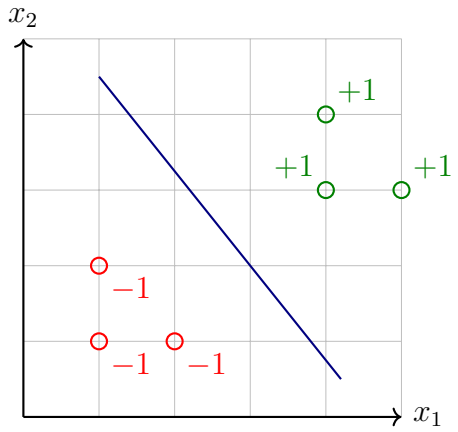
Lecture 5: Unsupervised learning

CS 3780/5780, Sp25

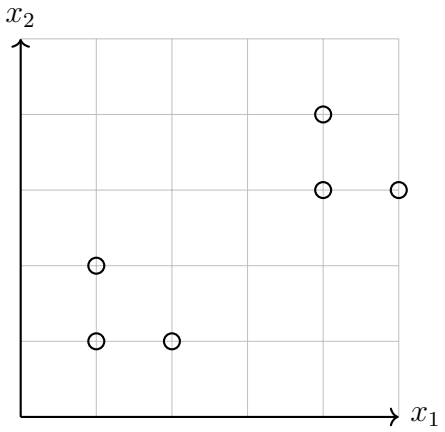
Tushaar Gangavarapu (TG352@cornell.edu)

In this lecture (and the next), we will look at a different regime than supervised learning—*unsupervised* learning. We will make this more concrete momentarily, in essence, we are going to tackle the fundamental problem of "what do we do when we don't have labels?" One of the traditional examples of such unsupervised learning is the cocktail party problem, where the individual speech signals are separated from a recording of people talking simultaneously in a room.

Let's make the setup for unsupervised learning more concrete: given an unlabeled dataset (no $y^{(j)}$s), $\mathcal{D} = \{x^{(j)} | 1 \leq j \leq n\}$, we want to uncover the (latent) structure in the data (if it exists).



Supervised setting
(*includes* class labels)



Unsupervised setting
(*no* class labels)

So, what can we expect in this no-label regime? In case of supervised learning, we can argue why a particular separating line (or, hyperplane) is the "right" one and what guarantees it gives us on the algorithmic convergence (e.g., Perceptron convergence depends on the margin width, γ). The lack of labels makes it difficult for us to make similar guarantees or statements on what it means to find the right structure. Consequently, we are going to:

(a) allow for stronger assumptions—in supervised setting, the assumption was that the classes are separable; in unsupervised setting, we are going to assume that there exists some underlying latent structure in the data (e.g., assume our data is generated a certain way),

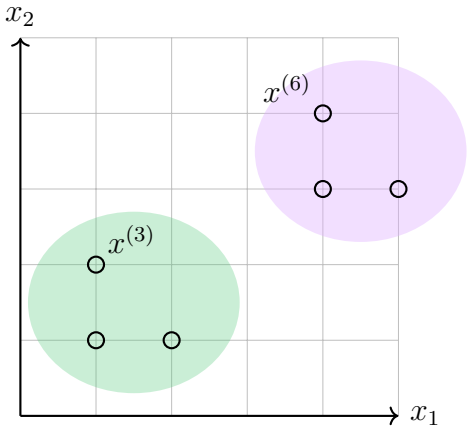
(b) accept weaker guarantees—in Perceptron, we assumed that a $w^\star$ exists and realized how fast we can reach $w^\star$, starting from some w; we cannot make such statements in the unsupervised setting (e.g., how many clusters is the correct number of clusters?).

Aside. While we noted the ends of the learning spectrum with supervised on one end and unsupervised on the other, there are other, recently-popularized flavors of "less" supervised learning which include: (a) "weak" supervision, where noisy or lower-quality labels are used to create large training datasets for training commercial models; (b) self-supervised learning, a.k.a., the large language model regime, where we train on a simpler task (e.g., predicting the next word), which can then be used for other downstream tasks (e.g., sentiment classification).

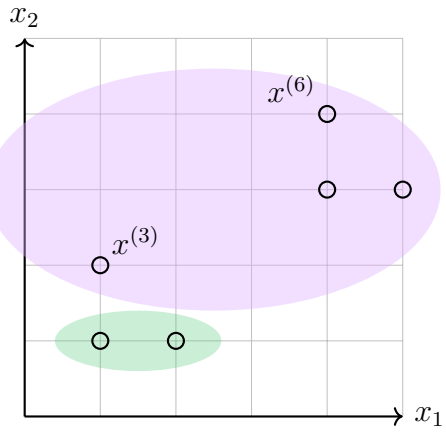
While we will not cover these regimes in this class extensively, it is still interesting to realize the ties of these now-widely-used approaches to unsupervised learning.

1 K-means clustering

Given the number of clusters, k,[1] our goal is to find a good clustering if the data points. Now, a natural question is to ask what it means for clusters to be "good"?



Two "good" clusters ($k = 2$)



Two clusters ($k = 2$)

(As the name hints,) in k-means, we are going to define "good" clustering based on the distance of the cluster mean (or, centroid) to the points in the cluster. This goes back to our notion of "similar points points are labeled similarly" in k-nearest neighbors. In the example above, on the right, $x^{(3)}$ is closer to the centroid of the green cluster, than it is to the blue cluster.

