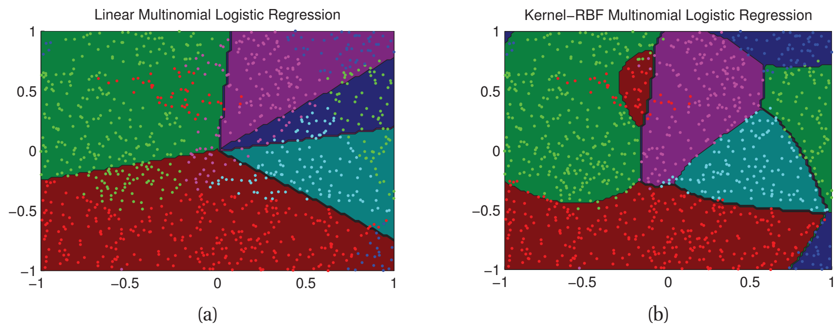


Figure 8.9 (a) Multinomial logistic regression for 5 classes in the original feature space. (b) After basis function expansion, using RBF kernels with a bandwidth of 1, and using all the data points as centers. Figure generated by `logregMultinomKernelDemo`.

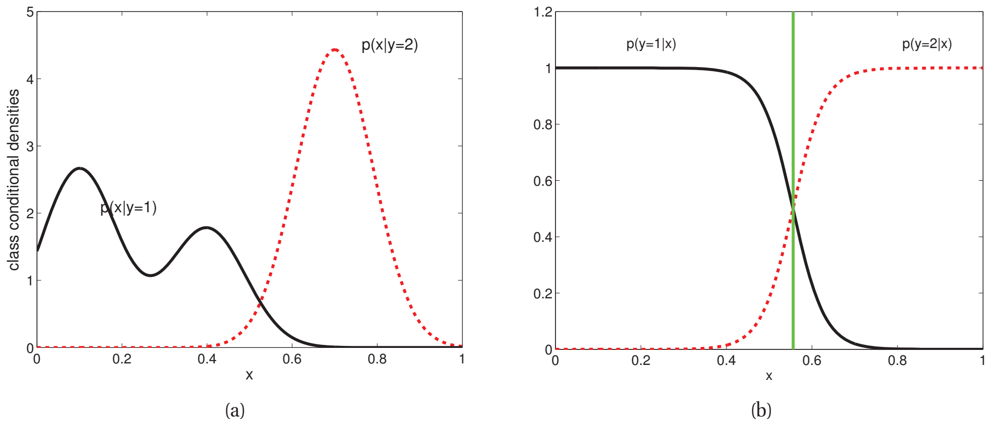


Figure 8.10 The class-conditional densities $p(x|y = c)$ (left) may be more complex than the class posteriors $p(y = c|x)$ (right). Based on Figure 1.27 of (Bishop 2006a). Figure generated by `generativeVsDiscrim`.

regression techniques we cover in this book.

8.6.2 Dealing with missing data

Sometimes some of the inputs (components of $\mathbf{x}$) are not observed; this could be due to a sensor failure, or a failure to complete an entry in a survey, etc. This is called the **missing data problem** (Little. and Rubin 1987). The ability to handle missing data in a principled way is one of the biggest advantages of generative models.

To formalize our assumptions, we can associate a binary response variable $r_i \in \{0, 1\}$, that specifies whether each value $\mathbf{x}_i$ is observed or not. The joint model has the form $p(\mathbf{x}_i, r_i|\boldsymbol{\theta}, \boldsymbol{\phi}) = p(r_i|\mathbf{x}_i, \boldsymbol{\phi})p(\mathbf{x}_i|\boldsymbol{\theta})$, where $\boldsymbol{\phi}$ are the parameters controlling whether the item

Model	Classif/regr	Gen/Discr	Param/Non	Section
Discriminant analysis	Classif	Gen	Param	Sec. 4.2.2, 4.2.4
Naive Bayes classifier	Classif	Gen	Param	Sec. 3.5, 3.5.1.2
Tree-augmented Naive Bayes classifier	Classif	Gen	Param	Sec. 10.2.1
Linear regression	Regr	Discrim	Param	Sec. 1.4.5, 7.3, 7.6,
Logistic regression	Classif	Discrim	Param	Sec. 1.4.6, 8.3.4, 8.4.3, 21.8.1.1
Sparse linear/ logistic regression	Both	Discrim	Param	Ch. 13
Mixture of experts	Both	Discrim	Param	Sec. 11.2.4
Multilayer perceptron (MLP)/ Neural network	Both	Discrim	Param	Ch. 16
Conditional random field (CRF)	Classif	Discrim	Param	Sec. 19.6
K nearest neighbor classifier	Classif	Gen	Non	Sec. 1.4.2, 14.7.3
(Infinite) Mixture Discriminant analysis	Classif	Gen	Non	Sec. 14.7.3
Classification and regression trees (CART)	Both	Discrim	Non	Sec. 16.2
Boosted model	Both	Discrim	Non	Sec. 16.4
Sparse kernelized lin/logreg (SKLR)	Both	Discrim	Non	Sec. 14.3.2
Relevance vector machine (RVM)	Both	Discrim	Non	Sec. 14.3.2
Support vector machine (SVM)	Both	Discrim	Non	Sec. 14.5
Gaussian processes (GP)	Both	Discrim	Non	Ch. 15
Smoothing splines	Regr	Discrim	Non	Section 15.4.6

Table 8.1 List of various models for classification and regression which we discuss in this book. Columns are as follows: Model name; is the model suitable for classification, regression, or both; is the model generative or discriminative; is the model parametric or non-parametric; list of sections in book which discuss the model. See also `http://pmtk3.googlecode.com/svn/trunk/docs/tutorial/html/tutSupervised.html` for the PMTK equivalents of these models. Any generative probabilistic model (e.g., HMMs, Boltzmann machines, Bayesian networks, etc.) can be turned into a classifier by using it as a class conditional density.

is observed or not. If we assume $p(r_i|\mathbf{x}_i, \boldsymbol{\phi}) = p(r_i|\boldsymbol{\phi})$, we say the data is **missing completely at random** or **MCAR**. If we assume $p(r_i|\mathbf{x}_i, \boldsymbol{\phi}) = p(r_i|\mathbf{x}_i^o, \boldsymbol{\phi})$, where $\mathbf{x}_i^o$ is the observed part of $\mathbf{x}_i$, we say the data is **missing at random** or **MAR**. If neither of these assumptions hold, we say the data is **not missing at random** or **NMAR**. In this case, we have to model the missing data mechanism, since the pattern of missingness is informative about the values of the missing data and the corresponding parameters. This is the case in most collaborative filtering problems, for example. See e.g., (Marlin 2008) for further discussion. We will henceforth assume the data is MAR.

When dealing with missing data, it is helpful to distinguish the cases when there is missingness only at test time (so the training data is **complete data**), from the harder case when there is missingness also at training time. We will discuss these two cases below. Note that the class label is always missing at test time, by definition; if the class label is also sometimes missing at training time, the problem is called semi-supervised learning.

