Clustering 聚类

Lectured by Shangsong Liang 梁上松
School of Computer Science, 计算机学院
Sun Yat-sen University

Outline

- Prototype-based
  - Fuzzy c-means
  - Mixture Model Clustering
  - Self-Organizing Maps
- Density-based
  - Grid-based clustering
  - Subspace clustering
- Graph-based
  - Chameleon
  - Jarvis-Patrick
  - Shared Nearest Neighbor (SNN)
- Characteristics of Clustering Algorithms

Hard (Crisp) vs Soft (Fuzzy) Clustering

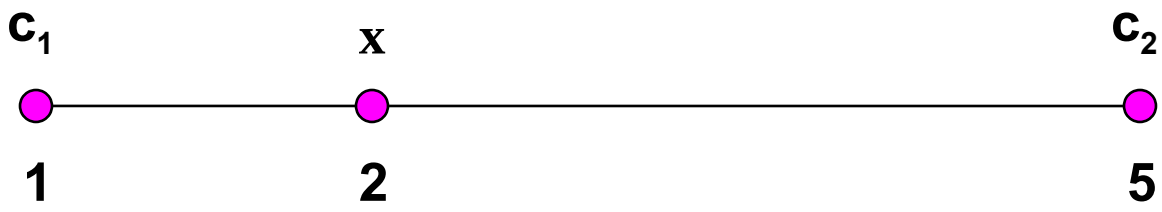
- Hard (Crisp) vs. Soft (Fuzzy) clustering
  - For soft clustering allow point to belong to more than one cluster
  - For K-means, generalize objective function

$$SSE = \sum_{j=1}^{k} \sum_{i=1}^{m} w_{ij}\, dist\left(\boldsymbol{x}_i, \boldsymbol{c}_j\right)^2 \qquad \sum_{j=1}^{k} w_{ij} = 1$$

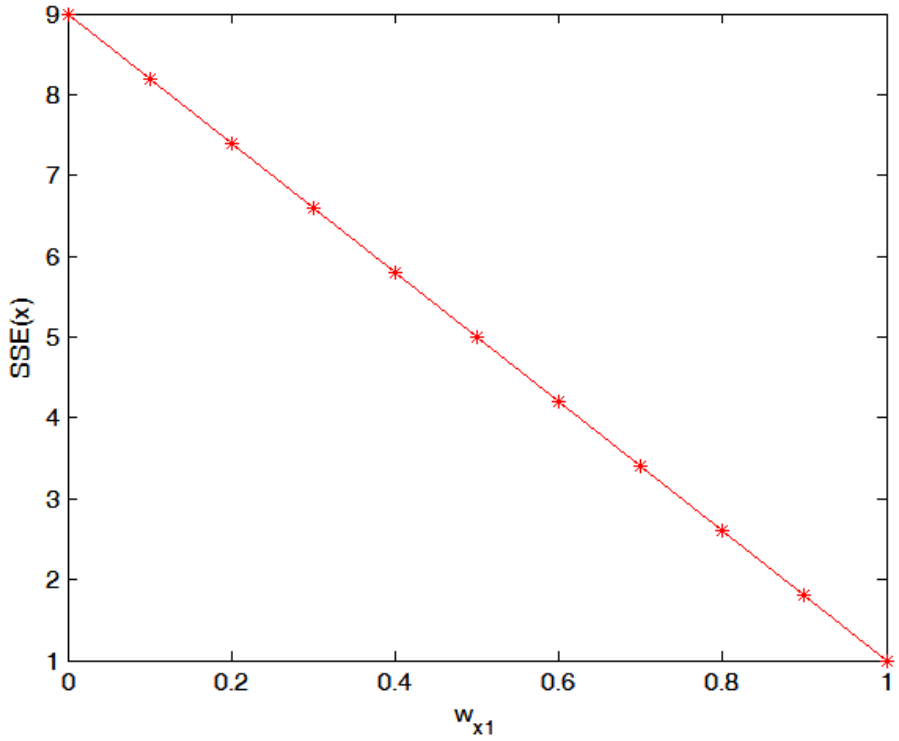
    : weight with which object x_i belongs to cluster

  - To minimize SSE (Error of Sum of Squares), repeat the following steps:
    - Fix and determine *w*(cluster assignment)
    - Fix*w* and recompute
  - Hard clustering:$w \in \{0,1\}$

Soft (Fuzzy) Clustering: Estimating Weights



$$SSE(x) = w_{x1}(2-1)^2 + w_{x2}(5-2)^2$$
$$= w_{x1} + 9w_{x2}$$



SSE(x) is minimized when $w_{x1} = 1$, $w_{x2} = 0$

Fuzzy C-means

- Objective function

p: fuzzifier (p > 1)

$$SSE = \sum_{j=1}^{k} \sum_{i=1}^{m} w_{ij}^{p} \, dist\left(\boldsymbol{x}_i, \boldsymbol{c}_j\right)^2 \qquad \sum_{j=1}^{k} w_{ij} = 1$$

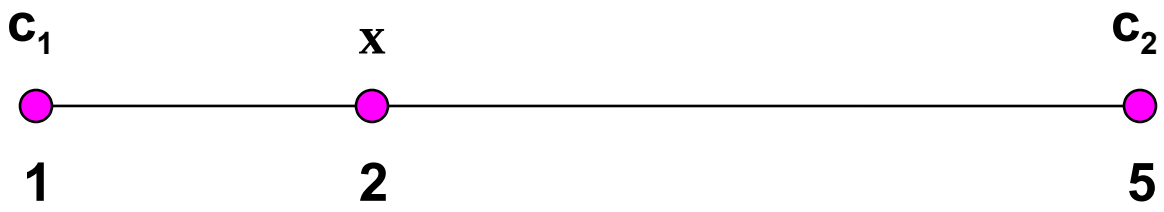
  - : weight with which object belongs to cluster
  - a power for the weight not a superscript and controls how “fuzzy” the clustering is

  - To minimize objective function, repeat the following:
    - Fix and determine*w*
    - Fix*w*and recompute

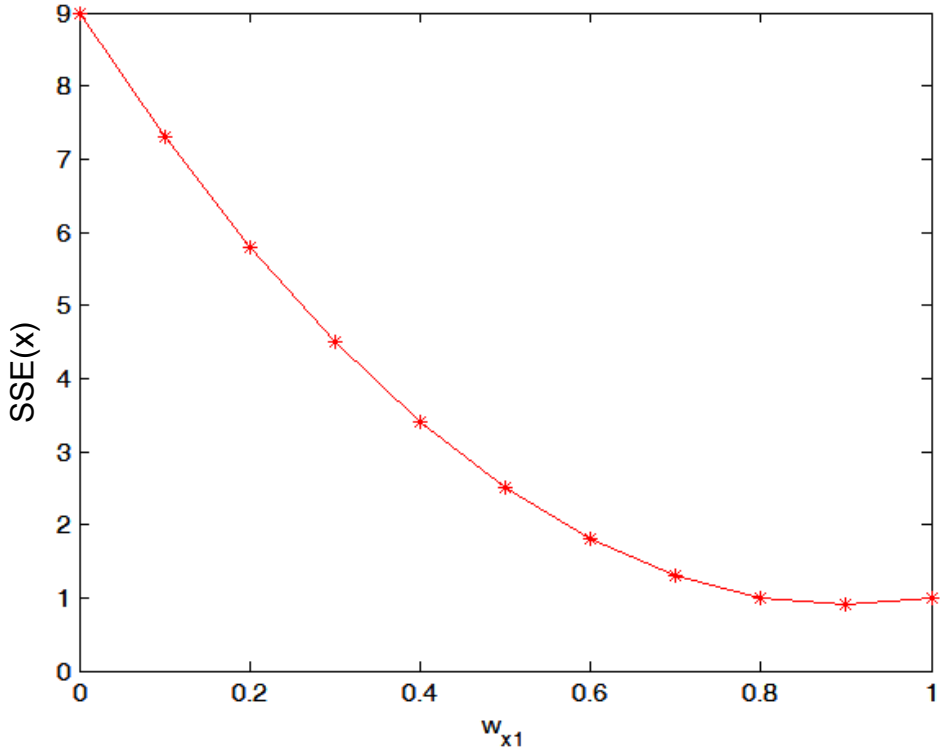
  - Fuzzy c-means clustering:*w*∈[0,1]

Bezdek, James C. *Pattern recognition with fuzzy objective function algorithms*. Kluwer Academic Publishers, 1981.

Fuzzy C-means



$$SSE(x) = w_{x1}^2(2-1)^2 + w_{x2}^2(5-2)^2$$
$$= w_{x1}^2 + 9w_{x2}^2$$



SSE(x) is minimized when w_{x1} = 0.9, w_{x2} = 0.1

Fuzzy C-means

- Objective function:

p: fuzzifier (p > 1)

$$SSE = \sum_{j=1}^{k} \sum_{i=1}^{m} w_{ij}^{p} \, dist\left(\boldsymbol{x}_i, \boldsymbol{c}_j\right)^2 \qquad \sum_{j=1}^{k} w_{ij} = 1$$

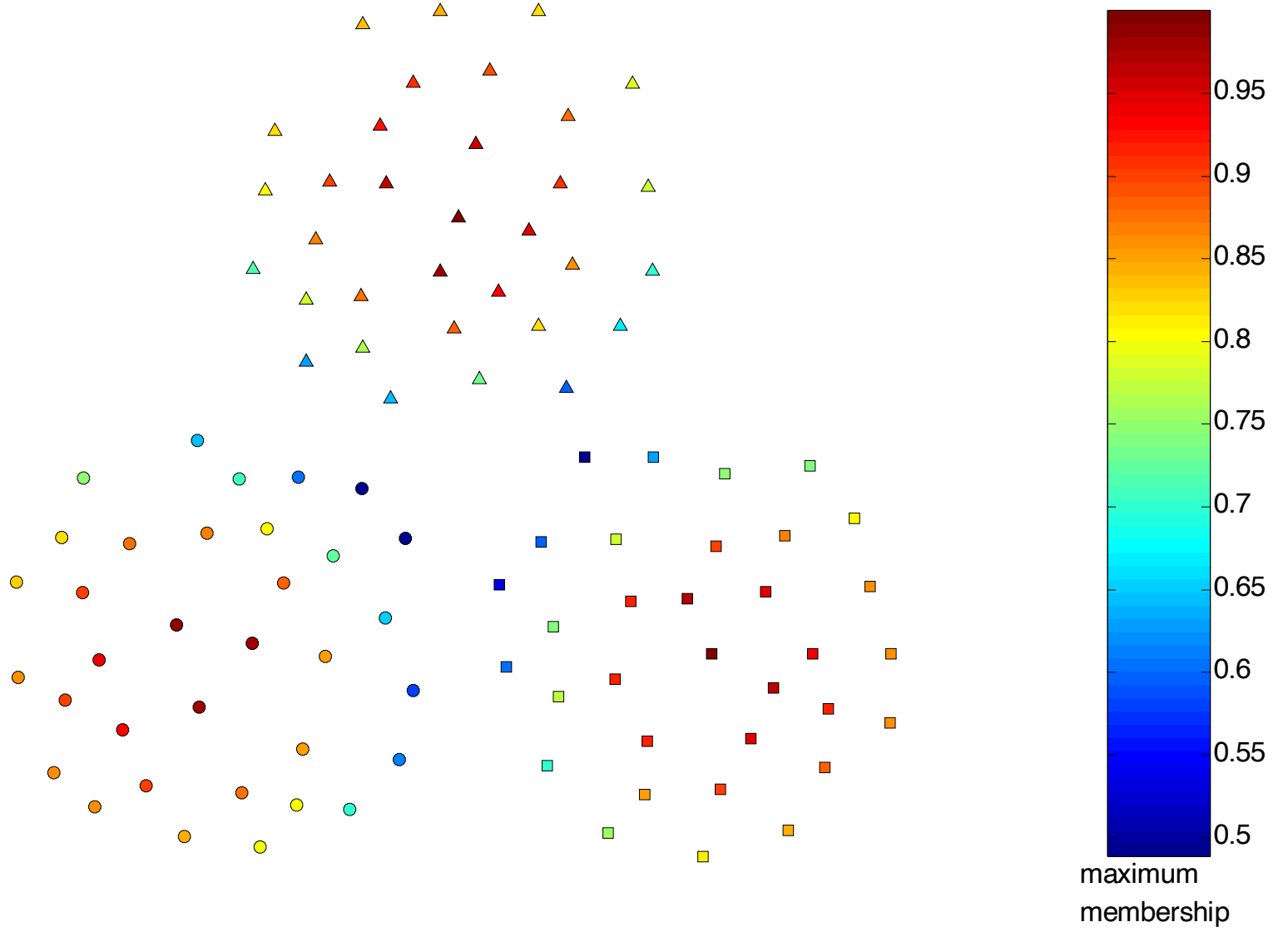
- Initialization: choose the weights w_{ij} randomly
- Repeat:
  - Update centroids:

$$\boldsymbol{c}_j = \sum_{i=1}^{m} w_{ij} \boldsymbol{x}_i / ¿ \sum_{i=1}^{m} w_{ij} ¿$$

  - Update weights:

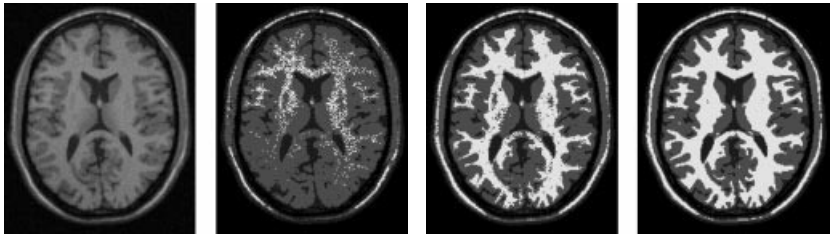
$$w_{ij} = \left(1/dist\left(\boldsymbol{x}_i, \boldsymbol{c}_j\right) ¿¿ 2\right)^{\frac{1}{p-1}} / \sum_{j=1}^{k} \left(1/dist\left(\boldsymbol{x}_i, \boldsymbol{c}_j\right) ¿¿ 2\right)^{\frac{1}{p-1}} ¿¿$$

Fuzzy K-means Applied to Sample Data



An Example Application: Image Segmentation

- Modified versions of fuzzy c-means have been used for image segmentation
  - Especially fMRI images (functional magnetic resonance images)
- References
  - Gong, Maoguo, Yan Liang, Jiao Shi, Wenping Ma, and Jingjing Ma. "Fuzzy c-means clustering with local information and kernel metric for image segmentation." *Image Processing, IEEE Transactions on* 22, no. 2 (2013): 573-584.



From left to right: original images, fuzzy c-means, EM, BCFCM

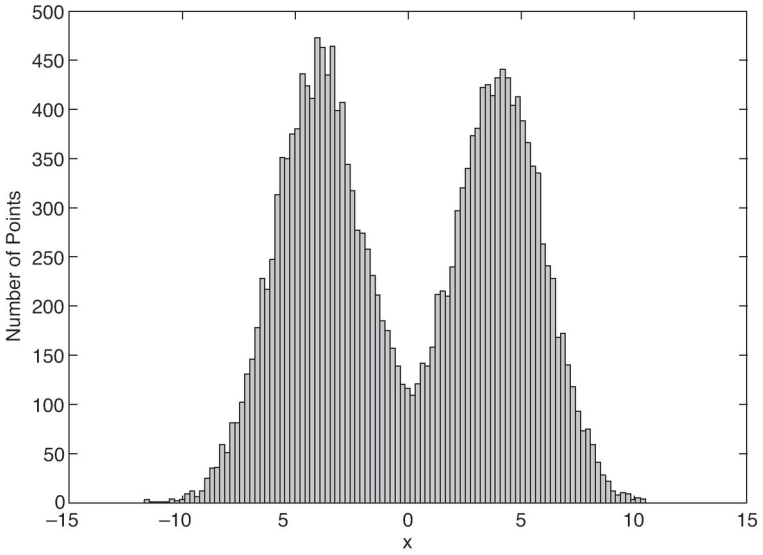
  - Ahmed, Mohamed N., Sameh M. Yamany, Nevin Mohamed, Aly A. Farag, and Thomas Moriarty. "A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data." *Medical Imaging, IEEE Transactions on* 21, no. 3 (2002): 193-199.

Hard (Crisp) vs Soft (Probabilistic) Clustering

- Idea is to model the set of data points as arising from a mixture of distributions
  - Typically, normal (Gaussian) distribution is used
  - But other distributions have been very profitably used

- Clusters are found by estimating the parameters of the statistical distributions
  - Can use a k-means like algorithm, called the Expectation-Maximization (EM) algorithm, to estimate these parameters
    - Actually, k-means is a special case of this approach
  - Provides a compact representation of clusters
  - The probabilities with which point belongs to each cluster provide a functionality similar to fuzzy clustering.

Probabilistic Clustering: Example



- Informal example: consider modeling the points that generate the following histogram.
- Looks like a combination of two normal (Gaussian) distributions
- Suppose we can estimate the mean and standard deviation of each normal distribution.
  - This completely describes the two clusters
  - We can compute the probabilities with which each point belongs to each cluster
  - Can assign each point to the cluster (distribution) for which it is most probable.

$$prob(x_i|\Theta) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Probabilistic Clustering: EM Algorithm

Initialize the parameters

Repeat

For each point, compute its probability under each distribution

Using these probabilities, update the parameters of each distribution

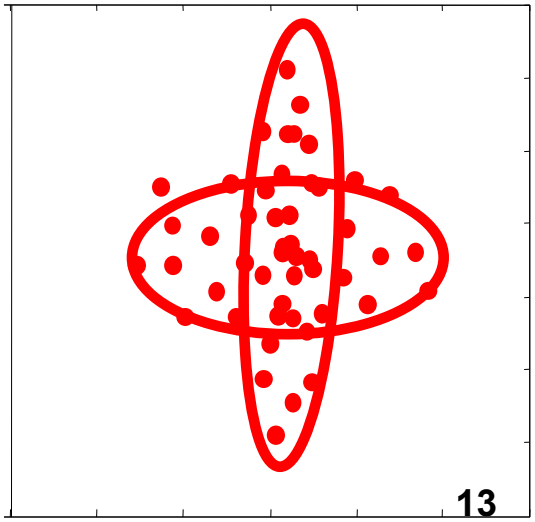
Until there is no change

- Very similar to K-means
- Consists of assignment and update steps
- Can use random initialization
  - Problem of local minima
- For normal distributions, typically use K-means to initialize
- If using normal distributions, can find elliptical as well as spherical shapes.

Mixtures of Gaussians

- K-means algorithm
  - Assigned each example to exactly one cluster
  - What if clusters are overlapping?
    - Hard to tell which cluster is right
    - Maybe we should try to remain uncertain
  - Used Euclidean distance
  - What if cluster has a non-circular shape?

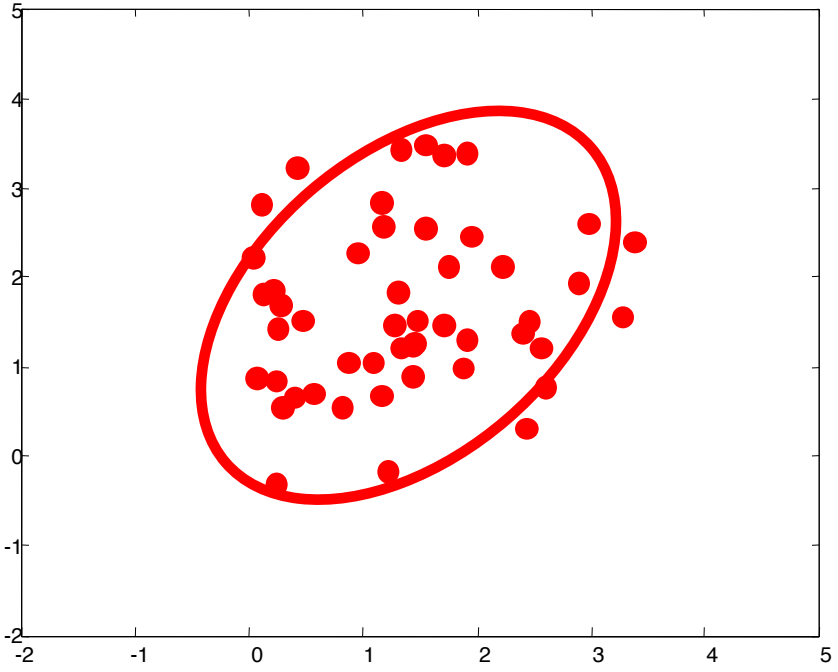
- Gaussian mixture models
  - Clusters modeled as Gaussians
    - Not just by their mean
  - EM algorithm: assign data to cluster with some *probability*



Multivariate Gaussian models

$$\mathcal{N}(\underline{x}\ ;\ \underline{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2} \exp\left\{ -\frac{1}{2} (\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu}) \right\}$$

$$\log p(\underline{X}) = \sum_i \log \left[\sum_c \pi_c \, \mathcal{N}(x_i\ ;\ \mu_c, \Sigma_c) \right]$$



Maximum Likelihood estimates

$$\hat{\mu} = \frac{1}{N} \sum_i x^{(i)}$$

$$\hat{\Sigma} = \frac{1}{N} \sum_i (x^{(i)} - \hat{\mu})^T (x^{(i)} - \hat{\mu})$$

We'll model each cluster using one of these Gaussian "bells"…

Multivariate Gaussian models

The goal is to maximize the following objective

$$
\begin{aligned}
p(X) &= \prod_i p(x_i) \\
&= \prod_i \sum_c p(x_i \mid c) p(c) \\
&= \prod_i \pi_c \sum_c p(x_i \mid c) \\
&= \prod_i \pi_c \mathcal{N}\left(x_i \mid \mu_c, \Sigma_c\right)
\end{aligned}
$$

Multivariate Gaussian models

Maximizing the above objective is equal to maximizing the following:

$$\log p(X) = \sum_i \log \pi_c \mathcal{N}\left(x_i \mid \mu_c, \Sigma_c\right)$$

Multivariate Gaussian models

If we have know all the parameters: means, the covariances, and the sizes. We can cluster the data as the following:

$$
\begin{aligned}
p(c \mid x_i) &= \frac{p(c) \cdot p(x_i \mid c)}{p(x_i)} \\
&= \frac{\pi_c \, p(x_i \mid \mu_c, \Sigma_c)}{\sum_j \pi_{c_j} \cdot p(x_i \mid \mu_{c_j}, \Sigma_{c_j})} \\
&= \frac{\pi_c \, \mathcal{N}(x_i; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i; \mu_{c'}, \Sigma_{c'})}
\end{aligned}
$$

Let $p(c \mid x_i)$ denotes as r_{ic}. Then, we can cluster x_i into the cluster c_{j*} by:

$$
c_{j*} = \arg\max_{j \in 1,2,\ldots,J} r_{ij}
$$

EM Algorithm: E-step

- Start with parameters describing each cluster
- Mean, Covariance, “size”

- E-step (“Expectation”)
  - For each datum (example) x_i,
  - Compute “r_{ic}”, the probability that it belongs to cluster c
    - Compute its probability under model c
    - Normalize to sum to one (over clusters c)

$$r_{ic} = \frac{\pi_c \mathcal{N}(x_i \; ; \; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i \; ; \; \mu_{c'}, \Sigma_{c'})}$$

  - If x_I is very likely under the c^{th} Gaussian, it gets high weight
  - Denominator just makes r’s sum to one

EM Algorithm: M-step

- Start with assignment probabilities r_{ic}
- Update parameters: means, Covariances, "sizes"

- M-step ("Maximization")
  - For each cluster (Gaussian) x_c,
  - Update its parameters using the (weighted) data points

$$N_c = \sum_i r_{ic}$$ **Total responsibility allocated to cluster c**

$$\pi_c = \frac{N_c}{N}$$ **Fraction of total assigned to cluster c**

$$\mu_c = \frac{1}{N_c} \sum_i r_{ic} x_i$$ **Weighted mean of assigned data**

$$\Sigma_c = \frac{1}{N_c} \sum_i r_{ic} (x_i - \mu_c)^T (x_i - \mu_c)$$ **Weighted covariance of assigned data (use new weighted means here)**

