UCI University of California, Irvine

CS 273A: Machine Learning

Fall 2021

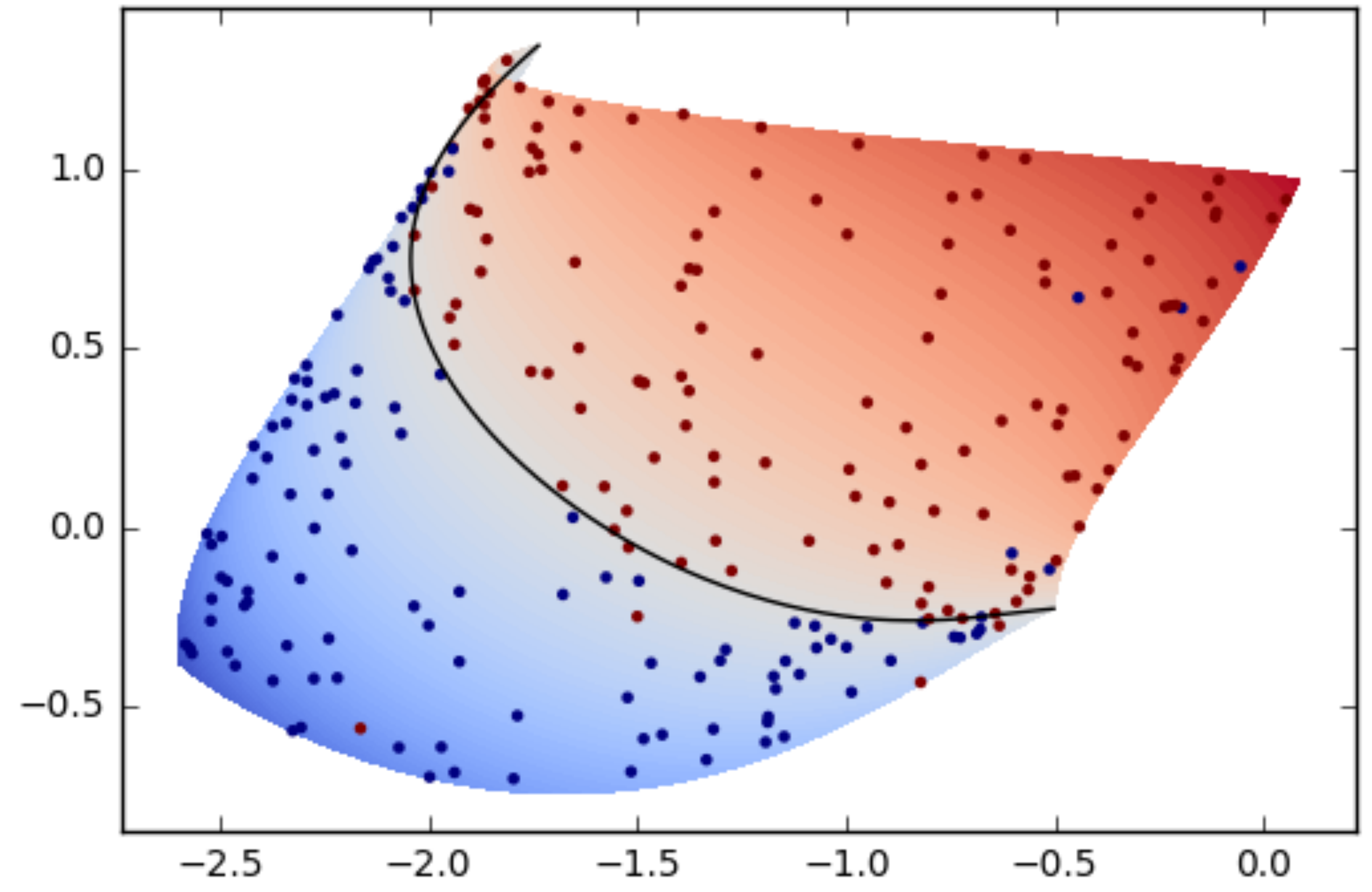
Lecture 14: Clustering

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

project

- Project abstract due today on Canvas

assignments

- Assignment 5 due Tuesday, Nov 23

Today's lecture

Gradient boosting

AdaBoost

k-Means

Agglomerative clustering

Growing ensembles

- Ensemble = collection of models: $\hat{y}(x) = \sum_k f_k(x)$
  - Models should “cover” for each other
- If we could add a model to a given ensemble, what would we add?

$$\mathscr{L}(y, \hat{y}') = \mathscr{L}(y, \hat{y} + f_{K+1}(x))$$

- Let's find $f_{K+1}(x)$ that minimizes this loss
  - If we could do this well — done in one step
  - Instead, let's do it badly but many times → gradually improve

Boosting

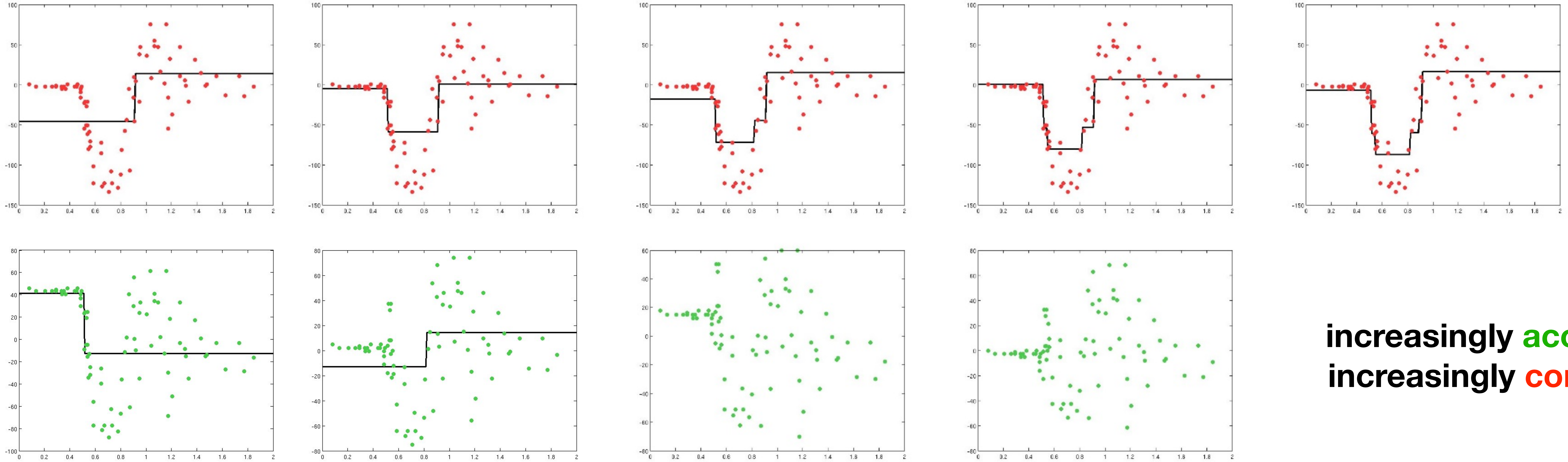
- Question: can we create a strong learner from many weak learners?
  - Weak learner = underfits, but fast and simple (e.g., decision stump, perceptron)
  - Strong learner = performs well but increasingly complex
- Boosting: focus new learners on instances that current ensemble gets wrong
  - Train new learner
  - Measure errors
  - Re-weight data points to emphasize large residuals
  - Repeat

Example: MSE loss

- Ensemble: $\hat{y}_K = \sum_k f_k(x)$; MSE loss: $\mathscr{L}(y, \hat{y}_k) = \frac{1}{2}(y - \hat{y}_{k-1} - f_k(x))^2$
  - To minimize: have $f_k(x)$ try to predict $y - \hat{y}_{k-1}$
  - Then update $\hat{y}_k = \hat{y}_{k-1} + f_k(x)$

data
prediction

residual
weak model



increasingly accurate
increasingly complex

Gradient Boosting

- More generally: pseudo-residuals $r_k^{(j)} = -\partial_{\hat{y}}\mathscr{L}(y^{(j)}, \hat{y})\Big|_{\hat{y}=\hat{y}_{k-1}^{(j)}}$
  - $r_k^{(j)}$ = steepest descent of loss in "prediction space" (how $\hat{y}_{k-1}^{(j)}$ should change)
  - For MSE loss: $r_k^{(j)} = y^{(j)} - \hat{y}_{k-1}^{(j)}$ as before
- Gradient Boosting:
  - Learn weak model to predict $f_k : x^{(j)} \mapsto r_k^{(j)}$
  - Find best step size $\alpha_k = \arg\min_\alpha \frac{1}{m}\sum_j \mathscr{L}\left(y^{(j)}, \hat{y}_{k-1}^{(j)} + \alpha f_k(x^{(j)})\right)$ (line search)

Demo

- http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

Today's lecture

Gradient boosting

AdaBoost

k-Means

Agglomerative clustering

Growing ensembles

- Ensemble = collection of models: $\hat{y}(x) = \sum_k \alpha_k f_k(x)$
  - Models should "cover" for each other
- If we could add a model to a given ensemble, what would we add?

$$\mathscr{L}(y, \hat{y}_k) = \mathscr{L}(y, \hat{y}_{k-1} + \alpha_k f_k(x))$$

- Let's find $\alpha_k, f_k(x)$ that minimize this loss
  - If we could do this well — done in one step
  - Instead, let's do it badly but many times → gradually improve

Example: exponential loss

- Exponential loss: $\mathcal{L}(y, \hat{y}) = e^{-y\hat{y}}$
  - Optimal $\hat{y}(x)$: $\arg\min_{\hat{y}} \mathbb{E}_{y|x}[\mathcal{L}(y, \hat{y})] = \frac{1}{2} \ln \frac{p(y=+1\,|\,x)}{p(y=-1\,|\,x)}$ (proof by derivative)
  - If we can minimize the loss $\implies$ $\operatorname{sign}(\hat{y})$ is the more likely label
- Let's find model $f_k : x \mapsto \{+1, -1\}$ that minimizes

$$\sum_j \mathcal{L}(y^{(j)}, \hat{y}_k^{(j)}) = \sum_j \mathcal{L}(y^{(j)}, \hat{y}_{k-1}^{(j)} + \alpha_k f_k(x^{(j)})) = \sum_j \overbrace{e^{-y^{(j)}\hat{y}_{k-1}^{(j)}}}^{w_{k-1}^{(j)}} e^{-y^{(j)}\alpha_k f_k(x^{(j)})}$$

independent of f_k

$$= (e^{\alpha_k} - e^{-\alpha_k}) \sum_j w_{k-1}^{(j)} \delta[y^{(j)} \neq f_k(x^{(j)})] + e^{-\alpha_k} \sum_j w_{k-1}^{(j)}$$

independent of f_k

Minimizing weighted loss

- So far, we minimized average loss: $\frac{1}{m}\sum_j \mathscr{L}(y^{(j)}, \hat{y}^{(j)})$
- We can also minimize weighted loss: $\sum_j w^{(j)}\mathscr{L}(y^{(j)}, \hat{y}^{(j)})$
    - Every data point "counts" as $w^{(j)}$
    - E.g., in decision trees, weighted info gain obtained by $p(y = c) \propto \sum_{j:y^{(j)}=c} w^{(j)}$
- In our current case, weighted 0–1 loss: $\sum_j w_{k-1}^{(j)}\delta[y^{(j)} \neq f_k(x^{(j)})]$

Boosting with exponential loss (cont.)

- The best classifier to add to the ensemble minimizes weighted 0–1 loss:

$$\sum_j w_{k-1}^{(j)} \delta[y^{(j)} \neq f_k(x^{(j)})] \quad \text{with } w_{k-1}^{(j)} = e^{-y^{(j)}\hat{y}_{k-1}^{(j)}}$$

- It gives weighted error rate $\epsilon_k = \dfrac{\sum_j w_{k-1}^{(j)} \delta[y^{(j)} \neq f_k(x^{(j)})]}{\sum_j w_{k-1}^{(j)}}$

- Plugging into the loss and solving: $\alpha_k = \dfrac{1}{2} \ln \dfrac{1 - \epsilon_k}{\epsilon_k}$

- Now add the model and update the ensemble $\hat{y}_k(x) = \hat{y}_{k-1}(x) + \alpha_k f_k(x)$

AdaBoost

- AdaBoost = adaptive boosting:
  - Initialize $w_0^{(j)} = \frac{1}{m}$
  - Train classifier f_k on training data with weights w_{k-1}
  - Compute weighted error rate $\epsilon_k = \frac{\sum_j w_{k-1}^{(j)} \delta[y^{(j)} \neq f_k(x^{(j)})]}{\sum_j w_{k-1}^{(j)}}$
  - Compute $\alpha_k = \frac{1}{2} \ln \frac{1 - \epsilon_k}{\epsilon_k}$
  - Update weights $w_k^{(j)} = w_{k-1}^{(j)} e^{-y^{(j)} \alpha_k f_k(x^{(j)})}$ (increase weight for misclassified points)
- Predict $\hat{y}(x) = \operatorname{sign} \sum_k \alpha_k f_k(x)$

Recap

- Ensembles = collections of predictors
  - Combine predictions to improve performance
- Boosting: Gradient Boost, AdaBoost, ...
  - Build strong predictor from many weak ones
  - Train sequentially; later predictors focus on mistakes by earlier
    - Weight “hard” examples more

