K-means Clustering, Gaussian Mixture Models and the EM Algorithm

Brian Azizi

Cavendish Laboratory, Department of Physics, J J Thomson Avenue, Cambridge. CB3 0HE

Abstract

With the accelerating amount of unstructed information in the world, unsupervised machine learning gains in importantance. Clustering is one of the main tools for knowledge discovery within machine learning. This paper provides a summary of two of the most widely used clustering techniques, the K-means algorithm and the Gaussian mixture model, as well as the popular Expectation-Maximization algorithm. We will give a general derivation of the methods and describe details of our implementation.

1. Introduction

The goal of *clustering* (also called *cluster analysis*) is to divide a set of data points into different clusters such that data points withing the same cluster are similar to each other and dissimilar to data points in different clusters. When faced with an unlabeled and high-dimensional data set, performing a cluster analysis can give us useful initial insights into the underlying system.

Due its general use as a tool for knowledge discovery, clustering has found practical applications in a range of disciplines. In Biology, and particularly in genetics, clustering methods have been used to infer structures in genetic populations. Clustering has become a common tool in market research, allowing for automatic market segmentation. More recently, text clustering has been used by Google News to aggregate reports on a particular news story from many different publications. Another common use of clustering is in methods of automatic image segmentation. Important applications of image segmentation include medical imaging and machine vision.

We begin in section 2 by describing and deriving the standard K-means algorithm. Section 3 gives a general discussion of mixture models and we derive the expectation-maximization algorithm for inference in latent variable models. In section 4, we explore the use of Gaussian mixture models to clustering. Section 5 briefly discusses extensions to the Gaussian mixture model and concludes this paper.

2. K-Means Clustering

In this section we introduce the *K-Means algorithm* [1] as a simple and intuitive approache to clustering. The algorithm requires only a single parameter to be set, namely the number of desired clusters K. It outputs a *flat* and *non-probabilistic* clustering of the data.

We start off by giving a general description and statement of the K-means method. Following that, we look at its convergence properties and derive the K-means algorithm. We then discuss model selection in the context of K-means clustering and demonstrate the algorithm in a simple setting.

2.1. The K-Means algorithm

Suppose we have data set $S = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)}\}$ consisting of N observations of the D dimensional random variable $\boldsymbol{X} \in \mathbb{R}^D$. We would like to partition the data into K sets, where each set corresponds to a cluster. For now, we assume that K is given. In section 2.3, we will discuss some common strategies for how K can be set.

Each cluster k is represented be a *cluster centroids* $\boldsymbol{\mu}_k \in \mathbb{R}^D$. We also need to introduce a latent variable $z^{(i)}$ for each data point $x^{(i)}$ that contains the *cluster assignment* for sample i. If $z^{(i)} = k$, then $x^{(i)}$ belongs to cluster k.

The K-Means algorithm starts by initializing the centroids $\boldsymbol{\mu}_k$. Typically, this is done by randomly selecting K distinct data points as the initial values for the centroid variables.

The algorithm then repeats the following two steps until convergence:

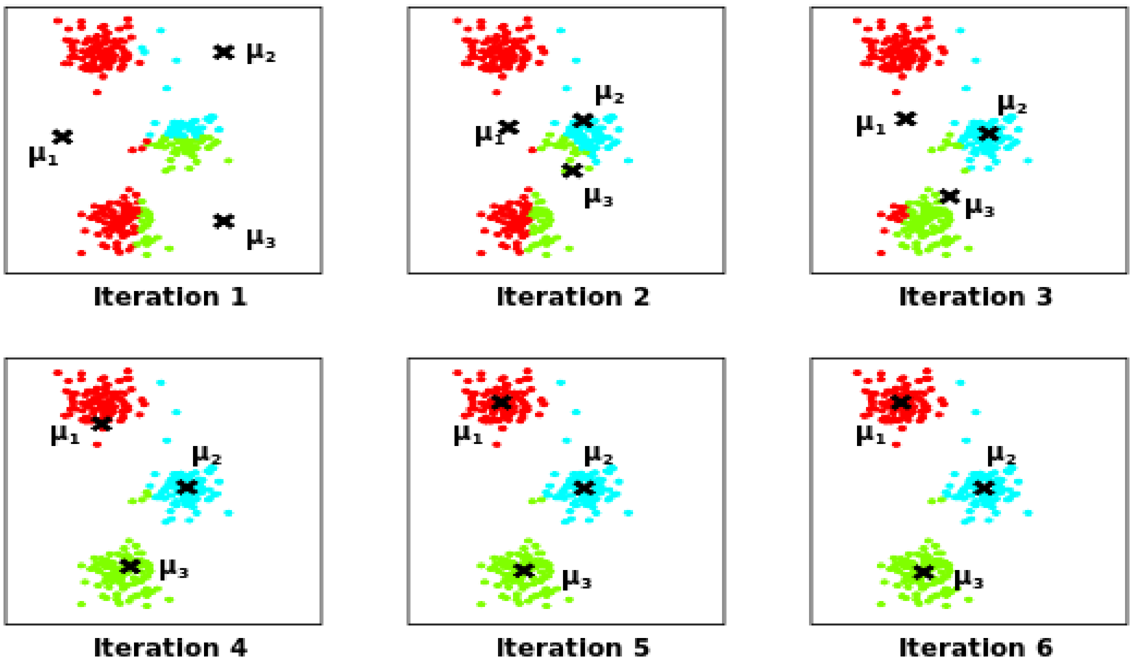

Figure 1: Illustration of the K-means algorithm on a 2d data set with $K = 3$. Cluster assignment is shown by colour. The algorithm converged in 6 iterations

1. Assign each sample $\boldsymbol{x}^{(i)}$ to the cluster represented by the centroid $\boldsymbol{\mu}_k$ that is closest to it.
2. Assign each cluster centroid $\boldsymbol{\mu}_k$ to the mean of all samples that currently belong to cluster k.

The algorithm has converged once there are no more changes. We have summarized the K-means algorithm in Algorithm 1.

Figure 1 illustrates the algorithm with $K = 3$ on a 2d data set consisting of 300 data points. We chose a poor initialization for the cluster centroids so that we could show several iterations. With a better initialization, as suggested above, convergence would have been faster.

2.2. *Derivation and Convergence of K-Means*

In order to prove convergence of K-means, we define the *distortion function*[1]

$$J(z^{(1)}, \ldots, z^{(N)}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{1}\{z^{(i)} = k\} \, \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k\|^2$$

[1]We use the notation $\mathbb{1}\{A\}$ to denote the *indicator function* which is equal to 1 whenever A is true and 0 otherwise.

The objective of K-Means clustering is equivalent to finding the parameters $z^{(i)} \in \{1, \ldots, K\}$ and $\boldsymbol{\mu}_k \in \mathbb{R}^D$, for all i and k, that minimize the distortion function J.

The K-means algorithm is obtained by applying the *coordinate descent algorithm* to minimize the distortion function. Coordinate descent is a simple optimization method that can be used to find a local minimum of a multivariate function $F(\boldsymbol{y})$. It starts off by forming an initial guess $\boldsymbol{y}^0$ for the minimum $\boldsymbol{y}^*$. It then repeatedly cycles through each coordinate direction y_j and minimizes F along that direction.

In the case of the distortion function J, we only need to initialize the cluster centroids since the cluster assignments are independent of one another (the optimal value for $z^{(i)}$ tells us nothing about what $z^{(j)}$ should be if $i \neq j$). We then minimize J with respect to each $z^{(i)}$ keeping all other variables constant. This can be done by brute force by individually trying each value for $z^{(i)}$. The optimal value for $z^{(i)}$ is given by

$$z^{(i)} = \arg\min_k \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k\|^2 \qquad (1)$$

giving us the first inner loop of the K-means algorithm (lines 3-5 in Algorithm 1).

