CS 383 – Machine Learning

Clustering

Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Overview

- Unsupervised Learning
- Clustering
  - K-means/mediods
  - Mixture of Gaussians/Expectation Maximization (EM)
  - Agglomerative Clustering
- Reading
  - Springer Section 10.3

Supervised vs. Unsupervised Learning

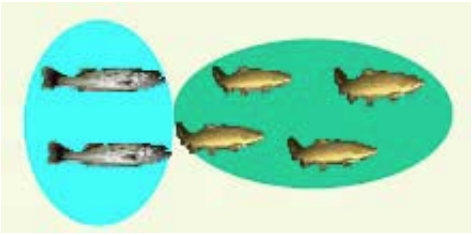
- In general our data comes in two flavors:
  1. Supervised
  2. Unsupervised
- With *supervised* data we have not only the observations,$\{X_i\}_{i=1}^N$ but also associated labels, $\{Y_i\}_{i=1}^N$.
  - Together these form our dataset: $\{X_i, Y_i\}_{i=1}^N$
  - We will later use this type of data to do
    - Regression
    - Classification
- With *unsupervised* data we only have the observations, $\{X_i\}_{i=1}^N$

Why Unsupervised Learning?

- It's harder ☹
  - How do we know if results are meaningful since there no answers (labels) to test against?
    - Let experts look at the results (external evaluation)
    - Define some objective function on clustering (internal evaluation)
- We need it though
  - Labeling large datasets is very costly
  - May have no idea what/how many classes there are (data mining)
  - May want to use clustering to gain some insight into the structure of the data before designing a classifier

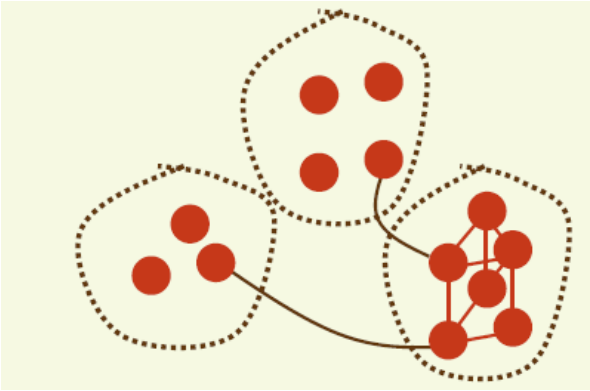
Unsupervised Learning

- The most common type of unsupervised learning is *clustering* where we attempt to learn underlying patters from data



What is Clustering?

- Clustering: The process of grouping a set of objects into classes of similar objects
  - Items within cluster should be similar
  - Observations from different clusters should be dissimilar
- This is the most common form of *unsupervised learning*



Issues for Clustering

- Need a notion of similarity/distance
  - Similarity: Gaussian, Cosine
  - Distance: L2 (Euclidian), L1 (Manhattan)
  - See Blackboard for these
- How many clusters?
  - Fixed a priori?
  - Completely data driven?
    - Avoid “trivial” clusters – too small or too large
    - Use some statistics?
- How to evaluate cluster quality?
  - It’s unsupervised after all….

Hard vs Soft Clustering

- Hard clustering: Each observation belongs to exactly one cluster
  - More common and easier to do
- Soft clustering: An observation can belong to more than one cluster
  - And/or can belong to clusters with some probability

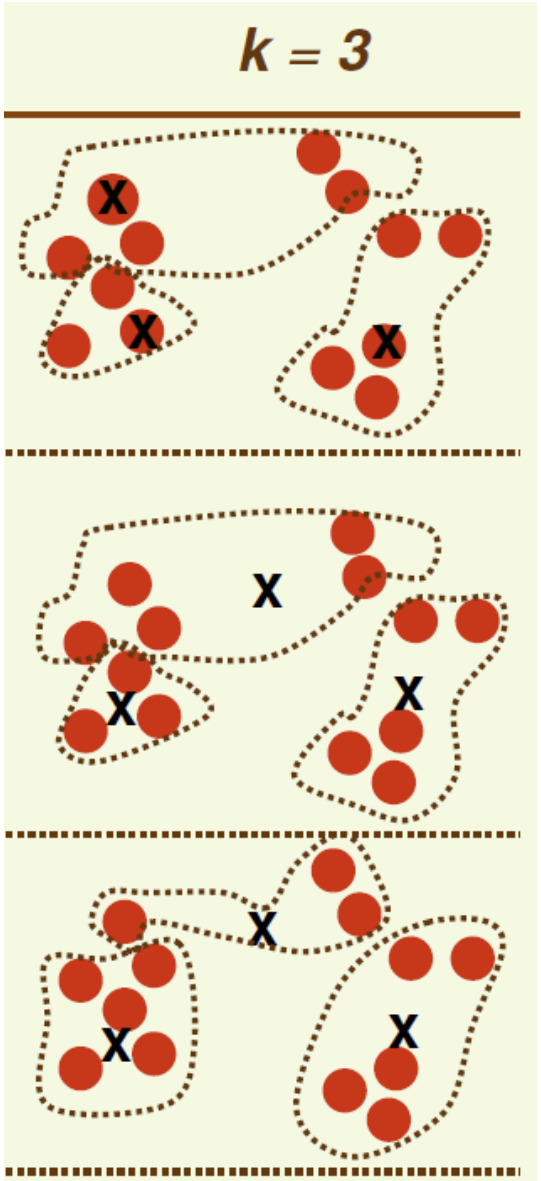
Clustering Algorithms

- Flat algorithms
  - Usually start with a random (partial) partitioning
  - Refine it iteratively
    - k-means/medoids clustering
    - Model-based clustering (GMM)
- Hierarchical algorithms
  - Bottom-up, agglomerative
  - Top-down, divisive

k-Means/Mediods/Modes

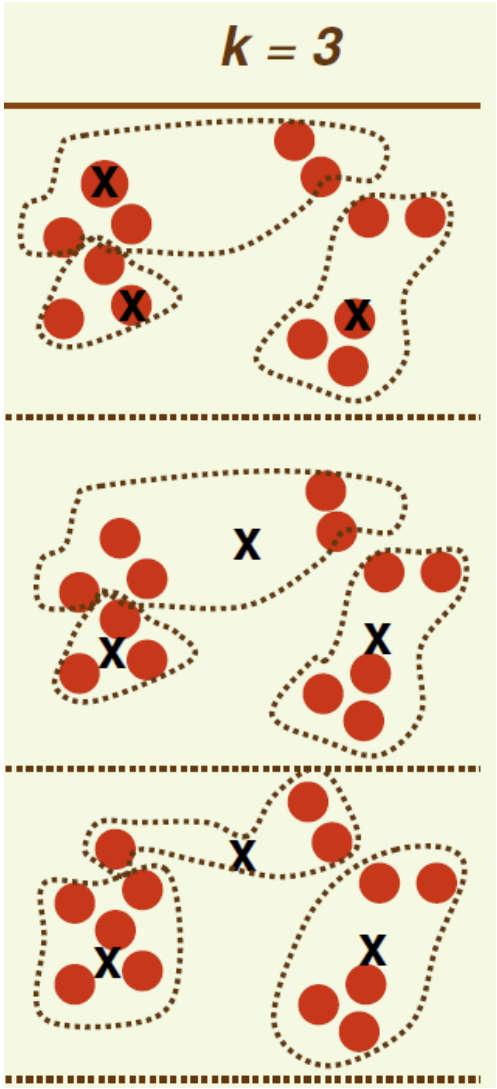
k-means/medoids

- Assumes observations are real-valued vectors
- Each observation is associated with a single *reference vector* (i.e cluster center).
  - This is typically the reference vector that the observation is closest to using some distance measurement
- The initial reference vectors are often chosen at random



