ECE 20875

Python for Data Science

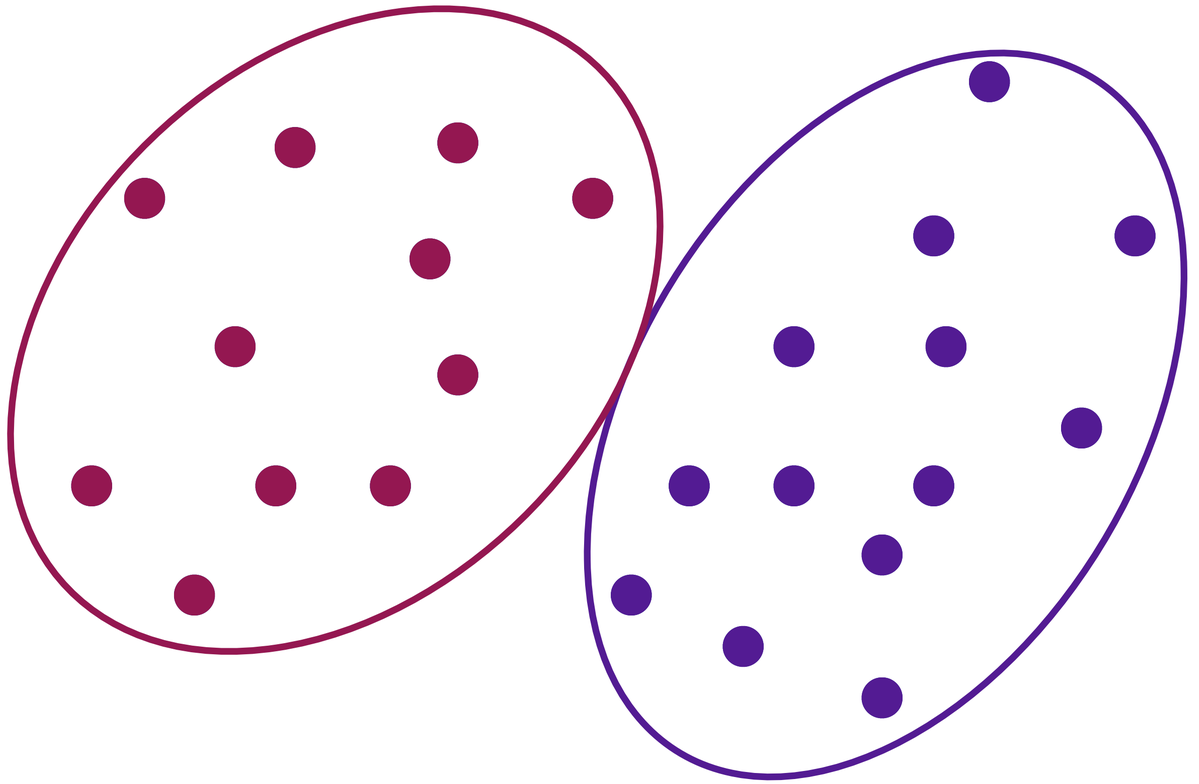
Chris Brinton, Qiang Qiu, and Mahsa Ghasemi

(Adapted from material developed by Profs. Milind Kulkarni, Stanley Chan, Chris Brinton, David Inouye, Qiang Qiu)

clustering

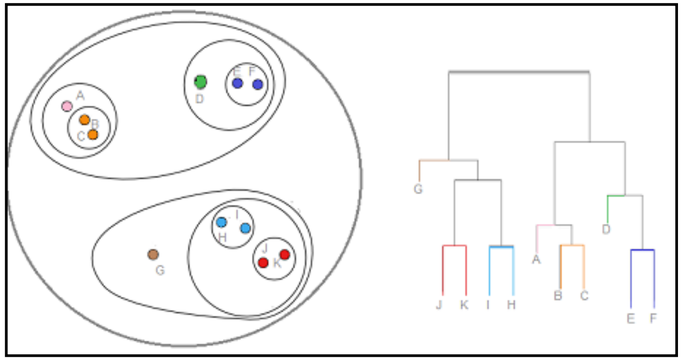
what is clustering?

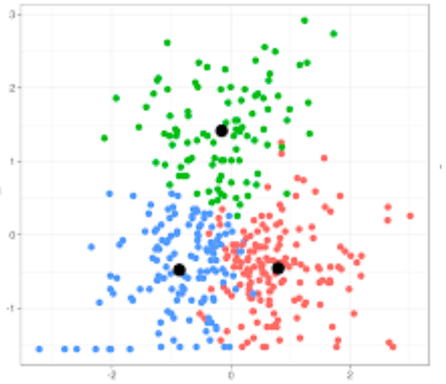
- Given a set of data points, group them into **clusters**, i.e., subsets of the data set that are “similar”
- What “similar” means depends on what clustering algorithm you use
  - K-means: Points are “near” each other
  - Gaussian mixture models: Points come from same Gaussian distribution
- Basic goal: Identify structure in data *without any labels*
- Lack of labels for the data points makes this **unsupervised learning**
  - We will discuss **supervised learning** more again later (where have we seen it already?)

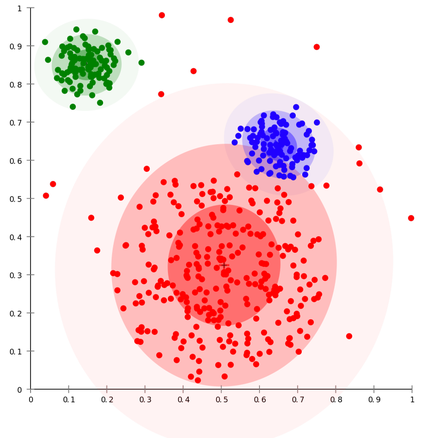


clustering algorithms

- There are probably hundreds of published algorithms for clustering datasets
- Clustering algorithms fall into a few general categories:
  - **Hierarchical**: Group datapoints together based on how “close” they are to one another
  - **Centroid-based**: Find center points that can be interpreted as cluster centers, and assign each point to one center
  - **Distribution-based**: Datapoints are grouped based on their likelihood of belonging to the same probability distribution
  - **Density-based**: Clusters are defined as areas of higher density in the dataset







