11 *Mixture models and the EM algorithm*

11.1 Latent variable models

In Chapter 10 we showed how graphical models can be used to define high-dimensional joint probability distributions. The basic idea is to model dependence between two variables by adding an edge between them in the graph. (Technically the graph represents conditional independence, but you get the point.)

An alternative approach is to assume that the observed variables are correlated because they arise from a hidden common "cause". Model with hidden variables are also known as **latent variable models** or **LVM**s. As we will see in this chapter, such models are harder to fit than models with no latent variables. However, they can have significant advantages, for two main reasons. First, LVMs often have fewer parameters than models that directly represent correlation in the visible space. This is illustrated in Figure 11.1. If all nodes (including H) are binary and all CPDs are tabular, the model on the left has 17 free parameters, whereas the model on the right has 59 free parameters.

Second, the hidden variables in an LVM can serve as a **bottleneck**, which computes a compressed representation of the data. This forms the basis of unsupervised learning, as we will see. Figure 11.2 illustrates some generic LVM structures that can be used for this purpose. In general there are L latent variables, $z_{i1}, \ldots, z_{IL}$, and D visible variables, $x_{i1}, \ldots, x_{iD}$, where usually $D \gg L$. If we have $L > 1$, there are many latent factors contributing to each observation, so we have a many-to-many mapping. If $L = 1$, we we only have a single latent variable; in this case, z_i is usually discrete, and we have a one-to-many mapping. We can also have a many-to-one mapping, representing different competing factors or causes for each observed variable; such models form the basis of probabilistic matrix factorization, discussed in Section 27.6.2. Finally, we can have a one-to-one mapping, which can be represented as $\mathbf{z}_i \rightarrow \mathbf{x}_i$. By allowing $\mathbf{z}_i$ and/or $\mathbf{x}_i$ to be vector-valued, this representation can subsume all the others. Depending on the form of the likelihood $p(\mathbf{x}_i|\mathbf{z}_i)$ and the prior $p(\mathbf{z}_i)$, we can generate a variety of different models, as summarized in Table 11.1.

11.2 Mixture models

The simplest form of LVM is when $z_i \in \{1, \ldots, K\}$, representing a discrete latent state. We will use a discrete prior for this, $p(z_i) = \text{Cat}(\boldsymbol{\pi})$. For the likelihood, we use $p(\mathbf{x}_i|z_i = k) = p_k(\mathbf{x}_i)$,

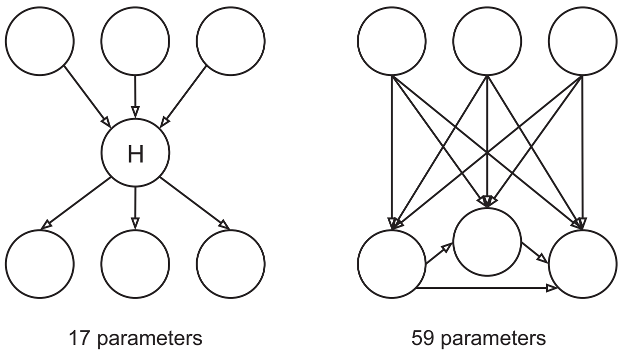


Figure 11.1 A DGM with and without hidden variables. The leaves represent medical symptoms. The roots represent primary causes, such as smoking, diet and exercise. The hidden variable can represent mediating factors, such as heart disease, which might not be directly visible.

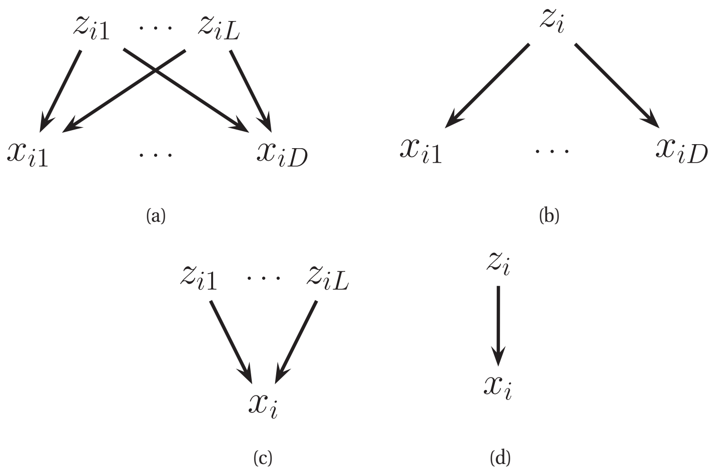


Figure 11.2 A latent variable model represented as a DGM. (a) Many-to-many. (b) One-to-many. (c) Many-to-one. (d) One-to-one.

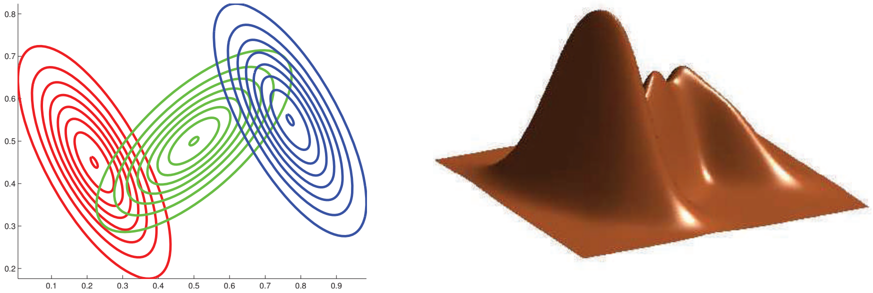
where p_k is the k'th **base distribution** for the observations; this can be of any type. The overall model is known as a **mixture model**, since we are mixing together the K base distributions as follows:

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k p_k(\mathbf{x}_i|\boldsymbol{\theta}) \tag{11.1}$$

This is a **convex combination** of the p_k's, since we are taking a weighted sum, where the **mixing weights** π_k satisfy $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$. We give some examples below.

$p(\mathbf{x}_i\|\mathbf{z}_i)$	$p(\mathbf{z}_i)$	Name	Section
MVN	Discrete	Mixture of Gaussians	11.2.1
Prod. Discrete	Discrete	Mixture of multinomials	11.2.2
Prod. Gaussian	Prod. Gaussian	Factor analysis/ probabilistic PCA	12.1.5
Prod. Gaussian	Prod. Laplace	Probabilistic ICA/ sparse coding	12.6
Prod. Discrete	Prod. Gaussian	Multinomial PCA	27.2.3
Prod. Discrete	Dirichlet	Latent Dirichlet allocation	27.3
Prod. Noisy-OR	Prod. Bernoulli	BN20/ QMR	10.2.3
Prod. Bernoulli	Prod. Bernoulli	Sigmoid belief net	27.7

Table 11.1 Summary of some popular directed latent variable models. Here "Prod" means product, so "Prod. Discrete" in the likelihood means a factored distribution of the form $\prod_j \text{Cat}(x_{ij}|\mathbf{z}_i)$, and "Prod. Gaussian" means a factored distribution of the form $\prod_j \mathcal{N}(x_{ij}|\mathbf{z}_i)$. "PCA" stands for "principal components analysis". "ICA" stands for "indepedendent components analysis".



(a) (b)

Figure 11.3 A mixture of 3 Gaussians in 2d. (a) We show the contours of constant probability for each component in the mixture. (b) A surface plot of the overall density. Based on Figure 2.23 of (Bishop 2006a). Figure generated by `mixGaussPlotDemo`.

11.2.1 Mixtures of Gaussians

The most widely used mixture model is the **mixture of Gaussians** (MOG), also called a **Gaussian mixture model** or **GMM**. In this model, each base distribution in the mixture is a multivariate Gaussian with mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$. Thus the model has the form

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{11.2}$$

Figure 11.3 shows a mixture of 3 Gaussians in 2D. Each mixture component is represented by a different set of eliptical contours. Given a sufficiently large number of mixture components, a GMM can be used to approximate any density defined on $\mathbb{R}^D$.

11.2.2 Mixture of multinoullis

We can use mixture models to define density models on many kinds of data. For example, suppose our data consist of D-dimensional bit vectors. In this case, an appropriate class-conditional density is a product of Bernoullis:

$$p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta}) = \prod_{j=1}^{D} \text{Ber}(x_{ij}|\mu_{jk}) = \prod_{j=1}^{D} \mu_{jk}^{x_{ij}}(1-\mu_{jk})^{1-x_{ij}} \tag{11.3}$$

where μ_{jk} is the probability that bit j turns on in cluster k.

The latent variables do not have to any meaning, we might simply introduce latent variables in order to make the model more powerful. For example, one can show (Exercise 11.8) that the mean and covariance of the mixture distribution are given by

$$\mathbb{E}[\mathbf{x}] = \sum_k \pi_k \boldsymbol{\mu}_k \tag{11.4}$$

$$\text{cov}[\mathbf{x}] = \sum_k \pi_k [\boldsymbol{\Sigma}_k + \boldsymbol{\mu}_k \boldsymbol{\mu}_k^T] - \mathbb{E}[\mathbf{x}]\,\mathbb{E}[\mathbf{x}]^T \tag{11.5}$$

where $\boldsymbol{\Sigma}_k = \text{diag}(\mu_{jk}(1-\mu_{jk}))$. So although the component distributions are factorized, the joint distribution is not. Thus the mixture distribution can capture correlations between variables, unlike a single product-of-Bernoullis model.

11.2.3 Using mixture models for clustering

There are two main applications of mixture models. The first is to use them as a **black-box** density model, $p(\mathbf{x}_i)$. This can be useful for a variety of tasks, such as data compression, outlier detection, and creating generative classifiers, where we model each class-conditional density $p(\mathbf{x}|y = c)$ by a mixture distribution (see Section 14.7.3).

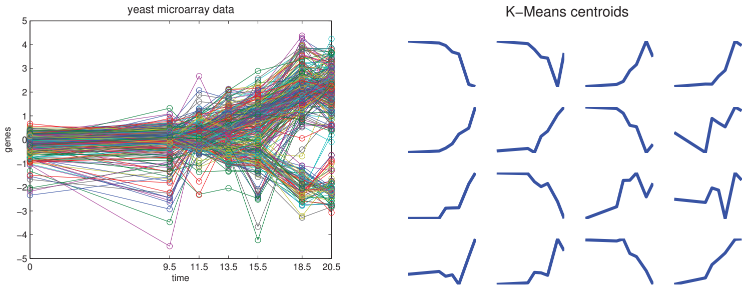
The second, and more common, application of mixture models is to use them for clustering. We discuss this topic in detail in Chapter 25, but the basic idea is simple. We first fit the mixture model, and then compute $p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta})$, which represents the posterior probability that point i belongs to cluster k. This is known as the **responsibility** of cluster k for point i, and can be computed using Bayes rule as follows:

$$r_{ik} \triangleq p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}) = \frac{p(z_i = k|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta})}{\sum_{k'=1}^{K} p(z_i = k'|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k', \boldsymbol{\theta})} \tag{11.6}$$

This procedure is called **soft clustering**, and is identical to the computations performed when using a generative classifier. The difference between the two models only arises at training time: in the mixture case, we never observe z_i, whereas with a generative classifier, we do observe y_i (which plays the role of z_i).

We can represent the amount of uncertainty in the cluster assignment by using $1 - \max_k r_{ik}$. Assuming this is small, it may be reasonable to compute a **hard clustering** using the MAP estimate, given by

$$z_i^* = \arg\max_k r_{ik} = \arg\max_k \log p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta}) + \log p(\mathbf{z}_i = k|\boldsymbol{\theta}) \tag{11.7}$$



(a) (b)

Figure 11.4 (a) Some yeast gene expression data plotted as a time series. (c) Visualizing the 16 cluster centers produced by K-means. Figure generated by `kmeansYeastDemo`.

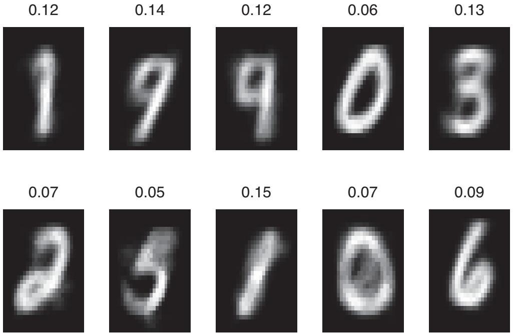


Figure 11.5 We fit a mixture of 10 Bernoullis to the binarized MNIST digit data. We show the MLE for the corresponding cluster means, $\boldsymbol{\mu}_k$. The numbers on top of each image represent the mixing weights $\hat{\pi}_k$. No labels were used when training the model. Figure generated by `mixBerMnistEM`.

Hard clustering using a GMM is illustrated in Figure 1.8, where we cluster some data representing the height and weight of people. The colors represent the hard assignments. Note that the identity of the labels (colors) used is immaterial; we are free to rename all the clusters, without affecting the partitioning of the data; this is called **label switching**.

Another example is shown in Figure 11.4. Here the data vectors $\mathbf{x}_i \in \mathbb{R}^7$ represent the expression levels of different genes at 7 different time points. We clustered them using a GMM. We see that there are several kinds of genes, such as those whose expression level goes up monotonically over time (in response to a given stimulus), those whose expression level goes down monotonically, and those with more complex response patterns. We have clustered the series into $K = 16$ groups. (See Section 11.5 for details on how to choose K.) For example, we can represent each cluster by a **prototype** or **centroid**. This is shown in Figure 11.4(b).

As an example of clustering binary data, consider a binarized version of the MNIST handwritten digit dataset (see Figure 1.5(a)), where we ignore the class labels. We can fit a mixture of

Bernoullis to this, using $K = 10$, and then visualize the resulting centroids, $\hat{\boldsymbol{\mu}}_k$, as shown in Figure 11.5. We see that the method correctly discovered some of the digit classes, but overall the results aren't great: it has created multiple clusters for some digits, and no clusters for others. There are several possible reasons for these "errors":

- The model is very simple and does not capture the relevant visual characteristics of a digit. For example, each pixel is treated independently, and there is no notion of shape or a stroke.

- Although we think there should be 10 clusters, some of the digits actually exhibit a fair degree of visual variety. For example, there are two ways of writing 7's (with and without the cross bar). Figure 1.5(a) illustrates some of the range in writing styles. Thus we need $K \gg 10$ clusters to adequately model this data. However, if we set K to be large, there is nothing in the model or algorithm preventing the extra clusters from being used to create multiple versions of the same digit, and indeed this is what happens. We can use model selection to prevent too many clusters from being chosen but what looks visually appealing and what makes a good density estimator may be quite different.

- The likelihood function is not convex, so we may be stuck in a local optimum, as we explain in Section 11.3.2.

This example is typical of mixture modeling, and goes to show one must be very cautious trying to "interpret" any clusters that are discovered by the method. (Adding a little bit of supervision, or using informative priors, can help a lot.)

11.2.4 Mixtures of experts

Section 14.7.3 described how to use mixture models in the context of generative classifiers. We can also use them to create discriminative models for classification and regression. For example, consider the data in Figure 11.6(a). It seems like a good model would be three different linear regression functions, each applying to a different part of the input space. We can model this by allowing the mixing weights and the mixture densities to be input-dependent:

$$p(y_i|\mathbf{x}_i, z_i = k, \boldsymbol{\theta}) = \mathcal{N}(y_i|\mathbf{w}_k^T\mathbf{x}_i, \sigma_k^2) \tag{11.8}$$

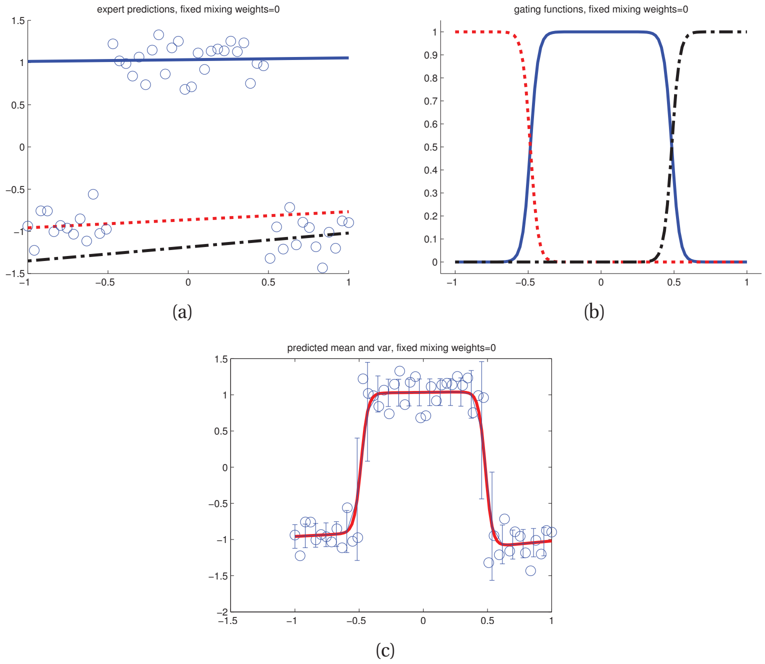
$$p(z_i|\mathbf{x}_i, \boldsymbol{\theta}) = \mathrm{Cat}(z_i|\mathcal{S}(\mathbf{V}^T\mathbf{x}_i)) \tag{11.9}$$

See Figure 11.7(a) for the DGM.