k-means/mediods

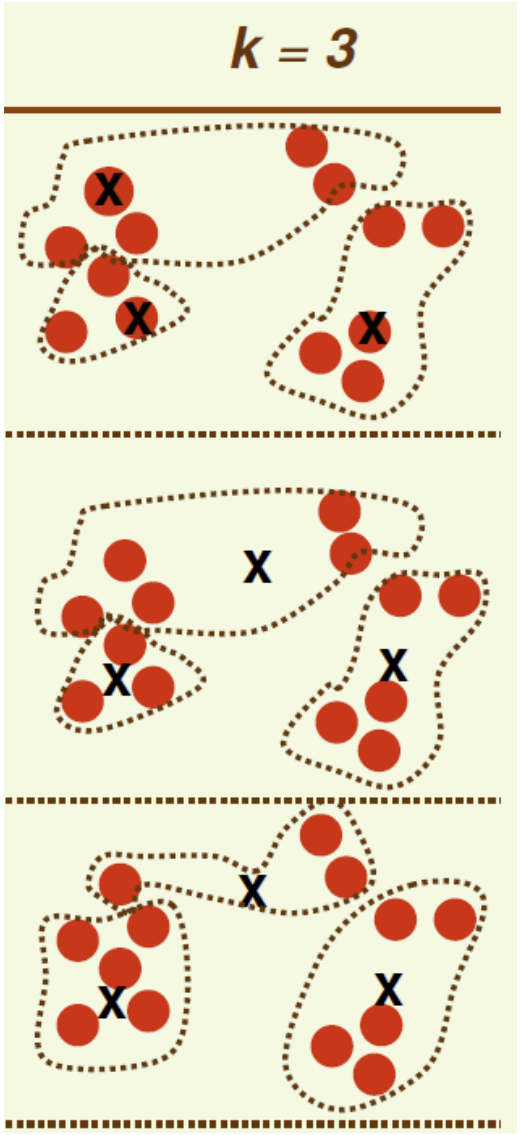
- At each iteration
  1. Each observation is reassignment to the reference vector closest to it
  2. A new reference vector is computed based on the observations that are associated with it
- This continues until some termination criteria is met
  - Number of iterations
  - Threshold on change of reference vectors.



k-means

- If the new reference vector is computed as the *mean* of the observations associated with it, then this is *k-means*
- Given cluster C_i (set of observations associated with reference vector i) of size $|C_i|$, we compute the center of gravity or mean of points in that cluster as:

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

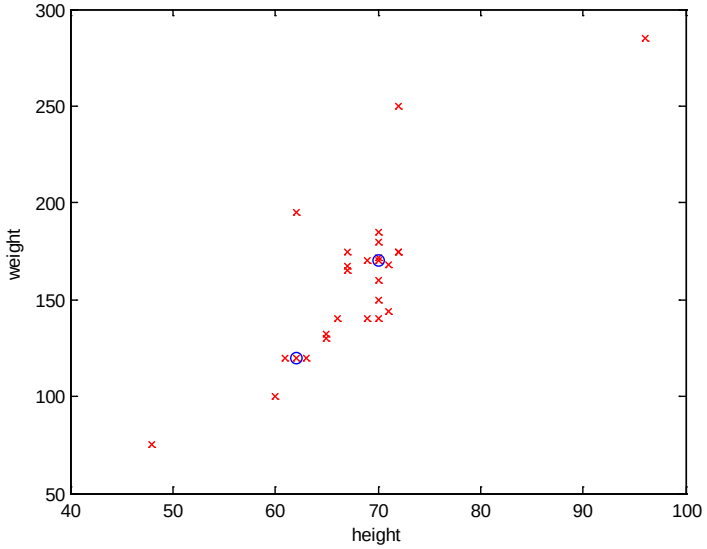


k-Means Algorithm

- Select k random vectors $\{s_1, s_2, \ldots, s_k\}$ as the initial reference vectors (cluster centers).
  - Therefore initially $\mu_i = s_i$
- Until clustering converges or other stopping condition:
  - For each observation x
    - Assign x to the cluster C_i such that
$$i = \underset{i}{\operatorname{argmin}}\big(dist(x, \mu_i)\big)$$
  - For each cluster C_i compute the new reference vector as the mean of those associated with it:
    - Update $\mu_i = \frac{1}{|C_i|}\sum_{x \in C_i} x$

k-Means Example

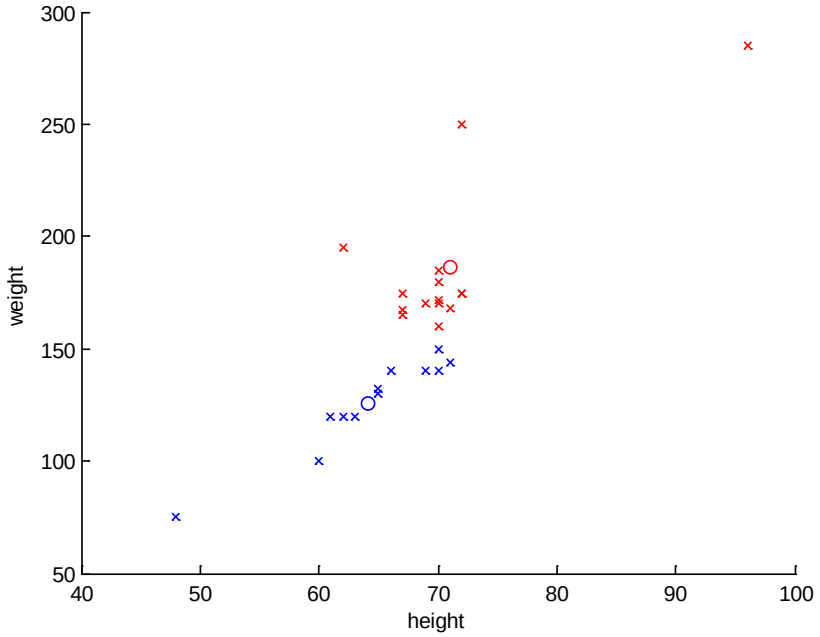
- We want to figure out t-shirt sizes based on people's height and weight
- Let's assume we want 2 shirt sizes
  - $k = 2$
- Initial reference vectors



height (inches)	weight (pounds)
61	120
65	130
72	250
63	120
62	195
62	120
60	100
70	140
70	160
65	132
48	75
72	175
67	167
69	140
96	285
70	172
70	185
71	168
70	180
69	170
70	150
70	170
71	144
66	140
67	175
67	165
72	175

Example

- After 4 iterations we get the following assignments
- And the reference vectors are at:
  - μ_1=[71, 168.133]
  - μ_2=[64.167, 125.9167]
- So we consider our final model $< \mu_1, \mu_2 >$
- Given a new person with measurements x, we can assign them a shirt size based on the model
  - Which are they closest to?



