Example: 1D GMM with *k* Clusters

Cluster 1

Probability of generating a point from cluster 1 = π_1

Gaussian mean = μ_1

Gaussian std dev = σ_1

…

Cluster *k*

Probability of generating a point from cluster *k* = π_k

Gaussian mean = μ_k

Gaussian std dev = σ_k

How to generate 1D points from this GMM:

1. Flip biased *k*-sided coin (the sides have probabilities $\pi_1, \ldots, \pi_k$)
2. Let Z be the side that we got (it is some value $1, \ldots, k$)
3. Sample 1 point from Gaussian mean μ_Z, std dev σ_Z

Example: 2D GMM with k Clusters

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = π_1

Gaussian mean = μ_1 2D point

Gaussian **covariance** = Σ_1 2x2 matrix

…

<u>Cluster k</u>

Probability of generating a point from cluster k = π_k

Gaussian mean = μ_k 2D point

Gaussian **covariance** = Σ_k 2x2 matrix

How to generate **2D** points from this GMM:

1. Flip biased k-sided coin (the sides have probabilities $\pi_1, \ldots, \pi_k$)
2. Let Z be the side that we got (it is some value $1, \ldots, k$)
3. Sample 1 point from Gaussian mean μ_Z, **covariance** Σ_Z

GMM with k Clusters

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = π_1

Gaussian mean = μ_1

Gaussian covariance = Σ_1

…

<u>Cluster k</u>

Probability of generating a point from cluster k = π_k

Gaussian mean = μ_k

Gaussian covariance = Σ_k

How to generate points from this GMM:

1. Flip biased k-sided coin (the sides have probabilities $\pi_1, \ldots, \pi_k$)
2. Let Z be the side that we got (it is some value $1, \ldots, k$)
3. Sample 1 point from Gaussian mean μ_Z, covariance Σ_Z

High-Level Idea of GMM

- Generative model that gives a *hypothesized* way in which data points are generated

In reality, data are unlikely generated the same way!

In reality, data points might not even be independent!

“All models are wrong, but some are useful.”

–George Edward Pelham Box

High-Level Idea of GMM

- Generative model that gives a *hypothesized* way in which data points are generated

In reality, data are unlikely generated the same way!

In reality, data points might not even be independent!

- Learning ("fitting") the parameters of a GMM
  - Input: *d*-dimensional data points, your guess for *k*
  - Output: $\pi_1, \ldots, \pi_k, \mu_1, \ldots, \mu_k, \Sigma_1, \ldots, \Sigma_k$
- *After* learning a GMM:
  - For *any* *d*-dimensional data point, can figure out probability of it belonging to each of the clusters