Expectation-Maximization

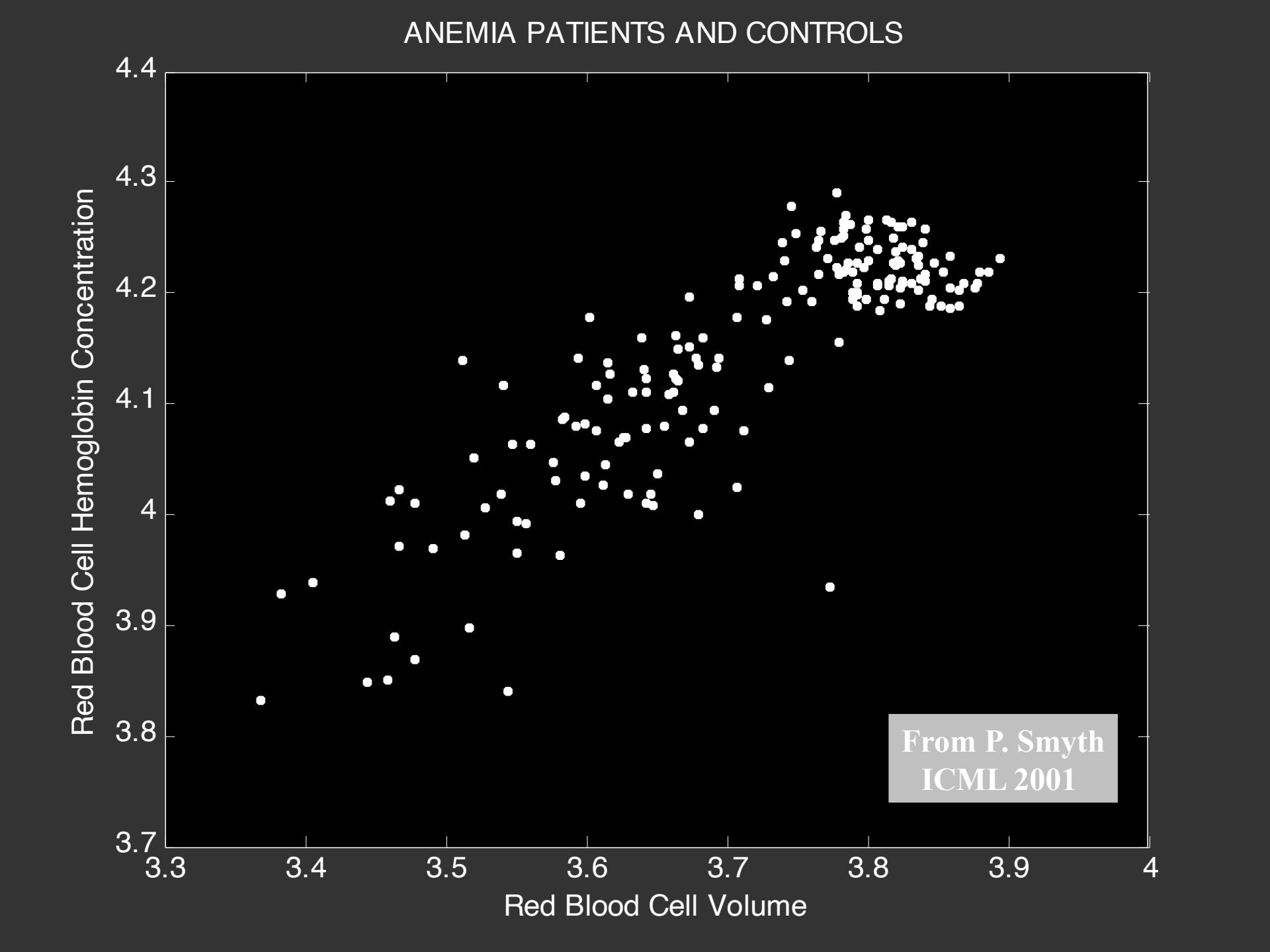
- Each step increases the log-likelihood of our model

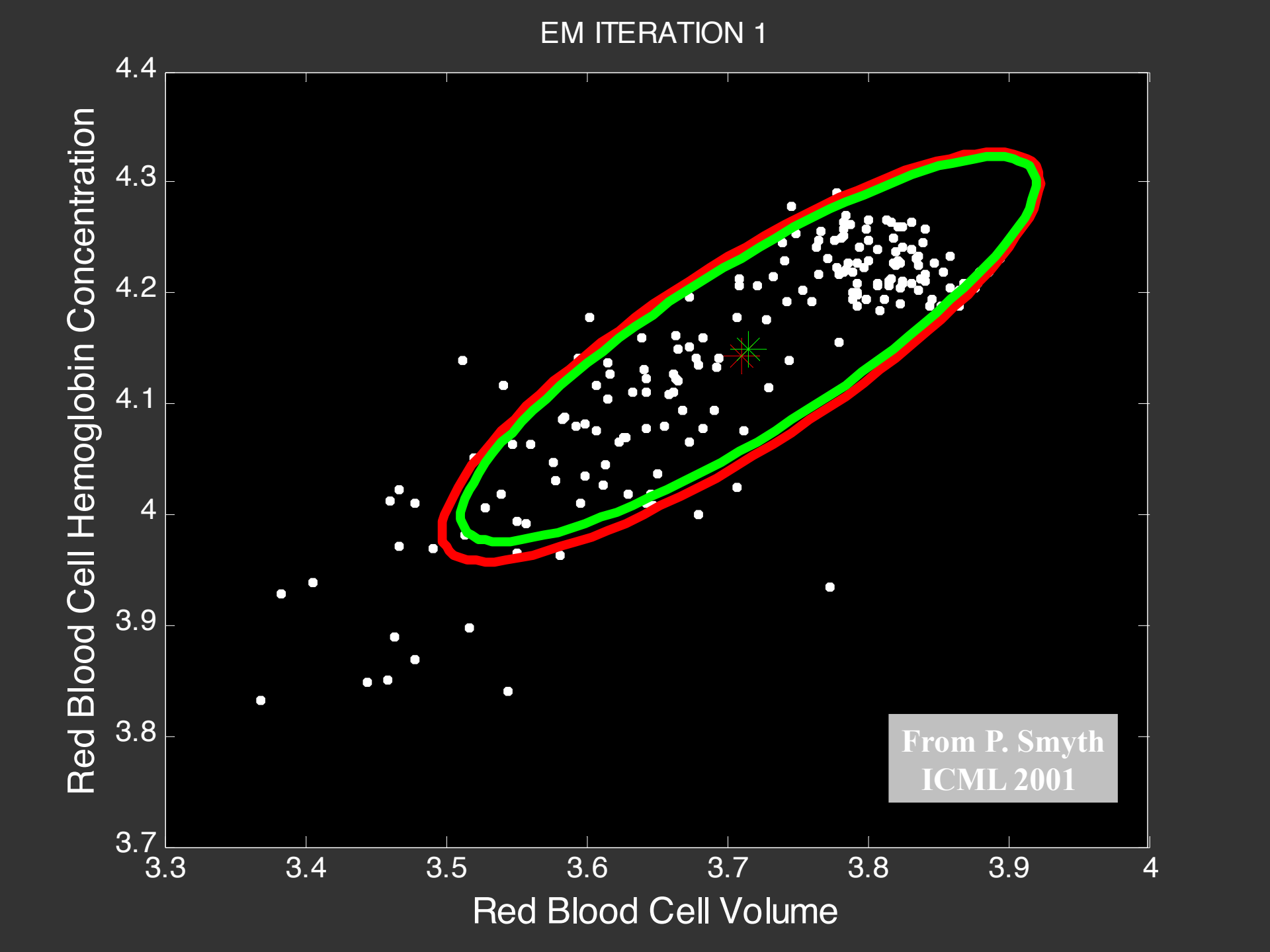
$$\log p(\underline{X}) = \sum_i \log \left[\sum_c \pi_c \, \mathcal{N}(x_i \; ; \; \mu_c, \Sigma_c) \right]$$

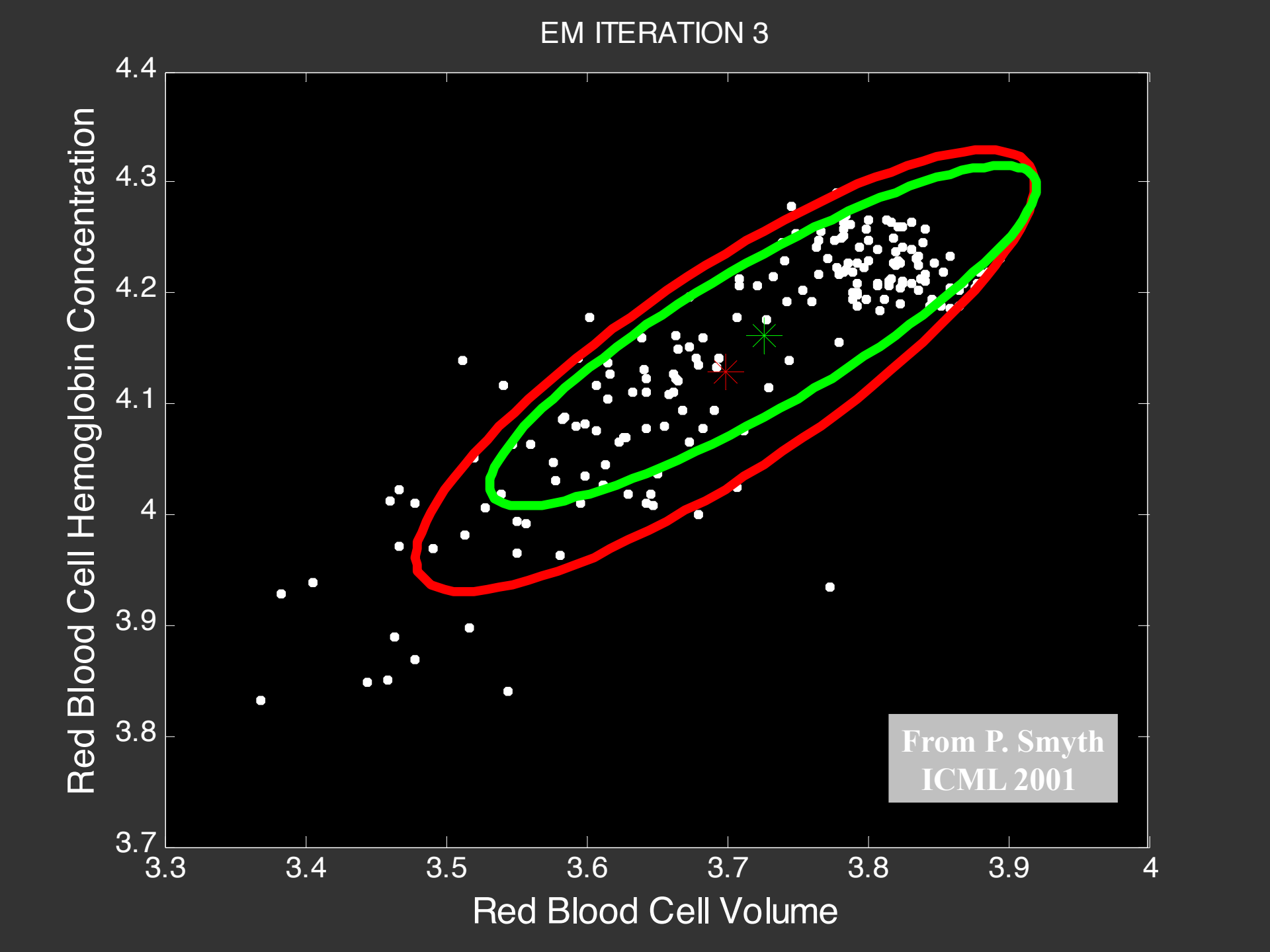
(we won't derive this, though)

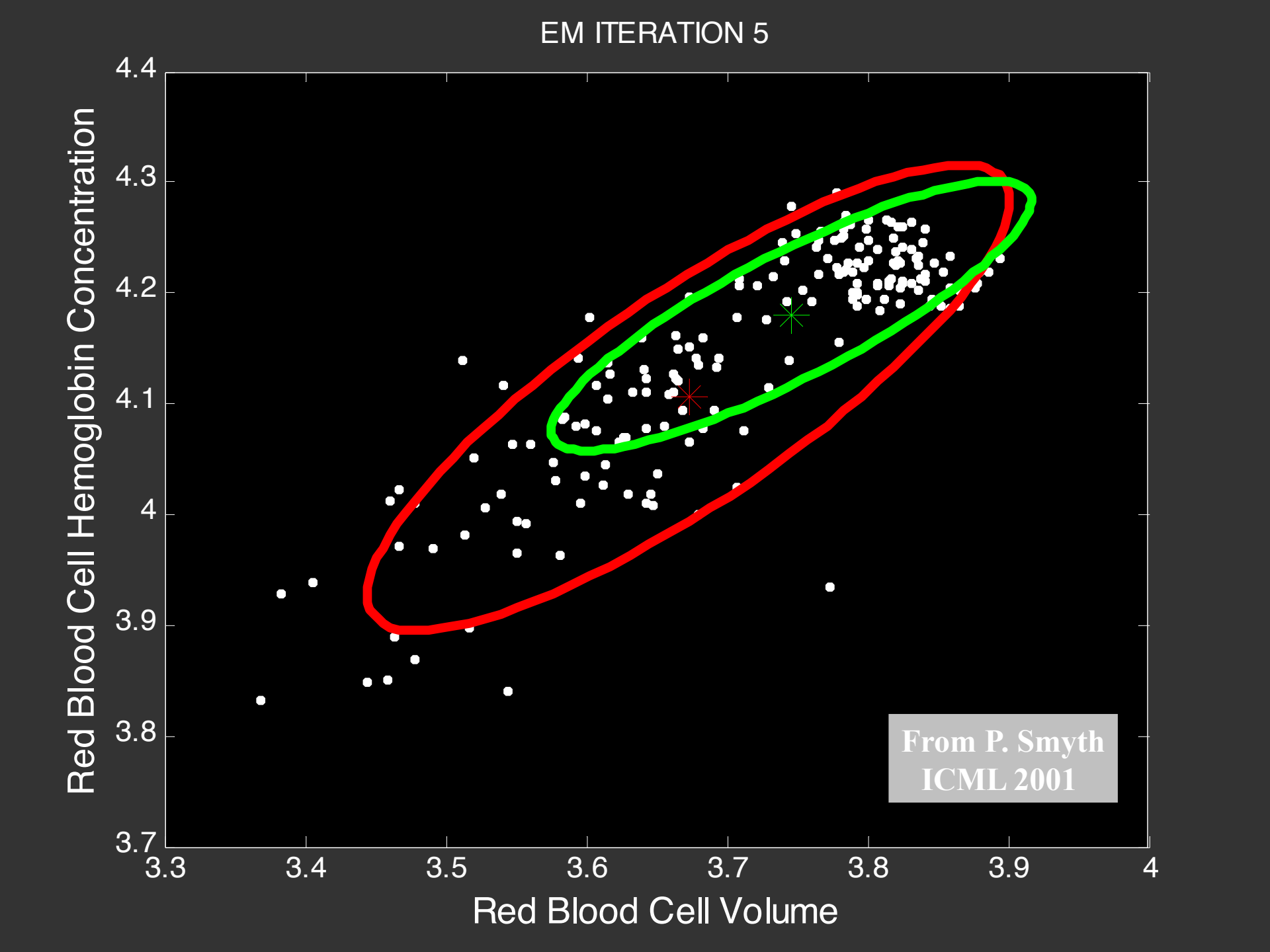
- Iterate until convergence
  - Convergence guaranteed – another ascent method

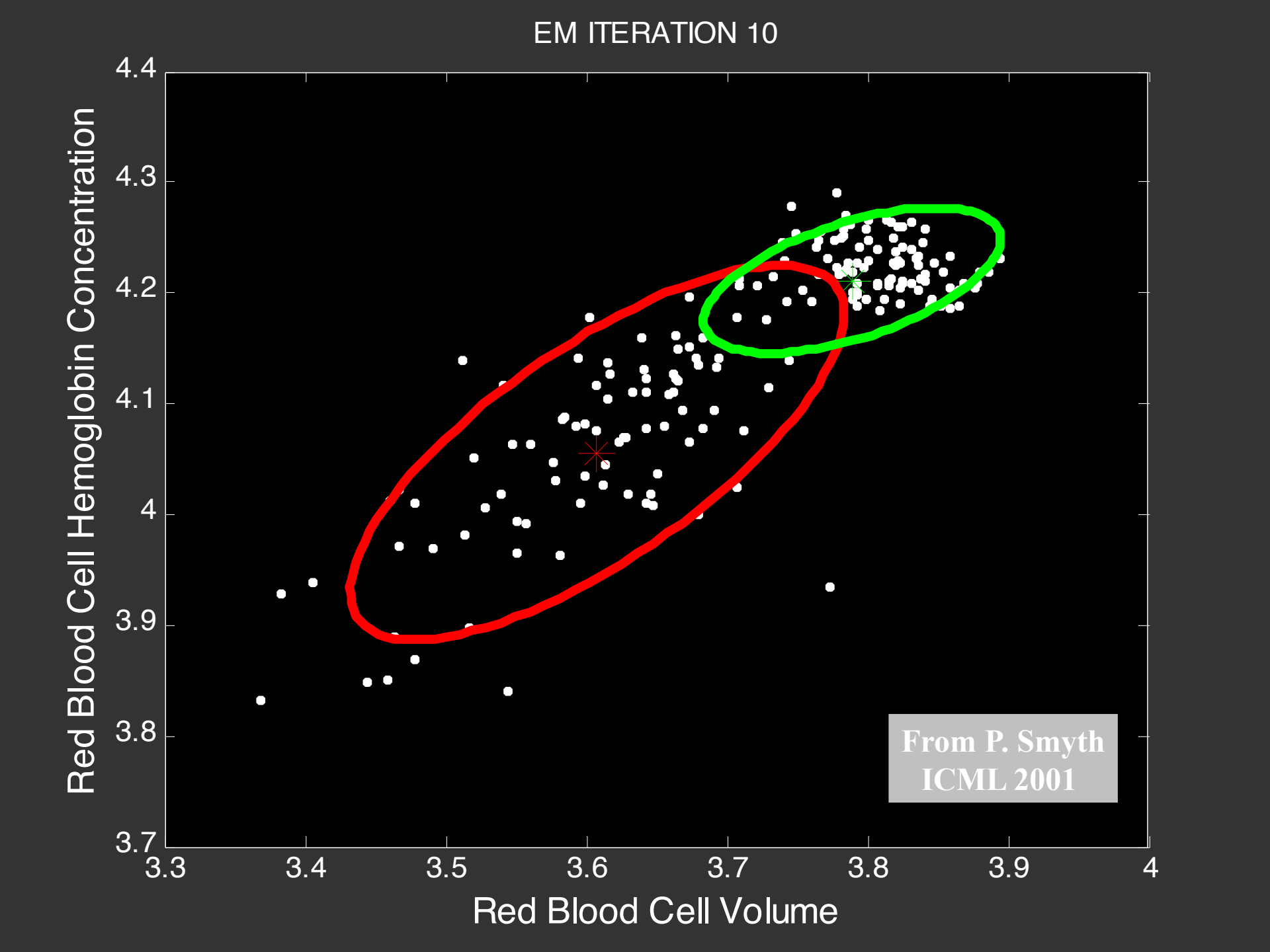
- What should we do
  - If we want to choose a single cluster for an "answer"?
  - With new data we didn't see during training?

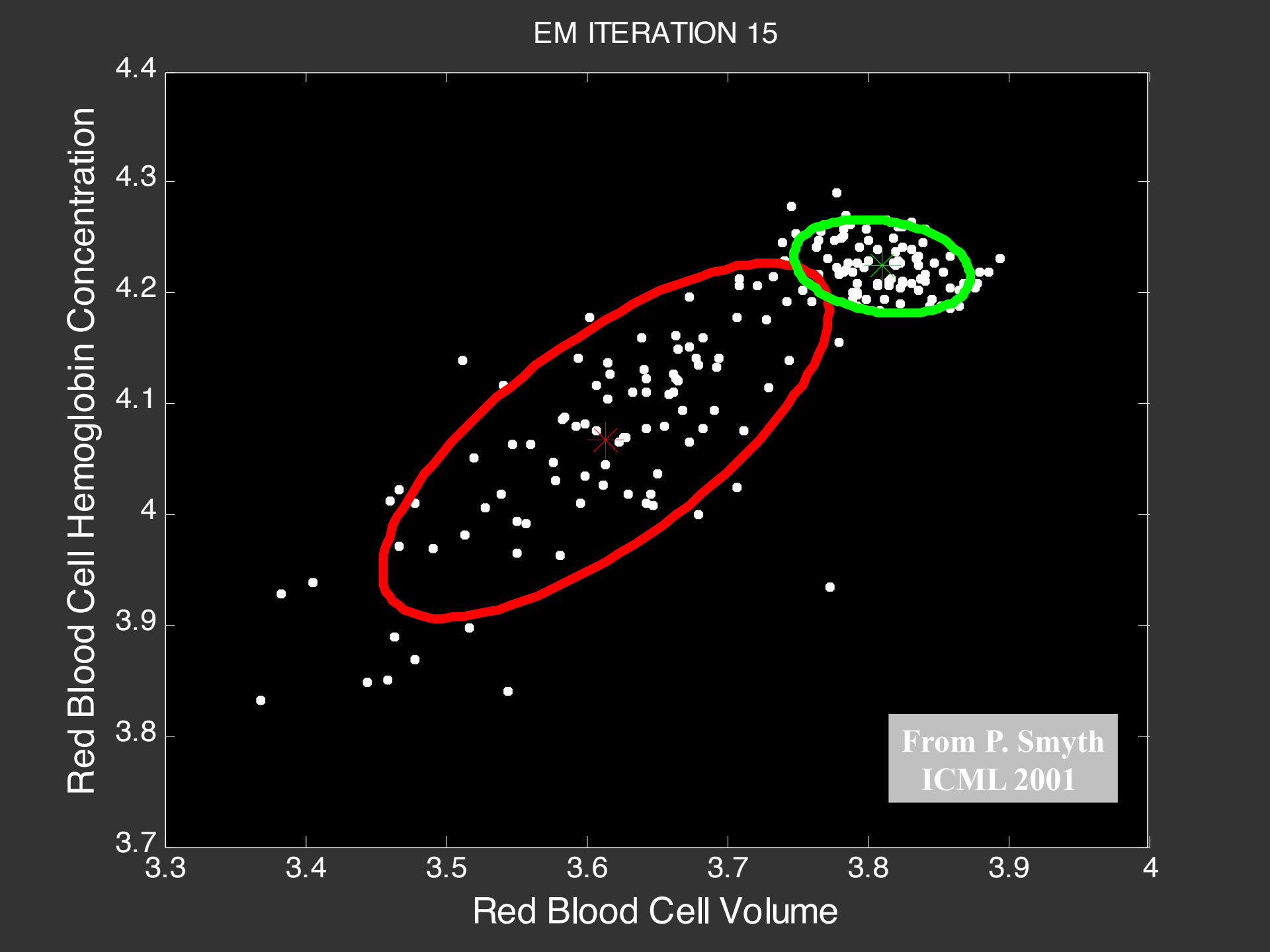

ANEMIA PATIENTS AND CONTROLS
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001

