Machine Learning and Data Mining

Lecture 9.1: Unsupervised Learning I - Clustering

CS 434

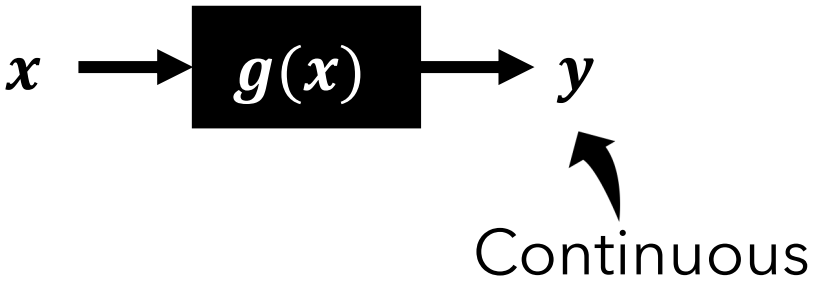
RECAP

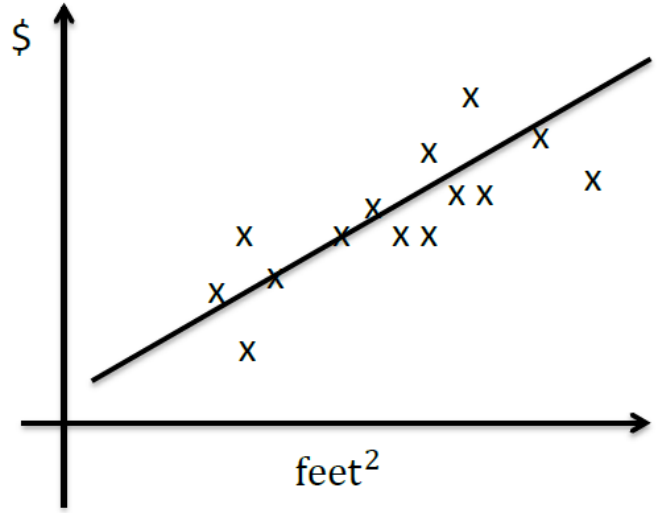
From Last Lecture

So Far, We've Mostly Focused on Supervised Learning

Supervised Learning
Learn to predict output from input given example (x,y) pairs

Regression: Predicting a (or multiple) continuous values as output



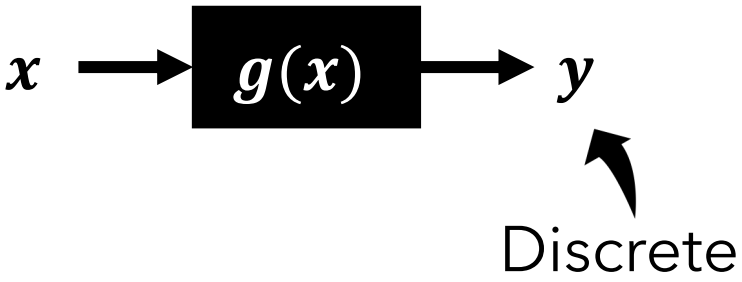


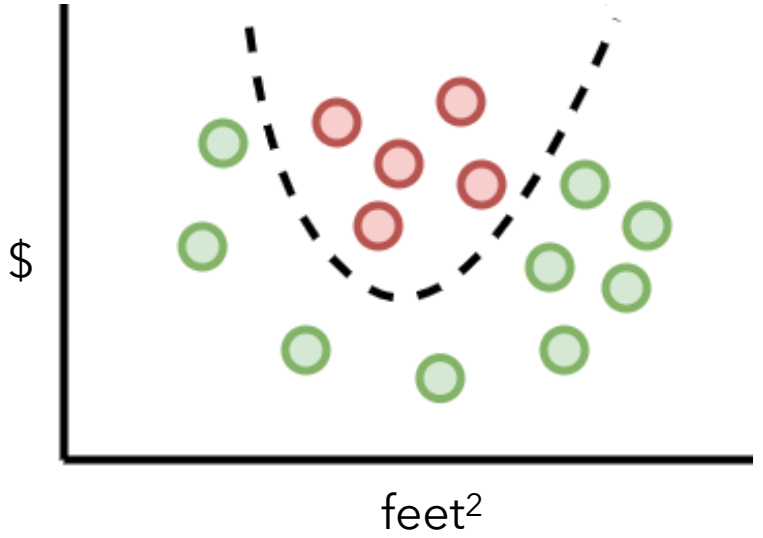
Example: Predicting the price of a house based on square footage

So Far, We've Mostly Focused on Supervised Learning

Supervised Learning
Learn to predict output from input given example (x,y) pairs

Classification: Predicting a (or multiple) discrete variable as output





Example: Predicting if a house will sell based on price and square footage

Moving to Unsupervised Learning

Supervised Learning

So far, our data has looked like:

$$\left(\begin{array}{c|c} x_{11}, x_{12}, \dots, x_{1d} & y_1 \\ x_{21}, x_{22}, \dots, x_{2d} & y_2 \\ \vdots & \vdots \\ x_{n1}, x_{n2}, \dots, x_{nd} & y_d \end{array}\right)$$

Features | Label

Goal: Prediction

Unsupervised Learning

Now, we'll consider **unlabeled data**

$$\left(\begin{array}{c} x_{11}, x_{12}, \dots, x_{1d} \\ x_{21}, x_{22}, \dots, x_{2d} \\ \vdots \\ x_{n1}, x_{n2}, \dots, x_{nd} \end{array}\right)$$

Features

Goal: Discovery

Today's Learning Objectives

Be able to answer:

- What is unsupervised learning and how does it differ form supervised learning?
  - What are some problems in unsupervised learning?

- What is clustering?

- How does k-means clustering work?
- What is k-means optimizing?
  - What is a coordinate descent algorithm?

- What is a Gaussian Mixture Model?
- What can this tell us about k-Means?

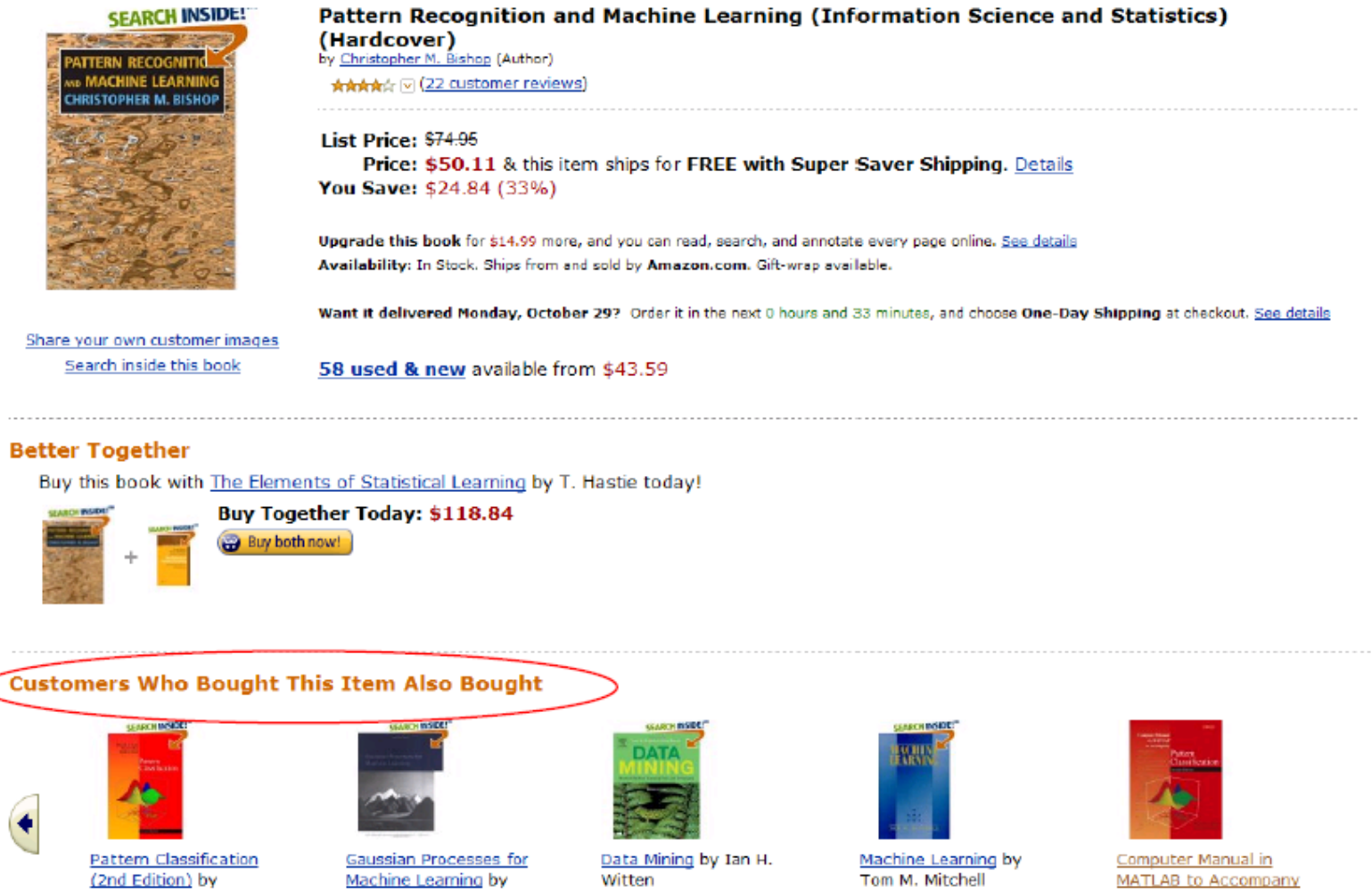
- How do we evaluate clustering?

What can we do without labels?

Organization: Find subgroups within the data that are self-similar. Why?

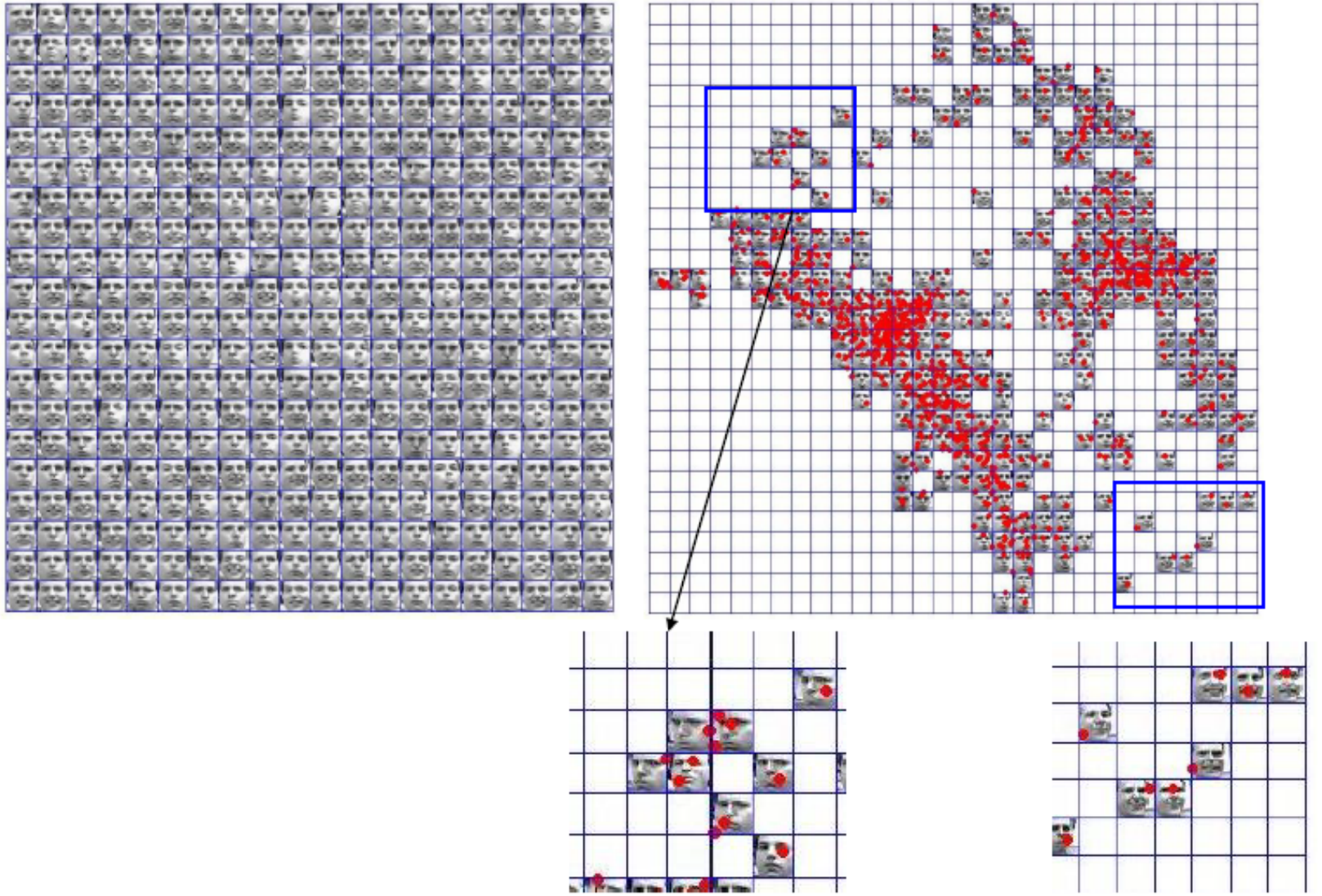
What can we do without labels?

Understanding: Figure out regular trends in the data



What can we do without labels?

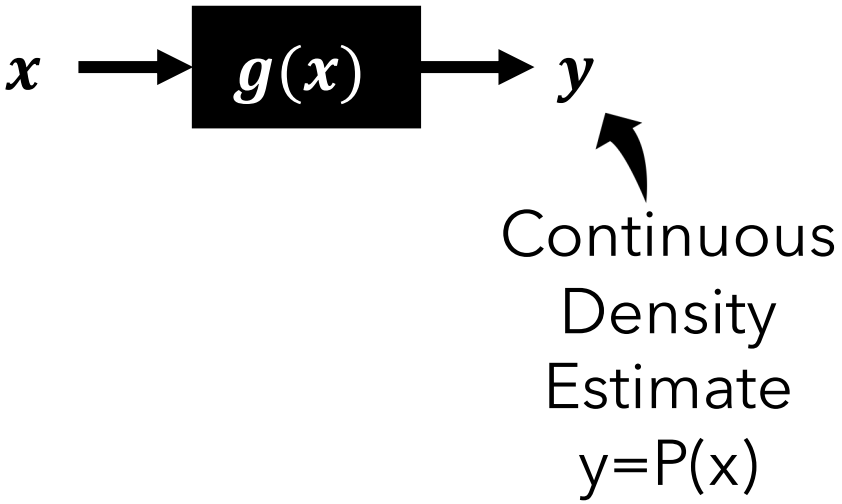
Representation: Find subspaces that represent that data well

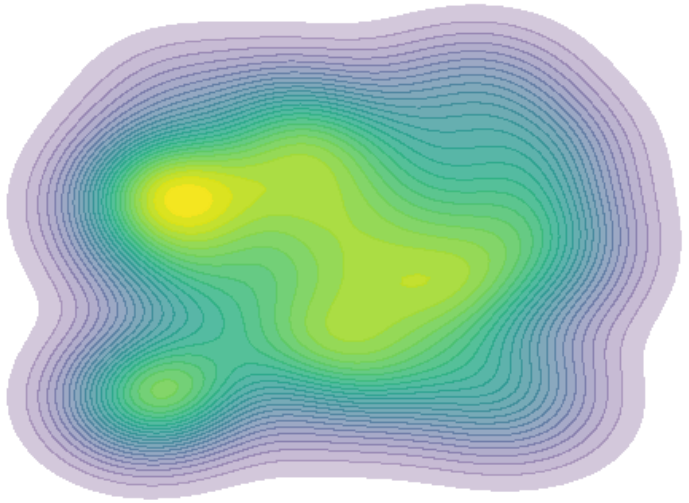


Now, We Are Turning Towards Unsupervised Learning

Unsupervised Learning
Only given a set of input instances (x)

Density Estimation: Estimate the underlying distribution generating our data



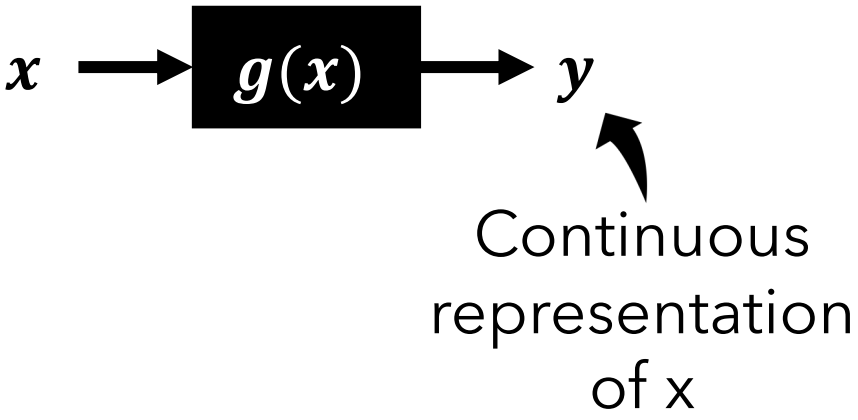


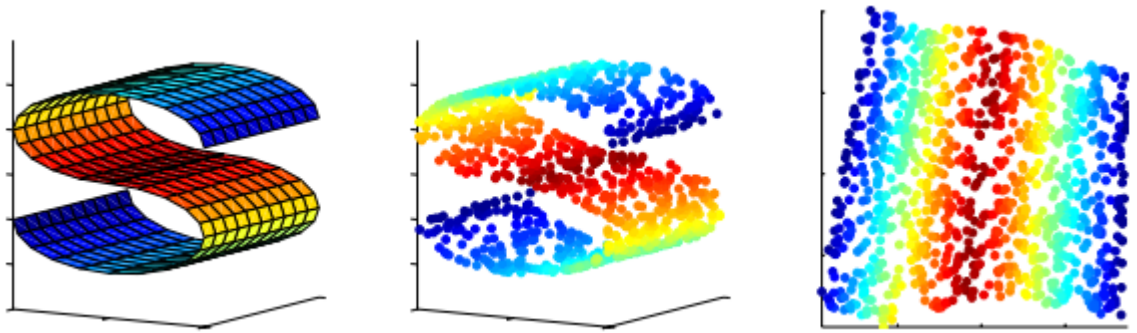
Example: Estimate how likely is a given house with these properties

Now, We Are Turning Towards Unsupervised Learning

Unsupervised Learning
Only given a set of input instances (x)

Dimensionality Reduction: Represent high-dim data as low-dim data



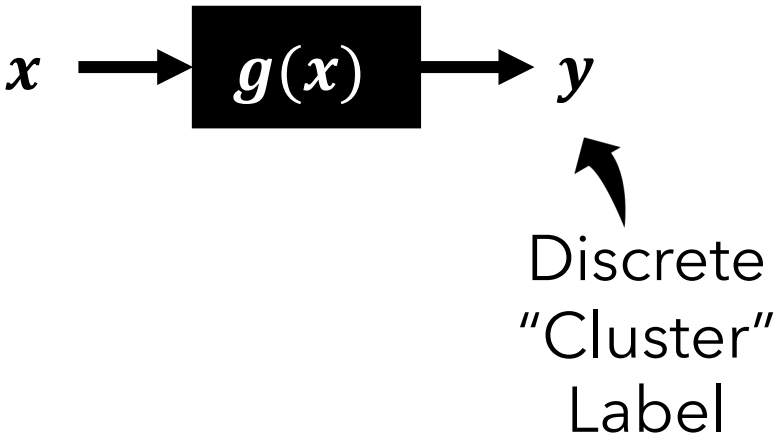


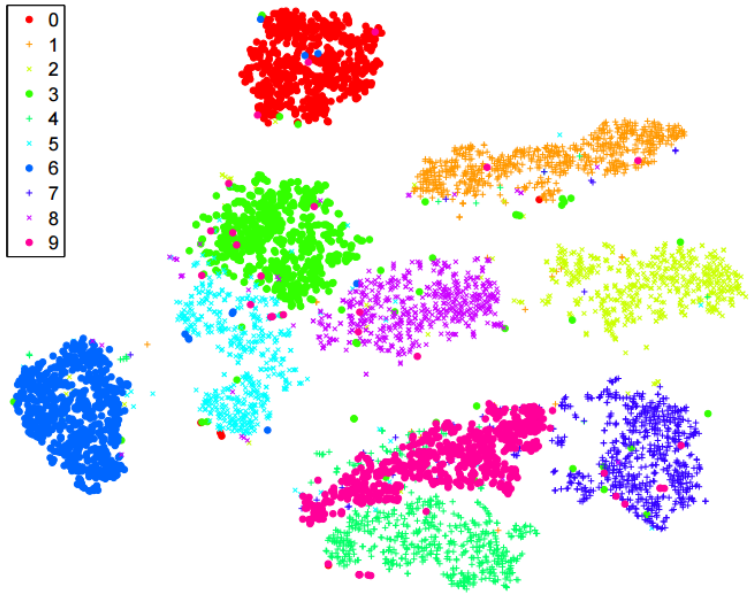
Example: Finding a 2D subspace in a 3D feature space

Now, We Are Turning Towards Unsupervised Learning

Unsupervised Learning
Only given a set of input instances (x)

Clustering: Discover self-similar groups within the data





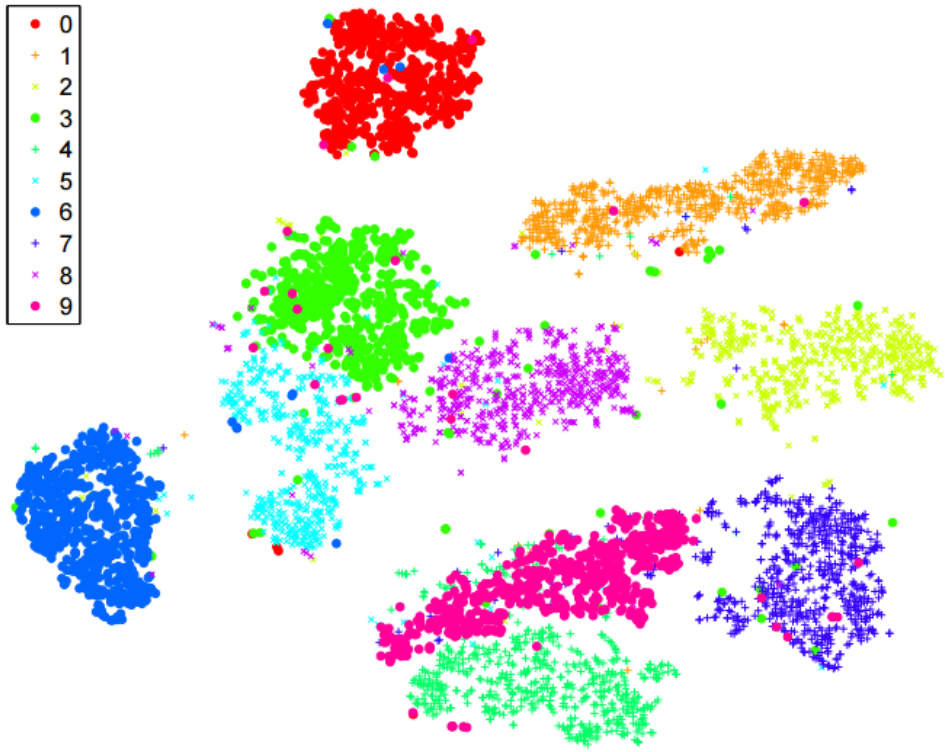
Example: Find groups of houses with similar properties

New Topic: Clustering

In general, clustering can be viewed as an exploratory procedure for finding interesting subgroups in a dataset.

Most work in clustering is on the setting where we want to:

- Group all given examples **(exhaustive)** into disjoint clusters **(partitional)** such that
  - Examples within a cluster are similar
  - Examples in different clusters are different



Slide adapted from Xiaoli Fern

Our First Clustering Algorithm: **k-Means**

Conceptual Overview of k-Means:

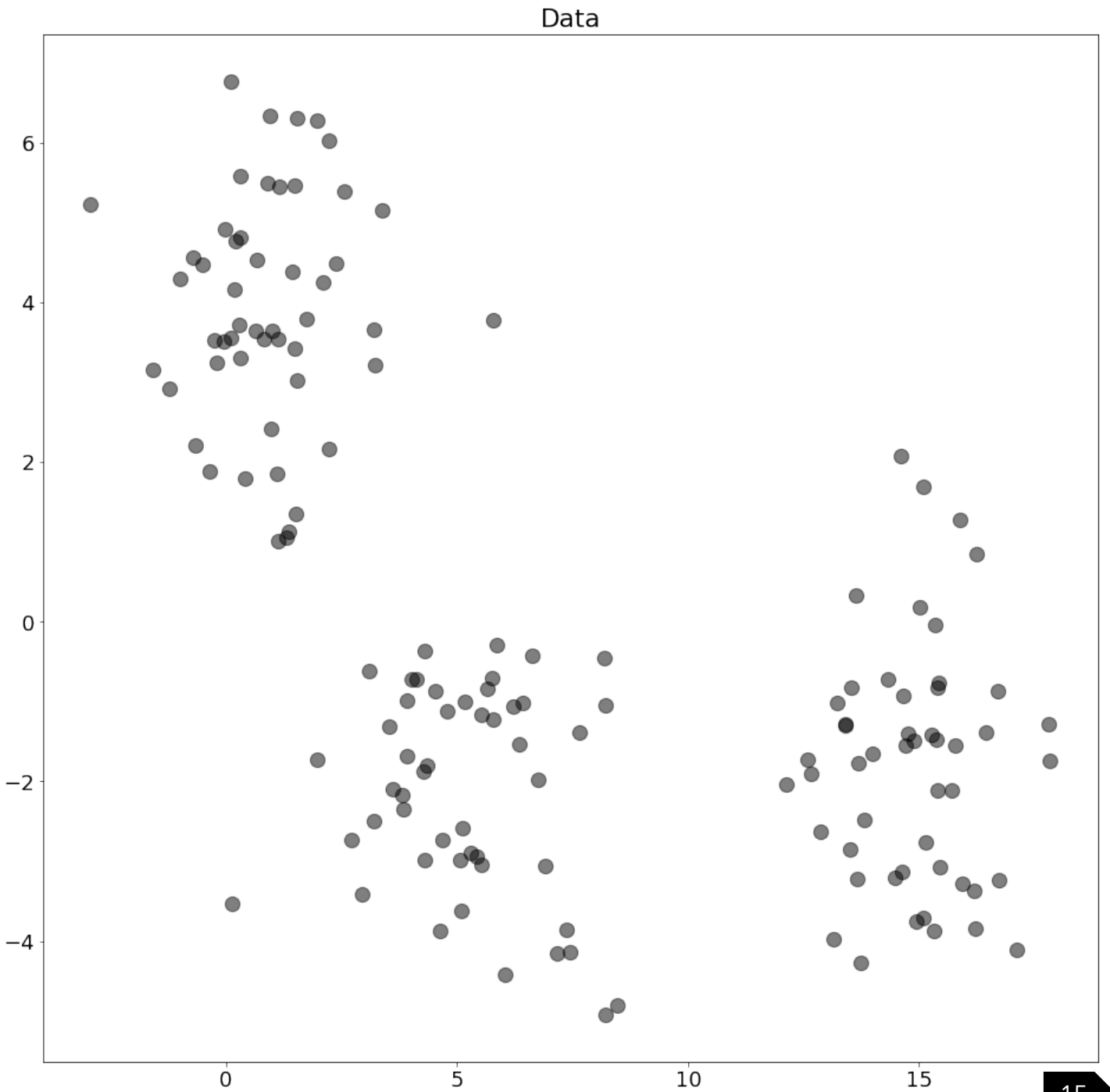
- Start with random initial points to represent the center of the k clusters.
- Form initial clusters based on these random starting points.
- Iteratively refine these clusters and stop when no longer making changes.

