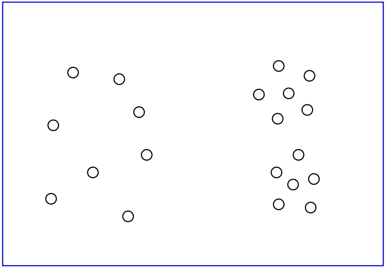
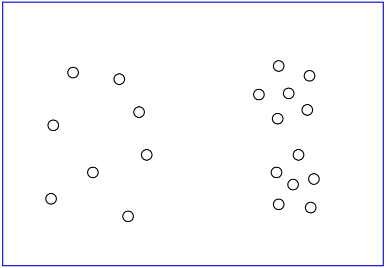
Clustering

ECE 407

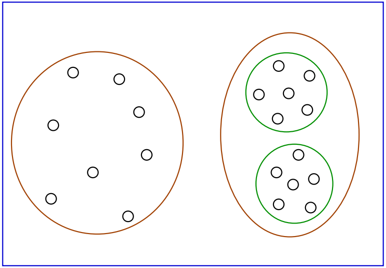
Clustering in $\mathbb{R}^p$



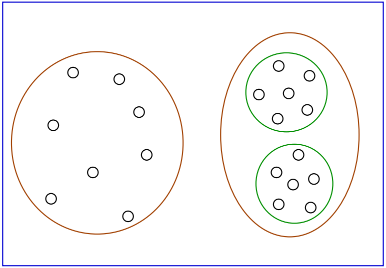
Clustering in $\mathbb{R}^p$



Clustering in $\mathbb{R}^p$

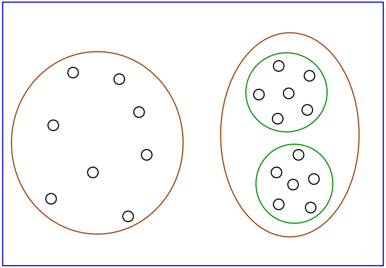


Clustering in $\mathbb{R}^p$



Two common uses of clustering:

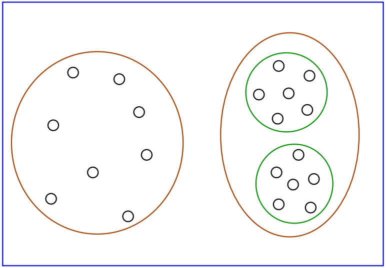
Clustering in $\mathbb{R}^p$



Two common uses of clustering:

- Vector quantization
  Find a finite set of representatives that provides good coverage of a complex, possibly infinite, high-dimensional space.

Clustering in $\mathbb{R}^p$



Three common uses of clustering:

- Vector quantization
  Find a finite set of representatives that provides good coverage of a complex, possibly infinite, high-dimensional space.
- Finding meaningful structure in data (unsupervised learning)
  Finding salient grouping in data.
- ML Estimation using Gaussian Mixture Models

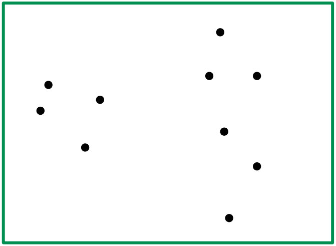
Widely-used clustering methods

1. K-means and its many variants
2. EM for mixtures of Gaussians
3. Agglomerative hierarchical clustering

The k-means optimization problem

- Input: Points $x_1, \ldots, x_n \in \mathbb{R}^p$; integer k
- Output: "Centers", or representatives, $\mu_1, \ldots, \mu_k \in \mathbb{R}^p$
- Goal: Minimize average squared distance between points and their nearest representatives:

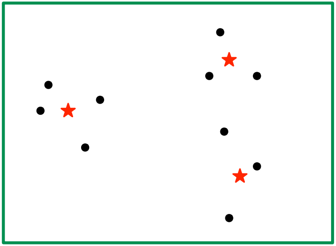
$$\mathrm{cost}(\mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \min_j \|x_i - \mu_j\|^2$$



The k-means optimization problem

- Input: Points $x_1, \ldots, x_n \in \mathbb{R}^p$; integer k
- Output: "Centers", or representatives, $\mu_1, \ldots, \mu_k \in \mathbb{R}^p$
- Goal: Minimize average squared distance between points and their nearest representatives:

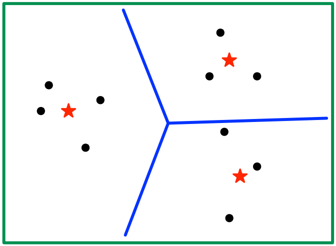
$$\text{cost}(\mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \min_j \|x_i - \mu_j\|^2$$



The k-means optimization problem

- Input: Points $x_1, \ldots, x_n \in \mathbb{R}^p$; integer k
- Output: "Centers", or representatives, $\mu_1, \ldots, \mu_k \in \mathbb{R}^p$
- Goal: Minimize average squared distance between points and their nearest representatives:

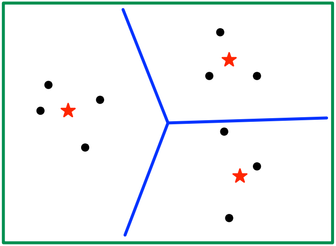
$$\mathrm{cost}(\mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \min_j \|x_i - \mu_j\|^2$$



The k-means optimization problem

- Input: Points $x_1, \ldots, x_n \in \mathbb{R}^p$; integer k
- Output: "Centers", or representatives, $\mu_1, \ldots, \mu_k \in \mathbb{R}^p$
- Goal: Minimize average squared distance between points and their nearest representatives:

$$\mathrm{cost}(\mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \min_j \|x_i - \mu_j\|^2$$

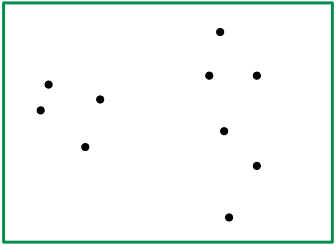


The centers carve $\mathbb{R}^p$ up into k convex regions: μ_j's region consists of points for which it is the closest center.

Lloyd's k-means algorithm

The k-means problem is NP-hard to solve. The most popular heuristic is called the "k-means algorithm".

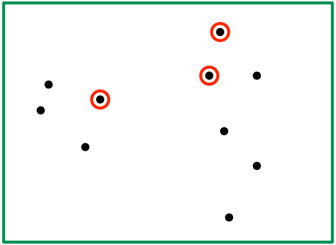
- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
  - Assign each point to its closest center.
  - Update each μ_j to the mean of the points assigned to it.



Lloyd's k-means algorithm

The k-means problem is NP-hard to solve. The most popular heuristic is called the "k-means algorithm" (or LBG algorithm).

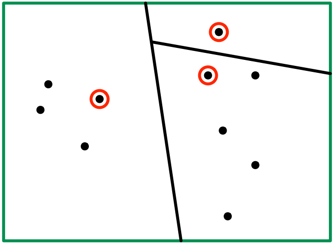
- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
  - Assign each point to its closest center.
  - Update each μ_j to the mean of the points assigned to it.



Lloyd's k-means algorithm

The k-means problem is NP-hard to solve. The most popular heuristic is called the "k-means algorithm".

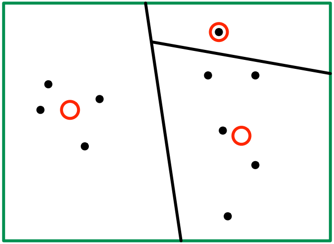
- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
  - Assign each point to its closest center.
  - Update each μ_j to the mean of the points assigned to it.



Lloyd's k-means algorithm

The k-means problem is NP-hard to solve. The most popular heuristic is called the "k-means algorithm".

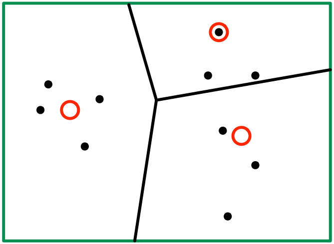
- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
  - Assign each point to its closest center.
  - Update each μ_j to the mean of the points assigned to it.



Lloyd's k-means algorithm

The k-means problem is NP-hard to solve. The most popular heuristic is called the "k-means algorithm".

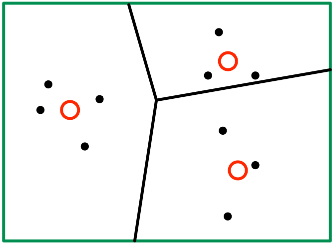
- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
  - Assign each point to its closest center.
  - Update each μ_j to the mean of the points assigned to it.



Lloyd's k-means algorithm

The k-means problem is NP-hard to solve. The most popular heuristic is called the "k-means algorithm".