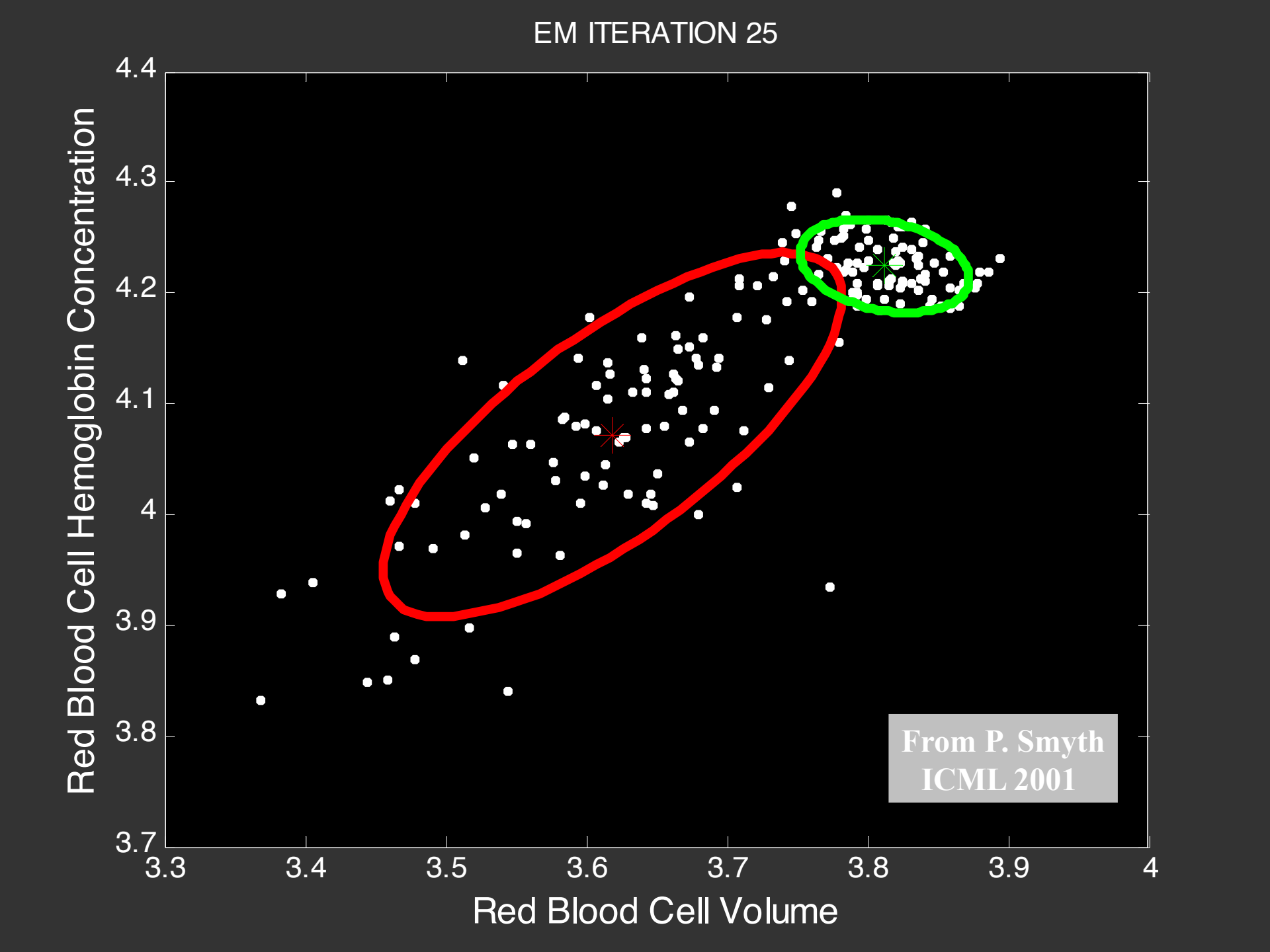

EM ITERATION 1
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001

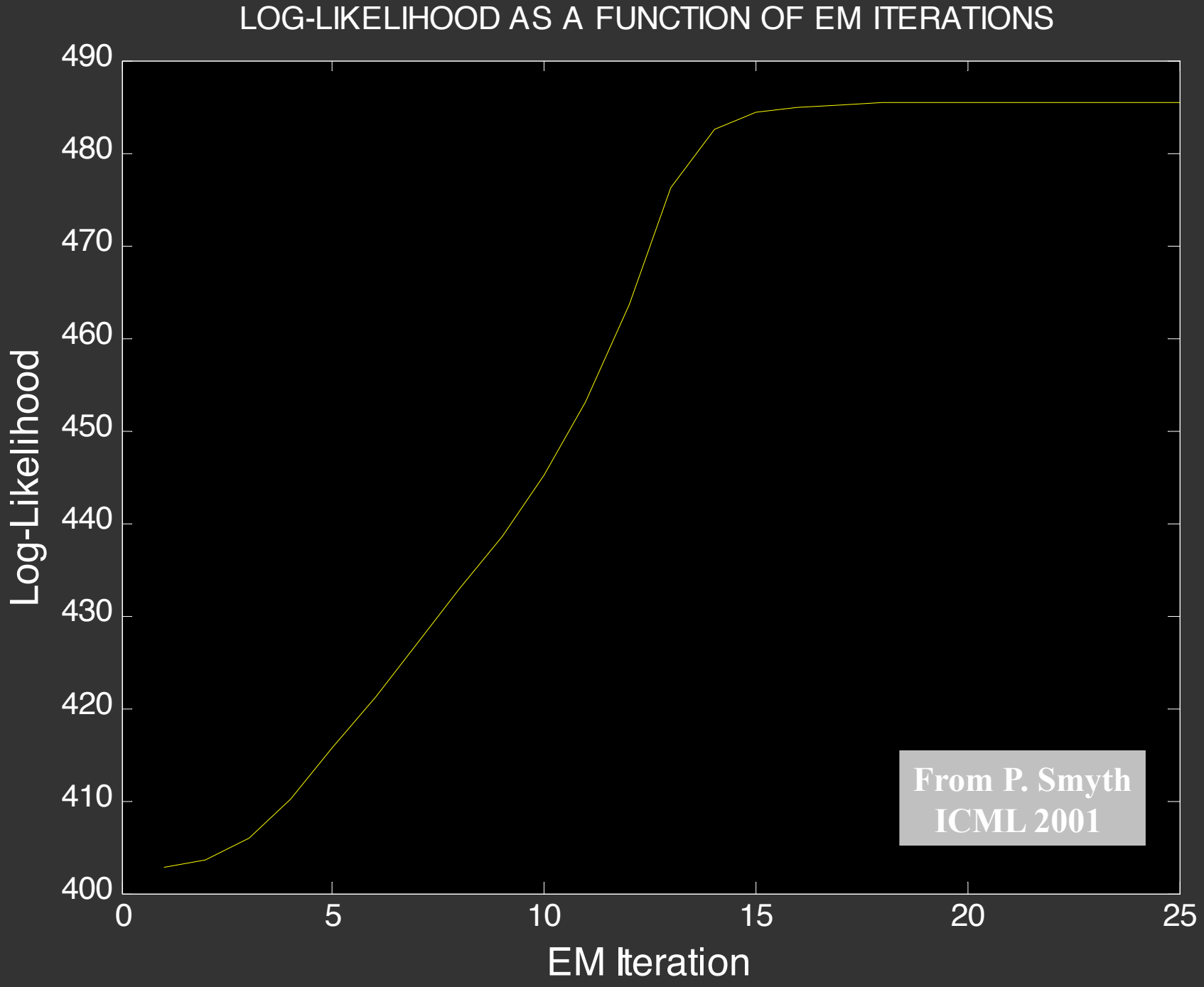

EM ITERATION 3
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 5
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 10
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 15
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 25
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001

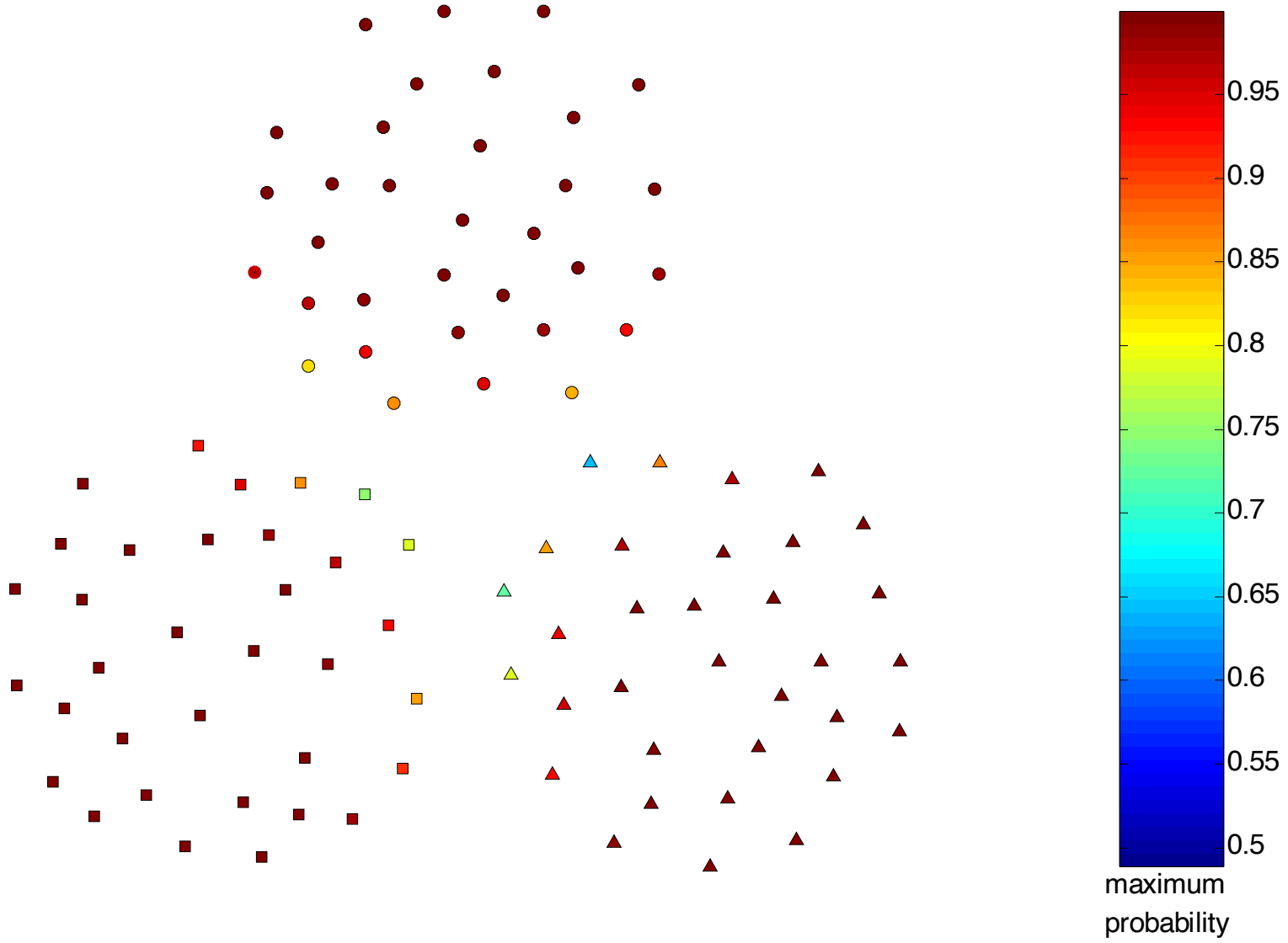

LOG-LIKELIHOOD AS A FUNCTION OF EM ITERATIONS
Log-Likelihood
EM Iteration
490
480
470
460
450
440
430
420
410
400
0
5
10
15
20
25
From P. Smyth
ICML 2001

More Detailed EM Algorithm

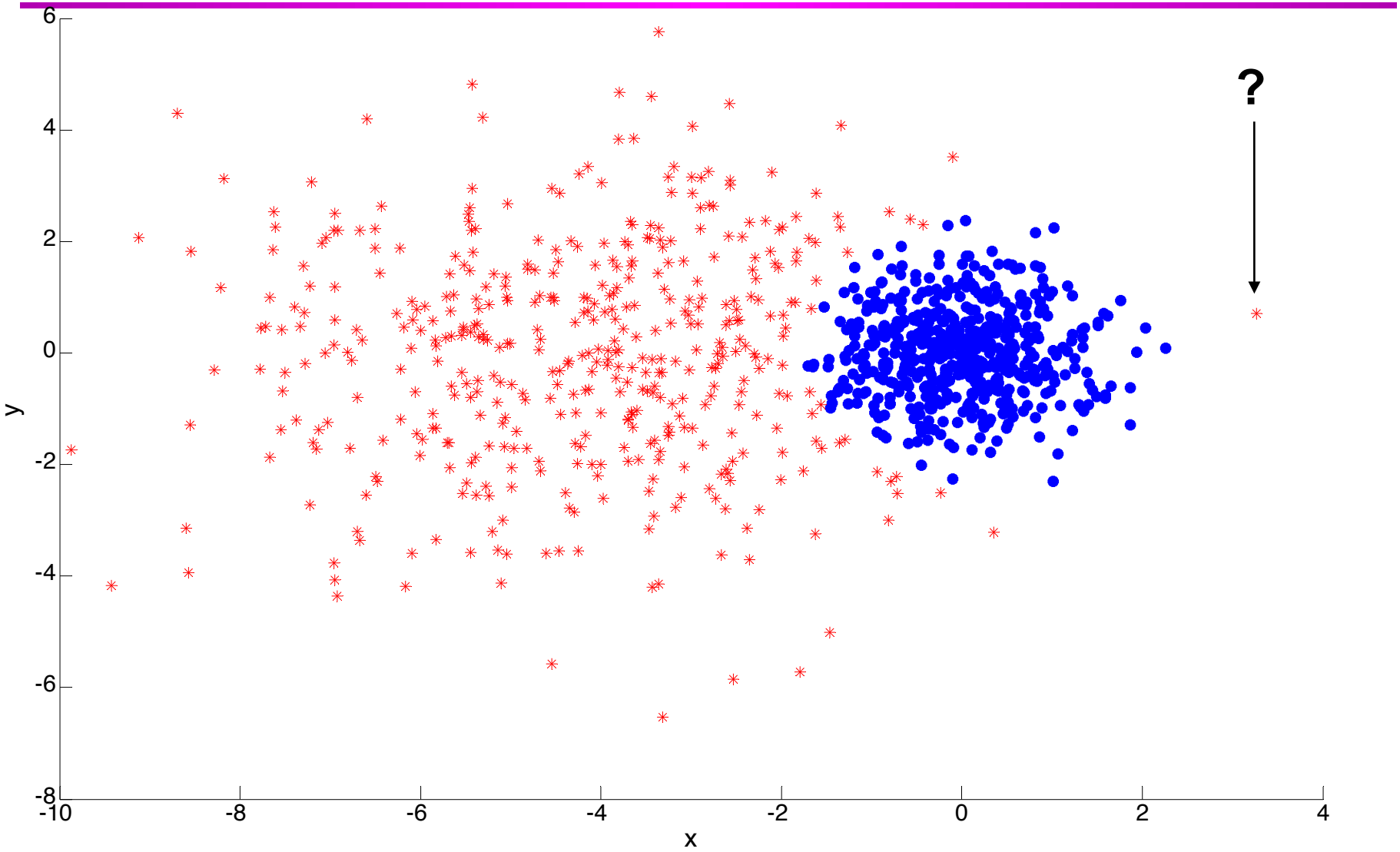
Algorithm 9.2 EM algorithm.

1: Select an initial set of model parameters.
(As with K-means, this can be done randomly or in a variety of ways.)
2: **repeat**
3: **Expectation Step** For each object, calculate the probability that each object belongs to each distribution, i.e., calculate $prob(distribution\ j|\mathbf{x}_i, \Theta)$.
4: **Maximization Step** Given the probabilities from the expectation step, find the new estimates of the parameters that maximize the expected likelihood.
5: **until** The parameters do not change.
(Alternatively, stop if the change in the parameters is below a specified threshold.)

Probabilistic Clustering Applied to Sample Data



Probabilistic Clustering: Dense and Sparse Clusters



Problems with EM

- Convergence can be slow
- Only guarantees finding local maxima
- Makes some significant statistical assumptions
- Number of parameters for Gaussian distribution grows as $O(d^2)$, d the number of dimensions
  - Parameters associated with covariance matrix
  - K-means only estimates cluster means, which grow as $O(d)$

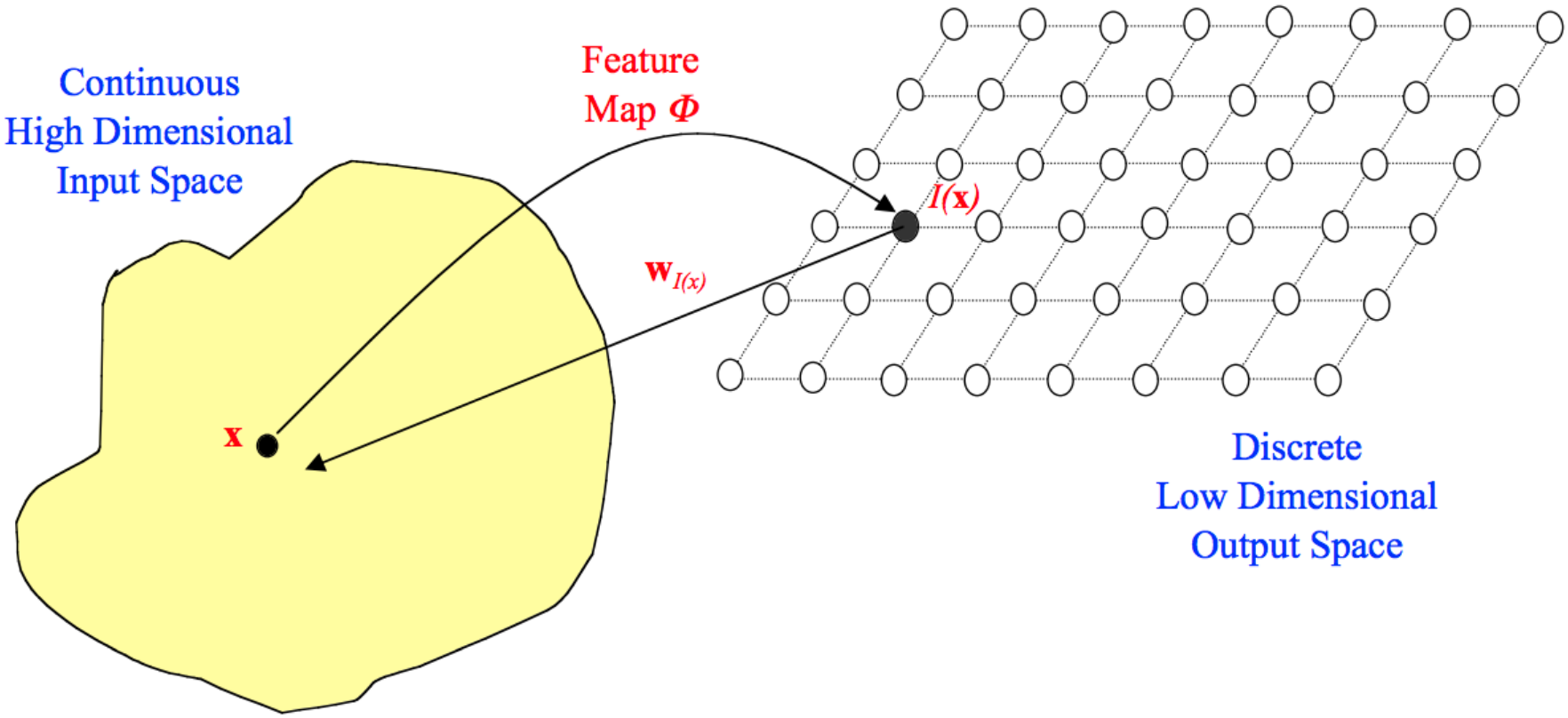
Alternatives to EM

- Method of moments / Spectral methods
  - ICML 2014 workshop bibliography

    https://sites.google.com/site/momentsicml2014/bibliography
- Markov chain Monte Carlo (MCMC)
- Other approaches

SOM: Self-Organizing Maps

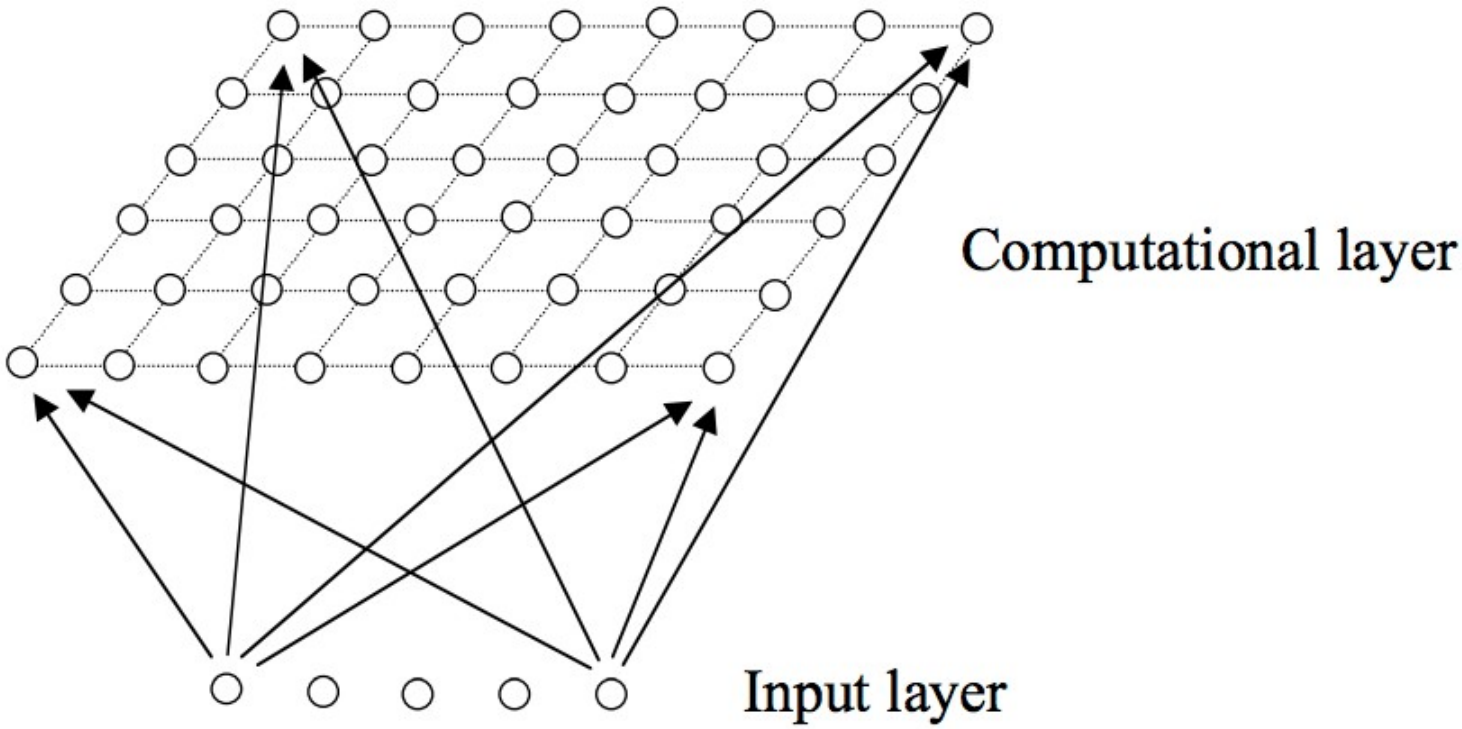
We have points **x** in the input space mapping to points $I(\mathbf{x})$ in the output space:



Each point I in the output space will map to a corresponding point $\mathbf{w}(I)$ in the input space.

SOM: Self-Organizing Maps

- A particular kind of SOM known as a Kohonen Network



SOM: Four major components

The self-organization process involves four major components:

Initialization: All the connection weights are initialized with small random values.

Competition: For each input pattern, the neurons compute their respective values of a *discriminant function* which provides the basis for competition. The particular neuron with the smallest value of the discriminant function is declared the winner.

SOM: Four major components

Cooperation: The winning neuron determines the spatial location of a topological neighbourhood of excited neurons, thereby providing the basis for cooperation among neighbouring neurons.

Adaptation: The excited neurons decrease their individual values of the discriminant function in relation to the input pattern through suitable adjustment of the associated connection weights, such that the response of the winning neuron to the subsequent application of a similar input pattern is enhanced.

SOM: Four major components

- The Competitive Process

If the input space is D dimensional (i.e. there are D input units) we can write the input patterns as $\mathbf{x} = \{x_i : i = 1, \ldots, D\}$ and the connection weights between the input units i and the neurons j in the computation layer can be written $\mathbf{w}_j = \{w_{ji} : j = 1, \ldots, N; i = 1, \ldots, D\}$ where N is the total number of neurons.

We can then define our ***discriminant function*** to be the squared Euclidean distance between the input vector $\mathbf{x}$ and the weight vector $\mathbf{w}_j$ for each neuron j

$$d_j(\mathbf{x}) = \sum_{i=1}^{D} (x_i - w_{ji})^2$$

In other words, the neuron whose weight vector comes closest to the input vector (i.e. is most similar to it) is declared the winner.

SOM: Four major components

- The Cooperative Process

We want to define a similar topological neighbourhood for the neurons in our SOM. If S_{ij} is the lateral distance between neurons i and j on the grid of neurons, we take

$$T_{j,I(\mathbf{x})} = \exp(-S^2_{j,I(\mathbf{x})} / 2\sigma^2)$$

as our topological neighbourhood, where $I(\mathbf{x})$ is the index of the winning neuron.

A special feature of the SOM is that the size σ of the neighbourhood needs to decrease with time. A popular time dependence is an exponential decay: $\sigma(t) = \sigma_0 \exp(-t / \tau_\sigma)$.