This model is called a **mixture of experts** or MoE (Jordan and Jacobs 1994). The idea is that each submodel is considered to be an "expert" in a certain region of input space. The function $p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta})$ is called a **gating function**, and decides which expert to use, depending on the input values. For example, Figure 11.6(b) shows how the three experts have "carved up" the 1d input space, Figure 11.6(a) shows the predictions of each expert individually (in this case, the experts are just linear regression models), and Figure 11.6(c) shows the overall prediction of the model, obtained using

$$p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) = \sum_k p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta})p(y_i|\mathbf{x}_i, z_i = k, \boldsymbol{\theta}) \tag{11.10}$$

We discuss how to fit this model in Section 11.4.3.

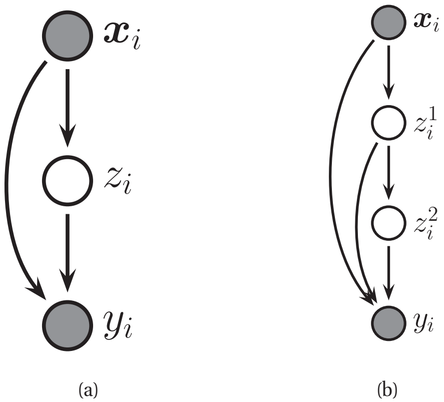


(a)

(b)

(c)

Figure 11.6 (a) Some data fit with three separate regression lines. (b) Gating functions for three different "experts". (c) The conditionally weighted average of the three expert predictions. Figure generated by `mixexpDemo`.



(a)

(b)

Figure 11.7 (a) A mixture of experts. (b) A hierarchical mixture of experts.

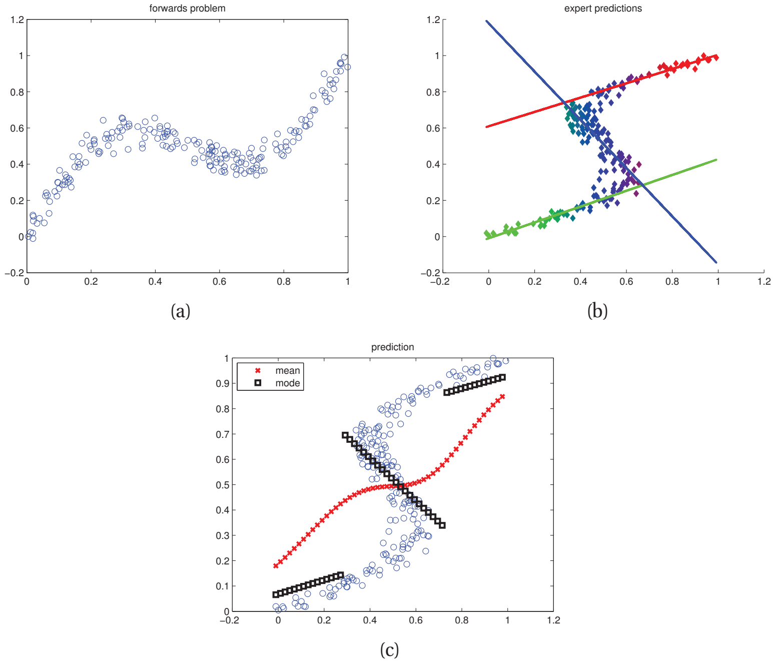


Figure 11.8 (a) Some data from a simple forwards model. (b) Some data from the inverse model, fit with a mixture of 3 linear regressions. Training points are color coded by their responsibilities. (c) The predictive mean (red cross) and mode (black square). Based on Figures 5.20 and 5.21 of (Bishop 2006b). Figure generated by `mixexpDemoOneToMany`.

It should be clear that we can "plug in" any model for the expert. For example, we can use neural networks (Chapter 16) to represent both the gating functions and the experts. The result is known as a **mixture density network**. Such models are slower to train, but can be more flexible than mixtures of experts. See (Bishop 1994) for details.

It is also possible to make each expert be itself a mixture of experts. This gives rise to a model known as the **hierarchical mixture of experts**. See Figure 11.7(b) for the DGM, and Section 16.2.6 for further details.

11.2.4.1 Application to inverse problems

Mixtures of experts are useful in solving **inverse problems**. These are problems where we have to invert a many-to-one mapping. A typical example is in robotics, where the location of the end effector (hand) $\mathbf{y}$ is uniquely determined by the joint angles of the motors, $\mathbf{x}$. However, for any given location $\mathbf{y}$, there are many settings of the joints $\mathbf{x}$ that can produce it. Thus the inverse mapping $\mathbf{x} = f^{-1}(\mathbf{y})$ is not unique. Another example is **kinematic tracking** of people from video (Bo et al. 2008), where the mapping from image appearance to pose is not unique, due to self occlusion, etc.

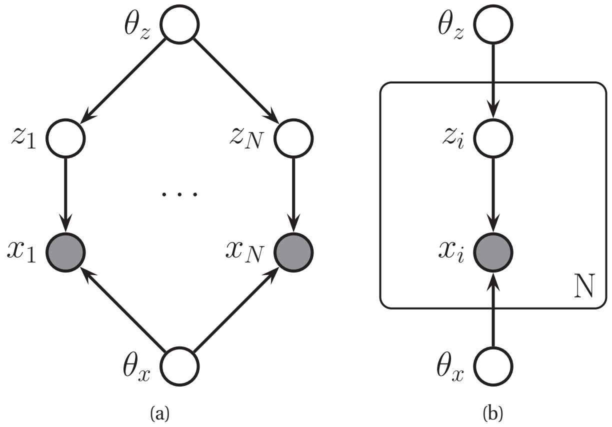


Figure 11.9 A LVM represented as a DGM. Left: Model is unrolled for N examples. Right: same model using plate notation.

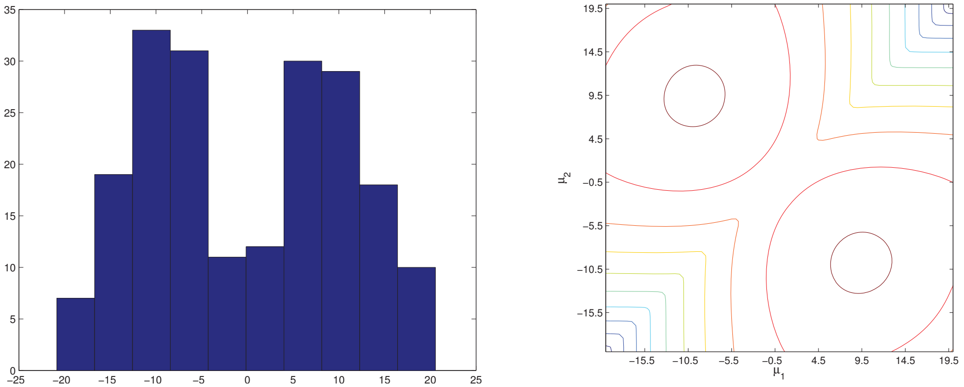
A simpler example, for illustration purposes, is shown in Figure 11.8(a). We see that this defines a function, $y = f(x)$, since for every value x along the horizontal axis, there is a unique response y. This is sometimes called the **forwards model**. Now consider the problem of computing $x = f^{-1}(y)$. The corresponding inverse model is shown in Figure 11.8(b); this is obtained by simply interchanging the x and y axes. Now we see that for some values along the horizontal axis, there are multiple possible outputs, so the inverse is not uniquely defined. For example, if $y = 0.8$, then x could be 0.2 or 0.8. Consequently, the predictive distribution, $p(x|y, \boldsymbol{\theta})$ is multimodal.

We can fit a mixture of linear experts to this data. Figure 11.8(b) shows the prediction of each expert, and Figure 11.8(c) shows (a plugin approximation to) the posterior predictive mode and mean. Note that the posterior mean does not yield good predictions. In fact, any model which is trained to minimize mean squared error — even if the model is a flexible nonlinear model, such as neural network — will work poorly on inverse problems such as this. However, the posterior mode, where the mode is input dependent, provides a reasonable approximation.

11.3 Parameter estimation for mixture models

We have seen how to compute the posterior over the hidden variables given the observed variables, assuming the parameters are known. In this section, we discuss how to learn the parameters.

In Section 10.4.2, we showed that when we have complete data and a factored prior, the posterior over the parameters also factorizes, making computation very simple. Unfortunately this is no longer true if we have hidden variables and/or missing data. The reason is apparent from looking at Figure 11.9. If the z_i were observed, then by d-separation, we see that $\boldsymbol{\theta}_z \perp \boldsymbol{\theta}_x|\mathcal{D}$, and hence the posterior will factorize. But since, in an LVM, the z_i are hidden, the parameters are no longer independent, and the posterior does not factorize, making it much harder to



(a) (b)

Figure 11.10 Left: $N = 200$ data points sampled from a mixture of 2 Gaussians in 1d, with $\pi_k = 0.5$, $\sigma_k = 5$, $\mu_1 = -10$ and $\mu_2 = 10$. Right: Likelihood surface $p(\mathcal{D}|\mu_1, \mu_2)$, with all other parameters set to their true values. We see the two symmetric modes, reflecting the unidentifiability of the parameters. Figure generated by `mixGaussLikSurfaceDemo`.

compute. This also complicates the computation of MAP and ML estimates, as we discus below.

11.3.1 Unidentifiability

The main problem with computing $p(\boldsymbol{\theta}|\mathcal{D})$ for an LVM is that the posterior may have multiple modes. To see why, consider a GMM. If the z_i were all observed, we would have a unimodal posterior for the parameters:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \text{Dir}(\boldsymbol{\pi}|\mathcal{D}) \prod_{k=1}^{K} \text{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k|\mathcal{D}) \tag{11.11}$$

Consequently we can easily find the globally optimal MAP estimate (and hence globally optimal MLE).

But now suppose the z_i's are hidden. In this case, for each of the possible ways of "filling in" the z_i's, we get a different unimodal likelihood. Thus when we marginalize out over the z_i's, we get a multi-modal posterior for $p(\boldsymbol{\theta}|\mathcal{D})$.[1] These modes correspond to different labelings of the clusters. This is illustrated in Figure 11.10(b), where we plot the likelihood function, $p(\mathcal{D}|\mu_1, \mu_2)$, for a 2D GMM with $K = 2$ for the data is shown in Figure 11.10(a). We see two peaks, one corresponding to the case where $\mu_1 = -10$, $\mu_2 = 10$, and the other to the case where $\mu_1 = 10$, $\mu_2 = -10$. We say the parameters are not **identifiable**, since there is not a unique MLE. Therefore there cannot be a unique MAP estimate (assuming the prior does not rule out certain labelings), and hence the posterior must be multimodal. The question of how many modes there

[1] Do not confuse multimodality of the parameter posterior, $p(\boldsymbol{\theta}|\mathcal{D})$, with the multimodality defined by the model, $p(\mathbf{x}|\boldsymbol{\theta})$. In the latter case, if we have K clusters, we would expect to only get K peaks, although it is theoretically possible to get more than K, at least if $D > 1$ (Carreira-Perpinan and Williams 2003).

are in the parameter posterior is hard to answer. There are $K!$ possible labelings, but some of the peaks might get merged. Nevertheless, there can be an exponential number, since finding the optimal MLE for a GMM is NP-hard (Aloise et al. 2009; Drineas et al. 2004).

Unidentifiability can cause a problem for Bayesian inference. For example, suppose we draw some samples from the posterior, $\boldsymbol{\theta}^{(s)} \sim p(\boldsymbol{\theta}|\mathcal{D})$, and then average them, to try to approximate the posterior mean, $\overline{\boldsymbol{\theta}} = \frac{1}{S}\sum_{s=1}^{S}\boldsymbol{\theta}^{(s)}$. (This kind of Monte Carlo approach is explained in more detail in Chapter 24.) If the samples come from different modes, the average will be meaningless. Note, however, that it is reasonable to average the posterior predictive distributions, $p(\mathbf{x}) \approx \frac{1}{S}\sum_{s=1}^{S} p(\mathbf{x}|\boldsymbol{\theta}^{(s)})$, since the likelihood function is invariant to which mode the parameters came from.

A variety of solutions have been proposed to the unidentifiability problem. These solutions depend on the details of the model and the inference algorithm that is used. For example, see (Stephens 2000) for an approach to handling unidentifiability in mixture models using MCMC.

The approach we will adopt in this chapter is much simpler: we just compute a single local mode, i.e., we perform approximate MAP estimation. (We say “approximate” since finding the globally optimal MLE, and hence MAP estimate, is NP-hard, at least for mixture models (Aloise et al. 2009).) This is by far the most common approach, because of its simplicity. It is also a reasonable approximation, at least if the sample size is sufficiently large. To see why, consider Figure 11.9(a). We see that there are N latent variables, each of which gets to “see” one data point each. However, there are only two latent parameters, each of which gets to see N data points. So the posterior uncertainty about the parameters is typically much less than the posterior uncertainty about the latent variables. This justifies the common strategy of computing $p(z_i|\mathbf{x}_i, \hat{\boldsymbol{\theta}})$, but not bothering to compute $p(\boldsymbol{\theta}|\mathcal{D})$. In Section 5.6, we will study hierarchical Bayesian models, which essentially put structure on top of the parameters. In such models, it is important to model $p(\boldsymbol{\theta}|\mathcal{D})$, so that the parameters can send information between themselves. If we used a point estimate, this would not be possible.

11.3.2 Computing a MAP estimate is non-convex

In the previous sections, we have argued, rather heuristically, that the likelihood function has multiple modes, and hence that finding an MAP or ML estimate will be hard. In this section, we show this result by more algebraic means, which sheds some additional insight into the problem. Our presentation is based in part on (Rennie 2004).

Consider the log-likelihood for an LVM:

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_i \log\left[\sum_{\mathbf{z}_i} p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta})\right] \tag{11.12}$$

Unfortunately, this objective is hard to maximize. since we cannot push the log inside the sum. This precludes certain algebraic simplications, but does not prove the problem is hard.

Now suppose the joint probability distribution $p(\mathbf{z}_i, \mathbf{x}_i|\boldsymbol{\theta})$ is in the exponential family, which means it can be written as follows:

$$p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})}\exp[\boldsymbol{\theta}^T\boldsymbol{\phi}(\mathbf{x}, \mathbf{z})] \tag{11.13}$$

where $\boldsymbol{\phi}(\mathbf{x}, \mathbf{z})$ are the sufficient statistics, and $Z(\boldsymbol{\theta})$ is the normalization constant (see Section 9.2 for more details). It can be shown (Exercise 9.2) that the MVN is in the exponential family, as are nearly all of the distributions we have encountered so far, including Dirichlet, multinomial, Gamma, Wishart, etc. (The Student distribution is a notable exception.) Furthermore, mixtures of exponential families are also in the exponential family, providing the mixing indicator variables are observed (Exercise 11.11).