1.1 The clustering algorithm

Let us formalize this: Given n data points, $x^{(1)}, \dots, x^{(n)}$, with $x^{(j)} \in \mathbb{R}^d$ and k, the number of clusters we are looking for, our goal is to find an assignment of points, $x^{(j)}$s, to k clusters, in a way that minimizes the distance of each point to the center of its cluster. We will denote this assignment as $\mathcal{C}^{(j)} = i$, indicating that point $x^{(j)}$ belongs to cluster i, where $i = 1, \dots, k$. In the example above, on the left, we have $x^{(3)}$ in cluster-1 (green) and $x^{(6)}$ in cluster-2 (blue); or, equivalently, $\mathcal{C}^{(3)} = 1$ and $\mathcal{C}^{(6)} = 2$. In some sense, we are performing *hard* assignment of points to clusters, i.e., a point can belong to a single cluster.

So, how do we find the clusters? A natural question to ask here is if there is a polynomial-time algorithm to assign n points to k clusters, and as it turns out, this is an NP-hard problem.[2] Let's instead take an iterative approach to facilitate cluster assignments:

(a) Randomly assign k cluster centers, $\mu^{(i)}$s.

(b) Next, (re)assign each point, $x^{(j)}$ to a cluster based on the distance of the point to the cluster center. When using ℓ_2 (or, Euclidean) distance,[3] this is mathematically equivalent

[1] We will come back to the modeling decision of selecting k later; but, note that one can intuitively choose the number of clusters a priori by simply looking at the data (e.g., in a class setting, one can use $k = 2$ to cluster 3780 and 5780 students).

[2] https://link.springer.com/content/pdf/10.1007/s10994-009-5103-0.pdf.

to

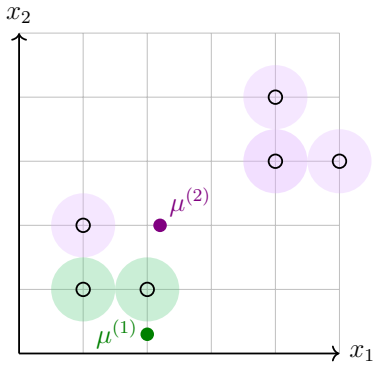
$$\mathcal{C}^{(j)} = \arg\min_{i=1,\ldots,k} \|x^{(j)} - \mu^{(i)}\|^2.$$

(c) Compute new cluster centers based on the points assigned to each cluster as

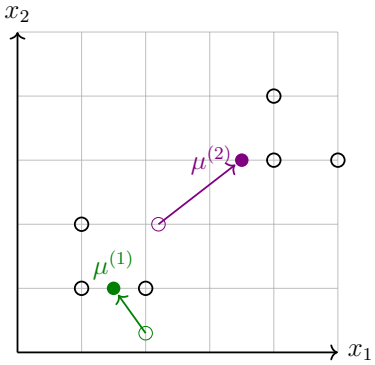
$$\mu^{(i)} = \frac{\sum_{j=1}^{n} \mathbf{1}\{\mathcal{C}^{(j)} = i\}\, x^{(j)}}{\sum_{j=1}^{n} \mathbf{1}\{\mathcal{C}^{(j)} = i\}},$$

where $\mathbf{1}\{\cdot\}$ is the indicator function, $\mathbf{1}\{\mathcal{C}^{(j)} = i\} = 1$ if point $x^{(j)}$ belongs to cluster-i, 0 otherwise.

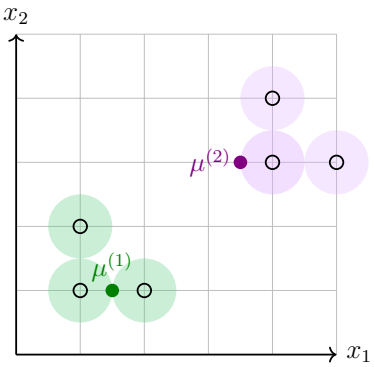
We will repeat steps (b) and (c) until the cluster assignments do not change, at which point, we note that the k-means algorithm has converged.



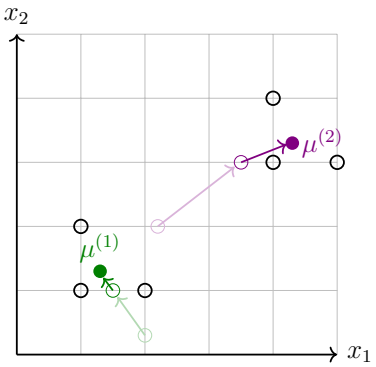
Randomly initialize centers and assign points to clusters (a, b)



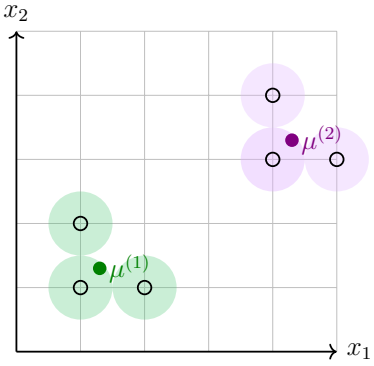
Recompute the centers (c)



