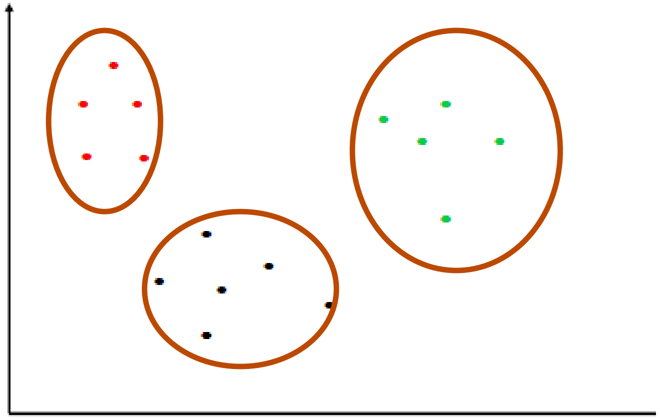
Lecture #12: Unsupervised Learning

Supervised vs. Unsupervised Learning

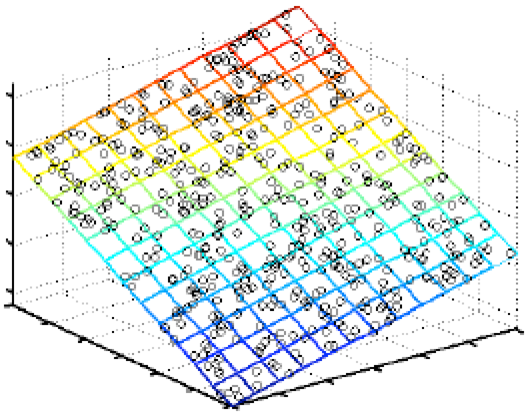
- **Supervised Learning**
  - Training examples are labeled with the class labels
- **Unsupervised Learning**
  - We are only provided with examples without specifying the class labels

What can we learn from unlabeled data?

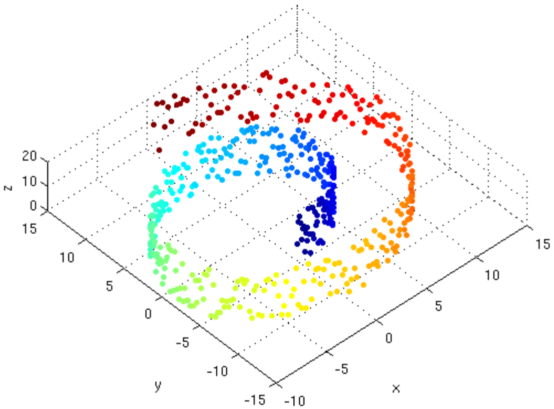
- **Group of clusters in the data**



- **Low dimensional structure**

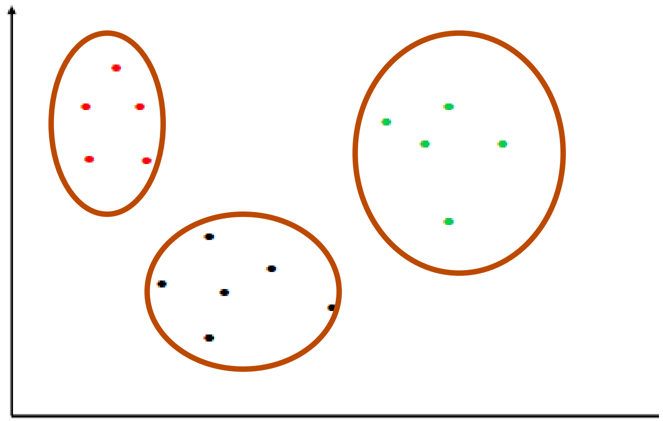


PCA



Nonlinear embedding

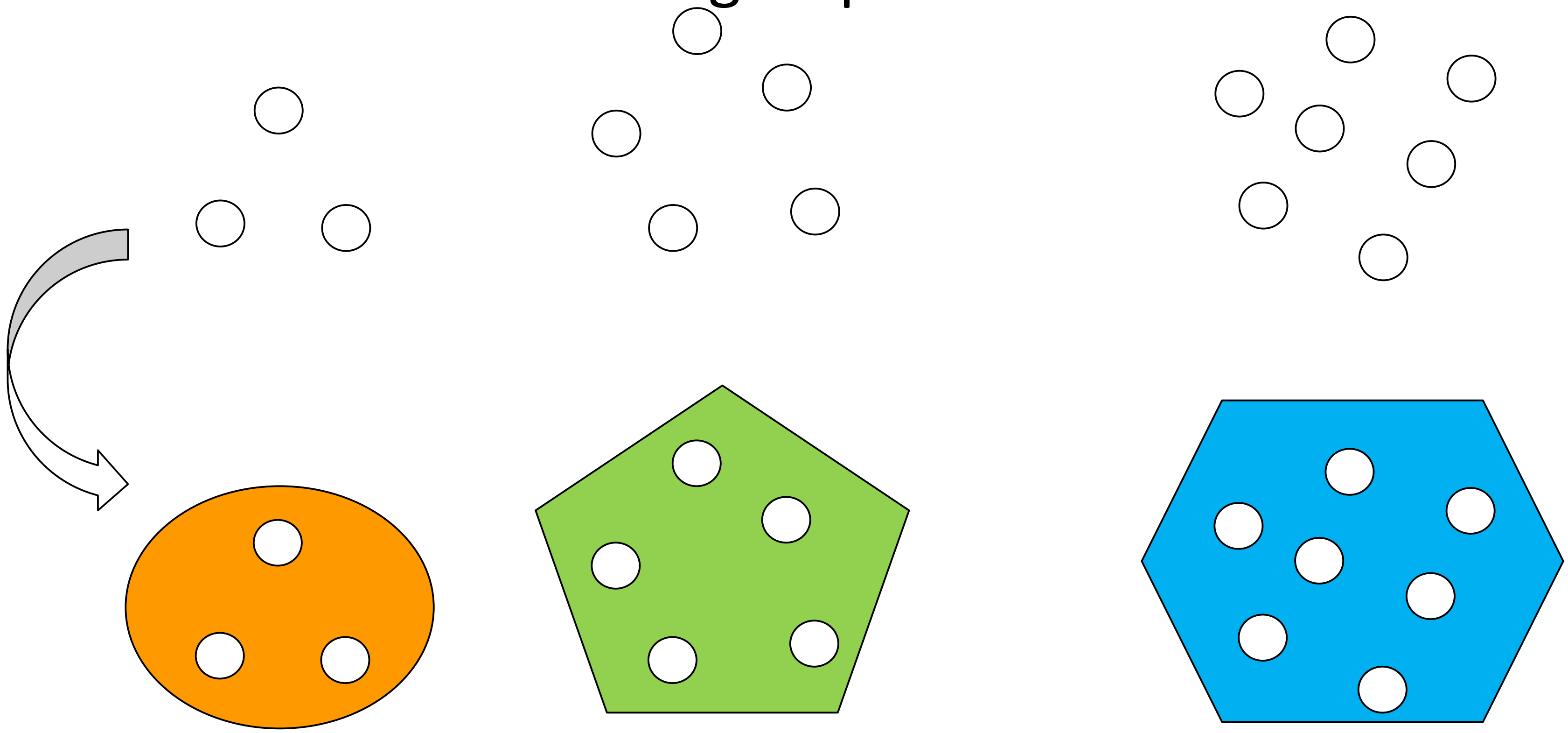
Clustering



- Are there any groups in the data?
- How to group?
- How many groups?