two approaches

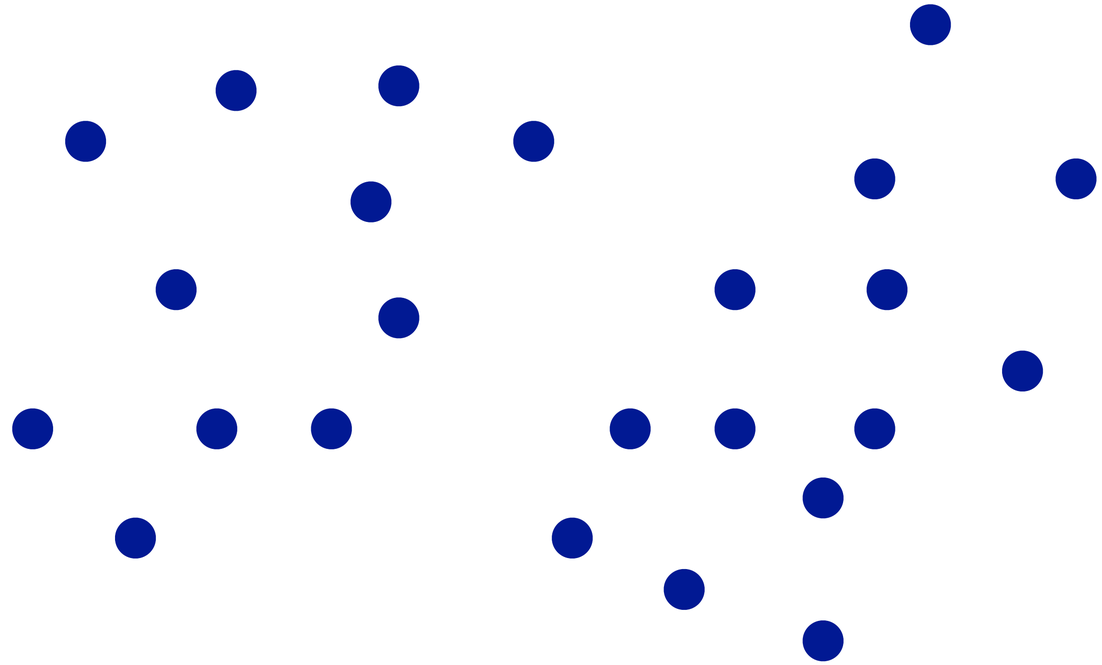
- We will study two of the most popular clustering methods: K-means and GMMs
  - **K-means**
    - Centroid-based
    - No model required
    - Can only find “simple” structure in data (points that are close together)
  - **Gaussian Mixture Models (GMMs)**
    - Distribution-based
    - Requires having a model in mind
    - Can find interesting structure in data (based on how complex the model is)
- Despite their different flavors, we will also see that both models …
  - require an initial assumption about the number of clusters
  - use **iterative algorithms** to find clusters: Form an initial guess, and refine it

k-means clustering

- Consider a dataset consisting of n points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, where $\mathbf{x}_j$ is the feature vector representation of observation j (and there is no y_j)

- With ***k*-means**, we seek to divide the dataset into k clusters $S_1, S_2, \ldots, S_k$, where each cluster S_i is defined by a centroid μ_i
  - μ_i is the mean of all the datapoints in S_i

- Formally, we seek to assign each $\mathbf{x}_j$ to a cluster S_i according to the following optimization problem:

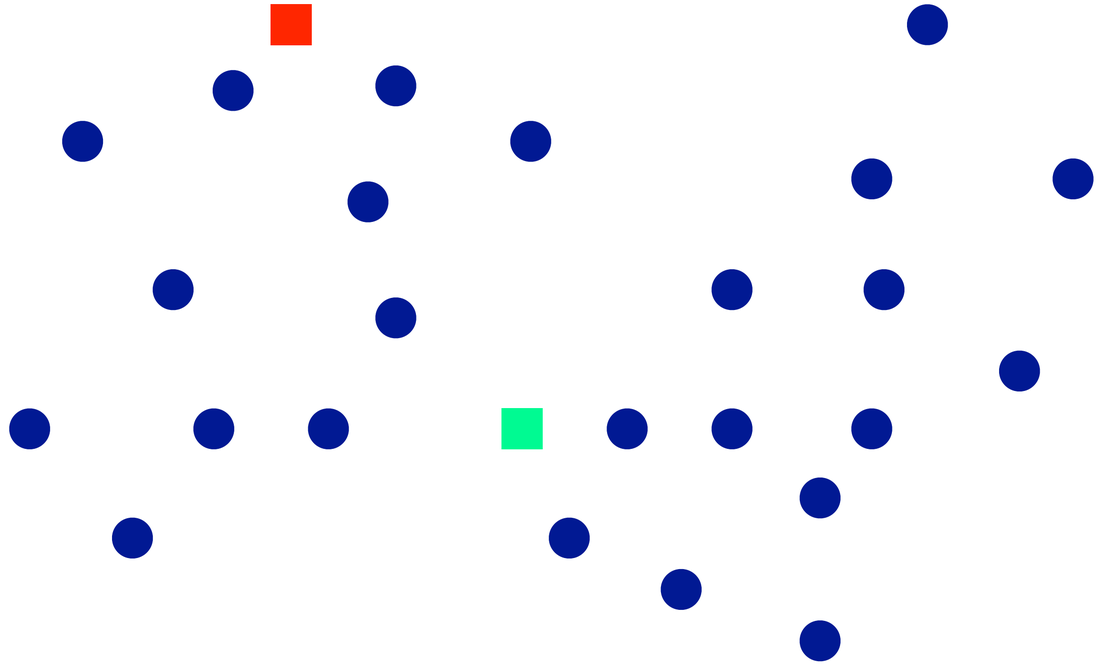


$$\underset{S_1,\ldots,S_k}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mu_i\|^2$$

Hard to solve: μ_i depends on S_i, and S_i depends on μ_i!

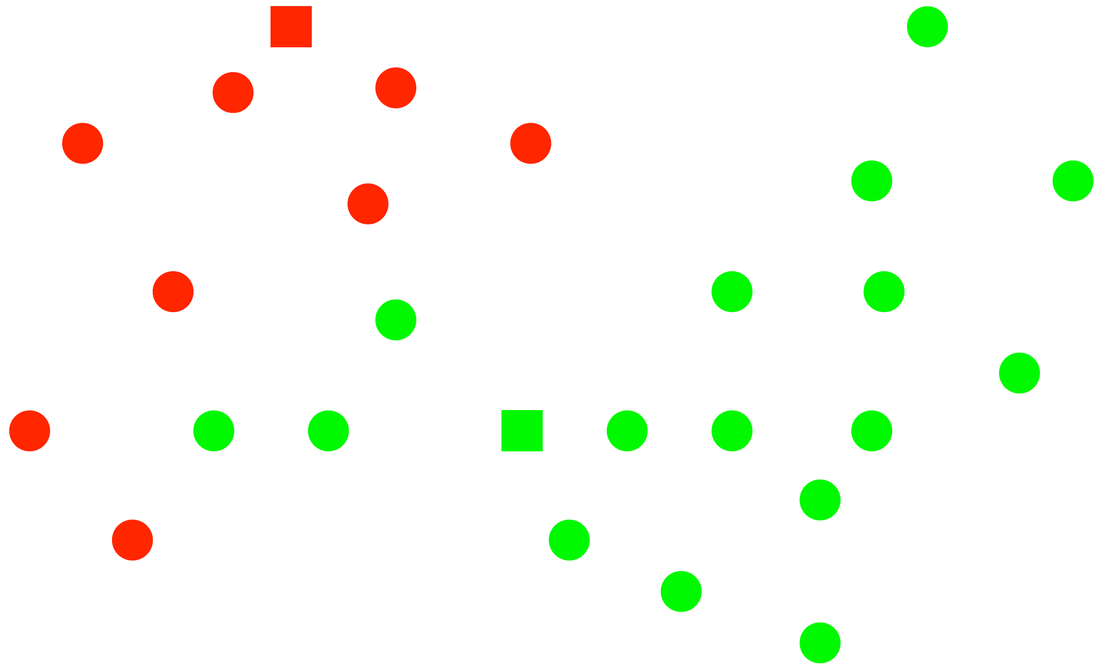
k-means algorithm

- Start out by initializing k “centroids” that define the clusters
  - Could just be random choice