Termination Conditions

- Several possibilities
  1. A fixed number of iterations
  2. Assignment partition unchanged
  3. Centroid positions don’t change

Convergence

- Why should the K-means algorithm ever reach a fixed point?
  - A state in which clusters don't change
- K-means is a special case of a general procedure known as *Expectation Maximization (EM)*.
  - EM is known to converge
  - Though number of iterations could be large (but generally isn't)
  - We're going to look at EM in a minute

Convergence of K-Means

- Let's prove that it converges
- Let a_i be the reference vector for cluster i
- Define "goodness" measure of cluster k as sum of squared distances from the reference vector:

$$G_i = \sum_{x \in C_i} (x - a_i)^2$$

Convergence of K-Means

- To converge, the change in this goodness should be zero.
- So we'll take the derivative of G_i with respect to a_i

$$\frac{d}{da_i}\left(\sum_{x \in C_i}(x - a_i)^2\right)$$

$$\sum_{x \in C_i} -2(x - a_i) = 0$$

- We can re-write this as

$$\sum_{x \in C_i} x = \sum_{x \in C_i} a_i$$

Convergence of K-Means

$$\sum_{x \in C_i} x = \sum_{x \in C_i} a_i$$

- Let $|C_i|$ be the number of members in cluster C_i
- Then $\sum_{x \in C_i} a_i = |C_i| a_i$
- Therefor (via substitution):

$$a_i = \left(\frac{1}{|C_i|} \sum_{x \in C_i} x \right)$$

- Which is the definition of the mean of the cluster, $\mu(C_i)$
- So if $a_i = \mu(C_i)$ then we have converged

Initial Reference Vector Choice

- Results can vary based on initial reference vector choice.
- Some choices can result in poor convergence rate, or convergence to sub-optimal clustering
- Some ideas might be:
  - Select good initial reference vectors using a heuristic (e.g. instances least similar to any existing mean)
  - Try out multiple starting points
  - Initialize with the results of another method

Weaknesses of k-Means

- The algorithms is only applicable if the *mean* is defined
  - For **categorical** data use k-mode where the centroid is represented by most frequent values
- The user needs to specify k
- The algorithm is sensitive to outliers
  - Outliers are data points that are very far away from other data points
  - Outliers could be errors in the data recording or some special data points with very different values
  - One solution is to use **k-medoids** (which uses the L1 distance and chooses the median of each feature)

Expectation Maximization

Via (Gaussian) Mixture Models

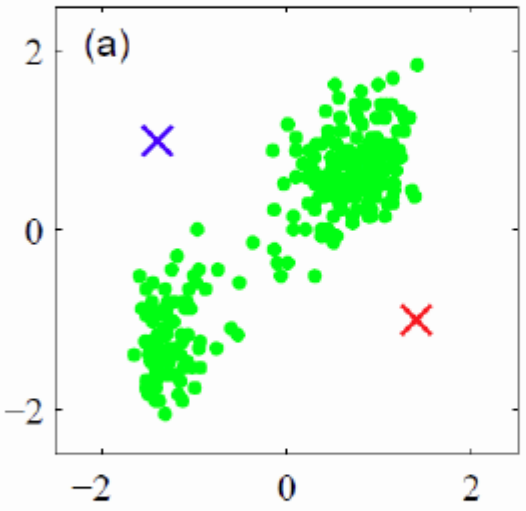
Expectation Maximization

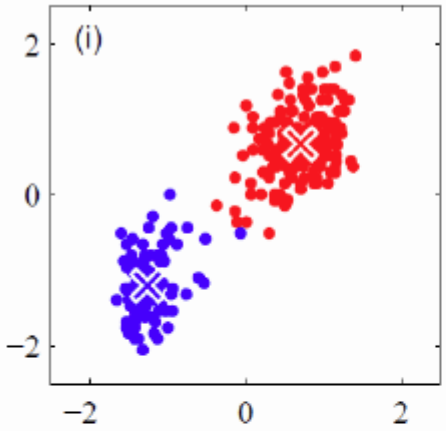
- k-means (medians, etc..) is an instance of an *expectation-maximization* (EM) algorithm.
- These types of algorithms try to find a solution by alternating between two steps:
  1. *Expectation* – Here's we predict out outcome.
     - For k-means, this is assigning each instance to a cluster.
  2. *Maximization* – Here we update our model to maximize/minimize something.
     - For k-means this is moving the reference vector to the mean of the cluster in order to minimize the distance.
     - Conversely we sometimes maximize the log-likelihood

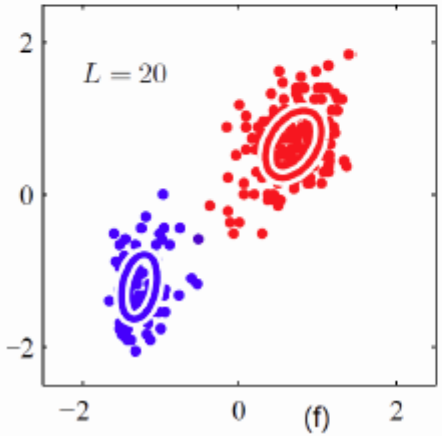
Gaussian Mixture Models

- Another example of expectation-maximization (EM) is discovering parameters for Gaussians
  - Our final "model" is set of Gaussians (Gaussian Mixture Model, or GMM)
- Here rather than identifying clusters by "nearest" centroids, we instead fit a set of k Gaussians to the data
  - This involves updating (M-Step) both the mean and variance of each Gaussian.

K-Means vs GMM





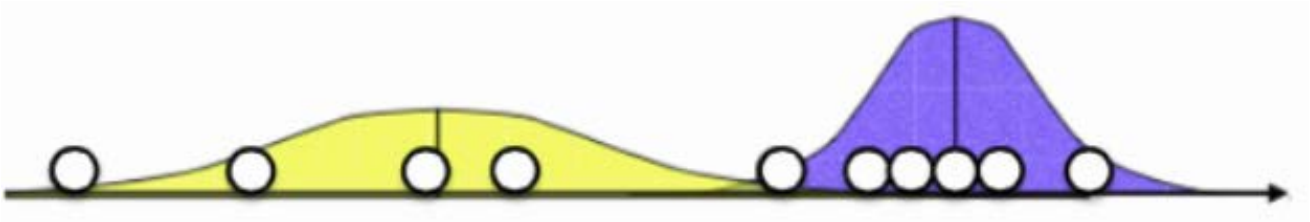


Hard clustering: a data point is assigned a cluster.

Soft clustering: a data point is explained by a mix of multiple Gaussians probabilistically.

