McCOMBS SCHOOL OF BUSINESS

Salem Center for Policy

Unsupervised learning

David Puelz

October 14, 2021

Outline

Clustering

Principal Components Analysis

Clustering

1. Introduction to clustering
2. K-means
3. Implementing K-means: some practical details
4. Hierarchical clustering

Introduction to clustering

You've seen a lot of models for $p(y \mid x)$. This is supervised learning (knowing outcomes y = "supervision").

The next few topics are all about models for x alone.

- Clustering means dividing data points into categories which are not defined in advance.

- It's different from classification: dividing data points into categories which *are* defined in advance, and for which we have explicit labels.

Clustering: Toy example

- The horizontal and vertical locations of each point give its location $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ in feature space.

- From these locations, we impute the cluster labels: $\gamma_i = k$ if point i is in cluster $k \in \{1, \ldots, K\}$.

Criteria for clustering

Clusters should partition the data:

- Partition = mutually exclusive, collectively exhaustive set of categories (each data point is in one and only one cluster).

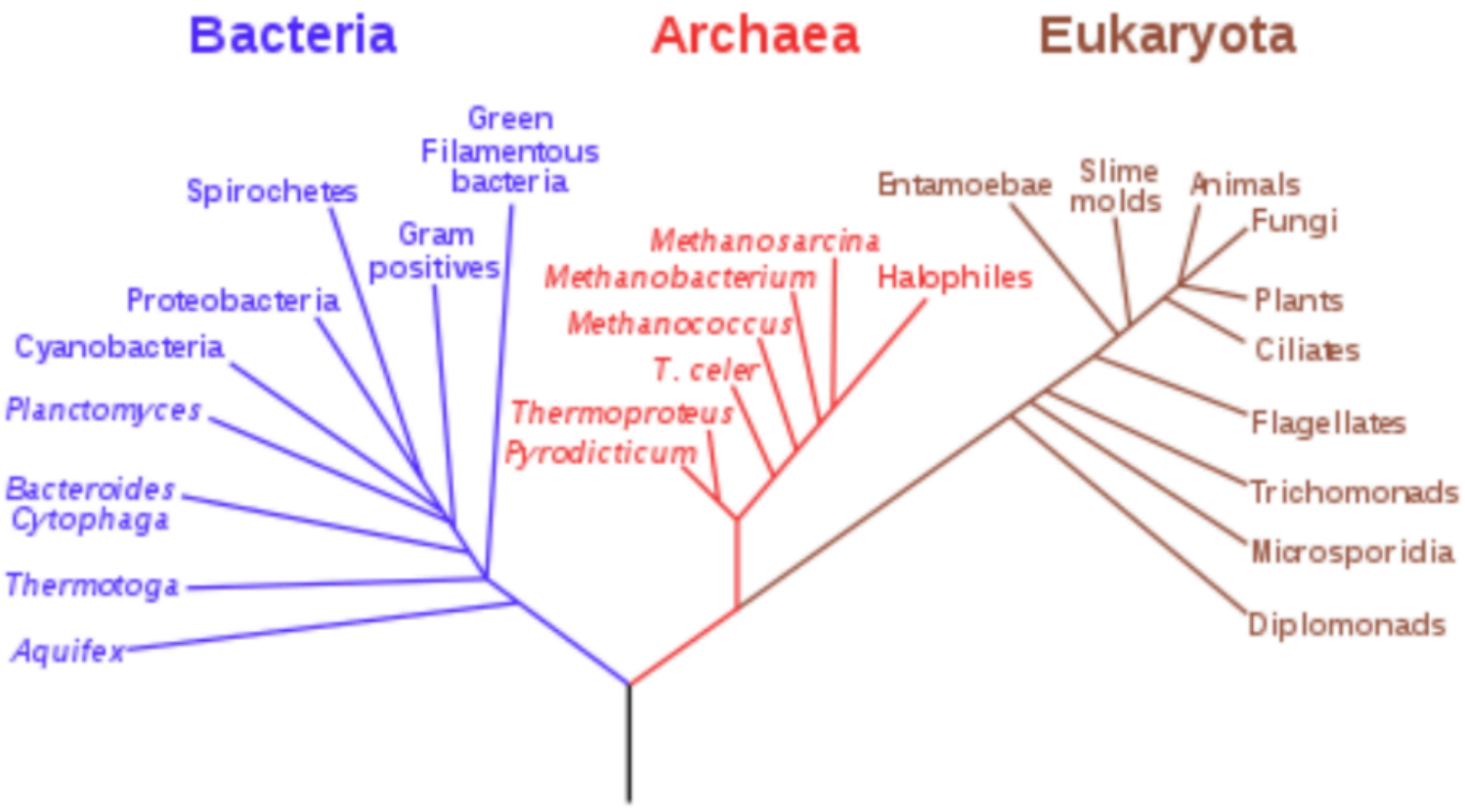
- No "mixed membership." If you encounter a Chiweenie, you need a new cluster!

Criteria for clustering

Data points within the same cluster should be close/similar, and data points in different clusters should be far apart/dissimilar.

Criteria for clustering

Clusters should (ideally) be balanced. A sensible clustering of living creatures:



Criteria for clustering

A less sensible clustering of living creatures:

1

2

All other living creatures

3

How can we cluster without labels?

There are many algorithms which do this, i.e. try to find clusters that are homogenous, well separated, balanced, etc.

Key fact: we need to know about **distances** to quantify similarity (within a cluster) and difference (between clusters).

Generically, if x_i and x_j are two data points, we let $d(x_i, x_j)$ denote the distance between them.

Distance functions

Properties of distance functions:

1. $d(x_i, x_j) \geq 0$

2. $d(x_i, x_j) = 0 \iff x_i = x_j$.

3. $d(x_i, x_j) = d(x_j, x_i)$ (symmetry)

4. $d(x_i, x_j) \leq d(x_i, x_k) + d(x_j, x_k)$ (triangle inequality, i.e. "If you want to get from Austin to Houston, don't go through Dallas!")