k-means algorithm

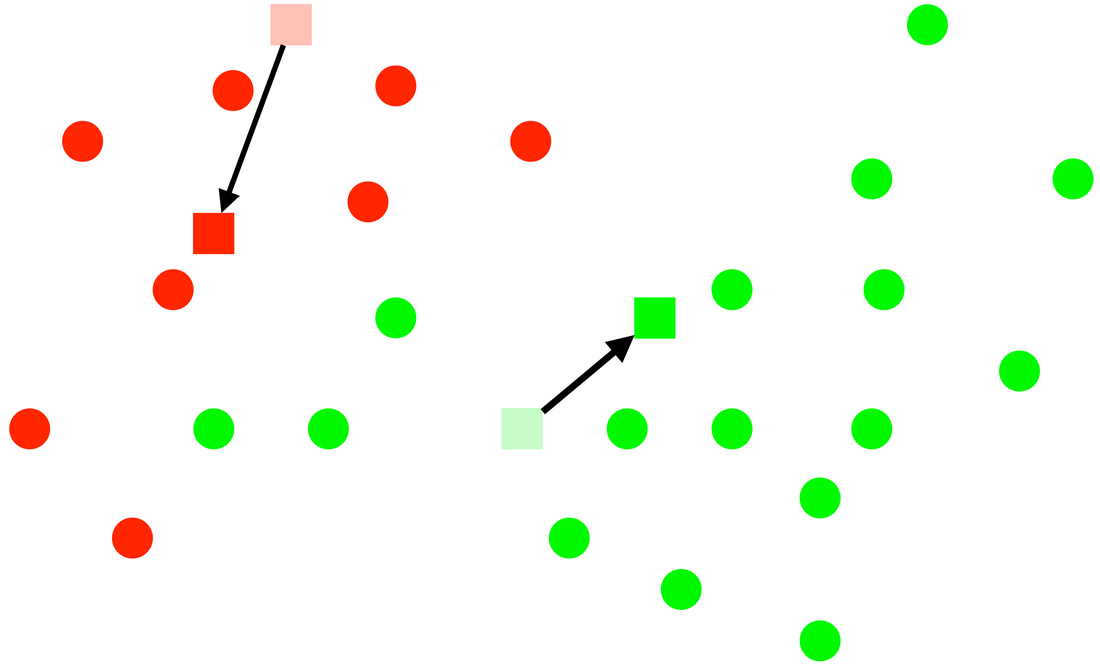
- Start out by initializing k “centroids” that define the clusters
- **Assignment step**: Assign each data point to a cluster
  - Each data point is assigned to the cluster it is closest to
  - According to Euclidean distance, i.e., $\arg\min_i \|\mathbf{x}_j - \mu_i\|$



k-means algorithm

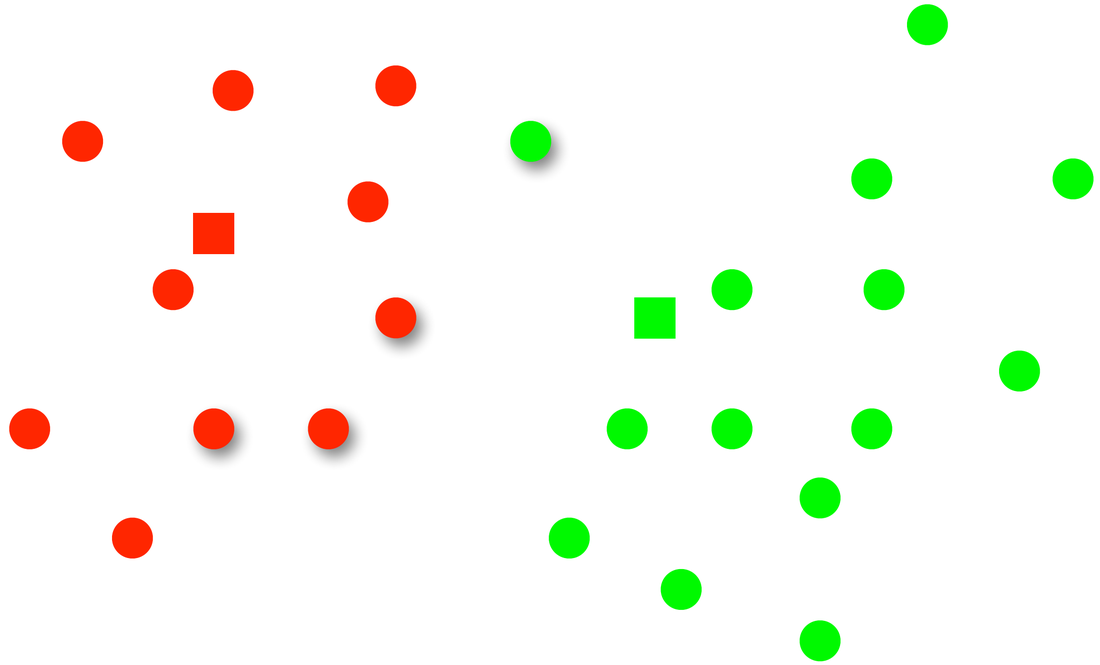
- Start out by initializing k “centroids” that define the clusters
- **Assignment step**: Assign each data point to a cluster
- **Update step**: Move each centroid to the “middle” of its cluster
  - Compute the average position of the data points
  - Compute mean according to

$$\mu_i = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \mathbf{x}$$



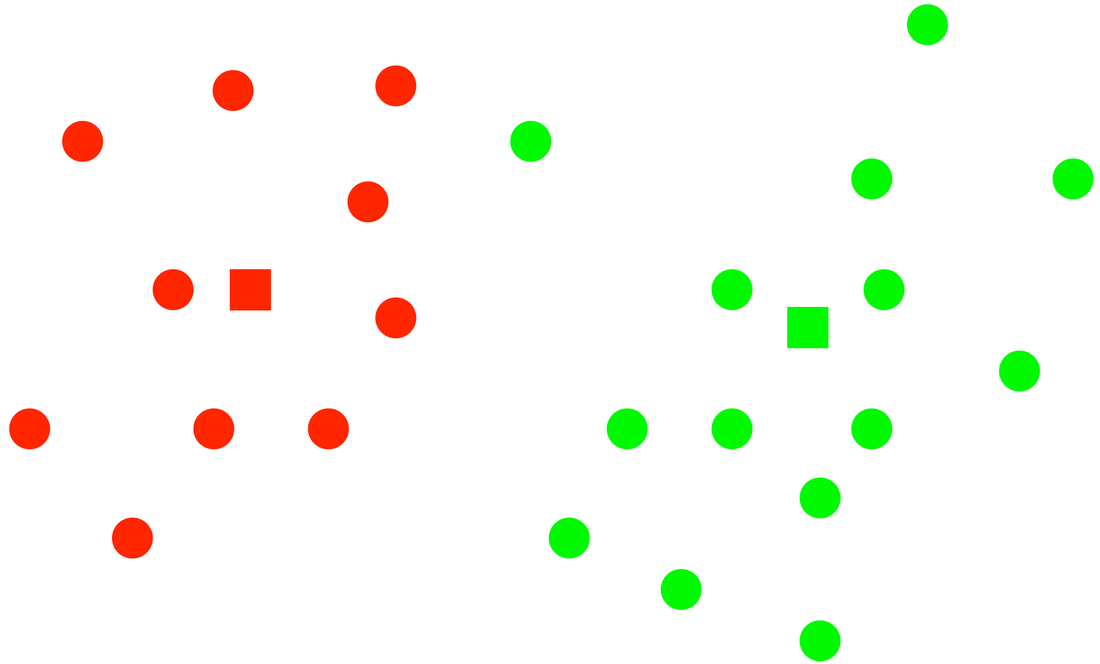
k-means algorithm

- Start out by initializing k “centroids” that define the clusters
- **Assignment step**: Assign each data point to a cluster
- **Update step**: Move each centroid to the “middle” of its cluster
- Repeat assignment step with new centroid locations