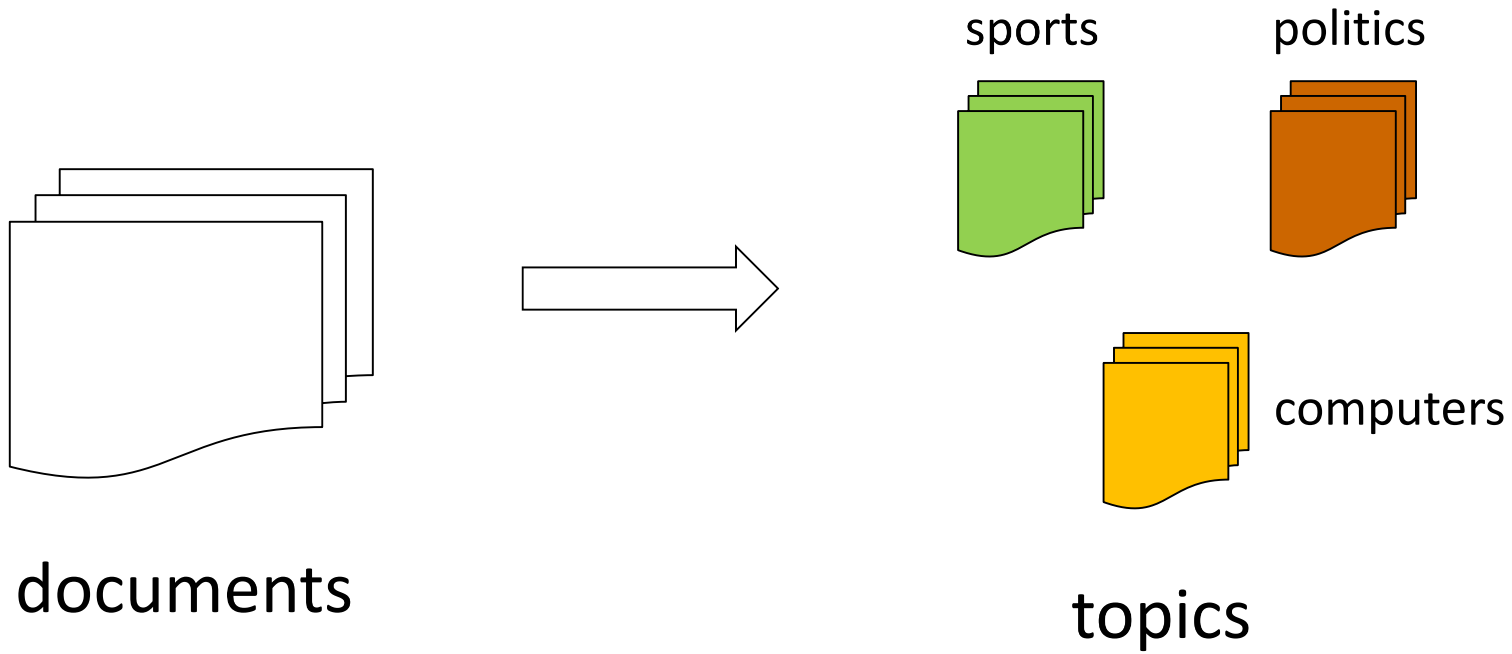
Unsupervised Learning

- **Clustering:** most common form of unsupervised learning
  - Given a collection of unlabeled examples (objects), discover self-similar groups in the data



Unsupervised Learning

- **Text Clustering**



Unsupervised Learning

- **Image Segmentation**

Clustering Applications

- Find genes that are similar in their functions
- Group documents based on topics
- Categorize customers based on their buying habit
- Group images based on their contents

Clustering Issues

- What is a natural grouping among these objects?
  - Definition of "group“
- What makes objects “related”?
  - Definition of "similarity/distance“
- Representation for objects
  - Vector, normalization?
- How many clusters?
  - Fixed a priori?
  - Completely data driven?
  - Avoid “trivial” clusters - too large or small

What is a natural grouping?

- By color? By pattern? By weight?
- The definition of natural grouping is subjective
- This is why we call clustering ***<u>exploratory</u>*** data analysis

What is similarity?

- This is a philosophical question. We will take a more pragmatic approach.
  - Depends on representation and algorithm. For many representations/algorithms, it is easier to think in terms of a distance (rather than similarity) between vectors

Hard to define but

We know it when we see it

Properties of a distance measure?

- **D must be Symmetric**
  - $D(A, B) = D(B, A)$
  - Otherwise, we can say A looks like B but B does not look like A

- **Positivity, and self-similarity**
  - $D(A, B) \geq 0$, and $D(A, B) = 0$ iff $A = B$
  - Otherwise, there will different objects that we cannot tell apart

- **Must satisfy triangle inequality**
  - $D(A, B) + D(B, C) \geq D(A, C)$
  - Otherwise, one can say "A is like B, B is like C, but A is not like C at all"

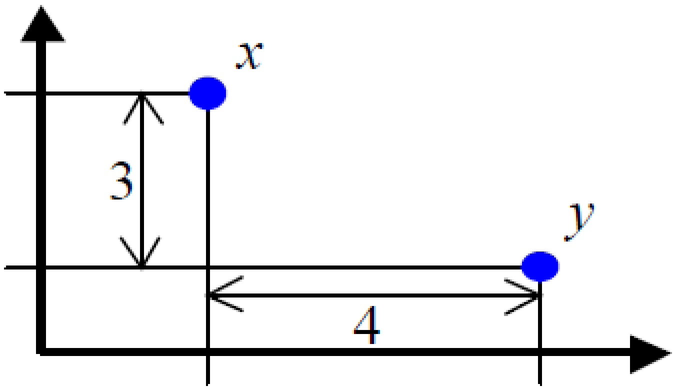
Distance Measures: Minkowski Metric

- Suppose two object x and y both have d features
  - $x = (x_1, \cdots, x_d), y = (y_1, \cdots, y_d)$
- The Minkowski metric of order r is defined by

$$d(x, y) = \sqrt[r]{\sum_i |x_i - y_i|^r}$$

- Common Minkowski metrics:
  - Euclidean(r=2): $d(x, y) = \sqrt[2]{\sum_i (x_i - y_i)^2}$, also called L_2 distance
  - Manhattan distance(r=1) : $d(x, y) = \sum_i |x_i - y_i|$, also called L_1 distance
  - "Sup" distance(r = +∞): $d(x, y) = \max_i |x_i - y_i|$, also called L_∞ distance

A simple example



$1:$ Euclidean distance: $\sqrt[2]{4^2+3^2}=5.$

$2:$ Manhattan distance: $4+3=7.$

$3:$ "sup" distance: $\max\{4,3\}=4.$

Similarities

- Cosine similarity – commonly used to measure document similarity

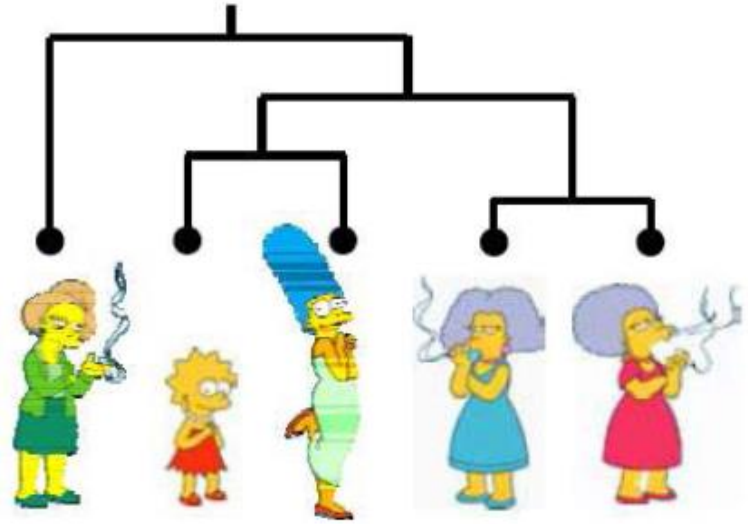
$$cos(\mathrm{x}, \mathrm{x}') = \frac{<\mathrm{x} \cdot \mathrm{x}'>}{|\mathrm{x}| \cdot |\mathrm{x}'|}$$

- Kernels: RBF (Gaussian) Kernel

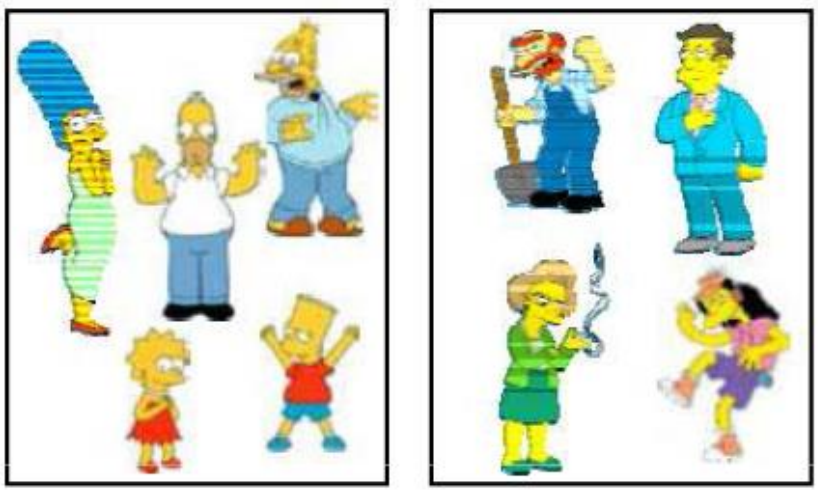
$$K(X, X') = \exp \frac{-|X - X'|^2}{2\sigma^2}$$

Clustering Algorithms

- **Hierarchical algorithms**
  - Bottom up (agglomerative)
  - Top down (divisive)

