CSCI 567: Machine Learning

Vatsal Sharan

Spring 2026

Lecture 12, Apr 17

USC University of Southern California

Administrivia

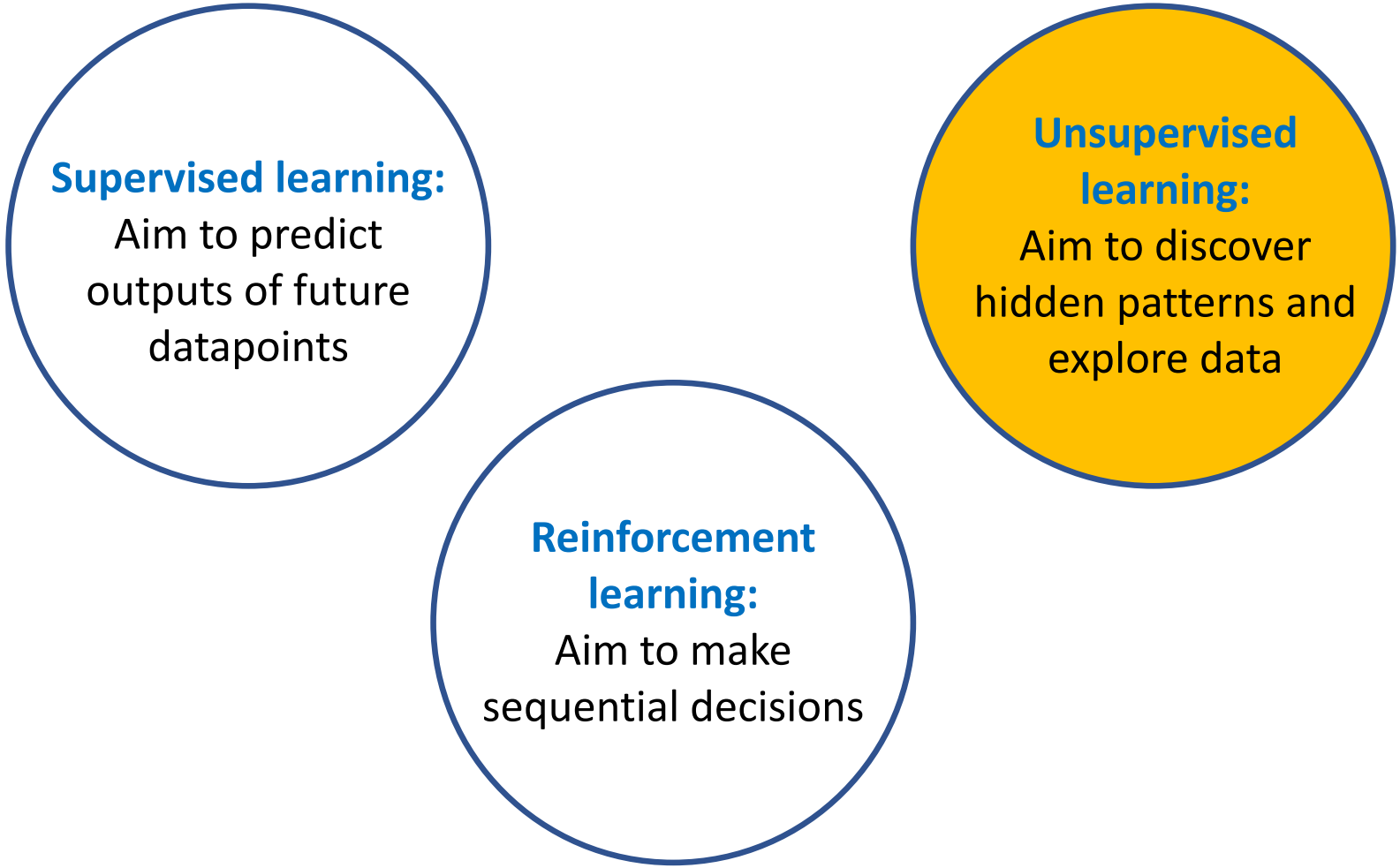
- Exam 2 is on May 1 in class (1pm-3:20pm)
  - Similar format to Exam 1
  - Syllabus is (primarily) multiclass classification (end of lecture 6) onwards

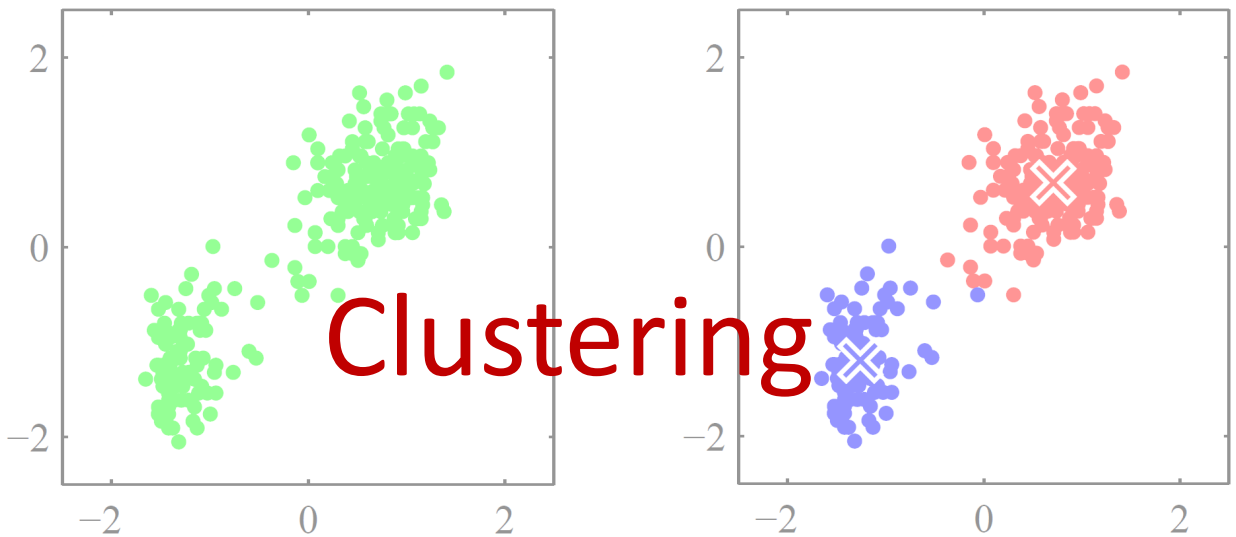
- Project pre-final check-ins next week
  - Upload your slides to Gradescope after your check-in

- Lecture next week will be on Tuesday April 21 from 5:30pm-7:50pm in OHE 132
- Discussion next week will be at 1pm (till around 3pm) on Friday April 24

- Today's plan:
  - Clustering
  - Gaussian Mixture Models and Expectation Maximization (EM)

