Unsupervised learning

David Puelz

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).

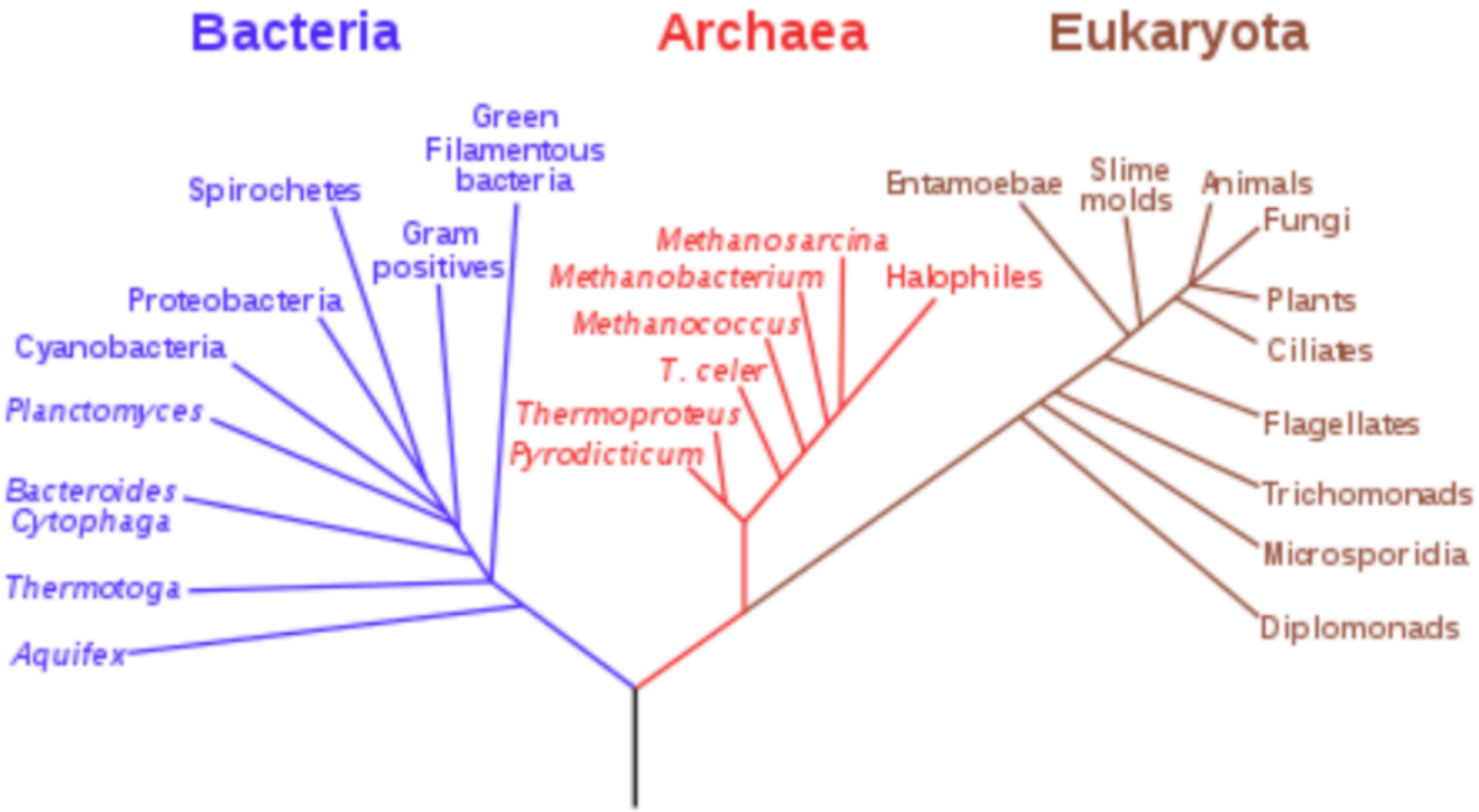
- No "mixed membership." If you encounter a Chiweenie, you need a new cluster!

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

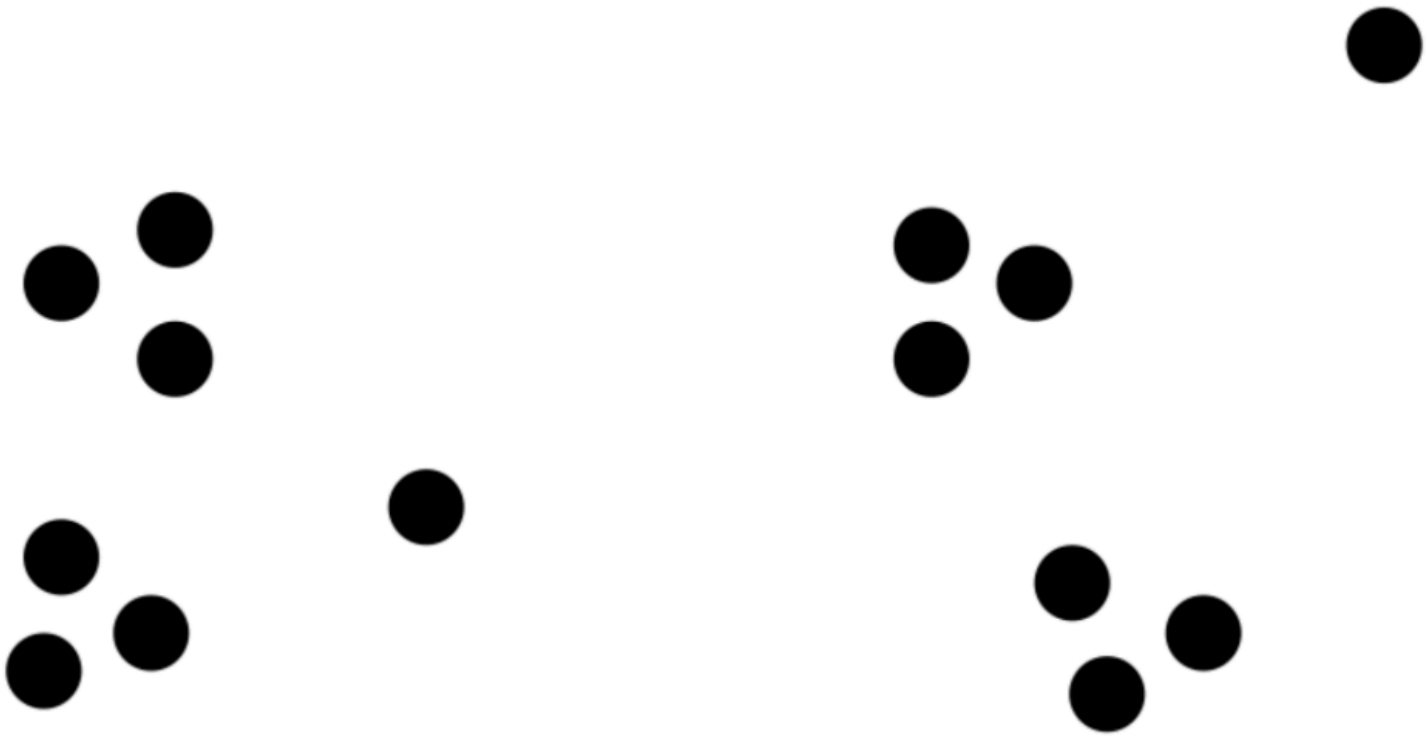
The x points can be arbitrary objects in potentially crazy spaces:

- 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

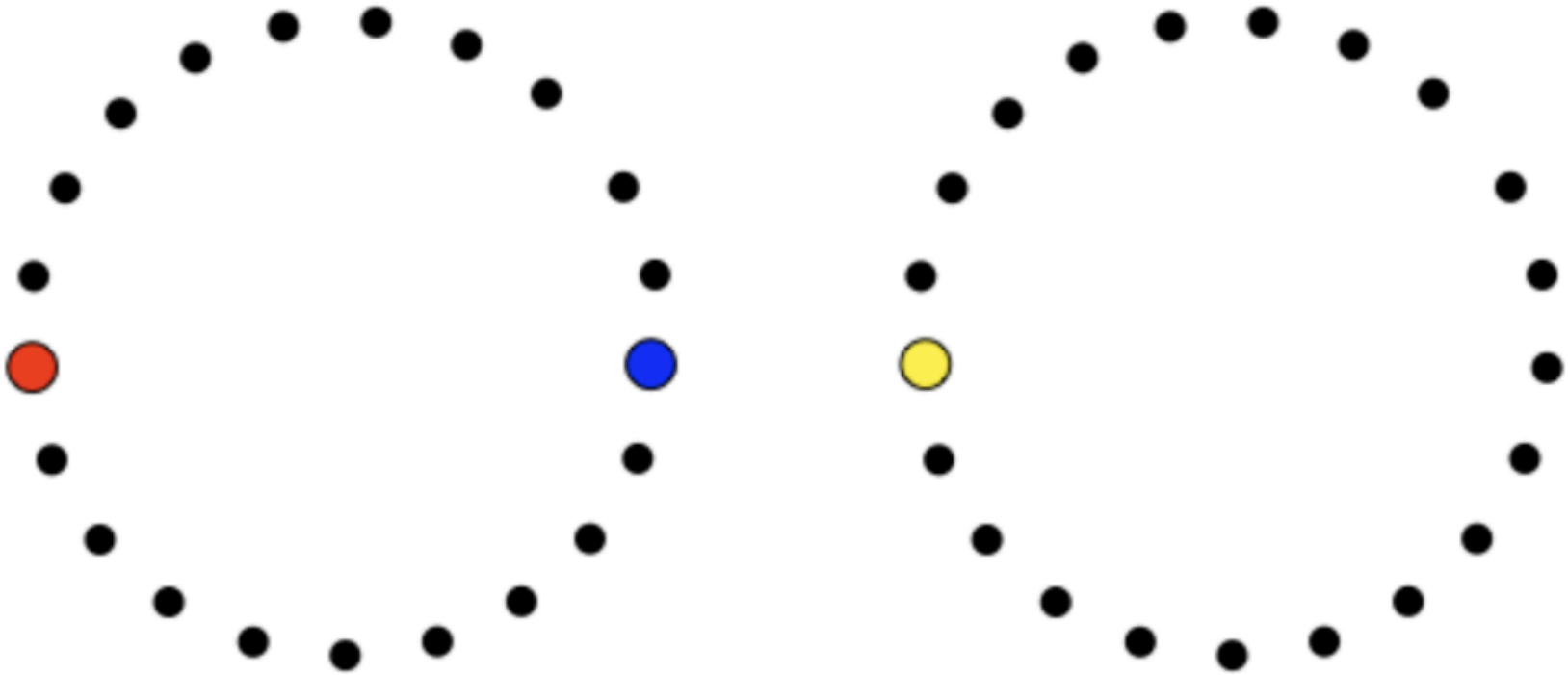
We can organize algorithms into two broad classes:

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 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)

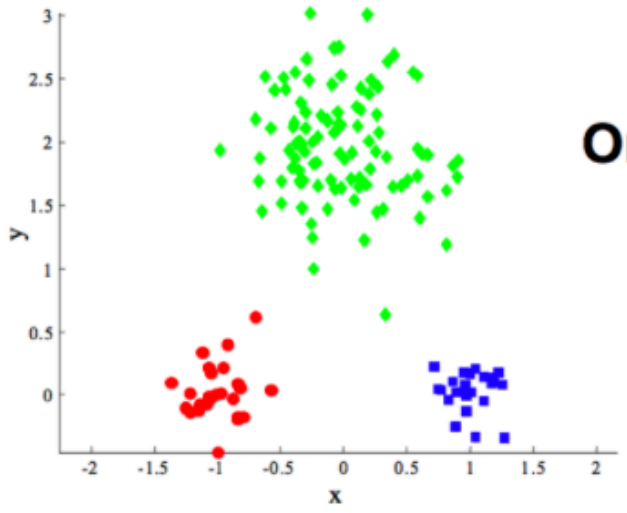
- Initial centroids are often chosen randomly. Thus the clusters produced vary from one run to another