Hyperparameters:

- k – the number of clusters
- Some distance function, we'll assume Euclidean (aka L2) for now

K-Mean Clustering

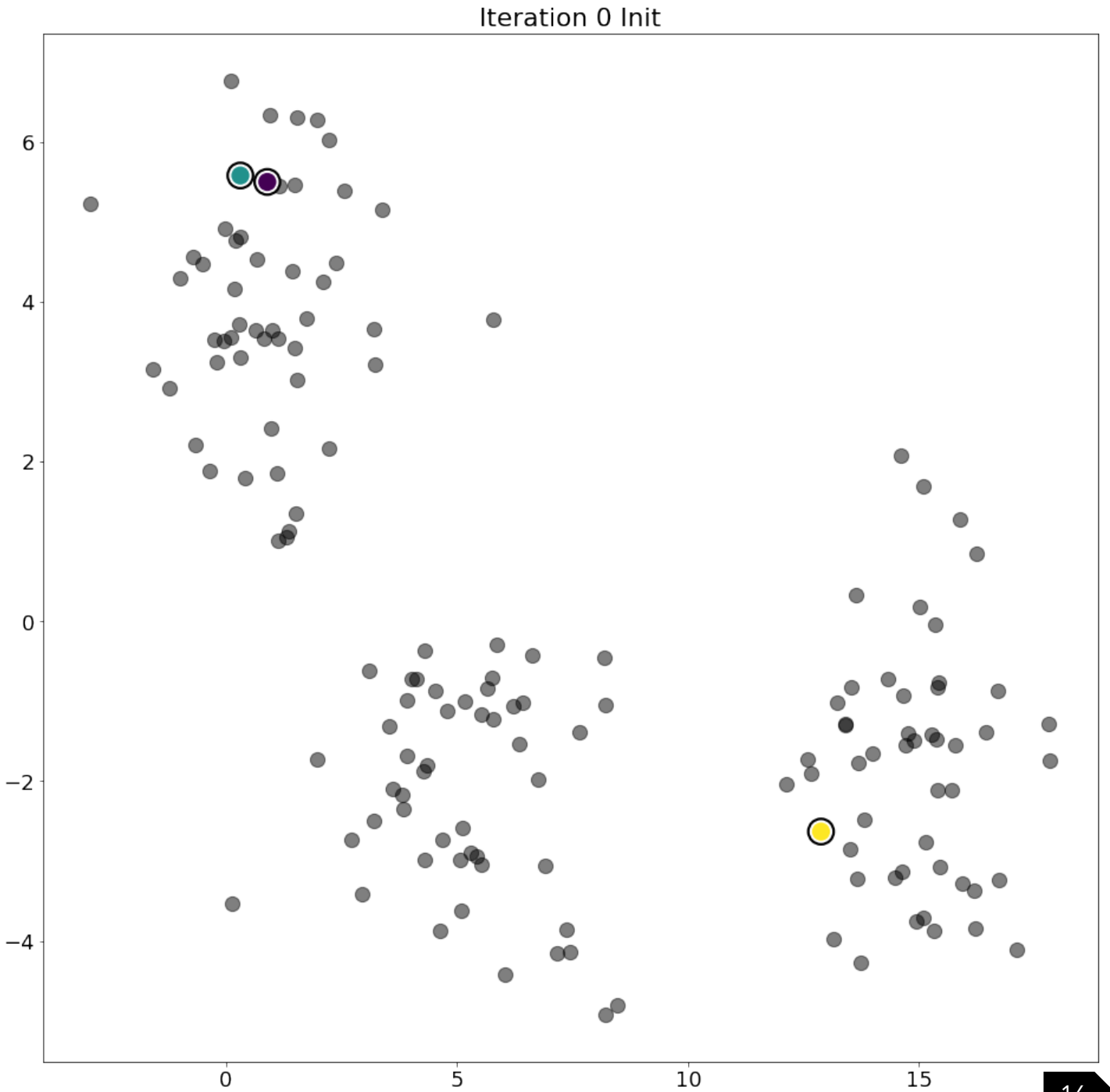
Given some data, find clusters.



K-Mean Clustering

Algorithm:

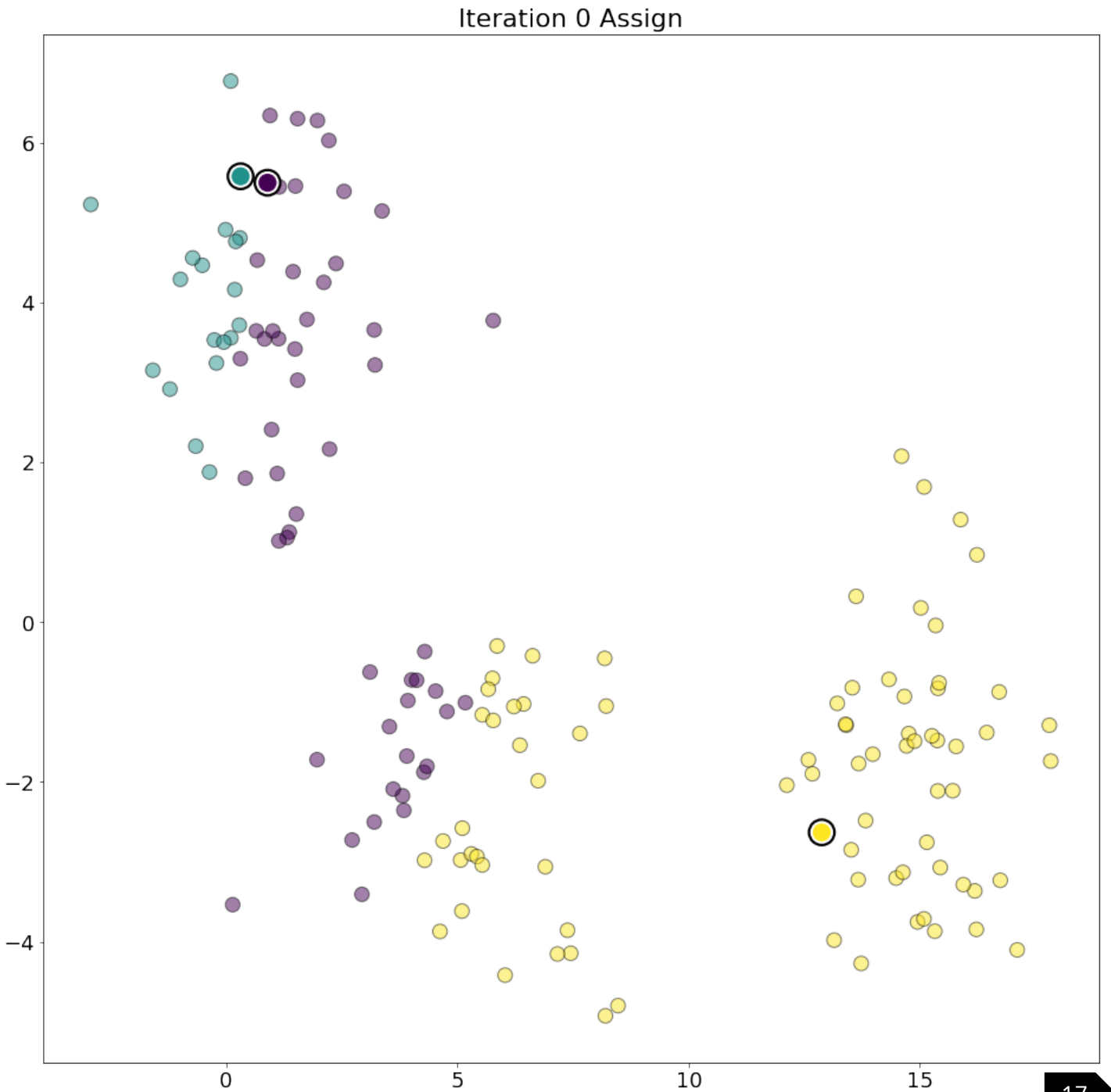
1. Initialize *k* centroids randomly



K-Mean Clustering

Algorithm:

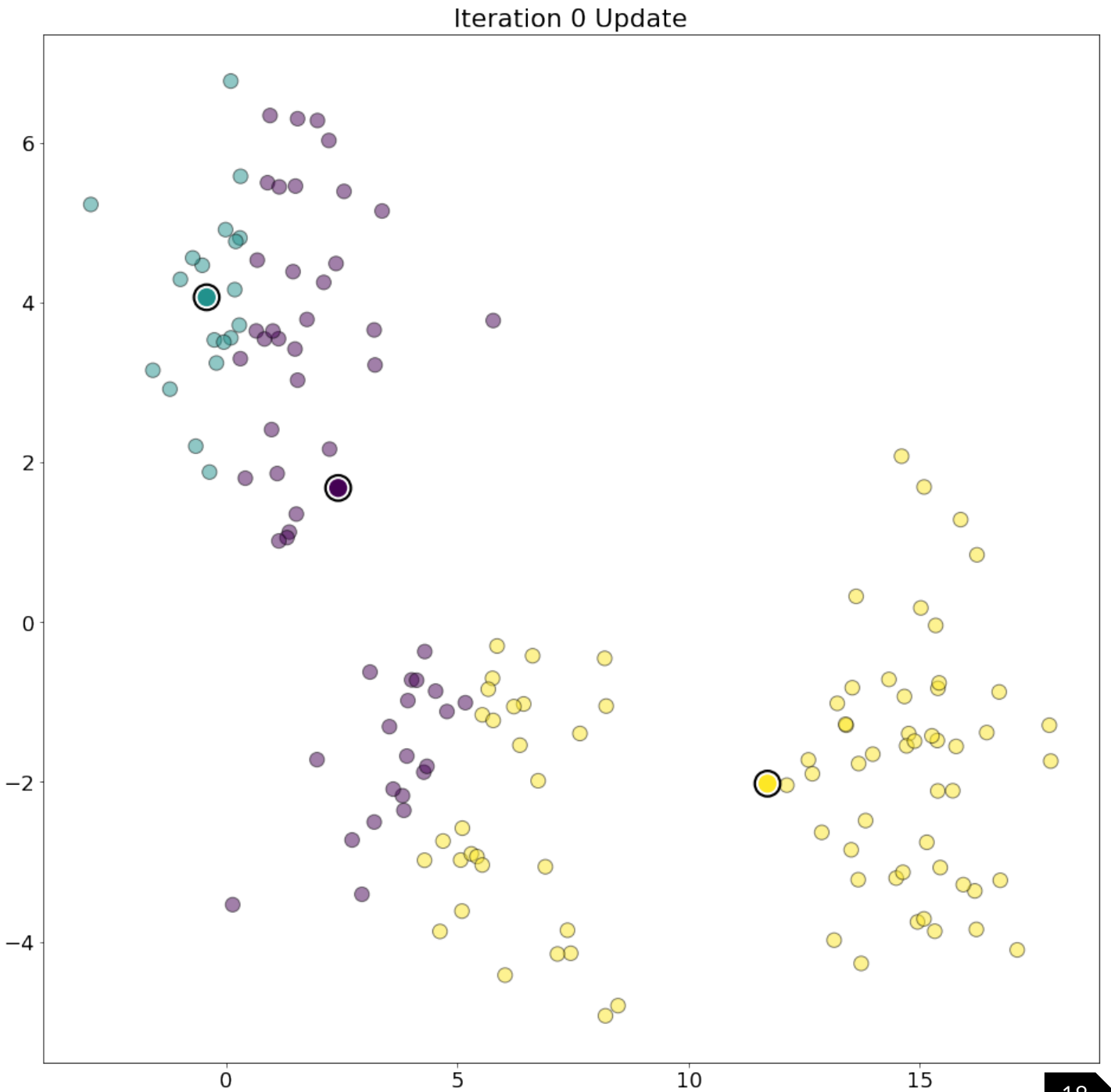
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid



K-Mean Clustering

Algorithm:

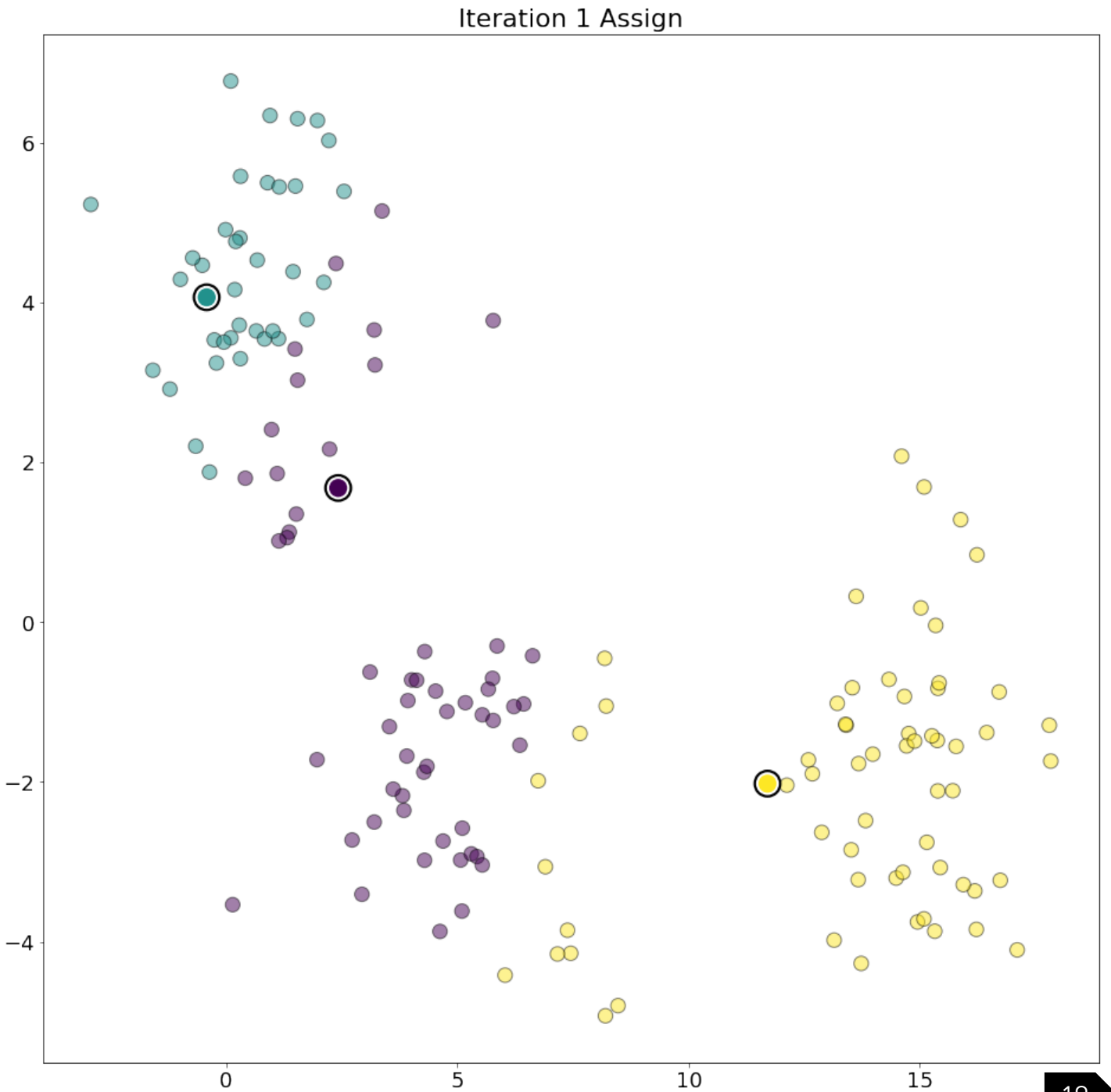
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid
   b) Update each centroid as the average of associated points



K-Mean Clustering

Algorithm:

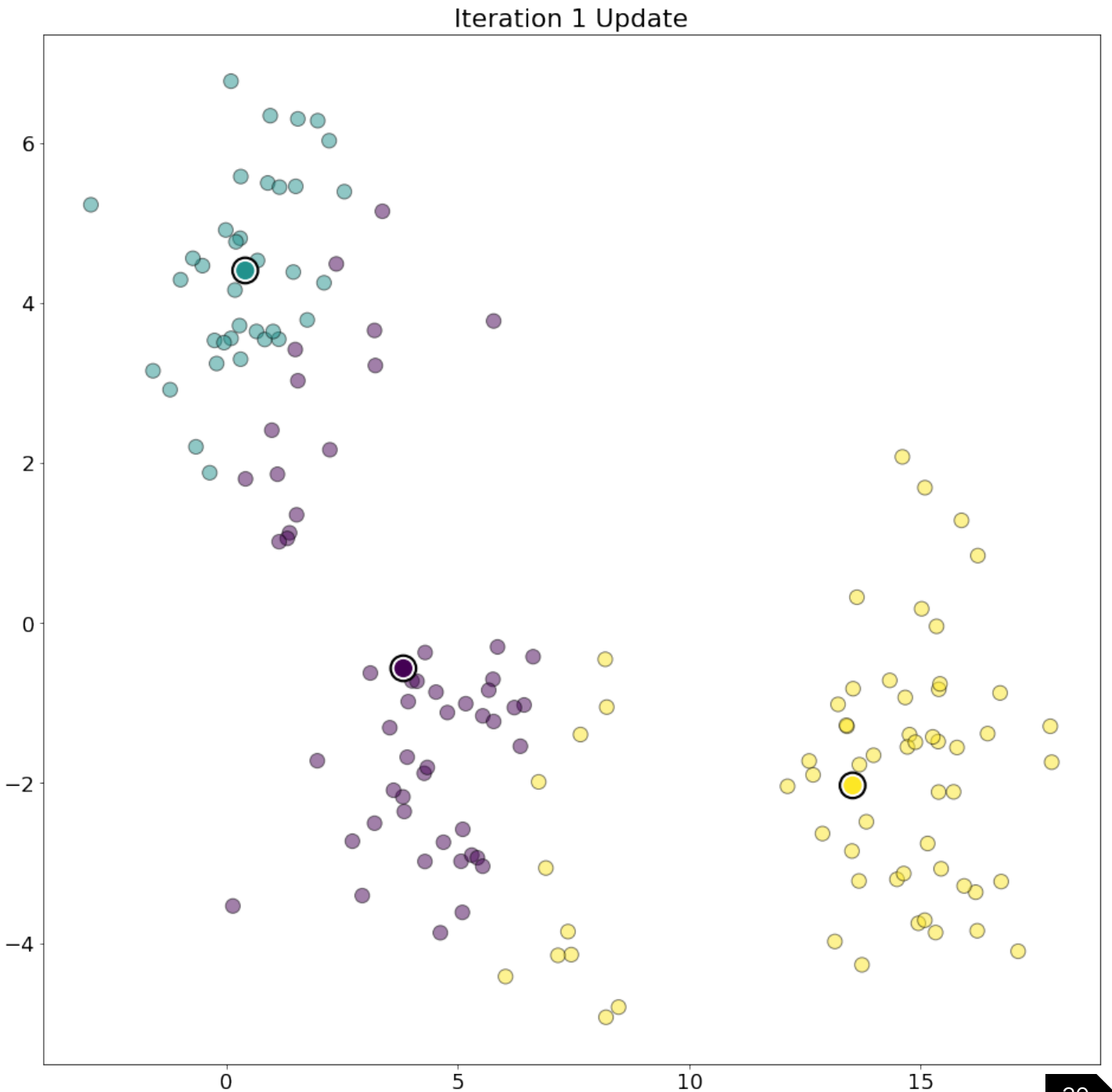
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid
   b) Update each centroid as the average of associated points



K-Mean Clustering

Algorithm:

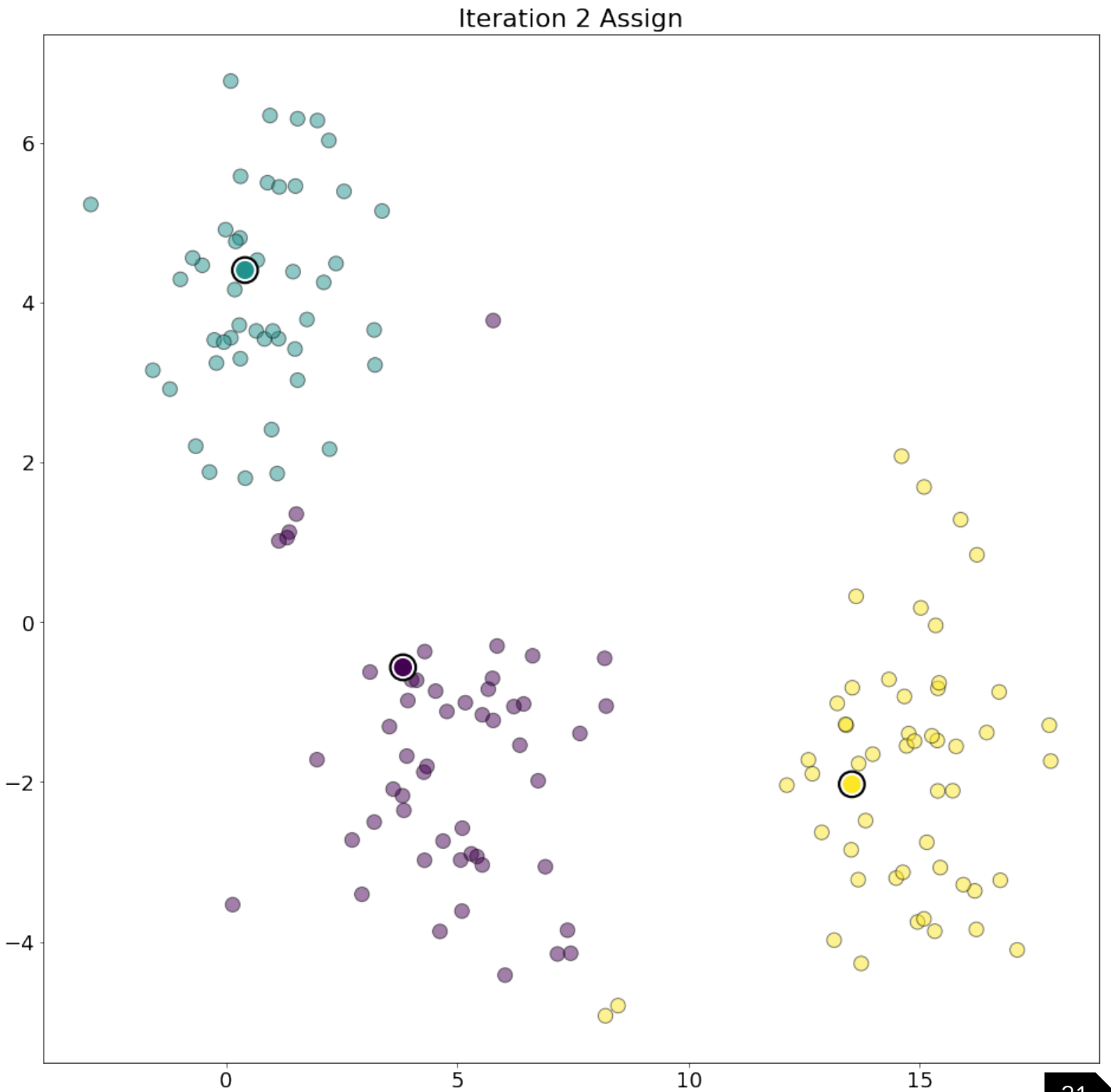
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid
   b) Update each centroid as the average of associated points



K-Mean Clustering

Algorithm:

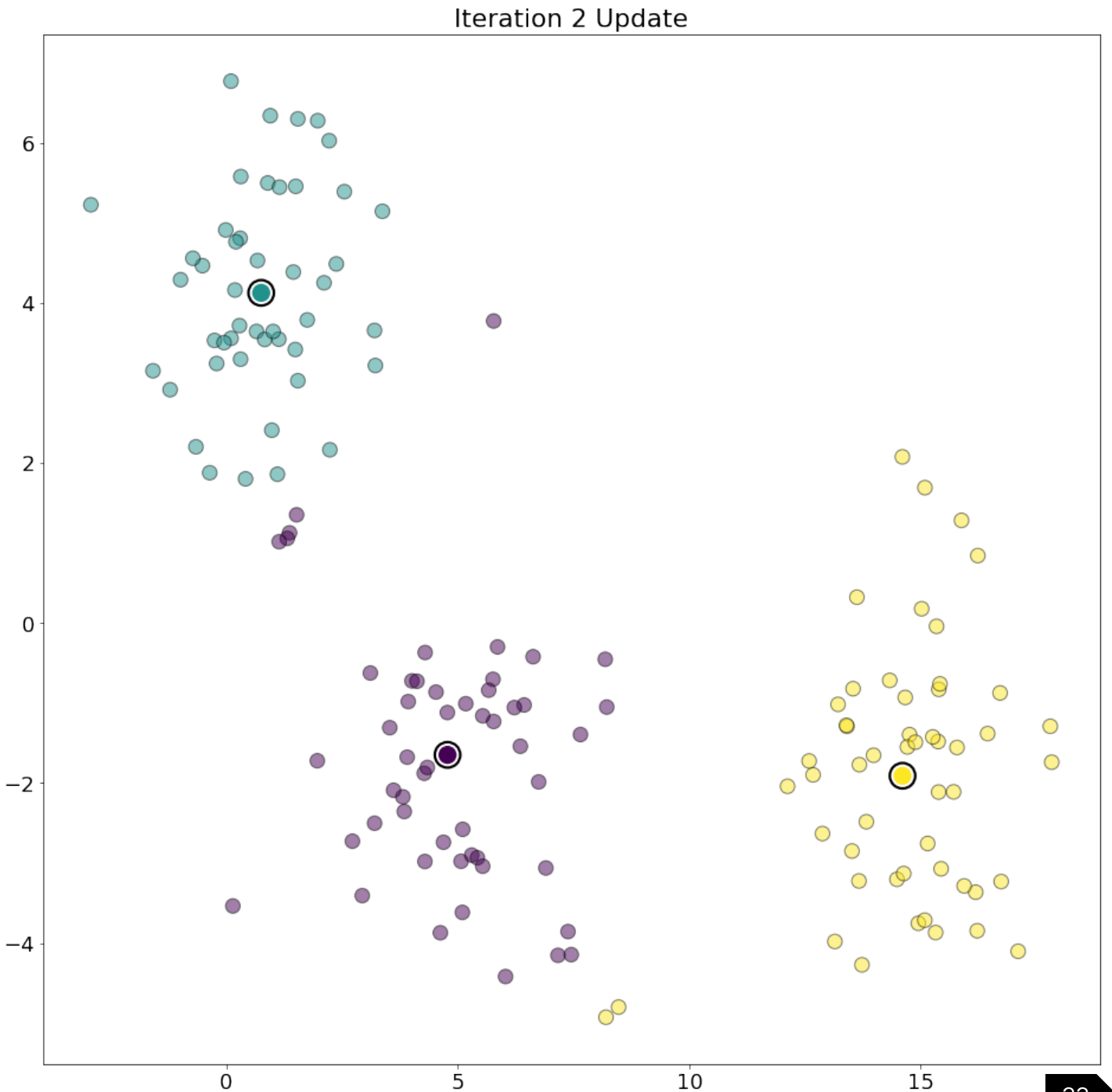
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid
   b) Update each centroid as the average of associated points