Reassign points to clusters (b)



Recompute the centers (c)



Reassign points to clusters (b) (no change in assignments)

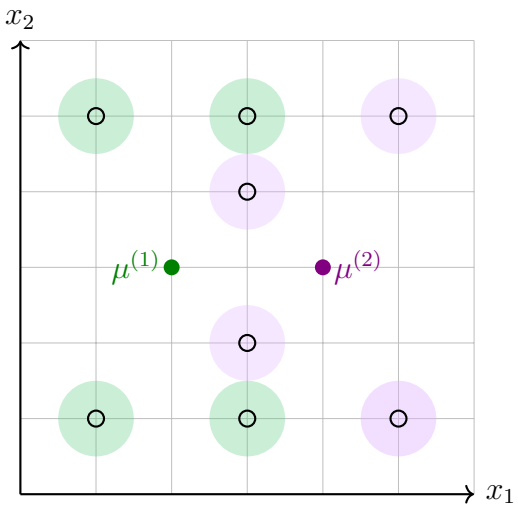
1.2 Properties of K-means

While such an iterative approach makes sense intuitively, it is not obvious that the algorithm always converges, i.e., is there some setting in which the cluster means simply oscillate? To understand this, let us look at the cost function that measures the distance of each point to its centroid:

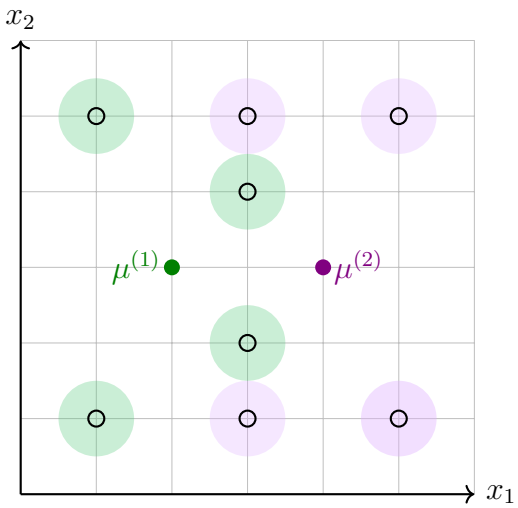
$$J(\mathcal{C}, \mu) = \sum_{j=1}^{n} \|x^{(j)} - \mu^{(\mathcal{C}^{(j)})}\|^2.$$

[3] We saw in "P0: Getting started" that $\|x^{(1)} - x^{(2)}\|_2$ computes the Euclidean distance between $x^{(1)}$ and $x^{(2)}$.

Observe that the k-means algorithm minimizes J with respect to $\mathcal{C}$ by fixing μ (assign points to clusters in step (b)), and then minimizes J with respect to μ by keeping $\mathcal{C}$ fixed (recompute new centers in step (c)). Hence, J must decrease monotonically (either decreases or remains the same), and the value of J must converge. While unlikely in practice, it is possible for k-means to oscillate between two different clusterings—different values of $\mathcal{C}$ and/or μ result in the same J. For instance, in the following setup, J is always the same and k-means oscillates between two assignments, while the centroids remain the same.

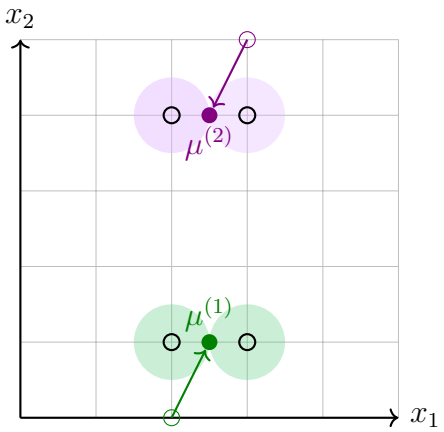


For the cluster assignments, compute centroids (step (c))

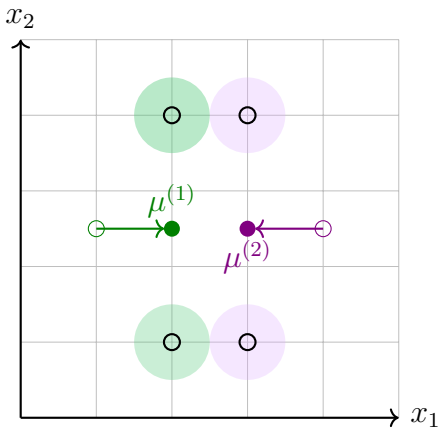


For the computed centroids, assign clusters (step (b))

The next question to ask is if we can find a global minima of J?—no! As a simple exercise, we show below that different starting centroids results in different clusterings. As noted earlier, the problem of k-means is an NP-hard problem, thus using an exact algorithm to calculate the centroids doesn't run in polynomial time. (Recall that we said we were going to make stronger assumptions and accept weaker guarantees in unsupervised settings.)