Algorithm 1 K-Means algorithm

```
1:  Initialize cluster centroids $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$

2:  repeat
3:      for $i = 1, \dots, N$ do
4:          $z^{(i)} := \arg\min_k \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k\|^2$
5:      end for

6:      for $k = 1, \dots, K$ do
7:          $\boldsymbol{\mu}_k := \frac{\sum_{i=1}^N \mathbb{1}\{z^{(i)} = k\} \boldsymbol{x}^{(i)}}{\sum_{i=1}^N \mathbb{1}\{z^{(i)} = k\}}$
8:      end for
9:  until Convergence

10: return $z^{(1)}, \dots, z^{(N)}, \boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$
```

Next, we minimize J with respect to each cluster centroid $\boldsymbol{\mu}_k$ keeping all other variables constant. We can do so by setting the gradient of J with respect to $\boldsymbol{\mu}_k$ to zero:

$$\begin{aligned}\nabla_{\boldsymbol{\mu}_k} J &= \sum_{i=1}^{N} \mathbb{1}\{z^{(i)} = k\} \nabla_{\boldsymbol{\mu}_k} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k)^T (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k) \\ &= -2 \sum_{i=1}^{N} \mathbb{1}\{z^{(i)} = k\} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k) = 0\end{aligned}$$

where we made use of the matrix derivative identity $\nabla_{\boldsymbol{y}} \frac{1}{2} \boldsymbol{y}^T \boldsymbol{y} = \boldsymbol{y}$. Rearranging this equation gives us

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^N \mathbb{1}\{z^{(i)} = k\} \boldsymbol{x}^{(i)}}{\sum_{i=1}^N \mathbb{1}\{z^{(i)} = k\}} \tag{2}$$

This is the second inner loop of the algorithm (lines 6-8 in Algorithm 1). Furthermore, as already mentioned, (2) corresponds to setting $\boldsymbol{\mu}_k$ to the means of all samples $\boldsymbol{x}^{(i)}$ which currently assigned to cluster k.[2]

One property of the coordinate descent algorithm is that each iteration is a (weak) improvement on the previous one. This can be easily seen by noting that none of the individual updates along the coordinate axes can make us worse off. So if the approximation to the solution at iteration j is $\boldsymbol{y}^{(j)}$, then $F(\boldsymbol{y}^{(j+1)}) \leq F(\boldsymbol{y}^{(j)})$. Hence, as long as the objective function is bounded below, the algorithm is guaranteed to converge to a local minimum. Using the fact that $J \geq 0$, we have thus established convergence of the K-means algorithm.

[2]It is possible that $\sum_{i=1}^N \mathbb{1}\{z^{(i)} = k\} = 0$. In other words, there is a possibility that a cluster becomes empty in the course of the algorithm. If this happens, we can simply re-initialize the corresponding cluster centroid. However, this may be an indication that K is too large.

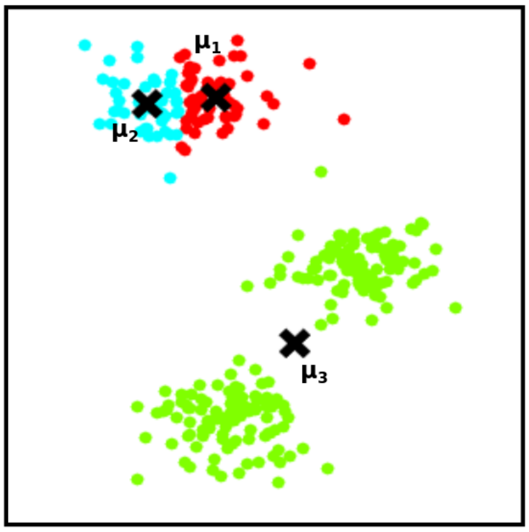

Figure 2: Local optimum of the K-Means algorithm applied to the data set from Figure 1 with $K = 3$.

Note, however, that we are only guaranteed to converge to a *local* minimum. Figure 2 shows an example of a local optimum of the K-means algorithm applied to the same data set as in Figure 1.

In two dimensions, we are able to visualize the data and might be able to recognize that we are stuck in a local solution simply by looking at the output of the algorithm. This is less straightforward in higher dimensions. In practice, this problem is dealt with by performing multiple runs of the entire K-Means algorithm. We have to make sure that each run uses a different set of initial values for the the cluster centroids since, given an initialization, the K-means algorithm is completely deterministic. We also need to record the final value of the distortion function J after each run. The overall output is then chosen to be the solution corresponding to the run for which the final value of J was smallest. This method does not necessarily result in the global optimum either. However, generally lets us avoid particularly poor local optima.

Convergence of the K-means algorithm was extensively studied by [2].

2.3. Model selection for K-Means Clustering

The only user-set parameter in the K-means algorithm is the number of clusters. Therefore, model selection for K-means amounts to selecting the optimal value for K.

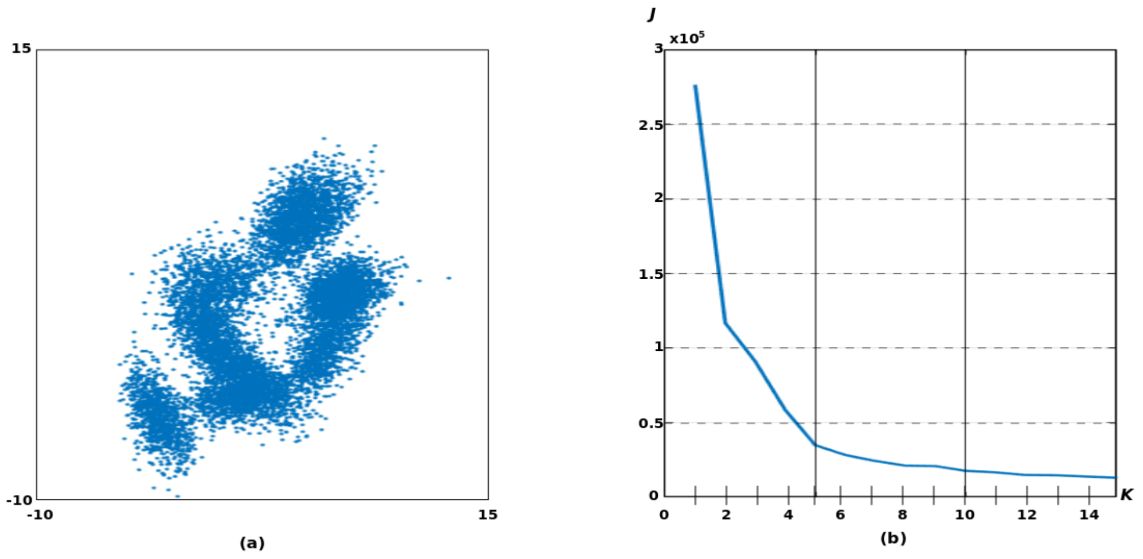

Figure 3: Illustration of the elbow method for selecting K in K-means clustering. (a) Example 2d data set. (b) We ran K-means on the data set in (a) for $1 \leq K \leq 15$ and plotted the optimal value of the distortion function J against K.

The standard model selection tool for K-means clustering is the so-called “elbow method” (also referred to as the “kink method”). We run the K-Means algorithm for a range of different values of K, say for $1 \leq K \leq K_{max}$, where K_{max} is some preset maximum value for K. For each run, we save the optimal value of the distortion function, J^*_K. If necessary, we perform multiple random initializations for each K to avoid poor local optima. Finally, we plot J^*_K against K.

In some cases, it is possible to identify a distinct “kink” in the curve at some K^* (so that the curve has a noticable “elbow”). If that is the case, the value at which the kink occurs is a reasonable choice for K.

The intuition is that, if there is some true number of clusters K^*, we would expect a steep slope in the elbow curve for $K < K^*$. If $K < K^*$, we are putting several clusters into the same group. Splitting the group into its constituent clusters should have a large effect on the distortion function.

If $K > K^*$, we have broken individual clusters into smaller groups. Thus, we would expect only a small reduction in the distortion compared to K^*.

We have illustrated the elbow method in Figure 3. Note the disctinct “kinks” at $K = 2$ and $K = 5$. Looking at panel (a), either choice for K seems to be reasonable.

In practical applications, the optimal value for K is often ambiguous. In fact, a common behaviour of real-world data is that the number of clusters grows with the size of the data set.