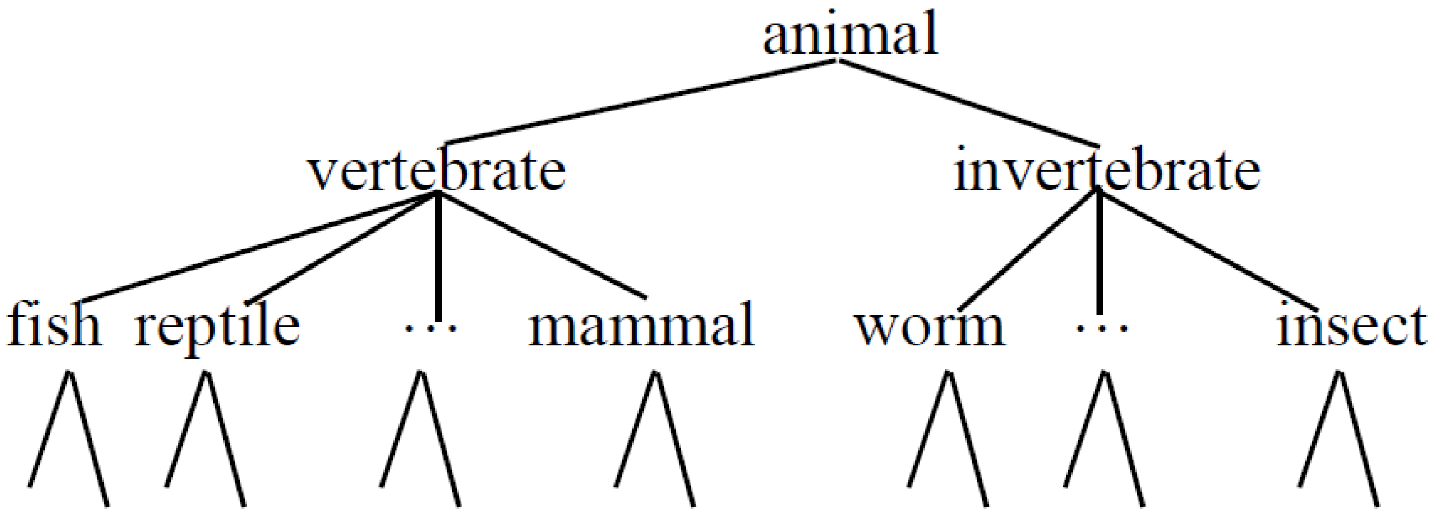


- **Partition algorithms (Flat)**
  - K-means
  - Mixture of Gaussians
  - Spectral clustering



Hierarchical Clustering

- Given a set of objects, build a tree-based taxonomy



- Hierarchies are a convenient way for organizing information, used frequently by web-portals

Hierarchical Agglomerative Clustering (HAC)

- Start with each object in a separate cluster
- Repeatedly join the closest pair of clusters
- until there is only one cluster

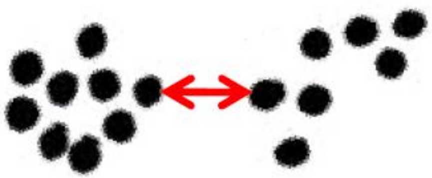
- The history of merging forms a tree of hierarchy

- ***Question***: how to measure the "**closeness**" of two clusters?

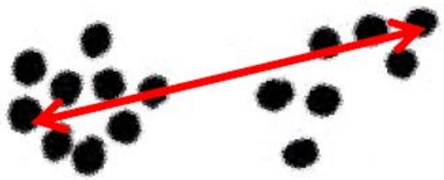
Closest Pair of Clusters?

The distance between two clusters is defined as the distance between:

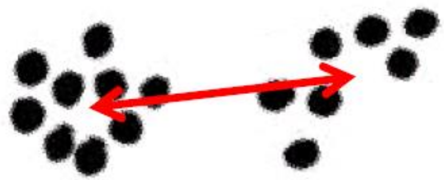
- Single-link
  - The nearest pair of points



- Complete-link
  - The farthest pair of points



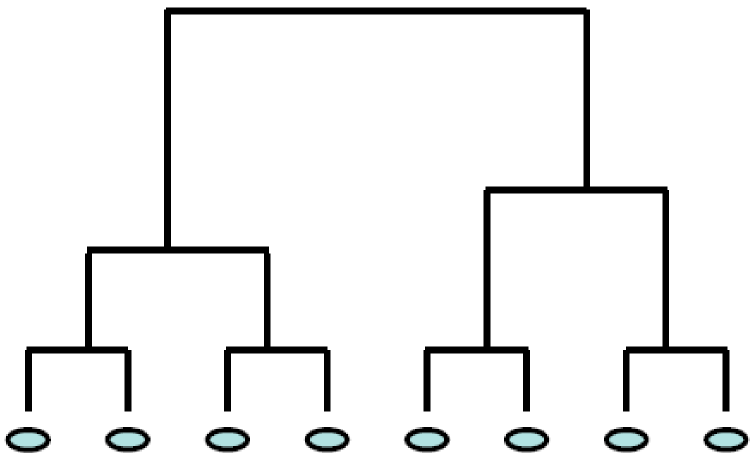
- Centroid
  - The center of gravity



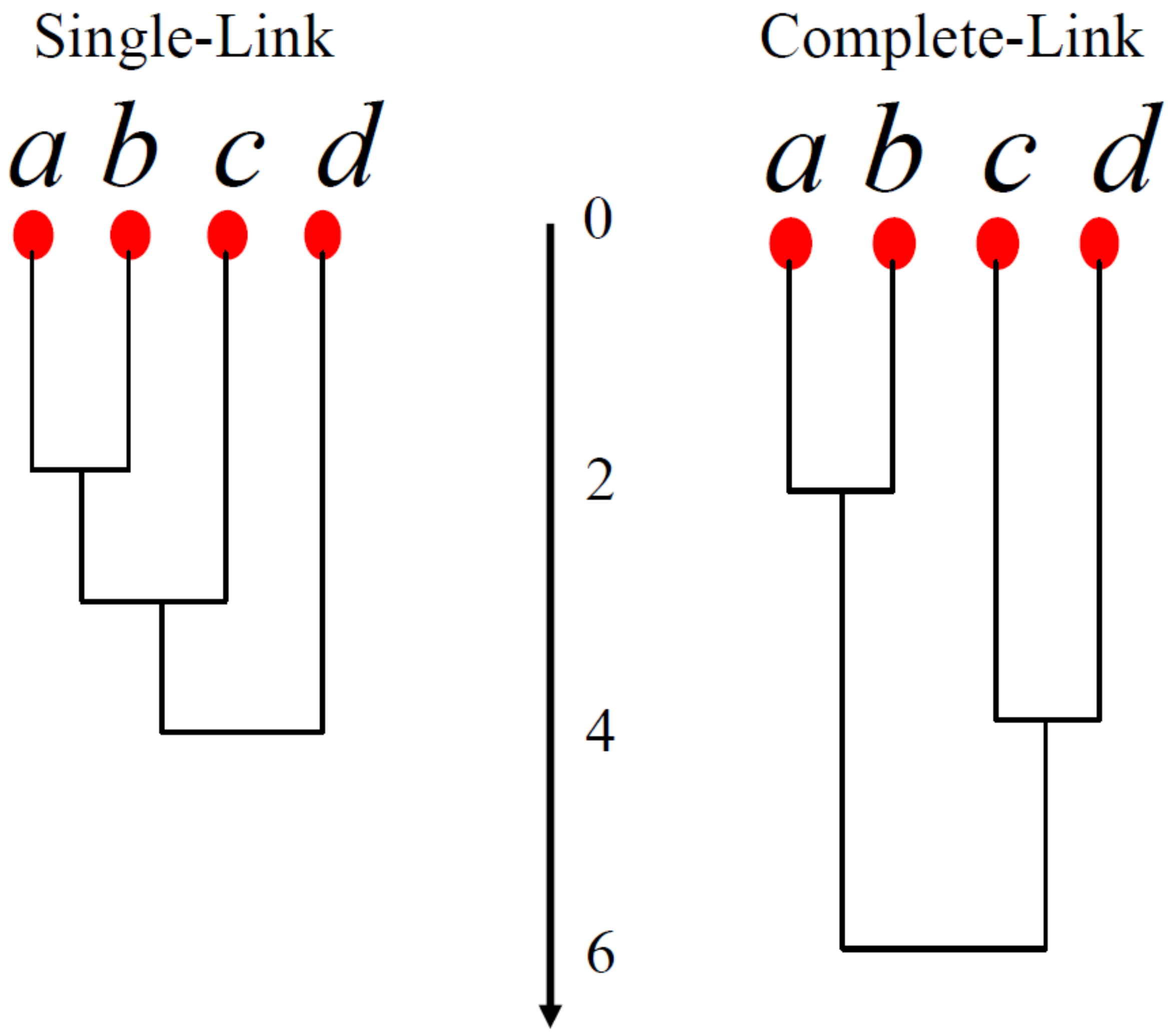
- Average-link
  - Average of all cross-cluster pairs

