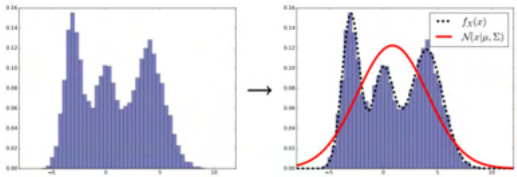
JNOTE.

Gaussian Mixture Models

We have seen that we can solve classification problems by building generative models that describe the distribution of samples

The Gaussian classifier is an example that assumes that class-conditional distributions are Gaussian

In many cases, however, the assumption can be quite inaccurate



single Gaussian fall short because:

- Real-world data rarely follows a perfect bell curve
- Many datasets have multiple peaks (multimodal distributions)
- Some distributions have asymmetric shapes or heavy tails
- Data clusters might appear in separate regions

So when a single Gaussian isn't sufficient we need more flexible models. We have several options:

① Domain-specific distribution: In some fields, we might know that data follows specific distributions (e.g. Poisson for count data)

② Gaussian Mixture Models (GMMs): A more general solution that can approximate virtually any smooth distribution by combining multiple Gaussian components.

GMMs can model complex multimodal distributions, it's possible to control the approximation precision by adding more components, GMMs maintain mathematical tractability while handling complex shape

Of course, since we are estimating the density from data, we require a sufficient amount of data to obtain good estimates

The use of GMMs is not restricted to classification

GMMs can be employed also in other tasks that require estimating a population density (e.g. data generation)

As we will see, they also allow to solve different kind of problems

For example, GMMs provide an alternative to K-means for clustering

It can be use also for open set classification

We have already encountered an example of GMM in the Gaussian classifier model

- In this classifier we have modeled each class with its own Gaussian distributions: $f_{X|C}(x|c) = N(x|\mu_c, \Sigma_c)$
- Assigned prior probabilities to each class $P(C=c) = \pi_c$
- Calculate the overall data distribution as a weighted sum of these class-conditional distributions.

This overall distribution is what we now call a Gaussian Mixture Model:

$$f_X(x) = \sum_{c=1}^{M} \pi_c N(x|\mu_c, \Sigma_c)$$

Digression on it

First, remember that in a classification setting, we have K different classes. For each class c, we have:

A prior probability πc that represents how likely any random data point belongs to class c $P(C=c) = \pi_c$, $c = 1, \dots, M$

A class-conditional density N(x | μc,Σc) that describes the distribution of data points within that class

Now, to find the probability density of observing any particular data point x (regardless of which class it comes from), we need to consider all possible classes. This uses a fundamental concept in probability called the "law of total probability." For each potential class c:

We multiply the probability that a point comes from class c (πc)

By the probability density of observing x given that it's from class c (N(x | μc,Σc)) N(x | μc,Σc))

Then we add up these products for all classes

This gives us the overall probability density for any point x, without knowing which class it belongs to:

$$f_X(x) = \sum_{c=1}^{K} \pi_c N(x|\mu_c, \Sigma_c)$$

Think of it like if we randomly pick a data point from our entire dataset, this formula tells us how likely we are to observe a specific value x, considering all possible classes that could have generated it.

Each component represent a different source that could have generated our data point.

Note in Gaussian Classifier —

POICHÉ SIAMO IN UN MODELLO DI CLASSIFICAZIONE A INSIEME CHIUSO (TUTTE LE POSSIBILI CLASSI SONO NOTE E IL CAMPIONE DEVE APPARTENERE AD UNA DI ESSE. APPLICHIAMO BAYES PER TROVARE UNA PROBABILITÀ

$$P(C=c \mid X_t=x_t) = \frac{f_{X,C}(x_t, c)}{\sum_{c' \in C} f_{X,C}(x_t, c) = f_X(x_t)}$$

DOVE IL DENOMINATORE SOMMA TUTTE LE POSSIBILI CLASSI IN [È UN VALORE COSTANTE CHE NORMALIZZA LA LA PROBABILITÀ A POSTERIORI GARANTENDO CHE SOMMANDOLE TUTTE PER C IL VALORE SIA 1]

DI FATTO È LA DENSITÀ DI PROBABILITÀ MARGINALE DOVE $X = x_t$ VALE FISSO E C ASSUME TUTTE LE POSSIBILITÀ

Formal definition of GMMs

More generally a Gaussian Mixture model represents a probability distribution as a weighted sum of multiple Gaussian components:

$$f_X(x) = \sum_{c=1}^{k} w_c \mathcal{N}(x \mid \mu_c, \Sigma_c)$$

Where:

- w_c are the mixture weights (analogous to the class priors in classification)
- μ_c are the means of each Gaussian component.
- Σ_c are the covariance matrix for each component
- k is the number of Gaussian components in our mixture.

Visual intuition: Imagine trying to model a mountain range. Each individual mountain can be represented by a Gaussian "bump," and the entire range is the combination of all these bumps, each with its own height, width, and position.

To fully specify a GMM, we need three sets of parameters:

1. Component means $M = [\mu_1, \mu_2, \ldots, \mu_k]$, probabilmente
   - These determine the center of each Gaussian "bump"
   - Each μ_i is a vector with the same dimensionality of our data.
2. Component Covariance $S = [\Sigma_1, \Sigma_2, \ldots \Sigma_k]$
   - This determine the shape, orientation, and spread of each Gaussian.
   - Each Σ_i is a positive-definite matrix of size $d \times d$ (d is the data dimensionality)
3. Mixture weights: $w = [w_1, w_2, \ldots, w_k]$
   - This determine the relative contribution of each component the the overall mixture
   - Large weights mean that component has more influence in the final distribution.

$$X \sim GMM(M, S, w)$$

$$f_X(x) = \sum_{c=1}^{k} w_c \mathcal{N}(x \mid \mu_c, \Sigma_c)$$

The distribution parameters are the component means

$$M = [\mu_1 \dots \mu_K]$$

the component covariances

$$S = [\Sigma_1 \dots \Sigma_K]$$

and the weights

$$w = [w_1 \dots w_K]$$

Remember that, for f_X to be a density, we need that its integral is equal to 1. Integrating w.r.t. x we have:

$$\int f_X(x) = \int \sum_{c=1}^{K} w_c \mathcal{N}(x; \mu_c, \Sigma_c) dx \sum_{c=1}^{K} w_c \int \mathcal{N}(x|\mu_c, \Sigma_c) dx = \sum_{c=1}^{K} w_c = 1$$

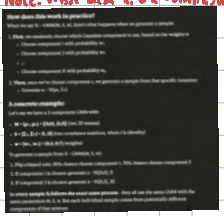
if the weights are the priors(c) then sum is 1

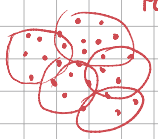
i.e., the weights must sum to 1

When working with a dataset D = [x_1, x_2, ..., x_n], GMMs provide us with a powerful framework for modeling the underlying distribution. We make a fundamental assumption: all samples have been independently generated by the same GMM distribution.
More formally, we treat the data points as independent and identically distributed (i.i.d.) random variables, where each X_i follows the same distribution X ~ GMM(M, S, w).
This is important to understand in context: $X_i \sim X \sim GMM(M, S, w)$ ↳ sum of diff. real Gaussian component

- In density estimation, we're simply trying to model the overall distribution of our data
- Unlike supervised classification, we're not assigning predefined class labels
- The dataset D is considered unlabeled - we don't know in advance which Gaussian component generated each point
- The GMM could be modeling data from a single class (if used within a classification framework) or an entire dataset without any class distinctions

Note: What does $x_i \sim X \sim GMM(M, S, w)$





Just as with "simpler" Gaussian model we want to find the parameters that maximize the likelihood of observing our data, however, GMM model introduce a significant challenge that single Gaussian don't have: the likelihood function is unbounded.

This creates what mathematicians call an "ill-posed-problem".
Imagine we place one Gaussian component directly on a single data point and make his variance extremly small (approaching zero) This create an extremly narrow tall peak, As the variance gets smaller and smaller:

- The probability density at that exact point approaches infinity
- The likelihood becomes unbounded.
- The solution becomes degenerate (mathematically valid but practically useless)

In practice we address this issue with regularization techniques and constraint on the covariance matrices to prevent these degenerate solutions. With these safeguards, maximum likelihood estimation work well for GMMs

Going deeper

A degenerate solution occurs when one of the Gaussian components becomes pathologically "concentrated" on a very small number of data points (often just one point).

Here's what happens step by step:

The Pathological Scenario

Imagine we have a GMM with K=2 components, and we place one component directly on a single data point, say x_1:

1. **Set the mean**: $\mu_1 = x_1$ (place the Gaussian exactly on the data point)
2. **Shrink the variance**: Make $\Sigma_1 = \sigma^2 I$ where $\sigma \to 0$ (make the Gaussian extremely narrow)
3. **Keep reasonable weight**: w_1 could be any reasonable value, say 0.3

What Happens to the Likelihood?

The probability density of a Gaussian at point x is:

$$N(x|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

When we place $\mu_1 = x_1$ and make $\Sigma_1 = \sigma^2 I$ with $\sigma \to 0$:

- The exponent becomes: $\exp\left(-\frac{1}{2}\frac{\|x_1 - x_1\|^2}{\sigma^2}\right) = \exp(0) = 1$
- The normalization factor becomes: $\frac{1}{(2\pi\sigma^2)^{d/2}} = \frac{1}{(2\pi)^{d/2}\sigma^d}$

So: $N(x_1|\mu_1, \Sigma_1) = \frac{1}{(2\pi)^{d/2}\sigma^d}$

As $\sigma \to 0$, this probability density $\to \infty$!

Impact on the Total Likelihood

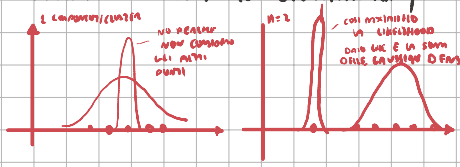
The likelihood contribution from point x_1 becomes:

$$L_1 = w_1 N(x_1|\mu_1, \Sigma_1) + w_2 N(x_1|\mu_2, \Sigma_2) \approx w_1 \cdot \frac{1}{\sigma^d} + \text{bounded term}$$

As $\sigma \to 0$, this grows without bound, making the total likelihood unbounded.

Indeed, as long as we have more than 1 component, we can devise degenerate solutions for which the likelihood is not bounded above

A DIFFERENZA DEL GAUSSIAN CLASSIFIER IN CUI HO UN UNICA NORMALE CHE MI DESCRIVE L'INTERO DATASET, QUI VE NE HO DI PIÙ, (ALCUNE UN UNICA N NORMALE CON GAUSSIANE SU PIÙ PUNTI, QUI INVECE LA NORMALE PUÒ ESSERE CENTRATA SULL'UNICO PUNTO DEL DATASET...)