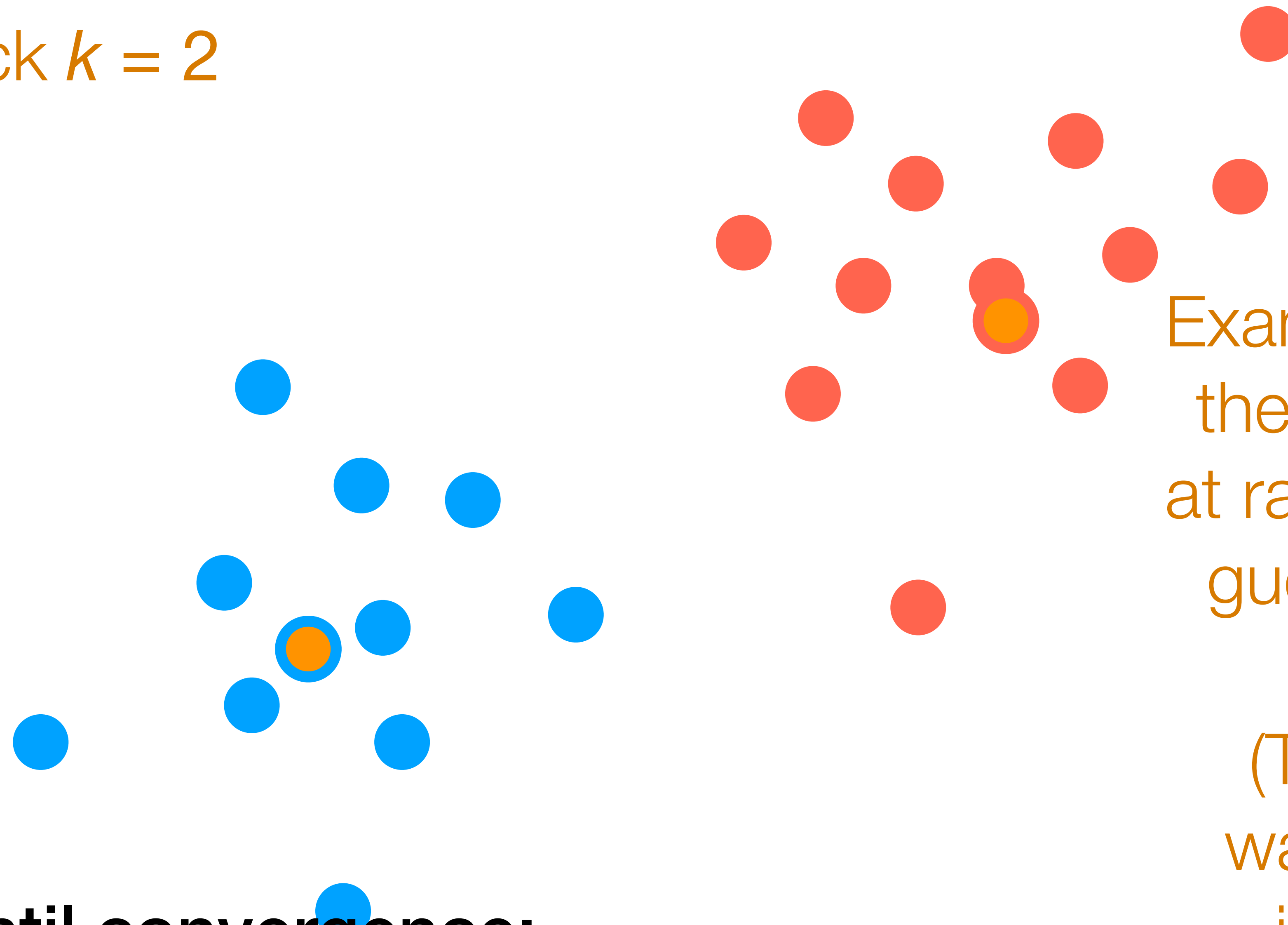
How do you turn this into a cluster assignment?

k-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick <u>guesses</u> for where cluster centers are



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Repeat until convergence:

Step 2: Assign each point to belong to the closest cluster

Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Step 0: Pick *k*

Step 1: Pick <u>guesses</u> for
where cluster centers are

Repeat until convergence:

Step 2: Assign each point to belong to the closest cluster

Step 3: Update cluster means (to be the center of mass per cluster)

(Rough Intuition) Learning a GMM

Step 0: Pick *k*

Step 1: Pick <u>guesses</u> for **cluster means and covariances**

Repeat until convergence:

Step 2: Compute probability of each point belonging to each of the *k* clusters

Step 3: Update **cluster means and covariances** carefully accounting for probabilities of each point belonging to each of the clusters

This algorithm is called the Expectation-Maximization (EM) algorithm specifically for GMM's (and approximately does maximum likelihood)

(Note: EM by itself is a general algorithm not just for GMM's)