Visualization of the hierarchy: Dendogram

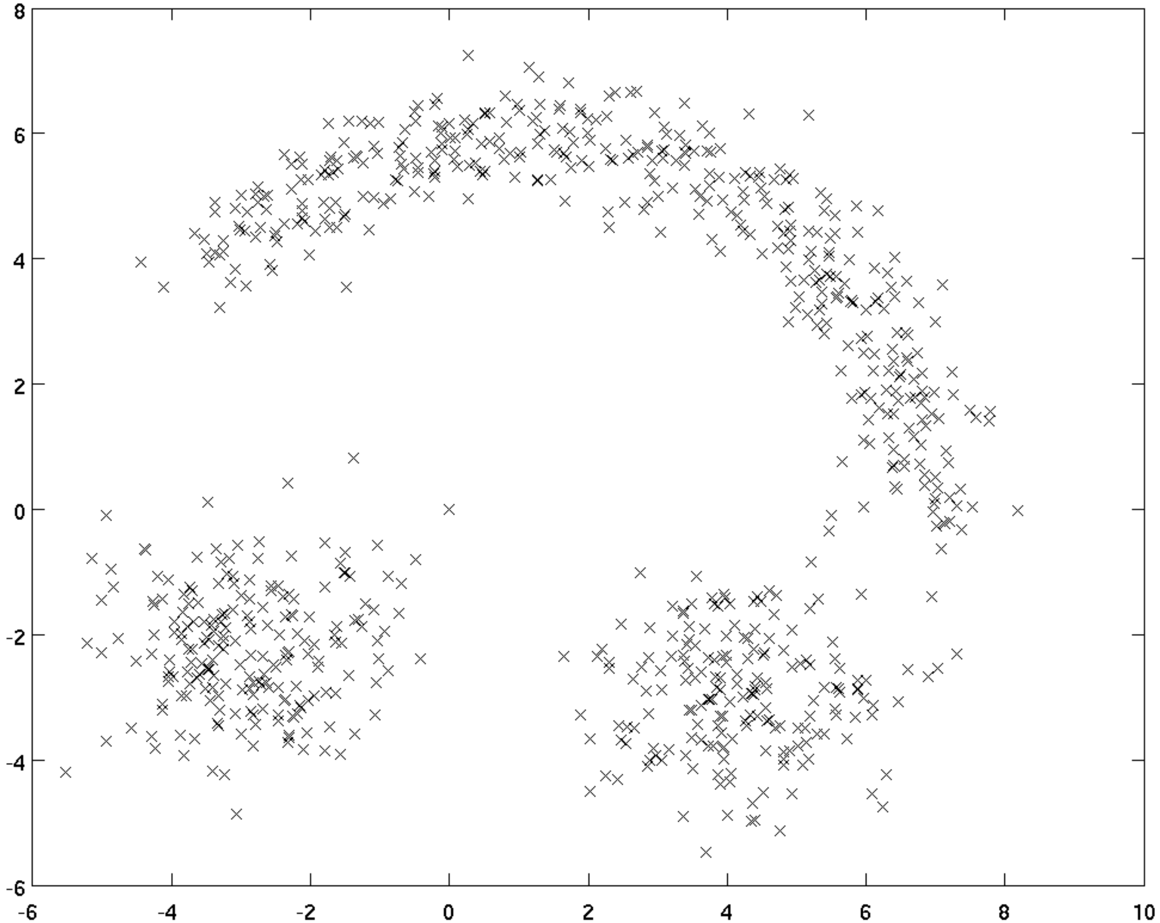
- Can be used to identify the number of clusters in data
  - A horizontal cut will create a unique clustering
  - Moving the cut from root down creates more clusters
  - Large gaps between the merging nodes indicate a good cutting point



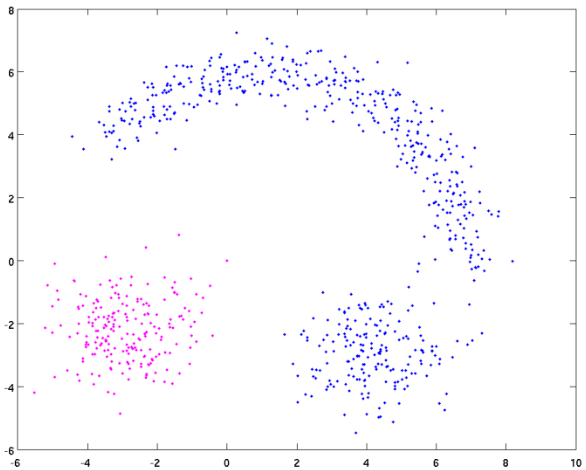
Dendograms

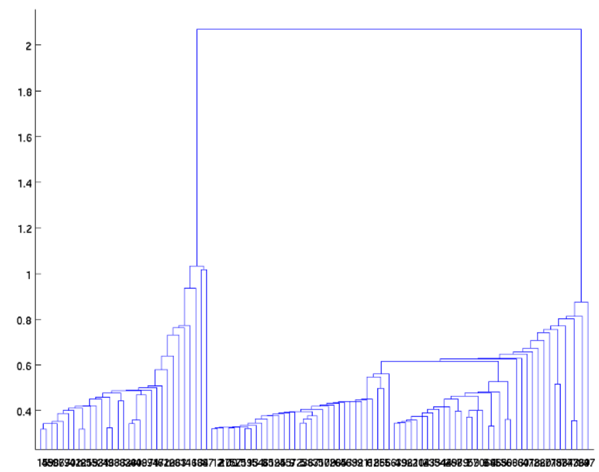


Another example

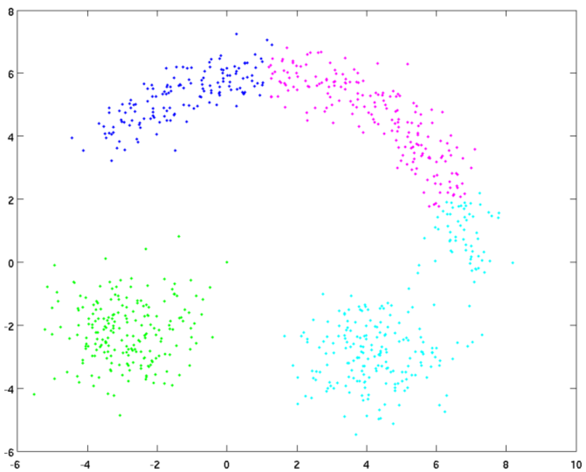


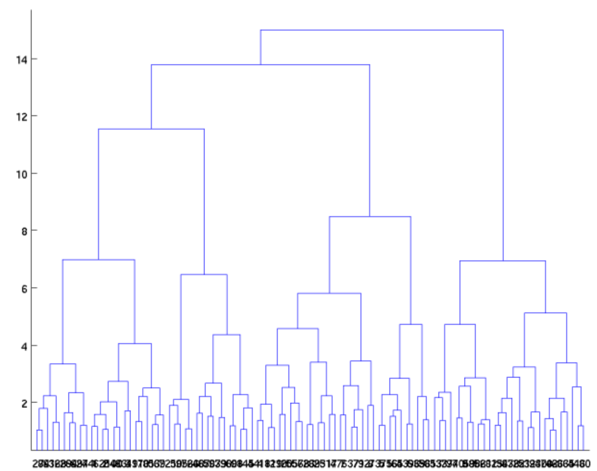
Single Link vs. Complete Link





Single





Complete

- Single-link creates straggly clusters due to chaining effect

Computational Complexity

- All hierarchical clustering methods need to compute distance of all pairs of n individual instances which is $O(n^2)$
- There are n-1 iterations, at each iteration after the merge we must compute the distance between new cluster and all other clusters

$$\sum_{i=2}^{n-1} n - i = O(n^2)$$

- In order to maintain an overall $O(n^2)$ performance, distance update must be done in constant time – trivial for complete-link and single-link

Partition Clustering

- Given a data set of n points, we know that there are k clusters in the data, how to find these clusters?
- Roughly speaking there are $O(k^n)$ ways to partition the data, Which one is better?
- One intuition says that we want tight clusters, i.e., points should be in a tight ball
- This leads to the following objective function

  $\sum_{i=1}^{k} \sum_{x \in C_i} |x - \mu_i|^2$ --- squared distance between data point x and its cluster center
- Optimizing this objective is a combinatorial optimization problem
  - Exhaustive search for an optimal solution is not feasible

Combinatorial optimization: An iterative solution

- ***Initialization:*** Start with a random partition of the data
- ***Iterative step***: the cluster assignments and cluster centers are updated to improve the objective
- ***Stopping criterion***: if no improvement can be achieved.