Remember

$$f_{X+Y}(v) = \int f_{X+Y}(x,v)\,dx = \int f_{X+Y|X}(v|x) f_X(x)\,dx$$

(sum of random variables; sum of conditional random variables)

Let's define our model parameters as $\theta = [\pi, \mu, \Sigma, w]$ which includes all component means, covariance and weights. The likelihood function is:

$$L(\theta) = \prod_{i=1}^{n} f_{X_i}(x_i) = \prod_{i=1}^{n} GM(x_i|\pi,\mu,\Sigma,w) = \prod_{i=1}^{n}\left(\sum_{c=1}^{K} w_c N(x_i|\mu_c, \Sigma_c)\right)$$

($f_{X_i}(x_i)$)

Taking the logarithm gives us the log-likelihood:

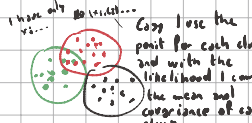
$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} \log\left(\sum_{c=1}^{K} w_c N(x_i|\mu_c, \Sigma_c)\right)$$

USARE ANCHE ANCHE QUESTA E TROVARE I PARAMETRI CON UN NUMERO DI CLUSTER MAGGIORE DI 1, PUÒ PORTARE A MODELLI QUALE ● ?

A key insight for working with GMMs is interpreting them as a marginal distribution of a joint distribution between observed data points and unobserved (latent) cluster assignments.

For each data point x_i:

- We have an associate (but unobserved) cluster label c_i
- The probability that x_i belongs to cluster c is $P(c_i = c) = w_c$
- The conditional distribution of x_i given $c_i = c$ is $N(x_i|\mu_c, \Sigma_c)$



Visually, imagine our complex data distribution as composed of several simpler Gaussian "sub-populations" or clusters:

- Each cluster follow its own Gaussian distribution
- Points are probabilistically assigned to one of these clusters
- The overall distribution is the weighted combination of these cluster-specific distributions.

The problem is that we don't know which data point belongs to with cluster. if we knew the cluster assignments (we don't know nothing, even the number K of cluster)

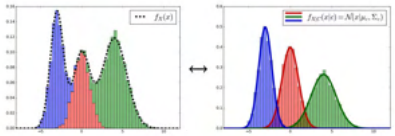
① We could sliced the data into K separate groups
② For each group we could easily estimate the mean and the covariance using standard ML techniques for Gaussians.
③ The mixture weights would simply be the propotion of points in each cluster. (if we know at which cluster belongs each point w will be 0.0.1)

If we knew the component responsible for each sample (i.e. its cluster label), we could estimate the parameters of each Gaussian by ML from the points of each cluster

Unfortunately, in general the clusters are unknown

We treat cluster membership as an unobserved (latent) random variables[1]

Intuitively, we want to estimate both cluster assignments and model parameters as to maximize the *marginal* distribution of the data



[1]Note that the model is not identifiable: for example, exchanging any two components results in the same marginal likelihood

$w_1\ \mu_1\ \Sigma_1$
$w_2\ \mu_2\ \Sigma_2$
$w_3\ \mu_3\ \Sigma_3$

Let's consider a set of GM parameters $\theta = (\pi, \mu, \Sigma)$, we said that we don't know the cluster assignament, so we suppose that we know that.
For any sample x_i and component (cluster) c, we can define a joint density

$$f_{x_i, C_i}(x_i, c) = w_c\, N(x_i / \mu_c, \Sigma_c)$$

This represents the probability of both observing data point x_i AND having it come from component c. Think of this as answering the question: "What's the probability that we see this specific data point and it was generated by this specific component?"

From the joint probability using the Bayes theorem we can compute the posterior probabilities or responsibilities:

$$\gamma_{c,i} = P(C_i = c / x_i = x_i) = \frac{f_{x_i, C_i}(x_i, c)}{f_{X_i}(x_i)}$$

$$= \frac{w_c \, N(x_i | \mu_c, \Sigma_c)}{\sum_{c'} w_{c'} N(x_i | \mu_{c'}, \Sigma_{c'})} \quad \text{[our GM density]}$$

This tell us, given that we observed data point x_i, what's the probability it come from component c?

Now, as a first approximation, we can make hard assignments by giving each point to the component with the highest posterior probability:

$$c_i^* = \arg\max_c P(C_i = c | X_i = x_i)$$

So even though we're uncertain about cluster membership, let's pretend each point definitely belongs to its most likely cluster" (This is the absolute truth

Given that we can then estimate by ML the new GM parameters $\theta^{new} = (M^{new}, S^{new}, w^{new})$

The log-likelihood becomes:

$$L(\theta) = \prod f_{X_i, C_i}(x_i, c_i^*)$$

DIFFERENZA RISPETTO A MVG: LA PRIOR PROBABILITÀ FA PARTE DEL MODELLO E LA DOBBIAMO STIMARE, NON È UNA CONOSCENZA A PRIORI (CHE POSSIAMO TOGLIERE) AS IN THE MVG

$$\ell(\theta) = \sum_{i=1}^{n} \log f_{X_i|C_i}(x_i | c_i^*) + \sum_{i=1}^{n} \log P(C_i = c_i^*)$$

$$= \sum_{i=1}^{n} \left[\log N(x_i | \mu_{c_i^*}, \Sigma_{c_i^*})\right] + \sum_{i=1}^{n} \left[\log w_{c_i^*}\right]$$

The first term depends only on the Gaussian parameters (M and S), while the second term depends only on the mixture weights (w). They don't interact with each other mathematically, which means we can optimize them completely independently

$$\ell(\theta) = \ell_N(M, S) + \ell_c(w)$$

Let's focus on the first term:

$$\ell_N(M, S) = \sum_{i=1}^{n} \left[\log N(x_i | \mu_{c_i^*}, \Sigma_{c_i^*})\right]$$

corresponds to the log-likelihood of a (multivariate) Gaussian classification model, where the class labels are assumed to be the estimated c_i^*.

$$\ell_N(M, S) = \sum_{c=1}^{k} \sum_{i | c_i^* = c} \log N(x_i | \mu_{c_i^*}, \Sigma_{c_i^*})$$

This is exactly the same as if I had k separate datasets, one for each cluster, and I was fitting a single Gaussian to each dataset independently. oint

Let N_c be the number of samples for which $c_i^* = c$. The solution for μ_c and Σ_c is thus [the maximum likelihood estimate]

$$\mu_c^* = \frac{1}{N_c} \sum_{i|c_i^*=c} x_i \,, \quad \Sigma_c^* = \frac{1}{N_c} \sum_{i|c_i^*=c} (x_i - \mu_c^*)(x_i - \mu_c^*)^T$$

$c_1^* = 2$
$c_2^* = 2$
$c_3^* = 2$
$c_4^* = 3$

$\sum_{c=1}^{n} N_c \log w_c$ =

" $\log w_2 + \log w_2 + \log w_2 + \log w_3 + \dots$

For the second term, we have $l_L(w) = \sum_{i=1}^{N} [\log w_{c_i^*}] = \sum_{c=1}^{n} \sum_{i|c_i^*=c} \log w_c$

This is the log-likelihood of a categorical distribution with n categories, where we observed N_c occurrences of category c. The maximum likelihood estimate for a categorical distribution is simply the relative frequency

$$w_c^* = \frac{N_c}{\sum_{c=1}^{n} N_c}$$

" Total number of samples

What is a Categorical Distribution? Recap

The categorical distribution is one of the simplest probability distributions you can imagine. It describes the outcome of a single trial where exactly one of K possible outcomes can occur, and each outcome has its own probability.

Think of it as the mathematical way to describe any situation where you're choosing one option from several mutually exclusive possibilities. The classic example is rolling a die: you have 6 possible outcomes (1, 2, 3, 4, 5, 6), and exactly one will happen on each roll.

The Mathematical Setup

For a categorical distribution with K categories, we need K parameters: $p_1, p_2, \dots, p_K$, where p_i represents the probability of outcome i occurring. These probabilities must satisfy two important constraints:

First, each probability must be non-negative: $p_i \geq 0$ for all i.

Second, since exactly one outcome must occur, the probabilities must sum to one: $p_1 + p_2 + \dots + p_K = 1$.

When we observe a single outcome, say outcome j, the probability of this observation is simply p_j. This seems almost trivially simple, but it becomes more interesting when we have multiple observations.

Multiple Observations and Maximum Likelihood

Now suppose we conduct n independent trials and observe the following counts: we see outcome 1 exactly N_1 times, outcome 2 exactly N_2 times, and so on, where $N_1 + N_2 + \dots + N_K = n$.

The likelihood of observing this particular sequence of outcomes is the product of all individual probabilities. Since we observed outcome 1 exactly N_1 times, outcome 2 exactly N_2 times, and so forth, the likelihood becomes:

$L(p_1, p_2, \dots, p_K) = p_1^{N_1} \times p_2^{N_2} \times \dots \times p_K^{N_K}$

Taking the logarithm gives us the log-likelihood:

$\ell(p_1, p_2, \dots, p_K) = N_1 \log p_1 + N_2 \log p_2 + \dots + N_K \log p_K$

The Maximum Likelihood Solution

To find the parameters that maximize this log-likelihood, we need to use calculus, but we must respect the constraint that the probabilities sum to one. Using the method of Lagrange multipliers (which I can explain separately if you'd like), the solution turns out to be beautifully simple:

$\hat{p}_i = N_i/n$

In other words, the maximum likelihood estimate for the probability of outcome i is simply the relative frequency with which we observed outcome i. This matches our intuitive understanding perfectly.

Connecting Back to GMM Weights

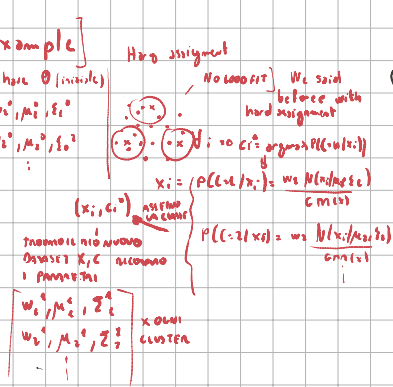
Now let's see how this connects to our Gaussian Mixture Model context. In our GMM, each data point is assigned to exactly one cluster. We can think of this assignment process as a categorical distribution where:

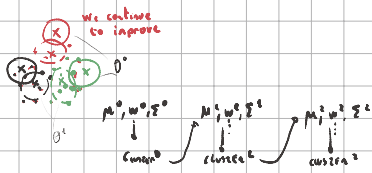
The K categories correspond to the K mixture components. The probability parameters correspond to the mixture weights $w_1, w_2, \dots, w_K$. When we observe that N_c data points are assigned to component c, we're essentially observing N_c occurrences of "category c" in our categorical distribution.

Following the maximum likelihood principle for categorical distributions, our estimate for the mixture weight w_c becomes:

$w_c^* = N_c/n$

This is exactly the formula we saw in the GMM parameter estimation! The mixture weights are simply the maximum likelihood estimates for a categorical distribution where the "outcomes" are cluster assignments.





The Chicken-and-Egg Problem Visualized

Imagine you're looking at a dataset of people's heights and weights, and you suspect there are three groups hidden in your data - perhaps children, women, and men. But you don't know who belongs to which group! This creates our fundamental problem:

To figure out the parameters of each group (like average height and weight), you need to know who belongs to each group. But to assign people to groups, you need to know what each group looks like. It's like trying to sort mail without knowing the addresses, but needing to know the addresses to sort the mail.