8.6.2.1 Missing data at test time

In a generative classifier, we can handle features that are MAR by marginalizing them out. For example, if we are missing the value of x_1, we can compute

$$\begin{aligned} p(y=c|\mathbf{x}_{2:D},\boldsymbol{\theta}) &\propto p(y=c|\boldsymbol{\theta})p(\mathbf{x}_{2:D}|y=c,\boldsymbol{\theta}) && (8.94) \\ &= p(y=c|\boldsymbol{\theta})\sum_{x_1} p(x_1,\mathbf{x}_{2:D}|y=c,\boldsymbol{\theta}) && (8.95) \end{aligned}$$

If we make the naive Bayes assumption, the marginalization can be performed as follows:

$$\sum_{x_1} p(x_1, x_{2:D}|y=c,\boldsymbol{\theta}) = \left[\sum_{x_1} p(x_1|\boldsymbol{\theta}_{1c})\right]\prod_{j=2}^{D} p(x_j|\boldsymbol{\theta}_{jc}) = \prod_{j=2}^{D} p(x_j|\boldsymbol{\theta}_{jc}) \quad (8.96)$$

where we exploited the fact that $\sum_{x_1} p(x_1|y=c,\boldsymbol{\theta}) = 1$. Hence in a naive Bayes classifier, we can simply ignore missing features at test time. Similarly, in discriminant analysis, no matter what regularization method was used to estimate the parameters, we can always analytically marginalize out the missing variables (see Section 4.3):

$$p(\mathbf{x}_{2:D}|y=c,\boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_{2:D}|\boldsymbol{\mu}_{c,2:D}, \boldsymbol{\Sigma}_{c,2:D,2:D}) \quad (8.97)$$

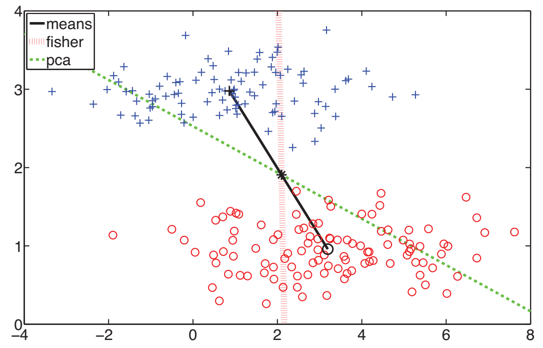
8.6.2.2 Missing data at training time

Missing data at training time is harder to deal with. In particular, computing the MLE or MAP estimate is no longer a simple optimization problem, for reasons discussed in Section 11.3.2. However, soon we will study are a variety of more sophisticated algorithms (such as EM algorithm, in Section 11.4) for finding approximate ML or MAP estimates in such cases.

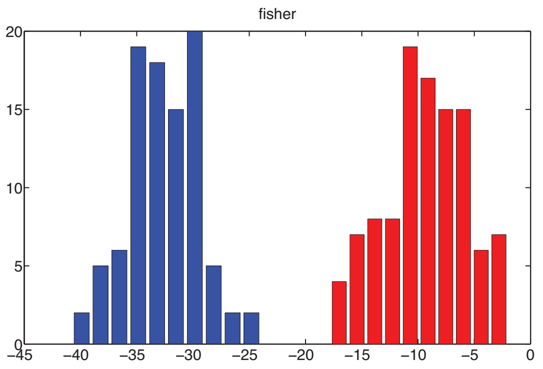
8.6.3 Fisher's linear discriminant analysis (FLDA) *

Discriminant analysis is a generative approach to classification, which requires fitting an MVN to the features. As we have discussed, this can be problematic in high dimensions. An alternative approach is to reduce the dimensionality of the features $\mathbf{x} \in \mathbb{R}^D$ and then fit an MVN to the resulting low-dimensional features $\mathbf{z} \in \mathbb{R}^L$. The simplest approach is to use a linear projection matrix, $\mathbf{z} = \mathbf{W}\mathbf{x}$, where $\mathbf{W}$ is a $L \times D$ matrix. One approach to finding $\mathbf{W}$ would be to use PCA (Section 12.2); the result would be very similar to RDA (Section 4.2.6), since SVD and PCA are essentially equivalent. However, PCA is an unsupervised technique that does not take class labels into account. Thus the resulting low dimensional features are not necessarily optimal for classification, as illustrated in Figure 8.11. An alternative approach is to find the matrix $\mathbf{W}$ such that the low-dimensional data can be classified as well as possible using a Gaussian class-conditional density model. The assumption of Gaussianity is reasonable since we are computing linear combinations of (potentially non-Gaussian) features. This approach is called **Fisher's linear discriminant analysis**, or **FLDA**.

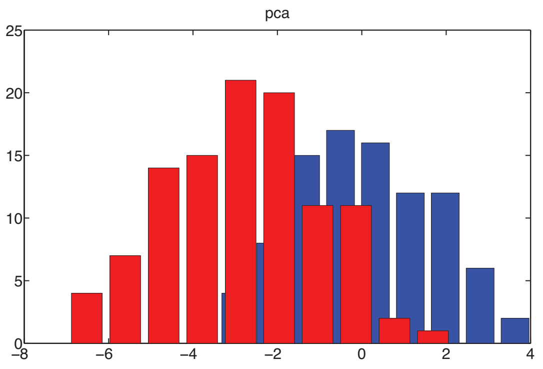
FLDA is an interesting hybrid of discriminative and generative techniques. The drawback of this technique is that it is restricted to using $L \leq C-1$ dimensions, regardless of D, for reasons that we will explain below. In the two-class case, this means we are seeking a single vector $\mathbf{w}$ onto which we can project the data. Below we derive the optimal $\mathbf{w}$ in the two-class case. We



(a)



(b)



(c)

Figure 8.11 Example of Fisher's linear discriminant. (a) Two class data in 2D. Dashed green line = first principal basis vector. Dotted red line = Fisher's linear discriminant vector. Solid black line joins the class-conditional means. (b) Projection of points onto Fisher's vector shows good class separation. (c) Projection of points onto PCA vector shows poor class separation. Figure generated by `fisherLDAdemo`.

then generalize to the multi-class case, and finally we give a probabilistic interpretation of this technique.

8.6.3.1 Derivation of the optimal 1d projection

We now derive this optimal direction $\mathbf{w}$, for the two-class case, following the presentation of (Bishop 2006b, Sec 4.1.4). Define the class-conditional means as

$$\boldsymbol{\mu}_1 = \frac{1}{N_1} \sum_{i:y_i=1} \mathbf{x}_i, \ \boldsymbol{\mu}_2 = \frac{1}{N_2} \sum_{i:y_i=2} \mathbf{x}_i \tag{8.98}$$

Let $m_k = \mathbf{w}^T \boldsymbol{\mu}_k$ be the projection of each mean onto the line $\mathbf{w}$. Also, let $z_i = \mathbf{w}^T \mathbf{x}_i$ be the projection of the data onto the line. The variance of the projected points is proportional to

$$s_k^2 = \sum_{i:y_i=k} (z_i - m_k)^2 \tag{8.99}$$

The goal is to find $\mathbf{w}$ such that we maximize the distance between the means, $m_2 - m_1$, while also ensuring the projected clusters are "tight":

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \tag{8.100}$$

We can rewrite the right hand side of the above in terms of $\mathbf{w}$ as follows

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{8.101}$$

where $\mathbf{S}_B$ is the between-class scatter matrix given by

$$\mathbf{S}_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \tag{8.102}$$

and $\mathbf{S}_W$ is the within-class scatter matrix, given by

$$\mathbf{S}_W = \sum_{i:y_i=1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T + \sum_{i:y_i=2} (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^T \tag{8.103}$$

To see this, note that

$$\mathbf{w}^T \mathbf{S}_B \mathbf{w} = \mathbf{w}^T (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \mathbf{w} = (m_2 - m_1)(m_2 - m_1) \tag{8.104}$$

and

$$\mathbf{w}^T \mathbf{S}_W \mathbf{w} = \sum_{i:y_i=1} \mathbf{w}^T (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T \mathbf{w} + \sum_{i:y_i=2} \mathbf{w}^T (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^T \mathbf{w} \tag{8.105}$$

$$= \sum_{i:y_i=1} (z_i - m_1)^2 + \sum_{i:y_i=2} (z_i - m_2)^2 \tag{8.106}$$

Equation 8.101 is a ratio of two scalars; we can take its derivative with respect to $\mathbf{w}$ and equate to zero. One can show (Exercise 12.6) that that $J(\mathbf{w})$ is maximized when

