UCI University of California, Irvine

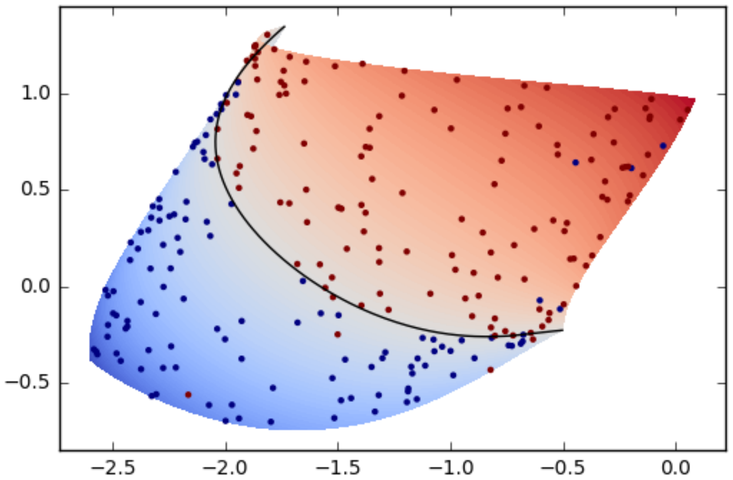
CS 273A: Machine Learning

Fall 2021

Lecture 15: Latent-Space Models

Roy Fox
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine

All slides in this course adapted from Alex Ihler & Sameer Singh



Logistics

assignments

- Assignment 5 due Tuesday, Nov 23

Today's lecture

k-Means

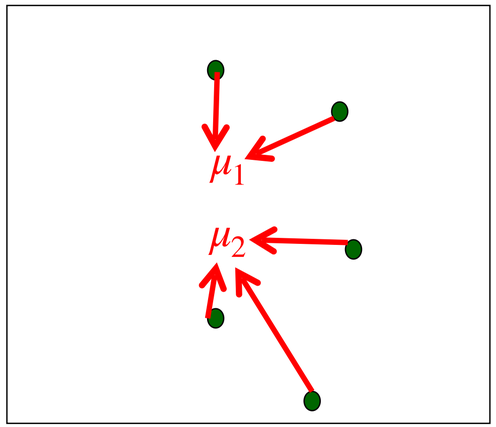
Agglomerative clustering

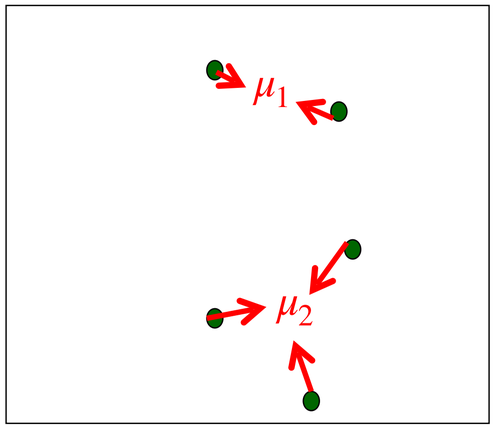
Gaussian Mixture Models

Latent-space models

k-Means

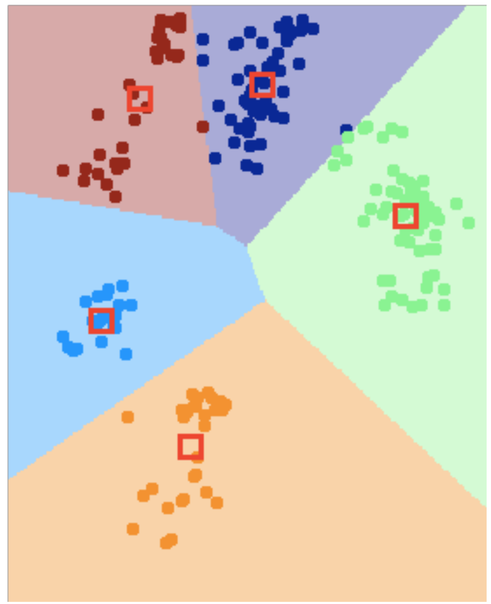
- Iterate until convergence:
  - For each $x_i \in \mathcal{D}$, find the closest cluster: $z_i = \arg\min_c \|x_i - \mu_c\|^2$
  - Set each cluster centroid μ_c to the mean of assigned points: $\mu_c = \frac{1}{m_c} \sum_{i:z_i=c} x_i$





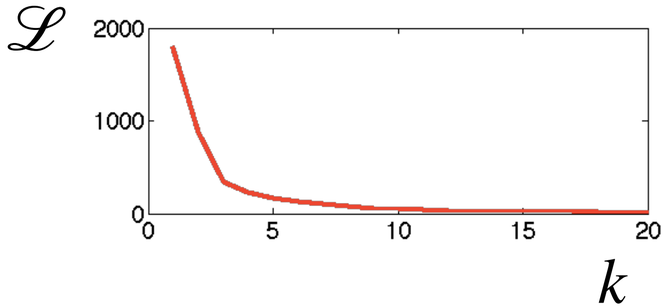
Out-of-sample clustering

- How can we use clustering to assign new data points?
- In k-Means: choose nearest centroid
  - 1-NN with learned centroids



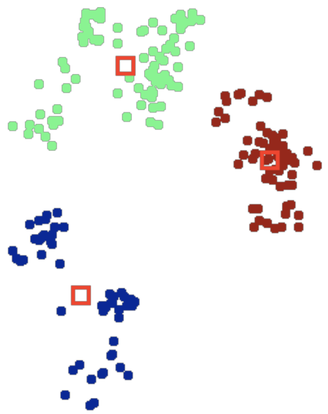
Choosing k

- How to choose the number of clusters k?

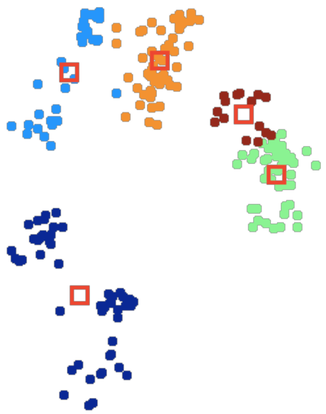


- More clusters $\Longrightarrow$ can make them closer to more points
  - $\Longrightarrow$ Loss $\mathscr{L}(z, \mu) = \sum_i \|x_i - \mu_{z_i}\|^2$ generally decreases with k (validation loss too...)
  - Larger $k \Longrightarrow$ larger model complexity

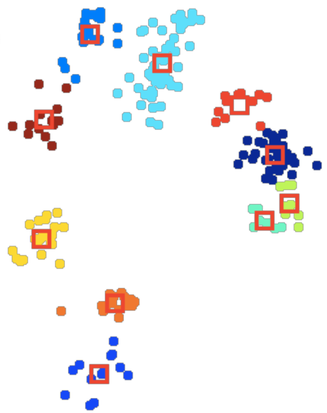
$k = 3$



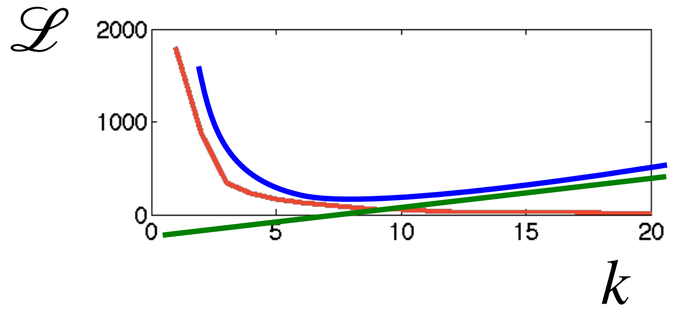
$k = 5$



$k = 10$



Choosing k



- How to choose the number of clusters k?
- More clusters $\implies$ can make them closer to more points
  - $\implies$ Loss $\mathscr{L}(z, \mu) = \sum_i \|x_i - \mu_{z_i}\|^2$ generally decreases with k (validation loss too...)
  - Larger $k \implies$ larger model complexity
- One solution: penalize complexity; loss = MSE + regularizer
  - More clusters may increase loss if they don't help much
  - Example: simplified BIC $\mathscr{L}(z, \mu) = \log\left(\frac{1}{mn}\sum_i \|x_i - \mu_{z_i}\|^2\right) + k\frac{\log m}{m}$

Recap: k-means

- Clusters represented as centroids in feature space
- Initialize centroids; repeat:
  - Assign each data point to its closest centroid
  - Move centroids minimize mean squared error (i.e. means of assigned points)
- Coordinate descent on MSE loss
- Prone to local optima; initialization important
- Can use to assign out-of-sample data
- Choosing k = #clusters: model selection; penalize for complexity (BIC, etc.)

Today's lecture

k-Means

Agglomerative clustering

Gaussian Mixture Models

Latent-space models

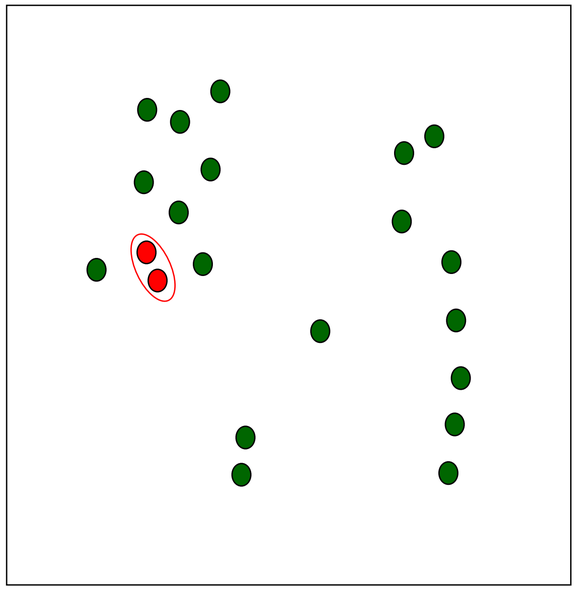
Hierarchical agglomerative clustering

- Another simple clustering algorithm
- Define distance (dissimilarity) between clusters $d(C_i, C_j)$
- Initialize: every data point is its own cluster
- Repeat:
  - Compute distance between each pair of clusters
  - Merge two closest clusters
- Output: tree of merge operations ("dendrogram")
- Complexity: in $m - 1$ iterations, merge distances and sort $\implies O(m^2 \log m)$