NB: in math, distance functions are called "metrics."

Distance functions

Euclidean (ℓ^2):

$$d(x_i, x_j) = \left[\sum_{d=1}^{D}(x_{id} - x_{jd})^2\right]^{1/2}$$

(just the Pythagorean theorem!)

Manhattan (ℓ^1):

$$d(x_i, x_j) = \sum_{d=1}^{D}|x_{id} - x_{jd}|$$

(also called "taxicab" distance)

Distance functions

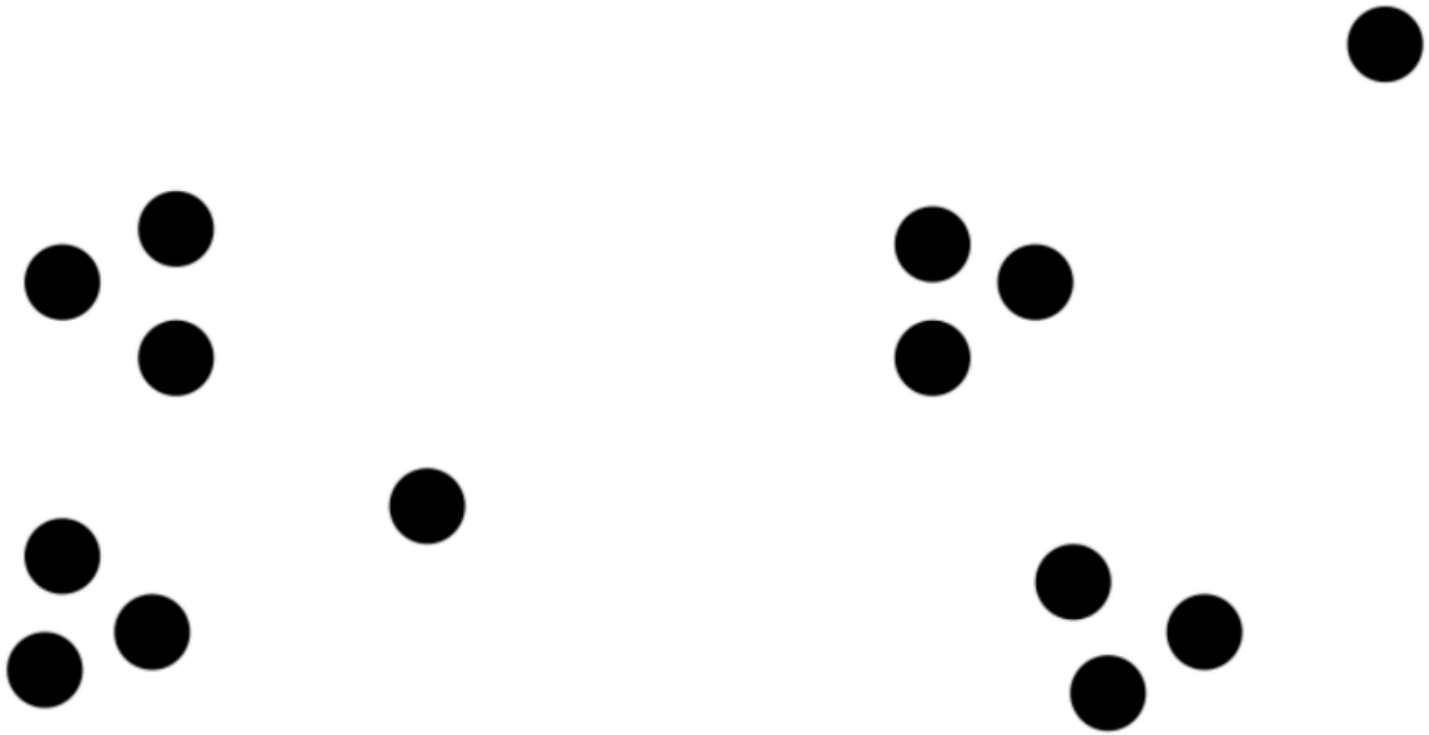
The x points can be arbitrary objects in potentially crazy spaces:

- playlists on Spotify
- phrase counts for books
- economic indicators
- DNA sequences ("Hamming distance")
- etc.

As long as we can measure the distance between any two points, we can cluster them!

A few miscellaneous notes

Clusters can be ambiguous:



How many clusters do you see?

