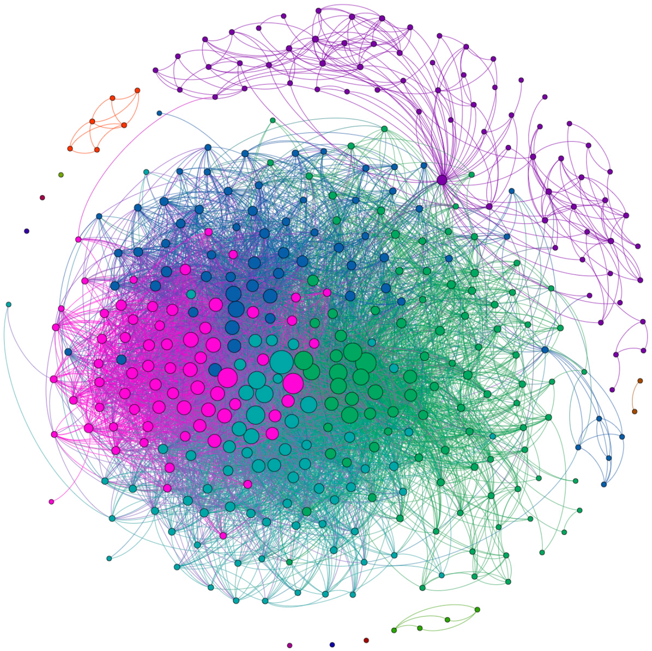
Clustering



INF 553:
Foundations and Applications of Data Mining

Roadmap

- Problem, types, and distance functions
- **Algorithms:**
- Hierarchical clustering
- Point assignment
  - K-means
  - BFR: extend k-means to handle large data set
  - CURE

With slide contributions from J. Leskovec, Anand Rajaraman, Jeffrey D. Ullman

Learning Approaches

Supervised Learning

- The **training data** is **annotated** with information to help the learning system
  - Eg. the class for each instance

Unsupervised Learning

- The training data is **not annotated** with any extra information to help the learning system
  - Eg. clustering of data

Semi-Supervised Learning

High Dimensional Data

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	SVM	Recommender systems
Clustering	Network Analysis	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

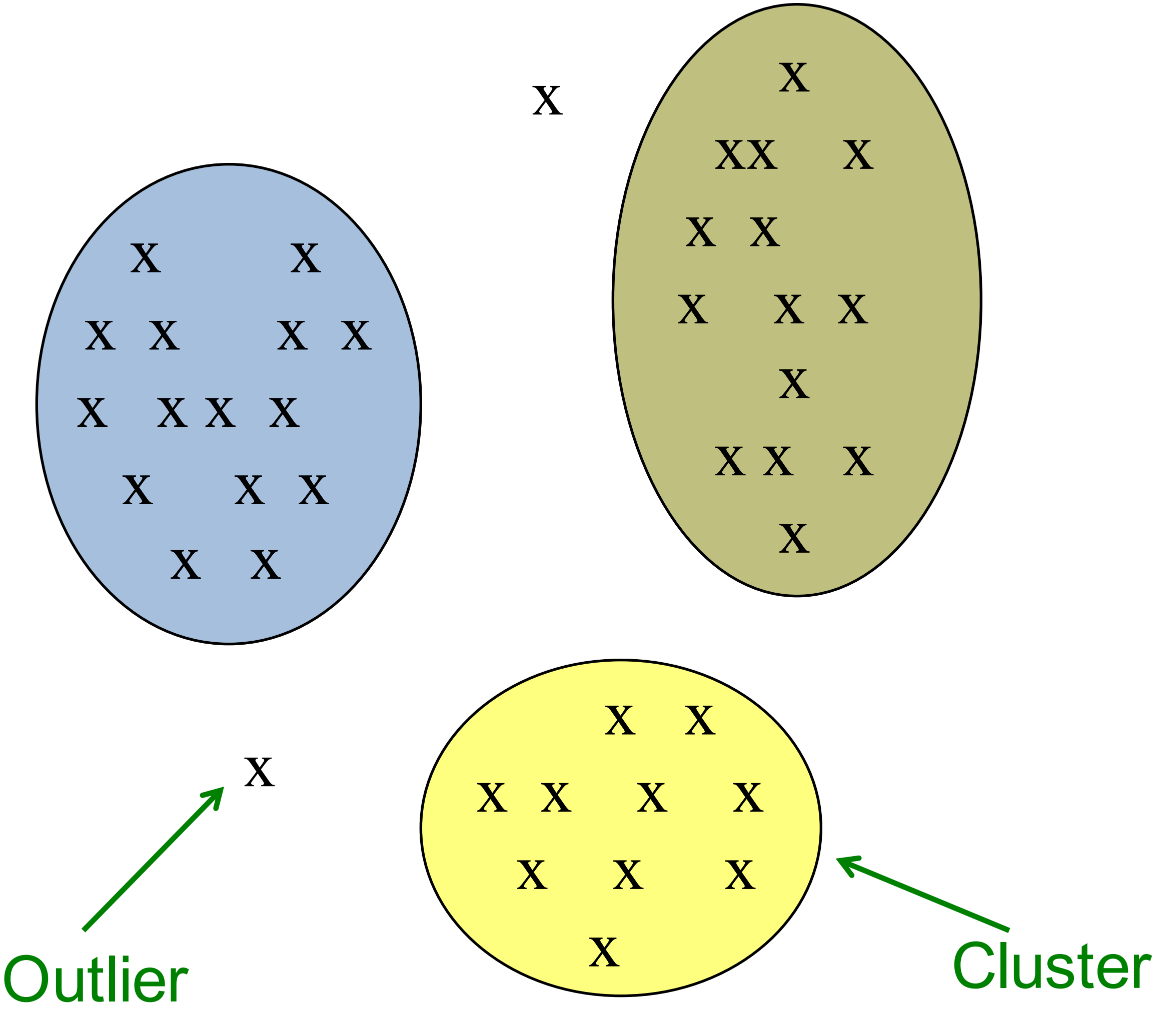
High Dimensional Data

- Given a cloud of data points we want to understand its **structure**
- Group points into **"clusters"** according to some **distance measure**.