SOM: Four major components

- The Adaptive Process

The point of the topographic neighbourhood is that not only the winning neuron gets its weights updated, but its neighbours will have their weights updated as well, although by not as much as the winner itself. In practice, the appropriate weight update equation is

$$\Delta w_{ji} = \eta(t) \, . \, T_{j,I(\mathbf{x})}(t) \, . \, (x_i - w_{ji})$$

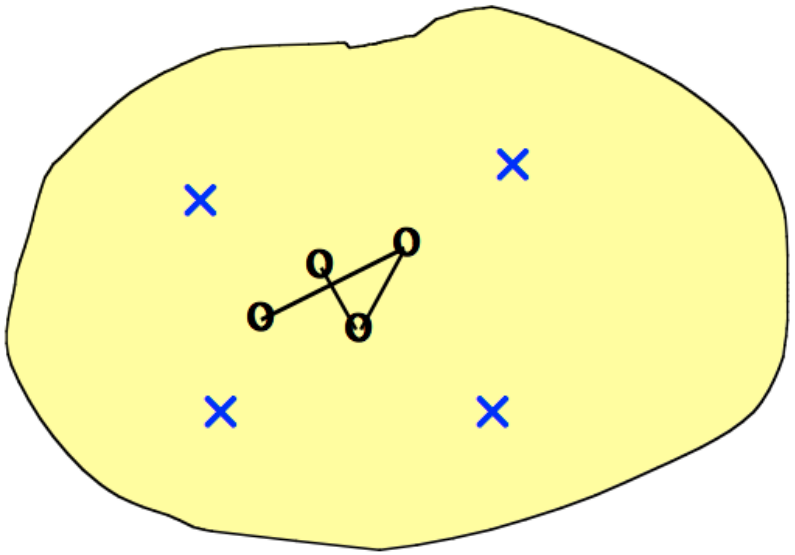
in which we have a time (epoch) t dependent learning rate $\eta(t) = \eta_0 \exp(-t / \tau_\eta)$, and the updates are applied for all the training patterns $\mathbf{x}$ over many epochs.

SOM: Four major components

The stages of the SOM algorithm can be summarised as follows:

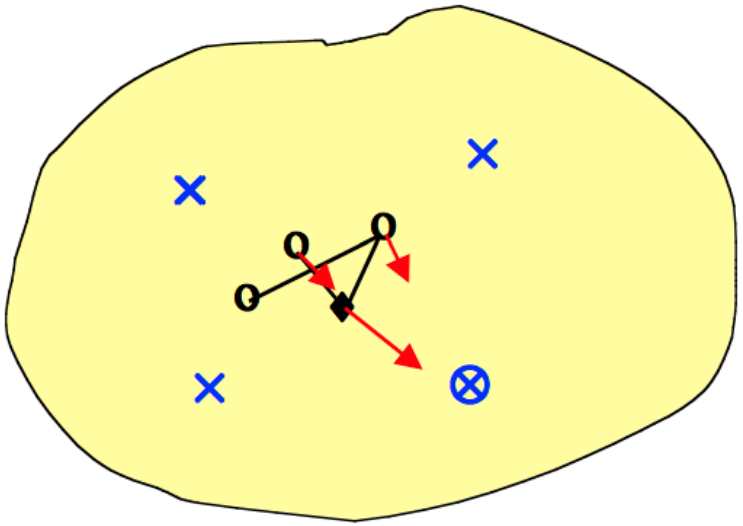
1. ***Initialization*** – Choose random values for the initial weight vectors $\mathbf{w}_j$.
2. ***Sampling*** – Draw a sample training input vector $\mathbf{x}$ from the input space.
3. ***Matching*** – Find the winning neuron $I(\mathbf{x})$ with weight vector closest to input vector.
4. ***Updating*** – Apply the weight update equation $\Delta w_{ji} = \eta(t)\, T_{j,I(\mathbf{x})}(t)\, (x_i - w_{ji})$.
5. ***Continuation*** – keep returning to step 2 until the feature map stops changing.

SOM: Visualizing



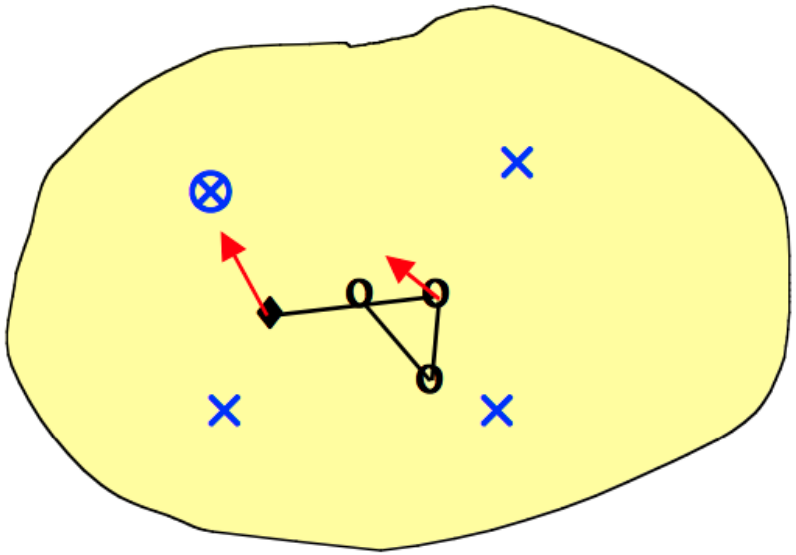
Suppose we have four data points (crosses) in our continuous 2D input space, and want to map this onto four points in a discrete 1D output space. The output nodes map to points in the input space (circles). Random initial weights start the circles at random positions in the centre of the input space.

SOM: Visualizing



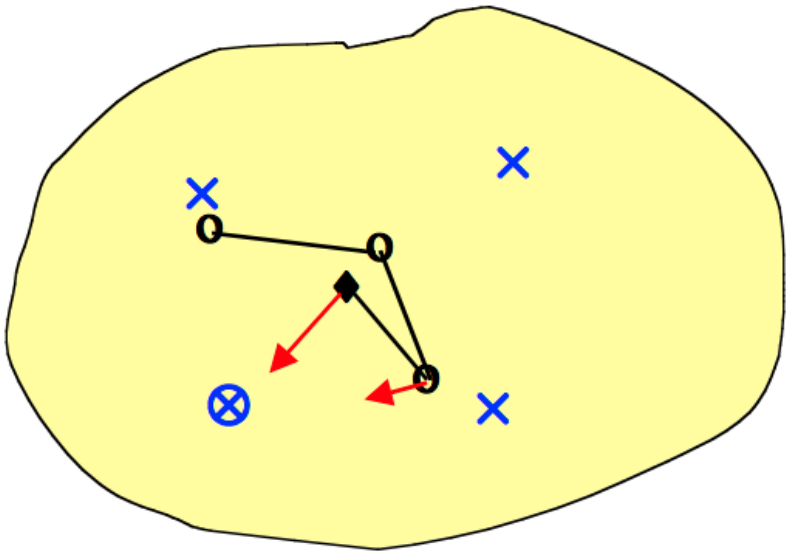
We randomly pick one of the data points for training (cross in circle). The closest output point represents the winning neuron (solid diamond). That winning neuron is moved towards the data point by a certain amount, and the two neighbouring neurons move by smaller amounts (arrows).

SOM: Visualizing



Next we randomly pick another data point for training (cross in circle). The closest output point gives the new winning neuron (solid diamond). The winning neuron moves towards the data point by a certain amount, and the one neighbouring neuron moves by a smaller amount (arrows).

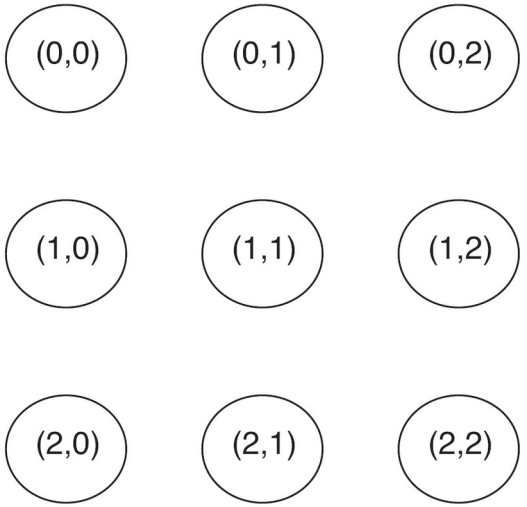
SOM: Visualizing



We carry on randomly picking data points for training (cross in circle). Each winning neuron moves towards the data point by a certain amount, and its neighbouring neuron(s) move by smaller amounts (arrows). Eventually the whole output grid unravels itself to represent the input space.

SOM: Self-Organizing Maps

- Self-organizing maps (SOM)
  - Centroid based clustering scheme
  - Like K-means, a fixed number of clusters are specified
  - However, the spatial relationship of clusters is also specified, typically as a grid
  - Points are considered one by one
  - Each point is assigned to the closest centroid
  - Other centroids are updated based on their nearness to the closest centroid



Kohonen, Teuvo, and Self-Organizing Maps. "Springer series in information sciences." *Self-organizing maps* 30 (1995).

SOM: Self-Organizing Maps

Algorithm 9.3 Basic SOM Algorithm.

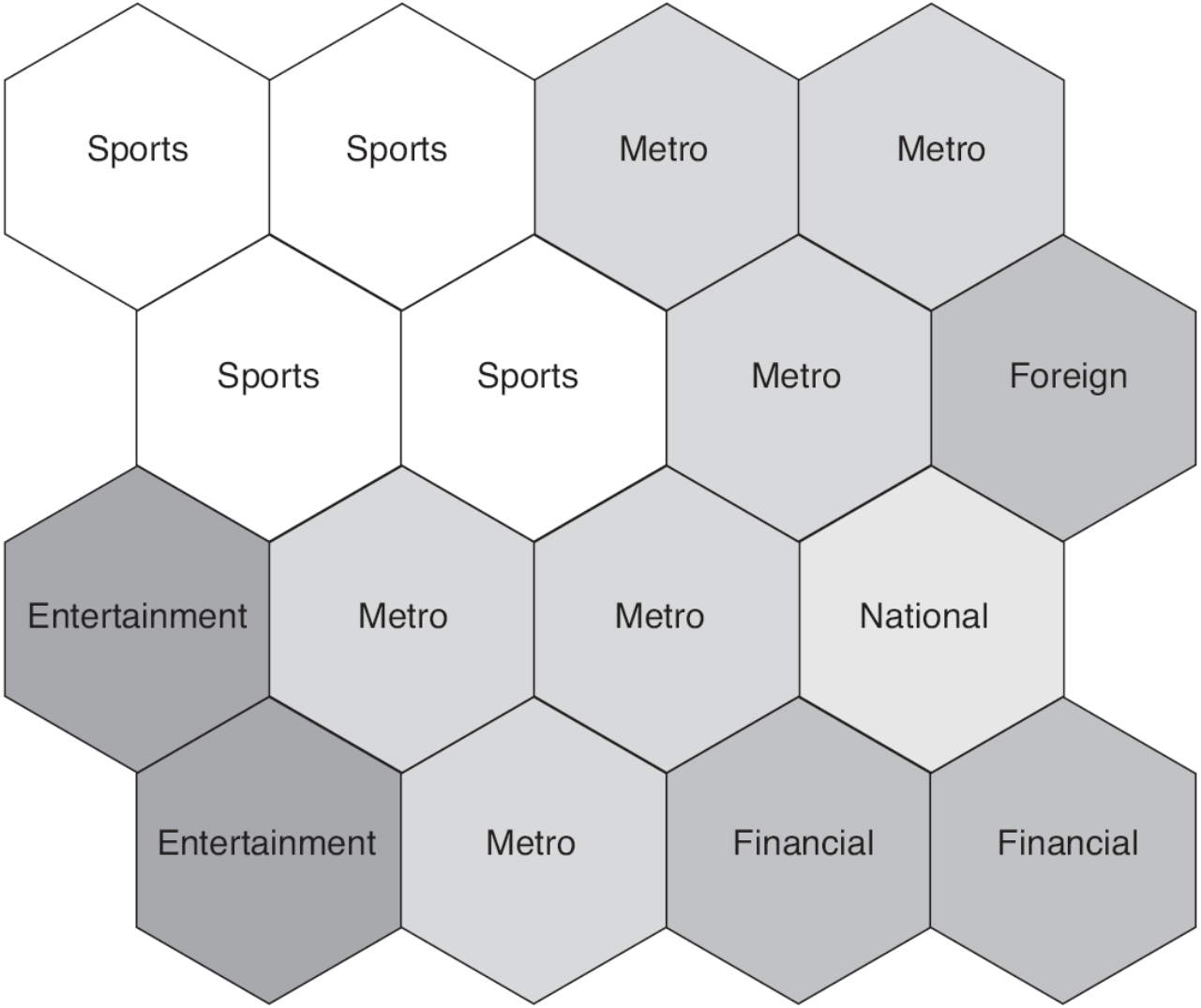
1: Initialize the centroids.
2: **repeat**
3: Select the next object.
4: Determine the closest centroid to the object.
5: Update this centroid and the centroids that are close, i.e., in a specified neighborhood.
6: **until** The centroids don't change much or a threshold is exceeded.
7: Assign each object to its closest centroid and return the centroids and clusters.

- Updates are weighted by distance
  - Centroids farther away are affected less
- The impact of the updates decreases with each time
  - At some point the centroids will not change much

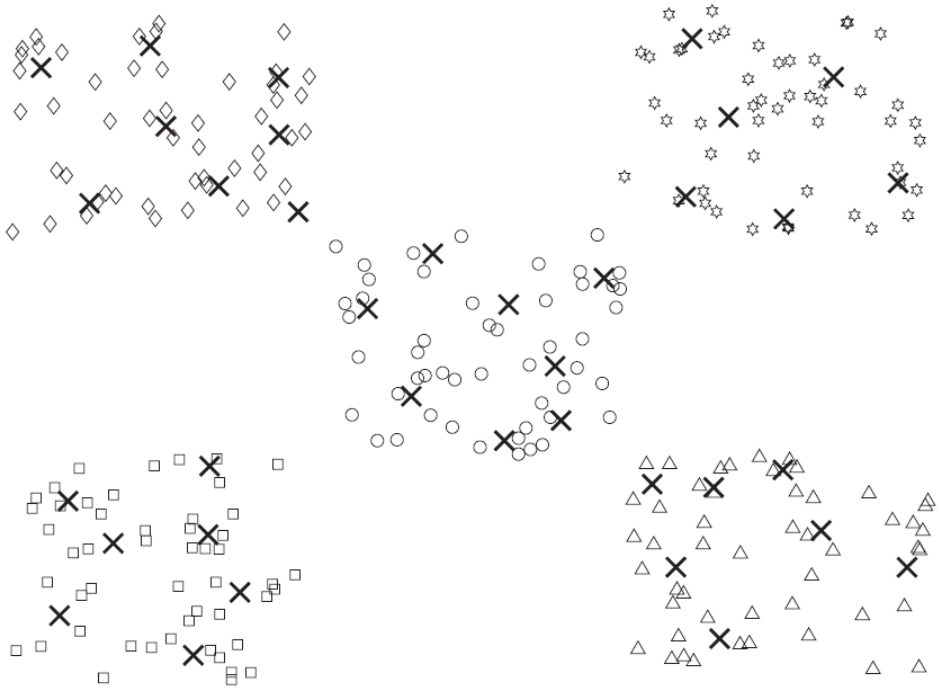
SOM: Self-Organizing Maps

- SOM can be viewed as a type of dimensionality reduction
- If a two-dimensional grid is used, the results can be visualized

SOM Clusters of LA Times Document Data



Another SOM Example: 2D Points



(a) Distribution of SOM reference vectors (**X**'s) for a two-dimensional point set.

diamond	diamond	diamond	hexagon	hexagon	hexagon
diamond	diamond	diamond	circle	hexagon	hexagon
diamond	diamond	circle	circle	circle	hexagon
square	square	circle	circle	triangle	triangle
square	square	circle	circle	triangle	triangle
square	square	square	triangle	triangle	triangle

(b) Classes of the SOM centroids.

Issues with SOM

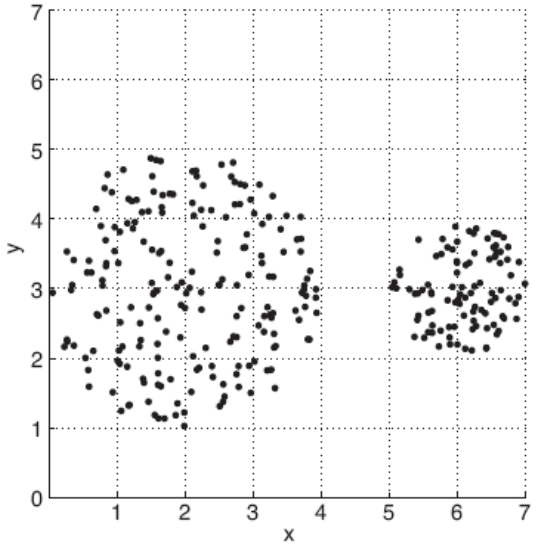
- Computational complexity
- Locally optimal solution
- Grid is somewhat arbitrary

Grid-based Clustering

- A type of density-based clustering

Algorithm 9.4 Basic grid-based clustering algorithm.

1: Define a set of grid cells.
2: Assign objects to the appropriate cells and compute the density of each cell.
3: Eliminate cells having a density below a specified threshold, τ.
4: Form clusters from contiguous (adjacent) groups of dense cells.

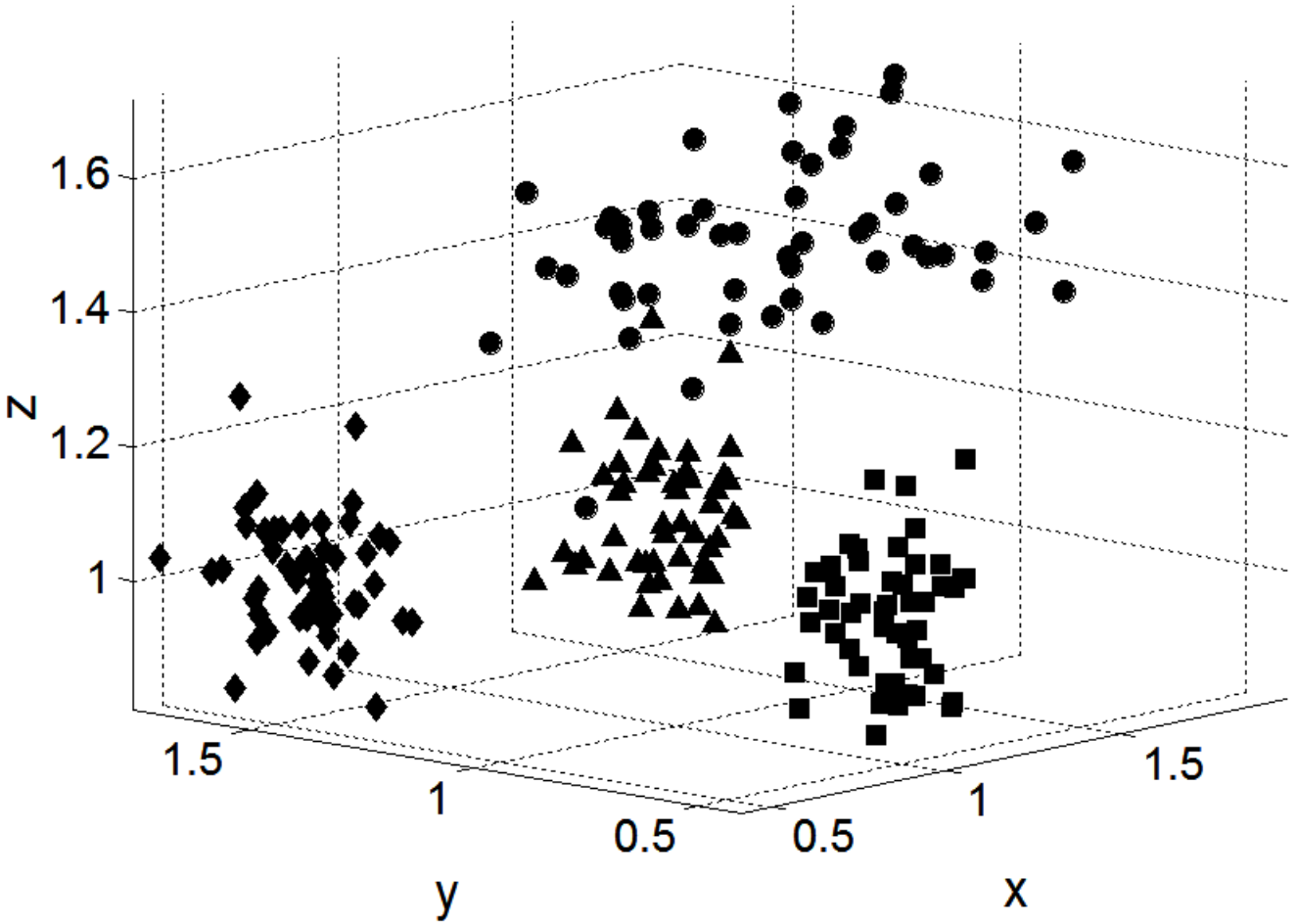


0	0	0	0	0	0	0
0	0	0	0	0	0	0
4	17	18	6	0	0	0
14	14	13	13	0	18	27
11	18	10	21	0	24	31
3	20	14	4	0	0	0
0	0	0	0	0	0	0

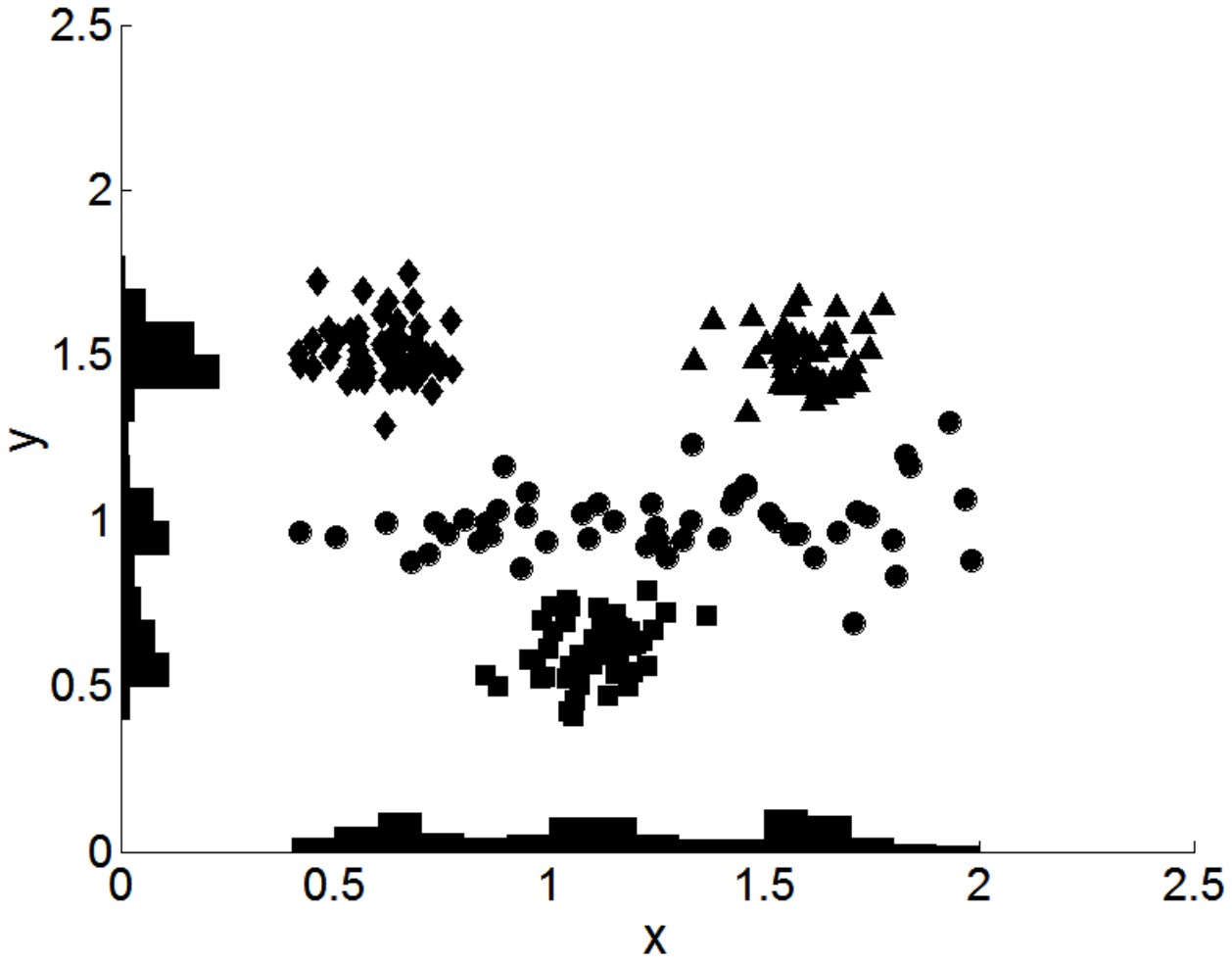
Subspace Clustering

- Until now, we found clusters by considering all of the attributes
- Some clusters may involve only a subset of attributes, i.e., subspaces of the data
  - Example:
    - When k-means is used to find document clusters, the resulting clusters can typically be characterized by 10 or so terms