A few miscellaneous notes

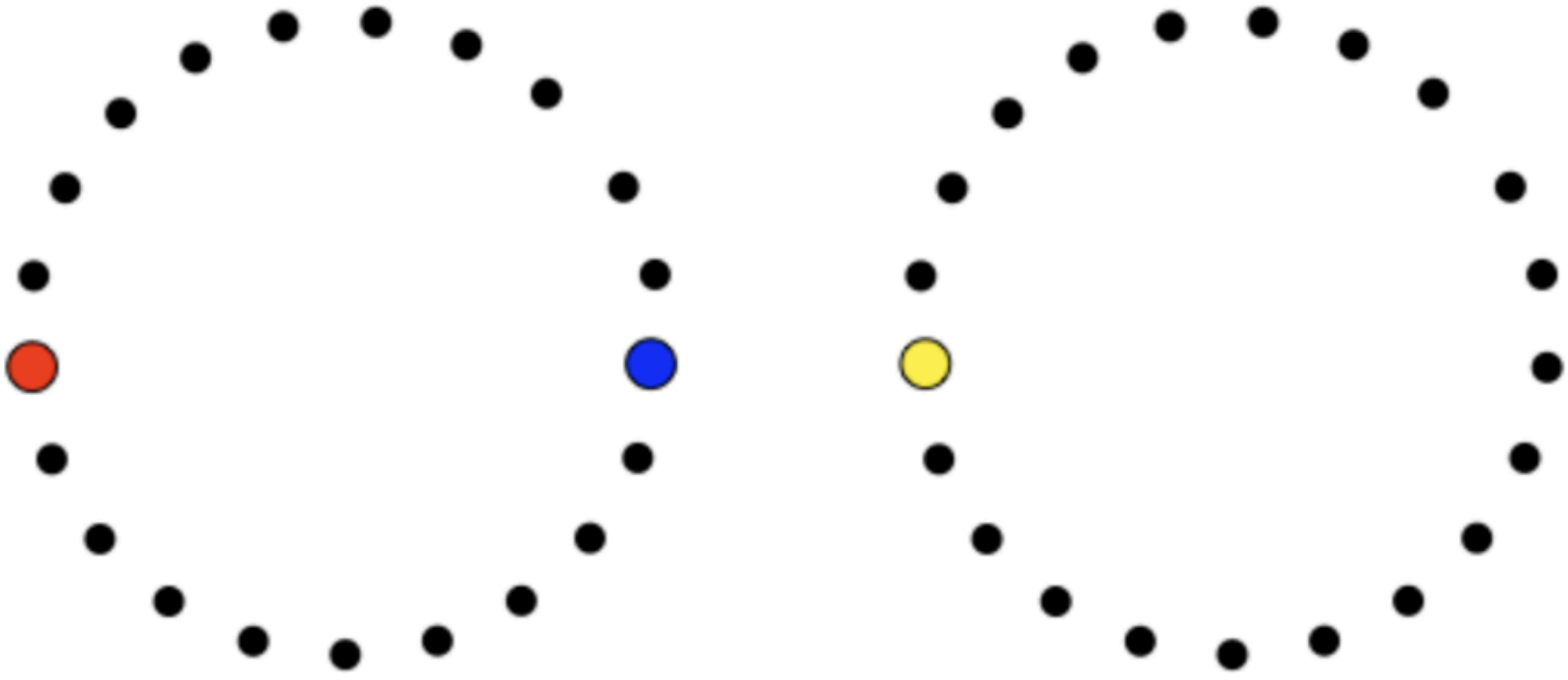
We can organize algorithms into two broad classes:

1. "hierarchical" clustering methods. Think the tree of life! Mammals and reptiles are both vertebrates. Vertebrates and invertebrates are both animals. Animals and plants are both eukaryotes. Etc.

2. "partitional" clustering methods. Here there is no tree or hierarchy, just a "flat" set of clusters.

A few miscellaneous notes

Distance-based clustering isn't magic.



Should blue cluster with red or with yellow?

(We can often deal with situations like this by redefining what distance is. The term here is "manifold learning.")

K-means clustering

→ K-means is a partitional clustering approach. It's the "Least Sqaures Regression of clustering"

→ Each cluster is associated with a centroid (center point). The number of clusters K must be chosen in advance

→ Each point is assigned to the cluster with the closest centroid

K-means clustering

The basic algorithm is super simple:

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

That's it! Algorithms don't get any simpler in machine learning

K-means clustering (considerations)

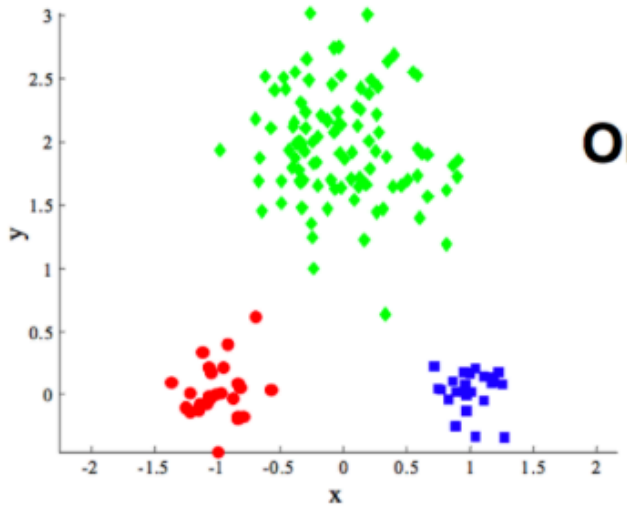
- Initial centroids are often chosen randomly. Thus the clusters produced vary from one run to another

- The centroid is (typically) the mean of the points in the cluster

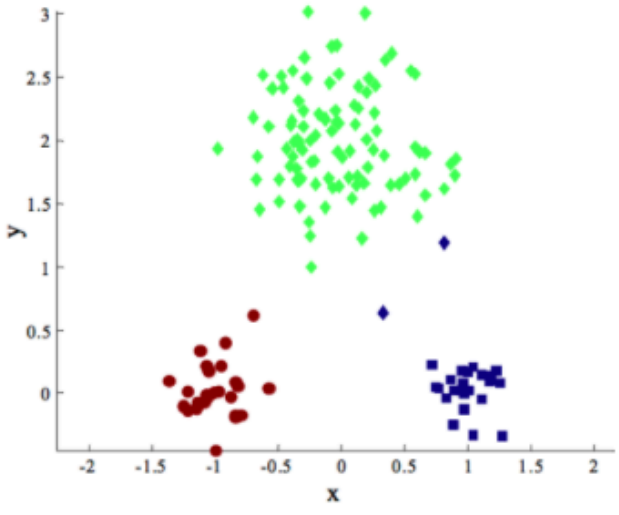
- Closeness is measured by Euclidean (default) or any valid distance

- K-means will converge pretty rapidly for these common distance measures. Most of the convergence happens in the first few iterations