Iteration 1

- Build clustering hierarchically, bottom up (“agglomerative”)

data



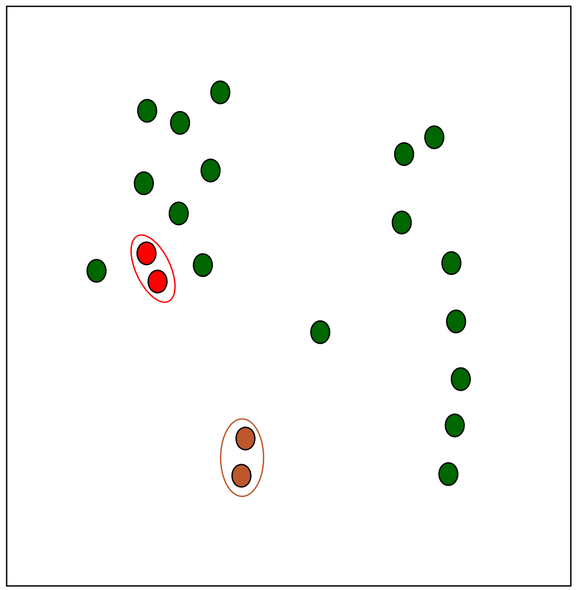
dendrogram

height of join indicates dissimilarity

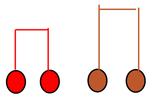
Iteration 2

- Build clustering hierarchically, bottom up (“agglomerative”)

data



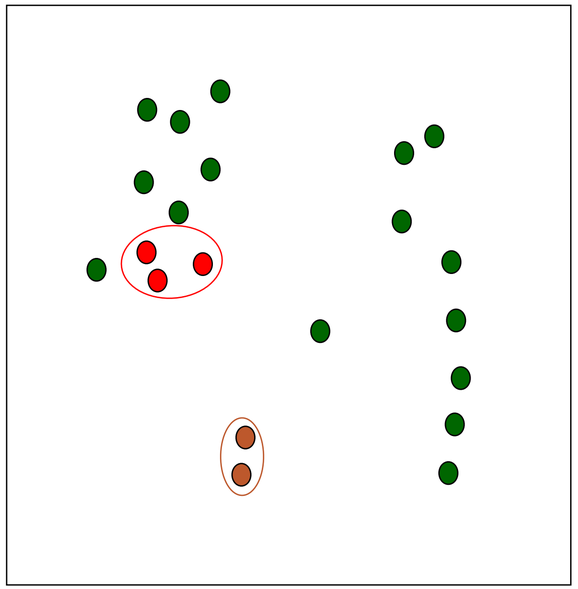
dendrogram



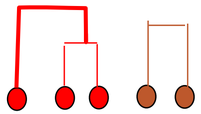
Iteration 3

- Build clustering hierarchically, bottom up (“agglomerative”)

data



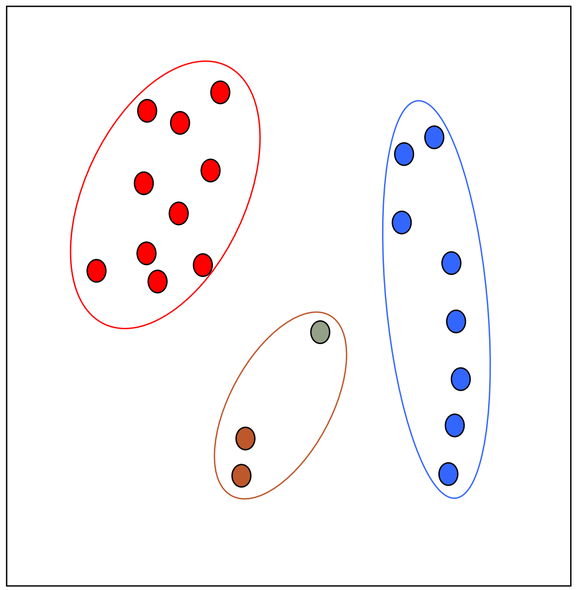
dendrogram



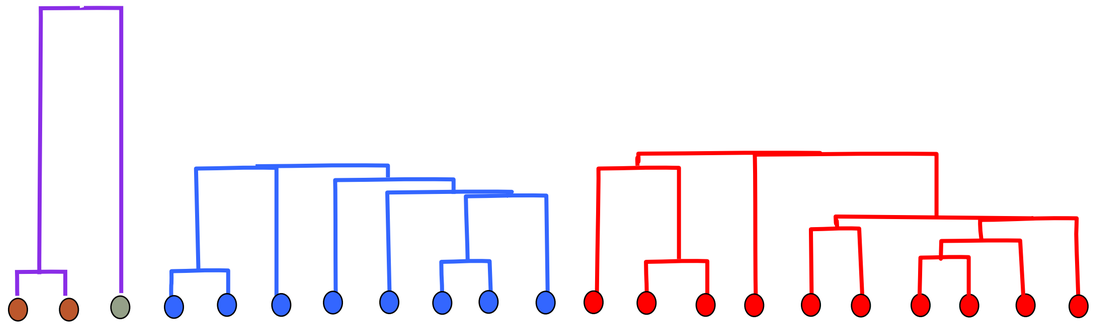
Iteration $m - 3$

- Build clustering hierarchically, bottom up ("agglomerative")

data



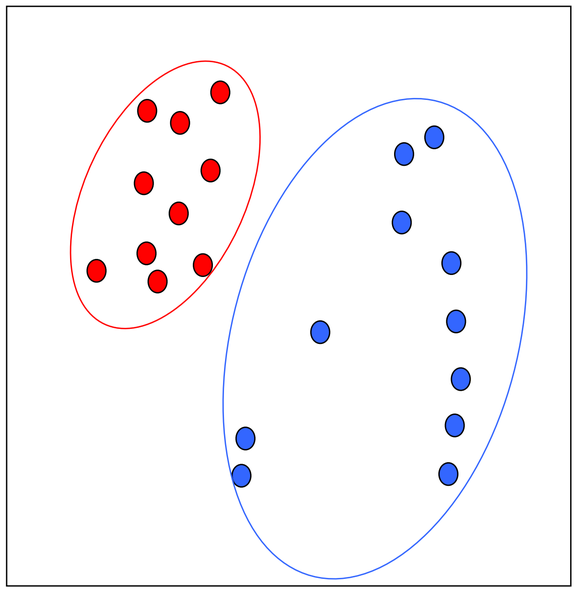
dendrogram



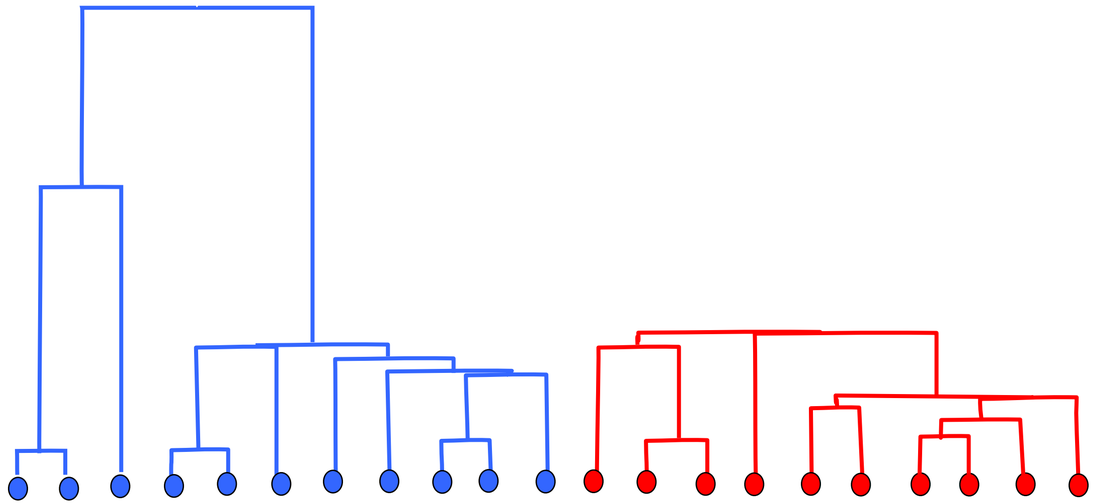
Iteration $m - 2$

- Build clustering hierarchically, bottom up (“agglomerative”)

data



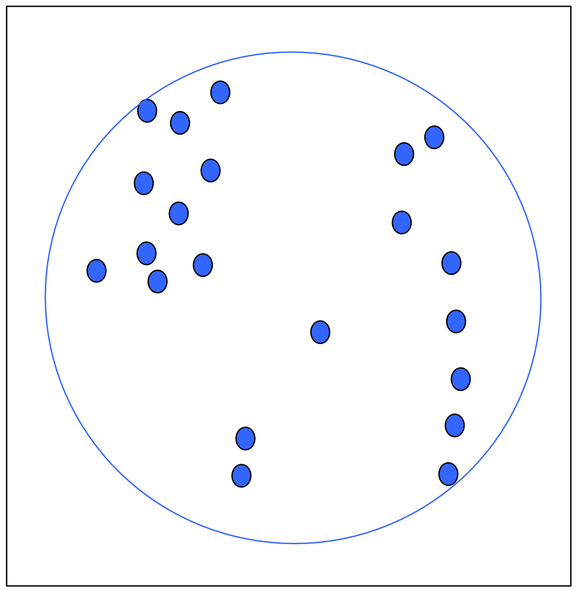
dendrogram



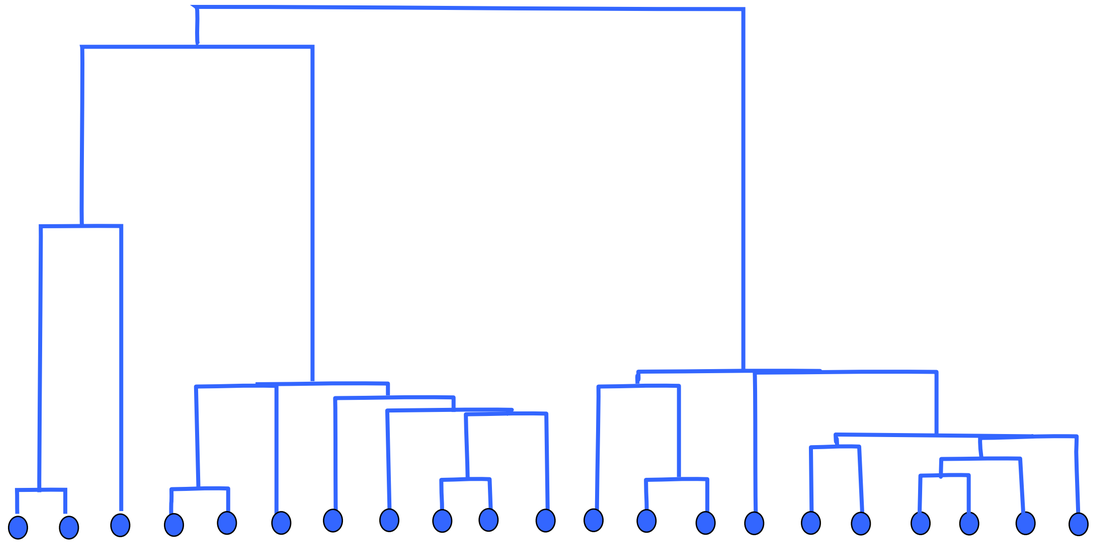
Iteration $m - 1$

- Build clustering hierarchically, bottom up (“agglomerative”)