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w} \tag{8.107}$$

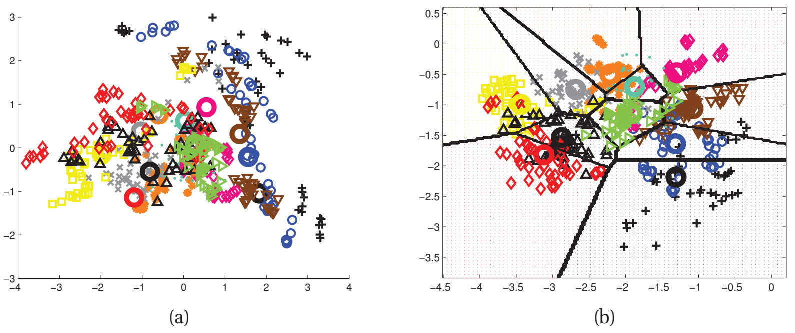


Figure 8.12 (a) PCA projection of vowel data to 2d. (b) FLDA projection of vowel data to 2d. We see there is better class separation in the FLDA case. Based on Figure 4.11 of (Hastie et al. 2009). Figure generated by `fisherDiscrimVowelDemo`, by Hannes Bretschneider.

where

$$\lambda = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{8.108}$$

Equation 8.107 is called a **generalized eigenvalue** problem. If $\mathbf{S}_W$ is invertible, we can convert it to a regular eigenvalue problem:

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w} \tag{8.109}$$

However, in the two class case, there is a simpler solution. In particular, since

$$\mathbf{S}_B \mathbf{w} = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \mathbf{w} = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(m_2 - m_1) \tag{8.110}$$

then, from Equation 8.109 we have

$$\lambda\, \mathbf{w} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(m_2 - m_1) \tag{8.111}$$

$$\mathbf{w} \propto \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \tag{8.112}$$

Since we only care about the directionality, and not the scale factor, we can just set

$$\mathbf{w} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \tag{8.113}$$

This is the optimal solution in the two-class case. If $\mathbf{S}_W \propto \mathbf{I}$, meaning the pooled covariance matrix is isotropic, then $\mathbf{w}$ is proportional to the vector that joins the class means. This is an intuitively reasonable direction to project onto, as shown in Figure 8.11.

8.6.3.2 Extension to higher dimensions and multiple classes

We can extend the above idea to multiple classes, and to higher dimensional subspaces, by finding a projection *matrix* $\mathbf{W}$ which maps from D to L so as to maximize

$$J(\mathbf{W}) = \frac{|\mathbf{W}\boldsymbol{\Sigma}_B \mathbf{W}^T|}{|\mathbf{W}\Sigma_W \mathbf{W}^T|} \tag{8.114}$$

where

$$\boldsymbol{\Sigma}_B \triangleq \sum_c \frac{N_c}{N}(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T \quad (8.115)$$

$$\boldsymbol{\Sigma}_W \triangleq \sum_c \frac{N_c}{N}\boldsymbol{\Sigma}_c \quad (8.116)$$

$$\boldsymbol{\Sigma}_c \triangleq \frac{1}{N_c}\sum_{i:y_i=c}(\mathbf{x}_i - \boldsymbol{\mu}_c)(\mathbf{x}_i - \boldsymbol{\mu}_c)^T \quad (8.117)$$

The solution can be shown to be

$$\mathbf{W} = \boldsymbol{\Sigma}_W^{-\frac{1}{2}}\mathbf{U} \quad (8.118)$$

where $\mathbf{U}$ are the L leading eigenvectors of $\boldsymbol{\Sigma}_W^{-\frac{1}{2}}\boldsymbol{\Sigma}_B\boldsymbol{\Sigma}_W^{-\frac{1}{2}}$, assuming $\boldsymbol{\Sigma}_W$ is non-singular. (If it is singular, we can first perform PCA on all the data.)

Figure 8.12 gives an example of this method applied to some $D = 10$ dimensional speech data, representing $C = 11$ different vowel sounds. We see that FLDA gives better class separation than PCA.

Note that FLDA is restricted to finding at most a $L \leq C - 1$ dimensional linear subspace, no matter how large D, because the rank of the between class covariance matrix $\boldsymbol{\Sigma}_B$ is $C - 1$. (The -1 term arises because of the $\boldsymbol{\mu}$ term, which is a linear function of the $\boldsymbol{\mu}_c$.) This is a rather severe restriction which limits the usefulness of FLDA.

8.6.3.3 Probabilistic interpretation of FLDA *

To find a valid probabilistic interpretation of FLDA, we follow the approach of (Kumar and Andreo 1998; Zhou et al. 2009). They proposed a model known as **heteroscedastic LDA** (HLDA), which works as follows. Let $\mathbf{W}$ be a $D \times D$ invertible matrix, and let $\mathbf{z}_i = \mathbf{W}\mathbf{x}_i$ be a transformed version of the data. We now fit full covariance Gaussians to the transformed data, one per class, but with the constraint that only the first L components will be class-specific; the remaining $H = D - L$ components will be shared across classes, and will thus not be discriminative. That is, we use

$$p(\mathbf{z}_i|\boldsymbol{\theta}, y_i = c) = \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \quad (8.119)$$

$$\boldsymbol{\mu}_c \triangleq (\mathbf{m}_c; \mathbf{m}_0) \quad (8.120)$$

$$\boldsymbol{\Sigma}_c \triangleq \begin{pmatrix} \mathbf{S}_c & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_0 \end{pmatrix} \quad (8.121)$$

where $\mathbf{m}_0$ is the shared H dimensional mean and $\mathbf{S}_0$ is the shared $H \times H$ covariace. The pdf of the original (untransformed) data is given by

$$p(\mathbf{x}_i|y_i = c, \mathbf{W}, \boldsymbol{\theta}) = |\mathbf{W}|\, \mathcal{N}(\mathbf{W}\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \quad (8.122)$$

$$= |\mathbf{W}|\, \mathcal{N}(\mathbf{W}_L\mathbf{x}_i|\mathbf{m}_c, \mathbf{S}_c)\, \mathcal{N}(\mathbf{W}_H\mathbf{x}_i|\mathbf{m}_0, \mathbf{S}_0) \quad (8.123)$$

where $\mathbf{W} = \begin{pmatrix} \mathbf{W}_L \\ \mathbf{W}_H \end{pmatrix}$. For fixed $\mathbf{W}$, it is easy to derive the MLE for $\boldsymbol{\theta}$. One can then optimize $\mathbf{W}$ using gradient methods.

In the special case that the $\mathbf{\Sigma}_c$ are diagonal, there is a closed-form solution for $\mathbf{W}$ (Gales 1999). And in the special case the $\mathbf{\Sigma}_c$ are all equal, we recover classical LDA (Zhou et al. 2009).

In view of this this result, it should be clear that HLDA will outperform LDA if the class covariances are not equal within the discriminative subspace (i.e., if the assumption that $\mathbf{\Sigma}_c$ is independent of c is a poor assumption). This is easy to demonstrate on synthetic data, and is also the case on more challenging tasks such as speech recognition (Kumar and Andreo 1998). Furthermore, we can extend the model by allowing each class to use its own projection matrix; this is known as **multiple LDA** (Gales 2002).