Iterative greedy descent

– convergence is guaranteed, but to local optimal

K-Means

Algorithm

Input – Desired number of clusters, *k*

Initialize – the *k* cluster centers (randomly if necessary)

Iterate –

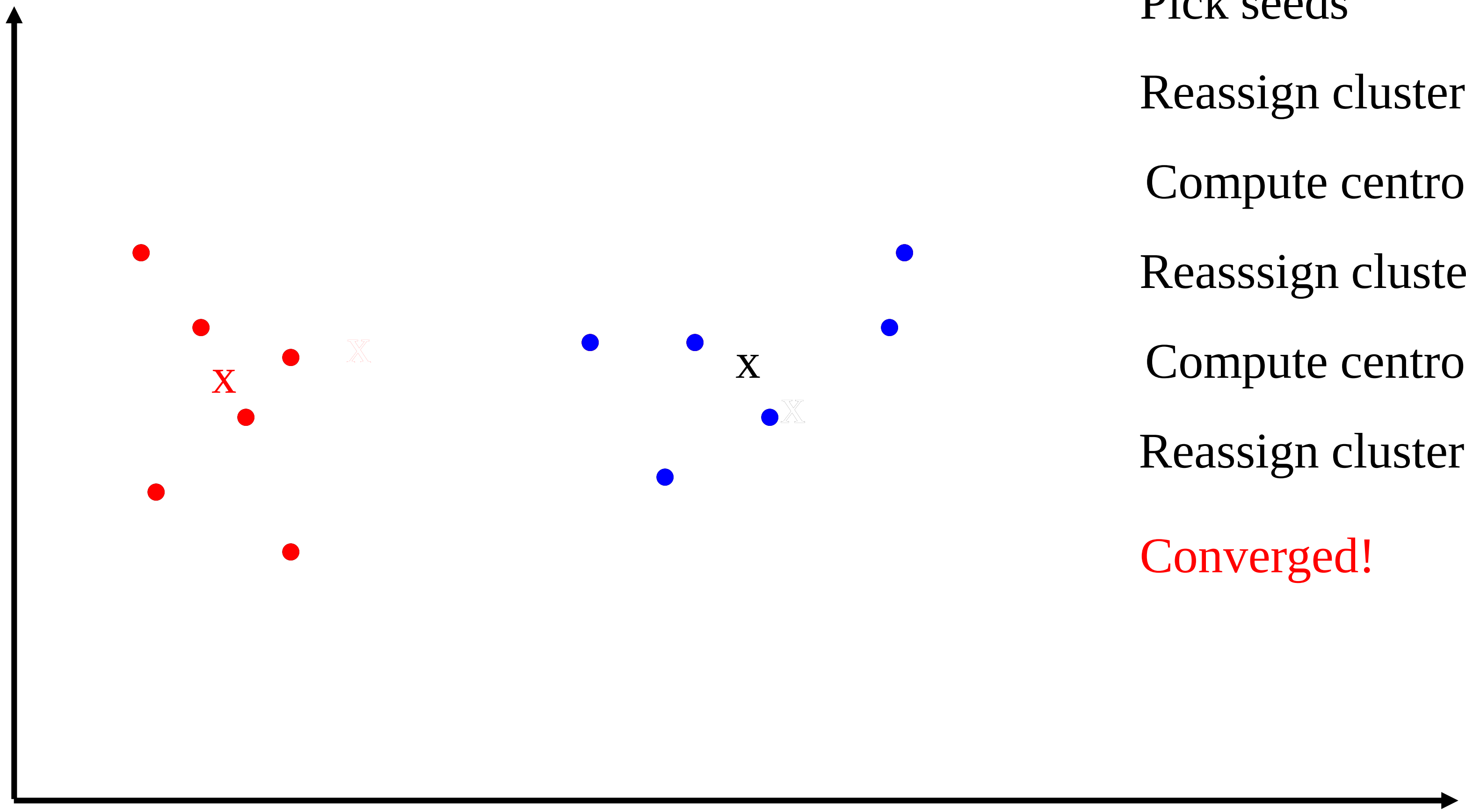
1. Assigning each of the N data points to its nearest cluster centers
2. Re-estimate the cluster center by assuming that the current assignment is correct

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Termination –

If none of the data points changed membership in the last iteration, exit. Otherwise, go to 1

K-Means Example (K=2)



Pick seeds

Reassign clusters

Compute centroids

Reasssign clusters

Compute centroids

Reassign clusters

Converged!

Computational Complexity

- At each iteration:
  - Reassigning clusters: $O(kn)$ distance computations
  - Computing centroids: Each instance vector gets added once to some centroid: $O(n)$
- Assume these two steps are each done once for I iterations: $O(Ikn)$.
- Linear in all relevant factors, assuming a fixed number of iterations, more efficient than $O(n^2)$ HAC
- Does it always converge?

K-means Convergence

Objective

$$\min_{\mu}\min_{C}\sum_{i=1}^{k}\sum_{x\in C_i}|x-\mu_i|^2$$

1. Fix μ, optimize C:

Step 1 of kmeans

$$\min_{C}\sum_{i=1}^{k}\sum_{x\in C_i}|x-\mu_i|^2 = \min_{c}\sum_{i}^{n}\left|x_i-\mu_{x_i}\right|^2$$

2. Fix C, optimize μ:

$$\min_{\mu}\sum_{i=1}^{k}\sum_{x\in C_i}|x-\mu_i|^2$$

- Take partial derivative of μ_i and set to zero, we have

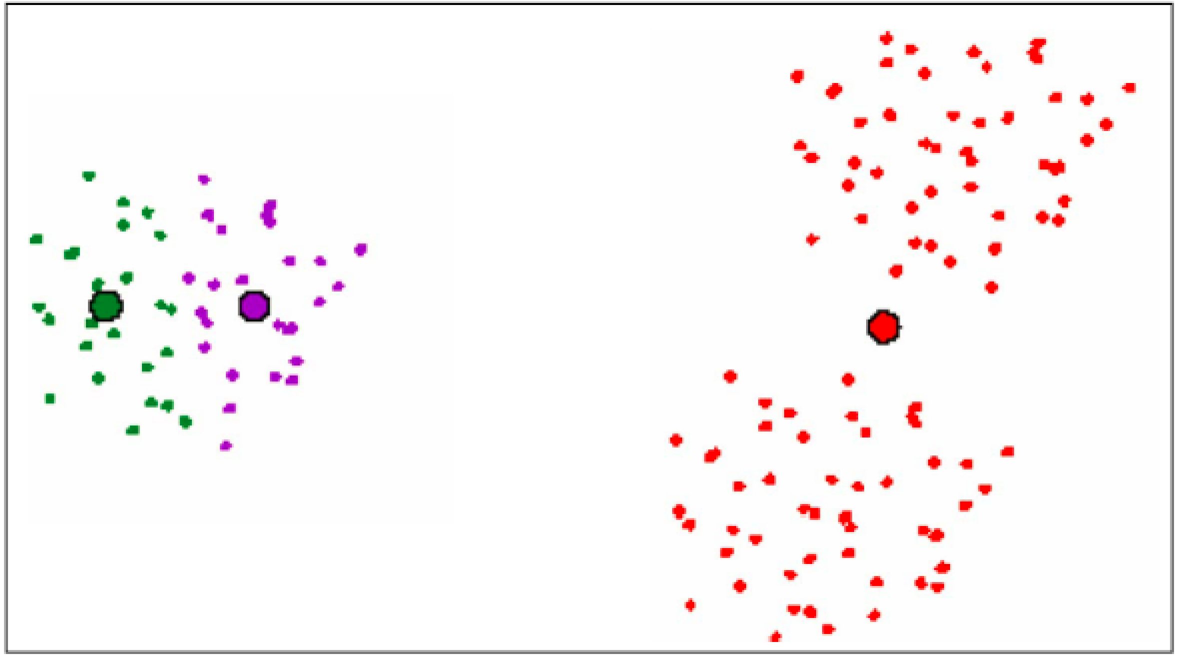
$$\mu_i = \frac{1}{|C_i|}\sum_{x\in C_i} x$$

Step 2 of kmeans

K-means takes an alternating optimization approach, each step is guaranteed to decrease the objective – thus guaranteed to converge

Impact of Initial Seeds

- Highly sensitive to the initial seeds



- Multiple random trials: choose the one with best sum of squared loss (important!)
- Heuristics for choosing better centers
  - choose initial centers to be far apart – furthest first traversal (K-Means++ algorithm)
  - Initialize with results of other clustering method

More Comments

- K-Means is exhaustive:
  - Cluster every data point, no notion of outlier
  - Outliers cause problems
    - Become singular clusters
    - Bias the centroid estimation
- K-medoids methods is more robust to outliers
  - Cluster medoid: the point that has minimum sum squared distance to all data points in the cluster
  - More expensive to compute
    - For each point: sum squared distance with all other pts in cluster $O(|C|^2)$
- Need to specify *k*: difficult in practice
  - Automatically deciding *k*? more on this later…

Soft Clustering

- Hard clustering:
  - Data point is deterministically assigned to one and only one cluster
  - But in reality clusters may overlap
- Soft-clustering:
  - Data points are assigned to clusters with certain probabilities
- Model-based clustering

Aside: Gaussian Bayes Classifier

- We have k classes in our data
- Each class contains data generated from a particular Gaussian distribution
- Data is generated by
  - first randomly select one of the classes according to class prior *p(y)*
  - draw random samples from the Gaussian distribution of that particular class

$$P(\mathbf{x}, y) = P(\mathbf{x} \mid y)P(y)$$

$$P(\mathbf{x} \mid y = i) = \frac{1}{(2\pi)^{d/2}\left|\Sigma_i\right|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x}-\boldsymbol{\mu}_i)}$$

Back to Unsupervised Learning

- Now assume we know our data is generated in the same way
- If we know the labels, we can estimate the mean and covariance of the each class using ML (Maximum Likelihood) estimation
  - Bayes Gaussian Classifier
- But for unsupervised learning, we don’t have the labels
- How can we learn the correct model from the incomplete data?

