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:

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

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:

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

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

$$
\pi_k = \frac{1}{N}\sum_i r_{ik} = \frac{r_k}{N} \tag{11.28}
$$

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.

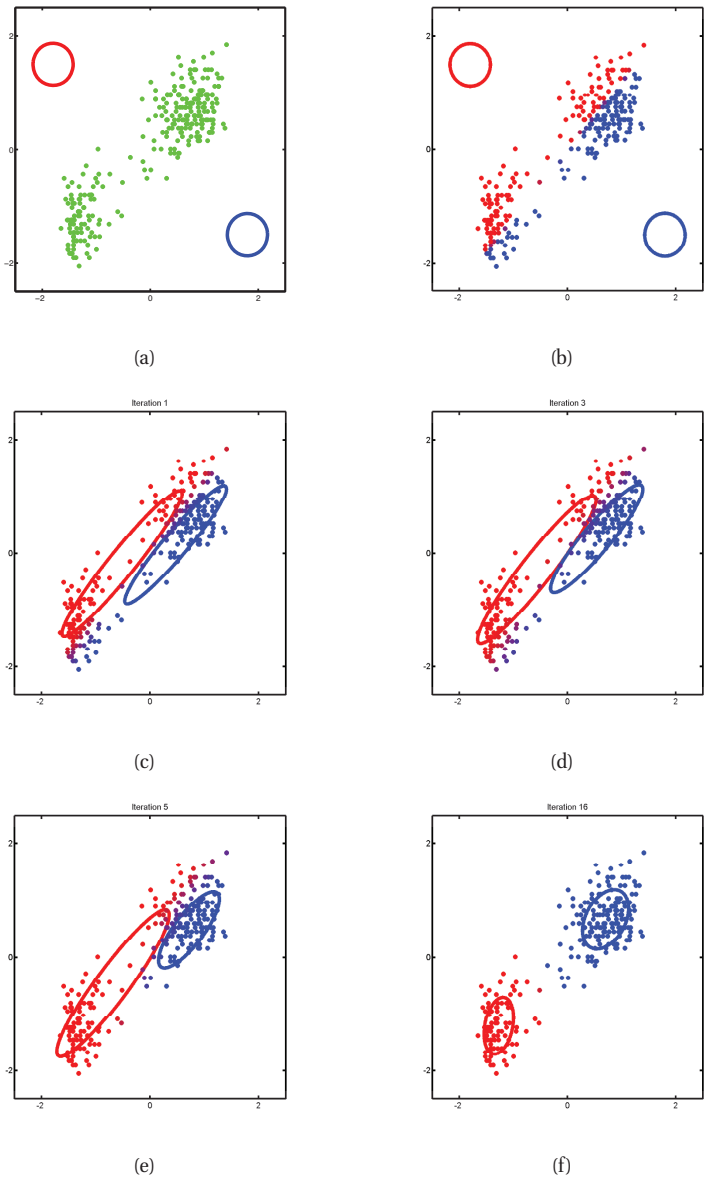
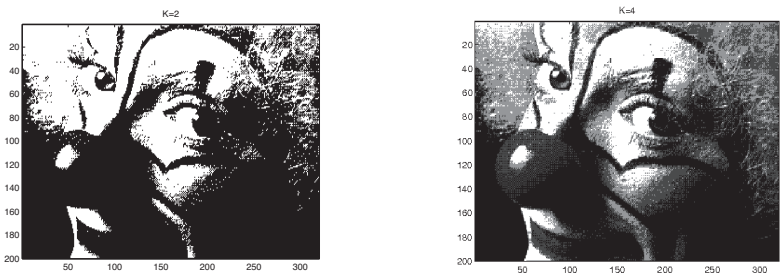


Figure 11.11 Illustration of the EM for a GMM applied to the Old Faithful data. (a) Initial (random) values of the parameters. (b) Posterior responsibility of each point computed in the first E step. The degree of redness indicates the degree to which the point belongs to the red cluster, and similarly 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:

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

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

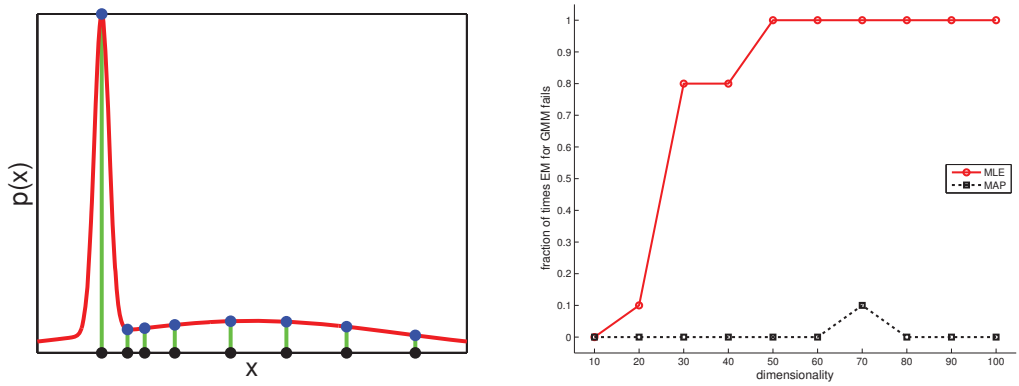
Both K-means and EM need to be initialized. It is common to pick K data points at random, and to make these be the initial cluster centers. Or we can pick the centers sequentially so as to try to "cover" the data. That is, we pick the initial point uniformly at random. Then each subsequent point is picked 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 $\boldsymbol{\Sigma}_k = \sigma_k^2 I$, 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 $\sigma_2 \to 0$, 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:

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

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, $\boldsymbol{\pi} \sim \mathrm{Dir}(\boldsymbol{\alpha})$, since this is conjugate to the categorical distribution. The MAP estimate is given by

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

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

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

From Section 4.6.3, the MAP estimate is given by

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

(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

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

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 $|\mathbf{S}_0| = \frac{1}{K}|\mathrm{diag}(s_1^2, \ldots, s_D^2)|$.) 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 $\mathbf{R}_k = \mathrm{diag}(r_{:,k})$. 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:

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

The log-likelihood of the complete data is given by

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

where $N_{tck} = \sum_{i=1}^{N} \mathbb{I}(x_{it} = i, \mathbf{x}_{i,\mathrm{pa}(t)} = c)$ 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

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

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

The quantity $p(x_{it}, \mathbf{x}_{i,\mathrm{pa}(t)}|\mathcal{D}_i, \boldsymbol{\theta})$ 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**:

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

(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

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

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

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

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

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

Now if $z_i \sim \mathrm{Ga}(a, b)$, 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

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

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

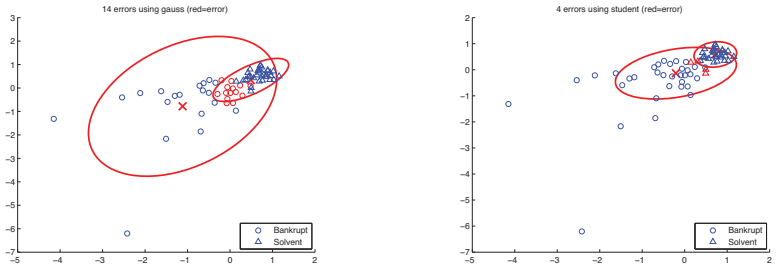
$$= \frac{1}{N} \left[\sum_i \overline{z}_i^{(t)} \mathbf{x}_i \mathbf{x}_i^T - \left(\sum_{i=1}^{N} \overline{z}_i^{(t)} \right) \hat{\boldsymbol{\mu}}^{(t+1)} (\hat{\boldsymbol{\mu}}^{(t+1)})^T \right] \tag{11.73}$$

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 $z_i \sim \mathrm{Ga}(a, b)$, 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

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

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

Substituting into Equation 11.68, we have

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

The gradient of this expression is equal to

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

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 $\mathbf{w}$:

$$\ell(\mathbf{z}, \mathbf{w}|\mathbf{V}_0) = \log p(\mathbf{y}|\mathbf{z}) + \log \mathcal{N}(\mathbf{z}|\mathbf{X}\mathbf{w}, \mathbf{I}) + \log \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{V}_0) \tag{11.79}$$

$$= \sum_i \log p(y_i|z_i) - \frac{1}{2}(\mathbf{z} - \mathbf{X}\mathbf{w})^T(\mathbf{z} - \mathbf{X}\mathbf{w}) - \frac{1}{2}\mathbf{w}^T\mathbf{V}_0^{-1}\mathbf{w} + \text{const} \tag{11.80}$$

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.