K-Mean Clustering

Algorithm:

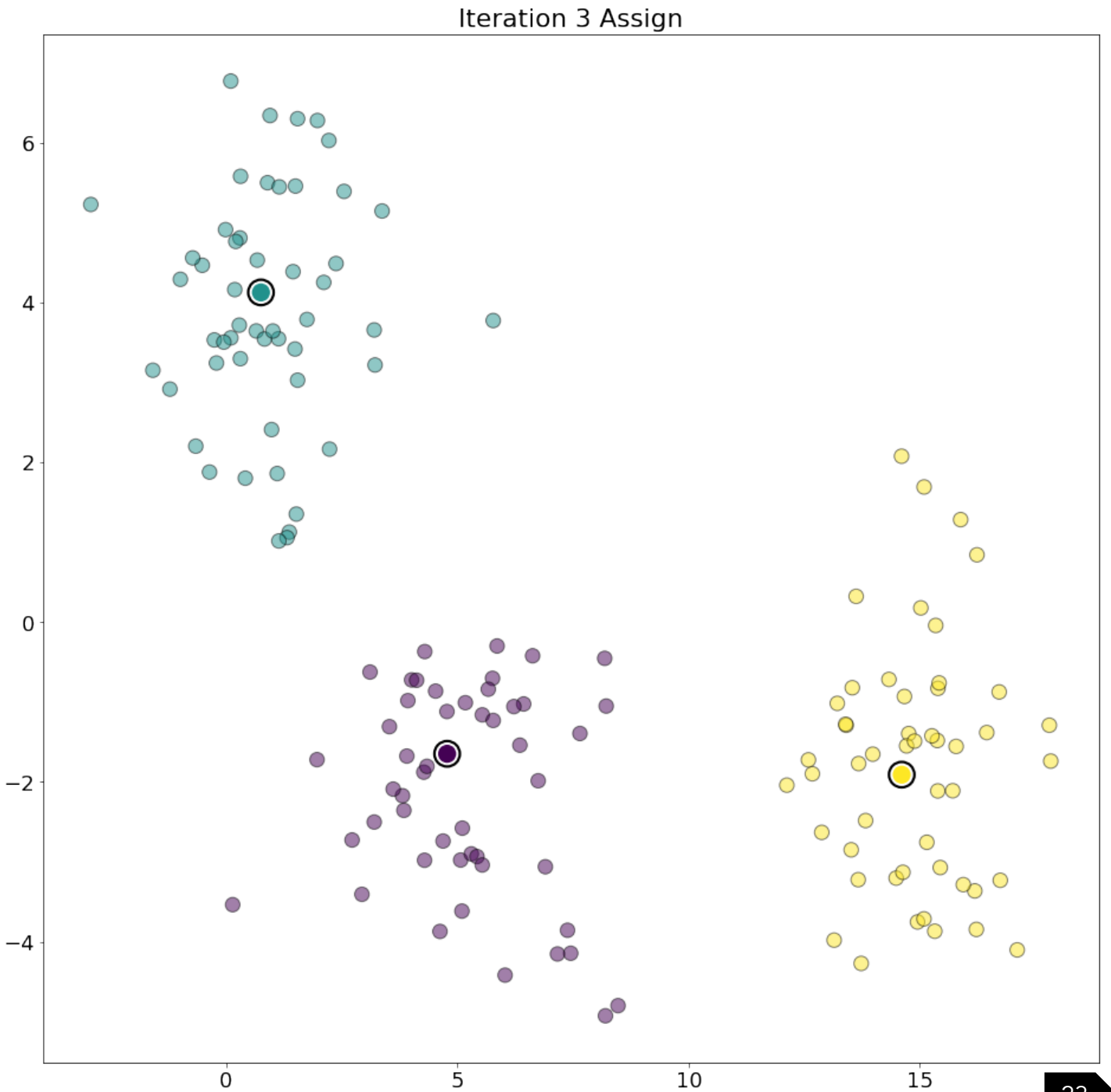
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid
   b) Update each centroid as the average of associated points



K-Mean Clustering

Algorithm:

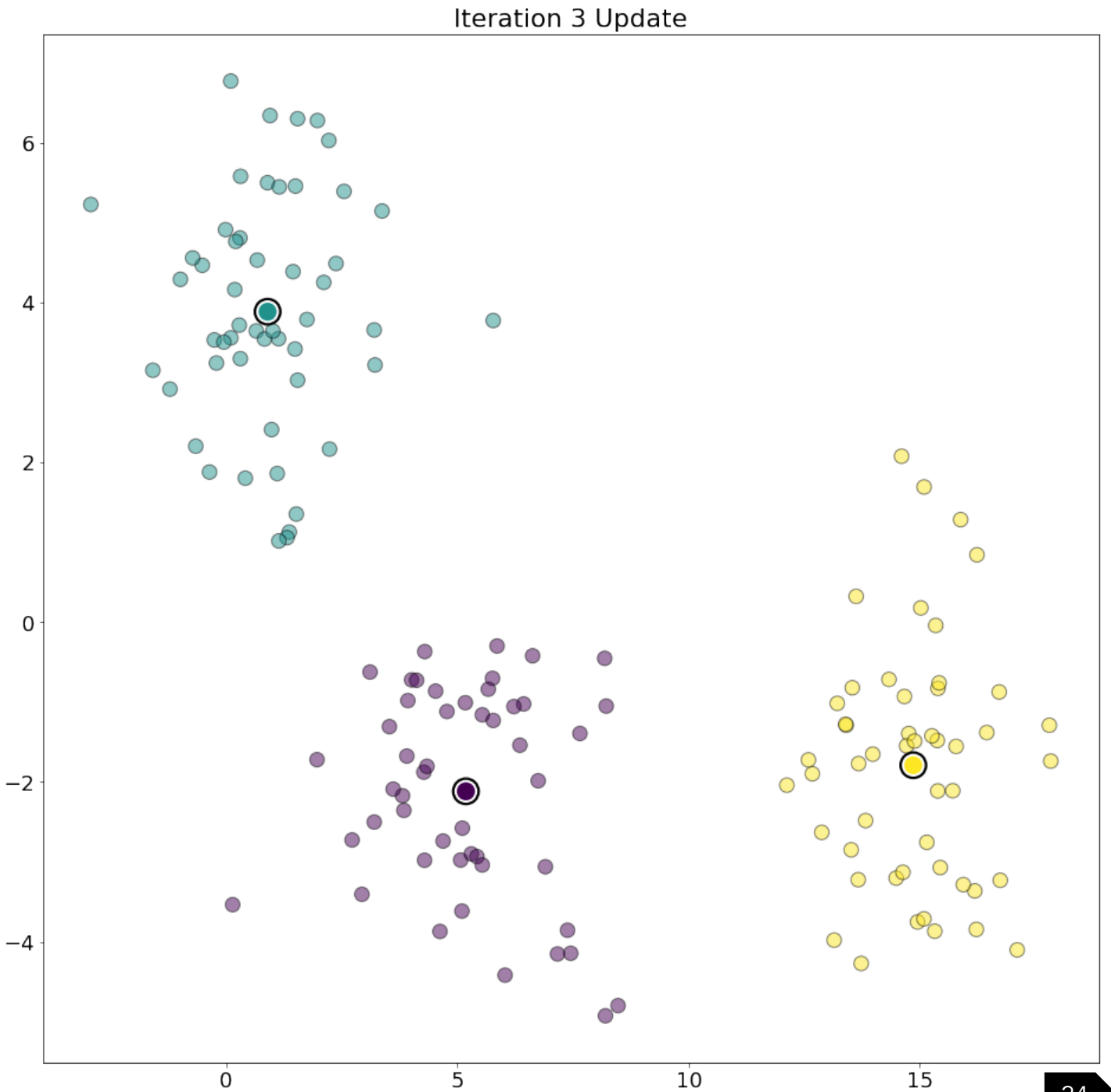
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid
   b) Update each centroid as the average of associated points



K-Mean Clustering

Algorithm:

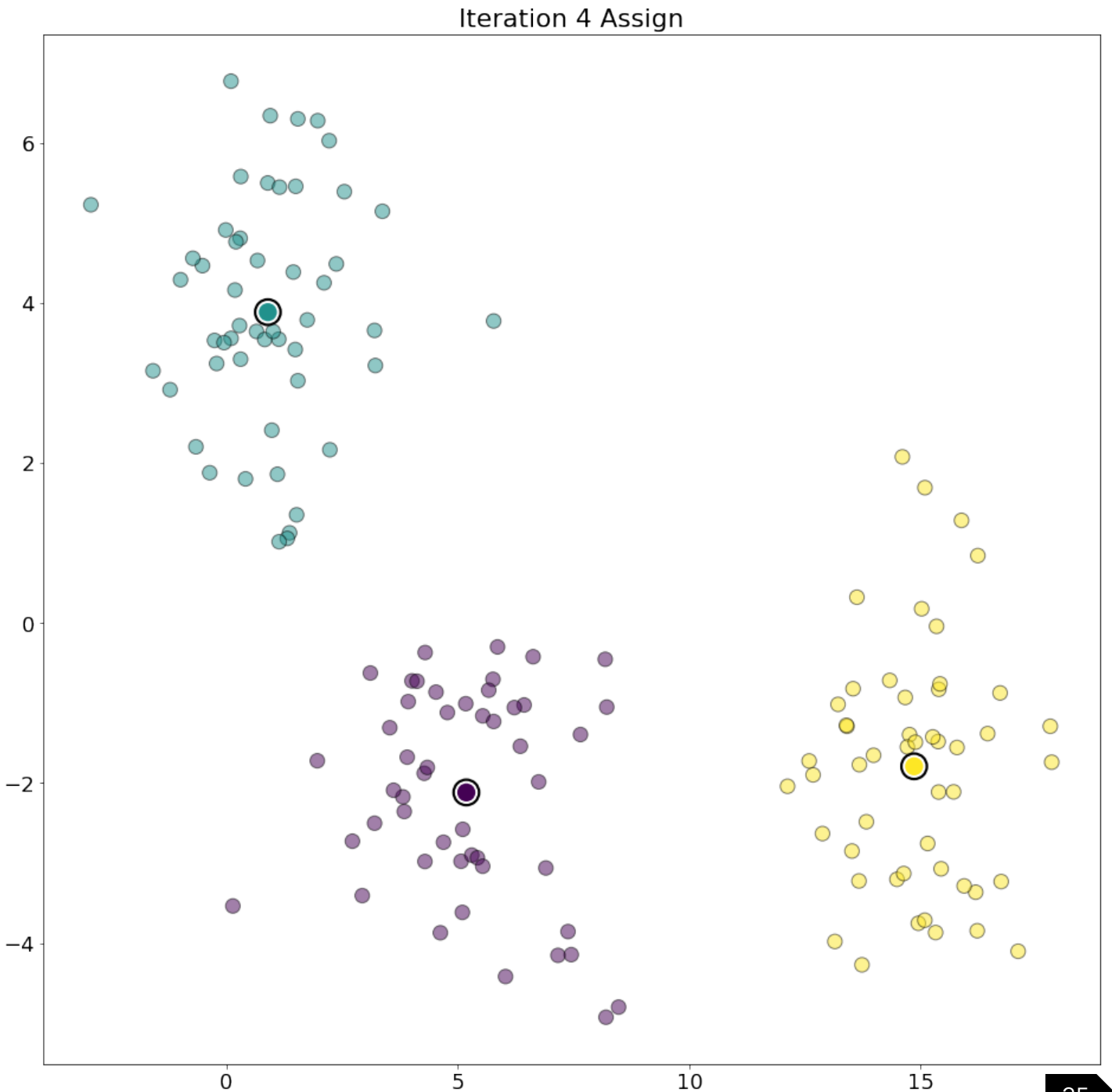
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid
   b) Update each centroid as the average of associated points



K-Mean Clustering

Algorithm:

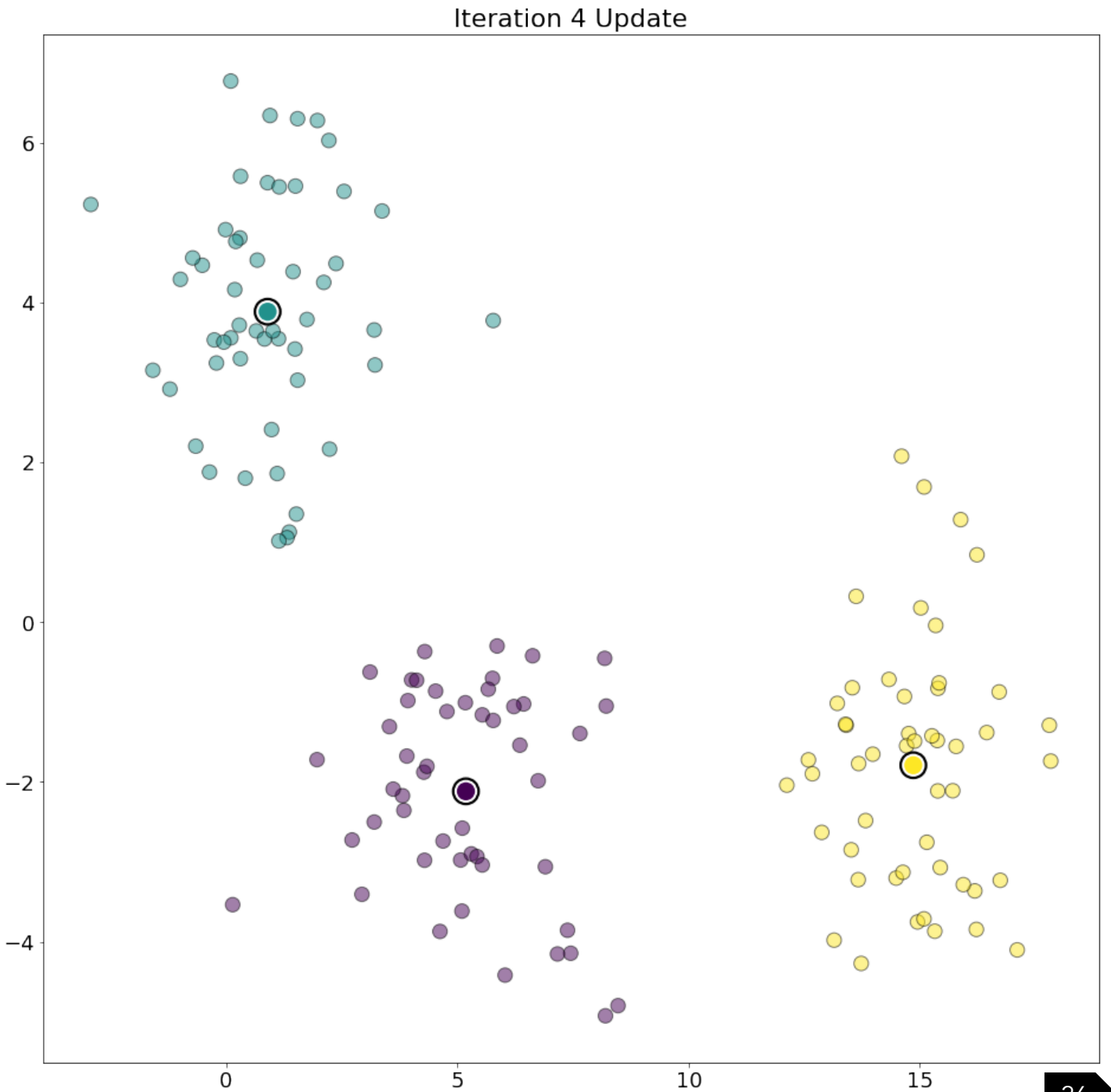
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid
   b) Update each centroid as the average of associated points



K-Mean Clustering

Algorithm:

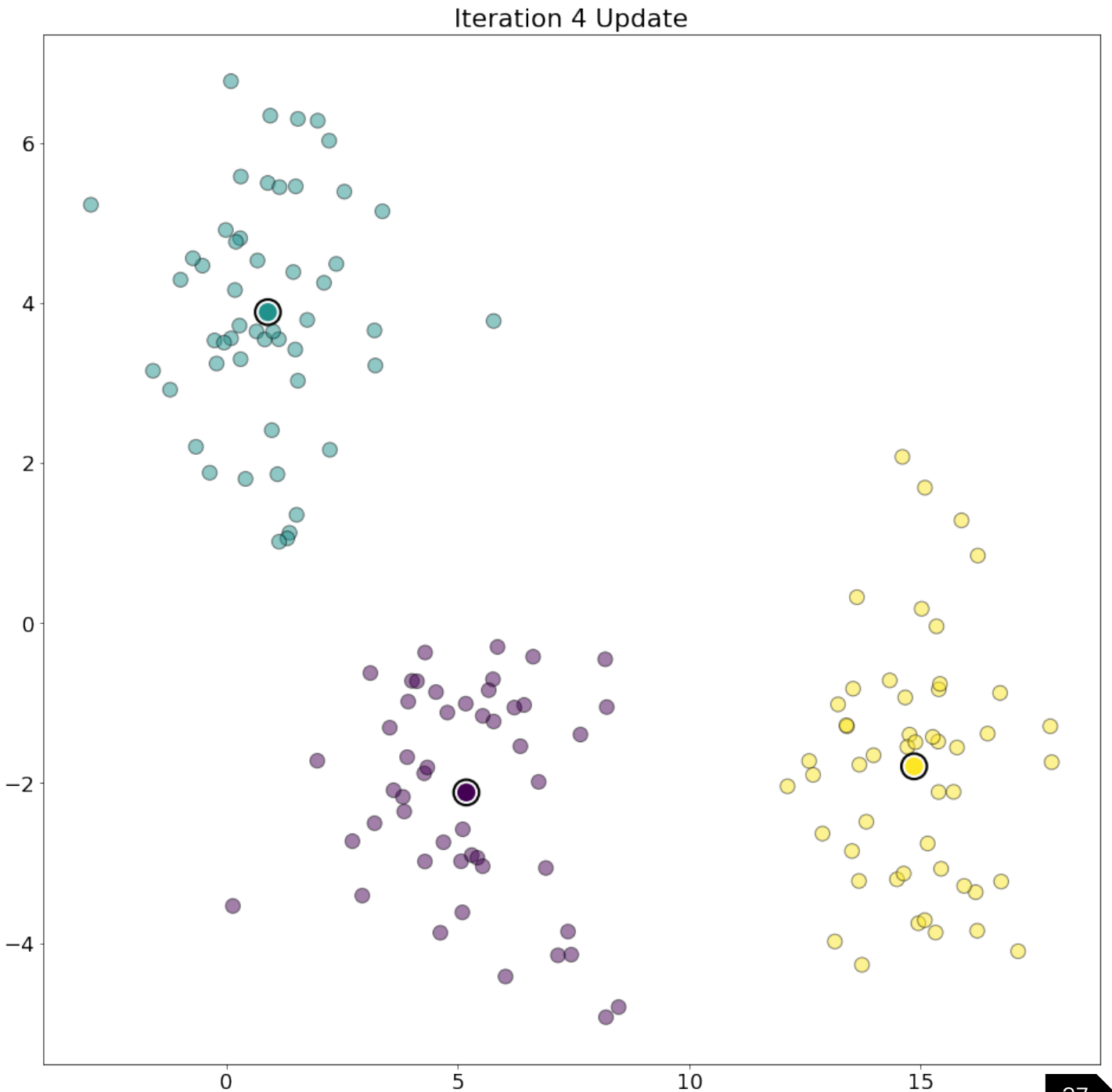
1. Initialize *k* centroids randomly
2. While not converged:
   a) Associate each point with its nearest centroid
   b) Update each centroid as the average of associated points



K-Mean Clustering

Algorithm:

1. Initialize *k* centroids randomly
2. While not converged:
    a) Associate each point with its nearest centroid
    b) Update each centroid as the average of associated points
3. Return centroids and associations



Animated Example

http://mlehman.github.io/kmeans-javascript/

Formalizing K-Mean Clustering

Given a dataset $X = \{\boldsymbol{x}_i\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and the number of desired clusters k, **produce** a set of assignments $Z = \{z_i\}_{i=1}^n$ where $z_i \in \{1,2,\ldots,k\}$ and a set of centroids $C = \{\boldsymbol{c}_j\}_{j=1}^k$ where $\boldsymbol{c}_j \in \mathbb{R}^d$.

Do this by alternating the following steps:

a) Assignment: For each datapoint *i*, update z_i:

$$z_i = \underset{j=1,2,\ldots,k}{\operatorname{argmin}} \left\| \boldsymbol{x}_i - \boldsymbol{c}_j \right\|_2^2$$

b) Update: For each centroid j, update $\boldsymbol{c}_j$:

$$\boldsymbol{c}_j = \frac{\mathbf{1}}{\sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z}_i = j]} \sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z}_i = j] \ \boldsymbol{x}_i$$

Question Break!

Assuming the dataset consists of ***n*** ***d***-dimensional features, what is the computational complexity of running ***k***-means for ***m*** iterations?

Hint: You can assume computing distance between two points is O(d)

Do this by alternating the following steps:

a) Assignment: For each datapoint *i*, update z_i:

$$z_i = \underset{j=1,2,\ldots,k}{\operatorname{argmin}} \|\boldsymbol{x}_i - \boldsymbol{c}_j\|_2^2$$

b) Update: For each centroid j, update $\boldsymbol{c}_j$:

$$\boldsymbol{c}_j = \frac{\mathbf{1}}{\sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z}_i = j]} \sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z}_i = j] \ \boldsymbol{x}_i$$

A $O(kd)$

B $O(mkn)$

C $O(mknd)$

D $O(mk^2n)$

Complexity of k-Means

Computing L2 distance between two points is O(d).

Need distance between each datapoint and each centroid. **O(knd)**

Do this by alternating the following steps:

a) Assignment: For each datapoint *i*, update z_i:

$$z_i = \underset{j=1,2,\ldots,k}{\operatorname{argmin}} \|\boldsymbol{x}_i - \boldsymbol{c}_j\|_2^2$$

b) Update: For each centroid j, update $\boldsymbol{c_j}$:

$$\boldsymbol{c_j} = \frac{\mathbf{1}}{\sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z_i} = j]} \sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z_i} = j] \ \boldsymbol{x_i}$$

Each datapoint contributes to only one cluster, total of n vector additions. **O(nd)**

Repeat this for ***m*** iterations.

Overall computational complexity is therefor O(mknd)

- linear in relevant inputs (for fixed iteration count)

Formalizing K-Mean Clustering

Given a dataset $X = \{\boldsymbol{x}_i\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and the number of desired clusters k, **produce** a set of assignments $Z = \{z_i\}_{i=1}^n$ where $z_i \in \{1,2,\dots,k\}$ and a set of centroids $C = \{\boldsymbol{c}_j\}_{j=1}^k$ where $\boldsymbol{c}_j \in \mathbb{R}^d$.

Do this by alternating the following steps:

a) Assignment: For each datapoint *i*, update z_i:

$$z_i = \underset{j=1,2,\dots,k}{\operatorname{argmin}} \left\| \boldsymbol{x}_i - \boldsymbol{c}_j \right\|_2^2$$

b) Update: For each centroid j, update $\boldsymbol{c}_j$:

$$\boldsymbol{c}_j = \frac{1}{\sum_{i=1}^n \mathbb{I}[z_i = j]} \sum_{i=1}^n \mathbb{I}[z_i = j]\ \boldsymbol{x}_i$$

This is a *procedural* view of k-Means –
i.e., it describes the "how" but not the "why".

What is k-Means optimizing?

Claim: k-Means is minimizing the sum of squared error between points and their associated cluster centroid. That is, the optimal centroids and assignments are:

$$C^*, Z^* = \underset{C,Z}{\text{argmin}} \; SSE(X, C, Z) = \underset{C,Z}{\text{argmin}} \sum_{i=1}^{n} \left\| \boldsymbol{x}_i - \boldsymbol{c}_{z_i} \right\|_2^2$$

This is an optimization over two sets of variable – assignments and centroids – and mixes discrete / continuous variables. Very hard in general.