Today's lecture

Gradient boosting

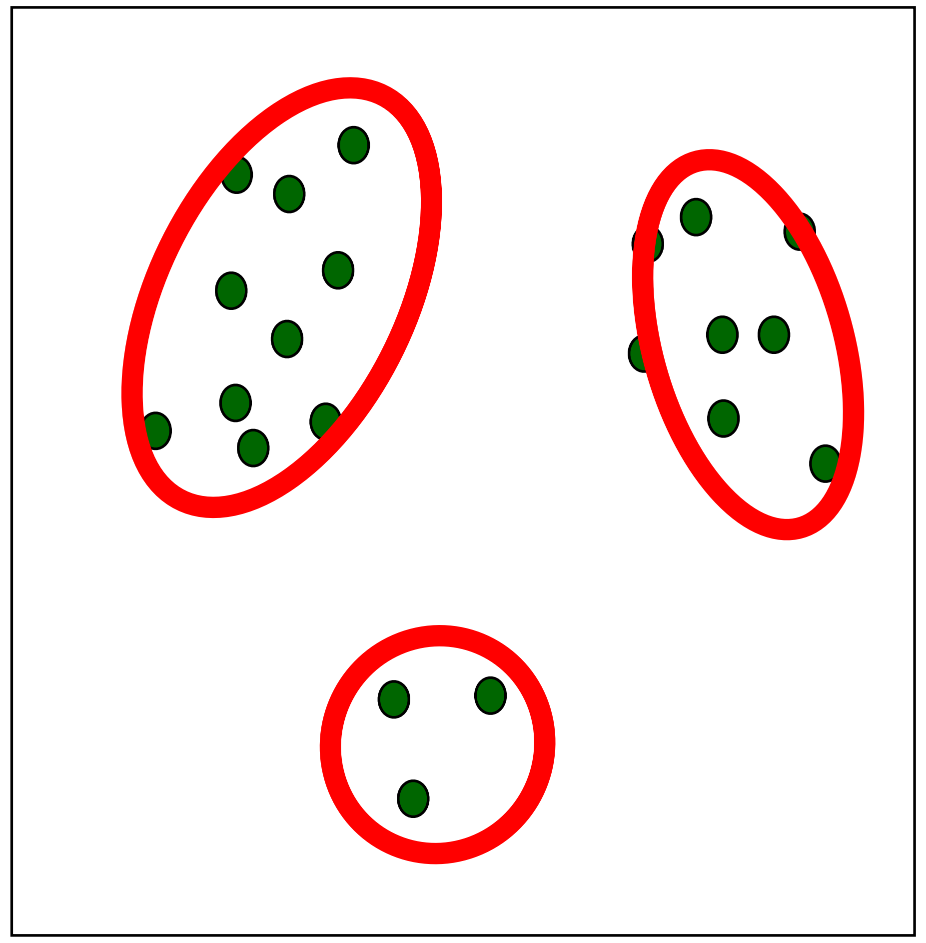
AdaBoost

k-Means

Agglomerative clustering

Unsupervised learning

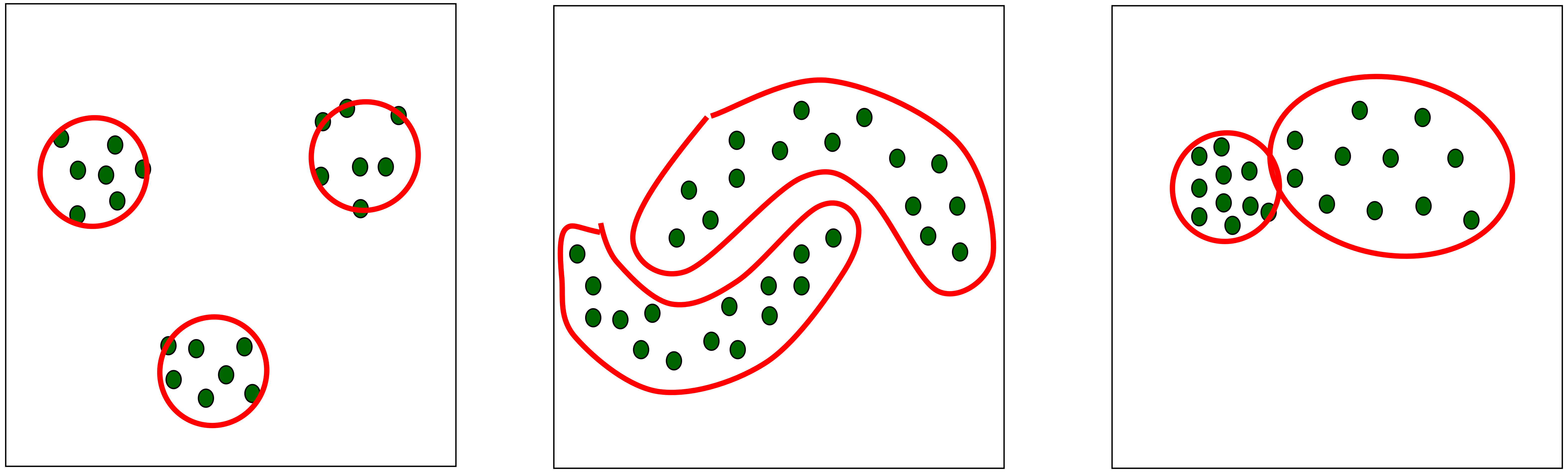
- Supervised learning: learn decision $f : x \mapsto y$ from $\mathscr{D} = \{(x^{(j)}, y^{(j)})\}$
- Unsupervised learning: discover patterns in x from $\mathscr{D} = \{x^{(j)}\}$
  - Explain some features in terms of others
  - Impute missing values
  - Estimate data density (for data generation or anomaly detection)
  - Generate succinct representation (via feature selection or generation)



- Example: clustering
  - Represent data point as member of one of few sets (clusters) with some property

Clustering

- Group data points into few sets
  - Clustering function: $f : x \mapsto c$
  - Similar to classification, except true labels never seen (latent)
- Examples:



close to each other
far from others

large margin

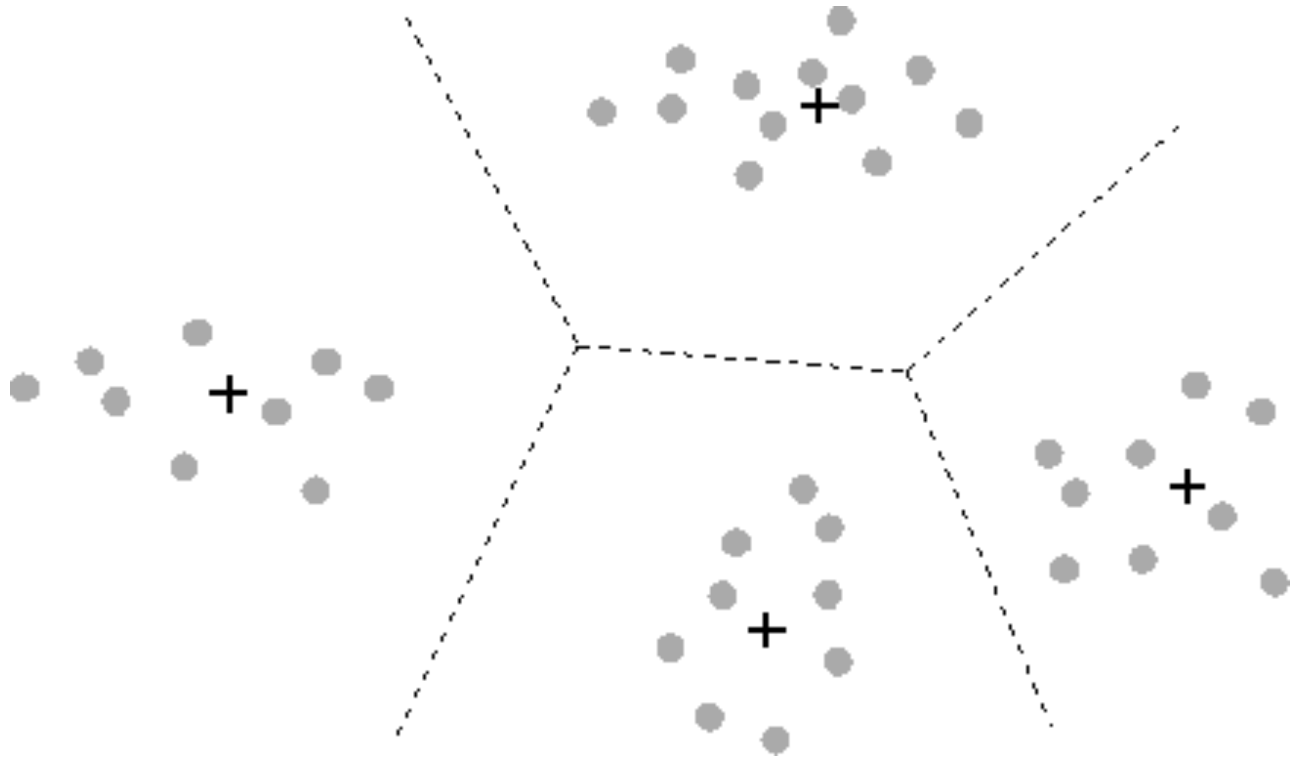
density

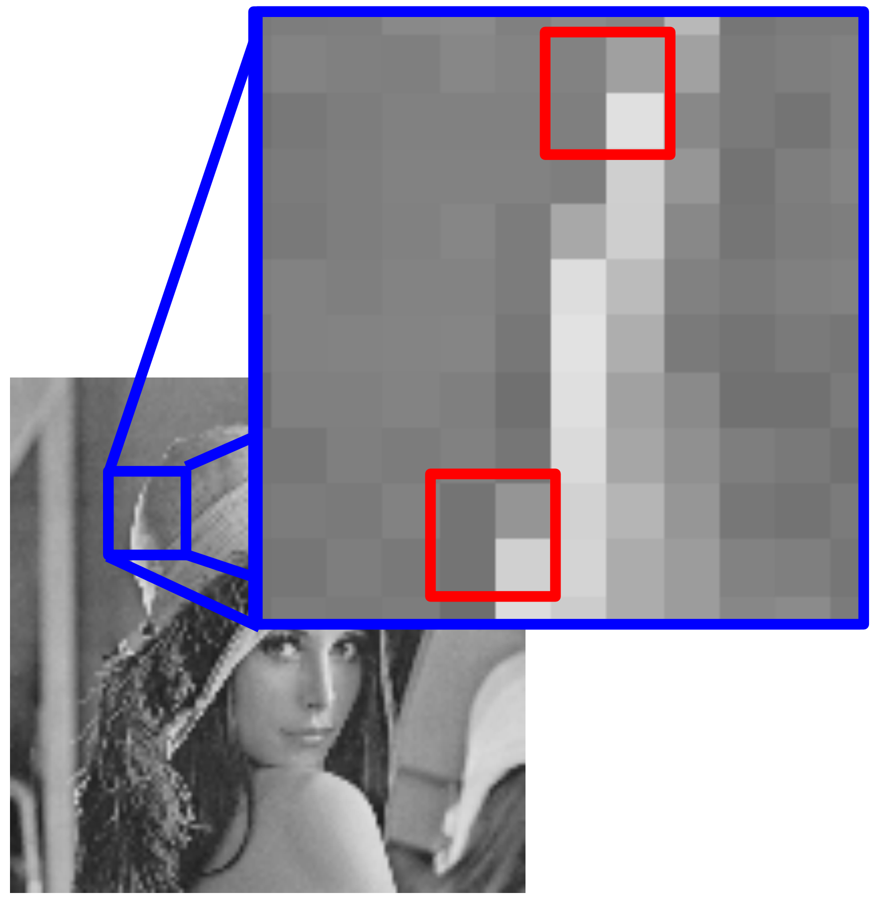
Clustering & compression

$$x \xrightarrow{} \boxed{f} \xrightarrow{c} \boxed{g} \xrightarrow{} \hat{x}$$

- Suppose we must communicate x using only finite symbols (bit string, word)
  - We need an encoder $f : x \mapsto c$ and decoder $g : c \mapsto \hat{x}$
  - Codebook = dictionary of the possible codewords = values of c
- Vector quantization = encoding vector to the nearest dictionary vector