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

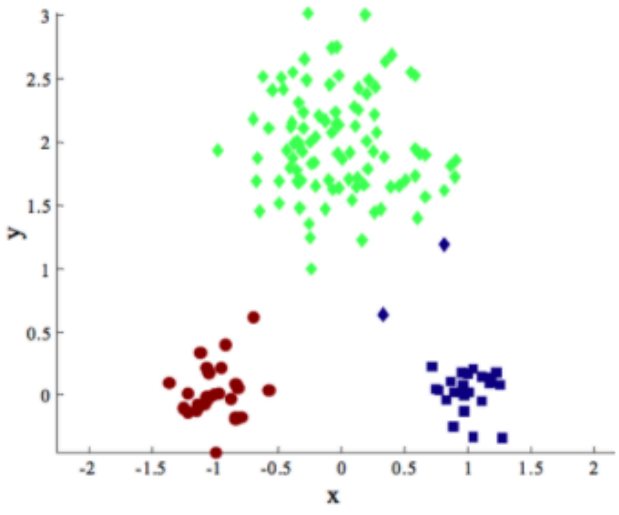
- Closeness is measured by Euclidean (default) or any valid distance

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

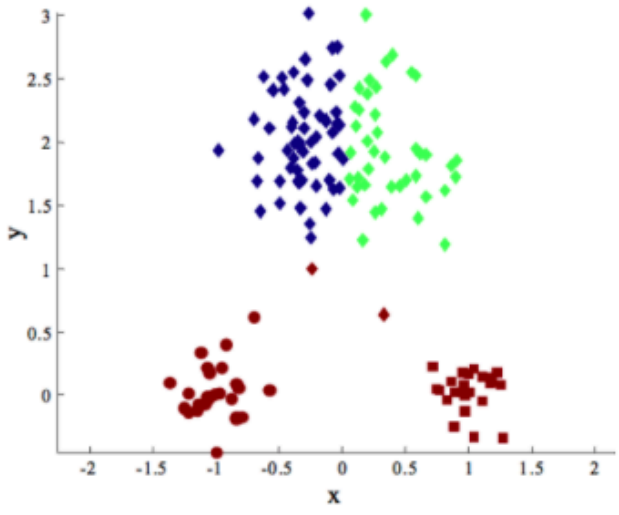
Two different k-means clusterings



Original Points

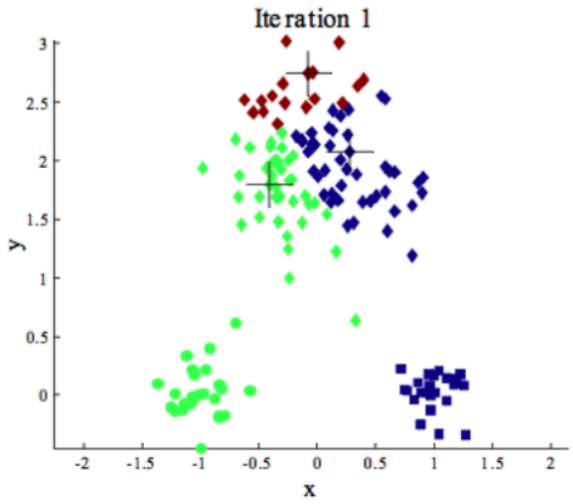


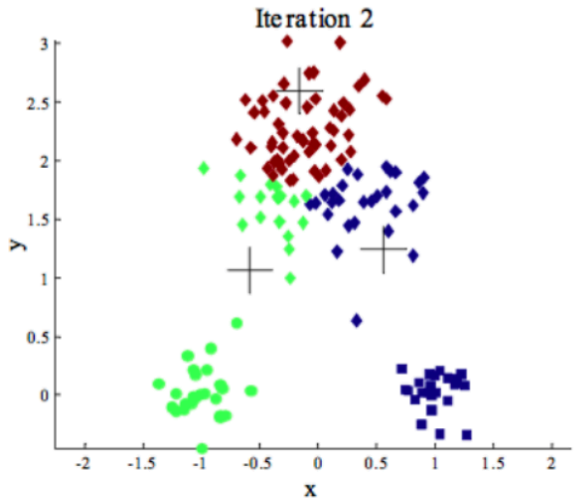
Optimal Clustering

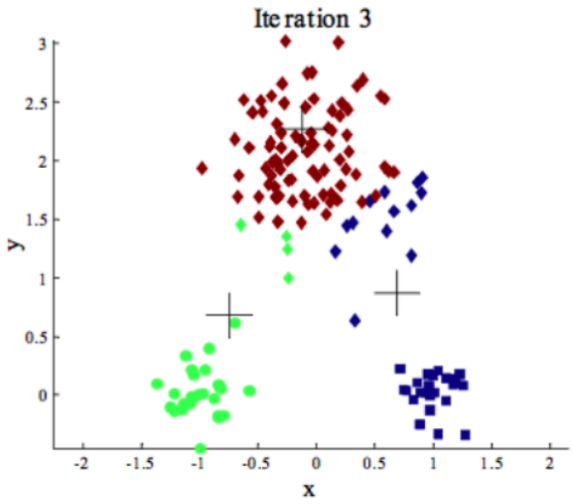


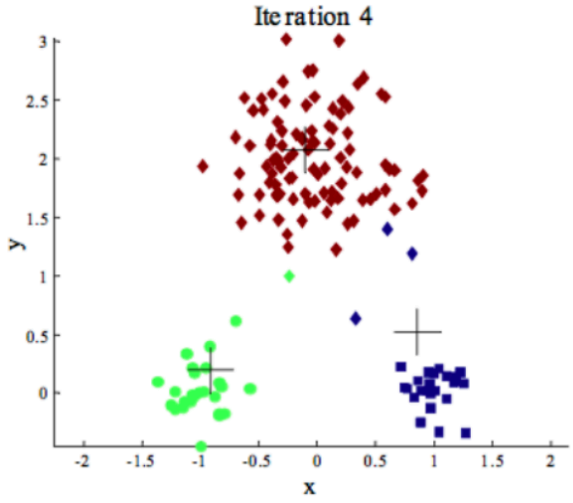
Sub-optimal Clustering

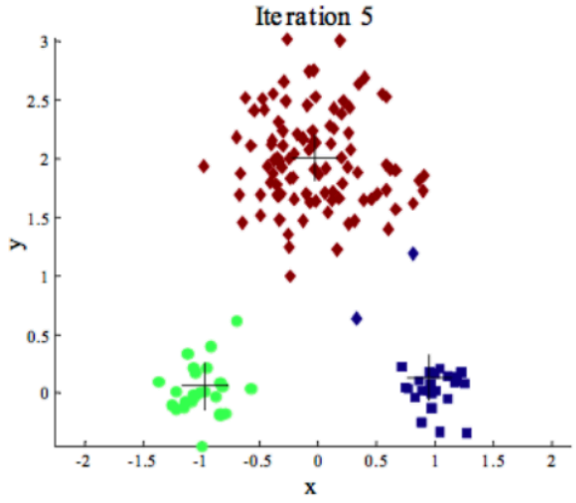
From one set of initial centroids

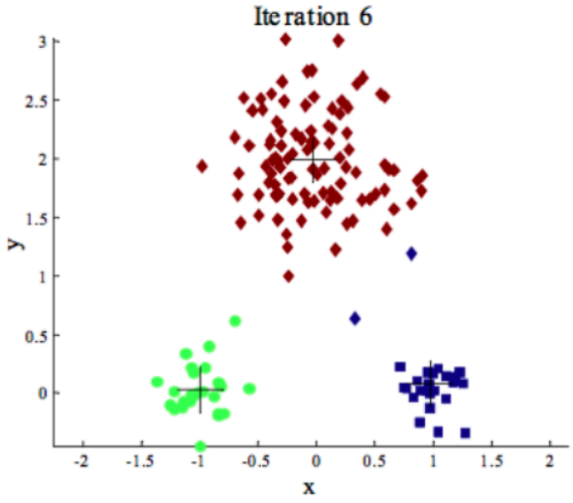




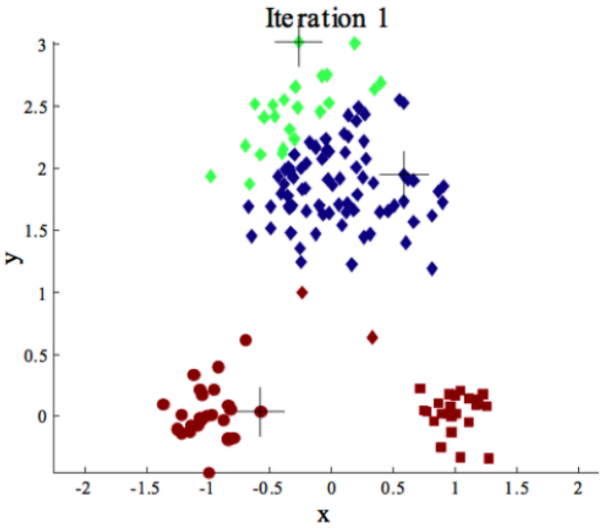


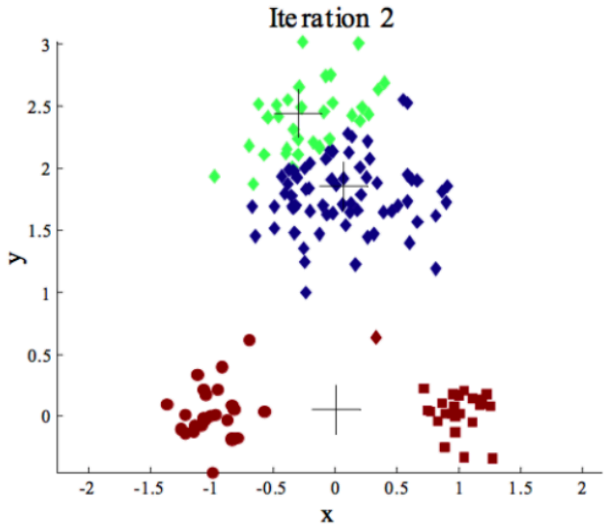


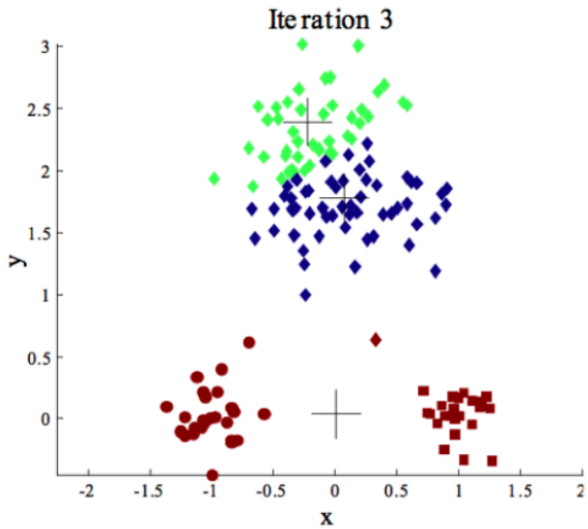


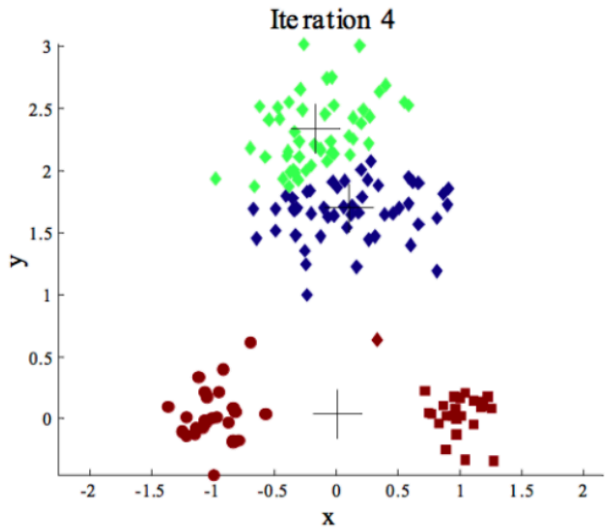


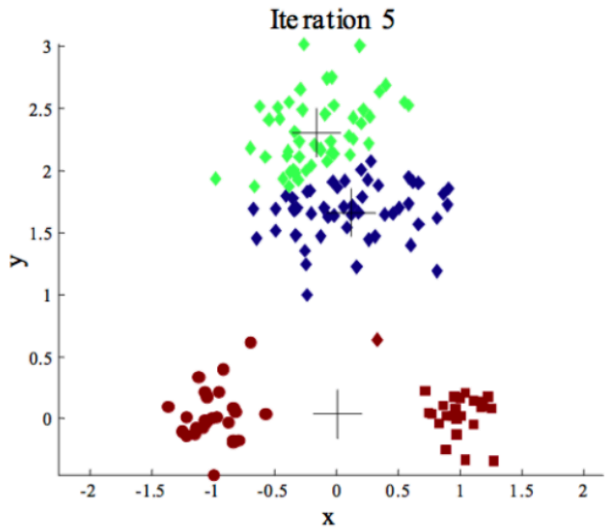
From a different set of initial centroids