Mixture Models in 1D

- Given:
  - Observations $\{X_i\}_{i=1}^{N}$
  - We decided that we want to model the data using $\mathrm{k} = 2$ Gaussians.
- Output
  - Our goal is to find the parameters of the two Gaussians, $\mathcal{N}_1(\mu_1, \sigma_1)$ and $\mathcal{N}_2(\mu_2, \sigma_2)$, pertaining to the two clusters, C_1 and C_2, respectively, that maximize the log likelihood of the data (or minimize some distance)



1D EM for GMM

- Similar to k-means…
- EM for GMM
  1. Start with two random starting vectors μ_1 and μ_2 and compute sample standard deviations σ_1 and σ_2 to initialize Gaussians $\mathcal{N}_1(\mu_1, \sigma_1)$ and $\mathcal{N}_2(\mu_2, \sigma_2)$
  2. Initialize probabilities of each Gaussian. We will need this for the E-Step computations. Typically this will be done uniformly unless there is prior information: $P(\mathcal{N}_i) = \frac{1}{k}$ where k is the number of Gaussians.
  3. (E-Step): For each observation compute the likelihood (expectation) of each Gaussian given the observation.
     - That is, compute $P(\mathcal{N}_1|x)$ and $P(\mathcal{N}_2|x)$
     - We'll see how to do this in the next slide
  4. (M-Step): Adjust $\mathcal{N}_1(\mu_1, \sigma_1)$ and $\mathcal{N}_2(\mu_2, \sigma_2)$, $P(\mathcal{N}_1)$ and $P(\mathcal{N}_2)$ to better fit the estimates (maximize the likelihood).
     - We'll also see how to do this in the next slide as well
  5. Iterate (repeat 3-4) until convergence

Mixture Models in 1D

Computing the expectation $P(\mathcal{N}_i|x)$

- Given $P(x|\mathcal{N}_i)$ and $P(\mathcal{N}_i)$ we can compute $P(\mathcal{N}_i|x)$ using Bayes' Rule!

$$P(\mathcal{N}_i|x) = \frac{P(x|\mathcal{N}_i)P(\mathcal{N}_i)}{\sum_{j=1}^{C} P(x|\mathcal{N}_j)P(\mathcal{N}_j)}$$

- Since the denominator is independent of the Gaussian $\mathcal{N}_i$ we can compute

$$P(\mathcal{N}_i|x) \propto P(x|\mathcal{N}_i)P(\mathcal{N}_i)$$

- $P(\mathcal{N}_i)$ has some current value (to be updated in the E-Step)
- Assuming a Gaussian distribution, $P(x|\mathcal{N}_i)$ is proportional to the probability density function (pdf) as

$$P(x|\mathcal{N}) \propto p(x|\mathcal{N}(\mu,\sigma)) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

  - In 1D space we can use MatLab's `normpdf` (normal probability density function) to compute this
  - In multi-dimensional space we'll use `mdnpdf` (see multi-dimensional slides later)

- Finally we can normalize the distributions so that

$$\sum_{j=1}^{C} P(\mathcal{N}_j|x) = 1$$

Mixture Models in 1D

To Maximization!

- Just like with k-means, in order to find the new centers (part of the maximization step) we want to compute the average vector of those observations associated with the cluster.
- However unlike k-means, observations don't just belong to one cluster.
- Therefor to compute the Gaussian parameters we will weight the samples according to $P(\mathcal{N}|x)$ computed during the E-Step

$$\mu_i = \frac{\sum_{j=1}^{N}\left(P\left(\mathcal{N}_i\middle|X_j\right)X_j\right)}{\sum_j^N P\left(\mathcal{N}_i\middle|X_j\right)},\ \sigma_i^2 = \frac{\sum_{j=1}^{N}\left(P\left(\mathcal{N}_i\middle|X_j\right)(X_j-\mu_i)^2\right)}{\sum_j^N P\left(\mathcal{N}_i\middle|X_j\right)}$$

- Finally we need to update $P(\mathcal{N}_i)$ which is just the average weight:

$$P(\mathcal{N}_i) = \frac{1}{N}\sum_{j=1}^{N} P\left(\mathcal{N}_i|X_j\right)$$

1D EM for GMM

Putting this all together…

1. Start with two random starting vectors μ_1 and μ_2 and compute sample standard deviations σ_1 and σ_2 to initialize Gaussians $\mathcal{N}_1(\mu_1, \sigma_1)$ and $\mathcal{N}_2(\mu_2, \sigma_2)$
2. Initialize probabilities of each Gaussian. Typically this will be done uniformly unless there is prior information:

$$P(\mathcal{N}_i) = \frac{1}{k}$$

3. (E-Step): For each observation compute the probability of each Gaussian given the observation

$$P(\mathcal{N}_1|x) \text{ and } P(\mathcal{N}_2|x)$$

4. (M-Step):
   a. Using the expectations computed in the E-Step, update the Gaussian Parameter μ_i, σ_i
   b. Update the priors $P(\mathcal{N}_i)$
5. Iterate (repeat 3-4) until convergence

Example: 1D GMM

- Datapoints (D=1)
  - $X = [.78, .72, .66, .51, .86, .83, .53, .32, .79, .97]^T$
- Let's assume two Gaussians ($k = 2$)
- Pick two samples as initial means
  - μ_1=.78, μ_2=.51
- Use the standard deviation of the data relative to the means for the initial standard deviation
  - σ_1=0.2135., σ_2=0.2771
- Assuming uniform distribution of classes (initially)
  - $P(\mathcal{N}_1) = P(\mathcal{N}_2) = 0.5$

Example: 1D GMM

- Start the EM process!
- (E-Step)
  - For data point 0.78
    - $p(0.78|\mathcal{N}_1) = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-(0.78-\mu_1)^2/(2\sigma_1^2)}$
    - $p(0.78|\mathcal{N}_1) = 1.8689$
    - $p(0.78|\mathcal{N}_2) = 0.8956$
    - $P(\mathcal{N}_1|0.78) \propto p(0.78|\mathcal{N}_1)P(\mathcal{N}_1) = 0.9345$
    - $P(\mathcal{N}_2|0.78) \propto p(0.78|\mathcal{N}_2)P(\mathcal{N}_2) = 0.4478$
    - Normalize so $P(\mathcal{N}_1|0.78) + P(\mathcal{N}_2|0.78) = 1$
      - $P(\mathcal{N}_1|0.78) = 0.6760$
      - $P(\mathcal{N}_2|0.78) = 0.3240$
  - Etc.. for all other points

1D GMM

- (M-Step)
  - $\mu_1 = \frac{\sum_{i=1}^{N} P(\mathcal{N}_1|x_i)x_i}{\sum_{i=1}^{N} P(\mathcal{N}_1|x_i)}$
  - $\sigma_1^2 = \frac{\sum_{i=1}^{N} P(\mathcal{N}_1|x_i)(x_i-\mu_1)^2}{\sum_{i=1}^{N} P(\mathcal{N}_1|x_i)}$
  - $P(\mathcal{N}_1) = \frac{1}{N}\sum_{i=1}^{N} P(\mathcal{N}_1|x_i)$