Gaussian Mixture Model

$$P(\mathbf{x}) = \sum_{i=1}^{k} P(\mathbf{x}, y=i)$$

$$= \sum_{i=1}^{k} P(\mathbf{x} \mid y=i) P(y=i)$$

$$= \sum_{i=1}^{k} \alpha_i P(\mathbf{x} | \theta_i)$$

$\underline{\alpha_i}$ = p(y=i): the class prior
Mixing parameter

$\theta_i = \{\mu_i, \Sigma_i\}$

Goal of unsupervised learning:

- Given a set of x's, estimate $\{\alpha_1, \cdots, \alpha_k, \theta_1, \cdots, \theta_k\}$
- Once the model is identified, we can compute $p(y = i|\mathbf{x})$ for $i = 1, .., k$

Maximum Marginal Likelihood

$$\arg\max_{\theta} \prod_j P(\mathbf{x}^j) = \arg\max_{\theta} \prod_j \sum_{i=1}^{k} P(\mathbf{x}^j, y^j = i)$$

$$= \arg\max_{\theta} \sum_{j=1}^{n} \underbrace{\log \sum_{i=1}^{k} P(\mathbf{x}^j, y^j = i)}$$

log sum is difficult to optimize !

Gradient ascent is doable but very inefficient

Expectation Maximization (EM)

- A highly used approach for dealing with hidden (missing) data
  - Here the cluster labels are hidden
- Much simpler than gradient methods
- It is an iterative algorithm that starts with some initial guess of the model parameters
- Iteratively performs two linked steps:
  - **Expectation (E-step)**: given current model parameters λ_t, compute the expectation for the hidden (missing) data
  - **Maximization (M-step)**: re-estimate the parameters λ_{t+1} assuming that the expected values computed in the E-step are the true values
- We will first show how it works for mixture of Gaussian

EM – simple case

- A simple case:
  - We have unlabeled data $x^1, \cdots, x^m$
  - We know there are k classes
  - We know $\alpha_1 = P(y = 1), \ldots \alpha_k = P(y = k)$
  - We don't know $\mu_1 \cdots \mu_k$, but know the common variance σ^2

Start with an initial guess for $\mu_1, \ldots, \mu_k$,

1. If we know $\mu_1, \ldots, \mu_k$, we can easily compute probability that a point x^j belongs to class i:

$$p(y = i|x^j) \propto \exp\left(-\frac{1}{2\sigma^2}|x^j - \mu_i|^2\right) p(y = i)$$

Simply evaluate this, then normalize

E-step

2. If we know the probability that each point belongs to each class, we can estimate the $\mu_1, \ldots, \mu_k$

$$\mu_i = \frac{\sum_{j=1}^m p(y = i|x^j) x^j}{\sum_{j=1}^m p(y=i|x^j)}$$

M-step

EM – Axis-aligned Gaussian

- We have unlabeled data $x^1, \cdots, x^m$
- We know there are k classes
- We know that the Gaussians are axis aligned

$$\Sigma_i = \begin{pmatrix} \sigma^2_{i,1} & 0 & 0 & \cdots & 0 & 0 \\ 0 & \sigma^2_{i,2} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \sigma^2_{i,3} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \sigma^2_{i,m-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & \sigma^2_{i,m} \end{pmatrix}$$

Start with an initial guess for $\mu_1, \dots, \mu_k, \Sigma_1, \cdots, \Sigma_k, \alpha_1, \cdots, \alpha_k,$

1. If we know the parameters, we can easily compute probability that a point x^j belongs to class i:

$$p\left(y = i \middle| x^j\right) \propto p\left(x^j \middle| \mu_i, \Sigma_i\right) p(y = i)$$

Simply evaluate this, then normalize

E-step

2. If we know the probability that each point belongs to each class, we can estimate the $\mu_1, \dots, \mu_k, \Sigma_1, \cdots, \Sigma_k, \alpha_1, \cdots, \alpha_k,$

$$\mu_i = \frac{\sum_{j=1}^m p\left(y = i \middle| x^j\right) x^j}{\sum_{j=1}^m p(y=i|x^j)}$$

$$\alpha_i = \frac{\sum_{j=1}^m p\left(y = i \middle| x^j\right)}{m}$$

$$\sigma_{il}^2 = \frac{\sum_{j=1}^m p\left(y = i \middle| x^j\right)\left(x_l^j - \mu_{il}\right)^2}{\sum_{j=1}^m p\left(y = i \middle| x^j\right)}$$

M-step

EM – General Gaussian

Start with an initial guess for $\mu_1, \ldots, \mu_k, \Sigma_1, \cdots, \Sigma_k, \alpha_1, \cdots, \alpha_k,$

1. If we know the parameters, we can easily compute probability that a point x^j belongs to class i:

$$p\left(y = i \middle| x^j\right) \propto p\left(x^j \middle| \mu_i, \Sigma_i\right) p(y = i)$$

Simply evaluate this, then normalize

E-step

2. If we know the probability that each point belongs to each class, we can estimate the $\mu_1, \ldots, \mu_k, \Sigma_1, \cdots, \Sigma_k, \alpha_1, \cdots, \alpha_k,$

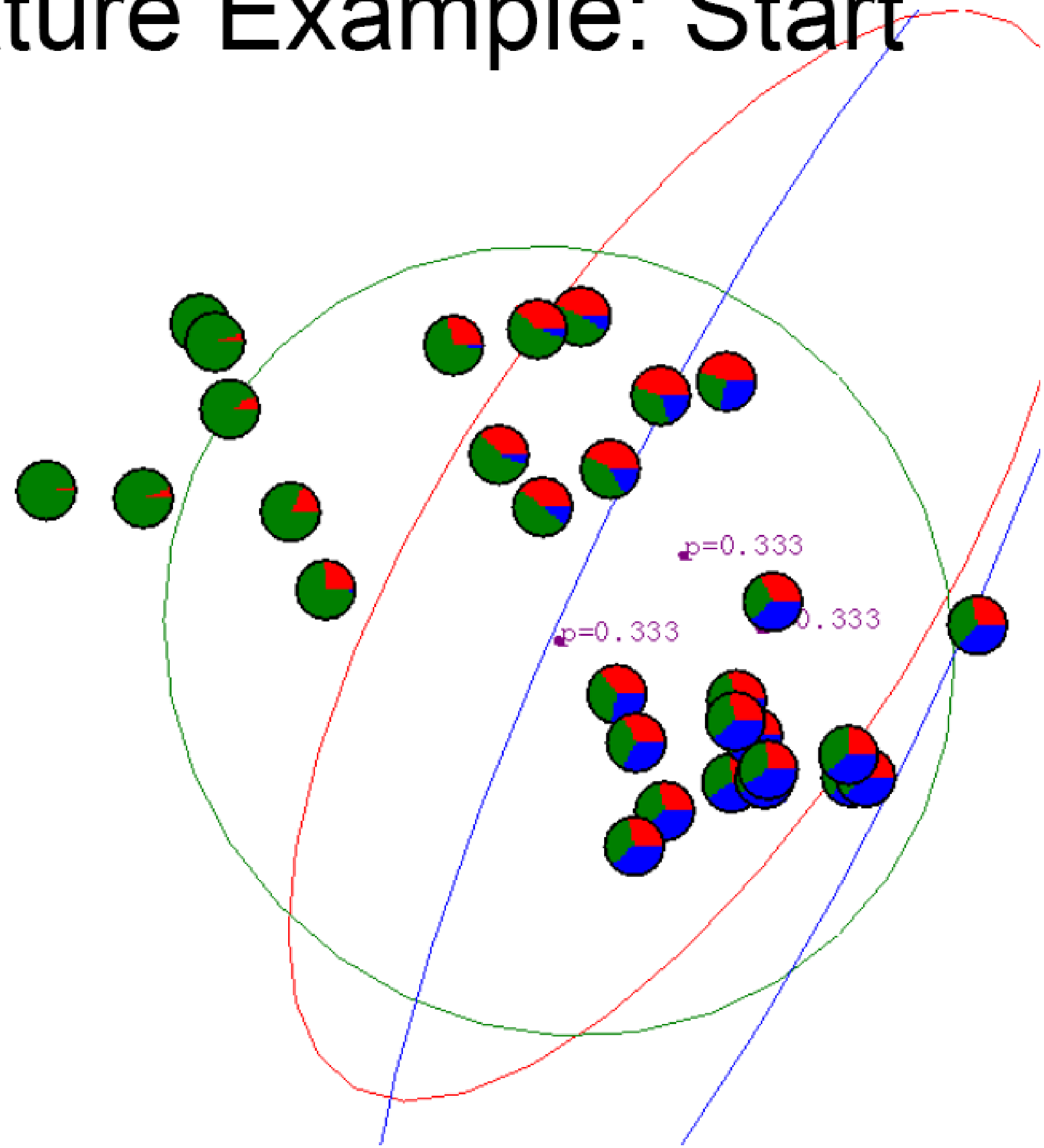
$$\mu_i = \frac{\sum_{j=1}^{m} p\left(y = i \middle| x^j\right) x^j}{\sum_{j=1}^{m} p(y=i|x^j)}$$