Exercises

Exercise 8.1 Spam classification using logistic regression

Consider the email spam data set discussed on p300 of (Hastie et al. 2009). This consists of 4601 email messages, from which 57 features have been extracted. These are as follows:

- 48 features, in $[0, 100]$, giving the percentage of words in a given message which match a given word on the list. The list contains words such as "business", "free", "george", etc. (The data was collected by George Forman, so his name occurs quite a lot.)
- 6 features, in $[0, 100]$, giving the percentage of characters in the email that match a given character on the list. The characters are ; ([! $ #
- Feature 55: The average length of an uninterrupted sequence of capital letters (max is 40.3, mean is 4.9)
- Feature 56: The length of the longest uninterrupted sequence of capital letters (max is 45.0, mean is 52.6)
- Feature 57: The sum of the lengts of uninterrupted sequence of capital letters (max is 25.6, mean is 282.2)

Load the data from `spamData.mat`, which contains a training set (of size 3065) and a test set (of size 1536).

One can imagine performing several kinds of preprocessing to this data. Try each of the following separately:

a. Standardize the columns so they all have mean 0 and unit variance.
b. Transform the features using $\log(x_{ij} + 0.1)$.
c. Binarize the features using $\mathbb{I}(x_{ij} > 0)$.

For each version of the data, fit a logistic regression model. Use cross validation to choose the strength of the ℓ_2 regularizer. Report the mean error rate on the training and test sets. You should get numbers similar to this:

method	train	test
stnd	0.082	0.079
log	0.052	0.059
binary	0.065	0.072

(The precise values will depend on what regularization value you choose.) Turn in your code and numerical results.

(See also Exercise 8.2.

Exercise 8.2 Spam classification using naive Bayes

We will re-examine the dataset from Exercise 8.1.

a. Use `naiveBayesFit` and `naiveBayesPredict` on the binarized spam data. What is the training and test error? (You can try different settings of the pseudocount α if you like (this corresponds to the $\mathrm{Beta}(\alpha, \alpha)$ prior each θ_{jc}), although the default of $\alpha = 1$ is probably fine.) Turn in your error rates.

b. Modify the code so it can handle real-valued features. Use a Gaussian density for each feature; fit it with maximum likelihood. What are the training and test error rates on the standardized data and the log transformed data? Turn in your 4 error rates and code.

Exercise 8.3 Gradient and Hessian of log-likelihood for logistic regression

a. Let $\sigma(a) = \frac{1}{1+e^{-a}}$ be the sigmoid function. Show that

$$\frac{d\sigma(a)}{da} = \sigma(a)(1 - \sigma(a)) \tag{8.124}$$

b. Using the previous result and the chain rule of calculus, derive an expression for the gradient of the log likelihood (Equation 8.5).

c. The Hessian can be written as $\mathbf{H} = \mathbf{X}^T\mathbf{S}\mathbf{X}$, where $\mathbf{S} \triangleq \mathrm{diag}(\mu_1(1-\mu_1), \ldots, \mu_n(1-\mu_n))$. Show that $\mathbf{H}$ is positive definite. (You may assume that $0 < \mu_i < 1$, so the elements of $\mathbf{S}$ will be strictly positive, and that $\mathbf{X}$ is full rank.)

Exercise 8.4 Gradient and Hessian of log-likelihood for multinomial logistic regression

a. Let $\mu_{ik} = \mathcal{S}(\boldsymbol{\eta}_i)_k$. Prove that the Jacobian of the softmax is

$$\frac{\partial \mu_{ik}}{\partial \eta_{ij}} = \mu_{ik}(\delta_{kj} - \mu_{ij}) \tag{8.125}$$

where $\delta_{kj} = I(k = j)$.

b. Hence show that

$$\nabla_{\mathbf{w}_c} \ell = \sum_i (y_{ic} - \mu_{ic})\mathbf{x}_i \tag{8.126}$$

Hint: use the chain rule and the fact that $\sum_c y_{ic} = 1$.

c. Show that the block submatrix of the Hessian for classes c and c' is given by

$$\mathbf{H}_{c,c'} = -\sum_i \mu_{ic}(\delta_{c,c'} - \mu_{i,c'})\mathbf{x}_i\mathbf{x}_i^T \tag{8.127}$$

Exercise 8.5 Symmetric version of ℓ_2 regularized multinomial logistic regression

(Source: Ex 18.3 of (Hastie et al. 2009).)

Multiclass logistic regression has the form

$$p(y = c|\mathbf{x}, \mathbf{W}) = \frac{\exp(w_{c0} + \mathbf{w}_c^T\mathbf{x})}{\sum_{k=1}^C \exp(w_{k0} + \mathbf{w}_k^T\mathbf{x})} \tag{8.128}$$

where $\mathbf{W}$ is a $(D+1) \times C$ weight matrix. We can arbitrarily define $\mathbf{w}_c = \mathbf{0}$ for one of the classes, say $c = C$, since $p(y = C|\mathbf{x}, \mathbf{W}) = 1 - \sum_{c=1}^{C-1} p(y = c|\mathbf{x}, \mathbf{w})$. In this case, the model has the form

$$p(y = c|\mathbf{x}, \mathbf{W}) = \frac{\exp(w_{c0} + \mathbf{w}_c^T\mathbf{x})}{1 + \sum_{k=1}^{C-1} \exp(w_{k0} + \mathbf{w}_k^T\mathbf{x})} \tag{8.129}$$

If we don't "clamp" one of the vectors to some constant value, the parameters will be unidentifiable. However, suppose we don't clamp $\mathbf{w}_c = \mathbf{0}$, so we are using Equation 8.128, but we add ℓ_2 regularization by optimizing

$$\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \mathbf{W}) - \lambda \sum_{c=1}^{C} ||\mathbf{w}_c||_2^2 \tag{8.130}$$

Show that at the optimum we have $\sum_{c=1}^{C} \hat{w}_{cj} = 0$ for $j = 1 : D$. (For the unregularized $\hat{w}_{c0}$ terms, we still need to enforce that $w_{0C} = 0$ to ensure identifiability of the offset.)

Exercise 8.6 Elementary properties of ℓ_2 regularized logistic regression

(Source: Jaaakkola.). Consider minimizing

$$J(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda ||\mathbf{w}||_2^2 \tag{8.131}$$

where

$$\ell(\mathbf{w}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \log \sigma(y_i \mathbf{x}_i^T \mathbf{w}) \tag{8.132}$$

is the average log-likelihood on data set $\mathcal{D}$, for $y_i \in \{-1, +1\}$. Answer the following true/ false questions.

a. $J(\mathbf{w})$ has multiple locally optimal solutions: T/F?
b. Let $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$ be a global optimum. $\hat{\mathbf{w}}$ is sparse (has many zero entries): T/F?
c. If the training data is linearly separable, then some weights w_j might become infinite if $\lambda = 0$: T/F?
d. $\ell(\hat{\mathbf{w}}, \mathcal{D}_{\text{train}})$ always increases as we increase λ: T/F?
e. $\ell(\hat{\mathbf{w}}, \mathcal{D}_{\text{test}})$ always increases as we increase λ: T/F?

Exercise 8.7 Regularizing separate terms in 2d logistic regression

(Source: Jaaakkola.)

a. Consider the data in Figure 8.13, where we fit the model $p(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$. Suppose we fit the model by maximum likelihood, i.e., we minimize

$$J(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) \tag{8.133}$$

where $\ell(\mathbf{w}, \mathcal{D}_{\text{train}})$ is the log likelihood on the training set. Sketch a possible decision boundary corresponding to $\hat{\mathbf{w}}$. (Copy the figure first (a rough sketch is enough), and then superimpose your answer on your copy, since you will need multiple versions of this figure). Is your answer (decision boundary) unique? How many classification errors does your method make on the training set?

b. Now suppose we regularize only the w_0 parameter, i.e., we minimize

$$J_0(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda w_0^2 \tag{8.134}$$

Suppose λ is a very large number, so we regularize w_0 all the way to 0, but all other parameters are unregularized. Sketch a possible decision boundary. How many classification errors does your method make on the training set? Hint: consider the behavior of simple linear regression, $w_0 + w_1 x_1 + w_2 x_2$ when $x_1 = x_2 = 0$.

c. Now suppose we heavily regularize only the w_1 parameter, i.e., we minimize

$$J_1(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda w_1^2 \tag{8.135}$$

Sketch a possible decision boundary. How many classification errors does your method make on the training set?