Because of this, many practitioners tend to select K manually, making use of domain-specific knowledge if possible.

2.4. Implementation Details and Demonstration

We have implemented the K-means algorithm in C++ using the external linear algebra library “Armadillo” [3]. Translating the pseudo-code in algorithm 1 into C++ is straight-forward. We have only added two further features.

1. We set a maximum value for the number of iterations in order to put a bound on the running time. This is not strictly necessary since, as we saw in section 2.2, the algorithm is guaranteed to converge.
2. We run the algorithm multiple times, randomly re-initializing the centroids before each run. We then save the output of the run for which J was smallest. As explained in section 2.2, this is done in order to mitigate the risk of getting a poor local optimum.

A common demonstration of the K-means algorithm is its application in image compression and image segmentation [4].

Given a colour image, we can treat each pixel of the image as an individual data point in 3 dimensions. The dimensions corresponds to the three colour channels (red, green and blue). The K-means algorithm will cluster the pixels according to their colour. Once we

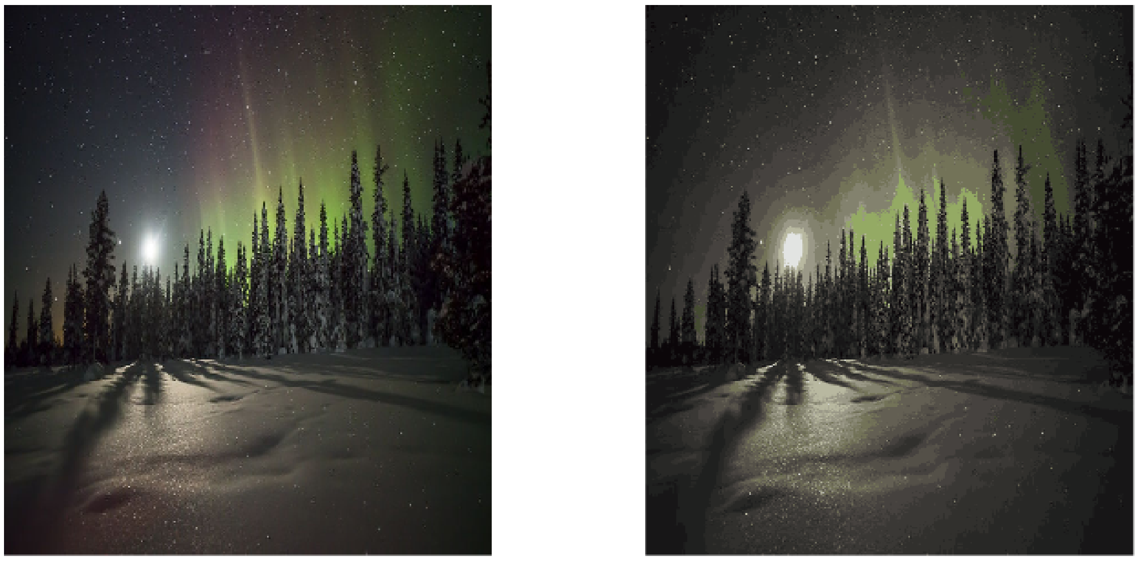

Figure 4: Illustration of K-means to image compression. Right: Original image. Left: Compressed image with $K = 16$.

have a clustering, we replace each pixel with the centroid of its assigned cluster. The resulting image is a compressed version of the original one and consists of only K distinct colours.

We have illustrated this approach in Figure 4 on an example with the choice $K = 16$. In general, smaller values of K give a higher compression at the cost of poorer quality.

2.5. Extensions and generalizations

In the introduction, we said that we would like data points inside the same cluster to be similar to each other and dissimilar to data points in other clusters.

The notion of similarity that we have been using in our discussion so far is based on squared Euclidean distance: Two points $\boldsymbol{x}$ and $\boldsymbol{x}'$ are dissimilar if $\|\boldsymbol{x} - \boldsymbol{x}'\|^2$ is large.

It is possible to formulate the K-means algorithm using a general dissimilarity metric $V(\boldsymbol{x}, \boldsymbol{x}')$. In that case, the distortion function takes the form

$$J(z^{(1)}, \ldots, z^{(N)}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{1}\{z^{(i)} = k\}\, V(\boldsymbol{x}^{(i)}, \boldsymbol{\mu}_k)$$

and the resulting algorithm is a generalization of K-means known as the *K-medoids algorithm*.

Updating the cluster centroids in the K-medoids algorithm could be more complicated than in the K-means algorithm since the metric V may not offer the same analytical convenience as the squared Euclidean metric. One possible way of dealing with this problem is to restrict the cluster centroids $\boldsymbol{\mu}_k$ to be one of the data points $\boldsymbol{x}^{(i)}$ in cluster k. The cluster update step can then be implemented as a discrete search over the cluster.

Other extensions of the K-means algorithm are achieved by improving the cluster assignment step. In its current form, we need to compute the squared Euclidean distance a total of K times for each of the N data points. There have been proposals for speeding up this step. See [5] for a discussion and further references.

In section 4, we will derive the Gaussian mixture model which can be regarded as a generalization of K-means to a probabilistic framework. But first, we will give a general discussion of mixture models in the next section.

For more information on K-Means clustering, consult [5, 6].

3. Mixture Models and the EM Algorithm

The K-means algorithm provided us with a *hard clustering* of the data. This means that every data point $\boldsymbol{x}^{(i)}$ was assigned to a single cluster and we have no way of measuring the confidence of that assignment. It could be that $\boldsymbol{x}^{(i)}$ ends up being very close to its cluster's centroid in which case the hard assignment is reasonable. But it may also happen that $\boldsymbol{x}^{(i)}$ ends up being roughly midway between two centroids. In that case, the hard assignment to the nearest centroid seems a lot less appropriate.

Ideally, we would like our cluster assignments to come with a measure of confidence. That is, we want a *soft clustering*. This section introduces a popular and simple approach to soft clustering. We estimate the density of the data using a mixture model with K components. This will provide us with a posterior probabilities of each sample $\boldsymbol{x}^{(i)}$ belonging to the individual components of the model. By viewing the components of the model as clusters, we end up with a soft clustering of our data into K clusters.

We start off with a general desciption of finite mixture models in section 3.1. We then discuss how to fit the parameters of the model using the *EM algorithm*.

This section introduces mixture models and the EM algorithm in a general framework. In section 4, we will discuss Gaussian mixtures, the most popular instance of finite mixture models.

3.1. Mixture Models

Suppose we have a data set $S = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)}\}$ consisting of N independent observations of a random variable $\boldsymbol{X} \in \mathbb{R}^D$. We are interested in modelling the distribution of $\boldsymbol{X}$, i.e. we would like to use the data S to estimate the probability density function $p(\boldsymbol{x})$ of $\boldsymbol{X}$.

There are many ways of tackling this problem. The standard approach is to assume that $p(\boldsymbol{x}) = f(\boldsymbol{x}|\,\theta)$, i.e. that $p(\boldsymbol{x})$ belongs to some family of distributions parametrized by θ and then use the data set S to estimate θ with some inference method. For example, a common technique is to assume X is a Gaussian variable, such that

$$f(\boldsymbol{x}|\,\theta) = \mathcal{N}(\boldsymbol{x}|\,\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

and then use maximum likelihood estimation (explained in the section 3.2) to infer the value of the parameter $\theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

The standard approach works well for simple data sets, but is somewhat limitted in more complex systems. A simple way of extending this method is to assume that $p(\boldsymbol{x})$ is composed of K distinct *base distributions*. Let $f_k(\boldsymbol{x})$ denote the density function of the kth base distribution F_K. The resulting density of $\boldsymbol{X}$ is then

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k f_k(\boldsymbol{x}) \tag{3}$$

We require $\pi_k \geq 0$ for all k and $\sum_{k=1}^{K} = 1$ in order for the density function to be well-defined. Models of the form (3) are called *mixture models*.