A simplistic taxonomy of ML





Clustering

Clustering

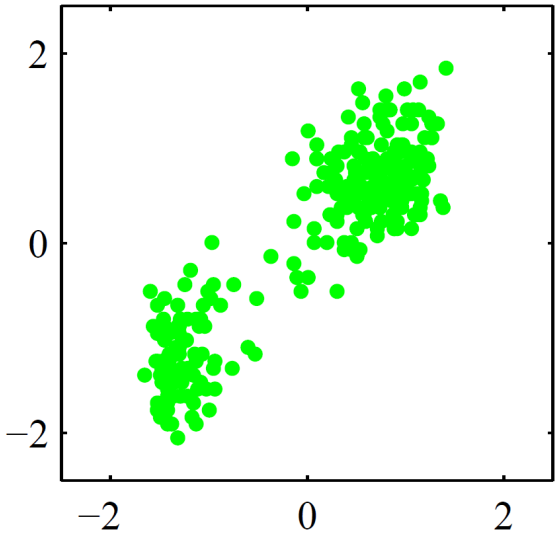
- Introduction
- Formalizing and solving the objective (alternating minimization)
- k-means algorithm

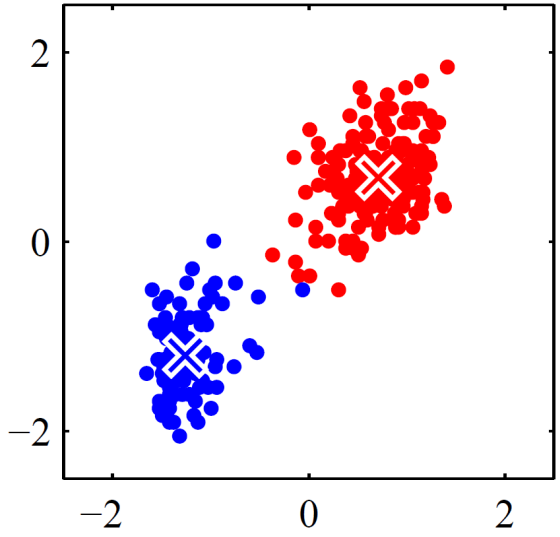
Clustering: Informal definition

Given: a set of data points (feature vectors), *without labels*

Output: group the data into some clusters, which means

- assign each point to a specific cluster
- find the center (representative/prototype/...) of each cluster



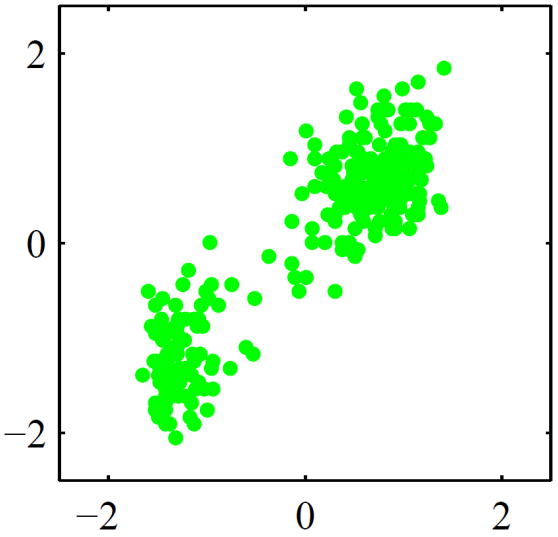


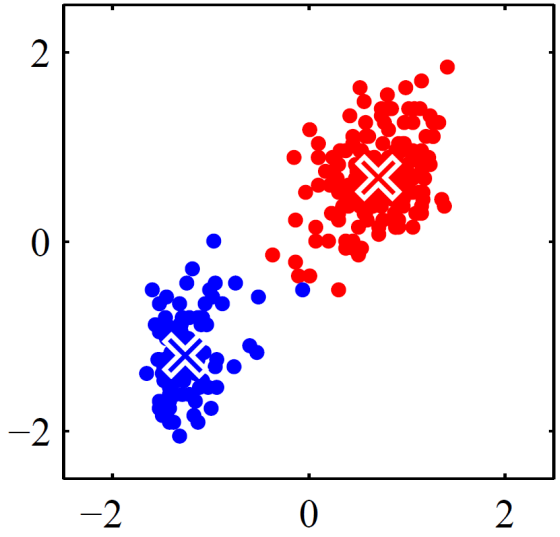
Clustering: More formal definition

Given: data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and #clusters k we want

Output: group the data into k clusters, which means,

- find assignment $\gamma_{ij} \in \{0, 1\}$ for each data point $i \in [n]$ and $j \in [k]$ s.t. $\sum_{j \in [k]} \gamma_{ij} = 1$ for any fixed i each datapoint is assigned to exactly 1 cluster
- find the cluster centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k \in \mathbb{R}^d$





Many applications

Clustering is one of the most fundamental ML tasks, with many applications:

- recognize communities in a social network
- group similar customers in market research
- image segmentation
- accelerate other algorithms (e.g. nearest neighbor classification)
- . . .

Clustering

- Introduction
- Formalizing and solving the objective (alternating minimization)
- k-means algorithm

Formal objective

As with PCA, no ground-truth to even measure the quality of the answer (*no labels given*).

What is the high-level goal here?

We want to partition the points into k clusters, such that points within each cluster are close to their cluster center.

We can turn this into an optimization problem, find γ_{ij} and $\boldsymbol{\mu}_j$ to minimize

$$F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j\}\right) = \sum_{i=1}^{n} \sum_{j=1}^{k} \gamma_{ij} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$

i.e. the **sum of squared distances of each point to its center**. This is the **"k-means" objective**.

How to solve this? Alternating minimization

Unfortunately, finding the exact minimizer of the k-means objective is *NP-hard!*

don't expect to solve the problem efficiently.

Therefore, we use a heuristic (*alternating minimization*) that alternatingly minimizes over $\{\gamma_{ij}\}$ and $\{\boldsymbol{\mu}_j\}$:

Initialize $\{\boldsymbol{\mu}_j^{(1)} : j \in [k]\}$

For $t = 1, 2, \ldots$

- find

$$\{\gamma_{ij}^{(t+1)}\} = \underset{\{\gamma_{ij}\}}{\operatorname{argmin}} F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j^{(t)}\}\right)$$

fix $\{\mu_j\}$, find $\{\gamma_{ij}\}$

- find

$$\{\boldsymbol{\mu}_j^{(t+1)}\} = \underset{\{\boldsymbol{\mu}_j\}}{\operatorname{argmin}} F\left(\{\gamma_{ij}^{(t+1)}\}, \{\boldsymbol{\mu}_j\}\right)$$

fix $\{\gamma_{ij}\}$, find $\{\mu_j\}$

Alternating minimization: Closer look

The first step

$$\min_{\{\gamma_{ij}\}} F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j\}\right) = \min_{\{\gamma_{ij}\}} \sum_i \sum_j \gamma_{ij} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$
$$= \sum_i \min_{\{\gamma_{ij}\}} \sum_j \gamma_{ij} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$

is simply to **assign each $\boldsymbol{x}_i$ to the closest $\boldsymbol{\mu}_j$**, i.e.

$$\gamma_{ij} = \mathbb{I}\left[j == \operatorname*{argmin}_{c \in [k]} \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|_2^2\right]$$

for all $j \in [k]$ and $i \in [n]$.

$$\begin{cases} 1, & \text{if } k_j \text{ is closest center to } x_i \\ 0, & \text{otw} \end{cases}$$

Alternating minimization: Closer look

The second step

$$\min_{\{\boldsymbol{\mu}_j\}} F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j\}\right) = \min_{\{\boldsymbol{\mu}_j\}} \sum_i \sum_j \gamma_{ij} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$
$$= \sum_j \min_{\boldsymbol{\mu}_j} \sum_{i:\gamma_{ij}=1} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$

is simply **to average the points of each cluster** (hence the name)

$$\boldsymbol{\mu}_j = \frac{\sum_{i:\gamma_{ij}=1} \boldsymbol{x}_i}{|\{i : \gamma_{ij} = 1\}|} = \frac{\sum_i \gamma_{ij} \boldsymbol{x}_i}{\sum_i \gamma_{ij}}$$

→ vectorized operation

for each $j \in [k]$.