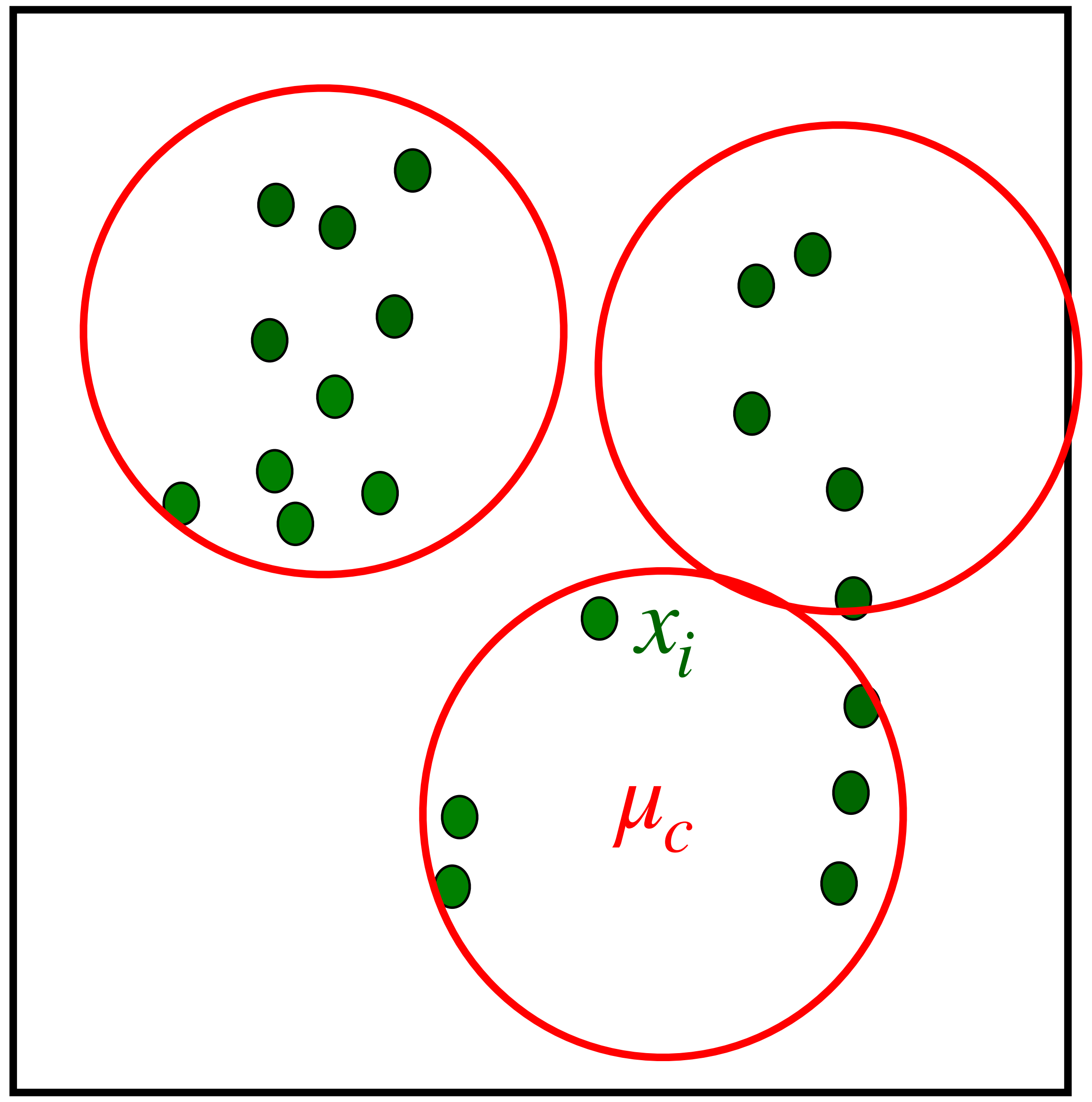
Voronoi regions





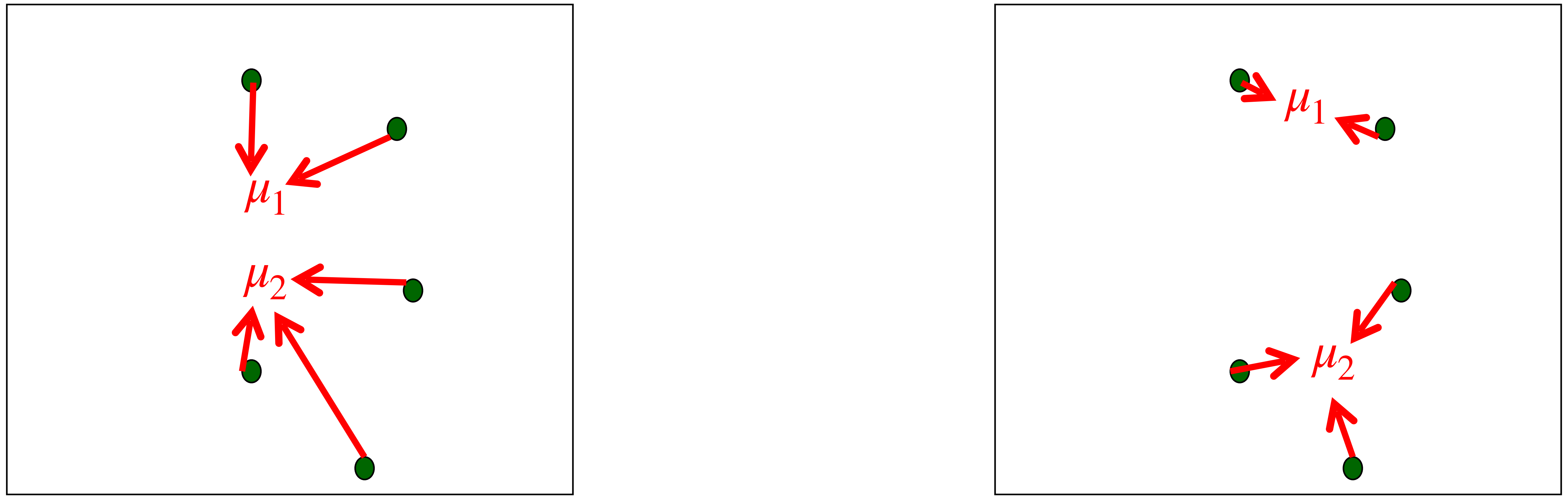
k-Means

- Simple clustering algorithm
- Repeat:
  - Update the clustering = assignment of data points to clusters
  - Update the cluster's representation to match the assigned points
- Notation:
  - x_i = data point in the dataset
  - k = number of clusters
  - μ_c = representation of cluster c



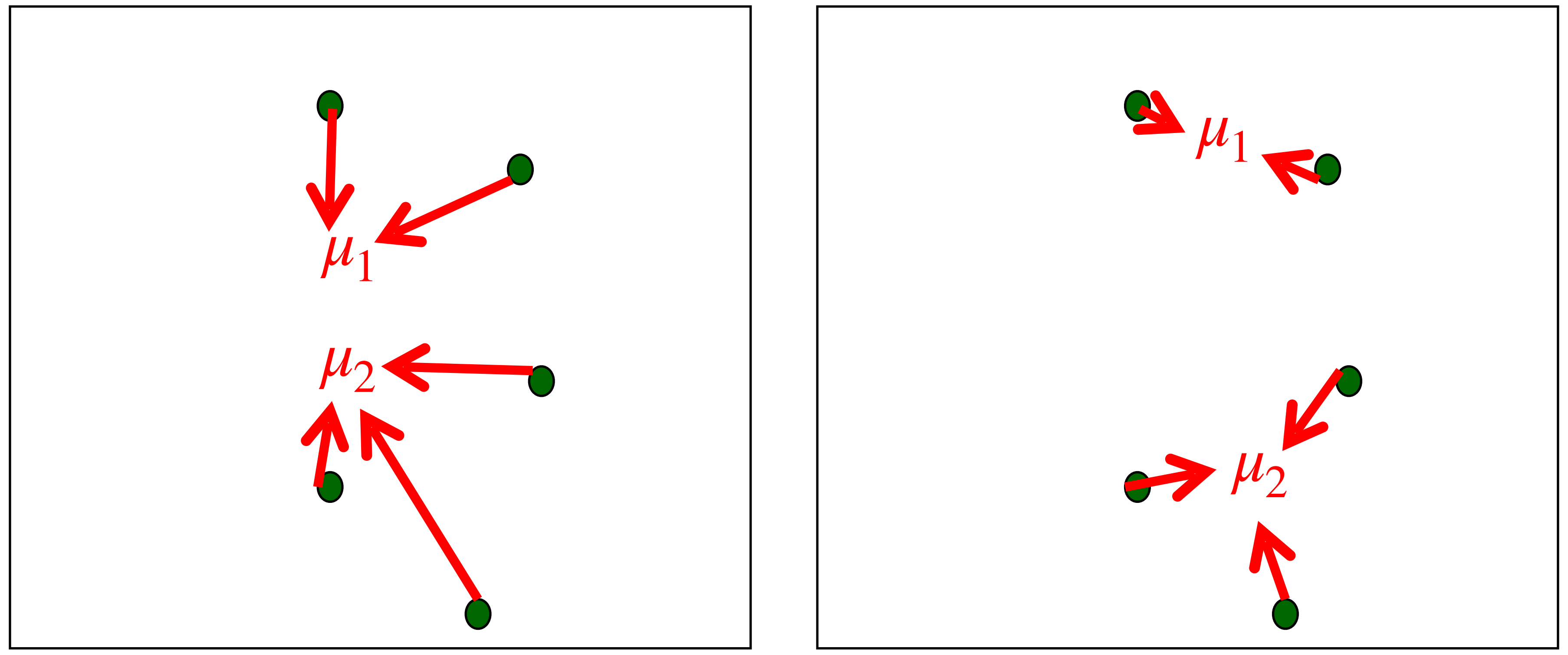
k-Means

- Iterate until convergence:
  - For each $x_i \in \mathscr{D}$, 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: $\mu_c = \frac{1}{m_c} \sum_{i:z_i=c} x_i$



k-Means

- k-Means optimizes the MSE loss: $\mathscr{L}(z, \mu) = \sum_i \|x_i - \mu_{z_i}\|^2$
  - Optimize with respect to z: closest centroid
  - Optimize with respect to μ: cluster mean



- Coordinate descent = each step descends on subset of parameters
- k-Means is guaranteed to converge:
  - $\mathscr{L} \geq 0$, and decreasing every step
  - But convergence may not be to global optimum

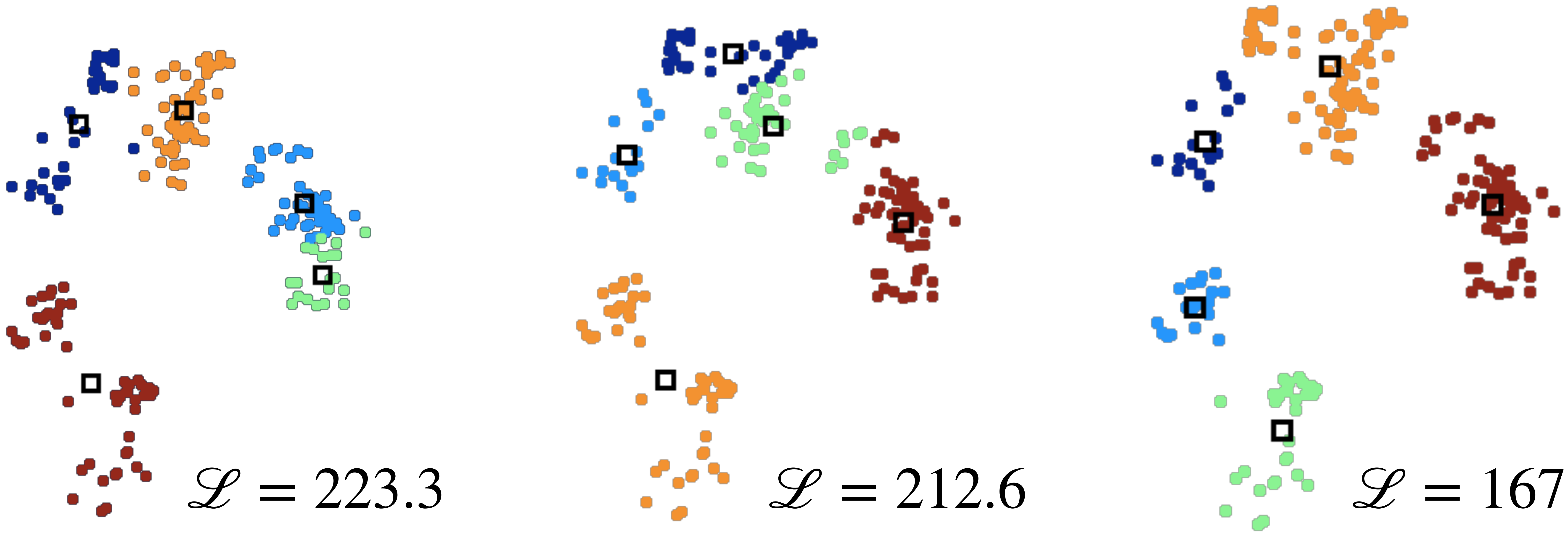
Sensitivity to initialization

- The loss landscape has many local optima ← **Not a problem in the supervised version:** μ **given** $\Longrightarrow$ **1-Nearest Neighbor**

- Different initializations of μ lead to different results

  - Randomly try various initializations

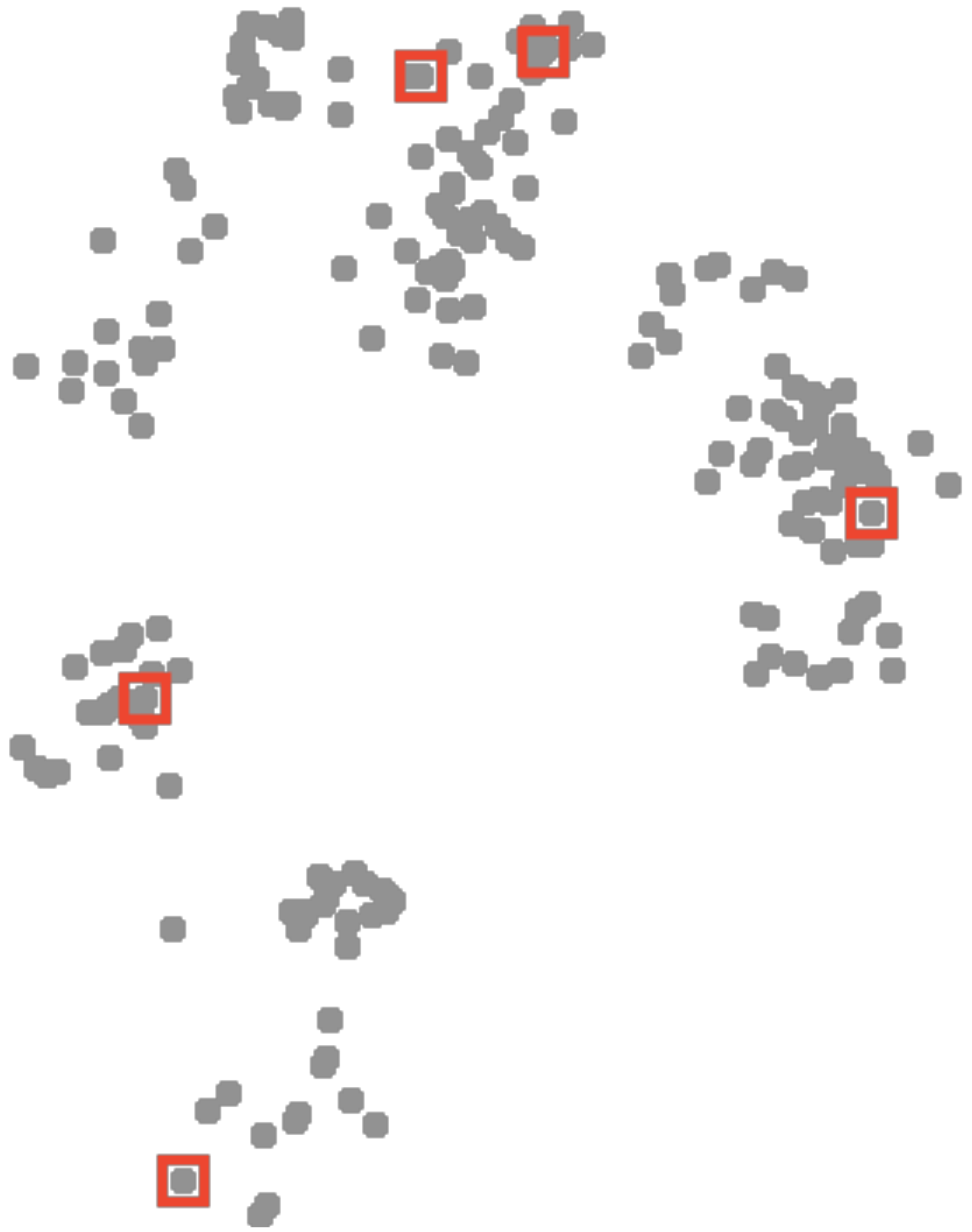
  - Use $\mathscr{L}$ ("training loss") to select best initialization