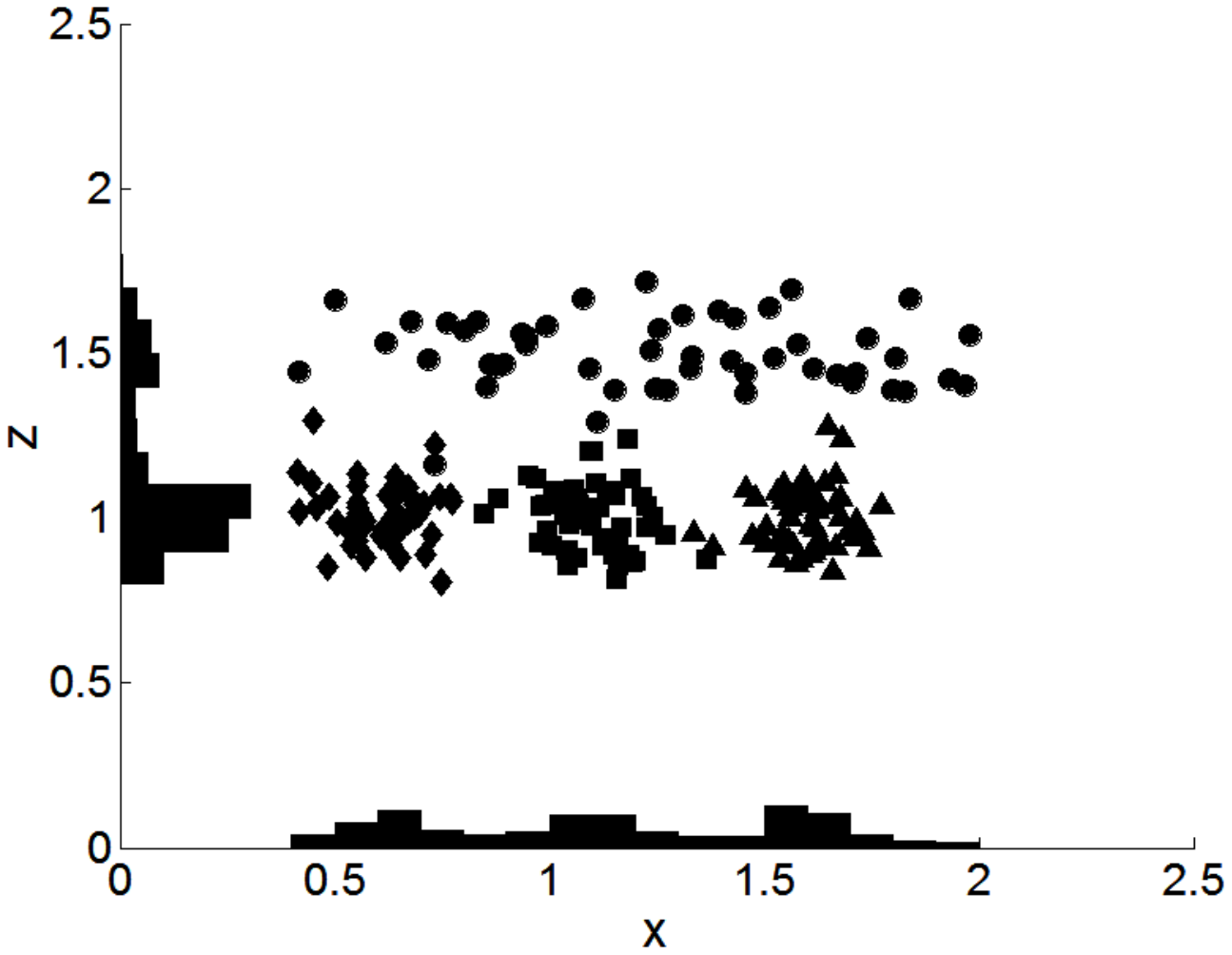
Example



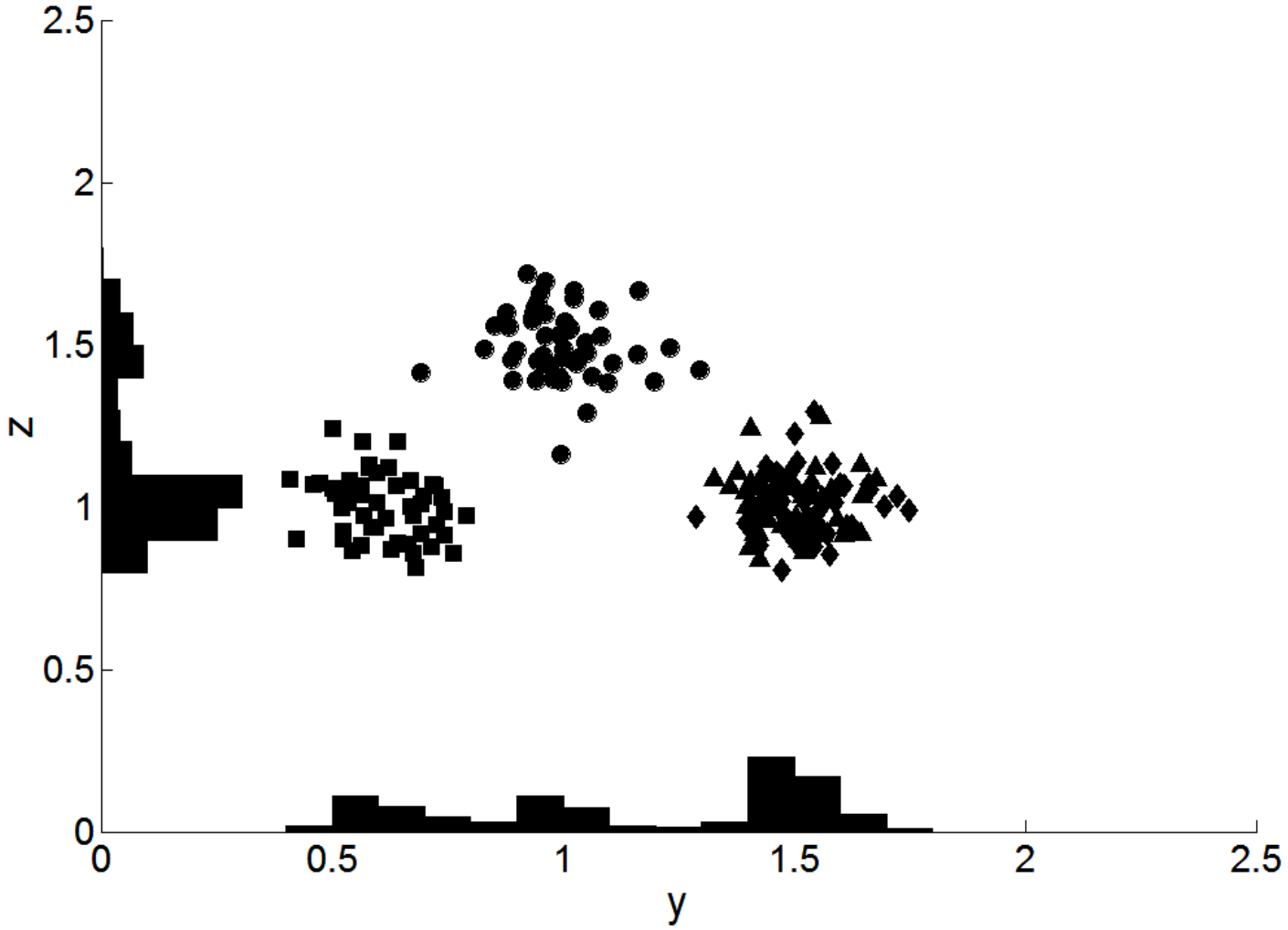
Example



Example



Example



Clique Algorithm - Overview

- A grid-based clustering algorithm that methodically finds subspace clusters
  - Partitions the data space into rectangular units of equal volume
  - Measures the density of each unit by the fraction of points it contains
  - A unit is dense if the fraction of overall points it contains is above a user specified threshold, τ
  - A cluster is a group of collections of contiguous (touching) dense units

Clique Algorithm

- It is impractical to check each volume unit to see if it is dense since there is exponential number of such units
- Monotone property of density-based clusters:
  - If a set of points forms a density based cluster in k dimensions, then the same set of points is also part of a density based cluster in all possible subsets of those dimensions
- Very similar to Apriori algorithm
- Can find overlapping clusters

Clique Algorithm

Algorithm 9.5 CLIQUE.

1: Find all the dense areas in the one-dimensional spaces corresponding to each attribute. This is the set of dense one-dimensional cells.
2: $k \leftarrow 2$
3: **repeat**
4: Generate all candidate dense k-dimensional cells from dense $(k-1)$-dimensional cells.
5: Eliminate cells that have fewer than ξ points.
6: $k \leftarrow k + 1$
7: **until** There are no candidate dense k-dimensional cells.
8: Find clusters by taking the union of all adjacent, high-density cells.
9: Summarize each cluster using a small set of inequalities that describe the attribute ranges of the cells in the cluster.

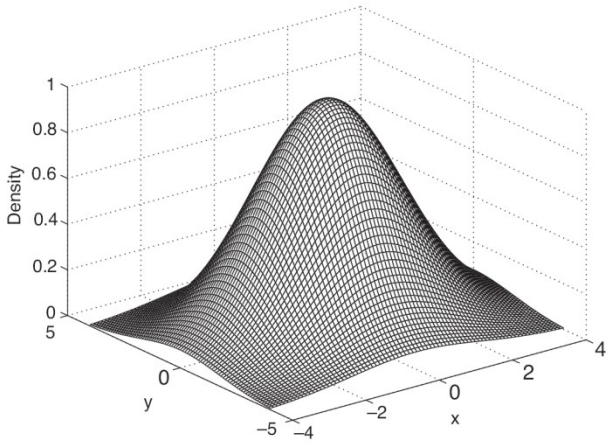
Limitations of Clique

- Time complexity is exponential in number of dimensions
  - Especially if “too many” dense units are generated at lower stages
- May fail if clusters are of widely differing densities, since the threshold is fixed
  - Determining appropriate threshold and unit interval length can be challenging

Denclue (DENsity CLUstering)

- Based on the notion of kernel-density estimation
  - Contribution of each point to the density is given by an influence or kernel function

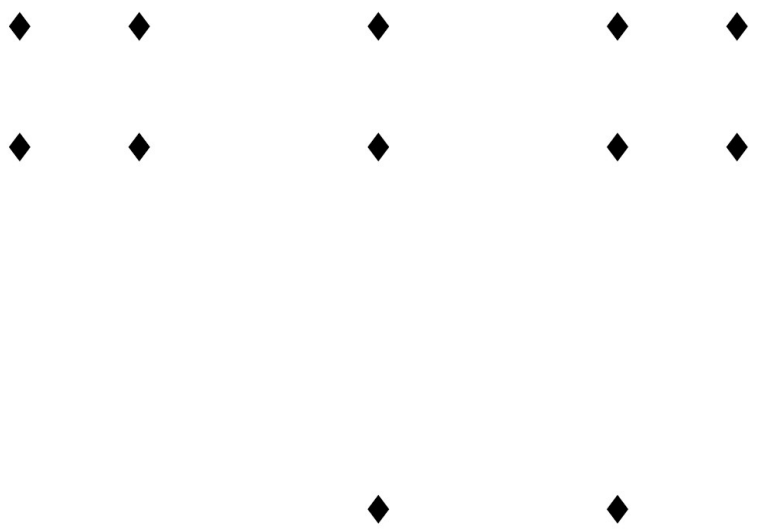
$$K(y) = e^{-distance(\mathbf{x},\mathbf{y})^2/2\sigma^2}$$



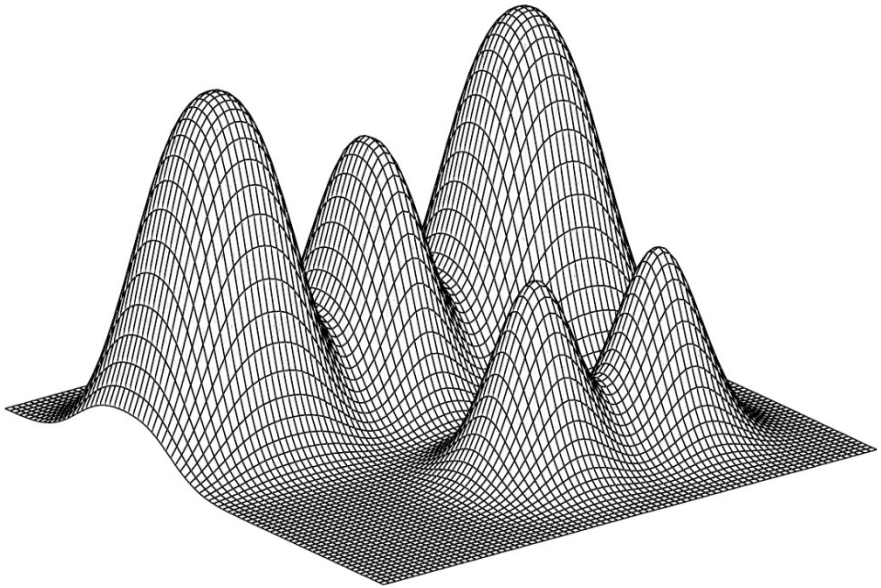
Formula and plot of Gaussian Kernel

  - Overall density is the sum of the contributions of all points

Example of Density from Gaussian Kernel

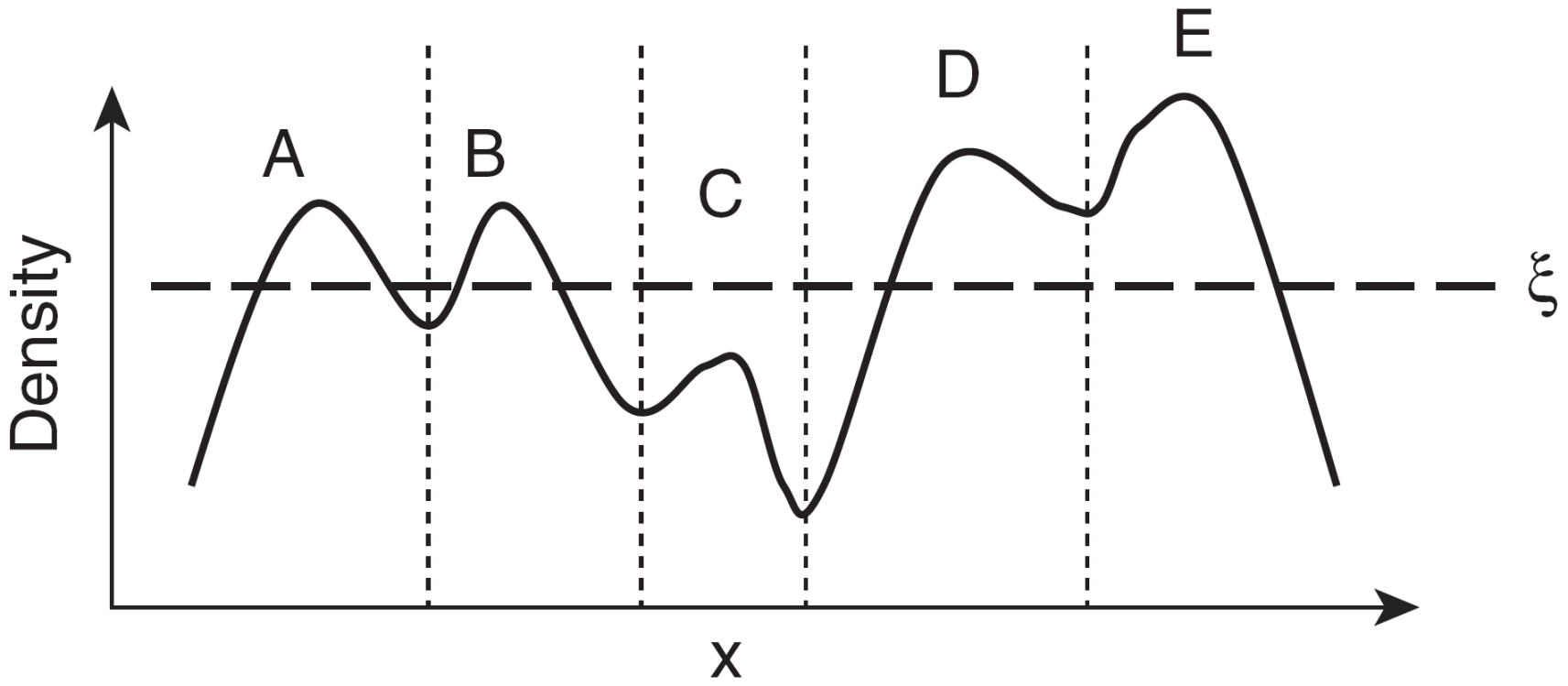


Set of 12 points.



Overall density—surface plot.

DENCLUE Algorithm



DENCLUE Algorithm

- Find the density function
- Identify local maxima (density attractors)
- Assign each point to the density attractor
  - Follow direction of maximum increase in density
- Define clusters as groups consisting of points associated with density attractor
- Discard clusters whose density attractor has a density less than a user specified minimum, ξ
- Combine clusters connected by paths of points that are connected by points with density above ξ

Graph-Based Clustering: General Concepts

- Graph-Based clustering uses the proximity graph
  - Start with the proximity matrix
  - Consider each point as a node in a graph
  - Each edge between two nodes has a weight which is the proximity between the two points
  - Initially the proximity graph is fully connected
  - MIN (single-link) and MAX (complete-link) can be viewed in graph terms

- In the simplest case, clusters are connected components in the graph.

CURE Algorithm: Graph-Based Clustering

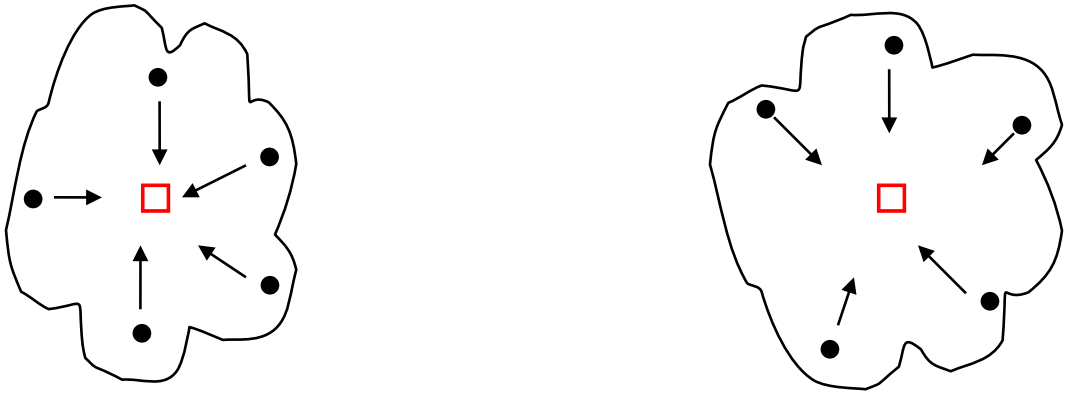
- Agglomerative hierarchical clustering algorithms vary in terms of how the proximity of two clusters are computed
  - MIN (single link)
    - susceptible to noise/outliers
  - MAX (complete link)/GROUP AVERAGE/Centroid/Ward's:
    - may not work well with non-globular clusters
- CURE algorithm tries to handle both problems

CURE Algorithm

- Represents a cluster using multiple representative points
  - Representative points are found by selecting a constant number of points from a cluster
    - First representative point is chosen to be the point furthest from the center of the cluster
    - Remaining representative points are chosen so that they are farthest from all previously chosen points

CURE Algorithm

- “Shrink” representative points toward the center of the cluster by a factor, α



- Shrinking representative points toward the center helps avoid problems with noise and outliers

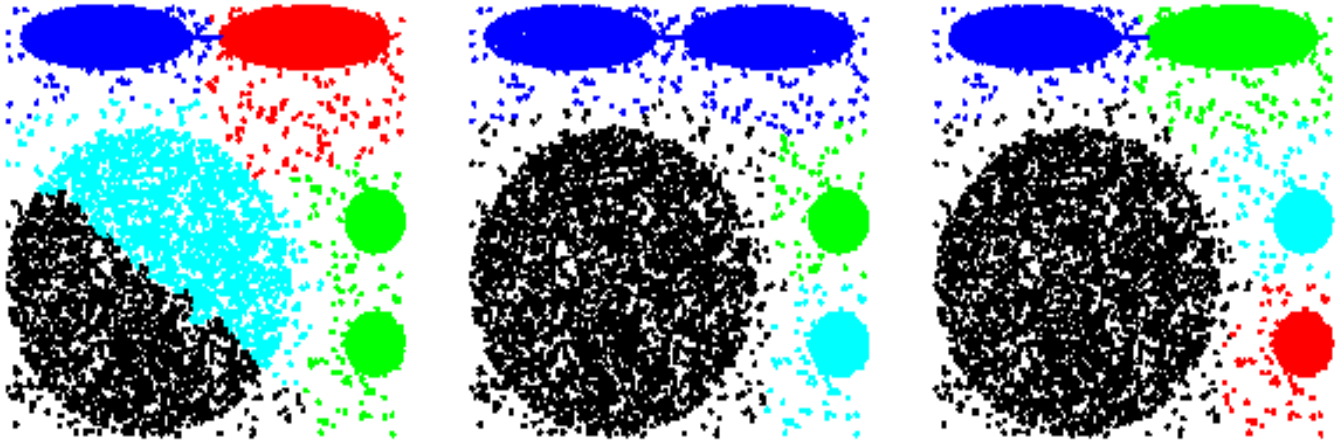
- Cluster similarity is the similarity of the closest pair of representative points from different clusters

CURE Algorithm

- Uses agglomerative hierarchical scheme to perform clustering;
  - α = 0: similar to centroid-based
  - α = 1: somewhat similar to single-link

- CURE is better able to handle clusters of arbitrary shapes and sizes

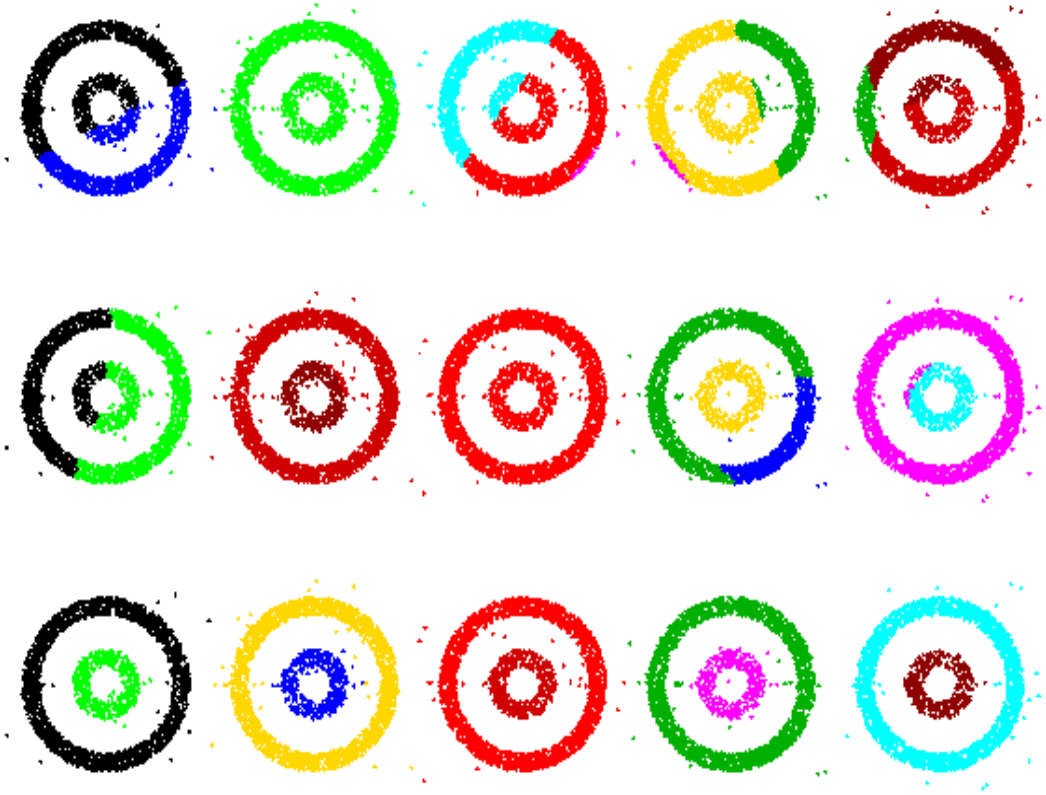
Experimental Results: CURE



a) BIRCH b) MST METHOD c) CURE

Picture from *CURE*, Guha, Rastogi, Shim.

