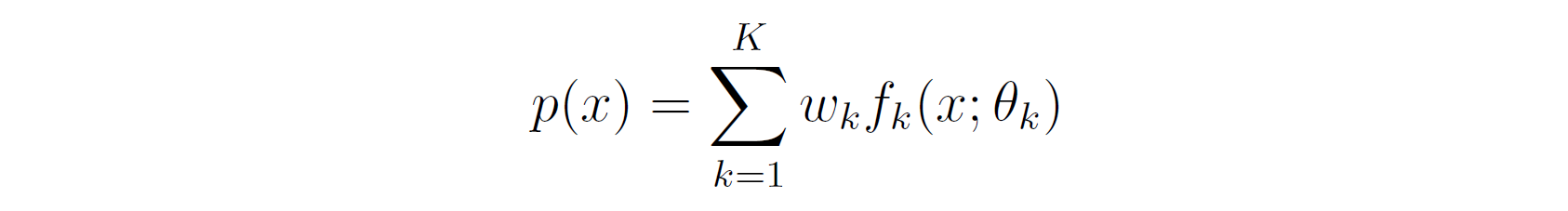
**Gaussian Mixture Model**

A Gaussian mixture model is a soft clustering technique used in unsupervised learning to determine the probability that a given data point belongs to a cluster. It’s composed of several Gaussians, each identified by *k* ∈ {1,…, *K*}, where *K* is the number of clusters in a data set.

Gaussian Mixture Models (GMMs) are statistical models that represent the data as a mixture of Gaussian (normal) distributions. These models can be used to identify groups within the dataset, and to capture the complex, multi-modal structure of data distributions.

GMMs are used in a variety of machine learning applications, including clustering, density estimation, and pattern recognition.

Formally, if *X* is a random variable whose distribution is a mixture of *K* component distributions, the probability density function (PDF) or probability mass function (PMF) of *X* can be written as:

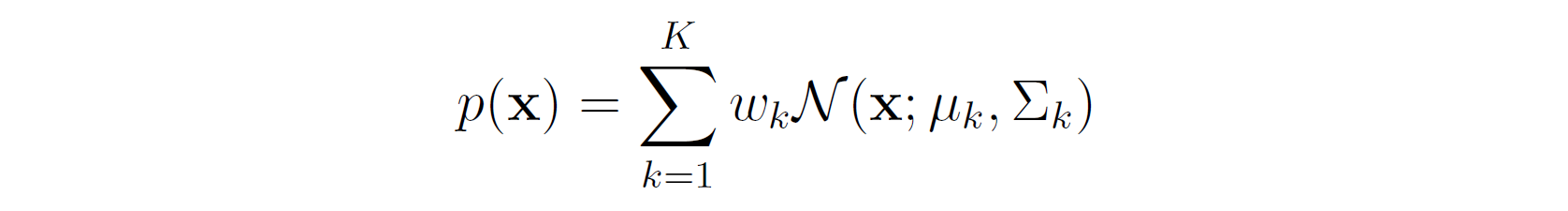


where:

* *p*(*x*) is the overall density or mass function of the mixture model.
* *K* is the number of component distributions in the mixture.
* *fₖ*(*x*; θk) is the density or mass function of the *k*-th component distribution, parametrized by θk.
* *wₖ* is the mixing weight of the *k*-th component, with 0 ≤ *wₖ* ≤ 1 and the sum of the weights being 1. *wₖ* is also known as the prior probability of component *k*.
* θk represents the parameters of the *k*-th component, such as the mean and standard deviation in the case of Gaussian distributions.

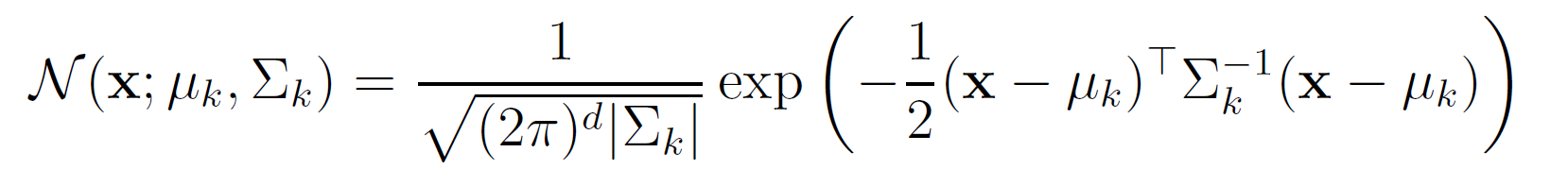
The mixture model assumes that each data point comes from one of the *K* component distributions, with the specific distribution being selected according to the mixing weights *wₖ*. The model does not require knowing which component each data point belongs to.