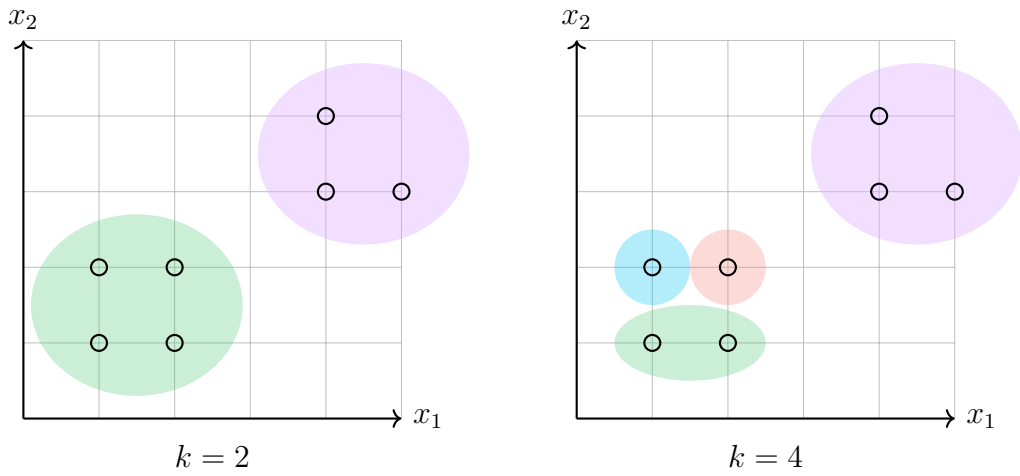
Optimal clustering



Sub-optimal clustering

Initializing cluster centroids. In practice, we randomly choose k training samples and assign their values as the initial cluster centers (in step (a)). While this strategy works fine, if you are worried about getting stuck in local minima, one common strategy is to run the k-means algorithm with different initial centroids, $\mu^{(i)}$s and choose the clusterings that achieve lowest J.

1.3 On choosing the number of clusters



There is no right way to choose the value of k; domain knowledge and the application-specific requirements often help. For instance, how many clusters to form in this lecture hall? Split by course code—3780 vs. 5780? Or, split by seating columns?

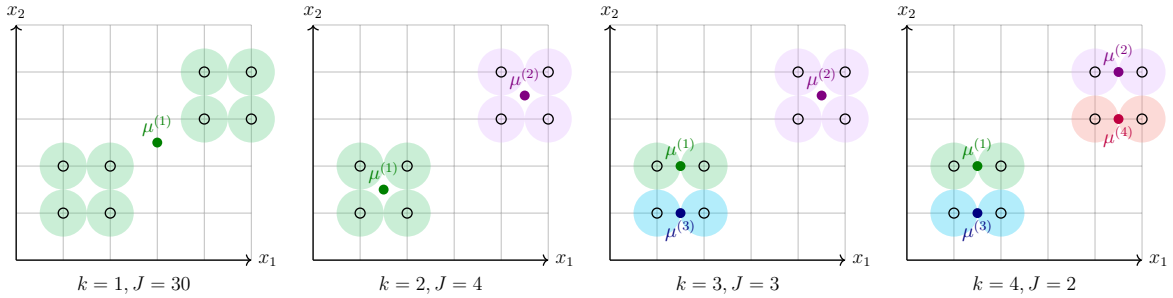
That said, one strategy we can adapt to potentially avoid over-splitting clusters is the elbow method, which is as follows:

(a) Run k-means clustering for different values of k.

(b) For each k, upon convergence, compute the cost estimate, $J(\mathcal{C}, \mu)$ (we will have to average the values of J over multiple runs, to account for the randomness in cluster initialization).

(c) Choose the "elbow" point as the point beyond which the decrease in J is minimal.

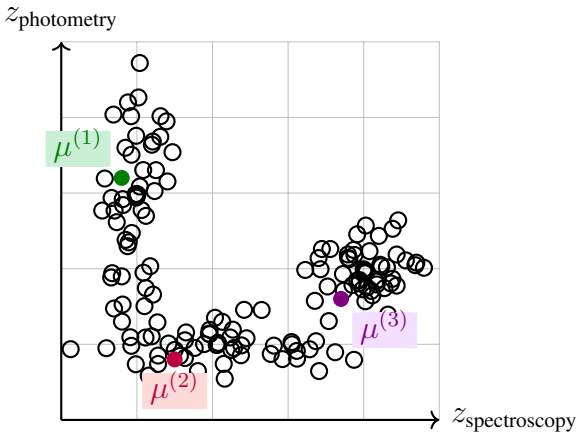
Note: Increasing k will *always* decrease J (why?—think: what happens if each point is its own cluster?), but the gains are bigger until we realize the "true" k. In the example below, we observe the biggest drop from $k = 1$ to $k = 2$, beyond which the drop is insignificant; hence, we can choose $k = 2$.



2 Mixture of Gaussians

As a motivating example, let us consider the following measurements of spectral energy distributions from three different light sources (say, stars, galaxies, and quasars):[4]

[4] Inspired from (Miller et al., 2015).



Cluster means from k-means

Are the clusterings in the above depiction "good"?

Recall that in k-means, we made no assumptions about how the data was generated, and we made "hard" assignments. In this section, we will see, what feels like, a generalization of the k-means. The goal is the same as before: assign each data point (a reading) to a cluster (a light source), but we are going to make "soft" assignments. Instead of saying $\mathcal{C}^{(j)} = i$ (or, $x^{(j)}$ belong to cluster-i), we are going to model $P(Z^{(j)} = i|x^{(j)})$, which notes the *posterior probability* that $x^{(j)}$ belongs to cluster-i, given that we observed $x^{(j)}$.