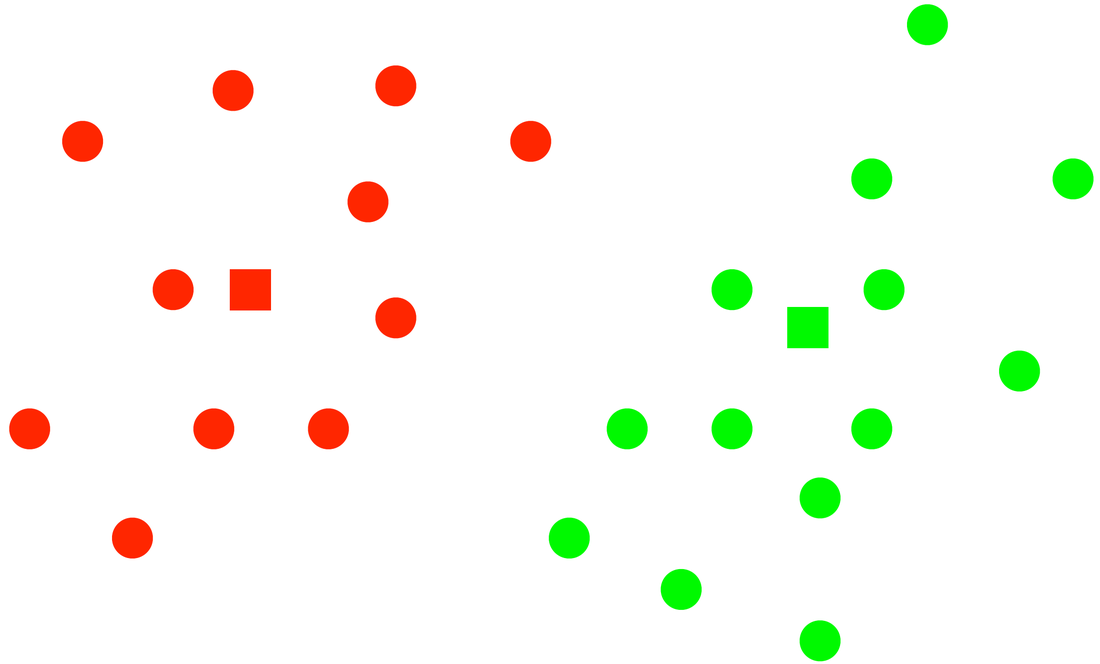
k-means algorithm

- Start out by initializing k “centroids” that define the clusters
- **Assignment step**: Assign each data point to a cluster
- **Update step**: Move each centroid to the “middle” of its cluster
- Repeat assignment step with new centroid locations
- Repeat update step with new clusters



k-means algorithm summary

- Start out by initializing k “centroids” that define the clusters
- **Assignment step**: Assign each data point to a cluster
- **Update step**: Move each centroid to the “middle” of its cluster
- Repeat assignment, update, assignment, update, … until convergence



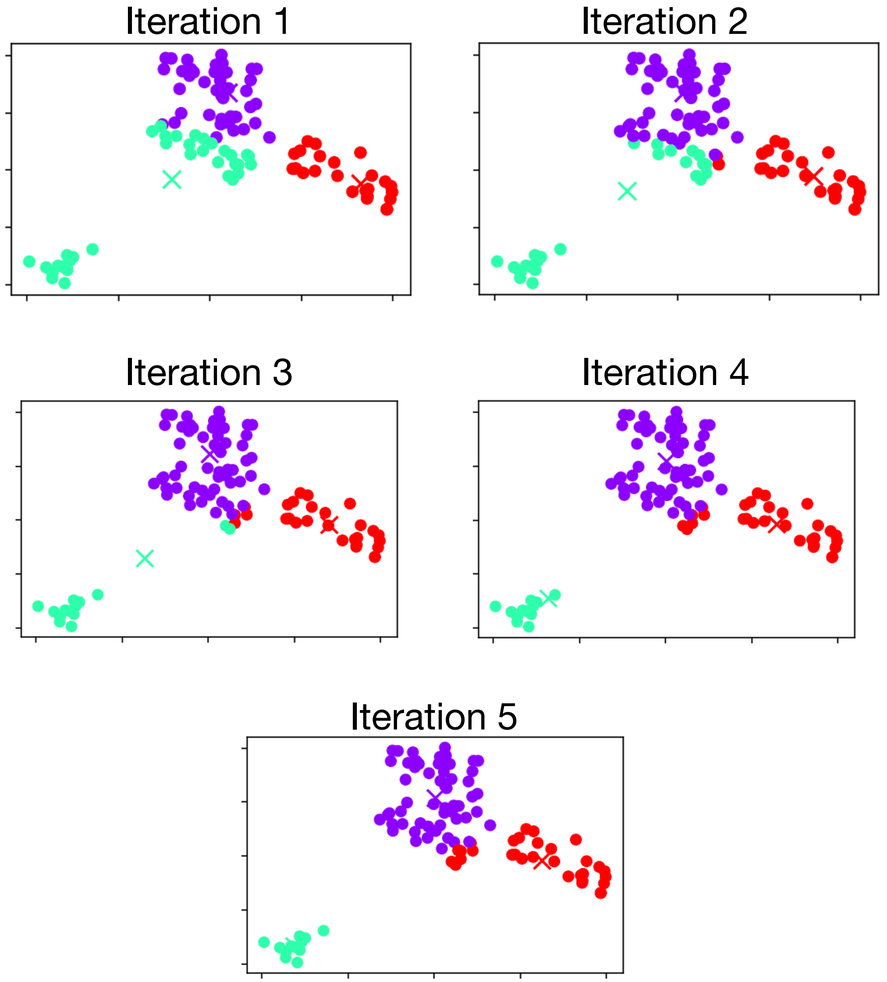
implementation and considerations

- In Python: `KMeans` class from `sklearn.cluster` (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)

```
kmeans = KMeans(n_clusters, n_init,
random_state, …)   #kmeans object

kmeans.fit(X)   #fit kmeans to X

kmeans.labels_  #Cluster assignments of X

kmeans.cluster_centers_  #Cluster centers
```

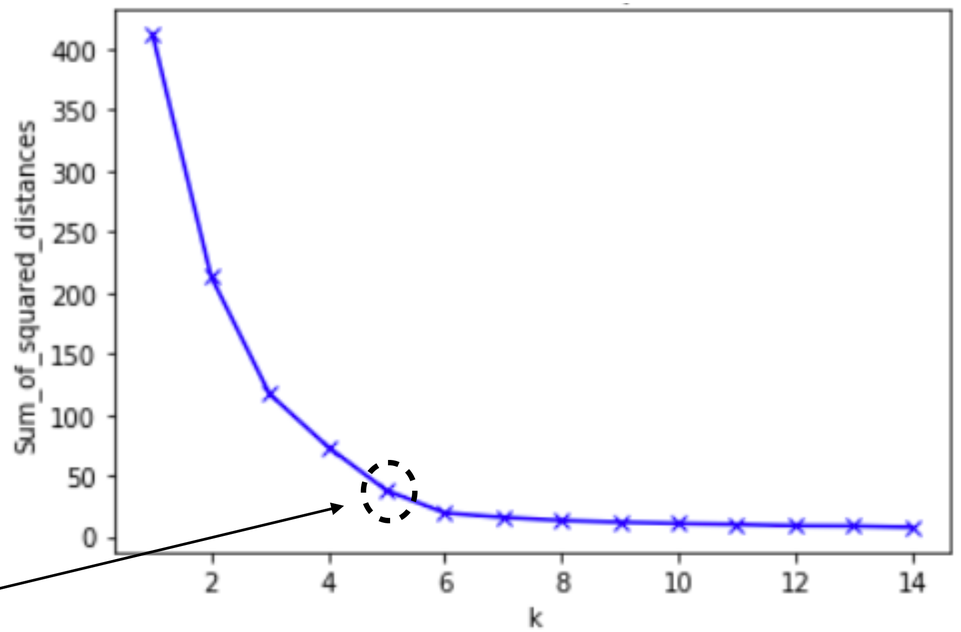
- A simpler approach than GMMs (which we will see next): No need for an *a priori* model