data



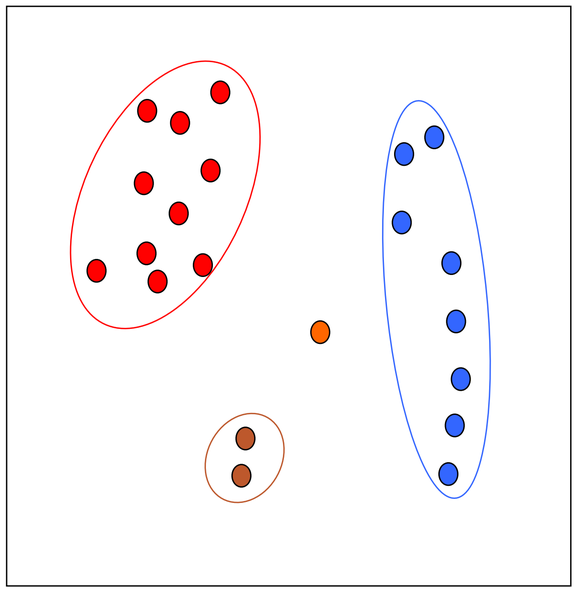
dendrogram



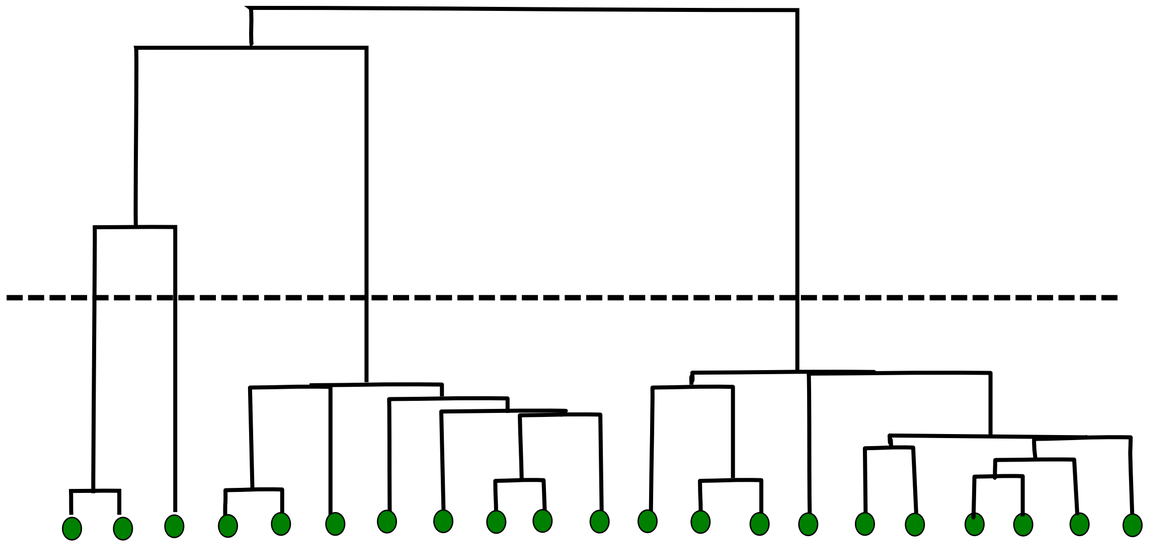
From dendrogram to clusters

- Given the hierarchy of clusters, choose a frontier of subtrees = clusters

data



dendrogram

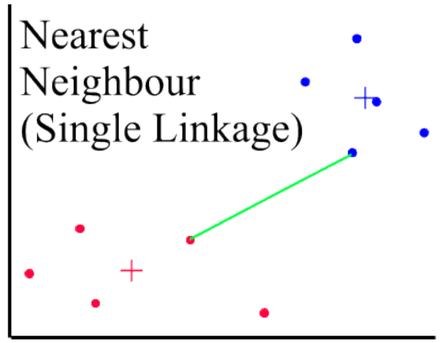


  - For a given k, or a given level of dissimilarity

Distance measures

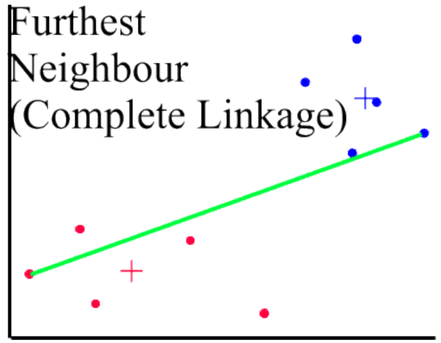
- $d_{\min}(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|^2$

produces minimum spanning tree



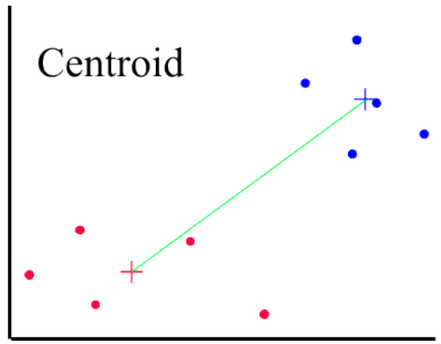
- $d_{\max}(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\|^2$

avoids elongated clusters



- $d_{\text{avg}}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i, y \in C_j} \|x - y\|^2$

- $d_{\text{means}}(C_i, C_j) = \|\mu_i - \mu_j\|^2$

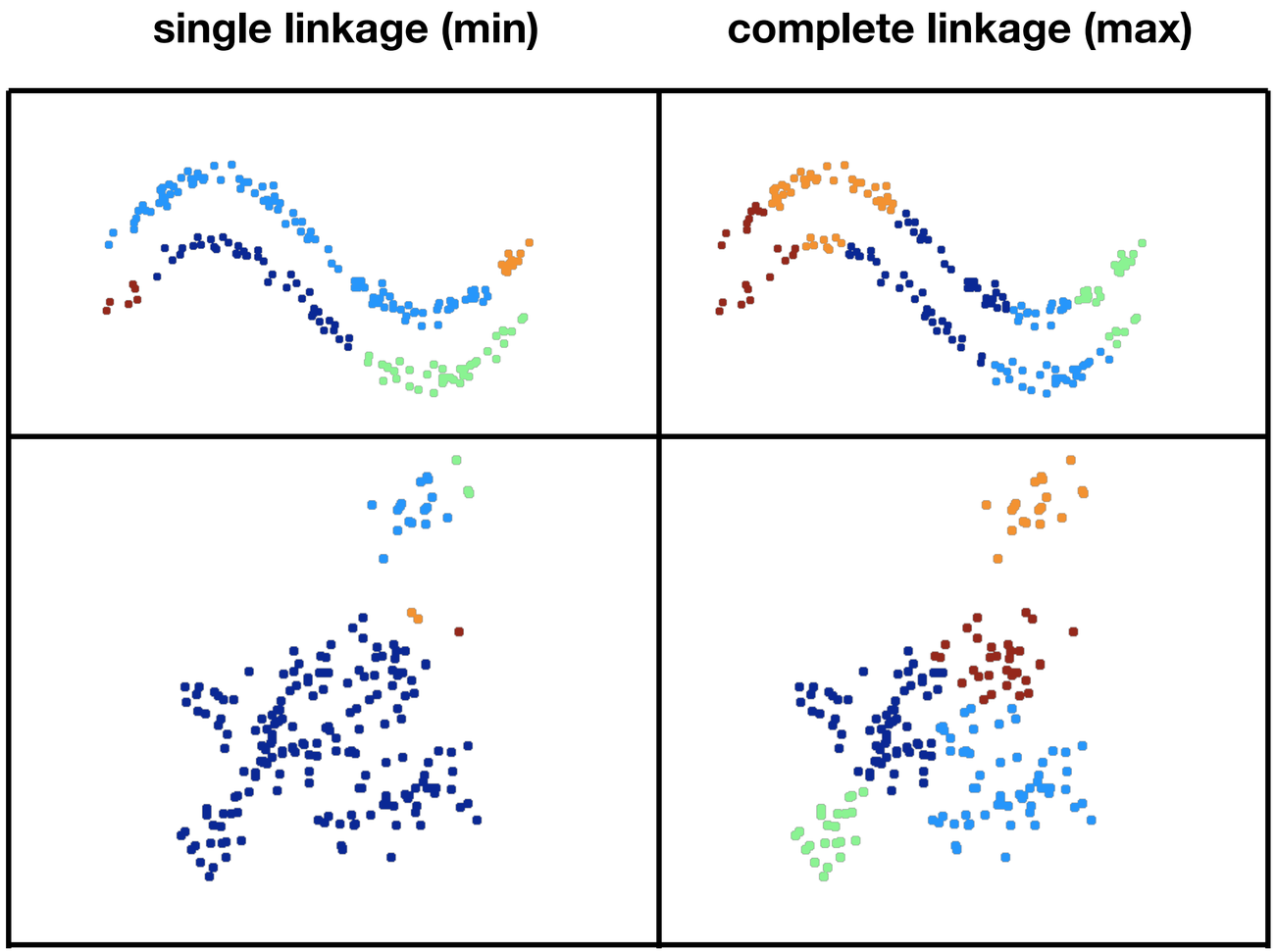


- Important property: iterative computation

$$d(C_i \cup C_j, C_k) = f(d(C_i, C_k), d(C_j, C_k))$$

Distance measures

- Dissimilarity measure affects the clustering qualitatively



Recap: agglomerative clustering

- Hierarchical clustering: build “dendrogram”
  - Bottom-up: agglomerative clustering
- Successively merge closest pair of clusters
  - Dendrogram = tree of merges & distances
  - Complexity = $O(m^2 \log m)$
- Clusters quality depend on choice of a distance / dissimilarity measure

Today's lecture

k-Means

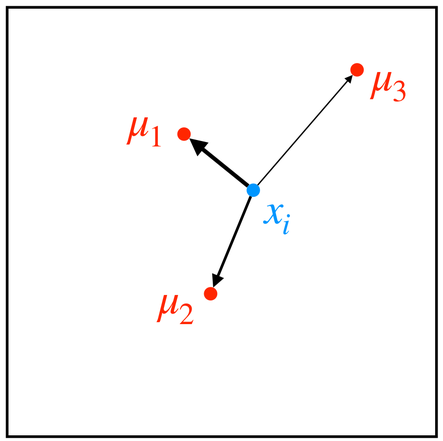
Agglomerative clustering

Gaussian Mixture Models

Latent-space models

Mixture Models

- k-Means assigns each instance to one cluster
  - Could it be assigned to another cluster equally well? Almost equally?
  - Hard assignment $f : x \mapsto c$ loses information on:
    - Which clusters are "close seconds"
    - Uncertainty = how sure are we of the assignment

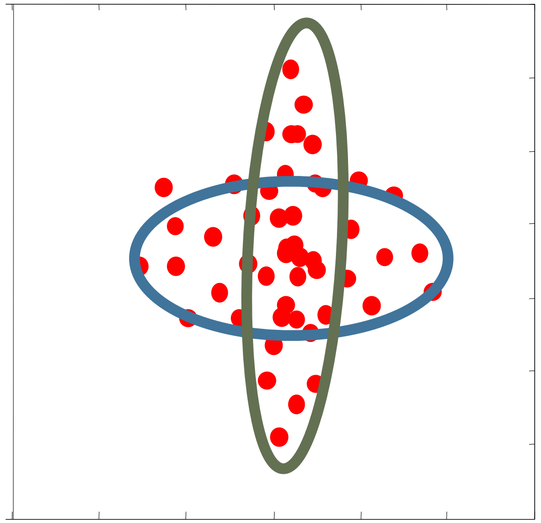


- Mixture Model = prior over clusters $p(c)$ + distribution in each cluster $p(x \mid c)$
  - $\Longrightarrow$ Posterior $p(c \mid x)$ = probabilistic (soft) assignment of x to c

Gaussian Mixture Models (GMMs)