$\mathscr{L} = 223.3$ $\mathscr{L} = 212.6$ $\mathscr{L} = 167.0$

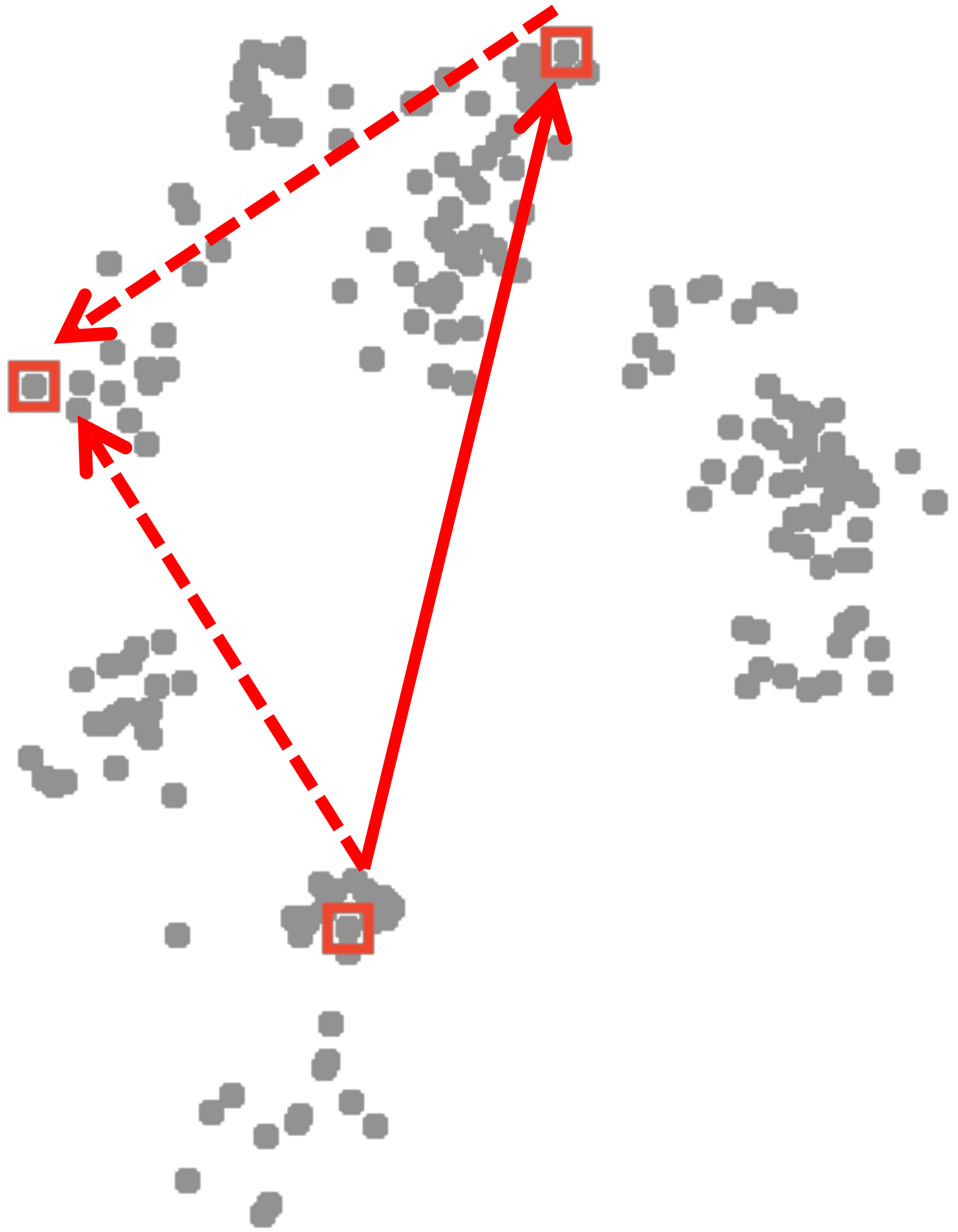
Initialization methods

- Random
  - Initialize each centroid to a random data point
  - Ensures centroids are near some data
  - Issue: may initialize several centroids close together



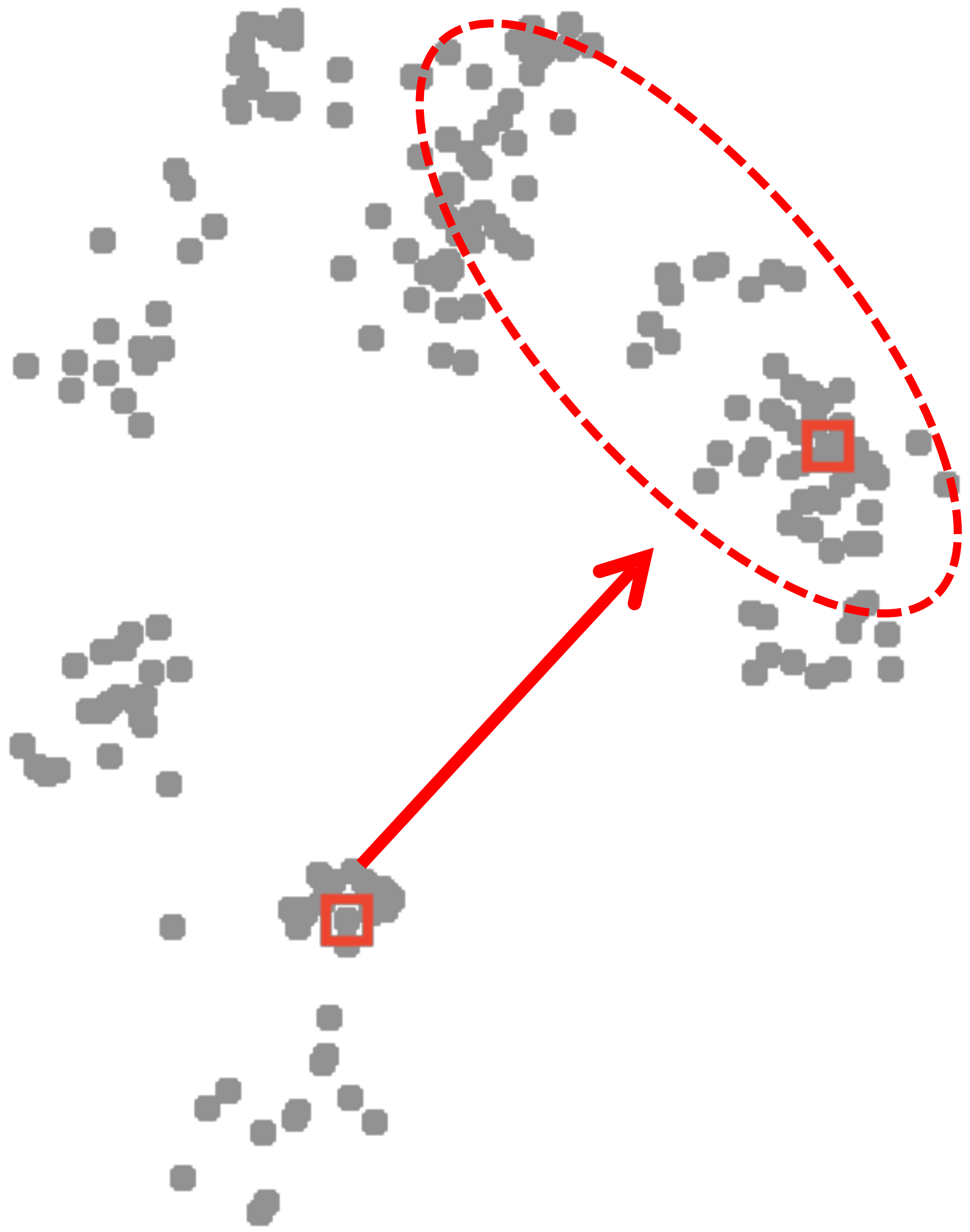
Initialization methods

- Random
  - Initialize each centroid to a random data point
  - Ensures centroids are near some data
  - Issue: may initialize several centroids close together
- Distance-based
  - Initialize first centroid to a random data point
  - Initialize each next centroid to the point farthest from other centroids
  - Issue: may choose outliers



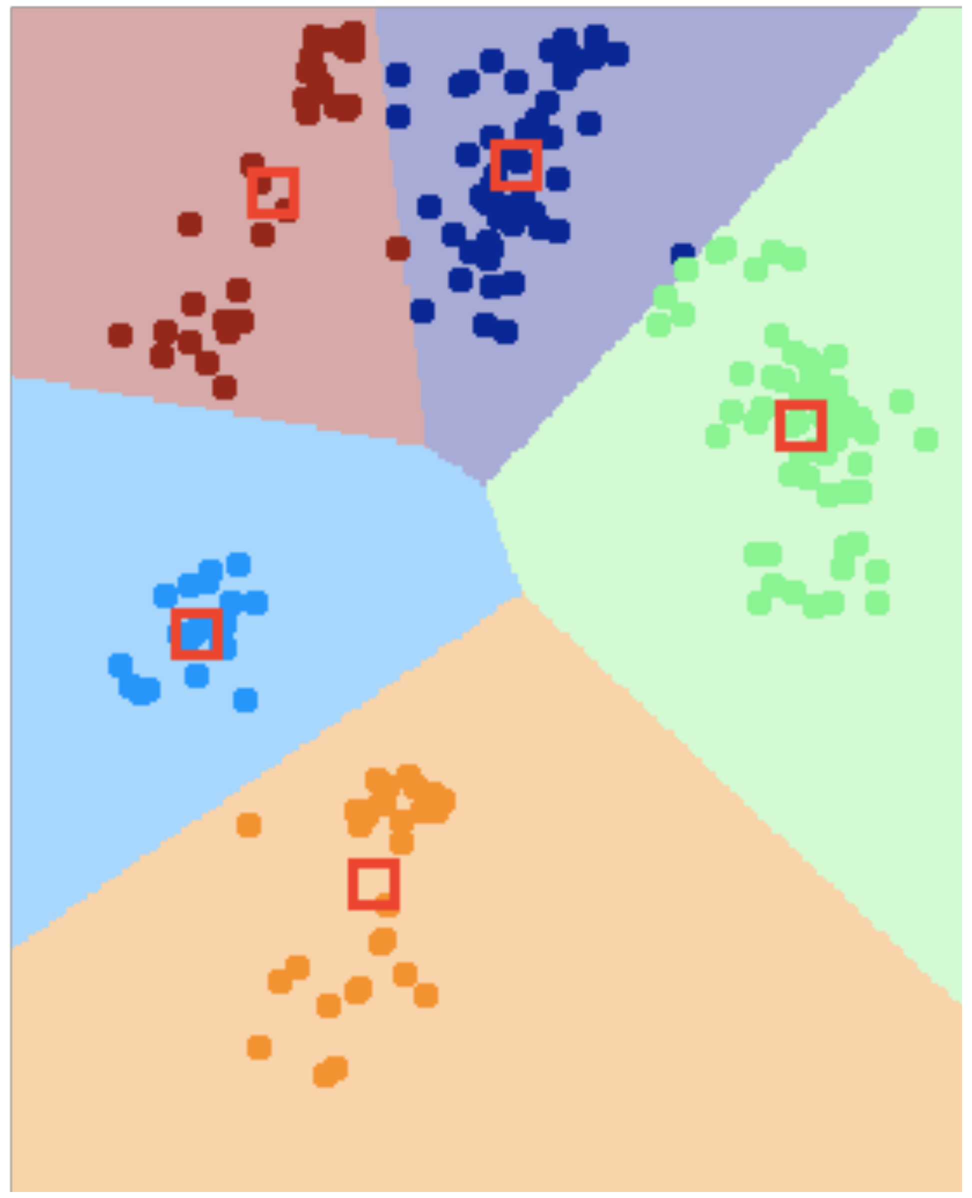
Initialization methods

- Random
  - Initialize each centroid to a random data point
  - Ensures centroids are near some data
  - Issue: may initialize several centroids close together
- Distance-based
  - Initialize first centroid to a random data point
  - Initialize each next centroid to the point farthest from other centroids
  - Issue: may choose outliers
- Randomized distance-based (“k-means++”)
  - Randomize over far points
  - Distribution of next initial centroid: $p(x) \propto (d(x, \mu))^2$
  - Likely to put a cluster far away, in a region with lots of data