Breaking the Deadlock with Iteration

The brilliant insight is that we can break this deadlock by making educated guesses and gradually improving them. Let me walk you through exactly how this works with a concrete example.

Step 1: Make an Initial Guess

Let's say we start by randomly guessing that our three groups have these characteristics:

Group 1 (children): average height 120cm, average weight 30kg
Group 2 (women): average height 160cm, average weight 60kg
Group 3 (men): average height 180cm, average weight 80kg

These initial guesses are probably wrong, but that's okay - we'll improve them.

Step 2: Compute Responsibilities

Now, for each person in our dataset, we ask: "Given their actual height and weight, which group are they most likely to belong to?"

For example, if we see someone who is 175cm tall and weighs 75kg:

Probability they're from Group 1 (children): very low (they're too tall and heavy)
Probability they're from Group 2 (women): moderate (possible, but on the tall/heavy side)
Probability they're from Group 3 (men): high (fits well with our current guess)

We compute these probabilities mathematically using our current parameter estimates.

Step 3: Make Hard Assignments

We then assign each person to their most likely group. Our 175cm, 75kg person gets assigned to Group 3 (men) because that had the highest probability.

Step 4: Update Parameters our assignment was based on the probabilities so it wasnt correct for force.

Now here's the key insight: we use these assignments to compute better parameter estimates. We look at all the people we assigned to Group 3 and compute their actual average height and weight. Maybe it turns out to be 177cm and 78kg instead of our initial guess of 180cm and 80kg.

We do this for all three groups, getting updated estimates based on who we think belongs to each group.

Step 5: Repeat and Improve

We then go back to Step 2, but now with our improved parameter estimates. This time, when we compute responsibilities for our 175cm, 75kg person, the probabilities might be different because our group characteristics have changed. Maybe now they're more likely to belong to Group 2 than we initially thought.

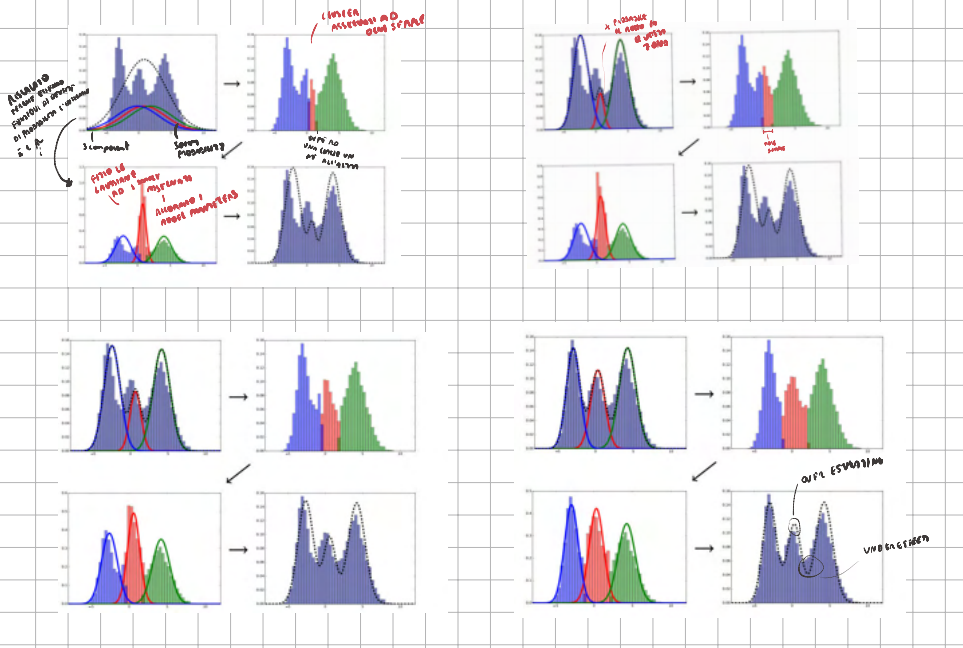
Why This Creates Progress

Each iteration improves our estimates in a specific way. The responsibilities in Step 2 are computed using the best information we currently have about each group. The parameter updates in Step 4 are the mathematically optimal estimates given our current assignments. So each step is making the best possible decision given what we know at that moment.

The beautiful mathematical property is that each iteration increases the likelihood of our data under the model. Think of likelihood as a measure of "how well does our current model explain what we actually observed?" Each iteration makes our model explain the data better.

Why It Eventually Stops

The process converges because there are only so many ways to improve. Eventually, the assignments become stable - people stop switching between groups, and the parameter estimates stop changing significantly. At this point, we've found a local optimum where our model does a good job of explaining the data structure.



The hard assignment approach has a critical flaw that become apparent when we examinate what happens at cluster boundaries. Consider a data point that lies exactly between two cluster centers. When we compute the posterior probability, we might get something like

- $P(C_i = 1 | X_i = x_i) = 0,52$ (52% of belonging to C_1)
- $P(C_i = 2 | X_i = x_i) = 0,48$ (48% of belonging to C_2)

With hard assignments we would definitively assign this point to cluster 2 since $0.52 > 0.48$. But think about how cruel this approximation is! We're treating a point that's almost equally likely to belong to either cluster as if it definitely belongs to just one cluster. This throws away valuable information about the uncertainty in our assignments.

This becomes even more problematic when posterior probabilities are more evenly distributed, such as $P(C_i = 1 | X_i = x_i) = 0,4$, $P(C_i = 2 | X_i = x_i) = 0,35$, $P(C_i = 3 | X_i = x_i) = 0,35$. We will assign the point to cluster 1 but is actually quite uncertain which cluster generate it.

So the hard assignment algorithms is not actually maximizing the likelihood of our observed data. We are making approximation that lead us away from the optimal solution.

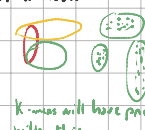
So before we develop a full solution to estimate a local maximum to the likelihood, let's see what happens when we make some simplifying assumptions.

We still consider hard-assignments,
We constraint our GM so that:
- All covariance matrices are identity matrices, $\Sigma_c = I$
- All mixture weights are equal: $w_c = \frac{1}{N}$ (N = number of clusters)

In this case cluster assignment corresponds to the rule

$$c_i^* = \arg\max_c P(c_i = c \mid x_i = x_i) = \arg\min_c \|x_i - \mu_c\|^2$$

Is a generalization of K-means



K-means will have problem with this

↓

because K=3. The GM is better, the GM gives us a probability (K-means no)

So the posterior probabilities $P(c_i = c \mid x_i = x_i)$ becomes proportional to the Gaussian density.

Our algorithm becomes

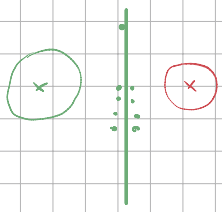
- Compute the component or cluster c_i^* whose centroid $\mu_{c_i^*}$ is closest to our point and assign x_i to that cluster
- Re-estimate the cluster centroids from the given points, and iterate until convergence

This is the K-Means clustering algorithm

GMMs can also be applied to clustering tasks as a generalization of K-Means

The algorithm we considered can be extended to handle soft assignments

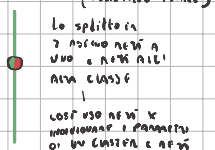
We will see that a point is not completely associated to a single Gaussian component, but contributes to the estimation of different components according to its cluster (component) posterior probability



Tutti point have the same probability more or less $p(c_1) = \frac{1}{2}$ $p(c_2) = \frac{1}{2}$

Posso assegnare meglio a uno e meno all'altro

Se ho ho solo uno? (obiettivo finale)



Lo splitto in 2 ascolto meglio a uno e meno all'altro

Nella classe

↓

Cosi uso meglio k indipendenti i parametri di un cluster e meglio di un altro

The key insight for improving our algorithm is to embrace uncertainty rather than eliminate it. Instead of forcing each point to belong to exactly one cluster, we can allow points to contribute to multiple clusters, according to its posterior probability

We consider the full log-likelihood of our data

Combino tutti i cluster non chiudo cluster a cui appartiene ogni data point

$$\sum_{i=1}^{n} \log f_{X_i}(x_i \mid \Theta) = \sum_{i=1}^{n} \log\left(\sum_{c=1}^{K} w_c \, N(x_i \mid \mu_c, \Sigma_c)\right)$$

This expression capture the probability of observing our entire dataset under the current parameter estimates. Each term in the outher sum represents the log probability of one data point, where that probabilities come from the weighted mixture of all K Gaussian components.

Now to find the optimal parameter we need to take the derivative with respect to each parameter and set it equal to zero.

Let's take the gradient with respect to μ_c

$$\frac{\partial \ell}{\partial \mu_c} = \sum_{i=1}^{n} \frac{\partial}{\partial \mu_c} \log\left(\sum_{c'=1}^{k} w_{c'} N(x_i/\mu_{c'}, \Sigma_{c'})\right) =$$

$$\sum_{i=1}^{n} \frac{1}{\sum_{c'=1}^{k} w_{c'} N(x_i/\mu_{c'}, \Sigma_{c'})} \frac{\partial}{\partial \mu_c}\left(\sum_{c'=1}^{k} w_{c'} N(x_i/\mu_{c'}, \Sigma_{c'})\right) =$$

only the c-th term in the inner term sum depend on μ_c, w_c so

$$\Rightarrow \frac{\partial}{\partial \mu_c}\left(\sum_{c'=1}^{k} w_{c'} N(x_i/\mu_{c'}, \Sigma_{c'})\right) = \frac{\partial}{\partial \mu_c} w_c N(x_i/\mu_c, \Sigma_c) = w_c \frac{\partial}{\partial \mu_c} N(x_i/\mu_c, \Sigma_c)$$

$$N(x_i/\mu_c, \Sigma_c) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_c|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(x_i-\mu_c)^T \Sigma_c^{-1}(x_i-\mu_c)\right)}$$

$$\frac{\partial}{\partial \mu_c} N(x_i/\mu_c, \Sigma_c) = -N(x_i/\mu_c, \Sigma_c)\, \Sigma_c^{-1}(x_i-\mu_c)$$

The derivative of the quadratic form $(x_i-\mu_c)^T \Sigma_c^{-1}(x_i-\mu_c)$ with respect to μ_c is $-2\Sigma_c^{-1}(x_i-\mu_c)$

Putting all together

$$\frac{\partial \ell}{\partial \mu_c} = -\sum_{i=1}^{n} \frac{w_c N(x_i/\mu_c, \Sigma_c)}{\sum_{c'=1}^{k} w_{c'} N(x_i/\mu_{c'}, \Sigma_{c'})} \Sigma_c^{-1}(x_i-\mu_c) \overset{\text{responsibility formula}}{=} -\sum_{i=1}^{n} \gamma_{c,i} \Sigma_c^{-1}(x_i-\mu_c)$$

$$\frac{\partial \ell}{\partial \mu_c} = 0 \Rightarrow \sum_{i=1}^{n} \gamma_{c,i} \Sigma_c^{-1}(x_i-\mu_c) = 0 \Rightarrow \Sigma_c^{-1} \sum_{i=1}^{n} \gamma_{c,i} x_i - \Sigma_c^{-1} \mu_c \sum_{i=1}^{n} \gamma_{c,i} =$$

$$\left(\mu_c = f(\mu_c)\right) \Rightarrow \mu_c = \frac{\Sigma_c^{-1} \sum_{i=1}^{n} \gamma_{c,i} x_i}{\Sigma_c^{-1} \sum_{i=1}^{n} \gamma_{c,i}} = \frac{\sum_{i=1}^{n} \gamma_{c,i} x_i}{\sum_{i=1}^{n} \gamma_{c,i}}$$

13/

The optimal mean for component c is simply the weighted average of all data points, where each point is weighted by is responsibility to that component. Point that are very likely to belong to component c (high $\gamma_{c,i}$) contribute strongly to determining that component's center, while point that are unlikely to belong to component c (low $\gamma_{c,i}$) have minimal influence.