With this assumption, the **complete data log likelihood** can be written as follows:

$$\ell_c(\boldsymbol{\theta}) = \sum_i \log p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) = \boldsymbol{\theta}^T (\sum_i \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z}_i)) - NZ(\boldsymbol{\theta}) \tag{11.14}$$

The first term is clearly linear in $\boldsymbol{\theta}$. One can show that $Z(\boldsymbol{\theta})$ is a convex function (Boyd and Vandenberghe 2004), so the overall objective is concave (due to the minus sign), and hence has a unique maximum.

Now consider what happens when we have missing data. The **observed data log likelihood** is given by

$$\ell(\boldsymbol{\theta}) = \sum_i \log \sum_{\mathbf{z}_i} p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) = \sum_i \log \left[\sum_{\mathbf{z}_i} e^{\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{z}_i, \mathbf{x}_i)} \right] - N \log Z(\boldsymbol{\theta}) \tag{11.15}$$

One can show that the log-sum-exp function is convex (Boyd and Vandenberghe 2004), and we know that $Z(\boldsymbol{\theta})$ is convex. However, the difference of two convex functions is not, in general, convex. So the objective is neither convex nor concave, and has local optima.

The disadvantage of non-convex functions is that it is usually hard to find their global optimum. Most optimization algorithms will only find a local optimum; which one they find depends on where they start. There are some algorithms, such as simulated annealing (Section 24.6.1) or genetic algorithms, that claim to always find the global optimum, but this is only under unrealistic assumptions (e.g., if they are allowed to be cooled "infinitely slowly", or allowed to run "infinitely long"). In practice, we will run a local optimizer, perhaps using **multiple random restarts** to increase out chance of finding a "good" local optimum. Of course, careful initialization can help a lot, too. We give examples of how to do this on a case-by-case basis.

Note that a convex method for fitting mixtures of Gaussians has been proposed. The idea is to assign one cluster per data point, and select from amongst them, using a convex ℓ_1-type penalty, rather than trying to optimize the locations of the cluster centers. See (Lashkari and Golland 2007) for details. This is essentially an unsupervised version of the approach used in sparse kernel logistic regression, which we will discuss in Section 14.3.2. Note, however, that the ℓ_1 penalty, although convex, is not necessarily a good way to promote sparsity, as discussed in Chapter 13. In fact, as we will see in that Chapter, some of the best sparsity-promoting methods use non-convex penalties, and use EM to optimie them! The moral of the story is: do not be afraid of non-convexity.

11.4 The EM algorithm

For many models in machine learning and statistics, computing the ML or MAP parameter estimate is easy provided we observe all the values of all the relevant random variables, i.e., if

Model	Section
Mix. Gaussians	11.4.2
Mix. experts	11.4.3
Factor analysis	12.1.5
Student T	11.4.5
Probit regression	11.4.6
DGM with hidden variables	11.4.4
MVN with missing data	11.6.1
HMMs	17.5.2
Shrinkage estimates of Gaussian means	Exercise 11.13

Table 11.2 Some models discussed in this book for which EM can be easily applied to find the ML/ MAP parameter estimate.

we have complete data. However, if we have missing data and/or latent variables, then computing the ML/MAP estimate becomes hard.

One approach is to use a generic gradient-based optimizer to find a local minimum of the **negative log likelihood** or **NLL**, given by

$$\text{NLL}(\boldsymbol{\theta}) = - \triangleq \frac{1}{N} \log p(\mathcal{D}|\boldsymbol{\theta}) \tag{11.16}$$

However, we often have to enforce constraints, such as the fact that covariance matrices must be positive definite, mixing weights must sum to one, etc., which can be tricky (see Exercise 11.5). In such cases, it is often much simpler (but not always faster) to use an algorithm called **expectation maximization**, or **EM** for short (Dempster et al. 1977; Meng and van Dyk 1997; McLachlan and Krishnan 1997). This is a simple iterative algorithm, often with closed-form updates at each step. Furthermore, the algorithm automatically enforce the required constraints.

EM exploits the fact that if the data were fully observed, then the ML/ MAP estimate would be easy to compute. In particular, EM is an iterative algorithm which alternates between inferring the missing values given the parameters (E step), and then optimizing the parameters given the "filled in" data (M step). We give the details below, followed by several examples. We end with a more theoretical discussion, where we put the algorithm in a larger context. See Table 11.2 for a summary of the applications of EM in this book.

11.4.1 Basic idea

Let $\mathbf{x}_i$ be the visible or observed variables in case i, and let $\mathbf{z}_i$ be the hidden or missing variables. The goal is to maximize the log likelihood of the observed data:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{i=1}^{N} \log \left[\sum_{\mathbf{z}_i} p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) \right] \tag{11.17}$$

Unfortunately this is hard to optimize, since the log cannot be pushed inside the sum.

EM gets around this problem as follows. Define the **complete data log likelihood** to be

$$\ell_c(\boldsymbol{\theta}) \triangleq \sum_{i=1}^{N} \log p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) \tag{11.18}$$

This cannot be computed, since $\mathbf{z}_i$ is unknown. So let us define the **expected complete data log likelihood** as follows:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) = \mathbb{E}\left[\ell_c(\boldsymbol{\theta})\middle|\mathcal{D}, \boldsymbol{\theta}^{t-1}\right] \tag{11.19}$$

where t is the current iteration number. Q is called the **auxiliary function**. The expectation is taken wrt the old parameters, $\boldsymbol{\theta}^{t-1}$, and the observed data $\mathcal{D}$. The goal of the **E step** is to compute $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$, or rather, the terms inside of it which the MLE depends on; these are known as the **expected sufficient statistics** or ESS. In the **M step**, we optimize the Q function wrt $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^t = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) \tag{11.20}$$

To perform MAP estimation, we modify the M step as follows:

$$\boldsymbol{\theta}^t = \underset{\boldsymbol{\theta}}{\text{argmax}}\, Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) + \log p(\boldsymbol{\theta}) \tag{11.21}$$

The E step remains unchanged.

In Section 11.4.7 we show that the EM algorithm monotonically increases the log likelihood of the observed data (plus the log prior, if doing MAP estimation), or it stays the same. So if the objective ever goes down, there must be a bug in our math or our code. (This is a surprisingly useful debugging tool!)

Below we explain how to perform the E and M steps for several simple models, that should make things clearer.

11.4.2 EM for GMMs

In this section, we discuss how to fit a mixture of Gaussians using EM. Fitting other kinds of mixture models requires a straightforward modification — see Exercise 11.3. We assume the number of mixture components, K, is known (see Section 11.5 for discussion of this point).

11.4.2.1 Auxiliary function

The expected complete data log likelihood is given by

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t-1)}) &\triangleq \mathbb{E}\left[\sum_i \log p(\mathbf{x}_i, z_i|\boldsymbol{\theta})\right] &(11.22)\\ &= \sum_i \mathbb{E}\left[\log\left[\prod_{k=1}^K (\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k))^{\mathbb{I}(z_i=k)}\right]\right] &(11.23)\\ &= \sum_i \sum_k \mathbb{E}\left[\mathbb{I}(z_i = k)\right] \log[\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k)] &(11.24)\\ &= \sum_i \sum_k p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}^{t-1}) \log[\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k)] &(11.25)\\ &= \sum_i \sum_k r_{ik} \log \pi_k + \sum_i \sum_k r_{ik} \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) &(11.26) \end{aligned}$$

where $r_{ik} \triangleq p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}^{(t-1)})$ is the **responsibility** that cluster k takes for data point i. This is computed in the E step, described below.

11.4.2.2 E step

The E step has the following simple form, which is the same for any mixture model:

$$r_{ik} = \frac{\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k^{(t-1)})}{\sum_{k'} \pi_{k'} p(\mathbf{x}_i|\boldsymbol{\theta}_{k'}^{(t-1)})} \qquad (11.27)$$

11.4.2.3 M step

In the M step, we optimize Q wrt $\boldsymbol{\pi}$ and the $\boldsymbol{\theta}_k$. For $\boldsymbol{\pi}$, we obviously have

$$\pi_k = \frac{1}{N}\sum_i r_{ik} = \frac{r_k}{N} \qquad (11.28)$$

where $r_k \triangleq \sum_i r_{ik}$ is the weighted number of points assigned to cluster k.

To derive the M step for the $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ terms, we look at the parts of Q that depend on $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. We see that the result is

$$\begin{aligned} \ell(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) &= \sum_k \sum_i r_{ik} \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) &(11.29)\\ &= -\frac{1}{2}\sum_i r_{ik}\left[\log|\boldsymbol{\Sigma}_k| + (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)\right] &(11.30) \end{aligned}$$

This is just a weighted version of the standard problem of computing the MLEs of an MVN (see Section 4.1.3). One can show (Exercise 11.2) that the new parameter estimates are given by

$$\begin{aligned} \boldsymbol{\mu}_k &= \frac{\sum_i r_{ik}\mathbf{x}_i}{r_k} &(11.31)\\ \boldsymbol{\Sigma}_k &= \frac{\sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{r_k} = \frac{\sum_i r_{ik}\mathbf{x}_i\mathbf{x}_i^T}{r_k} - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T &(11.32) \end{aligned}$$

These equations make intuitive sense: the mean of cluster k is just the weighted average of all points assigned to cluster k, and the covariance is proportional to the weighted empirical scatter matrix.

After computing the new estimates, we set $\boldsymbol{\theta}^t = (\pi_k, \mu_k, \boldsymbol{\Sigma}_k)$ for $k = 1 : K$, and go to the next E step.

11.4.2.4 Example

An example of the algorithm in action is shown in Figure 11.11. We start with $\boldsymbol{\mu}_1 = (-1, 1)$, $\boldsymbol{\Sigma}_1 = \mathbf{I}$, $\boldsymbol{\mu}_2 = (1, -1)$, $\boldsymbol{\Sigma}_2 = \mathbf{I}$. We color code points such that blue points come from cluster 1 and red points from cluster 2. More precisely, we set the color to

$$\text{color}(i) = r_{i1}\text{blue} + r_{i2}\text{red} \tag{11.33}$$

so ambiguous points appear purple. After 20 iterations, the algorithm has converged on a good clustering. (The data was standardized, by removing the mean and dividing by the standard deviation, before processing. This often helps convergence.)

11.4.2.5 K-means algorithm

There is a popular variant of the EM algorithm for GMMs known as the **K-means algorithm**, which we now discuss. Consider a GMM in which we make the following assumptions: $\boldsymbol{\Sigma}_k = \sigma^2 \mathbf{I}_D$ is fixed, and $\pi_k = 1/K$ is fixed, so only the cluster centers, $\boldsymbol{\mu}_k \in \mathbb{R}^D$, have to be estimated. Now consider the following delta-function approximation to the posterior computed during the E step:

$$p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}) \approx \mathbb{I}(k = z_i^*) \tag{11.34}$$

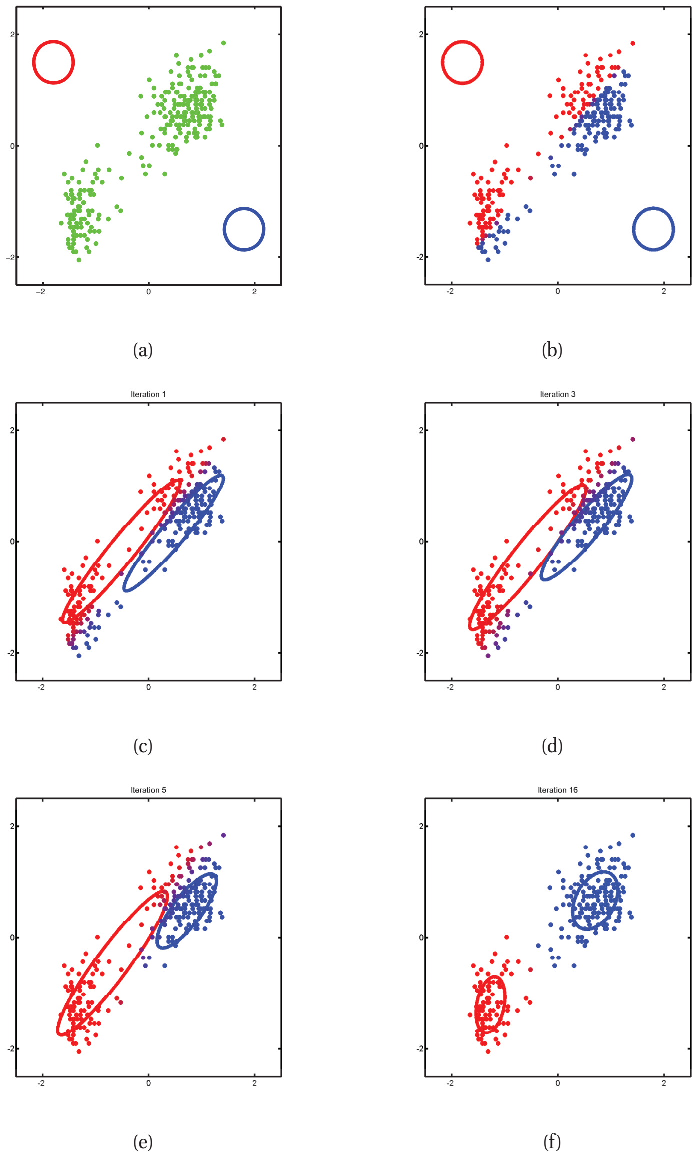
where $z_i* = \text{argmax}_k\, p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta})$. This is sometimes called **hard EM**, since we are making a hard assignment of points to clusters. Since we assumed an equal spherical covariance matrix for each cluster, the most probable cluster for $\mathbf{x}_i$ can be computed by finding the nearest prototype:

$$z_i^* = \arg\min_k ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2 \tag{11.35}$$

Hence in each E step, we must find the Euclidean distance between N data points and K cluster centers, which takes $O(NKD)$ time. However, this can be sped up using various techniques, such as applying the triangle inequality to avoid some redundant computations (Elkan 2003). Given the hard cluster assignments, the M step updates each cluster center by computing the mean of all points assigned to it:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i \tag{11.36}$$

See Algorithm 5 for the pseudo-code.



(a) (b)

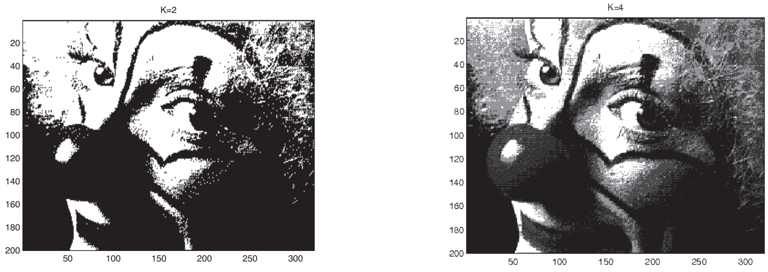
(c) (d)

(e) (f)

Figure 11.11 Illustration of the EM for a GMM applied to the Old Faithful data. (a) Initial (random) values of the parameters. (b) Posterior responsibility of each point computed in the first E step. The degree of redness indicates the degree to which the point belongs to the red cluster, and similarly for blue; this purple points have a roughly uniform posterior over clusters. (c) We show the updated parameters after the first M step. (d) After 3 iterations. (e) After 5 iterations. (f) After 16 iterations. Based on (Bishop 2006a) Figure 9.8. Figure generated by `mixGaussDemoFaithful`.

Algorithm 11.1: K-means algorithm

```
initialize $\mathbf{m}_k$;
repeat
    Assign each data point to its closest cluster center: $z_i = \arg\min_k ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$;
    Update each cluster center by computing the mean of all points assigned to it:
    $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i$;
until converged;
```



(a) (b)

Figure 11.12 An image compressed using vector quantization with a codebook of size K. (a) $K = 2$. (b) $K = 4$. Figure generated by `vqDemo`.

11.4.2.6 Vector quantization

Since K-means is not a proper EM algorithm, it is not maximizing likelihood. Instead, it can be interpreted as a greedy algorithm for approximately minimizing a loss function related to data compression, as we now explain.