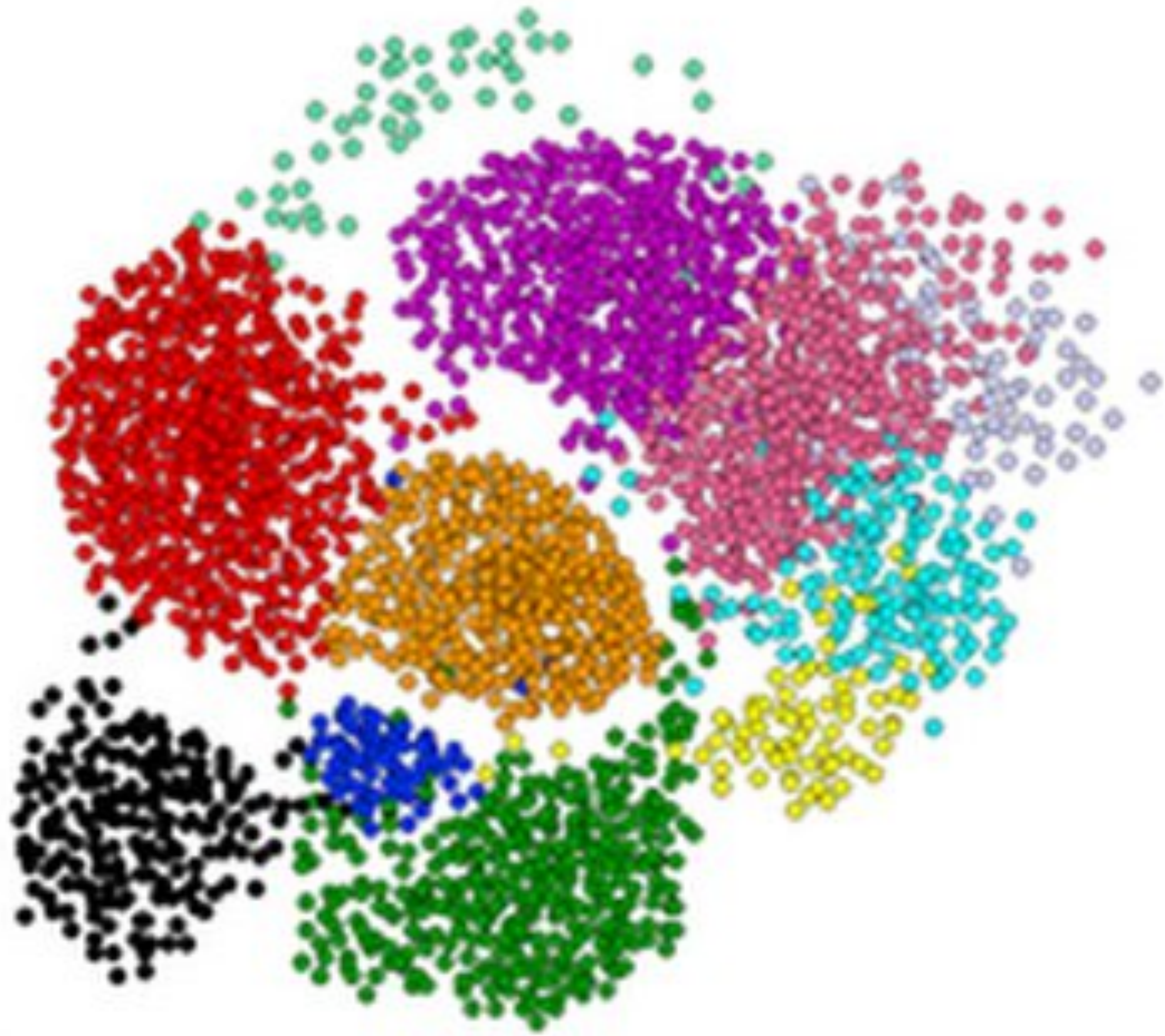
The Problem of Clustering

- Given a **set of points** that belong to some **space**, with a notion of **distance** between points
- **Group the points** into some number of ***clusters***, so that:
  - Members of **a cluster** are **close/similar** to each other
  - Members of **different clusters** are **dissimilar**
- **Usually:**
  - Points are in a high-dimensional space
  - Similarity is defined using a **distance measure**
    - Euclidean, Cosine, Jaccard, edit distance, …

Example: Clusters & Outliers



Clustering is a hard problem!



Why is it hard?

- Clustering in **two dimensions** looks easy
- Clustering **small amounts** of data looks easy
  - And in most cases, looks are not deceiving

- BUT: Many applications involve not 2, but 10 or 10,000 dimensions
- **High-dimensional spaces look different:** Almost all pairs of points are at about the same distance
  - **Curse of dimensionality**: In high dimensions, almost all pairs of points are **equally far away** from one another; almost any **two vectors are orthogonal**.

Clustering Sky Objects: SkyCat

- A catalog of 2 billion "sky objects" represents objects by their radiation in 7 dimensions (frequency bands)
- Problem: Cluster into similar objects, e.g., galaxies, nearby stars, quasars, etc.
- Sloan Digital Sky Survey is a newer, better version of this

Clustering Problem: Songs

- **Intuitively: Music divides into categories, and customers prefer a few categories**
  - But what are categories really?

- Represent a song by a **set of customers** who bought it

- **Similar songs** have similar sets of customers, and vice-versa.

Clustering Problem: Songs

Space of all songs:

- Think of a space with one dimension for each customer
  - Values in a dimension may be 0 or 1 only
  - A **song is a point** in this space $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i^{th} customer bought the song

- For Amazon, the dimension is tens of millions

- **Task:** Find clusters of similar songs.

Clustering Problem: Documents

Finding topics:

- Represent a document by a **vector** $(x_1, x_2,\dots, x_k)$, where $x_i = 1$ iff the i^{th} word (in some order) appears in the document
  - It actually doesn't matter if k is infinite; i.e., we don't limit the set of words
- Representing documents by sets of shingles in another example
- **Documents with similar sets of words or same shingles may be about the same topic.**

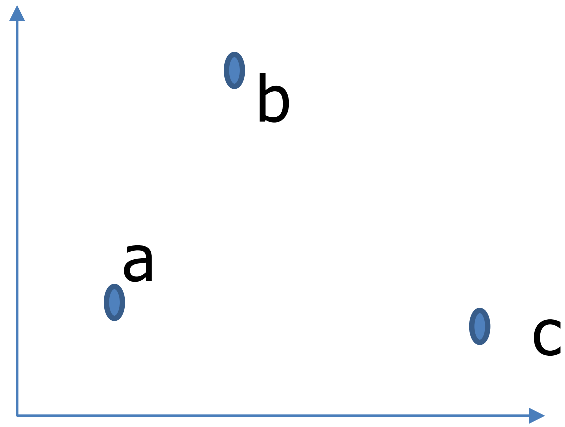
Jaccard, Euclidean and Cosine Distance

- Different ways of representing documents (**as sets of words or shingles)** lead to different distance measures
- Document = **set** of words
  - **Jaccard distance**
- Document = **point** in space of words
  - $(x_1, x_2, \ldots, x_n)$, where $x_i=1$ iff word *i* appears in doc
  - **Euclidean distance**
- Document = **vector** in space of words
  - Vector from origin to $(x_1, x_2, \ldots, x_n)$
  - **Cosine distance.**

Euclidean Distance

- Measures distance of two points in Euclidean space

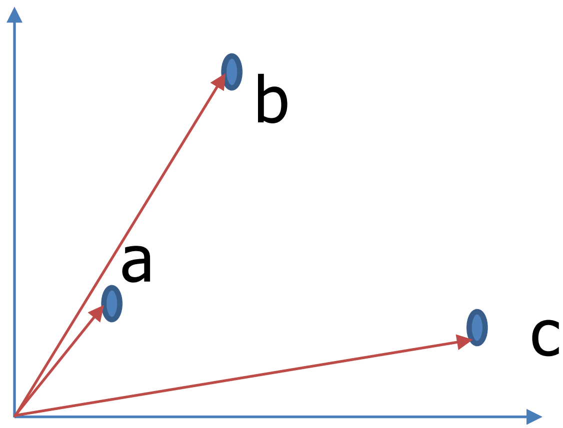
$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$



Cosine Distance

- Similarity = Cosine of angle btw vectors: A & B
  - Numerator is the **dot product** of vectors A and B
  - Denominator is the product of the **Euclidean distance** of each vector from the origin

    (length of the vector)



- distance = 1- Cosine(A, B)

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Example-Cosine Distance

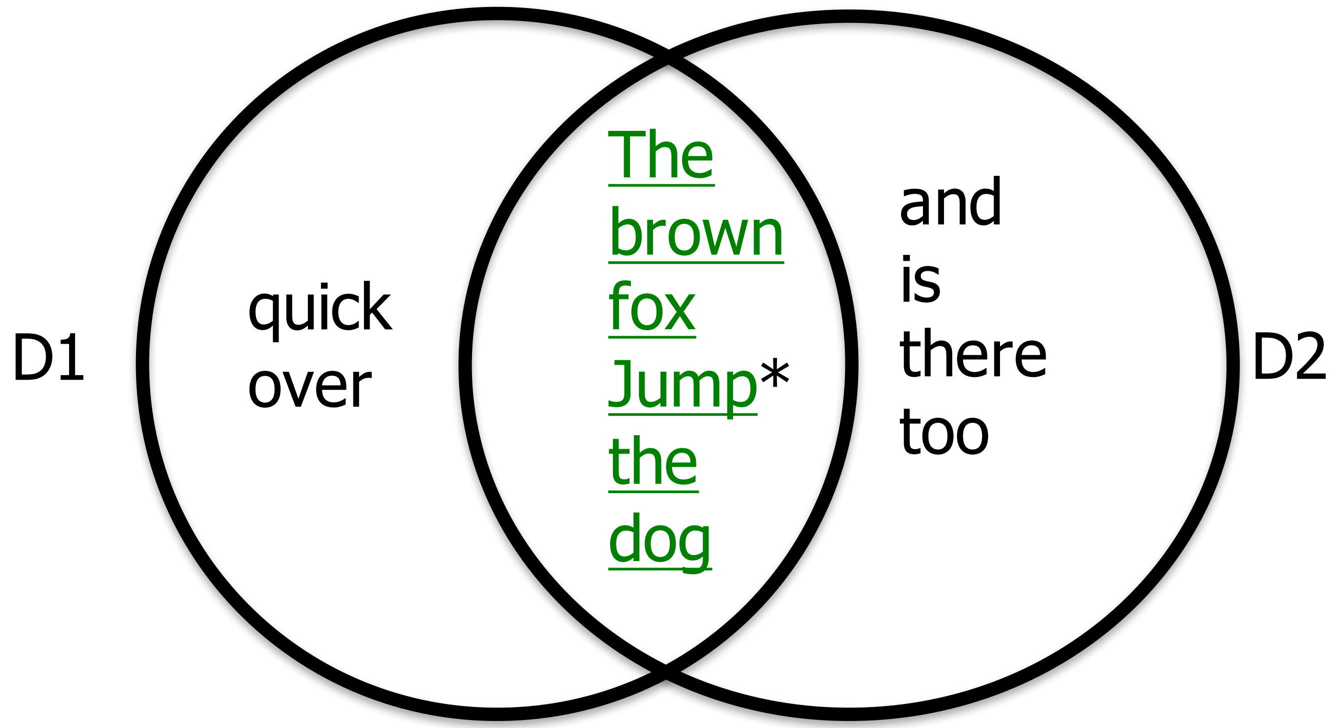
- Think of a point as a vector from the origin (0,0,...,0) to its location
- Two points'vectors make an angle, whose cosine is the normalized dot product of the vectors: p1.p2/|p2||p1|
- **Example**: p1=00**11**; p2=100**11**
- p1.p2=2
- |p2|=|p1|=√*3*
- Cos(θ)=2/3; θ is about 48 degrees.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

For Sets, Jaccard *Similarity*

D1: The quick brown fox jumped over the dog.