Verify: take gradients!

Clustering

- Introduction
- Formalizing and solving the objective (alternating minimization)
- k-means algorithm

k-means algorithm

Step 0 Initialize $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$

Step 1 For the centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ being fixed, assign each point to the closest center:

$$\gamma_{ij} = \mathbb{I}\left[j == \operatorname*{argmin}_c \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|_2^2\right]$$

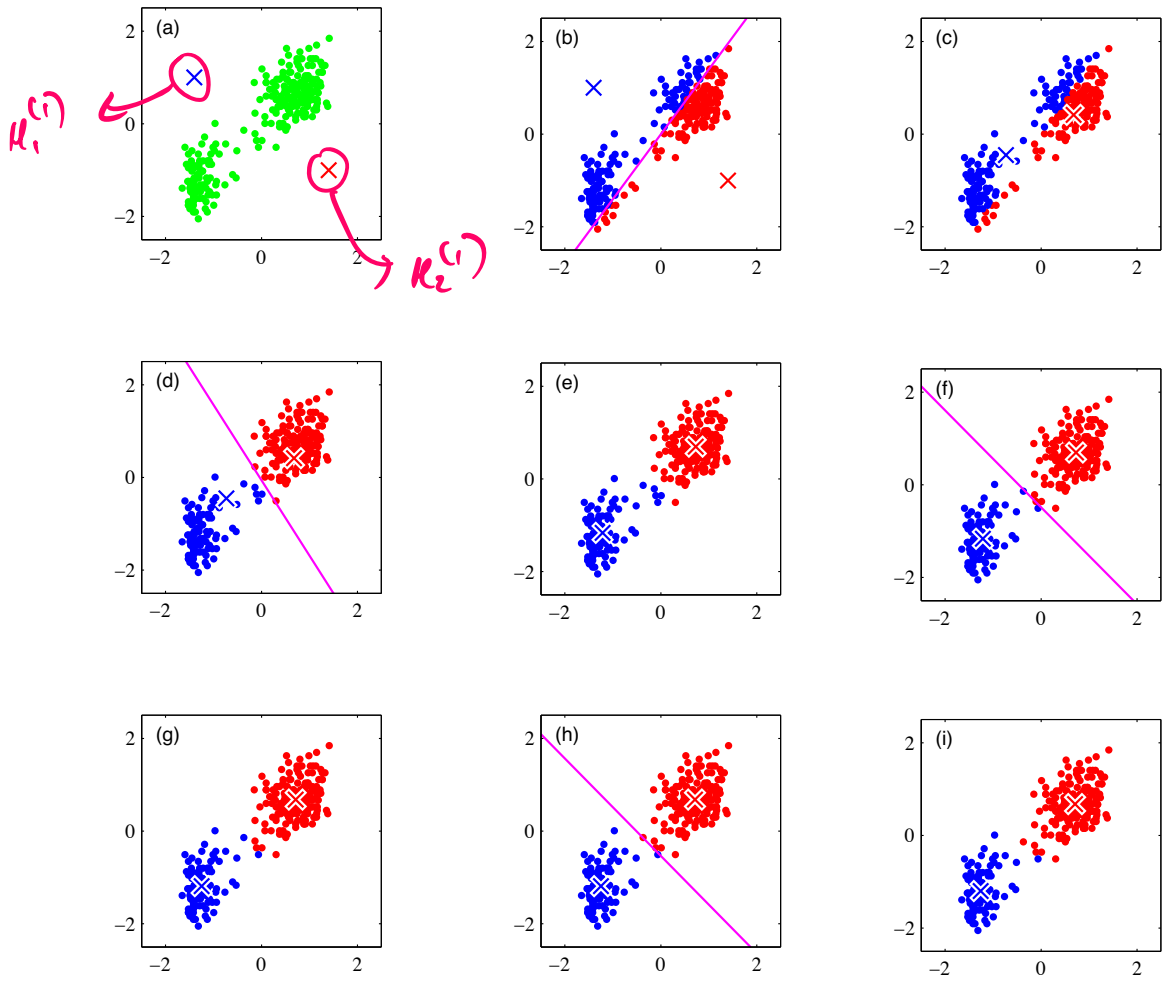
Step 2 For the assignments $\{\gamma_{ij}\}$ being fixed, update the centers

$$\boldsymbol{\mu}_j = \frac{\sum_i \gamma_{ij} \boldsymbol{x}_i}{\sum_i \gamma_{ij}}$$

Step 3 Return to Step 1 if not converged (convergence means that all the assigments γ_{ij} and means $\boldsymbol{\mu}_j$ are unchanged in Step 1 and 2).

k-means algorithm: Example

$k=2$



k-means algorithm: Convergence

k-means will converge in a finite number of iterations, why?

- objective strictly decreases at each step if the algorithm has not converged.

Why? For $t = 1, 2, \ldots$

- find

$$\{\gamma_{ij}^{(t+1)}\} = \underset{\{\gamma_{ij}\}}{\operatorname{argmin}} F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j^{(t)}\}\right)$$
$$= \mathbb{I}\left[j == \underset{c}{\operatorname{argmin}} \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|_2^2\right]$$

this step will never increase obj. function value (as long as there are no ties, then it decreases function value)

- find

$$\{\boldsymbol{\mu}_j^{(t+1)}\} = \underset{\{\boldsymbol{\mu}_j\}}{\operatorname{argmin}} F\left(\{\gamma_{ij}^{(t+1)}\}, \{\boldsymbol{\mu}_j\}\right)$$
$$= \frac{\sum_i \gamma_{ij} \boldsymbol{x}_i}{\sum_i \gamma_{ij}}$$

If the means change, then this reduces obj. function value (mean is unique minimizer of sum of squares objective)

k-means algorithm: Convergence

k-means will converge in a finite number of iterations, why?

- objective strictly decreases at each step if the algorithm has not converged.
- #possible_assignments is finite (k^n, exponentially large though)

k assignments to each point

Therefore, the algorithm must converge in at most k^n steps.

Why? More specifically, why can't the algorithm cycle between different clusterings?

