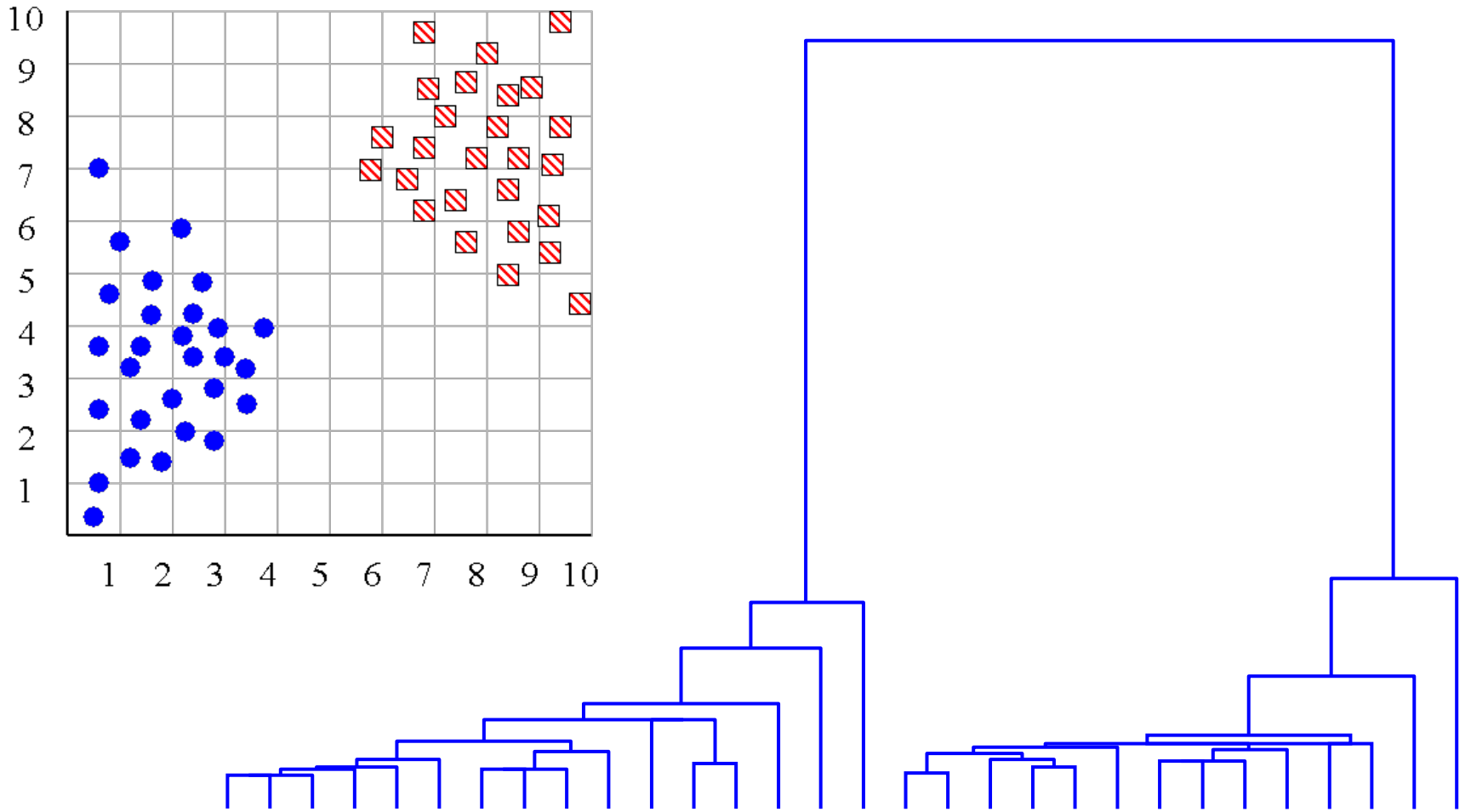
- Agglomerative (bottom up) clustering: It builds the dendrogram (tree) from the bottom level, and
  - merges the most similar (or nearest) pair of clusters
  - stops when all the data points are merged into a single cluster (i.e., the root cluster).
- Divisive (top down) clustering: It starts with all data points in one cluster, the root.
  - Splits the root into a set of child clusters. Each child cluster is recursively divided further
  - stops when only singleton clusters of individual data points remain, i.e., each cluster with only a single point

Agglomerative clustering

It is more popular then divisive methods.