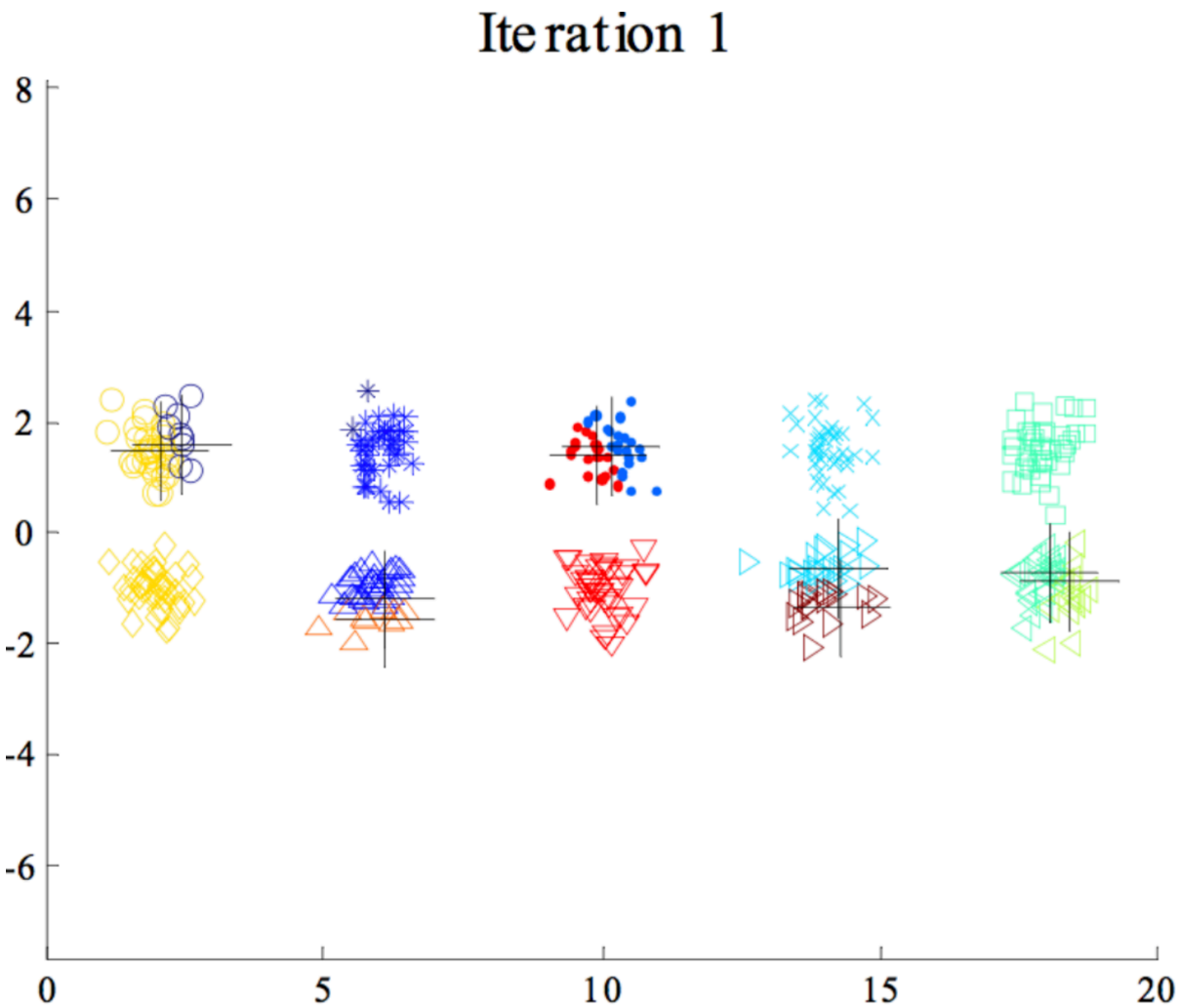






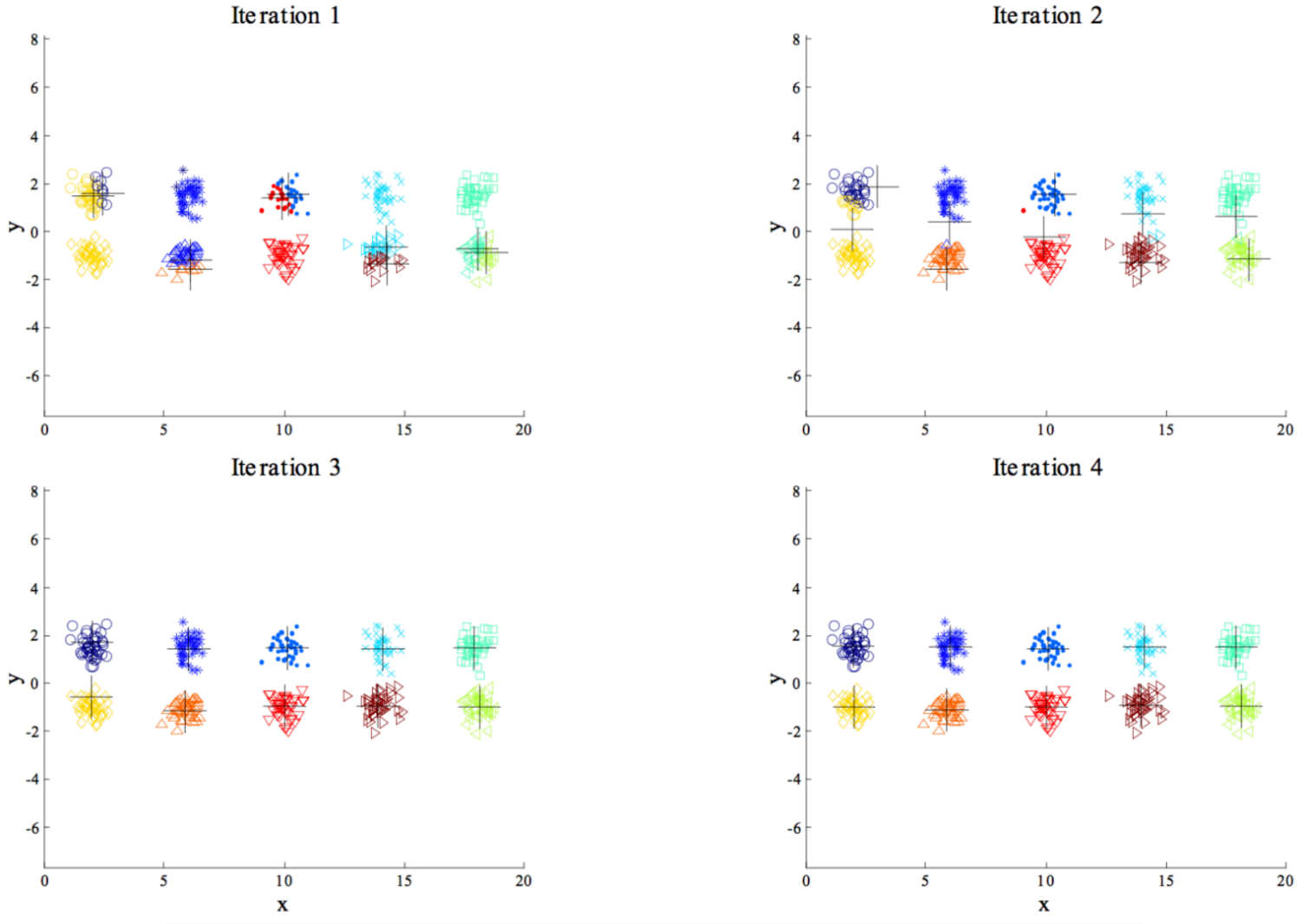


Ten initial centroids



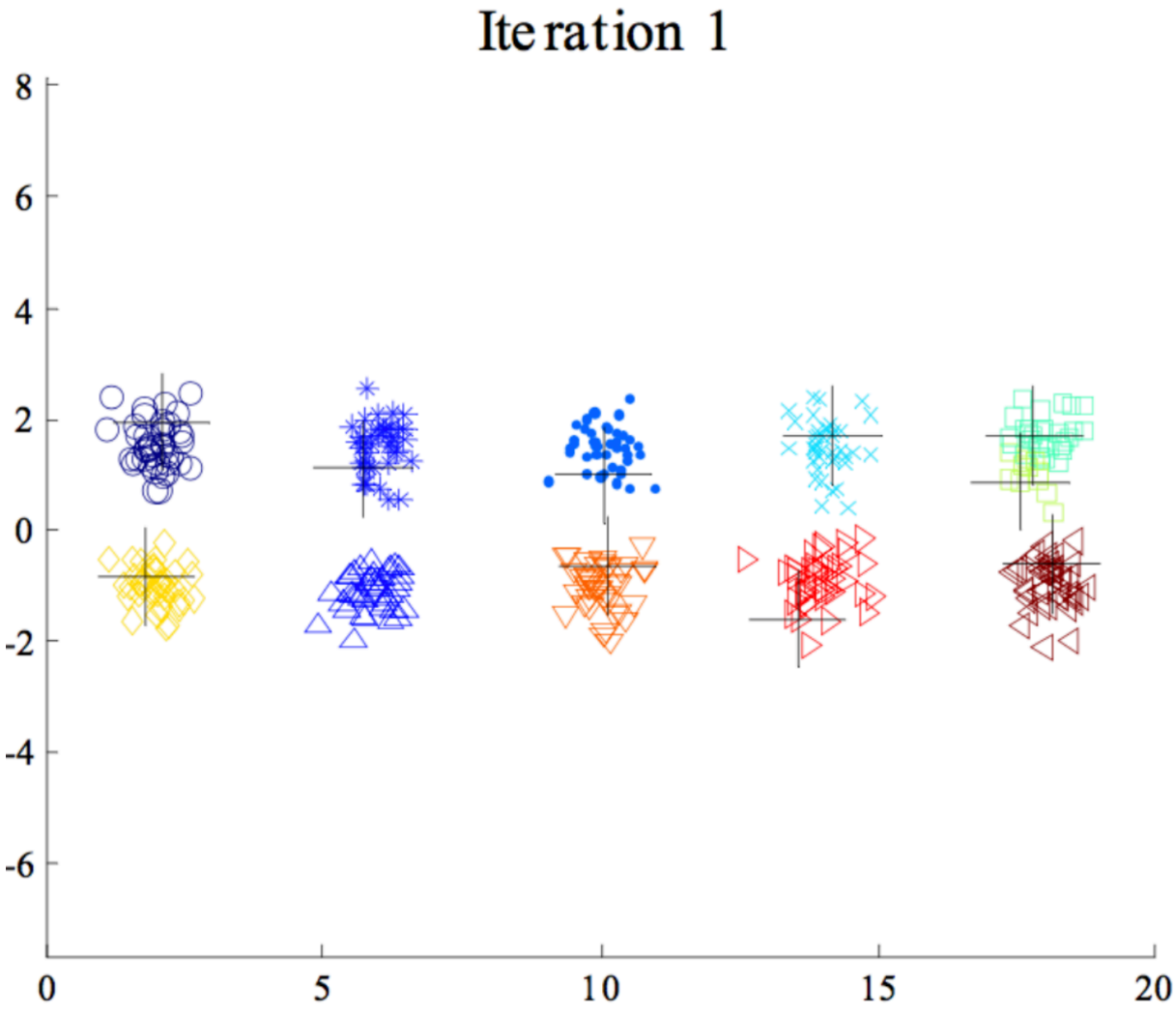
Each pair of clusters has two initial centroids