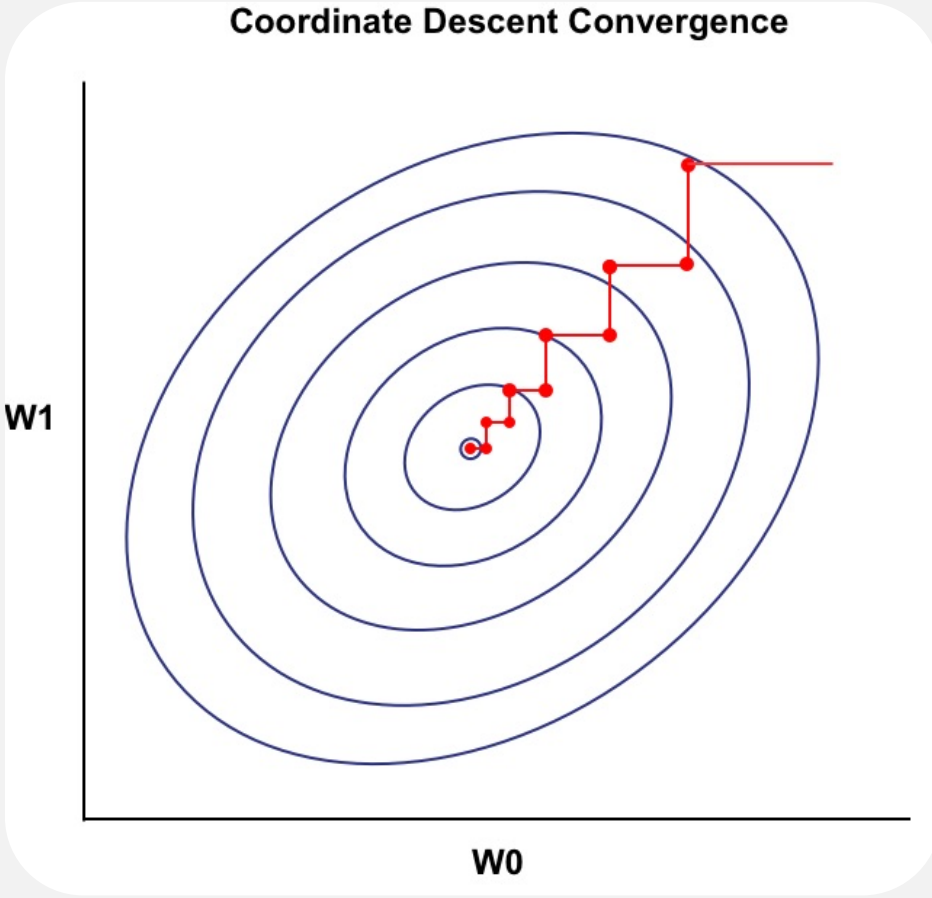
How do we get from this problem to the 2-step process we defined before?

Coordinate Descent

Suppose we have some function $f(w_1, w_2)$ we want to minimize.

Coordinate Descent would alternative between the following steps:

1. Fix w_2 as a constant and optimize w_1
2. Fix w_1 as a constant and optimize w_2



Why do this? Sometimes functions have closed-form / easy solutions in some variables if the others are fixed.

Does it converge to an optima? Yes for convex, differentiable f. Not generally.

What is k-Means optimizing?

Coordinate Descent on SSE. Will alternate between solving for C and Z that minimize SSE with the other fixed.

$$SSE(X, C, Z) = \sum_{i=1}^{n} \| \boldsymbol{x}_i - \boldsymbol{c}_{z_i} \|_2^2 = \sum_{i=1}^{n} \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t} \right)^T \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t} \right)$$

Minimize objective with assignments Z^t fixed from previous round:

$$C^{t+1} = \underset{C}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t} \right)^T \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t} \right)$$

Minimize objective with centroids C^t fixed from previous round:

$$Z^{t+1} = \underset{Z}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i}^{\boldsymbol{t+1}} \right)^T \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i}^{\boldsymbol{t+1}} \right)$$

What is k-Means optimizing?

Minimize objective with assignments Z^t fixed from previous round:

$$C^{t+1} = \underset{C}{\text{argmin}} \sum_{i=1}^{n} \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t}\right)^T \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i^t}\right)$$

Observe c_j only affects the overall SSE for points assigned to it, so could optimize each independently. Let's consider SSE over the set $\mathrm{M_j} = \{x_i \,|z_i^t = j\}$

$$c_j^{t+1} = \underset{\boldsymbol{c}}{\text{argmin}} \sum_{x_i \in M_j} (\boldsymbol{x}_i - \boldsymbol{c})^T (\boldsymbol{x}_i - \boldsymbol{c})$$

Let's take a derivative and set it equal to 0.

$$\frac{dSSE}{dc} = \sum_{x_i \in M_j} 2(\boldsymbol{x}_i - \boldsymbol{c}) \quad \Rightarrow \quad \boldsymbol{c} = \frac{1}{|M_j|} \sum_{x_i \in M_j} \boldsymbol{x}_i$$

What is k-Means optimizing?

Minimize objective with centroids C^t fixed from previous round:

$$Z^{t+1} = \underset{Z}{\text{argmin}} \sum_{i=1}^{n} \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i}^{\boldsymbol{t+1}}\right)^T \left(\boldsymbol{x}_i - \boldsymbol{c}_{z_i}^{\boldsymbol{t+1}}\right)$$

Each z_i only affects the distance of x_i to its assigned centroid. Can consider each independently:

$$z_i = \underset{z}{\text{argmin}} \ (\boldsymbol{x}_i - \boldsymbol{c}_{\boldsymbol{z}})^T (\boldsymbol{x}_i - \boldsymbol{c}_{\boldsymbol{z}})$$

As the number of centroids is finite, we can solve this by simply computing the distance to each centroid and taking the id of the nearest centroid.

Formalizing K-Mean Clustering

Given a dataset $X = \{\boldsymbol{x}_i\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and the number of desired clusters k, **produce** a set of assignments $Z = \{z_i\}_{i=1}^n$ where $z_i \in \{1,2,\dots,k\}$ and a set of centroids $C = \{c_j\}_{j=1}^k$ where $c_j \in \mathbb{R}^d$.

Do this by alternating the following steps:

a) Assignment: For each datapoint *i*, update z_i:

$$z_i = \underset{j=1,2,\dots,k}{\operatorname{argmin}} \left\| \boldsymbol{x}_i - \boldsymbol{c}_j \right\|_2^2$$

b) Update: For each centroid j, update $\boldsymbol{c_j}$:

$$\boldsymbol{c_j} = \frac{\mathbf{1}}{\sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z_i} = j]} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \mathbb{I}[\boldsymbol{z_i} = j] \;\; \boldsymbol{x_i}$$

Now we know this is coordinate descent to minimize sum-of-squared error between centroids and datapoints!

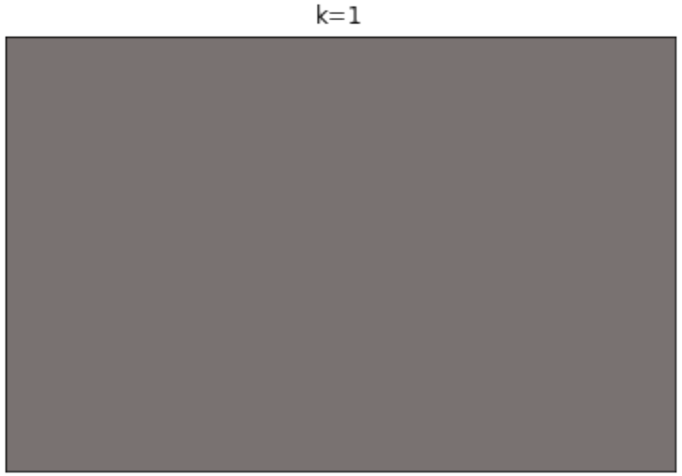
(There is a layer deeper than this we'll get to later.)

Some Examples of k-Means in Action

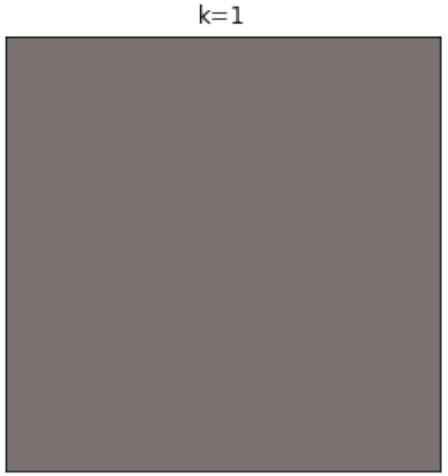
Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction



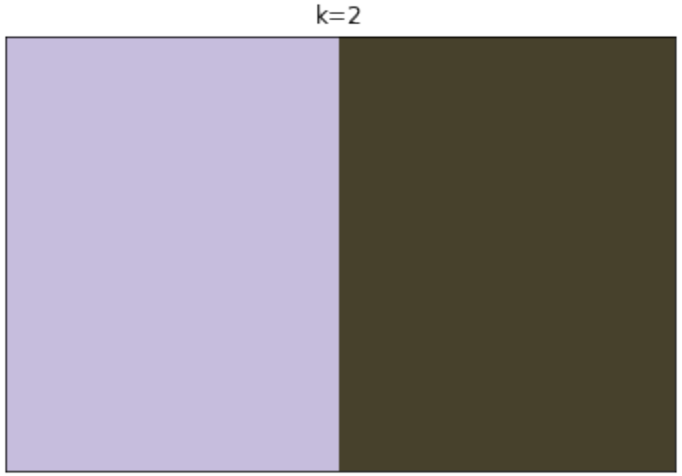
Only store id of centroid at each pixel. Store the centroids as well.

Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction

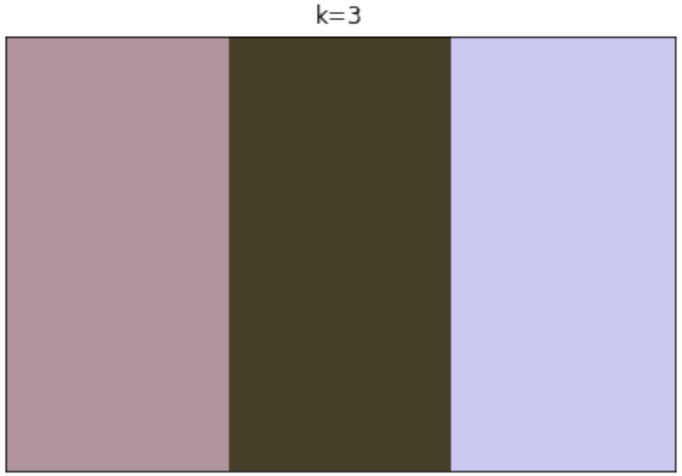
Only store id of centroid at each pixel. Store the centroids as well.

Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction

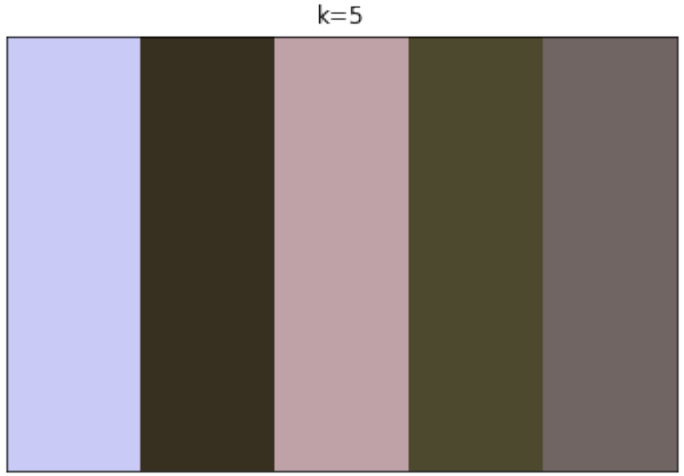
Only store id of centroid at each pixel. Store the centroids as well.

Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction

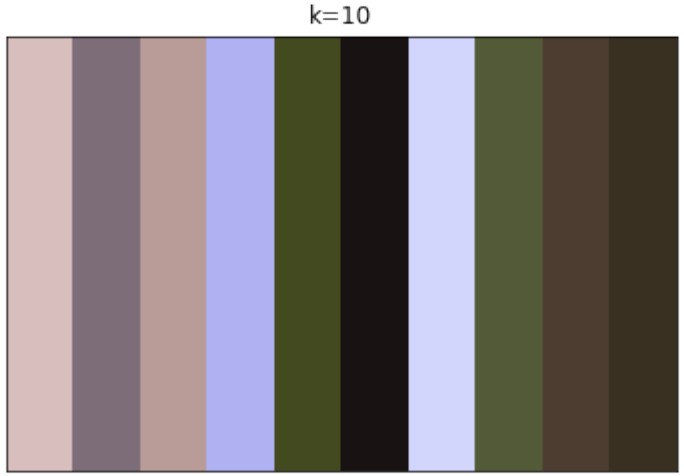
Only store id of centroid at each pixel. Store the centroids as well.

Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction

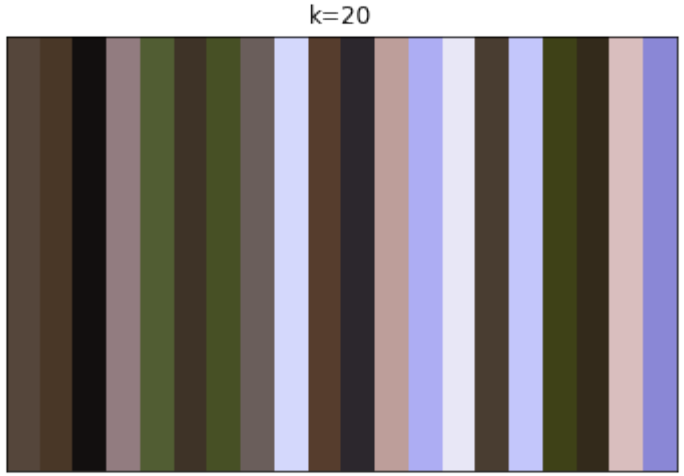
Only store id of centroid at each pixel. Store the centroids as well.

Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction

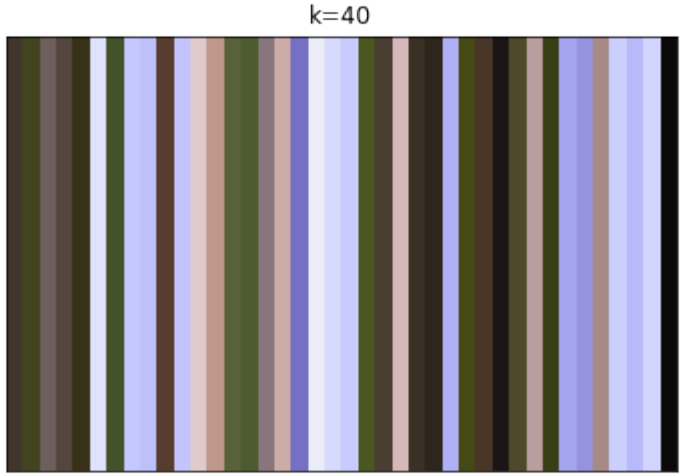
Only store id of centroid at each pixel. Store the centroids as well.

Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction

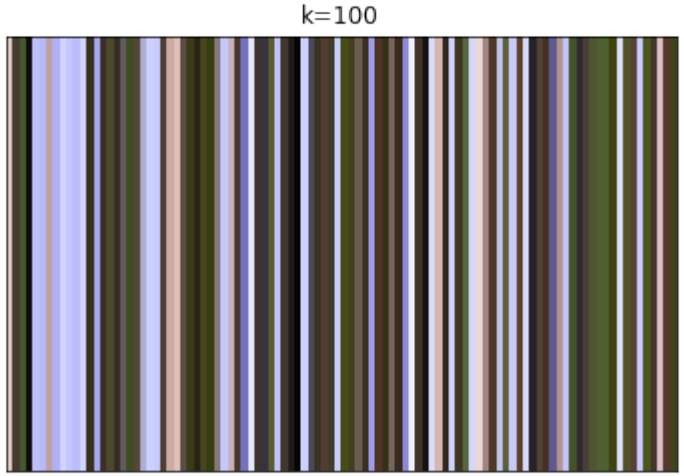
Only store id of centroid at each pixel. Store the centroids as well.

Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction

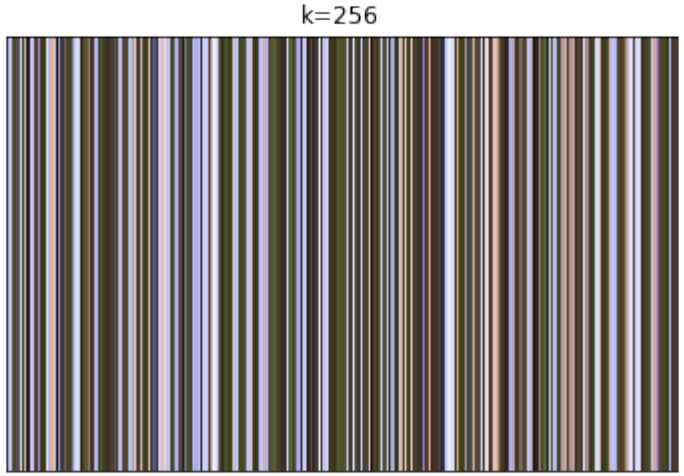
Only store id of centroid at each pixel. Store the centroids as well.

Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors



Reconstruction

Only store id of centroid at each pixel. Store the centroids as well.

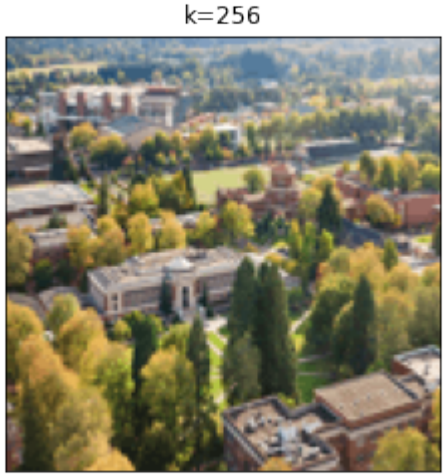
Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids – significant size reduction!

Original

Centroid Colors

Reconstruction



Only store id of centroid at each pixel. Store the centroids as well.

Some Examples of k-Means in Action

Discovering Similar Datapoints: Took a dataset of faces and represented them with an edge-sensitive image feature called HoG (see your homework for more details). Clustered them!

One of the Clusters

Random Sample

Some Examples of k-Means in Action

Discovering Similar Datapoints: Took a dataset of faces and represented them with an edge-sensitive image feature called HoG (see your homework for more details). Clustered them!

One of the Clusters

Random Sample

Some Examples of k-Means in Action

Discovering Similar Datapoints: Took a dataset of faces and represented them with an edge-sensitive image feature called HoG (see your homework for more details). Clustered them!

One of the Clusters

Random Sample

Question Break!

Intuition Check: Following this coordinate descent algorithm we described; k-Means is _____.

A Guaranteed to converge in finite steps to the global minima of SSE

B Guaranteed to converge in finite steps to a local minima of SSE

C Not guaranteed to converge in finite steps

D I don't know how to think about this.

Monotonicity of k-Means Optimization

Monotonicity: Each iteration of k-Means strictly decreases the SSE until convergence. Proof relies on showing that both the assignment and update steps cannot increase SSE.

a) Assignment: Each point only switches to a new cluster if the squared error is *lower*:

$$z_i = \underset{j=1,2,\ldots,k}{\operatorname{argmin}} \left\| \boldsymbol{x}_i - \boldsymbol{c}_j \right\|_2^2$$

b) Update: Each centroid is updated to the mean of its assignments, which we already showed minimizes the sum-of-squared error with respect to a fixed assignment.

$$\boldsymbol{c_j} = \frac{\mathbf{1}}{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \mathbb{I}[\boldsymbol{z_i} = j]} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \mathbb{I}[\boldsymbol{z_i} = j] \ \boldsymbol{x_i}$$

Question Break!

If I increase the number of clusters (k), the sum-of-squared error will _____.

A Increase

B Decrease

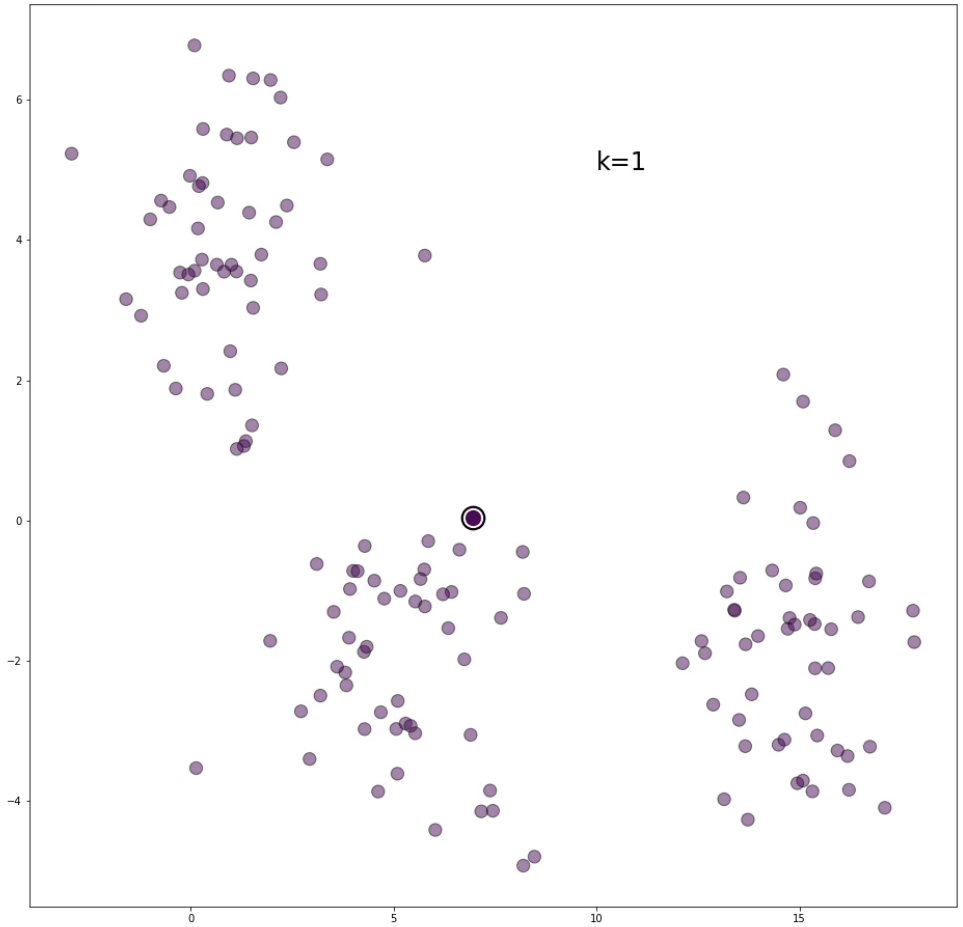
C Increase or decrease depending on the dataset

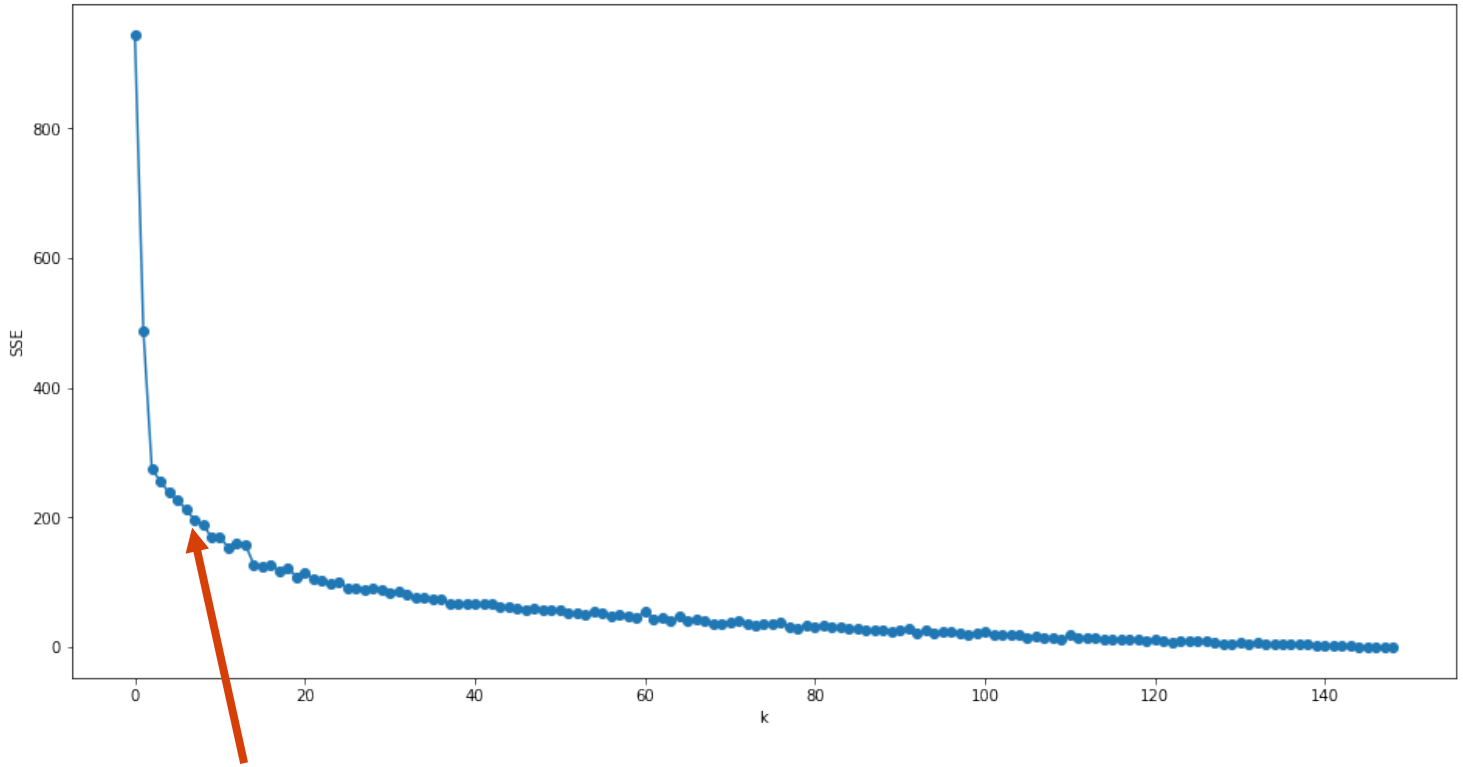
D Stay the same

Properties: How do we choose **k**?

The number of clusters **k** is an important hyperpameter. How can we choose it?

- Unfortunately, SSE decreases with k so training set won't tell us much.





One common heuristic is to pick the "elbow/knee" of the SSE vs. k graph.

Question Break!

If I run this algorithm multiple times, should I generally expect the same or different clusterings to be produced?

