4. Latent Variable Models and EM

Gholamreza Haffari

4. Latent Variable Models and EM

Gholamreza Haffari

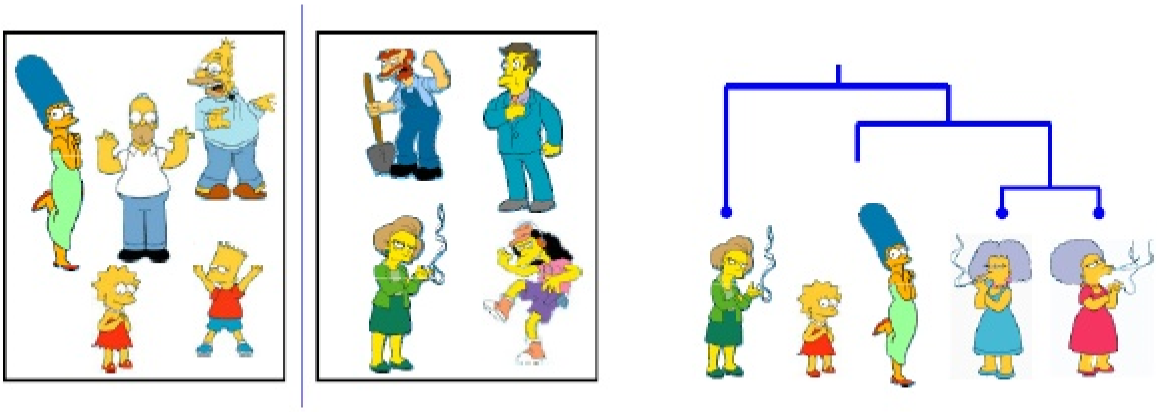
Contents

1
Clustering and Kmeans

Data Clustering

Data clustering or cluster analysis is one of the most common forms of unsupervised learning that aims to find a sensible *structure* from unlabeled data. In other words, a clustering algorithm groups data into their natural categories based on the similarities between them without knowing their actual groups. Clustering algorithms help data scientists to do several tasks such as revealing the structure of the data, which has many applications in hypothesis generation and anomaly detection, categorizing data when labeling (e.g., by human experts) are costly, and compressing data (only storing the cluster representatives/prototypes or the parameters of the generative models).

Some of the challenges in data clustering are scalability to big data, the capability to handle different types of data with a variety of distributions and insensitivity to noise and outliers. So far, heaps of clustering algorithms are developed by statistician and computer scientists. Center-based (http://Center-based) partitioning (e.g., *KMeans*), density-based (https://en.wikipedia.org/wiki/Cluster_analysis#Density-based_clustering) clustering (e.g., *DBSCAN*), hierarchical (https://en.wikipedia.org/wiki/Hierarchical_clustering) clustering, and graph-based (http://www.slideshare.net/ssakpi/graph-based-clustering) clustering are just a few examples of popular clustering approaches. Figure **4.1.1.** compares the partitioning clustering (the left side) versus the hierarchical clustering (the right side).



(http://www.slideshare.net/aorriols/lecture17)

Figure 4.1.1. Partitioning (left) vs. hierarchical clustering (right). Source: http://www.slideshare.net/aorriols/lecture17

In this chapter, we focus on the *KMeans* algorithm not only because of its simplicity, but also because it provides us some useful anchors to important concepts and techniques that will be discussed in the next chapters.

The KMeans Algorithm

Suppose N unlabeled data points $\boldsymbol{x}_n$ are given, and the goal is to partition them into K distinct groups (or clusters) such that *similar* points are grouped together. The similarity is measured usually by the inverse of a *distance* measure $d(.)$ e.g., Euclidean distance. The simplest center-based algorithms to solve the clustering problem is *KMeans*.

KMeans, is an iterative algorithm that starts with initial *random* guesses of K cluster centers $(\boldsymbol{\mu}_1^{(0)}, \ldots, \boldsymbol{\mu}_K^{(0)})$ and continues by iteratively executing the following two steps until a stopping criterion is met:

1. **Update Assignment of Datapoints to Clusters:** Calculate the distance of each data point $\boldsymbol{x}_n$ to all cluster centers, and assign the datapoint to the cluster with the minimum distance.
2. **Update Centers of the Clusters:** For each cluster, calculate the new center as the average of all datapoints assigned to it. More formally,

$$\boldsymbol{\mu}_k^{(\tau+1)} = \frac{\sum_n r_{nk}\boldsymbol{x}_n}{\sum_n r_{nk}}$$

   where r_{nk} indicates whether $\boldsymbol{x}_n$ is assigned to k cluster:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j d(\boldsymbol{x}_n, \boldsymbol{\mu}_j^{(\tau)}) \\ 0 & \text{otherwise.} \end{cases}$$

Based on the above description,in one step *KMeans* assigns data points to their closest centres, and in the consecutive step it updates the each cluster mean by simply averaging over all the datapoints that are currently assigned to that cluster. Figure **4.1.2** illustrates the *KMeans* steps.