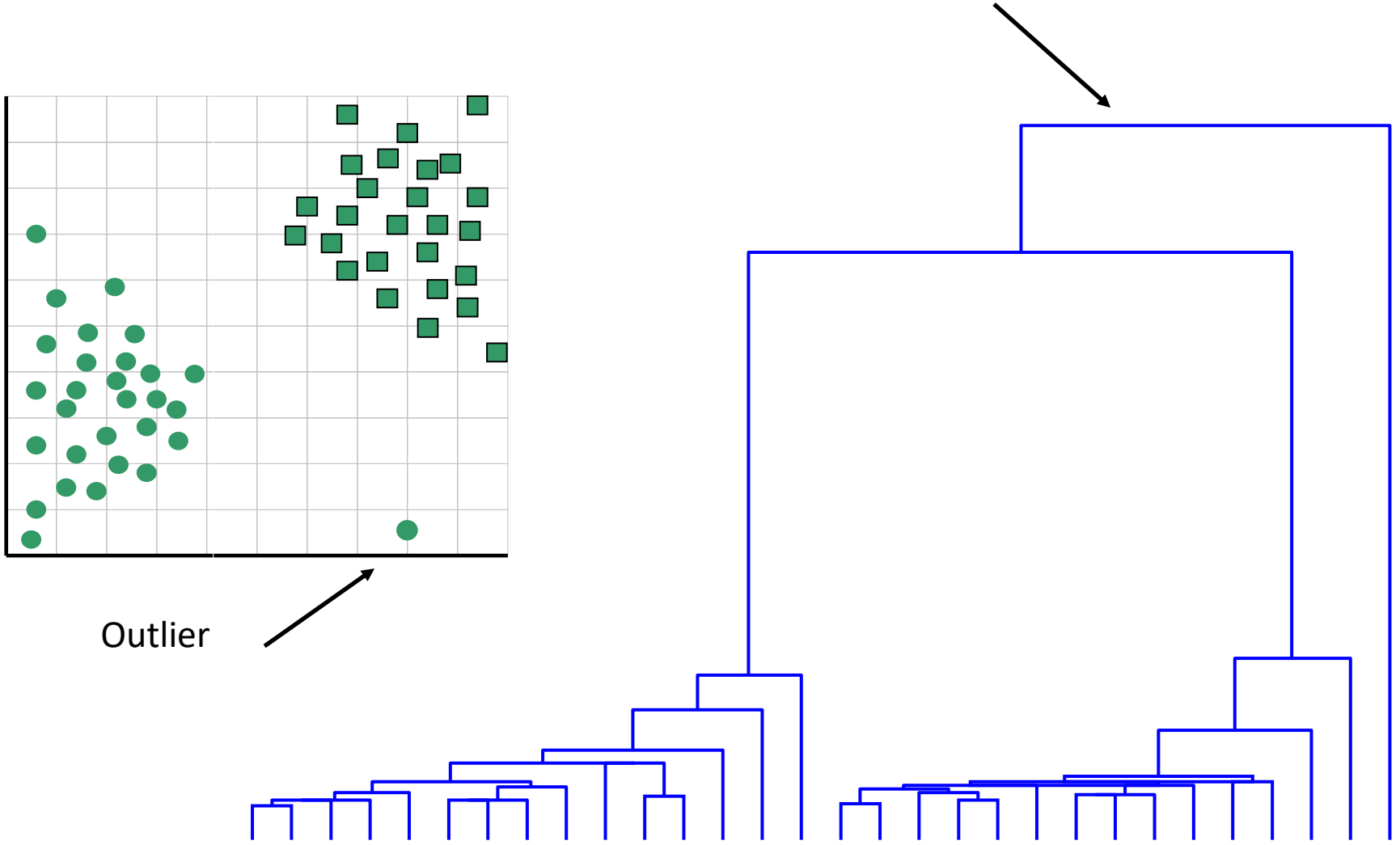
- At the beginning, each data point forms a cluster (also called a node).
- Merge nodes/clusters that have the least distance.
- Go on merging
- Eventually all nodes belong to one cluster

We can look at the dendrogram to determine the “correct” number of clusters. In this case, the two highly separated subtrees are highly suggestive of two clusters. (Things are rarely this clear cut, unfortunately)



One potential use of a dendrogram is to detect outliers

The single isolated branch is suggestive of a data point that is very different to all others