D2: The brown fox jumps and the dog is there too.



Jaccard = $|S \wedge T| \;/\; |S \vee T| = 6/12 = 0.5$

Jaccard ***<u>Distance</u>***

1 – (Jaccard Similarity)

Hamming Distance

- For two bit vectors, distance btw x and y =
  - # of corresponding bits that differ

- x = **1**01**1**01, y = **1**1**1**10
  - Hamming(x, y) = 3

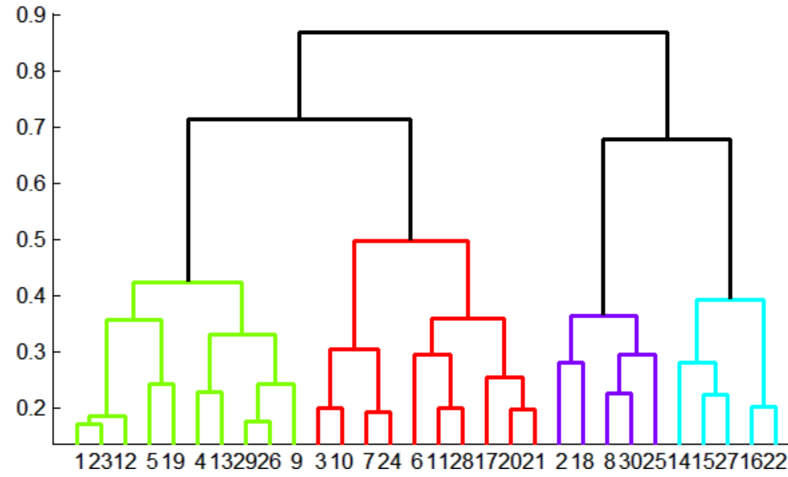
Edit Distance

- Use when **points are strings**
- Distance between strings $x = x_1x_2 \ldots x_n$ and $y=y_1y_2 \ldots y_m$
- **Smallest number of insertions and deletions of single characters that will convert x to y**
- Example:
  - x = abcde and y = acfdeg
  - To convert x to y: Delete b, insert f after c, insert g after e
  - Edit distance = 3

Roadmap

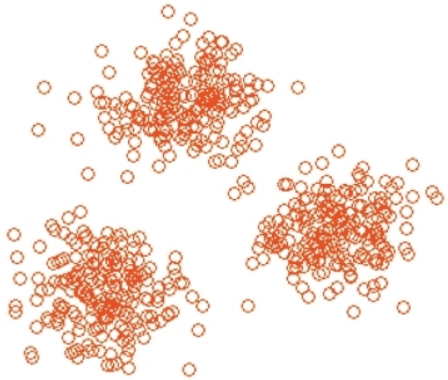
- Problem, types and distance functions
- Hierarchical clustering
- Point assignment
  - K-means
  - BFR
  - CURE

Overview: Methods of Clustering



- **Hierarchical:**
  - **Agglomerative** (bottom up):
    - Initially, each point is a cluster
    - Repeatedly combine the two "nearest" clusters into one
  - **Divisive** (top down):
    - Start with one cluster and recursively split it

- **Point assignment:**
  - Maintain a set of clusters
  - Points belong to "nearest" cluster.

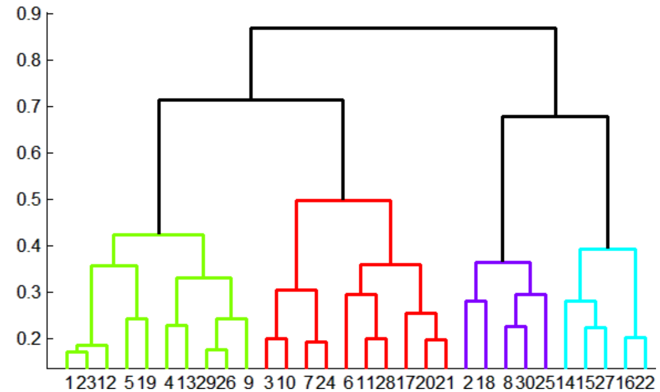


Clustering Strategies (cont.)

Also distinguish clustering algorithms by:

- Whether the algorithm assumes a **Euclidean space** or uses some other **distance measure**
- Whether the algorithm assumes data are **small enough** to fit into memory.

Hierarchical Clustering (Agglomerative)



- **Key operation:**
  Repeatedly combine two nearest clusters

- **Three important questions:**
  - **1)** How do you represent a **cluster of more than one** point?
  - **2)** How do you determine the **"nearness"** of clusters?
  - **3)** When to **stop combining** clusters?

Hierarchical Clustering

- **Key operation: Repeatedly combine two nearest clusters**
- **(1) How to represent a cluster of many points?**
  - **Key problem:** As you merge clusters, how do you represent the "location" of each cluster, to tell which pair of clusters is closest?
- **Euclidean case:** each cluster has a ***centroid*** **= average of its (data) points**
- **(2) How to determine "nearness" of clusters?**
  - **Measure cluster distances by distances of centroids.**

Hierarchical Clustering

- Initially, a point is in a cluster by itself

How to pick and combine efficiently?

When to stop?

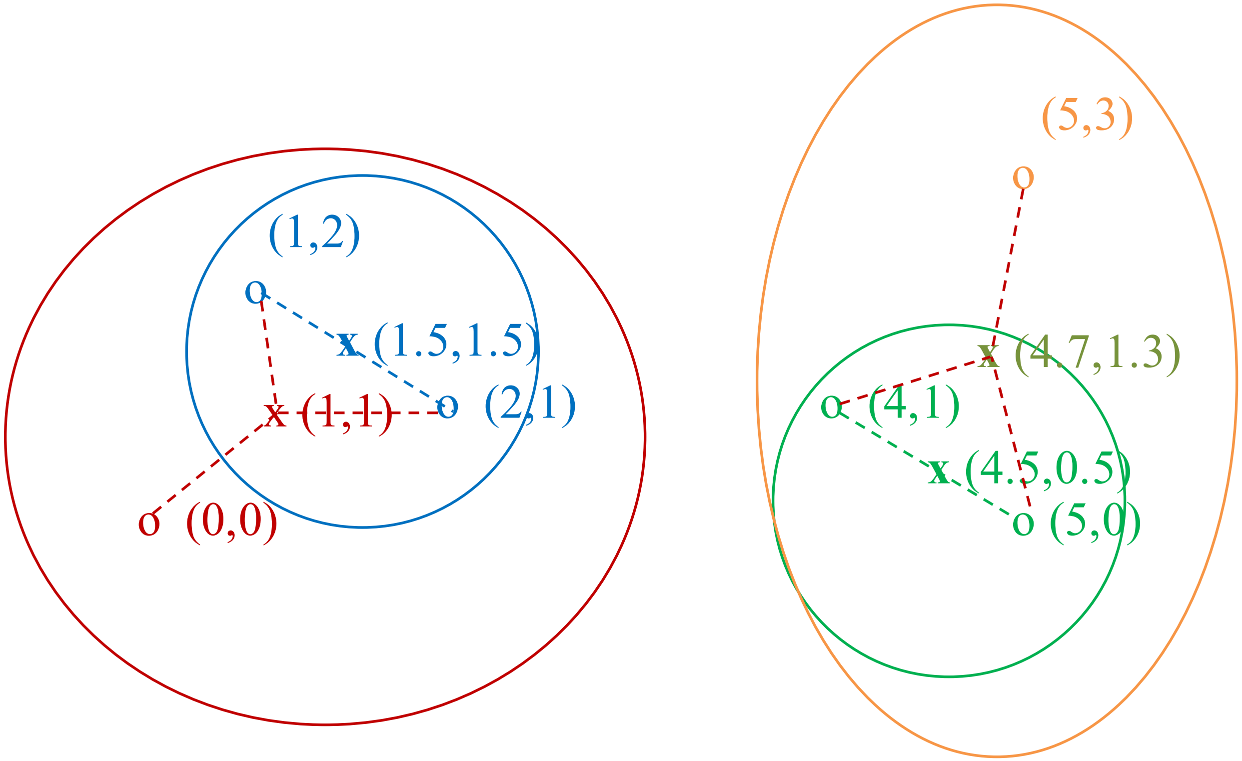
```
WHILE it is not time to stop DO
    pick the best two clusters to merge;
    combine those two clusters into one cluster;
END;
```

How to measure cluster distance?