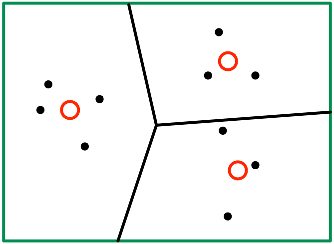
- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
  - Assign each point to its closest center.
  - Update each μ_j to the mean of the points assigned to it.



Lloyd's k-means algorithm

The k-means problem is NP-hard to solve. The most popular heuristic is called the "k-means algorithm".

- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
    - Assign each point to its closest center.
    - Update each μ_j to the mean of the points assigned to it.



Each iteration reduces the cost $\Rightarrow$ convergence to a local optimum
We cannot prove the global optimality.

Initializing the k-means algorithm

Typical practice: choose k data points at random as the initial centers.

Initializing the k-means algorithm

Typical practice: choose k data points at random as the initial centers.

Another common trick: start with extra centers, then prune later.

Initializing the k-means algorithm

Typical practice: choose k data points at random as the initial centers.

Another common trick: start with extra centers, then prune later.

A particularly good initializer: k-**means++**

- Pick a data point x at random as the first center
- Let $C = \{x\}$ (centers chosen so far)
- Repeat until desired number of centers is attained:
  - Pick a data point x at random from the following distribution:

    $$\Pr(x) \propto \mathrm{dist}(x, C)^2,$$

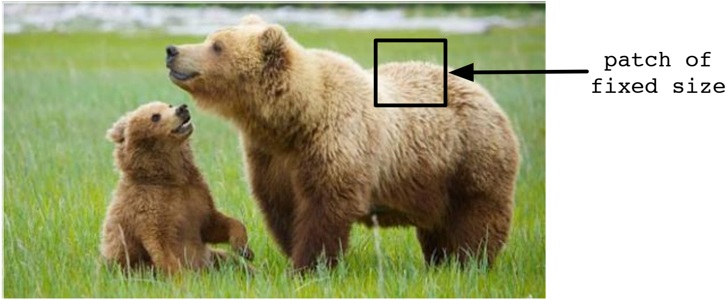
    where $\mathrm{dist}(x, C) = \min_{z \in C} \|x - z\|$
  - Add x to C

Representing images using k-means codewords

Given a collection of images, how to represent as fixed-length vectors?

Representing images using k-means codewords

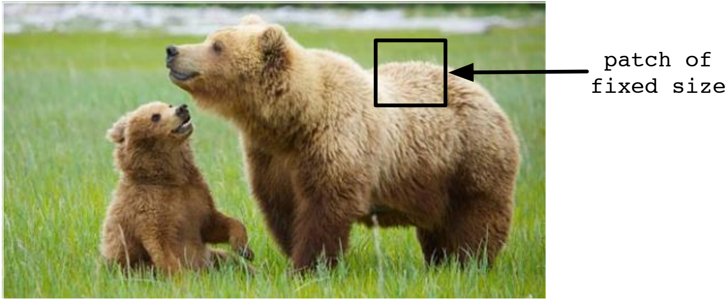
Given a collection of images, how to represent as fixed-length vectors?



- Look at all $\ell \times \ell$ patches in all images.
- Run k-means on this entire collection to get k centers.
- Now associate any image patch with its nearest center.
- Represent an image by a histogram over $\{1, 2, \ldots, k\}$.

Representing images using k-means codewords

Given a collection of images, how to represent as fixed-length vectors?



- Look at all $\ell \times \ell$ patches in all images.
- Run k-means on this entire collection to get k centers.
- Now associate any image patch with its nearest center.
- Represent an image by a histogram over $\{1, 2, \ldots, k\}$.

This is called Vector Quantization (VQ) based Feature Extraction.

Streaming and online computation

Streaming computation: for data sets that are too large to fit in memory.

- Make one pass (or maybe a few passes) through the data.
- On each pass:
  - See data points one at a time, in order.
  - Update models/parameters along the way.
- There is only enough space to store a tiny fraction of the data, or a perhaps short summary.

Streaming and online computation

Streaming computation: for data sets that are too large to fit in memory.

- Make one pass (or maybe a few passes) through the data.
- On each pass:
  - See data points one at a time, in order.
  - Update models/parameters along the way.
- There is only enough space to store a tiny fraction of the data, or a perhaps short summary.

Online computation: an even more lightweight setup, for data that is continuously being collected.

- Initialize a model.
- Repeat forever:
  - See a new data point.
  - Update model if need be.