Experimental Results: CURE



a) BIRCH

(centroid)

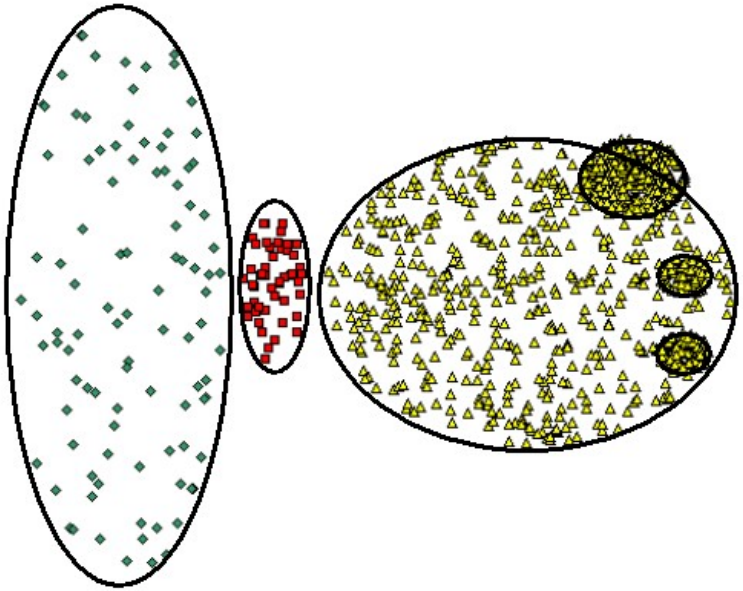
b) MST METHOD (sing

(single link)

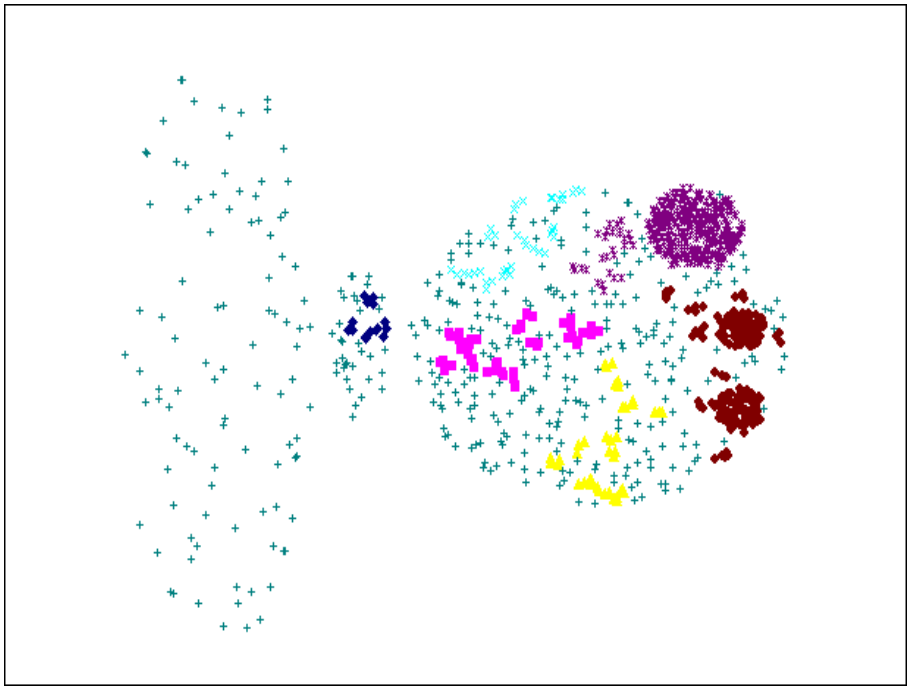
c) CURE

Picture from *CURE*, Guha, Rastogi, Shim.

CURE Cannot Handle Differing Densities



Original Points



CURE

Graph-Based Clustering: Chameleon

- Based on several key ideas
  - Sparsification of the proximity graph
  - Partitioning the data into clusters that are relatively pure subclusters of the “true” clusters
  - Merging based on preserving characteristics of clusters

Graph-Based Clustering: Sparsification

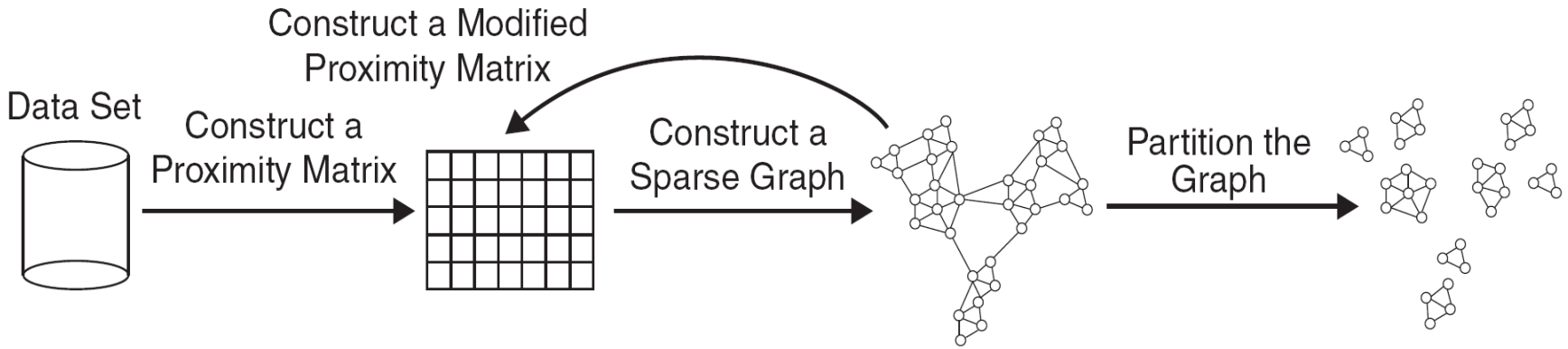
- The amount of data that needs to be processed is drastically reduced
  - Sparsification can eliminate more than 99% of the entries in a proximity matrix
  - The amount of time required to cluster the data is drastically reduced
  - The size of the problems that can be handled is increased

Graph-Based Clustering: Sparsification …

- Clustering may work better
  - Sparsification techniques keep the connections to the most similar (nearest) neighbors of a point while breaking the connections to less similar points.
  - The nearest neighbors of a point tend to belong to the same class as the point itself.
  - This reduces the impact of noise and outliers and sharpens the distinction between clusters.

- Sparsification facilitates the use of graph partitioning algorithms (or algorithms based on graph partitioning algorithms)
  - Chameleon and Hypergraph-based Clustering

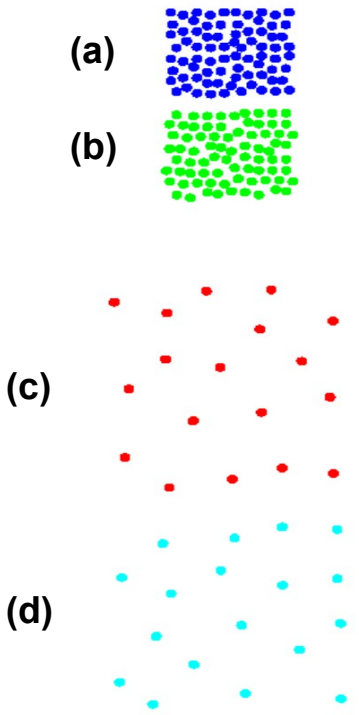
Sparsification in the Clustering Process



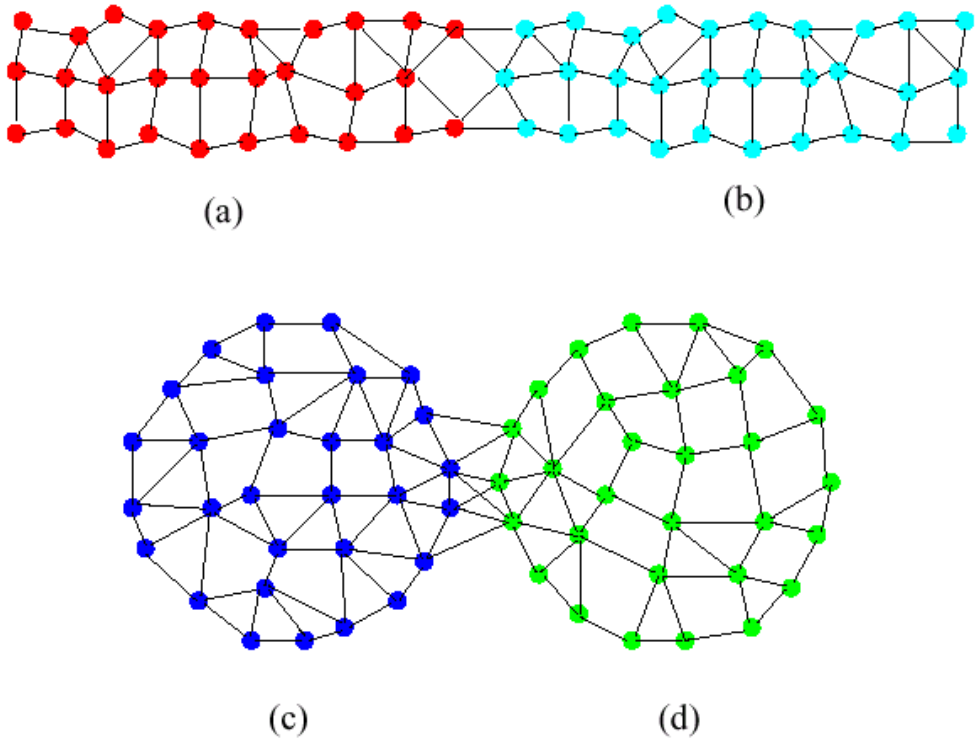
Limitations of Current Merging Schemes

- Existing merging schemes in hierarchical clustering algorithms are static in nature
  - MIN or CURE:
    - Merge two clusters based on their *closeness* (or minimum distance)
  - GROUP-AVERAGE:
    - Merge two clusters based on their average *connectivity*

Limitations of Current Merging Schemes



Closeness schemes will merge (a) and (b)



Average connectivity schemes will merge (c) and (d)

Chameleon: Clustering Using Dynamic Modeling

- Adapt to the characteristics of the data set to find the natural clusters
- Use a dynamic model to measure the similarity between clusters
  - Main properties are the relative closeness and relative inter-connectivity of the cluster
  - Two clusters are combined if the resulting cluster shares certain *properties* with the constituent clusters
  - The merging scheme preserves *self-similarity*

Relative Interconnectivity

- **Relative Interconnectivity (RI)** is the absolute interconnectivity of two clusters normalized by the internal connectivity of the clusters. Two clusters are combined if the points in the resulting cluster are almost as strongly connected as points in each of the original clusters. Mathematically,

$$RI = \frac{EC(C_i, C_j)}{\frac{1}{2}(EC(C_i) + EC(C_j))}, \tag{9.18}$$

where $EC(C_i, C_j)$ is the sum of the edges (of the k-nearest neighbor graph) that connect clusters C_i and C_j; $EC(C_i)$ is the minimum sum of the cut edges if we bisect cluster C_i; and $EC(C_j)$ is the minimum sum of the cut edges if we bisect cluster C_j.

Relative Closeness

- **Relative Closeness (RC)** is the absolute closeness of two clusters normalized by the internal closeness of the clusters. Two clusters are combined only if the points in the resulting cluster are almost as close to each other as in each of the original clusters. Mathematically,

$$RC = \frac{\bar{S}_{EC}(C_i, C_j)}{\frac{m_i}{m_i+m_j}\bar{S}_{EC}(C_i) + \frac{m_j}{m_i+m_j}\bar{S}_{EC}(C_j)}, \tag{9.17}$$

where m_i and m_j are the sizes of clusters C_i and C_j, respectively, $\bar{S}_{EC}(C_i, C_j)$ is the average weight of the edges (of the k-nearest neighbor graph) that connect clusters C_i and C_j; $\bar{S}_{EC}(C_i)$ is the average weight of edges if we bisect cluster C_i; and $\bar{S}_{EC}(C_j)$ is the average weight of edges if we bisect cluster C_j. (EC stands for edge cut.)

Chameleon: Steps

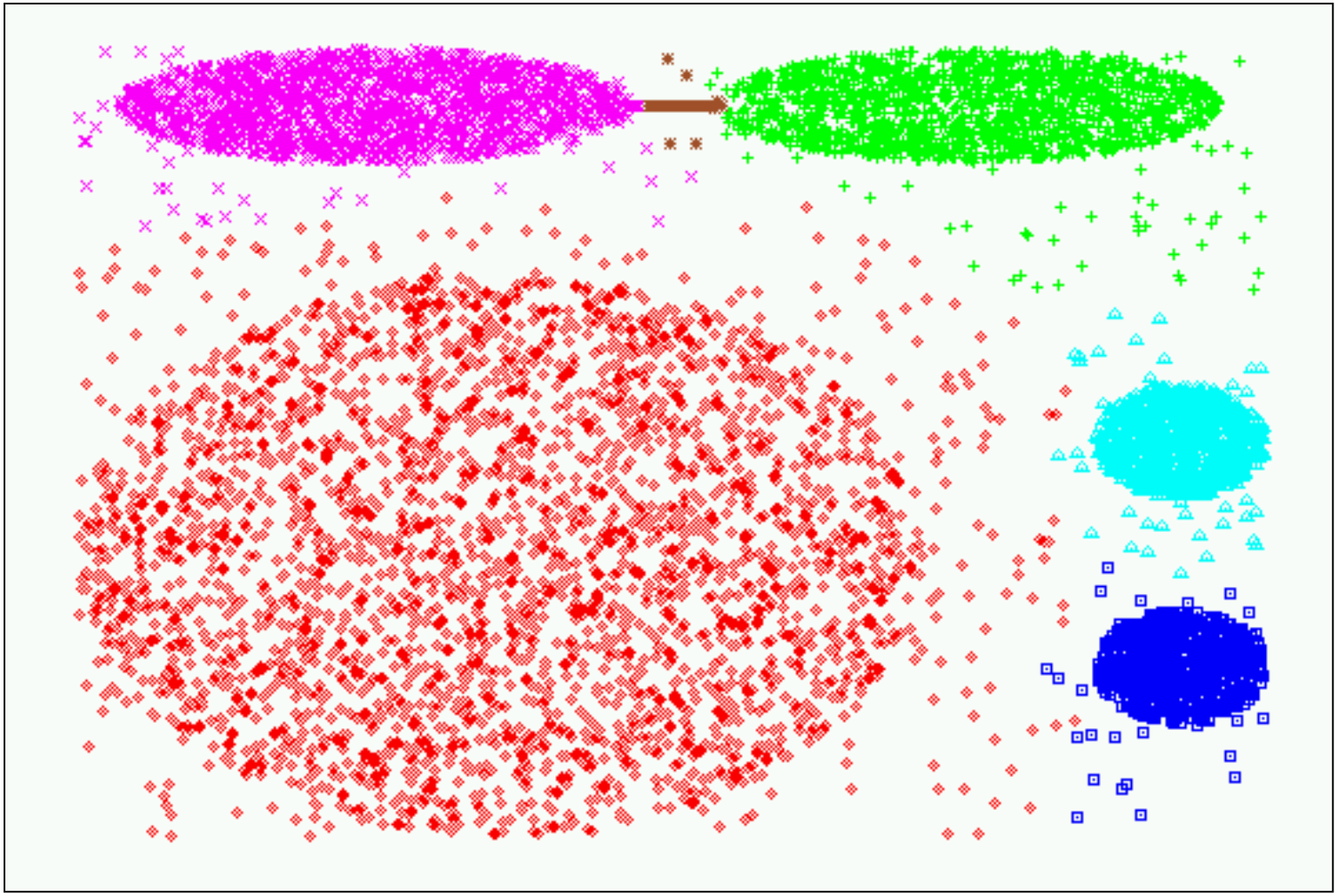
- Preprocessing Step: Represent the data by a Graph
  - Given a set of points, construct the k-nearest-neighbor (k-NN) graph to capture the relationship between a point and its k nearest neighbors
  - Concept of neighborhood is captured dynamically (even if region is sparse)

- Phase 1: Use a multilevel graph partitioning algorithm on the graph to find a large number of clusters of well-connected vertices
  - Each cluster should contain mostly points from one “true” cluster, i.e., be a sub-cluster of a “real” cluster

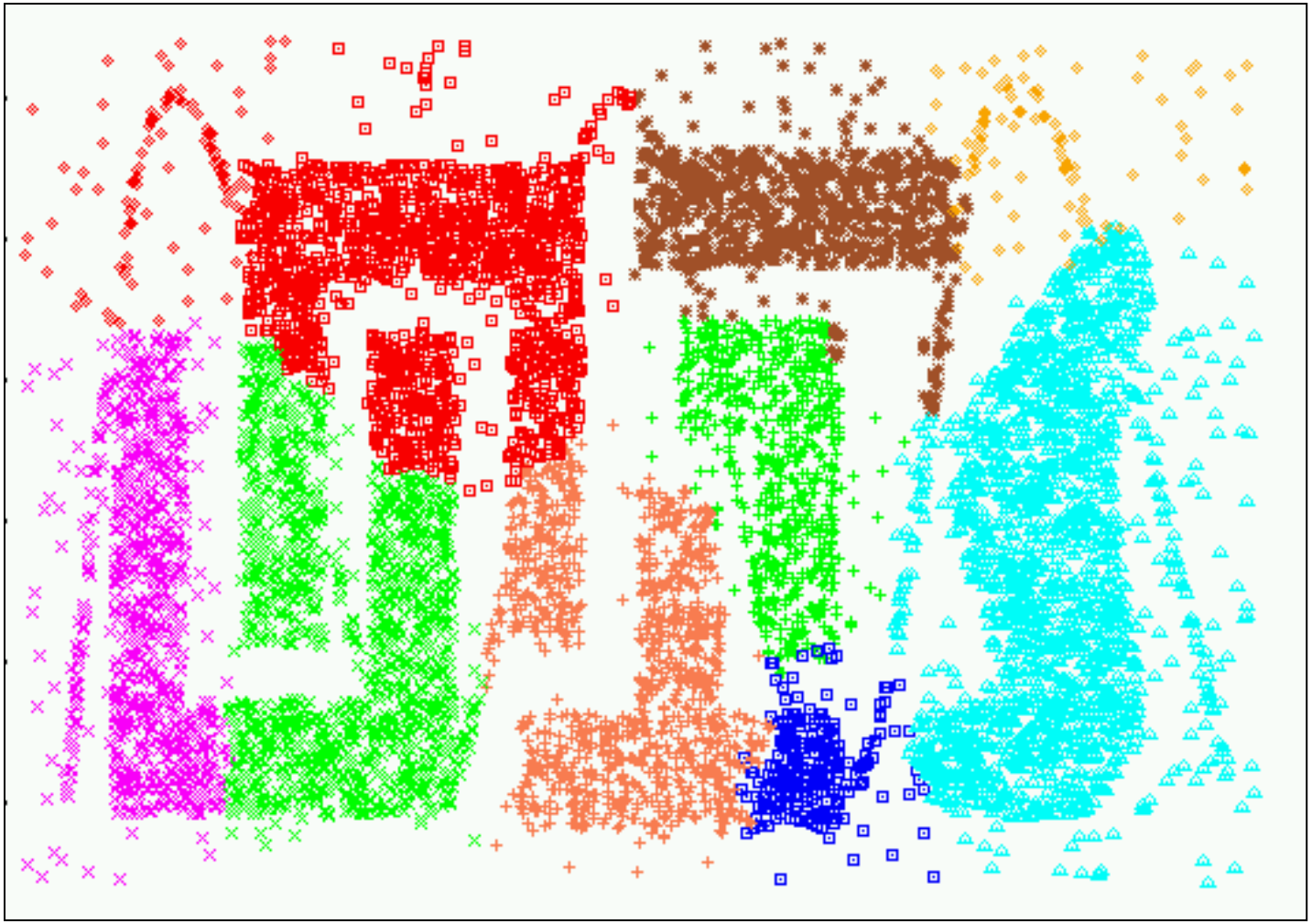
Chameleon: Steps …

- Phase 2: Use Hierarchical Agglomerative Clustering to merge sub-clusters
  - Two clusters are combined if the *resulting cluster shares certain properties with the constituent clusters*
  - Two key properties used to model cluster similarity:
    - Relative Interconnectivity: Absolute interconnectivity of two clusters normalized by the internal connectivity of the clusters
    - Relative Closeness: Absolute closeness of two clusters normalized by the internal closeness of the clusters