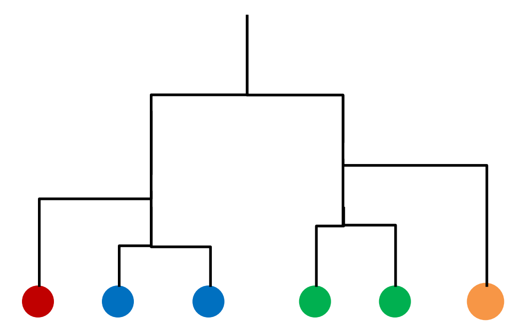
Hierarchical Agglomerative (Bottom-Up) Clustering Algorithm

- **First assume the space is Euclidean**
- Represent cluster by its **centroid** or average of points in the cluster
- **Merging rule:**
  - the distance between two clusters is distance between their centroids
  - dist(C1, C2) = distance of their centroids
    - Coordinate of centroid = avg of that of all points in the cluster
  - C1: {(1, 2), (2, 2)}
    - Centroid = (1.5, 2)
- **merge two clusters at shortest distance.**

Example: Hierarchical clustering



Data:
o … data point
x … centroid



Dendrogram

When to Stop Clustering Process

- Several approaches:

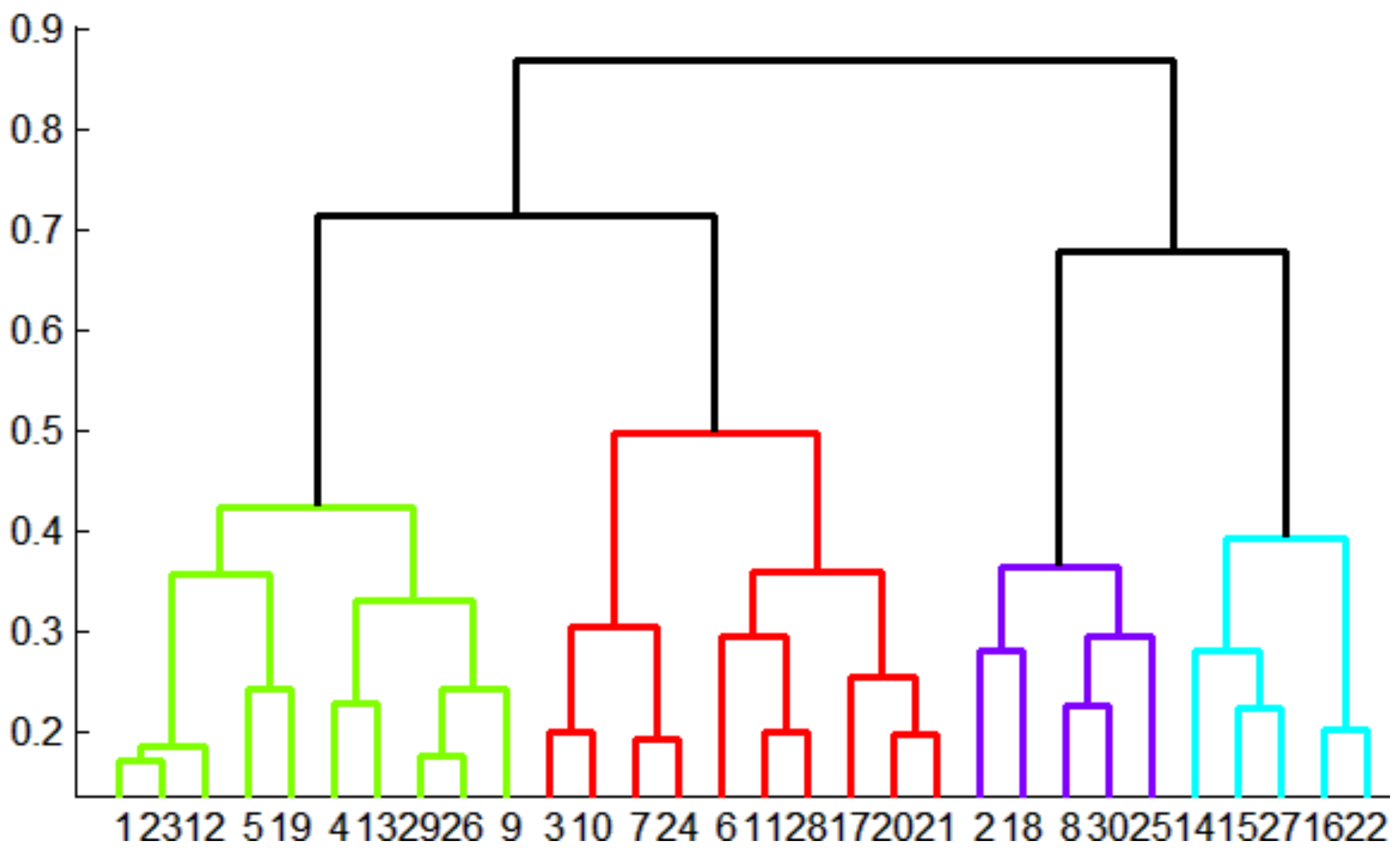
1. **May know how many clusters** there are in the data
   - Have been told or some **intuitive number of clusters**
2. Stop combining **when best combination of existing clusters produces a cluster that is inadequate**
   - E.g., Average distance between centroid and its points should be below some limit.

Rules for Controlling Hierarchical Clustering: Picking Clusters to Merge

1. Find pair with **smallest distance between centroids** (previous)
2. Take distance between two clusters **as minimum of distances between any two points, one chosen from each cluster**
   - **Merge two clusters with minimum distance**
   - May result in entirely different clustering from distance-of-centroids
3. Take distance between two clusters to be **average distance of all pairs of points, one from each cluster**
   - **Merge two clusters with smallest average distance**
4. **Radius of cluster = maximum distance between all points and the centroid**
   - **Combine two clusters whose resulting cluster has lowest radius**
5. **Diameter of cluster = maximum distance between any two points of the cluster**
   - **Merge the clusters whose resulting cluster has smallest diameter.**

Dendrogram

- Can obtain k clusters from result for desired k
  - k can be any value between 1 and n



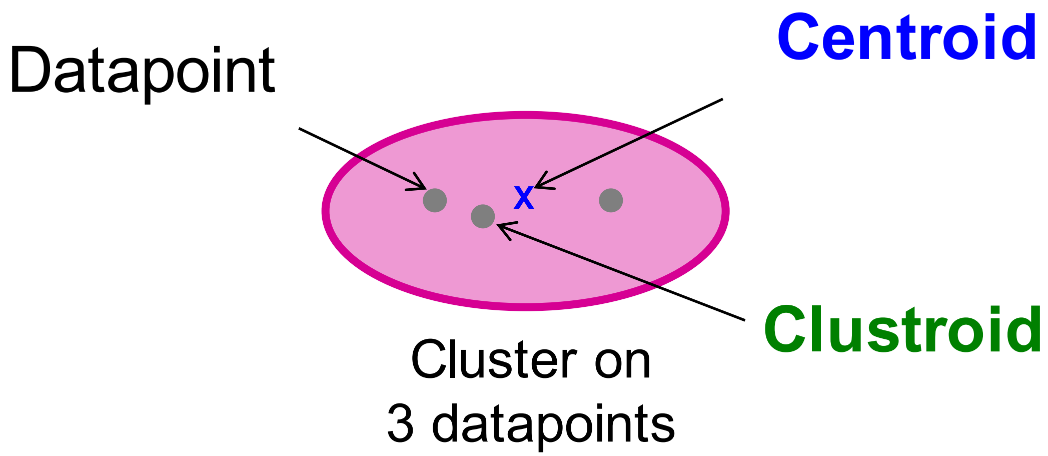
And in the Non-Euclidean Case?

What about the Non-Euclidean case?

- The only "locations" we can talk about are the points themselves
  - i.e., there is no "average" of two points

- **Approach 1:**
  - **(1) How to represent a cluster of many points?**
    - ***clustroid*** = (data)point "***closest***" to other points
  - **(2) How do you determine the "nearness" of clusters?**
    - Treat clustroid as if it were centroid, when computing inter-cluster distances.

Clustroid

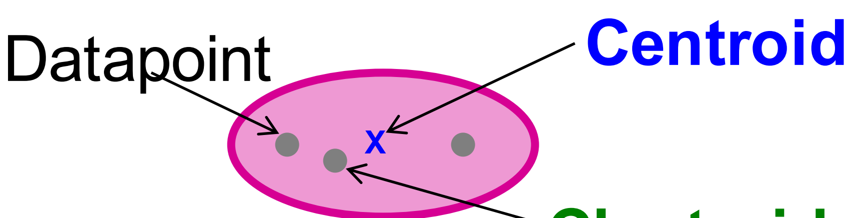


Centroid is the avg. of all (data)points in the cluster. This means centroid is an **“artificial” point**

Clustroid is an **existing** (data)point that is “closest” to all other points in the cluster.