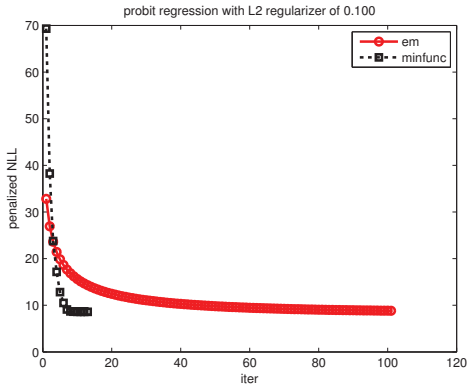


Figure 11.15 Fitting a probit regression model in 2d using a quasi-Newton method or EM. Figure generated by `probitRegDemo`.

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:

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

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:

$$L(\boldsymbol{\theta}, q_i) = \sum_{\mathbf{z}_i} q_i(\mathbf{z}_i) \log \frac{p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta})}{q_i(\mathbf{z}_i)} \tag{11.87}$$

$$= \sum_{\mathbf{z}_i} q_i(\mathbf{z}_i) \log \frac{p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta}) p(\mathbf{x}_i|\boldsymbol{\theta})}{q_i(\mathbf{z}_i)} \tag{11.88}$$

$$= \sum_{\mathbf{z}_i} q_i(\mathbf{z}_i) \log \frac{p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta})}{q_i(\mathbf{z}_i)} + \sum_{\mathbf{z}_i} q_i(\mathbf{z}_i) \log p(\mathbf{x}_i|\boldsymbol{\theta}) \tag{11.89}$$

$$= -\mathbb{KL}\left(q_i(\mathbf{z}_i) || p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta})\right) + \log p(\mathbf{x}_i|\boldsymbol{\theta}) \tag{11.90}$$

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

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

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

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

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

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

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

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

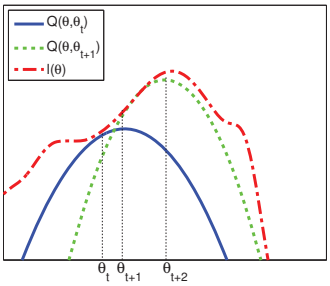


Figure 11.16 Illustration of EM as a bound optimization algorithm. Based on Figure 9.14 of (Bishop 2006a). Figure generated by `emLogLikelihoodMax`.

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.

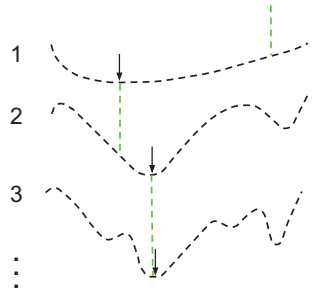


Figure 11.17 Illustration of deterministic annealing. Based on `http://en.wikipedia.org/wiki/Graduated_optimization`.

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

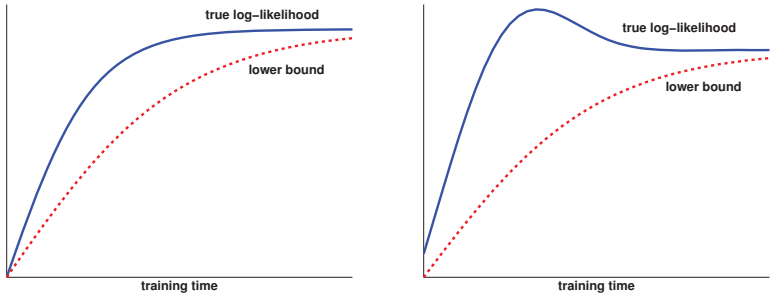
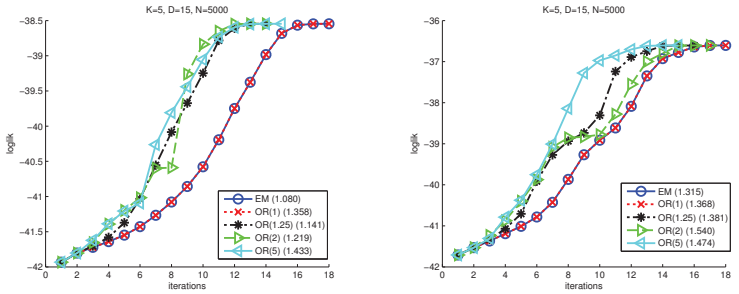


Figure 11.18 Illustration of possible behaviors of variational EM. (a) The lower bound increases at each iteration, and so does the likelihood. (b) The lower bound increases but the likelihood decreases. In this case, the algorithm is closing the gap between the approximate and true posterior. This can 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.