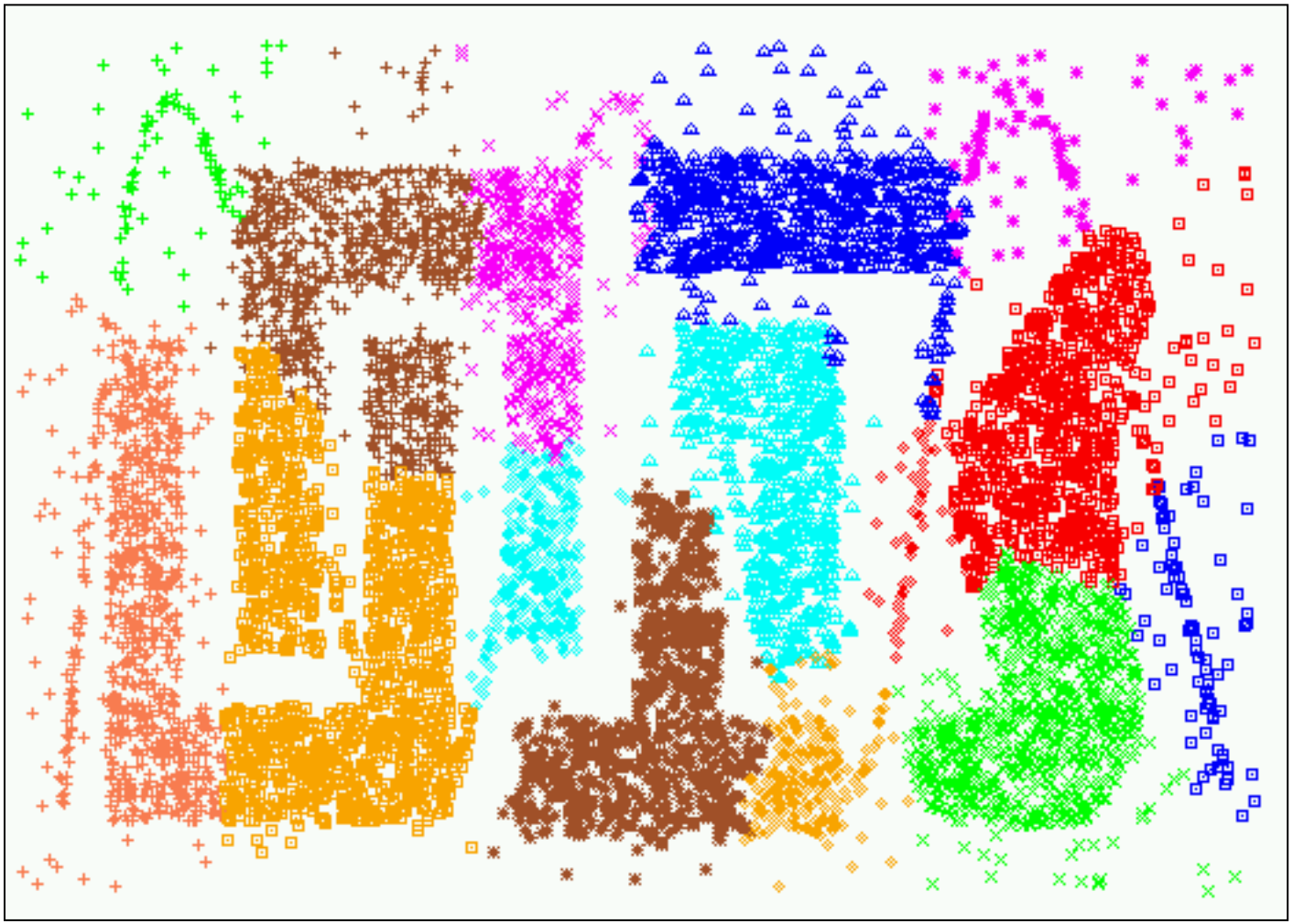
Experimental Results: CHAMELEON



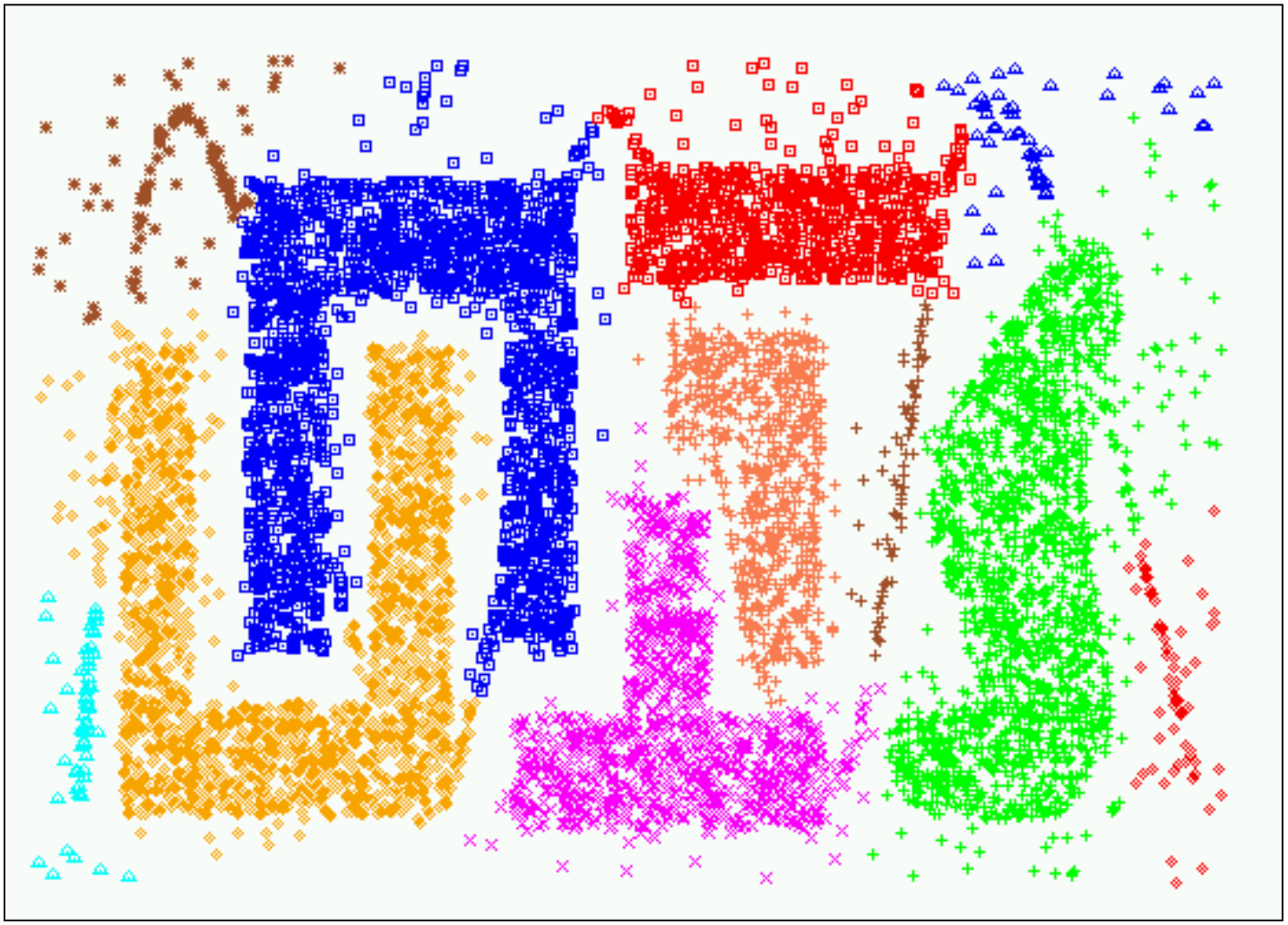
Experimental Results: CURE (*10 clusters*)



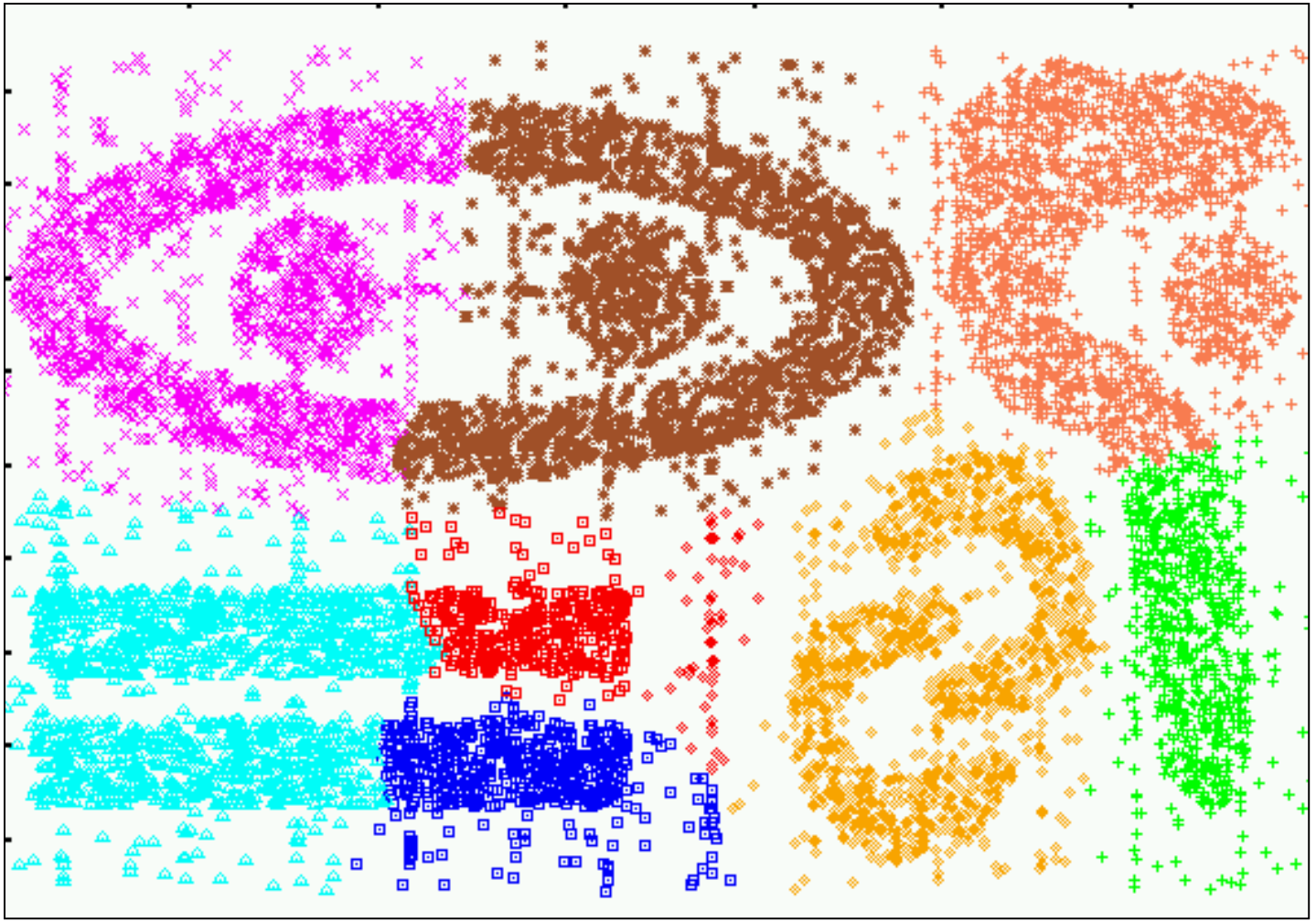
Experimental Results: CURE (*15 clusters*)



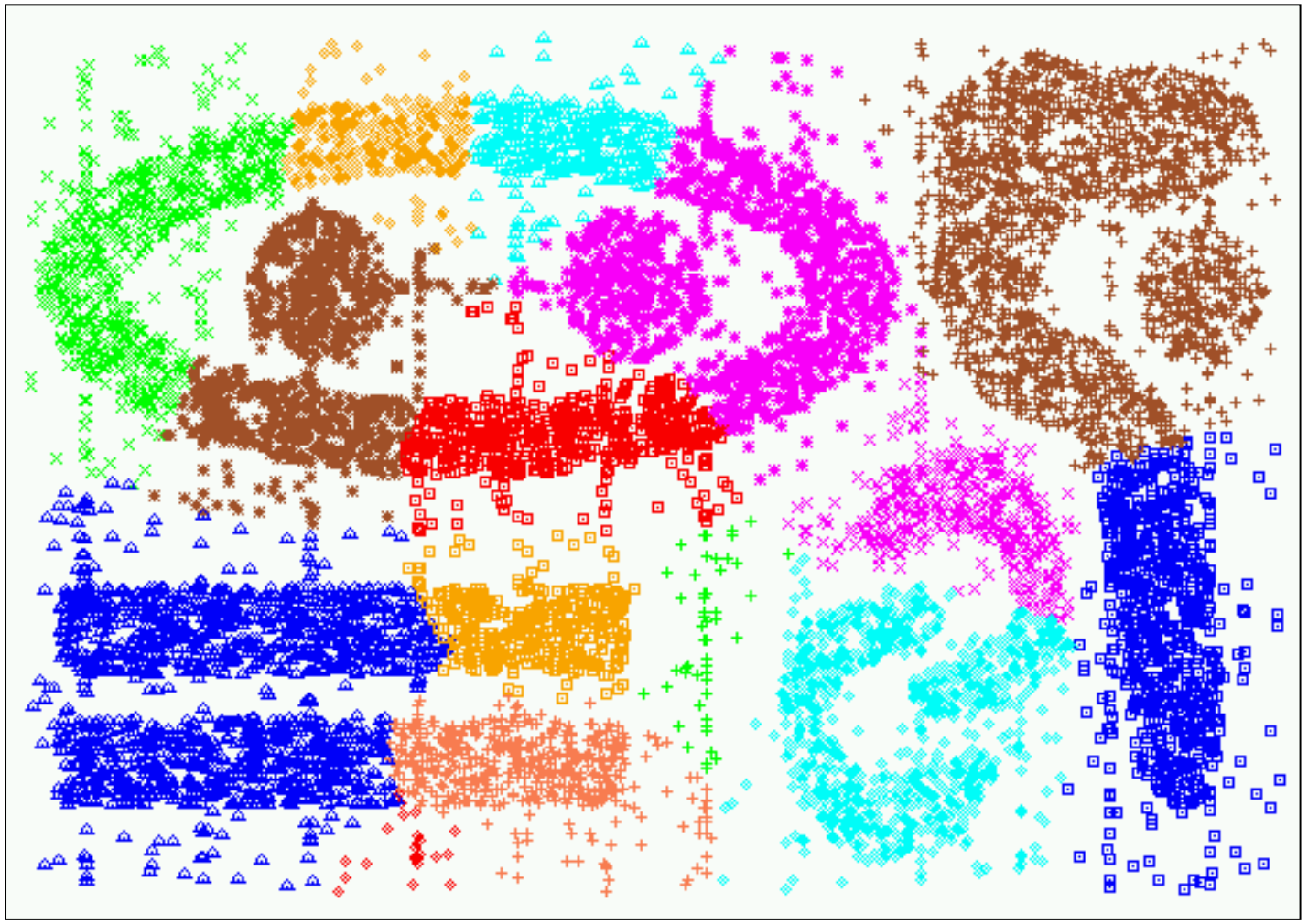
Experimental Results: CHAMELEON



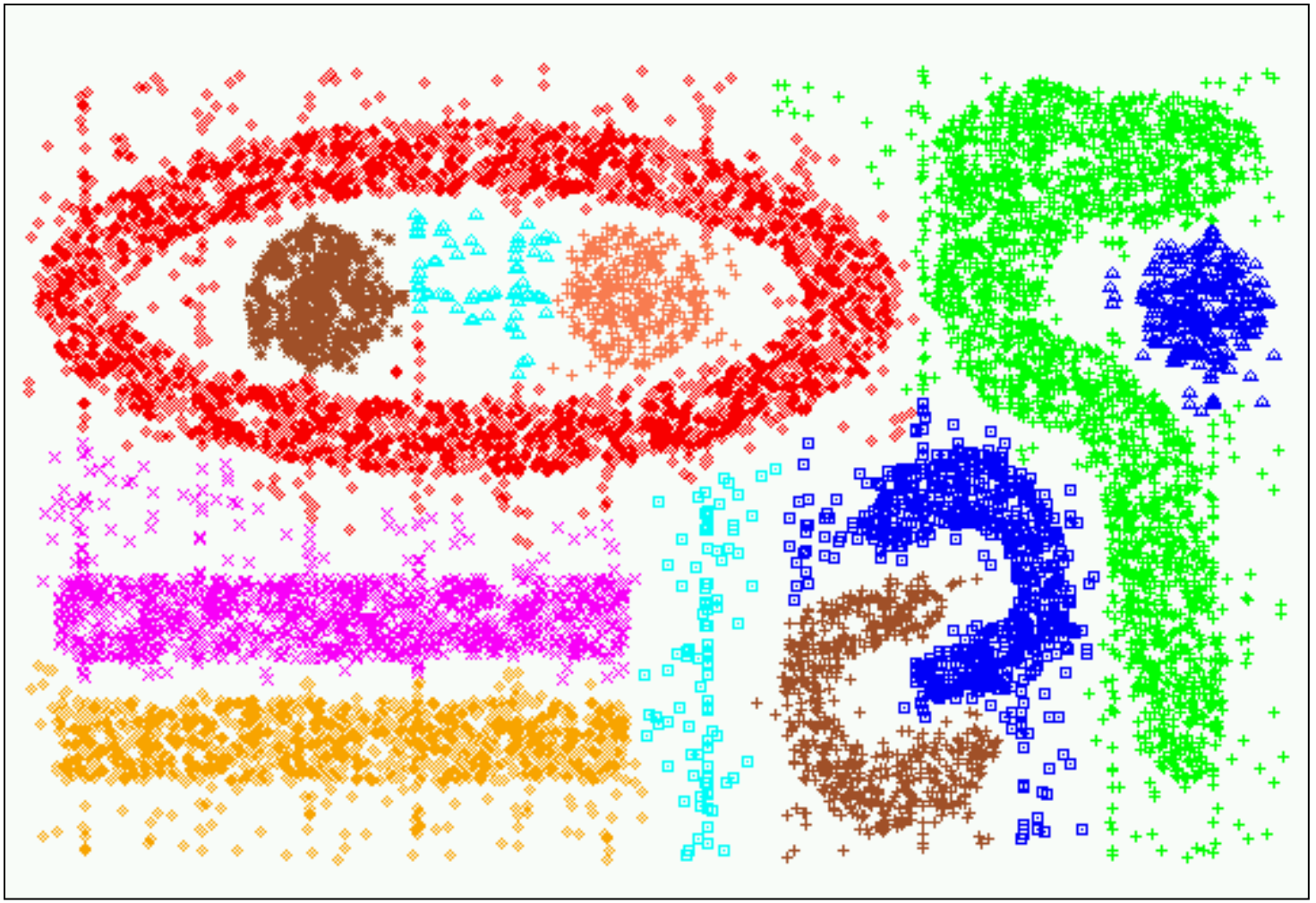
Experimental Results: CURE (*9 clusters*)



Experimental Results: CURE (*15 clusters*)

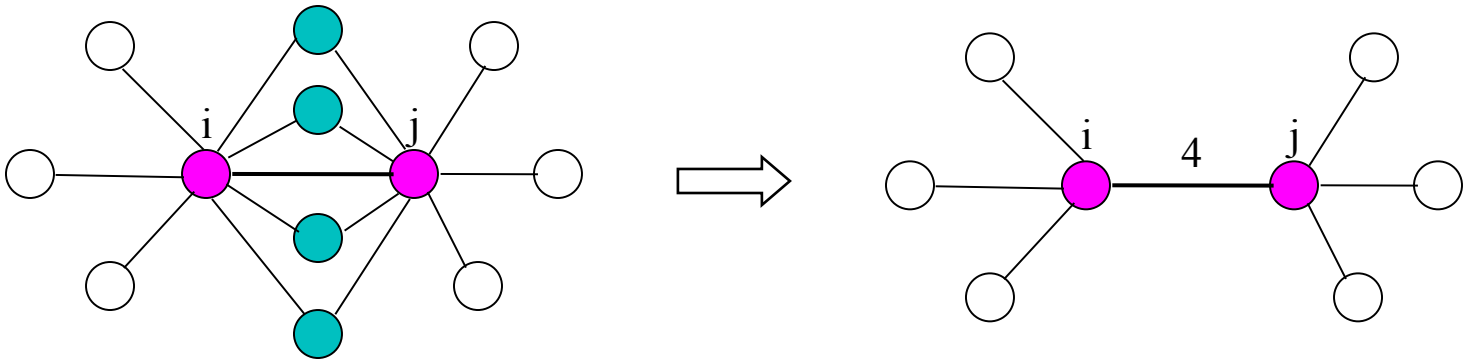


Experimental Results: CHAMELEON

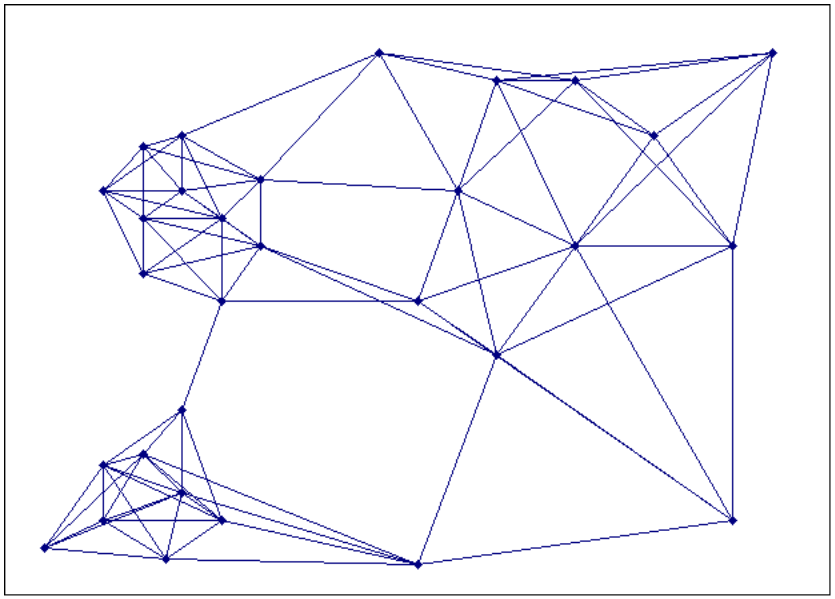


Graph-Based Clustering: SNN Approach

Shared Nearest Neighbor (SNN) graph: the weight of an edge is the number of shared neighbors between vertices given that the vertices are connected

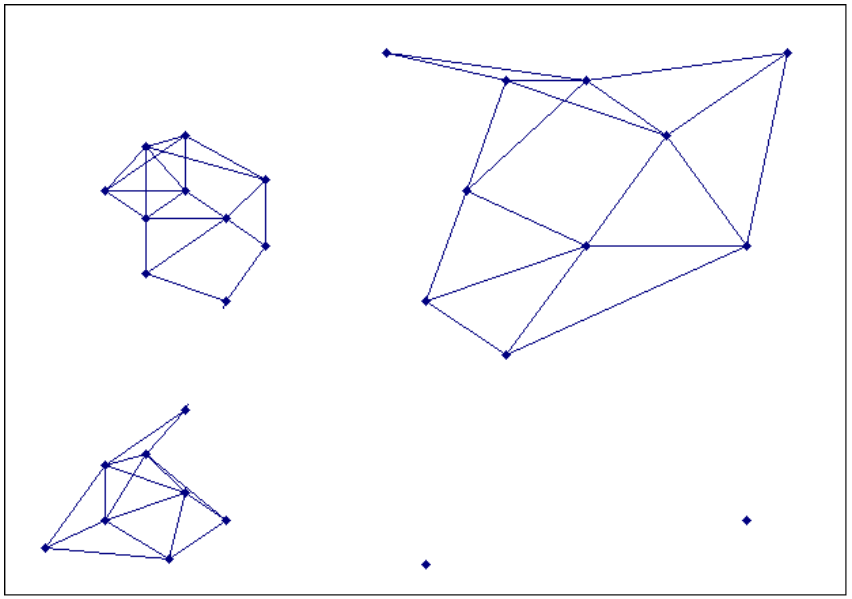


Creating the SNN Graph



Sparse Graph

Link weights are similarities between neighboring points



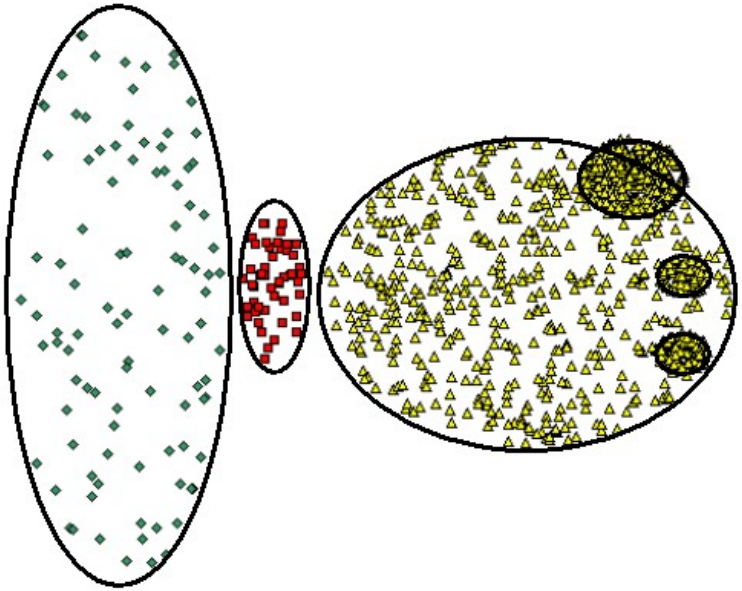
Shared Near Neighbor Graph

Link weights are number of Shared Nearest Neighbors

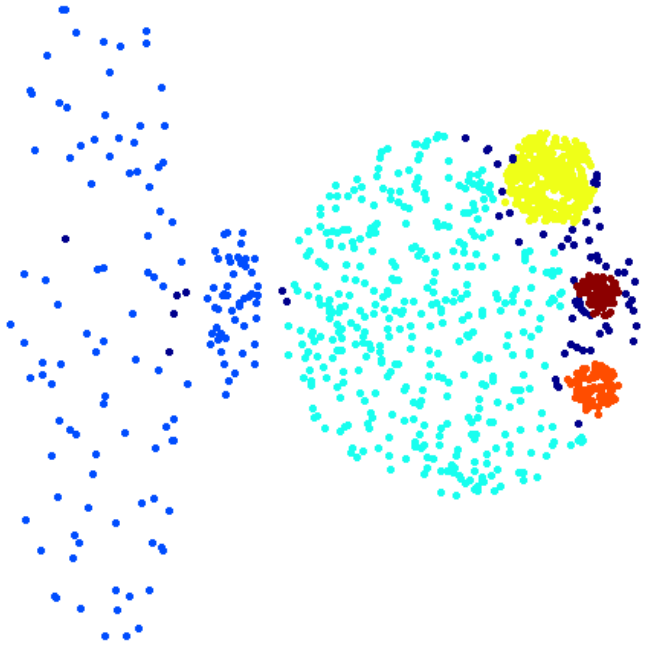
Jarvis-Patrick Clustering

- First, the k-nearest neighbors of all points are found
  - In graph terms this can be regarded as breaking all but the k strongest links from a point to other points in the proximity graph
- A pair of points is put in the same cluster if
  - any two points share more than T neighbors and
  - the two points are in each others k nearest neighbor list
- For instance, we might choose a nearest neighbor list of size 20 and put points in the same cluster if they share more than 10 near neighbors
- Jarvis-Patrick clustering is too brittle

