Clustering

Reference: Introduction to Statistical Learning Chapter 10.3

Outline

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

Introduction to clustering

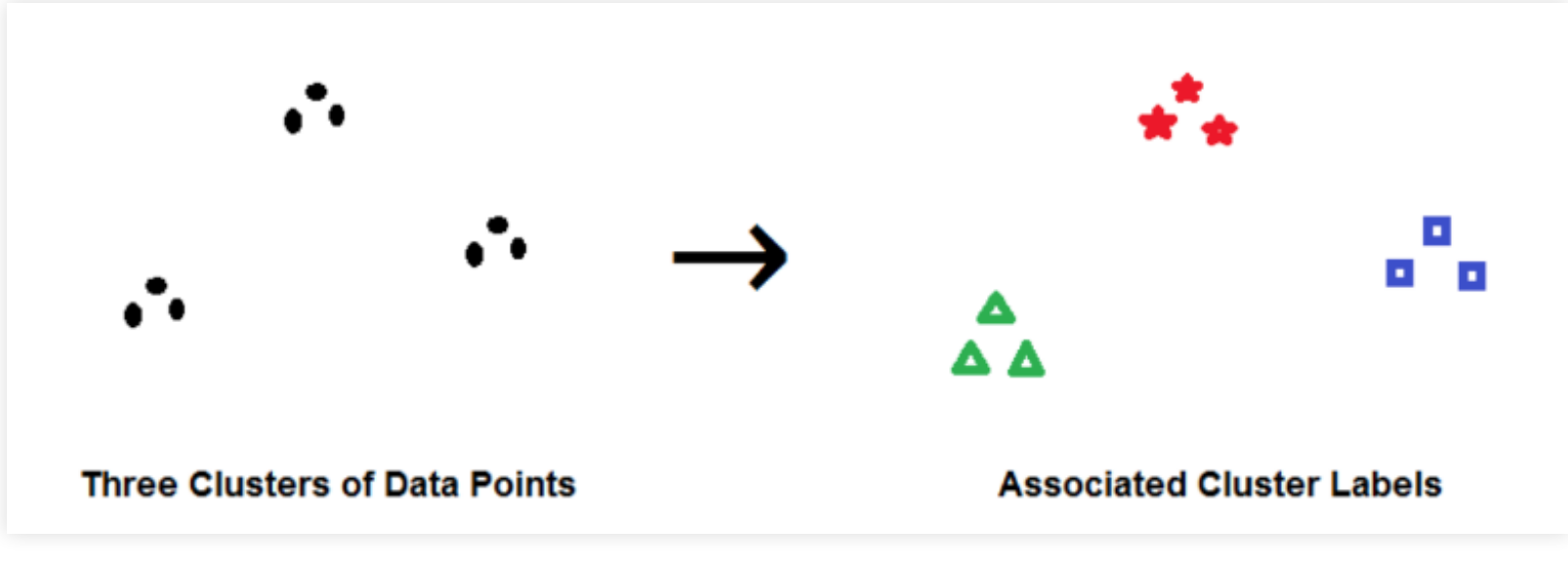
You've seen a lot of models for $p(y \mid x)$. This is *supervised learning* (knowing outcomes y = "supervision").

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

Today: **clustering**

- 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:

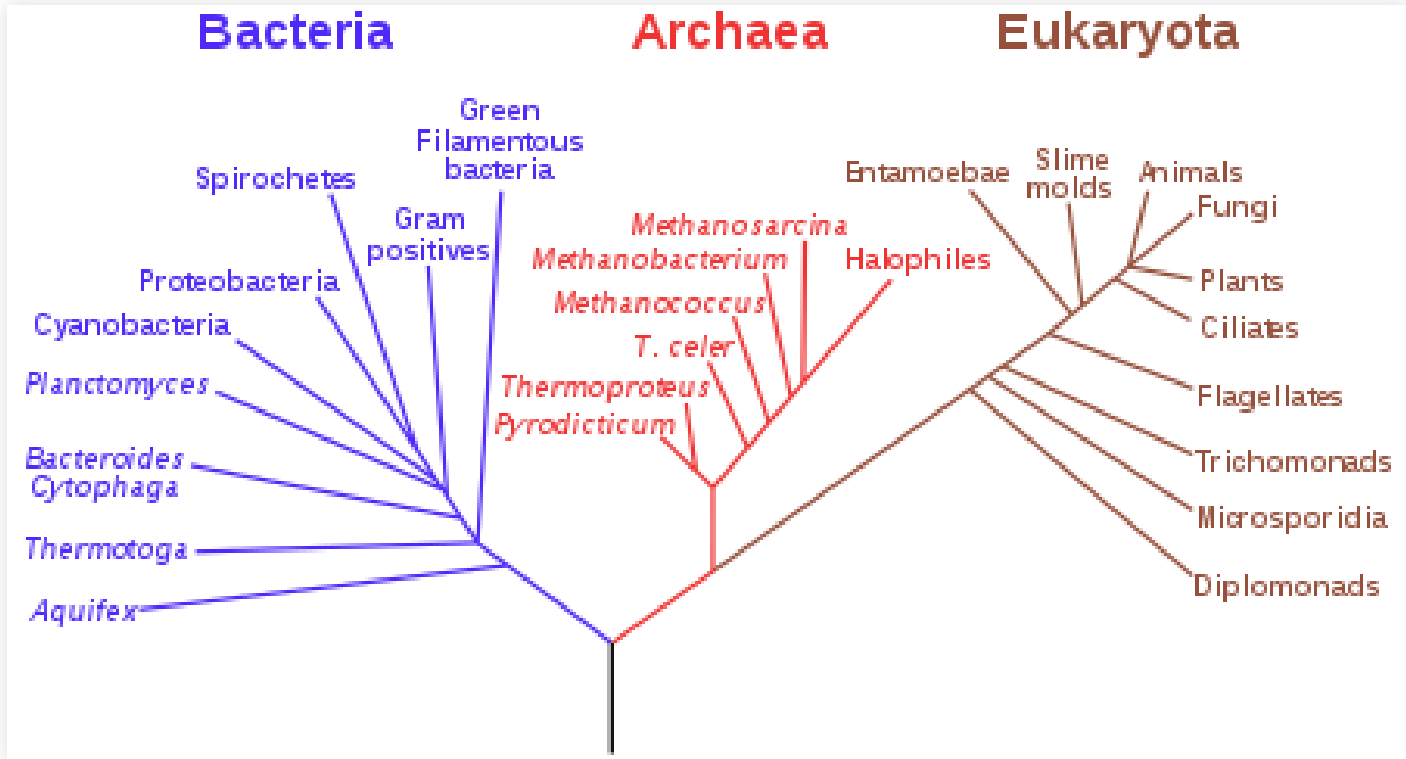
- Partition = mutually exclusive, collectively exhaustive set of categories (each data point is in one and only one cluster).
- No "mixed membership." If you encounter a Chiweenie, you need a new cluster!

Criteria for clustering

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

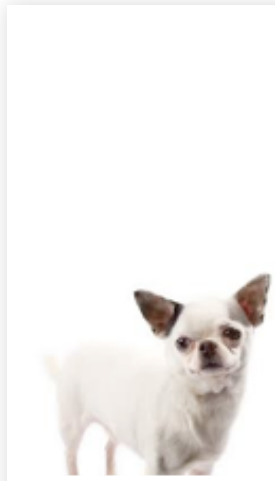
Criteria for clustering

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

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

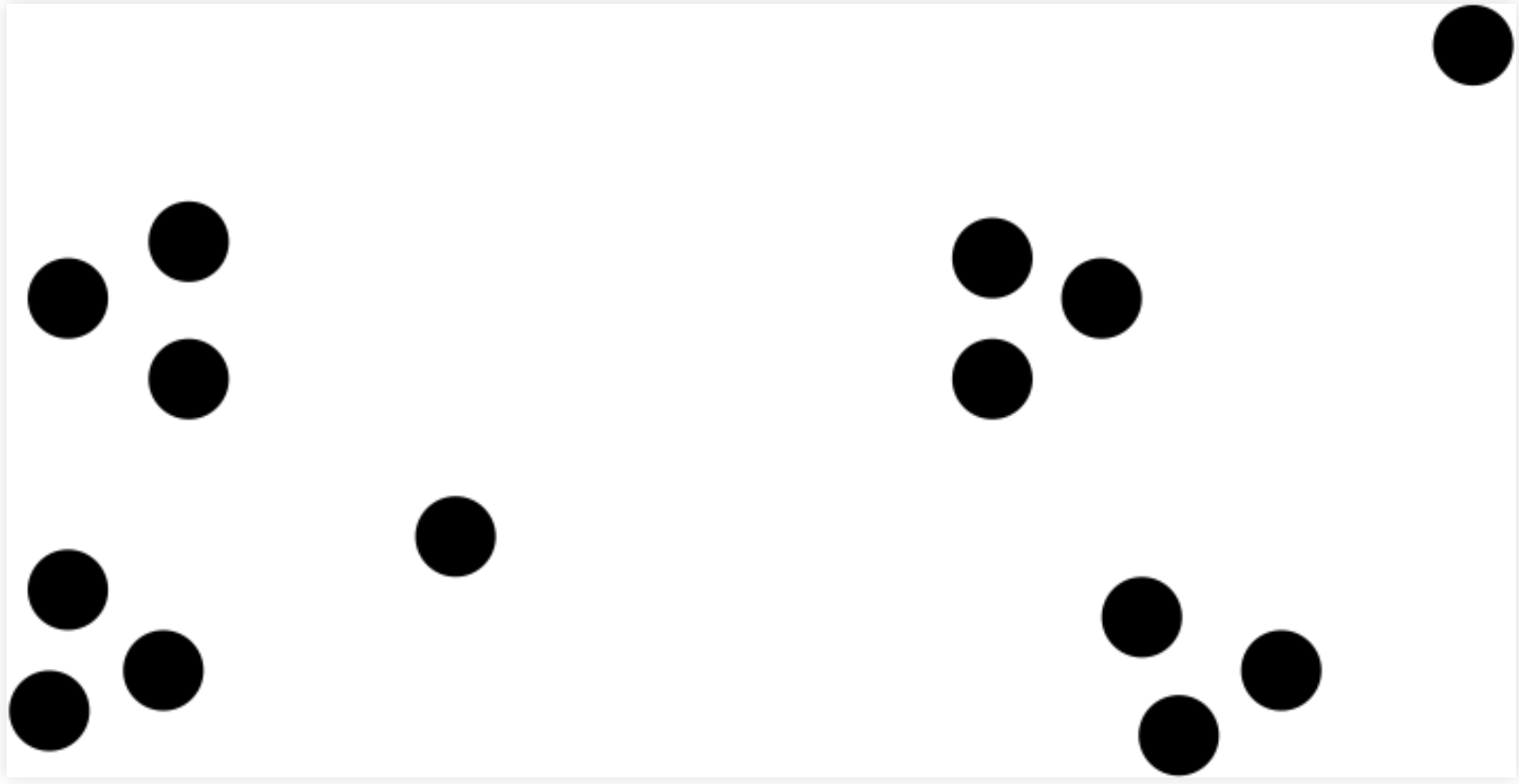
The x points can be arbitrary objects in potentially crazy spaces:

- playlists on Spotify
- phrase counts for books
- DNA sequences
- 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?