"Closest" Point?

- **(1) How to represent a cluster of many points?**
  clustroid = point "***closest***" to other points
- **Possible meanings of "closest":**
  - **Smallest maximum distance** to other points
  - **Smallest average distance** to other points in the cluster
  - **Smallest sum of squares of distances** to other points
    - For distance metric ***d*** clustroid ***c*** of cluster ***C*** is: $\min_{c} \sum_{x \in C} d(x,c)^2$



Cluster on
3 datapoints

Defining "Nearness" of Clusters

- **(2) How do you determine the "nearness" of clusters?**
  - **Approach 1:**
    Intercluster distance = minimum of the distances between any two points, one from each cluster
  - **Approach 2:**
    Pick a notion of "**cohesion**" of clusters, *e.g.*, maximum distance from the **clustroid**
    - Merge clusters whose *union* is most cohesive.

Termination condition

- **(3) When to stop merging**
  - **Approach 1:** Pick a number **k** upfront, and stop when we have **k** clusters
    - Makes sense when we know that the data naturally falls into k classes
  - **Approach 2:** Stop when the next merge would create a cluster with low "cohesion"
    - i.e, a "bad" cluster.

Cohesion

- **Merge clusters whose *union* is most cohesive**
- **Approach 3.1: Diameter** of the merged cluster = maximum distance between points in the cluster
- **Approach 3.2: Radius**= maximum distance of a point from **centroid** (or clustroid)
- **Approach 3.3:** Use a **density-based approach**
  - Density = number of points per unit volume
  - E.g., divide number of points in cluster by **diameter or radius of the cluster**
  - Perhaps use a power of the radius (e.g., square or cube).

Example

- Consider a cluster of 4 points:
  - abcd, aecdb, abecb, ecdab

- Their edit distances:

	aecdb	abecb	ecdab
abcd	3	3	5
aecdb		2	2
abecb			4

Determine Clusteroid

- aecdb will be chosen as clusteroid
  - Located in "center" judged by all 3 measures

	aecdb	abecb	ecdab
abcd	3	3	5
aecdb		2	2
abecb			4

Point	**Sum**	**Sum-sq**	**Max**
abcd	11	43	5
aecdb	**7**	**17**	**3**
abecb	9	29	4
ecdab	11	45	5

Complexity of Hierarchical Clustering

- n data points
- At most n – 1 step of merging

- Naive implementation, e.g., storing pairwise cluster distances in a matrix

	C1	**C2**	**C3**	**C4**
C1	0	2	3	2
C2		0	4	5
C3			0	3
C4				0

Implementation

- Naïve implementation of hierarchical clustering:
  - At each step, compute pairwise distance between all pairs of clusters, then merge.

- Initially, $O(n^2)$ for creating matrix and finding pair with minimum distance
- Subsequent merge,

=> Overall complexity: $O(n^3)$

Improved Version

- Use priority queue (e.g., heap-based) instead of matrix

1. Compute pairwise dist. of all points: $O(n^2)$
2. Build priority queue (time linear to size of queue), so: $O(n^2)$
3. Each merge:
   a) Remove entries for old clusters: 2n* O(log(n))
   b) Add entries for new cluster: n * O(log(n))

=> Overall complexity: $O(n^2 \log(n))$

Implementation

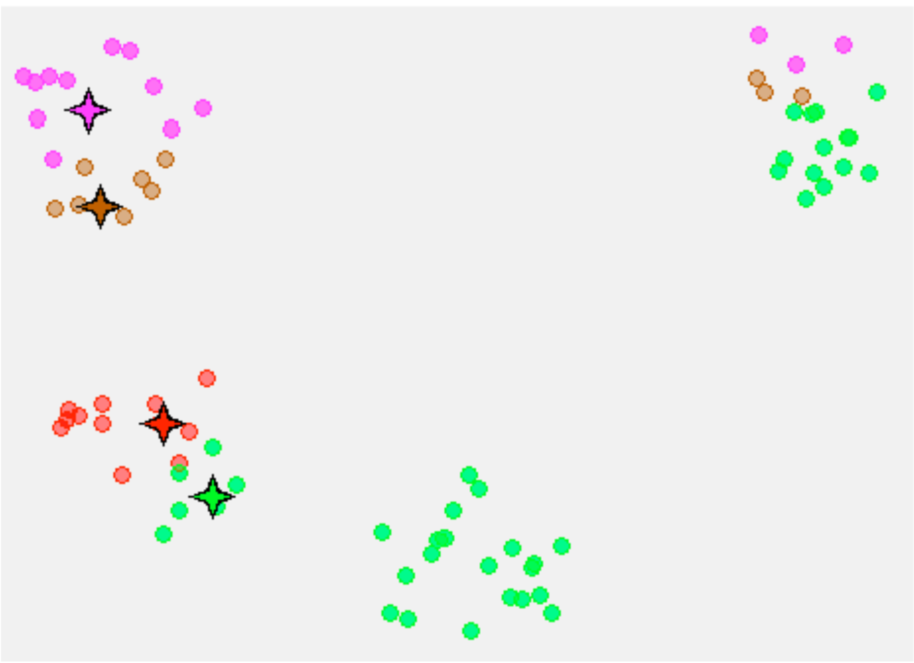
- **Naïve implementation of hierarchical clustering:**
  - At each step, compute pairwise distances between all pairs of clusters, then merge
    - Initially, $O(n^2)$ for creating matrix and finding pair with minimum distance
    - Subsequent merge,

      => Overall complexity: $O(n^3)$
- Careful implementation using **priority queue** can reduce time to $O(N^2 \log N)$
  - **Still too expensive for really big datasets that do not fit in memory.**

Roadmap

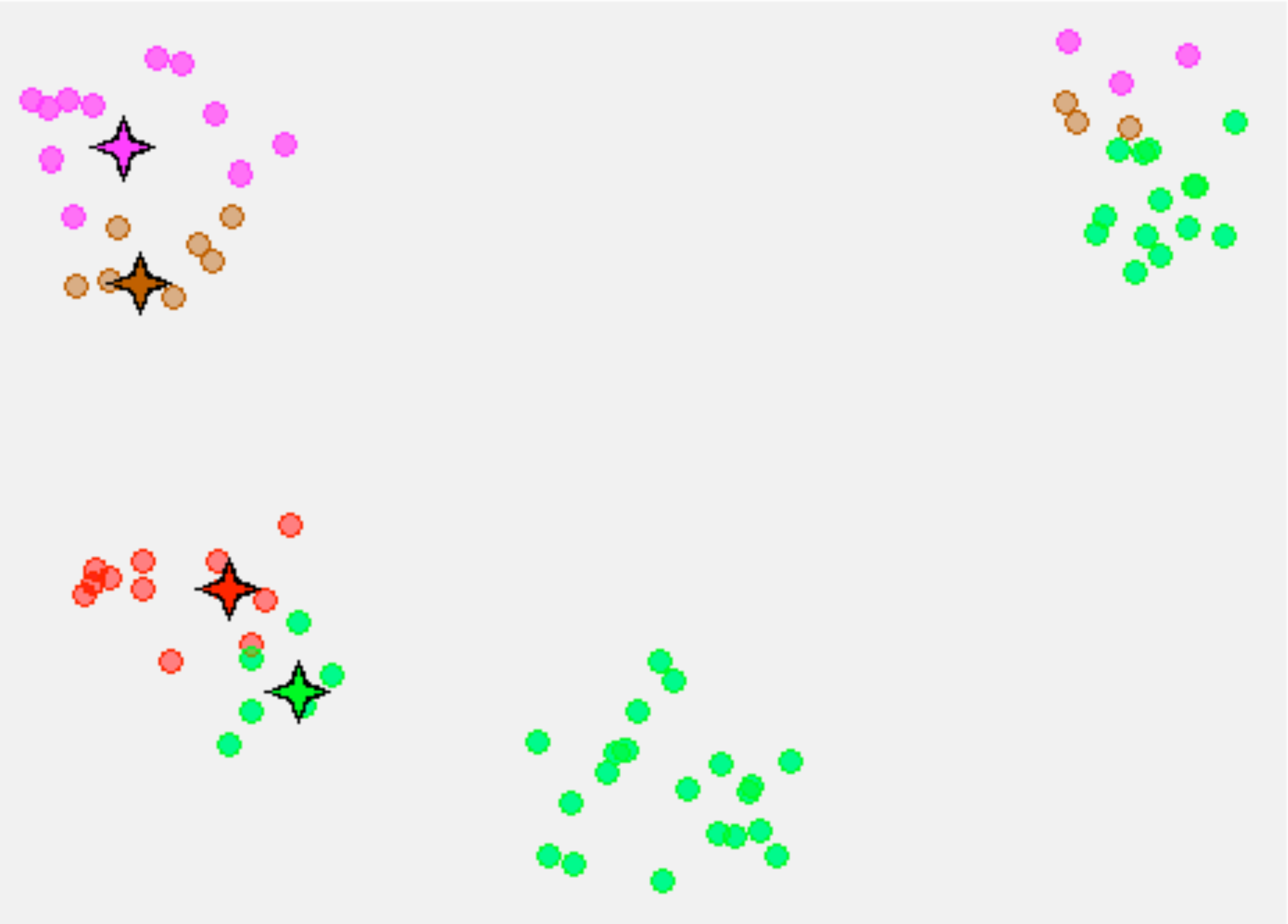
- Problem, types and distance functions
- Hierarchical clustering
- Point assignment
  - K-means
  - BFR
  - CURE