- Each cluster is modeled by a Gaussian $p(x \mid c) = \mathcal{N}(x; \mu_c, \Sigma_c)$
  - Σ_c allows non-isotropic clusters $\Longrightarrow$ weighted Euclidean distance
- Mixture = distribution over Gaussians is given by a probability vector $p(c)$
- Generative model = we can sample $p(x)$:
  - Sample $z \sim p(c)$
  - Sample $x \sim p(x \mid c = z)$
  - Probability of this x: $\sum_c p(c = z)p(x \mid c = z) = \sum_c p(c, x) = p(x)$

we don't output z, it is "latent" = hidden $\Longrightarrow$ can be any of them



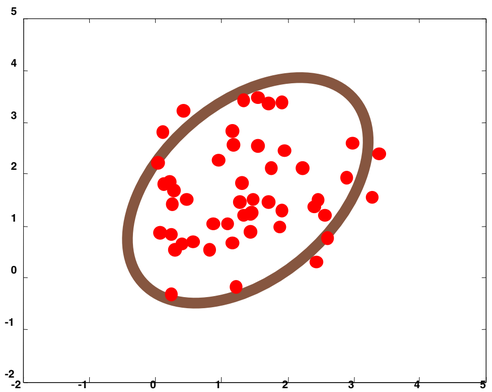
Multivariate Gaussian distributions

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

- For data points $\{x_i\}$, maximum log-likelihood estimator of μ, Σ:

$$\nabla_\mu \sum_i \log \mathcal{N}(x_i;\mu,\Sigma) = \frac{1}{2}\sum_i (x_i-\mu)^\intercal\Sigma^{-1} = 0$$

$$\implies \mu = \frac{1}{m}\sum_i x_i$$



$$\nabla_{\Sigma^{-1}} \sum_i \log \mathcal{N}(x_i;\mu,\Sigma) = -\frac{1}{2}\sum_i \left((x_i-\mu)(x_i-\mu)^\intercal - \Sigma\right) = 0$$

$$\implies \Sigma = \frac{1}{m}\sum_i (x_i-\mu)(x_i-\mu)^\intercal$$

matrix calculus identity:
$\nabla_{\Sigma^{-1}}\log|\Sigma|^{-1} = \Sigma$

Training GMMs

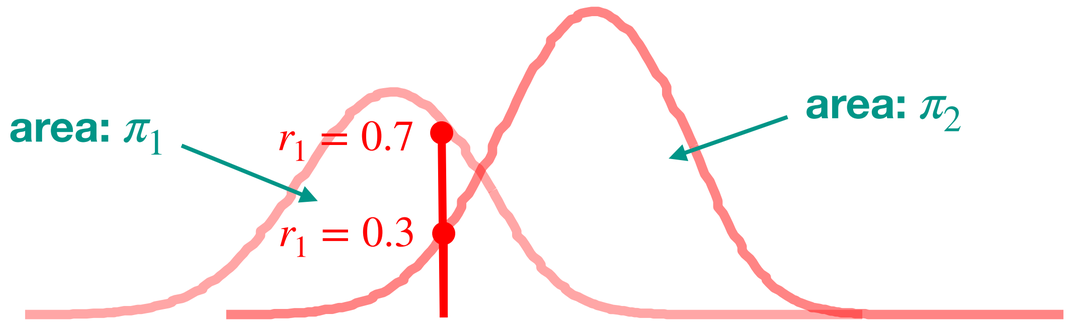
- k-Means:
  - Assign data points to clusters z_i
  - Update each cluster's parameters μ_c
- A “soft” version of k-Means: Expectation–Maximization (EM) algorithm
  - Find a “soft” assignment $p(c \mid x)$
  - Update model parameters $p(c)$, $p(x \mid c)$
- The EM algorithm is extremely general, GMMs are a very special case

Expectation–Maximization: E-step

- Initialize model parameters $\pi_c = p(c)$, μ_c, Σ_c
- E-step (Expectation): [why “expectation”? comes from the general EM algorithm]
  - For each data point x_i, use Bayes' rule to compute:

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

  - High weight to clusters that are likely a-priori, or in which x_i is relatively probable



Expectation–Maximization: M-step

- Given assignment probabilities r_{ic}
- M-step (Maximization):
  - For each cluster c, fit the best Gaussian to the weighted assignment

total weight assigned to cluster c

$$m_c = \sum_i r_{ic}$$

what is $\sum_c m_c$? m

fraction of weight assigned to cluster c

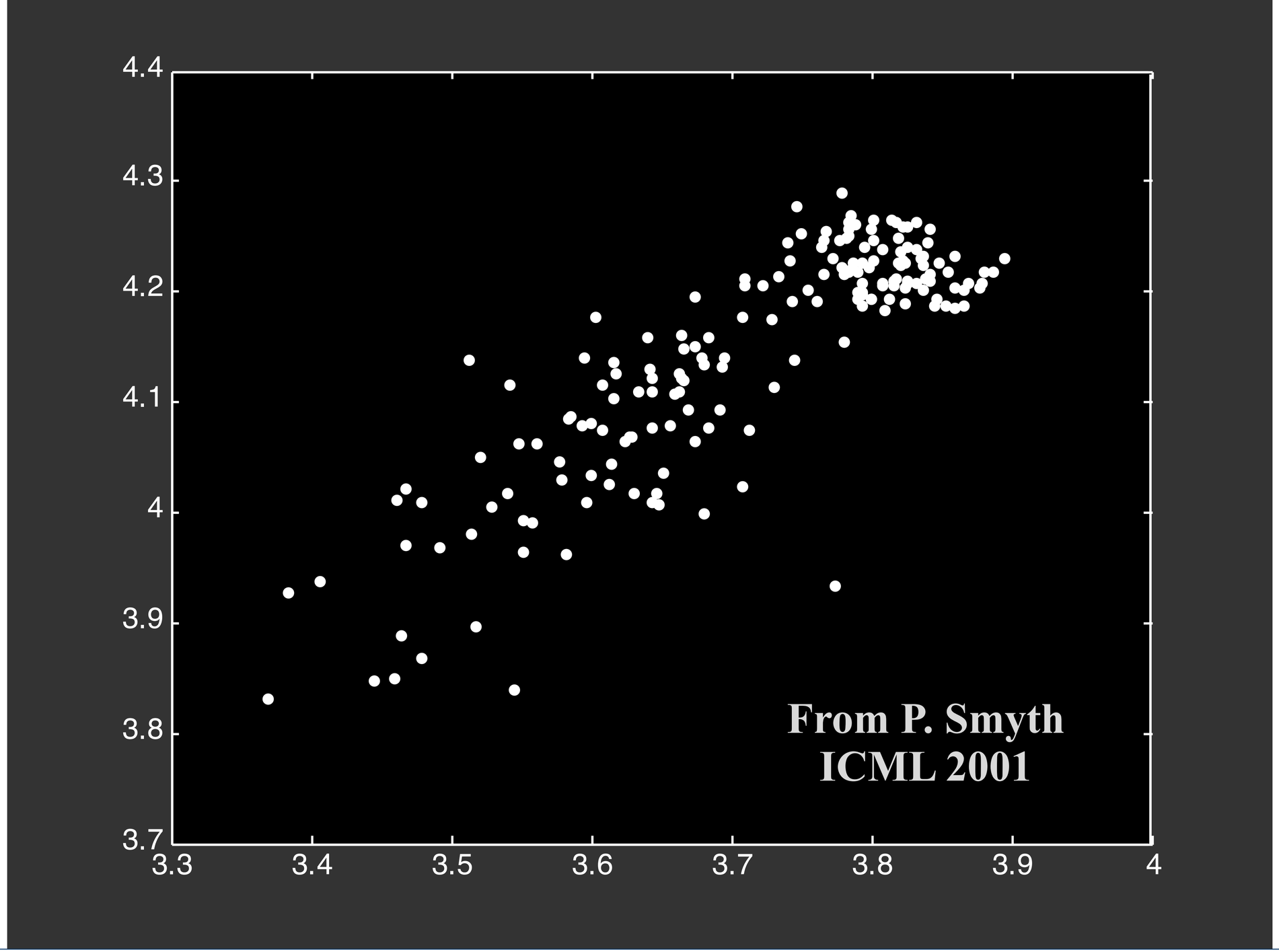
$$\pi_c = \frac{m_c}{m}$$

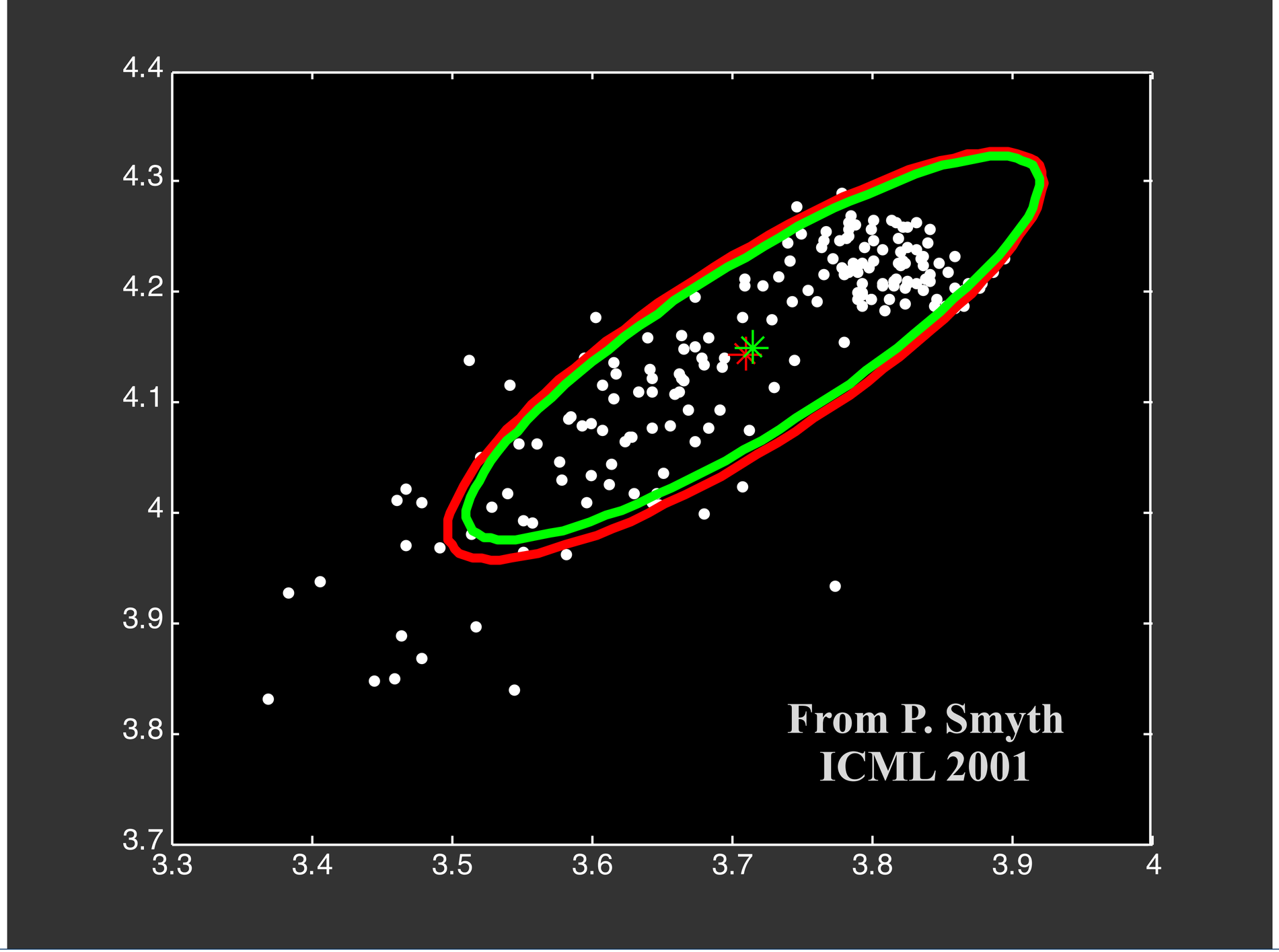
$$\mu_c = \frac{1}{m_c} \sum_i r_{ic} x_i$$