A Gaussian mixture model (GMM) is a common mixture model, where the probability density is given by a mixture of Gaussian distributions:



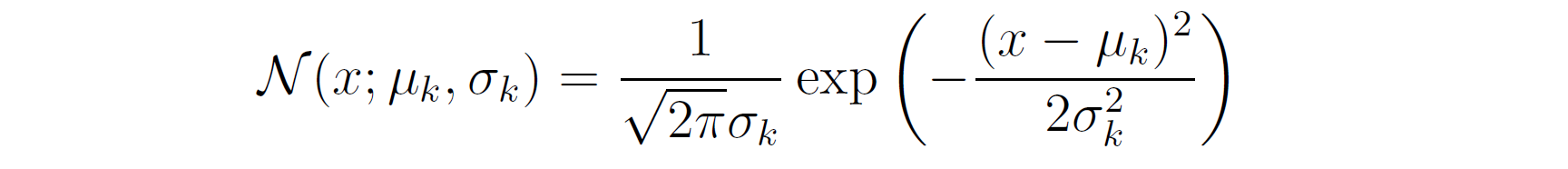
where:

* x is a *d*-dimensional vector.
* *μₖ* is the mean vector of the *k*-th Gaussian component.
* Σ*ₖ* is the covariance matrix of the *k*-th Gaussian component.
* *N*(x; *μₖ*, Σ*ₖ*) is the multivariate normal density function for the *k*-th component:



where:

* *μₖ* is the mean of the *k*-th Gaussian component.
* *σₖ* is the covariance matrix of the *k*-th Gaussian component.
* *N*(*x*; *μₖ*, *σₖ*) is the univariate normal density function for the *k*-th component:



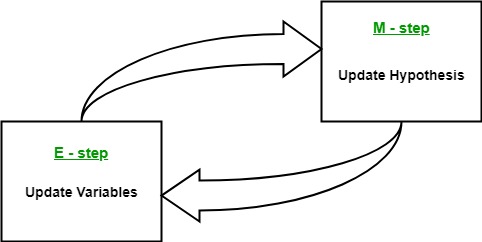
**What is Expectation-Maximization (EM) algorithm?**

A latent variable model consists of observable variables along with unobservable variables. Observed variables are those variables in the dataset that can be measured whereas unobserved (latent/hidden) variables are inferred from the observed variables.

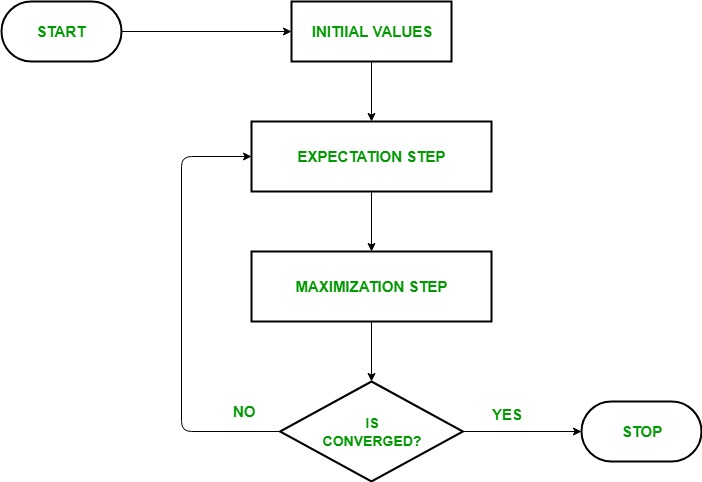
* It can be used to find the local maximum likelihood (MLE) parameters or maximum a posteriori (MAP) parameter for latent variables in a statistical or mathematical model.
* It is used to predict these missing values in the dataset, provided we know the general form of probability distribution associated with these latent variables.
* In simple words, the basic idea behind this algorithm is to use the observable samples of latent variables to predict the values of samples that are unobservable for learning. This process is repeated until the convergence of the values occurs.

**Detailed Explanation of the EM Algorithm**

* Given a set of incomplete data, start with a set of initialized parameters.
* **Expectation step (E – step):** Given the current parameter estimates (θ), this step calculates the expected value of the complete-data log-likelihood function, treating the missing data as random variables. Essentially, it estimates the missing data based on the current model parameters. Finally, after this step, we get complete data having no missing values.
* **Maximization step (M – step):** Now, we have to use the complete data, which is prepared in the expectation step. Given the estimated missing data (from the E-step), this step updates the model parameters (θ) (i.e, update the hypothesis) to maximize the expected complete-data log-likelihood. This step essentially finds the parameter values that best explain the observed data, along with the estimated missing data.
* **Checking of convergence Step:** Now, in this step, we checked whether the values are converging or not. The E-step and M-step are repeated **iteratively**. In each iteration, the parameter estimates are refined, and the estimates of the missing data are updated. This process continues until the parameter estimates converge (until the convergence occurs).