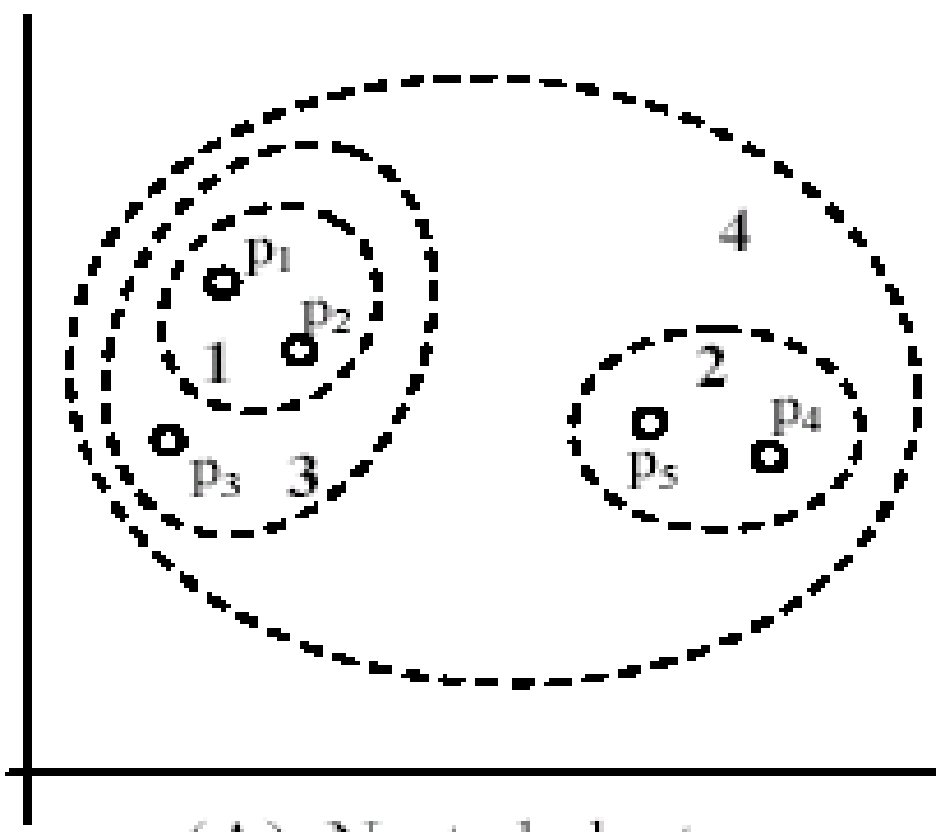


Agglomerative clustering algorithm

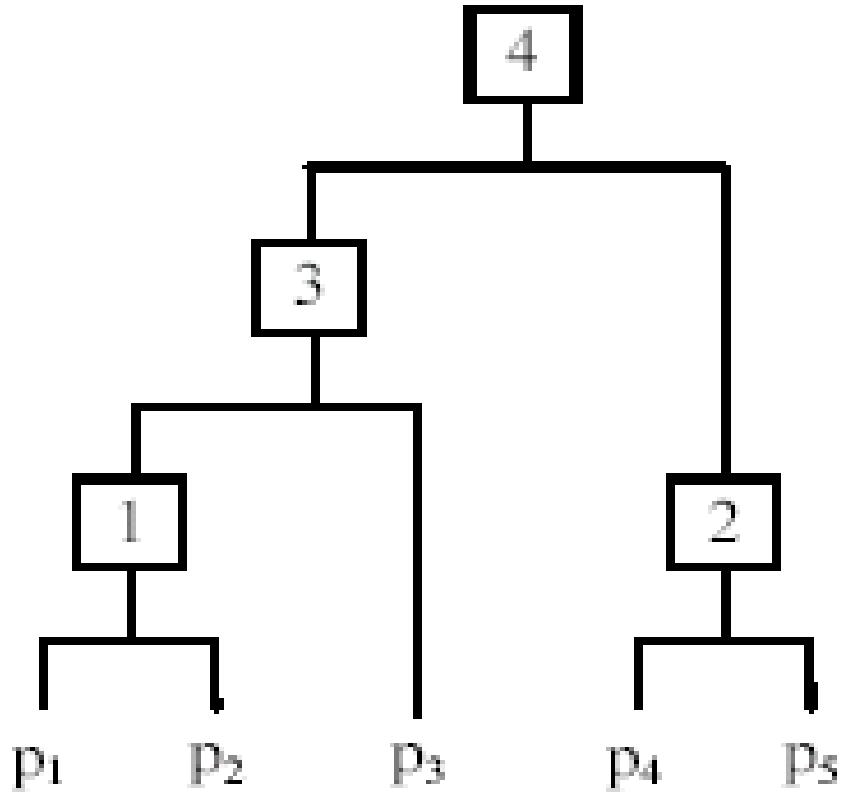
Algorithm Agglomerative(D)

1 Make each data point in the data set D a cluster,
2 Compute all pair-wise distances of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in D$;
2 **repeat**
3 find two clusters that are nearest to each other;
4 merge the two clusters form a new cluster c;
5 compute the distance from c to all other clusters;
12 **until** there is only one cluster left

An example: working of the algorithm



(A). Nested clusters



(B) Dendrogram

Measuring the distance of two clusters

- A few ways to measure distances of two clusters.
- Results in different variations of the algorithm.
  - Single link
  - Complete link
  - Average link
  - Centroids
  - …