$$\alpha_i = \frac{\sum_{j=1}^{m} p\left(y = i \middle| x^j\right)}{m}$$

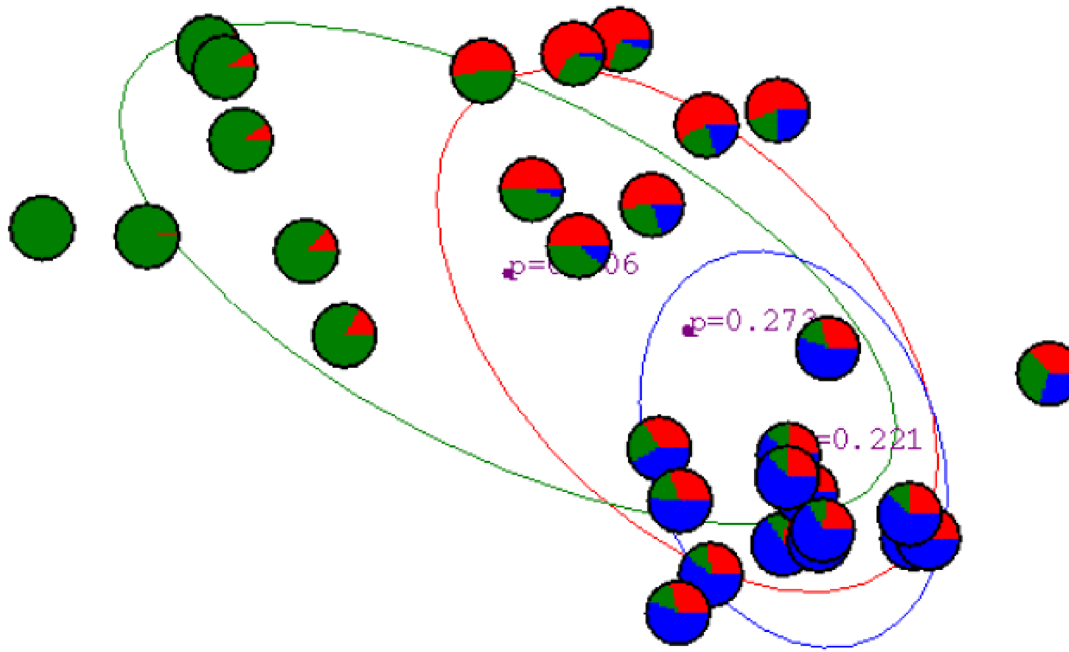
$$\Sigma_i = \frac{\sum_{j=1}^{m} p\left(y = 1 \middle| x^j\right)(x^j - \mu_i)(x^j - \mu_i)^T}{\sum_{j=1}^{m} p\left(y = j \middle| x^j\right)}$$