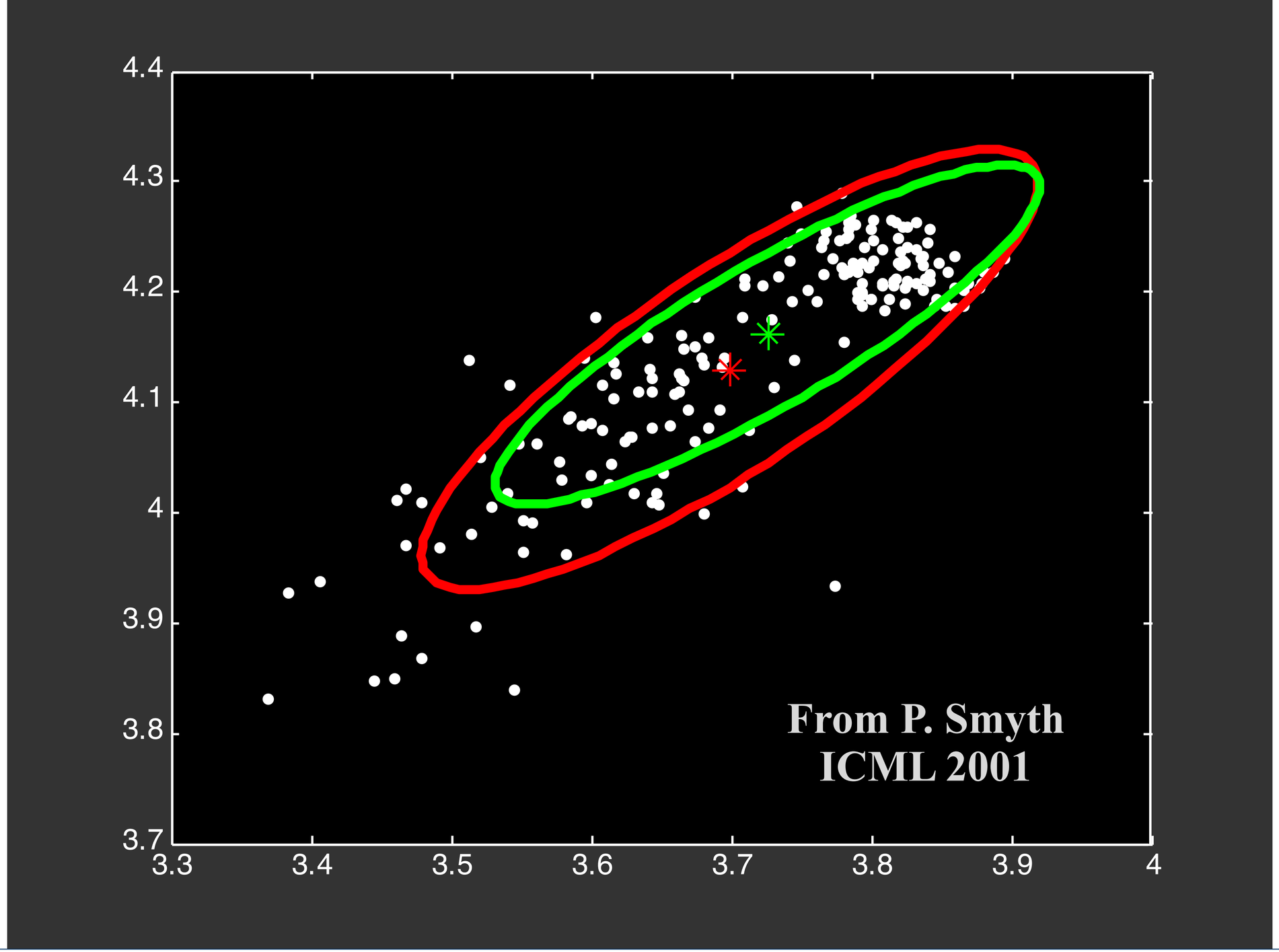
weighted mean of data in cluster c

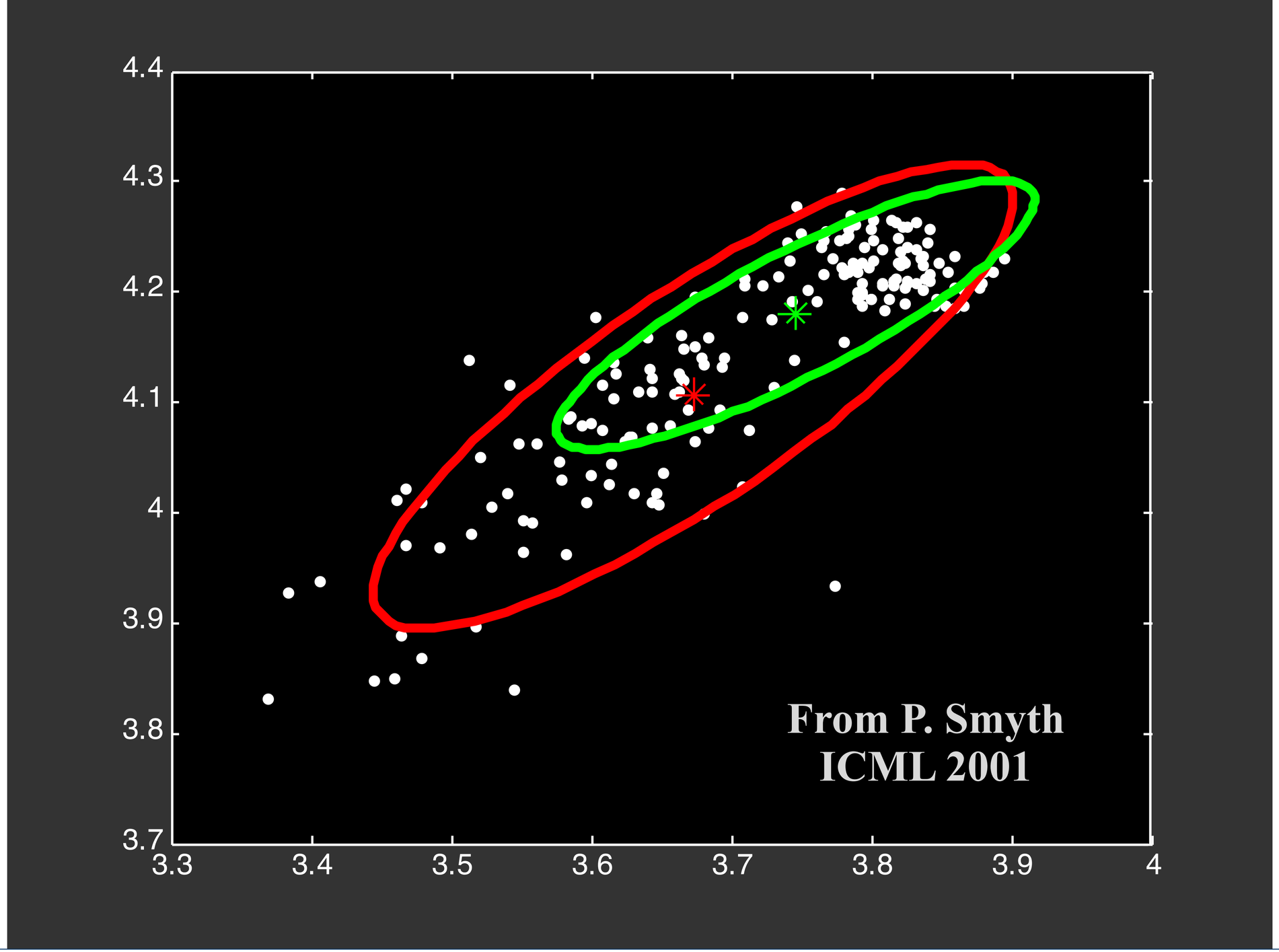
$$\Sigma_c = \frac{1}{m_c} \sum_i r_{ic} (x_i - \mu_c)(x_i - \mu_c)^\intercal$$

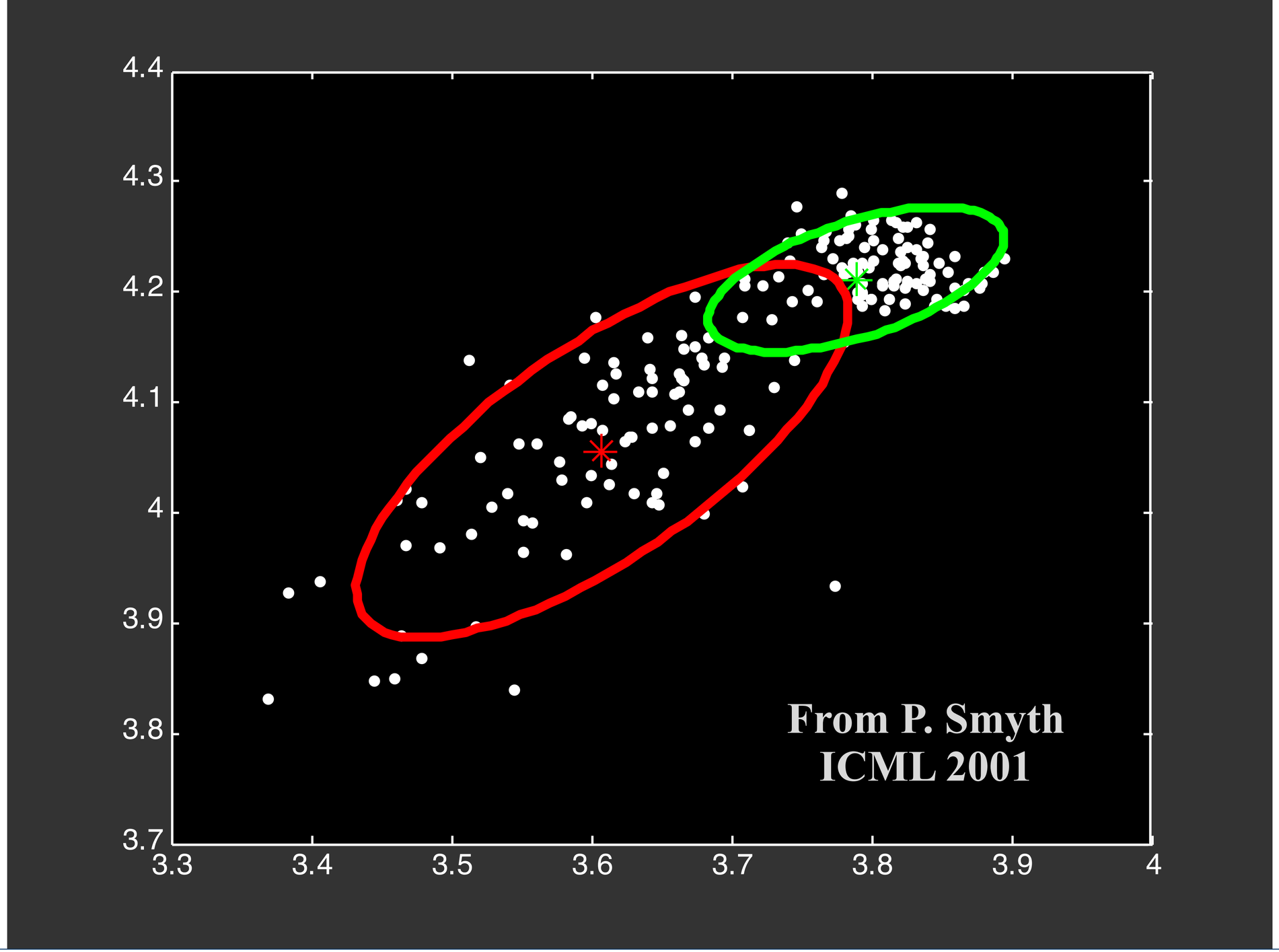
weighted covariance of data in cluster c