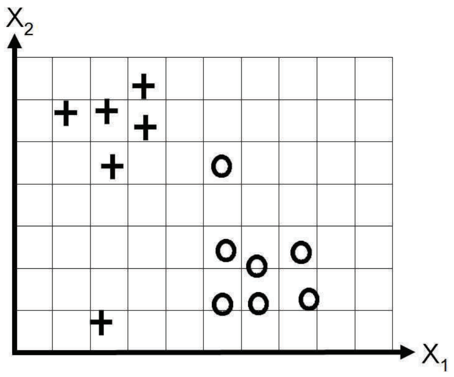


Figure 8.13 Data for logistic regression question.

d. Now suppose we heavily regularize only the w_2 parameter. Sketch a possible decision boundary. How many classification errors does your method make on the training set?

9 *Generalized linear models and the exponential family*

9.1 Introduction

We have now encountered a wide variety of probability distributions: the Gaussian, the Bernoulli, the Student t, the uniform, the gamma, etc. It turns out that most of these are members of a broader class of distributions known as the **exponential family**.[1] In this chapter, we discuss various properties of this family. This allows us to derive theorems and algorithms with very broad applicability.

We will see how we can easily use any member of the exponential family as a class-conditional density in order to make a generative classifier. In addition, we will discuss how to build discriminative models, where the response variable has an exponential family distribution, whose mean is a linear function of the inputs; this is known as a generalized linear model, and generalizes the idea of logistic regression to other kinds of response variables.

9.2 The exponential family

Before defining the exponential family, we mention several reasons why it is important:

- It can be shown that, under certain regularity conditions, the exponential family is the only family of distributions with finite-sized sufficient statistics, meaning that we can compress the data into a fixed-sized summary without loss of information. This is particularly useful for online learning, as we will see later.
- The exponential family is the only family of distributions for which conjugate priors exist, which simplifies the computation of the posterior (see Section 9.2.5).
- The exponential family can be shown to be the family of distributions that makes the least set of assumptions subject to some user-chosen constraints (see Section 9.2.6).
- The exponential family is at the core of generalized linear models, as discussed in Section 9.3.
- The exponential family is at the core of variational inference, as discussed in Section 21.2.

1. The exceptions are the Student t, which does not have the right form, and the uniform distribution, which does not have fixed support independent of the parameter values.

9.2.1 Definition

A pdf or pmf $p(\mathbf{x}|\boldsymbol{\theta})$, for $\mathbf{x} = (x_1, \ldots, x_m) \in \mathcal{X}^m$ and $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^d$, is said to be in the **exponential family** if it is of the form

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} h(\mathbf{x}) \exp[\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})] \tag{9.1}$$

$$= h(\mathbf{x}) \exp[\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}) - A(\boldsymbol{\theta})] \tag{9.2}$$

where

$$Z(\boldsymbol{\theta}) = \int_{\mathcal{X}^m} h(\mathbf{x}) \exp[\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})] d\mathbf{x} \tag{9.3}$$

$$A(\boldsymbol{\theta}) = \log Z(\boldsymbol{\theta}) \tag{9.4}$$

Here $\boldsymbol{\theta}$ are called the **natural parameters** or **canonical parameters**, $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^d$ is called a vector of **sufficient statistics**, $Z(\boldsymbol{\theta})$ is called the **partition function**, $A(\boldsymbol{\theta})$ is called the **log partition function** or **cumulant function**, and $h(\mathbf{x})$ is the a scaling constant, often 1. If $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$, we say it is a **natural exponential family**.

Equation 9.2 can be generalized by writing

$$p(\mathbf{x}|\boldsymbol{\theta}) = h(\mathbf{x}) \exp[\eta(\boldsymbol{\theta})^T \boldsymbol{\phi}(\mathbf{x}) - A(\eta(\boldsymbol{\theta}))] \tag{9.5}$$

where η is a function that maps the parameters $\boldsymbol{\theta}$ to the canonical parameters $\boldsymbol{\eta} = \eta(\boldsymbol{\theta})$. If $\dim(\boldsymbol{\theta}) < \dim(\eta(\boldsymbol{\theta}))$, it is called a **curved exponential family**, which means we have more sufficient statistics than parameters. If $\eta(\boldsymbol{\theta}) = \boldsymbol{\theta}$, the model is said to be in **canonical form**. We will assume models are in canonical form unless we state otherwise.

9.2.2 Examples

Let us consider some examples to make things clearer.

9.2.2.1 Bernoulli

The Bernoulli for $x \in \{0, 1\}$ can be written in exponential family form as follows:

$$\text{Ber}(x|\mu) = \mu^x (1-\mu)^{1-x} = \exp[x \log(\mu) + (1-x) \log(1-\mu)] = \exp[\boldsymbol{\phi}(x)^T \boldsymbol{\theta}] \tag{9.6}$$

where $\boldsymbol{\phi}(x) = [\mathbb{I}(x=0), \mathbb{I}(x=1)]$ and $\boldsymbol{\theta} = [\log(\mu), \log(1-\mu)]$. However, this representation is **over-complete** since there is a linear dependendence between the features:

$$\mathbf{1}^T \boldsymbol{\phi}(x) = \mathbb{I}(x=0) + \mathbb{I}(x=1) = 1 \tag{9.7}$$

Consequently $\boldsymbol{\theta}$ is not uniquely identifiable. It is common to require that the representation be **minimal**, which means there is a unique $\boldsymbol{\theta}$ associated with the distribution. In this case, we can just define

$$\text{Ber}(x|\mu) = (1-\mu) \exp\left[x \log\left(\frac{\mu}{1-\mu}\right)\right] \tag{9.8}$$

Now we have $\phi(x) = x$, $\theta = \log\left(\frac{\mu}{1-\mu}\right)$, which is the **log-odds ratio**, and $Z = 1/(1-\mu)$. We can recover the mean parameter μ from the canonical parameter using

$$\mu = \text{sigm}(\theta) = \frac{1}{1+e^{-\theta}} \tag{9.9}$$

9.2.2.2 Multinoulli

We can represent the multinoulli as a minimal exponential family as follows (where $x_k = \mathbb{I}(x = k)$):

$$\begin{aligned}
\text{Cat}(x|\boldsymbol{\mu}) &= \prod_{k=1}^{K} \mu_k^{x_k} = \exp\left[\sum_{k=1}^{K} x_k \log \mu_k\right] && (9.10)\\
&= \exp\left[\sum_{k=1}^{K-1} x_k \log \mu_k + \left(1 - \sum_{k=1}^{K-1} x_k\right) \log(1 - \sum_{k=1}^{K-1} \mu_k)\right] && (9.11)\\
&= \exp\left[\sum_{k=1}^{K-1} x_k \log\left(\frac{\mu_k}{1 - \sum_{j=1}^{K-1} \mu_j}\right) + \log(1 - \sum_{k=1}^{K-1} \mu_k)\right] && (9.12)\\
&= \exp\left[\sum_{k=1}^{K-1} x_k \log\left(\frac{\mu_k}{\mu_K}\right) + \log \mu_K\right] && (9.13)
\end{aligned}$$

where $\mu_K = 1 - \sum_{k=1}^{K-1} \mu_k$. We can write this in exponential family form as follows:

$$\begin{aligned}
\text{Cat}(x|\boldsymbol{\theta}) &= \exp(\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}) - A(\boldsymbol{\theta})) && (9.14)\\
\boldsymbol{\theta} &= [\log \frac{\mu_1}{\mu_K}, \ldots, \log \frac{\mu_{K-1}}{\mu_K}] && (9.15)\\
\boldsymbol{\phi}(x) &= [\mathbb{I}(x=1), \ldots, \mathbb{I}(x=K-1)] && (9.16)
\end{aligned}$$