Mixture models are a special case of *latent variable models*. To see why, suppose we have discrete latent variable Z taking values in $\{1, \dots, K\}$. The distribution of Z is fully specified by the quantities $\pi_1, \dots, \pi_K$, where $\pi_k = \Pr(Z = k)$, and is referred to as the *categorical distribution*, denoted $Z \sim \mathrm{Cat}(\pi_1, \dots, \pi_K)$. We can express the probability mass function of Z by

$$\mathrm{Cat}(z|\,\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{\mathbb{1}\{z=k\}}$$

Note that, due their probabilistic nature, the parameters $\boldsymbol{\pi} = (\pi_1, \dots, \pi_K)$ must satisfy $\pi_k \geq 0$, $\sum_{k=1}^{K} \pi_k = 1$.

Our mixture models can then be built as follows:

$$\begin{aligned} Z|\,\boldsymbol{\pi} &\sim \mathrm{Cat}(\boldsymbol{\pi}) \\ \boldsymbol{X}|\,Z = k &\sim F_k \end{aligned} \tag{4}$$

The joint distribution of $\boldsymbol{X}$ and Z is obtained from the product rule of probability:

$$p(\boldsymbol{x}, z) = p(z)p(\boldsymbol{x}|\,z)$$

Finally, to get the marginal ditribution of $\boldsymbol{X}$, we apply the sum rule of probability:

$$p(\boldsymbol{x}) = \sum_z p(\boldsymbol{x}, z) \tag{5}$$

The resulting model is thus derived as follows

$$\begin{aligned} p(\boldsymbol{x}) &= \sum_z p(\boldsymbol{x}, z) \\ &= \sum_z p(z)p(\boldsymbol{x}|\,z) \\ &= \sum_{k=1}^{K} \Pr(Z = k)p(\boldsymbol{x}|\,Z = k) \\ &= \sum_{k=1}^{K} \pi_k f_k(\boldsymbol{x}) \end{aligned}$$

This is the same as the mixture model in (3).

Typically, the base distributions are chosen to belong to the same parametric family and differ only in the value of their parameter, so that $F_k = F(\theta_k)$ and $f_k(\boldsymbol{x}) = f(\boldsymbol{x}|\,\theta_k)$. In that case, the mixture distribution is given by

$$p(\boldsymbol{x}|\,\boldsymbol{\pi}, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k f(\boldsymbol{x}|\,\theta_k) \tag{6}$$

where we have made the dependence on the parameters $\boldsymbol{\pi}$ and $\boldsymbol{\theta} = (\theta_1, \dots, \theta_K)$ explicit.

3.2. The EM algorithm

Given our data set $S = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)}\}$, how do we infer the parameters of our mixture model (6)? The standard inference method in frequentist statistics is to find the parameter value that maximizes the likelihood of the data. If our model for the joint distribution of $\boldsymbol{X}$ is $p(\boldsymbol{x}|\boldsymbol{\theta})$, then the likelihood of our data set S is given by

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} p(\boldsymbol{x}^{(i)}|\boldsymbol{\theta})$$

and we would like to find $\boldsymbol{\theta}$ that maximises $\mathcal{L}(\boldsymbol{\theta})$. This inference method is known as *maximum likelihood estimation*.

For simple models this problem can often be solved analytically. Unfortunately, this is generally not the case for latent variable models.

In this section, we will introduce the *Expectation-Maximization* (EM) algorithm [7]. It is powerful variational technique that can be used to find a maximum likelihood solution in models with latent variables.

In a general latent variable framework, we have a model for $p(\boldsymbol{x}, z\,|\,\boldsymbol{\theta})$, the joint distribution of the observed variable $\boldsymbol{X}$ and the latent variable Z, parametrized by $\boldsymbol{\theta}$. The log likelihood of the data S is

$$\begin{aligned}\ell(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)}|\boldsymbol{\theta}) \\ &= \sum_{i=1}^{N} \log\left(\sum_{z^{(i)}} p(\boldsymbol{x}^{(i)}, z^{(i)}|\boldsymbol{\theta})\right)\end{aligned} \tag{7}$$

The sum inside the logarithm is what generally makes this optimization problem analytically intractable. The expectation-maximization algorithm uses a simple iterative approach, that often has closed-form updates at each step. It alternated between two steps, the expectation (E) step and the maximization M step. In the E-step, it forms a function that is a lower bound to $\ell(\boldsymbol{\theta})$ and that is tight at the current value of $\boldsymbol{\theta}$. In the M-step, it finds $\boldsymbol{\theta}$ that maximizes this lower bound. This guarantees that we increase the value of $\ell(\theta)$ in each step until we find a local maximum.

We will derive the general form of the EM algorithm for maximizing the log likelihood function (7). We are treating Z as a discrete latent variable, but the derivation is analogous for continuous Z by simply exchanging the relevant sums with integrals.

We will make repeated use of *Jensen's Inequality* which states that, for any random variable Y and convex function f, we have

$$f(\mathbb{E}[Y]) \leq \mathbb{E}[f(Y)]$$

Further, if f is strictly convex, then the inequality is

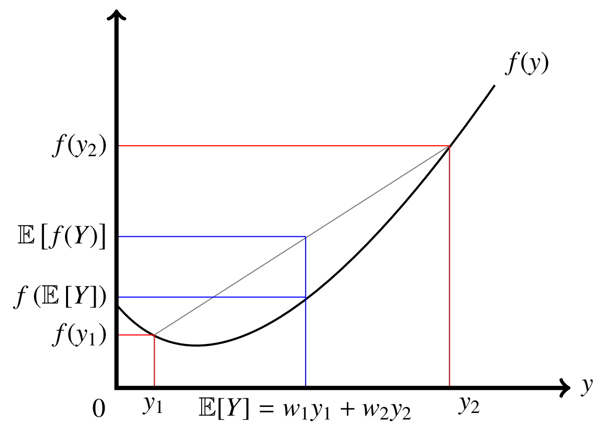

Figure 5: Illustration of Jensen's Inequality. The random variable Y can take two possible states y_1 and y_2 with probabilities w_1 and $w_2 = 1 - w_1$ respectively. It Y had only one possible state, say y_1, so that $\Pr(Y = y_1) = 1$, then $f(\mathbb{E}[Y])$ and $\mathbb{E}[f(Y)]$ would coincide.

strict and we have

$$\mathbb{E}[f(Y)] = f(\mathbb{E}[Y]) \iff \Pr(Y = E[Y]) = 1$$

An illustration of Jensen's inequality is shown in Figure 5. If f is concave, then the holds in inequality reverse since $(-f)$ is convex in that case.

We start by letting $Q_i(z^{(i)})$ be any probability function for $Z^{(i)}$. That means $Q_i(z^{(i)}) \geq 0$ and $\sum_{z^{(i)}} Q_i(z^{(i)}) = 1$. Our objective function can then be expressed as

$$\begin{aligned}\ell(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \log\left(\sum_{z^{(i)}} p(\boldsymbol{x}^{(i)}, z^{(i)}\,|\,\boldsymbol{\theta})\right) \\ &= \sum_{i=1}^{N} \log\left(\sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(\boldsymbol{x}^{(i)}, z^{(i)}\,|\,\boldsymbol{\theta})}{Q_i(z^{(i)})}\right) \\ &= \sum_{i=1}^{N} \log \mathbb{E}_{z^{(i)}\sim Q_i}\left[\frac{p(\boldsymbol{x}^{(i)}, z^{(i)}\,|\,\boldsymbol{\theta})}{Q_i(z^{(i)})}\right]\end{aligned}$$

where we used the definition of the expectation operator.

Applying Jensen's inequality to (7) and using the fact that the log function is strictly concave gives us

$$\begin{aligned}\ell(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \log \mathbb{E}_{z^{(i)}\sim Q_i}\left[\frac{p(\boldsymbol{x}^{(i)}, z^{(i)}\,|\,\boldsymbol{\theta})}{Q_i(z^{(i)})}\right] \\ &\geq \sum_{i=1}^{N} \mathbb{E}_{z^{(i)}\sim Q_i}\left[\log \frac{p(\boldsymbol{x}^{(i)}, z^{(i)}\,|\,\boldsymbol{\theta})}{Q_i(z^{(i)})}\right] \\ &= \sum_{i=1}^{N} \sum_{z^{(i)}} Q_i(z^{(i)}) \log\left(\frac{p(\boldsymbol{x}^{(i)}, z^{(i)}\,|\,\boldsymbol{\theta})}{Q_i(z^{(i)})}\right) := J(\boldsymbol{\theta}, \boldsymbol{Q})\end{aligned}$$

where we used $\boldsymbol{Q}$ to denote $\{Q_1, \ldots, Q_N\}$.