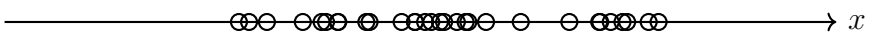
You can think of Z as a multinomial random variable the indicates the outcome of rolling a k-faced die.

2.1 Assumptions

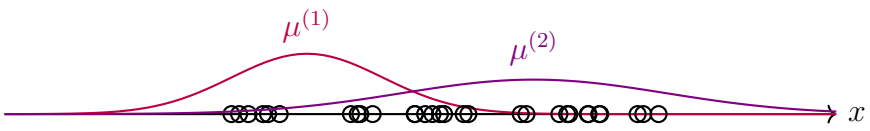
We are going to assume that there are many clusters (or, light sources), and we know how many sources there are, i.e., we know k. Additionally, we are going to assume that the data generated by each light source is well modeled as a Gaussian—in 1D, this is characterized by μ and σ^2—and that each light source has different intensities (or, μ and σ^2).[5] Note that we are not assuming any sampling frequency, i.e., all light sources generate the same number of photons. This is often referred to as the "unknown" mixture (of Gaussians).

In summary, we assume that the data we observed is a result of k light sources with different, unknown μ and σ^2, and we do not know how often they are sampled.

2.2 The mixture of Gaussians model



What we observe (no knowledge of the underlying model)



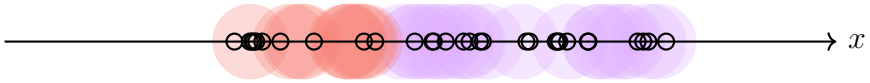
Two Gaussians generating the observed data (the "god" view)

[5] This Gaussian assumption can still work as an approximation even when the underlying distribution is not Gaussian, since a mixture of Gaussians can approximate many continuous distributions arbitrarily well, given enough components.

For simplicity, let us restrict our discussion to the 1D setting. Now, let us assume that there are two Gaussians (two light sources) generating data, with means $\mu^{(1)}$ and $\mu^{(2)}$. Further, assume we sample from $\mu^{(1)}$-Gaussian with probability p and $\mu^{(2)}$-Gaussian with probability $1-p$.

In the above example, we have $\mu^{(1)} = 4, \mu^{(2)} = 7$ and $p = 1/3$. Consequently, the prior (or, belief) $P(Z^{(j)} = 1) = 1/3$ and $P(Z^{(j)} = 2) = 2/3$.

Aside. What if we knew that specific data points came from specific sources? For example, consider the following assignment of points:



Then, all we need to do to estimate our two Gaussians is compute the means and variances of the points belonging to a specific cluster. Additionally, based on the number of samples in each cluster, we can estimate p. For convenience, let's collect all our parameters as $\Theta = \{\mu^{(1)}, \mu^{(2)}, \sigma^{(1)}, \sigma^{(2)}, p\}$. Now, we can estimate our posterior $P(Z^{(j)} = i|x^{(j)})$ as:[6,7]

$$P(Z^{(j)} = i|x^{(j)}; \Theta) = \frac{P(x^{(j)}|Z^{(j)} = i; \Theta)P(Z^{(j)} = i; \Theta)}{P(x^{(j)})}.$$

Note that by our assumption, $x^{(j)}|Z^{(j)} = i; \Theta \sim \mathcal{N}(\mu^{(i)}, {\sigma^{(i)}}^2)$, which can be computed for $x^{(j)}$ and the marginal $P(x^{(j)})$ can also be computed as:

$$\begin{aligned} P(x^{(j)}) &= \sum_{\ell=1}^{k} P(x^{(j)}|Z^{(j)} = \ell; \Theta)P(Z^{(j)} = \ell; \Theta) \\ &= P(x^{(j)}|Z^{(j)} = 1; \Theta)P(Z^{(j)} = 1; \Theta) + P(x^{(j)}|Z^{(j)} = 2; \Theta)P(Z^{(j)} = 2; \Theta) \\ &= \frac{1}{3}\underbrace{P(x^{(j)}|Z^{(j)} = 1; \Theta)}_{\mathcal{N}(\mu^{(1)}, {\sigma^{(1)}}^2)} + \frac{2}{3}\underbrace{P(x^{(j)}|Z^{(j)} = 2; \Theta)}_{\mathcal{N}(\mu^{(2)}, {\sigma^{(2)}}^2)} \end{aligned}$$

All this to say that if we knew which points came from which light source, we can then make our soft assignments by estimating $P(Z^{(j)} = i|x^{(j)})$.

2.3 Formalization

Let us formalize the setup so far: Given n data points, $x^{(1)}, \ldots, x^{(n)}$, with $x^{(j)} \in \mathbb{R}^d$ and k, the number of clusters we are looking for, our goal is to estimate $P(Z^{(j)} = i|x^{(j)})$, the probability that $x^{(j)}$ belongs to cluster-i, given that we observed $x^{(j)}$.