When Jarvis-Patrick Works Reasonably Well



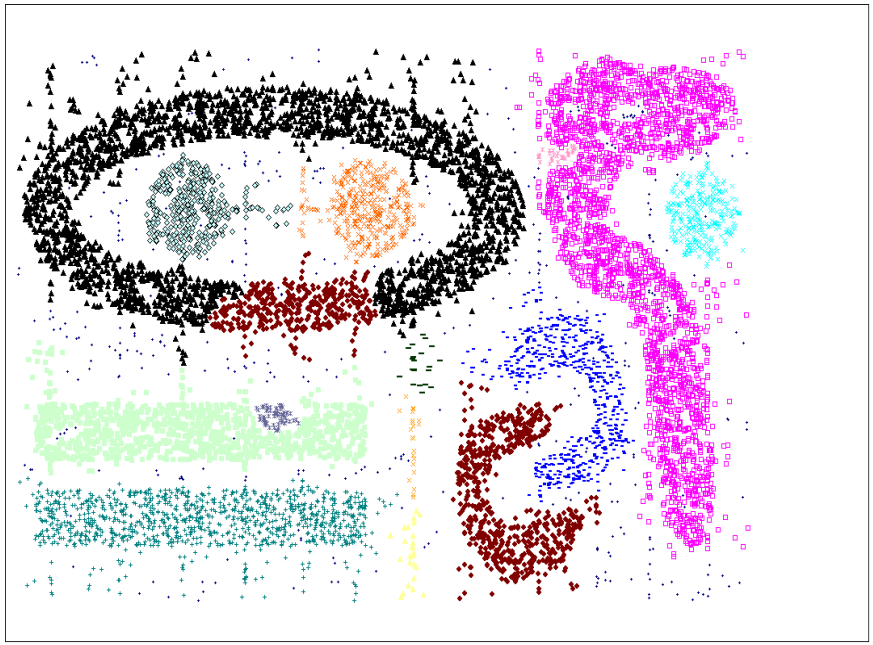
Original Points



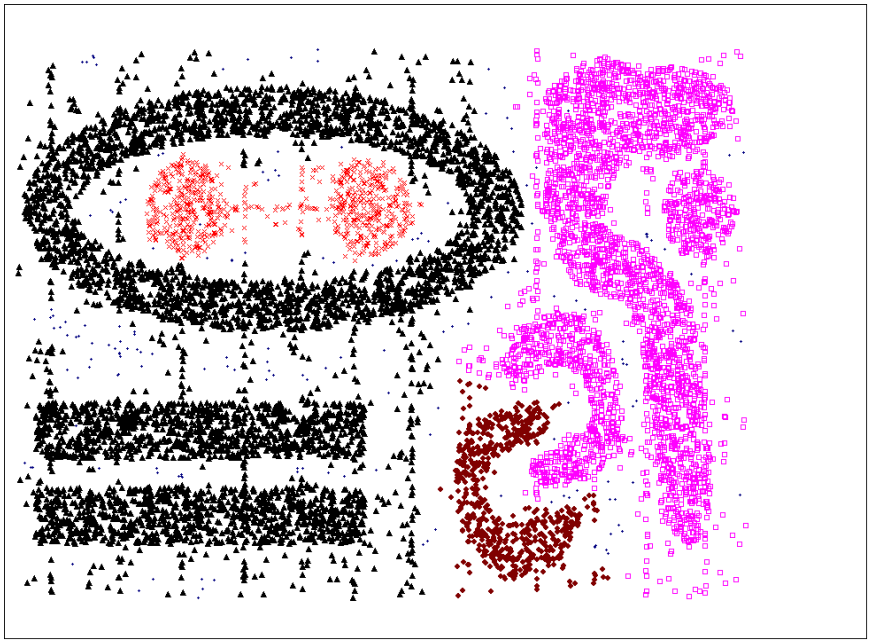
Jarvis Patrick Clustering

6 shared neighbors out of 20

When Jarvis-Patrick Does NOT Work Well



Smallest threshold, T, that does not merge clusters.



Threshold of T - 1

SNN Density-Based Clustering

- Combines:
  - Graph based clustering (similarity definition based on number of shared nearest neighbors)
  - Density based clustering (DBSCAN-like approach)

- SNN density measures whether a point is surrounded by similar points (with respect to its nearest neighbors)

SNN Clustering Algorithm

1. **Compute the similarity matrix**
   This corresponds to a similarity graph with data points for nodes and edges whose weights are the similarities between data points

2. **Sparsify the similarity matrix by keeping only the *k* most similar neighbors**
   This corresponds to only keeping the *k* strongest links of the similarity graph

3. **Construct the shared nearest neighbor graph from the sparsified similarity matrix.**
   At this point, we could apply a similarity threshold and find the connected components to obtain the clusters (Jarvis-Patrick algorithm)

4. **Find the SNN density of each Point.**
   Using a user specified parameters, *Eps*, find the number points that have an SNN similarity of *Eps* or greater to each point. This is the SNN density of the point

SNN Clustering Algorithm …

5. **Find the core points**
   Using a user specified parameter, *MinPts*, find the core points, i.e., all points that have an SNN density greater than *MinPts*

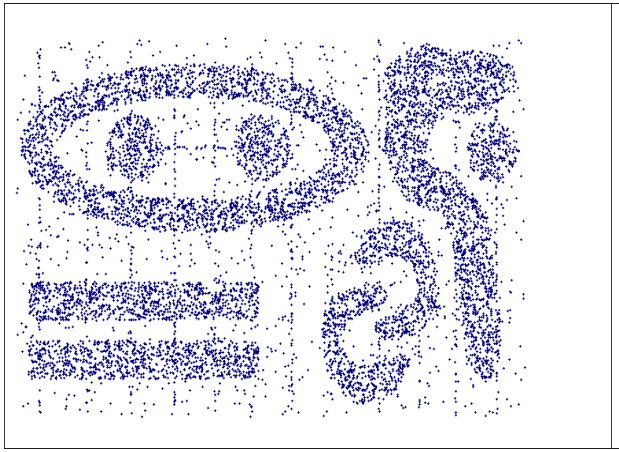
6. **Form clusters from the core points**
   If two core points are within a "radius", *Eps*, of each other they are placed in the same cluster

7. **Discard all noise points**
   All non-core points that are not within a "radius" of *Eps* of a core point are discarded

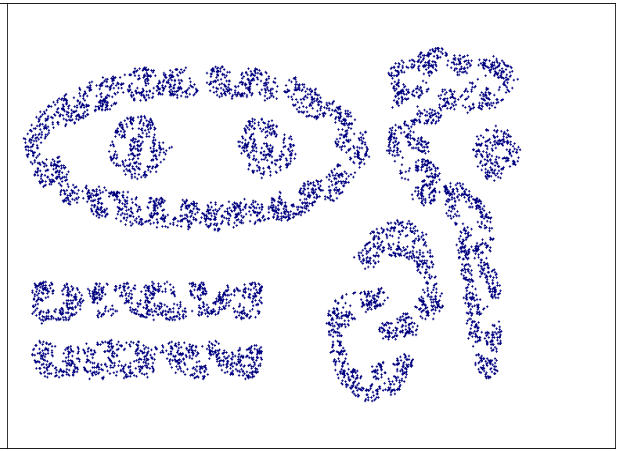
8. **Assign all non-noise, non-core points to clusters**
   This can be done by assigning such points to the nearest core point

   (Note that steps 4-8 are DBSCAN)

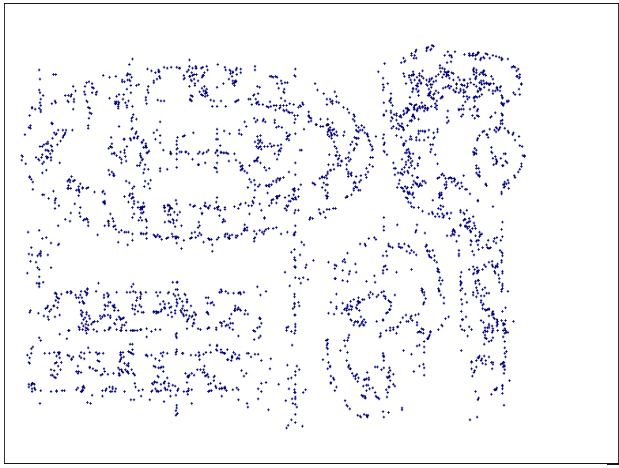
SNN Density



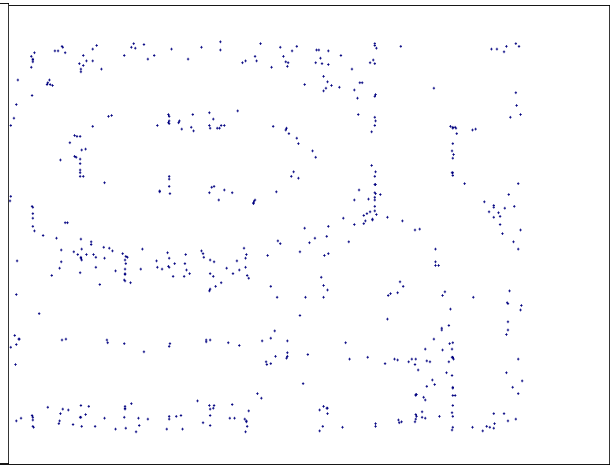
a) All Points



b) High SNN Density

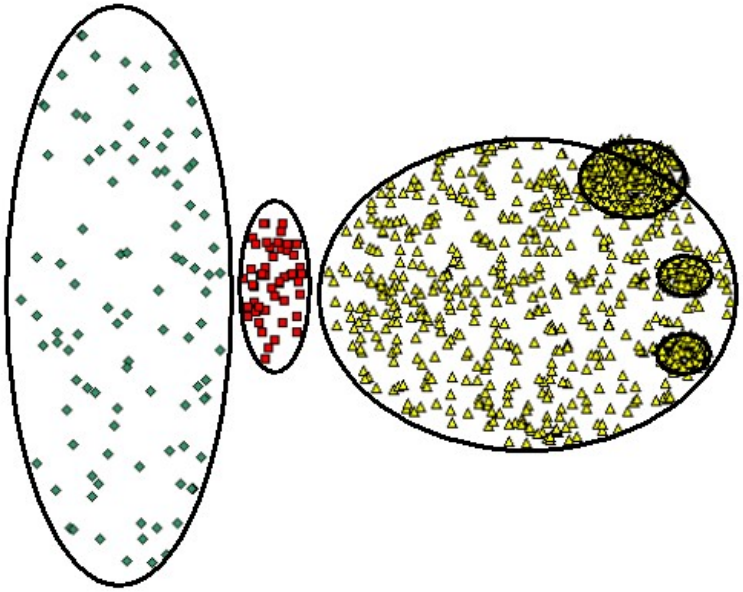


c) Medium SNN Density

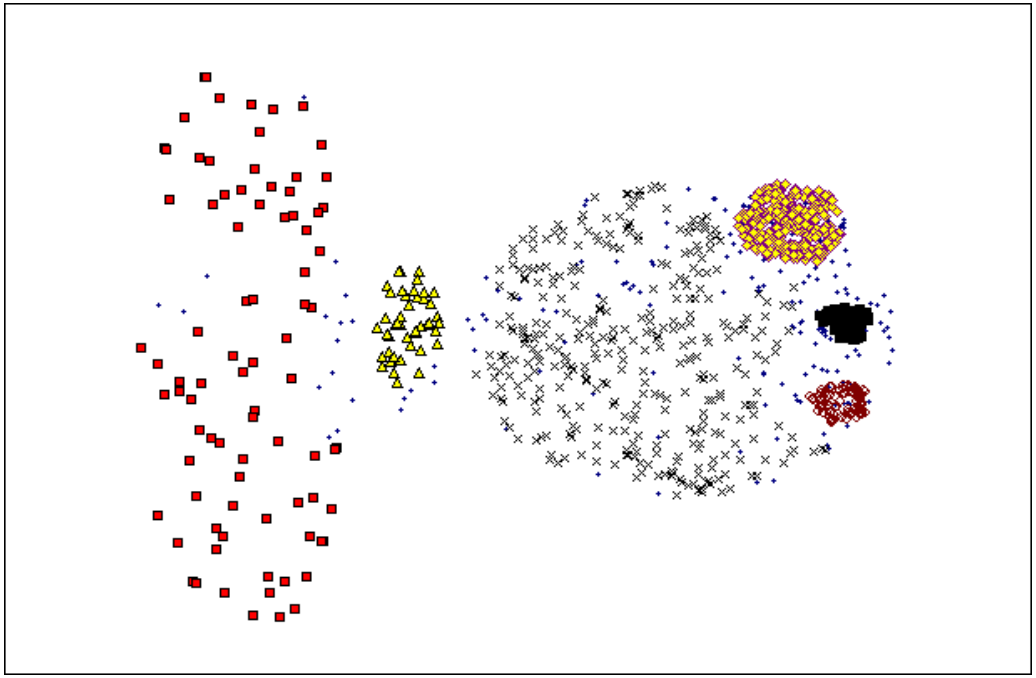


d) Low SNN Density

SNN Clustering Can Handle Differing Densities

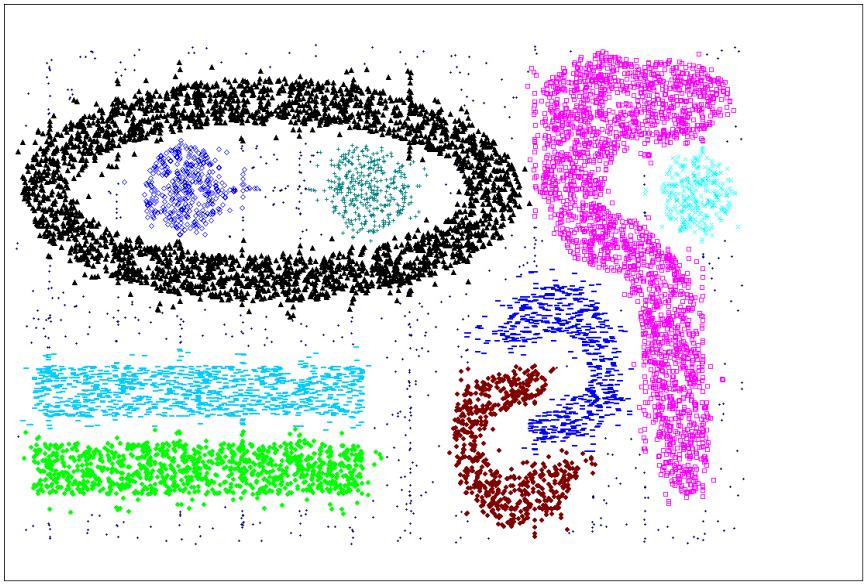


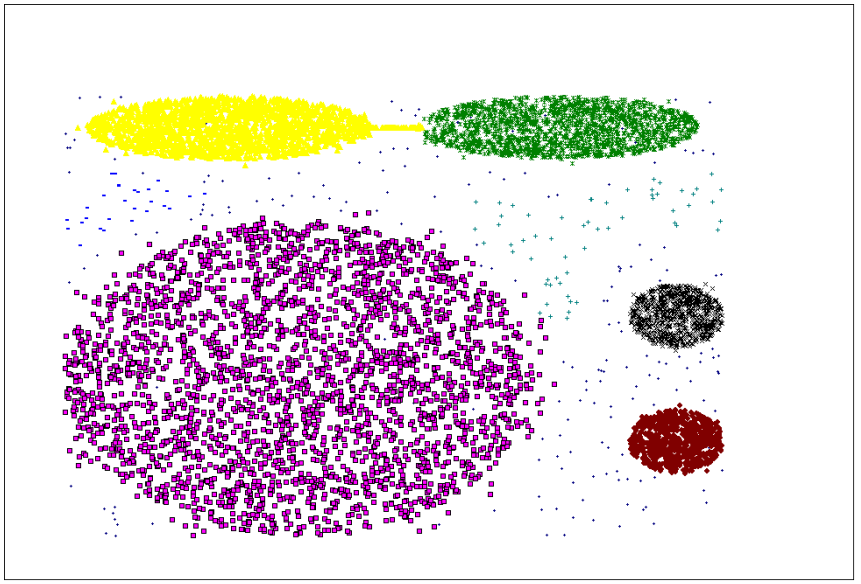
Original Points



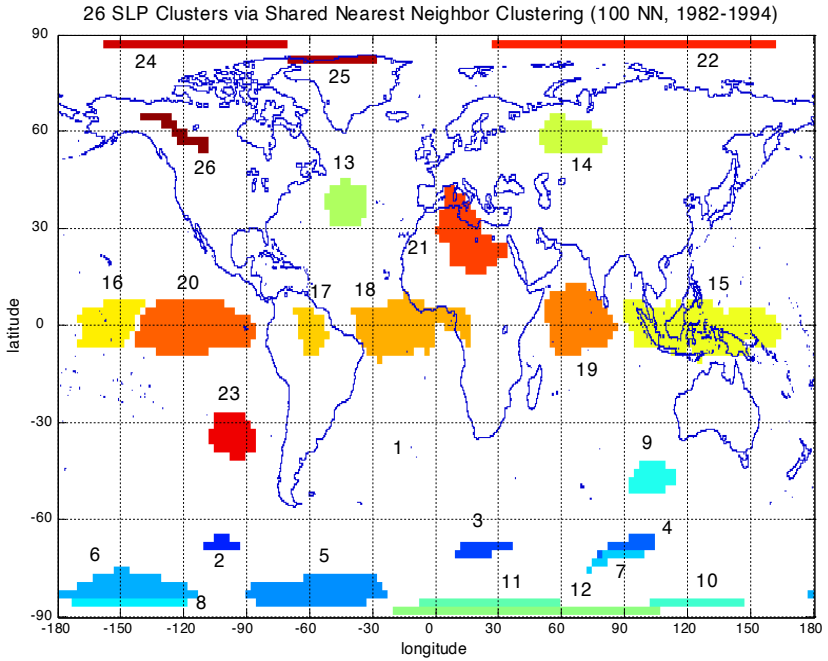
SNN Clustering

SNN Clustering Can Handle Other Difficult Situations

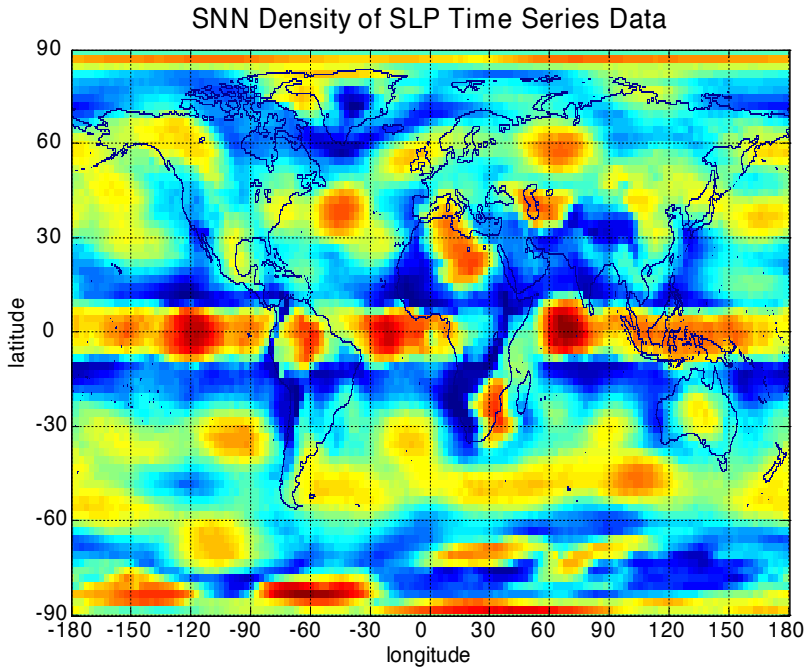




Finding Clusters of Time Series In Spatio-Temporal Data



SNN Clusters of SLP.



SNN Density of Points on the Globe.

Limitations of SNN Clustering

- Does not cluster all the points
- Complexity of SNN Clustering is high
  - O(n * time to find numbers of neighbor within *Eps*)
  - In worst case, this is $O(n^2)$
  - For lower dimensions, there are more efficient ways to find the nearest neighbors
    - R* Tree
    - k-d Trees
- Parameteriziation is not easy

Characteristics of Data, Clusters, and Clustering Algorithms

- A cluster analysis is affected by characteristics of
  - Data
  - Clusters
  - Clustering algorithms

- Looking at these characteristics gives us a number of dimensions that you can use to describe clustering algorithms and the results that they produce

Characteristics of Data

- High dimensionality
- Size of data set
- Sparsity of attribute values
- Noise and Outliers
- Types of attributes and type of data sets
- Differences in attribute scales
- Properties of the data space
  - Can you define a meaningful centroid

Characteristics of Clusters

- Data distribution
- Shape
- Differing sizes
- Differing densities
- Poor separation
- Relationship of clusters
- Types of clusters
  - Center-based, contiguity-based, density-based
- Subspace clusters

Characteristics of Clustering Algorithms

- Order dependence
- Non-determinism
- Parameter selection
- Scalability
- Underlying model
- Optimization based approach

Comparison of MIN and EM-Clustering

- *We assume EM clustering using the Gaussian (normal) distribution.*
- MIN is hierarchical, EM clustering is partitional.
- Both MIN and EM clustering are complete.
- MIN has a graph-based (contiguity-based) notion of a cluster, while EM clustering has a prototype (or model-based) notion of a cluster.
- MIN will not be able to distinguish poorly separated clusters, but EM can manage this in many situations.
- MIN can find clusters of different shapes and sizes; EM clustering prefers globular clusters and can have trouble with clusters of different sizes.
- Min has trouble with clusters of different densities, while EM can often handle this.
- Neither MIN nor EM clustering finds subspace clusters.

Comparison of MIN and EM-Clustering

- MIN can handle outliers, but noise can join clusters; EM clustering can tolerate noise, but can be strongly affected by outliers.
- EM can only be applied to data for which a centroid is meaningful; MIN only requires a meaningful definition of proximity.
- EM will have trouble as dimensionality increases and the number of its parameters (the number of entries in the covariance matrix) increases as the square of the number of dimensions; MIN can work well with a suitable definition of proximity.
- EM is designed for Euclidean data, although versions of EM clustering have been developed for other types of data. MIN is shielded from the data type by the fact that it uses a similarity matrix.
- MIN makes no distribution assumptions; the version of EM we are considering assumes Gaussian distributions.

Comparison of MIN and EM-Clustering

- EM has an O(n) time complexity; MIN is $O(n^2 \log(n))$.
- Because of random initialization, the clusters found by EM can vary from one run to another; MIN produces the same clusters unless there are ties in the similarity matrix.
- Neither MIN nor EM automatically determine the number of clusters.
- MIN does not have any user-specified parameters; EM has the number of clusters and possibly the weights of the clusters.
- EM clustering can be viewed as an optimization problem; MIN uses a graph model of the data.
- Neither EM or MIN are order dependent.

Comparison of DBSCAN and K-means

- Both are partitional.
- K-means is complete; DBSCAN is not.
- K-means has a prototype-based notion of a cluster; DB uses a density-based notion.
- K-means can find clusters that are not well-separated. DBSCAN will merge clusters that touch.
- DBSCAN handles clusters of different shapes and sizes; K-means prefers globular clusters.

Comparison of DBSCAN and K-means

- DBSCAN can handle noise and outliers; K-means performs poorly in the presence of outliers
- K-means can only be applied to data for which a centroid is meaningful; DBSCAN requires a meaningful definition of density
- DBSCAN works poorly on high-dimensional data; K-means works well for some types of high-dimensional data
- Both techniques were designed for Euclidean data, but extended to other types of data
- DBSCAN makes no distribution assumptions; K-means is really assuming spherical Gaussian distributions

Comparison of DBSCAN and K-means

- K-means has an O(n) time complexity; DBSCAN is O(n^2)
- Because of random initialization, the clusters found by K-means can vary from one run to another; DBSCAN always produces the same clusters
- DBSCAN automatically determines the number of clusters; K-means does not
- K-means has only one parameter, DBSCAN has two.
- K-means clustering can be viewed as an optimization problem and as a special case of EM clustering; DBSCAN is not based on a formal model.