Example: sequential k-means

1. Set the centers $\mu_1, \ldots, \mu_k$ to the first k data points
2. Set their counts to $n_1 = n_2 = \cdots = n_k = 1$
3. Repeat, possibly foreover:
   - Get next data point x
   - Let μ_j be the center closest to x
   - Update μ_j and n_j:

$$\mu_j = \frac{n_j \mu_j + x}{n_j + 1} \quad \text{and} \quad n_j = n_j + 1$$

K-means: the good and the bad

The good:

- Fast and easy.
- Effective in quantization.

The bad:

- Geared towards data in which the clusters are spherical, and of roughly the same radius.

K-means: the good and the bad

The good:

- Fast and easy.
- Effective in quantization.

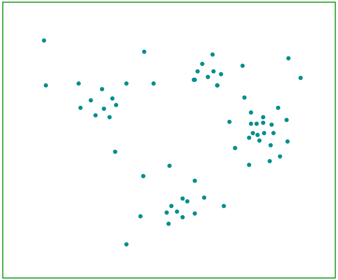
The bad:

- Geared towards data in which the clusters are spherical, and of roughly the same radius.

Is there is a similarly-simple algorithm in which clusters of more general shape are accommodated?

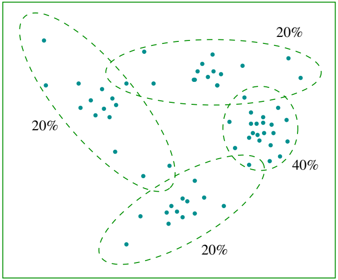
Mixtures of Gaussians

Idea: model each cluster by a Gaussian:



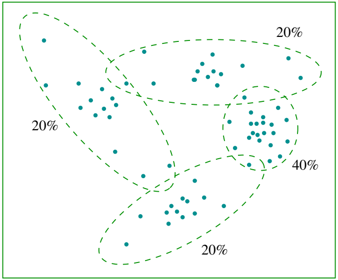
Mixtures of Gaussians

Idea: model each cluster by a Gaussian:



Mixtures of Gaussians

Idea: model each cluster by a Gaussian:

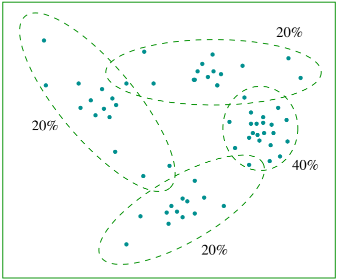


Each of the k clusters is specified by:

- a Gaussian distribution $P_j = N(\mu_j, \Sigma_j)$
- a mixing weight π_j

Mixtures of Gaussians

Idea: model each cluster by a Gaussian:



Each of the k clusters is specified by:

- a Gaussian distribution $P_j = N(\mu_j, \Sigma_j)$
- a mixing weight π_j

Overall distribution over $\mathbb{R}^p$: a **mixture of Gaussians**

$$\Pr(x) = \pi_1 P_1(x) + \cdots + \pi_k P_k(x)$$

The clustering task

Given data $x_1, \ldots, x_n \in \mathbb{R}^P$, find the maximum-likelihood mixture of Gaussians: that is, find parameters

- $\pi_1, \ldots, \pi_k \geq 0$ summing to one
- $\mu_1, \ldots, \mu_k \in \mathbb{R}^p$
- $\Sigma_1, \ldots, \Sigma_k \in \mathbb{R}^{p \times p}$

to maximize

$$
\begin{aligned}
&\Pr\left(\text{data} \mid \pi_1 P_1 + \cdots + \pi_k P_k\right) \\
&= \prod_{i=1}^{n} \left(\sum_{j=1}^{k} \pi_j P_j(x_i) \right) \\
&= \prod_{i=1}^{n} \left(\sum_{j=1}^{k} \frac{\pi_j}{(2\pi)^{p/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2} (x_i - \mu_j)^T \Sigma_j^{-1} (x_i - \mu_j) \right) \right)
\end{aligned}
$$

where P_j is the distribution of the jth cluster, $N(\mu_j, \Sigma_j)$.