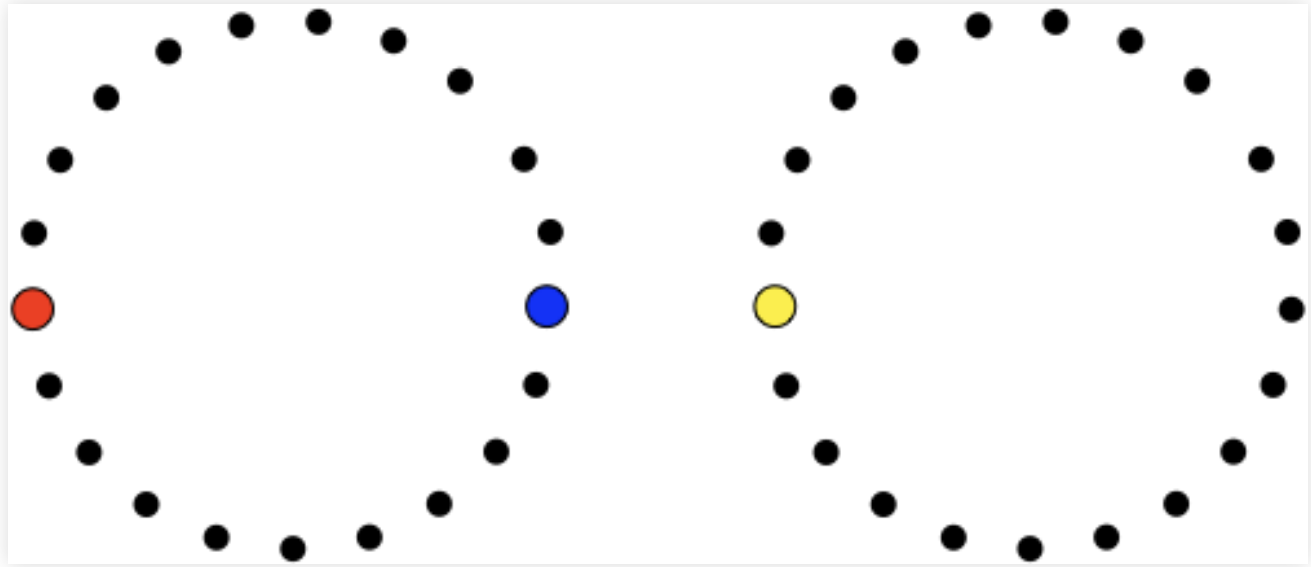
A few miscellaneous notes

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 "OLS 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

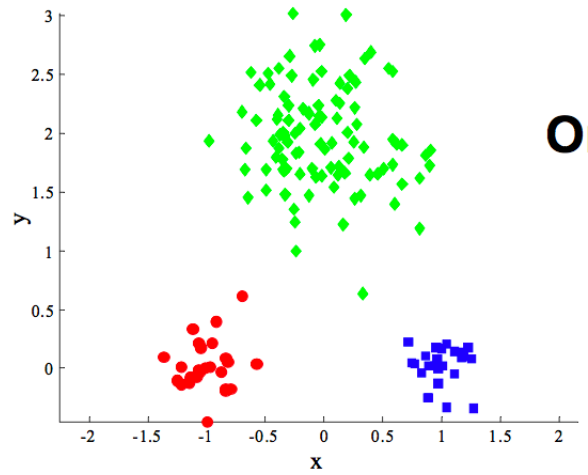
The basic algorithm is super simple:

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

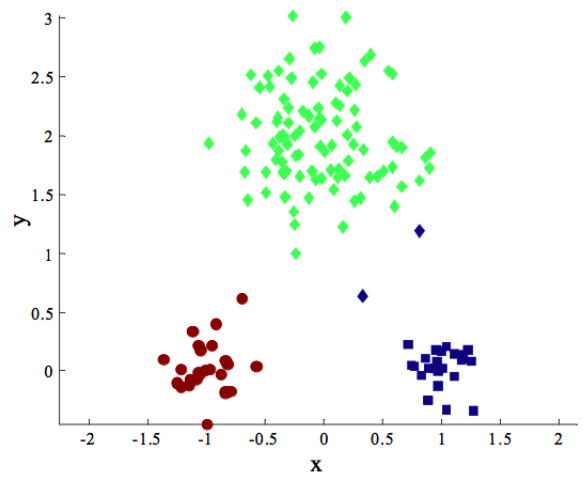
K-means clustering

- 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.
- Complexity of each iteration is is $O(n \cdot K \cdot D)$ where n is the number of data points, K is the number of clusters, and D is the number of features.