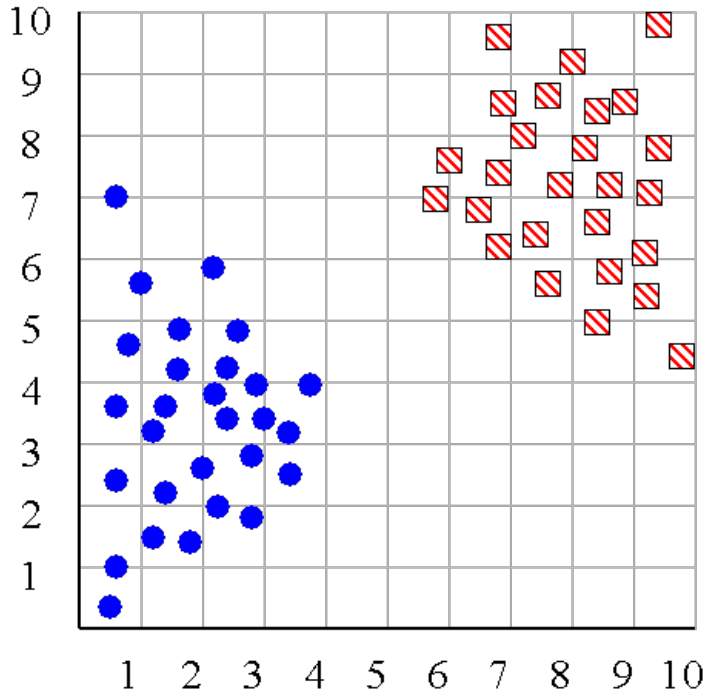
We know how to measure the distance between two objects, but defining the distance between an object and a cluster, or defining the distance between two clusters is non obvious.

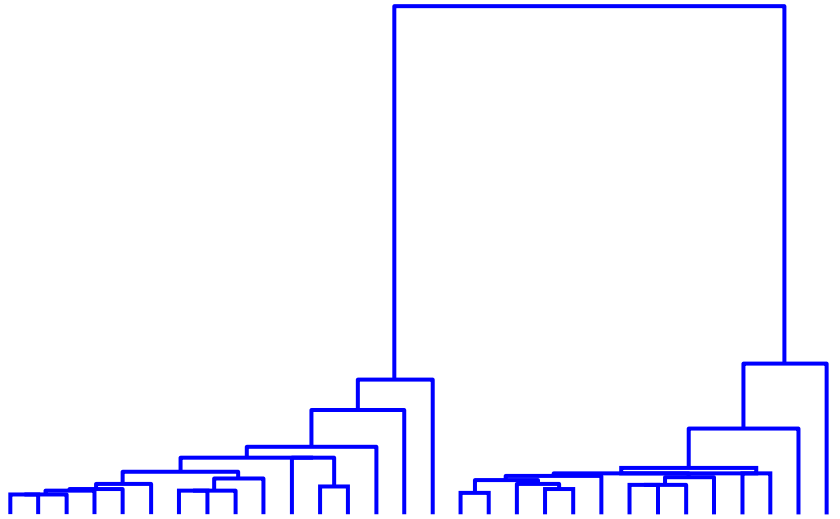
Single linkage (nearest neighbor): In this method the distance between two clusters is determined by the distance of the two closest objects (nearest neighbors) in the different clusters.

Complete linkage (furthest neighbor): In this method, the distances between clusters are determined by the greatest distance between any two objects in the different clusters (i.e., by the "furthest neighbors").

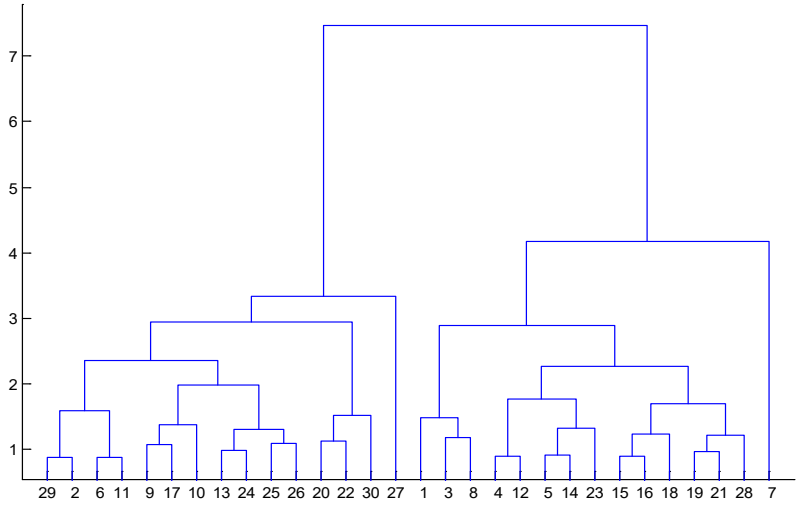
Group average linkage: In this method, the distance between two clusters is calculated as the average distance between all pairs of objects in the two different clusters.