**Flow chart for EM algorithm**



Advantages and Disadvantages of EM algorithm

**Advantages**

* The basic two steps of the EM algorithm i.e, E-step and M-step are often pretty easy for many of the machine learning problems in terms of implementation.
* The solution to the M-steps often exists in the closed-form.
* It is always guaranteed that the value of likelihood will increase after each iteration.

**Disadvantages**

* It has slow convergence.
* It converges to the local optimum only.
* It takes both forward and backward probabilities into account. This thing is in contrast to that of numerical optimization which considers only forward probabilities.

**Applications of EM Algorithm**

The latent variable model has several real-life applications in Machine learning:

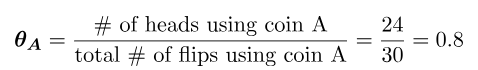
* **Gaussian Mixture Models (GMMs):** Used for mixed models like Gaussian Mixture Models to clustering data points into groups based on Gaussian distributions.
* **Hidden Markov Models (HMMs):** Used for estimating the parameters of the Hidden Markov Model (HMM). Modeling sequences of data where the underlying state is hidden.
* **Image segmentation:** Clustering pixels in an image based on their color and other features.
* **Bioinformatics:** Analyzing gene expression data and protein structures.
* Used to calculate the Gaussian density of a function.
* Helpful to fill in the missing data during a sample.
* It finds plenty of use in different domains such as Natural Language Processing (NLP), Computer Vision, etc.
* Used in image reconstruction in the field of Medicine and Structural Engineering.

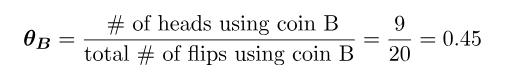
**Intuitive Coin Flipping Example**

Let’s first look at a simple coin-flipping example to develop some intuition. We have two coins which we’ll name A and B. Each has some unknown probability of getting heads  and . Suppose that we want to find what these values are. We can experimentally find them by gathering data. So, we take a coin randomly and flip it ten times. And then let’s say we repeat that procedure five times, amounting to 50-coin tosses worth of data:

| **Coin** | **Flips** | **# coin A heads** | **# coin B heads** |
| --- | --- | --- | --- |
| B | HTTTHHTHTH | 0 | 5 |
| A | HHHHTHHHHH | 9 | 0 |
| A | HTHHHHHTHH | 8 | 0 |
| B | HTHTTTHHTT | 0 | 4 |
| A | THHHTHHHTH | 7 | 0 |

Since we denoted each time which coin we were using to take the series of flips, answering would be pretty easy – it’s just the proportion of heads relative to the number of tosses with each coin:





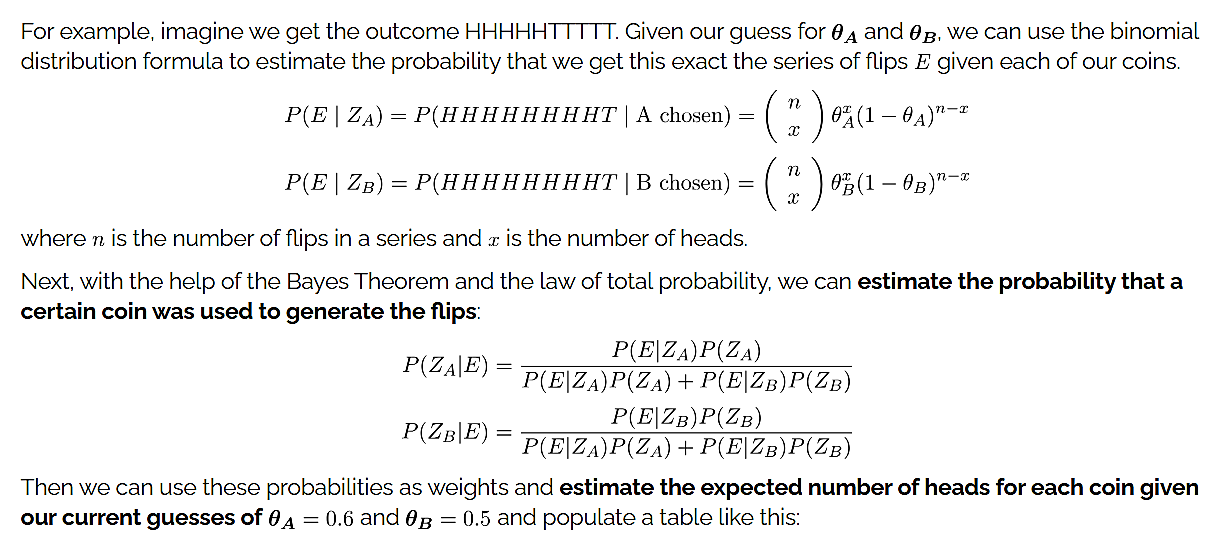
This intuitive solution can be proven to be the maximum likelihood estimate by analytically solving the equations needed to maximize the likelihood function of a Binomial model of the coins here. But what if somehow, we lost the data about the identity of each coin we were using throughout the experiment? We’ll end up with data that is incomplete and has one latent variable, which we’ll call  – the identity of the coin used.