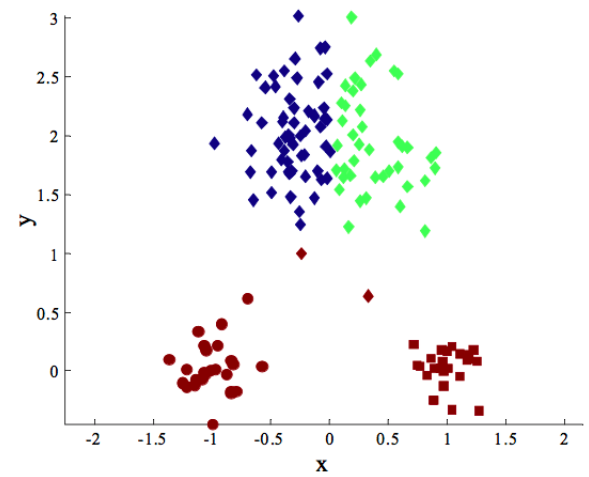
Two different K-means Clusterings



Original Points

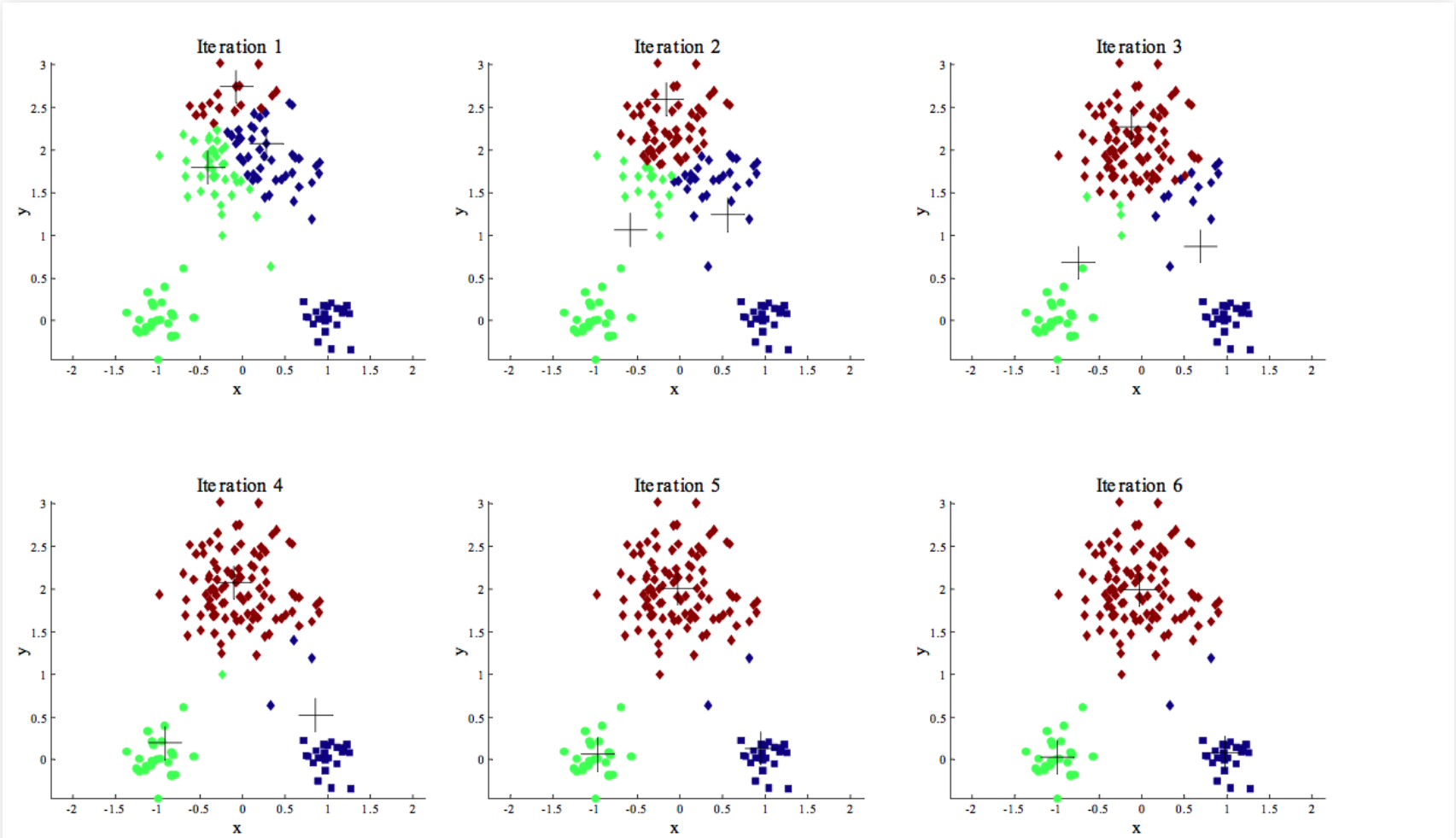


Optimal Clustering

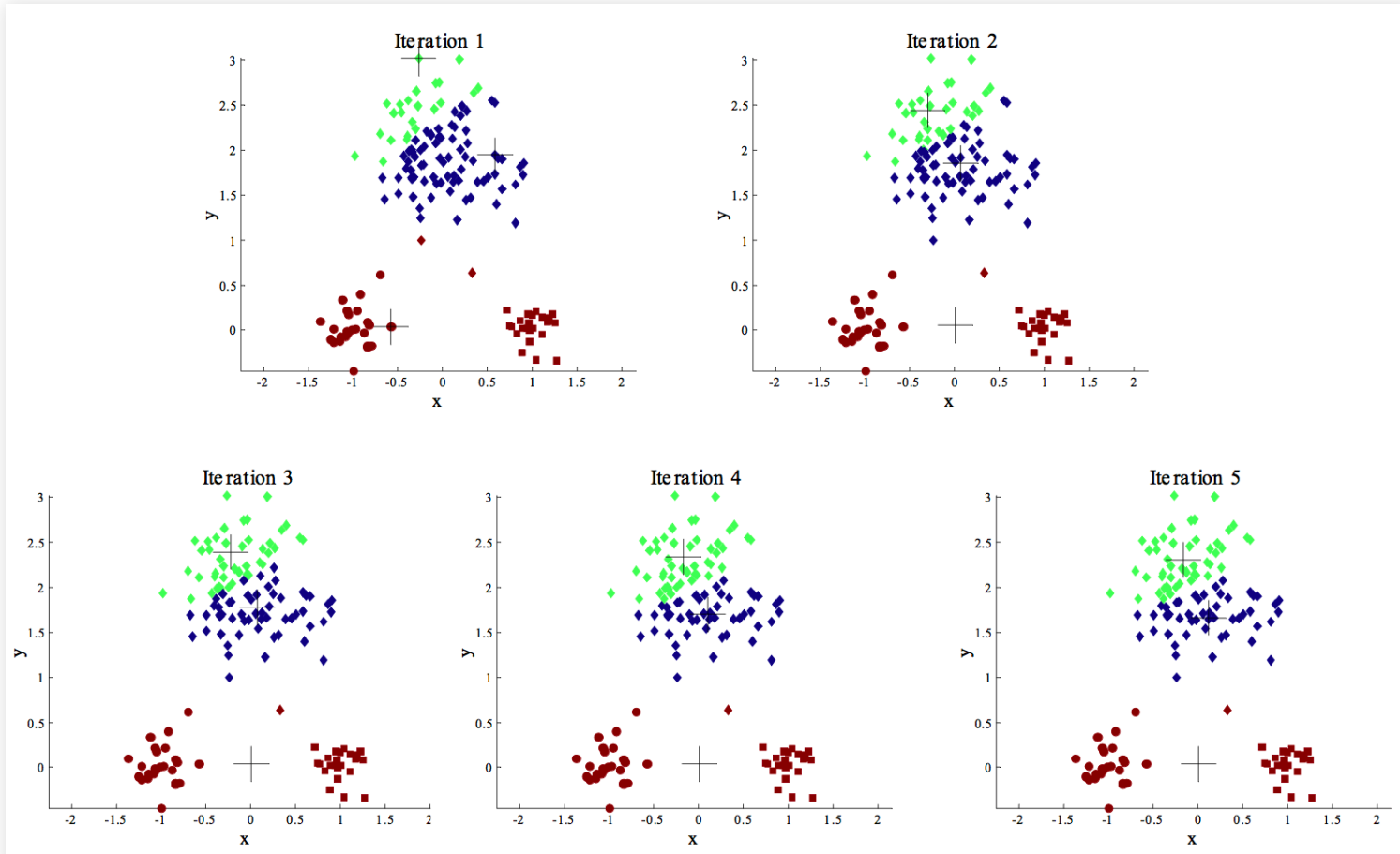


Sub-optimal Clustering

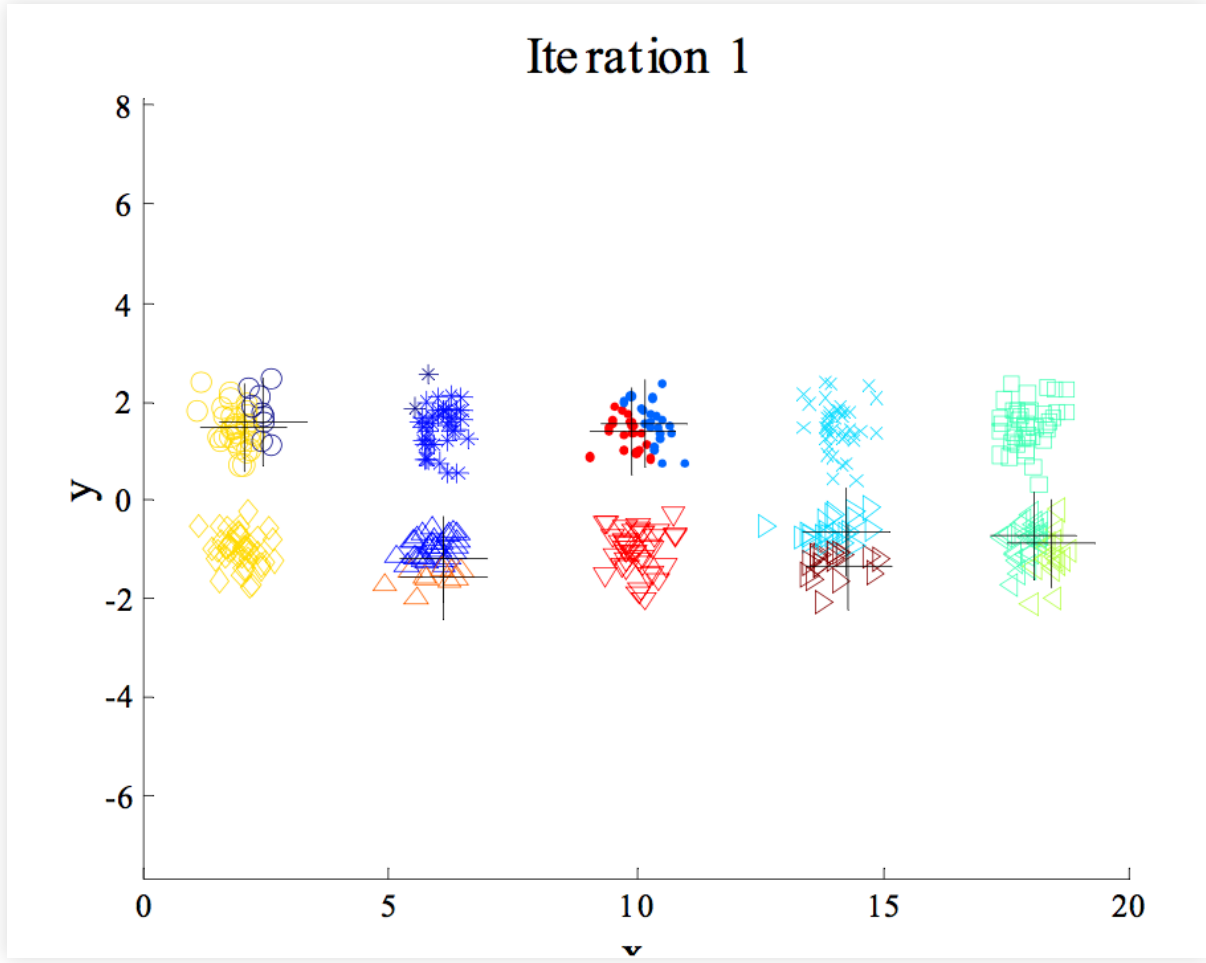
From one set of initial centroids



From a different set of initial centroids

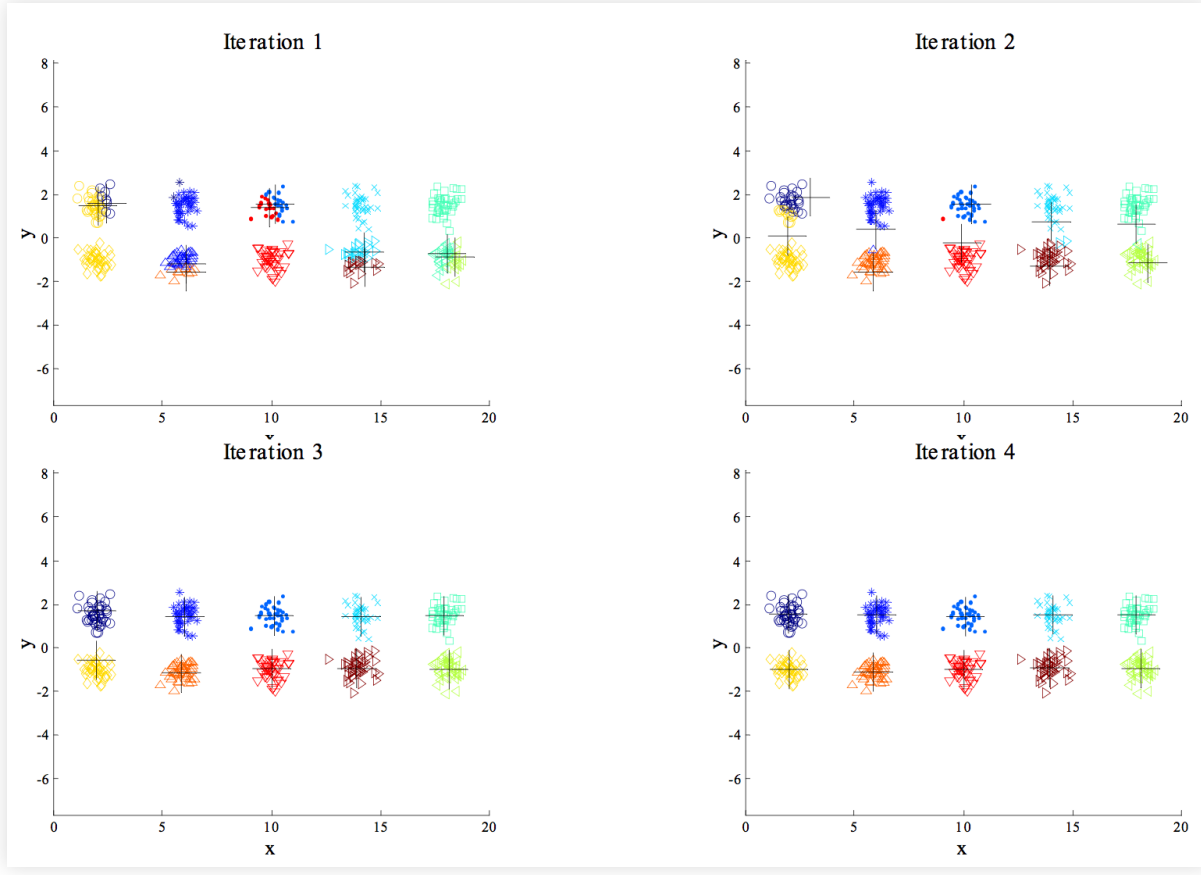


Ten initial centroids



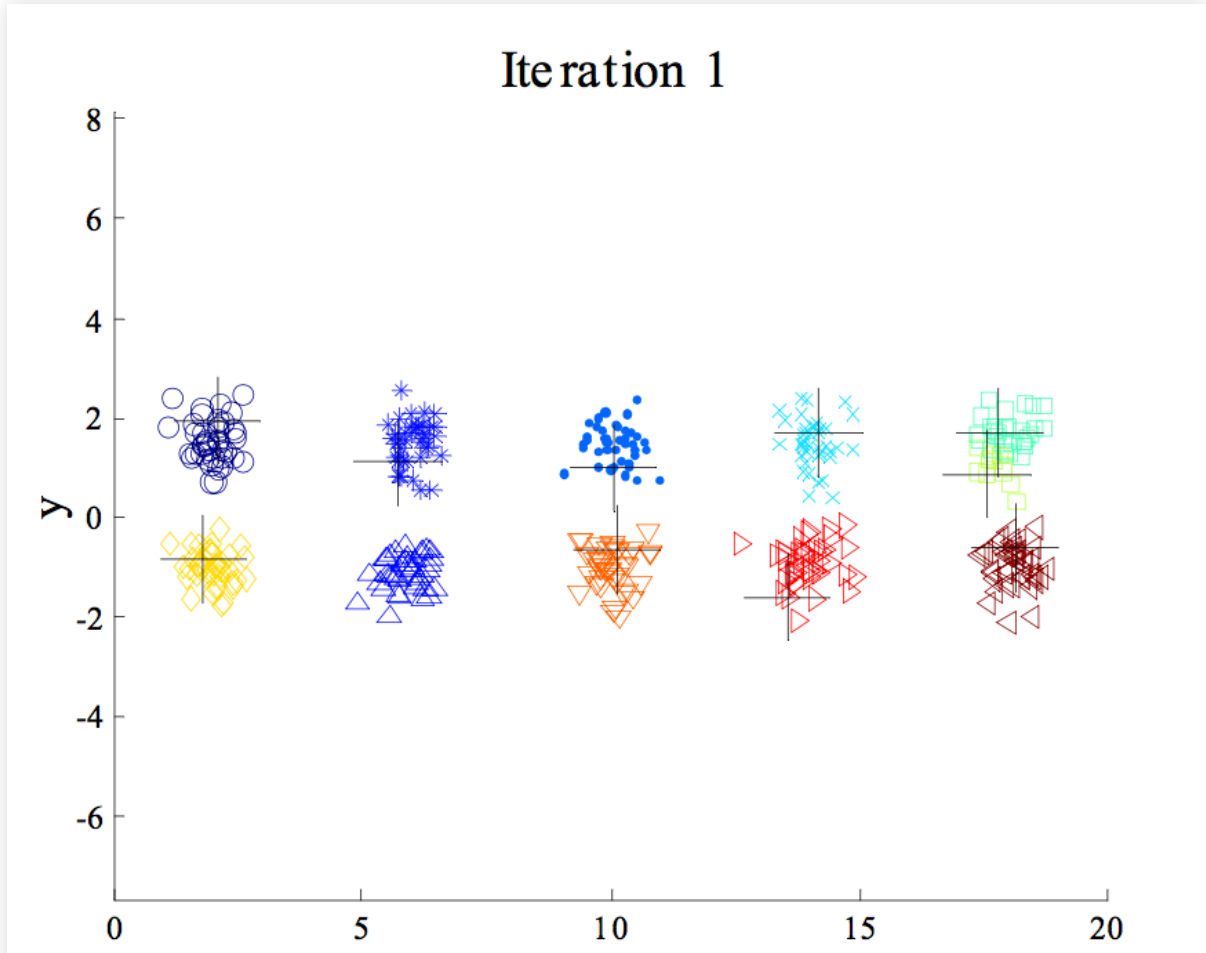
Each pair of clusters has two initial centroids.

Ten initial centroids



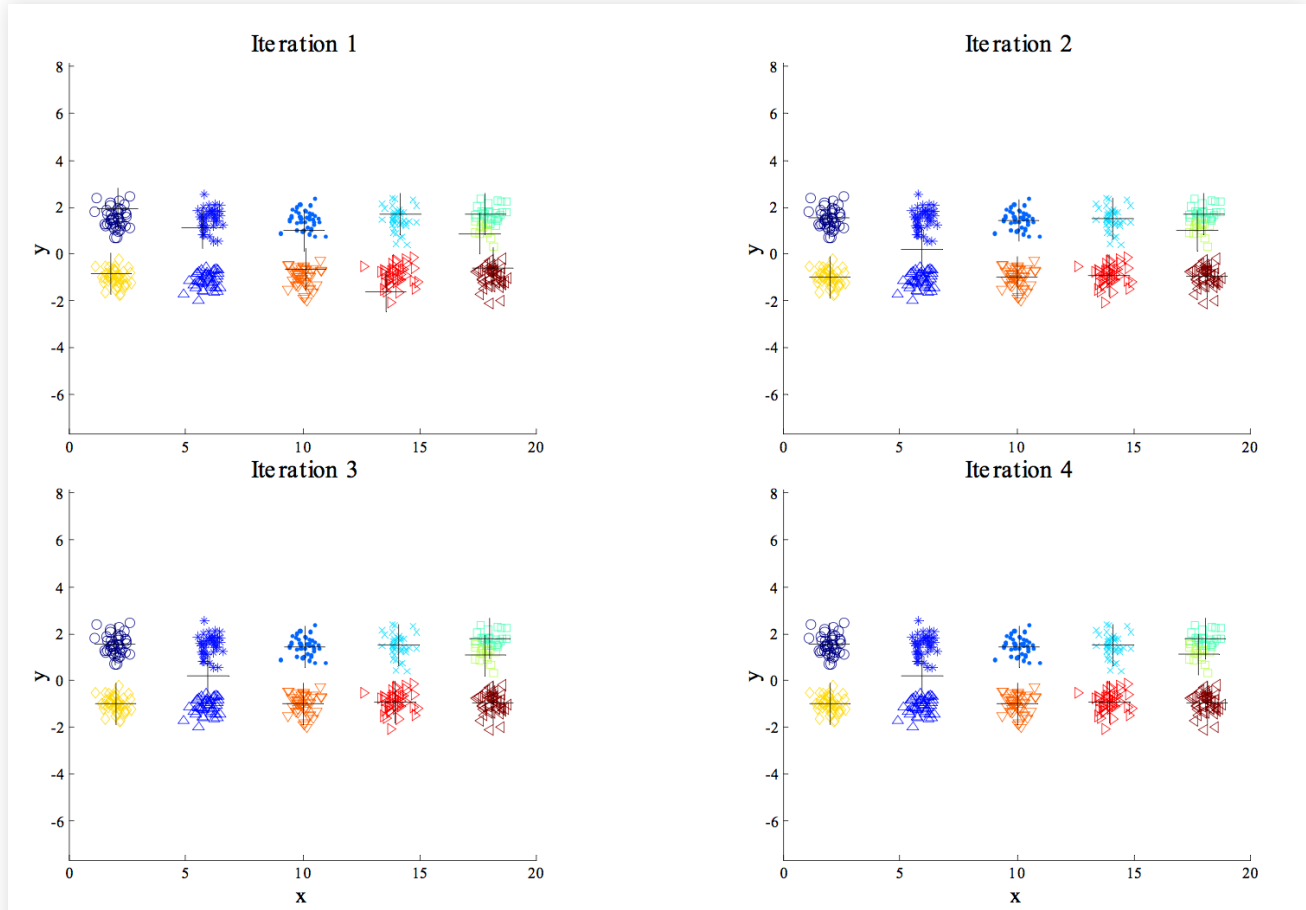
Not bad!