Wards Linkage: In this method, we try to minimize the variance of the merged clusters

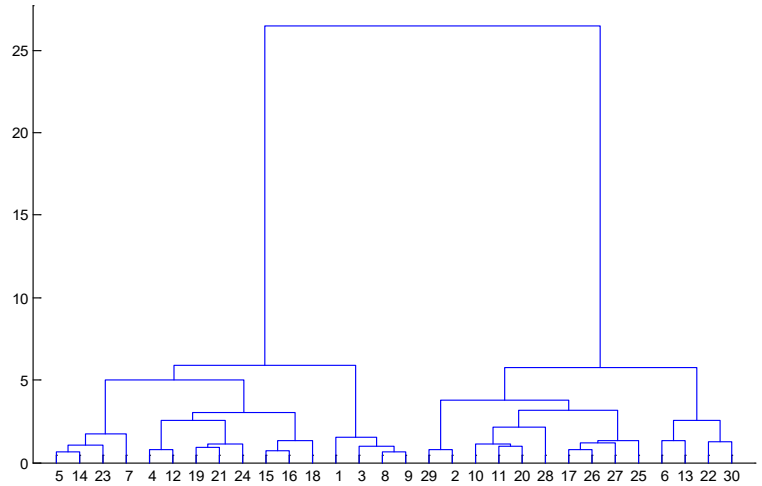




Single linkage



Average linkage



Wards linkage

Summary of Hierarchal Clustering Methods

- No need to specify the number of clusters in advance.
- Hierarchal nature maps nicely onto human intuition for some domains
- They do not scale well: time complexity of at least $O(n^2)$, where n is the number of total objects.
- Like any heuristic search algorithms, local optima are a problem.
- Interpretation of results is (very) subjective.

Density based clustering algorithms

Density based clustering algorithms make an assumption that clusters are dense regions in space separated by regions of lower density.

A dense cluster is a region which is “density connected”, i.e. the density of points in that region is greater than a minimum.

Since these algorithms expand clusters based on dense connectivity, they can find clusters of arbitrary shapes.

DBSCAN is an example of density based clustering algorithm.

DBSCAN (Density Based Spatial Clustering of Applications with Noise)

Published by Ester et. al. in 1996

The algorithm finds dense areas and expands these recursively to find dense arbitrarily shaped clusters.

Two main parameters to DBSCAN are ‘ε’ and ‘minPoints’. ‘ε’ defines radius of the ‘neighborhood region’ and ‘minPoints’ defines the minimum number of points that should be contained within that neighborhood.

Since it has a concept of noise, it works well even with noisy datasets.

Epsilon neighborhood *(Nε)* : set of all points within a distance ‘ε’.

Core point : A point that has at least ‘minPoint’ (including itself) points within it’s *N*ε .

Direct Density Reachable (DDR) : A point *q* is directly density reachable from a point *p* if *p* is core point and *q* ∈ *N*ε .

Density Reachable (DR) : Two points are DR if there is a chain of DDR points that link these two points.

Border Point: Point that are DDR but not a core point.

Noise : Points that do not belong to any point’s *N*ε .