Ten initial centroids



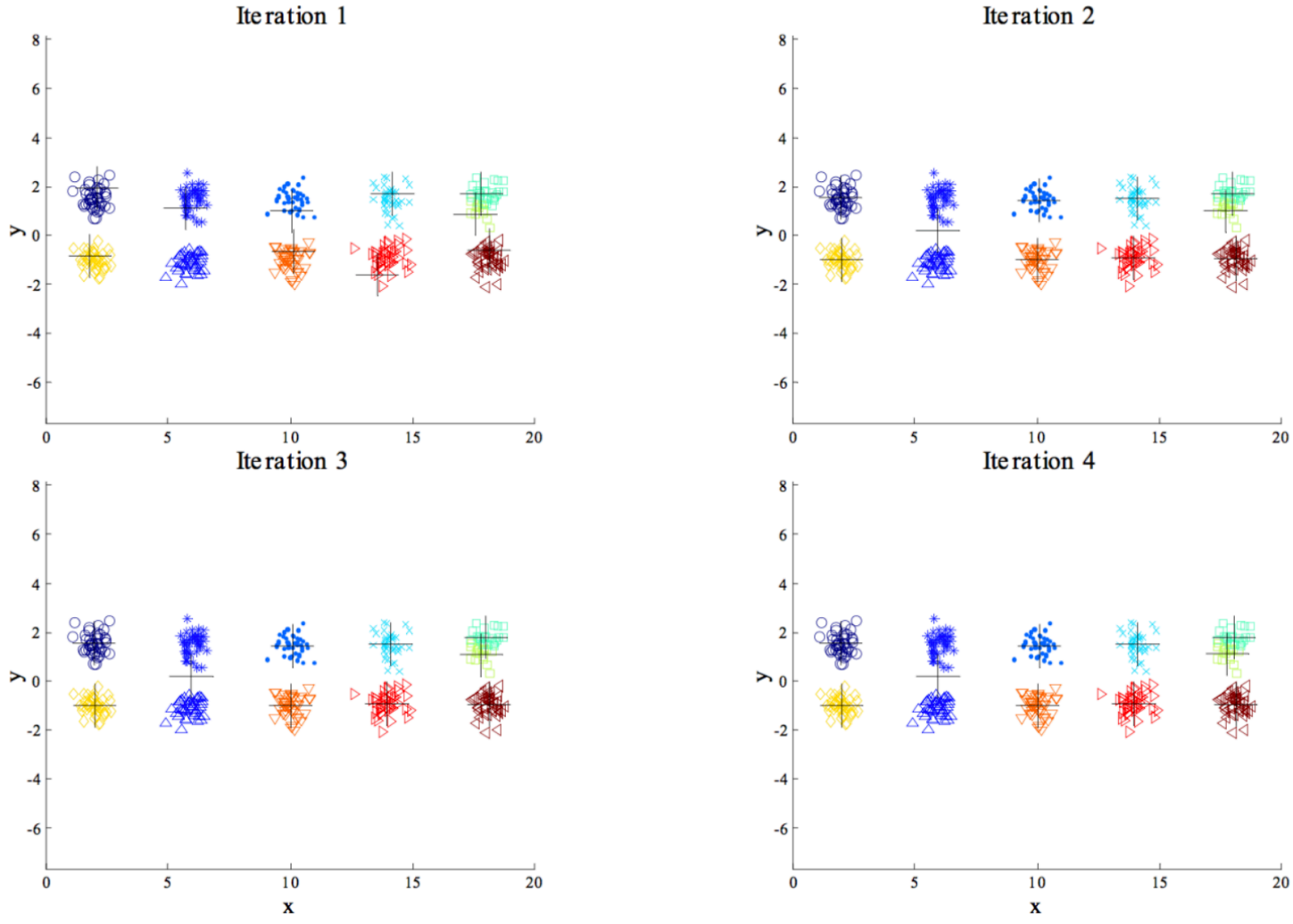
Not bad!

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:

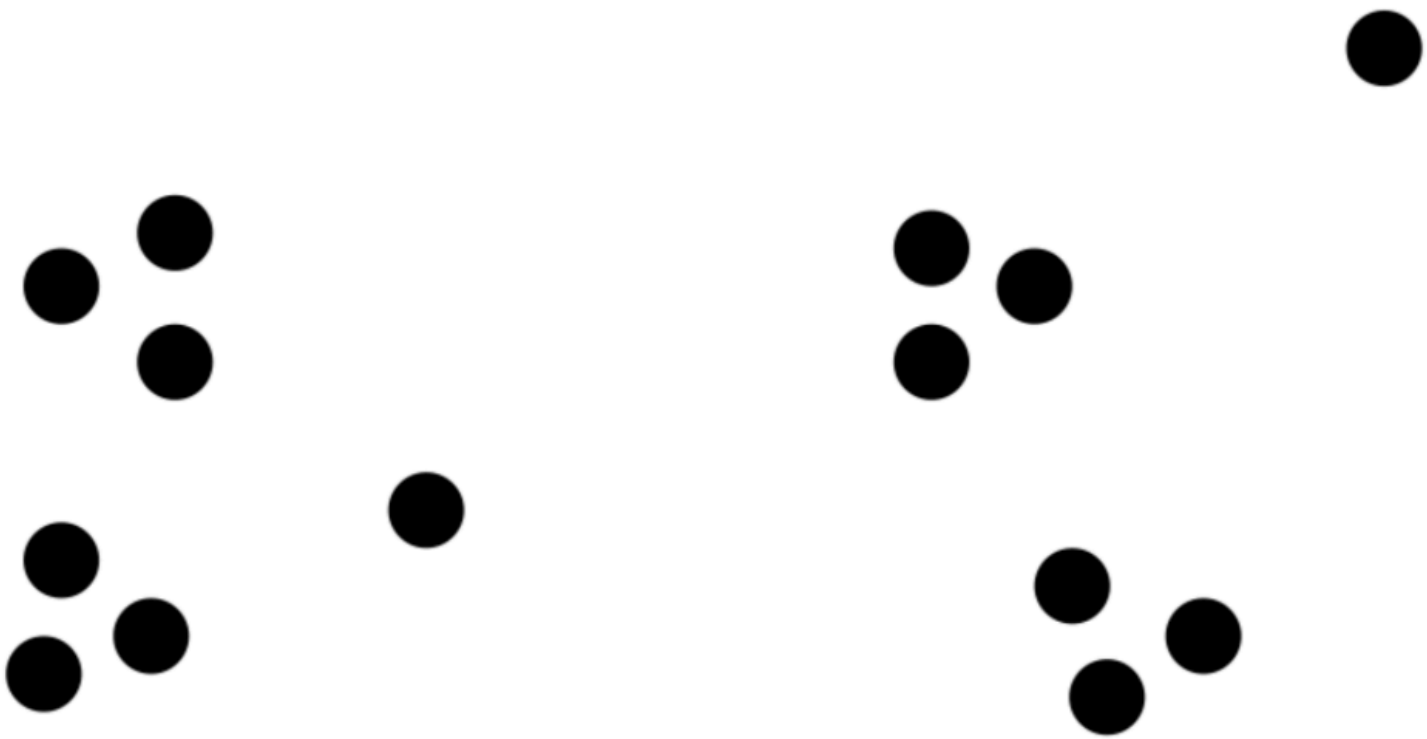
- It basically never finds a global optimizer, except in simple problems.
- There are too many possible clusterings to check them all!

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

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

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

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.

Choosing K

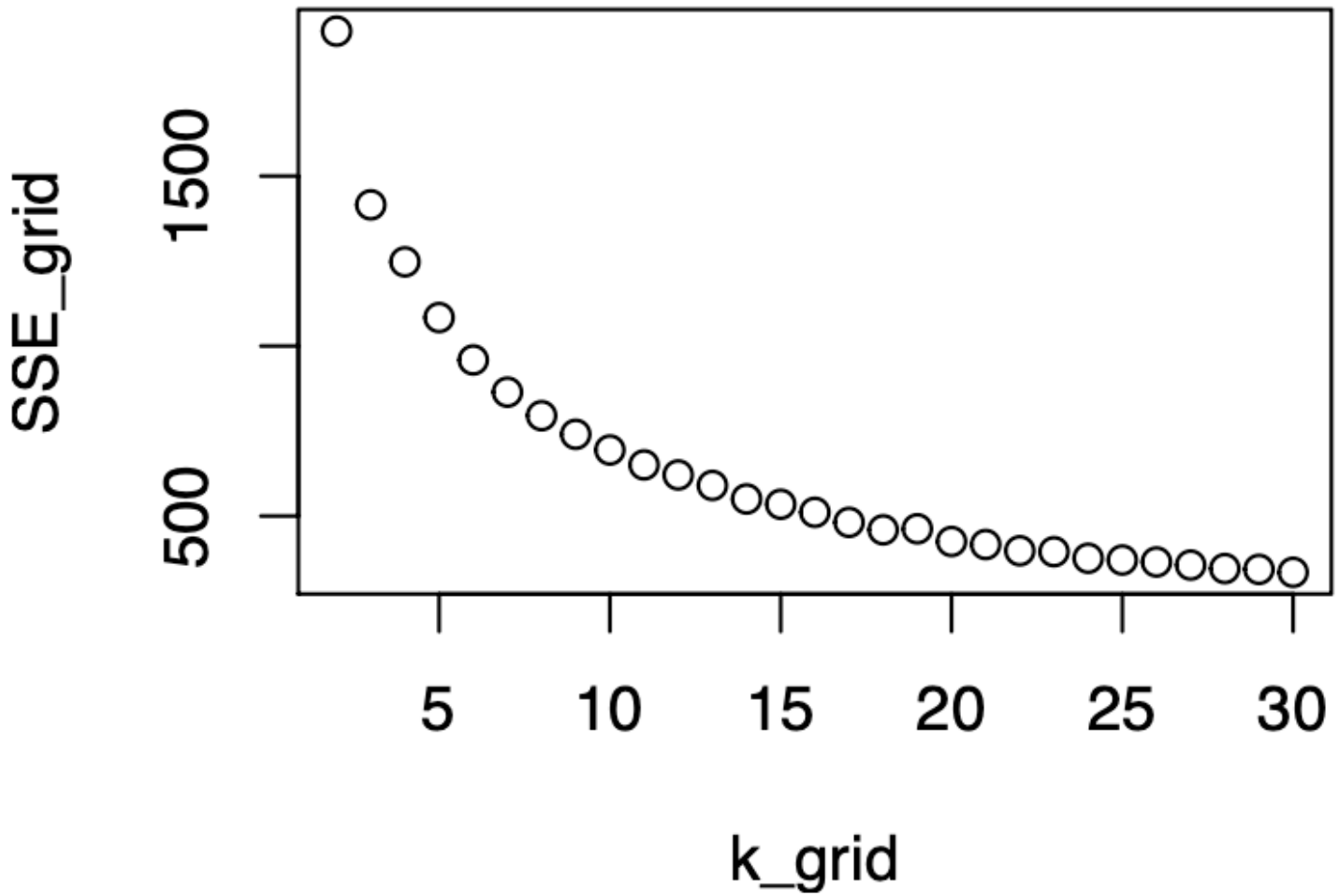
Let's try many values of K for the `cars` data ...

```
library(foreach)
cars = read.csv('data/cars.csv')
cars = scale(cars[,10:18]) # cluster on measurables
k_grid = seq(2, 30, by=1)
SSE_grid = foreach(k = k_grid, .combine='c') %do% {
  cluster_k = kmeans(cars, k, nstart=50)
  cluster_k$tot.withinss
}
```

What do we expect this plot to look like?

To the board!

What does the plot actually look like?



Do you see an elbow?

Elbow plots: Reality

A lot of elbow plots I've seen look something like this. It depends on the data whether or not they will be useful.

Elbow plots and other statistics

There are a couple other statistics one can look at to discern a reasonable value for K:

$\rightarrow$ CH statistic

$\rightarrow$ Gap statistic

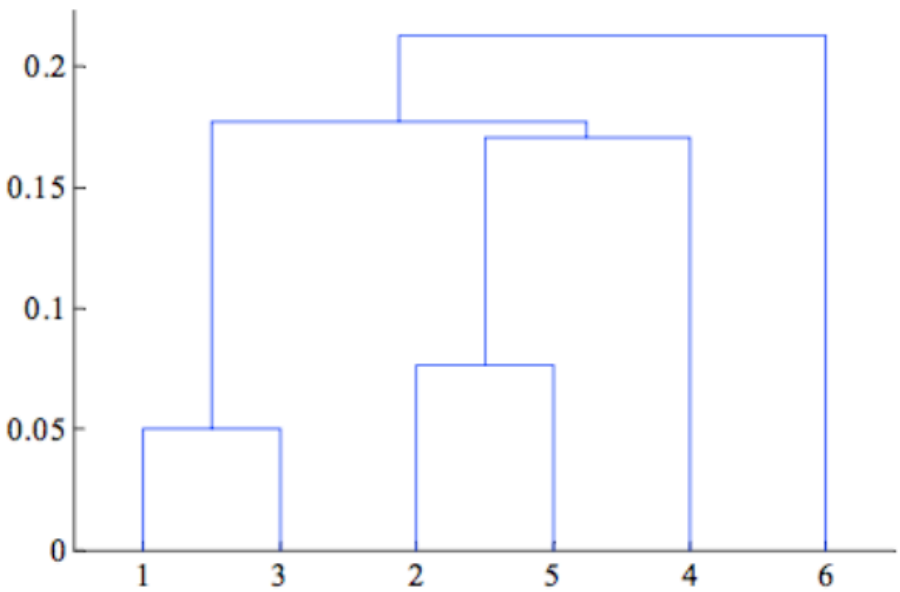
Both of these are potentially useful. If you're interested, you can read more about them in the `R` documentation. I just want you to know that they exist!