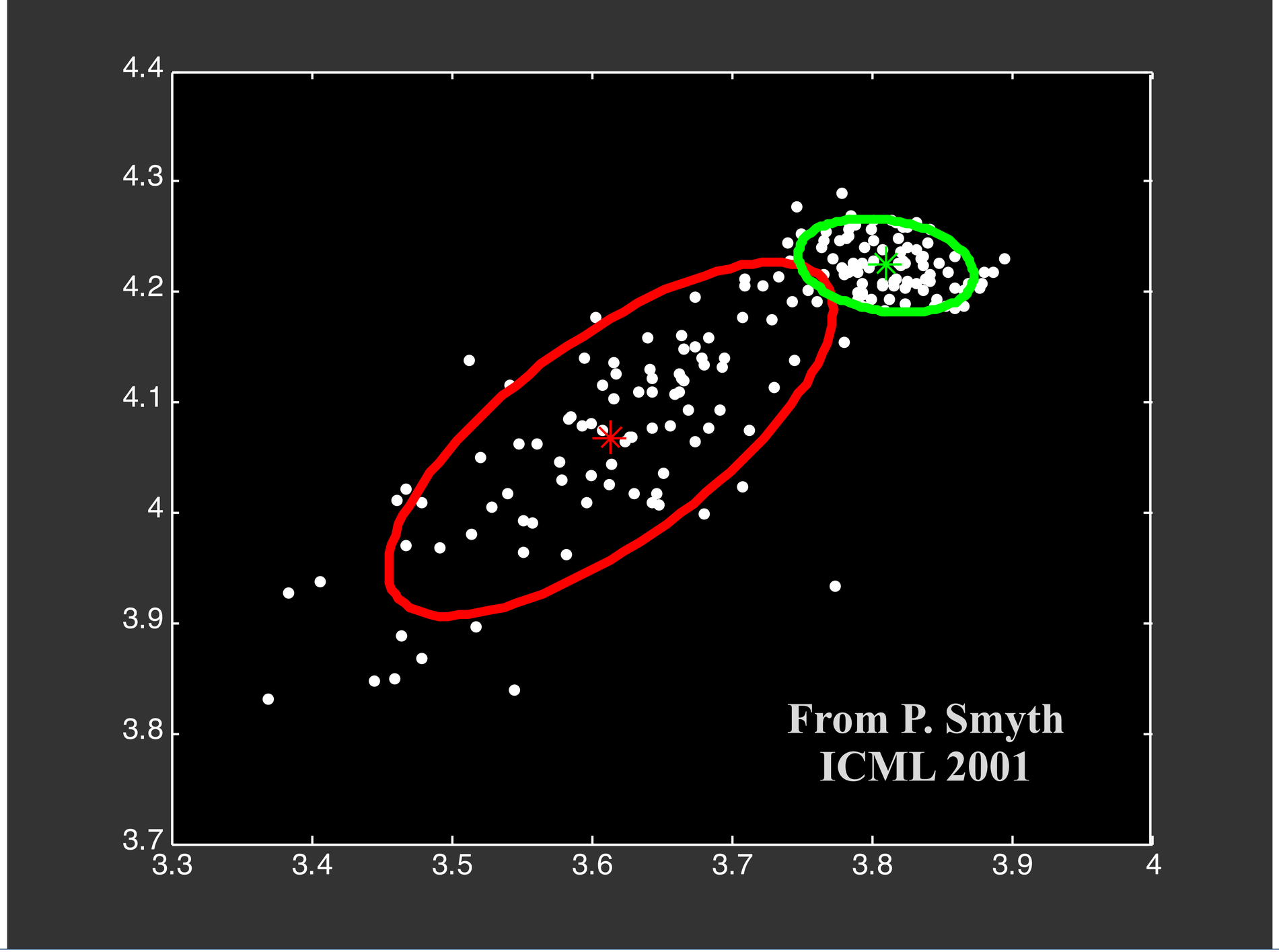

4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001

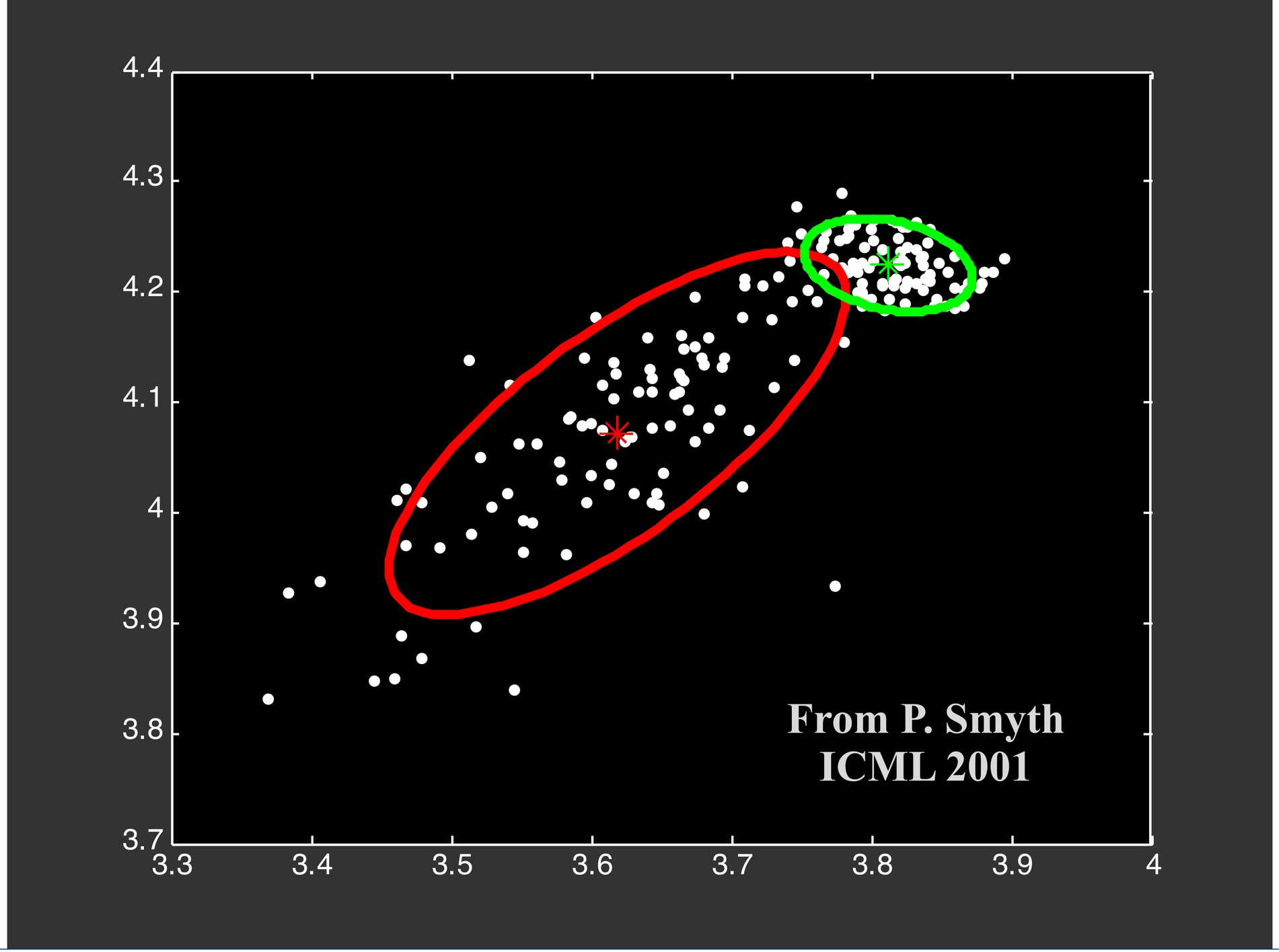

4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001

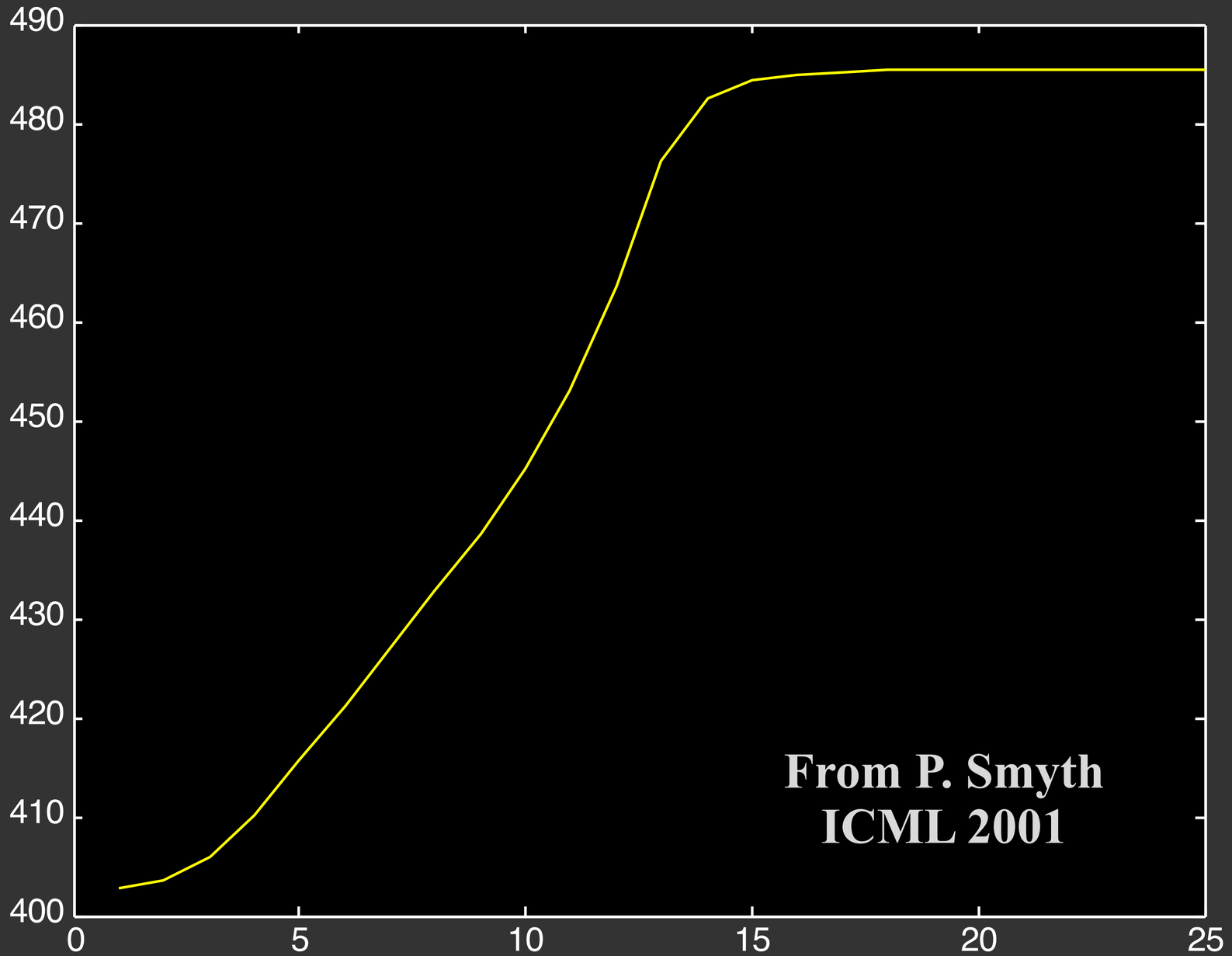

4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


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


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


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


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


490
480
470
460
450
440
430
420
410
400
0
5
10
15
20
25
From P. Smyth
ICML 2001

Demo

- https://lukapopijac.github.io/gaussian-mixture-model/

Expectation–Maximization: considerations

- Each iteration of EM is guaranteed to increase the data log likelihood

$$\log p(\mathscr{D}) = \sum_i \log p(x_i) = \sum_i \log \sum_c \pi_c \mathcal{N}(x_i; \mu_c, \Sigma_c)$$

we won't show this
but proof is very insightful!

  - Convergence guaranteed — descends NLL

    - But could be local optima $\Longrightarrow$ initialization important

- Out-of-sample data: can find soft assignment = probabilistic prediction

- Choosing #clusters: regularized training log-likelihood (as in k-Means)

  - Or: validate log-likelihood on held out data; many clusters $\Longrightarrow$ overfitting!