- Suppose the algorithm finds the same clustering at time steps t_1 and t_2.
- Since the objective function value decreases at every step, this means the same clustering (at time steps t_1 and t_2) has two different costs, which is not possible.
- Therefore, by contradiction, the algorithm cannot cycle between clusterings.

k-means algorithm: Convergence

k-means will converge in a finite number of iterations, why?

- objective strictly decreases at each step if the algorithm has not converged.
- #possible_assignments is finite (k^n, exponentially large though)

However

- it could take *exponentially many iterations* to converge
- and it *might not converge to the global minimum* of the k-means objective

k-means algorithm: How to initialize?

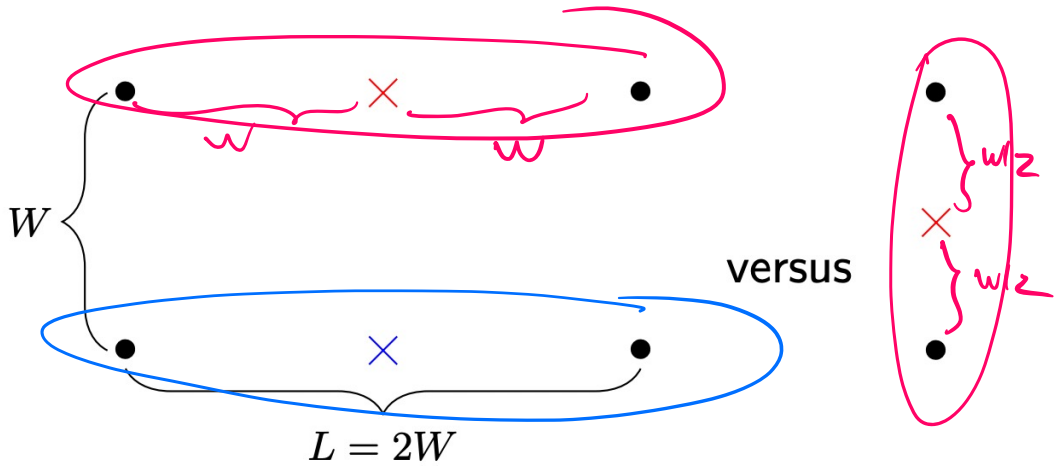
There are different ways to initialize:

- randomly pick k points as initial centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
- or randomly assign each point to a cluster, then average to find centers
- or more sophisticated approaches (e.g. k-means++)

Initialization matters for **convergence**.

k-means algorithm: Local vs Global minima

Simple example: 4 data points, 2 clusters, 2 different initializations



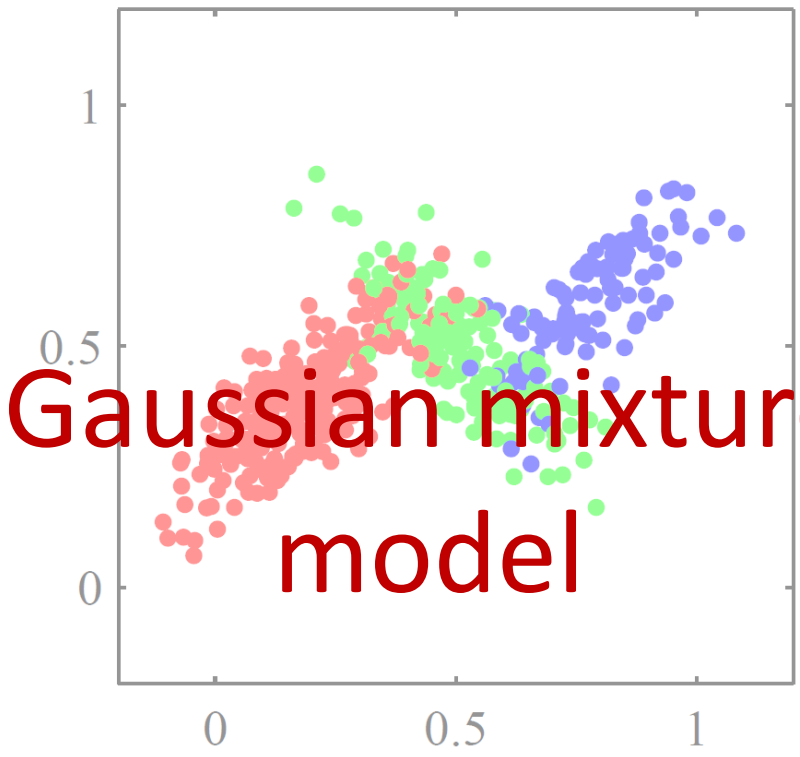
K-means converges immediately in both cases, but

- left has K-means objective $L^2 = 4W^2$
- right has K-means objective W^2, *4 times better than left!*
- in fact, left is **local minimum**, and right is **global minimum**.

as we increase L, we can make the local minimizer arbitarily bad! ∴ initialization matters for convergence!

k-means algorithm: Summary

- Clustering is a fundamental unsupervised learning task.
- k-means is a alternating minimization algorithm for the k-means objective.
- The algorithm always converges, but it can converge to a local minimum.
- Initialization matters a lot for the convergence. There are principled initialization schemes, which have guarantees on the solution they find (e.g. k-means++).



Gaussian mixture model

Gaussian Mixture Model

- Introduction
- Learning the parameters
- EM algorithm
- EM for the Gaussian Mixture Model

Gaussian mixture models

Gaussian mixture models (GMM) is a probabilistic approach for clustering

- more explanatory than minimizing the k-means objective
- can be seen as a soft version of k-means

To solve GMM, we will introduce a powerful method for learning probabilistic models:
the **Expectation Maximization (EM) algorithm**.

A generative model

$\Pr[y \mid x, w] = \sigma(y w^T x)$

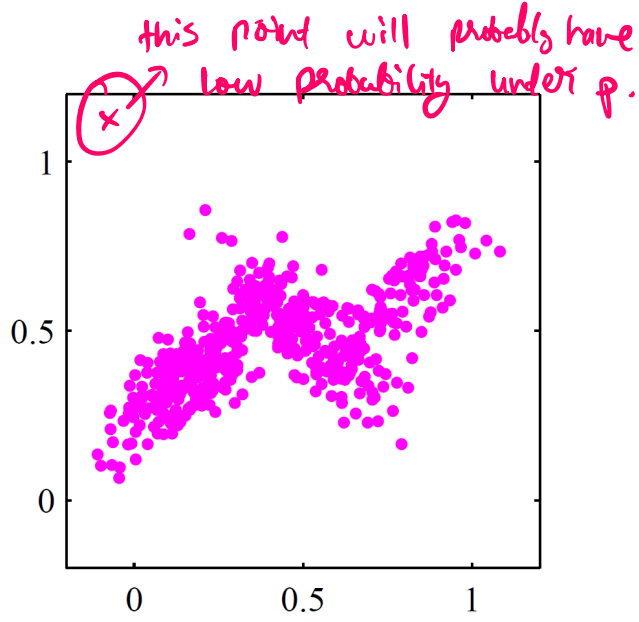
For classification, we discussed the sigmoid model to "explain" how the labels are generated.