- And continue until convergence…

Mixture Models in D>1

- If we have more than 1 dimensional data (as is usually the case) then we must replace the standard deviation **s** with the covariance Σ:

$$p(x|\mathcal{N}_1) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_1|}} \exp\left(-\frac{(x-\mu_1)\Sigma_1^{-1}}{2}\right)$$

  - Where D is the dimensionality of the data and $|\Sigma_1|$ is the determinant of Σ_1
  - Which we can compute in MatLab with `mvnpdf`

Mixture Models in D>1

- Other than that it's pretty much the same!
  - With some allowance for matrix algebra
- Expectation
  - $p(x|\mathcal{N}_i) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_i|}} \exp\left(-\frac{(x-\mu_i)\Sigma_i^{-1}(x-\mu_i)^T}{2}\right)$
  - $P(\mathcal{N}_i|x) \propto p(x|\mathcal{N}_i)P(\mathcal{N}_i)$
  - Then normalize the distributions

Mixture Models in D>1

- Maximization
  - Update mean for feature j of Gaussian i (using updated prior)

$$\mu_{i,j} = \frac{\sum_{n=1}^{N} P(\mathcal{N}_i|X_n) X_{n,j}}{\sum_{n=1}^{N} P(\mathcal{N}_i|X_n)}$$

  - Covariance (using posterior from E stage, updated prior and mean from M stage) between features j and t of Gaussian i:

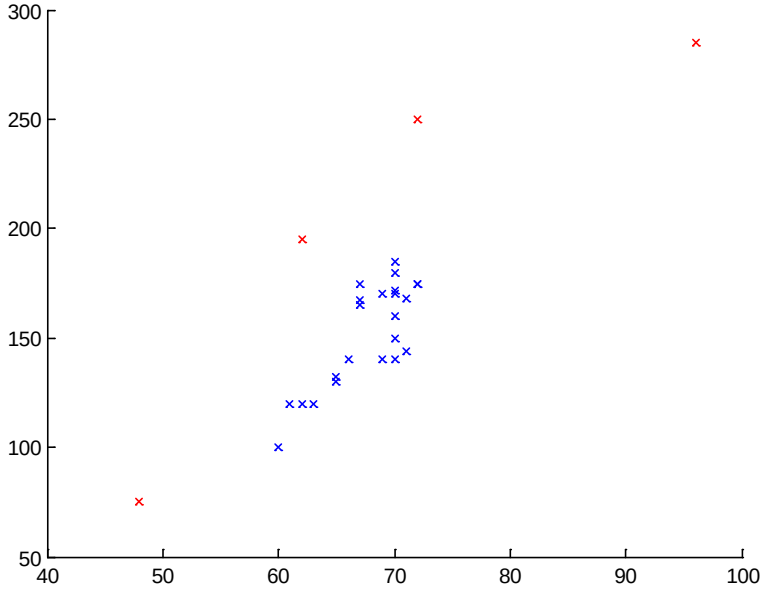
$$(\Sigma_i)_{j,t} = \frac{\sum_{n=1}^{N} P(\mathcal{N}_i|X_n)\left(X_{n,j} - \mu_{i,j}\right)\left(X_{n,t} - \mu_{i,t}\right)}{\sum_{n=1}^{N} P(\mathcal{N}_i|X_n)}$$

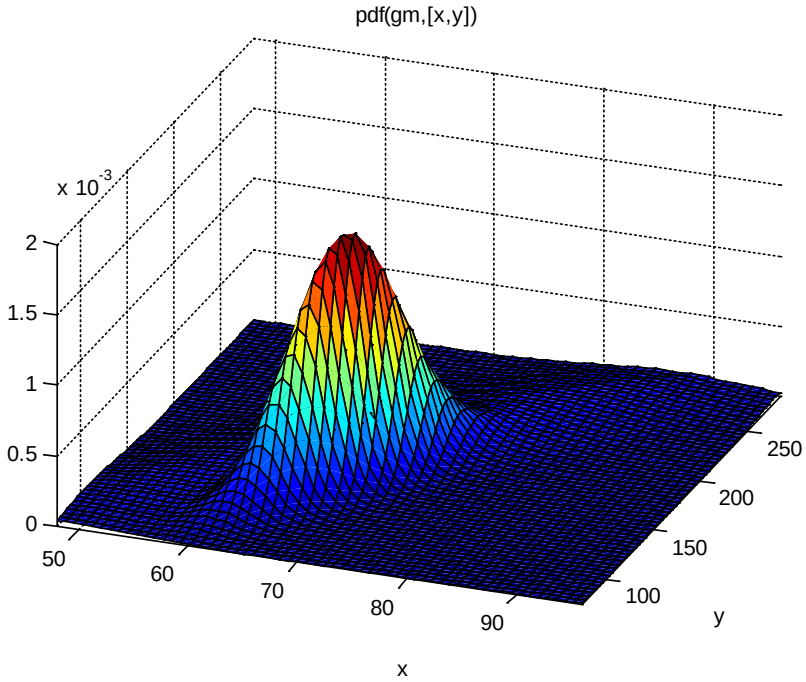
  - Prior for Gaussian i:

$$p(\mathcal{N}_i) = \frac{1}{N} \sum_{j=1}^{N} P(\mathcal{N}_i|X_j)$$

Multivariate GMM

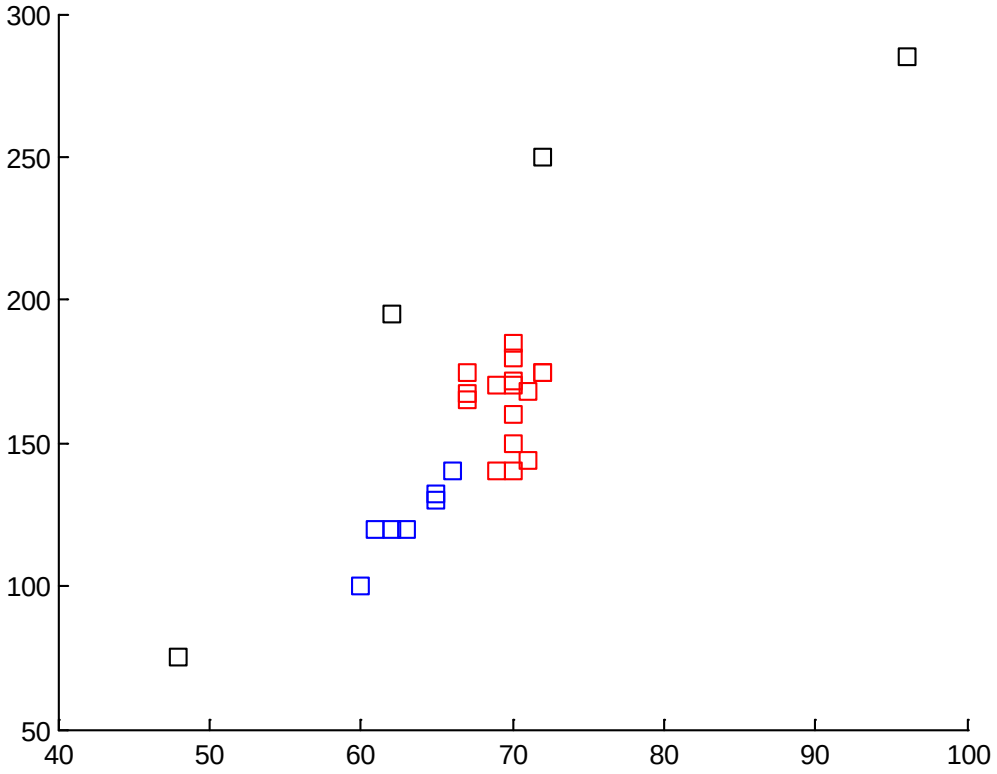
- Interesting… (from the 2D data used with k-Means)
  - Since GMM does "soft clustering" we seem to have identified the outliers as one cluster
  - Kinda cool





Multivariate GMM

- With 3 clusters ….