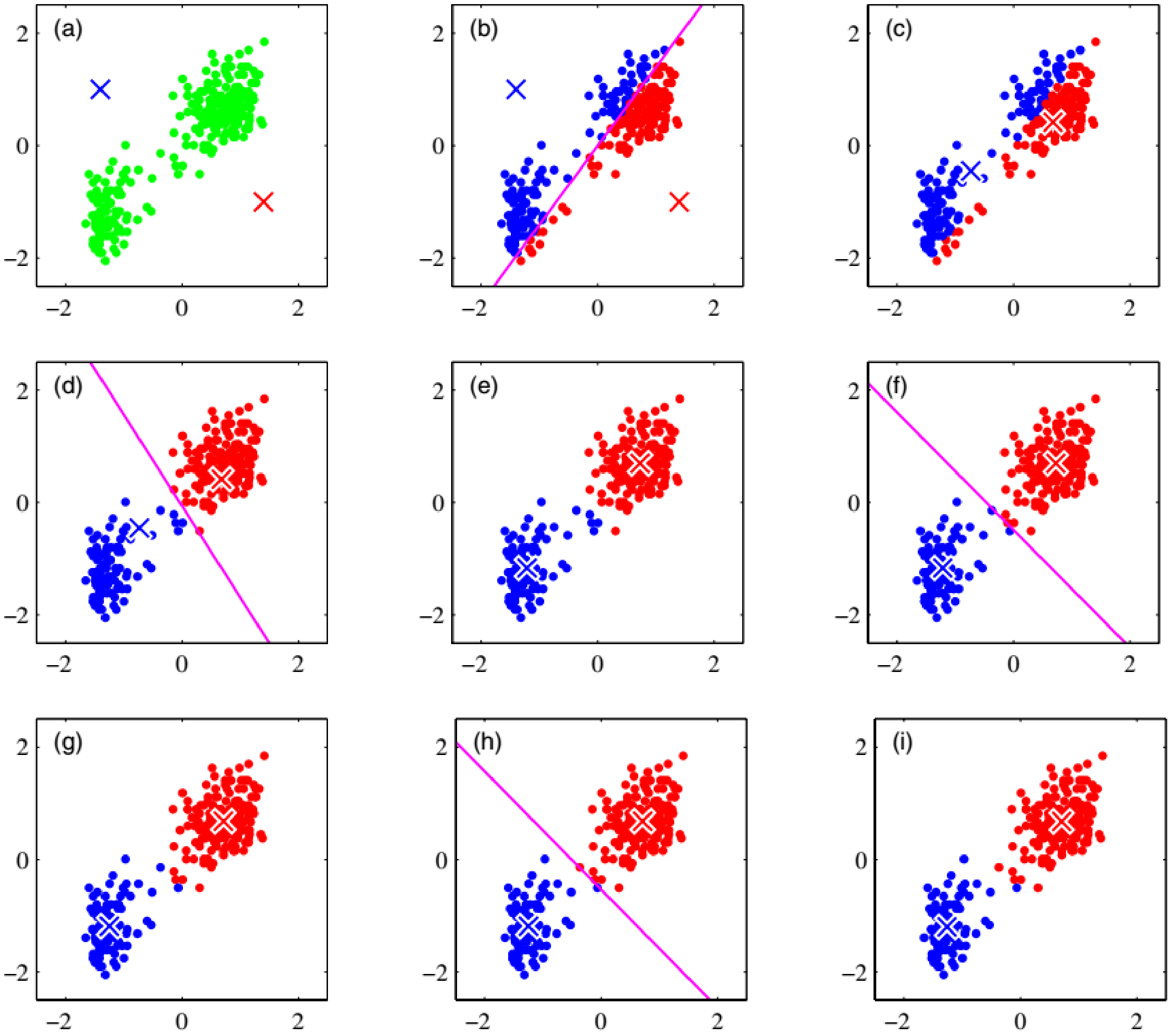


Figure 4.1.2. Illustration of KMeans steps.

The green points in Plot (**a**) from Figure **4.1.2.** are unlabeled data points, and the crosses are the randomly generated cluster centers which are far from the center of the clusters. The red and blue points in Plot (**b**) indicate the samples that are assigned to the closest clusters to them. and the pink solid line is the border between the two so found clusters. Plot (**c**) shows the cluster prototypes (i.e., crosses) after an update. Plots (**d**)-(**h**) illustrate the next steps of the KMeans and Plot (**i**) demonstrates the final cluster centers and clustered points.

Important Remarks

1. We can view KMeans as an optimisation algorithm to minimise he following objective function:

$$J(\boldsymbol{\mu}, \boldsymbol{r}) := \sum_1^N \sum_1^K r_{nk} \times d(\boldsymbol{x}_n, \boldsymbol{\mu}_k)$$

with the constraints that $r_{nk} \in \{0, 1\}$ and $\sum_{k=1}^K r_{nk} = 1$ for all n. Note that J is a function of the cluster centres $\boldsymbol{\mu} := (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K)$ and the cluster assignments $\boldsymbol{r} := (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N)$, and is optimised with respect to these variables.

2. *KMeans* is sensitive to initial values, which means the different execution of *KMeans* (with different initial cluster centers) may result in different solutions.
3. *KMeans* is a non-probabilistic algorithm which only supports *hard-assignment*. This means a data point can only be assigned to one and only one of the clusters.

Application

Figure **4.1.3.** shows an application of KMeans in image compression. In this particular application, the pixels colors (i.e., RGB intensities) are treated as the datapoints. We can cluster the pixel colors into some clusters, and then replace the colors in a cluster with the cluster centroid. Therefore, the color resolution of the picture is reduced, which in turn leads to compression. Apparently, the lower the K is set, the compression rate would be higher, and accordingly, the image quality would be lower.