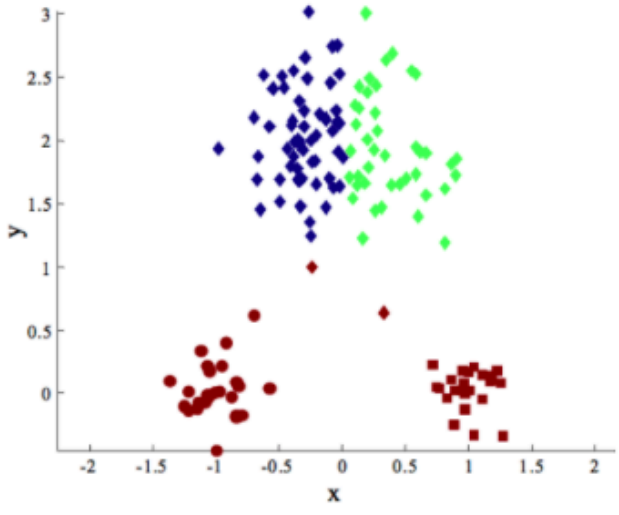
Two different k-means clusterings



Original Points

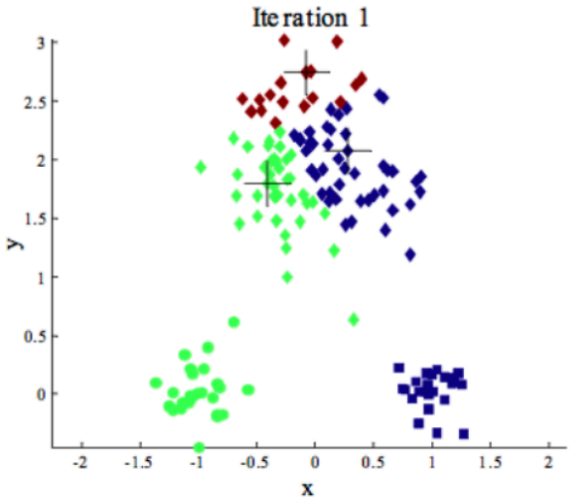


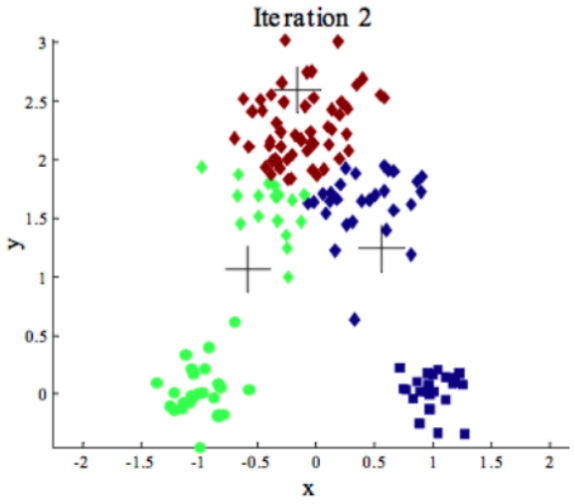
Optimal Clustering

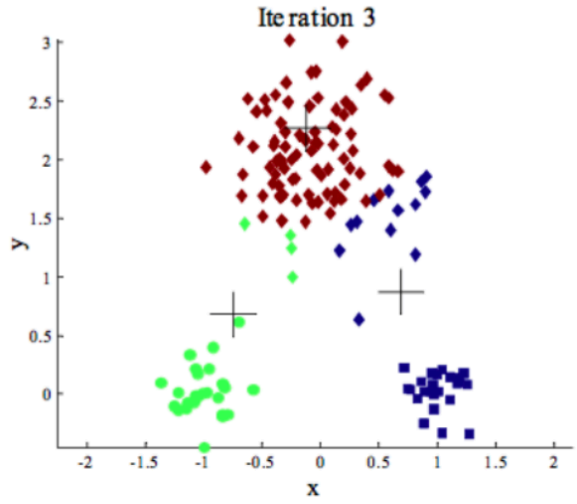


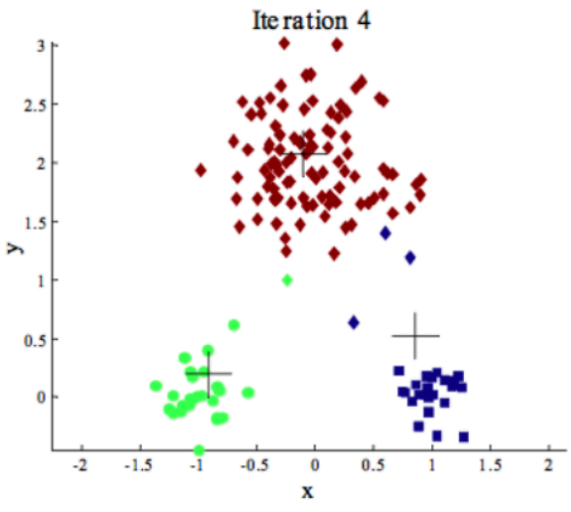
Sub-optimal Clustering

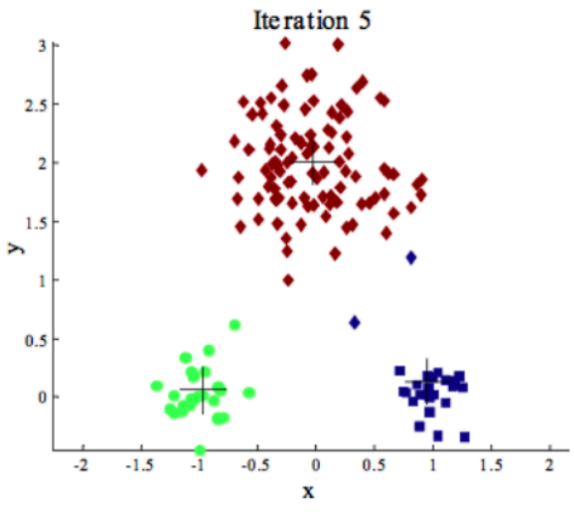