Suppose we want to perform lossy compression of some real-valued vectors, $\mathbf{x}_i \in \mathbb{R}^D$. A very simple approach to this is to use **vector quantization** or **VQ**. The basic idea is to replace each real-valued vector $\mathbf{x}_i \in \mathbb{R}^D$ with a discrete symbol $z_i \in \{1, \ldots, K\}$, which is an index into a **codebook** of K prototypes, $\boldsymbol{\mu}_k \in \mathbb{R}^D$. Each data vector is encoded by using the index of the most similar prototype, where similarity is measured in terms of Euclidean distance:

$$\text{encode}(\mathbf{x}_i) = \arg\min_k ||\mathbf{x}_i - \boldsymbol{\mu}_k||^2 \tag{11.37}$$

We can define a cost function that measures the quality of a codebook by computing the **reconstruction error** or **distortion** it induces:

$$J(\boldsymbol{\mu}, \mathbf{z}|K, \mathbf{X}) \triangleq \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{x}_i - \text{decode}(\text{encode}(\mathbf{x}_i))||^2 = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{x}_i - \boldsymbol{\mu}_{z_i}||^2 \tag{11.38}$$

where $\text{decode}(k) = \boldsymbol{\mu}_k$. The K-means algorithm can be thought of as a simple iterative scheme for minimizing this objective.

Of course, we can achieve zero distortion if we assign one prototype to every data vector, but that takes $O(NDC)$ space, where N is the number of real-valued data vectors, each of length D, and C is the number of bits needed to represent a real-valued scalar (the quantization accuracy). However, in many data sets, we see similar vectors repeatedly, so rather than storing them many times, we can store them once and then create pointers to them. Hence we can reduce the space requirement to $O(N \log_2 K + KDC)$: the $O(N \log_2 K)$ term arises because each of the N data vectors needs to specify which of the K codewords it is using (the pointers); and the $O(KDC)$ term arises because we have to store each codebook entry, each of which is a D-dimensional vector. Typically the first term dominates the second, so we can approximate the **rate** of the encoding scheme (number of bits needed per object) as $O(\log_2 K)$, which is typically much less than $O(DC)$.

One application of VQ is to image compression. Consider the $N = 200 \times 320 = 64,000$ pixel image in Figure 11.12; this is gray-scale, so $D = 1$. If we use one byte to represent each pixel (a gray-scale intensity of 0 to 255), then $C = 8$, so we need $NC = 512,000$ bits to represent the image. For the compressed image, we need $N \log_2 K + KC$ bits. For $K = 4$, this is about 128kb, a factor of 4 compression. For $K = 8$, this is about 192kb, a factor of 2.6 compression, at negligible perceptual loss (see Figure 11.12(b)). Greater compression could be achieved if we modelled spatial correlation between the pixels, e.g., if we encoded 5x5 blocks (as used by JPEG). This is because the residual errors (differences from the model's predictions) would be smaller, and would take fewer bits to encode.

11.4.2.7 Initialization and avoiding local minima

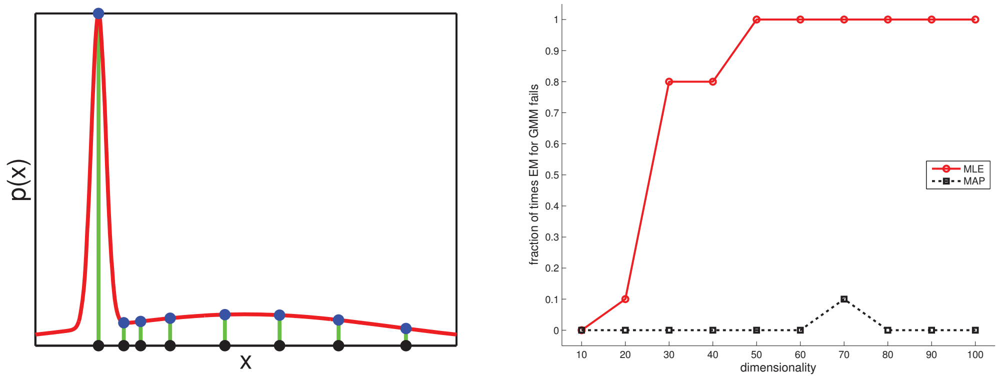
Both K-means and EM need to be initialized. It is common to pick K data points at random, and to make these be the initial cluster centers. Or we can pick the centers sequentially so as to try to "cover" the data. That is, we pick the initial point uniformly at random. Then each subsequent point is picked from the remaining points with probability proportional to its squared distance to the points's closest cluster center. This is known as **farthest point clustering** (Gonzales 1985), or **k-means++** (Arthur and Vassilvitskii 2007; Bahmani et al. 2012). Surprisingly, this simple trick can be shown to guarantee that the distortion is never more than $O(\log K)$ worse than optimal (Arthur and Vassilvitskii 2007).

An heuristic that is commonly used in the speech recognition community is to incrementally "grow" GMMs: we initially give each cluster a score based on its mixture weight; after each round of training, we consider splitting the cluster with the highest score into two, with the new centroids being random perturbations of the original centroid, and the new scores being half of the old scores. If a new cluster has too small a score, or too narrow a variance, it is removed. We continue in this way until the desired number of clusters is reached. See (Figueiredo and Jain 2002) for a similar incremental approach.

11.4.2.8 MAP estimation

As usual, the MLE may overfit. The overfitting problem is particularly severe in the case of GMMs. To understand the problem, suppose for simplicity that $\mathbf{\Sigma}_k = \sigma_k^2 I$, and that $K = 2$. It is possible to get an infinite likelihood by assigning one of the centers, say $\boldsymbol{\mu}_2$, to a single data point, say $\mathbf{x}_1$, since then the 1st term makes the following contribution to the likelihood:

$$\mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}_2, \sigma_2^2 I) = \frac{1}{\sqrt{2\pi\sigma_2^2}} e^0 \tag{11.39}$$



(a)

(b)

Figure 11.13 (a) Illustration of how singularities can arise in the likelihood function of GMMs. Based on (Bishop 2006a) Figure 9.7. Figure generated by `mixGaussSingularity`. (b) Illustration of the benefit of MAP estimation vs ML estimation when fitting a Gaussian mixture model. We plot the fraction of times (out of 5 random trials) each method encounters numerical problems vs the dimensionality of the problem, for $N = 100$ samples. Solid red (upper curve): MLE. Dotted black (lower curve): MAP. Figure generated by `mixGaussMLvsMAP`.

Hence we can drive this term to infinity by letting $\sigma_2 \rightarrow 0$, as shown in Figure 11.13(a). We will call this the "collapsing variance problem".

An easy solution to this is to perform MAP estimation. The new auxiliary function is the expected complete data log-likelihood plus the log prior:

$$Q'(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) = \left[\sum_i \sum_k r_{ik} \log \pi_{ik} + \sum_i \sum_k r_{ik} \log p(\mathbf{x}_i | \boldsymbol{\theta}_k) \right] + \log p(\boldsymbol{\pi}) + \sum_k \log p(\boldsymbol{\theta}_k) \tag{11.40}$$

Note that the E step remains unchanged, but the M step needs to be modified, as we now explain.

For the prior on the mixture weights, it is natural to use a Dirichlet prior, $\boldsymbol{\pi} \sim \text{Dir}(\boldsymbol{\alpha})$, since this is conjugate to the categorical distribution. The MAP estimate is given by

$$\pi_k = \frac{r_k + \alpha_k - 1}{N + \sum_k \alpha_k - K} \tag{11.41}$$

If we use a uniform prior, $\alpha_k = 1$, this reduces to Equation 11.28.

The prior on the parameters of the class conditional densities, $p(\boldsymbol{\theta}_k)$, depends on the form of the class conditional densities. We discuss the case of GMMs below, and leave MAP estimation for mixtures of Bernoullis to Exercise 11.3.

For simplicity, let us consider a conjugate prior of the form

$$p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \text{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k | \mathbf{m}_0, \kappa_0, \nu_0, \mathbf{S}_0) \tag{11.42}$$

From Section 4.6.3, the MAP estimate is given by

$$\hat{\boldsymbol{\mu}}_k = \frac{r_k \overline{\mathbf{x}}_k + \kappa_0 \mathbf{m}_0}{r_k + \kappa_0} \tag{11.43}$$

$$\tag{11.44}$$

$$\overline{\mathbf{x}}_k \triangleq \frac{\sum_i r_{ik} \mathbf{x}_i}{r_k} \tag{11.45}$$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{\mathbf{S}_0 + \mathbf{S}_k + \frac{\kappa_0 r_k}{\kappa_0 + r_k}(\overline{\mathbf{x}}_k - \mathbf{m}_0)(\overline{\mathbf{x}}_k - \mathbf{m}_0)^T}{\nu_0 + r_k + D + 2} \tag{11.46}$$

$$\mathbf{S}_k \triangleq \sum_i r_{ik}(\mathbf{x}_i - \overline{\mathbf{x}}_k)(\mathbf{x}_i - \overline{\mathbf{x}}_k)^T \tag{11.47}$$

We now illustrate the benefits of using MAP estimation instead of ML estimation in the context of GMMs. We apply EM to some synthetic data in D dimensions, using either ML or MAP estimation. We count the trial as a "failure" if there are numerical issues involving singular matrices. For each dimensionality, we conduct 5 random trials. The results are illustrated in Figure 11.13(b) using $N = 100$. We see that as soon as D becomes even moderately large, ML estimation crashes and burns, whereas MAP estimation never encounters numerical problems.

When using MAP estimation, we need to specify the hyper-parameters. Here we mention some simple heuristics for setting them (Fraley and Raftery 2007, p163). We can set $\kappa_0 = 0$, so that the $\boldsymbol{\mu}_k$ are unregularized, since the numerical problems only arise from $\boldsymbol{\Sigma}_k$. In this case, the MAP estimates simplify to $\hat{\boldsymbol{\mu}}_k = \overline{\mathbf{x}}_k$ and $\hat{\boldsymbol{\Sigma}}_k = \frac{\mathbf{S}_0 + \mathbf{S}_k}{\nu_0 + r_k + D + 2}$, which is not quite so scary-looking.

Now we discuss how to set $\mathbf{S}_0$. One possibility is to use

$$\mathbf{S}_0 = \frac{1}{K^{1/D}} \text{diag}(s_1^2, \ldots, s_D^2) \tag{11.48}$$

where $s_j = (1/N) \sum_{i=1}^N (x_{ij} - \overline{x}_j)^2$ is the pooled variance for dimension j. (The reason for the $\frac{1}{K^{1/D}}$ term is that the resulting volume of each ellipsoid is then given by $|\mathbf{S}_0| = \frac{1}{K} |\text{diag}(s_1^2, \ldots, s_D^2)|$.) The parameter ν_0 controls how strongly we believe this prior. The weakest prior we can use, while still being proper, is to set $\nu_0 = D + 2$, so this is a common choice.

11.4.3 EM for mixture of experts

We can fit a mixture of experts model using EM in a straightforward manner. The expected complete data log likelihood is given by

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) = \sum_{i=1}^N \sum_{k=1}^K r_{ik} \log[\pi_{ik} \mathcal{N}(y_i | \mathbf{w}_k^T \mathbf{x}_i, \sigma_k^2)] \tag{11.49}$$

$$\pi_{i,k} \triangleq \mathcal{S}(\mathbf{V}^T \mathbf{x}_i)_k \tag{11.50}$$

$$r_{ik} \propto \pi_{ik}^{old} \mathcal{N}(y_i | \mathbf{x}_i^T \mathbf{w}_k^{old}, (\sigma_k^{old})^2) \tag{11.51}$$

So the E step is the same as in a standard mixture model, except we have to replace π_k with $\pi_{i,k}$ when computing r_{ik}.

In the M step, we need to maximize $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old})$ wrt $\mathbf{w}_k$, σ_k^2 and $\mathbf{V}$. For the regression parameters for model k, the objective has the form

$$Q(\boldsymbol{\theta}_k, \boldsymbol{\theta}^{old}) = \sum_{i=1}^{N} r_{ik} \left\{ -\frac{1}{\sigma_k^2}(y_i - \mathbf{w}_k^T \mathbf{x}_i) \right\} \tag{11.52}$$

We recognize this as a weighted least squares problem, which makes intuitive sense: if r_{ik} is small, then data point i will be downweighted when estimating model k's parameters. From Section 8.3.4 we can immediately write down the MLE as

$$\mathbf{w}_k = (\mathbf{X}^T \mathbf{R}_k \mathbf{X})^{-1} \mathbf{X}^T \mathbf{R}_k \mathbf{y} \tag{11.53}$$

where $\mathbf{R}_k = \text{diag}(r_{:,k})$. The MLE for the variance is given by

$$\sigma_k^2 = \frac{\sum_{i=1}^N r_{ik}(y_i - \mathbf{w}_k^T \mathbf{x}_i)^2}{\sum_{i=1}^N r_{ik}} \tag{11.54}$$

We replace the estimate of the unconditional mixing weights $\boldsymbol{\pi}$ with the estimate of the gating parameters, $\mathbf{V}$. The objective has the form

$$\ell(\mathbf{V}) \quad = \quad \sum_i \sum_k r_{ik} \log \pi_{i,k} \tag{11.55}$$

We recognize this as equivalent to the log-likelihood for multinomial logistic regression in Equation 8.34, except we replace the "hard" 1-of-C encoding $\mathbf{y}_i$ with the "soft" 1-of-K encoding $\mathbf{r}_i$. Thus we can estimate $\mathbf{V}$ by fitting a logistic regression model to soft target labels.

11.4.4 EM for DGMs with hidden variables

We can generalize the ideas behind EM for mixtures of experts to compute the MLE or MAP estimate for an arbitrary DGM. We could use gradient-based methods (Binder et al. 1997), but it is much simpler to use EM (Lauritzen 1995): in the E step, we just estimate the hidden variables, and in the M step, we will compute the MLE using these filled-in values. We give the details below.

For simplicity of presentation, we will assume all CPDs are tabular. Based on Section 10.4.2, let us write each CPT as follows:

$$p(x_{it}|\mathbf{x}_{i,\text{pa}(t)}, \boldsymbol{\theta}_t) = \prod_{c=1}^{K_{\text{pa}(t)}} \prod_{k=1}^{K_t} \theta_{tck}^{\mathbb{I}(x_{it}=i, \mathbf{x}_{i,\text{pa}(t)}=c)} \tag{11.56}$$

The log-likelihood of the complete data is given by

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{t=1}^{V} \sum_{c=1}^{K_{\text{pa}(t)}} \sum_{k=1}^{K_t} N_{tck} \log \theta_{tck} \tag{11.57}$$

where $N_{tck} = \sum_{i=1}^N \mathbb{I}(x_{it} = i, \mathbf{x}_{i,\text{pa}(t)} = c)$ are the empirical counts. Hence the expected complete data log-likelihood has the form

$$\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] \quad = \quad \sum_t \sum_c \sum_k \overline{N}_{tck} \log \theta_{tck} \tag{11.58}$$

where

$$\overline{N}_{tck} = \sum_{i=1}^{N} \mathbb{E}\left[\mathbb{I}(x_{it} = i, \mathbf{x}_{i,\text{pa}(t)} = c)\right] = \sum_{i} p(x_{it} = k, \mathbf{x}_{i,\text{pa}(t)} = c|\mathcal{D}_i) \tag{11.59}$$

where $\mathcal{D}_i$ are all the visible variables in case i.

The quantity $p(x_{it}, \mathbf{x}_{i,\text{pa}(t)}|\mathcal{D}_i, \boldsymbol{\theta})$ is known as a **family marginal**, and can be computed using any GM inference algorithm. The $\overline{N}_{tjk}$ are the expected sufficient statistics, and constitute the output of the E step.

Given these ESS, the M step has the simple form

$$\hat{\theta}_{tck} = \frac{\overline{N}_{tck}}{\sum_{k'} \overline{N}_{tjk'}} \tag{11.60}$$

This can be proved by adding Lagrange multipliers (to enforce the constraint $\sum_k \theta_{tjk} = 1$) to the expected complete data log likelihood, and then optimizing each parameter vector $\boldsymbol{\theta}_{tc}$ separately. We can modify this to perform MAP estimation with a Dirichlet prior by simply adding pseudo counts to the expected counts.

11.4.5 EM for the Student distribution *

One problem with the Gaussian distribution is that it is sensitive to outliers, since the log-probability only decays quadratically with distance from the center. A more robust alternative is the Student t distribution, as discussed in Section **??**.