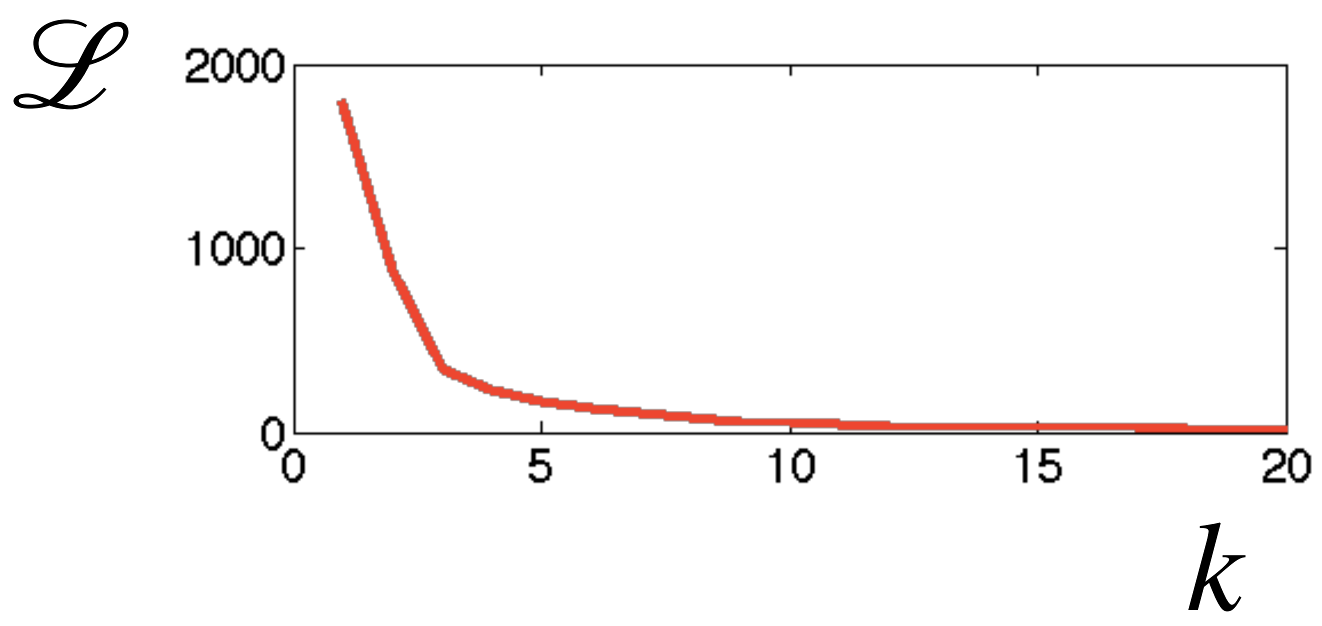


Out-of-sample clustering

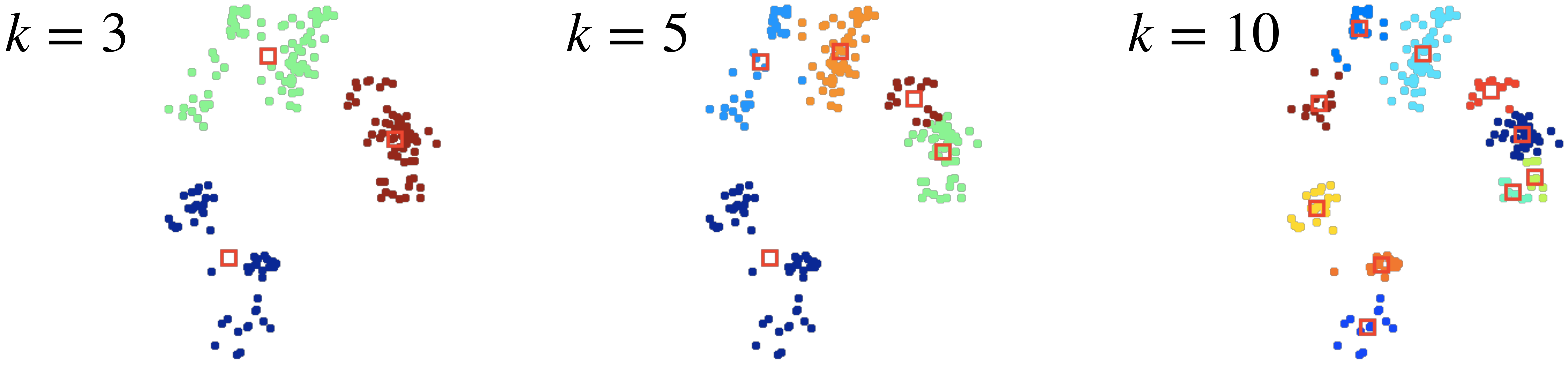
- How can we use clustering to assign new data points?
- In k-Means: choose nearest centroid
  - 1-NN with learned centroids



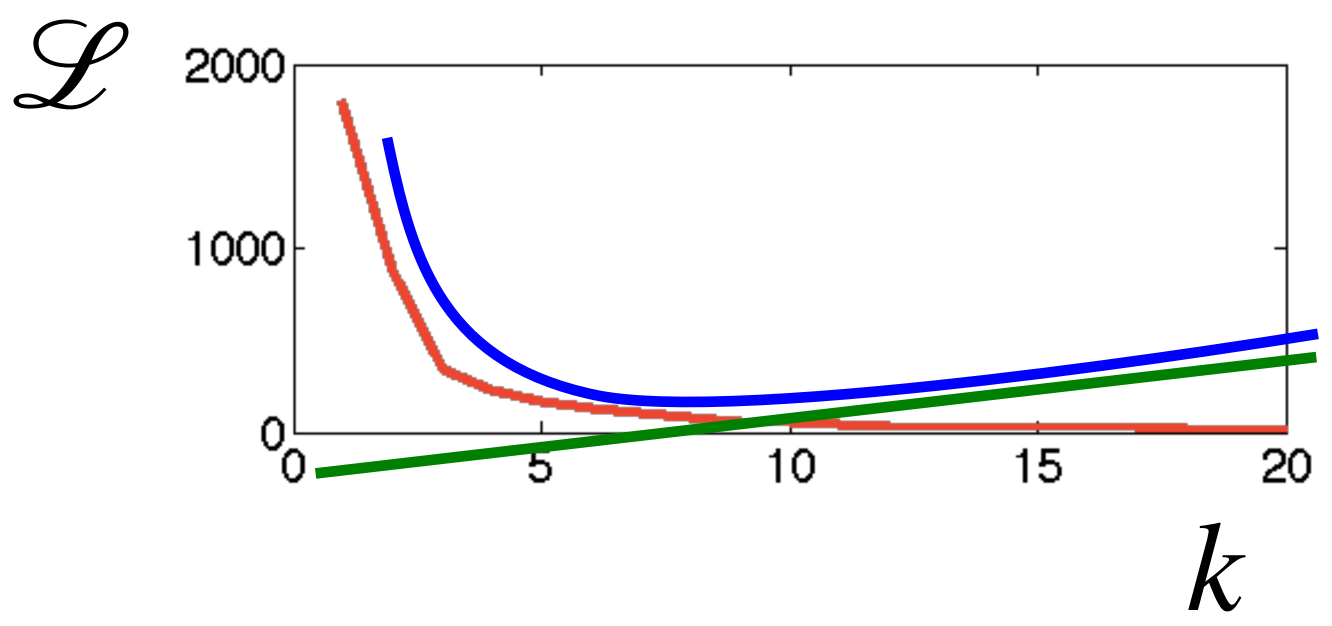
Choosing k



- How to choose the number of clusters k?
- More clusters $\implies$ 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



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 $\mathscr{L}(z, \mu) = \log\left(\frac{1}{md}\sum_i \|x_i - \mu_{z_i}\|^2\right) + k\frac{\log m}{m}$