Recap

- Gaussian Mixture Models (GMMs)
  - Expressive class of generative models $p(x)$
  - Explain variation with latent clusters + cluster distribution
  - Given cluster (= mode), feature values are Gaussian
- Expectation–Maximization (EM)
  - Compute soft assignment probabilities, “responsibility” r_{ic}
  - Update model parameters: mixture π_c, cluster mean and covariance μ_c, Σ_c
  - Ascent on log-likelihood: convergent, but local optima
- Selecting the number of clusters
  - Regularized training log-likelihood, or validation log-likelihood

Today's lecture

k-Means

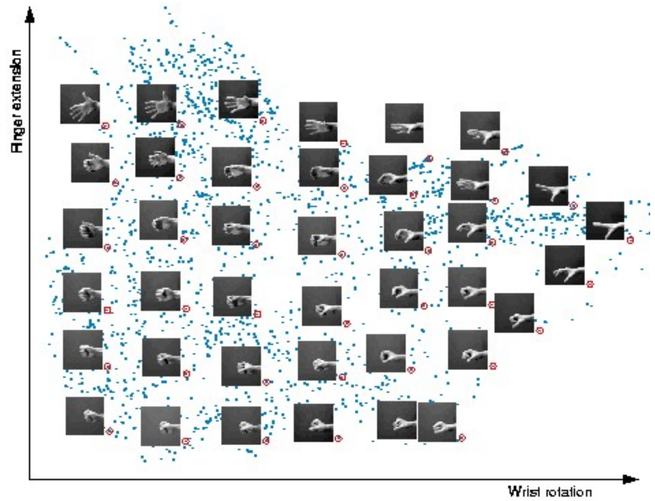
Agglomerative clustering

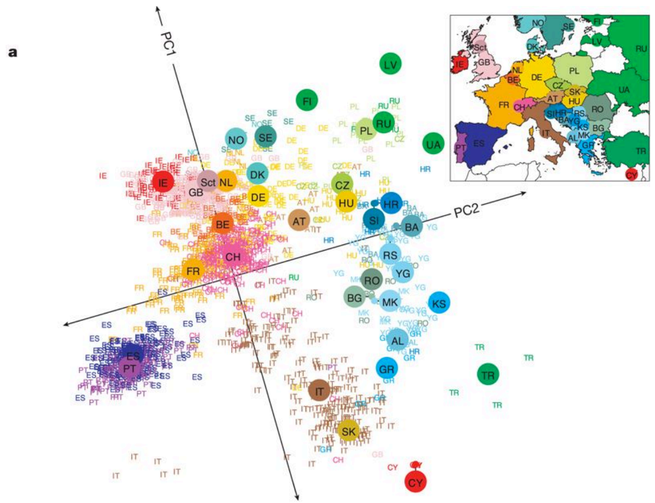
Gaussian Mixture Models

Latent-space models

Why reduce dimensionality?

- Data is often high-dimensional = many features
  - Images (even at 28x28 pixels)
  - Text (even a “bag of words”)
  - Stock prices (e.g. S&P500)
- Issues with high-dimensionality:
  - Computational complexity of analyzing the data
  - Model complexity (more parameters)
  - Sparse data = cannot cover all combinations of features
  - Correlated features can be independently noisy
  - Hard to visualize





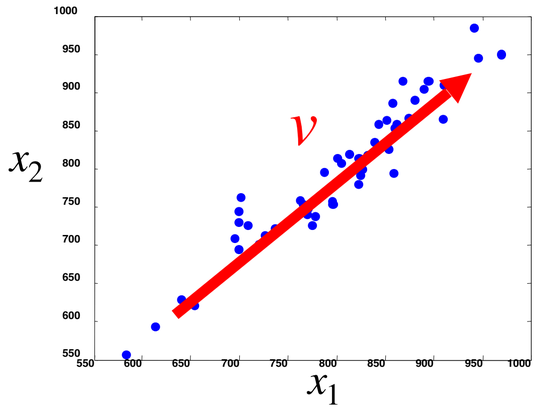
Dimensionality reduction

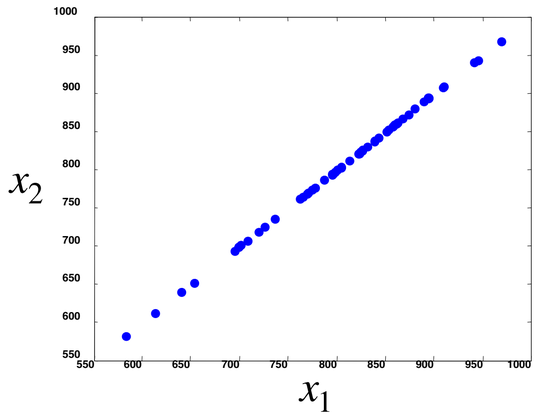
- With many features, some tend to change together
  - Can be summarized together
  - Others may have little or irrelevant change
- Example: S&P500 → "Tech stocks up 2x, manufacturing up 1.5x, …"
- Embed instances in lower-dimensional space $f : \mathbb{R}^n \mapsto \mathbb{R}^d$
  - Keep dimensions of "interesting" variability of data
  - Discard dimensions of noise or unimportant variability; or no variability at all
  - Keep "similar" data close $\implies$ may preserve cluster structure, other insights

Linear features

- Example: summarize two real features $x = [x_1, x_2] \rightarrow$ one real feature z
  - If z preserves much information about x, should be able to find $x \approx f(z)$
- Linear embedding:
  - $x \approx zv$
  - zv should be the closest point to x along v
    - $z = \arg\min \|x - zv\|^2 \implies z = \dfrac{x^\intercal v}{v^\intercal v}$

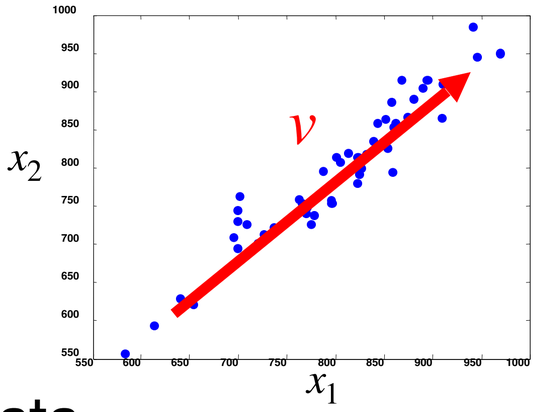
projection of x on v





Principal Component Analysis (PCA)

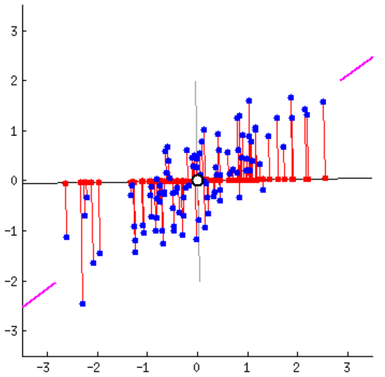
- How to find a good v?
  - Assume X has mean 0; otherwise, subtract the mean $\tilde{X} = X - \mu$
  - Idea: find the direction v of maximum "spread" (variance) of the data
  - Project $\tilde{X}$ on v: $z = \tilde{X}v$



empirical covariance

$$\max_{v:\|v\|=1} \sum_i (z_i)^2 = z^\intercal z = v^\intercal \tilde{X}^\intercal \tilde{X} v \implies v \text{ is eigenvector of } \tilde{X}^\intercal \tilde{X} \text{ of largest eigenvalue}$$

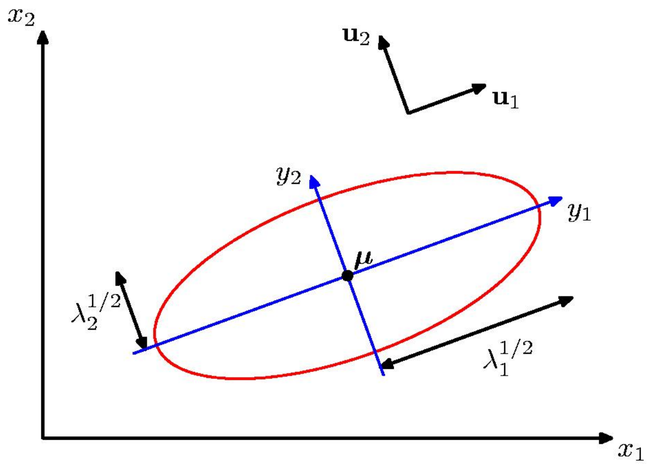
  - = minimum MSE of the residual $\tilde{X} - zv^\intercal = \tilde{X} - \tilde{X}vv^\intercal$



Source

Geometry of a Gaussian

- Data covariance: $\Sigma = \frac{1}{m}\tilde{X}^\intercal\tilde{X} \qquad \tilde{X} = X - \mu$
- Gaussian fit: $p(x) \sim \mathcal{N}(\mu, \Sigma)$
- Value contour for $p(x)$: $\Delta^2 = (x-\mu)^\intercal\Sigma^{-1}(x-\mu) = \text{const}$
- It's always possible to write Σ in terms of its eigenvectors U, eigenvalues λ:
  - $\Sigma = U\Lambda U^\intercal = \sum_{i=1}^{n} \lambda_i u_i u_i^\intercal \Longrightarrow \Sigma^{-1} = \sum_{i=1}^{n} \frac{1}{\lambda_i} u_i u_i^\intercal$
  - In the eigenvector basis: $\Delta^2 = \sum_{i=1}^{n} \frac{y_i^2}{\lambda_i}$, with $y_i = u_i^\intercal(x-\mu)$



PCA representation

- Subtract data mean from data points
- (Optional) Scale each dimension by its variance
  - Don't just focus on large-scale features (e.g., +1 mileage $\ll$ +1yr ownership)
  - Focus on correlation between features
- Compute empirical covariance matrix $\Sigma = \frac{1}{m} \sum_i \tilde{x}_i \tilde{x}_i^\intercal$
- Take k largest eigenvectors of $\Sigma = U \Lambda U^\intercal$

Singular Value Decomposition (SVD)

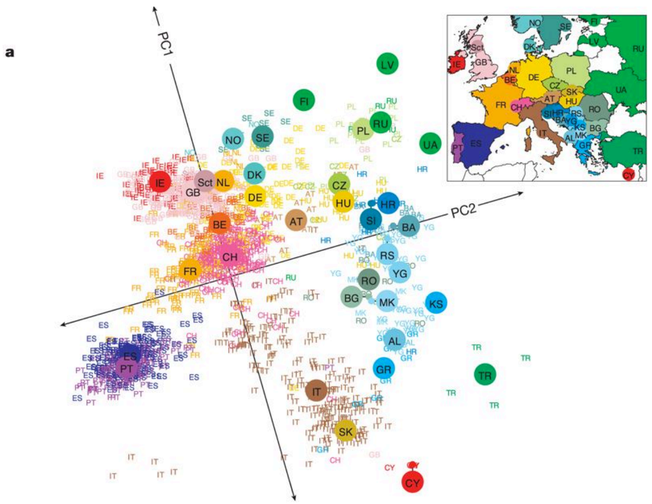
- Alternative method for finding covariance eigenvectors
  - Has many other uses
- Singular Value Decomposition (SVD): $X = UDV^\intercal$
  - U and V (left- and right singular vectors) are orthogonal: $U^\intercal U = I$, $V^\intercal V = I$
  - D (singular values) is rectangular-diagonal
  - $\Sigma = X^\intercal X = VD^\intercal U^\intercal UDV^\intercal = V(D^\intercal D)V^\intercal$

$$\underset{m \times n}{X} \approx \underset{m \times k}{U} \cdot \underset{k \times k}{D} \cdot \underset{k \times n}{V^\intercal}$$

- UD matrix gives coefficients to reconstruct data: $x_i = U_{i,1}D_{1,1}v_1 + U_{i,2}D_{2,2}v_2 + \cdots$
  - We can truncate this after top k singular values (square root of eigenvalues)

Latent-space representations: uses

- Remove unneeded features
  - Features that add very little information (e.g. low variability, high noise)
  - Features that are similar to others (e.g. almost linearly dependent)
  - Reduce dimensionality for downstream application
    - Supervised learning: fewer parameters, need less data
    - Compression: less bandwidth
- Can also add features
  - Summarize multiple features into few cleaner / higher-level ones