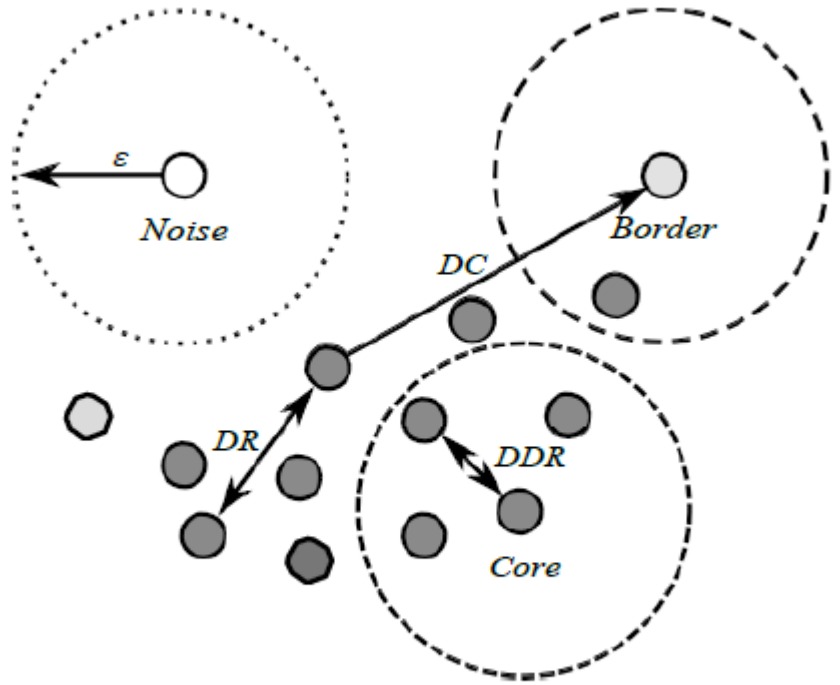


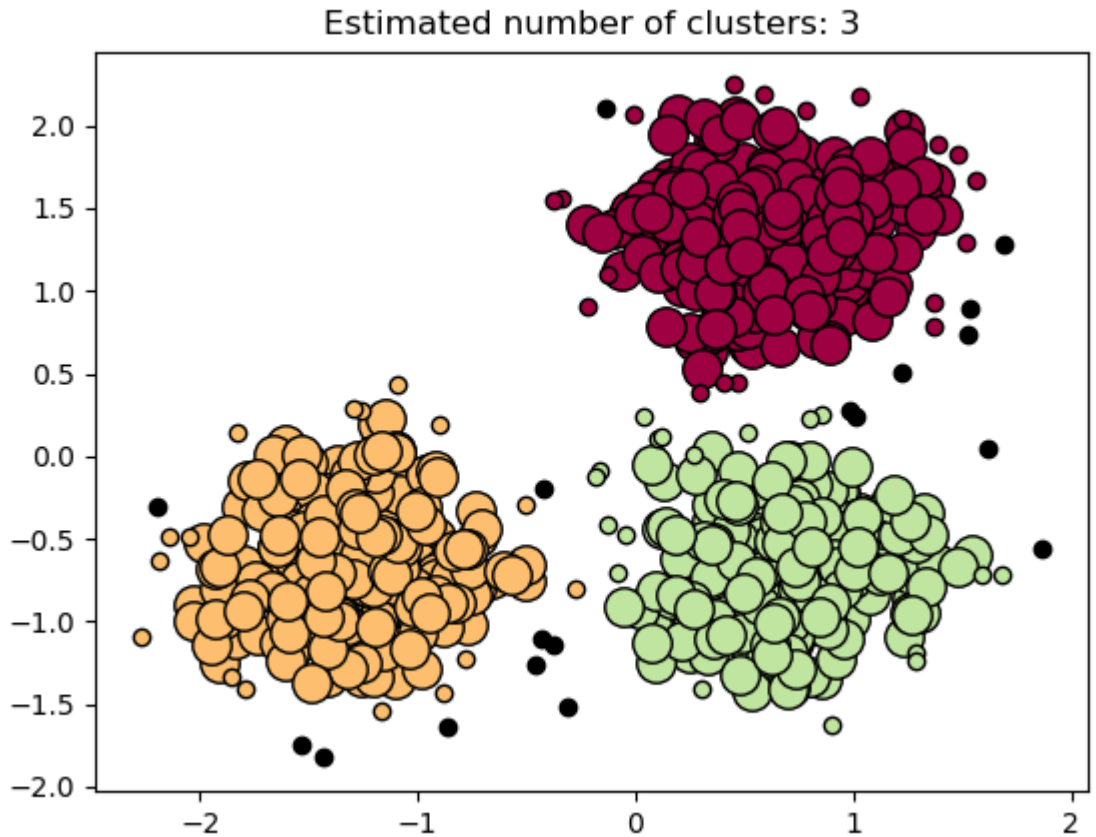
Figure 1: *DBSCAN* **clustering with** $minPoints = 4$

The steps to the DBSCAN algorithm are:

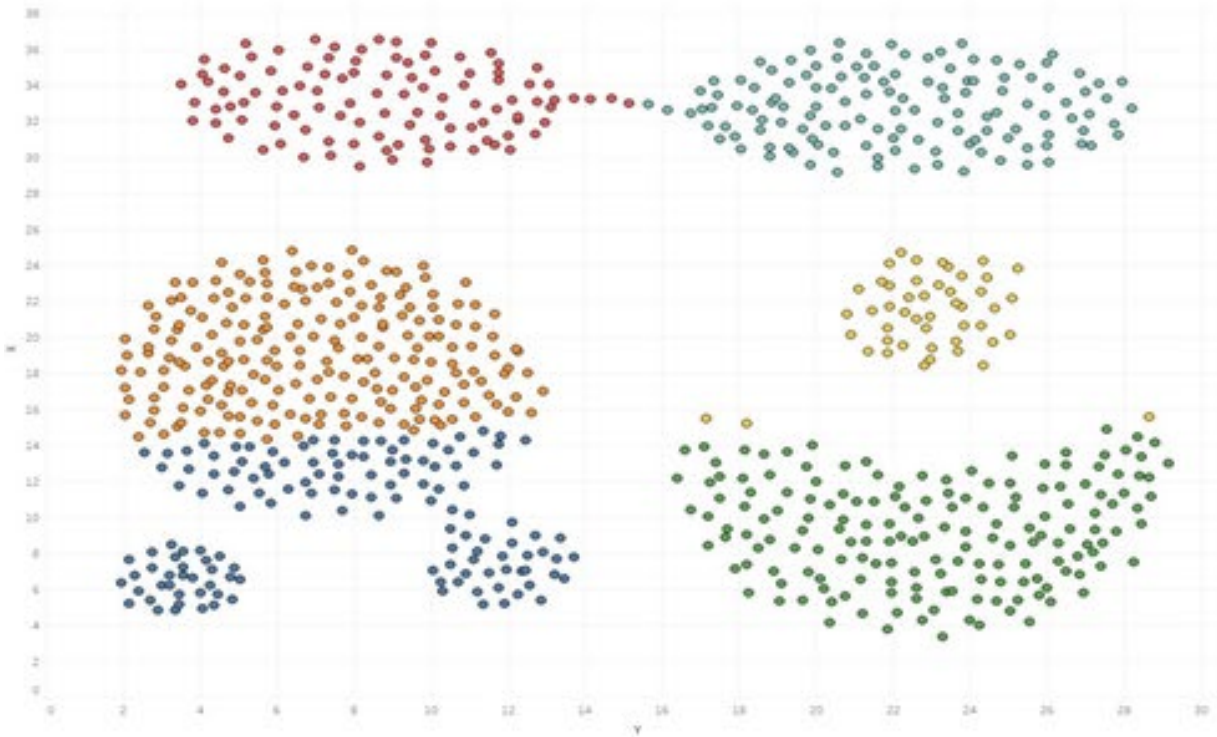
Pick a point at random that has not been assigned to a cluster or been designated as an *outlier*. Compute its neighborhood to determine if it's a *core point*. If yes, start a cluster around this point. If no, label the point as an *outlier*.

Once we find a *core point* and thus a cluster, expand the cluster by adding all *directly-reachable* points to the cluster. Perform "neighborhood jumps" to find all *density-reachable* points and add them to the cluster. If an an *outlier* is added, change that point's status from *outlier* to *border point*.

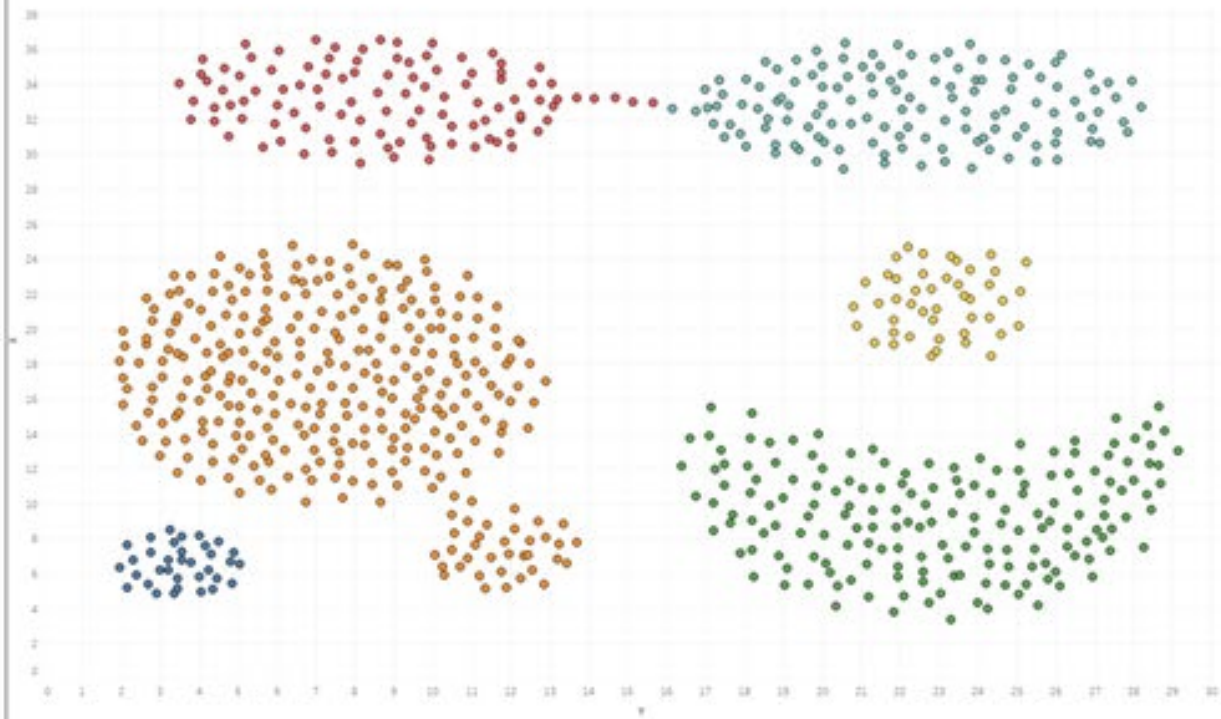
Repeat these two steps until all points are either assigned to a cluster or designated as an *outlier*.