Similarly, for clustering, we want to come up with a probabilistic model p to **"explain" how the data is generated**.

That is, each point is an independent sample of $\boldsymbol{x} \sim p$.

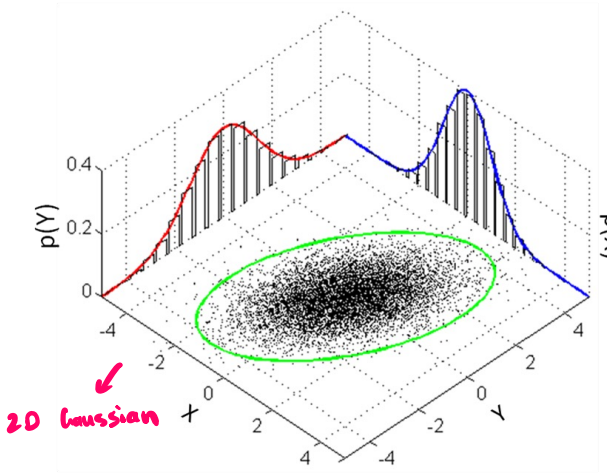
Why do generative modelling?

- can generate data from p
- can estimate probability of seeing any datapoint (useful for many tasks, such as for finding outliers/anomalies in data)



What probabilistic model generates data like this?

GMM: Intuition



GMM is a natural model to explain such data.

Assume there are 3 ground-truth Gaussian models.

To generate a point, we

- first **randomly pick one of the Gaussian models**,
- then **draw a point according this Gaussian**.

Hence the name "**Gaussian mixture model**".

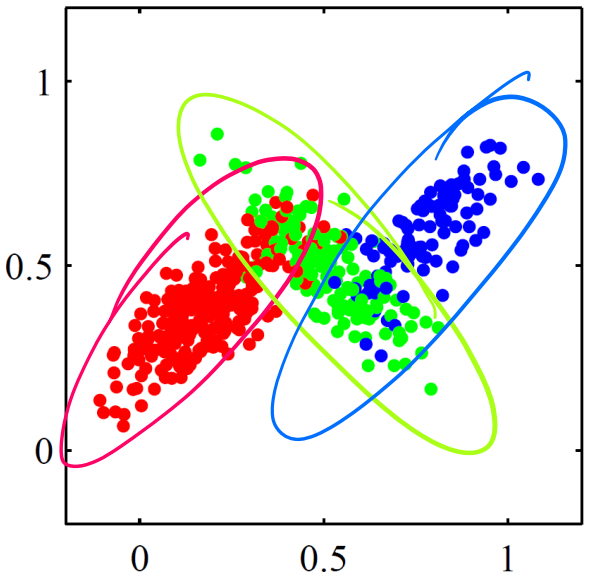


Figure from Wikipedia

GMM: Formal definition

A GMM has the following density function:

$$p(\boldsymbol{x}) = \sum_{j=1}^{k} \pi_j N(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

k Gaussians

then sample datapoint from Gaussian $\{\mu_j, \Sigma_j\}$

probability of picking Gaussian j

where

- k: the number of Gaussian components (same as #clusters we want)
- $\pi_1, \ldots, \pi_k$: mixture weights, a distribution over k components
- $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$: mean and covariance matrix of the j-th Gaussian
- N: the density function for a Gaussian

Another view

unobserved

By introducing a **latent variable** $z \in [k]$, which indicates cluster membership, we can see p as a **marginal distribution**

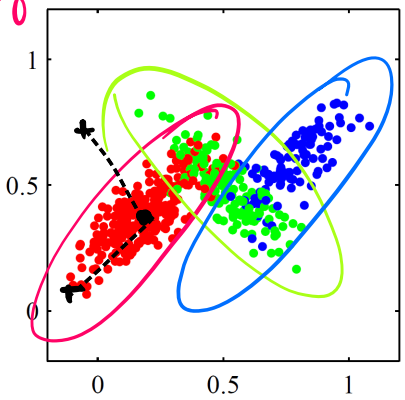
$$p(\boldsymbol{x}) = \sum_{j=1}^{k} p(\boldsymbol{x}, z = j) = \sum_{j=1}^{k} p(z = j) p(\boldsymbol{x} | z = j) = \sum_{j=1}^{k} \pi_j N(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

$\boldsymbol{x}$ and z are both random variables drawn from the model

- $\boldsymbol{x}$ is observed
- z is unobserved/latent

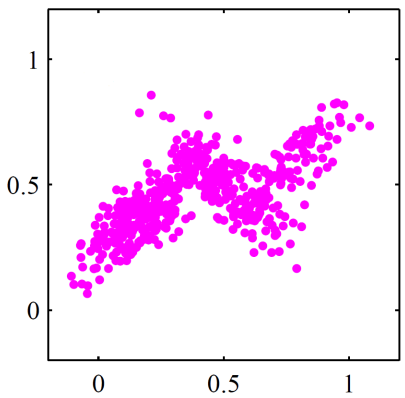
An example

GMMs can be better at handling scaling



The conditional distributions are

$$p(\boldsymbol{x} \mid z = \text{red}) = N(\boldsymbol{x} \mid \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$$
$$p(\boldsymbol{x} \mid z = \text{blue}) = N(\boldsymbol{x} \mid \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$$
$$p(\boldsymbol{x} \mid z = \text{green}) = N(\boldsymbol{x} \mid \boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$$



The marginal distribution is

$$p(\boldsymbol{x}) = p(\text{red})N(\boldsymbol{x} \mid \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + p(\text{blue})N(\boldsymbol{x} \mid \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) + p(\text{green})N(\boldsymbol{x} \mid \boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$$

Learning GMMs

Learning a GMM means finding all the parameters $\boldsymbol{\theta} = \{\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}_{j=1}^k$.

In the process, we will learn the distribution of the latent variable z_i as well:

$$p(z_i = j \mid \boldsymbol{x}_i) := \gamma_{ij} \in [0, 1]$$

i.e. "soft assignment" of each point to each cluster, as opposed to "hard assignment" by k-means.

GMM is more explanatory than k-means

- both learn the cluster centers $\boldsymbol{\mu}_j$'s
- in addition, GMM learns cluster weight π_j and covariance $\boldsymbol{\Sigma}_j$, thus
  - we can *predict probability of seeing a new point*
  - we can *generate synthetic data*

Gaussian Mixture Model

- Introduction
- Learning the parameters
- EM algorithm
- EM for the Gaussian Mixture Model

Density estimation

With clustering using GMMs, our high-level goal is the following:

Given a training set $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, **estimate a density function p that could have generated this data** (via $\boldsymbol{x}_i \overset{i.i.d.}{\sim} p$).