Algorithm 2 The EM algorithm for LVMs

```
1: Initialize $\boldsymbol{\theta}^{(current)}$

2: repeat
       E-Step:
3:     for $i = 1, \ldots, N$ do
4:         $Q_i(z^{(i)}) = p(z^{(i)} \,|\, \boldsymbol{x}^{(i)}, \boldsymbol{\theta}^{(current)})$
5:     end for

       M-Step:
6:     $\boldsymbol{\theta}^{(new)} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \sum_{z^{(i)}} Q_i(z^{(i)}) \log\left(\frac{p(\boldsymbol{x}^{(i)}, z^{(i)} \,|\, \boldsymbol{\theta})}{Q_i(z^{(i)})}\right)$
7:     Set $\theta^{(current)} = \theta^{(new)}$
8: until Convergence

9: return $\boldsymbol{\theta}^{(current)}, z^{(1)}, \ldots, z^{(N)}$
```

$J(\boldsymbol{\theta}, \boldsymbol{Q})$ is a lower bound to the log likelihood function $\ell(\boldsymbol{\theta})$ for *any* valid choice of probability functions $\boldsymbol{Q}$. What should our choice for the Q_i be?

The EM algorithm chooses $\boldsymbol{Q}$ in the following way. Suppose we currently have an estimate of our parameters $\boldsymbol{\theta}^{(current)}$. In the E-step, the EM algorithm chooses Q_i so that the lower bound is tight at $\boldsymbol{\theta}^{(current)}$, i.e. so that

$$J\left(\boldsymbol{\theta}^{(current)}, \boldsymbol{Q}\right) = \ell\left(\boldsymbol{\theta}^{(current)}\right)$$

Jensen's inequality tells us that this can be achieved by setting

$$\frac{p(\boldsymbol{x}^{(i)}, z^{(i)} \,|\, \boldsymbol{\theta}^{(current)})}{Q_i(z^{(i)})} = \mathit{constant}$$

for all values of $z^{(i)}$. This implies that

$$Q_i(z^{(i)}) \propto p(\boldsymbol{x}^{(i)}, z^{(i)} \,|\, \boldsymbol{\theta}^{(current)})$$

The constant of proportionality can be calculated by using the constraint that $\sum_{z^{(i)}} Q_i(z^{(i)}) = 1$. Thus,

$$\begin{aligned} Q_i(z^{(i)}) &= \frac{p(\boldsymbol{x}^{(i)}, z^{(i)} \,|\, \boldsymbol{\theta}^{(current)})}{\sum_{z^{(i)}} p(\boldsymbol{x}^{(i)}, z^{(i)} \,|\, \boldsymbol{\theta}^{(current)})} \\ &= \frac{p(\boldsymbol{x}^{(i)}, z^{(i)} \,|\, \boldsymbol{\theta}^{(current)})}{p(\boldsymbol{x}^{(i)} \,|\, \boldsymbol{\theta}^{(current)})} \qquad (8) \\ &= p(z^{(i)} \,|\, \boldsymbol{x}^{(i)}, \boldsymbol{\theta}^{(current)}) \end{aligned}$$

where the second line follows from the sum rule of probability (5) and the third line follows from the definition of conditional probabilities.

These quantities have an important interpretation in the context of soft clustering using mixture models. They are the posterior distributions of the cluster assignments of our data are referred to as *responsibilities*. We say that $\Pr(z^{(i)} = k \,|\, \boldsymbol{x}^{(i)}, \boldsymbol{\theta})$ is the responsibility that cluster k takes in explaining sample $\boldsymbol{x}^{(i)}$, and denote it $\gamma_k^{(i)}$.

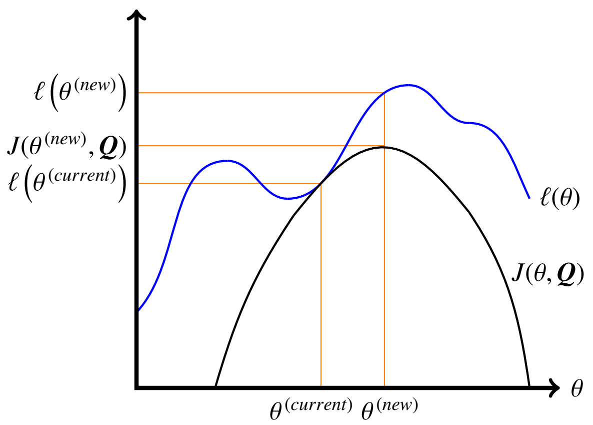

Figure 6: Illustration of the EM algorithm. $J(\theta, \boldsymbol{Q})$ (black curve) is a lower bound to $\ell(\theta)$ (blue curve). The E-step sets $\boldsymbol{Q}$ such that the blue curve and the black curve touch at $\theta^{(current)}$. The M-step finds $\theta^{(new)}$ that maximizes the black curve. Note how this brings us closer to a maximum of $\ell(\theta)$.

Setting $Q_i(z^{(i)})$ completes the E-step of the EM algorithm. In the M-Step, we update our parameters by finding $\boldsymbol{\theta}^{(new)}$ that maximizes this lower bound, keeping the Q_i fixed. In other words, we solve the following optimization problem

$$\begin{aligned} \boldsymbol{\theta}^{(new)} &= \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \overline{\boldsymbol{Q}}) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \sum_{z^{(i)}} \overline{Q}_i(z^{(i)}) \log\left(\frac{p(\boldsymbol{x}^{(i)}, z^{(i)} \,|\, \boldsymbol{\theta})}{\overline{Q}_i(z^{(i)})}\right) \end{aligned} \qquad (9)$$

where $\overline{Q}_i(z^{(i)}) = p(z^{(i)} \,|\, \boldsymbol{x}^{(i)}, \boldsymbol{\theta}^{(current)})$.[3] This optimization problem is simpler than the direct optimization of the original log likelihood (7). In particular, it is possible to find a simple closed-form solution for a wide range of models, including the Gaussian mixture model.

We have summarized the general form of the EM algorithm in Algorithm 2. Figure 6 provides a conceptual illustration of the method.

An important caveat is that the EM algorithm does not guarantee convergence to the global optimum. Thus, different initializations of $\boldsymbol{\theta}$ may lead to different outcomes.

Thus, similar to the local optima problem in K-means, we can try running the EM algorithm multiple

[3]Thus, it is possible to interpret the EM algorithm as a particular instance of the coordinate ascent algorithm applied to the function $J(\boldsymbol{\theta}, \boldsymbol{Q})$.

times with different initial values in order to increase the chance of obtaining a sufficiently good local optimum. This is no coincidence. We will see later that the K-means algorithm is, in fact, just an instance of the EM algorithm.

4. The Gaussian Mixture Model

This section discusses *Gaussian mixture models* (GMM) and their use in clustering. We begin with a general description of the model. Next we show how to fit the parameters of the model using the EM algorithm and derive the update formulas. We then briefly discuss model selection for GMMs. Following that, we give a short description and demonstration of our implementation of GMMs.

4.1. Clustering with the Gaussian Mixture Model

Consider again the general mixture model (4) and the resulting mixture distribution (6). The Gaussian mixture model is a particular instance of this model corresponding to the choice $F_k = \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. In other words, the GMM is a mixture model with Gaussian base distributions.

The probability density function for a general D-dimensional Gaussian variable $\boldsymbol{X}$ is

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{D}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left((\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{10}$$

where $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{X}]$ is the mean parameter and $\Sigma = \mathrm{Cov}(\boldsymbol{X})$ is the covariance parameter. The mean $\boldsymbol{\mu}$ is a D-dimensional vector and $\boldsymbol{\Sigma}$ is a $D \times D$ symmetric and positive definite matrix.