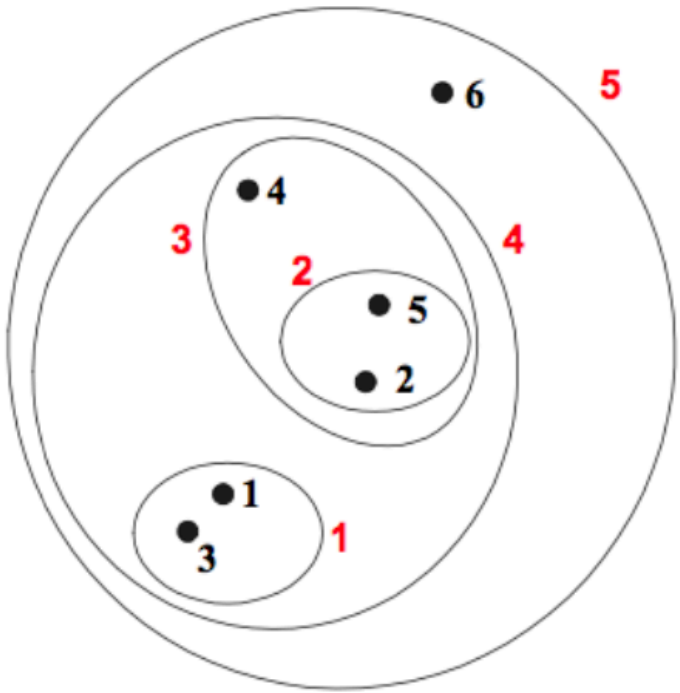
Hierarchical clustering

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 dendrogram.

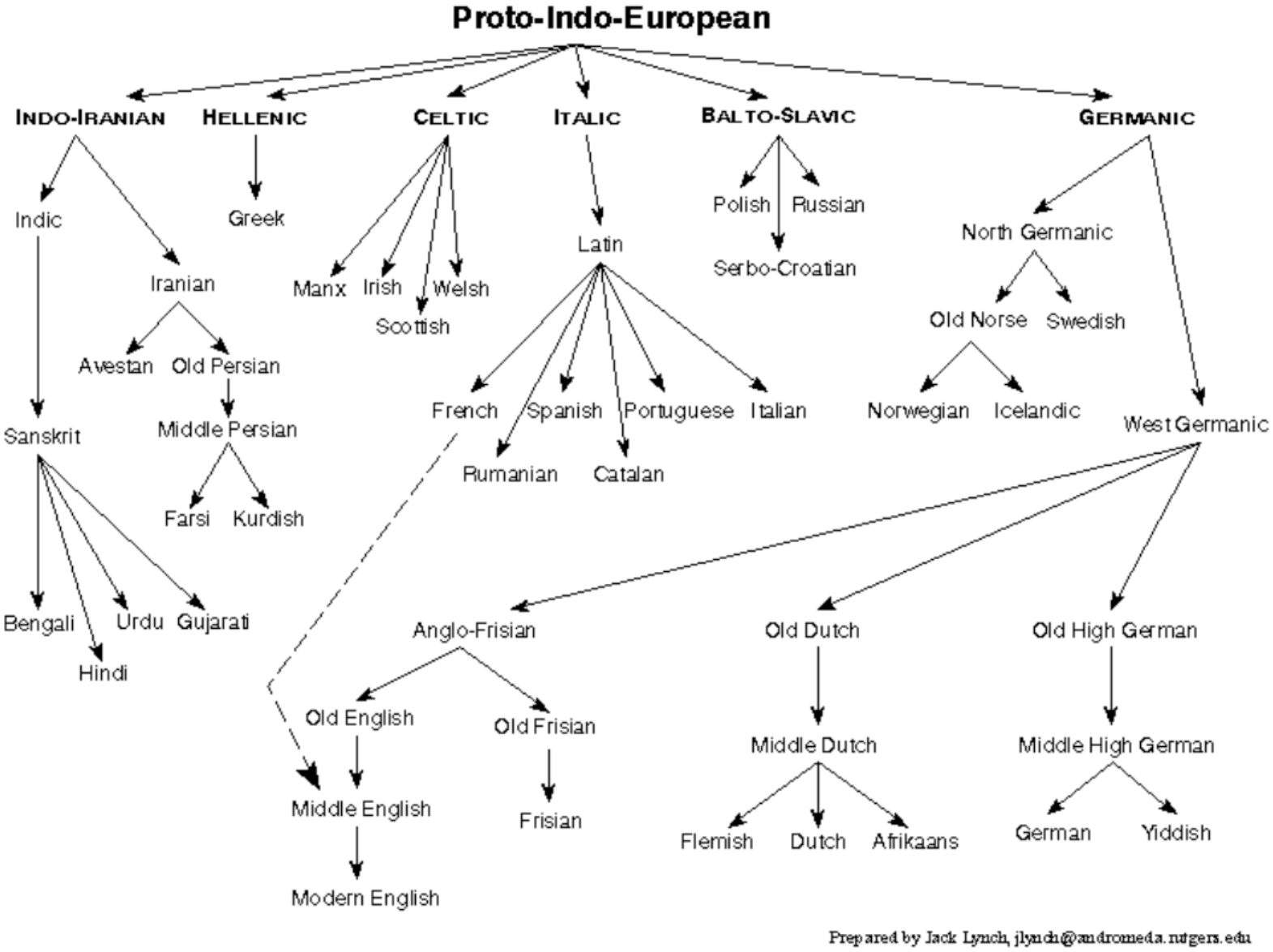
- Just "cut" the tree at the proper level.

The hierarchy itself may correspond to meaningful taxonomies:

- the tree of life

- languages

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:

Agglomerative (bottom up):

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 (top down):

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:

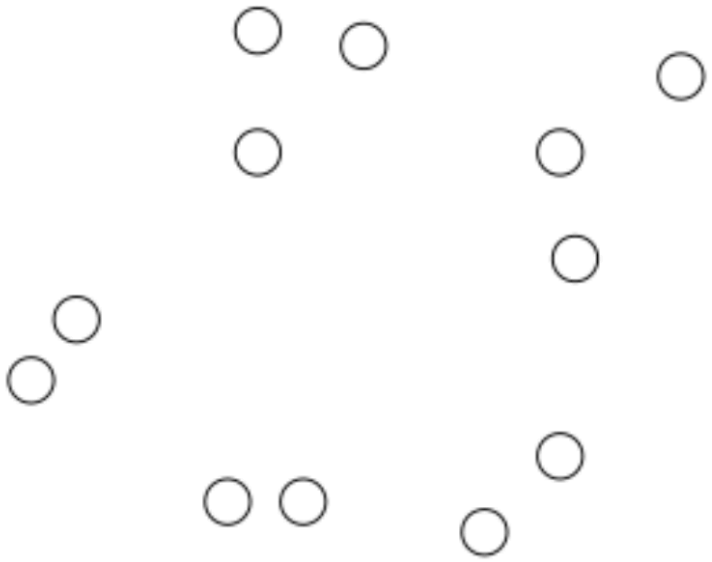
1. Merge the two closest clusters.
2. Update the proximity matrix.

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.

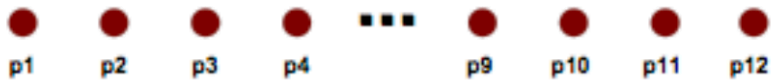
Agglomerative clustering

Start with a bunch of points and a proximity matrix:



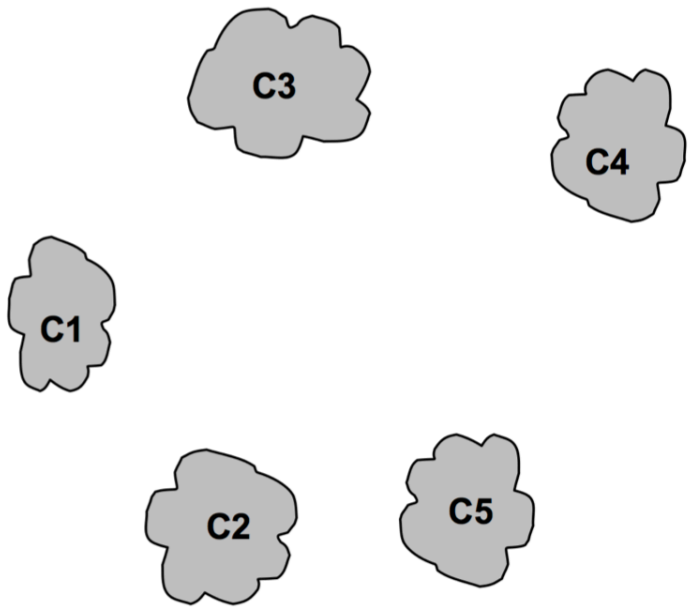
	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix



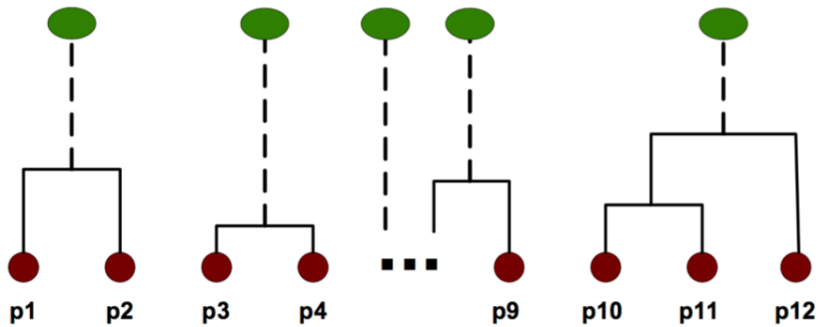
Agglomerative clustering

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



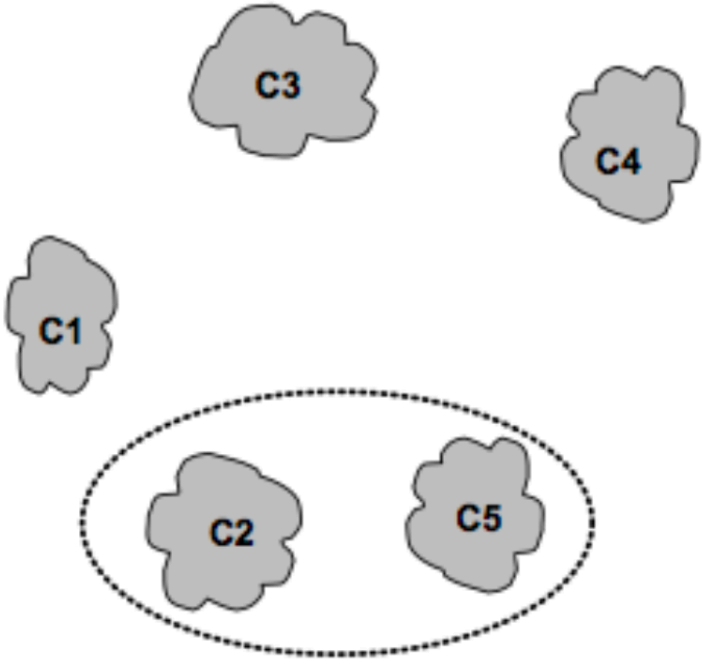
	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