- But is less sophisticated than GMMs: Clusters that *k*-means finds have limitations



choosing k

- How do we know how many centroids to start with?
- One possibility: Can pick a k and see how far points in a cluster are from their centroid
  - If there are too few centroids, average distance is high
- As seen on the right, as k increases, the distance drops
  - But drop “slows down” after a while
  - **Knee** or **elbow method**: Look for the “knee” of the curve
- Can also use cross validation!
  - Too many clusters: New data points are not well represented by the clusters



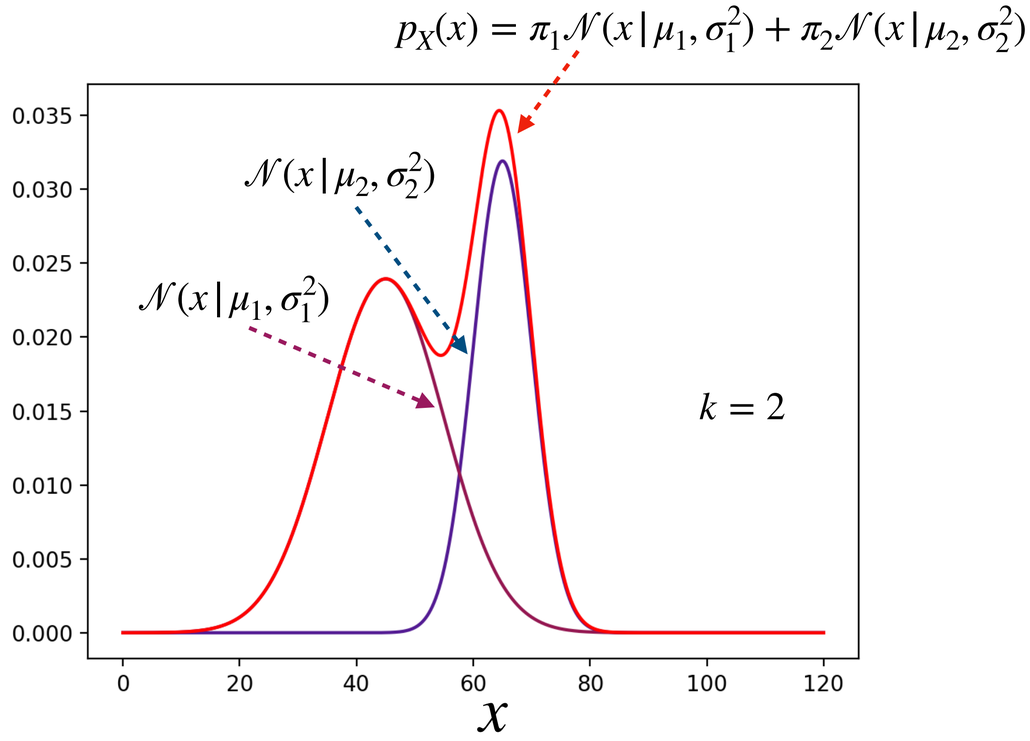
gaussian mixture models

- A **Gaussian mixture model (GMM)** with k components (clusters) is a probability distribution that is a weighted sum of k Gaussians:

$$p_X(x) = \sum_{i=1}^{k} \pi_i \mathcal{N}(x \mid \mu_i, \sigma_i^2)$$

  - μ_i : mean of ith Gaussian
  - σ_i^2 : variance of ith Gaussian
  - π_i : weight of ith Gaussian

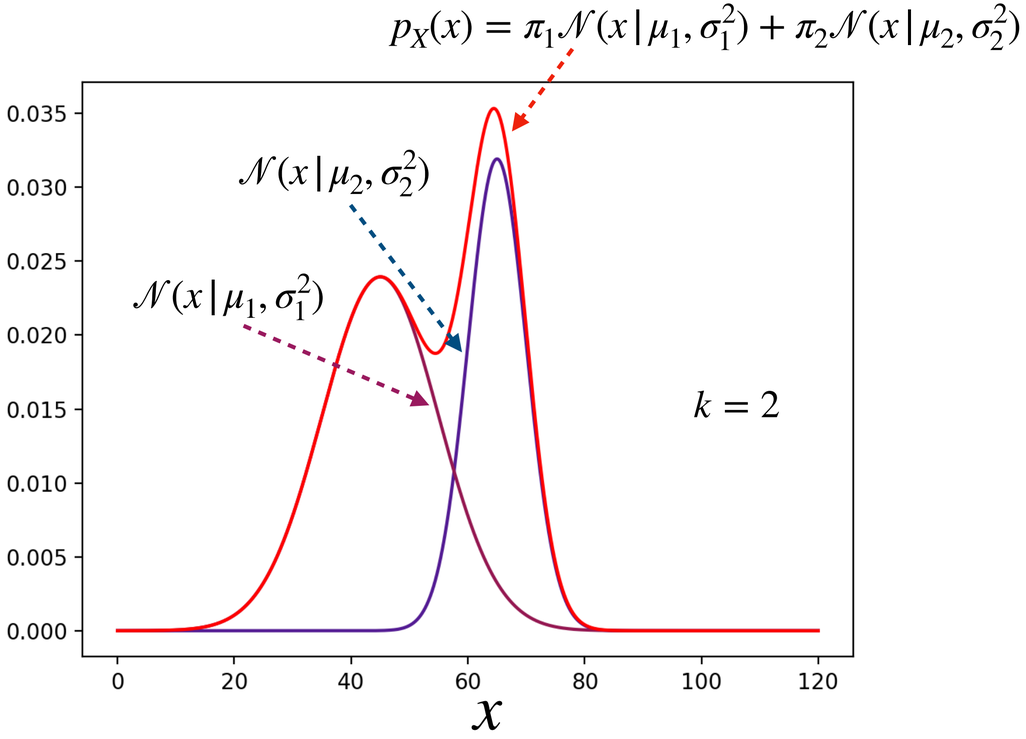
Note: $\pi_i \geq 0, \sum_i \pi_i = 1$ (why?)