From one set of initial centroids

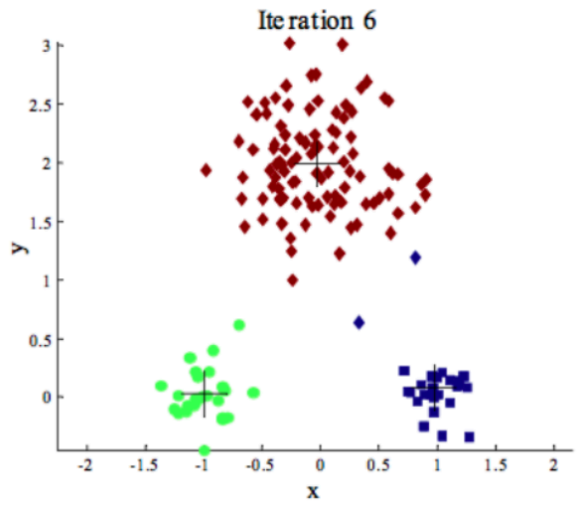




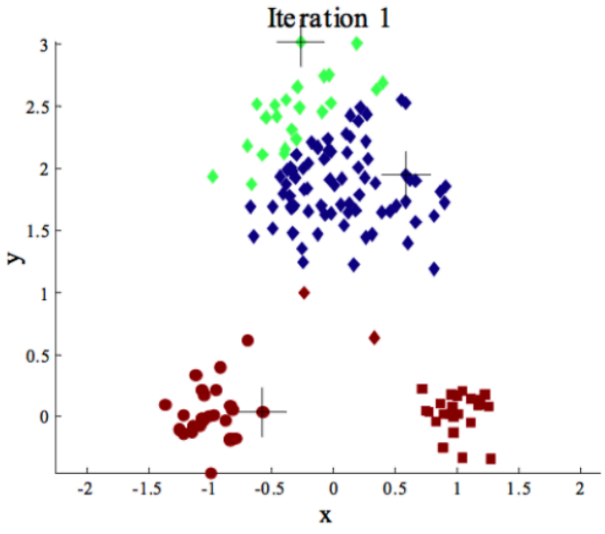


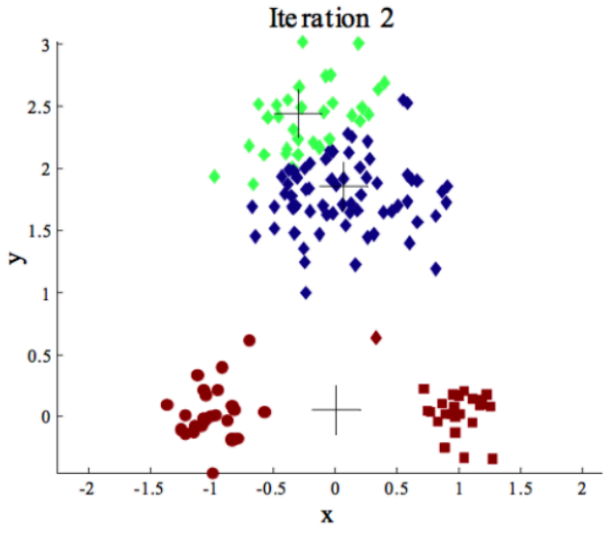


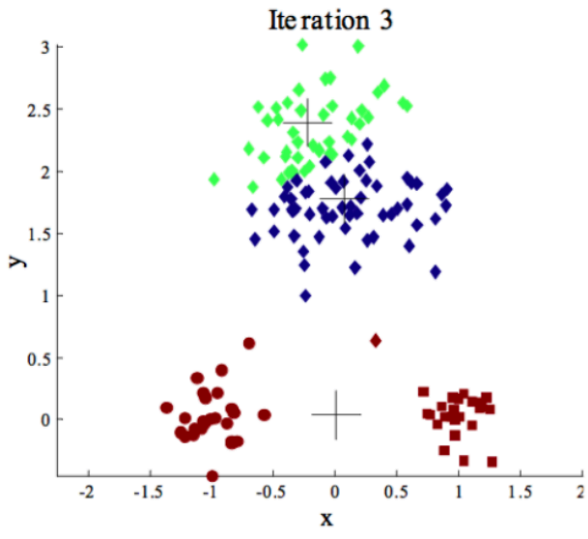


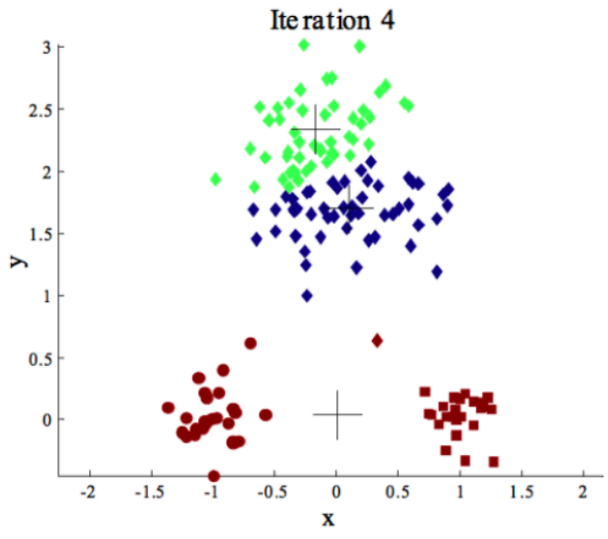


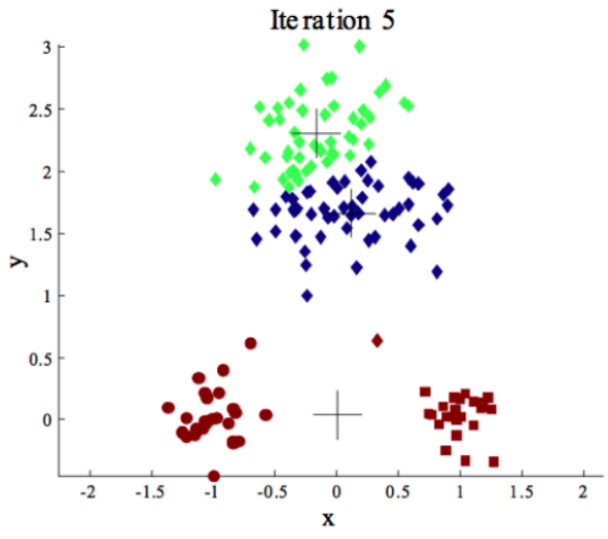
From a different set of initial centroids