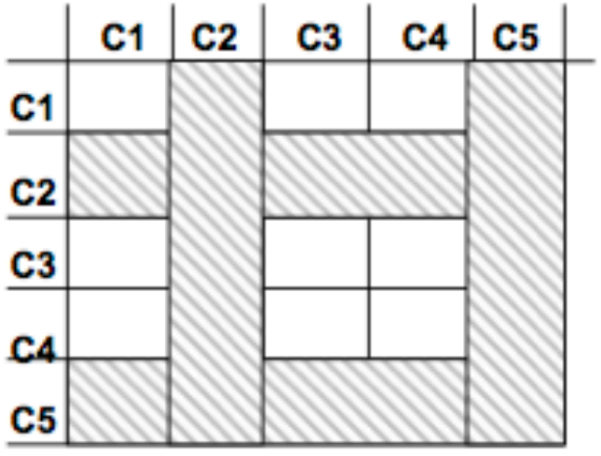
Proximity Matrix



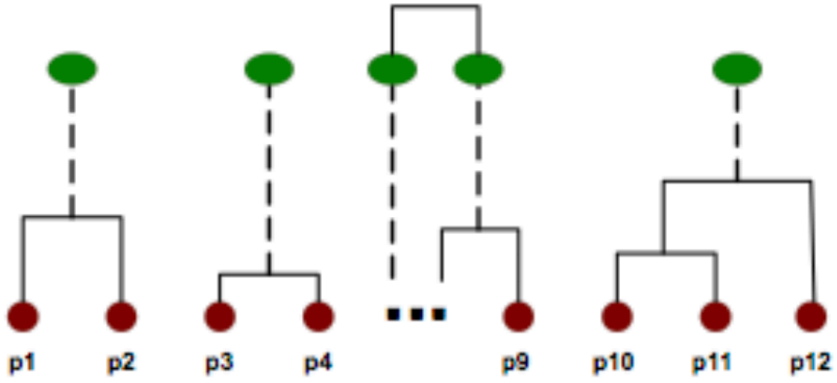
Agglomerative clustering

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



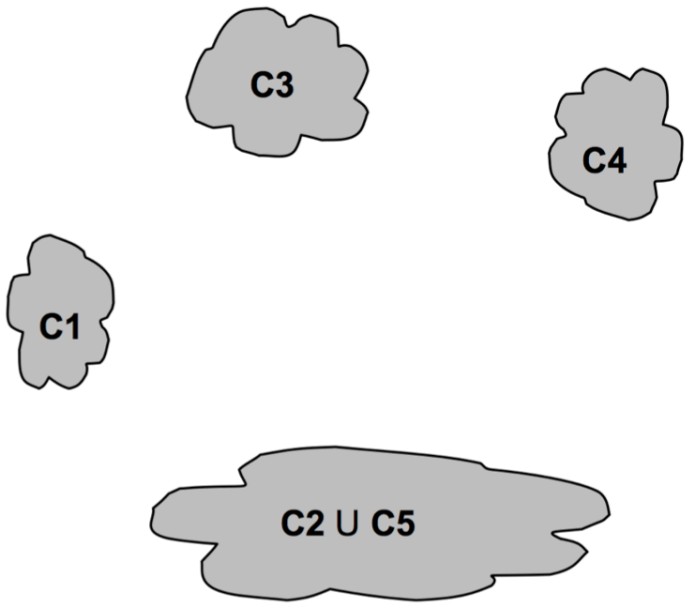


Proximity Matrix



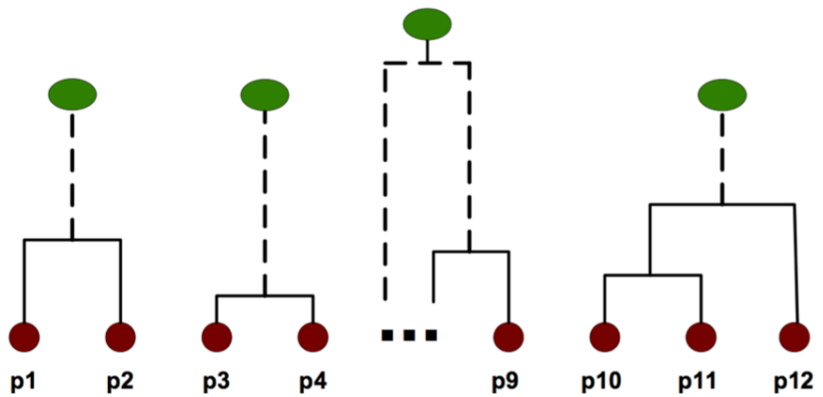
Agglomerative clustering

How do we do this?



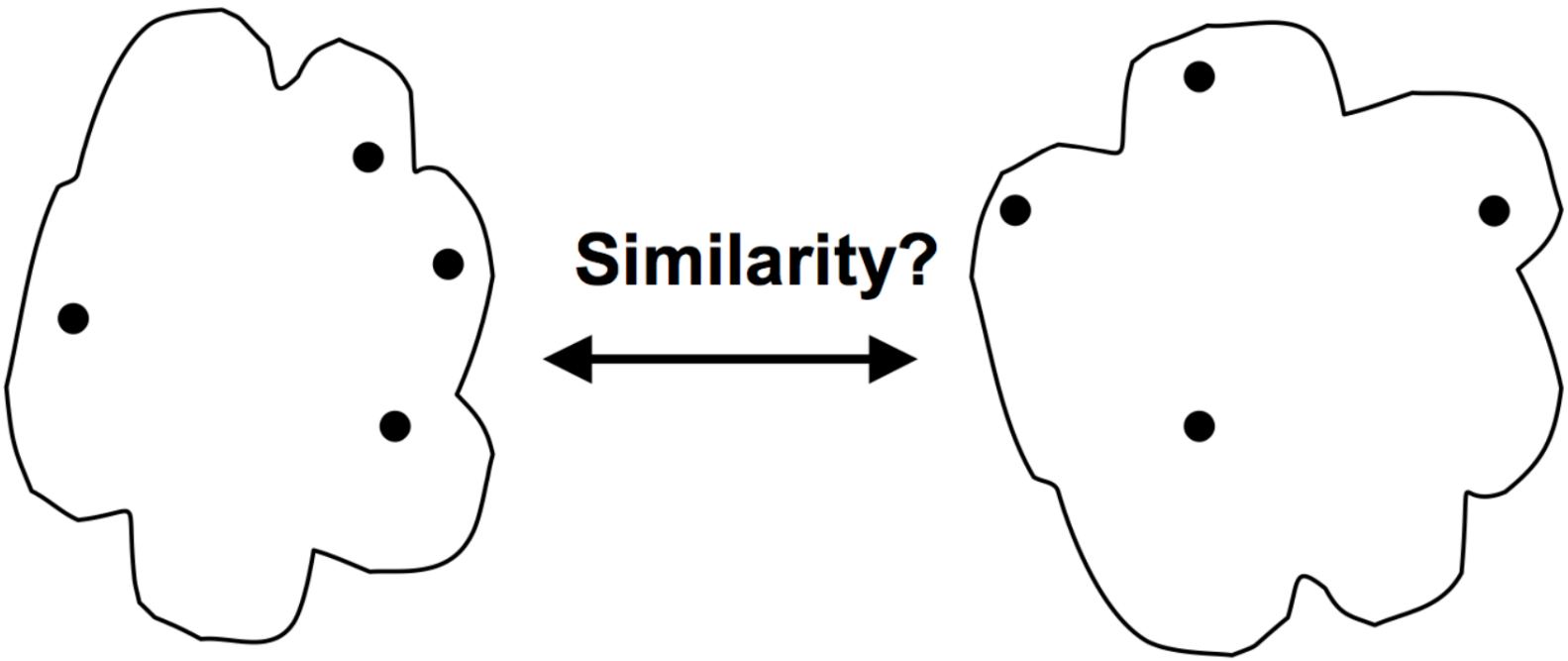
	C1	C2 U C5	C3	C4
C1		?		
C2 U C5	?	?	?	?
C3		?		
C4		?		

Proximity Matrix

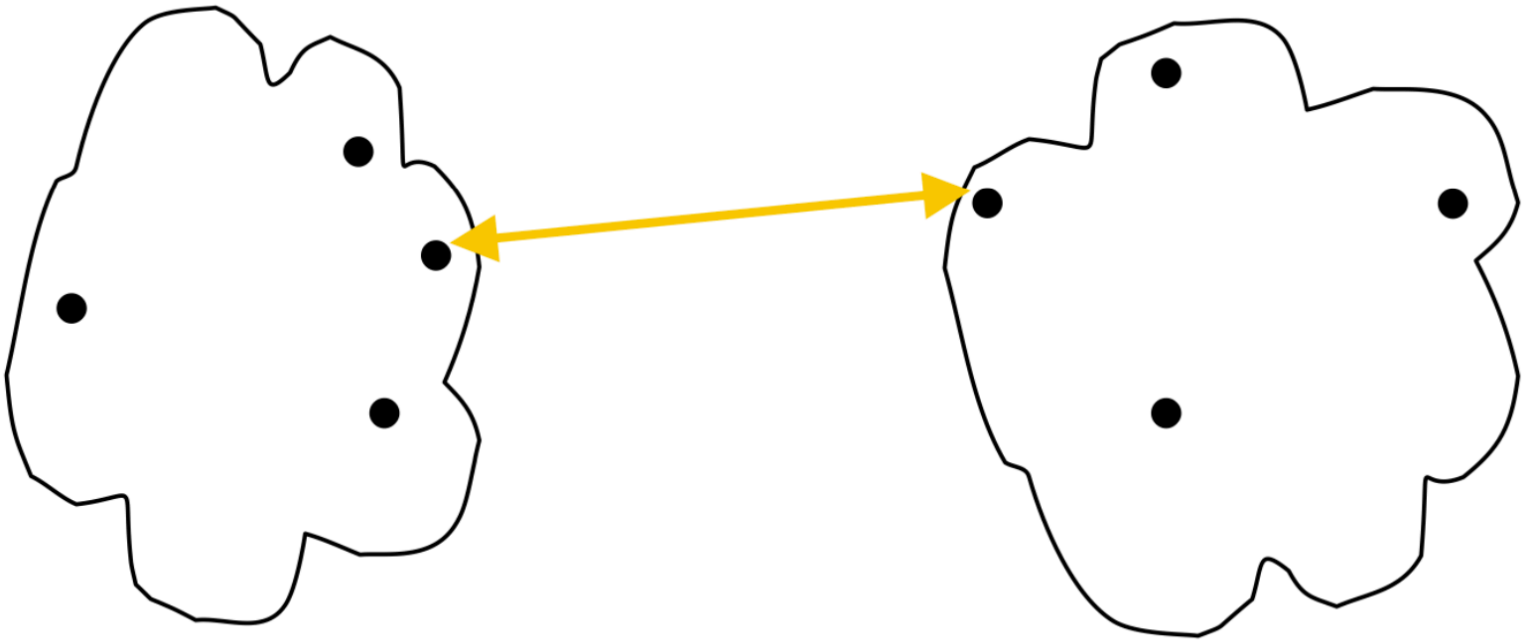


Agglomerative clustering

To frame the question very precisely, how do we measure the similarity/distance between two clusters of points?