This makes perfect intuitive sense. If we were trying to find the center of a cluster and we knew exactly which points belonged to it, we would simply average those points. Here, we're doing the same thing, but accounting for our uncertainty about cluster membership by using weighted averages instead of simple averages.

Notice that the responsibilities $\gamma_{c,i}$ depend on μ_c. If we knew the responsibilities we could compute μ_c as in (3)

We can interpret (3) as a weighted empirical mean. The weight of each sample is the corresponding responsibility.

The terms

$$N_c = \sum_{i=1}^{N} \gamma_{c,i}$$

and

$$\boldsymbol{F}_c = \sum_{i=1}^{N} \gamma_{c,i} \boldsymbol{x}_i$$

are also called zero and first order statistics

Note that we are summing over all samples in $\mathcal{D}$

We can adopt a similar strategy for the covariance matrix, obtaining

$$\boldsymbol{\Sigma}_c = \frac{1}{N_c} \sum_i \gamma_{c,i} (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T = \frac{1}{N_c} \sum_i \gamma_{c,i} \boldsymbol{x}_i \boldsymbol{x}_i^T - \boldsymbol{\mu}_c \boldsymbol{\mu}_c^T$$

The terms

$$\boldsymbol{S}_c = \sum_i \gamma_{c,i} \boldsymbol{x}_i \boldsymbol{x}_i^T$$

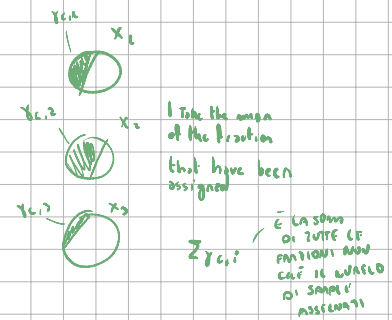
are also called second order statistics

The weights can be re-estimated as

$$w_c = \frac{N_c}{N}$$

where N is the number of samples $N = \sum_{c=1}^{K} N_c$

Nc represent the effective number of point assigned to component c.
While Fc represent the weighted sum of points assigned to component c.



Covariance Formula Intuitively

Think about what this formula is telling us. If we knew exactly which points belonged to component c, we would compute the sample covariance as the average of the outer products (xi - μc)(xi - μc)^T for all points in that component. Here, we're doing exactly the same thing, but we're weighting each point's contribution by how likely it is to belong to component c.
This makes perfect sense. A point that's very likely to belong to component c (high γc,i) should strongly influence our estimate of that component's covariance structure. A point that's unlikely to belong to component c (low γc,i) should have minimal impact on that component's covariance estimate.

Weight Formula Intuitively

This result is remarkably intuitive. The weight for component c is simply the proportion of the total "responsibility mass" that component c accounts for. Since Nc represents the effective number of points assigned to component c, and n is the total number of points, w_c = Nc/n tells us what fraction of the data is effectively explained by component c.
This makes perfect sense from a probabilistic standpoint. If component c is responsible for explaining a large portion of the data points (high Nc), it should have a correspondingly large weight in our mixture. If component c explains very few data points (low Nc), its weight should be small.

(On the slide there is an example)

Note: We didn't find a solution, we still need a way to compute $\gamma_{c,i} = \frac{w_c \mathcal{N}(x_i|\mu_c,\Sigma_c)}{\sum_{c'} w_{c'} \mathcal{N}(x_i|\mu_{c'},\Sigma_{c'})}$ so $\gamma_{c,i}$ is a $f(\mu_c, \Sigma_c)$, so we need initial parameter θ

(This not stop us we will have a starting point w.r.t. to compare with)

Think back to our GMM log-likelihood function (DIMOSTRARE PERCHÉ FUNZIONA A PICCOLE ANOMALIE AD INCREMENTARE DI VOLTA IN VOLTA LA LIKELIHOOD)

$$\ell(\theta) = \sum_{i=0}^{n} \log\left(\sum_{c=0}^{k} w_c \mathcal{N}(x_i|\mu_c, \Sigma_c)\right)$$

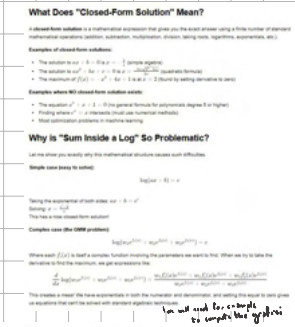
The fundamental problem lies in that inner summation inside the logarithm. When we try to derivate to find maximum, we end up with expressions that involve ratios of complicated sums, leading to equations that have no closed-form solutions. (it's not solvable using a finite number of standard mathematical operations (addition, subtraction, multiplication...))

Contrast this with what happen when we know the cluster assignments. If we knew that point x_i came from cluster c_i, then our log-likelihood would become beautifully simple:

$$\ell(\theta) = \sum_{i=0}^{n} \left[\log \mathcal{N}(x_i|\mu_{c_i}, \Sigma_{c_i}) + \log w_{c_i}\right]$$

As we said we don't have this cluster assignments - they are hidden from us

Note: What does it mean Closed-Form-Solution?



What Does "Closed-Form Solution" Mean?

A **closed-form solution** is a mathematical expression that gives you the exact answer using a finite number of standard mathematical operations (addition, subtraction, multiplication, division, taking roots, logarithms, exponentials, etc.)

Examples of closed-form solutions:

- The solution to $ax + b = 0$ is $x = -\frac{b}{a}$ (simple algebra)
- The solution to $ax^2 + bx + c = 0$ is $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ (quadratic formula)
- The maximum of $f(x) = -x^2 + 4x + 1$ is at $x = 2$ (found by setting derivative to zero)

Examples where NO closed-form solution exists:

- The equation $x^5 + x + 1 = 0$ (no general formula for polynomials degree 5 or higher)
- Finding where $e^x = x$ intersects (must use numerical methods)
- Most optimization problems in machine learning

Why is "Sum Inside a Log" So Problematic?

Let me show you exactly why this mathematical structure causes such difficulties.

Simple case (easy to solve):

$$\log(ax + b) = c$$

Taking the exponential of both sides: $ax + b = e^c$

Solving: $x = \frac{e^c - b}{a}$

This has a nice closed-form solution!

Complex case (the GMM problem):

$$\log(w_1 e^{f_1(x)} + w_2 e^{f_2(x)} + w_3 e^{f_3(x)}) = c$$

Where each $f_i(x)$ is itself a complex function involving the parameters we want to find. When we try to take the derivative to find the maximum, we get expressions like:

$$\frac{d}{dx}\log(w_1 e^{f_1(x)} + w_2 e^{f_2(x)} + w_3 e^{f_3(x)}) = \frac{w_1 f_1'(x) e^{f_1(x)} + w_2 f_2'(x) e^{f_2(x)} + w_3 f_3'(x) e^{f_3(x)}}{w_1 e^{f_1(x)} + w_2 e^{f_2(x)} + w_3 e^{f_3(x)}}$$

This creates a mess! We have exponentials in both the numerator and denominator, and setting this equal to zero gives us equations that can't be solved with standard algebraic techniques.

I'm not used to examples to compute like algebra

The EM is an iterative procedure suited for the ML estimation of the parameters of complex likelihoods[2] $f_X(x|\theta)$ that can be expressed through marginalization of joint likelihoods $f_{X,H}(x,h|\theta)$:

$$f_X(x) = \int f_{X,H}(x,h)dh = \int f_{X|H}(x|h) f_H(h) dh$$

NOTE: Here and in the following we do not make any assumption on what X represents. We simply assume that it's a random variable or a random vector, for which we observe a value x. We shall see later that, for our GMM estimation task, X represents the set of random vectors that describe the feature vectors in our dataset $\mathcal{D}$

The EM algorithm applies to any situation where we have observed data x and hidden (latent) variables H, and where the joint likelihood $f_{X,H}(x,h|\theta)$ is much easier to work with than the marginal likelihood $f_X(x|\theta)$

The derivation holds for both continuous and discrete latent variable replacing integrals

with sums.

H represents a latent (or hidden) random variable (or vector) i.e. a R.V. whose value has not been observed, i.e., is unknown

As we shall see, the EM transforms the maximization of a log-likelihood $\log f_X(x|\theta)$ into a sequence of optimizations of expectations of the joint log-likelihood $\log f_{X,H}(x,h|\theta)$

Let's conside again the marginal log-likelihood

Bayes theorem

$$l(\theta) = \log f_X(x|\theta) = \log \frac{f_{X,H}(x,h|\theta)}{f_{H|X}(h|x,\theta)}$$

Given a density $Q(h)$ with the same support of $f_H(h)$ ($Q(h)$ and $f_H(h)$ are both no zero on exactly the same domain, this prevent us to have $Q(h)>0$ in region where $f_H(h)=0$)
we can rewrite the log-pdf as

$$\log f_X(x|\theta) \cdot 1 = \log f_X(x|\theta) \int Q(h)\,dh =$$

$$= \int Q(h) \log f_X(x|\theta)\,dh = \int Q(h) \cdot \log \frac{f_{X,H}(x,h|\theta)}{f_{H|X}(h|x,\theta)} =$$

$$= \int Q(h) \log f_{X,H}(x,h|\theta)\,dh - \int Q(h) \log f_{H|X}(h|x,\theta)\,dh =$$

$$= \int Q(h) \log f_{X,H}(x,h|\theta)\,dh - \int Q(h) \log f_{H|X}(h|x,\theta)\,dh =$$

$$+ \int Q(h) \log Q(h)\,dh - \int Q(h) \log Q(h)\,dh =$$

$$= \int Q(h) \log \frac{f_{X,H}(x,h|\theta)}{Q(h)}\,dh - \int Q(h) \log \frac{f_{H|X}(h|x,\theta)}{Q(h)}\,dh$$

[$\int Q(h)\,dh = 1$, is a probability distribution, so we can use it as a weighting function]

In our case both $Q(h)$ and $f_H(h)$ represent the distribution over the cluster assignment $(1,2,\dots,n)$ so they naturally have the same support

The term:

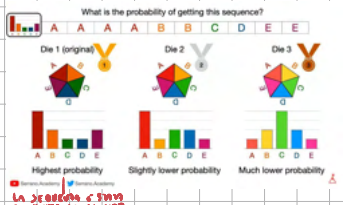
$$D_h(Q(h) \| f_{H|X}(h|x,\theta)) = -\int Q(h) \log \frac{f_{H|X}(h|x,\theta)}{Q(h)}\,dh = -E_Q\left[\log \frac{f_{H|X}(h|x,\theta)}{Q(h)}\right]$$

This term is called Kullback-Leibler (KL) divergence (usually denoted simply as $D(Q\|f_{H|X})$)
D stand for divergencies standing that this term measure di differenze between the two distribution "the distance".
The $\|$ bars mean "measured against", or "compared to", so when we write $D_h(Q(h)\|f_{H|X}(h|x,\theta))$ we are asking the question how far away is our approximation Q from the true target distribution $f_{H|X}(h,\theta)$

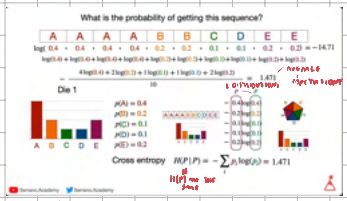
So how goal will be to minimize $D_{KL}(Q||f_{MIX})$, which means making our auxiliary distribution as close as possible to the posterior distribution $f_{MIX}(\mu|x,\sigma)$ [NON CONSIDERIAMO IL DENO QUANTI MINIMIZZARE QUESTO TERMINE CONDUCE CI PORTA A MASSIMIZZARE LA LOG-LIKELIHOOD

BRO ☝ CHIARO CHE È UNA DIFFERENZA, STAI CONSIDERANDO FUNZIONI DI DENSITÀ DI PROBABILITÀ IL CUI VALORE DI PROBABILITÀ EFFETTIVO È IL POTENZIALE INTEGRALE, HAI MESSO IL LOG DAVANTI X SEMPLICITÀ MA LA PROPRIETÀ DEI LOG SAI TU QUALE? $\log\frac{a}{b} = \log a - \log b$ = DIFFERENZA TRA DISTRIBUZIONI (TRA PROBABILITÀ DATO L'INTEGRALE). QUEL $Q(\mu)$ È IL PESO ($f_{MIX}(\mu)$) DATO AL RAPPORTO DELL'OCCORRENZA

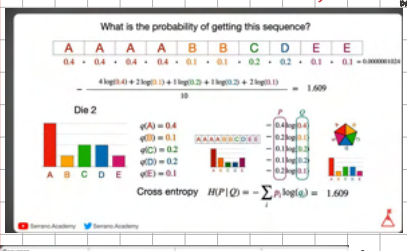
Example

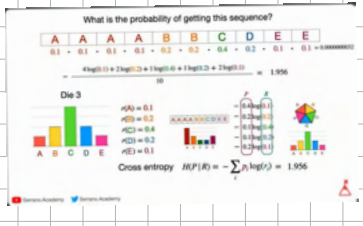


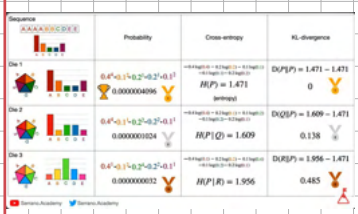
LA SEQUENZA È STATA OTTENUTA LANCIANDO QUATTRO DADI



PROBABILITÀ DI OTTENERE LA STESSA SEQUENZA CON I DIVERSI DADI







So KL divergencies is calculated as a difference of the cross-entropy is not simmetrical (we cant change the order of the term)

$Q(\mu)$ È LA DISTRIBUZIONE MASSIMA CHE VOGLIAMO TROVARE E CHE SI SPOSTA MEGLIO CON I DATI CHE OSSERVO IN QUESTO CASO LA SEQUENZA DI 3015.

What is cross-entropy??. How it concat with divergencies?

Il Concetto Fondamentale: La Sorpresa

Prima di immergerci nelle formule matematiche, dobbiamo comprendere il concetto di **sorpresa** (o "surprisal"), che è il cuore di tutto. La sorpresa misura quanto è inaspettato un evento. Pensa a quando lanci una moneta: se esce testa, non sei molto sorpreso perché sapevi che aveva il 50% di probabilità. Ma se lanci un dado e esce un sei specifico, sei più sorpreso perché aveva solo 1/6 di probabilità.

La sorpresa deve avere proprietà molto specifiche che riflettano la nostra intuizione. Primo, eventi più rari dovrebbero causare più sorpresa. Secondo, eventi certi (con probabilità 1) non dovrebbero causare alcuna sorpresa. Terzo, e questo è cruciale, la sorpresa deve essere **additiva** per eventi indipendenti.

Cosa significa "additiva"? Se lanci tre dadi simultaneamente e prevedi correttamente tutti e tre i risultati, la sorpresa totale dovrebbe essere circa tre volte la sorpresa di prevedere un singolo dado. Ma qui nasce un problema matematico interessante: mentre le sorprese si sommano, le probabilità di eventi indipendenti si moltiplicano. La probabilità di ottenere tre risultati specifici è (1/6) × (1/6) × (1/6).

Perché il Logaritmo È Essenziale

Per risolvere questo problema matematico, abbiamo bisogno di una funzione che trasformi la moltiplicazione in addizione. Questa funzione magica è il logaritmo! La sorpresa di un evento con probabilità p è definita come -log(p) o log(1/p).

Questa definizione è brillante perché soddisfa tutte le nostre esigenze. Primo, -log(p) è sempre positivo quando 0 < p < 1, perché il logaritmo di un numero minore di 1 è negativo. Secondo, diminuisce quando p aumenta, raggiungendo zero quando p = 1 (poiché log(1) = 0). Terzo, e più importante, trasforma la moltiplicazione in addizione: $\log(1/(p_1 \times p_2 \times p_3)) = \log(1/p_1) + \log(1/p_2) + \log(1/p_3)$.

L'Entropia: La Sorpresa Media

Ora che abbiamo definito la sorpresa per eventi singoli, possiamo estendere il concetto a intere distribuzioni di probabilità. L'**entropia** di una distribuzione P misura la sorpresa media che ci aspettiamo da quella distribuzione.

Matematicamente, l'entropia è:

$$H(P) = \sum_i P(x_i) \times (-\log P(x_i)) = -\sum_i P(x_i) \log P(x_i)$$

Questa formula ci dice di prendere ogni possibile risultato, moltiplicare la sua probabilità per la sua sorpresa, e sommare tutto. Il risultato ci dice quanto è "incerta" o "imprevedibile" la distribuzione. Una distribuzione uniforme (dove tutti gli esiti sono ugualmente probabili) ha alta entropia perché è molto imprevedibile. Una distribuzione dove un esito ha probabilità molto alta e gli altri molto bassa ha bassa entropia perché è molto prevedibile.

La Cross-Entropia: Quando Usiamo il Modello Sbagliato

La **cross-entropia** estende il concetto di entropia a situazioni dove abbiamo due distribuzioni diverse. Immagina di avere la vera distribuzione P, ma stai usando una distribuzione modello Q per fare previsioni. La cross-entropia misura la sorpresa media che ti aspetti quando la realtà segue P ma tu credi che segua Q.

La formula è:

$$H(P, Q) = \sum_i P(x_i) \times (-\log Q(x_i)) = -\sum_i P(x_i) \log Q(x_i)$$

Nota la differenza sottile ma cruciale: stiamo ancora pesando con le probabilità vere P(x_i), ma ora prendiamo il logaritmo delle probabilità del modello Q(x_i). Questo cattura il costo di usare la distribuzione sbagliata per le nostre previsioni.

Pensa a un esempio pratico di previsioni meteorologiche. Supponi che la vera probabilità di pioggia domani sia 70%, ma il tuo modello prevede solo 30%. La cross-entropia misura quanto costo informativo extra paghi perché le tue previsioni non corrispondono alla realtà.

La Divergenza KL: Il Ponte Tra i Concetti

Ora arriviamo alla connessione elegante che lega tutto insieme. La **divergenza di Kullback-Leibler** è definita come:

$$D_{KL}(P||Q) = H(P, Q) - H(P)$$

Questa relazione rivela che la divergenza KL misura il costo informativo extra quando usiamo la distribuzione Q invece della distribuzione ottimale P. Matematicamente:

$$D_{KL}(P||Q) = \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)} = -H(P) + H(P, Q)$$

Il primo termine, -H(P), rappresenta il costo informativo minimo possibile quando usiamo lo schema di codifica ottimale basato sulla vera distribuzione P. Il secondo termine, H(P,Q), rappresenta il costo informativo effettivo quando usiamo lo schema subottimale basato sulla distribuzione Q. La loro differenza ci dà la divergenza KL, che misura esattamente quanto costo extra sosteniamo usando la distribuzione sbagliata.

La Connessione con l'Apprendimento Automatico

Questi concetti diventano particolarmente potenti quando comprendiamo il loro ruolo nell'ottimizzazione dell'apprendimento automatico. In molti problemi di machine learning, stiamo cercando di trovare una distribuzione modello Q che approssimi il più possibile una distribuzione target P. La divergenza KL fornisce una misura naturale di quanto bene la nostra approssimazione si comporta.

Nei problemi di classificazione, la loss della cross-entropia è una delle funzioni obiettivo più comuni. Quando minimizziamo la cross-entropia tra le probabilità predette delle classi e le etichette vere delle classi, stiamo essenzialmente minimizzando la divergenza KL tra le previsioni del nostro modello e la distribuzione vera.

Questa connessione spiega perché la cross-entropia funziona così bene come funzione di loss. Non stiamo solo minimizzando una misura di distanza arbitraria; stiamo minimizzando il costo teorico-informativo di usare il nostro modello invece del predittore ottimale.

Perché Spesso Usiamo Cross-Entropia Invece di Divergenza KL

Nella pratica, il codice per l'addestramento dei modelli spesso non usa la divergenza KL come funzione di errore da minimizzare, ma piuttosto la cross-entropia. Questo accade per una ragione molto pratica e elegante.

Guardando la definizione di divergenza KL, il termine entropia di P (H(P)) non dipende dal modello Q. È una costante determinata dalla vera distribuzione P, che deriva dall'incertezza intrinseca nei dati di addestramento e non può essere modificata modificando i parametri del modello.

Pertanto, qualsiasi modello Q che minimizza la divergenza KL minimizza anche, per definizione, la cross-entropia. L'entropia dei dati H(P) non influisce su quale Q è ottimale; sposta solo il valore della divergenza KL di una quantità costante.

Poiché il calcolo esatto dell'entropia dei dati H(P) da un numero finito di campioni richiede una stima che di solito non è necessaria per l'ottimizzazione, si evita di stimarla per risparmiare computazioni. In sintesi, minimizzare la cross-entropia è equivalente a minimizzare la divergenza KL rispetto al modello Q.