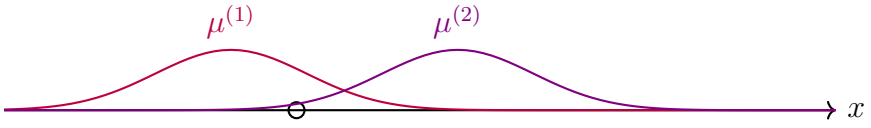
Now, according the mixture of Gaussians model, we have: $Z^{(j)} \sim \text{Multinomial}(\Phi)$ (which simplifies to $\text{Bernoulli}(p)$ for $k = 2$) that indicates the outcome of picking a cluster, and $P(Z^{(j)} = i)$ indicates the (prior) *probability* of picking a cluster-i. From the Gaussian assumption, we have $x^{(j)}|Z^{(j)} = i \sim \mathcal{N}(\mu, \sigma^2)$. One final note is that we don't observe $Z^{(j)}$ in our data, i.e., it is a "latent" variable, and the data we observe are the remnants of this data generation process.

[6] In the notation below, we clump all parameters into Θ for typesetting convenience, if we were being more precise, we would instead write: $P(x^{(j)}|Z^{(j)} = i; \mu^{(i)}, \sigma^{(i)})$ and $P(Z^{(j)} = i; p)$.

[7] To be strict in our notation, we use $P(x^{(j)}|Z^{(j)} = i; \Theta)$ to indicate the *probability density* (not probability) or likelihood of $x^{(j)}$ under Gaussian-i, which can be computed using the "$1/\sqrt{2\pi}\sigma^{(i)} \ldots$," which tells us how "dense" probability mass is around $x^{(j)}$, but not the probability of any single point.

The idea here is that of all the possible estimates for model parameters, $\mu^{(i)}$s, $\sigma^{(i)}$s, and $p^{(i)}$s in a way that maximizes the likelihood of the data we observed. We will formalize this notion of maximizing data likelihood into a framework that can used in various settings, in the upcoming weeks. For now, think of this as reverse engineering the data generation process based on the data observed.

Example. To solidify our understanding of what's going on, let us consider a simple example. (You're gonna have the "god" view running in the back of your mind.) Consider $p^{(1)} = 0.6$ (meaning, $p^{(2)} = 1 - p^{(1)} = 0.4$), $\mu^{(1)} = 3$ and $\mu^{(2)} = 6$, ${\sigma^{(1)}}^2 = {\sigma^{(2)}}^2 = 1$.



(a) Pick a Gaussian with $P(Z^{(j)} = 1) = p^{(1)} = 0.6$
(b) Sample a point from the chosen Gaussian: $x^{(j)}|Z^{(j)} = i \sim \mathcal{N}(\mu^{(i)}, {\sigma^{(i)}}^2)$
(c) Repeat steps (a), (b)

2.4 The algorithm

While we outline the algorithm, we want you have k-means to be in the back of your mind. Precisely, the steps of k-means are: (a) guess the centroids, (b) make cluster assignments based on the centroids, and (c) recompute the centroids based on the clusters (and repeat (b), (c)).

The clustering approach using the mixture of Gaussians is as follows:

(a) Randomly initialize the values of Θ, i.e., $\mu^{(1)}, \mu^{(2)}, \sigma^{(1)}, \sigma^{(2)}, p$ in our case.

(b) Compute the posterior, $P(Z^{(j)} = i|x^{(j)}; \Theta)$, indicating the probability of $x^{(j)}$ having generated by Gaussian-i, given that we observed the point $x^{(j)}$.

We can compute this posterior using the Bayes flip:

$$P(Z^{(j)} = i|x^{(j)}; \Theta) = \frac{\overbrace{P(x^{(j)}|Z^{(j)} = i; \Theta)}^{=\exp(-(x^{(j)}-\mu^{(i)})/2{\sigma^{(i)}}^2)/\sqrt{2\pi}\sigma^{(i)}} \quad \overbrace{P(Z^{(j)} = i; \Theta)}^{=p \text{ or } 1-p}}{P(x^{(j)})}$$
$$= \frac{P(x^{(j)}|Z^{(j)} = i; \Theta)P(Z^{(j)} = i; \Theta)}{p \times P(x^{(j)}|Z^{(j)} = 1; \Theta) + (1-p) \times P(x^{(j)}|Z^{(j)} = 2; \Theta)}.$$

Once computed, we have "soft" assignment of points to clusters.

(c) Update the parameters, Θ, based on the (soft) cluster assignments. (Recall that this is exactly the same setting as our previous exercise where we assumed that we knew which samples came from which Gaussian.)

$$p^{(1)} = p = \frac{1}{n}\sum_{j=1}^{n} P(Z^{(j)} = 1|x^{(j)}); \quad p^{(2)} = 1 - p,$$

$$\mu^{(1)} = \frac{\sum_{j=1}^{n} P(Z^{(j)} = 1|x^{(j)})x^{(j)}}{\sum_{j=1}^{n} P(Z^{(j)} = 1|x^{(j)})}; \quad \mu^{(2)} = \frac{\sum_{j=1}^{n} P(Z^{(j)} = 2|x^{(j)})x^{(j)}}{\sum_{j=1}^{n} P(Z^{(j)} = 2|x^{(j)})},$$

$${\sigma^{(1)}}^2 = \frac{\sum_{j=1}^{n} P(Z^{(j)} = 1|x^{(j)})(x^{(j)} - \mu^{(1)})^2}{\sum_{j=1}^{n} P(Z^{(j)} = 1|x^{(j)})}; \quad {\sigma^{(2)}}^2 = \frac{\sum_{j=1}^{n} P(Z^{(j)} = 2|x^{(j)})(x^{(j)} - \mu^{(2)})^2}{\sum_{j=1}^{n} P(Z^{(j)} = 2|x^{(j)})}.$$