PCA: applications

- Eigen-faces
  - Represent image data (e.g. faces) using PCA
- Latent-Space Analysis (topic models)
  - Represent text data (e.g. bag of words) using PCA
- Collaborative Filtering for Recommendation Systems
  - Represent sentiment data (e.g. ratings) using PCA

Eigen-faces

- "Eigen-X" = represent X using its principal components

$$\underset{m \times n}{X} \approx \underset{m \times k}{U} \cdot \underset{k \times k}{D} \cdot \underset{k \times n}{V^{\intercal}}$$

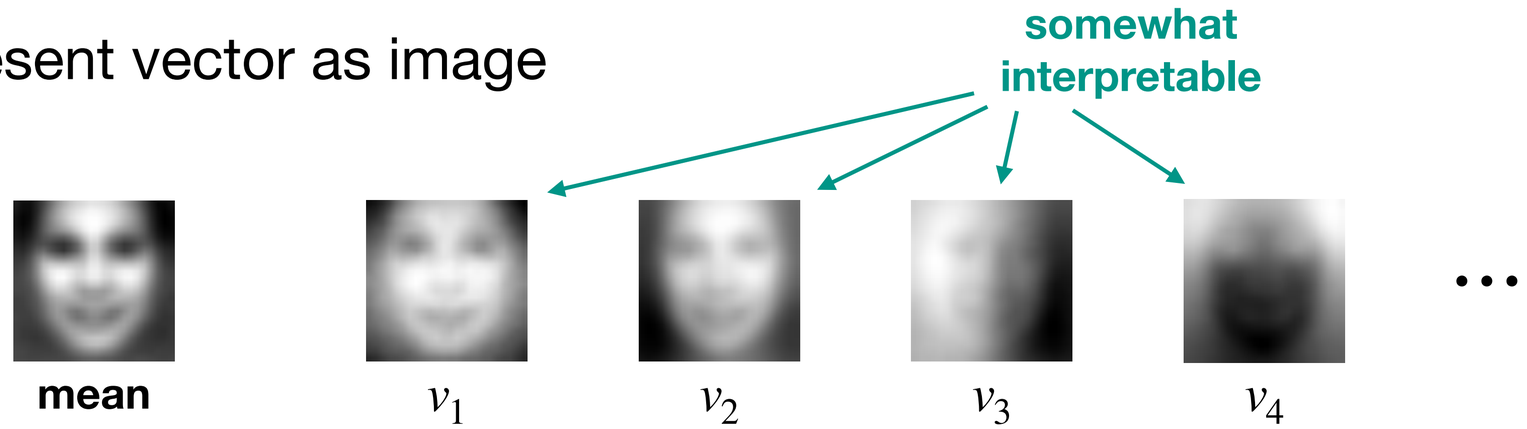
- Viola Jones dataset: 24×24 images $\in \mathbb{R}^{576}$
  - Can represent vector as image

Eigen-faces

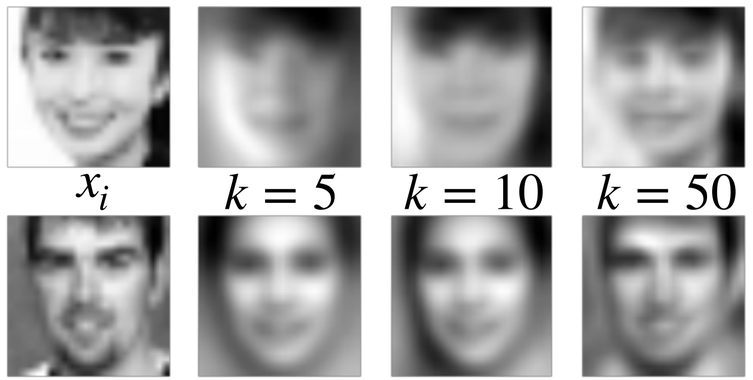
- "Eigen-X" = represent X using its principal components

$$\underset{m \times n}{X} \approx \underset{m \times k}{U} \cdot \underset{k \times k}{D} \cdot \underset{k \times n}{V^\intercal}$$

- Viola Jones dataset: 24×24 images $\in \mathbb{R}^{576}$
  - Can represent vector as image



  - Project data on k principal components



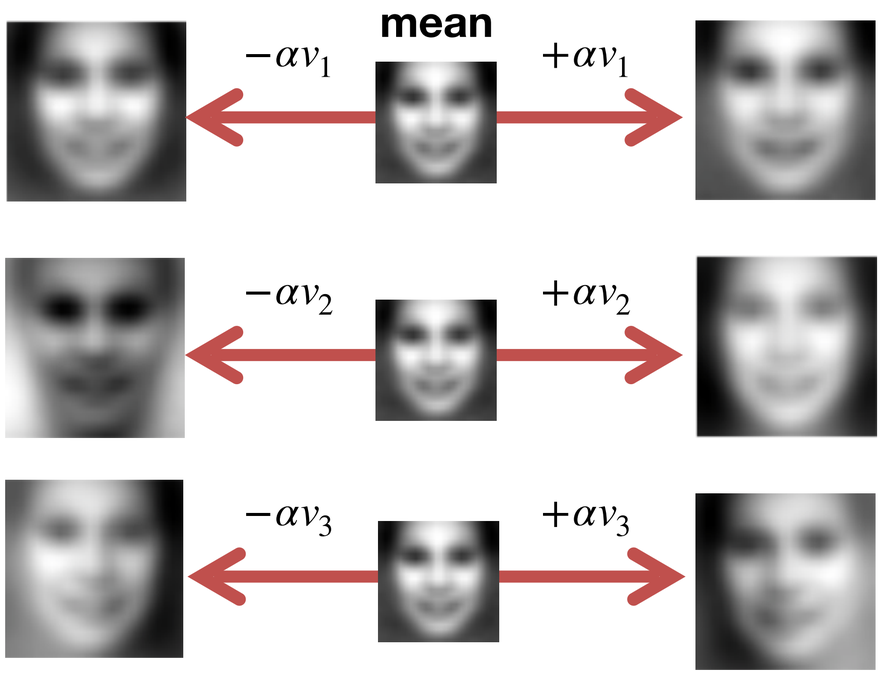
Eigen-faces

- "Eigen-X" = represent X using its principal components

$$\underset{m \times n}{X} \approx \underset{m \times k}{U} \cdot \underset{k \times k}{D} \cdot \underset{k \times n}{V^\intercal}$$

- Viola Jones dataset: 24×24 images $\in \mathbb{R}^{576}$
  - Can represent vector as image
  - Visualize basis vectors v_i

    as $\mu \pm \alpha v_i$

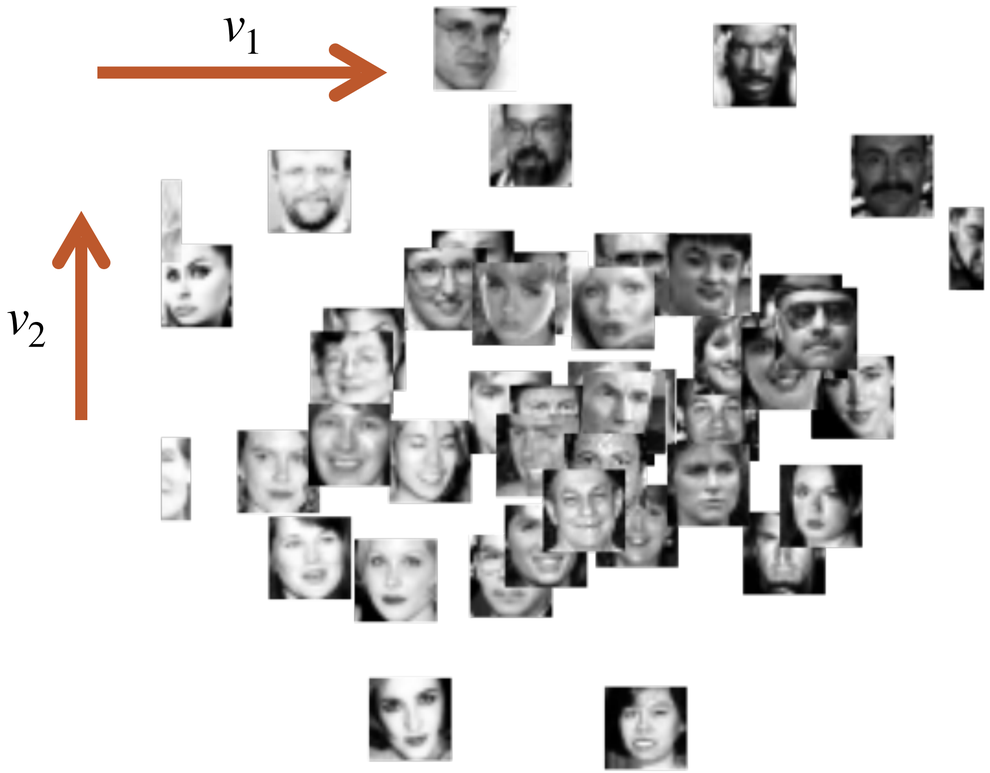


Eigen-faces

- "Eigen-X" = represent X using its principal components

$$\underset{m \times n}{X} \approx \underset{m \times k}{U} \cdot \underset{k \times k}{D} \cdot \underset{k \times n}{V^\intercal}$$

- Viola Jones dataset: 24×24 images $\in \mathbb{R}^{576}$
  - Can represent vector as image

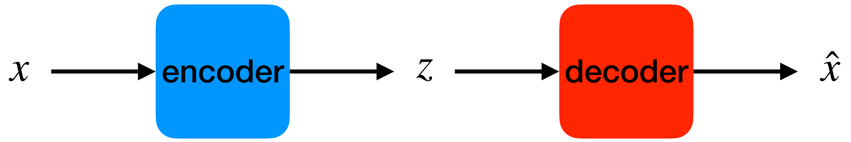


  - Visualize data by projecting

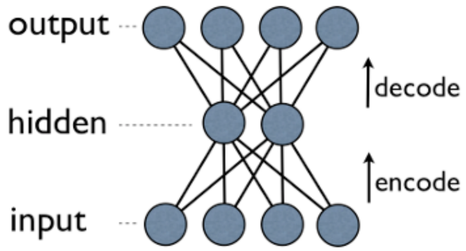
    onto 2 principal components

Nonlinear latent spaces

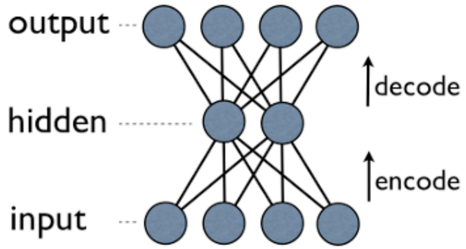
- Latent-space representation = represent x_i as z_i
  - Usually more succinct, less noisy
  - Preserves most (interesting) information on $x_i \implies$ can reconstruct $\hat{x}_i \approx x_i$
  - Auto-encoder = encode $x \to z$, decode $z \to \hat{x}$



- Linear latent-space representation:
  - Encode: $Z = XV_{\leq k} = (UDV^\intercal V)_{\leq k} = U_{\leq k}D_{\leq k}$; Decode: $X \approx ZV_{\leq k}^\intercal$
- Nonlinear: e.g., encoder + decoder are neural networks
  - Restrict z to be shorter than $x \implies$ requires succinctness



Variational Auto-Encoders (VAE)



- Probabilistic model:
  - Simple prior over latent space $p(z)$ (e.g. Gaussian)
  - Decoder = generator $p_\theta(x \mid z)$, tries to match data distribution $p_\theta(x) \approx \mathscr{D}$
  - Encoder = inference $q_\phi(z \mid x)$, tries to match posterior $q_\phi(z \mid x) \approx \dfrac{p(z)p_\theta(x \mid z)}{p_\theta(x)}$

Logistics

assignments

- Assignment 5 due Tuesday, Nov 23