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

Gradient boosting

AdaBoost

k-Means

Agglomerative clustering

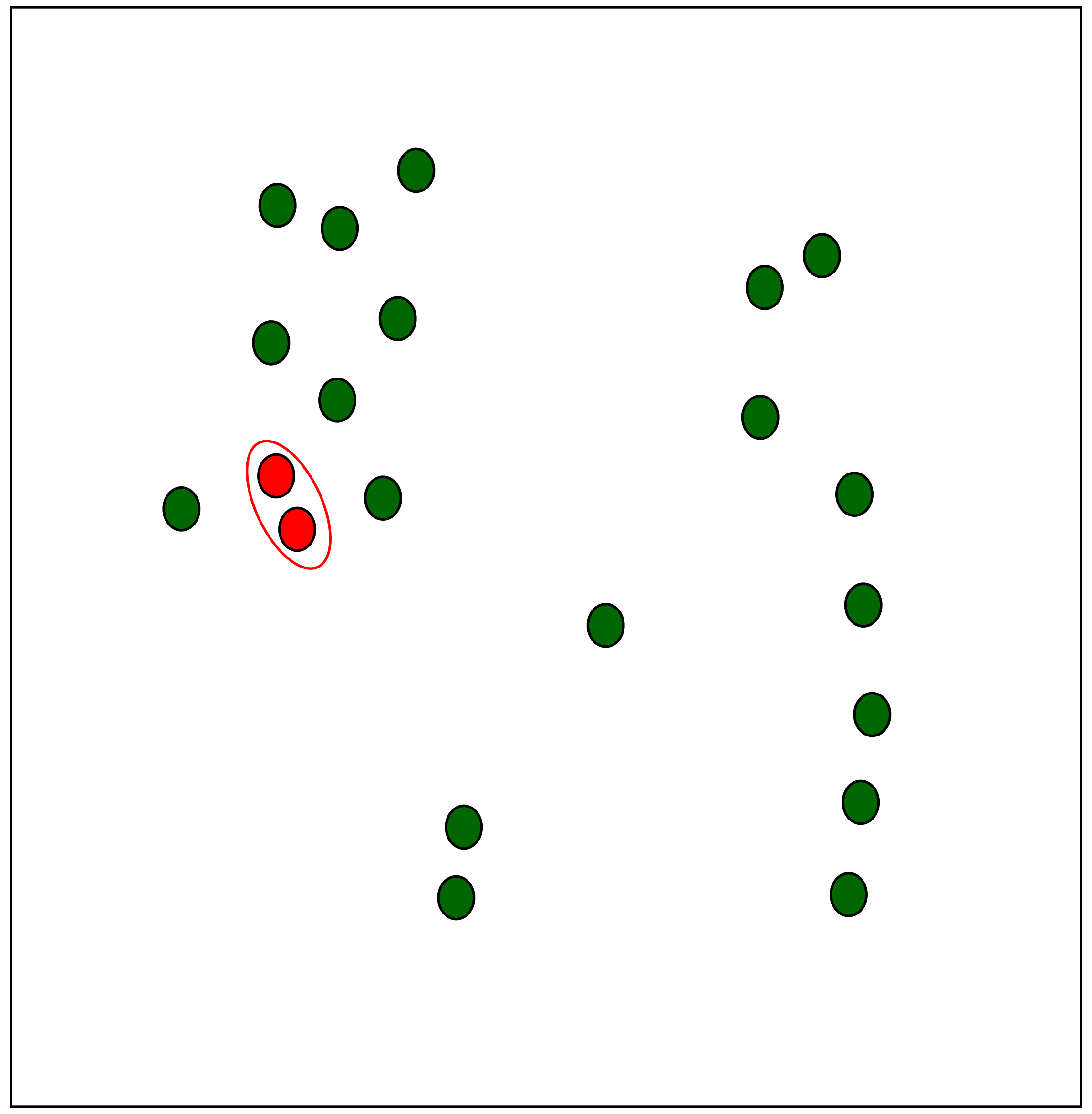
Hierarchical agglomerative clustering

- 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 $\implies O(m^2 \log m)$

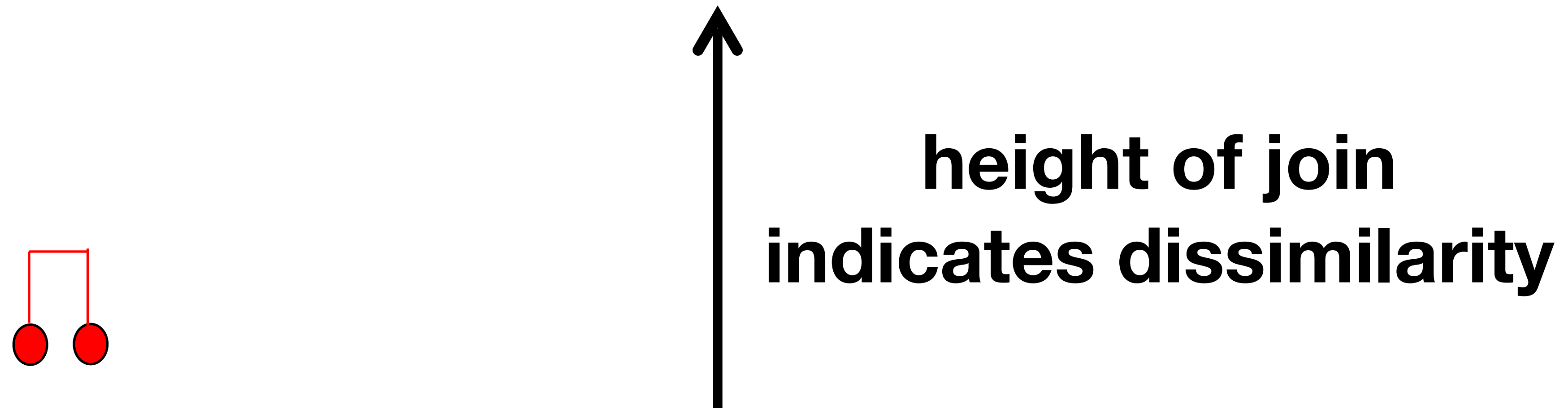
Iteration 1

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

data



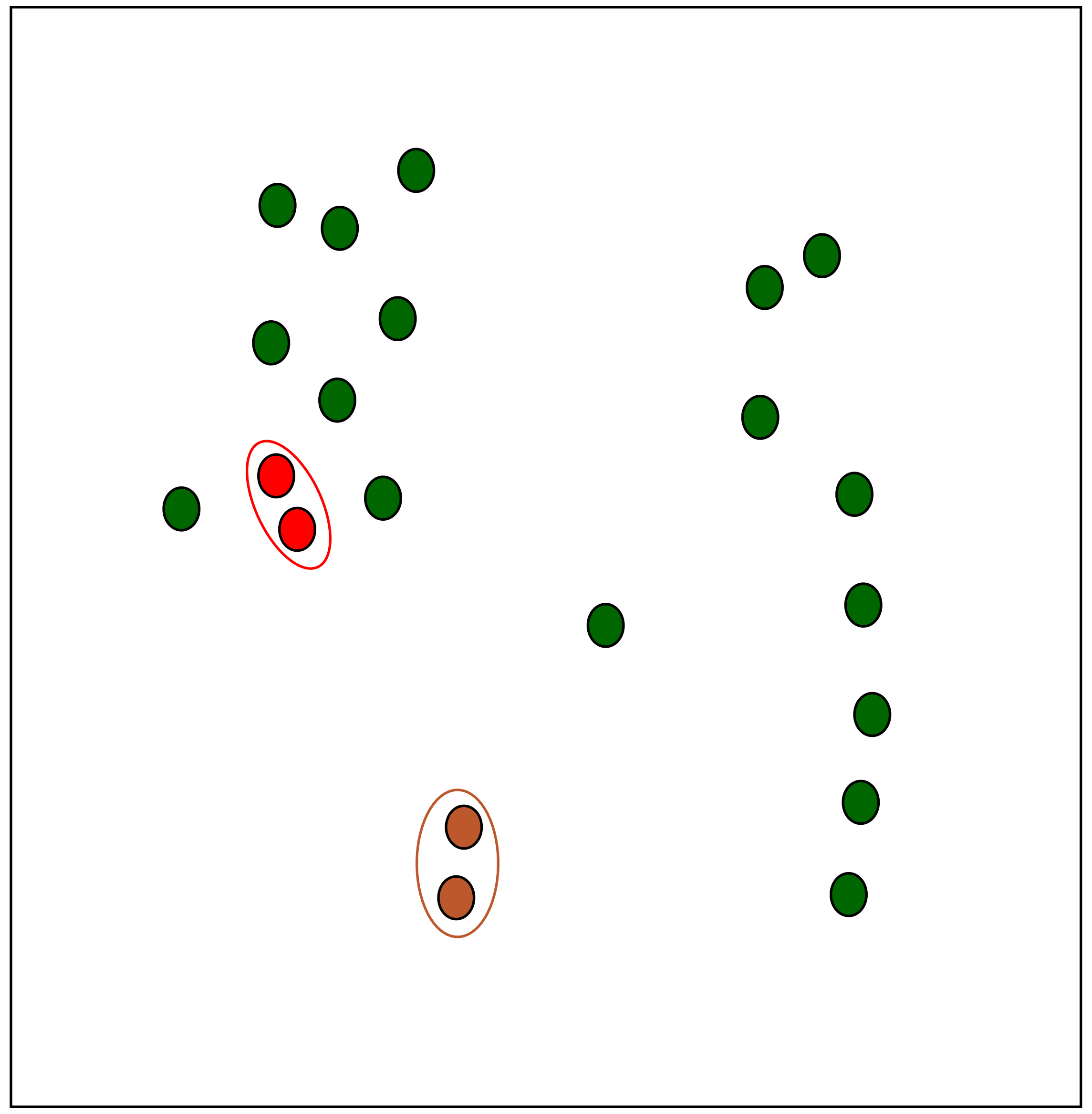
dendrogram



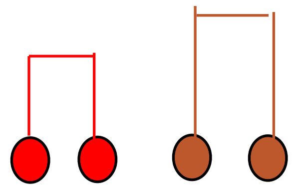
Iteration 2

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

data



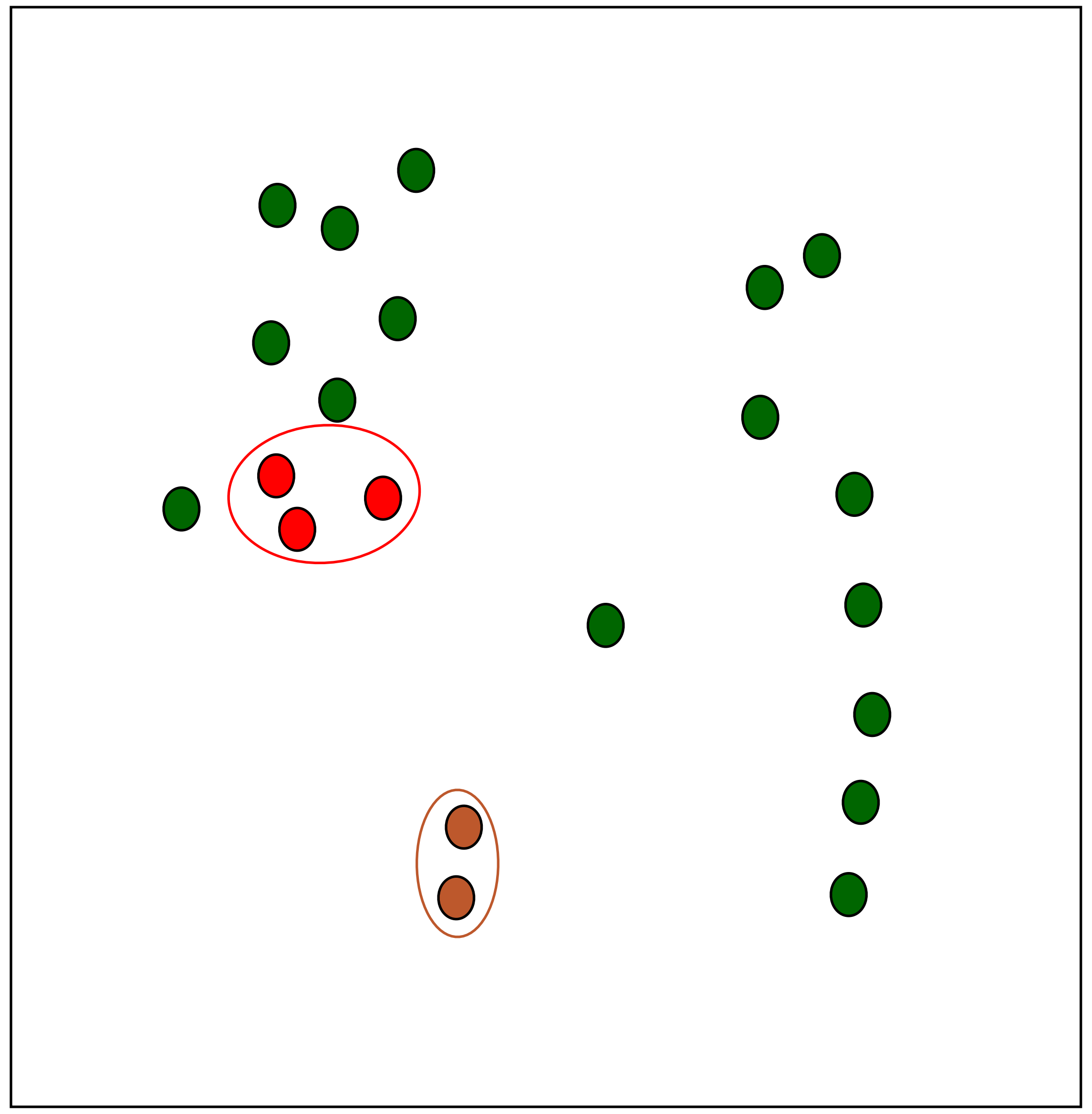
dendrogram

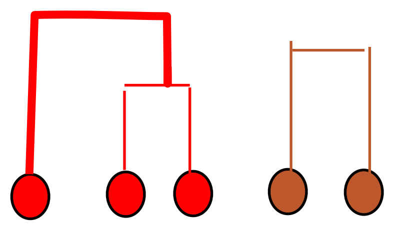


Iteration 3

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

data **dendrogram**