We will repeat steps (b) and (c) until the cluster assignments do not change, at which point, we note convergence. Take a moment to realize how algorithm is similar to k-means, except we now have soft assignments via $P(Z^{(j)} = 1|x^{(j)})$ instead of $\mathbf{1}\{\mathcal{C}^{(j)} = i\}$. Furthermore, note that this follows a similar pattern of coordinate descent from k-means, meaning this clustering scheme is also susceptible to local optima.

Now, let's take this setting to the extreme: what happens if all our Gaussians had $\sigma^{(i)} \to 0$, i.e., our variances are infinitesimally small? Observe the behavior of

$$P(x^j|Z^{(j)} = i) = \frac{1}{\sqrt{2\pi}\sigma^{(i)}} \exp\left(-\frac{(x^{(j)} - \mu^{(i)})^2}{2{\sigma^{(i)}}^2}\right)$$

as $\sigma^{(i)} \to 0$. For $x^{(j)} = \mu^{(i)}$, i.e., point is the cluster center, we have $P(x^j|Z^{(j)} = i) \to \inf$. And, for points that are not clusters, we have $P(x^j|Z^{(j)} = i) \to 0$. This means, our Gaussian clustering model collapses to k-means clustering with hard assignments, where each point is assigned to the closest centroid. For this reason, we view mixture of Gaussians clustering as a generalization of the k-means.

Aside. If you're interested in learning how we came about the estimates in step-(c), see Appendix C; however, as noted earlier, we will cover the framework of estimating parameters from maximizing data likelihood in the upcoming weeks.

3 Conclusion

In conclusion, we looked at a different regime of unsupervised setting, where we have no labels associated with the data points. We then saw k-means, a simple heuristic algorithm that was easy to implement and fast for small k, but the algorithm inherently assumed all clusters are spherical. Furthermore, we noted that k-means only made hard cluster assignments. To facilitate soft assignments, we discussed the mixture of Gaussians clustering, where we assumed that the data was generated from, well, a mixture of Gaussians, and the model aimed at recovering the data generation parameters iteratively.

A Notation

$\mathcal{D}$	The training dataset of n samples
n	The number of training samples in the dataset $\mathcal{D}$
d	The number of feature dimensions
k	The number of clusters
$x^{(j)} \in \mathbb{R}^d$	The d-dimensional (feature) vector associated with the j-th training sample
$y^{(j)}$	The class label associated with the j-th training sample (not present in unsupervised setting case)
$x_\ell^{(j)} \in \mathbb{R}$	The ℓ-th element of $x^{(j)}$
$\mathcal{C}^{(j)} = i$	The data point $x^{(j)}$ is assigned to cluster-i
$\mu^{(i)}$	The center of the cluster-i or the mean of a set of points in cluster-i
$\sigma^{(i)}$	The standard deviation of the data points in cluster-i
$\mathbf{1}\{a = b\}$	Indicator function that outputs 1 when $a = b$, 0 otherwise
J	The cost function (often referred to as the distortion function)
$\mathcal{N}(\mu, \sigma)$	Normal distribution defined by mean, μ, and variance, σ^2
$P(a\|b; \alpha)$	The probability of a given b, *parameterized* by α. Note: α is a parameter of the model rather than a random variable
$X \sim \text{Bernoulli}(p)$	X is a Bernoulli random variable with parameter p. Think: X indicates the outcome of a coin toss, with $P(H) = p$
$X \sim \text{Multinomial}(\Phi)$	X is a Multinomial random variable with parameter Φ and $n = 1$—this is the generalization of the Bernoulli random variable. Think: X indicates the outcome of rolling a dice, with $P(\text{side-}i) = p^{(i)}$; $\Phi = \{p^{(1)}, \ldots, p^{(6)}\}$
Z	A random variable to indicate the outcome of rolling a k-faced die ($k = 2$: Bernoulli; Multinomial otherwise)
$P(Z^{(j)} = i)$	The probability of drawing a data point from Gaussian-i. This is more of a belief or prior, and is independent of the data. Think: God set this a priori
$P(Z^{(j)} = i\|x^{(j)})$	The probability that point $x^{(j)}$ was to generated from Gaussian-i, given that we observed $x^{(j)}$. Think of this as: we observed $x^{(j)}$, now, was it drawn from Gaussian-i?
$P(x^{(j)}\|Z^{(j)} = i)$	The probability of observing $x^{(j)}$ given that we are generating data from $Z^{(j)} = i$; in this lecture, we assume $x^{(j)}\|Z^{(j)} = i \sim \mathcal{N}(\mu^{(i)}, \sigma^{(i)})$
Θ	A set of model parameters; in case of $k = 2$, $\Theta = \{\mu^{(1)}, \mu^{(2)}, \sigma^{(1)}, \sigma^{(2)}, p\}$

B Probability fundamentals

Bayes rule (or, flip). For two events, A and B, we are often interested in the probability that *both* these event happen. For example, consider A to be "Did LeBron make his first shot?" and B to be "Did LeBron make his second shot?," and we are interested in the probability that LeBron makes his first shot *and* his second shot.