We can recover the mean parameters from the canonical parameters using

$$\mu_k = \frac{e^{\theta_k}}{1 + \sum_{j=1}^{K-1} e^{\theta_j}} \tag{9.17}$$

From this, we find

$$\mu_K = 1 - \frac{\sum_{j=1}^{K-1} e^{\theta_j}}{1 + \sum_{j=1}^{K-1} e^{\theta_j}} = \frac{1}{\sum_{j=1}^{K-1} e^{\theta_j}} \tag{9.18}$$

and hence

$$A(\boldsymbol{\theta}) = \log\left(1 + \sum_{k=1}^{K-1} e^{\theta_k}\right) \tag{9.19}$$

If we define $\theta_K = 0$, we can write $\boldsymbol{\mu} = \mathcal{S}(\boldsymbol{\theta})$ and $A(\boldsymbol{\theta}) = \log \sum_{k=1}^{K} e^{\theta_k}$, where $\mathcal{S}$ is the softmax function in Equation 4.39.

9.2.2.3 Univariate Gaussian

The univariate Gaussian can be written in exponential family form as follows:

$$\begin{aligned}
\mathcal{N}(x|\mu,\sigma^2) &= \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}}\exp[-\frac{1}{2\sigma^2}(x-\mu)^2] &(9.20)\\
&= \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}}\exp[-\frac{1}{2\sigma^2}x^2+\frac{\mu}{\sigma^2}x-\frac{1}{2\sigma^2}\mu^2] &(9.21)\\
&= \frac{1}{Z(\boldsymbol{\theta})}\exp(\boldsymbol{\theta}^T\boldsymbol{\phi}(x)) &(9.22)
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{\theta} &= \begin{pmatrix}\mu/\sigma^2\\ \frac{-1}{2\sigma^2}\end{pmatrix} &(9.23)\\
\boldsymbol{\phi}(x) &= \begin{pmatrix}x\\ x^2\end{pmatrix} &(9.24)\\
Z(\mu,\sigma^2) &= \sqrt{2\pi}\sigma\exp[\frac{\mu^2}{2\sigma^2}] &(9.25)\\
A(\boldsymbol{\theta}) &= \frac{-\theta_1^2}{4\theta_2}-\frac{1}{2}\log(-2\theta_2)-\frac{1}{2}\log(2\pi) &(9.26)
\end{aligned}$$

9.2.2.4 Non-examples

Not all distributions of interest belong to the exponential family. For example, the uniform distribution, $X \sim \text{Unif}(a,b)$, does not, since the support of the distribution depends on the parameters. Also, the Student T distribution (Section 11.4.5) does not belong, since it does not have the required form.

9.2.3 Log partition function

An important property of the exponential family is that derivatives of the log partition function can be used to generate **cumulants** of the sufficient statistics.[2] For this reason, $A(\boldsymbol{\theta})$ is sometimes called a **cumulant function**. We will prove this for a 1-parameter distribution; this can be generalized to a K-parameter distribution in a straightforward way. For the first

2. The first and second cumulants of a distribution are its mean $\mathbb{E}[X]$ and variance $\text{var}[X]$, whereas the first and second moments are its mean $\mathbb{E}[X]$ and $\mathbb{E}[X^2]$.

derivative we have

$$\begin{aligned}
\frac{dA}{d\theta} &= \frac{d}{d\theta}\left(\log \int \exp(\theta\phi(x))h(x)dx\right) && (9.27)\\
&= \frac{\frac{d}{d\theta}\int \exp(\theta\phi(x))h(x)dx}{\int \exp(\theta\phi(x))h(x)dx} && (9.28)\\
&= \frac{\int \phi(x)\exp(\theta\phi(x))h(x)dx}{\exp(A(\theta))} && (9.29)\\
&= \int \phi(x)\exp(\theta\phi(x) - A(\theta))h(x)dx && (9.30)\\
&= \int \phi(x)p(x)dx = \mathbb{E}\left[\phi(x)\right] && (9.31)
\end{aligned}$$

For the second derivative we have

$$\begin{aligned}
\frac{d^2A}{d\theta^2} &= \int \phi(x)\exp\left(\theta\phi(x) - A(\theta)\right)h(x)(\phi(x) - A'(\theta))dx && (9.32)\\
&= \int \phi(x)p(x)(\phi(x) - A'(\theta))dx && (9.33)\\
&= \int \phi^2(x)p(x)dx - A'(\theta)\int \phi(x)p(x)dx && (9.34)\\
&= \mathbb{E}\left[\phi^2(X)\right] - \mathbb{E}\left[\phi(x)\right]^2 = \text{var}\left[\phi(x)\right] && (9.35)
\end{aligned}$$

where we used the fact that $A'(\theta) = \frac{dA}{d\theta} = \mathbb{E}\left[\phi(x)\right]$.

In the multivariate case, we have that

$$\frac{\partial^2 A}{\partial\theta_i\partial\theta_j} = \mathbb{E}\left[\phi_i(x)\phi_j(x)\right] - \mathbb{E}\left[\phi_i(x)\right]\mathbb{E}\left[\phi_j(x)\right] \quad (9.36)$$

and hence

$$\nabla^2 A(\boldsymbol{\theta}) = \text{cov}\left[\boldsymbol{\phi}(\mathbf{x})\right] \quad (9.37)$$

Since the covariance is positive definite, we see that $A(\boldsymbol{\theta})$ is a convex function (see Section 7.3.3).

9.2.3.1 Example: the Bernoulli distribution

For example, consider the Bernoulli distribution. We have $A(\theta) = \log(1 + e^\theta)$, so the mean is given by

$$\frac{dA}{d\theta} = \frac{e^\theta}{1 + e^\theta} = \frac{1}{1 + e^{-\theta}} = \text{sigm}(\theta) = \mu \quad (9.38)$$

The variance is given by

$$\begin{aligned}
\frac{d^2A}{d\theta^2} &= \frac{d}{d\theta}(1 + e^{-\theta})^{-1} = (1 + e^{-\theta})^{-2}.e^{-\theta} && (9.39)\\
&= \frac{e^{-\theta}}{1 + e^{-\theta}}\frac{1}{1 + e^{-\theta}} = \frac{1}{e^\theta + 1}\frac{1}{1 + e^{-\theta}} = (1 - \mu)\mu && (9.40)
\end{aligned}$$

9.2.4 MLE for the exponential family

The likelihood of an exponential family model has the form

$$p(\mathcal{D}|\boldsymbol{\theta}) = \left[\prod_{i=1}^{N} h(\mathbf{x}_i)\right] g(\boldsymbol{\theta})^N \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta})^T [\sum_{i=1}^{N} \boldsymbol{\phi}(\mathbf{x}_i)]\right) \tag{9.41}$$

We see that the sufficient statistics are N and

$$\boldsymbol{\phi}(\mathcal{D}) = [\sum_{i=1}^{N} \phi_1(\mathbf{x}_i), \ldots, \sum_{i=1}^{N} \phi_K(\mathbf{x}_i)] \tag{9.42}$$

For example, for the Bernoulli model we have $\boldsymbol{\phi} = [\sum_i \mathbb{I}(x_i = 1)]$, and for the univariate Gaussian, we have $\boldsymbol{\phi} = [\sum_i x_i, \sum_i x_i^2]$. (We also need to know the sample size, N.)