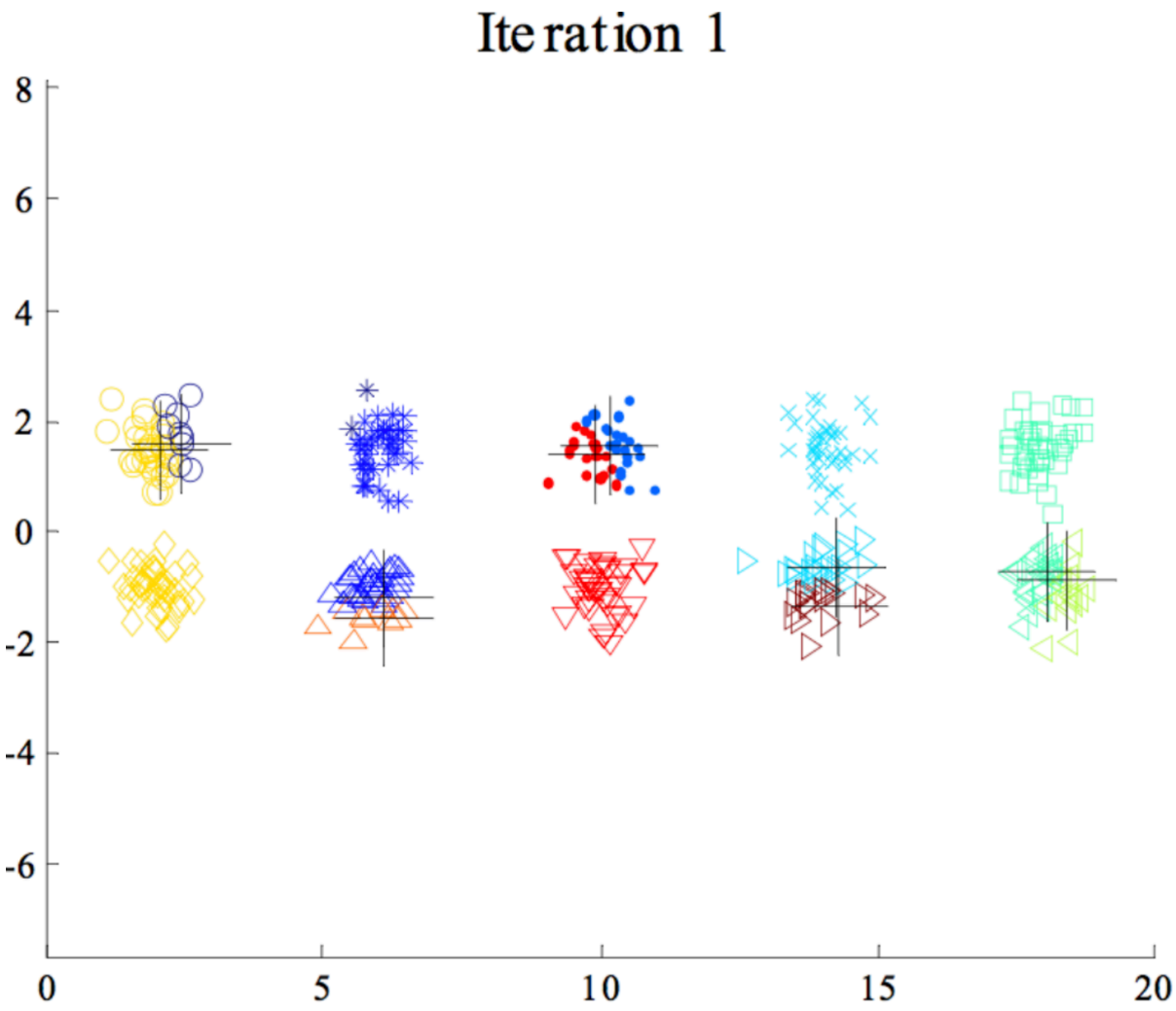






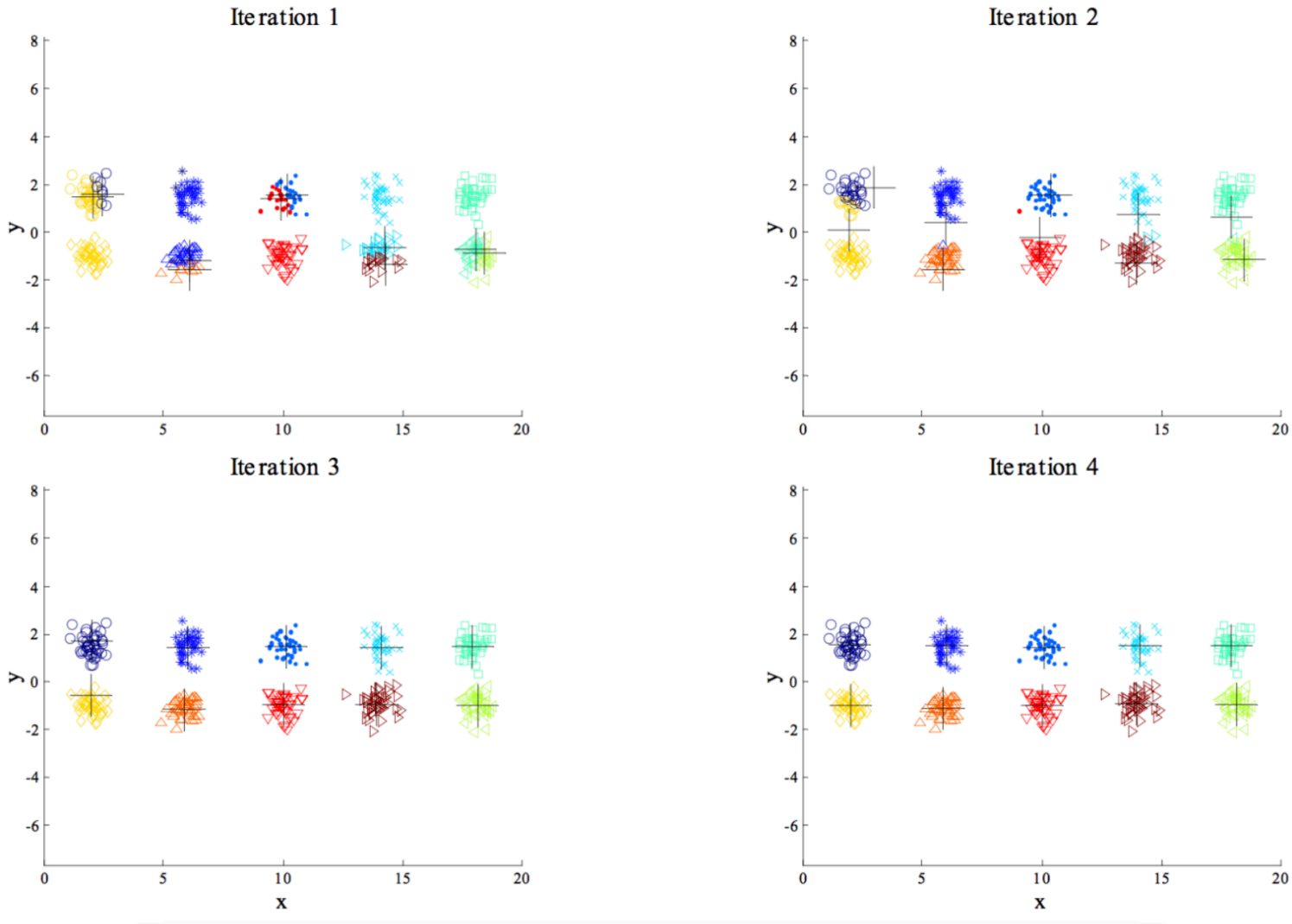