A Same

B Different

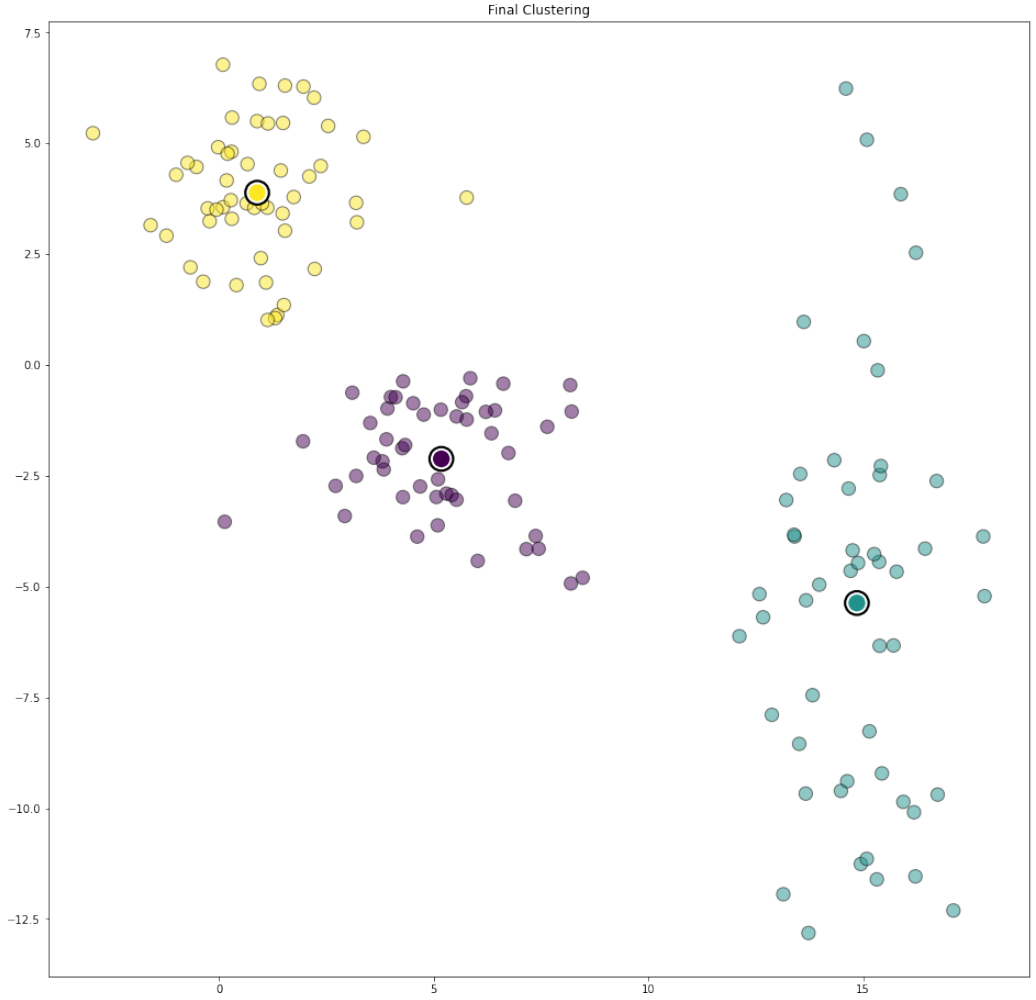
C

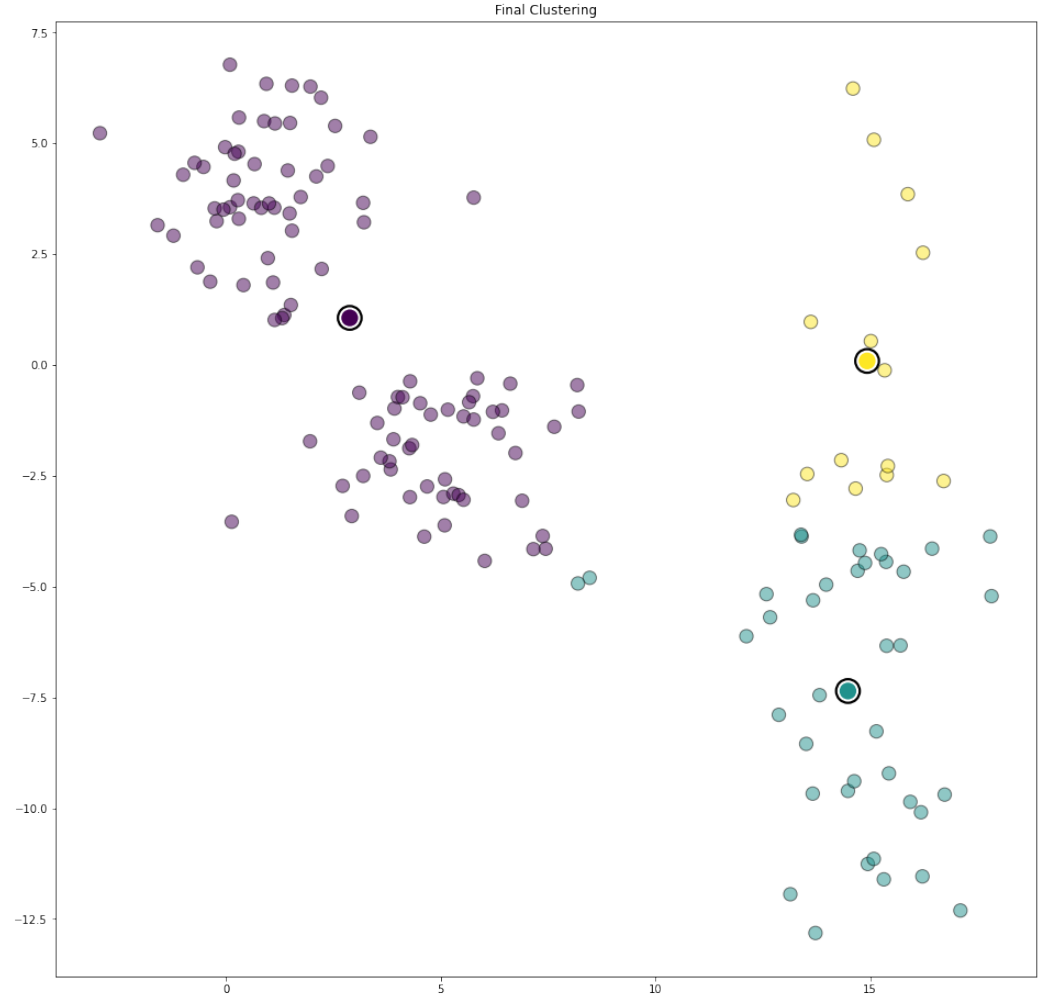
D

Properties: Highly Sensitive to Initialization

k-Means is highly sensitive to initialization of the centroids.

- Sum-of-squared error has many local minima where no local reassignments can reduce SSE.





Properties: Highly Sensitive to Initialization

How can we deal with this sensitivity?

Run multiple trials and choose the one with the lowest SSE (often used in practice)

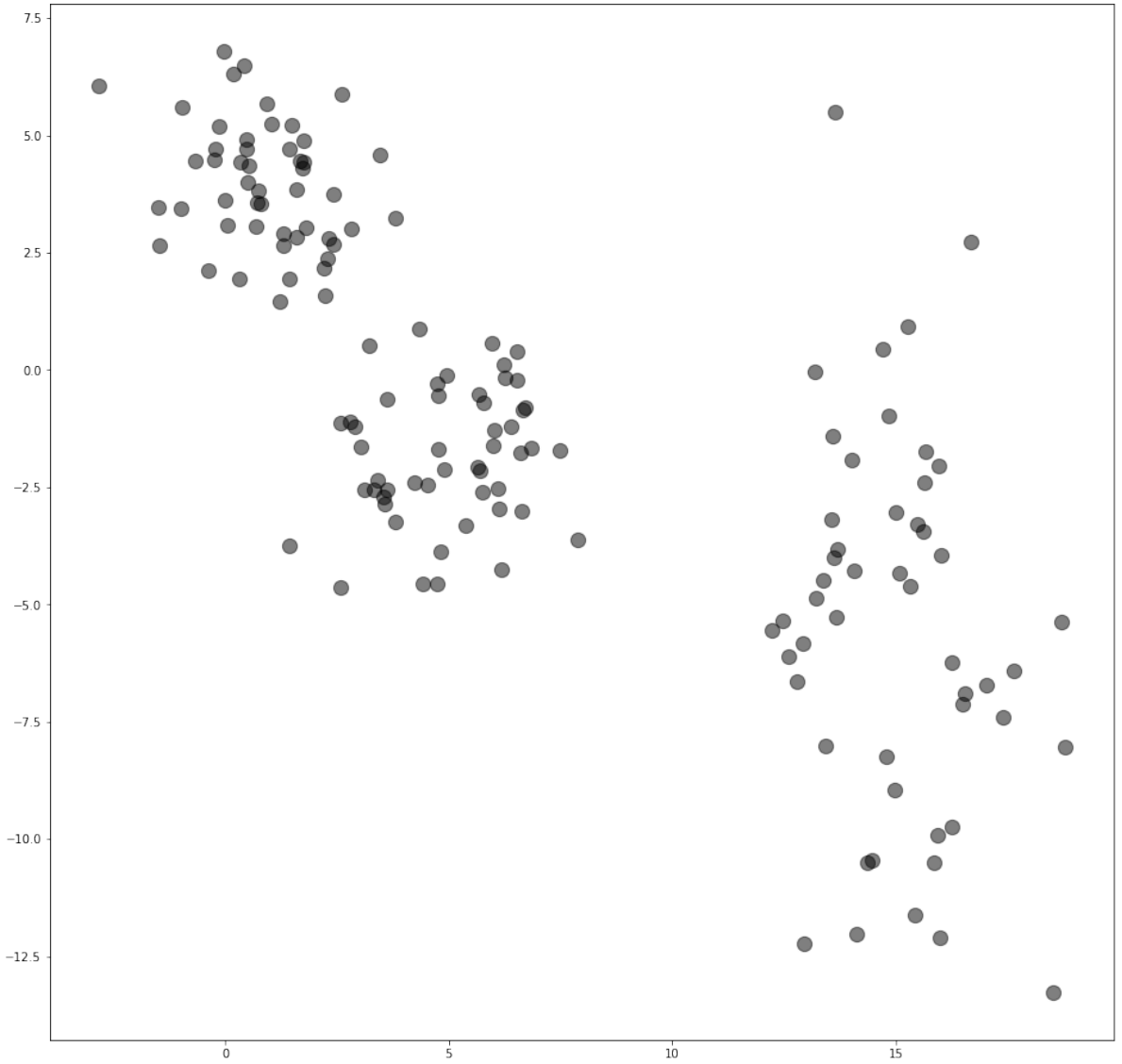
Try to initialize the centroids "well" – not clear how to do this. Some common heuristics:

- **Random Points:** Initialize the centroids by copying random datapoints – ensures your centroids are near data.
- **Far-Away Points:** Try to make the cluster centers far from each other:
  - Samples a datapoint and set the first centroid c_1 to its value
  - Compute a weight w_i for each datapoint proportional to its distance from c_1. Then sample according to this distribution.
  - Repeat with weights proportional to the nearest centroid.

Properties: Highly Sensitive to Initialization

Example of Iterative Distance-Based Sampling Initialization: $w_i \propto e^{\left(\min_{c \in C}(x_i - c)^2\right)}$

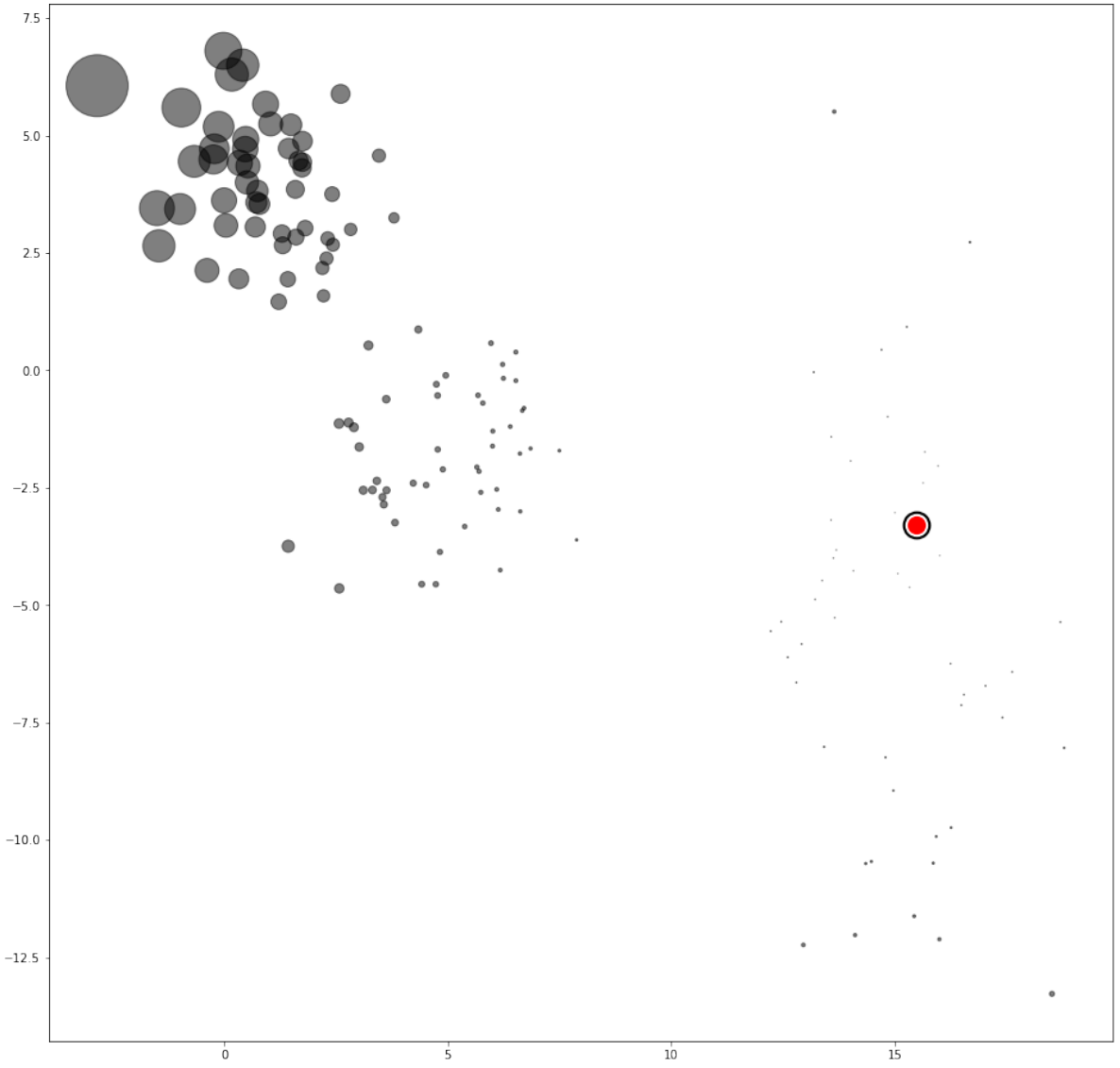
Point size proportional to probability of being selected this iteration.



Properties: Highly Sensitive to Initialization

Example of Iterative Distance-Based Sampling Initialization: $w_i \propto e^{\left(\min_{c \in C}(x_i - c)^2\right)}$

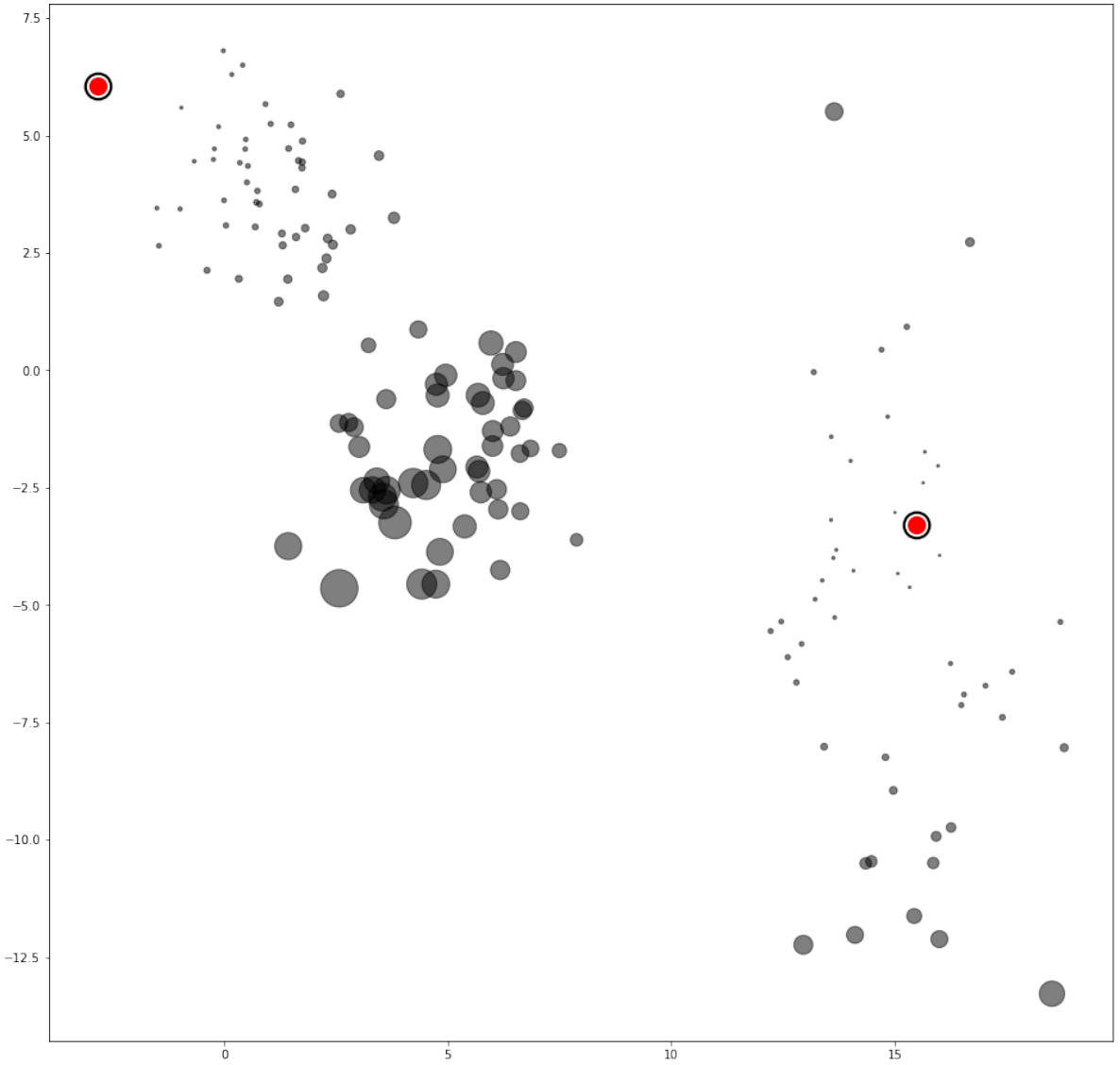
Point size proportional to probability of being selected this iteration.



Properties: Highly Sensitive to Initialization

Example of Iterative Distance-Based Sampling Initialization: $w_i \propto e^{\left(\min_{c \in C}(x_i - c)^2\right)}$

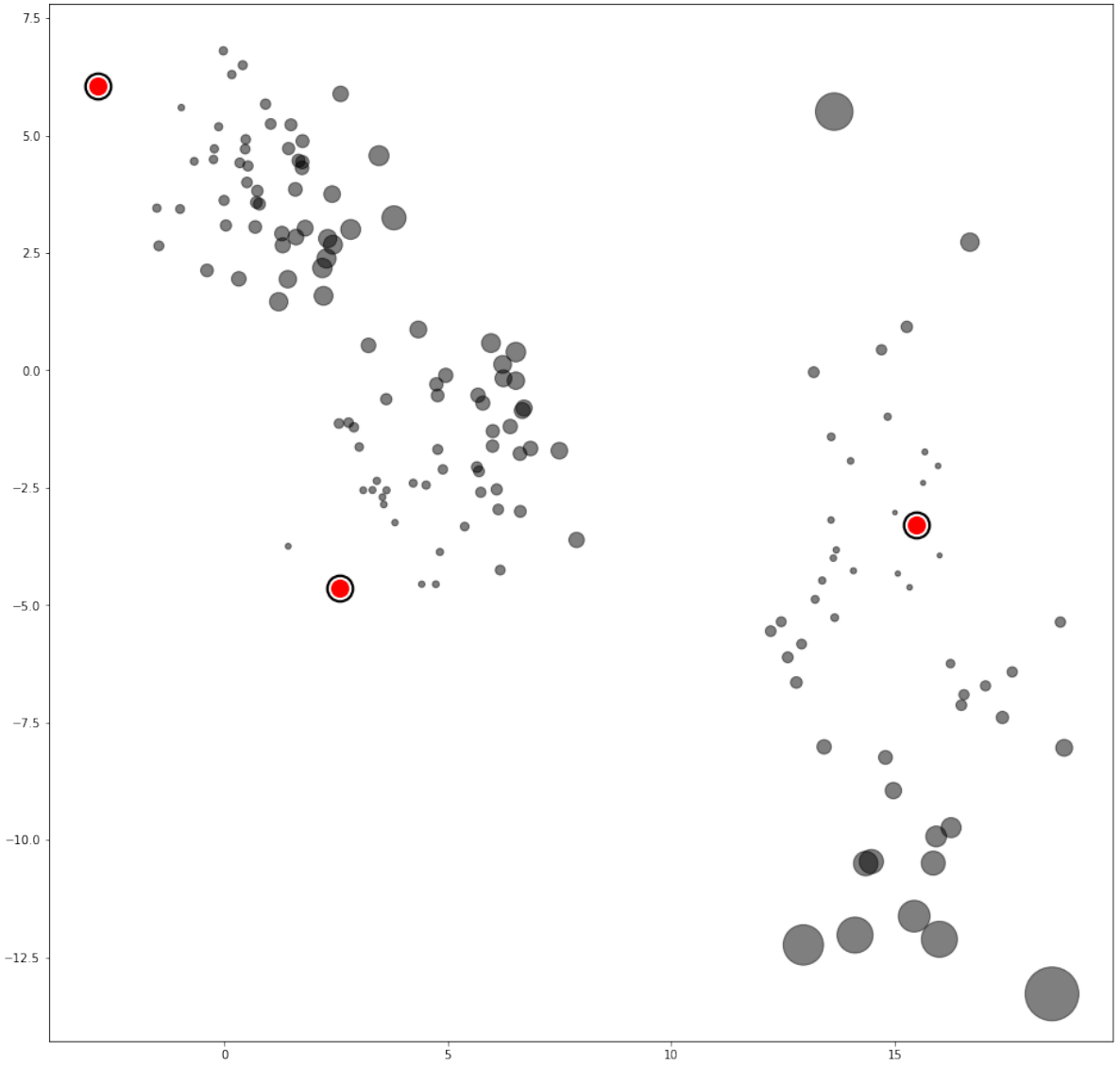
Point size proportional to probability of being selected this iteration.



Properties: Highly Sensitive to Initialization

Example of Iterative Distance-Based Sampling Initialization: $w_i \propto e^{\left(\min_{c \in C}(x_i - c)^2\right)}$

Point size proportional to probability of being selected this iteration.



Question Break!

Do you expect the k-means algorithm
with L2 distance to be robust to outliers?

(By robust to outliers, I mean the solution won't be changed too much by adding any individual point)

A Not Robust

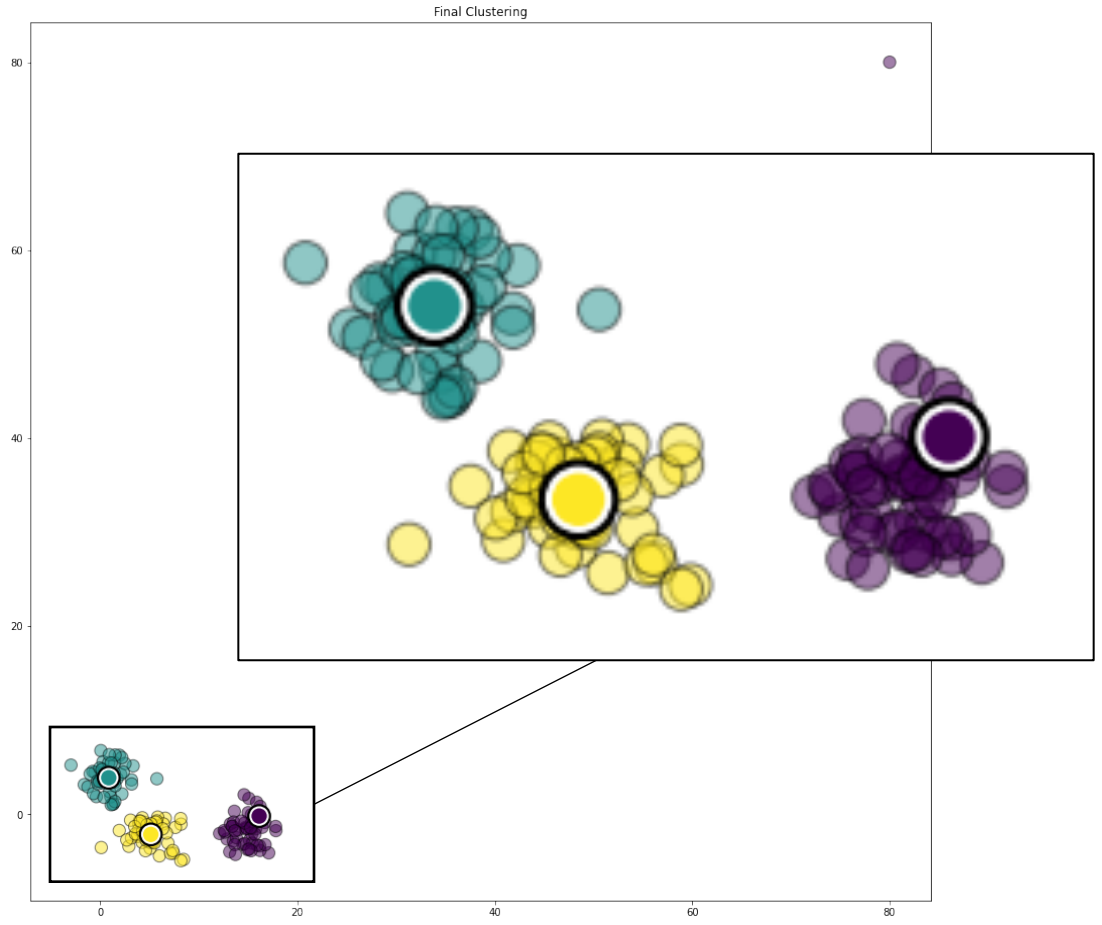
B Robust

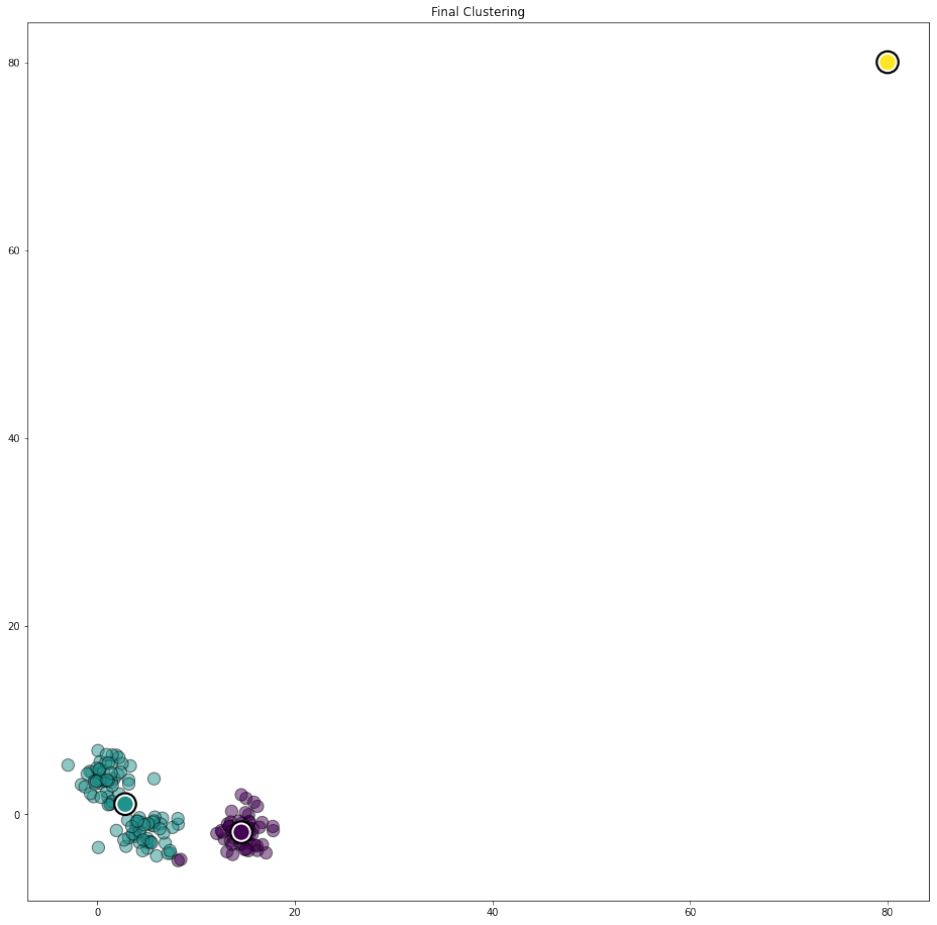
C

D

Properties: Sensitive to Outliers

Every point needs to have a cluster and every cluster center is the average of its members.