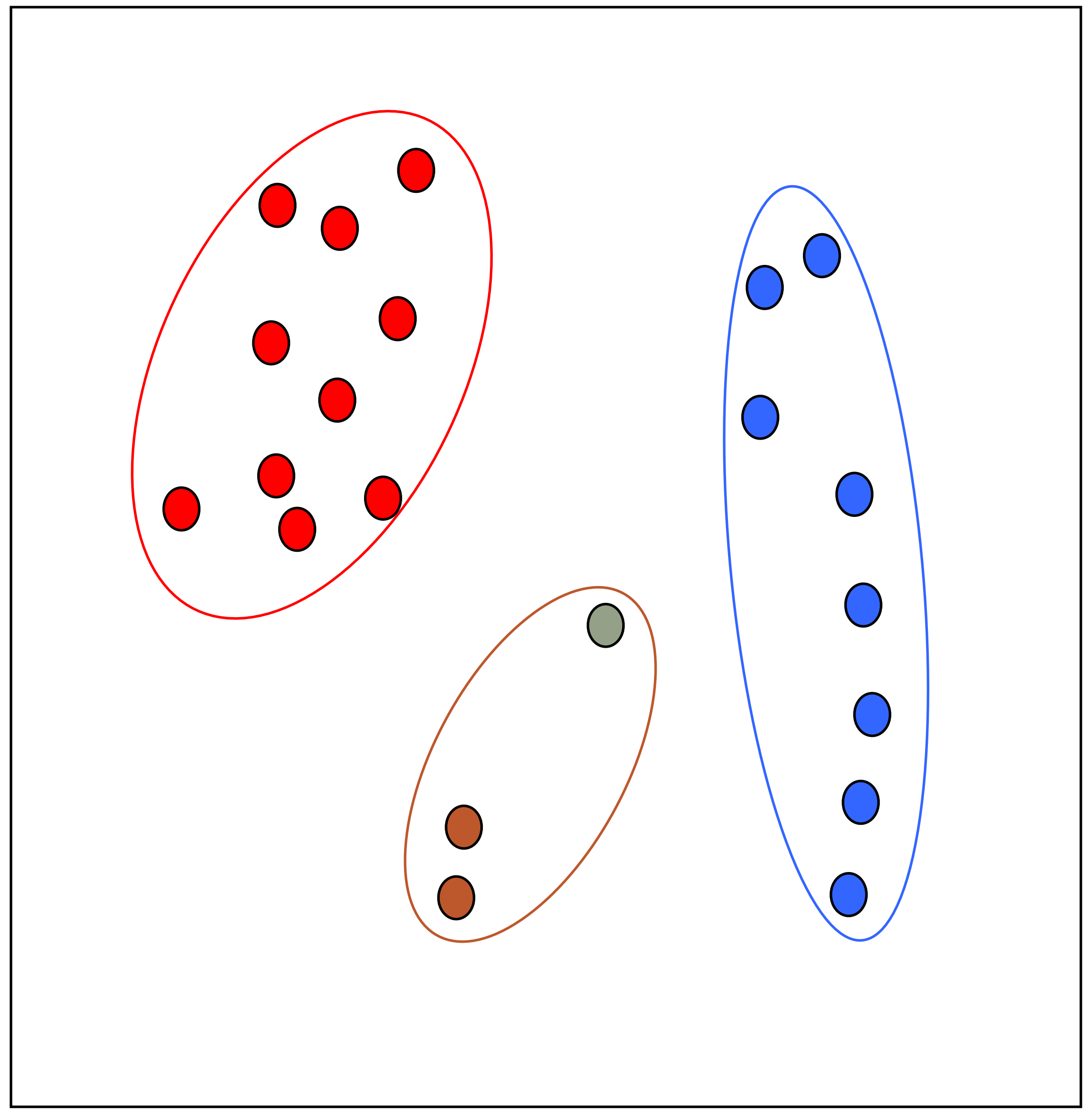




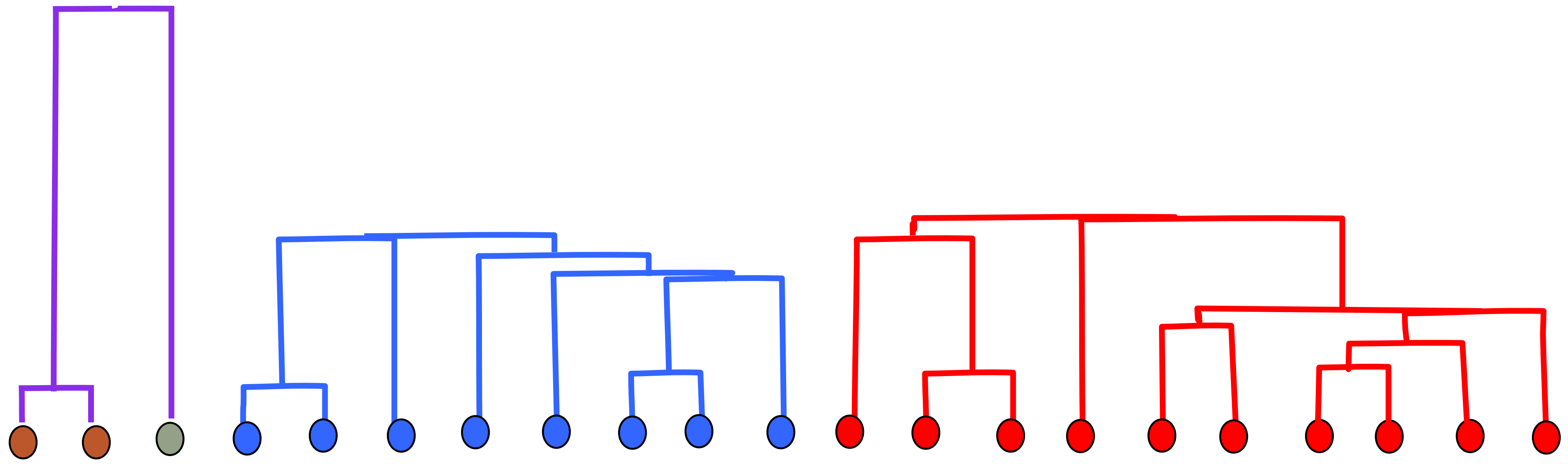
Iteration $m - 3$

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

data



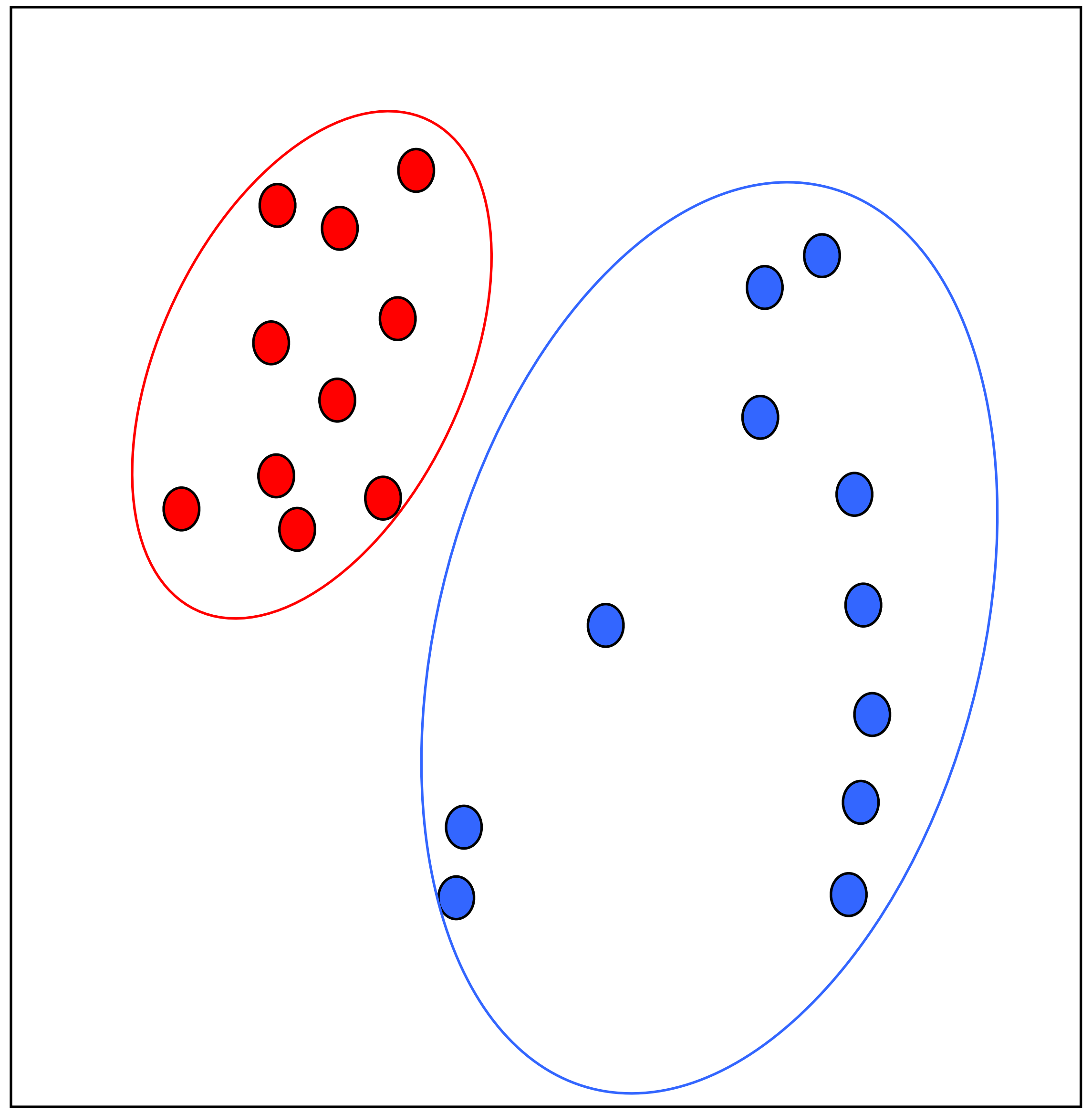
dendrogram



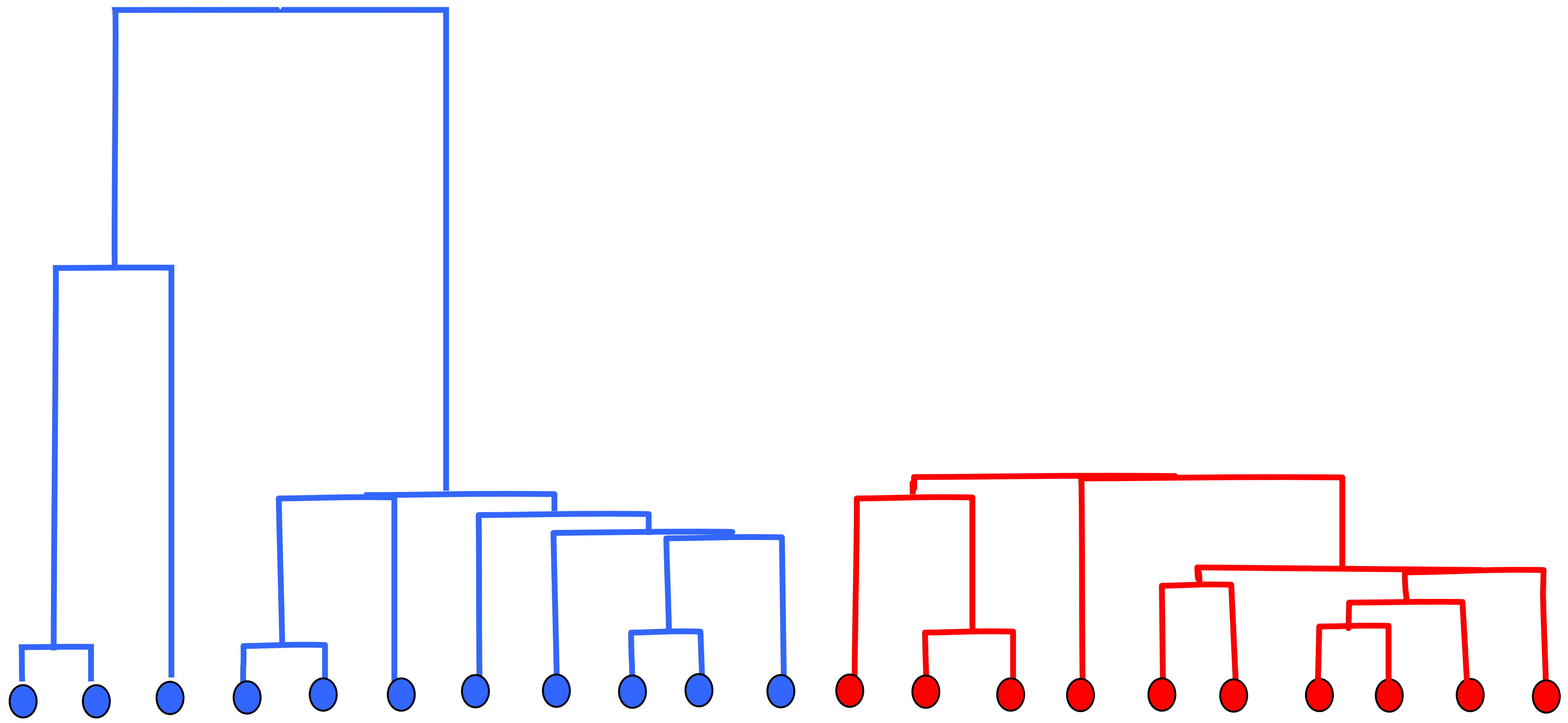
Iteration $m - 2$

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

data



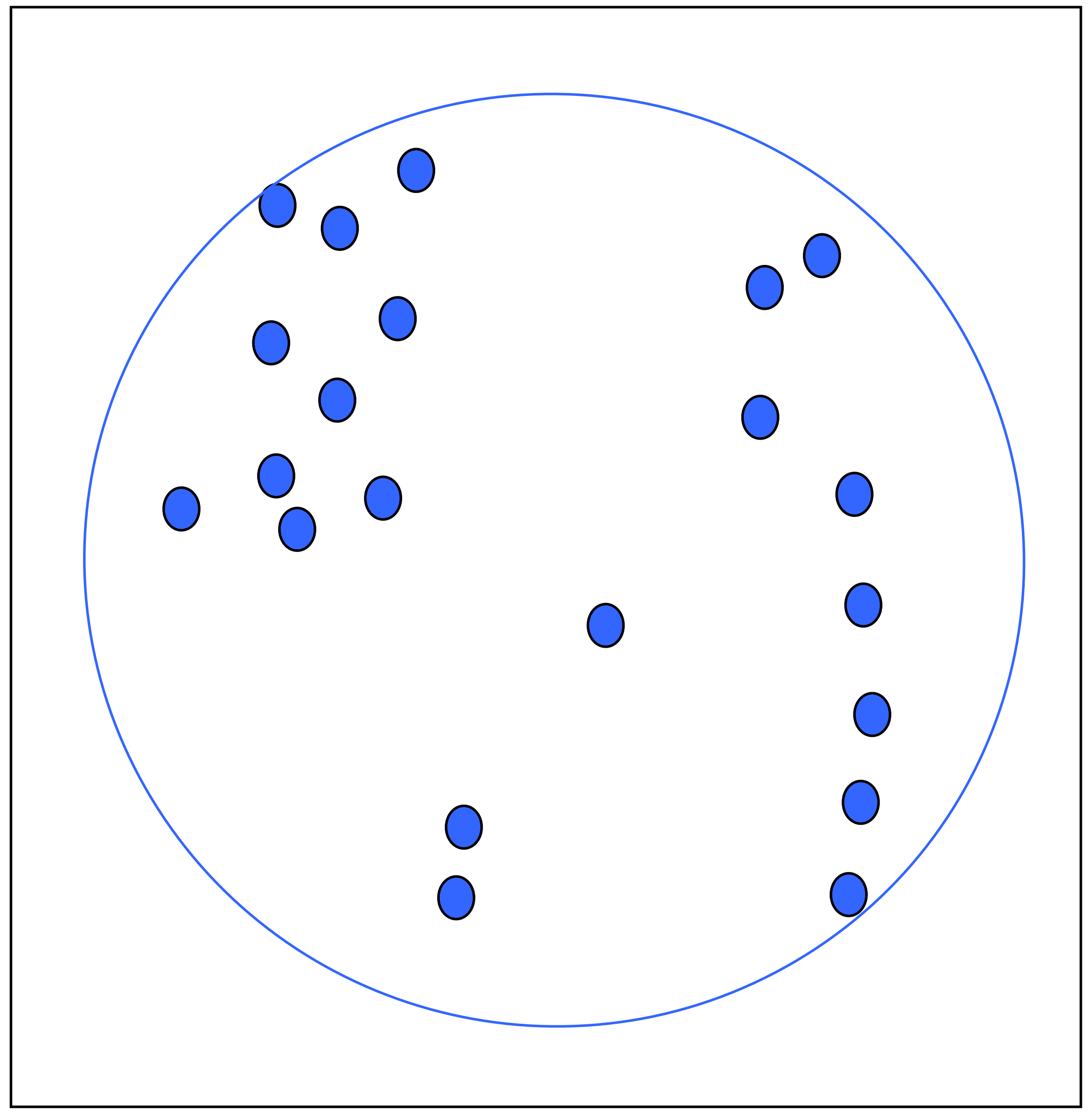
dendrogram



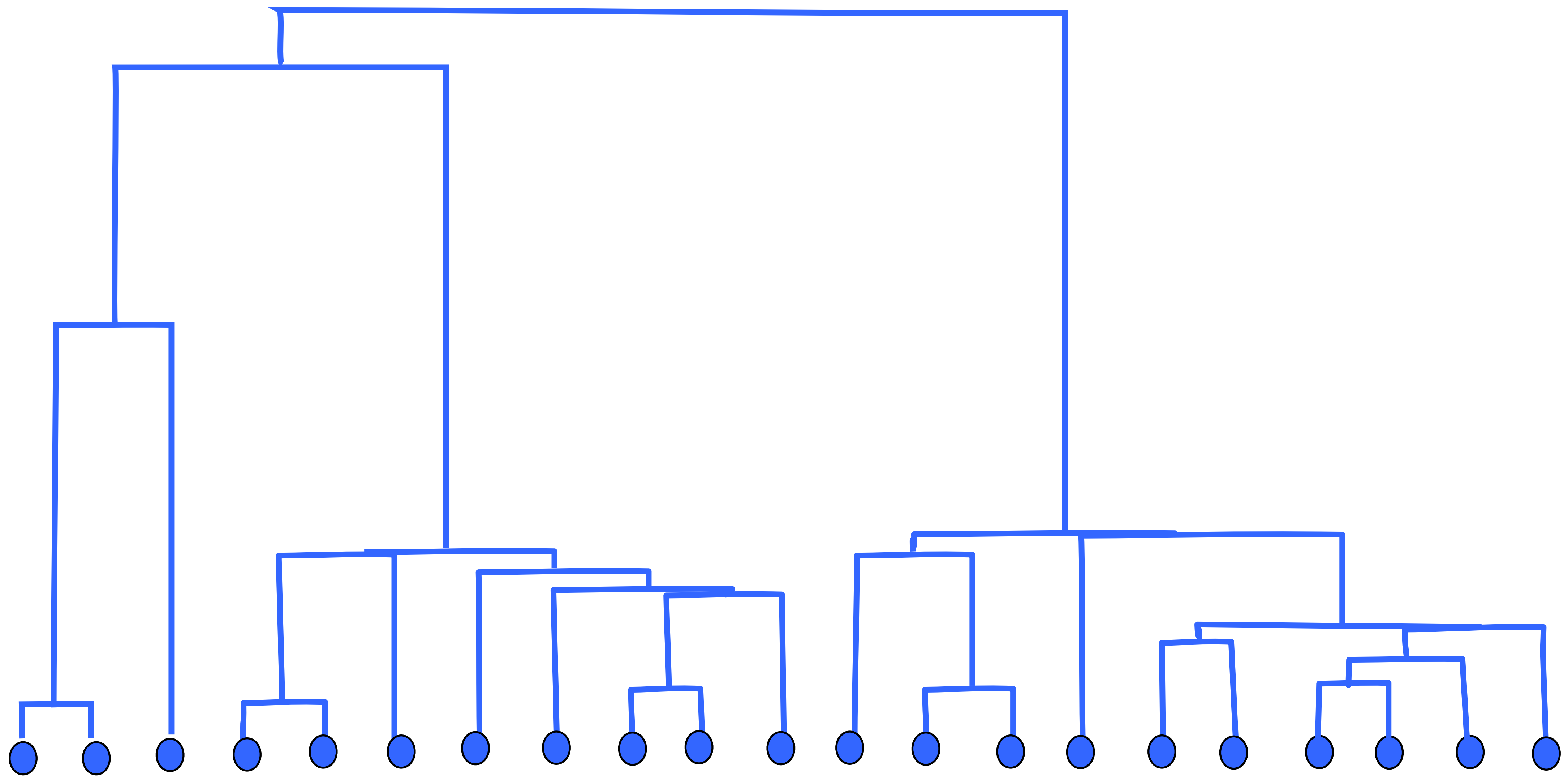
Iteration $m-1$

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

data

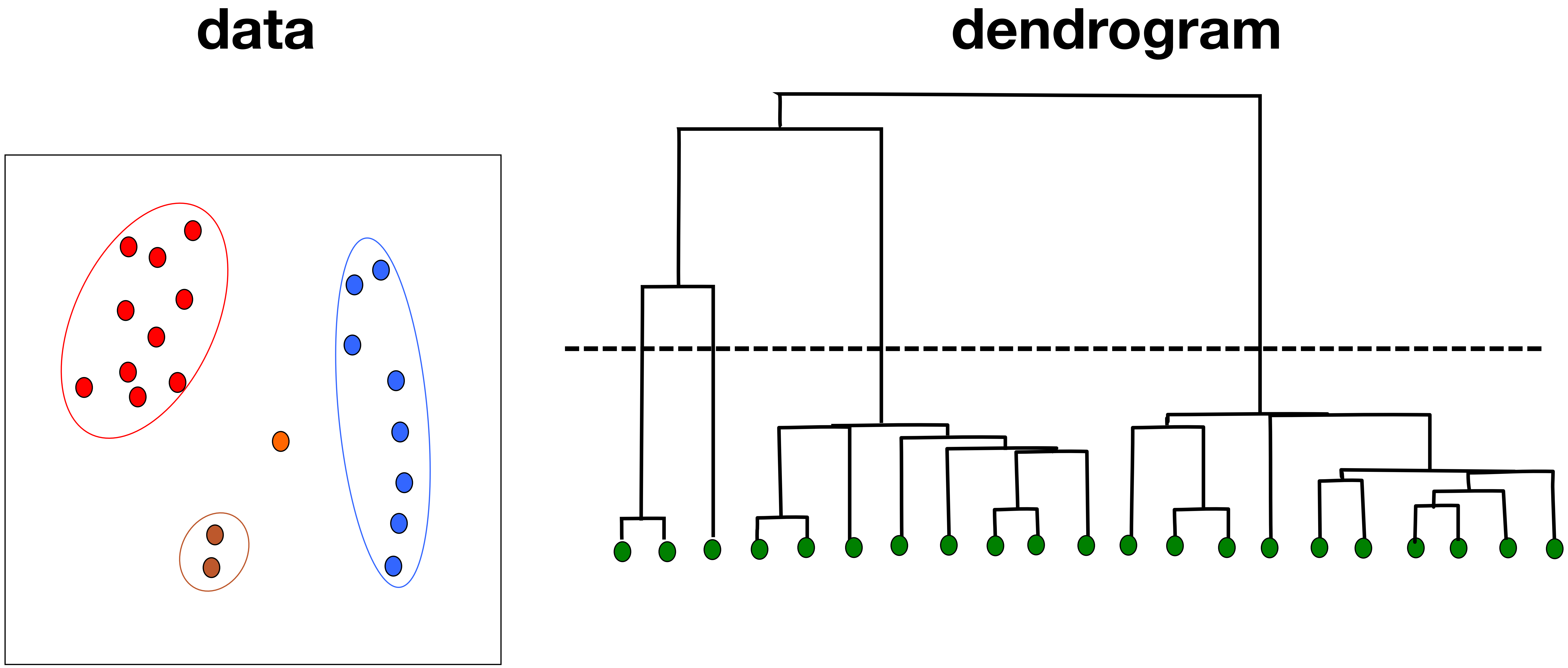


dendrogram



From dendrogram to clusters

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

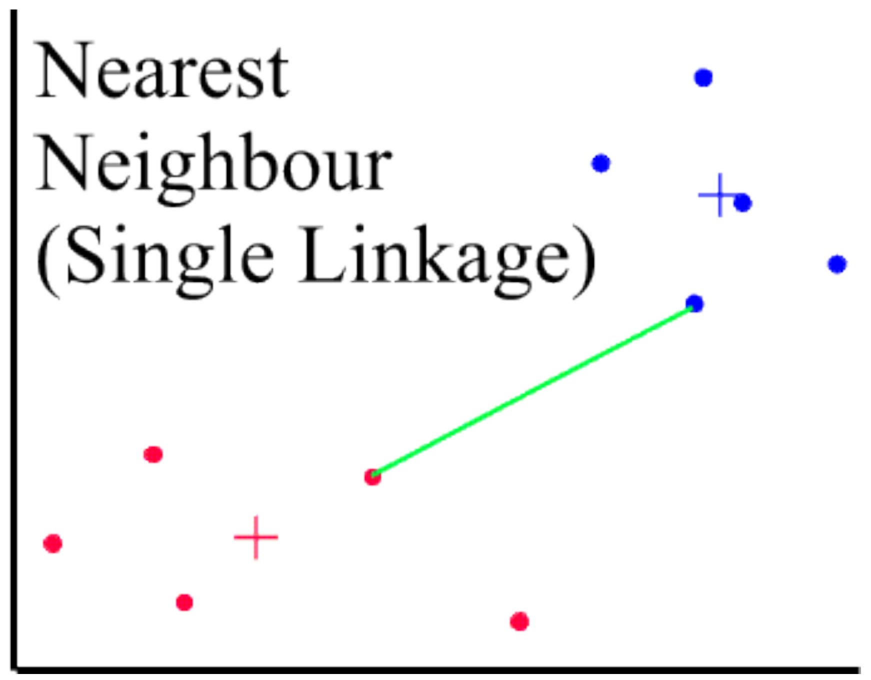


  - For a given k, or a given level of dissimilarity