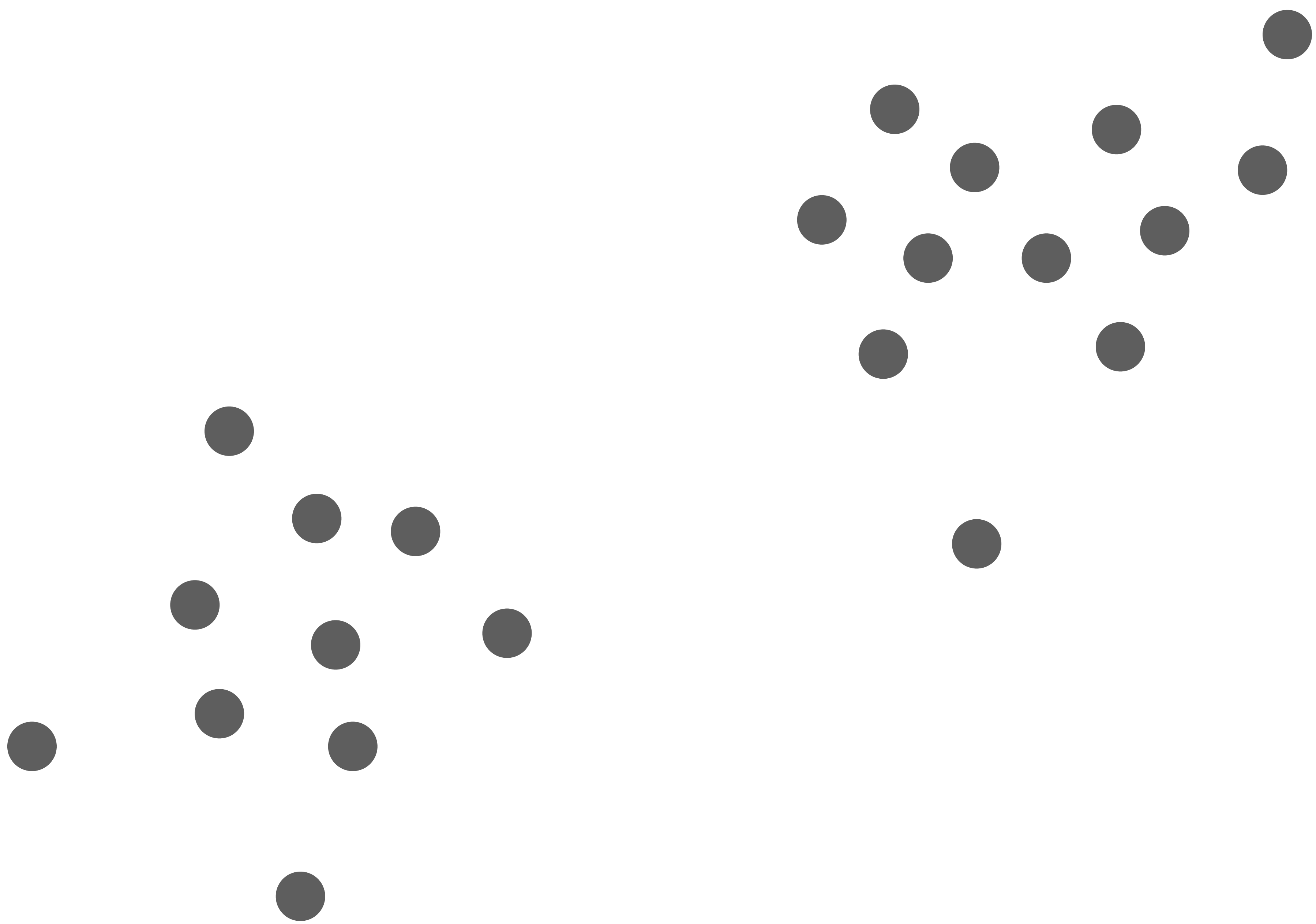
Relating *k*-means to GMM's

If the ellipses are all circles and have the same "skinniness" (e.g., in the 1D case it means they all have same std dev):