The **Pitman-Koopman-Darmois theorem** states that, under certain regularity conditions, the exponential family is the only family of distributions with finite sufficient statistics. (Here, finite means of a size independent of the size of the data set.)

One of the conditions required in this theorem is that the support of the distribution not be dependent on the parameter. For a simple example of such a distribution, consider the uniform distribution

$$p(x|\theta) = U(x|\theta) = \frac{1}{\theta}\mathbb{I}(0 \le x \le \theta) \tag{9.43}$$

The likelihood is given by

$$p(\mathcal{D}|\boldsymbol{\theta}) = \theta^{-N}\mathbb{I}(0 \le \max\{x_i\} \le \theta) \tag{9.44}$$

So the sufficient statistics are N and $s(\mathcal{D}) = \max_i x_i$. This is finite in size, but the uniform distribution is not in the exponential family because its support set, $\mathcal{X}$, depends on the parameters.

We now descibe how to compute the MLE for a canonical exponential family model. Given N iid data points $\mathcal{D} = (x_1, \ldots, x_N)$, the log-likelihood is

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathcal{D}) - NA(\boldsymbol{\theta}) \tag{9.45}$$

Since $-A(\boldsymbol{\theta})$ is concave in $\boldsymbol{\theta}$, and $\boldsymbol{\theta}^T\boldsymbol{\phi}(\mathcal{D})$ is linear in $\boldsymbol{\theta}$, we see that the log likelihood is concave, and hence has a unique global maximum. To derive this maximum, we use the fact that the derivative of the log partition function yields the expected value of the sufficient statistic vector (Section 9.2.3):

$$\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) = \boldsymbol{\phi}(\mathcal{D}) - N\mathbb{E}[\boldsymbol{\phi}(\mathbf{X})] \tag{9.46}$$

Setting this gradient to zero, we see that at the MLE, the empirical average of the sufficient statistics must equal the model's theoretical expected sufficient statistics, i.e., $\hat{\boldsymbol{\theta}}$ must satisfy

$$\mathbb{E}[\boldsymbol{\phi}(\mathbf{X})] = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\phi}(\mathbf{x}_i) \tag{9.47}$$

This is called **moment matching**. For example, in the Bernoulli distribution, we have $\phi(X) = \mathbb{I}(X = 1)$, so the MLE satisfies

$$\mathbb{E}[\phi(X)] = p(X = 1) = \hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(x_i = 1) \tag{9.48}$$

9.2.5 Bayes for the exponential family *

We have seen that exact Bayesian analysis is considerably simplified if the prior is conjugate to the likelihood. Informally this means that the prior $p(\boldsymbol{\theta}|\boldsymbol{\tau})$ has the same form as the likelihood $p(\mathcal{D}|\boldsymbol{\theta})$. For this to make sense, we require that the likelihood have finite sufficient statistics, so that we can write $p(\mathcal{D}|\boldsymbol{\theta}) = p(\mathbf{s}(\mathcal{D})|\boldsymbol{\theta})$. This suggests that the only family of distributions for which conjugate priors exist is the exponential family. We will derive the form of the prior and posterior below.

9.2.5.1 Likelihood

The likelihood of the exponential family is given by

$$p(\mathcal{D}|\boldsymbol{\theta}) \propto g(\boldsymbol{\theta})^N \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta})^T \mathbf{s}_N\right) \tag{9.49}$$

where $\mathbf{s}_N = \sum_{i=1}^{N} \mathbf{s}(\mathbf{x}_i)$. In terms of the canonical parameters this becomes

$$p(\mathcal{D}|\boldsymbol{\eta}) \propto \exp(N\boldsymbol{\eta}^T\overline{\mathbf{s}} - NA(\boldsymbol{\eta})) \tag{9.50}$$

where $\overline{\mathbf{s}} = \frac{1}{N}\mathbf{s}_N$.

9.2.5.2 Prior

The natural conjugate prior has the form

$$p(\boldsymbol{\theta}|\nu_0, \boldsymbol{\tau}_0) \propto g(\boldsymbol{\theta})^{\nu_0} \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta})^T \boldsymbol{\tau}_0\right) \tag{9.51}$$

Let us write $\boldsymbol{\tau}_0 = \nu_0\overline{\boldsymbol{\tau}}_0$, to separate out the size of the prior pseudo-data, ν_0, from the mean of the sufficient statistics on this pseudo-data, $\overline{\boldsymbol{\tau}}_0$. In canonical form, the prior becomes

$$p(\boldsymbol{\eta}|\nu_0, \overline{\boldsymbol{\tau}}_0) \propto \exp(\nu_0\boldsymbol{\eta}^T\overline{\boldsymbol{\tau}}_0 - \nu_0 A(\boldsymbol{\eta})) \tag{9.52}$$

9.2.5.3 Posterior

The posterior is given by

$$p(\boldsymbol{\theta}|\mathcal{D}) = p(\boldsymbol{\theta}|\nu_N, \boldsymbol{\tau}_N) = p(\boldsymbol{\theta}|\nu_0 + N, \boldsymbol{\tau}_0 + \mathbf{s}_N) \tag{9.53}$$

So we see that we just update the hyper-parameters by adding. In canonical form, this becomes

$$\begin{aligned} p(\boldsymbol{\eta}|\mathcal{D}) &\propto \exp\left(\boldsymbol{\eta}^T(\nu_0\overline{\boldsymbol{\tau}}_0 + N\overline{\mathbf{s}}) - (\nu_0 + N)A(\boldsymbol{\eta})\right) & (9.54) \\ &= p(\boldsymbol{\eta}|\nu_0 + N, \frac{\nu_0\overline{\boldsymbol{\tau}}_0 + N\overline{\mathbf{s}}}{\nu_0 + N}) & (9.55) \end{aligned}$$

So we see that the posterior hyper-parameters are a convex combination of the prior mean hyper-parameters and the average of the sufficient statistics.

9.2.5.4 Posterior predictive density

Let us derive a generic expression for the predictive density for future observables $\mathcal{D}' = (\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_{N'})$ given past data $\mathcal{D} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ as follows. For notational brevity, we will combine the sufficient statistics with the size of the data, as follows: $\tilde{\boldsymbol{\tau}}_0 = (\nu_0, \boldsymbol{\tau}_0)$, $\tilde{\mathbf{s}}(\mathcal{D}) = (N, \mathbf{s}(\mathcal{D}))$, and $\tilde{\mathbf{s}}(\mathcal{D}') = (N', \mathbf{s}(\mathcal{D}'))$. So the prior becomes

$$p(\boldsymbol{\theta}|\tilde{\boldsymbol{\tau}}_0) = \frac{1}{Z(\tilde{\boldsymbol{\tau}}_0)} g(\boldsymbol{\theta})^{\nu_0} \exp(\boldsymbol{\eta}(\boldsymbol{\theta})^T \boldsymbol{\tau}_0) \tag{9.56}$$

The likelihood and posterior have a similar form. Hence

$$\begin{aligned}
p(\mathcal{D}'|\mathcal{D}) &= \int p(\mathcal{D}'|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} & (9.57)\\
&= \left[\prod_{i=1}^{N'} h(\tilde{\mathbf{x}}_i)\right] Z(\tilde{\boldsymbol{\tau}}_0 + \tilde{\mathbf{s}}(\mathcal{D}))^{-1} \int g(\boldsymbol{\theta})^{\nu_0 + N + N'} d\boldsymbol{\theta} & (9.58)\\
&\quad \times \exp\left(\sum_k \eta_k(\boldsymbol{\theta})(\tau_k + \sum_{i=1}^{N} s_k(\mathbf{x}_i) + \sum_{i=1}^{N'} s_k(\tilde{\mathbf{x}}_i)\right) d\boldsymbol{\theta} & (9.59)\\
&= \left[\prod_{i=1}^{N'} h(\tilde{\mathbf{x}}_i)\right] \frac{Z(\tilde{\boldsymbol{\tau}}_0 + \tilde{\mathbf{s}}(\mathcal{D}) + \tilde{\mathbf{s}}(\mathcal{D}'))}{Z(\tilde{\boldsymbol{\tau}}_0 + \tilde{\mathbf{s}}(\mathcal{D}))} & (9.60)
\end{aligned}$$

If $N = 0$, this becomes the marginal likelihood of $\mathcal{D}'$, which reduces to the familiar form of normalizer of the posterior divided by the normalizer of the prior, multiplied by a constant.