Ten initial centroids



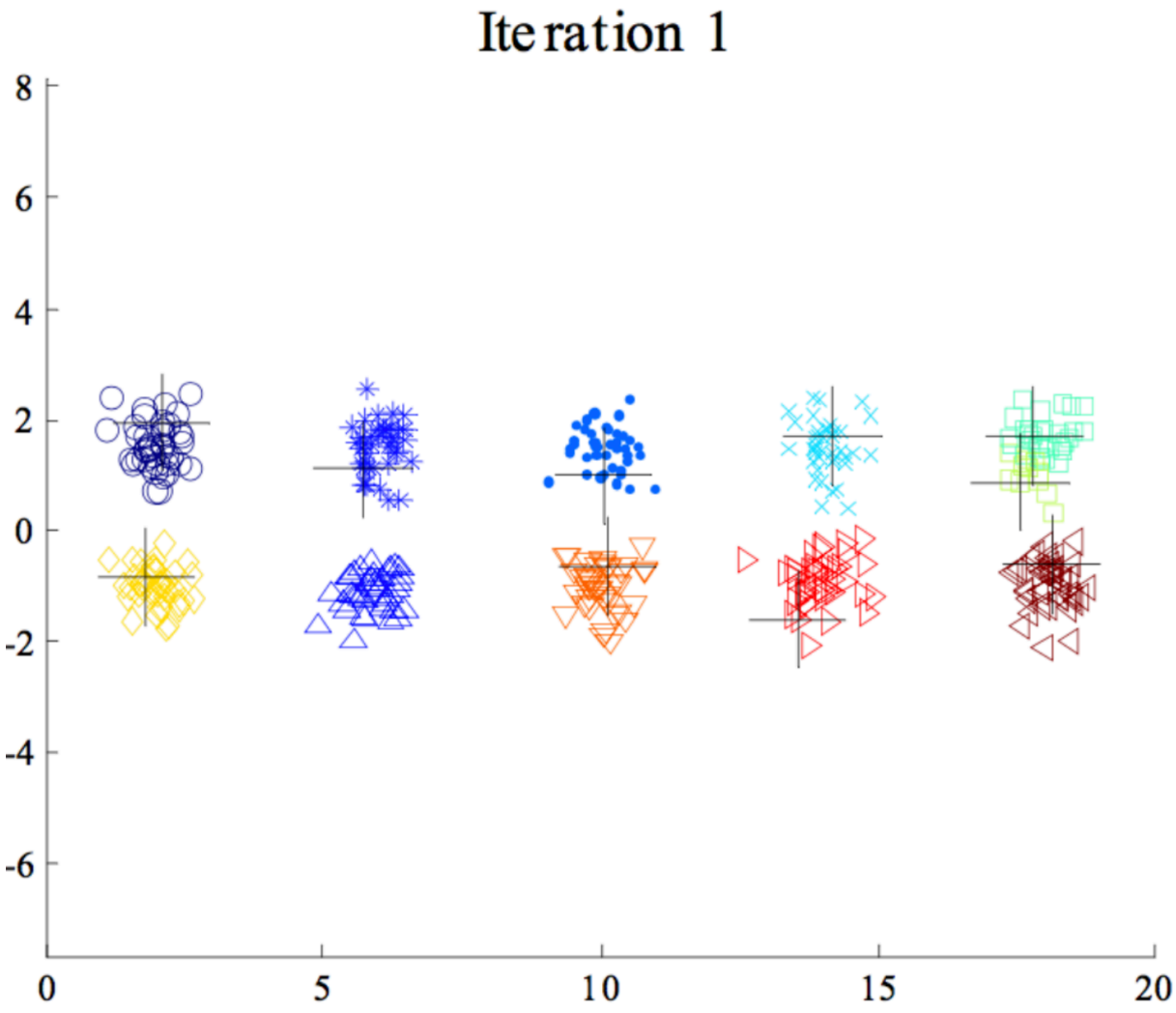
Each pair of clusters has two initial centroids