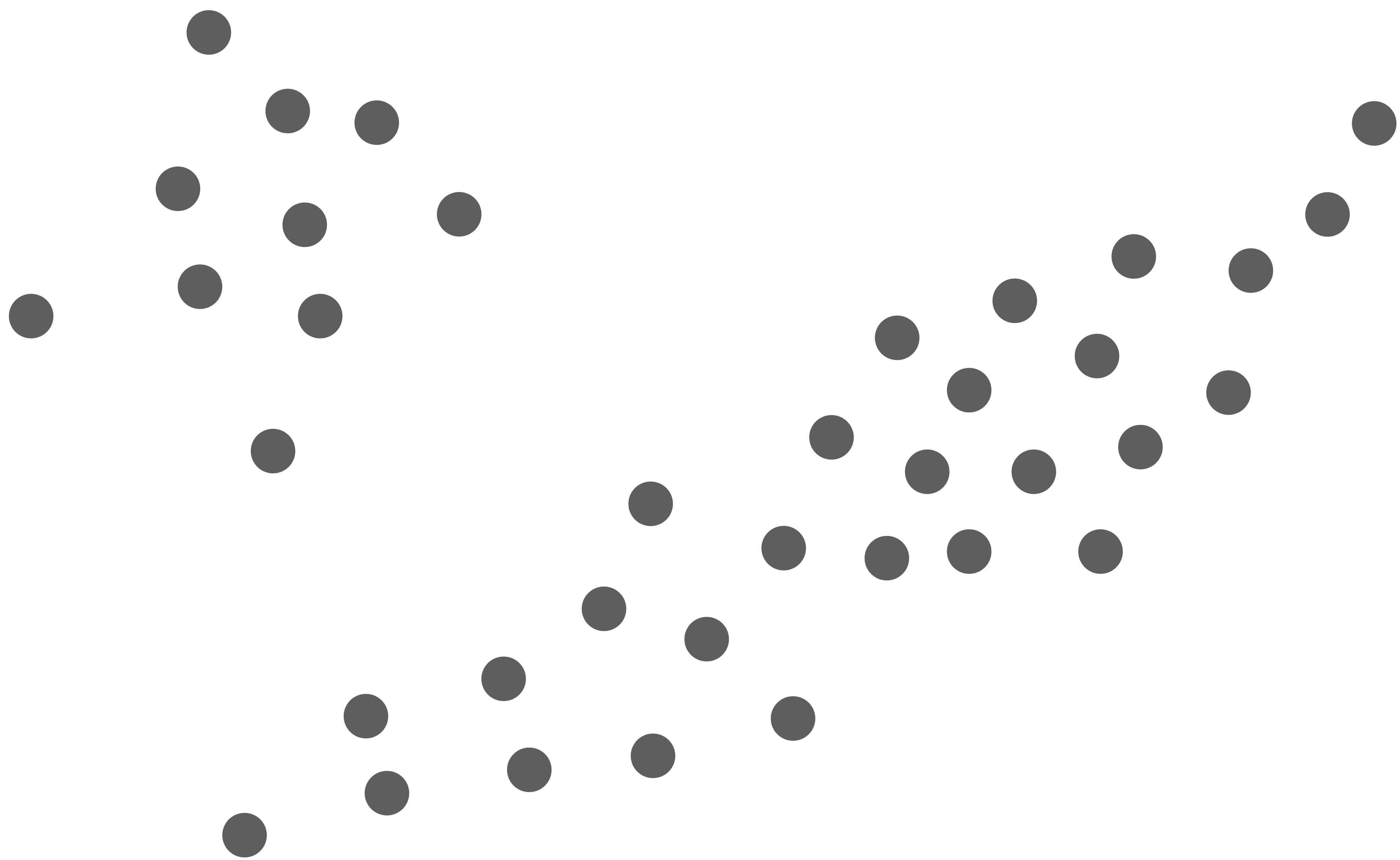
- *k*-means approximates the EM algorithm for GMM's
- Notice that *k*-means does a "hard" assignment of each point to a cluster, whereas the EM algorithm does a "soft" (probabilistic) assignment of each point to a cluster

Interpretation: *We know when k-means should work! It should work when the data appear as if they're from a GMM with true clusters that "look like circles"*

k-means should do well on this



But not on this



Learning a GMM

Demo

Automatic Selection of *k*

Dirichlet Process Gaussian Mixture Model (DP-GMM):

- Number of clusters is effectively random, and *can grow with the amount of data you have!*

- While you don't have to choose *k*, you have to choose a different parameter which says basically how likely new points are to form new clusters vs join existing clusters

DP-GMM High-Level Idea

Cluster 1	Cluster 2	Cluster 3	
Probability of generating a point from cluster 1 = π_1	π_2	π_3	…
Gaussian mean = μ_1	μ_2	μ_3	It goes on forever!
Gaussian covariance = Σ_1	Σ_2	Σ_3	

There is a parameter that controls how these π values roughly decay

There are an infinite number of parameters

(Rough idea) How to generate points from this DP-GMM:

1. Flip biased ∞-sided coin (the sides have probabilities π_1, π_2, π_3, …)
2. Let Z be the side that we got (it is a positive integer)
3. Sample 1 point from Gaussian mean μ_Z, covariance Σ_Z

Remark: For any given dataset, when learning the DP-GMM, there aren't going to be an infinite number of clusters found

Automatic Selection of *k*

Dirichlet Process Gaussian Mixture Model (DP-GMM):

- Number of clusters is effectively random, and *can grow with the amount of data you have!*

- While you don't have to choose k, you have to choose a different parameter which says basically how likely you are to form new clusters vs try to stick to already existing clusters

- An example of a *Bayesian nonparametric model* (roughly: a generative model with an *infinite number of parameters*, where the *parameters are random*)

Learning a DP-GMM

Two main approaches:

- Finite approximation where you specify some maximum number of possible clusters (the algorithm will find up to that many clusters)

  This is what's implemented in *sklearn*

  - Algorithm is somewhat similar to *k*-means/EM for GMMs
  - Algorithm output: very similar to regular GMM fitting

- Random sampling approach (no finite approximation needed!)
  - Algorithm output: a bunch of samples of different cluster assignments (can pick one with highest probability)

  This is what's implemented in R (package *dpmixsim*)

Learning a DP-GMM

Demo