Properties: Sensitive to Outliers

Rough Idea of k-medoids Algorithm: Instead of taking the mean in the centroid update step, take the median – i.e., the data point that is closest to the other points in the cluster.

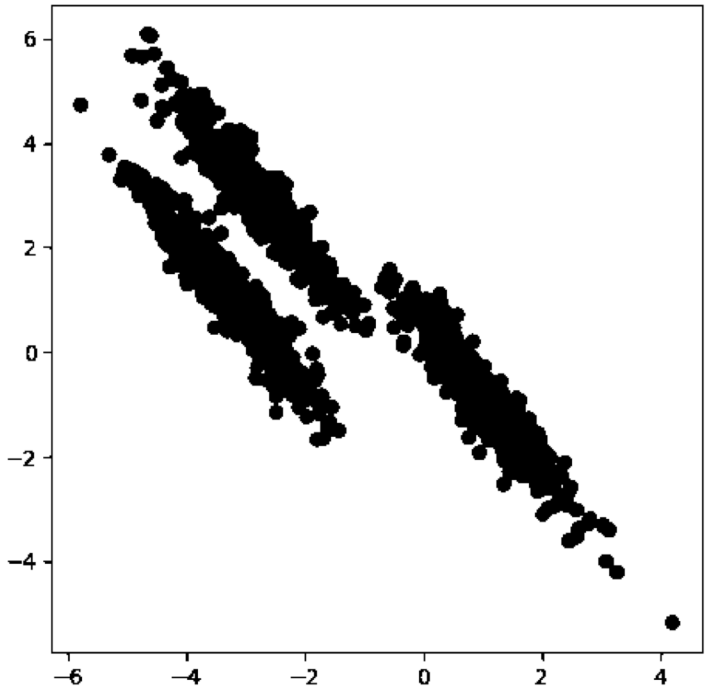
$$c_j = \frac{1}{\sum_{i=1}^{n} \mathbb{I}[z_i = j]} \sum_{i=1}^{n} \mathbb{I}[z_i = j]\ x_i \quad \longrightarrow \quad c_j = \underset{z \in M_j}{\operatorname{argmin}} \sum_{x_i \in M_j} \|x_i - z\|_2^2$$

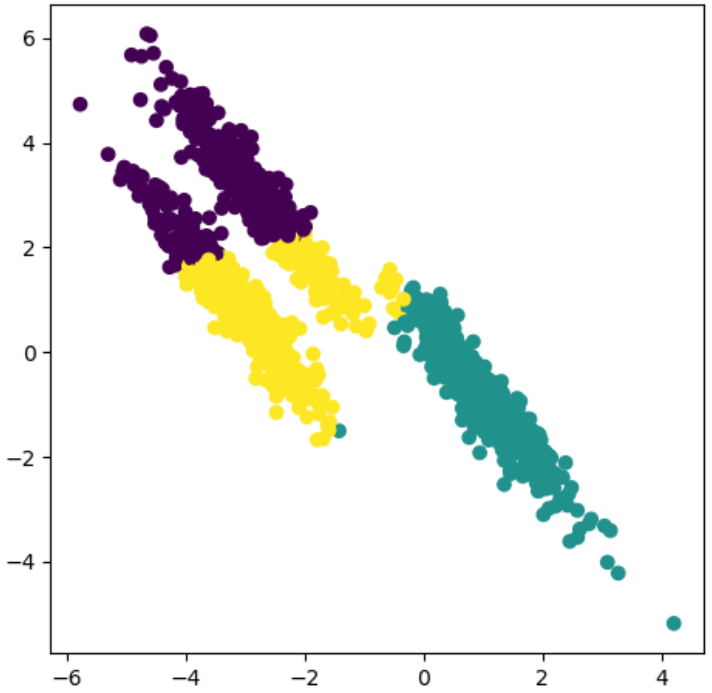
K-Medoids is computationally more expensive but more robust to outliers.

Slide adapted from Xiaoli Fern

Questions Break!

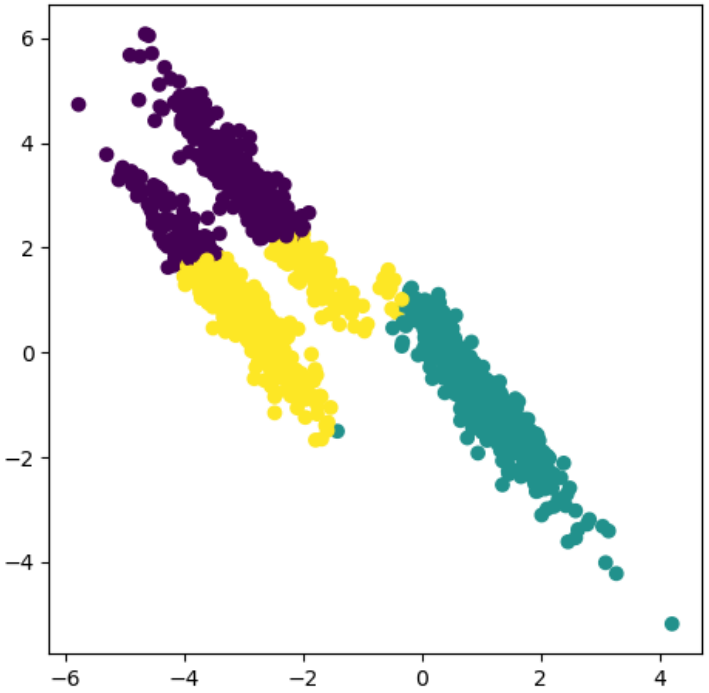
Consider the dataset below. How many clusters do you see? If I ran k-Means with that value for k, what do you expect the resulting clustering to be?

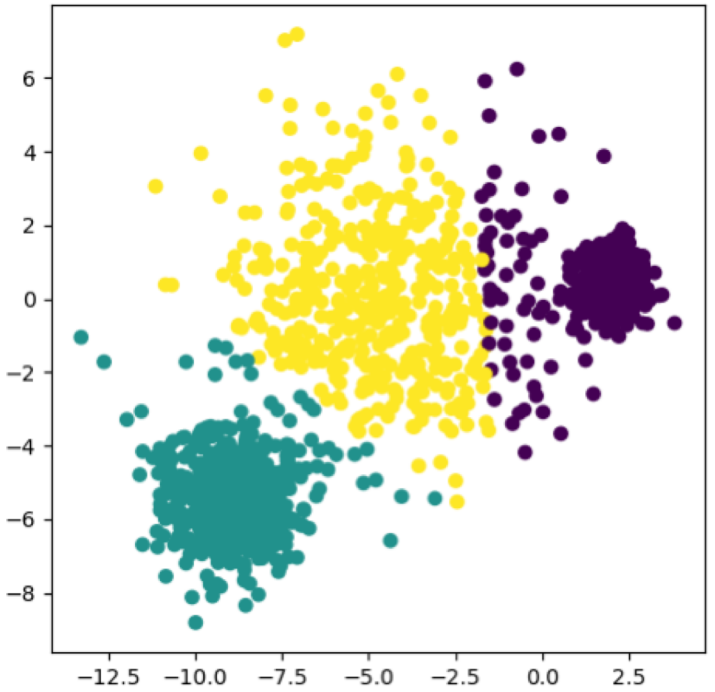




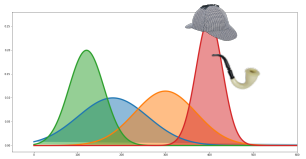
Properties: Some Hidden Assumptions

k-Means performs poorly when the clusters are not spherical or when different clusters have very different spreads. Seems like there are some hidden assumptions here…





Where do these assumptions come from? Spoiler: Its Gaussian again!



Figures adapted from Phil Roth

Summary of k-Means

Given a dataset, **k-Means splits it into k disjoint groups.**

- Setting k is largely heuristic as larger k produces lower SSE
  - Unsupervised learning has no validation set because it has no labels

- Guaranteed to converge in finite steps to a local minima
  - Often in a few iterations in practice

- O(mknd) computational complexity makes running k-Means tractable

Properties of k-means:

- Not particularly resistant to outliers
- Very sensitive to initialization of centroids – need to run multiple times
- Only seems to work on spherical clusters with similar sizes

Today's Learning Objectives

Be able to answer:

- ~~What is unsupervised learning and how does it differ form supervised learning?~~
  - ~~What are some problems in unsupervised learning?~~

- ~~What is clustering?~~

- ~~How does k-means clustering work?~~
- ~~What is k-means optimizing?~~
  - ~~What is a coordinate descent algorithm?~~

- What is a Gaussian Mixture Model?
- What can this tell us about k-Means?

- How do we evaluate clustering?

Gaussian Mixture Models (GMM)

Given: a dataset $D = \{\boldsymbol{x}_i\}_{i=1}^{n}$ where $x \in \mathbb{R}^d$

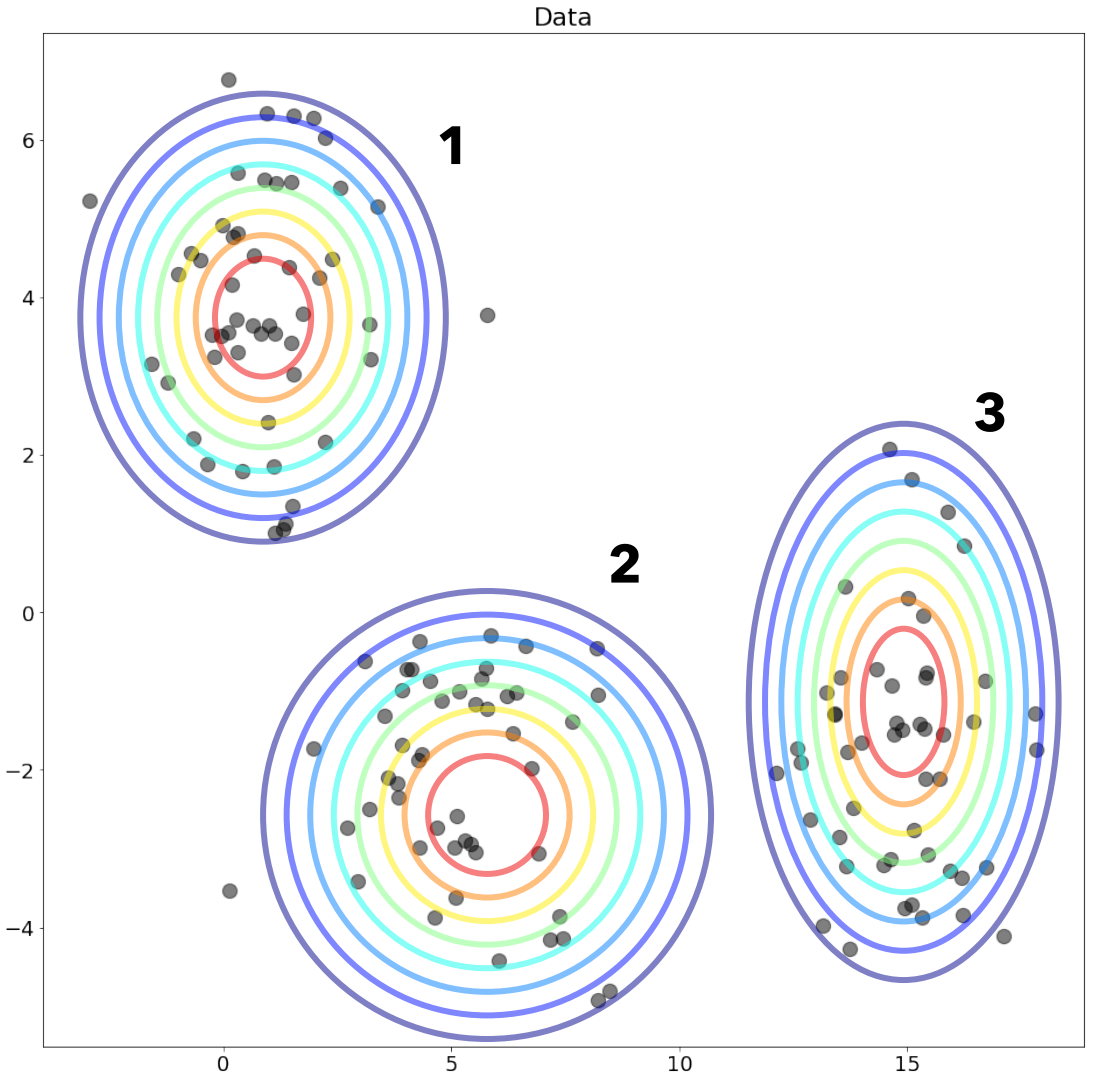
Goal: Fit the distribution $P(\boldsymbol{x})$

But this seems like a weird distribution… what if I approximate it as multiple Gaussians?

An Idea: Assume the following generative story of how this data was created.

Assume each point was generated by:

1. Sampling one of the three clusters, then
2. Sampling a point from a Gaussian defining that cluster



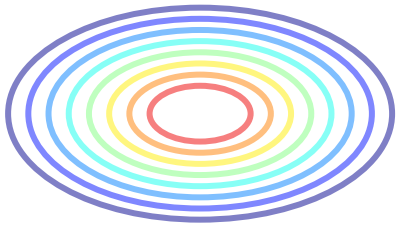
Recall Multidimensional Gaussians

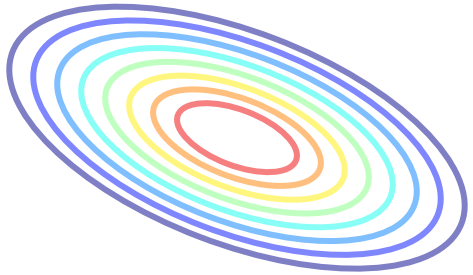
Gaussians can be defined in multiple dimensions. Describe the probability of a vector given the

- Mean vector μ – describing the location / center of the distribution
- Covariance matrix Σ – describing the spread of the distribution along different directions

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

Can draw them in 2D as a contour plot where lines show rings of equal probability and color shows relative probability:





Recall Multidimensional Gaussians

Different covariance matrix structures capture different shapes. Consider three structures:

Isotropic

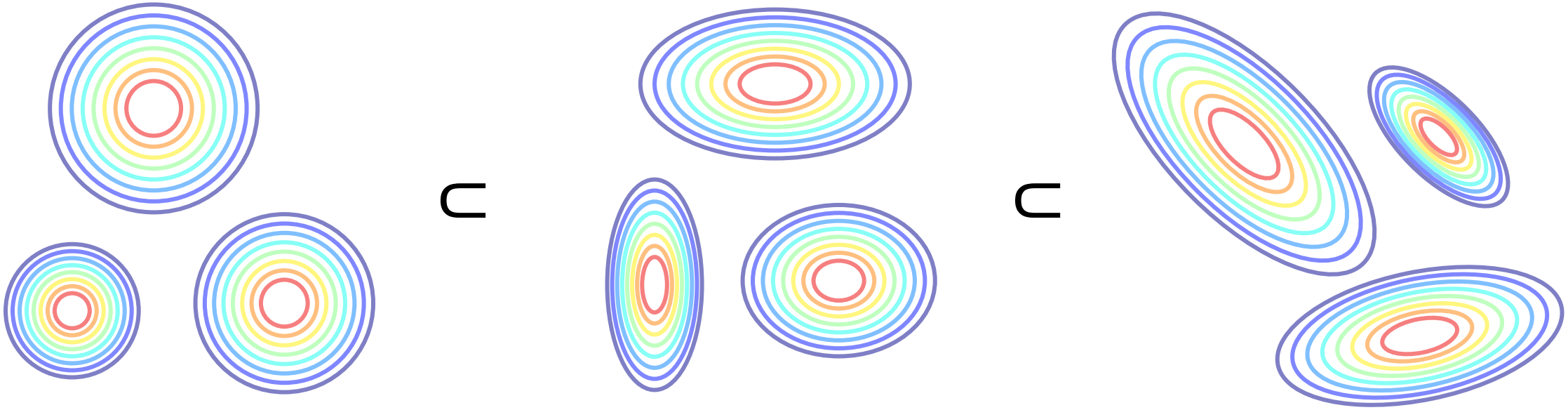
$$\Sigma = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$$

Axis-Aligned

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

Full Covariance

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{bmatrix}$$

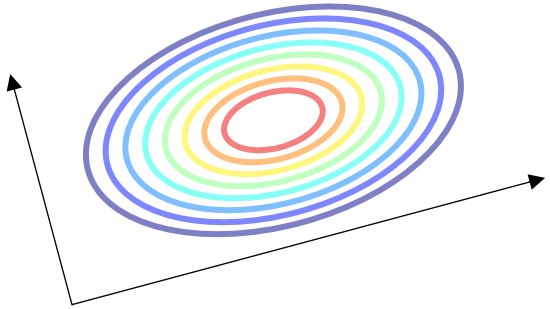


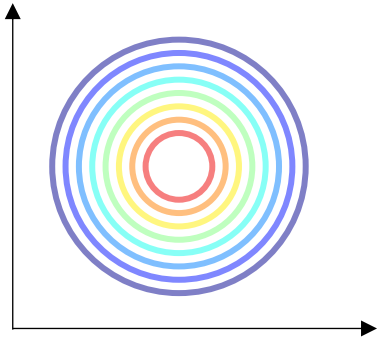
Slide idea adapted from Xiaoli Fern

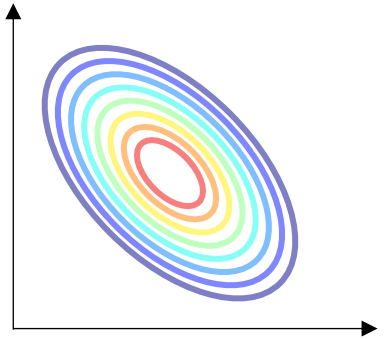
Question Break!

What would you describe these Gaussians as? (Be as specific as possible)

Isotropic ⊂ Axis-Aligned ⊂ Full Covariance







A Isotropic, Isotropic, Axis-Aligned

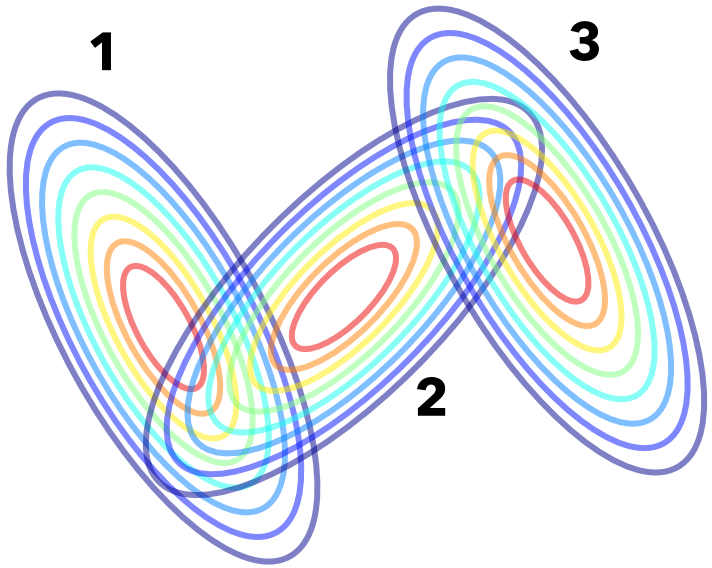
B Full Covariance, Isotropic, Full Covariance

C Axis-Aligned, Axis-Aligned, Isotropic

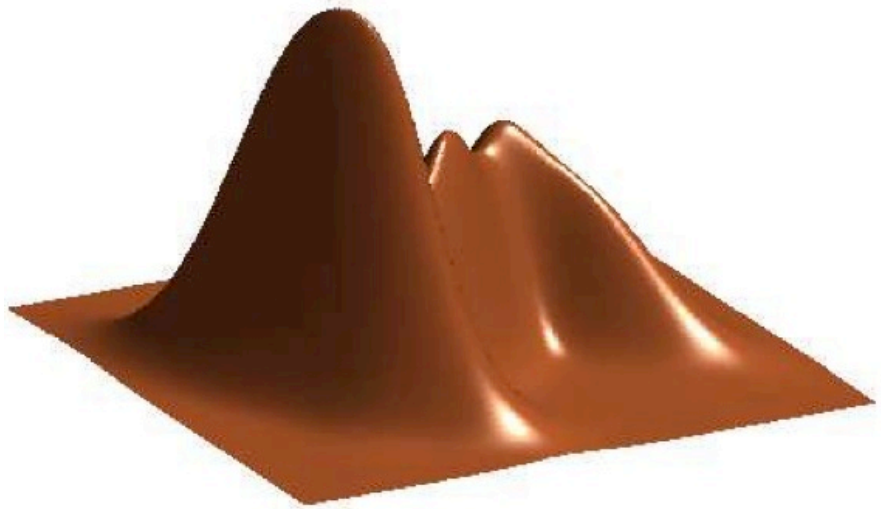
D Axis-Aligned, Isotropic, Full Covariance

An Example Gaussian Mixture Model

$$P(\boldsymbol{x}) = 0.5 * \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_1}, \Sigma_1) + 0.25 * \mathcal{N}(\mathbf{x}; \boldsymbol{\mu_2}, \Sigma_2) + 0.25 * \mathcal{N}(\mathbf{x}; \boldsymbol{\mu_3}, \Sigma_3)$$



$\boldsymbol{P(x)}$



$\boldsymbol{P(x)}$

Rightmost figure adapted from Kevin Murphy

Gaussian Mixture Models (GMM)

Given: a dataset $D = \{\boldsymbol{x}_i\}_{i=1}^{n}$ where $x \in \mathbb{R}^d$

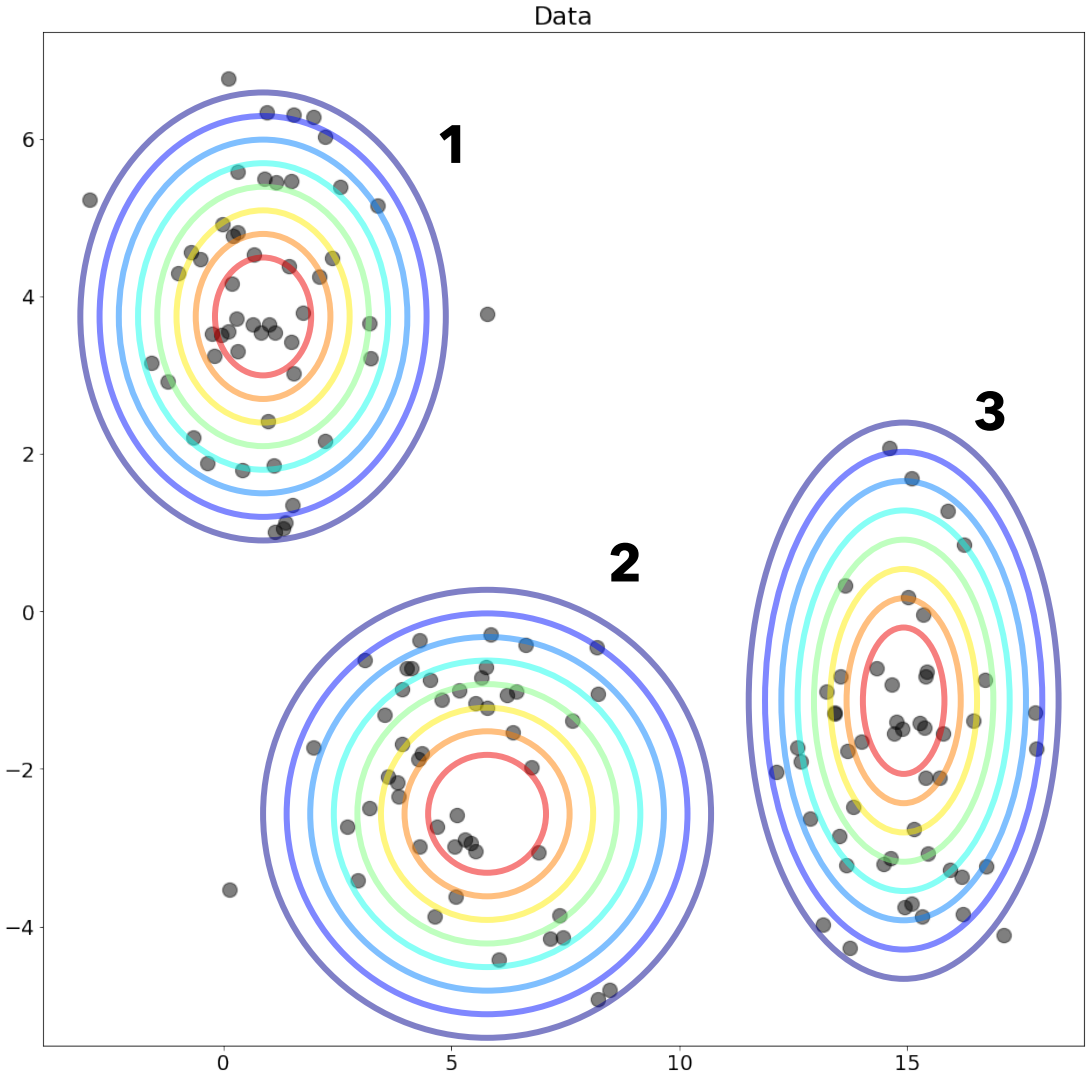
Goal: Fit the distribution $P(\boldsymbol{x})$ as a Gaussian Mixture Model with *k* components.