Unlike the case of a Gaussian, there is no closed form formula for the MLE of a Student, even if we have no missing data, so we must resort to iterative optimization methods. The easiest one to use is EM, since it automatically enforces the constraints that ν is positive and that $\boldsymbol{\Sigma}$ is symmetric positive definite. In addition, the resulting algorithm turns out to have a simple intuitive form, as we see below.

At first blush, it might not be apparent why EM can be used, since there is no missing data. The key idea is to introduce an "artificial" hidden or auxiliary variable in order to simplify the algorithm. In particular, we will exploit the fact that a Student distribution can be written as a **Gaussian scale mixture**:

$$\mathcal{T}(\mathbf{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu) = \int \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma}/z_i)\text{Ga}(z_i|\frac{\nu}{2}, \frac{\nu}{2})dz_i \tag{11.61}$$

(See Exercise 11.1 for a proof of this in the 1d case.) This can be thought of as an "infinite" mixture of Gaussians, each one with a slightly different covariance matrix.

Treating the z_i as missing data, we can write the complete data log likelihood as

$$\ell_c(\boldsymbol{\theta}) = \sum_{i=1}^{N} [\log \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma}/z_i) + \log \text{Ga}(z_i|\nu/2, \nu/2)] \tag{11.62}$$

$$= \sum_{i=1}^{N} \left[-\frac{D}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{z_i}{2}\delta_i + \frac{\nu}{2}\log\frac{\nu}{2} - \log\Gamma(\frac{\nu}{2}) \right. \tag{11.63}$$

$$\left. + \frac{\nu}{2}(\log z_i - z_i) + (\frac{D}{2} - 1)\log z_i \right] \tag{11.64}$$

where we have defined the Mahalanobis distance to be

$$\delta_i = (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) \tag{11.65}$$

We can partition this into two terms, one involving $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, and the other involving ν. We have, dropping irrelevant constants,

$$\ell_c(\boldsymbol{\theta}) = L_N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) + L_G(\nu) \tag{11.66}$$

$$L_N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq -\frac{1}{2} N \log |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{i=1}^N z_i \delta_i \tag{11.67}$$

$$L_G(\nu) \triangleq -N \log \Gamma(\nu/2) + \frac{1}{2} N\nu \log(\nu/2) + \frac{1}{2} \nu \sum_{i=1}^N (\log z_i - z_i) \tag{11.68}$$

11.4.5.1 EM with ν known

Let us first derive the algorithm with ν assumed known, for simplicity. In this case, we can ignore the L_G term, so we only need to figure out how to compute $\mathbb{E}[z_i]$ wrt the old parameters.

From Section 4.6.2.2 we have

$$p(z_i | \mathbf{x}_i, \boldsymbol{\theta}) = \text{Ga}(z_i | \frac{\nu + D}{2}, \frac{\nu + \delta_i}{2}) \tag{11.69}$$

Now if $z_i \sim \text{Ga}(a, b)$, then $\mathbb{E}[z_i] = a/b$. Hence the E step at iteration t is

$$\overline{z}_i^{(t)} \triangleq \mathbb{E}\left[z_i | \mathbf{x}_i, \boldsymbol{\theta}^{(t)}\right] = \frac{\nu^{(t)} + D}{\nu^{(t)} + \delta_i^{(t)}} \tag{11.70}$$

The M step is obtained by maximizing $\mathbb{E}[L_N(\boldsymbol{\mu}, \boldsymbol{\Sigma})]$ to yield

$$\hat{\boldsymbol{\mu}}^{(t+1)} = \frac{\sum_i \overline{z}_i^{(t)} \mathbf{x}_i}{\sum_i \overline{z}_i^{(t)}} \tag{11.71}$$

$$\hat{\boldsymbol{\Sigma}}^{(t+1)} = \frac{1}{N} \sum_i \overline{z}_i^{(t)} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}^{(t+1)})(\mathbf{x}_i - \hat{\boldsymbol{\mu}}^{(t+1)})^T \tag{11.72}$$

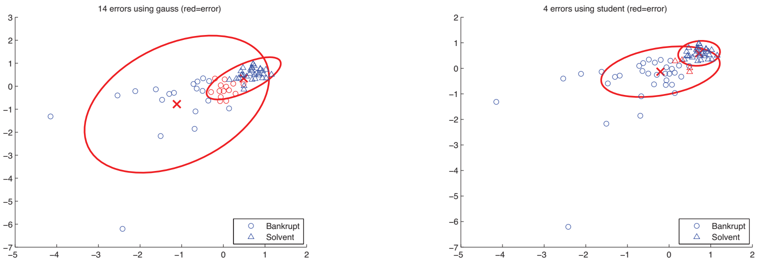
$$= \frac{1}{N} \left[\sum_i \overline{z}_i^{(t)} \mathbf{x}_i \mathbf{x}_i^T - \left(\sum_{i=1}^N \overline{z}_i^{(t)} \right) \hat{\boldsymbol{\mu}}^{(t+1)} (\hat{\boldsymbol{\mu}}^{(t+1)})^T \right] \tag{11.73}$$

These results are quite intuitive: the quantity $\overline{z}_i$ is the precision of measurement i, so if it is small, the corresponding data point is down-weighted when estimating the mean and covariance. This is how the Student achieves robustness to outliers.

11.4.5.2 EM with ν unknown

To compute the MLE for the degrees of freedom, we first need to compute the expectation of $L_G(\nu)$, which involves z_i and $\log z_i$. Now if $z_i \sim \text{Ga}(a, b)$, then one can show that

$$\overline{\ell}_i^{(t)} \triangleq \mathbb{E}\left[\log z_i | \boldsymbol{\theta}^{(t)}\right] = \Psi(a) - \log b \tag{11.74}$$



(a)

(b)

Figure 11.14 Mixture modeling on the bankruptcy data set. Left: Gaussian class conditional densities. Right: Student class conditional densities. Points that belong to class 1 are shown as triangles, points that belong to class 2 are shown as circles The estimated labels, based on the posterior probability of belonging to each mixture component, are computed. If these are incorrect, the point is colored red, otherwise it is colored blue. (Training data is in black.) Figure generated by `mixStudentBankruptcyDemo`.

where $\Psi(x) = \frac{d}{dx}\log\Gamma(x)$ is the digamma function. Hence, from Equation 11.69, we have

$$\overline{\ell}_i^{(t)} = \Psi(\frac{\nu^{(t)} + D}{2}) - \log(\frac{\nu^{(t)} + \delta_i^{(t)}}{2}) \tag{11.75}$$

$$= \log(\overline{z}_i^{(t)}) + \Psi(\frac{\nu^{(t)} + D}{2}) - \log(\frac{\nu^{(t)} + D}{2}) \tag{11.76}$$

Substituting into Equation 11.68, we have

$$\mathbb{E}\left[L_G(\nu)\right] = -N\log\Gamma(\nu/2) + \frac{N\nu}{2}\log(\nu/2) + \frac{\nu}{2}\sum_i(\overline{\ell}_i^{(t)} - \overline{z}_i^{(t)}) \tag{11.77}$$

The gradient of this expression is equal to

$$\frac{d}{d\nu}\mathbb{E}\left[L_G(\nu)\right] = -\frac{N}{2}\Psi(\nu/2) + \frac{N}{2}\log(\nu/2) + \frac{N}{2} + \frac{1}{2}\sum_i(\overline{\ell}_i^{(t)} - \overline{z}_i^{(t)}) \tag{11.78}$$

This has a unique solution in the interval $(0, +\infty]$ which can be found using a 1d constrained optimizer.

Performing a gradient-based optimization in the M step, rather than a closed-form update, is an example of what is known as the **generalized EM** algorithm. One can show that EM will still converge to a local optimum even if we only perform a "partial" improvement to the parameters in the M step.

11.4.5.3 Mixtures of Student distributions

It is easy to extend the above methods to fit a mixture of Student distributions. See Exercise 11.4 for the details.

Let us consider a small example from (Lo 2009, ch3). We have a $N = 66$, $D = 2$ data set regarding the bankrupty patterns of certain companies. The first feature specifies the ratio of retained earnings (RE) to total assets, and the second feature specifies the ratio of earnings before interests and taxes (EBIT) to total assets. We fit two models to this data, ignoring the class labels: a mixture of 2 Gaussians, and a mixture of 2 Students. We then use each fitted model to classify the data. We compute the most probable cluster membership and treat this as $\hat{y}_i$. We then compare $\hat{y}_i$ to the true labels y_i and compute an error rate. If this is more than 50%, we permute the latent labels (i.e., we consider cluster 1 to represent class 2 and vice versa), and then recompute the error rate. Points which are misclassified are then shown in red. The result is shown in Figure 11.14. We see that the Student model made 4 errors, the Gaussian model made 21. This is because the class-conditional densities contain some extreme values, causing the Gaussian to be a poor choice.

11.4.6 EM for probit regression *

In Section 9.4.2, we described the latent variable interpretation of probit regression. Recall that this has the form $p(y_i = 1|z_i) = \mathbb{I}(z_i > 0)$, where $z_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}_i, 1)$ is latent. We now show how to fit this model using EM. (Although it is possible to fit probit regression models using gradient based methods, as shown in Section 9.4.1, this EM-based approach has the advantage that it generalized to many other kinds of models, as we will see later on.)

The complete data log likelihood has the following form, assuming a $\mathcal{N}(\mathbf{0}, \mathbf{V}_0)$ prior on $\mathbf{w}$:

$$\begin{aligned} \ell(\mathbf{z}, \mathbf{w}|\mathbf{V}_0) &= \log p(\mathbf{y}|\mathbf{z}) + \log \mathcal{N}(\mathbf{z}|\mathbf{X}\mathbf{w}, \mathbf{I}) + \log \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{V}_0) && (11.79) \\ &= \sum_i \log p(y_i|z_i) - \frac{1}{2}(\mathbf{z} - \mathbf{X}\mathbf{w})^T(\mathbf{z} - \mathbf{X}\mathbf{w}) - \frac{1}{2}\mathbf{w}^T\mathbf{V}_0^{-1}\mathbf{w} + \text{const} && (11.80) \end{aligned}$$

The posterior in the E step is a **truncated Gaussian**:

$$p(z_i|y_i, \mathbf{x}_i, \mathbf{w}) = \begin{cases} \mathcal{N}(z_i|\mathbf{w}^T\mathbf{x}_i, 1)\mathbb{I}(z_i > 0) & \text{if } y_i = 1 \\ \mathcal{N}(z_i|\mathbf{w}^T\mathbf{x}_i, 1)\mathbb{I}(z_i < 0) & \text{if } y_i = 0 \end{cases} \tag{11.81}$$

In Equation 11.80, we see that $\mathbf{w}$ only depends linearly on $\mathbf{z}$, so we just need to compute $\mathbb{E}[z_i|y_i, \mathbf{x}_i, \mathbf{w}]$. Exercise 11.15 asks you to show that the posterior mean is given by

$$\mathbb{E}[z_i|\mathbf{w}, \mathbf{x}_i] = \begin{cases} \mu_i + \frac{\phi(\mu_i)}{1-\Phi(-\mu_i)} = \mu_i + \frac{\phi(\mu_i)}{\Phi(\mu_i)} & \text{if } y_i = 1 \\ \mu_i - \frac{\phi(\mu_i)}{\Phi(-\mu_i)} = \mu_i - \frac{\phi(\mu_i)}{1-\Phi(\mu_i)} & \text{if } y_i = 0 \end{cases} \tag{11.82}$$

where $\mu_i = \mathbf{w}^T\mathbf{x}_i$.

In the M step, we estimate $\mathbf{w}$ using ridge regression, where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{z}]$ is the output we are trying to predict. Specifically, we have

$$\hat{\mathbf{w}} = (\mathbf{V}_0^{-1} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{\mu} \tag{11.83}$$

The EM algorithm is simple, but can be much slower than direct gradient methods, as illustrated in Figure 11.15. This is because the posterior entropy in the E step is quite high, since we only observe that z is positive or negative, but are given no information from the likelihood about its magnitude. Using a stronger regularizer can help speed convergence, because it constrains the range of plausible z values. In addition, one can use various speedup tricks, such as data augmentation (van Dyk and Meng 2001), but we do not discuss that here.

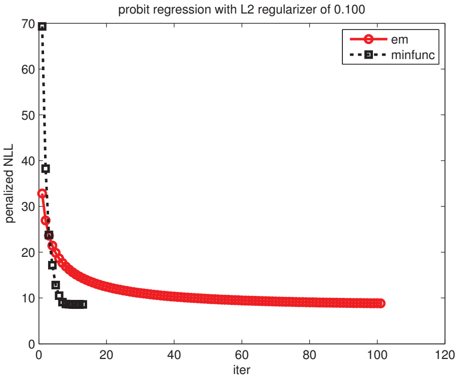


Figure 11.15 Fitting a probit regression model in 2d using a quasi-Newton method or EM. Figure generated by `probitRegDemo`.

11.4.7 Theoretical basis for EM *

In this section, we show that EM monotonically increases the observed data log likelihood until it reaches a local maximum (or saddle point, although such points are usually unstable). Our derivation will also serve as the basis for various generalizations of EM that we will discuss later.

11.4.7.1 Expected complete data log likelihood is a lower bound

Consider an arbitrary distribution $q(\mathbf{z}_i)$ over the hidden variables. The observed data log likelihood can be written as follows:

$$\ell(\boldsymbol{\theta}) \triangleq \sum_{i=1}^{N} \log \left[\sum_{\mathbf{z}_i} p(\mathbf{x}_i, \mathbf{z}_i | \boldsymbol{\theta}) \right] = \sum_{i=1}^{N} \log \left[\sum_{\mathbf{z}_i} q(\mathbf{z}_i) \frac{p(\mathbf{x}_i, \mathbf{z}_i | \boldsymbol{\theta})}{q(\mathbf{z}_i)} \right] \tag{11.84}$$

Now $\log(u)$ is a *concave* function, so from Jensen's inequality (Equation 2.113) we have the following *lower bound*:

$$\ell(\boldsymbol{\theta}) \geq \sum_{i} \sum_{\mathbf{z}_i} q_i(\mathbf{z}_i) \log \frac{p(\mathbf{x}_i, \mathbf{z}_i | \boldsymbol{\theta})}{q_i(\mathbf{z}_i)} \tag{11.85}$$

Let us denote this lower bound as follows:

$$Q(\boldsymbol{\theta}, q) \triangleq \sum_{i} \mathbb{E}_{q_i} \left[\log p(\mathbf{x}_i, \mathbf{z}_i | \boldsymbol{\theta}) \right] + \mathbb{H}\left(q_i\right) \tag{11.86}$$

where $\mathbb{H}(q_i)$ is the entropy of q_i.

The above argument holds for any positive distribution q. Which one should we choose? Intuitively we should pick the q that yields the tightest lower bound. The lower bound is a sum over i of terms of the following form:

$$\begin{aligned} L(\boldsymbol{\theta}, q_i) &= \sum_{\mathbf{z}_i} q_i(\mathbf{z}_i) \log \frac{p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta})}{q_i(\mathbf{z}_i)} && (11.87) \\ &= \sum_{\mathbf{z}_i} q_i(\mathbf{z}_i) \log \frac{p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta}) p(\mathbf{x}_i|\boldsymbol{\theta})}{q_i(\mathbf{z}_i)} && (11.88) \\ &= \sum_{\mathbf{z}_i} q_i(\mathbf{z}_i) \log \frac{p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta})}{q_i(\mathbf{z}_i)} + \sum_{\mathbf{z}_i} q_i(\mathbf{z}_i) \log p(\mathbf{x}_i|\boldsymbol{\theta}) && (11.89) \\ &= -\mathbb{KL}\left(q_i(\mathbf{z}_i)||p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta})\right) + \log p(\mathbf{x}_i|\boldsymbol{\theta}) && (11.90) \end{aligned}$$

The $p(\mathbf{x}_i|\boldsymbol{\theta})$ term is independent of q_i, so we can maximize the lower bound by setting $q_i(\mathbf{z}_i) = p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta})$. Of course, $\boldsymbol{\theta}$ is unknown, so instead we use $q_i^t(\mathbf{z}_i) = p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta}^t)$, where $\boldsymbol{\theta}^t$ is our estimate of the parameters at iteration t. This is the output of the E step.