Distance measures

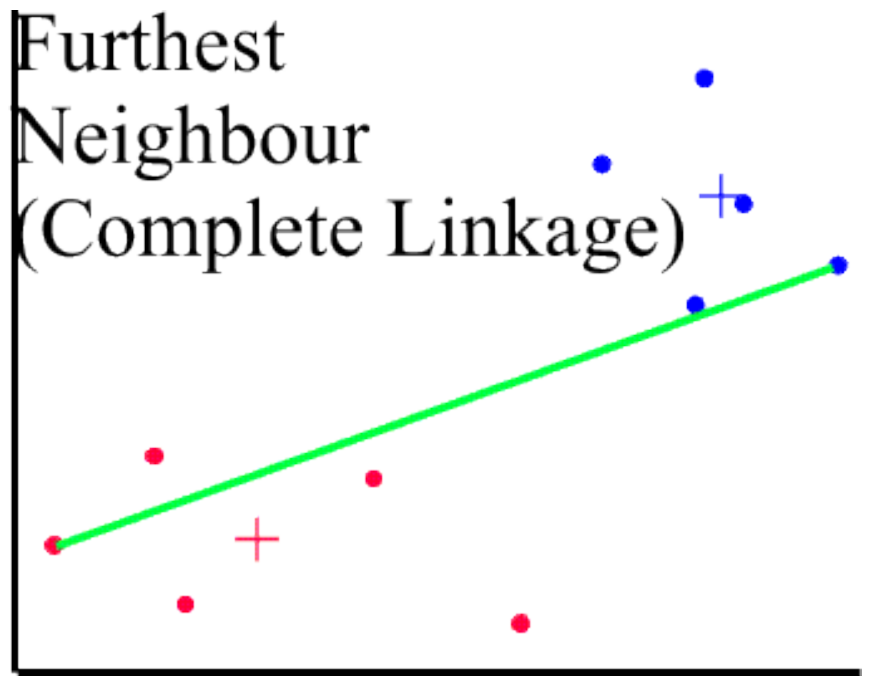
- $d_{\min}(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|^2$

produces minimum spanning tree



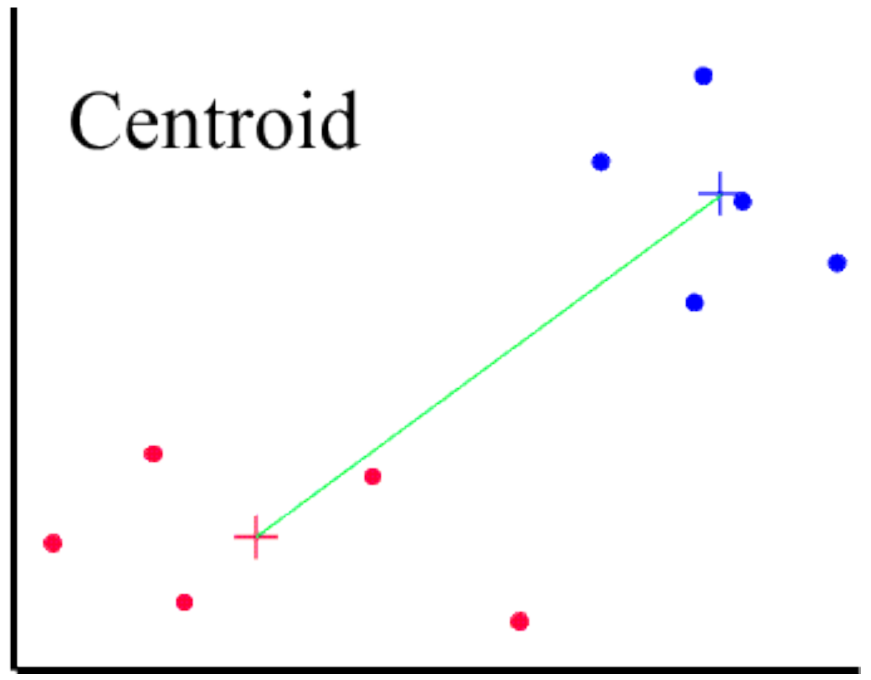
- $d_{\max}(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\|^2$

avoids elongated clusters



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

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

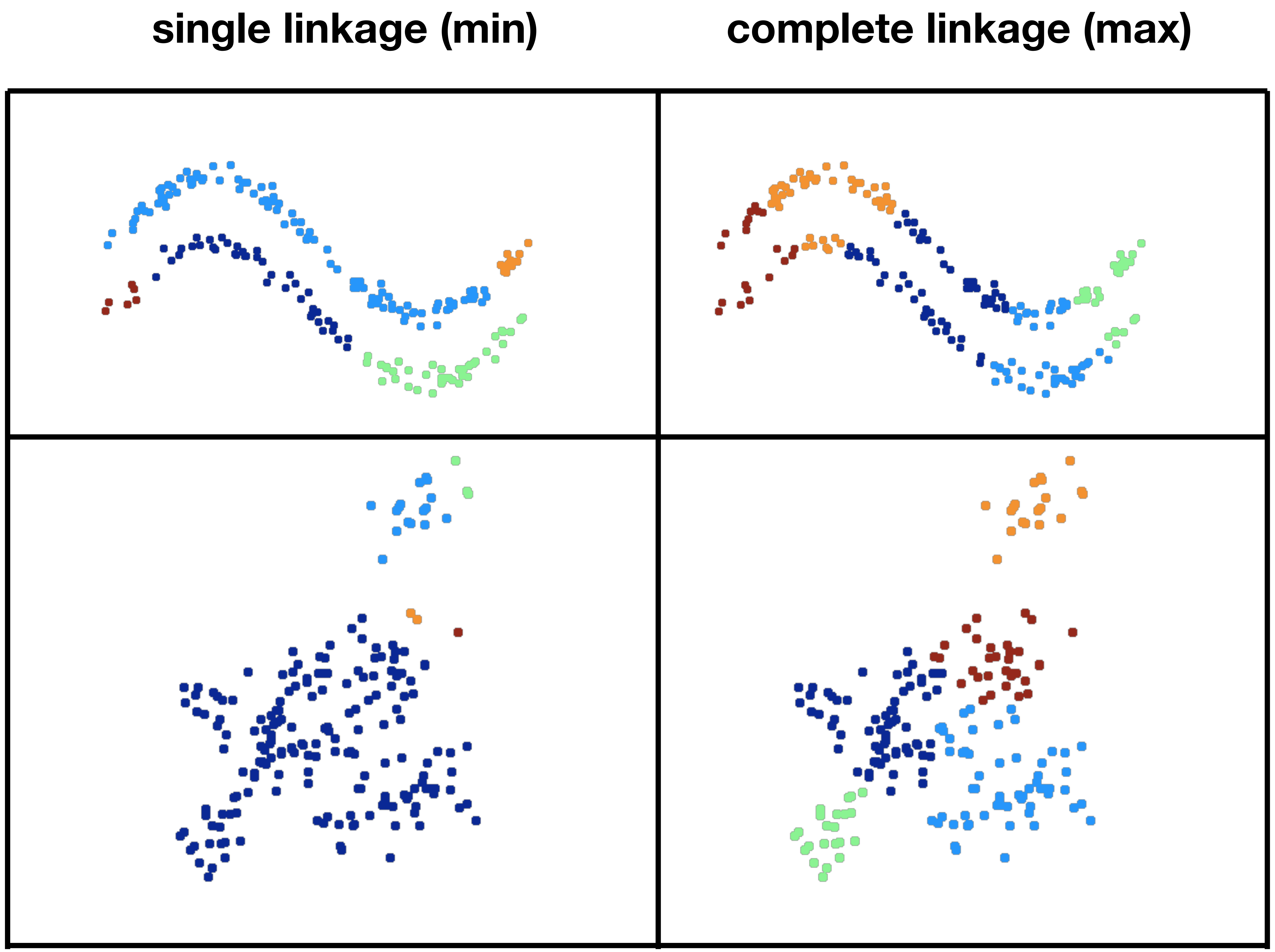


- Important property: iterative computation

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

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

Logistics

project

- Project abstract due today on Canvas

assignments

- Assignment 5 due Tuesday, Nov 23