Formalizing our generative story: Assume each point x_i was generated by the process:

$$z_i \sim P(z)$$

$$x_i \sim \mathcal{N}(x; \mu_{z_i}, \Sigma_{z_i})$$

Call the z's "latent or hidden variables" as they are not observed from our data and must be inferred.

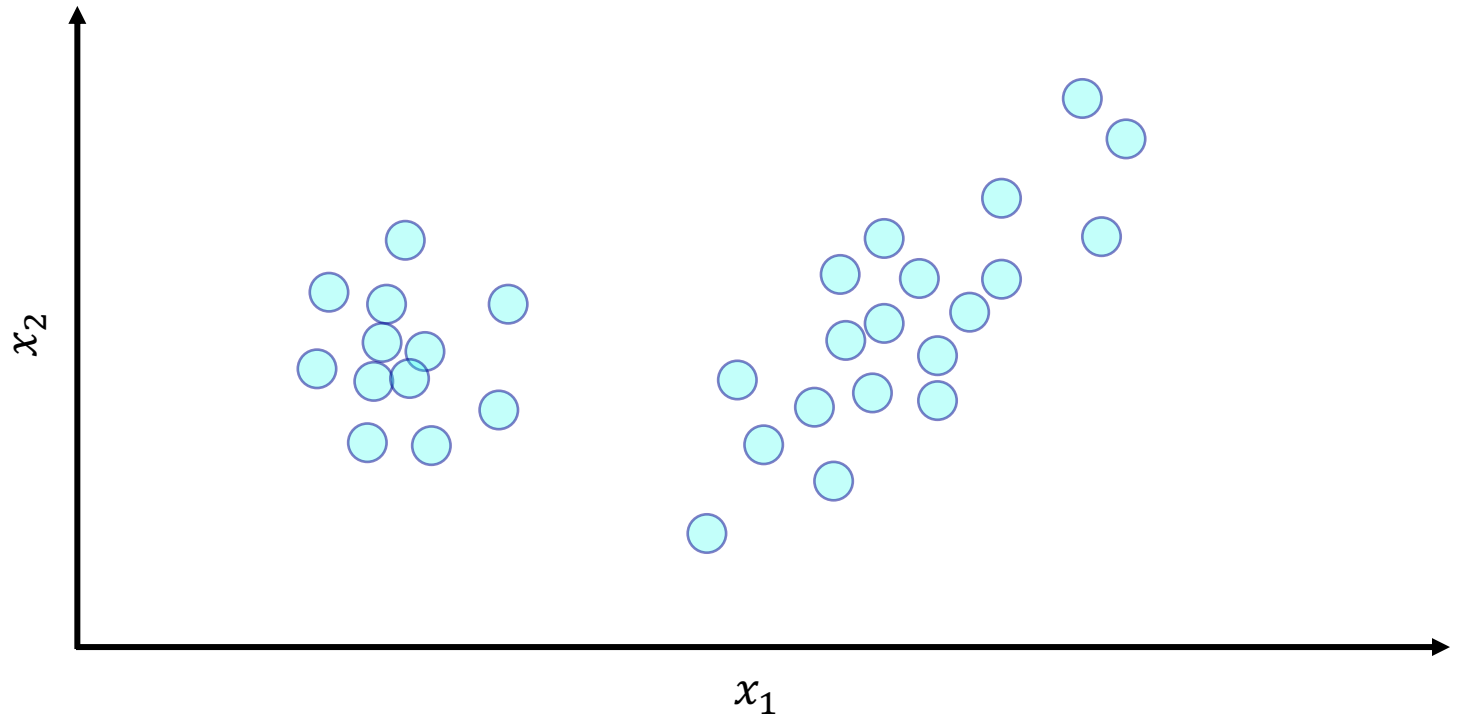


Need to learn:

- The means and covariances of the k Gaussians.
- The Categorical distribution P(z) over the k Gaussians.

Let's Start Simple – A 2d Case with k=2

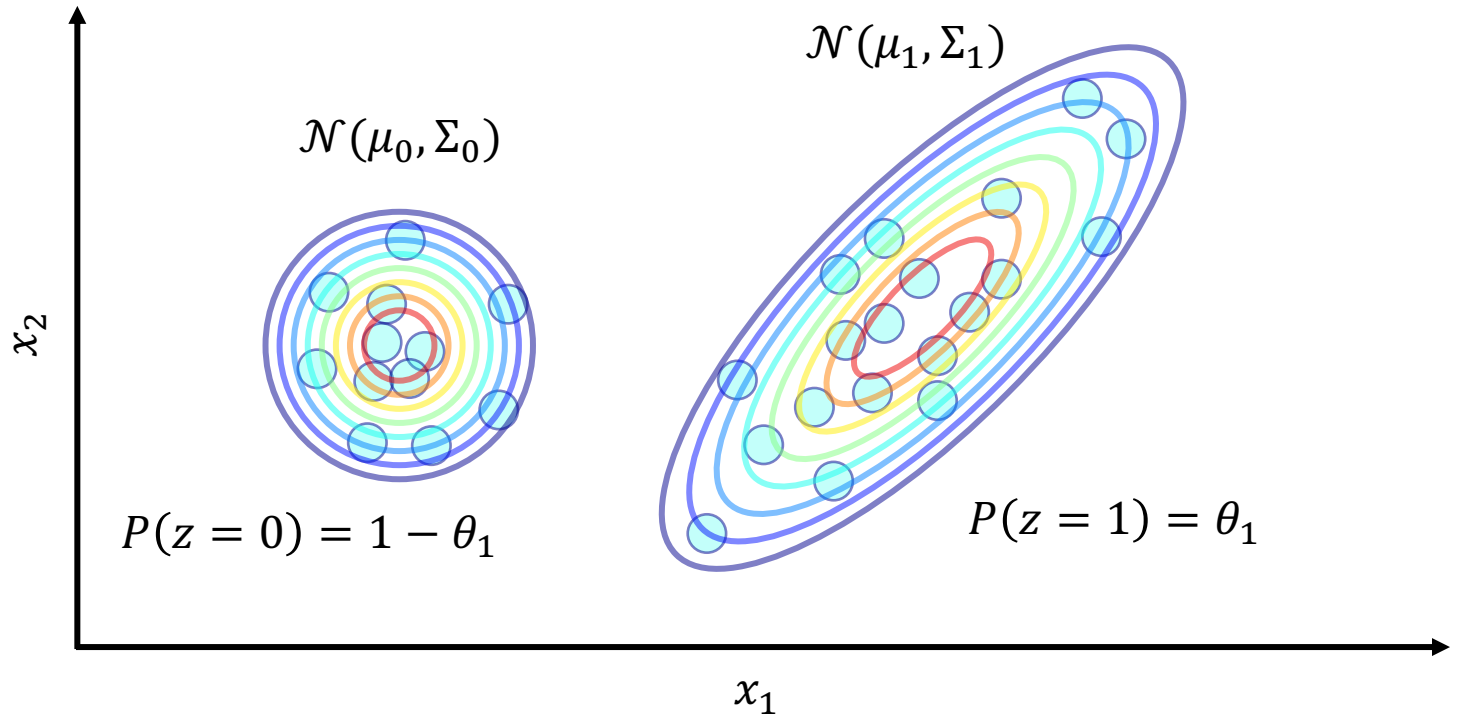
Suppose I observe the following 2-dimensional dataset. It appears to have two clusters, one on the left and one on the right.



Let's try modelling this as a mixture of two Gaussian distributions – that is, assume the data was generated by selecting one of two Gaussians and then sampling a point from the selected Gaussian.

Let's Start Simple – A 2d Case with k=2

Given a dataset of $D = \{x_i\}_{i=1}^n$ where $x \in \mathbb{R}$, want to learn $\mu_0, \mu_1, \Sigma_0, \Sigma_1$, and θ_1. $(\theta_0 = 1 - \theta_1)$



We don't know which Gaussian generated each point (especially when we haven't even learned the Gaussian's parameters yet). So we'll introduce the latent variables $\{z_i\}_{i=1}^n$

Questions Break!

Suppose a point $\boldsymbol{x_i}$ is distributed according to a Gaussian, i.e.,

$$\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

Write an expression for the probability $P(\boldsymbol{x_i})$.

Questions Break!

We have the following model for how a point $\boldsymbol{x}_i$ was generated:

$$z_i \sim Cat(\theta_0, \theta_1, \ldots, \theta_k) \qquad \boldsymbol{x}_i \sim \mathcal{N}\left(\boldsymbol{\mu}_{z_i}, \Sigma_{z_i}\right)$$

where z_i is an unobserved variable (i.e., we don't know z_i just like we don't know the parameters of our Gaussians or the prior Categorical)

Write an expression for the probability $P(\boldsymbol{x}_i)$.

Hint: needs marginalization.

Let's Start Simple – A 2d Case with k=2

What is the probability of a single datapoint under this model?

$$z_i \sim P(z)$$
$$\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{z_i}, \Sigma_{z_i})$$

$$P(x_i) = \sum_{c=0}^{1} P(\boldsymbol{x}_i, z_i = c) = \sum_{c=0}^{1} P(\boldsymbol{x}_i | \, z_i = c) \, P(z_i = c)$$

$$= P(z_i = 0)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_0, \Sigma_0) + P(z_i = 1)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_1, \Sigma_1)$$

Notice that we must sum over values of z_i to marginalize out the unknown latent variable. This will be a cause of great trouble for us later one.

Let's Start Simple – A 2d Case with k=2

Let's write out the log-likelihood and think about MLE to fit the parameters.

$$LL(\mu_1, \mu_2, \sigma_1, \sigma_2, \theta) = \log \prod_{i=1}^{n} P(\boldsymbol{x}_i) = \sum_{i=1}^{n} \log P(\boldsymbol{x}_i)$$

$$= \sum_{i=1}^{n} \log\big(P(z_i = 0)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_0, \Sigma_0) + P(z_i = 1)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_1, \Sigma_1)\big)$$

Crap. That is a sum inside of a log. This doesn't factorize nicely, and all our parameters will be intertwined if we take derivatives…

Let's Start Simple – A 2d Case with k=2

$$LL = \sum_{i=1}^{n} \log\Big(P(z_i = 0)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_0, \Sigma_0) + P(z_i = 1)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_1, \Sigma_1)\Big)$$

What if we magically *knew* what value z_i… wouldn't need to marginalize over it anymore

$$LL_{magic} = \sum_{i=1}^{n} \log\Big(P(z_i)\mathcal{N}\big(\boldsymbol{x}_i; \boldsymbol{\mu}_{z_i}, \Sigma_{z_i}\big)\Big) = \sum_{i=1}^{n} \log \mathcal{N}\big(\boldsymbol{x}_i; \boldsymbol{\mu}_{z_i}, \Sigma_{z_i}\big) + \log P(z_i)$$

Easy. This would be great. Things factorize out nicely and its just fitting some Gaussians and a Categorical distribution.

Let's Start Simple – A 2d Case with k=2

What if we magically *knew* what value z_i… wouldn't need to marginalize anymore

$$LL_{magic} = \sum_{i=1}^{n} \log\left(P(z_i)\mathcal{N}\left(\boldsymbol{x}_i; \boldsymbol{\mu}_{z_i}, \Sigma_{z_i}\right)\right) = \sum_{i=1}^{n} \log \mathcal{N}\left(\boldsymbol{x}_i; \boldsymbol{\mu}_{z_i}, \Sigma_{z_i}\right) + \log P(z_i)$$

Could easily find the MLE solutions to this problem:

$$\theta_c^* = \frac{\sum_i \mathbb{I}[z_i = c]}{n}$$

Fraction of points with z = c

$$\boldsymbol{\mu}_c^* = \frac{1}{\sum_i \mathbb{I}[z_i = c]} \sum_i \mathbb{I}[z_i = c]\, \boldsymbol{x}_i$$

Mean of points with z = c

$$\Sigma_c^* = \frac{1}{\sum_i \mathbb{I}[z_i = c]} \sum_i \mathbb{I}[z_i = c]\, (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T$$

Covariance of points with z = c

Could do something very similar if someone gave us "soft" values for z's too.

Note: We didn't derive this. It comes from maximizing a tight lower-bound of the log-likelihood. See the textbook if you are curious.

The Expectation Maximization (EM) Algorithm for GMM

Initialize: probabilities of being in each Gaussian $\theta_1, \dots, \theta_k$ all to 1/k
means of the Gaussians $\mu_1, \dots, \mu_k$ to random points
covariances of the Gaussians $\Sigma_1, \dots, \Sigma_k$ to identity matrices

E-Step: Compute fractional assignment of point $\boldsymbol{i}$ coming from class $\boldsymbol{c}$

$$P(z_i = c \mid \boldsymbol{x}_i) \propto P(z_i = c)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_c, \Sigma_c) \quad \text{denote this} \quad p_{c|x_i}$$

Normalize these probabilities such that $\sum_c p_{c|x_i} = 1$

M-Step: Update the parameters based on the current fractional assignments

$$\theta_c^* = \frac{\sum_i p_{c|x_i}}{n}$$

Fraction of mass assigned to c

$$\boldsymbol{\mu}_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} \boldsymbol{x}_i$$

Weighted mean of fractional points assigned to c

$$\Sigma_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T$$

Weighted covariance of fractional points assigned to c

An Example GMM

Initialization:

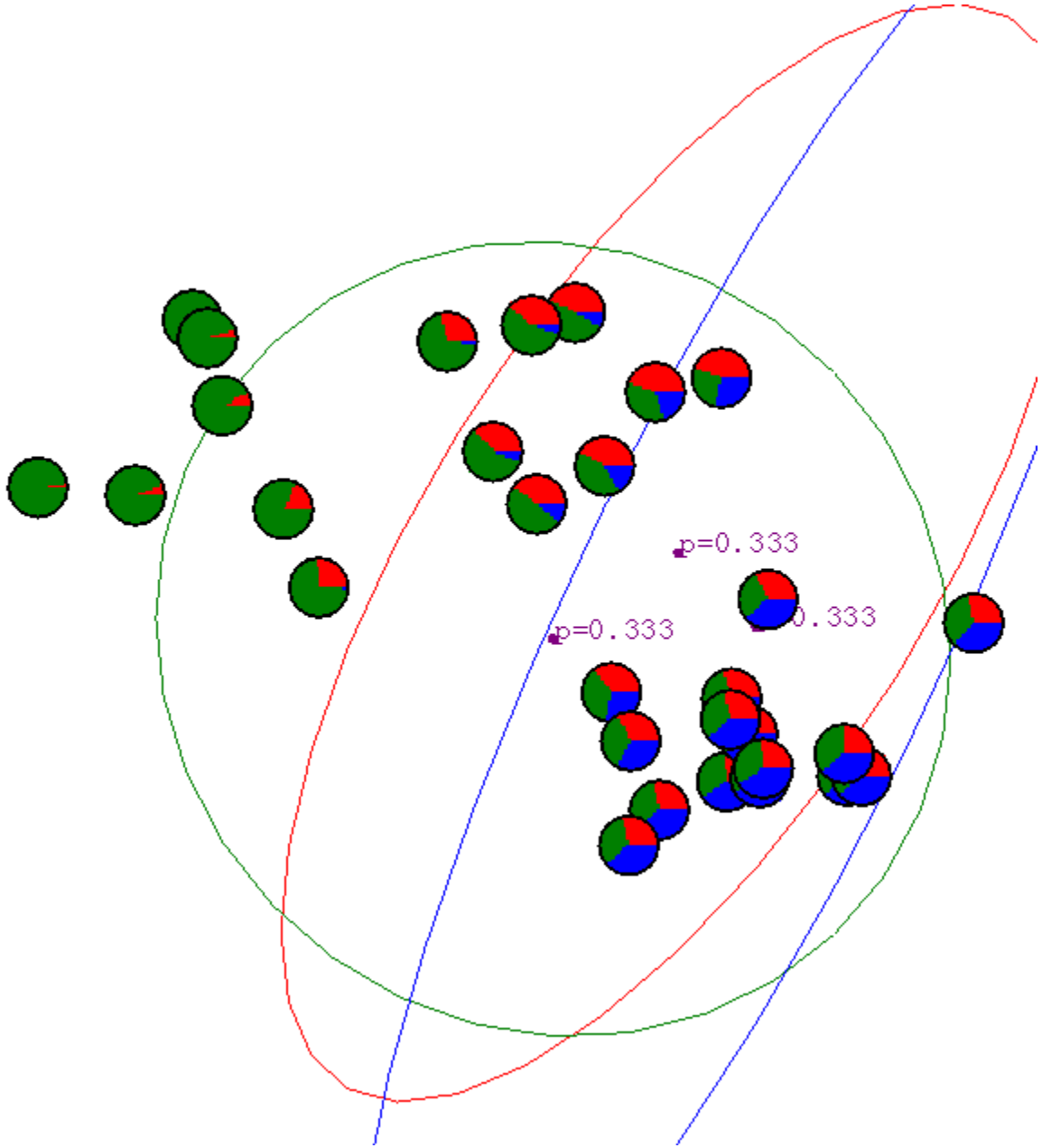


Figure credit Carlos Guestrin

An Example GMM

Iteration 1:

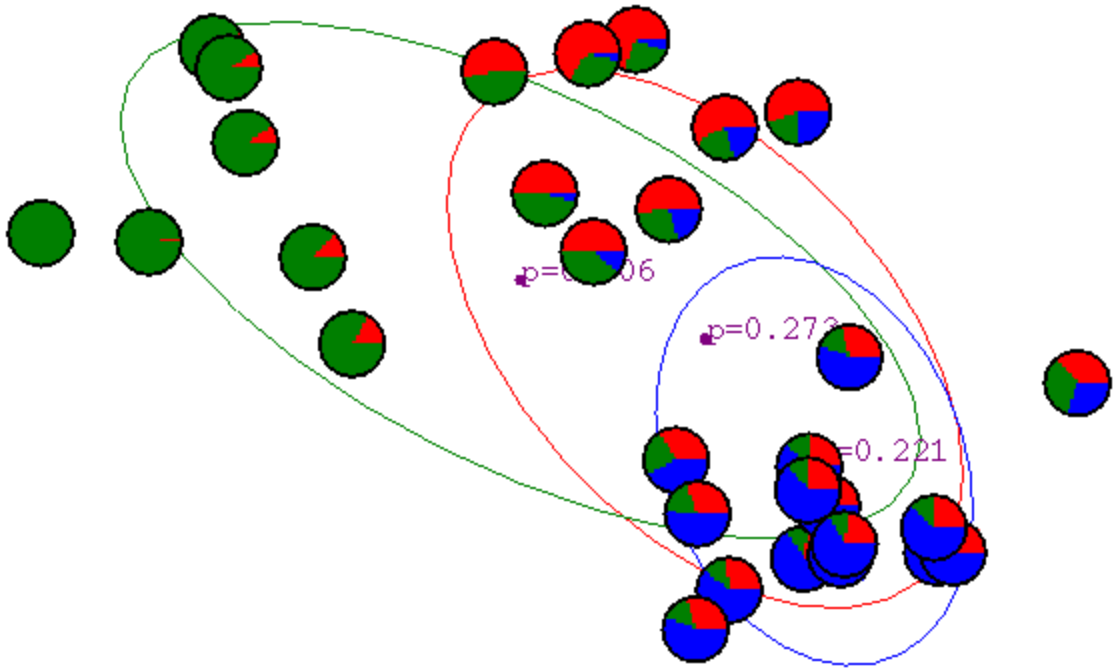


Figure credit Carlos Guestrin

An Example GMM

Iteration 2:

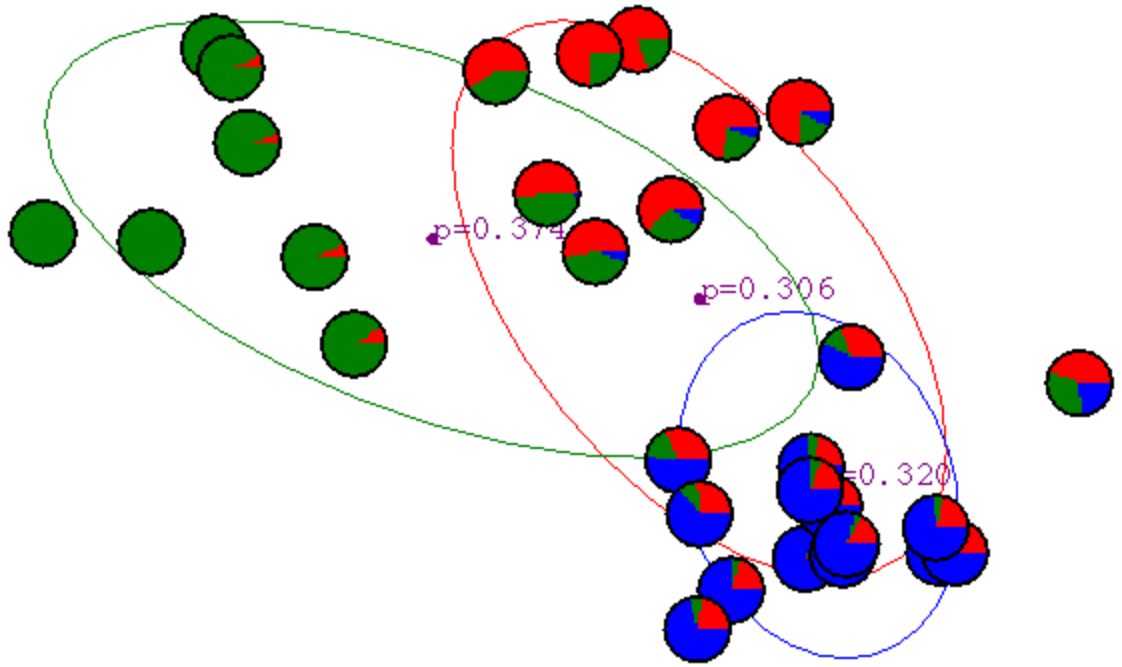


Figure credit Carlos Guestrin

An Example GMM

Iteration 3:

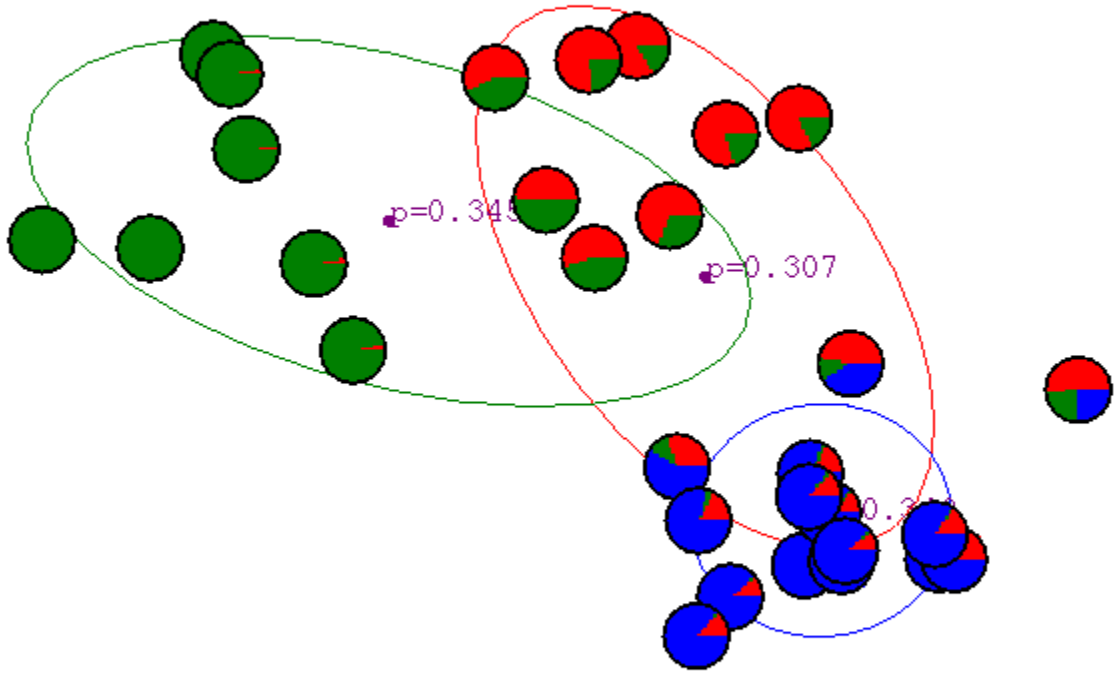


Figure credit Carlos Guestrin

An Example GMM

Iteration 4:

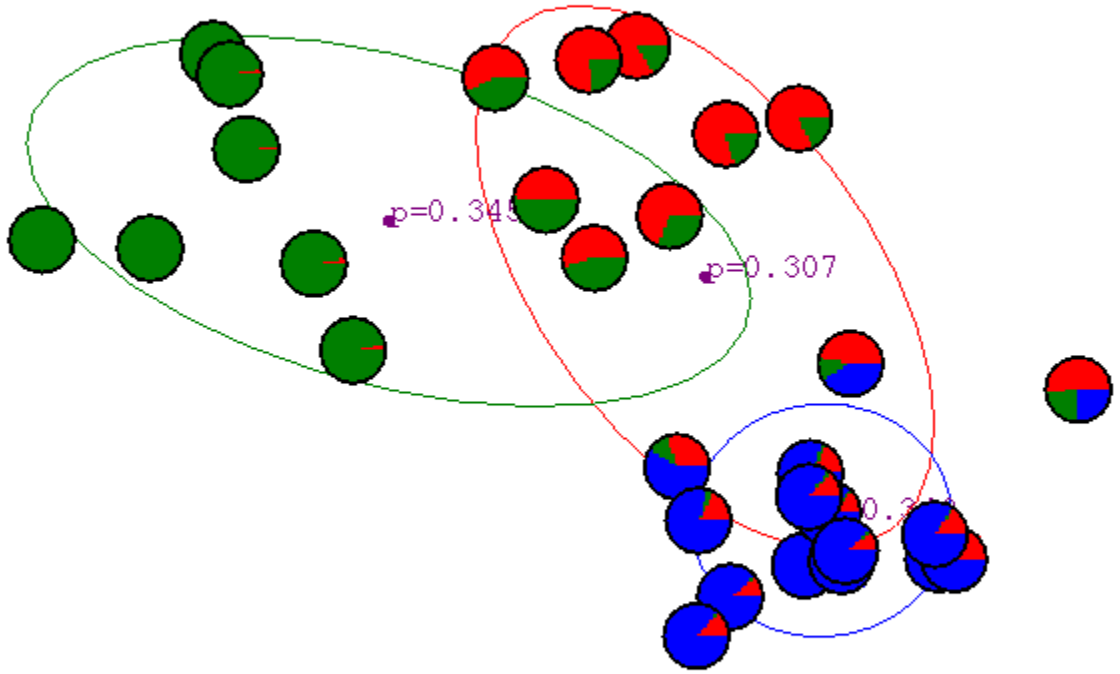


Figure credit Carlos Guestrin

An Example GMM

Iteration 5:

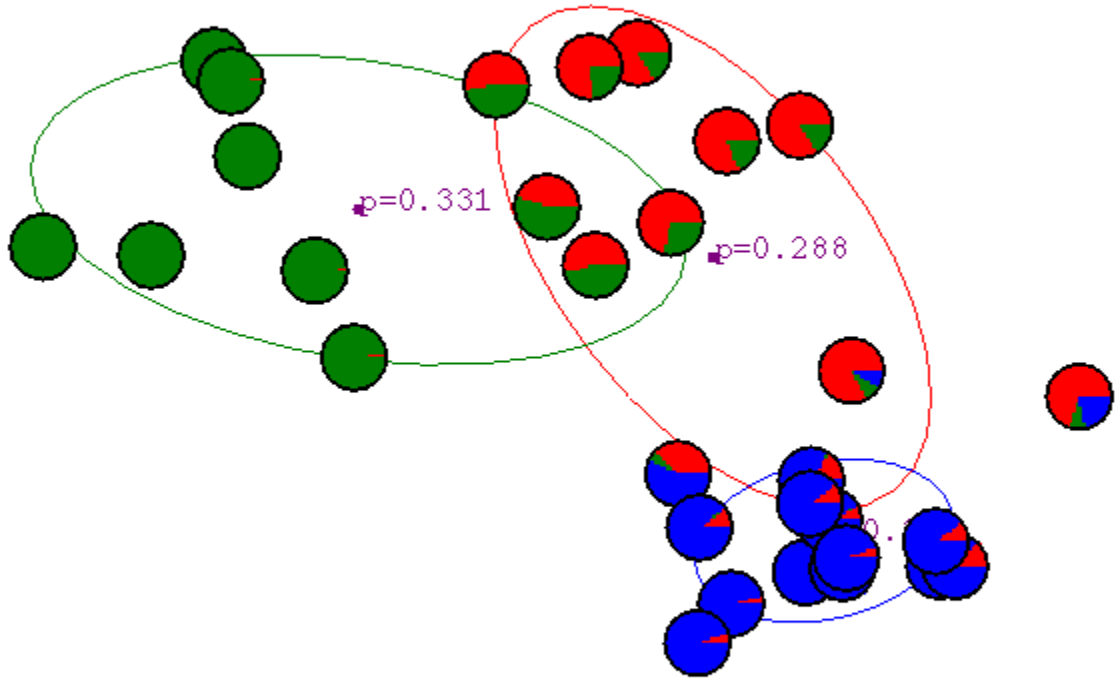


Figure credit Carlos Guestrin

An Example GMM

Iteration 6:

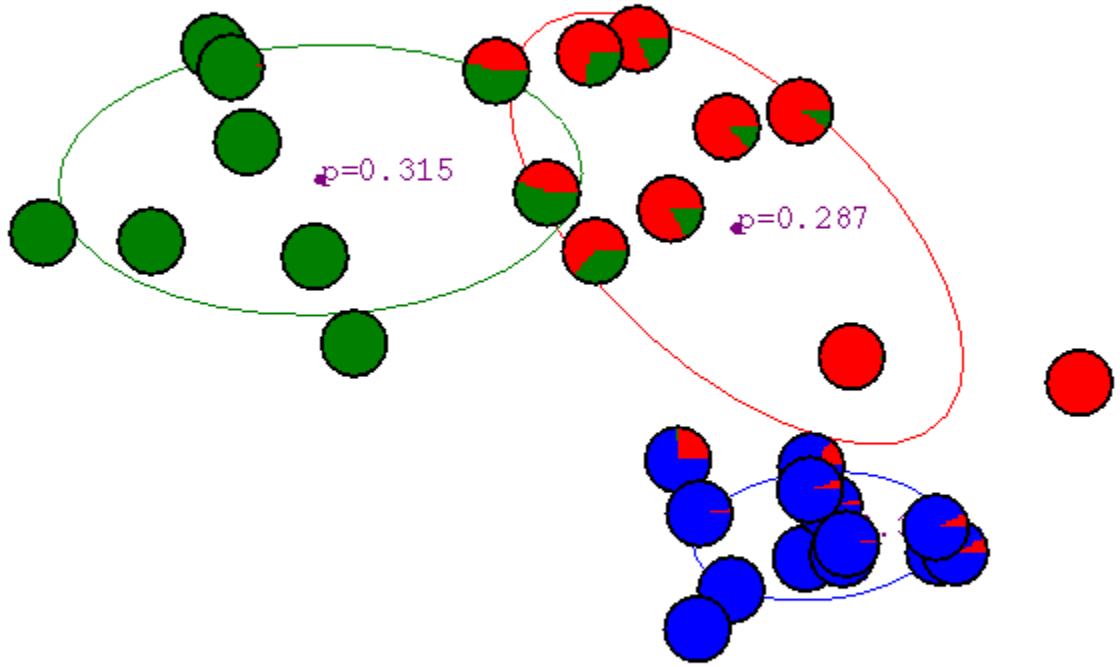


Figure credit Carlos Guestrin

An Example GMM

Iteration 20:

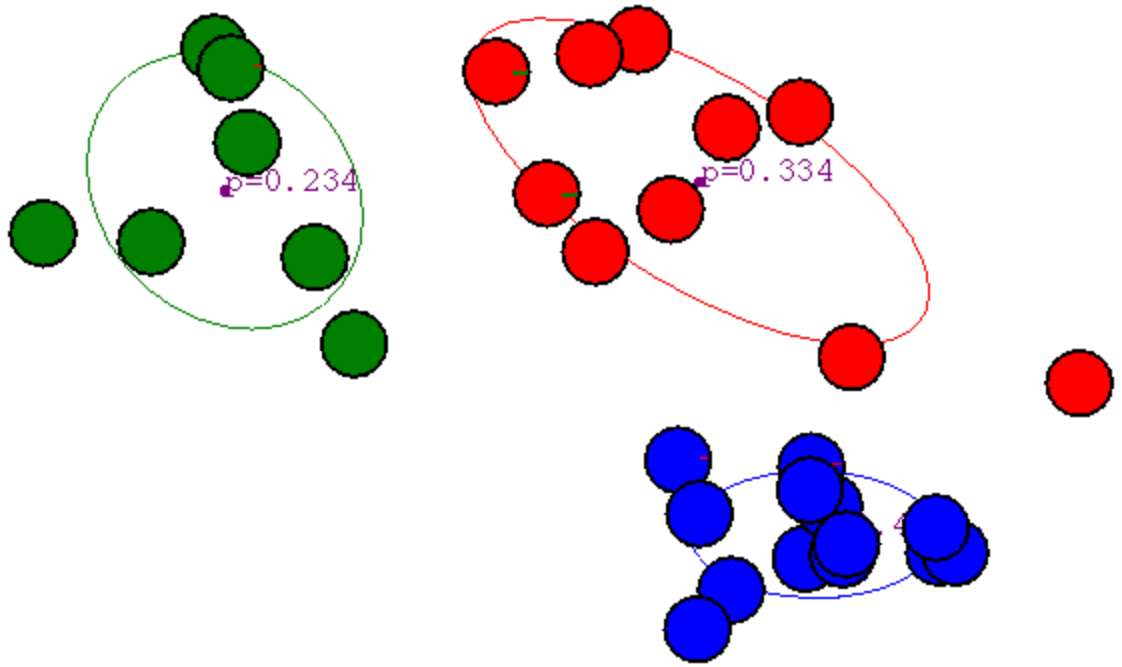
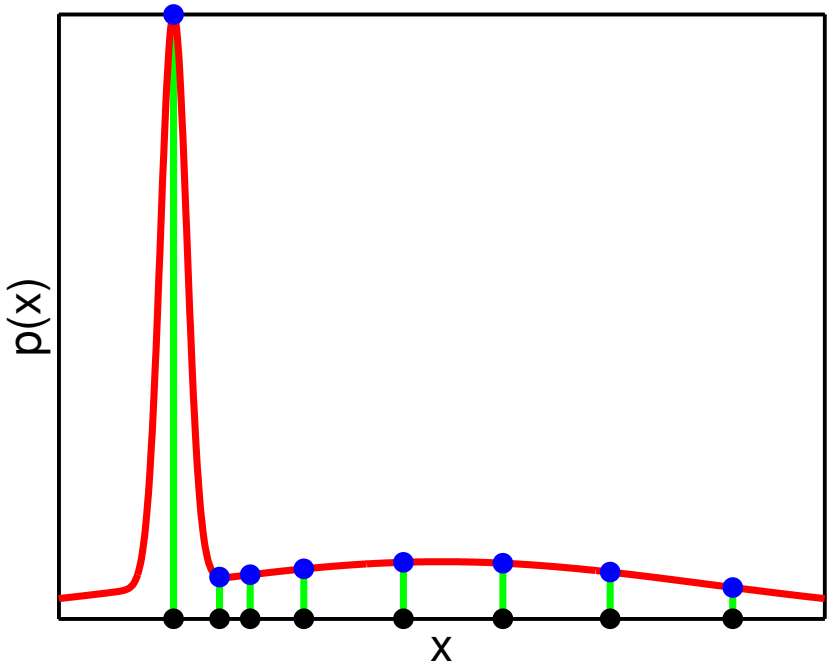


Figure credit Carlos Guestrin

Properties: Singularities in GMM

Log-likelihood can go to infinity if one of the Gaussians "collapses". Assume one of the Gaussians has only one point assigned and the covariance heads towards zero -- $\mu = x$ and $|\Sigma| \to 0$



$$\mathcal{N}(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}$$

$$\lim_{|\Sigma|\to 0} \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} = \infty$$

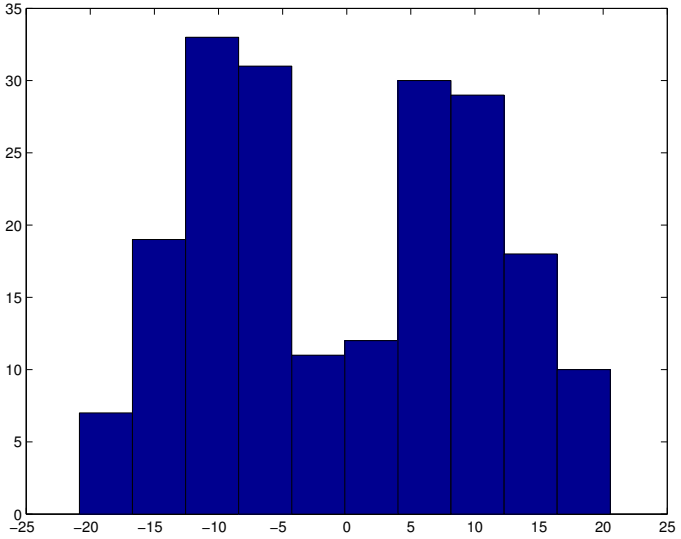
How to deal with this? Monitor each component and reset it to something random if it starts collapsing (random value, large covariance). Or add a prior to the covariance and do MAP in the M-Step of the EM algorithm.

Figure credit Chris Bishop from "Pattern Recognition and Machine Learning" book.

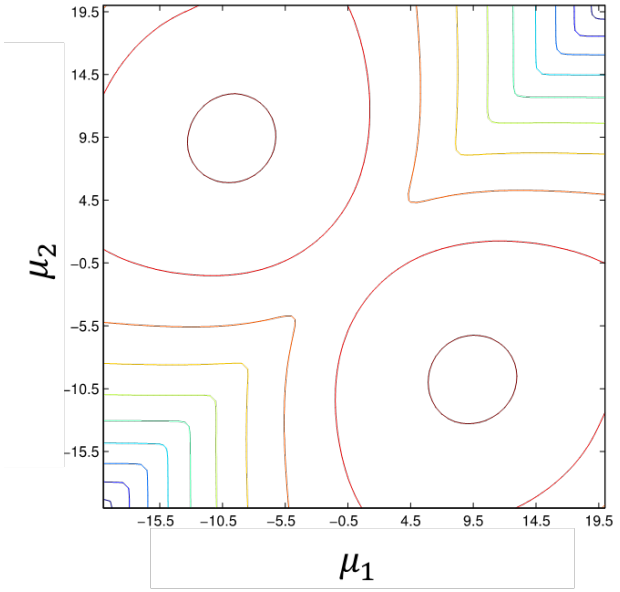
Properties: Identifiability Issues

The log-likelihood in Gaussian Mixture Models has multiple identical maxima. Simple consider swapping the parameters between different models to see why.

Histogram of Data



Log-Likelihood for Different Means of a 2-component GMM



No easy fix. Also limited harm, part of the reason convergence may be slow.

Figure credit Kevin Murphy from "Machine Learning: A Probabilistic Approach" book.

Behavior of Expectation Maximization for GMM

It is guaranteed to converge in finitely many steps:

- Proof looks like the k-Means version but harder – shows that log-likelihood must increase or remain the same between iterations

- In practice, it may converge slowly. Can stop early if no progress is made on log-likelihood for a long time.

Not guaranteed to converge to the global optima:

- Like k-Means, do multiple restarts and then choose the one with highest log likelihood.

- Has a couple of "divergent" solutions

Note: Expectation Maximization (EM) is a whole family of algorithms for dealing with hidden/latent variables. Not just used in Gaussian Mixture Models.

Summary of Gaussian Mixture Models

Assumes data is generated from *k* independent Gaussians:

- Can model more complex configurations than k-Means but is slightly more costly and difficult to implement.

- Produces a full density model of the data → enables you to sample new synthetic data or evaluate the probability of some new point.
  - More on density estimation next class.

- Fractional assignments can be turned into hard clusterings by taking the argmax for each point.

Uses the expectation maximization algorithm for optimization:

- May be slow to converge or end up in trivial optima.
- May need multiple restarts.

What can Gaussian Mixture Models Tell Us About k-Means?

How to get k-Means back from GMMs:

- Assume hard-assignment rather than fractional.
- Assume all Gaussians have the same isotropic covariance.

The Expectation Maximization (EM) Algorithm for GMM

Initialize: probabilities of being in each Gaussian $\theta_1, \ldots, \theta_k$ all to 1/k
means of the Gaussians $\mu_1, \ldots, \mu_k$ to random points
covariances of the Gaussians $\Sigma_1, \ldots, \Sigma_k$ to identity matrices

E-Step: Compute fractional assignment of point $\boldsymbol{i}$ coming from class $\boldsymbol{c}$

$$P(z_i = c \mid \boldsymbol{x}_i) \propto P(z_i = c)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_c, \Sigma_c) \quad \text{denote this} \quad p_{c|x_i}$$

Normalize these probabilities such that $\sum_c p_{c|x_i} = 1$

M-Step: Update the parameters based on the current fractional assignments

$$\theta_c^* = \frac{\sum_i p_{c|x_i}}{n}$$

Fraction of mass assigned to c

$$\boldsymbol{\mu}_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} \boldsymbol{x}_i$$

Weighted mean of fractional points assigned to c

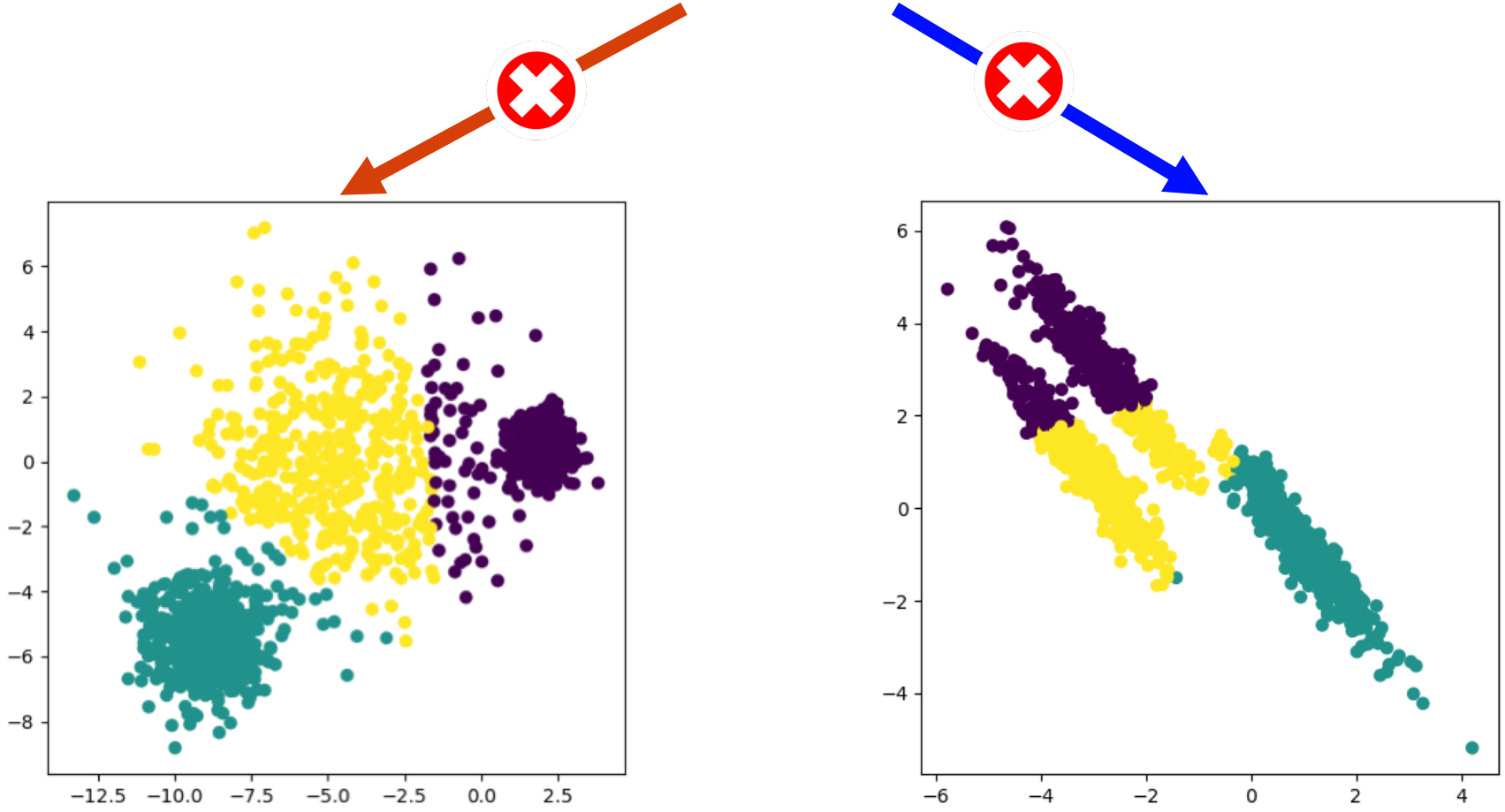
$$\Sigma_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T$$

Weighted covariance of fractional points assigned to c

What can Gaussian Mixture Models Tell Us About k-Means?

How to get k-Means back from GMMs:

- Assume hard-assignment rather than fractional.
- Assume all Gaussians have the **same isotropic** covariance.



Figures adapted from Phil Roth

Question Break!

What all can Gaussian Mixture Models do that k-Means can't?

A Fit clusters with different spreads

B Fit clusters that aren't isotropic / spherical

C Provide an estimate of P(X)

D Allow for new synthetic points to be sampled from the distribution of the training data

Today's Learning Objectives

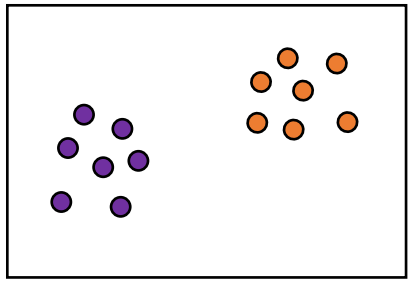
Be able to answer:

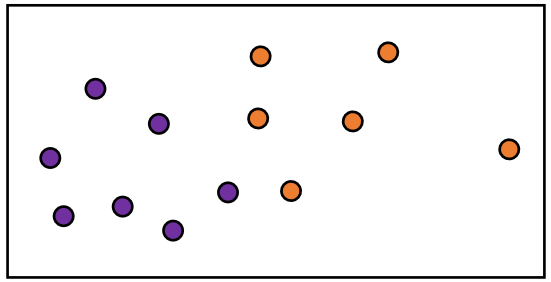
- ~~What is unsupervised learning and how does it differ form supervised learning?~~
  - ~~What are some problems in unsupervised learning?~~
- ~~What is clustering?~~
- ~~How does k-means clustering work?~~
- ~~What is k-means optimizing?~~
  - ~~What is a coordinate descent algorithm?~~
- ~~What is a Gaussian Mixture Model?~~
- ~~What can this tell us about k-Means?~~
- ~~How do we evaluate clustering?~~

This is all cool, but how do I know if I made a good clustering?

Without external data:

- User inspection – (aka just look at it) Does a cluster seem to have a common theme?
  - CAUTION: HUMANS ARE GOOD AT IMAGINING PATTERNS
- Internal Criterion – measure properties of a clustering presumed to be "good"
  - High within-cluster similarity: $s_w = \sum_{j=1}^{k} \sum_{x,x' \in c_j} sim(x, x')$
  - Low between-cluster similarity: $s_b = \sum_{x \in c_i} \sum_{x' \notin c_i} sim(x, x')$





  - This measure depends on the dataset and measure of distance used.

Slide adapted from Xiaoli Fern

Evaluating Clustering

With external data:

- Suppose someone comes along afterwards and gives you labels for all the points, can evaluate the clustering by how well it separated points with different labels.
- **Rand Index** – Given a clustering P and a ground truth label set G, measure the number of vector pairs that are
  - a: in the same group in both P and G (same cluster, same labels)
  - b: in the same group in P but different in G (same cluster, different labels)
  - c: in different groups in P but same in G (different cluster, same labels)
  - d: in different groups in both P and G (different clusters, different labels)

$$Rand\ Index = \frac{a + d}{a + b + c + d}$$

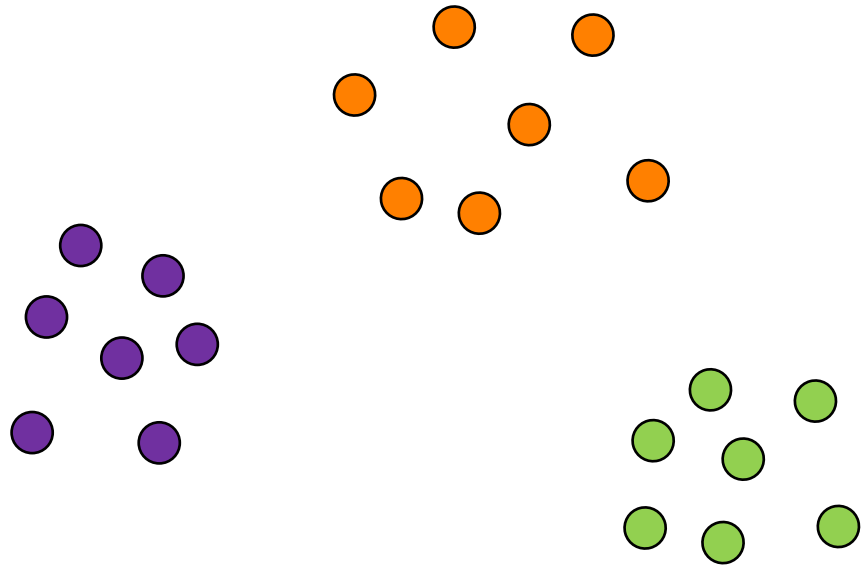
- **Adjusted Rand Index:** correct rand-index by the average rand-index of a random clustering of the data.

Slide adapted from Xiaoli Fern

Evaluating Clustering

With external data:

- Suppose someone comes along afterwards and gives you labels for all the points, can evaluate the clustering by how well it separated points with different labels.
- **Purity** – Fraction of points that would be correctly classified by a "majority vote" per cluster where all points get the label of the majority.

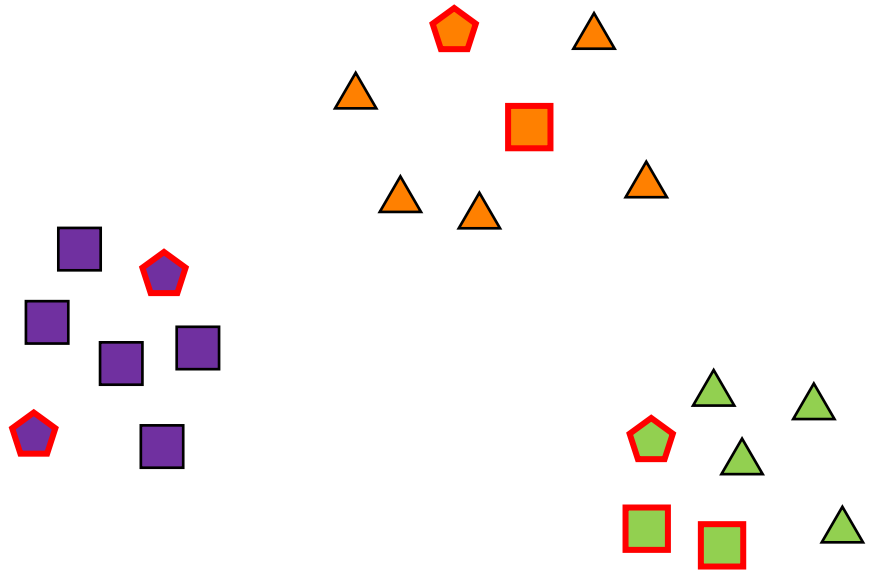


Slide adapted from Xiaoli Fern

Evaluating Clustering

With external data:

- Suppose someone comes along afterwards and gives you labels for all the points, can evaluate the clustering by how well it separated points with different labels.
- **Purity** – Fraction of points that would be correctly classified by a "majority vote" per cluster where all points get the label of the majority.



$$Purity = \frac{5 + 5 + 4}{21}$$

Slide adapted from Xiaoli Fern

Next time

Next Time: We'll talk a bit about hierarchical clustering and then move on to dimensionality reduction and density estimation.