gaussian mixture models

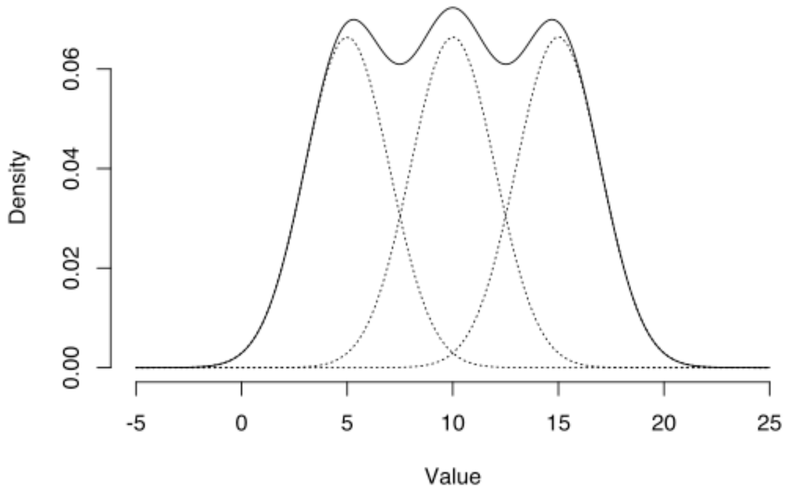
- Using GMMs for clustering:
  - Given N data points $x_1, \ldots, x_N$, how do we determine what the parameters of the k Gaussians are that best fit the data?
  - The parameters are the π_i, μ_i, σ_i
- Intuition:
  - Move the Gaussians around until their sum best fits the red curve (i.e., the dataset)

$$p_X(x) = \sum_{i=1}^{k} \pi_i \mathcal{N}(x \mid \mu_i, \sigma_i^2)$$



expectation maximization

- Like with KMeans, we will use an iterative approach to fit the Gaussian parameters
- **Expectation maximization** is an iterative approach to finding the parameters of a statistical model, where the model depends on unobserved, **latent variables**
  - Here, our latent variables are cluster labels (i.e., which of the *k* Gaussians each point belongs to)
- Start with a random guess for the Gaussian parameters
- Compute **expectation** (E-step)
  - Given the current parameters, what is the likelihood that each point comes from a particular Gaussian?
- Perform **maximization** (M-step)
  - Given these new likelihoods (which are essentially weights), update the means, (co)variances, and weights of the Gaussians using weighted averages



E step

- For each data point x_j, compute the likelihood that the data point comes from Gaussian *i*'s random variable G_i:

$$\gamma_{ij} = P(G_i \mid x_j) = \frac{P(x_j \mid G_i)P(G_i)}{P(x_j)}$$

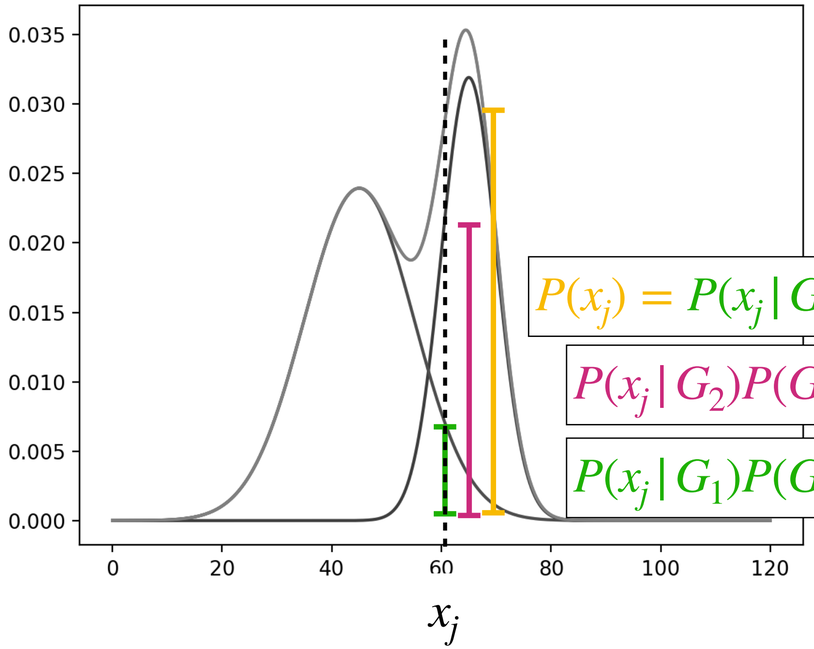
Bayes' Theorem

  - $P(x_j \mid G_i)$ is the (conditional) probability of observing x_j from G_i
  - $P(G_i)$ is the (unconditional) probability of observing the Gaussian G_i
  - $P(x_j)$ is the (unconditional) probability of observing x_j (from any Gaussian)

E step

- For each data point x_j, compute the likelihood that the data point comes from Gaussian *i*'s random variable G_i:

$$\gamma_{ij} = P(G_i \mid x_j) = \frac{P(x_j \mid G_i)P(G_i)}{P(x_j)}$$



Compute straight from Gaussian

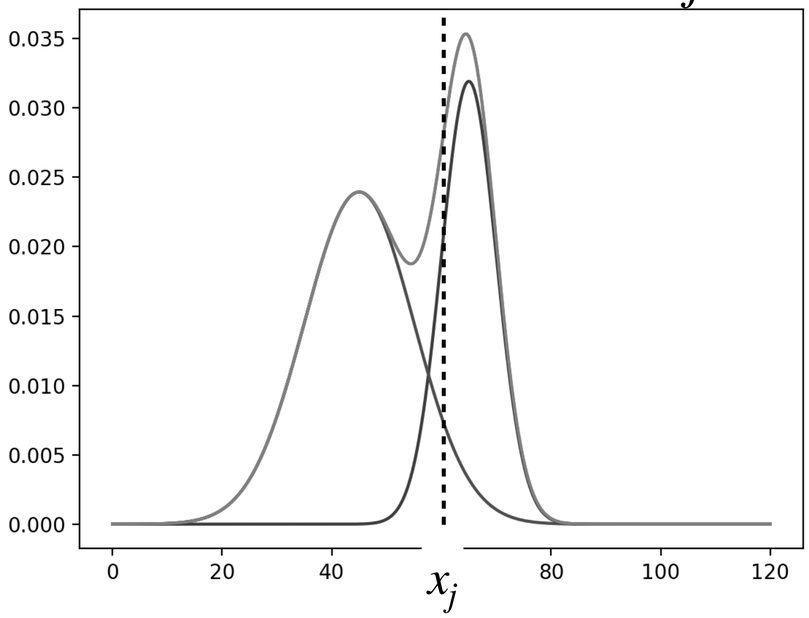
Weight of this Gaussian

Estimate from overall distribution (normalizes so probabilities sum to 1)

E step

- For each data point x_j, compute the likelihood that the data point comes from Gaussian *i*'s random variable G_i:

$$\gamma_{ij} = P(G_i \mid x_j) = \frac{P(x_j \mid G_i)P(G_i)}{P(x_j)}$$



$$= \frac{\pi_i \mathcal{N}(x_j \mid \mu_i, \sigma_i^2)}{\sum_{g=1}^{k} \pi_g \mathcal{N}(x_j \mid \mu_g, \sigma_g^2)}$$

M step

- Now that we have the likelihoods for each datapoint (how likely each is to come from each Gaussian), we *re-estimate* the parameters of each Gaussian i using those weights:

$$N_i = \sum_{j=1}^{N} \gamma_{ij} \qquad \pi_i' = \frac{N_i}{N} \qquad \mu_i' = \frac{\sum_{j=1}^{N} \gamma_{ij} x_j}{N_i} \qquad \sigma_i'^2 = \frac{\sum_{j=1}^{N} \gamma_{ij} (x_j - \mu_i')^2}{N_i}$$

- These expressions are the **maximum likelihood estimators** for Gaussian distributions
  - Derived by setting the derivative of $\log \prod_j p_X(x_j)$ to 0 for each parameter