We are going to model this "and" clause as: in how many cases did LeBron make his first shot, and in how many *of those* did LeBron make the second shot. Mathematically, of the cases where A occurred, B would also occur in $B|A$ of them. Hence, we have

$$P(A, B) = P(A)P(B|A) = P(B)P(A|B).$$

From the above, we observe that

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)},$$

which is often referred to as the Bayes rule or the Bayes flip (since we flip the conditional from $A|B$ to $B|A$).

Marginal probability. Consider the simple case of two events, A: the point x was generated from Gaussian-1 and B: we observe point x in our data. Now, we wish to compute $P(B)$—we have two possibilities here: we observe x and it was generated from Gaussian-1, or we observe x and it was generated from Gaussian-2. This follows,

$$\begin{aligned} P(B) &= P(A, B) + P(A', B) \\ &= P(B|A)P(A) + P(B|A')P(A'), \end{aligned}$$

where A' is the complement of A.

C Maximum likelihood estimates

In §2.4 (step (c)), we noted that the parameters, $\Theta = \{p, \mu^{(1)}, \mu^{(2)}, \sigma^{(1)}, \sigma^{(2)}\}$, were estimated to maximize the likelihood of the data we observed. While we will make this more concrete in the upcoming lectures, we preset a proof here for completeness.

We write down the likelihood of our data as:

$$L(\Theta) = \prod_{j=1}^{n} P(x^{(j)}; \Theta),$$

and our goal is choose Θ such that $L(\Theta)$ is maximized. A simple observation here is that maximizing $L(\Theta)$ is the same as maximizing $\log(L(\Theta))$. This follows from $\log(x)$ being strictly increasing, meaning that whenever x increases, so does $\log(x)$—therefore, the value of x that reaches the highest point will also correspond to the highest value of $\log(x)$ on the graph; essentially, they both occur at the same point. Hence, we are going to maximize $\ell(\Theta) = \log(L(\Theta))$ instead:

$$\begin{aligned} \ell(\Theta) &= \log(L(\Theta)) \\ &= \log\left(\prod_{j=1}^{n} P(x^{(j)}; \Theta)\right) \\ &= \sum_{j=1}^{n} \log P(x^{(j)}; \Theta). \end{aligned}$$

Now, we can estimate the marginal probability $P(x^{(j)}$ similar to our estimates in §B:

$$\begin{aligned} \ell(\Theta) &= \sum_{j=1}^{n} \log P(x^{(j)}; \Theta) \\ &= \sum_{j=1}^{n} \log \sum_{i \in \{1,2\}} P(x^{(j)}|Z^{(j)} = i; \Theta) P(Z^{(j)} = i; \Theta). \end{aligned}$$

Now, if we were to set the derivates of the above equation to zero and try to solve for Θ, we would quickly notice that it isn't possible to estimate the values of Θ in closed form.

Now, if we assume that we "know" the cluster assignments beforehand, the above reduces to:

$$\begin{aligned}\ell(\Theta) &= \sum_{j=1}^{n} \log P(x^{(j)}; \Theta) \\ &= \sum_{j=1}^{n} \log(P(x^{(j)}|Z^{(j)} = i; \Theta)P(Z^{(j)} = i; \Theta)),\end{aligned}$$

where we no longer need to optimize over all cluster assignments, k; we can use our knowledge of assignments instead. Now, maximizing the above with respect to μ, σ, p gives you:

$$\begin{aligned}p^{(i)} &= \frac{1}{n}\sum_{j=1}^{n} \mathbf{1}\{Z^{(j)} = i\} \\ \mu^{(i)} &= \frac{\sum_{j=1}^{n} \mathbf{1}\{Z^{(j)} = i\}x^{(j)}}{\sum_{j=1}^{n} \mathbf{1}\{Z^{(j)} = i\}} \\ \sigma^{(i)} &= \frac{\sum_{j=1}^{n} \mathbf{1}\{Z^{(j)} = i\}(x^{(j)} - \mu^{(i)})^2}{\sum_{j=1}^{n} \mathbf{1}\{Z^{(j)} = i\}}.\end{aligned}$$

Since we don't have "hard" assignments, $\mathbf{1}\{Z^{(j)} = i\}$, noting which Gaussian the point was sampled from, we can replace them with our soft assignments, $P(Z^{(j)} = i|x^{(j)})$.

References

CS4/5780. Unsupervised Learning. URL https://www.cs.cornell.edu/courses/cs4780/2023sp/lectures/UnsupervisedLearning.html. Accessed: 02/03/2025.

A. Miller, A. Wu, J. Regier, J. McAuliffe, D. Lang, M. Prabhat, D. Schlegel, and R. P. Adams. A Gaussian process model of quasar spectral energy distributions. *Advances in Neural Information Processing Systems*, 28, 2015.

A. Ng. CS229 Lecture notes. *CS229 Lecture notes*, 1(1):145–151, 2000. URL https://cs229.stanford.edu/main_notes.pdf. Version: June 11, 2023.

(Last compiled: 3/10/2025, 2.24pm ET.)