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 "Initial Centroids" 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++)
- Sample and use hierarchical clustering to determine initial centroids (sampling reduces computational cost of hierarchical clustering)

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 with 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 see an example in `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!

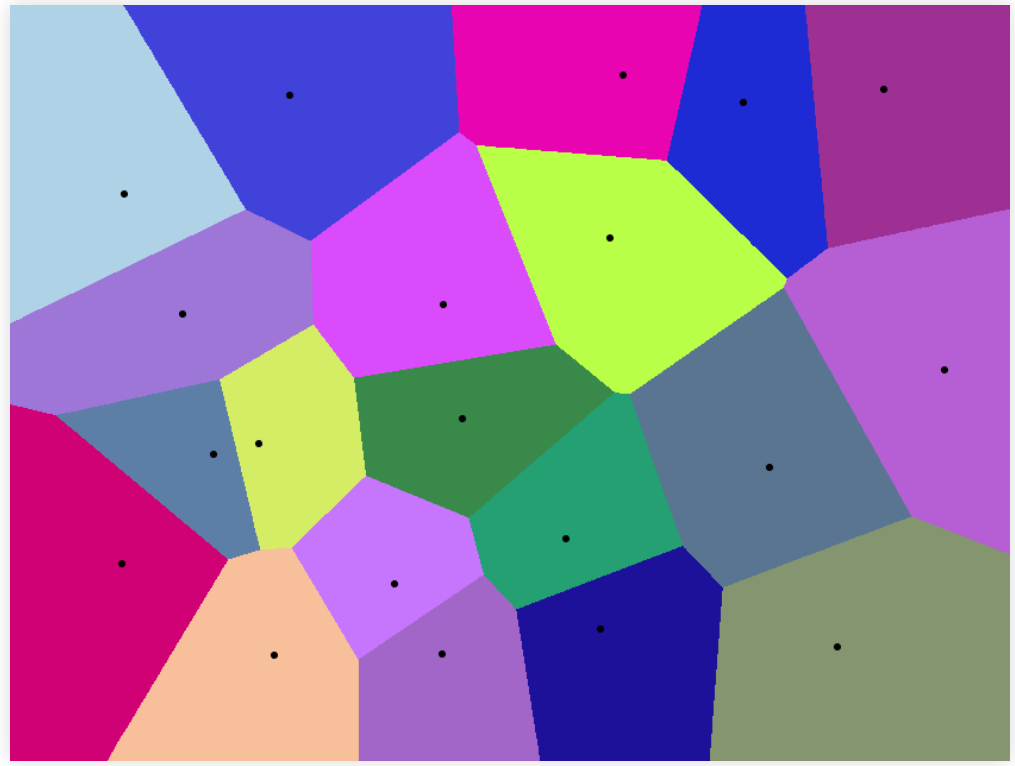
$$A(N, K) = \#\text{ of assignments of } N \text{ points to } K \text{ groups}$$
$$= \frac{1}{K!} \sum_{j=1}^{K} (-1)^{K-j} \cdot \binom{K}{j} \cdot j^N$$

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

The geometry of K-means

Let $m_1, \ldots, m_K$ be the cluster centroids, and define

$$V_k = \{x \in \mathcal{R}^D : d(x - m_k) \leq d(x, m_j) \text{ for } j = 1, \ldots, K\}$$



The geometry of K-means

These V_k are convex polyhedra, and they form a Voronoi tesselation.

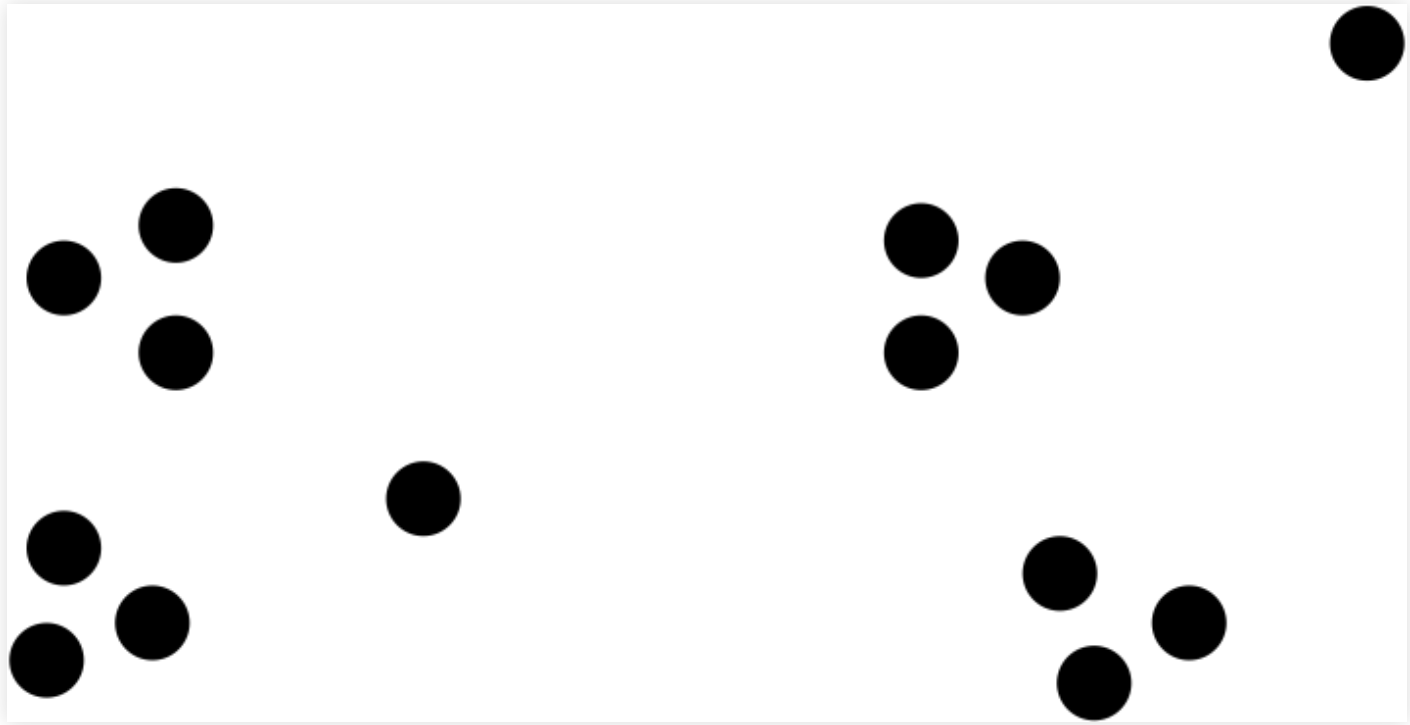
Upshot: K-means will not successfully recover clusters of points that aren't shaped like a convex polyhedron (rings, arcs, amoeba-like blobs, bananas, etc)

K-means is best at recovering clusters shaped like multivariate Gaussians.

But: k-means can often be successfully applied in conjunction with another algorithm to find non-convex clusters.

Choosing K

Remember: K is an *input* to K-means, not an estimated parameter. **There is no generally accepted "best" method for choosing K,** and there never will be. Recall this example:



Choosing K

Unlike in supervised learning, we can't really cross validate, because we don't know the "true" cluster labels of points in a training set.

Some heuristics that people use in practice:

- choose it in advance based on prior knowledge or prior utility
- find the "elbow" on a plot of SSE_W versus K.
- optimize one of many quantitative model-selection criteria (CH index, AIC, BIC, gap statistic…)

Choosing K

But probably the most popular principle is this:

- satisfice, don't optimize: pick a value of K that gives clusters you and your stakeholders can interpret, and call it a day.
- **There is absolutely no shame in this, and smart people do it all the time.**

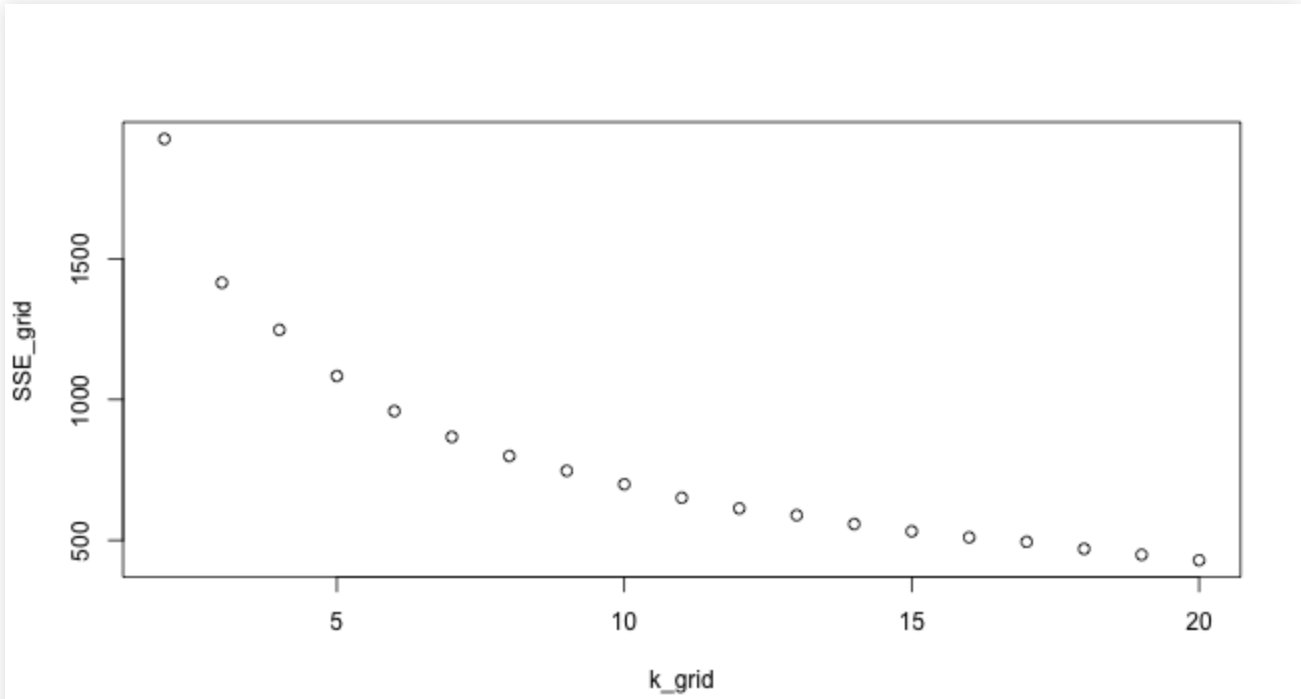
Elbow plot

Let's try many clustering with many different values of K:

```
library(foreach)
cars = read.csv('../data/cars.csv')
cars = scale(cars[,10:18]) # cluster on
measurables
k_grid = seq(2, 20, by=1)
SSE_grid = foreach(k = k_grid, .combine='c')
%do% {
  cluster_k = kmeans(cars, k, nstart=50)
  cluster_k$tot.withinss
}
```

Elbow plot

```
plot(k_grid, SSE_grid)
```



Do you see an elbow?