This is a special case of the general problem of *density estimation*, an important unsupervised learning problem.
Parametric methods for density estimation assume a generative model parameterized by $\boldsymbol{\theta}$: $p(\boldsymbol{x}) = p(\boldsymbol{x}\,; \boldsymbol{\theta})$

Examples:

- GMM: $p(\boldsymbol{x}\,; \boldsymbol{\theta}) = \sum_{j=1}^{k} \pi_j N(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ where $\boldsymbol{\theta} = \{\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}$
- Multinomial: a discrete variable with values in $\{1, 2, \ldots, k\}$ s.t.

$$p(x = j\,; \boldsymbol{\theta}) = \theta_j$$

  where $\boldsymbol{\theta}$ is a distribution over the k elements.

Size of $\boldsymbol{\theta}$ is independent of the size of the training set, so it's parametric.

Parametric methods: estimation via MLE

As usual, we can apply **MLE** to learn the parameters $\boldsymbol{\theta}$:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ln \prod_{i=1}^{n} p(\boldsymbol{x}_i ; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{n} \ln p(\boldsymbol{x}_i ; \boldsymbol{\theta})$$

log-likelihood of getting the data

For some cases this admits a simple closed-form solution. An example is the multinomial, for the multinomial the log-likelihood is

$$\sum_{i=1}^{n} \ln p(x = x_i ; \boldsymbol{\theta}) = \sum_{i=1}^{n} \ln \theta_{x_i} = \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbf{1}(x_i = j) \ln \theta_j = \sum_{j=1}^{k} \sum_{i:x_i=j} \ln \theta_j = \sum_{j=1}^{k} z_j \ln \theta_j$$

where $z_j = |\{i : x_i = j\}|$ is **the number of examples with value** j.

The solution is simply

$$\theta_j = \frac{z_j}{n} \propto z_j,$$

i.e. the fraction of examples with value j.

What about MLE for mixture of Gaussians?

We want to do MLE for Gaussian mixtures:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ \ln \prod_{i=1}^{n} p(\boldsymbol{x}_i \, ; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{n} \ln p(\boldsymbol{x}_i \, ; \boldsymbol{\theta}) := \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \, P(\boldsymbol{\theta}).$$

This is an incomplete log-likelihood (since z_i's are unobserved). We can still write it down as an optimization problem by marginalizing out the z_i's.

$$\begin{aligned} P(\boldsymbol{\theta}) &= \sum_{i=1}^{n} \ln p(\boldsymbol{x}_i \, ; \boldsymbol{\theta}) = \sum_{i=1}^{n} \ln \left(\sum_{j=1}^{k} p(\boldsymbol{x}_i, z_i = j \, ; \boldsymbol{\theta}) \right) \\ &= \sum_{i=1}^{n} \ln \left(\sum_{j=1}^{k} p(z_i = j \, ; \boldsymbol{\theta}) p(\boldsymbol{x}_i | z_i = j \, ; \boldsymbol{\theta}) \right) = \sum_{i=1}^{n} \ln \left(\sum_{j=1}^{k} \pi_j N(\boldsymbol{x}_i \mid \mu_j, \boldsymbol{\Sigma}_j) \right). \end{aligned}$$

This is a non-concave problem, and does not have a closed-form solution.

One solution is to still apply GD/SGD, but a much more effective approach is the **Expectation Maximization (EM) algorithm**.

Preview of EM for learning GMMs

Step 0 Initialize $\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j$ for each $j \in [k]$

Step 1 (E-Step) update the "soft assignment" (fixing parameters)

$$\gamma_{ij} = p(z_i = j \mid \boldsymbol{x}_i) \propto \pi_j \mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)$$

prior

likelihood

fix π_j, μ_j, Σ_j

find γ_{ij}

Step 2 (M-Step) update the model parameter (fixing assignments)

$$\pi_j = \frac{\sum_i \gamma_{ij}}{n} \qquad \boldsymbol{\mu}_j = \frac{\sum_i \gamma_{ij} \boldsymbol{x}_i}{\sum_i \gamma_{ij}}$$

→ weighted mean

$$\boldsymbol{\Sigma}_j = \frac{1}{\sum_i \gamma_{ij}} \sum_i \gamma_{ij} (\boldsymbol{x}_i - \boldsymbol{\mu}_j)(\boldsymbol{x}_i - \boldsymbol{\mu}_j)^{\mathrm{T}}$$

→ weighted covariance

fix γ_{ij},

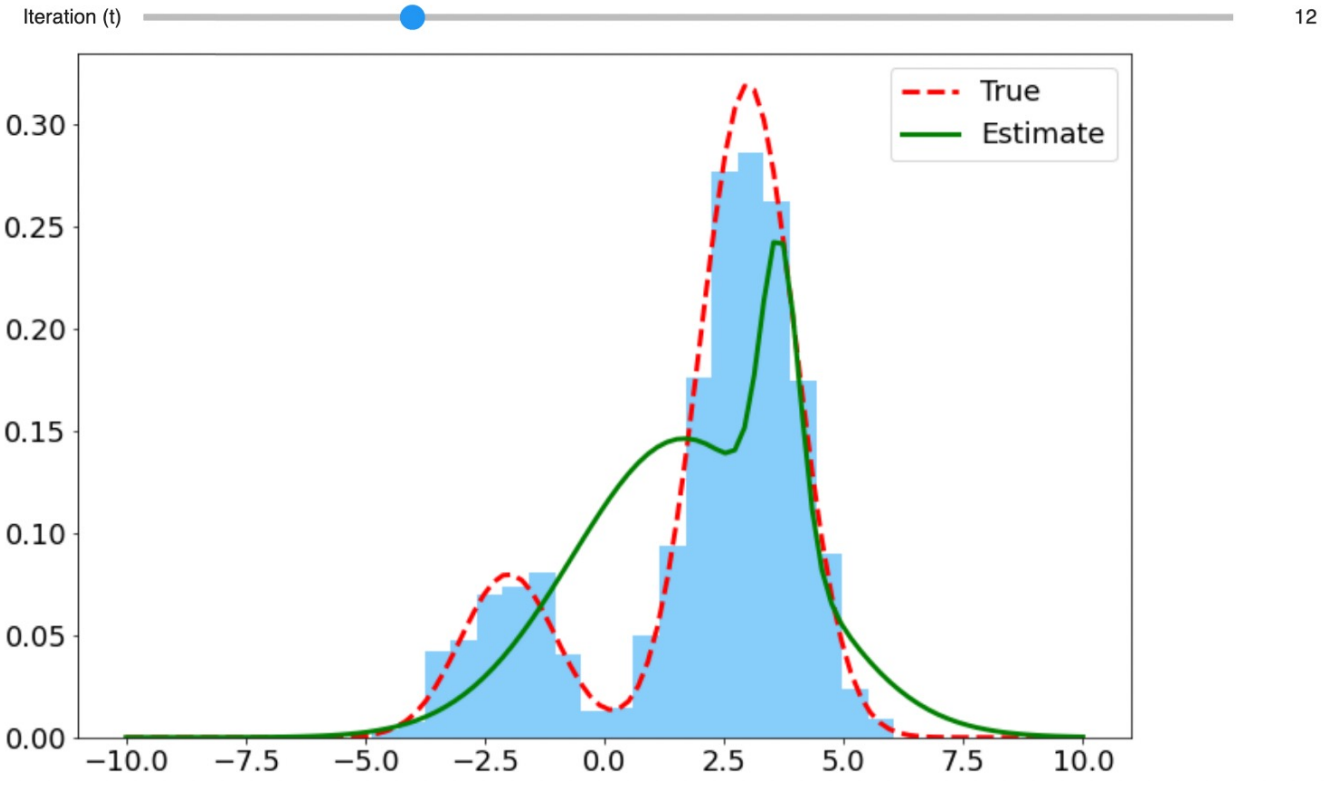
find

π_j, μ_j, Σ_j

Step 3 return to Step 1 if not converged

We will see how this is a special case of EM.