Ten initial centroids



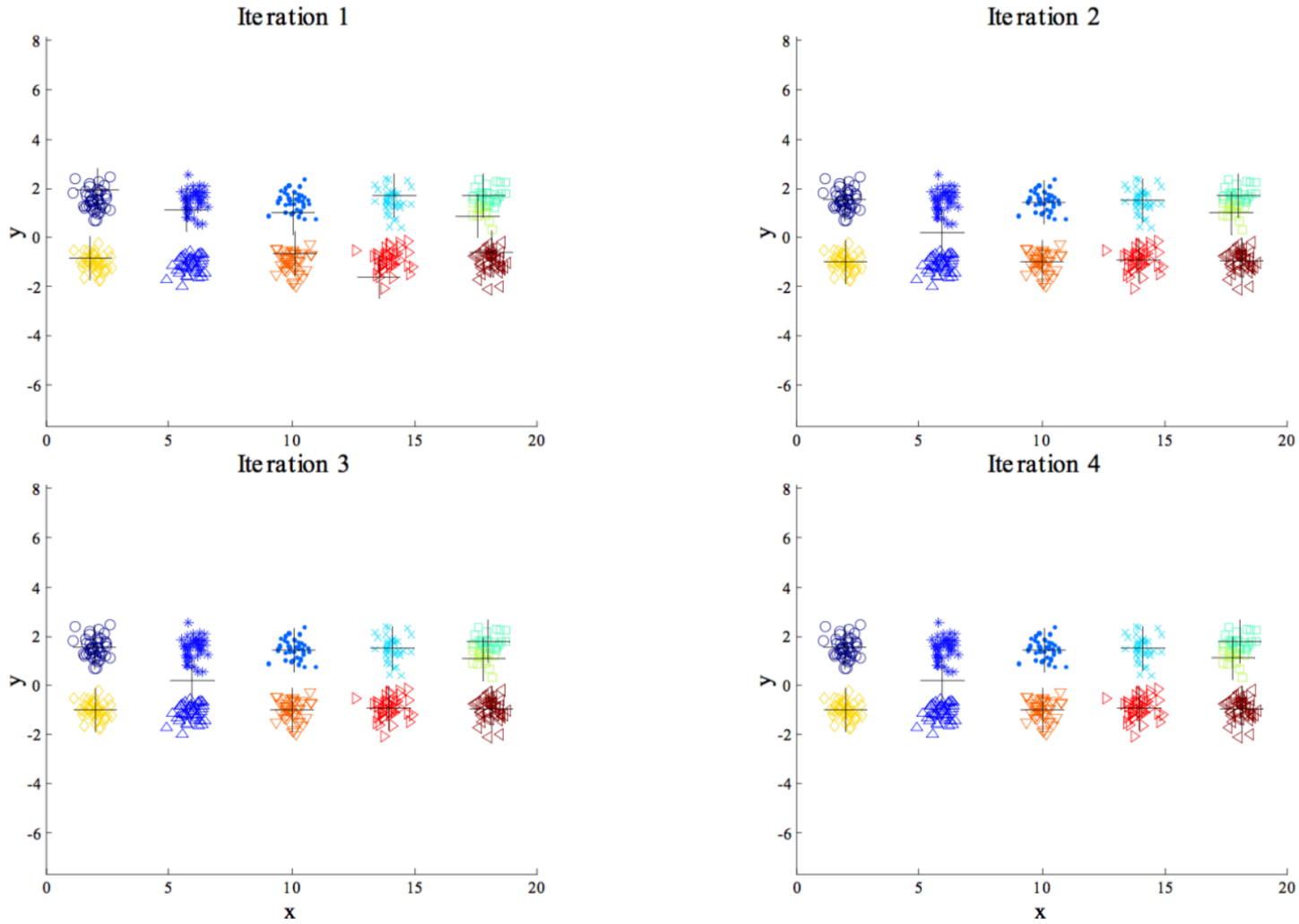
Not bad!

Ten initial centroids: Another try



Now one pair of clusters has one centroid (another pair has three)

Ten initial centroids: Another try



Not as good!

Solutions to the "initial centroid" problem

→ Multiple restarts (helps, but probability is not on your side)

→ Select more than K initial centroids and then select the most widely separated among these initial centroids

→ Postprocessing

→ Different initialization strategies (e.g. K-means++)

K-means++

Initialize by choosing 1 centroid at random, m_1.

Then for $k = 2, \ldots, K$:

1) For each point, compute d_i as the minimum distance of x_i to the existing centroids.

2) Choose x_i as initial centroid k probability proportional to d_i^2. Thus points that are far away from existing cluster centroids are more likely to get chosen.

Note: this is just a different initialization strategy. After initialization, it works the same as K-means

K-means: Evaluating in-sample fit

Let m_k be cluster centroid k, and let C_k be the set of points in cluster k (i.e. for which $\gamma_i = k$).

Within-cluster sum of squares should be low:

$$\text{SSE}_W = \sum_{k=1}^{K} \sum_{x_i \in C_k} d(m_k, x_i)^2$$

Between-cluster sum of squares should be high:

$$\text{SSE}_B = \sum_{k=1}^{K} d(m_k, \bar{x})^2$$

where $\bar{x}$ is the overall sample mean.

K-means: Example

Let's go to `cars.R`!

K-means only finds a local optimum

K-means tries to minimize within-cluster SSE. But:

- It basically never finds a global optimizer, except in simple problems.
- There are too many possible clusterings to check them all!

$$A(N,K) = \#\text{ of assignments of } N \text{ points to } K \text{ groups}$$
$$= \frac{1}{K!}\sum_{j=1}^{K}(-1)^{K-j}\cdot\binom{K}{j}\cdot j^N$$

$$A(10,4) = 34,105$$
$$A(25,4) \approx 5\times 10^{13} \quad \text{(ouch!)}$$