Plugging this in to the lower bound we get

$$Q(\boldsymbol{\theta}, q^t) = \sum_i \mathbb{E}_{q_i^t}\left[\log p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta})\right] + \mathbb{H}\left(q_i^t\right) \qquad (11.91)$$

We recognize the first term as the expected complete data log likelihood. The second term is a constant wrt $\boldsymbol{\theta}$. So the M step becomes

$$\boldsymbol{\theta}^{t+1} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) = \arg\max_{\boldsymbol{\theta}} \sum_i \mathbb{E}_{q_i^t}\left[\log p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta})\right] \qquad (11.92)$$

as usual.

Now comes the punchline. Since we used $q_i^t(\mathbf{z}_i) = p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta}^t)$, the KL divergence becomes zero, so $L(\boldsymbol{\theta}^t, q_i) = \log p(\mathbf{x}_i|\boldsymbol{\theta}^t)$, and hence

$$Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = \sum_i \log p(\mathbf{x}_i|\boldsymbol{\theta}^t) = \ell(\boldsymbol{\theta}^t) \qquad (11.93)$$

We see that the lower bound is tight after the E step. Since the lower bound "touches" the function, maximizing the lower bound will also "push up" on the function itself. That is, the M step is guaranteed to modify the parameters so as to increase the likelihood of the observed data (unless it is already at a local maximum).

This process is sketched in Figure 11.16. The dashed red curve is the original function (the observed data log-likelihood). The solid blue curve is the lower bound, evaluated at $\boldsymbol{\theta}^t$; this touches the objective function at $\boldsymbol{\theta}^t$. We then set $\boldsymbol{\theta}^{t+1}$ to the maximum of the lower bound (blue curve), and fit a new bound at that point (dotted green curve). The maximum of this new bound becomes $\boldsymbol{\theta}^{t+2}$, etc. (Compare this to Newton's method in Figure 8.4(a), which repeatedly fits and then optimizes a quadratic approximation.)

11.4.7.2 EM monotonically increases the observed data log likelihood

We now prove that EM monotonically increases the observed data log likelihood until it reaches a local optimum. We have

$$\ell(\boldsymbol{\theta}^{t+1}) \geq Q(\boldsymbol{\theta}^{t+1}, \boldsymbol{\theta}^t) \geq Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = \ell(\boldsymbol{\theta}^t) \qquad (11.94)$$

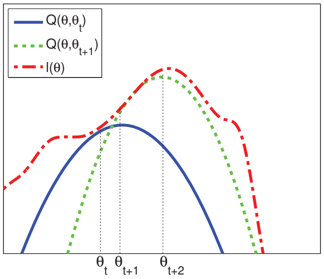


Figure 11.16 Illustration of EM as a bound optimization algorithm. Based on Figure 9.14 of (Bishop 2006a). Figure generated by `emLogLikelihoodMax`.

where the first inequality follows since $Q(\boldsymbol{\theta}, \cdot)$ is a lower bound on $\ell(\boldsymbol{\theta})$; the second inequality follows since, by definition, $Q(\boldsymbol{\theta}^{t+1}, \boldsymbol{\theta}^t) = \max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \geq Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t)$; and the final equality follows Equation 11.93.

As a consequence of this result, if you do not observe monotonic increase of the observed data log likelihood, you must have an error in your math and/or code. (If you are performing MAP estimation, you must add on the log prior term to the objective.) This is a surprisingly powerful debugging tool.

11.4.8 Online EM

When dealing with large or streaming datasets, it is important to be able to learn online, as we discussed in Section 8.5. There are two main approaches to **online EM** in the literature. The first approach, known as **incremental EM** (Neal and Hinton 1998), optimizes the lower bound $Q(\boldsymbol{\theta}, q_1, \ldots, q_N)$ one q_i at a time; however, this requires storing the expected sufficient statistics for each data case. The second approach, known as **stepwise EM** (Sato and Ishii 2000; Cappe and Mouline 2009; Cappe 2010), is based on stochastic approximation theory, and only requires constant memory use. We explain both approaches in more detail below, following the presentation of (Liang and Klein Liang and Klein).

11.4.8.1 Batch EM review

Before explaining online EM, we review batch EM in a more abstract setting. Let $\boldsymbol{\phi}(\mathbf{x}, \mathbf{z})$ be a vector of sufficient statistics for a single data case. (For example, for a mixture of multinoullis, this would be the count vector $a(j)$, which is the number of cluster j was used in $\mathbf{z}$, plus the matrix $B(j, v)$, which is of the number of times the hidden state was j and the observed letter was v.) Let $\mathbf{s}_i = \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}_i, \boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z})$ be the expected sufficient statistics for case i, and $\boldsymbol{\mu} = \sum_{i=1}^N \mathbf{s}_i$ be the sum of the ESS. Given $\boldsymbol{\mu}$, we can derive an ML or MAP estimate of the parameters in the M step; we will denote this operation by $\boldsymbol{\theta}(\boldsymbol{\mu})$. (For example, in the case of mixtures of multinoullis, we just need to normalize $\mathbf{a}$ and each row of $\mathbf{B}$.) With this notation under our belt, the pseudo code for batch EM is as shown in Algorithm 8.

Algorithm 11.2: Batch EM algorithm

```
initialize $\boldsymbol{\mu}$;
repeat
    $\boldsymbol{\mu}^{new} = \mathbf{0}$ ;
    for each example $i = 1 : N$ do
        $\mathbf{s}_i := \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}_i, \boldsymbol{\theta}(\boldsymbol{\mu}))\boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z})$ ;
        $\boldsymbol{\mu}^{new} := \boldsymbol{\mu}^{new} + \mathbf{s}_i$; ;
    $\boldsymbol{\mu} := \boldsymbol{\mu}^{new}$;
until converged;
```

11.4.8.2 Incremental EM

In incremental EM (Neal and Hinton 1998), we keep track of $\boldsymbol{\mu}$ as well as the $\mathbf{s}_i$. When we come to a data case, we swap out the old $\mathbf{s}_i$ and replace it with the new $\mathbf{s}_i^{new}$, as shown in the code in Algorithm 8. Note that we can exploit the sparsity of $\mathbf{s}_i^{new}$ to speedup the computation of $\boldsymbol{\theta}$, since most components of $\boldsymbol{\mu}$ wil not have changed.

Algorithm 11.3: Incremental EM algorithm

```
initialize $\mathbf{s}_i$ for $i = 1 : N$;
$\boldsymbol{\mu} = \sum_i \mathbf{s}_i$;
repeat
    for each example $i = 1 : N$ in a random order do
        $\mathbf{s}_i^{new} := \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}_i, \boldsymbol{\theta}(\boldsymbol{\mu}))\boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z})$ ;
        $\boldsymbol{\mu} := \boldsymbol{\mu} + \mathbf{s}_i^{new} - \mathbf{s}_i$;
        $\mathbf{s}_i := \mathbf{s}_i^{new}$;
until converged;
```

This can be viewed as maximizing the lower bound $Q(\boldsymbol{\theta}, q_1, \ldots, q_N)$ by optimizing q_1, then $\boldsymbol{\theta}$, then q_2, then $\boldsymbol{\theta}$, etc. As such, this method is guaranteed to monotonically converge to a local maximum of the lower bound and to the log likelihood itself.

11.4.8.3 Stepwise EM

In stepwise EM, whenever we compute a new $\mathbf{s}_i$, we move $\boldsymbol{\mu}$ towards it, as shown in Algorithm 7.[2] At iteration k, the stepsize has value η_k, which must satisfy the Robbins-Monro conditions in Equation 8.82. For example, (Liang and Klein Liang and Klein) use $\eta_k = (2 + k)^{-\kappa}$ for $0.5 < \kappa \leq 1$. We can get somewhat better behavior by using a minibatch of size m before each update. It is possible to optimize m and κ to maximize the training set likelihood, by

[2] A detail: As written the update for $\boldsymbol{\mu}$ does not exploit the sparsity of $\mathbf{s}_i$. We can fix this by storing $\mathbf{m} = \frac{\boldsymbol{\mu}}{\prod_{j<k}(1-\eta_j)}$ instead of $\boldsymbol{\mu}$, and then using the sparse update $\mathbf{m} := \mathbf{m} + \frac{\eta_k}{\prod_{j<k}(1-\eta_j)}\mathbf{s}_i$. This will not affect the results (i.e., $\boldsymbol{\theta}(\boldsymbol{\mu}) = \boldsymbol{\theta}(\mathbf{m})$), since scaling the counts by a global constant has no effect.

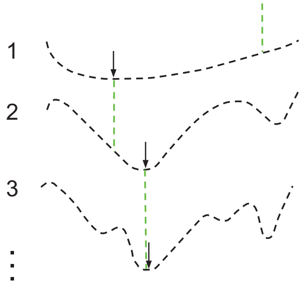


Figure 11.17 Illustration of deterministic annealing. Based on `http://en.wikipedia.org/wiki/Graduated_optimization`.

trying different values in parallel for an initial trial period; this can significantly speed up the algorithm.

Algorithm 11.4: Stepwise EM algorithm

```
initialize $\boldsymbol{\mu}$; $k = 0$ ;
repeat
    for each example $i = 1 : N$ in a random order do
        $\mathbf{s}_i := \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}_i, \boldsymbol{\theta}(\boldsymbol{\mu}))\boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z})$ ;
        $\boldsymbol{\mu} := (1 - \eta_k)\boldsymbol{\mu} + \eta_k \mathbf{s}_i$;
        $k := k + 1$
until converged;
```

(Liang and Klein Liang and Klein) compare batch EM, incremental EM, and stepwise EM on four different unsupervised language modeling tasks. They found that stepwise EM (using $\kappa \approx 0.7$ and $m \approx 1000$) was faster than incremental EM, and both were much faster than batch EM. In terms of accuracy, stepwise EM was usually as good or sometimes even better than batch EM; incremental EM was often worse than either of the other methods.

11.4.9 Other EM variants *

EM is one of the most widely used algorithms in statistics and machine learning. Not surprisingly, many variations have been proposed. We briefly mention a few below, some of which we will use in later chapters. See (McLachlan and Krishnan 1997) for more information.

- **Annealed EM** In general, EM will only converge to a local maximum. To increase the chance of finding the global maximum, we can use a variety of methods. One approach is to use a method known as **deterministic annealing** (Rose 1998). The basic idea is to “smooth” the posterior “landscape” by raising it to a temperature, and then gradually cooling it, all the while slowly tracking the global maximum. See Figure 11.17. for a sketch. (A stochastic version

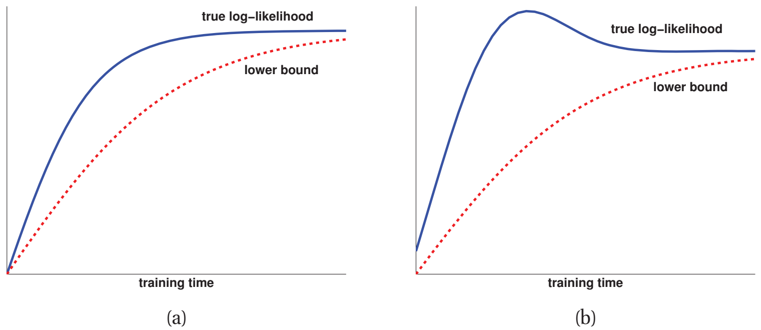
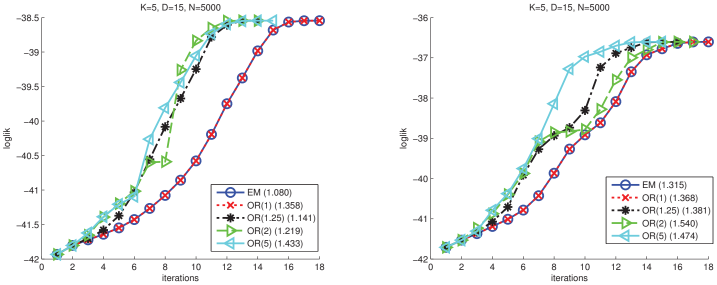


Figure 11.18 Illustration of possible behaviors of variational EM. (a) The lower bound increases at each iteration, and so does the likelihood. (b) The lower bound increases but the likelihood decreases. In this case, the algorithm is closing the gap between the approximate and true posterior. This can have a regularizing effect. Based on Figure 6 of (Saul et al. 1996). Figure generated by `varEMbound`.

of this algorithm is described in Section 24.6.1.) An annealed version of EM is described in (Ueda and Nakano 1998).

- **Variational EM** In Section 11.4.7, we showed that the optimal thing to do in the E step is to make q_i be the exact posterior over the latent variables, $q_i^t(\mathbf{z}_i) = p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta}^t)$. In this case, the lower bound on the log likelihood will be tight, so the M step will "push up" on the log-likelihood itself. However, sometimes it is computationally intractable to perform exact inference in the E step, but we may be able to perform approximate inference. If we can ensure that the E step is performing inference based on a a lower bound to the likelihood, then the M step can be seen as monotonically increasing this lower bound (see Figure 11.18). This is called **variational EM** (Neal and Hinton 1998). See Chapter 21 for some variational inference methods that can be used in the E step.
- **Monte Carlo EM** Another approach to handling an intractable E step is to use a Monte Carlo approximation to the expected sufficient statistics. That is, we draw samples from the posterior, $\mathbf{z}_i^s \sim p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta}^t)$, and then compute the sufficient statistics for each completed vector, $(\mathbf{x}_i, \mathbf{z}_i^s)$, and then average the results. This is called **Monte Carlo EM** or **MCEM** (Wei and Tanner 1990). (If we only draw a single sample, it is called **stochastic EM** (Celeux and Diebolt 1985).) One way to draw samples is to use MCMC (see Chapter 24). However, if we have to wait for MCMC to converge inside each E step, the method becomes very slow. An alternative is to use stochastic approximation, and only perform "brief" sampling in the E step, followed by a partial parameter update. This is called **stochastic approximation EM** (Delyon et al. 1999) and tends to work better than MCEM. Another alternative is to apply MCMC to infer the parameters as well as the latent variables (a fully Bayesian approach), thus eliminating the distinction between E and M steps. See Chapter 24 for details.
- **Generalized EM** Sometimes we can perform the E step exactly, but we cannot perform the M step exactly. However, we can still monotonically increase the log likelihood by performing a "partial" M step, in which we merely increase the expected complete data log likelihood, rather than maximizing it. For example, we might follow a few gradient steps. This is called



(a) (b)

Figure 11.19 Illustration of adaptive over-relaxed EM applied to a mixture of 5 Gaussians in 15 dimensions. We show the algorithm applied to two different datasets, randomly sampled from a mixture of 10 Gaussians. We plot the convergence for different update rates η. Using $\eta = 1$ gives the same results as regular EM. The actual running time is printed in the legend. Figure generated by `mixGaussOverRelaxedEmDemo`.

the **generalized EM** or **GEM** algorithm. (This is an unfortunate term, since there are many ways to generalize EM....)

- **ECM(E) algorithm** The **ECM** algorithm stands for "expectation conditional maximization", and refers to optimizing the parameters in the M step sequentially, if they turn out to be dependent. The **ECME** algorithm, which stands for "ECM either" (Liu and Rubin 1995), is a variant of ECM in which we maximize the expected complete data log likelihood (the Q function) as usual, or the observed data log likelihood, during one or more of the conditional maximization steps. The latter can be much faster, since it ignores the results of the E step, and directly optimizes the objective of interest. A standard example of this is when fitting the Student T distribution. For fixed ν, we can update $\boldsymbol{\Sigma}$ as usual, but then to update ν, we replace the standard update of the form $\nu^{t+1} = \arg\max_\nu Q((\boldsymbol{\mu}^{t+1}, \boldsymbol{\Sigma}^{t+1}, \nu), \boldsymbol{\theta}^t)$ with $\nu^{t+1} = \arg\max_\nu \log p(\mathcal{D}|\boldsymbol{\mu}^{t+1}, \boldsymbol{\Sigma}^{t+1}, \nu)$. See (McLachlan and Krishnan 1997) for more information.
- **Over-relaxed EM** Vanilla EM can be quite slow, especially if there is lots of missing data. The adaptive **overrelaxed EM algorithm** (Salakhutdinov and Roweis 2003) performs an update of the form $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \eta(M(\boldsymbol{\theta}^t) - \boldsymbol{\theta}^t)$, where η is a step-size parameter, and $M(\boldsymbol{\theta}^t)$ is the usual update computed during the M step. Obviously this reduces to standard EM if $\eta = 1$, but using larger values of η can result in faster convergence. See Figure 11.19 for an illustration. Unfortunately, using too large a value of η can cause the algorithm to fail to converge.