K-Means Clustering Algorithm



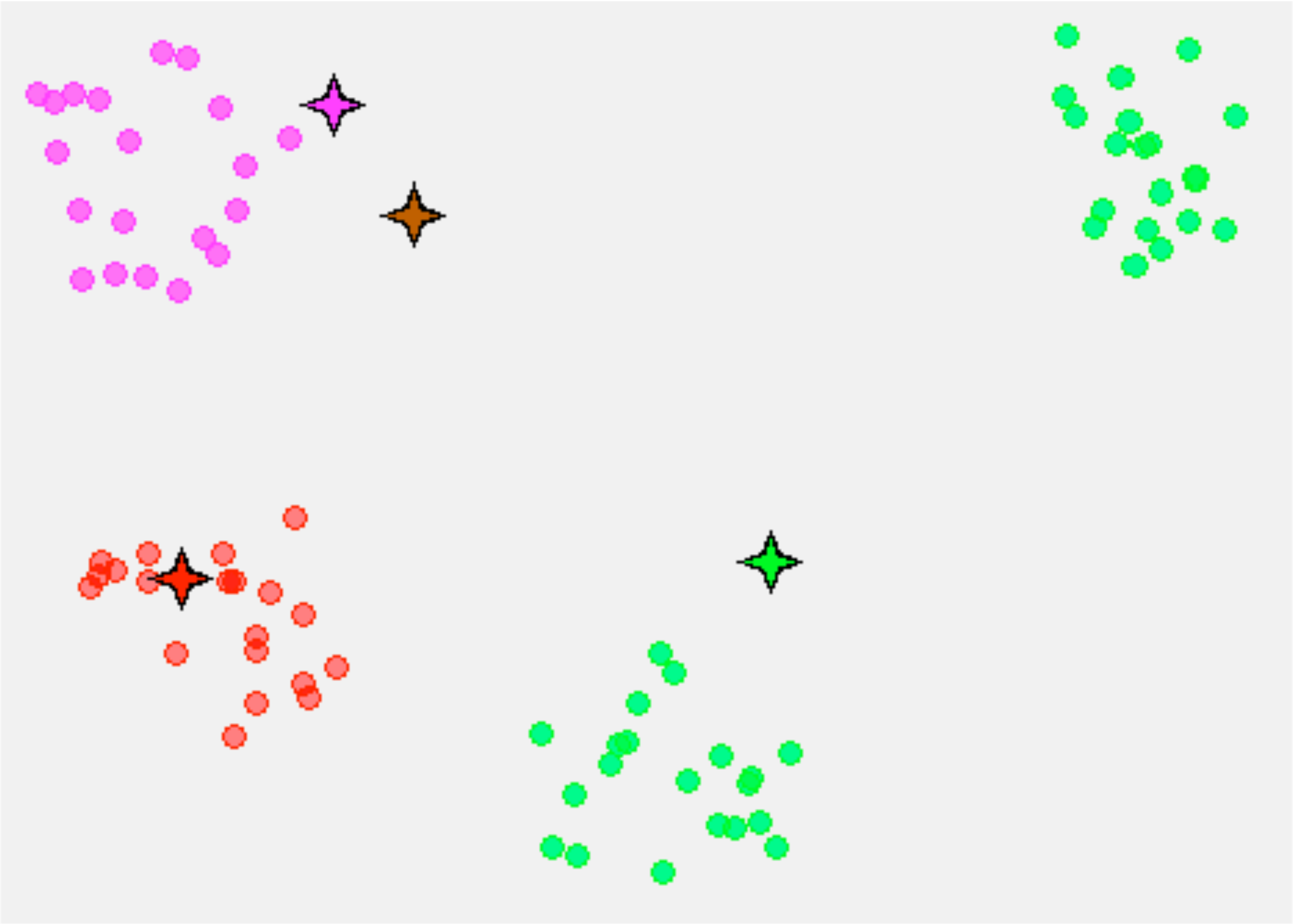
- User specifies a target number of clusters (k)
- Place randomly k cluster centers
- For each datapoint, attach it to the nearest cluster center
- For each center, find the centroid of all the datapoints attached to it
- Turn the centroids into cluster centers
- Repeat until the sum of all the datapoint distances to the cluster centers is minimized

K-Means Clustering (1)



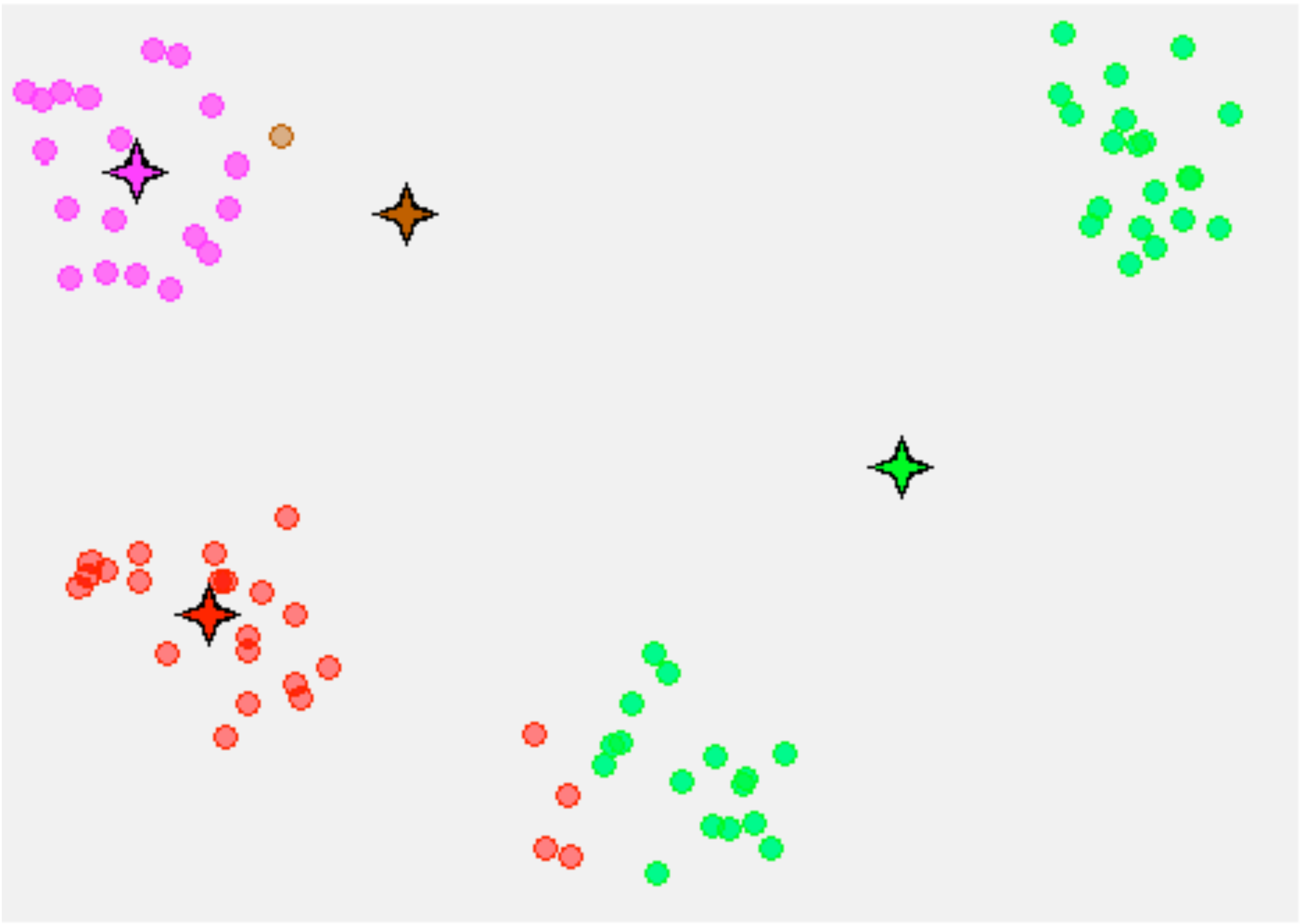
https://commons.wikimedia.org/wiki/File:K-means_convergence_to_a_local_minimum.png