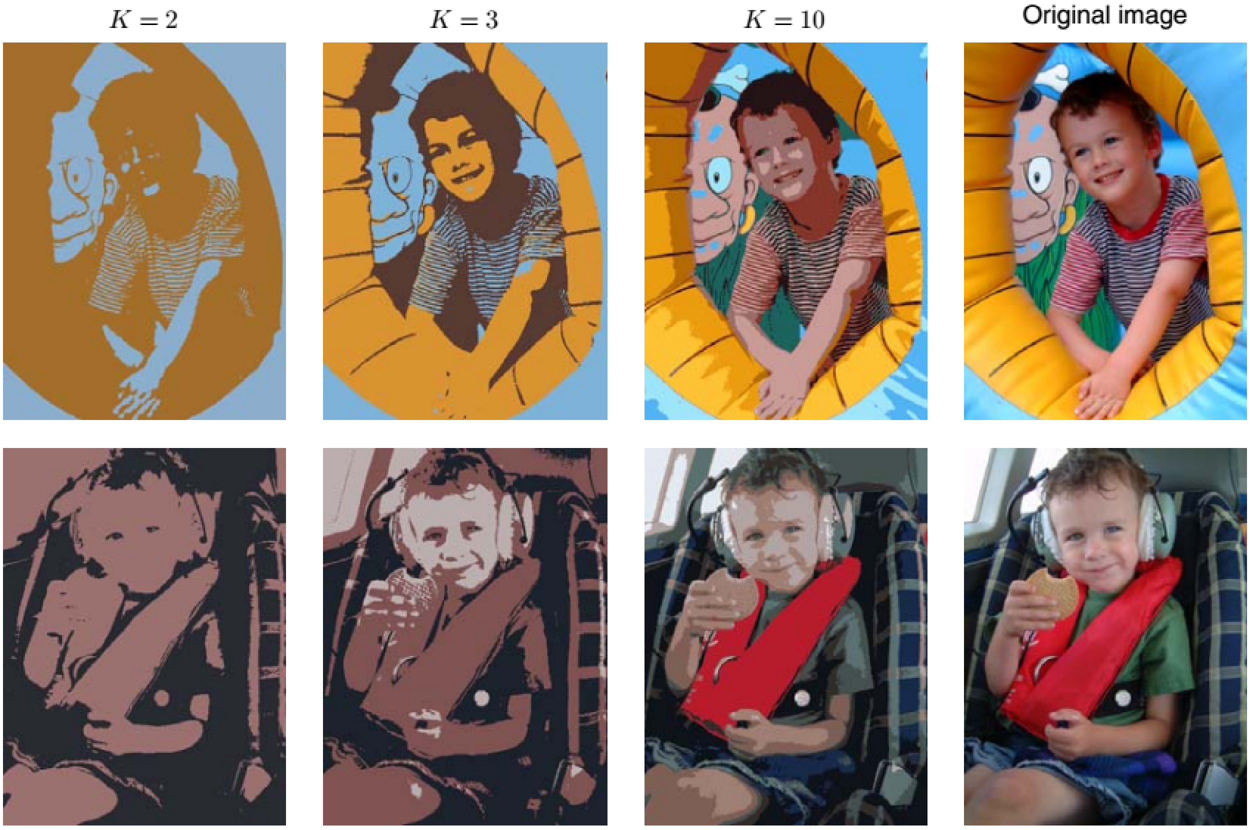


Figure 4.1.3. Application of KMeans algorithm in image compression.

2 Gaussian Mixture Models and Expectation-Maximisation

Consider the clustering problem that we saw in the previous chapter, where we wanted to partition a set of training data points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ into K groups of similar data points. Thinking differently, we would like to assign one of the *labels* $\{1, \ldots, K\}$ to each data point where a label is a group identifier. As opposed to supervised learning, we are *not* given labels for *training* data points in the clustering problem. In other words, the label of the training data points are *latent* or *hidden,* which we denote by *latent variables.* Latent variable is an important concept in probabilistic modelling, and its usage goes beyond the clustering problem.

In this chapter, we give a probabilistic modelling viewpoint to the clustering problem which involves latent variables. The parameters of the probabilistic model and best values for the latent variables are then learned based on the maximum likelihood principle. This is in contrast to the Kmeans algorithm for clustering, which is merely designed based on minimising an *intuitively* reasonable objective function (and is not based on any probabilistic principle).

Gaussian Mixture Models (GMMs)

A Generative Story. Consider the following hypothetical generative story for generating a label-datapoint pair $(k, \boldsymbol{x})$:

- First, generate a cluster label k, by tossing a dice with K faces where each face of the dice corresponds to a cluster label.
- Second, generate the data point $\boldsymbol{x}$, by sampling from the distribution $p_k(.)$ corresponding to the cluster label k

We use *oracle* to refer to the person who has hypothetically generated the data based on the above process. After generating the data, the oracle hides the label k from us and only gives us the datapoint $\boldsymbol{x}$. We denote the label which is hidden from us by a random variable $z \in \{1, \ldots, K\}$ which we call latent variable. Now given the training data, we would like to find the best value for the latent variables, *and* the best estimates for the parameters of the above generative story.

The Probabilistic Generative Model. Let us formulate the *probabilistic model* corresponding to the above generative story:

- Tossing a dice with K faces is the same as sampling from a *multinomial distribution* on K elements; the parameters of the multinomial are the probability φ_k of selecting elements, where $\sum_{k=1}^{K} \varphi_k = 1$ and $\varphi_k \geq 0$.
- For the face k,we assume data points are sampled from Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ where parameters include mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$. Note that we have a collection of these Gaussian distributions, each of which corresponds to one of K dice faces.

As said, the oracle hides the labels and only gives us the datapoints $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, and we aim to best guess the latent variables $(z_1, \ldots, z_N)$ where each $z_n \in \{1, .., K\}$ represents the *latent label* for a datapoint $\boldsymbol{x}_n$. Furthermore, we would like to find the best estimate for model parameters $\boldsymbol{\theta} := (\boldsymbol{\varphi}, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K)$. We achieve these two goals by resorting to the *maximum likelihood principle*.

Gaussian Mixture Model and the Likelihood. If we were given a *complete* data point $(k, \boldsymbol{x})$ where the label was *not* hidden, then the probability of the pair according to our generative story would be:

$$p(k, \boldsymbol{x}_n) = p(\text{face } k) p(\boldsymbol{x}_n | \text{face } k) = \varphi_k \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

where the probability of the dice face k is φ_k, and the probability of the datapoint $\boldsymbol{x}_n$ given the dice face is $\mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

In practice, the oracle has not given us the complete datapoint, instead it has given us only an *incomplete* data (or *observed* data) $\boldsymbol{x}$. Denoting the latent label by the latent variable z_n, we can write

$$\begin{aligned} p(\boldsymbol{x}_n) = \sum_{z_n \in \{1, \ldots, K\}} p(z_n, \boldsymbol{x}_n) &= \sum_{k=1}^{K} p(z_n = k) p(\boldsymbol{x} | \text{face } k) \\ &= \sum_{k=1}^{K} \varphi_k \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \end{aligned}$$

where we have *marginalised out* the random variable to get the probability of the incomplete data. This model is called the *Gaussian mixture model*, which is composed of *mixing coefficients* (corresponding to the multinomial distribution parameterised by $\boldsymbol{\varphi}$) and a collection of Gaussian *components*. We can now write down the log-likelihood of the *observed* data:

$$\mathcal{L}(\boldsymbol{\theta}) := \sum_{n=1}^{N} \ln p(\boldsymbol{x}_n) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \varphi_k \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

where $\boldsymbol{\theta} := (\boldsymbol{\varphi}, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K)$ are the model parameters.

The Prediction Rule. After estimating the model parameters, we can ask the question to which cluster an observed datapoint should be assigned to? Using the the Bayes' rule, we can find the conditional probability of a cluster k for a given datapoint $\boldsymbol{x}_n$:

$$\gamma(z_{nk}) := p(z_n = k | \boldsymbol{x}_n) = \frac{\varphi_k \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \varphi_j \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

where $\sum_{k=1}^{K} \gamma(z_{nk}) = 1$. Interestingly, this gives us a *partial* assignment, also called *soft* assignment, of a datapoint to a clusters based on our probabilistic model. Recall that in *Kmeans*, a datapoint belongs only to one cluster (i.e. *hard* assignment) and there was no notion of partial assignment.