M-step

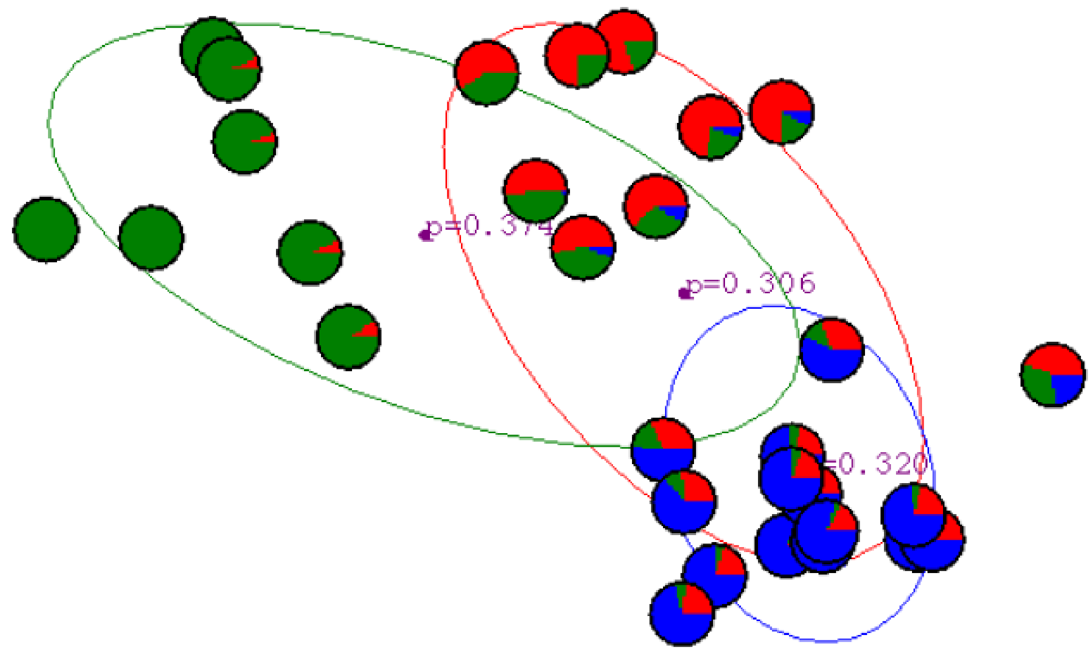
Gaussian Mixture Example: Start



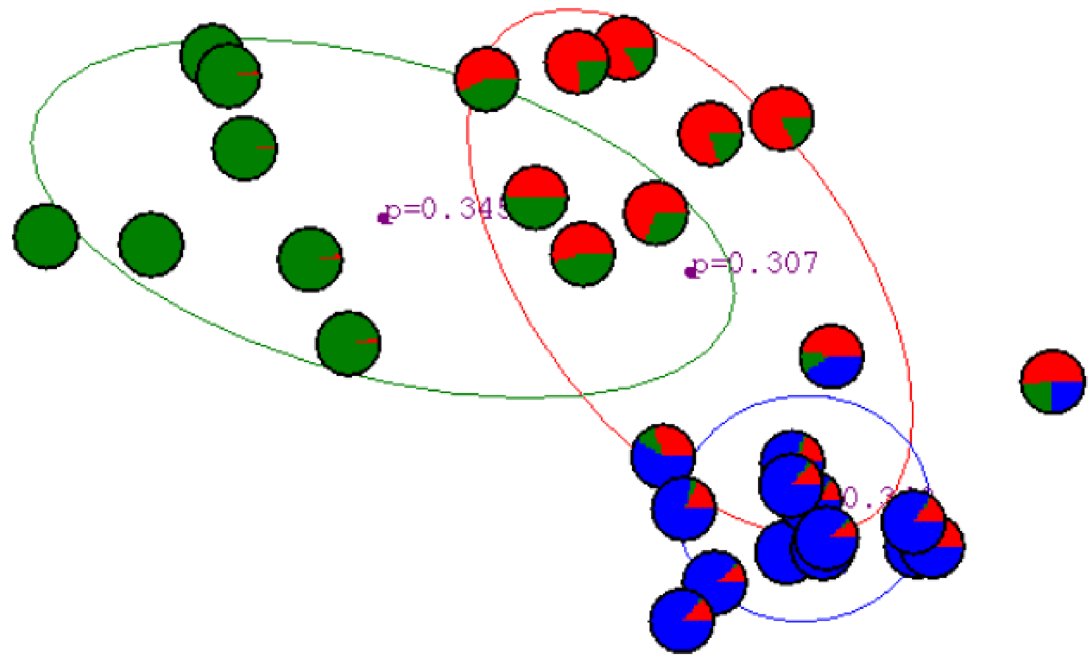
After first iteration



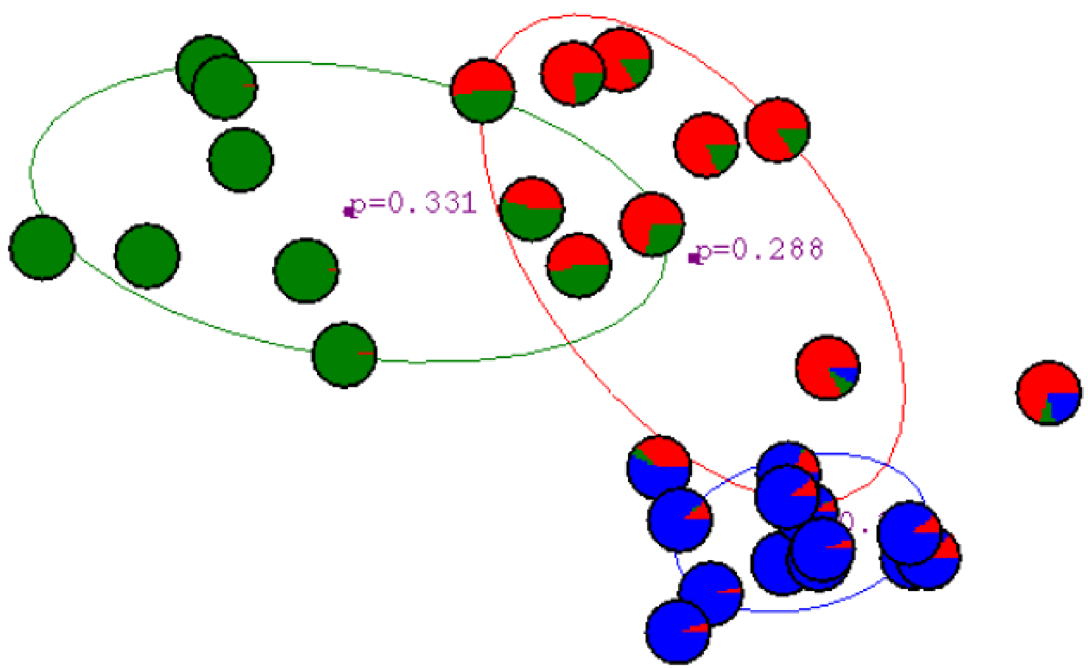
After 2nd iteration



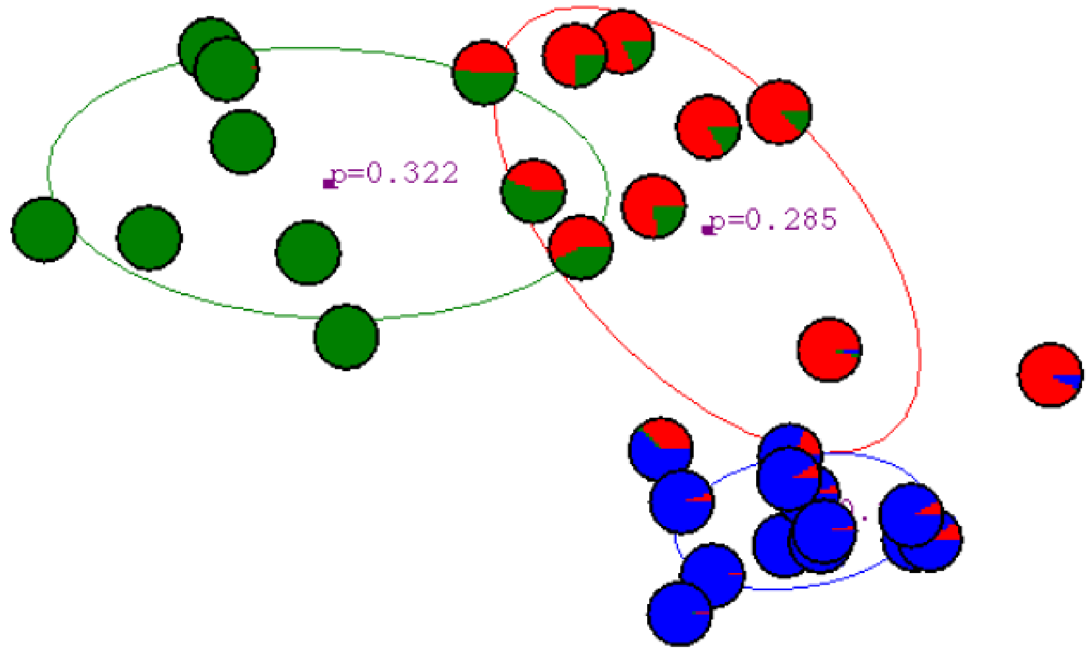
After 3rd iteration



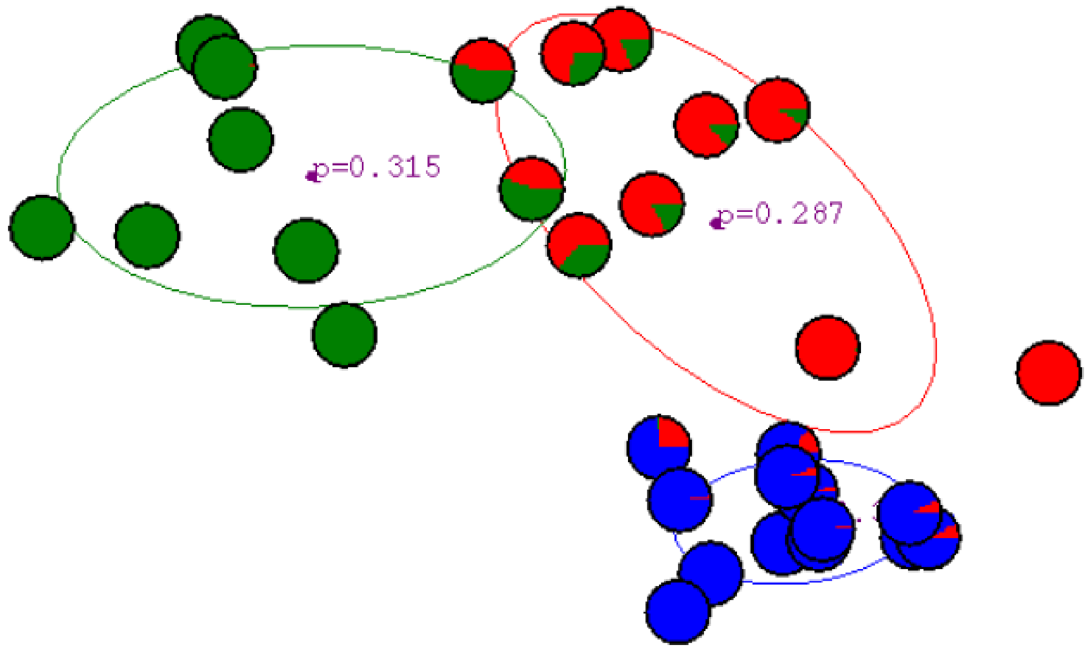
After 4th iteration



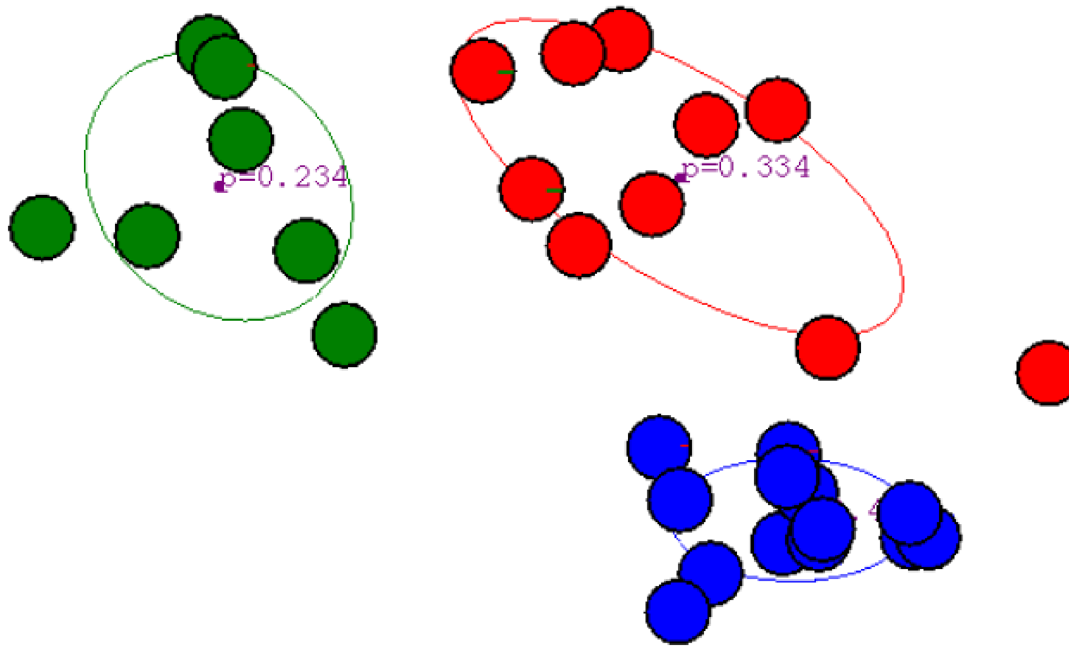
After 5th iteration



After 6th iteration



After 20th iteration



Behavior of EM

- It is guaranteed to converge
  - Convergence proof is based on the fact that $P(x|\theta)$ must increase or remain same between iterations (not obvious)
  - In practice it may converge slowly, one can stop early if the change in log-likelihood is smaller than a threshold
- It converges to a local optimum
  - Multiple restart is recommended