M step

- Now that we have the likelihoods for each datapoint (how likely each is to come from each Gaussian), we *re-estimate* the parameters of each Gaussian i using those weights:

$$N_i = \sum_{j=1}^{N} \gamma_{ij}$$

Total likelihood of points in this Gaussian

$$\pi_i' = \frac{N_i}{N}$$

Proportion of points that come from this Gaussian

$$\mu_i' = \frac{\sum_{j=1}^{N} \gamma_{ij} x_j}{N_i}$$

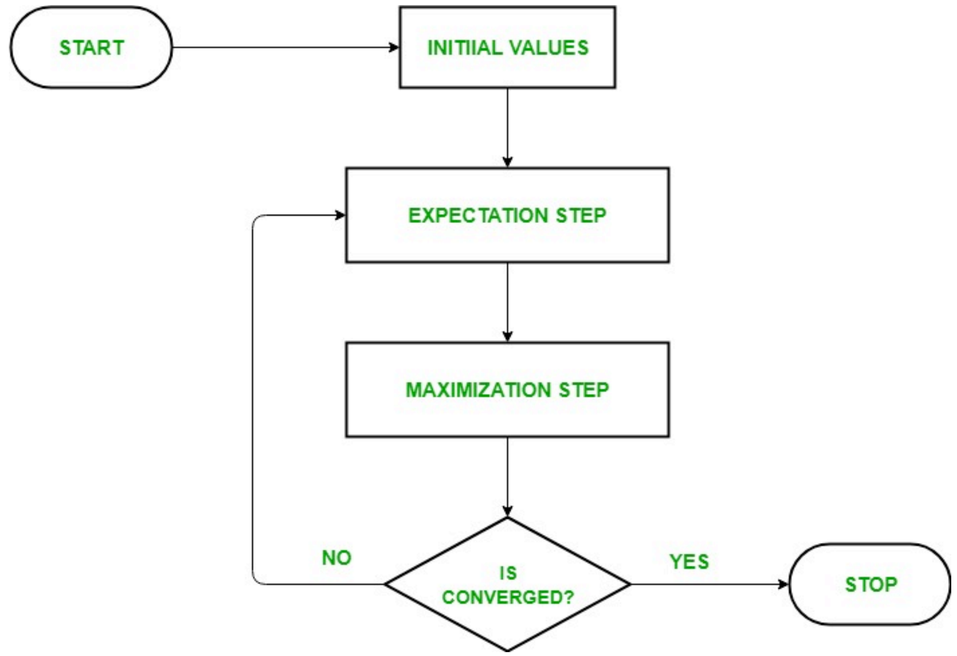
Weighted mean of this Gaussian

$$\sigma_i'^2 = \frac{\sum_{j=1}^{N} \gamma_{ij} (x_j - \mu_i')^2}{N_i}$$

Weighted variance of this Gaussian. Note that this uses the *updated* mean!

learning GMMs

- Repeat E and M steps until convergence
- Note that *what* you converge to can be sensitive to the initial estimates (like KMeans)
- When you are done, you have multiple Gaussians defined that “fit” the data you have
- This is a useful starting point for building Naïve Bayes classifiers!
  - We will discuss this later
- In Python: `GaussianMixture` class from `sklearn.mixture` (https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html)



determining convergence

- We continue alternating between E and M steps, but how do we know when the algorithm has converged?
  - We will use the log likelihood of the data given parameters:

$$\texttt{logl} = \log \prod_{j=1}^{N} p_X(x_j) = \sum_{j=1}^{N} \log p_X(x_j) = \sum_{j=1}^{N} \log \sum_{i=1}^{k} \pi_i \mathcal{N}(x_j \,|\, \mu_i, \sigma_i^2)$$

  - When the log-likelihood stops changing significantly, we can stop EM
  - Formally, we can stop once the change in `logl` is below a certain tolerance `tol`, e.g., `tol = 1`

numerical example

Consider a (very, very small) dataset of five points $x_1 = 12.14$, $x_2 = 4.55$, $x_3 = 2.57$, $x_4 = 12.19$, $x_5 = 12.78$. Find the GMM for this dataset with $k = 2$ clusters.

Assume an initialization of $\mu_1 = 2.57$, $\mu_2 = 7.68$ (i.e., chosen randomly), $\sigma_1 = 1$, $\sigma_2 = 1$, $\pi_1 = 0.5$, and $\pi_2 = 0.5$. Also assume `tol = 1`.

iteration 1

For the **E-step**, we need to find $\gamma = [\gamma_{ij}]$. We have $i = 1,2$ for clusters and $j = 1,...,5$ for datapoints.

To work out $\gamma_{ij} = \dfrac{\pi_i \mathcal{N}(x_j \mid \mu_i, \sigma_i^2)}{\sum_{g=1}^{k} \pi_g \mathcal{N}(x_j \mid \mu_g, \sigma_g^2)}$, we need to determine the unnormalized likelihoods $\tilde{\gamma}_{ij} = \pi_i \mathcal{N}(x_j \mid \mu_i, \sigma_i^2)$ for each i, j. We have:

$$\tilde{\gamma}_{1,1} = \pi_1 \mathcal{N}(x_1 \mid \mu_1, \sigma_1^2) = 0.5\mathcal{N}(12.14 \mid 2.57,1) = 2.487e-21 \qquad \tilde{\gamma}_{2,1} = \pi_2 \mathcal{N}(x_1 \mid \mu_2, \sigma_2^2) = 0.5\mathcal{N}(12.14 \mid 7.68,1) = 9.31e-06$$

$$\tilde{\gamma}_{1,2} = \pi_1 \mathcal{N}(x_2 \mid \mu_1, \sigma_1^2) = 0.5\mathcal{N}(4.55 \mid 2.57,1) = 2.77e-02 \qquad \tilde{\gamma}_{2,2} = \pi_2 \mathcal{N}(x_2 \mid \mu_2, \sigma_2^2) = 0.5\mathcal{N}(4.55 \mid 7.68,1) = 1.53e-03$$

$$\tilde{\gamma}_{1,3} = \pi_1 \mathcal{N}(x_3 \mid \mu_1, \sigma_1^2) = 0.5\mathcal{N}(2.57 \mid 2.57,1) = 1.99e-01 \qquad \tilde{\gamma}_{2,3} = \pi_2 \mathcal{N}(x_3 \mid \mu_2, \sigma_2^2) = 0.5\mathcal{N}(2.57 \mid 7.68,1) = 4.30e-07$$

$$\tilde{\gamma}_{1,4} = \pi_1 \mathcal{N}(x_4 \mid \mu_1, \sigma_1^2) = 0.5\mathcal{N}(12.19 \mid 2.57,1) = 1.51e-21 \qquad \tilde{\gamma}_{2,4} = \pi_2 \mathcal{N}(x_4 \mid \mu_2, \sigma_2^2) = 0.5\mathcal{N}(12.19 \mid 7.68,1) = 7.37e-06$$

$$\tilde{\gamma}_{1,5} = \pi_1 \mathcal{N}(x_5 \mid \mu_1, \sigma_1^2) = 0.5\mathcal{N}(12.78 \mid 2.57,1) = 4.32e-24 \qquad \tilde{\gamma}_{2,5} = \pi_2 \mathcal{N}(x_4 \mid \mu_2, \sigma_2^2) = 0.5\mathcal{N}(12.78 \mid 7.68,1) = 4.30e-07$$

We can then calculate $\gamma_{ij} = \dfrac{\tilde{\gamma}_{ij}}{\tilde{\gamma}_{1j} + \tilde{\gamma}_{2j}}$ for each i, j. For example, with $j = 2$, we have $\gamma_{:,2} = \left[\dfrac{\tilde{\gamma}_{1,2}}{\tilde{\gamma}_{1,2} + \tilde{\gamma}_{2,2}}, \dfrac{\tilde{\gamma}_{2,2}}{\tilde{\gamma}_{1,2} + \tilde{\gamma}_{2,2}}\right] = [0.95,0.05]$.