Alternatively, we shall view φ_k as the *prior* probability of $z_n = k$, and the quantity $\gamma(z_{nk})$ as the corresponding *posterior* probability once we have observed $\boldsymbol{x}_n$. We refer to $\gamma(z_{nk})$ as the *responsibility* that cluster k takes for *explaining* the observation $\boldsymbol{x}_n$.

We will see in the next section that cluster prediction is intertwined with parameter estimation, i.e. they are both done *simultaneously* in an iterative algorithm called the *Expectation Maximisation Algorithm* which finds *both* the best latent variable assignment and the best parameter estimates.

Expectation Maximisation (EM) for GMMs

The maximum likelihood principle instructs us to maximise the likelihood $\mathcal{L}(\boldsymbol{\theta})$ as a function of model parameters. Let us begin by writing down the conditions that must be satisfied at a maximum of the log-likelihood function.

The Mean Parameters. Firstly, setting the gradient of $\mathcal{L}(\boldsymbol{\theta})$ with respect to the means $\boldsymbol{\mu}_k$ of the Gaussian components to zero, we obtain

$$\frac{\partial}{\partial \boldsymbol{\mu}_k}\mathcal{L}(\boldsymbol{\theta}) = 0 \Rightarrow \sum_{n=1}^{N} \left(\underbrace{\frac{\varphi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_j \varphi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)}}_{\gamma(z_{nk})} \sum_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \right) = \mathbf{0}$$

where the responsibility quantities naturally appear on the right hand side. Rearranging the terms, we obtain

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n$$

where we have defined $N_k := \sum_{n=1}^{N} \gamma(z_{nk})$. We can interpret N_k as the *effective* number of points assigned to cluster k. Note carefully the form of this solution. We see that the mean $\boldsymbol{\mu}_k$ for the kth Gaussian component is obtained by taking a weighted mean of all of the points in the data set, in which the weighting factor for datapoint $\boldsymbol{x}_n$ is given by the posterior probability $\gamma(z_{nk})$ that component k was *responsible* for generating $\boldsymbol{x}_n$.

The Covariance Matrices. Secondly, setting the gradient of $\mathcal{L}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\Sigma}_k$ to zero, and following a similar line of reasoning, we obtain

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k}\mathcal{L}(\boldsymbol{\theta}) = 0 \Rightarrow \boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T$$

which has the same form as the corresponding result for a single Gaussian fitted to the data set, but again with each data point weighted by the corresponding posterior probability and with the denominator given by the effective number of points associated with the corresponding component.

The Mixing Coefficients. Finally, we maximise $\mathcal{L}(\boldsymbol{\theta})$ with respect to the mixing coefficients $\boldsymbol{\varphi}$. Here we must take account of the constraint that the mixing coefficients have to sum to one. This can be achieved using a *Lagrange multiplier* λ and maximizing the following objective function

$$\mathcal{L}(\boldsymbol{\theta}) + \lambda\big(\textstyle\sum_{k=1}^{K} \varphi_k - 1\big)$$

which, after setting the gradient to zero, gives:

$$\begin{aligned}\frac{\partial}{\partial \varphi_k}\Big[\mathcal{L}(\boldsymbol{\theta}) + \lambda(\sum_{k=1}^{K} \varphi_k - 1)\Big] = 0 &\Rightarrow \sum_{n=1}^{N} \frac{\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \varphi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda = 0 \\ \textbf{multiply both sides by } \varphi_k &\Rightarrow \sum_{n=1}^{N} \frac{\varphi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \varphi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda \varphi_k = 0 \\ &\Rightarrow \sum_{n=1}^{N} \gamma_{nk} + \lambda \varphi_k = 0 \Rightarrow \varphi_k = \frac{\sum_{n=1}^{N} \gamma_{nk}}{-\lambda}\end{aligned}$$

Making use of the constraint $\sum_k \varphi_k = 1$, we find $\lambda = -N$. Hence eliminating λ, we obtain

$$\varphi_k = \frac{N_k}{N}$$

so that the mixing coefficient for the kth component is given by the average responsibility which that component takes for explaining the data points.

EM Algorithm for Gaussian Mixture Models. It is worth emphasizing that the above results for the optimal parameter estimate do *not* constitute a *closed-form* solution because the responsibilities $\gamma(z_{nk})$ depend on those parameters in a complex way. However, these results do suggest a simple *iterative* scheme for finding a solution to the maximum likelihood problem, which as we shall see turns out to be an instance of the *Expectation Maximisation (EM)* algorithm for the particular case of the Gaussian mixture model.

Here is the EM algorithm for Gaussian mixture models. We first choose some initial values for the means, covariances, and mixing coefficients. Then we alternate between the following two updates that we shall call the E step and the M step (for reasons that will become apparent shortly) until a stopping condition is met:

- In the expectation step, or *E step*, we use the current values for the parameters to evaluate the posterior probabilities γ_{nk}
- In the maximisation step, or *M step*, to re-estimate the means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$, and mixing coefficients $\boldsymbol{\varphi}$ using the above results.

Relation to *KMeans*. The above algorithm is reminiscent of the *Kmeans* algorithm, with some differences: (i) The only parameters in *Kmeans* are the cluster means $\boldsymbol{\mu}_k$ (there are no $\boldsymbol{\varphi}$ and $\boldsymbol{\Sigma}_k$), and (ii) The assignment of data points to clusters are *hard* in *Kmeans* where each data point is associated *uniquely* with one cluster; however, the EM algorithm makes *soft* assignments based on the posterior probabilities γ_{nk}.

Important Remarks.

1. *Each update to the parameters resulting from an E step followed by an M step is guaranteed to increase the log likelihood function* (we do not prove it here).
2. In practice, the algorithm is deemed to have converged when the change in the log likelihood function, or alternatively in the parameters, falls below some threshold.
3. The EM algorithm takes many more iterations to reach convergence compared to the *Kmeans* algorithm, and that each cycle requires significantly more computations.

Example. We illustrate the EM algorithm for a mixture of two Gaussians on the data points in Figure **4.2.1.** Here a mixture of two Gaussians is used, with centres initialized using the same values as for the *Kmeans* algorithm in Figure **4.1.1**, and with covariance matrices initialized to be proportional to the unit matrix.