We can express the GMM with K mixture components as

$$\begin{aligned} Z\,|\,\pi_1,\ldots,\pi_K &\sim \mathrm{Cat}(\pi_1,\ldots,\pi_K) \\ X\,|\,Z = k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k &\sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \end{aligned}$$

and the resulting model for the joint distribution of $\boldsymbol{X}$ is given by

$$p(\boldsymbol{x}|\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) = \sum_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) \tag{11}$$

where we use $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ to denote the collections $(\pi_1,\ldots,\pi_K)$, $(\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K)$ and $(\boldsymbol{\Sigma}_1,\ldots,\boldsymbol{\Sigma}_K)$, respectively.

How can we use this model to obtain a clustering of a data set $S = \{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(N)}\}$? In the model, for each observation $\boldsymbol{x}^{(i)}$, there exists a categorical latent variable $z^{(i)}$ that contains the cluster assignment for $\boldsymbol{x}^{(i)}$. We do not actually observe $z^{(i)}$ but the model allows us to build their posterior distributions $p(z^{(i)}|\,\boldsymbol{x}^{(i)},\boldsymbol{\theta})$. These can be used to specify a soft clustering of the data:

$$\Pr(\boldsymbol{x}^{(i)} \text{ is in cluster } K) = \Pr(z^{(i)} = k|\,\boldsymbol{x}^{(i)},\boldsymbol{\theta})$$

As mentioned in the previous section, these quantities are referred to as responsibilities. We denote them with $\gamma_k^{(i)}$.

4.2. The EM algorithm for the GMM

The most popular inference method for the GMM is maximum likelihood estimation using the EM algorithm. Using (11), we can express the log likelihood of our data set S under the model as

$$\begin{aligned} \ell(\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) &= \sum_{i=1}^{N} \log\left(p(\boldsymbol{x}^{(i)}|\,\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}\right) \\ &= \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}^{(i)}|\,\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\right) \end{aligned}$$

We apply the EM algorithm since direct maximization of this log likelihood function is analytically intractable.

We start by setting some initial value for the parameters $\boldsymbol{\pi}^{(current)}$, $\boldsymbol{\mu}^{(current)}$ and $\boldsymbol{\Sigma}^{(current)}$. This is typically done randomly. However, we have to be careful when initializing the mixing coefficients $\boldsymbol{\pi}$ since they need satisfy the constraints that $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k \geq 0$ for all k. Once an initial value for the parameters is set, the EM algorithm alternates between the E-step and the M-step until some convergence criterion is met.

In the E-step (8), we need to find the $Q_i(z^{(i)})$. In the case of the GMM, this translates to calculating the responsibilities $\gamma_k^{(i)}$ for all values of i and k. Using Bayes' rule, we can derive a simple formula in terms of the current values of the parameter $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$:

$$\begin{aligned} \gamma_k^{(i)} &= \Pr(z^{(i)} = k|\,\boldsymbol{x}^{(i)},\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) \\ &= \frac{\Pr(z^{(i)} = k|\,\boldsymbol{\pi})p(\boldsymbol{x}^{(i)}|\,z^{(i)} = k,\boldsymbol{\mu},\boldsymbol{\Sigma})}{\sum_{j=1}^{K} \Pr(z^{(i)} = j|\,\boldsymbol{\pi})p(\boldsymbol{x}^{(i)}|\,z^{(i)} = j,\boldsymbol{\mu},\boldsymbol{\Sigma})} \\ &= \frac{\pi_k \mathcal{N}(\boldsymbol{x}^{(i)}|\,\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}^{(i)}|\,\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)} \end{aligned} \tag{12}$$

where, for clarity, we have temporarily dropped the "(*current*)" superscript.

In the M-Step (9), we find the parameters $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ that maximize $J(\boldsymbol{\theta},\overline{\boldsymbol{Q}})$. For Gaussian mixtures, J takes the form

$$J(\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma},\overline{\gamma}) = \sum_{i=1}^{N}\sum_{k=1}^{K} \overline{\gamma}_k^{(i)} \log\left(\frac{\pi_k \mathcal{N}(\boldsymbol{x}^{(i)}|\,\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\overline{\gamma}_k^{(i)}}\right)$$

where we put a bar on top of γ to indicate that the responsibilities are kept constant during the M-step. Rearranging and plugging in the pdf for the Gaussian pdf (10) gives us

$$\begin{aligned}
J(\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma},\overline{\gamma}) &= \sum_{i=1}^{N}\sum_{k=1}^{K}\overline{\gamma}_k^{(i)}\Big(\log\pi_k \\
&\quad + \log\mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) - \log\overline{\gamma}_k^{(i)}\Big) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K}\overline{\gamma}_k^{(i)}\Bigg(\log\pi_k - \frac{1}{2}D\log(2\pi) + \frac{1}{2}\log|\boldsymbol{\Sigma}_k^{-1}| \\
&\quad -\frac{1}{2}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}) - \log\overline{\gamma}_k^{(i)}\Bigg) \\
&= const + \sum_{i=1}^{N}\sum_{k=1}^{K}\overline{\gamma}_k^{(i)}\Bigg(\log\pi_k + \frac{1}{2}\log|\boldsymbol{\Sigma}_k^{-1}| \\
&\quad -\frac{1}{2}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu})\Bigg)
\end{aligned}$$

where *const* contains all the terms that are independent of the parameters.

In order to maximize J by need to take derivatives with respect to matrices and vectors. For later reference, we briefly state three useful matrix derivative identities: Suppose $\boldsymbol{y}$ is a D-dimensional vector and $\boldsymbol{A}$ is a $D\times D$ matrix with $|\boldsymbol{A}|>0$. Then

$$\nabla_{\boldsymbol{y}}(\boldsymbol{y}^T\boldsymbol{A}\boldsymbol{y}) = (\boldsymbol{A}+\boldsymbol{A}^T)\boldsymbol{y} \tag{13}$$

$$\nabla_{\boldsymbol{A}}(\boldsymbol{y}^T\boldsymbol{A}\boldsymbol{y}) = \boldsymbol{y}\boldsymbol{y}^T \tag{14}$$

$$\nabla_{\boldsymbol{A}}\log\left|\boldsymbol{A}^{-1}\right| = \boldsymbol{A}^T \tag{15}$$

To find the opimal parameter $\boldsymbol{\mu}_k$, we set the gradient of J with respect to $\boldsymbol{\mu}_k$ equal to zero:

$$\begin{aligned}
\nabla_{\boldsymbol{\mu}_k}J &= -\frac{1}{2}\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}\nabla_{\boldsymbol{\mu}_k}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k) \\
&= \frac{1}{2}\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}(\boldsymbol{\Sigma}_k^{-1}+\boldsymbol{\Sigma}_k^{-T})(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k) \\
&= \sum_{i=1}^{N}\overline{\gamma}_k^{(i)}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k) \\
&= 0
\end{aligned}$$

In the second line we used the matrix derivative identity (13) and in the third line we used the symmetry of $\boldsymbol{\Sigma}_k^{-1}$. Solving for $\boldsymbol{\mu}_k$ gives us the following update formula

$$\boldsymbol{\mu}_k^{(new)} = \frac{\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}\boldsymbol{x}^{(i)}}{\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}} \tag{16}$$

Next, we would like to maximize J with respect to $\boldsymbol{\Sigma}_k$. However, instead of taking the gradient with respect $\boldsymbol{\Sigma}_k$, it proves to be more convenient to take the gradient with respect to its inverse, $\boldsymbol{\Sigma}_k^{-1}$, instead.

$$\begin{aligned}
\nabla_{\boldsymbol{\Sigma}_k^{-1}}J &= \frac{1}{2}\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}\nabla_{\boldsymbol{\Sigma}_k^{-1}}\Big(\log\left|\boldsymbol{\Sigma}_k^{-1}\right| \\
&\quad -(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k)\Big) \\
&= \frac{1}{2}\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}\Big(\boldsymbol{\Sigma}_k - (\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k)(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k)^T\Big)
\end{aligned}$$

where we used matrix identities (14) and (15) and symmetry of $\boldsymbol{\Sigma}_k$. Setting the derivative to zero and rearranging for Σ_k yields the update formula

$$\boldsymbol{\Sigma}_k^{(new)} = \frac{\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k^{(new)})(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_k^{(new)})^T}{\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}} \tag{17}$$

Note that it is common to use the updated value $\boldsymbol{\mu}_k^{(new)}$ in the update formula for $\boldsymbol{\Sigma}$ as we have done here.