Continuing with this for each datapoint j, we arrive at:

$$\gamma = [\gamma_{ij}] = [[0,\ 0.95,\ 1,\ 0,\ 0],\ [1,\ 0.05,\ 0,\ 1,\ 1]]$$

iteration 1

For the **M-step**, we apply the formulas:

$$N_i = \sum_{j=1}^{N} \gamma_{ij} \qquad \pi_i' = \frac{N_i}{N} \qquad \mu_i' = \frac{\sum_{j=1}^{N} \gamma_{ij} x_j}{N_i} \qquad \sigma_i'^2 = \frac{\sum_{j=1}^{N} \gamma_{ij}(x_j - \mu_i')^2}{N_i}$$

We start with the calculations for N_1, N_2:

$$N_1 = \gamma_{1,1} + \gamma_{1,2} + \gamma_{1,3} + \gamma_{1,4} + \gamma_{1,5} = 0 + 0.95 + 1 + 0 + 0 = 1.95$$
$$N_2 = \gamma_{2,1} + \gamma_{2,2} + \gamma_{2,3} + \gamma_{2,4} + \gamma_{2,5} = 1 + 0.05 + 0 + 1 + 1 = 3.05$$

With this, we can find the new weights π_1, π_2 ($N = \sum_j N_j$ is also just the number of datapoints):

$$\pi_1 = N_1/(N_1 + N_2) = 0.39, \quad \pi_2 = N_2/(N_1 + N_2) = 0.61$$

The new means μ_1, μ_2 are calculated as:

$$\mu_1 = \frac{\sum_{j=1}^{N} \gamma_{1,j} x_j}{N_1} = \frac{0.95 \cdot 4.55 + 1.0 \cdot 2.57}{1.95} = 3.53, \quad \mu_2 = \frac{\sum_{j=1}^{N} \gamma_{2,j} x_j}{N_2} = \frac{1.0 \cdot 12.14 + 0.05 \cdot 4.55 + 1.0 \cdot 12.19 + 1.0 \cdot 12.78}{3.05} = 12.24$$

The new variances σ_1^2, σ_2^2 are calculated as:

$$\sigma_1^2 = \frac{\sum_{j=1}^{N} \gamma_{1j}(x_j - \mu_1')^2}{N_1} = \frac{0.95 \cdot (4.55 - 3.53)^2 + 1.0 \cdot (2.57 - 3.53)^2}{1.95} = 0.987,$$

$$\sigma_2^2 = \frac{\sum_{j=1}^{N} \gamma_{2,j}(x_j - \mu_2)^2}{N_2} = \frac{1.0 \cdot (12.14 - 12.24)^2 + 0.05 \cdot (4.55 - 12.24)^2 + 1.0 \cdot (12.19 - 12.24)^2 + 1.0 \cdot (12.78 - 12.24)^2}{3.05} = 1.12$$

log-likelihood after Iteration 1: $\texttt{logl} = -9.251$

iteration 2

Subsequent iterations are carried out the same way. Here are the results for iteration 2:

E-step

$\tilde{\gamma}$: [[7.82374842e-18, 9.22866077e-02, 9.73968107e-02, 4.95946483e-18, 2.34230944e-20]

[2.29617733e-01, 7.42592323e-13, 1.43390620e-19, 2.30381521e-01, 2.01831222e-01]]

γ: [[3.40729276e-17, 1.00000000e+00, 1.00000000e+00, 2.15271815e-17, 1.16052879e-19]

[1.00000000e+00, 8.04658813e-12, 1.47223116e-18, 1.00000000e+00, 1.00000000e+00]]

M-step

π: [0.4 0.6]

μ: [3.56112932 12.37288354]

σ^2: [0.98756428 0.08484898]

log-likelihood: -6.747

iteration 3

E-step

$\tilde{\gamma}$: [[1.73834797e-017, 1.00000000e+000, 1.00000000e+000, 9.73022463e-018, 1.04859007e-019]

[5.99253899e-001, 3.10915235e-157, 7.08580229e-247, 6.79831739e-001, 3.03920512e-001]]

γ: [[1.73834797e-017, 1.00000000e+000, 1.00000000e+000, 9.73022463e-018, 1.04859007e-019]

[1.00000000e+000, 3.19228660e-156, 7.27526641e-246, 1.00000000e+000, 1.00000000e+000]]

M-step

π: [0.4 0.6]

μ: [3.56112932 12.37288354]

σ^2: [0.98756428 0.08484898]

log-likelihood: -6.747

final result

After three iterations, we have the final model, which has these parameters:

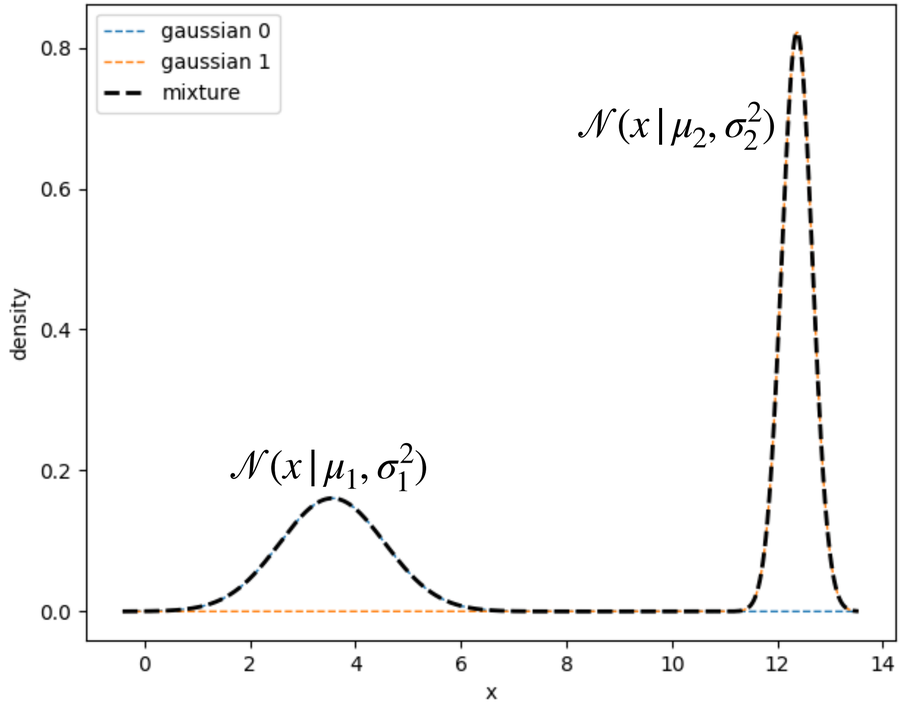
$$\gamma \approx [[0,1,1,0,0],[1,0,0,1,1]]$$

$$\pi \approx [0.4,0.6]$$

$$\mu \approx [3.56,12.37]$$

$$\sigma^2 \approx [0.99,0.08]$$

$$\texttt{logl} \approx -6.75$$



And the model is:

$$p_X(x) = 0.4\mathcal{N}(x\,|\,3.56,\ 0.99) + 0.6\mathcal{N}(x\,|\,12.37,\ 0.08)$$

The result is plotted in the graph on the right.