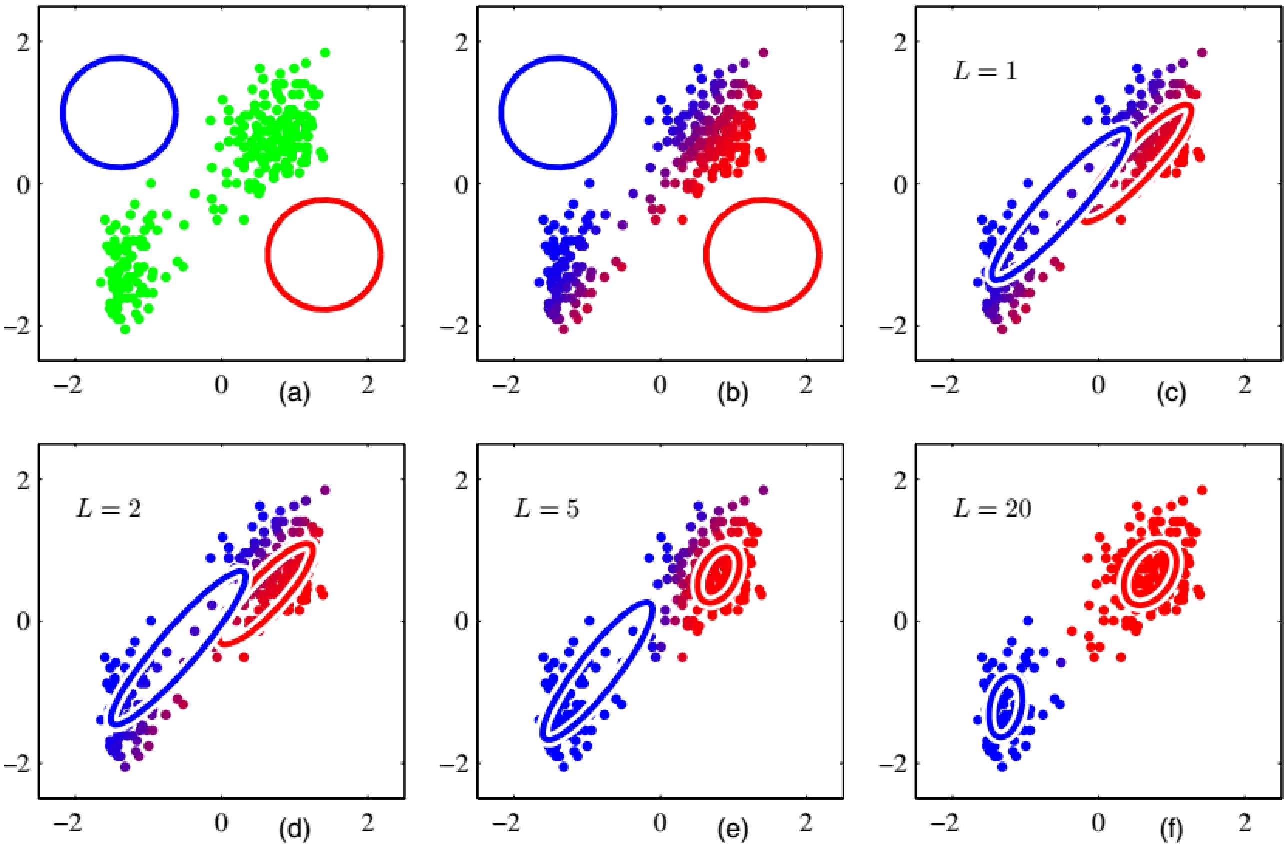


Figure 4.2.1. Illustration of EM steps in GMM.

Plot **(a)** shows the data points in green, together with the initial configuration of the mixture model in which the one standard-deviation contours for the two Gaussian components are shown as blue and red circles.

Plot **(b)** shows the result of the initial E step, in which each data point is depicted using a proportion of blue ink equal to the posterior probability of having been generated from the blue component, and a corresponding proportion of red ink given by the posterior probability of having been generated by the red component. Thus, points that have a significant probability for belonging to either cluster appear purple.

The situation after the first M step is shown in plot **(c)**, in which the mean of the blue Gaussian has moved to the mean of the data set, weighted by the probabilities of each data point belonging to the blue cluster, in other words it has moved to the centre of mass of the blue ink. Similarly, the covariance of the blue Gaussian is set equal to the covariance of the blue ink. Analogous results hold for the red component.

Plots **(d)**, **(e)**, and **(f)** show the results after 2, 5, and 20 complete cycles of EM, respectively. In Plot **(f)** the algorithm is close to convergence.

3
A General View to EM

In this section, we present a general view of the EM algorithm that recognizes the key role played by latent variables. We discuss this approach first of all in an abstract setting, and then for illustration we consider once again the case of Gaussian mixtures.

The EM Algorithm: General Case

The Training Objective. The goal of the EM algorithm is to find maximum likelihood solution for models having latent variables. We denote the observed data by $\boldsymbol{X}$, and similarly we denote the latent variables by $\boldsymbol{Z}$. The set of all model parameters is denoted by $\boldsymbol{\theta}$, and so the log likelihood function is given by

$$\ln p(\boldsymbol{X}|\boldsymbol{\theta}) = \ln \textstyle\sum_{\boldsymbol{Z}} p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$$

Note that our discussion will apply equally well to continuous latent variables simply by replacing the sum over $\boldsymbol{Z}$ with an integral.

Suppose that, for the observation $\boldsymbol{X}$, we were told the corresponding value of the latent variable $\boldsymbol{Z}$. We shall call $\{\boldsymbol{X}, \boldsymbol{Z}\}$ the *complete* data set, and we shall refer to the actual observed data $\boldsymbol{X}$ as *incomplete. The likelihood function for the complete data set simply takes the form* $\ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$*, and we shall suppose that maximization of this complete-data log likelihood function is straightforward.*

Maximising the Likelihood of Incomplete Data. In practice, we are not given the complete data set $\{\boldsymbol{X}, \boldsymbol{Z}\}$, but only the incomplete data $\boldsymbol{X}$. Our state of knowledge of the values of the latent variables in $\boldsymbol{Z}$ is given only by the posterior distribution $p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta})$. Because we cannot use the complete-data log likelihood, we consider instead its *expected* value under the posterior distribution of the latent variable, which corresponds to the E step of the EM algorithm. In the subsequent M step, we maximize this expectation as a function of model parameters. If the current estimate for the parameters is denoted $\boldsymbol{\theta}^{\text{old}}$, then a pair of successive E and M steps gives rise to a revised estimate $\boldsymbol{\theta}^{\text{new}}$. The algorithm is initialized by choosing some starting value for the parameters $\boldsymbol{\theta}^{0}$. The expectation and maximisation steps are iterated until a convergence condition is met.