K-Means Clustering (2)



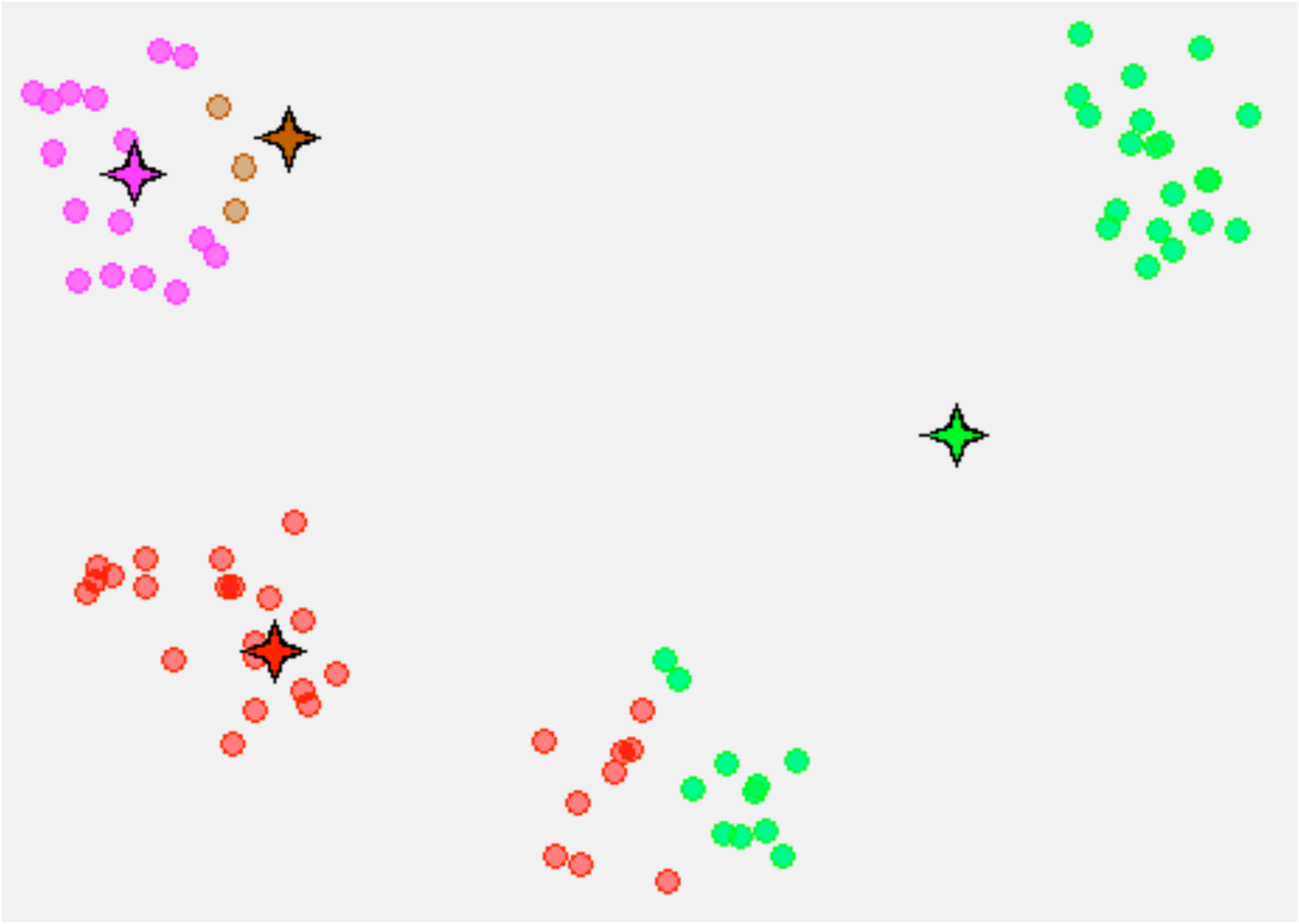
https://commons.wikimedia.org/wiki/File:K-means_convergence_to_a_local_minimum.png

K-Means Clustering (3)



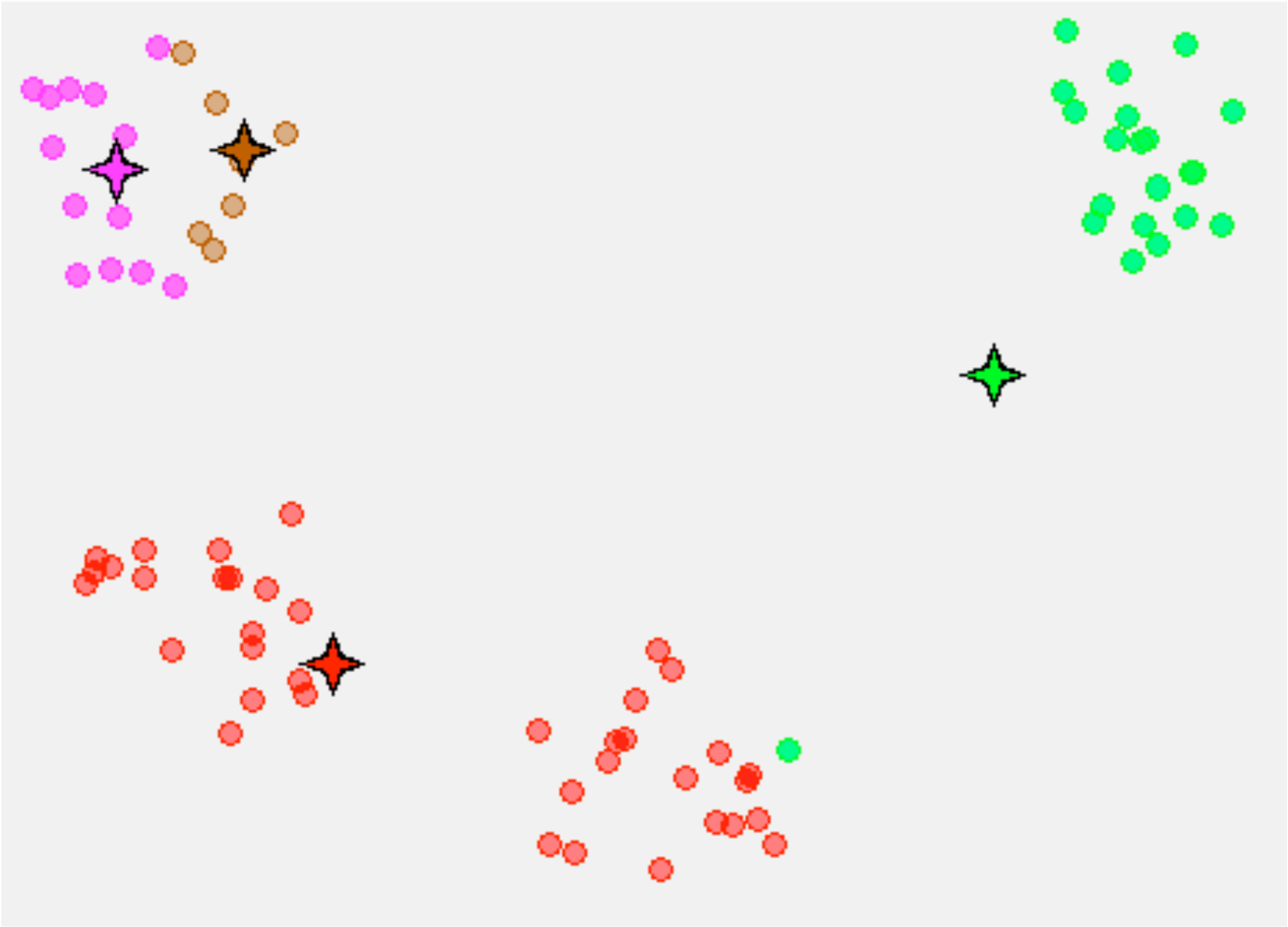
https://commons.wikimedia.org/wiki/File:K-means_convergence_to_a_local_minimum.png

K-Means Clustering (4)