Demo



See Colab notebook

Gaussian Mixture Model

- Introduction
- Learning the parameters
- EM algorithm
- EM for the Gaussian Mixture Model

EM algorithm

In general EM is **a heuristic to solve MLE with latent variables** (not just GMM), i.e. find the maximizer of

$$P(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ln p(\boldsymbol{x}_i \,; \boldsymbol{\theta}) = \sum_{i=1}^{n} \ln \int_{z_i} p(\boldsymbol{x}_i, z_i \,; \boldsymbol{\theta}) dz_i$$

- $\boldsymbol{\theta}$ is the parameters for a general probabilistic model
- $\boldsymbol{x}_i$'s are observed random variables
- z_i's are latent variables

Like for k-means, directly solving the objective is usually complicated and does not have a closed form solution.

General EM algorithm: Alternating minimization

Step 0 Initialize $\boldsymbol{\theta}^{(1)}$, $t = 1$

Step 1 (E-Step) update the posterior of latent variables z_i,

$$q_i^{(t)}(z_i) = p(z_i \mid \boldsymbol{x}_i \,; \boldsymbol{\theta}^{(t)})$$

and obtain **Expectation** of complete likelihood

$$Q(\boldsymbol{\theta} \,; \boldsymbol{\theta}^{(t)}) = \sum_{i=1}^{n} \mathbb{E}_{z_i \sim q_i^{(t)}} \left[\ln p(\boldsymbol{x}_i, z_i \,; \boldsymbol{\theta})\right]$$

fix θ, find distribution of z_i

Step 2 (M-Step) update the model parameter via **Maximization**

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \operatorname*{argmax}_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \,; \boldsymbol{\theta}^{(t)})$$

fix distribution of z_i, find θ

Step 3 $t \leftarrow t + 1$ and return to Step 1 if not converged

Gaussian Mixture Model

- Introduction
- Learning the parameters
- EM algorithm
- EM for the Gaussian Mixture Model

Applying EM to learn GMMs: E-Step

E-Step:

$$
\begin{aligned}
q_i^{(t)}(z_i = j) &= p\left(z_i = j \mid \boldsymbol{x}_i\,; \boldsymbol{\theta}^{(t)}\right) \\
&= \frac{p\left(\boldsymbol{x}_i, z_i = j\,; \boldsymbol{\theta}^{(t)}\right)}{p\left(\boldsymbol{x}_i\,; \boldsymbol{\theta}^{(t)}\right)} \quad \rightarrow \text{does not depend on } j \\
&\propto p\left(\boldsymbol{x}_i, z_i = j\,; \boldsymbol{\theta}^{(t)}\right) \\
&= p\left(z_i = j\,; \boldsymbol{\theta}^{(t)}\right) p\left(\boldsymbol{x}_i \mid z_i = j\,; \boldsymbol{\theta}^{(t)}\right) \\
&= \pi_j^{(t)} N\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j^{(t)}, \boldsymbol{\Sigma}_j^{(t)}\right)
\end{aligned}
$$

Prior

Likelihood

This computes the "soft assignment" $\gamma_{ij} = q_i^{(t)}(z_i = j)$, i.e. conditional probability of $\boldsymbol{x}_i$ belonging to cluster j.

Applying EM to learn GMMs: M-Step

M-Step:

$$\operatorname*{argmax}_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \mathbb{E}_{z_i \sim q_i^{(t)}} \left[\ln p(\boldsymbol{x}_i, z_i \,; \boldsymbol{\theta})\right]$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \mathbb{E}_{z_i \sim q_i^{(t)}} \left[\ln p(z_i \,; \boldsymbol{\theta}) + \ln p(\boldsymbol{x}_i | z_i \,; \boldsymbol{\theta})\right]$$

$$= \operatorname*{argmax}_{\{\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}} \sum_{i=1}^{n} \sum_{j=1}^{k} \gamma_{ij} \left(\ln \pi_j + \ln N(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\right)$$

$\sum_{i=1}^{k} \gamma_i^{(t)}(z_i = j) \ln p(z_i = j; \theta)$ — γ_{ij} — $\ln \pi_j$

only depends on π_j — only depends on μ_j, Σ_j

To find $\pi_1, \ldots, \pi_k$, solve

$$\operatorname*{argmax}_{\boldsymbol{\pi}} \sum_{i=1}^{n} \sum_{j=1}^{k} \gamma_{ij} \ln \pi_j$$

To find each $\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j$, solve

$$\operatorname*{argmax}_{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j} \sum_{i=1}^{n} \gamma_{ij} \ln N(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

Applying EM to learn GMMs: M-Step

Solutions to previous two problems are very natural, for each j

$$\pi_j = \frac{\sum_i \gamma_{ij}}{n}$$

i.e. (weighted) fraction of examples belonging to cluster j

$$\boldsymbol{\mu}_j = \frac{\sum_i \gamma_{ij} \boldsymbol{x}_i}{\sum_i \gamma_{ij}}$$

i.e. (weighted) average of examples belonging to cluster j

$$\boldsymbol{\Sigma}_j = \frac{1}{\sum_i \gamma_{ij}} \sum_i \gamma_{ij} (\boldsymbol{x}_i - \boldsymbol{\mu}_j)(\boldsymbol{x}_i - \boldsymbol{\mu}_j)^{\mathrm{T}}$$

i.e (weighted) covariance of examples belonging to cluster j

You will verify some of these in mini-discussions next week.