Finally, note that EM is in fact just a special case of a larger class of algorithms known as **bound optimization** or **MM** algorithms (MM stands for **minorize-maximize**). See (Hunter and Lange 2004) for further discussion.

11.5 Model selection for latent variable models

When using LVMs, we must specify the number of latent variables, which controls the model complexity. In particuarl, in the case of mixture models, we must specify K, the number of clusters. Choosing these parameters is an example of model selection. We discuss some approaches below.

11.5.1 Model selection for probabilistic models

The optimal Bayesian approach, discussed in Section 5.3, is to pick the model with the largest marginal likelihood, $K^* = \text{argmax}_k\, p(\mathcal{D}|K)$.

There are two problems with this. First, evaluating the marginal likelihood for LVMs is quite difficult. In practice, simple approximations, such as BIC, can be used (see e.g., (Fraley and Raftery 2002)). Alternatively, we can use the cross-validated likelihood as a performance measure, although this can be slow, since it requires fitting each model F times, where F is the number of CV folds.

The second issue is the need to search over a potentially large number of models. The usual approach is to perform exhaustive search over all candidate values of K. However, sometimes we can set the model to its maximal size, and then rely on the power of the Bayesian Occam's razor to "kill off" unwanted components. An example of this will be shown in Section 21.6.1.6, when we discuss variational Bayes.

An alternative approach is to perform stochastic sampling in the space of models. Traditional approaches, such as (Green 1998, 2003; Lunn et al. 2009), are based on reversible jump MCMC, and use birth moves to propose new centers, and death moves to kill off old centers. However, this can be slow and difficult to implement. A simpler approach is to use a Dirichlet process mixture model, which can be fit using Gibbs sampling, but still allows for an unbounded number of mixture components; see Section 25.2 for details.

Perhaps surprisingly, these sampling-based methods can be faster than the simple approach of evaluating the quality of each K separately. The reason is that fitting the model for each K is often slow. By contrast, the sampling methods can often quickly determine that a certain value of K is poor, and thus they need not waste time in that part of the posterior.

11.5.2 Model selection for non-probabilistic methods

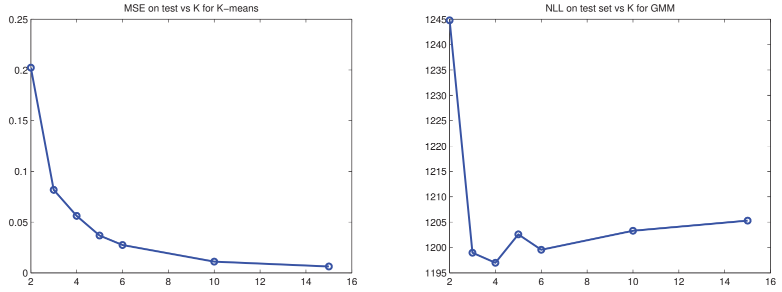
What if we are not using a probabilistic model? For example, how do we choose K for the K-means algorithm? Since this does not correspond to a probability model, there is no likelihood, so none of the methods described above can be used.

An obvious proxy for the likelihood is the reconstruction error. Define the squared reconstruction error of a data set $\mathcal{D}$, using model complexity K, as follows:

$$E(\mathcal{D}, K) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} ||\mathbf{x}_i - \hat{\mathbf{x}}_i||^2 \tag{11.95}$$

In the case of K-means, the reconstruction is given by $\hat{\mathbf{x}}_i = \boldsymbol{\mu}_{z_i}$, where $z_i = \text{argmin}_k\, ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$, as explained in Section 11.4.2.6.

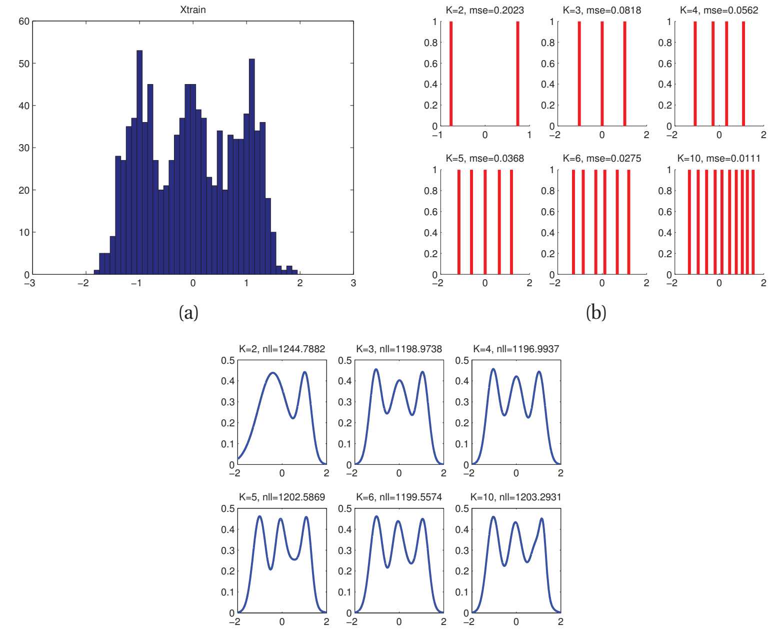
Figure 11.20(a) plots the reconstruction error on the *test set* for K-means. We notice that the error decreases with increasing model complexity! The reason for this behavior is as follows:



(a)

(b)

Figure 11.20 Test set performance vs K for data generated from a mixture of 3 Gaussians in 1d (data is shown in Figure 11.21(a)). (a) MSE on test set for K-means. (b) Negative log likelihood on test set for GMM. Figure generated by `kmeansModelSel1d`.



(a)

(b)

(c)

Figure 11.21 Synthetic data generated from a mixture of 3 Gaussians in 1d. (a) Histogram of training data. (Test data looks essentially the same.) (b) Centroids estimated by K-means for $K \in \{2, 3, 4, 5, 6, 10\}$. (c) GMM density model estimated by EM for for the same values of K. Figure generated by `kmeansModelSel1d`.

when we add more and more centroids to K-means, we can "tile" the space more densely, as shown in Figure 11.21(b). Hence any given test vector is more likely to find a close prototype to accurately represent it as K increases, thus decreasing reconstruction error. However, if we use a probabilistic model, such as the GMM, and plot the negative log-likelihood, we get the usual U-shaped curve on the test set, as shown in Figure 11.20(b).

In supervised learning, we can always use cross validation to select between non-probabilistic models of different complexity, but this is not the case with unsupervised learning. Although this is not a novel observation (e.g., it is mentioned in passing in (Hastie et al. 2009, p519), one of the standard references in this field), it is perhaps not as widely appreciated as it should be. In fact, it is one of the more compelling arguments in favor of probabilistic models.

Given that cross validation doesn't work, and supposing one is unwilling to use probabilistic models (for some bizarre reason...), how can one choose K? The most common approach is to plot the reconstruction error on the training set versus K, and to try to identify a **knee** or **kink** in the curve. The idea is that for $K < K^*$, where K^* is the "true" number of clusters, the rate of decrease in the error function will be high, since we are splitting apart things that should not be grouped together. However, for $K > K^*$, we are splitting apart "natural" clusters, which does not reduce the error by as much.

This kink-finding process can be automated by use of the **gap statistic** (Tibshirani et al. 2001). Nevertheless, identifying such kinks can be hard, as shown in Figure 11.20(a), since the loss function usually drops off gradually. A different approach to "kink finding" is described in Section 12.3.2.1.

11.6 Fitting models with missing data

Suppose we want to fit a joint density model by maximum likelihood, but we have "holes" in our data matrix, due to missing data (usually represented by NaNs). More formally, let $O_{ij} = 1$ if component j of data case i is observed, and let $O_{ij} = 0$ otherwise. Let $\mathbf{X}_v = \{x_{ij} : O_{ij} = 1\}$ be the visible data, and $\mathbf{X}_h = \{x_{ij} : O_{ij} = 0\}$ be the missing or hidden data. Our goal is to compute

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\mathbf{X}_v|\boldsymbol{\theta}, \mathbf{O}) \tag{11.96}$$

Under the missing at random assumption (see Section 8.6.2), we have

$$p(\mathbf{X}_v|\boldsymbol{\theta}, \mathbf{O}) = \prod_{i=1}^{N} p(\mathbf{x}_{iv}|\boldsymbol{\theta}) \tag{11.97}$$

where $\mathbf{x}_{iv}$ is a vector created from row i and the columns indexed by the set $\{j : O_{ij} = 1\}$. Hence the log-likelihood has the form

$$\log p(\mathbf{X}_v|\boldsymbol{\theta}) = \sum_{i} \log p(\mathbf{x}_{iv}|\boldsymbol{\theta}) \tag{11.98}$$

where

$$p(\mathbf{x}_{iv}|\boldsymbol{\theta}) = \sum_{\mathbf{x}_{ih}} p(\mathbf{x}_{iv}, \mathbf{x}_{ih}|\boldsymbol{\theta}) \tag{11.99}$$

and $\mathbf{x}_{ih}$ is the vector of hidden variables for case i (assumed discrete for notational simplicity). Substituting in, we get

$$\log p(\mathbf{X}_v|\boldsymbol{\theta}) = \sum_i \log \left[\sum_{\mathbf{x}_{ih}} p(\mathbf{x}_{iv}, \mathbf{x}_{ih}|\boldsymbol{\theta}) \right] \tag{11.100}$$

Unfortunately, this objective is hard to maximize. since we cannot push the log inside the sum. However, we can use the EM algorithm to compute a local optimum. We give an example of this below.

11.6.1 EM for the MLE of an MVN with missing data

Suppose we want to fit an MVN by maximum likelihood, but we have missing data. We can use EM to find a local maximum of the objective, as we explain below.

11.6.1.1 Getting started

To get the algorithm started, we can compute the MLE based on those rows of the data matrix that are fully observed. If there are no such rows, we can use some ad-hoc imputation procedures, and then compute an initial MLE.

11.6.1.2 E step

Once we have $\boldsymbol{\theta}^{t-1}$, we can compute the expected complete data log likelihood at iteration t as follows:

$$\begin{aligned}
Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) &= \mathbb{E}\left[\sum_{i=1}^{N} \log \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma}) | \mathcal{D}, \boldsymbol{\theta}^{t-1} \right] && (11.101) \\
&= -\frac{N}{2} \log |2\pi\boldsymbol{\Sigma}| - \frac{1}{2} \sum_i \mathbb{E}\left[(\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) \right] && (11.102) \\
&= -\frac{N}{2} \log |2\pi\boldsymbol{\Sigma}| - \frac{1}{2} \mathrm{tr}(\boldsymbol{\Sigma}^{-1} \sum_i \mathbb{E}\left[(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T \right] && (11.103) \\
&= -\frac{N}{2} \log |\boldsymbol{\Sigma}| - \frac{ND}{2} \log(2\pi) - \frac{1}{2} \mathrm{tr}(\boldsymbol{\Sigma}^{-1} \mathbb{E}\left[\mathbf{S}(\boldsymbol{\mu}) \right]) && (11.104)
\end{aligned}$$

where

$$\mathbb{E}\left[\mathbf{S}(\boldsymbol{\mu})\right] \triangleq \sum_i \left(\mathbb{E}\left[\mathbf{x}_i \mathbf{x}_i^T \right] + \boldsymbol{\mu}\boldsymbol{\mu}^T - 2\boldsymbol{\mu}\mathbb{E}\left[\mathbf{x}_i\right]^T \right) \tag{11.105}$$

(We drop the conditioning of the expectation on $\mathcal{D}$ and $\boldsymbol{\theta}^{t-1}$ for brevity.) We see that we need to compute $\sum_i \mathbb{E}\left[\mathbf{x}_i\right]$ and $\sum_i \mathbb{E}\left[\mathbf{x}_i \mathbf{x}_i^T\right]$; these are the expected sufficient statistics.

To compute these quantities, we use the results from Section 4.3.1. Specifically, consider case i, where components v are observed and components h are unobserved. We have

$$\begin{aligned}
\mathbf{x}_{ih}|\mathbf{x}_{iv}, \boldsymbol{\theta} &\sim \mathcal{N}(\mathbf{m}_i, \mathbf{V}_i) && (11.106) \\
\mathbf{m}_i &\triangleq \boldsymbol{\mu}_h + \boldsymbol{\Sigma}_{hv} \boldsymbol{\Sigma}_{vv}^{-1} (\mathbf{x}_{iv} - \boldsymbol{\mu}_v) && (11.107) \\
\mathbf{V}_i &\triangleq \boldsymbol{\Sigma}_{hh} - \boldsymbol{\Sigma}_{hv} \boldsymbol{\Sigma}_{vv}^{-1} \boldsymbol{\Sigma}_{vh} && (11.108)
\end{aligned}$$

Hence the expected sufficient statistics are

$$\mathbb{E}\left[\mathbf{x}_i\right] = \left(\mathbb{E}\left[\mathbf{x}_{ih}\right]; \mathbf{x}_{iv}\right) = \left(\mathbf{m}_i; \mathbf{x}_{iv}\right) \tag{11.109}$$

where we have assumed (without loss of generality) that the unobserved variables come before the observed variables in the node ordering.

To compute $\mathbb{E}\left[\mathbf{x}_i\mathbf{x}_i^T\right]$, we use the result that $\text{cov}\left[\mathbf{x}\right] = \mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right] - \mathbb{E}\left[\mathbf{x}\right]\mathbb{E}\left[\mathbf{x}^T\right]$. Hence

$$\begin{aligned} \mathbb{E}\left[\mathbf{x}_i\mathbf{x}_i^T\right] &= \mathbb{E}\left[\begin{pmatrix}\mathbf{x}_{ih}\\ \mathbf{x}_{iv}\end{pmatrix}\begin{pmatrix}\mathbf{x}_{ih}^T & \mathbf{x}_{iv}^T\end{pmatrix}\right] = \begin{pmatrix}\mathbb{E}\left[\mathbf{x}_{ih}\mathbf{x}_{ih}^T\right] & \mathbb{E}\left[\mathbf{x}_{ih}\right]\mathbf{x}_{iv}^T \\ \mathbf{x}_{iv}\mathbb{E}\left[\mathbf{x}_{ih}\right]^T & \mathbf{x}_{iv}\mathbf{x}_{iv}^T\end{pmatrix} && (11.110) \\ \mathbb{E}\left[\mathbf{x}_{ih}\mathbf{x}_{ih}^T\right] &= \mathbb{E}\left[\mathbf{x}_{ih}\right]\mathbb{E}\left[\mathbf{x}_{ih}\right]^T + \mathbf{V}_i && (11.111) \end{aligned}$$

11.6.1.3 M step

By solving $\nabla Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t-1)}) = \mathbf{0}$, we can show that the M step is equivalent to plugging these ESS into the usual MLE equations to get

$$\begin{aligned} \boldsymbol{\mu}^t &= \frac{1}{N}\sum_i \mathbb{E}\left[\mathbf{x}_i\right] && (11.112) \\ \boldsymbol{\Sigma}^t &= \frac{1}{N}\sum_i \mathbb{E}\left[\mathbf{x}_i\mathbf{x}_i^T\right] - \boldsymbol{\mu}^t(\boldsymbol{\mu}^t)^T && (11.113) \end{aligned}$$

Thus we see that EM is *not* equivalent to simply replacing variables by their expectations and applying the standard MLE formula; that would ignore the posterior variance and would result in an incorrect estimate. Instead we must compute the expectation of the sufficient statistics, and plug that into the usual equation for the MLE. We can easily modify the algorithm to perform MAP estimation, by plugging in the ESS into the equation for the MAP estimate. For an implementation, see `gaussMissingFitEm`.