CH index

The CH index is just one of many model-selection criteria that people have proposed for choosing K in K-means clustering. It tries to balance fit with simplicity.

Let n_j be the number of points in cluster j, and define

$$B(K) = \sum_{j=1}^{K} n_j d(m_j, \bar{x})^2 \quad \text{("between")}$$

$$W(K) = \sum_{j=1}^{K} \sum_{x_i \in C_j} d(x_i, m_j)^2 \quad \text{("within")}$$

CH index

The CH-index is defined as

$$CH(K) = \frac{B(K)}{W(K)} \cdot \frac{N-K}{K-1}$$

- The first term, $B(K)/W(K)$ gets larger as K gets larger (better fit).
- The second term, $(N-K)/(K-1)$ gets smaller as K gets larger (more complex model).

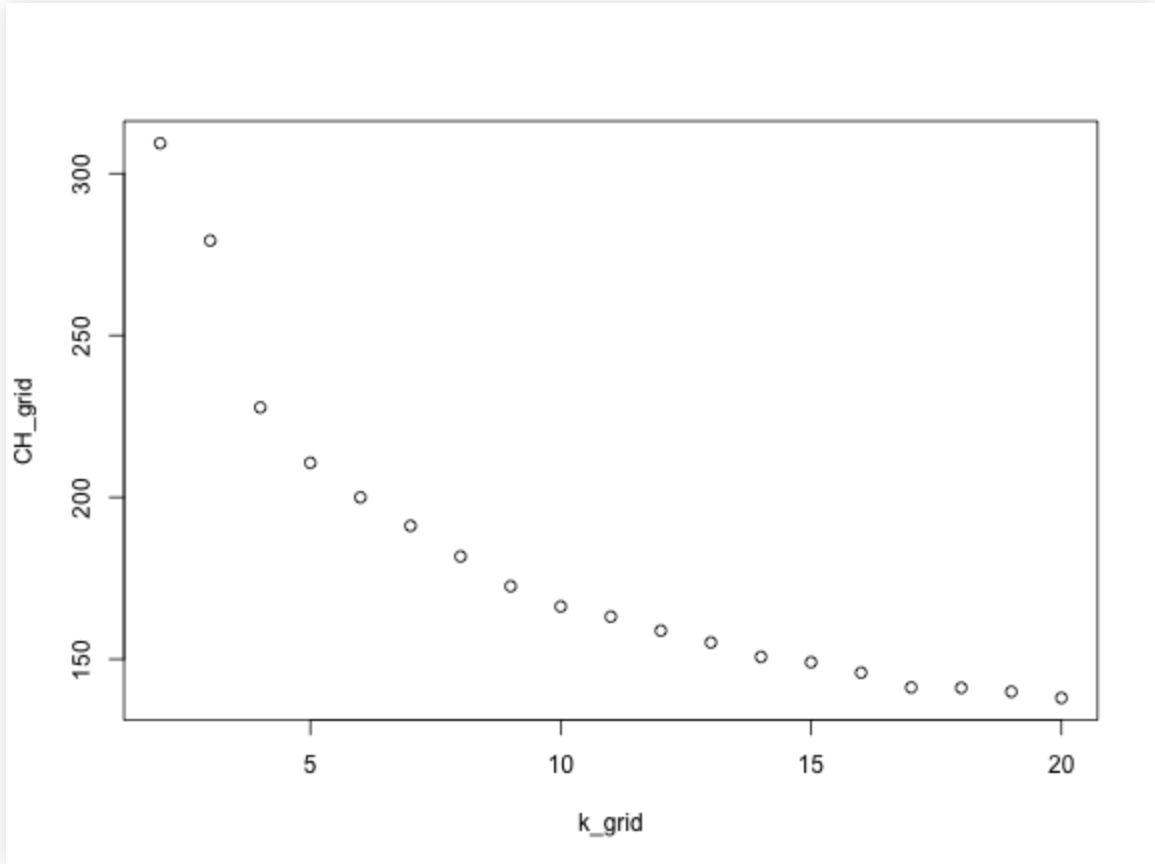
So choose $\hat{K} = \arg\max CH(K)$.

CH index

```
N = nrow(cars)
CH_grid = foreach(k = k_grid, .combine='c')
%do% {
  cluster_k = kmeans(cars, k, nstart=50)
  W = cluster_k$tot.withinss
  B = cluster_k$betweenss
  CH = (B/W)*((N-k)/(k-1))
  CH
}
```

CH index

```
plot(k_grid, CH_grid)
```

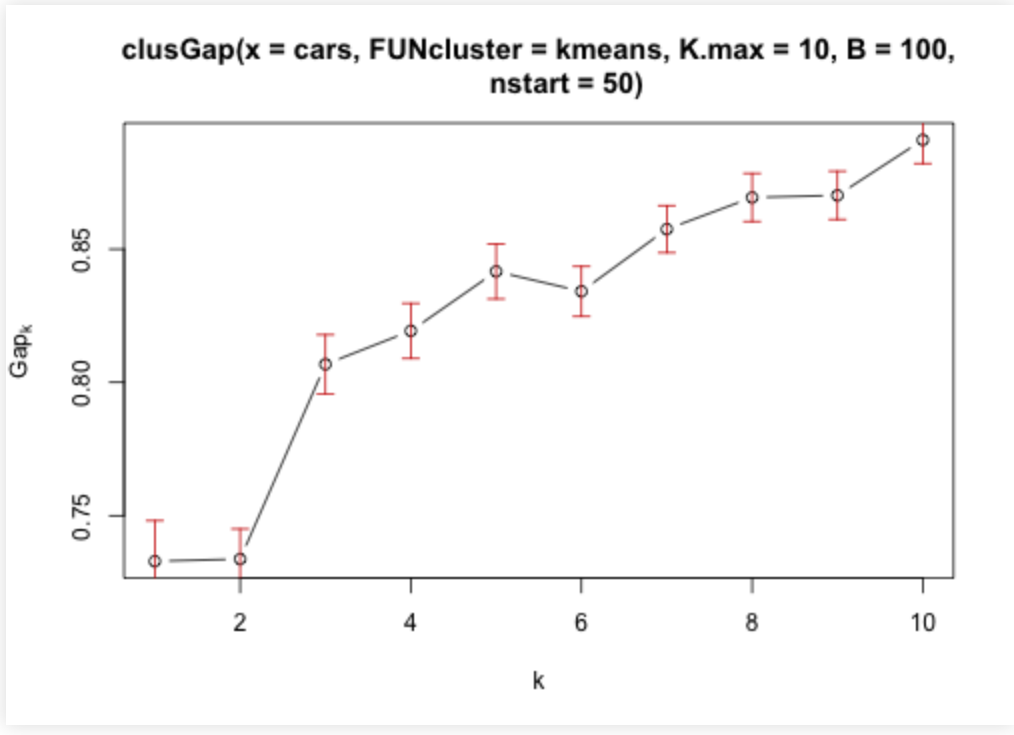


Probably K=2 or K=3?

Gap statistic

See the `clusGap` function in the `cluster` package.

```
library(cluster)
cars_gap = clusGap(cars, FUN = kmeans, nstart = 50, K.max = 10, B = 100)
plot(cars_gap)
```



Look for a dip: here after K=5.

Gap statistic: output

```
cars_gap
```

```
Clustering Gap statistic ["clusGap"] from call:
clusGap(x = cars, FUNcluster = kmeans, K.max = 10, B = 100, nstart = 50)
B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
 --> Number of clusters (method 'firstSEmax', SE.factor=1): 5
          logW   E.logW       gap      SE.sim
 [1,] 5.892765 6.625650 0.7328843 0.015251867
 [2,] 5.613163 6.346869 0.7337060 0.011313824
 [3,] 5.454826 6.261620 0.8067944 0.011098482
 [4,] 5.387105 6.206427 0.8193223 0.010320013
 [5,] 5.315312 6.156937 0.8416251 0.010310612
 [6,] 5.281250 6.115415 0.8341645 0.009379682
 [7,] 5.222616 6.080101 0.8574848 0.008807676
 [8,] 5.179327 6.048700 0.8693730 0.009031007
 [9,] 5.150857 6.021030 0.8701737 0.009058212
[10,] 5.105023 5.996002 0.8909787 0.008895814
```

Look for a dip: here after K=5. (The output helpfully tells you.)

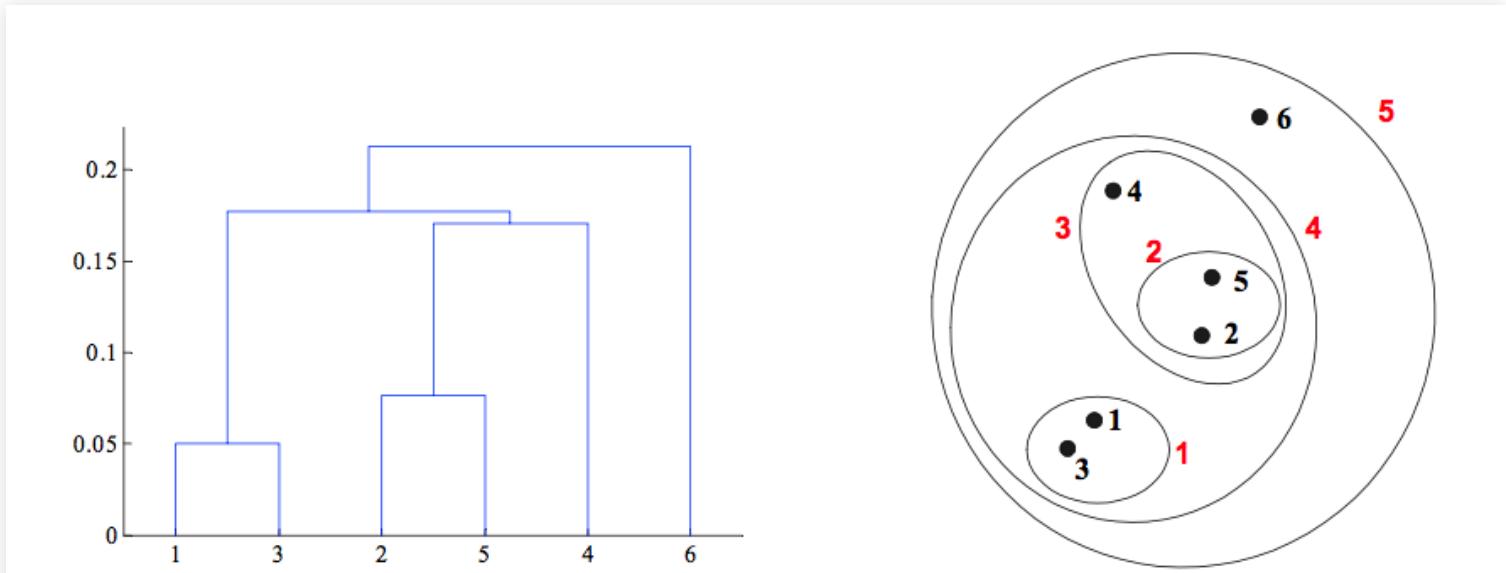
Summary of K-means

- Good for convex-shaped clusters, but less good for other shapes
- Have to choose K and no obvious way to do it
- Wicked fast, super scalable, and reasonably effective
- Usually the first thing that people try in a clustering problem

Note: I've generally had the most luck with the gap statistic as a model-selection heuristic. But don't be afraid to just pick a value of K that seems reasonable to you!