The EM Algorithm. More specifically, in the *E step*, we use the current parameter values $\boldsymbol{\theta}^{\text{old}}$ to find the posterior distribution of the latent variables given by $p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$. We then use this posterior distribution to find the expectation of the complete-data log likelihood evaluated for some general parameter value $\boldsymbol{\theta}$. This expectation, denoted $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$, is given by

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) := \textstyle\sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$$

In the *M step*, we determine the revised parameter estimate θnew by maximising this function

$$\boldsymbol{\theta}^{\text{new}} := \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$$

Note that in the definition of $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$, the logarithm acts directly on the joint distribution $p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$, and so the corresponding M-step maximization will, by supposition, be tractable.

To summarise, the general EM algorithm is as follows:

- Choose an initial setting for the parameters $\boldsymbol{\theta}^{\text{old}}$
- While the convergence is not met:
  - **E step:** Evaluate $p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$
  - **M Step:** Evaluate $\boldsymbol{\theta}^{\text{new}}$ given by $\boldsymbol{\theta}^{\text{new}} \leftarrow \arg\max_{\boldsymbol{\theta}} \underbrace{\sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})}_{Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})}$
  - $\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}}$

The incomplete-data log likelihood is guaranteed to increase in each cycle of the EM algorithm (we don't prove it here).

The Hard-EM Algorithm. In the Hard-EM Algorithm, there is no expectation in over the latent variables in the definition of the Q function. Instead, only the most probable value for the latent variable is chosen to define the Q function. In summary, this algorithm is as follows:

- Choose an initial setting for the parameters $\boldsymbol{\theta}^{\text{old}}$
- While the convergence is not met:
  - **E step:** Set $\boldsymbol{Z}^* \leftarrow \arg\max_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$
  - **M Step:** Set $\boldsymbol{\theta}^{\text{new}} \leftarrow \arg\max_{\boldsymbol{\theta}} \ln p(\boldsymbol{X}, \boldsymbol{Z}^*|\boldsymbol{\theta})$
  - $\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}}$

It can be shown that the incomplete-data log likelihood is guaranteed to increase in each cycle of the Hard-EM algorithm (we don't prove it here).

EM for GMMs: Revisited

The Complete Data Likelihood. Assume that in addition to the observed data set $\boldsymbol{X} := \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, we were also given the values of the corresponding latent variables

$\boldsymbol{Z} := \{\boldsymbol{z}_1, .., \boldsymbol{z}_N\}$, then

$$\begin{aligned}\ln p(X, Z|\mu, \Sigma, \varphi) &= \ln \prod_{n=1}^{N} \prod_{k=1}^{K} \varphi^{z_{n,k}} \mathcal{N}(x_n|\mu_k, \Sigma_k)^{z_{n,k}} \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} z_{n,k} \ln \varphi_k + z_{n,k} \ln \mathcal{N}(x_n|\mu_k, \Sigma_k)\end{aligned}$$

where $\boldsymbol{z}_n := (z_{n1}, \ldots, z_{nk})$ is the cluster assignment vector for the nth datapoint in which $z_{nk} = 1$ if this datapoint belongs to the cluster k and zero otherwise. Note that only one element of the cluster assignment vector is 1, and the rest are zero.

Maximising the complete data likelihood to find the parameters of the model is straightforward:

- The mixing components: $\varphi_k = \frac{N_k}{N}$ where $N_k := \sum_{n=1}^{N} z_{nk}$
- The mean parameters: $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} z_{nk} \boldsymbol{x}_n$
- The covariance matrices: $\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} z_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T$

The Q function. In practice we are not given the values for the latent variables, and we need to resort to the EM algorithm to find the best values for the parameters and the latent variables.

$$\begin{aligned}Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) &:= \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\theta}^{\text{old}}) \left[z_{n,k} \ln \varphi_k + z_{n,k} \ln \mathcal{N}(x_n|\mu_k, \Sigma_k)\right] \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} p(z_{nk} = 1|\boldsymbol{x}_n, \boldsymbol{\theta}^{\text{old}}) \ln \varphi_k + p(z_{nk} = 1|\boldsymbol{x}_n, \boldsymbol{\theta}^{\text{old}}) \ln \mathcal{N}(x_n|\mu_k, \Sigma_k)\end{aligned}$$

where $p(z_{nk} = 1|\boldsymbol{x}_n, \boldsymbol{\theta}^{\text{old}})$ is indeed the responsibility that we defined before:

$$\gamma(z_{nk}) := p(z_{nk} = 1|\boldsymbol{x}_n, \boldsymbol{\theta}^{\text{old}}) = \frac{\varphi_k^{\text{old}} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k^{\text{old}}, \boldsymbol{\Sigma}_k^{\text{old}})}{\sum_j \varphi_j^{\text{old}} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j^{\text{old}}, \boldsymbol{\Sigma}_j^{\text{old}})}$$

Maximising the Q function, leads to the following updated parameters:

- The mixing components: $\varphi_k^{\text{new}} = \frac{N_k}{N}$ where $N_k := \sum_{n=1}^{N} \gamma(z_{nk})$
- The mean parameters: $\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n$
- The covariance matrices: $\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T$

The EM Algorithm for GMMs. The EM Algorithms thus is as follows:

- Choose an initial setting for the parameters $\boldsymbol{\theta}^{\text{old}} = (\boldsymbol{\varphi}^{\text{old}}, \boldsymbol{\mu}_1^{\text{old}}, \dots, \boldsymbol{\mu}_K^{\text{old}}, \boldsymbol{\Sigma}_1^{\text{old}}, \dots, \boldsymbol{\Sigma}_K^{\text{old}})$
- While the convergence is not met:
    - **E step:** Set $\forall n, \forall k : \gamma(z_{nk})$ based on $\boldsymbol{\theta}^{\text{old}}$
    - **M Step:** Set $\boldsymbol{\theta}^{\text{new}}$ based on $\forall n, \forall k : \gamma(z_{nk})$
    - $\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}}$

This is exactly the EM Algorithm that we saw in the previous chapter.