The EM algorithm

1. Initialize $\pi_1, \ldots, \pi_k$ and $P_1 = N(\mu_1, \Sigma_1), \ldots, P_k = N(\mu_k, \Sigma_k)$ in some manner.
2. Repeat until convergence:
   - Assign each point x_i fractionally between the k clusters:

$$w_{ij} = \Pr(\text{cluster } j \mid x_i) = \frac{\pi_j P_j(x_i)}{\sum_\ell \pi_\ell P_\ell(x_i)} \qquad \text{j=1,2,..,k}$$

   - Now update the mixing weights, means, and covariances:

$$\pi_j = \frac{1}{n} \sum_{i=1}^{n} w_{ij}$$

(weighted mean)

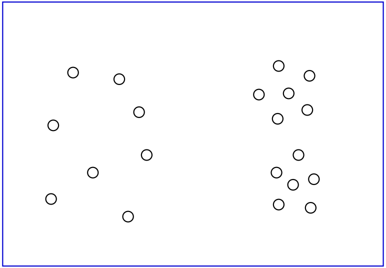
$$\mu_j = \frac{1}{n\pi_j} \sum_{i=1}^{n} w_{ij} x_i$$

(weighted covariance)

$$\Sigma_j = \frac{1}{n\pi_j} \sum_{i=1}^{n} w_{ij} (x_i - \mu_j)(x_i - \mu_j)^T$$

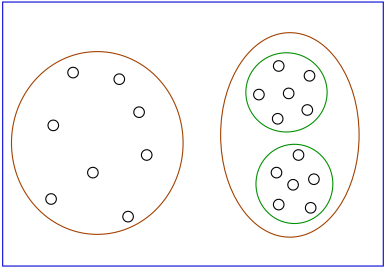
Hierarchical clustering

Choosing the number of clusters (k) is difficult.



Hierarchical clustering

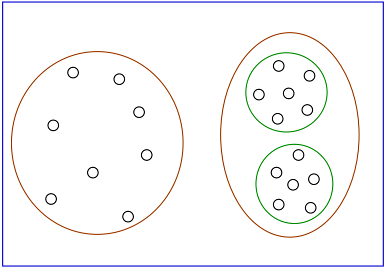
Choosing the number of clusters (k) is difficult.



Often there is no single right answer, because of multiscale structure.

Hierarchical clustering

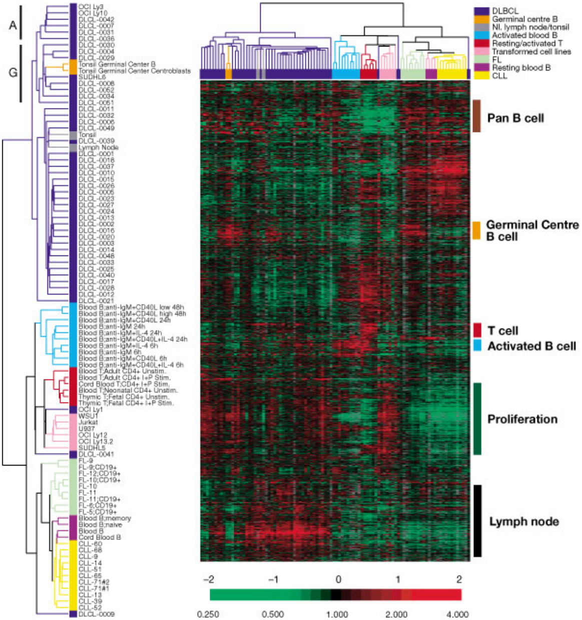
Choosing the number of clusters (k) is difficult.



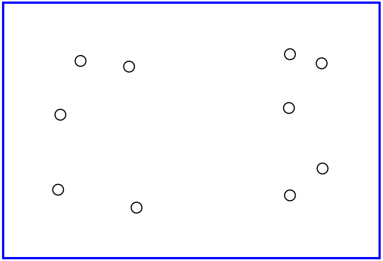
Often there is no single right answer, because of multiscale structure.

Hierarchical clustering avoids these problems.

Example: gene expression data

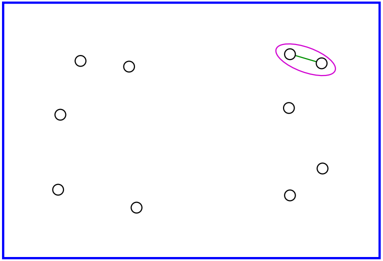


The single linkage algorithm



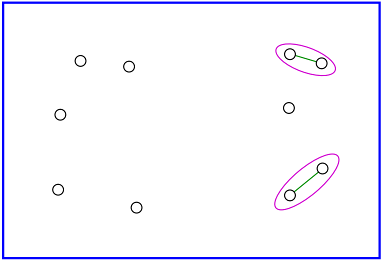
- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