The term

$$L_h(Q(h), \theta) = \int Q(h) \log \frac{f_{x,H}(x, h|\theta)}{Q(h)} dh = E_{Q(h)}[f_{x,H}(x, h|\theta)] + H(Q(h))$$

where $\mathcal{H}(Q(h))$ is the entropy of distribution $Q(h)$, provides a lower bound of the log-likelihood (again, the suffix h is usually omitted, but we keep it to remember we are integrating w.r.t. h)

$L_h(Q(h))$ is called evidence Lower Bound or ELBO

When we write

$$L_h(Q(h), \theta) = \int Q(h) \log \frac{f_{x,H}(x, h|\theta)}{Q(h)} dh$$

We can actually split this integral into two meaningful part, using the logarithm property that

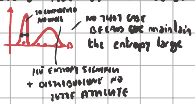
$\log\left(\frac{A}{C}\right) = \log(A) - \log(B)$

$$L_h(Q(h), \theta) = \int Q(h) \log f_{X,H}(x, h|\theta) dh - \int Q(h) \log Q(h) dh$$

The first integral, $\int Q(h) \log f_{X,H}(x,h|\theta) dh$ represent the expected value of the joint log-likelihood under the distribution $Q(h)$, This is exactly what we would want to maximize, if we knew the hidden variable - it's expectation of our complete log-likelihood function.

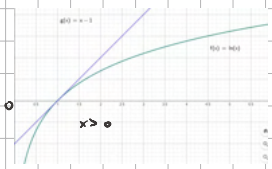
The second integral - $\int Q(h) \log Q(h) dh$ is the entropy $H(Q(h))$ of our auxiliar distribution Q Remember that entropy measures the uncertainty or randomness in a distribution. Higher entropy means the distribution Q is more spread out and uncertain about hidden variables

So ELBO as we well see, we want to find parameters θ that make the joint likelihood large (first term) but we also want to maintain high entropy in our auxiliary distribution (second term) This create a natural balance between fitting the data well and not being overly confident about our hidden variable assignments.



Why ELBO provides a valid lower bound?

The key insight relies on the fundamental inequity $\log z \leq z - 1$ for any positive number z, with equality if and only if z equals one. [$\log_2 z - z = 0 \Leftrightarrow z = 1$]



When we apply this inequality to our KL divergence, we set $z = \frac{f_{H|X}(h|x,\theta)}{Q(h)}$ and get:

$$\left[-\log \frac{f_{H|X}(h|x,\theta)}{Q(h)} \leq - \frac{f_{H|X}(h|x,\theta)}{Q(h)} + 1 \right]$$

(pointwise inequality)

with the equality holding if and only if $z = 1 \Rightarrow Q(h) = f_{H|X}(h|x,\theta)$

This inequality holds pointwise for every value of h in the support of $Q(h)$. When we integrate both sides with respect to $Q(h)$

$$-\int Q(h) \log \frac{f_{H|X}(h|x,\theta)}{Q(h)} dh \geq -\int \frac{Q(h) f_{H|X}(h|x,\theta)}{Q(h)} dh + \int Q(h) dh$$

$$\int f_{H|X}(h|x,\theta) dh = 1 \quad - \quad = 1$$

$$= D_{KL}\left(Q(h) \| f_{H|X}(h|x,\theta)\right) \geq 1 - 1 = 0$$

So, the KL divergencies is always non-negative. The equality conditions tell us also that $D_{KL}=0$ if and only if $Q(h) = f_{H|X}(h|x,\theta)$

(AVEVA INTEGRATO UNA CERTA MISURA, ANCHE PIÙ DI UN RAPPRESENTANTE CHE SI AVVÒ SEMPLICEMENTE L'UGUAGLIANZA NELLE SEMPRE ANALIZZATO DOVE PRIMA EQUIVALENTE $\log z \leq z-1$

$$D_h(Q(h)\|f_{H|X}(h|x,\theta)) = 0 \iff Q = f_{H|X} \text{ a. e.}$$

We have decomposed the log-likelihood as

$$\log f_X(x|\theta) = \mathcal{L}_h(Q(h),\theta) + D_h(Q(h)\|f_{H|X}(h|x,\theta))$$

Notice that the left side of the equation does not depend on the choice of Q at all. Furthermore since we've proven that the KL divergence is always not negative, we immediately know that

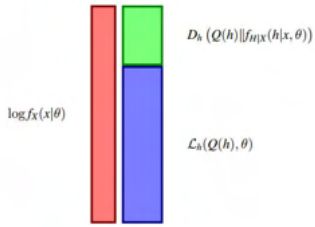
$$D_h(Q(h)\|f_{H|X}(h|x,\theta)) \geq 0 \implies \mathcal{L}_h(Q(h),\theta) \leq \log f_X(x|\theta)$$

IL DOMINIO DI QUESTO È ANCHE LA SOMMA DI 2 QUANTITÀ DI CUI È SEMPRE POSITIVA

$f_x = L + D_{KL}$ ANCHE SE NEGATIVA CONTA NEGATIVO

So we have proved that ELBO provides a lower bound that we can optimize instead of the original intractable log-likelihood

Decomposition of $\log f_X(x|\theta) = \mathcal{L}_h(Q(h),\theta) + D_h(Q(h)\|f_{H|X}(h|x,\theta))$



Our objective was to find the maximum log-likelihood we can do this directly maximizing the lower bound $L_h(Q(h),\theta)$ We can make this bound arbitrarily tight by choosing Q appropriately.

The EM algorithm optimizes the log-likelihood by iteratively

- Maximizing the lower bound $\mathcal{L}_h(Q(h),\theta)$ with respect to Q
- Maximizing the lower bound $\mathcal{L}_h(Q(h),\theta)$ with respect to θ

From an initial sets of parameters θ_0:

- $Q_0 = \arg\max_Q \mathcal{L}_h(Q(h),\theta_0)$
- $\theta_1 = \arg\max_\theta \mathcal{L}_h(Q_0(h),\theta)$
- $Q_1 = \arg\max_Q \mathcal{L}_h(Q(h),\theta_1)$
- $\theta_2 = \arg\max_\theta \mathcal{L}_h(Q_1(h),\theta)$
- ...

Think of this as a coordinated dance where each partner takes turns leading while the other follows (It's exactly what we have done until now, when we have tried to compute directly the sum of our dataset x we are just giving the mathematical interpretations

There are there are two step:

- The Expectation Step (E-step)

Given our current parameter estimates $\theta^{(t)}$ (for example μ_i, Σ_i, w_i in GM) we choose Q to make the lower bound as tight as possible. (we maximize our ELBO, setting it equal to its maximum possible value)

We have shown that

$$L_h(Q(h), \theta_t) \leq f_{H|X}(h|x, \theta_t)$$

We can maximize $L_h(Q(h), \theta t)$ with respect to Q by simply selecting

$$Q(h) = f_{H|X}(h|x, \theta_t)$$

We set our auxiliary distribution Q equal to the posterior distribution of the hidden variables given our current parameters.

But what this posterior actually represent? It answer the question: "Given my current model parameters and the data I've observed, what's the probabilities that each hidden variable takes each possible value?" In GM context, this becomes: "Given my current estimates of the cluster means, covariances, and weights, what's the probability that each datapoint belong to a cluster?"

This step is called expectation because we're computing expected values of the sufficient statistic under this posterior distribution. We're calculating things like "the expected number of points assigned to cluster 1" and "the expected number of points assigned to cluster 2," where the expectations account for our uncertainty about cluster assignments.

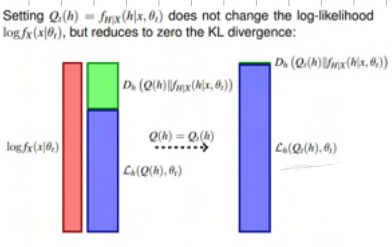
it's not the effective number but the portion instead

The KL divergence reaches its minimum value of zero when the two distributions are identical. Therefore, the optimal choice is always

$$Q_t(h) = f_{H|X}(h|x, \theta_t)$$

This makes intuitive sense - we're setting our auxiliary distribution equal to what we currently believe the true posterior distribution of the hidden variables should be. This decrease the divergence between the two distribution

This step doesn't change the log-likelihood value at all, since we're only adjusting our auxiliary distribution Q, not the actual parameters θ. But it sets us up perfectly for the M-step by ensuring that our lower bound is as tight as possible.



M-Step (Maximization with respect to Parameters) expected

Now given the distribution Q(h) we can maximize the log-likelihood computing the new model parameter θ (we actual maximize the log-likelihood following the new $Q_t(h)$ distribution

We fix our auxiliary distribution $Q_t(h)$ and find the parameters θ_{t+1} (new step) that maximize the lower bound. The optimization problem becomes:

$$\theta_{t+1} = \arg\max_{\theta} E_{Q(h)}[\log f_{X,H}(x,h|\theta)|x] = \int f_{H|X}(h|x, \theta_t) \log(f_{H,X}(h,x|\theta))\, dh$$

so $Q_t(h)$ does not depend on θ but on θ_t that is fixed.

This expectation has a beautiful interpretation. We're finding the parameters that maximize the expected complete data log-likelihood, where the expectation is take with respect to our current

best guess about the hidden variables

The expectation E_{Q^t(h)} appears because we don't know for certain which cluster each point belongs to. Instead of using hard assignments ("this point definitely belongs to cluster 1"), we use soft assignments ("this point is 70% likely to belong to cluster 1, 30% to cluster 2").

The expectation weights each possibility by its probability:

If a point is 70% likely to be chocolate, it contributes 70% of its weight to estimating chocolate properties

If a point is 30% likely to be vanilla, it contributes 30% of its weight to estimating vanilla properties

The $E_{Q_t(h)}$ operator weights each possible configuration of hidden variables by its probability under $Q_t(h)$. This creates a weighted average where more probable configurations have more influence on the parameter updates

Now we can also rewrite the problem as

$$E_{Q_t(h)}[\log f_{X|H}(x|h,\theta)] + E_{Q_t(h)}[\log f_H(h|\theta)]$$

Which corresponds exactly to what we used in GM derivation $\left(\zeta_i = \sum_{i=1}^{n} \gamma_{ij} \cdot Q_t(h)\right)$

The first term represents the expected log-likelihood of the observed data given the hidden variables and parameters. The second term represents the expected log-likelihood of the hidden variables themselves given the parameters.