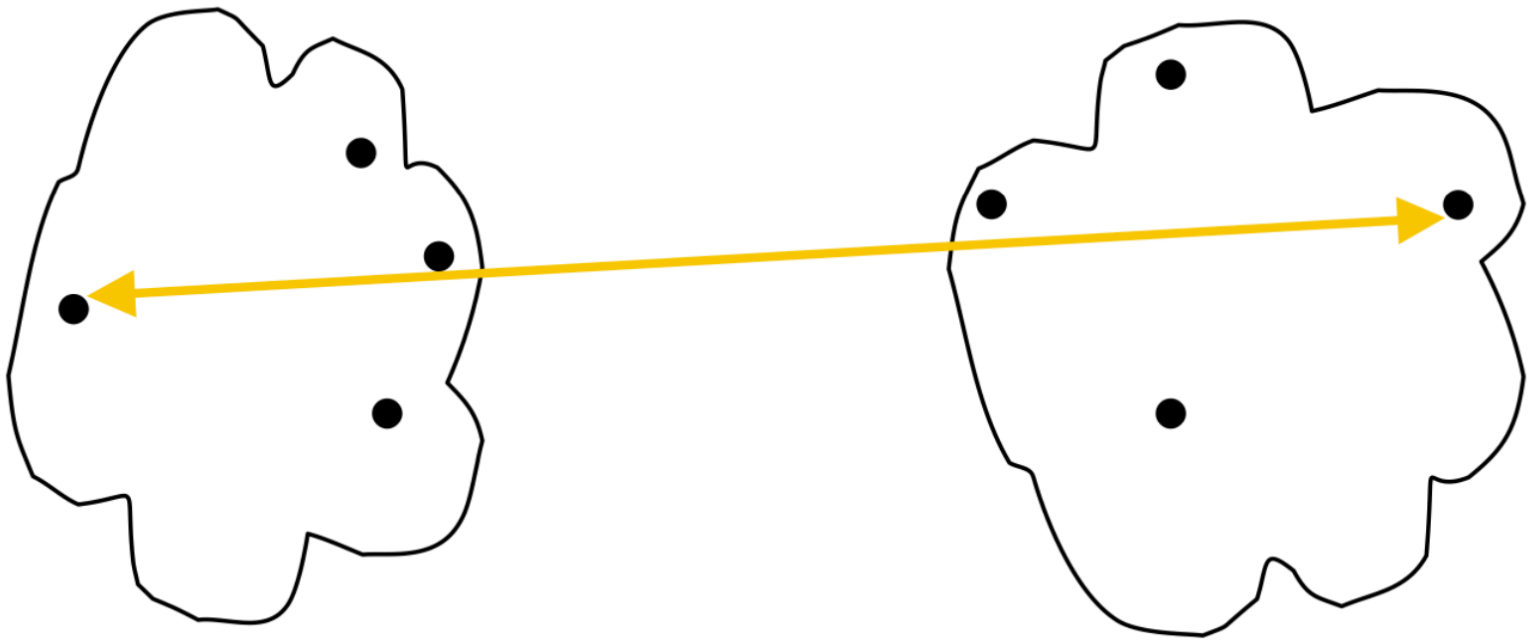


Min linkage ("single" in R)



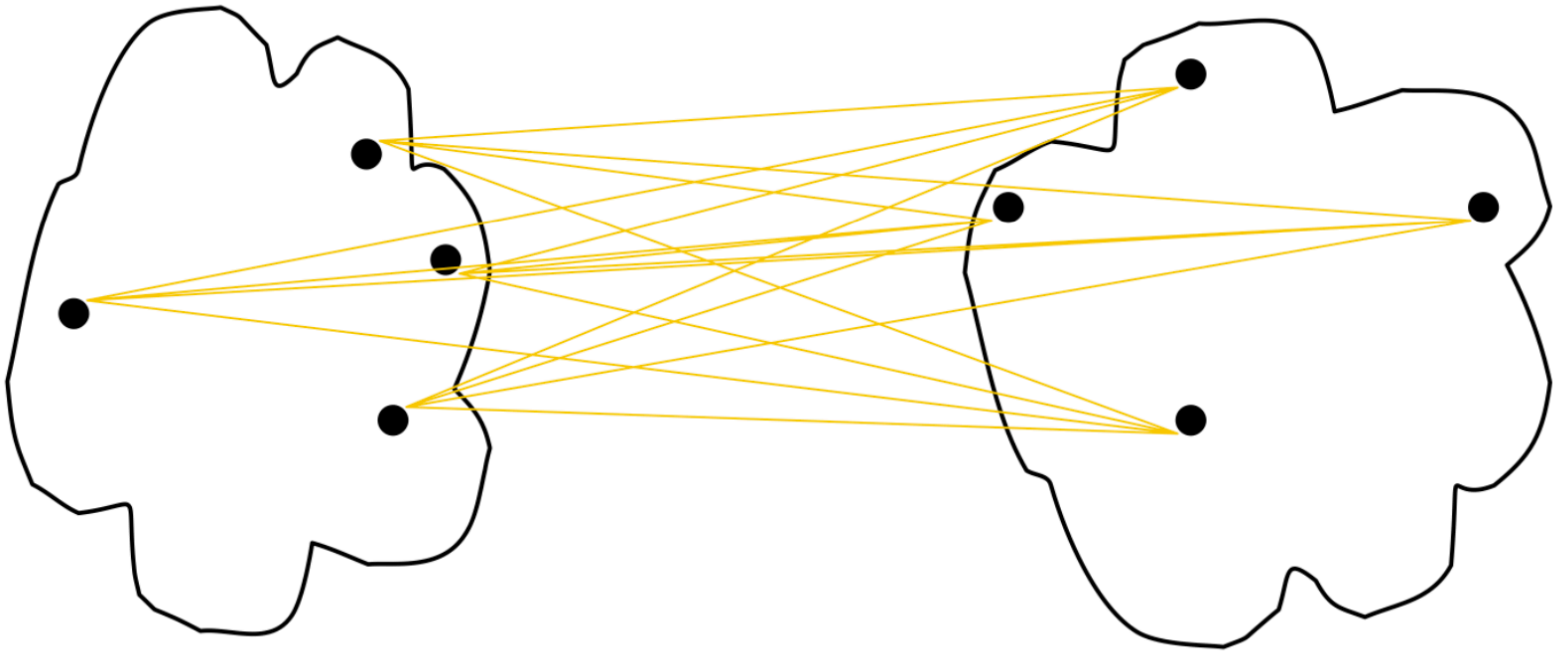
The minimum distance between two points, one in each cluster.

Max linkage ("complete" in R)



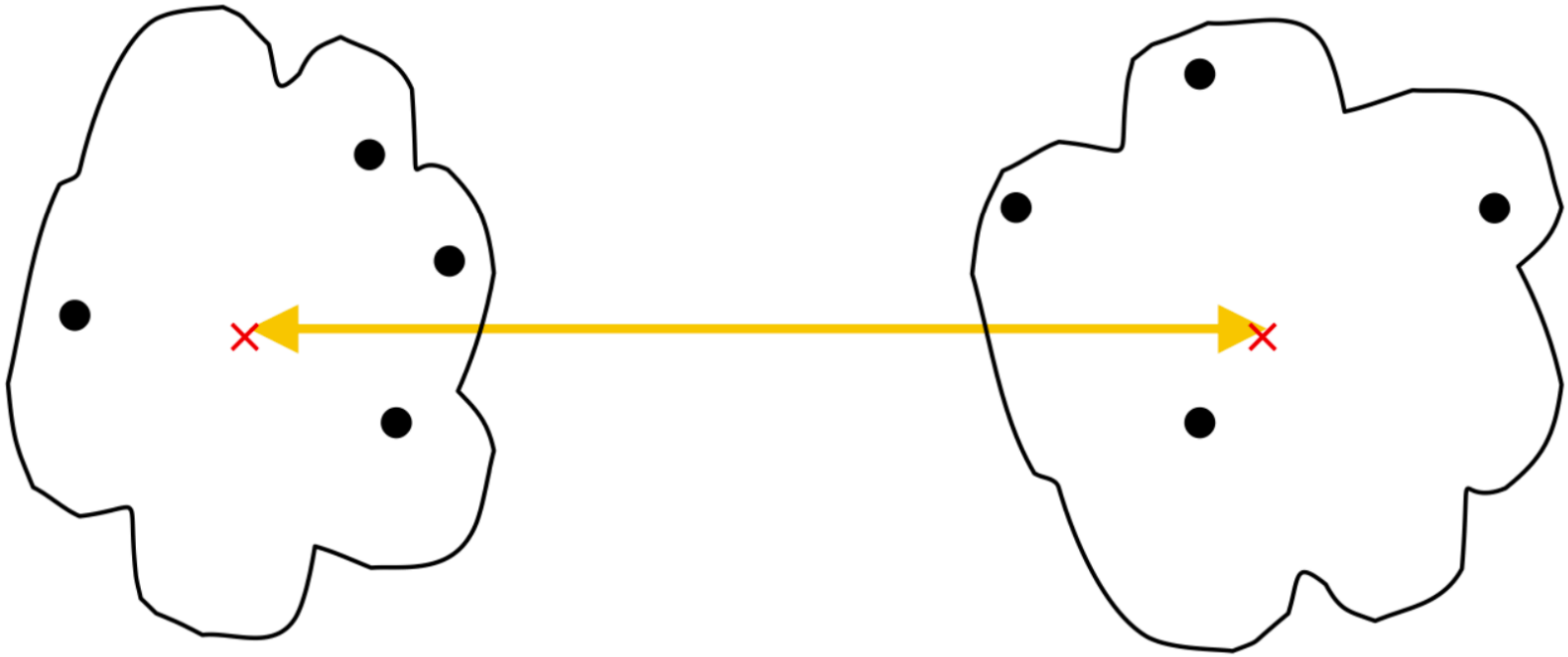
The maximum distance between two points, one in each cluster.

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`, where we'll see these ideas in action and getting the desired number of clusters by cutting the dendrogram.

Recap

So far, we've discussed two tools for unsupervised learning:

$\rightarrow$ K-means clustering – simply divide the data points into "flat clusters."

$\rightarrow$ Hierarchical clustering – develop a "hierarchy" or tree structure of your data. Choose clustering by cutting the tree.

Recap

So far, we've discussed two tools for unsupervised learning:

$\rightarrow$ K-means clustering – simply divide the data points into "flat clusters."

$\rightarrow$ Hierarchical clustering – develop a "hierarchy" or tree structure of your data. Choose clustering by cutting the tree.

Next, we'll move on to PCA, a powerful statistical tool for learning about your data.

Principal components analysis

Introduction to PCA

The goal of PCA is to find low-dimensional summaries of high-dimensional data sets.

This is useful for compression, for denoising, for plotting, and for making sense of data sets that initially seem too complicated to understand.

Introduction to PCA

The goal of PCA is to find low-dimensional summaries of high-dimensional data sets.

This is useful for compression, for denoising, for plotting, and for making sense of data sets that initially seem too complicated to understand.

It differs from clustering in that ...

- Clustering assumes that each data point is a member of one, and only one, cluster. (Clusters are mutually exclusive.)
- PCA assumes that each data point is like a combination of multiple basic "ingredients." (Ingredients are not mutually exclusive.)

Think about recipes

Nestle Toll House Chocolate-chip cookies:

→ 280 grams flour
→ 150 grams white sugar
→ 165 grams brown sugar
→ 225 grams butter
→ 2 eggs
→ 0 grams water

Think about recipes

Mary Berry's Victoria sponge cake:

$\rightarrow$ 225 grams flour
$\rightarrow$ 225 grams white sugar
$\rightarrow$ 0 grams brown sugar
$\rightarrow$ 225 grams butter
$\rightarrow$ 4 eggs
$\rightarrow$ 0 grams water

Think about recipes

seriouseats.com – old fashioned flaky pie dough:

→ 225 grams flour
→ 15 grams white sugar
→ 0 grams brown sugar
→ 225 grams butter
→ 0 eggs
→ 115 grams water

Think about recipes

Each baked good is constructed by following a recipe: a combination of the same basic ingredients.

→ Each data point x_i is like a baked good.
→ In PCA, the **principal components** are like the ingredients.

The amounts of each ingredient differ among baked goods:

→ For example, 225g sugar for sponge cake versus 15g sugar for pie dough.
→ In PCA, the **scores** are like the amounts of each ingredient in a given baked good.

Think about recipes

Our goal is to reverse-engineer both the **ingredients** and the **amounts/recipes** from an observed set of "baked goods" (i.e. original data points).

Hence, another unsupervised task!

The output of PCA

Before: Raw survey data on a bunch of TV shows

	Entertaining	Engaged	Original	Confusing	Funny
30 Rock	4.2	3.7	4.0	2.2	3.7
Next Top Model	4.2	3.8	3.8	2.0	3.5
American Chopper	4.2	3.6	3.9	2.1	3.5
Bones	4.3	4.1	3.8	1.9	3.4
Close to Home	4.1	3.8	3.8	1.9	2.9
Cold Case	4.2	3.9	4.0	1.9	3.0

+ 15 other columns

The output of PCA

After: Survey data on a bunch of TV shows run through PCA

	PC1	PC2
30 Rock	-2.64	0.12
Next Top Model	-0.78	-0.19
American Chopper	-2.31	0.68
Bones	3.24	2.22
Close to Home	-0.63	2.86
Cold Case	1.62	2.88

from 20 raw variables to 2 summaries. PC1 and PC2 are like the "ingredients." The numbers are the "amount" of each "ingredient."