The single linkage algorithm



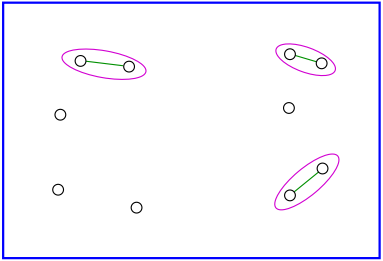
- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

The single linkage algorithm



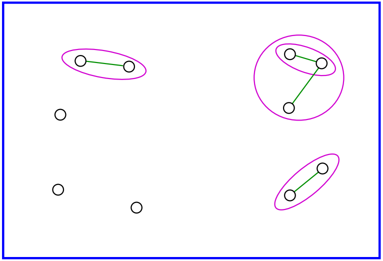
- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

The single linkage algorithm



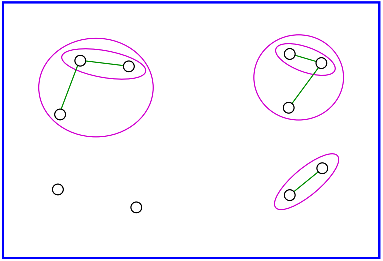
- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

The single linkage algorithm



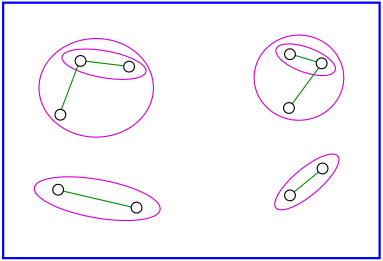
- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
    - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

The single linkage algorithm



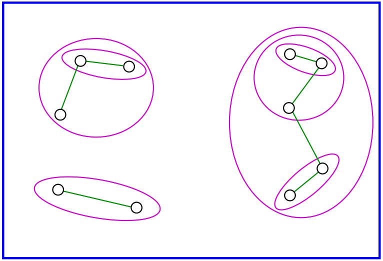
- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

The single linkage algorithm



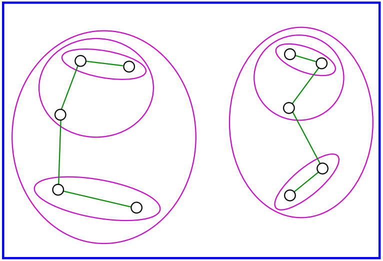
- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

The single linkage algorithm



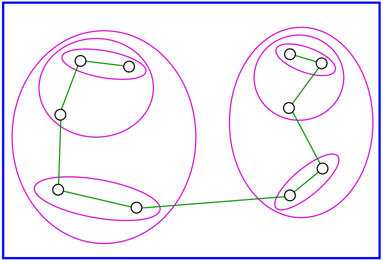
- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

The single linkage algorithm



- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

The single linkage algorithm



- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

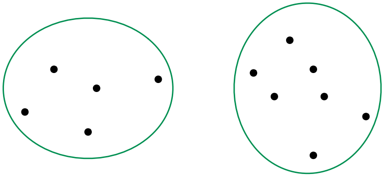
Linkage methods

- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
    - Merge the two "closest" clusters

Linkage methods

- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two "closest" clusters

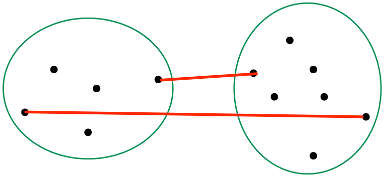
How to measure the distance between two clusters of points, C and C'?



Linkage methods

- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two "closest" clusters

How to measure the distance between two clusters of points, C and C'?



- Single linkage

$$\mathrm{dist}(C, C') = \min_{x \in C, x' \in C'} \|x - x'\|$$

- Complete linkage

$$\mathrm{dist}(C, C') = \max_{x \in C, x' \in C'} \|x - x'\|$$

Average linkage

Three commonly-used variants:

1. Average pairwise distance between points in the two clusters

$$\mathrm{dist}(C, C') = \frac{1}{|C| \cdot |C'|} \sum_{x \in C} \sum_{x' \in C'} \|x - x'\|$$

2. Distance between cluster centers

$$\mathrm{dist}(C, C') = \|\mathrm{mean}(C) - \mathrm{mean}(C')\|$$

3. Ward's method: the increase in k-means cost occasioned by merging the two clusters

$$\mathrm{dist}(C, C') = \frac{|C| \cdot |C'|}{|C| + |C'|} \|\mathrm{mean}(C) - \mathrm{mean}(C')\|^2$$