Since we are maximizing $L_h(Q_t(h), \theta)$ we have:

$$L(Q_t, \theta_{t+1}) \geq L(Q_t, \theta_t)$$

But as we said earlier $L_h(Q_t, \theta_{t+1})$ is the lower bound of $L_h(Q(h), \theta_{max}) \leq f_{max}(h|x, \theta_{t+1})$

$$L_h(Q_t, \theta_t) \leq L(Q_t, \theta_{t+1}) \leq f_{H|X}(h|x, \theta_{t+1})$$

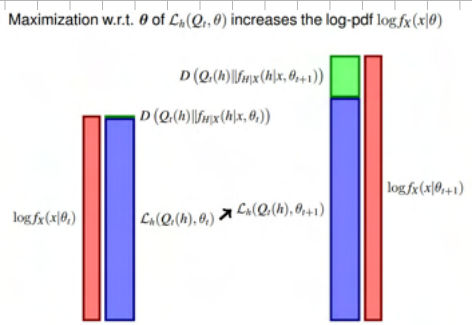
Indeed

$$\log f_X(x|\theta_{t+1}) = L(Q_t(h), \theta_{t+1}) + D(Q_t, f_{H|X}(h|x, \theta_{t+1}))$$

Maximization with respect to θ has increased both $\mathcal{L}_h$ and the KL-divergence, since, in general, $Q_t(h) \neq f_{H|X}(h|x, \theta_{t+1})$ unless we reached convergence

We thus have that

$$\log f_X(x|\theta_{t+1}) \geq \log f_X(x|\theta_t)$$



The algorithm iterates between two steps

- **Expectation (E) step:** Compute the posterior distribution $f_{H|X}(h|x, \theta_t)$ and compute the auxiliary function:

$$\mathcal{Q}(\theta, \theta_t) = \mathbb{E}_{f_{H|X}(h|x,\theta_t)}\left[\log f_{X,H}(x, h|\theta)\right]$$

stima ciò massimizzando

$Q_{\theta_t}(\theta)$

- **Maximization (M) step:** Maximize $\mathcal{Q}(\theta, \theta_t)$ w.r.t. θ to obtain

$$\theta_{t+1} = \arg\max_{\theta} \mathcal{Q}(\theta, \theta_t)$$

This create the sequence $\log f_X(x|\theta^0) \leq \log f_X(x|\theta^1) \leq \log f_X(x|\theta^2) \leq \ldots$ guaranteeing that each iteration improves our objective function or at least leaves it unchanged

The EM algorithm converges to a stationary point of the log-likelihood function under very general conditions. However, this stationary point might be a local maximum, a global (θ^*) maximum, or in rare cases, a saddle point. The algorithm cannot distinguish between these possibilities based solely on the convergence behavior.
This dependence on local optima makes initialization crucial for EM's success. Different starting points θ^0 can lead to different final solutions, and some of these solutions will be better than others. In practice, it's common to run EM multiple times with different random initializations and select the solution that achieves the highest final log-likelihood.
Think of the log-likelihood surface as a mountainous landscape with multiple peaks. EM is like a hiker who always walks uphill but can only see the immediate terrain around their current position. Depending on where the hiker starts, they might end up on a small local peak or discover the highest mountain in the region. Running multiple hiking expeditions from different starting points increases the chances of finding the tallest peak.

We have n hidden variables $h = (c_1, c_2, \ldots, c_n)$ where each c_i represents the cluster assignment for the i-th data point. Think of c_i as answering the question: "Which Gaussian component generated data point x_i?"

Each cluster assignment c_i, can take one of K possible values $[1, 2, \ldots K]$ corresponding to the K component in our mixture model, if $c_i = 3$ it means that data point x_i was generated by the third gaussian component in our mixture.

The joint-likelihood for a single sample and its cluster assignment is:

$$f_{x_i,c_i}(x_i,c_i) = w_{c_i} N(x_i|\mu_{c_i},\Sigma_{c_i})$$

The term w_c represents the prior probability that any randomly selected point belongs to cluster c. ($P(C_i=c)=w_c$) The term $N(x_i|\mu_c,\Sigma_c) = f_{x_i|c_i}(x_i|c)$ represent the probability of observing data point x_i given that it come from cluster c. Together they give the joint probability of observing both, the data point and its cluster assignment.

Think about this intuitively. If you randomly select a person from a mixed population of children and adults, wc tells you the probability they're from group c (children or adults), and $N(x_i|\mu c, \Sigma c)$ tells you the probability of observing their specific height and weight given their group membership.

We assume that samples are independent given the model parameters, so that we can express the joint log-likelihood for all the training set samples as:

$$\log f_{x_0,\dots x_N,c_1,\dots c_N}(x_0,\dots x_N,c_1,\dots c_N|\theta) = \sum_{i=0}^{N} \log f_{x_i,c_i}(x_i,c_i|\theta)$$

The EM algorithm requires computing the posterior for the hidden variables $C_1 \dots C_N | X_1 \dots X_N, \theta$.

Due to the independence assumptions, also the posterior distribution factorizes as

$$f_{C_1,\dots C_N|X_1,\dots X_N}(c_1 \dots c_N|x_1 \dots x_N,\theta) = \prod_{i=1}^{N} P(C_i = c_i|X_i = x_i,\theta)$$

The cluster assignment for one data point doesn't depend on the cluster assignments of other data points, given the model parameters.

We begin with the E-Step, that requires to compute the auxiliary function:

$$Q(\theta,\theta_t) = E_{c_0,\dots c_N|x_0,\dots x_N,\theta_t}[\log f_{x_0,\dots x_N,c_1,\dots c_N}(x_0,\dots x_N,c_1,\dots c_N|\theta)] =$$

$$= \sum_{i=0}^{N} E_{c_i|x_i=x_i,\theta_t}[\log f_{x_i,c_i}(x_i,\overbrace{c}^{c_i=?}|\theta)]$$

The expectation here is taken over all possible cluster assignments for each data point, weighted by their posterior probabilities under the current parameter estimates. This is where the famous "responsibilities" $\gamma c,i = P(Ci = c|Xi = xi, \theta^t)$ come into play.

Think of $\gamma c,i$ as answering the question: "Given the current model parameters and the observed data point xi, what's the probability that this point belongs to cluster c?" These responsibilities capture our uncertainty about cluster membership in a probabilistic way.

For example, if $\gamma_{1,i} = 0.7$ and $\gamma_{2,i} = 0.3$ for some data point i, it means we believe there's a 70% chance this point belongs to cluster 1 and a 30% chance it belongs to cluster 2. This soft assignment allows us to work with uncertainty rather than making premature hard decisions about cluster membership.

E-step: Compute $\gamma_{c,i} = P(C_i = c|\boldsymbol{X}_i = \boldsymbol{x}_i, \boldsymbol{\theta}^t)$. The auxiliary function is

$$\begin{aligned}
\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}_t) &= \sum_{i=1}^{N}\sum_{c=1}^{K} P(C_i = c|\boldsymbol{X}_i = \boldsymbol{x}_i, \boldsymbol{\theta}_t) \log f_{\boldsymbol{X}_i, C_i}(\boldsymbol{x}_i, c|\boldsymbol{\theta}) \\
&= \sum_{i=1}^{N}\sum_{c=1}^{K} \gamma_{c,i} \left[\log \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) + \gamma_{c,i} \log w_c\right] \\
&= \sum_{i=1}^{N}\sum_{c=1}^{K} \gamma_{c,i} \left(\frac{1}{2}\log|\boldsymbol{\Lambda}_c| - \frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T \boldsymbol{\Lambda}_c (\boldsymbol{x}_i - \boldsymbol{\mu}_c)\right) \\
&\quad + \sum_{i=1}^{N}\sum_{c=1}^{K} \gamma_{c,i} \log w_c
\end{aligned}$$

*For each data point i and cluster c, we're weighting the log-probability terms by how likely the point is to belong to that cluster. Point that are really likely to belong to that cluster contribute strongly to that cluster's parameter estimation, while point that are unlikely to belong to a cluster contribute weakly.

Note:

$\log(\mathcal{N}(x_i|\mu_c, \Sigma_c)) = -\frac{1}{2}\log|2\pi\Sigma_c| - \frac{1}{2}(x_i - \mu_c)^T \Sigma_c^{-1}(x_i - \mu_c)$

The precision matrix $\Lambda_c = \Sigma_c^{-1}$

$\log(\mathcal{N}(x_i|\mu_c, \Sigma_c)) = \frac{1}{2}\log|\Lambda_c| - \frac{1}{2}(x_i - \mu_c)^T \Lambda_c (x_i - \mu_c) + \text{constant}$

We find the parameters that maximize the auxiliary function

M-step: Maximize $\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}_t)$ w.r.t. $\boldsymbol{\theta} = (\boldsymbol{M}, \boldsymbol{\mathcal{S}}, \boldsymbol{w})$, subject to $\sum_{k=1}^{K} w_k = 1$:

$$\begin{aligned}
\boldsymbol{\mu}_c^* &= \frac{\sum_i \gamma_{c,i} \boldsymbol{x}_i}{\sum_i \gamma_{c,i}} \\
\boldsymbol{\Sigma}_c^* &= \frac{\sum_i \gamma_{c,i} (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T}{\sum_i \gamma_{c,i}} \\
w_c^* &= \frac{\sum_i \gamma_{c,i}}{\sum_i \sum_c \gamma_{c,i}}
\end{aligned}$$

The new estimate of the parameters is $\theta_{t+1} = (\boldsymbol{M}_{t+1}, \boldsymbol{\mathcal{S}}_{t+1}, w_{t+1})$:

$$\boldsymbol{M}_{t+1} = [\boldsymbol{\mu}_1^* \dots \boldsymbol{\mu}_K^*] \quad , \quad \boldsymbol{\mathcal{S}}_{t+1} = [\boldsymbol{\Sigma}_1^* \dots \boldsymbol{\Sigma}_K^*] \quad , \quad \boldsymbol{w}_{t+1} = [w_1^* \dots w_K^*]$$

Updating the Means: The optimal mean for cluster c is:

$$\mu_c^* = \frac{\sum_i \gamma_{c,i} x_i}{\sum_i \gamma_{c,i}}$$

This is a weighted average of all data points, where each point is weighted by its responsibility to cluster c. Think about how intuitive this is: points that are very likely to belong to cluster c (high $\gamma_{c,i}$) have a strong influence on where we place the cluster center, while points that are unlikely to belong to cluster c (low $\gamma_{c,i}$) have minimal influence.

Updating the Covariances: The optimal covariance for cluster c is:

$$\Sigma_c^* = \frac{\sum_i \gamma_{c,i}(x_i - \mu_c)(x_i - \mu_c)^T}{\sum_i \gamma_{c,i}}$$

This is a weighted sample covariance matrix. Each data point contributes to the covariance estimate in proportion to how likely it is to belong to that cluster. Points that are probably in the cluster strongly influence the cluster's shape and orientation, while points that probably aren't in the cluster have minimal impact.

Updating the Mixture Weights: The optimal weight for cluster c is:

$$w_c^* = \frac{\sum_i \gamma_{c,i}}{\sum_i \sum_c \gamma_{c,i}} = \frac{\sum_i \gamma_{c,i}}{N}$$

This represents the proportion of the data that is effectively assigned to cluster c. The numerator $\Sigma_i \gamma_{c,i}$ can be interpreted as the "effective number of points" assigned to cluster c. Since the responsibilities for each point sum to 1 across all clusters, the denominator equals N (the total number of data points).

The EM algorithm for GMMs creates a beautiful iterative refinement process. In each E-step, we update our beliefs about which cluster each point belongs to based on the current parameter estimates. In each M-step, we update our parameter estimates based on the current cluster assignment beliefs.

This creates a feedback loop where better cluster assignments lead to better parameter estimates, which in turn lead to better cluster assignments. The mathematical guarantee is that each iteration improves the log-likelihood, ensuring that we make steady progress toward a solution that at least locally optimizes our objective function.