Hierarchical clustering

- Produces a set of nested clusters organized hierarchically
- Can be visualized as a "dendrogram," a tree like diagram that records the sequences of merges or splits:



Strengths of hierarchical clustering

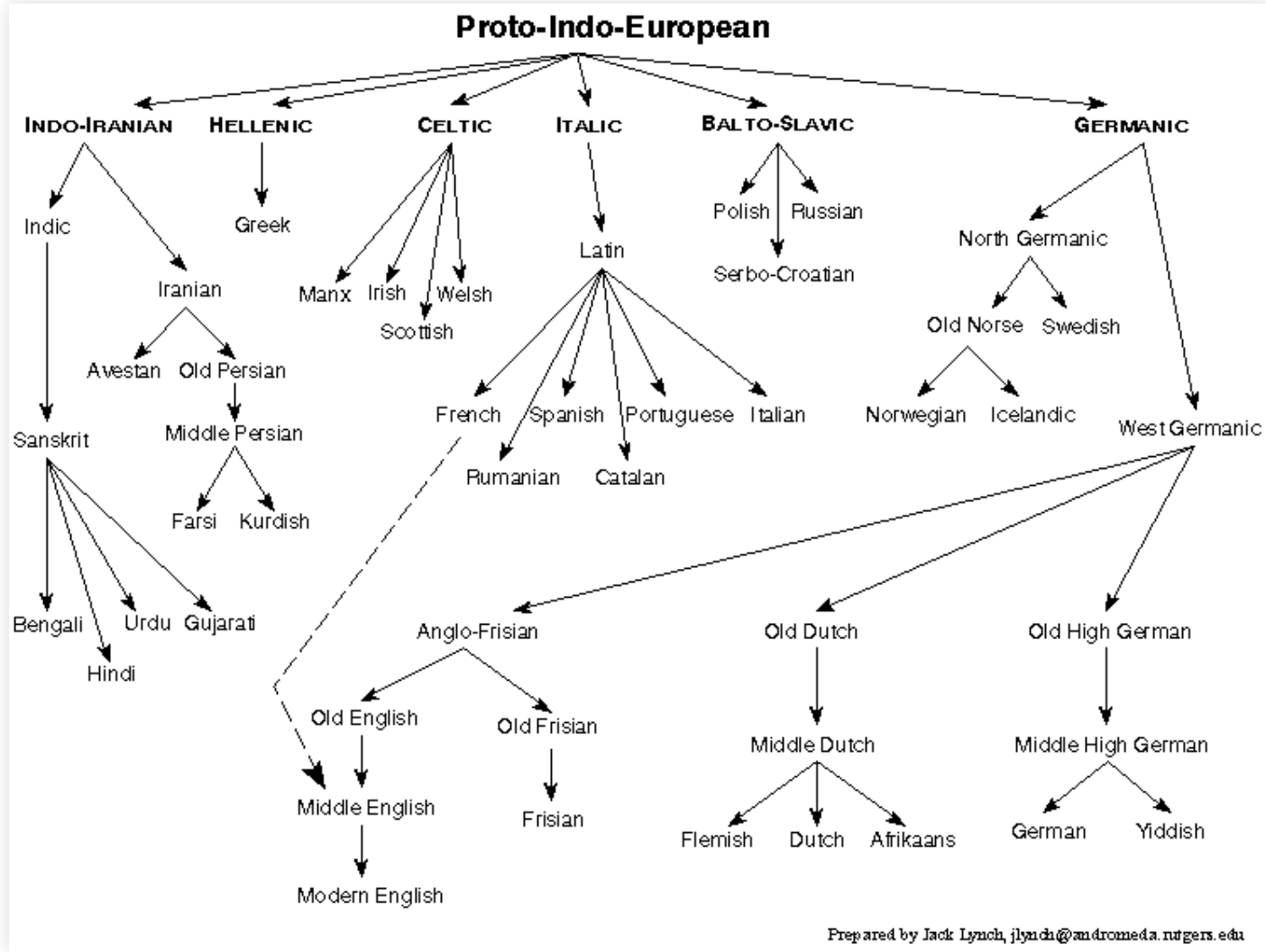
You don't have to assume any particular number of clusters.

- Any desired number of clusters can be obtained from a dendogram.
- Just “cut” the tree at the proper level.

The hierarchy itself may correspond to meaningful taxonomies:

- the tree of life
- languages
- etc.

For example


Proto-Indo-European
INDO-IRANIAN
HELLENIC
CELTIC
ITALIC
BALTO-SLAVIC
GERMANIC
Indic
Iranian
Greek
Manx
Irish
Welsh
Scottish
Latin
Polish
Russian
Serbo-Croatian
North Germanic
Old Norse
Swedish
Norwegian
Icelandic
Avestan
Old Persian
Middle Persian
Farsi
Kurdish
Sanskrit
Bengali
Urdu
Gujarati
Hindi
French
Spanish
Portuguese
Italian
Rumanian
Catalan
West Germanic
Anglo-Frisian
Old English
Old Frisian
Middle English
Frisian
Modern English
Old Dutch
Middle Dutch
Flemish
Dutch
Afrikaans
Old High German
Middle High German
German
Yiddish
Prepared by Jack Lynch, jlynch@andromeda.rutgers.edu

Hierarchical clustering

Two main types of hierarchical clustering

- Agglomerative:

    1. Start with the points as individual clusters
    2. At each step, merge the closest pair of clusters until only one cluster (or k clusters) left

- Divisive:

    1. Start with one, all-inclusive cluster
    2. At each step, split a cluster until each cluster contains a point (or there are k clusters)

Agglomerative clustering

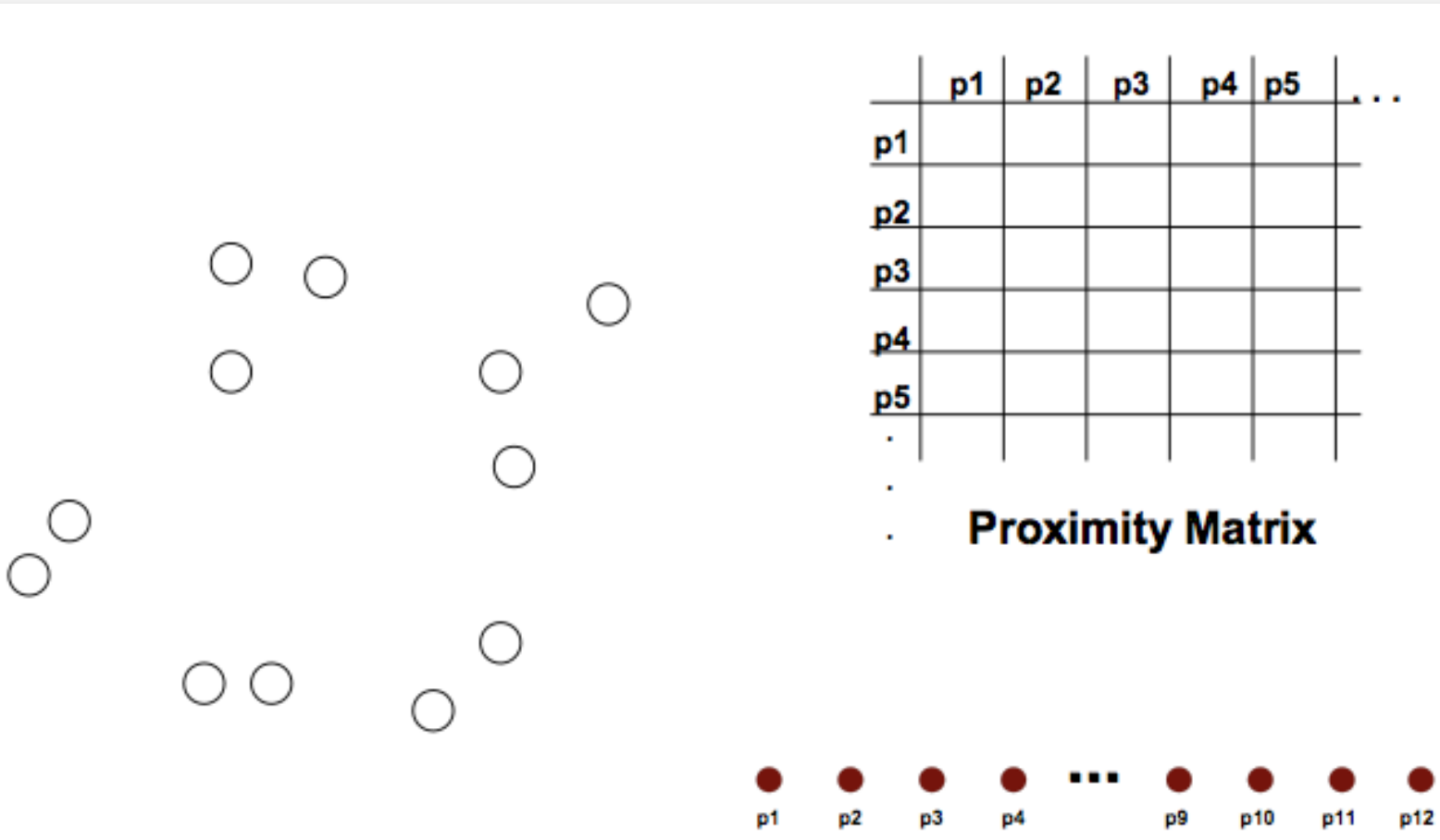
The basic algorithm is straightforward: start with every point in its own cluster and compute a proximity matrix $D(i, j)$ between every pair of points i and j.

- Then repeat:
    1. Merge the two closest clusters.
    2. Update the proximity matrix.
- Until only a single cluster remains.

Key operation is the computation of the proximity of two clusters. Different approaches to defining the distance between clusters distinguish the different algorithms.

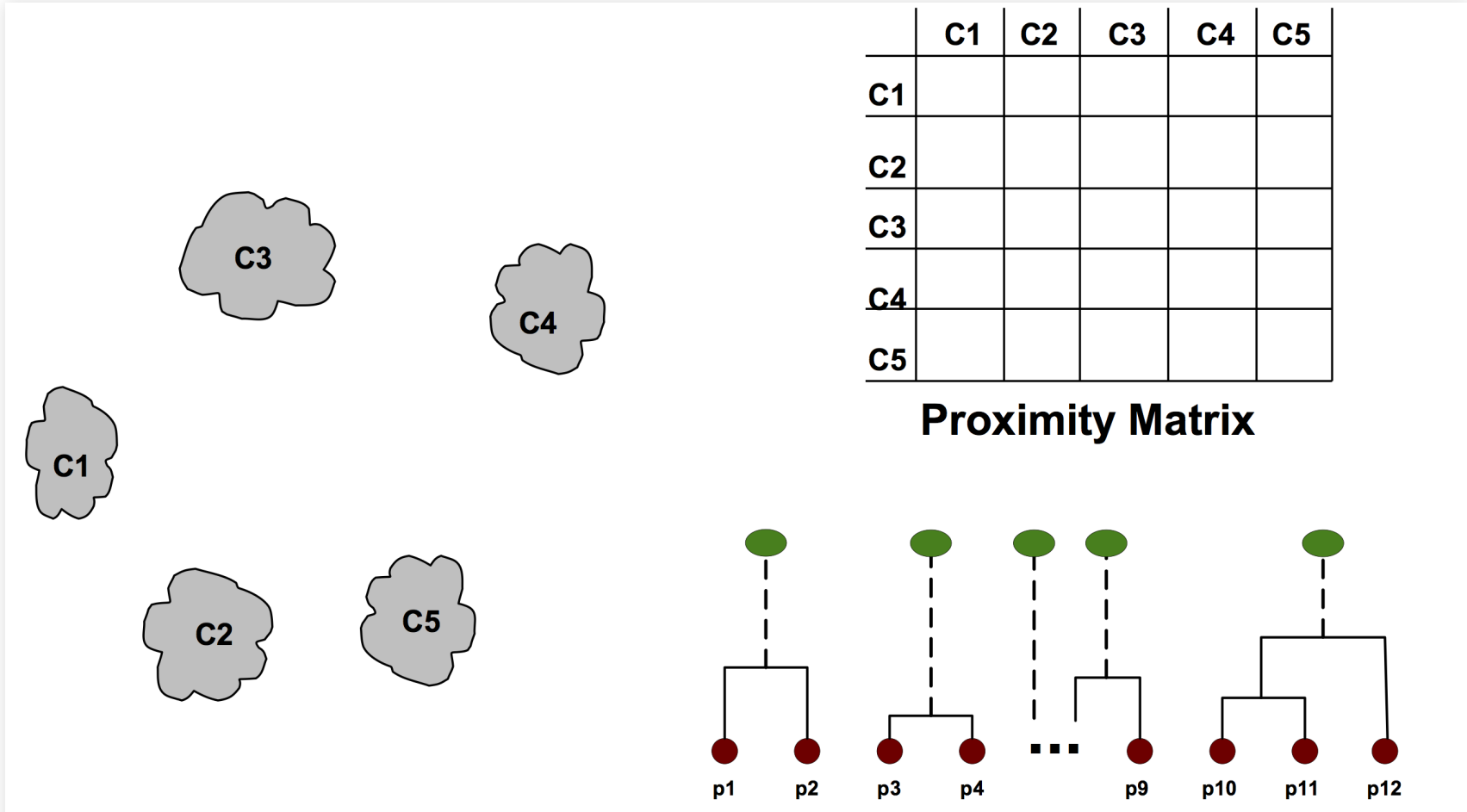
Agglomerative clustering

Start with a bunch of points and a proximity matrix:



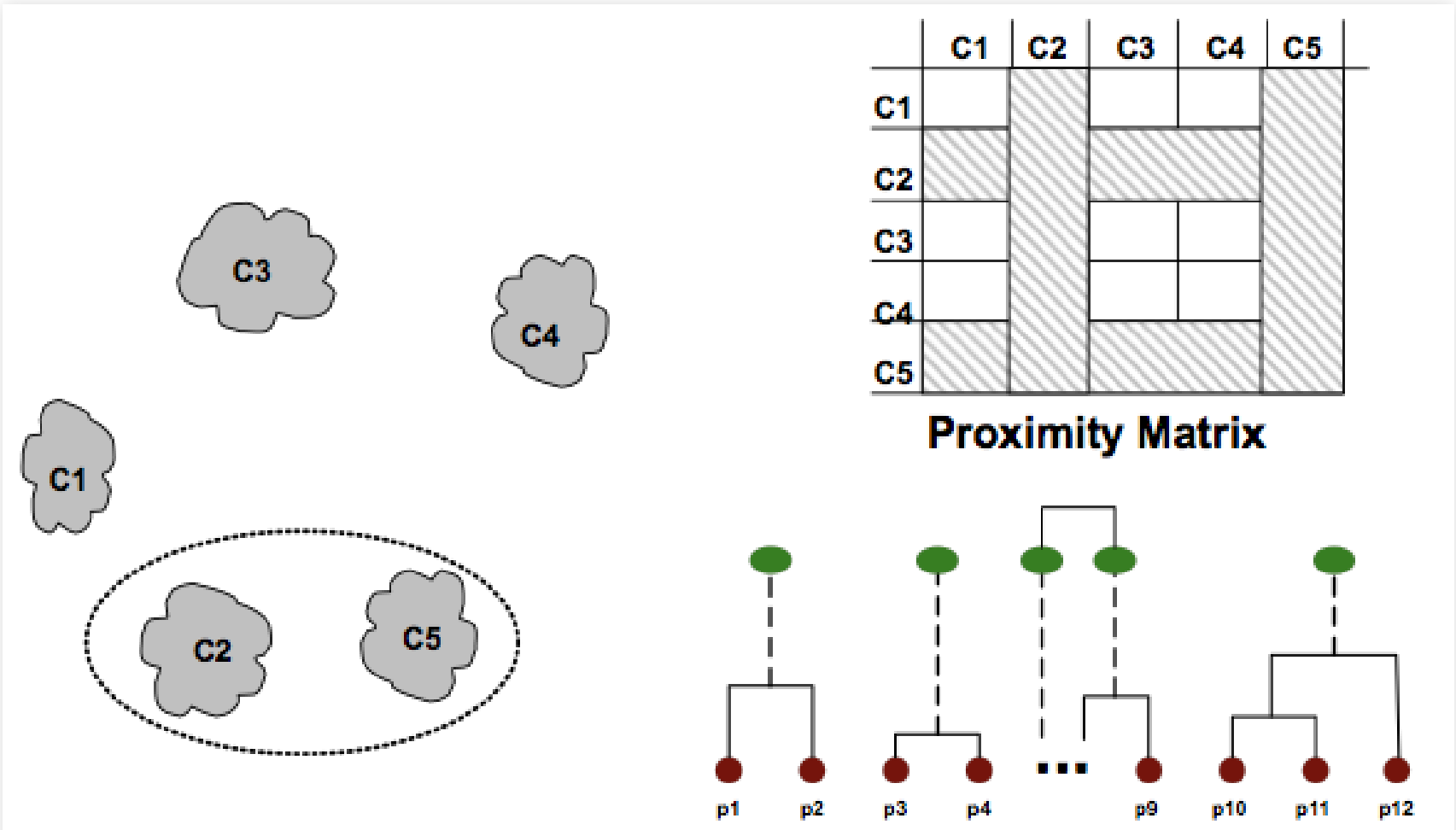
Agglomerative clustering

After some merging steps, we have a handful of clusters:



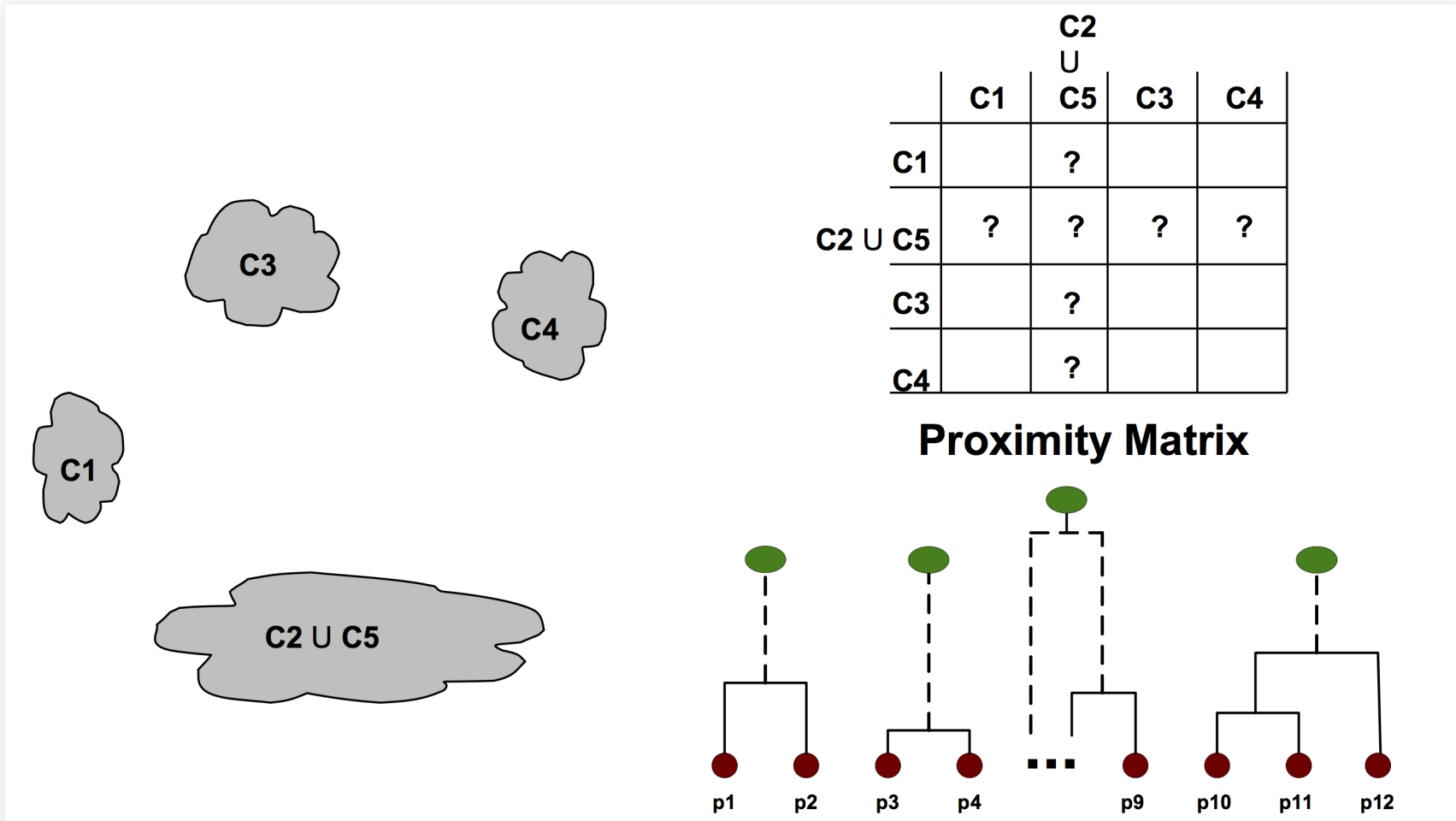
Agglomerative clustering

We want to merge C2 and C5 and update the proximity matrix:



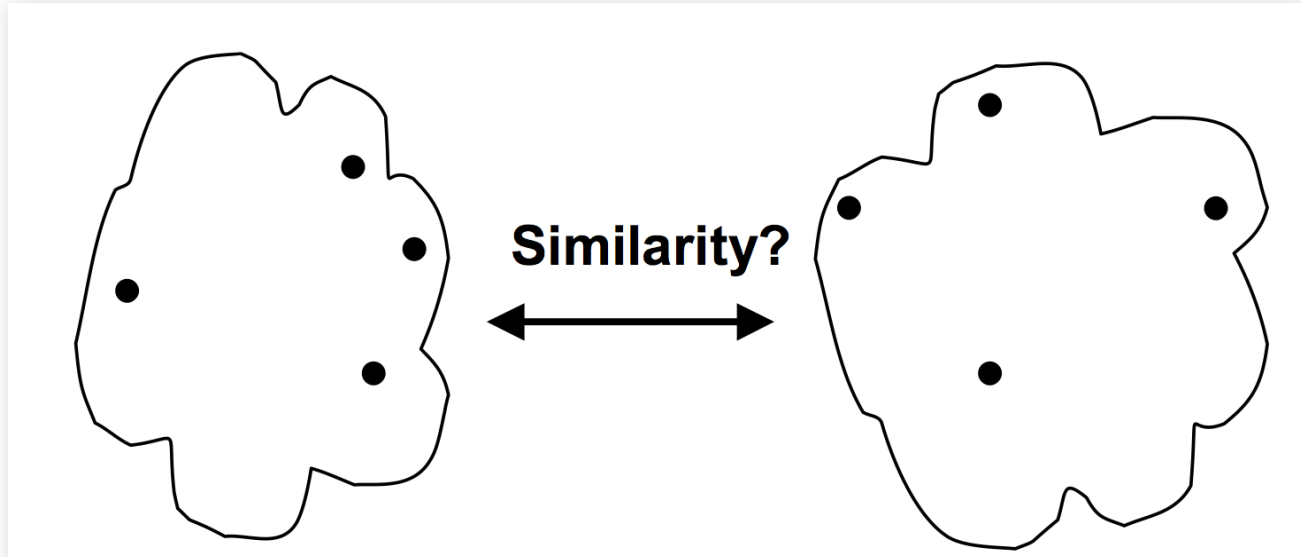
Agglomerative clustering

But how?



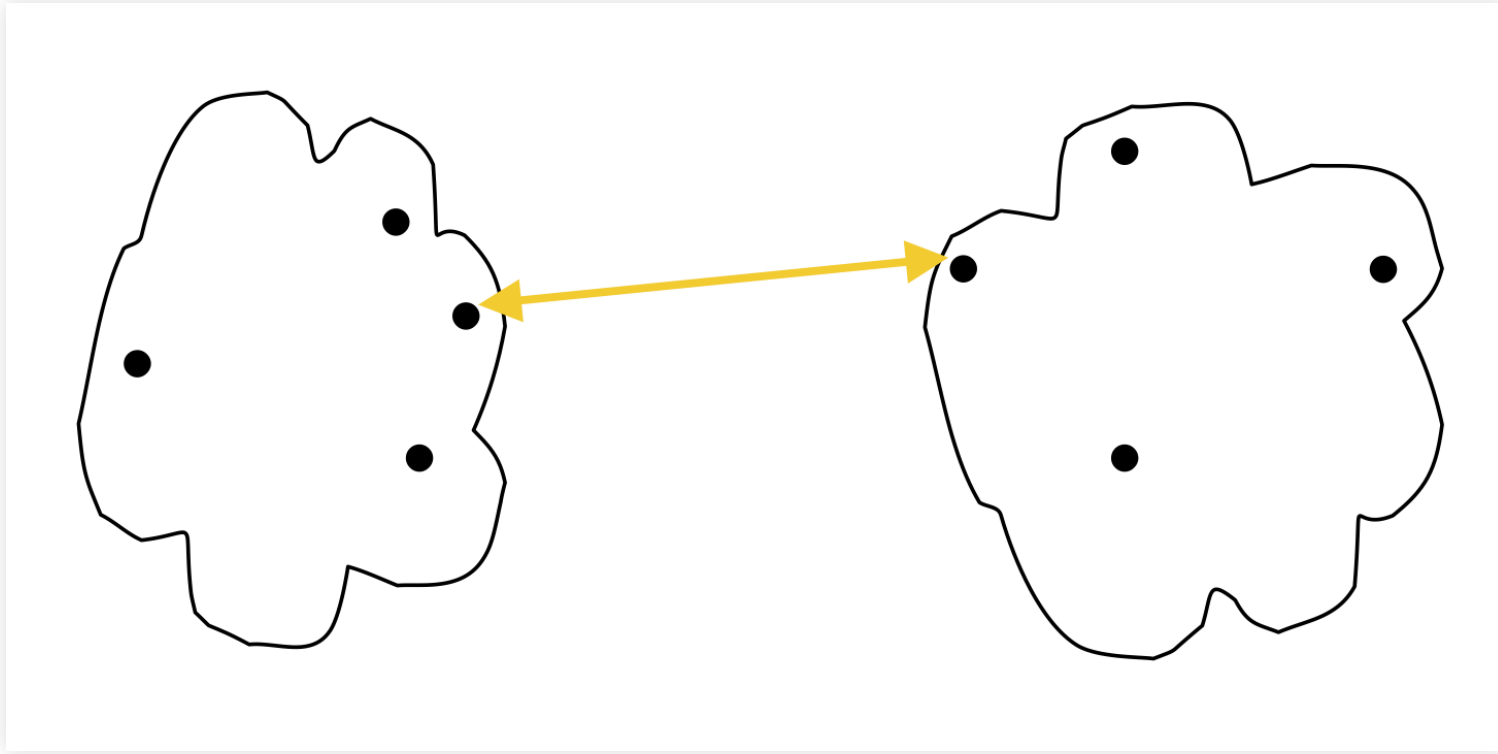
Agglomerative clustering

The key math question is: how do we measure similarity/distance between two clusters of points?