11.6.1.4 Example

As an example of this procedure in action, let us reconsider the imputation problem from Section 4.3.2.3, which had $N = 100$ 10-dimensional data cases, with 50% missing data. Let us fit the parameters using EM. Call the resulting parameters $\hat{\boldsymbol{\theta}}$. We can use our model for predictions by computing $\mathbb{E}\left[\mathbf{x}_{ih}|\mathbf{x}_{iv}, \hat{\boldsymbol{\theta}}\right]$. Figure 11.22(a-b) indicates that the results obtained using the learned parameters are almost as good as with the true parameters. Not surprisingly, performance improves with more data, or as the fraction of missing data is reduced.

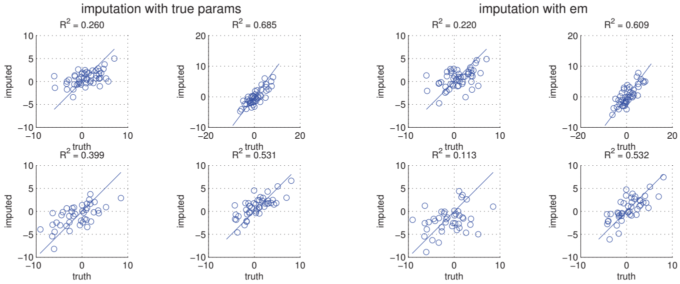
11.6.1.5 Extension to the GMM case

It is straightforward to fit a mixture of Gaussians in the presence of partially observed data vectors $\mathbf{x}_i$. We leave the details as an exercise.

Exercises

Exercise 11.1 Student T as infinite mixture of Gaussians

Derive Equation 11.61. For simplicity, assume a one-dimensional distribution.



(a) (b)

Figure 11.22 Illustration of data imputation. (a) Scatter plot of true values vs imputed values using true parameters. (b) Same as (b), but using parameters estimated with EM. Figure generated by `gaussImputationDemo`.

Exercise 11.2 EM for mixtures of Gaussians

Show that the M step for ML estimation of a mixture of Gaussians is given by

$$\boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\mathbf{x}_i}{r_k} \tag{11.114}$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{r_k} = \frac{\sum_i r_{ik}\mathbf{x}_i\mathbf{x}_i^T - r_k\boldsymbol{\mu}_k\boldsymbol{\mu}_k^T}{r_k} \tag{11.115}$$

Exercise 11.3 EM for mixtures of Bernoullis

- Show that the M step for ML estimation of a mixture of Bernoullis is given by

$$\mu_{kj} = \frac{\sum_i r_{ik}x_{ij}}{\sum_i r_{ik}} \tag{11.116}$$

- Show that the M step for MAP estimation of a mixture of Bernoullis with a $\beta(\alpha, \beta)$ prior is given by

$$\mu_{kj} = \frac{(\sum_i r_{ik}x_{ij}) + \alpha - 1}{(\sum_i r_{ik}) + \alpha + \beta - 2} \tag{11.117}$$

Exercise 11.4 EM for mixture of Student distributions

Derive the EM algorithm for ML estimation of a mixture of multivariate Student T distributions.

Exercise 11.5 Gradient descent for fitting GMM

Consider the Gaussian mixture model

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{11.118}$$

Define the log likelihood as

$$\ell(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}_n|\boldsymbol{\theta}) \tag{11.119}$$

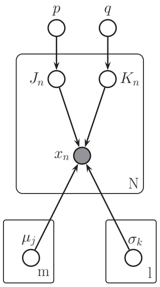


Figure 11.23 A mixture of Gaussians with two discrete latent indicators. J_n specifies which mean to use, and K_n specifies which variance to use.

Define the posterior responsibility that cluster k has for datapoint n as follows:

$$r_{nk} \triangleq p(z_n = k|\mathbf{x}_n, \boldsymbol{\theta}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'=1}^{K} \pi_{k'} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} \tag{11.120}$$

a. Show that the gradient of the log-likelihood wrt $\boldsymbol{\mu}_k$ is

$$\frac{d}{d\boldsymbol{\mu}_k}\ell(\boldsymbol{\theta}) = \sum_n r_{nk}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k) \tag{11.121}$$

b. Derive the gradient of the log-likelihood wrt π_k. (For now, ignore any constraints on π_k.)

c. One way to handle the constraint that $\sum_{k=1}^{K} \pi_k = 1$ is to reparameterize using the softmax function:

$$\pi_k \triangleq \frac{e^{w_k}}{\sum_{k'=1}^{K} e^{w_{k'}}} \tag{11.122}$$

Here $w_k \in \mathbb{R}$ are unconstrained parameters. Show that

$$\frac{d}{dw_k}\ell(\boldsymbol{\theta}) = \sum_n r_{nk} - \pi_k \tag{11.123}$$

(There may be a constant factor missing in the above expression...) Hint: use the chain rule and the fact that

$$\frac{d\pi_j}{dw_k} = \begin{cases} \pi_j(1-\pi_j) & \text{if } j = k \\ -\pi_j\pi_k & \text{if } j \neq k \end{cases} \tag{11.124}$$

which follows from Exercise 8.4(1).

d. Derive the gradient of the log-likelihood wrt $\boldsymbol{\Sigma}_k$. (For now, ignore any constraints on $\boldsymbol{\Sigma}_k$.)

e. One way to handle the constraint that $\boldsymbol{\Sigma}_k$ be a symmetric positive definite matrix is to reparameterize using a Cholesky decomposition, $\boldsymbol{\Sigma}_k = \mathbf{R}_k^T\mathbf{R}$, where $\mathbf{R}$ is an upper-triangular, but otherwise unconstrained matrix. Derive the gradient of the log-likelihood wrt $\mathbf{R}_k$.

Exercise 11.6 EM for a finite scale mixture of Gaussians

(Source: Jaakkola..) Consider the graphical model in Figure 11.23 which defines the following:

$$p(x_n|\theta) = \sum_{j=1}^{m} p_j \left[\sum_{k=1}^{l} q_k N(x_n|\mu_j, \sigma_k^2)\right] \tag{11.125}$$

where $\theta = \{p_1, \ldots, p_m, \mu_1, \ldots, \mu_m, q_1, \ldots, q_l, \sigma_1^2, \ldots, \sigma_l^2\}$ are all the parameters. Here $p_j \triangleq P(J_n = j)$ and $q_k \triangleq P(K_n = k)$ are the equivalent of mixture weights. We can think of this as a mixture of m non-Gaussian components, where each component distribution is a scale mixture, $p(x|j;\theta) = \sum_{k=1}^{l} q_k N(x; \mu_j, \sigma_k^2)$, combining Gaussians with different variances (scales).

We will now derive a generalized EM algorithm for this model. (Recall that in generalized EM, we do a partial update in the M step, rather than finding the exact maximum.)

a. Derive an expression for the responsibilities, $P(J_n = j, K_n = k|x_n, \theta)$, needed for the E step.

b. Write out a full expression for the expected complete log-likelihood

$$Q(\theta^{new}, \theta^{old}) = E_{\theta^{old}} \sum_{n=1}^{N} \log P(J_n, K_n, x_n|\theta^{new}) \tag{11.126}$$

c. Solving the M-step would require us to jointly optimize the means $\mu_1, \ldots, \mu_m$ and the variances $\sigma_1^2, \ldots, \sigma_l^2$. It will turn out to be simpler to first solve for the μ_j's given fixed σ_j^2's, and subsequently solve for σ_j^2's given the new values of μ_j's. For brevity, we will just do the first part. Derive an expression for the maximizing μ_j's given fixed $\sigma_{1:l}^2$, i.e., solve $\frac{\partial Q}{\partial \mu^{new}} = 0$.

Exercise 11.7 Manual calculation of the M step for a GMM

(Source: de Freitas.) In this question we consider clustering 1D data with a mixture of 2 Gaussians using the EM algorithm. You are given the 1-D data points $x = [1 \quad 10 \quad 20]$. Suppose the output of the E step is the following matrix:

$$\mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0.4 & 0.6 \\ 0 & 1 \end{bmatrix} \tag{11.127}$$

where entry $r_{i,c}$ is the probability of obervation x_i belonging to cluster c (the responsibility of cluster c for data point i). You just have to compute the M step. You may state the equations for maximum likelihood estimates of these quantities (which you should know) without proof; you just have to apply the equations to this data set. You may leave your answer in fractional form. Show your work.

a. Write down the likelihood function you are trying to optimize.

b. After performing the M step for the mixing weights π_1, π_2, what are the new values?

c. After performing the M step for the means μ_1 and μ_2, what are the new values?

Exercise 11.8 Moments of a mixture of Gaussians

Consider a mixture of K Gaussians

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{11.128}$$

a. Show that

$$\mathbb{E}[\mathbf{x}] = \sum_k \pi_k \boldsymbol{\mu}_k \tag{11.129}$$

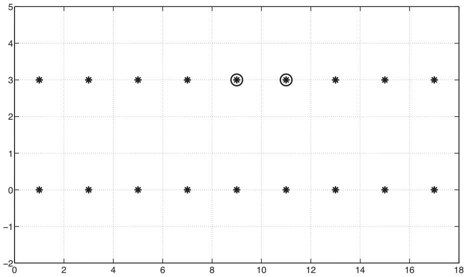


Figure 11.24 Some data points in 2d. Circles represent the initial guesses for $\mathbf{m}_1$ and $\mathbf{m}_2$.

b. Show that

$$\text{cov}[\mathbf{x}] = \sum_k \pi_k [\boldsymbol{\Sigma}_k + \boldsymbol{\mu}_k \boldsymbol{\mu}_k^T] - \mathbb{E}[\mathbf{x}]\,\mathbb{E}[\mathbf{x}]^T \tag{11.130}$$

Hint: use the fact that $\text{cov}[\mathbf{x}] = \mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right] - \mathbb{E}[\mathbf{x}]\,\mathbb{E}[\mathbf{x}]^T$.

Exercise 11.9 K-means clustering by hand

(Source: Jaakkola.)

In Figure 11.24, we show some data points which lie on the integer grid. (Note that the x-axis has been compressed; distances should be measured using the actual grid coordinates.) Suppose we apply the K-means algorithm to this data, using $K = 2$ and with the centers initialized at the two circled data points. Draw the final clusters obtained after K-means converges (show the approximate location of the new centers and group together all the points assigned to each center). Hint: think about shortest Euclidean distance.

Exercise 11.10 Deriving the K-means cost function

Show that

$$J_W(\mathbf{z}) = \frac{1}{2}\sum_{k=1}^{K} \sum_{i:z_i=k} \sum_{i':z_{i'}=k} (x_i - x_{i'})^2 = \sum_{k=1}^{K} n_k \sum_{i:z_i=k} (x_i - \overline{x}_k)^2 \tag{11.131}$$

Hint: note that, for any μ,

$$\begin{aligned}
\sum_i (x_i - \mu)^2 &= \sum_i [(x_i - \overline{x}) - (\mu - \overline{x})]^2 && (11.132)\\
&= \sum_i (x_i - \overline{x})^2 + \sum_i (\overline{x} - \mu)^2 - 2\sum_i (x_i - \overline{x})(\mu - \overline{x}) && (11.133)\\
&= ns^2 + n(\overline{x} - \mu)^2 && (11.134)
\end{aligned}$$

where $s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2$, since

$$\sum_i (x_i - \overline{x})(\mu - \overline{x}) = (\mu - \overline{x})\left((\sum_i x_i) - n\overline{x} \right) = (\mu - \overline{x})(n\overline{x} - n\overline{x}) = 0 \tag{11.135}$$

Exercise 11.11 Visible mixtures of Gaussians are in the exponential family

Show that the joint distribution $p(x, z|\boldsymbol{\theta})$ for a 1d GMM can be represented in exponential family form.

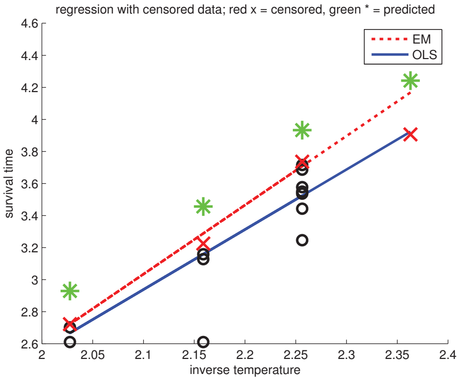


Figure 11.25 Example of censored linear regression. Black circles are observed training points, red crosses are observed but censored training points. Green stars are predicted values of the censored training points. We also show the lines fit by least squares (ignoring censoring) and by EM. Based on Figure 5.6 of (Tanner 1996). Figure generated by `linregCensoredSchmeeHahnDemo`, written by Hannes Bretschneider.

Exercise 11.12 EM for robust linear regression with a Student t likelihood

Consider a model of the form

$$p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2, \nu) = \mathcal{T}(y_i|\mathbf{w}^T\mathbf{x}_i, \sigma^2, \nu) \tag{11.136}$$

Derive an EM algorithm to compute the MLE for $\mathbf{w}$. You may assume ν and σ^2 are fixed, for simplicity. Hint: see Section 11.4.5.

Exercise 11.13 EM for EB estimation of Gaussian shrinkage model

Extend the results of Section 5.6.2.2 to the case where the σ_j^2 are not equal (but are known). Hint: treat the θ_j as hidden variables, and then to integrate them out in the E step, and maximize $\boldsymbol{\eta} = (\mu, \tau^2)$ in the M step.

Exercise 11.14 EM for censored linear regression

Censored regression refers to the case where one knows the outcome is at least (or at most) a certain value, but the precise value is unknown. This arises in many different settings. For example, suppose one is trying to learn a model that can predict how long a program will take to run, for different settings of its parameters. One may abort certain runs if they seem to be taking too long; the resulting run times are said to be **right censored**. For such runs, all we know is that $y_i \geq c_i$, where c_i is the censoring time, that is, $y_i = \min(z_i, c_i)$, where z_i is the true running time and y_i is the observed running time. We can also define **left censored** and **interval censored** models.[3] Derive an EM algorithm for fitting a linear regression model to right-censored data. Hint: use the results from Exercise 11.15. See Figure 11.25 for an example, based on the data from (Schmee and Hahn 1979). We notice that the EM line is tilted upwards more, since the model takes into account the fact that the truncated values are actually higher than the observed values.

[3] There is a closely related model in econometrics called the **Tobit model**, in which $y_i = \max(z_i, 0)$, so we only get to observe positive outcomes. An example of this is when z_i represents "desired investment", and y_i is actual investment. Probit regression (Section 9.4) is another example.

Exercise 11.15 Posterior mean and variance of a truncated Gaussian

Let $z_i = \mu_i + \sigma\epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0,1)$. Sometimes, such as in probit regression or censored regression, we do not observe z_i, but we observe the fact that it is above some threshold, namely we observe the event $E = \mathbb{I}(z_i \geq c_i) = \mathbb{I}(\epsilon_i \geq \frac{c_i-\mu_i}{\sigma})$. (See Exercise 11.14 for details on censored regression, and Section 11.4.6 for probit regression.) Show that

$$\mathbb{E}\left[z_i|z_i \geq c_i\right] \quad = \quad \mu_i + \sigma H\left(\frac{c_i-\mu_i}{\sigma}\right) \tag{11.137}$$

and

$$\mathbb{E}\left[z_i^2|z_i \geq c_i\right] \quad = \quad \mu_i^2 + \sigma^2 + \sigma(c_i+\mu_i)H\left(\frac{c_i-\mu_i}{\sigma}\right) \tag{11.138}$$

where we have defined

$$H(u) \triangleq \frac{\phi(u)}{1-\Phi(u)} \tag{11.139}$$

and where $\phi(u)$ is the pdf of a standard Gaussian, and $\Phi(u)$ is its cdf.

Hint 1: we have $p(\epsilon_i|E) = \frac{p(\epsilon_i,E)}{p(E)}$, where E is some event of interest.

Hint 2: It can be shown that

$$\frac{d}{dw}\mathcal{N}(w|0,1) = -w\mathcal{N}(w|0,1) \tag{11.140}$$

and hence

$$\int_b^c w\mathcal{N}(w|0,1) = \mathcal{N}(b|0,1) - \mathcal{N}(c|0,1) \tag{11.141}$$