The algorithm continues until convergence, which typically occurs when the parameter changes between iterations become negligibly small, or when the log-likelihood improvement falls below a specified threshold. At convergence, we have found parameter estimates that represent at least a local maximum of the likelihood function, giving us a GMM that provides a good probabilistic model of our data

From nve to GNN classification

Traditional nve classification makes a strong assumption that might not hold in practice. For each class c, it assumes that all data point follow a single Gaussian distribution: $X_i | C_i = c \sim N(\mu_c, \Sigma_c)$, This work well when each class has a roughly elliptical unimodal distribution, but real world data often violates this assumption dramatically.

MVG for classification:

- Fit a guassian density to samples of each class

$$X_i|C_i = c \sim \mathcal{N}(\mu_c, \Sigma_c) \quad \text{ONLY ONE GAUSSIAN}$$
$$f_{X|C}(x_t|c) = \mathcal{N}(x_t|\mu_c, \Sigma_c)$$
$$P(C_t = c|X_t = x_t) = \frac{\mathcal{N}(x_t|\mu_c, \Sigma_c)P(C_t = c)}{\sum_{c'} \mathcal{N}(x_t|\mu_{c'}, \Sigma_{c'})P(C_t = c')}$$

This is where GMM classification becomes powerfull, instead of forcing each class to follow a single gaussian distribution, we allow each class to be modeled by a mixture of multiple Gaussian components. Each component can capture different "substyle" or "subpattern" within the class.

In GMM classification we model each class c using its own GMM with parameters $(\pi_c, \mu_c, \Sigma_c, w_c)$

$$x_t|C_t = c \sim GMM(\pi_c, \mu_c, \Sigma_c, w_c)$$

The class conditional distribution:

$$f_{x_t|C_t} = \sum_{m=1}^{M_c} w_m \mathcal{N}(x_t|\mu_{c,m}, \Sigma_{c,m})$$

Each class can have a different number of components M_c.
Each class has his own complete set of parameters

The posterior probability

$$P(C_t = c|X_t = x_t) = \frac{P(C_t = c)\sum_{k=1}^{K_c} w_{c,k}\mathcal{N}(x_t|\mu_{c,k}, \Sigma_{c,k})}{\sum_{c'} P(C_t = c')\sum_{k=1}^{K_{c'}} w_{c',k}\mathcal{N}(x_t|\mu_{c',k}, \Sigma_{c',k})}$$

This formula elegantly combines the prior probability of each class with the GMM-based likelihood. Each class's likelihood is computed as a weighted combination of multiple Gaussian components, allowing for much more flexible modeling of within-class variation.

Open-set Classification

We can also exploit GMMs clustering capabilities for open-set multiclass classification

The difficulty in open-set classification consists in building a robust model for the none-of-the-others class

This class is usually very heterogeneous, and explicit modeling its sub-components requires labeled examples of its possible objects

We can partially alleviate the issue using a GMM.

We can assume that samples of known classes can be modeled by MVG distributions

We can collect a large set of unlabeled samples for the none-of-the-others class

We can model the samples of the none-of-the-others class using a GMM

The GMM will find homogeneous clusters (sub-classes) of the none-of-the-others population, and can be used as an estimate for the conditional density of a test sample assuming that it belongs to the none-of-the-others class[4]

[4]Given the complexity of the task, and due to the fact that different amounts of parameters are used for modeling the none-of-the-others class these kind of models often provide class-conditional likelihoods that are not calibrated, and may require further score processing (e.g. score calibration) to obtain good decisions

59

GMMs offer an elegant solution to this modeling challenge. We can collect a large set of unlabeled samples that we believe represent the none-of-the-others class and use EM to fit a GMM with many components. The algorithm will automatically discover homogeneous sub-clusters within this heterogeneous class, effectively learning a representation of the different types of "unknown" samples.

As we did for MVG, we can train GMM with diagonal covariance matrices to reduce the number of parameters to estimate (reducing overfitting and computational costs).

$C = \begin{bmatrix} \cdot & 0 \\ 0 & \cdot \end{bmatrix}$ no correlation

The solution is again given by the diagonals of Σ_c^*'s we defined before

We may need more components to model more complex distributions

The diagonal covariance assumption DOES NOT correspond to the Naive Bayes assumption in this case

The Naive Bayes assumption would correspond to training a different GMM (possibly with different number of components) for each subset of features that is assumed independent from the other features

We can also assume that all components of a GMM have the same covariance matrix (tied GMM)

Note that in this case we are tying the components of a single GMM, i.e. the Gaussian components of the GMM of a single class

This is different from the Tied Gaussian model, where parameters were shared across classes

Of course, we can extend GMM parameters tying across classes as well — we won't consider this model though, as it would require revisiting the EM estimation procedure, since sharing the parameters across classes would not allow us to independently estimate a GMM over the samples of each class

MNIST — GMM (PCA 50)

Components:	1	2	4	8	16
FullCov	3.6%	3.4%	2.8%	2.3%	2.2%
Diagonal	12.3%	10.1%	8.9%	7.6%	6.2%

Components:	32	64	128	256
FullCov	2.3%			
Diagonal	5.1%	4.3%	4.3%	4.3%

Performance Insights from Real Data

The MNIST results shown in your slides reveal important patterns about GMM classification performance. Several key insights emerge from this data:

The Power of Multiple Components: Moving from 1 to 16 components dramatically improves performance for both full and diagonal covariance models. With full covariances, error rates drop from 3.6% to 2.2%. This demonstrates that even relatively simple datasets like MNIST benefit significantly from multi-component modeling.

Covariance Structure Matters: Full covariance matrices consistently outperform diagonal ones, but the gap narrows as the number of components increases. With 1 component, full covariances achieve 3.6% error versus 12.3% for diagonal. But with 16 components, the gap closes to 2.2% versus 6.2%. This suggests that multiple components can partially compensate for the limitations of diagonal covariances.

Diminishing Returns: Performance improvements slow down as we add more components. The jump from 1 to 2 components provides substantial gains, but going from 128 to 256 components shows minimal improvement. This reflects the bias-variance trade-off: more components provide greater flexibility but require more data to estimate reliably.

Initialization problem

$\Sigma_c = I$ μ_c $w_c = \frac{1}{k}$ initial parameters k-means

Unlike simple optimization problems with convex objective functions, GMM training faces the challenge of a highly non-convex likelihood surface riddled with local maxima. The EM algorithm is guaranteed to find a local maximum, but there's no guarantee it will find the global maximum or even a good local maximum. This makes initialization absolutely critical.
Think of the likelihood surface as a mountainous landscape with many peaks of different heights. The EM algorithm is like a hiker who always walks uphill but can only see their immediate surroundings. Where you start your hike largely determines which peak you'll reach. A poor starting point might lead you to a small hill when there's a towering mountain just a few valleys away.
The Random Initialization Problem:
Random initialization often leads to poor results because it doesn't provide the algorithm with any guidance about the underlying structure in the data. You might initialize components in empty regions of the feature space, or multiple components might start so close together that they compete for the same data points throughout training.
K-means as a Natural Initializer:
The connection between GMMs and K-means provides an elegant initialization strategy. When we constrain GMMs to have isotropic (spherical) covariances and make hard cluster assignments, we essentially recover the K-means algorithm. This connection suggests using K-means clustering results to initialize GMM parameters.
The K-means initialization process works as follows:

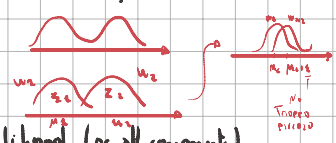
Run K-means clustering on your data to get initial cluster centers
- Initialize GMM means μc to the K-means centroids
- Initialize covariances Σc based on the sample covariances of the K-means clusters
- Initialize mixture weights wc based on the relative sizes of the K-means clusters

This approach ensures that each GMM component starts in a region where there's actual data, and the components are reasonably well-separated from each other.

An alternative approach is the LBG Algorithm

In the LBG instead of jumping directly to your target number of component you start symply and progressive split component to increase model complexity

1. Start with a single Gaussian component fitted to all your data
2. Split this component int two by creating a small perturbation: $\mu_c^+ = \mu_c + \epsilon$ and $\mu_c^- = \mu_c - \epsilon$
3. Run EM to converge on this 2-component model
4. Split the component that contributes most to the likelihood (or all components) to get 4 components
5. Continue until you reach your desired number of component.



A good value for ϵ can be a displacement along the principal eigenvector of the covariance matrix Σ_c

We have sample $\vec{x}$ with a lot of feature (so μ is a vector with the mean for each feature)

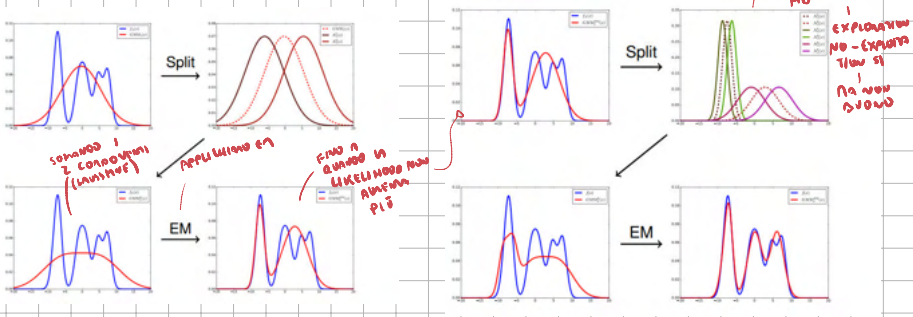
The eigenvectors of a covariance matrix tell us the principal direction of variation in the data, while the eigenvalues tell us how much variation exists along each direction.

So when we have $\Sigma = V \Lambda V^T$

eigenvector as column

the first eigenvector that corrispondo to the highest eigenvalue points in the direction where the data variate the most. This is the direction where it makes most sense to split a gaussian component into two sub-components. $\varepsilon = \lambda \cdot u_1$

magnitude of the split



One of the most difficult practical problems in GMM training is determining the appropriate number of components. This is particularly challenging because the likelihood always increases (or at least doesn't decrease) as you add more components, making it impossible to use likelihood alone for model selection. (OVER FITTING)

Why Likelihood Fails as a Selection Criterion:

Adding more components always provides the model with greater flexibility to fit the training data. In the extreme case, you could place one tiny Gaussian component on each training point and achieve arbitrarily high likelihood. But such a model would be useless for generalization because it would essentially memorize the training data rather than learning underlying patterns.

We cannot choose based on likelihood alone

Several criteria, more or less successful, have been proposed (AIC, BIC)

We can also resort to cross-validation USIAMO QUESTA

We also need to pay attention to degenerate models

As we said at the beginning, the log-likelihood for a GMM, as long as we have at least two components, is unbounded

The EM algorithm will usually find local maxima that are well-behaved, however, especially if we have too many components, we may obtain degenerate models which cause numerical issues

Some heuristics can be used to force models to be well-behaved (e.g. impose minimum values for the eigenvalues of the covariance matrices, tie the covariance of different components)

We can also modify our initialization so that the algorithm may end up in a different local maximum

Note: Degenerate solution typically occur when one or more components become pathologically concentrated on a small number of data points. As discussed earlier, this can lead to covariance matrix with very small eigenvalues, causing numerical instability and unbounded likelihood values