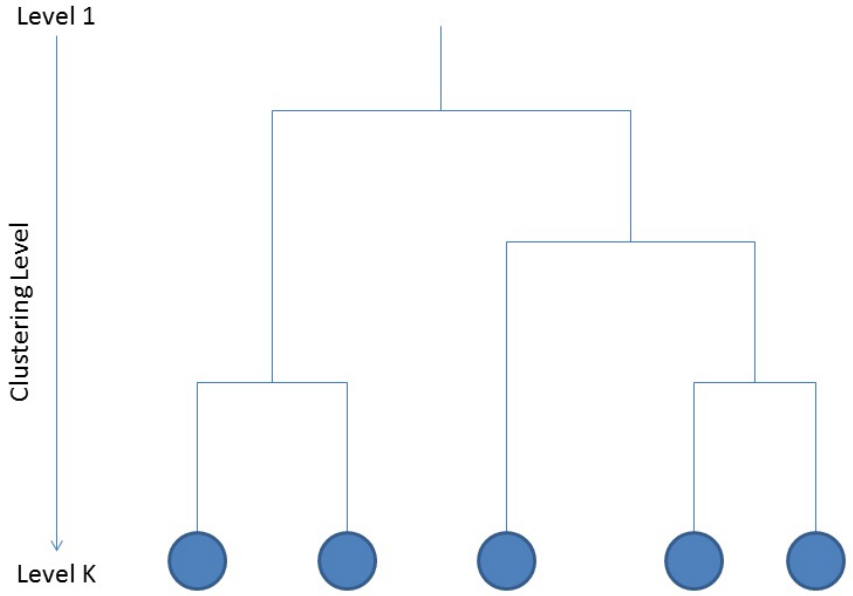
EM Summary

- Mixture Models
  - Advantages
    - If the assumed data distribution is correct it works well
  - Disadvantages
    - If the assumed data distribution is wrong, results can be quite bad
      - In particular type of distribution and number of components
- Expectation-Maximization is a general algorithm for parameter estimation
  - Like gradient descent
  - Remember k-means is actually just a version of EM

Hierarchical Clustering

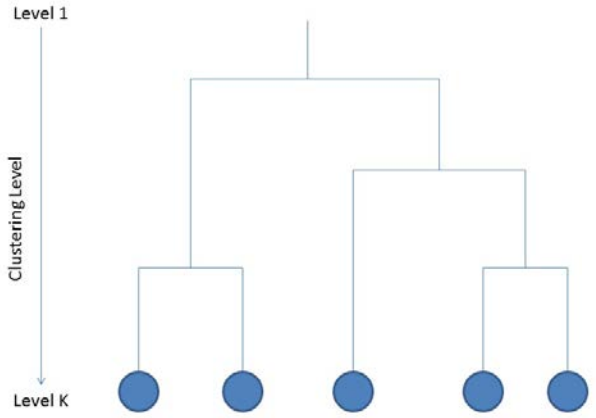
Hierarchical Agglomerative Clustering (HAC)

- With hierarchical agglomerative clustering we're building a clustering binary tree
- This can be done either bottom up, or top down
- At each iteration we have a new set of clusters



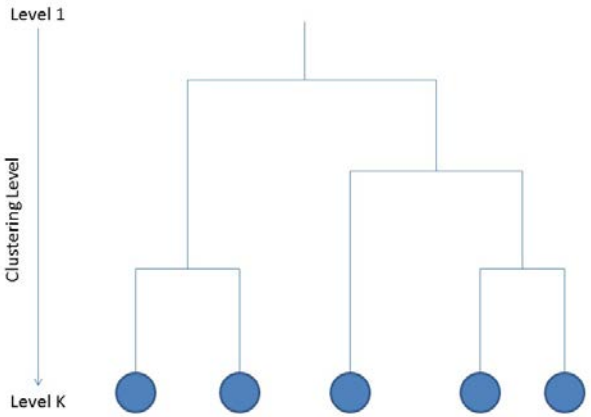
Hierarchical Agglomerative Clustering (HAC)

- Top-Down Approach
  - At first everything is part of a single cluster.
  - Then split this into two clusters based on some criterial
  - Now choose one of these two clusters, and split it into two
    - Now we have three total clusters
  - Etc.. until each cluster only has one observation in it (called a singleton)



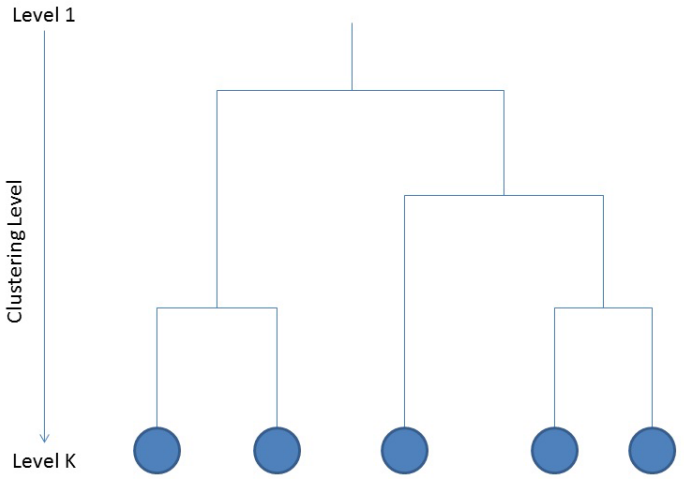
Hierarchical Agglomerative Clustering (HAC)

- Bottom-Up Approach
  - At first everything is its own cluster
    - If there's N observations, then there's N clusters
  - Choose two of these clusters to merge
    - Now there's $N - 1$ clusters
  - From these $N - 1$ clusters, choose two to merge
    - Now there's $N - 2$ clusters
  - Etc.. until there is only one cluster



Hierarchical Agglomerative Clustering (HAC)

- A top-down approach requires us to determine which cluster to split
  - And where to split it!
- A bottoms-up approach requires us to determine which clusters to merge
- A bottoms-up approach is more straight-forward so we'll focus on it