4
Activity 4.1: EM for GMM

In this activity we implement soft and hard expectation maximisation for training Gaussian mixture models. This activity helps you to complete Assignment 2.

Please download this (https://www.alexandriarepository.org/wp-content/uploads/20160701174342/Activity4.zip) zip file (right click and then choose "save link as") that contains the dataset (.txt) and Jupyter notebook (.ipynb).

For detailed discussion about setting your programming environment up, please refer to Appendix B: Setting up Your Programming Environme (https://www.alexandriarepository.org/?post_type=module&p=97654)

5 Latent Variable Models for Document Analysis

Document Clustering

Given a collection of documents $\{d_1, \ldots, d_N\}$ we would like to partition them into K clusters.

Document Representation. Each document d_n is made of some text. We treat a document as a *set* of words in its text irrespective of their positions, i.e. a document consisting of the text "Andi went to school to meet Bob" is represented as the following set of words: {Andi, went, to, school, to, meet, Bob}. Note that if we had another document with the text "Bob went to school to meet Andi", it would have the same representation as the previous sentence. We refer to this as the *bag of word* representation of the document. Also, we assume the words appearing in our collection of documents come from a dictionary denoted by $\mathcal{A}$.

Generative Model. We assume the following hypothetical generative story for generating the collection of our documents:

- For each document d_n
  - toss the K-face dice (with the parameter $\boldsymbol{\varphi}$) to choose the face (i.e. the cluster) k that the nth document belongs to
  - For each word placeholder in the document d_n
    - generate the word by tossing the dice (with parameter $\boldsymbol{\mu}_k$) corresponding to the face k.

The parameters of the model are (1) the clusters proportion $\boldsymbol{\varphi}$ which is a probability vector of size K where $\sum_{k=1}^{K} \varphi_k = 1$, and (2) the word proportion $\boldsymbol{\mu}_k$ corresponding to the kth face of the dice where $\sum_{w \in \mathcal{A}} \mu_{k,w} = 1$; note that we have K such word proportion vectors each of which corresponding to a face of the dice (or cluster).

The probability of generating a pair of a document and its cluster (k,d), according to our generative story, is

$$\begin{aligned} p(k,d) = p(k)p(d|k) &= \varphi_k \prod_{w \in d} \mu_{k,w} \\ &= \varphi_k \prod_{w \in \mathcal{A}} \mu_{k,w}^{c(w,d)} \end{aligned}$$

where $c(w,d)$ is simply the number of occurrences of the word w in the document d.

In practice, the document cluster ids are not given to us, so we use latent variables to denote the cluster assignments.

Complete Data

The Likelihood. Assume that in addition to the documents $\{d_1, \ldots, d_N\}$, we were also given the values of the corresponding latent variables $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$, then

$$p(d_1, \boldsymbol{z}_1, \ldots, d_N, \boldsymbol{z}_N) = \prod_{n=1}^{N} \prod_{k=1}^{K} \left(\varphi_k \prod_{w\in\mathcal{A}} \mu_{k,w}^{c(w,d_n)}\right)^{z_{n,k}}$$

where $\boldsymbol{z}_n := (z_{n1}, \ldots, z_{nk})$ is the cluster assignment vector for the nth document in which $z_{nk} = 1$ if this document belongs to the cluster k and zero otherwise. Note that only one element of the cluster assignment vector is 1, and the rest are zero. To summarise, the log-likelihood of the complete data is:

$$\ln p(d_1, \boldsymbol{z}_1, \ldots, d_N, \boldsymbol{z}_N) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{n,k} \left(\ln \varphi_k + \sum_{w\in\mathcal{A}} c(w, d_n) \ln \mu_{k,w}\right)$$

Learning the Parameters. Maximising the complete data log-likelihood to learn the parameters of the model leads to the following solution:

- The mixing components: $\varphi_k = \frac{N_k}{N}$ where $N_k := \sum_{n=1}^{N} z_{nk}$
- The word proportion parameters for each cluster: $\mu_{k,w} = \frac{\sum_{n=1}^{N} z_{n,k} c(w,d_n)}{\sum_{w'\in\mathcal{A}} \sum_{n=1}^{N} z_{n,k} c(w',d_n)}$

The above results can be obtained by forming the Lagrangian (to enforce the constraints, e.g. the sum of mixing coefficients must be zero), and then setting the derivatives with respect to the parameters to zero (see the Appendix A).

The above estimates for the optimal parameters are very intuitive. For example, the best value for $\boldsymbol{\mu}_k$ is obtained by counting the number of times that each word of the dictionary has been seen in the documents belonging to the cluster k, and then normalising this count vector so that it sums to 1 (by dividing each element of the count vector by the sum of the counts).

Incomplete Data and EM

The Likelihood. In practice, the document clusters are not given to us, so $\boldsymbol{z}_n$ is latent. So, the probability of the observed documents is

$$\begin{aligned} p(d_1, \ldots, d_N) = \prod_{n=1}^{N} p(d_n) &= \prod_{n=1}^{N} \sum_{k=1}^{K} p(z_{n,k} = 1, d_n) \\ &= \prod_{n=1}^{N} \sum_{k=1}^{K} \left(\varphi_k \prod_{w\in\mathcal{A}} \mu_{k,w}^{c(w,d_n)}\right) \end{aligned}$$

To summarise, the log-likelihood is:

$$\begin{aligned}\ln p(d_1,\ldots,d_N) = \sum_{n=1}^{N} \ln p(d_n) &= \sum_{n=1}^{N} \ln \sum_{k=1}^{K} p(z_{n,k}=1, d_n) \\ &= \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \left(\varphi_k \prod_{w\in\mathcal{A}} \mu_{k,w}^{c(w,d_n)} \right)\end{aligned}$$

To maximise the above incomplete data log-likelihood objective, we resort to the EM Algorithm.