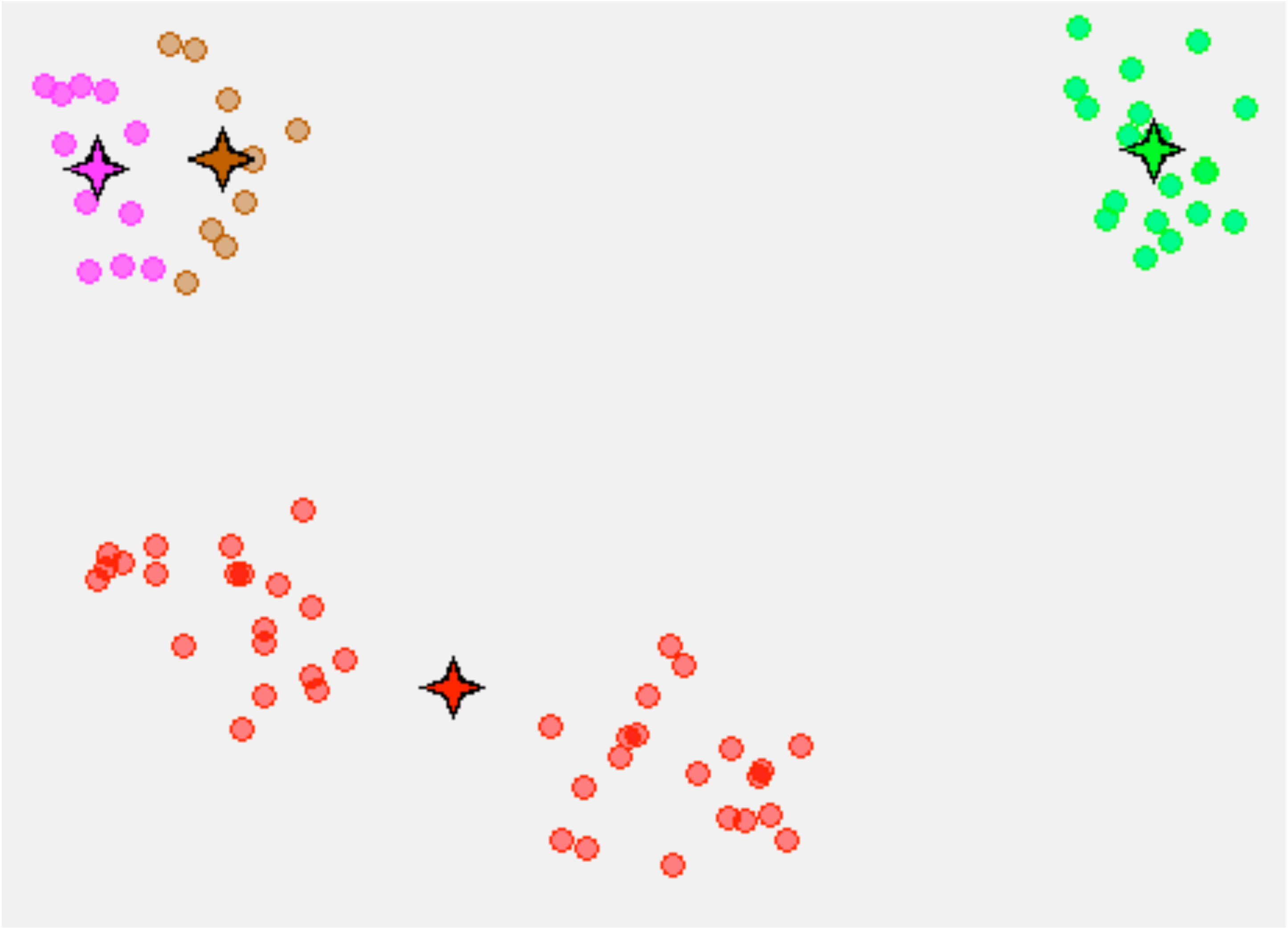
https://commons.wikimedia.org/wiki/File:K-means_convergence_to_a_local_minimum.png

K-Means Clustering (5)



https://commons.wikimedia.org/wiki/File:K-means_convergence_to_a_local_minimum.png

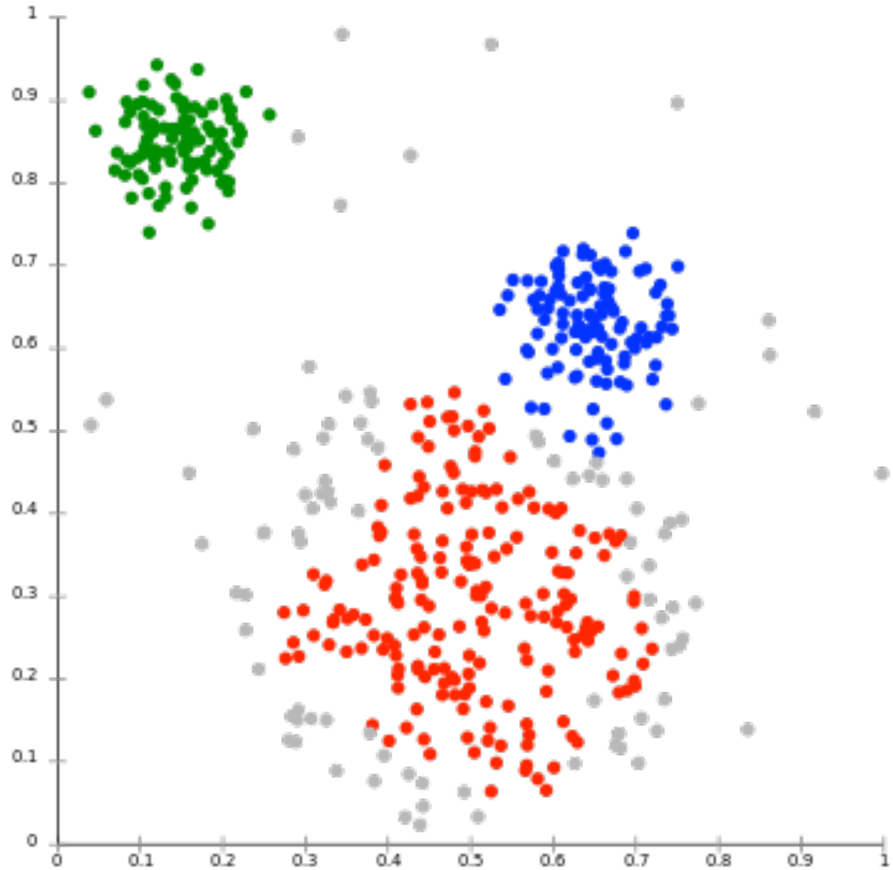
K-Means Clustering (6)



https://commons.wikimedia.org/wiki/File:K-means_convergence_to_a_local_minimum.png

Clustering Methods

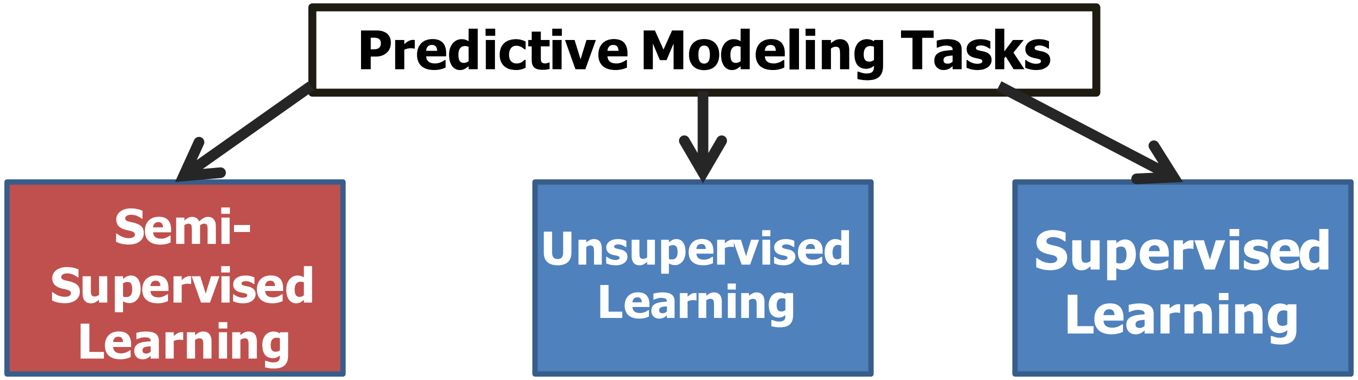
- Hierarchical clustering
  - Attach datapoints to root points
- K-Means clustering
  - Centroid-based
- Density-based methods
  - Clusters contain a minimal number of datapoints



https://commons.wikimedia.org/wiki/File:DBSCAN-Gaussian-data.svg

Summary of Concepts in Clustering

Clustering vs. Classification



- Classification is **supervised**
  - class labels are provided;
  - learn a **classifier to predict class labels** of novel/unseen data
- Clustering is **unsupervised** or **semi-supervised**;
  - No class label is give
  - Understand the structure underlying your data.

Summary of Concepts in Clustering

- Clustering

- **Algorithms:**
- Hierarchical clustering
  - centroid
  - clustroid
  - Dendrogram
- Point assignment
  - K-means: cluster centers, centroids
  - BFR: extend k-means to handle large data set
  - CURE.