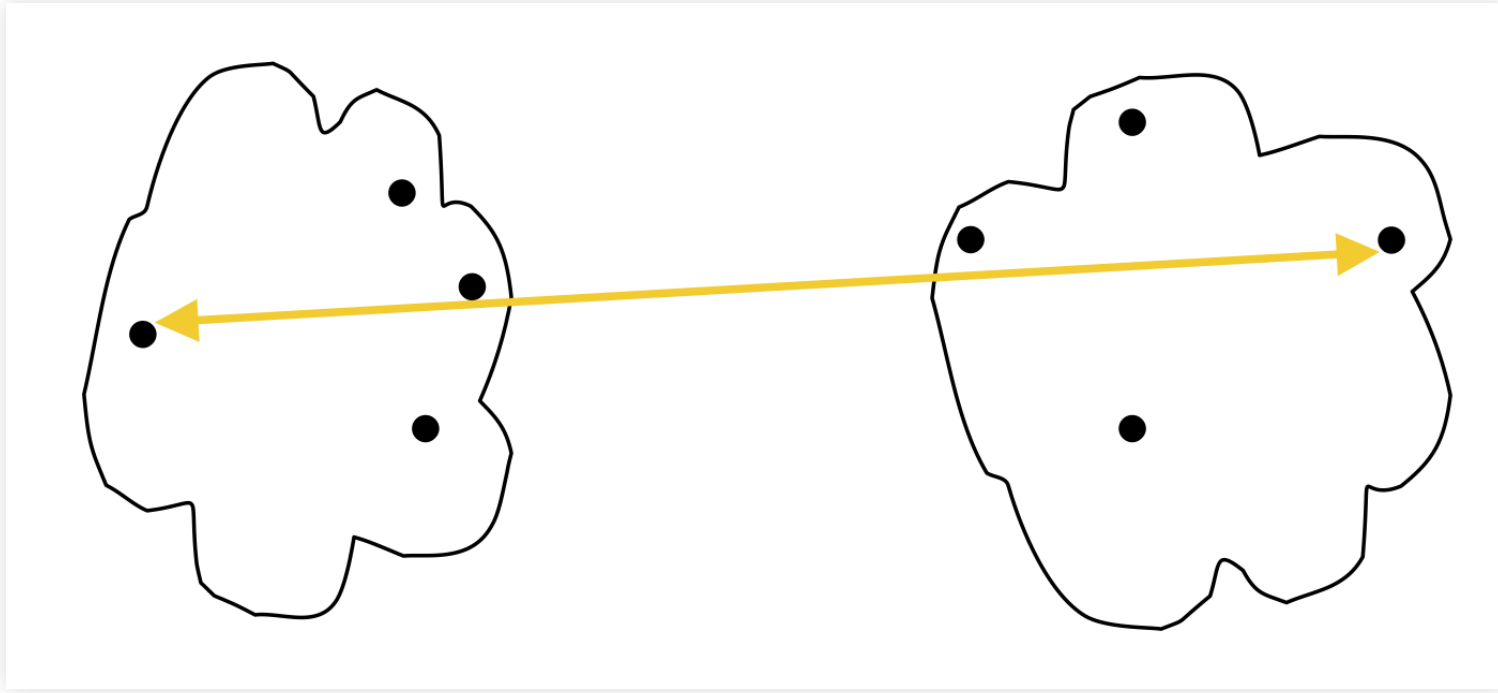
Four obvious options: min, max, average, centroids. There are called "linkage functions"

Min linkage ("single" in R)



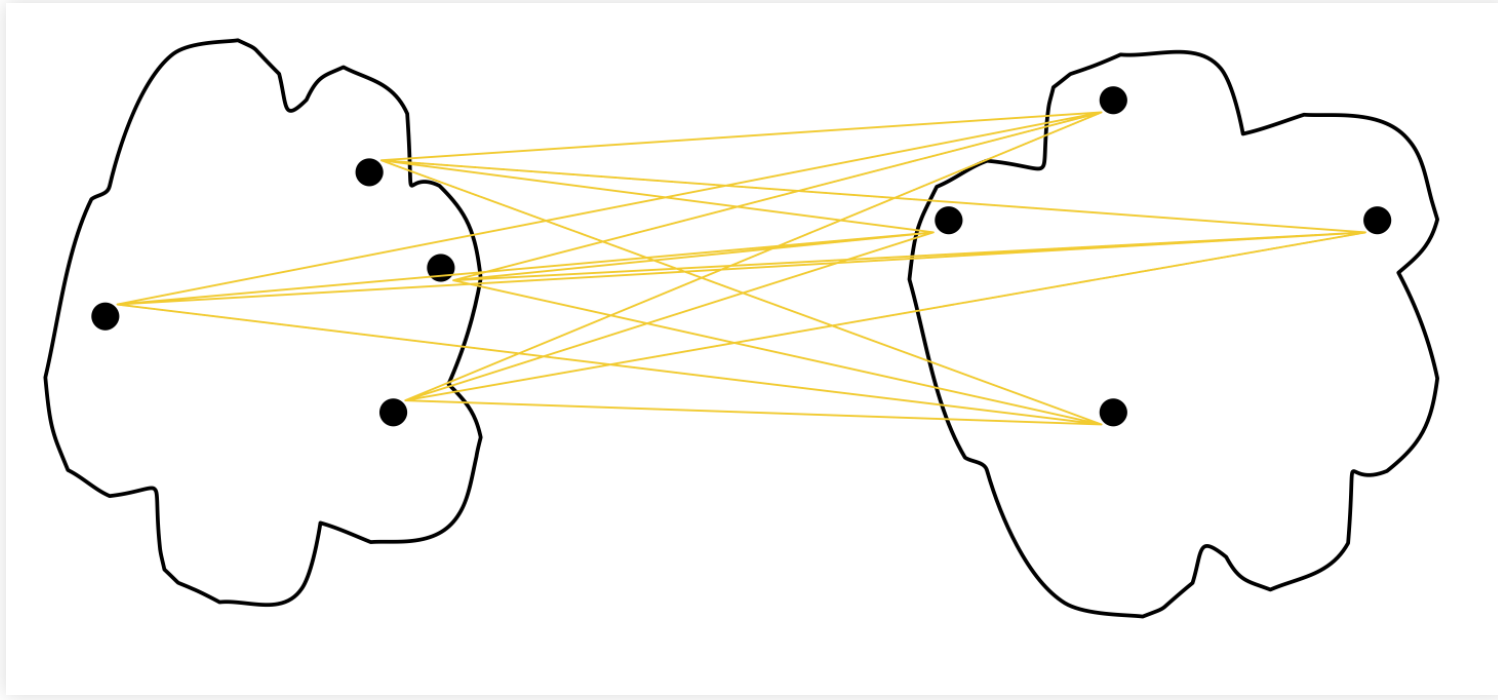
The minimum distance between two points, one in each cluster.

Max linkage ("complete" in R)



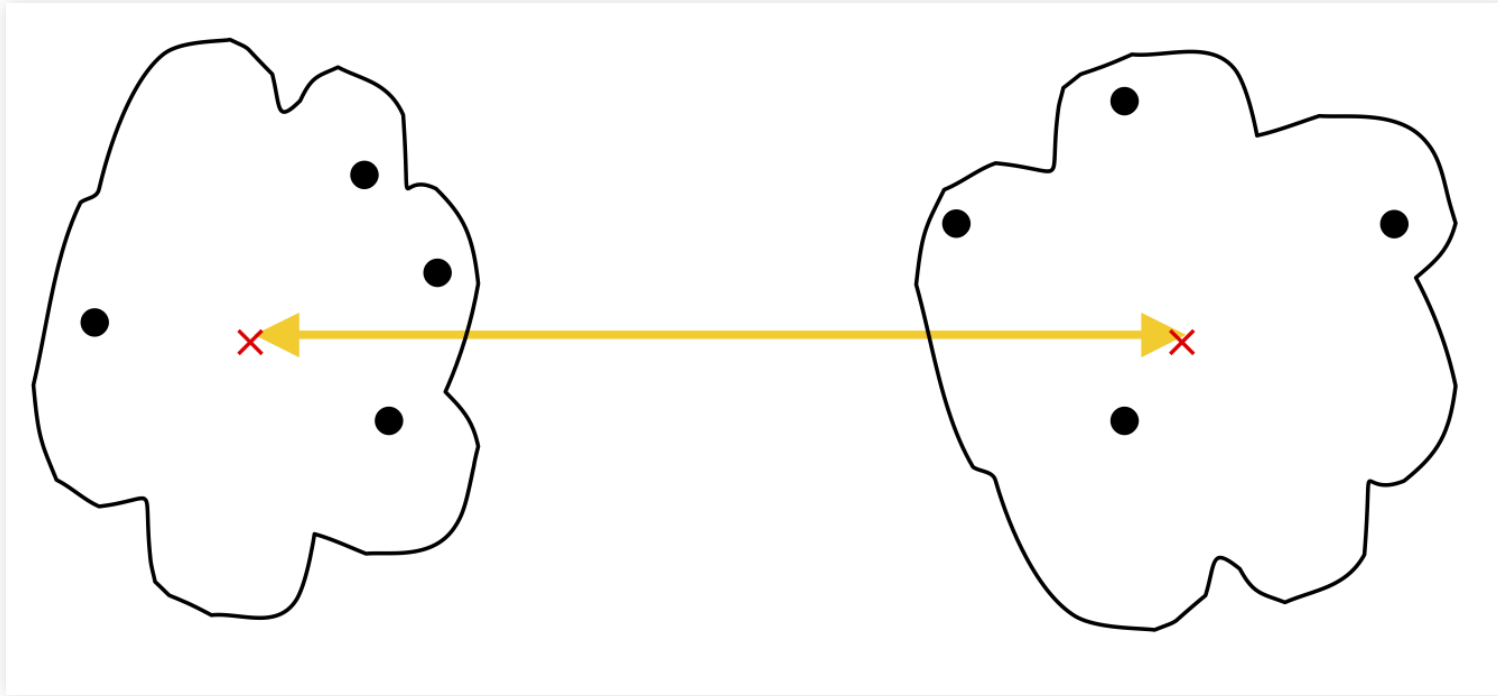
The maximum distance between two points, one in each cluster.

Average linkage ("average" in R)



The average distance between all pairs of points, one in each cluster.

Centroid linkage ("centroid" in R)



The distance between the centroids of each cluster.

Choice of linkage function

Each has its own pros and cons:

- Min: more sensitive to noise and outliers, but will leave large, obvious clusters mostly intact
- Max: more robust to noise and outliers, but tends to break up large clusters
- Average and centroid: kind of a compromise between the two

See `linkage_minmax.R` **and** `hclust_examples.R`

Agglomerative clustering: a few facts

- $O(N^2)$ space, since the proximity matrix must be stored
- Can be as slow as $O(N^3)$ in time, since there are N steps and each step might take $O(N^2)$ time to update the proximity matrix.
- All the same model-selection criteria for choosing K in K-means also work for choosing where to cut the tree in hierarchical clustering.