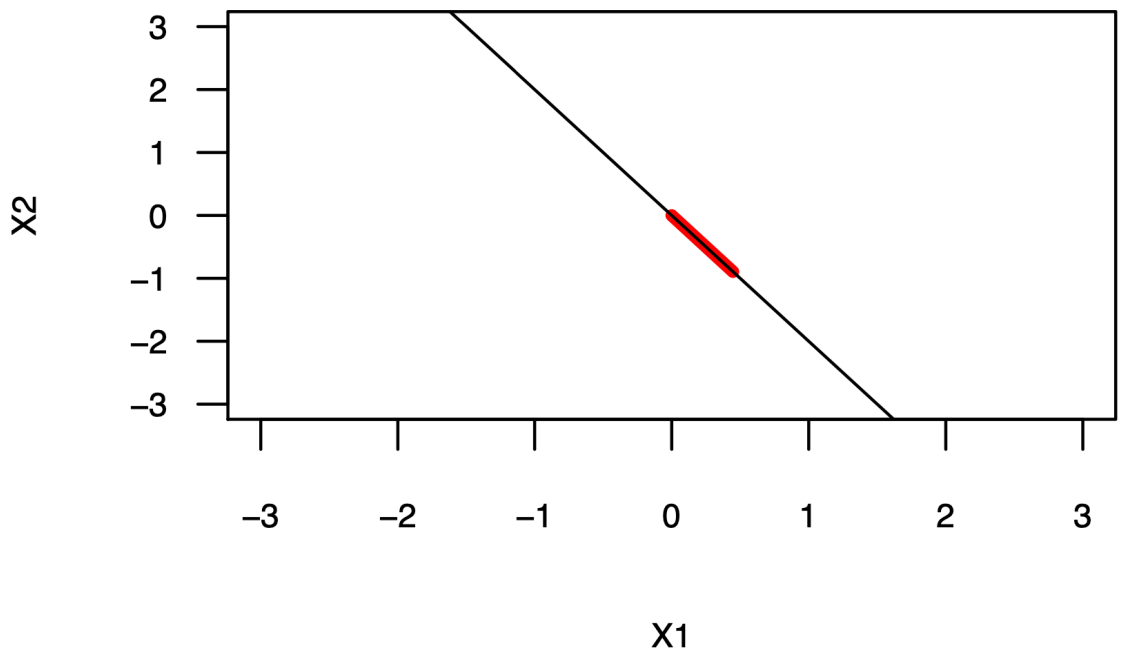
The trick to PCA is interpreting the summaries!

A brief detour into some linear algebra

Unfortunately, PCA is less delicious than baking, and it uses more linear algebra.

Say that $v \in \mathbb{R}^P$ is some vector. This defines a **subspace** of $\mathbb{R}^P$:

$$\mathcal{V} = \{z : z = \alpha_i v, \alpha_i \in \mathbb{R}\}$$



A brief detour into some linear algebra

Now let X be our usual $N \times P$ data matrix with rows x_i^T.

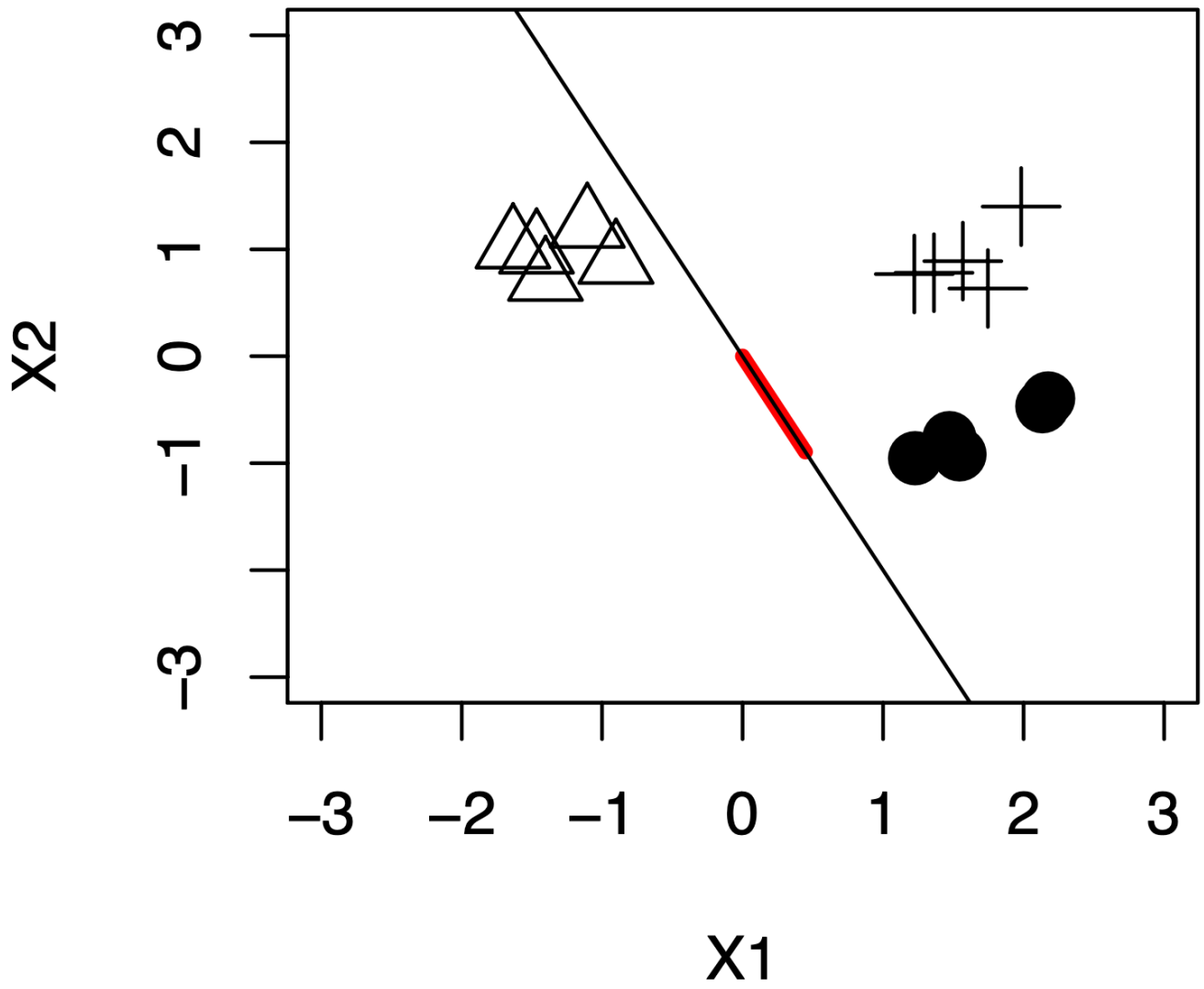
Suppose we **project** each x_i^T in our data matrix onto the subspace $\mathcal{V}$. The scalar location of this projected point in this subspace $\mathcal{V}$ is

$$\alpha_i = x_i \cdot v = x_i^T v$$

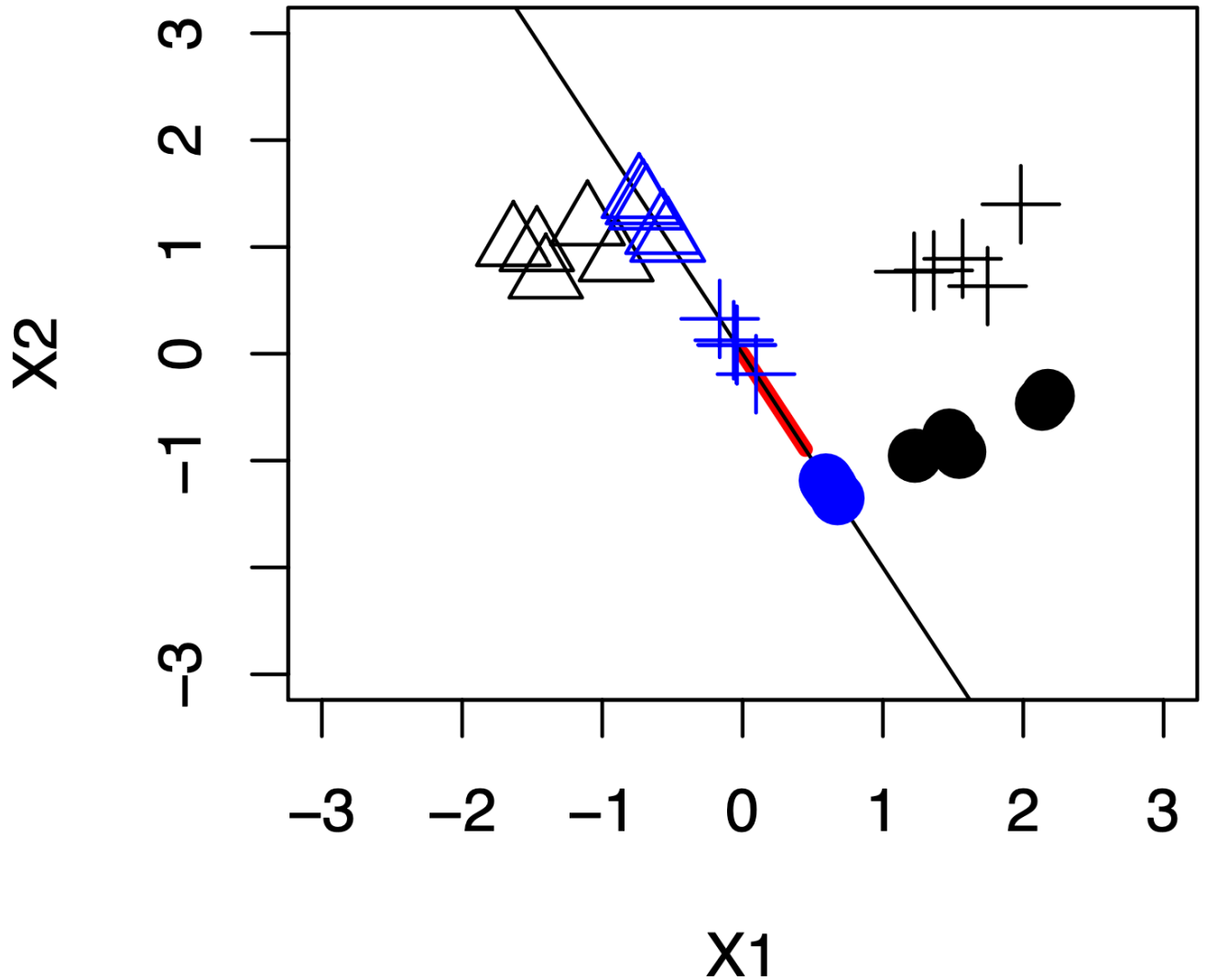
This is a one-number summary of our original point x_i. A different choice of v gives a different summary.

Let's see a picture.

The original points



The original points with the projected points



Key ideas

Key idea 1: projection = summary

- Each point's location along the subspace is a **one-number linear summary** of a P-dimensional data vector:

$$\alpha_i = x_i \cdot v = x_i^T v$$

- The goal of principal components analysis (PCA) is to find the "best" projection, i.e. the best linear summary of the data points.

Key ideas

Key idea 2: the "best summary" is the one that preserves as much of the **variance** in the original data points as possible.

- Intuition: we're already trying to crowd these P-dimensional points into 1-D. We should give them "room to breathe" by crowding them on top of each other as little as possible *within* that 1-D space!
- More variance in α_i means more "spread out" summaries in 1-D.
- And the more they're spread out, the more we have preserved differences between the projected points that were present in the original points. See `pca_intro.R`.

PCA as an optimization problem

Given data points $x_1, \ldots, x_N$, with each $x_i \in \mathbb{R}^P$, and a candidate vector v_1, the variance of the projected points is

$$\text{variance} = \frac{1}{n} \sum_{i=1}^{n} [\alpha_i - \bar{\alpha}]^2$$

where $\alpha_i x_i \cdot v_1$.
So we solve:

$$\underset{v_1 \in \mathbb{R},\, \|v\|_2 = 1}{\text{maximize}} \quad \sum_{i=1}^{n} \left[x_i \cdot v_1 - \left(\frac{1}{n} \sum_{i=1}^{N} x_i \cdot v_1 \right) \right]^2$$

Note: we constrain v_1 to have length 1; otherwise we could blow up the variance of the projected points as large as we wanted to.

PCA as an optimization problem

The solution v_1 to this optimization problem:

$\rightarrow$ is called the first principal component (synonyms: loading, rotation.)

$\rightarrow$ is the one-dimensional subspace capturing as much of the information in the original data matrix as possible.

The projected points $\alpha_i = x_i \cdot v$ are called the **scores** on the first **principal component**.

Higher order principal components

We define principal components 2 and up by the directions in the data space not explained by the previous components

- PC 2: run PCA on the "residual matrix" from PC 1.
- PC 3: run PCA on the "residual matrix" from PCs 1-2.
- ...
- PC P: run PCA on the "residual matrix" from PCs 1-(P-1).

Thus principal component M is defined recursively in terms of the fit from principal components 1 through $M - 1$. In practice, we often stop with far fewer than P principal components.