Applying EM to learn GMMs: Putting it together

EM for learning GMMs:

Step 0 Initialize $\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j$ for each $j \in [k]$

Step 1 (E-Step) **update the "soft assignment"** (fixing parameters)

$$\gamma_{ij} = p(z_i = j \mid \boldsymbol{x}_i) \propto \pi_j N\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)$$

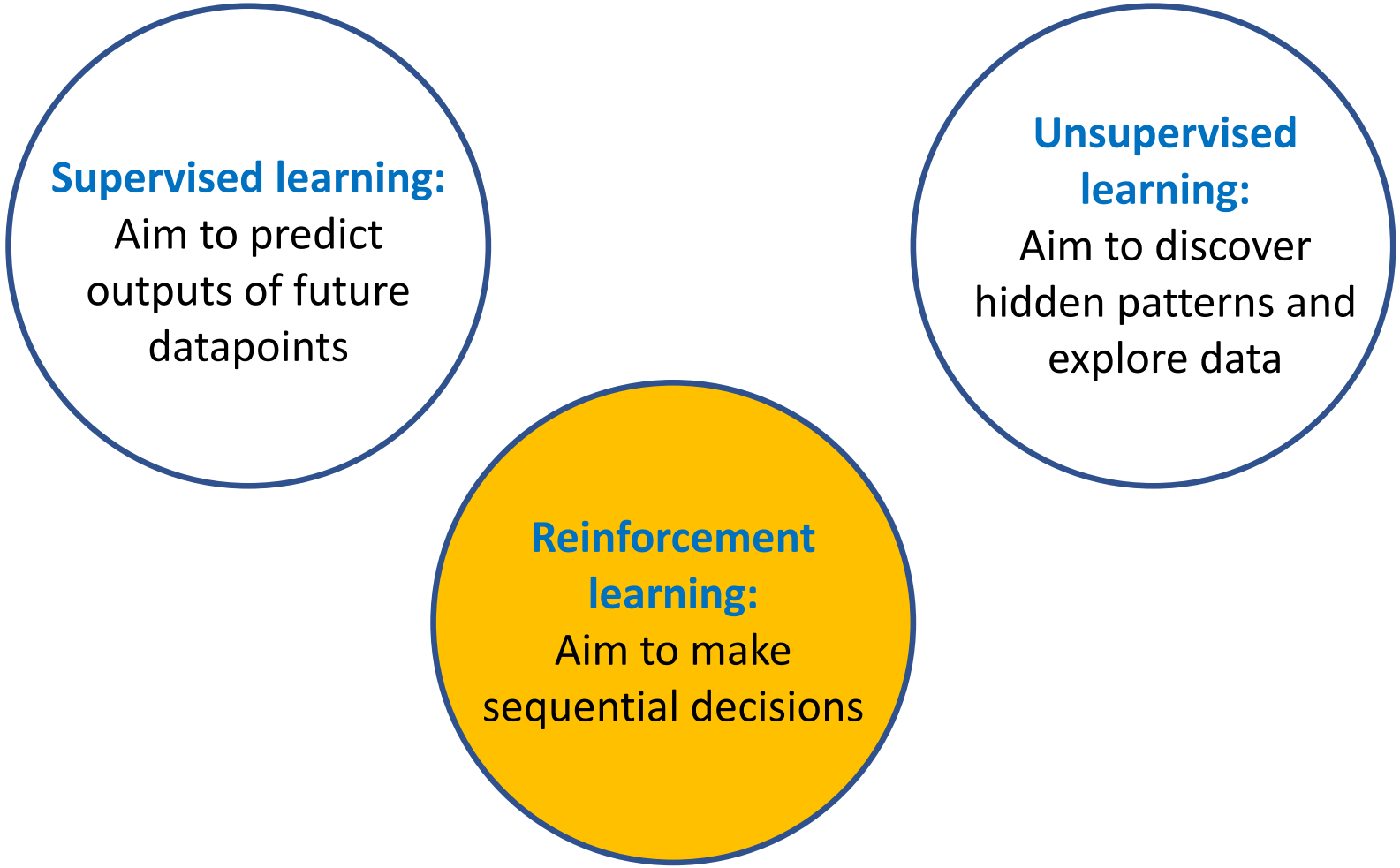
Step 2 (M-Step) **update the model parameter** (fixing assignments)

$$\pi_j = \frac{\sum_i \gamma_{ij}}{n} \qquad \boldsymbol{\mu}_j = \frac{\sum_i \gamma_{ij} \boldsymbol{x}_i}{\sum_i \gamma_{ij}}$$

$$\boldsymbol{\Sigma}_j = \frac{1}{\sum_i \gamma_{ij}} \sum_i \gamma_{ij} (\boldsymbol{x}_i - \boldsymbol{\mu}_j)(\boldsymbol{x}_i - \boldsymbol{\mu}_j)^{\mathrm{T}}$$

Step 3 return to Step 1 if not converged

A simplistic taxonomy of ML



Multi-armed bandits

- Motivation & setup
- Exploration vs. Exploitation

Decision making

Problems we have discussed so far:

- start with a fixed training dataset
- learn a predictor from the data or discover some patterns in the data

But many real-life problems are about **learning continuously**:

- make a prediction/decision
- receive some feedback
- repeat

Broadly, these are called **online decision making problems**.

Examples

Amazon/Netflix/Instagram **recommendation systems**:

- a user visits the website (or views a post etc.)
- the system recommends some products/movies/posts
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

- make a move
- receive some reward (e.g. score a point) or loss (e.g. fall down)
- make another move...

Multiarmed bandits: Motivation

Imagine going to a casino to play a slot machine

- it robs you, like a "bandit" with a single arm

Of course there are many slot machines in the casino

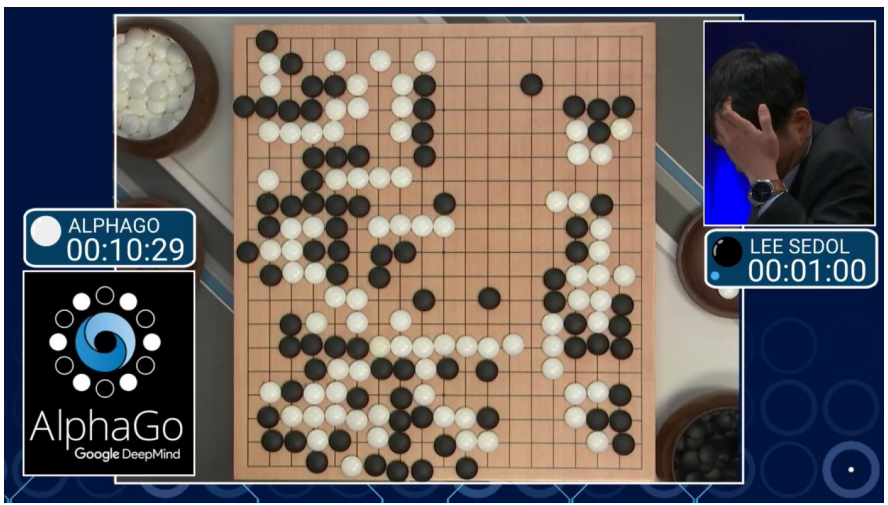
- like a bandit with multiple arms (hence the name)
- **if I can play 10 times, which machines should I play?**

Applications

This simple model and its variants capture many real-life applications:

- recommendation systems, each product/movie/news story is an arm
  (Netflix employs a variant of bandit algorithm)

- game playing, each possible move is an arm
  (AlphaGo has a bandit algorithm as one of the components)



Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward for arm a_t, i.e., r_{t,a_t}

Importantly, *learner does not observe rewards for arms not selected!*

This kind of limited feedback is usually referred to as bandit feedback

Evaluating performance

What should be the goal here?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural.

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*. A classic benchmark is

$$\max_{a \in [K]} \sum_{t=1}^{T} r_{t,a}$$

i.e. the largest reward one can achieve by always playing a fixed arm