The Q Function. Let's look at the Q function, which will be the basis of our EM Algorithm:

$$\begin{aligned}Q(\boldsymbol{\theta},\boldsymbol{\theta}^{\text{old}}) &:= \sum_{n=1}^{N}\sum_{k=1}^{K} p(z_{n,k}=1|d_n,\boldsymbol{\theta}^{\text{old}}) \ln p(z_{n,k}=1,d_n|\boldsymbol{\theta}) \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} p(z_{n,k}=1|d_n,\boldsymbol{\theta}^{\text{old}}) \left(\ln\varphi_k + \sum_{w\in\mathcal{A}} c(w,d_n)\ln\mu_{k,w} \right) \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{n,k}) \left(\ln\varphi_k + \sum_{w\in\mathcal{A}} c(w,d_n)\ln\mu_{k,w} \right)\end{aligned}$$

where $\boldsymbol{\theta} := (\boldsymbol{\varphi},\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K)$ is the collection of model parameters, and $\gamma(z_n,k) := p(z_{n,k}=1|d_n,\boldsymbol{\theta}^{\text{old}})$ are the responsibility factors.

To maximise the Q function, we can form the Lagrangian to enforce the constraints (See Appendix A), and set the derivatives to zero which leads to the following solution for the model parameters:

- The mixing components: $\varphi_k = \frac{N_k}{N}$ where $N_k := \sum_{n=1}^{N}\gamma(z_{n,k})$
- The word proportion parameters for each cluster: $\mu_{k,w} = \frac{\sum_{n=1}^{N}\gamma(z_{n,k})c(w,d_n)}{\sum_{w'\in\mathcal{A}}\sum_{n=1}^{N}\gamma(z_{n,k})c(w',d_n)}$

The EM Algorithm. Now let's put everything together to construct our EM algorithm to learn the parameters and find the best value for the latent variables:

- Choose an initial setting for the parameters $\boldsymbol{\theta}^{\text{old}} = (\boldsymbol{\varphi}^{\text{old}},\boldsymbol{\mu}_1^{\text{old}},\ldots,\boldsymbol{\mu}_K^{\text{old}})$
- While the convergence is not met:
  - **E step:** Set $\forall n,\forall k: \gamma(z_{n,k})$ based on $\boldsymbol{\theta}^{\text{old}}$
  - **M Step:** Set $\boldsymbol{\theta}^{\text{new}}$ based on the above equations

- $\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}}$

6
Activity 4.2: Document Clustering

In this activity we implement soft and hard expectation maximisation for document clustering problem. This activity helps you to complete Assignment 2.

Please download this (https://www.alexandriarepository.org/wp-content/uploads/20160701174342/Activity4.zip) zip file (right click and then choose "save link as") that contains the dataset (.txt), and Jupyter notebook (.ipynb).

For detailed discussion about setting your programming environment up, please refer to Appendix B: Setting up Your Programming Environme (https://www.alexandriarepository.org/?post_type=module&p=97654)

7 Appendix A: Constrained Optimisation

Constrained Optimisation

In these problems, we would like to maximise or minimise an objective function subject to some **constraints**. A constrain is a condition that the solution must satisfy. In this unit, the constrains are **equality** but in general they can be **inequality** as well. Briefly speaking, the constrained optimisation problem that we consider has the following form:

$$\begin{aligned} &\underset{x}{\text{maximise}} && f(x) \\ &\text{subject to} && g_i(x) = 0 \quad i = 1, \ldots, m \end{aligned}$$

Remember that previously we have seen gradient based algorithms for solving unconstrained optimisation problems, i.e. there were not any constraints on the solution for those problems other than achieving the best value for the objective function. Our strategy to solve unconstrained optimisation problem was to set the partial derivatives to zero, and either find a solution analytically or use an iterative algorithm like BGD or SGD to find a solution. Our strategy for solving constrained optimisation problem is to convert them to unconstrained optimisation problems using a technique called **the method of Lagrange multipliers**, and then solve them using the gradient based method that we already know.

Lagrange Multipliers

The method of Lagrange multipliers is a strategy for finding the local maxima and minima of a function subject to equality constraints. More specifically, for the constrained optimisation problem mentioned above, we introduce the notion of Lagrangian as follows:

$$\mathcal{L}(x, \lambda_1, \ldots, \lambda_m) := f(x) - \lambda_1 g_1(x) - \ldots - \lambda_m g_m(x)$$

where λ_i is called Lagrange multipliers.

If $f(x^*)$ is a maximum of f(x) for the original constrained optimisation problem, then there exists $\lambda_1^*, \ldots, \lambda_m^*$ such that $(x^*, \lambda_1^*, \ldots, \lambda_m^*)$ is a **stationary point** for the Lagrange function (stationary points are those points where the partial derivatives of $\mathcal{L}$ are zero). Therefore, to solve a constrained optimisation problem, the recipe is as follows: (1) Construct the Lagrangian, and (2) Set the partial derivatives to zero (with respect to the original variable and all the Lagrange multipliers).

To summarise, the original constrained optimisation problem is converted to an unconstrained optimisation problem via the Lagrangian. We then use the gradient-based techniques to find a stationary point of the Lagrangian, which in turns gives (hopefully) a solution of the original problem.

Example

Suppose we want to solve the following optimisation problem:

$$\begin{aligned} \underset{\theta_1,\theta_2}{\text{maximise}} \qquad & 2\log\theta_1 + 3\log\theta_2 \\ \text{subject to} \qquad & \theta_1 + \theta_2 = 1 \end{aligned}$$

Note that we have only one constraint $\theta_1 + \theta_2 - 1 = 0$. Therefore, the Lagrangian would be:

$$\mathcal{L}(\theta_1, \theta_2, \lambda) := 2\log\theta_1 + 3\log\theta_2 - \lambda(\theta_1 + \theta_2 - 1)$$

We now set the partial derivatives to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta_1} &= 0 \Rightarrow \frac{2}{\theta_1} - \lambda = 0 \\ \frac{\partial \mathcal{L}}{\partial \theta_2} &= 0 \Rightarrow \frac{3}{\theta_2} - \lambda = 0 \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= 0 \Rightarrow \theta_1 + \theta_2 - 1 = 0 \end{aligned}$$

Finding $\theta_1 = \frac{2}{\lambda}$ and $\theta_2 = \frac{3}{\lambda}$ from the first two equations and substituting in the third equation yields

$$\tfrac{2}{\lambda} + \tfrac{3}{\lambda} - 1 = 0 \Rightarrow \lambda = 5$$

Hence, the optimal values are $\theta_1 = \frac{2}{5}$ and $\theta_2 = \frac{3}{5}$.

If you are interested to know more about the method of Lagrangian multipliers, please see the wikipedia page (https://en.wikipedia.org/wiki/Lagrange_multiplier) on this subject.