Finally, when maximizing J with respect to the mixing coefficents $\boldsymbol{\pi}$ we need to respect the constraint that $\pi_k \geq 0$ and $\sum_{k=1}^{K}\pi_k = 1$. For our specific optimization problem, the most straightforward approach is to ignore the inequality constraints $\pi_k \geq 0$, and use the method of Lagrange multipliers to solve the resulting problem.

Let $\lambda > 0$ denote the Lagrange multiplier of the equality constraint $\sum_{k=1}^{K} = 1$. To find the optimal π_k, the method performs the unconstrained optimization of $\left(J + \lambda(1-\sum_{k=1}^{K}\pi_k)\right)$ with respect to λ and the π_k. We may do so because, as it turns out, we will get a solution that happens to satisfy the inequality constraints. Setting the derivative with respect to λ to zero yields

$$\begin{aligned}
\frac{\partial}{\partial\lambda}\left(J+\lambda(1-\sum_{k=1}^{K}\pi_k)\right) &= 1-\sum_{k=1}^{K}\pi_k \\
&= 0
\end{aligned} \tag{18}$$

which is simply our initial sum-to-one constraint. Differentiating with respect to π_k, we get

$$\begin{aligned}
\frac{\partial}{\partial\pi_k}\left(J+\lambda(1-\sum_{k=1}^{K}\pi_k)\right) &= \left(\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}\frac{\partial}{\partial\pi_k}\log\pi_k\right)-\lambda \\
&= \left(\sum_{i=1}^{N}\frac{\overline{\gamma}_k^{(i)}}{\pi_k}\right)-\lambda
\end{aligned}$$

Setting the derivative to zero and solving for π_k gives us

$$\pi_k = \frac{\sum_{i=1}^{N}\overline{\gamma}_k^{(i)}}{\lambda} \tag{19}$$

Algorithm 3 EM algorithm for GMMs

```
 1: Initialize $\boldsymbol{\pi}^{(current)}$, $\boldsymbol{\mu}^{(current)}$ and $\boldsymbol{\Sigma}^{(current)}$

 2: repeat
        E-Step:
 3:     for $i = 1, \ldots, N$ do
 4:         for $k = 1, \ldots, K$ do
 5:             Set $\gamma_k^{(i)} = \frac{\pi_k^{(current)} \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k^{(current)},\boldsymbol{\Sigma}_k^{(current)})}{\sum_{j=1}^K \pi_j^{(current)} \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_j^{(current)},\boldsymbol{\Sigma}_j^{(current)})}$
 6:         end for
 7:     end for

        M-Step:
 8:     for $k = 1, \ldots, K$ do
 9:         Set $N_k = \sum_{i=1}^N \gamma_k^{(i)}$
10:         Set $\pi_k^{(new)} = \frac{N_k}{N}$
11:         Set $\boldsymbol{\mu}_k^{(new)} = \frac{1}{N_k} \sum_{i=1}^N \gamma_k^{(i)} \boldsymbol{x}^{(i)}$
12:         Set $\boldsymbol{\Sigma}_k^{(new)}$
                $= \frac{1}{N_k} \sum_{i=1}^N \gamma_k^{(i)} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k^{(new)})(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k^{(new)})^T$

13:         Update $\boldsymbol{\pi}^{(current)} = \boldsymbol{\pi}^{(new)}$, $\boldsymbol{\mu}^{(current)} = \boldsymbol{\mu}^{(new)}$
              and $\boldsymbol{\Sigma}^{(current)} = \boldsymbol{\Sigma}^{(new)}$
14:     end for
15: until Convergence

16: return $\gamma_k^{(i)}, \pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \quad i = 1, \ldots, N, k = 1, \ldots, K$
```

Next, we note that $\sum_{k=1}^N \overline{\gamma}_k^{(i)} = 1$ due to their probabilistic nature (12). Taking the sum over k on both sides of (19) and making use of the sum-to-one constraint (18) therefore gives us

$$\begin{aligned} 1 &= \frac{\sum_{k=1}^K \sum_{i=1}^N \overline{\gamma}_k^{(i)}}{\lambda} \\ \Rightarrow \lambda &= \sum_{i=1}^N \sum_{k=1}^K \overline{\gamma}_k^{(i)} \\ &= \sum_{i=1}^N 1 \\ &= N \end{aligned}$$

Hence, we get the following update formulae for $\boldsymbol{\pi}$

$$\pi_k^{(new)} = \frac{\sum_{i=1}^N \overline{\gamma}_k^{(i)}}{N}. \tag{20}$$

This completes the M-step of the EM algorithm. We have summarized the EM algorithm for Gaussian mixture models in Algorithm 3.

In the context of clustering, there is an intuitive interpretation for the model parameters. We have already discussed the role of the responsibility $\gamma_k^{(i)}$ as a soft assignment of sample $\boldsymbol{x}^{(i)}$ to cluster k. These are analoguous to hard assignment indicator functions $\mathbb{1}\{z^{(i)} = k\}$ that were used in the K-means algorithm.

The quantity

$$N_k = \sum_{i=1}^N \gamma_k^{(i)}$$

can be interpreted as the *effective size* of cluster k.

With these interpretations of $\gamma_k^{(i)}$ and N_k, it is natural to view $\boldsymbol{\mu}_k$ as the mean of cluster k and $\boldsymbol{\Sigma}_k$ as its variance, measuring the spread of the data belonging to cluster k.

4.3. KMeans as limit of GMM

We have already noted the similarity between the formula for the mean parameters $\boldsymbol{\mu}_k$ in the GMM (16) and the formula for the cluster centroids in the K-means algorithm (2). It is in fact possible to derive the K-means algorithm as a special case of the Gaussian mixture model.

To show how, let us fix

$$\pi_k = \frac{1}{K}$$

and

$$\boldsymbol{\Sigma}_k = \sigma^2 \boldsymbol{I}_D$$

for all values of k, where σ^2 is a fixed positive scalar and we used $\boldsymbol{I}_D$ to denote the $D \times D$ identity matrix.

Further, let us approximate the responsibilities by indicator functions

$$\gamma_k^{(i)} = \mathbb{1}\{z^{(i)} = k^*\}$$

where

$$k^* = \arg\max_k \gamma_k^{(i)}.$$

This turns our soft clustering into hard cluster assignments.

Due to the spherical covariance matrix $\sigma^2 \boldsymbol{I}_D$, the cluster k whose mean parameter $\boldsymbol{\mu}_k$ is closest to $\boldsymbol{x}^{(i)}$ will hold the largest responsibility in explaining sample i. Thus, k^* will be set to the identity of the cluster that is closest to $\boldsymbol{x}^{(i)}$. This gives us the cluster assignment step of the K-means algorithm (1).