**4. Expectation-Maximization as a Solution**

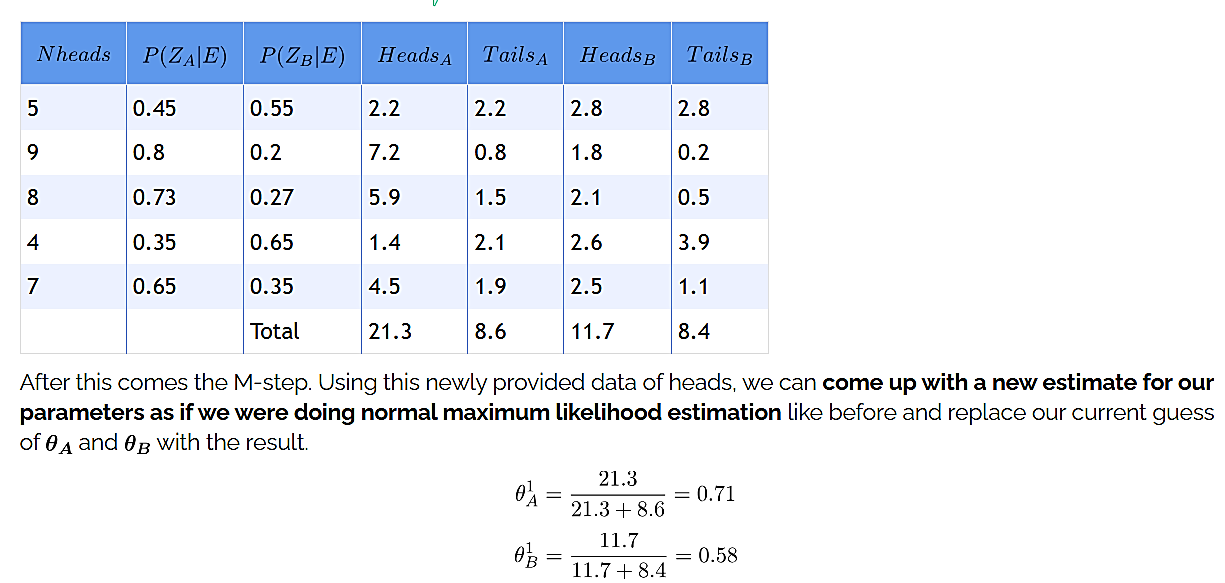
Even though the incomplete information makes things hard for us, the Expectation-Maximization can help us come up with an answer. The technique consists of two steps – the E(Expectation)-step and the M(Maximization)-step, which are repeated multiple times.

Lets’ look at the E-step first. You could say that this part is significantly related to having an educated guess about the missing data. So in our example, can we guess the coin that was used to generate the flips? Well, yes, but it won’t always be the most educated guess, so what can we do?

One thing that would be much better is not to force ourselves to assume one coin or the other, but instead, **estimate the probability that each coin is the true coin given the flips we see in the trial**. We can then use that to proportionally assign heads and tails counts to each coin. There is a small detail though. In order to estimate that, we would need to know  and . So let’s throw a wild guess at them instead, say  and  and give it a shot.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 0.45 | 0.55 | 2.2 | 2.2 | 2.8 | 2.8 |
| 9 | 0.8 | 0.2 | 7.2 | 0.8 | 1.8 | 0.2 |
| 8 | 0.73 | 0.27 | 5.9 | 1.5 | 2.1 | 0.5 |
| 4 | 0.35 | 0.65 | 1.4 | 2.1 | 2.6 | 3.9 |
| 7 | 0.65 | 0.35 | 4.5 | 1.9 | 2.5 | 1.1 |
|  |  | Total | 21.3 | 8.6 | 11.7 | 8.4 |



These values are indeed closer to the maximum likelihood estimate from earlier. So why not use them as an input and repeat the Expectation and Maximization steps again? Nothing is stopping us. And indeed, if we do this after a few iterations of this process, the values will converge, and we’ll get an answer.