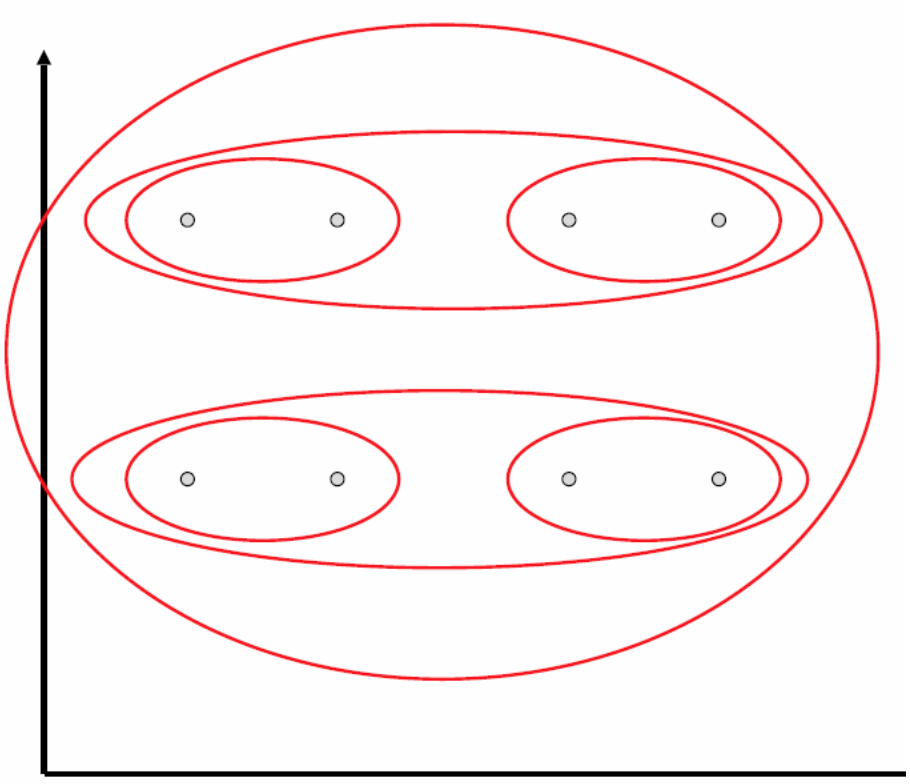
Closest Pair of Clusters

- When deciding which to clusters to merge we typically need to determine which two clusters are closest.
- There's many variants to defining closest pair of clusters
  - Single link – Similarity of the most similar
  - Complete link – Similarity of the furthest points
  - Average link – Average pair-wise similarity between clusters

Single Link Example

Single link – Similarity of the most similar

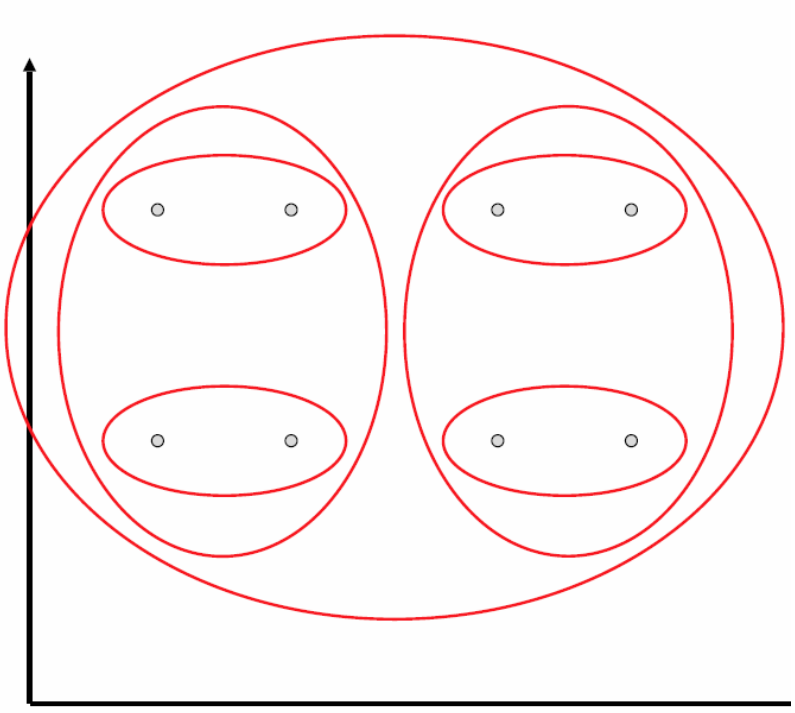
$$sim(C_i, C_j) = \max_{x \in C_i, y \in C_j} sim(x, y)$$



Complete Link HAC

Complete link – Similarity of the furthest points

$$sim(C_i, C_j) = \min_{x \in C_i, y \in C_j} sim(x, y)$$



Average HAC

- Average Link
  - Compromise between single and complete link
  - Average over all pairs *between* the two original clusters

$$sim(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} sim(x, y)$$

Choosing The Clustering Level

- Ok so we have a HAC tree.
- What can we use it for?
- If we know how many clusters we want (if the problem dictates k), then we just choose the level of the tree that has that many clusters.
- What if we don't know it a-priori?
- There's several ways, to attempt to find this
  - We'll just look at two basic graph-based ones

Clustering Quality

- First we need to provide some way to measure the quality of a given clustering
- A good clustering will produce high quality clusters in which
  - The intra-class (that is, intra-cluster) similarity is high
  - The inter-class similarity is low
- We will look at the graph of the weighted average intra-cluster distance in an attempt to determine the clustering level

Intra-Cluster Distance

- Intra-Cluster Distance
  - For a given cluster i and a chosen distance/similarity function d, the follow computes the pairwise intra-cluster distance G_i

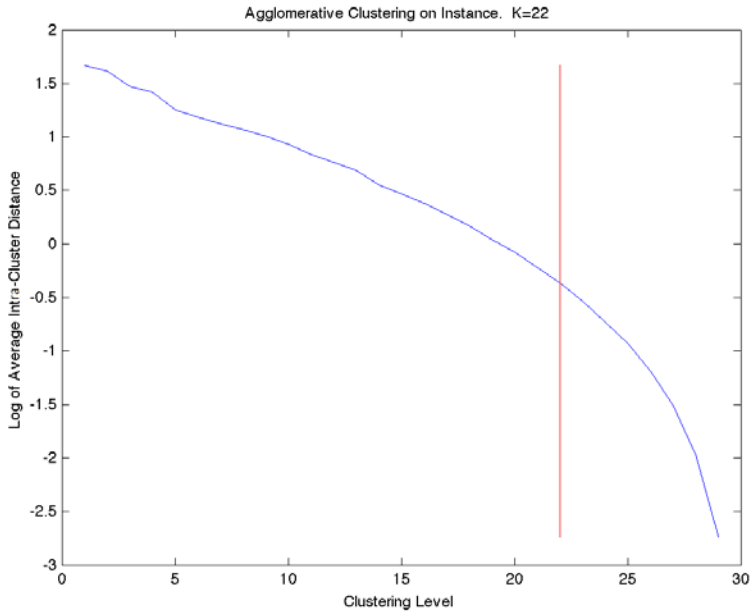
$$G_i = \sum_{x,y \in C_i} d(x, y)$$

  - The weighted average intra-cluster distance for clustering level j is then:

$$W_j = \sum_{i=1}^{j} \frac{G_i}{2|C_i|}$$

Graph Based Approaches

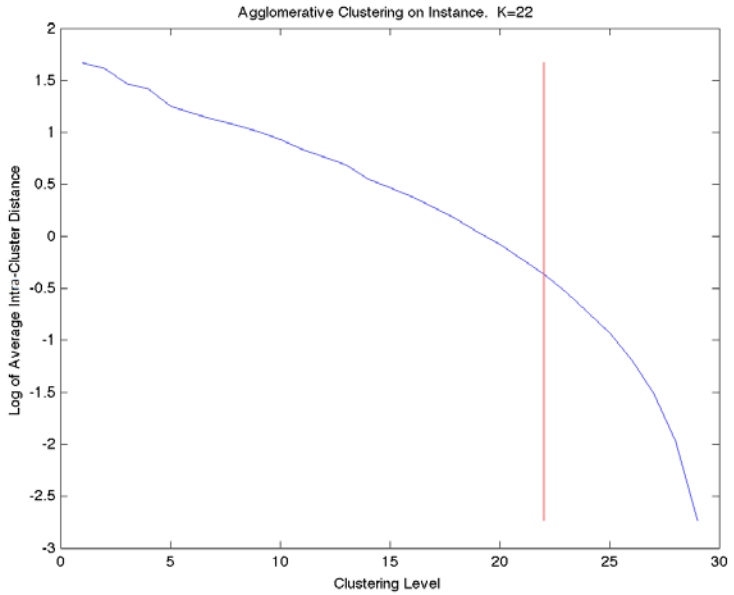
- Below is a graph showing the log of the average intra-cluster distance
- There are 30 observations
  - Clustering level 1 has everything in one cluster, and thus a large intra-cluster distance
  - Cluster level 30 has everything as its own cluster, and thus a small (infinitively negative) average intra-cluster distance



Graph Based Approaches

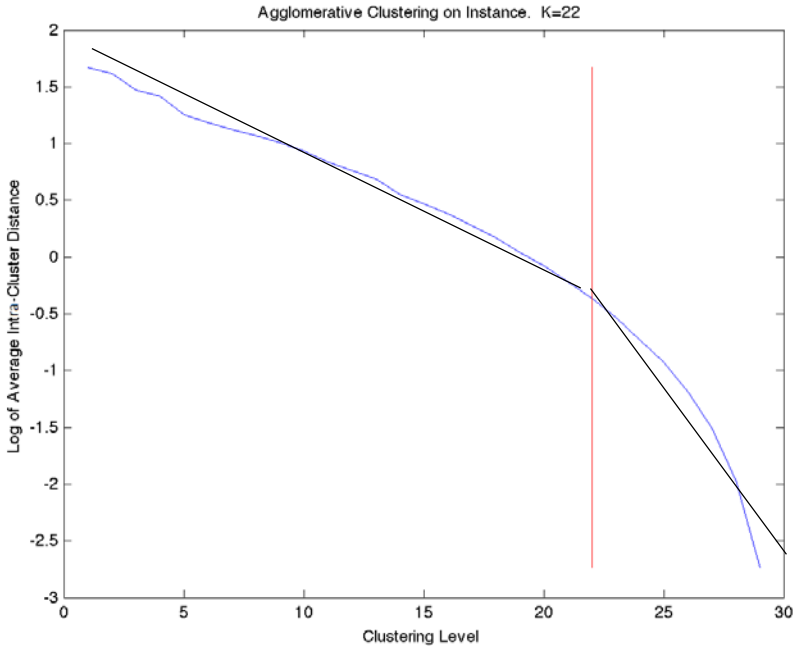
- Based on this graph we can attempt to find k by choosing the place where the graph becomes "less smooth"
- The Jump Method
  - Here we look for the largest discontinuity in the graph

$$k = \underset{j}{\operatorname{argmax}}\left(W_j - W_{j+1}\right)$$



Graph Based Approaches

- The L-Method
  - Here we attempt to find the "knee" in the graph.
  - This is done by fitting two lines that intersect at k' and finding the choice of $k = k$' that minimizes the reconstruction error



External Criteria for Clustering Quality

- Of course in the end we probably want to figure out how our algorithm is behaving.
- Maybe we can ask some people “after the fact” to do the clustering task and compare theirs to ours.
- Still unsupervised since the labels didn’t influence how we clustered. We just used it for evaluation

External Evaluation of Cluster Quality

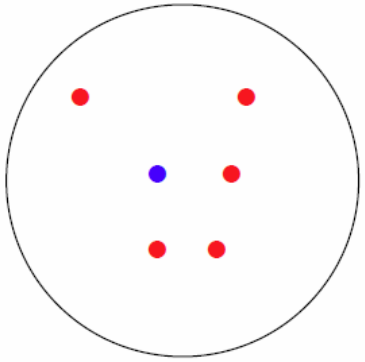
- Simple measure: **purity**
  - Let N_{ij} be the number of instances of (supervised) label *j* within cluster C_i
  - The purity of cluster C_i is then defined as

$$Purity(C_i) = \frac{1}{|C_i|} \max_j N_{ij}$$

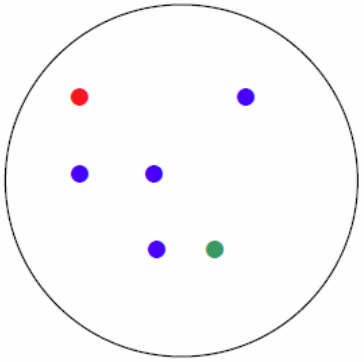
  - Then we can define the average purity of this clustering as

$$Purity = \frac{1}{k} \sum_{i=1}^{k} Purity(C_i)$$

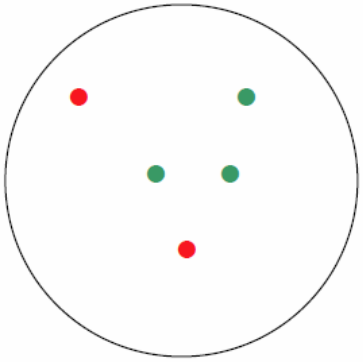
Purity Example



Cluster I



Cluster II



Cluster III

Cluster I: Purity = 1/6 (max(5, 1, 0)) = 5/6

Cluster II: Purity = 1/6 (max(1, 4, 1)) = 4/6

Cluster III: Purity = 1/5 (max(2, 0, 3)) = 3/5

External Evaluation of Cluster Quality

- Purity is biased because having $k = N$ clusters maximizes purity
  - But it can be useful in compare methods with the same clustering level
- Other measurements include
  - Entropy of a cluster using the supervised labels.
  - Mutual information between supervised labels and clusters
  - Rand Index