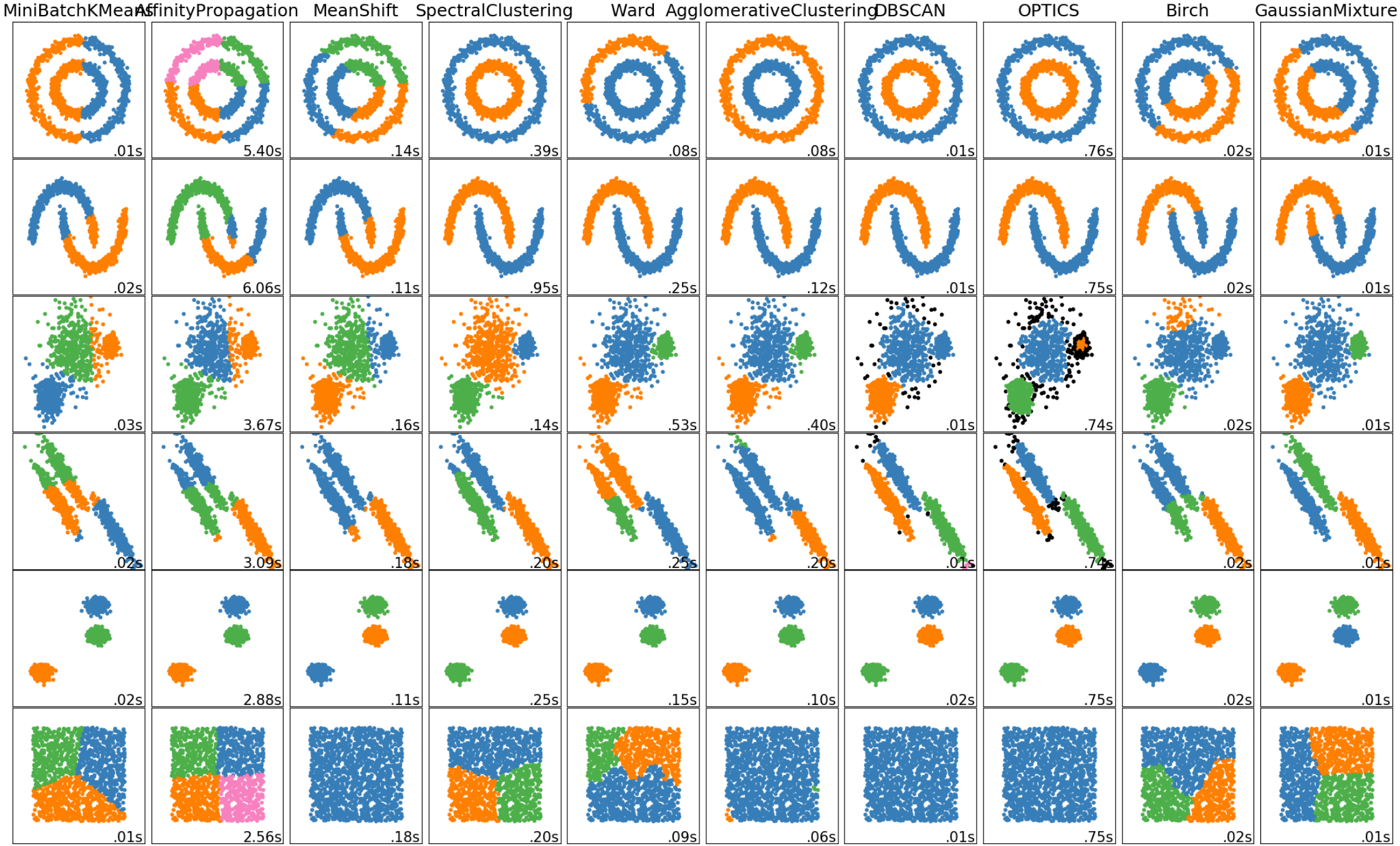

Estimated number of clusters: 3
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2

Traditional Clustering (K-means)



DBSCAN




MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
AgglomerativeClustering
DBSCAN
OPTICS
Birch
GaussianMixture
.01s
5.40s
.14s
.39s
.08s
.08s
.01s
.76s
.02s
.01s
.02s
6.06s
.11s
.95s
.25s
.12s
.01s
.75s
.02s
.01s
.03s
3.67s
.16s
.14s
.53s
.40s
.01s
.74s
.02s
.01s
.02s
3.09s
.18s
.20s
.25s
.20s
.01s
.74s
.02s
.01s
.02s
2.88s
.11s
.25s
.15s
.10s
.02s
.75s
.02s
.01s
.01s
2.56s
.18s
.20s
.09s
.06s
.01s
.75s
.02s
.01s