Next, we need to update the cluster means $\boldsymbol{\mu}_k$. Using our approximations for $\gamma_k^{(i)}$, the GMM update formula for $\boldsymbol{\mu}_k$ reduces to

$$\begin{aligned} \boldsymbol{\mu}_k &= \frac{\sum_{i=1} \gamma_l^{(i)} \boldsymbol{x}^{(i)}}{\sum_{i=1}^N \gamma_k^{(i)}} \\ &= \frac{\sum_{i=1}^N \mathbb{1}\{z^{(i)} = k\} \boldsymbol{x}^{(i)}}{\sum_{i=1}^N \mathbb{1}\{z^{(i)} = k\}} \end{aligned}$$

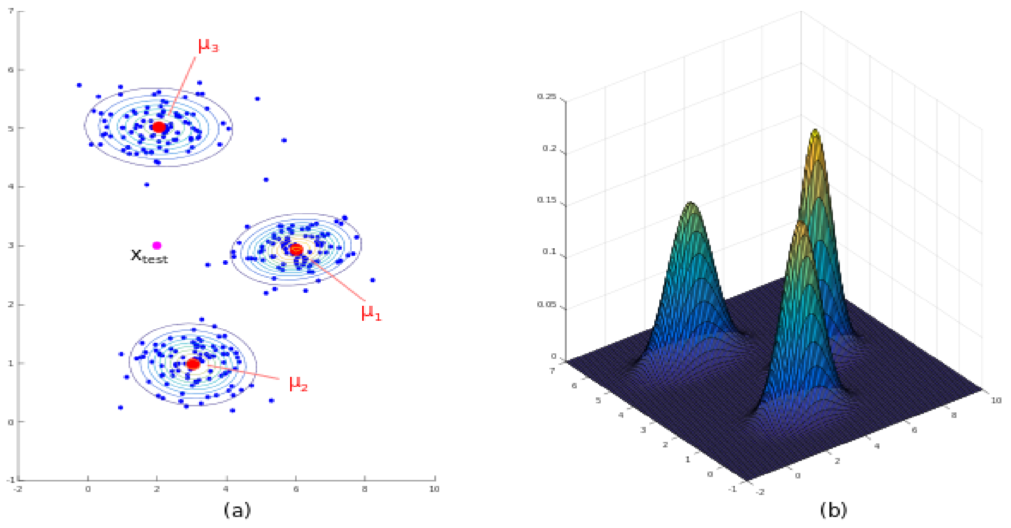

Figure 7: Illustration of GMM on the 2d data set from Figure 1 with $K = 3$. (a) Plot of the cluster means and variances indicated with ellipses. We have added a potential anomaly at $\boldsymbol{x}_{test}$. (b) Surface plot of the estimated density function $p(\boldsymbol{x})$.

giving us the same update formula that was used in the K-Means algorithm (2).

Thus, we see that the K-means algorithm is just an instance of the EM algorithm.

4.4. Model Selection in the Gaussian mixture model

Similar to the K-means algorithm, the only input parameter in the Gaussian mixture model is the number of clusters K. Thus, model selection in the context of the GMM is about selecting the optimal number of mixture components.

A simple model selection tool is the "elbow method". We have already discussed this method in section 2.3 in the context about K-means. Instead of the distortion function, we plot the negative value of the log likelihood, $-\ell(\boldsymbol{\theta})$, at the optimum parameter value of $\boldsymbol{\theta}$ against K and try to identify a "kink". Of course, this approach comes with the same drawbacks that were discussed earlier.

Alternative methods have been proposed that make use of the fact that the GMM is a probabilistic method. For instance, a simple practical approach is to fit the GMM for a range of values of K and use a statistical model selection tool such as the *Bayesian Information Criterion* (BIC) for each of the models. We then select the value of K with the lowest BIC score. See [8] for more details.

In general, model selection for finite mixture models is a difficult problem and virtually all model selection tools have significant drawbacks. One way of circumventing the problem is to make the move to Bayesian non-parametric models that allow for automatic inference of the number of clusters. A popular approach to non-parametric clustering is to use the Dirichlet process mixture model.

For more information on model selection in mixture models, see [6].

4.5. Implementation Details and Demonstration

Beside the K-means algorithm, we have also implemented the Gaussian mixture model in C++, again using Armadillo [3]. Implementation of the pseudo-code from Algorithm 3 is relatively straightforward. The only additions that we have made are

1. We added an upper limit to the number of iterations of the EM algorihtm.
2. We used a mixed error test as convergence criterion. This means that we stop the algorithm if

$$|\ell(\boldsymbol{\theta}^{(new)}) - \ell(\boldsymbol{\theta}^{(current)})| < \epsilon\left(1 + |\ell(\boldsymbol{\theta}^{(current)})|\right)$$

   where $\epsilon > 0$ is the target error.
3. Similar to K-means, we perform multiple runs of the EM algorithm with different random initializations to mitigate the risk of getting a poor local solution.

In Figure 7, we illustrate a common application of the GMM. We applied the model to the same data set as in Figure 1 with $K = 3$. The resulting clusters are shown in panel (a), where we used ellipses to indicate the covariance parameters. Panel (b) shows a surface plot of the estimated density function $p(\boldsymbol{x})$.

We also show how the Gaussian mixture model can be applied to *anomaly detection*. In panel (a), we have included a potential anomaly at the point marked as $\boldsymbol{x}_{test}$. Once we have an estimate for $p(\boldsymbol{x})$, we can evaluate it at $\boldsymbol{x}_{test}$. We flag $\boldsymbol{x}_{test}$ as an anomaly if

$$p(\boldsymbol{x}_{test}) < \epsilon$$

where $\epsilon > 0$ as a preset threshold.

5. Conclusion and Further Work

The EM algorithm is one of the most important techniques in machine learning and statistics. As such, many extensions and variations of it have been proposed in the literature. [6] gives a short description of several extensions of EM, while [9] offers a more in-depth treatment.

An important extension is *online EM*. We discussed the EM algorithm in the context of *batch learning*. However, when dealing with large data sets, this can be very slow, since the algorithm needs to go through the entire data set in order to produce a single parameter update. An alternative to batch learning is *online learning*. In online learning algorithms, each iteration considers only a single data point in order to update the parameters. In practice, this can lead to significant speedups.

There are two main variants of an online version of the EM algorithm. They are known as *incremental EM* and *stepwise EM* and are discussed in [10].

We showed how the EM algorithm can be used to fit latent variable models. In particular, we derived the EM algorithm for the Gaussian mixture model and K-means.

Both, K-means and Gaussian mixtures play an important role in unsupervised machine learning. They offer simple and intuitive approaches to clustering and are straightforward to implement. Typically, they are included in any major machine learning software package. In practice, this makes them a good "first thing to try" in a wide range of clustering problems.

There a various ways in which the two models have been extended and generalized.

As an example, the *elliptical K-means* algorithm [11] offers a hard-assignment version of the GMM with general covariance matrices. Other extensions to the GMM seek to find better ways of infering the parameters of the model, for instance through Markov Chain Monte Carlo methods.

However, as we said before, the biggest limitations of finite mixture models is that we need to manually set the number of components K. Model selection methods are fraught with difficulties and so it may be beneficial to use Bayesian nonparametric approaches to clustering that automatically infer the number of clusters. The most widely used of these methods is the Dirichlet process mixture model. For more information, see [6, 12].

Bibliography

[1] S. Lloyd, Least squares quantization in pcm, Information Theory, IEEE Transactions on 28 (2) (1982) 129–137.

[2] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, University of California Press, Berkeley, Calif., 1967, pp. 281–297.

[3] C. Sanderson, Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments, Tech. rep., NICTA (2010).

[4] D. A. Forsyth, J. Ponce, Computer Vision: A Modern Approach, Prentice Hall Professional Technical Reference, 2002.

[5] C. M. Bishop, Pattern recognition and machine learning, Vol. 1, springer, 2006.

[6] K. P. Murphy, Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.

[7] D. B. R. A. P. Dempster, N. M. Laird, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society. Series B (Methodological) 39 (1) (1977) 1–38.

[8] C. Fraley, A. E. Raftery, Model-based clustering, discriminant analysis, and density estimation, Journal of the American statistical Association 97 (458) (2002) 611–631.

[9] G. McLachlan, T. Krishnan, The EM Algorithm and Extensions, John Wiley & Sons, New York, 1997.

[10] P. Liang, D. Klein, Online em for unsupervised models, in: Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics, Association for Computational Linguistics, 2009, pp. 611–619.

[11] K.-K. Sung, T. Poggio, Example-based learning for view-based human face detection, Pattern Analysis and Machine Intelligence, IEEE Transactions on 20 (1) (1998) 39–51.

[12] Y. W. Teh, Dirichlet processes, in: Encyclopedia of Machine Learning, Springer, 2010.