9.2.5.5 Example: Bernoulli distribution

As a simple example, let us revisit the Beta-Bernoulli model in our new notation.

The likelihood is given by

$$p(\mathcal{D}|\theta) = (1-\theta)^N \exp\left(\log(\frac{\theta}{1-\theta}) \sum_i x_i\right) \tag{9.61}$$

Hence the conjugate prior is given by

$$\begin{aligned}
p(\theta|\nu_0, \tau_0) &\propto (1-\theta)^{\nu_0} \exp\left(\log(\frac{\theta}{1-\theta}) \tau_0\right) & (9.62)\\
&= \theta^{\tau_0}(1-\theta)^{\nu_0 - \tau_0} & (9.63)
\end{aligned}$$

If we define $\alpha = \tau_0 + 1$ and $\beta = \nu_0 - \tau_0 + 1$, we see that this is a beta distribution.

We can derive the posterior as follows, where $s = \sum_i \mathbb{I}(x_i = 1)$ is the sufficient statistic:

$$\begin{aligned}
p(\theta|\mathcal{D}) &\propto \theta^{\tau_0 + s}(1-\theta)^{\nu_0 - \tau_0 + n - s} & (9.64)\\
&= \theta^{\tau_n}(1-\theta)^{\nu_n - \tau_n} & (9.65)
\end{aligned}$$

We can derive the posterior predictive distribution as follows. Assume $p(\theta) = \text{Beta}(\theta|\alpha, \beta)$, and let $s = s(\mathcal{D})$ be the number of heads in the past data. We can predict the probability of a given sequence of future heads, $\mathcal{D}' = (\tilde{x}_1, \ldots, \tilde{x}_m)$, with sufficient statistic $s' = \sum_{i=1}^m \mathbb{I}(\tilde{x}_i = 1)$, as follows:

$$\begin{aligned} p(\mathcal{D}'|\mathcal{D}) &= \int_0^1 p(\mathcal{D}'|\theta|\text{Beta}(\theta|\alpha_n, \beta_n)d\theta & (9.66) \\ &= \frac{\Gamma(\alpha_n + \beta_n)}{\Gamma(\alpha_n)\Gamma(\beta_n)} \int_0^1 \theta^{\alpha_n + t' - 1}(1-\theta)^{\beta_n + m - t' - 1} d\theta & (9.67) \\ &= \frac{\Gamma(\alpha_n + \beta_n)}{\Gamma(\alpha_n)\Gamma(\beta_n)} \frac{\Gamma(\alpha_{n+m})\Gamma(\beta_{n+m})}{\Gamma(\alpha_{n+m} + \beta_{n+m})} & (9.68) \end{aligned}$$

where

$$\begin{aligned} \alpha_{n+m} &= \alpha_n + s' = \alpha + s + s' & (9.69) \\ \beta_{n+m} &= \beta_n + (m - s') = \beta + (n - s) + (m - s') & (9.70) \end{aligned}$$

9.2.6 Maximum entropy derivation of the exponential family *

Although the exponential family is convenient, is there any deeper justification for its use? It turns out that there is: it is the distribution that makes the least number of assumptions about the data, subject to a specific set of user-specified constraints, as we explain below. In particular, suppose all we know is the expected values of certain features or functions:

$$\sum_{\mathbf{x}} f_k(\mathbf{x})p(\mathbf{x}) = F_k \tag{9.71}$$

where F_k are known constants, and $f_k(\mathbf{x})$ is an arbitrary function. The principle of **maximum entropy** or **maxent** says we should pick the distribution with maximum entropy (closest to uniform), subject to the constraints that the moments of the distribution match the empirical moments of the specified functions.

To maximize entropy subject to the constraints in Equation 9.71, and the constraints that $p(\mathbf{x}) \geq 0$ and $\sum_{\mathbf{x}} p(\mathbf{x}) = 1$, we need to use Lagrange multipliers. The Lagrangian is given by

$$J(p, \boldsymbol{\lambda}) = -\sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x}) + \lambda_0(1 - \sum_{\mathbf{x}} p(\mathbf{x})) + \sum_k \lambda_k (F_k - \sum_{\mathbf{x}} p(\mathbf{x}) f_k(\mathbf{x})) \tag{9.72}$$

We can use the calculus of variations to take derivatives wrt the function p, but we will adopt a simpler approach and treat p as a fixed length vector (since we are assuming $\mathbf{x}$ is discrete). Then we have

$$\frac{\partial J}{\partial p(\mathbf{x})} = -1 - \log p(\mathbf{x}) - \lambda_0 - \sum_k \lambda_k f_k(\mathbf{x}) \tag{9.73}$$

Setting $\frac{\partial J}{\partial p(\mathbf{x})} = 0$ yields

$$p(\mathbf{x}) = \frac{1}{Z} \exp(-\sum_k \lambda_k f_k(\mathbf{x})) \tag{9.74}$$

$$\begin{tikzcd}
\boldsymbol{w} \arrow[dr] & & & \\
& \eta_i \arrow[r, shift left, "g^{-1}"] & \mu_i \arrow[l, shift left, "g"] \arrow[r, shift left, "\Psi"] & \theta_i \arrow[l, shift left, "\Psi^{-1}"] \\
\boldsymbol{x}_i \arrow[ur] & & &
\end{tikzcd}$$

Figure 9.1 A visualization of the various features of a GLM. Based on Figure 8.3 of (Jordan 2007).

where $Z = e^{1+\lambda_0}$. Using the sum to one constraint, we have

$$1 = \sum_{\mathbf{x}} p(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{x}} \exp(-\sum_k \lambda_k f_k(\mathbf{x})) \tag{9.75}$$

Hence the normalization constant is given by

$$Z = \sum_{\mathbf{x}} \exp(-\sum_k \lambda_k f_k(\mathbf{x})) \tag{9.76}$$

Thus the maxent distribution $p(\mathbf{x})$ has the form of the exponential family (Section 9.2), also known as the **Gibbs distribution**.

9.3 Generalized linear models (GLMs)

Linear and logistic regression are examples of **generalized linear model**s, or **GLM**s (McCullagh and Nelder 1989). These are models in which the output density is in the exponential family (Section 9.2), and in which the mean parameters are a linear combination of the inputs, passed through a possibly nonlinear function, such as the logistic function. We describe GLMs in more detail below. We focus on scalar outputs for notational simplicity. (This excludes multinomial logistic regression, but this is just to simplify the presentation.)

9.3.1 Basics

To understand GLMs, let us first consider the case of an unconditional dstribution for a scalar response variable:

$$p(y_i|\theta, \sigma^2) = \exp\left[\frac{y_i\theta - A(\theta)}{\sigma^2} + c(y_i, \sigma^2)\right] \tag{9.77}$$

where σ^2 is the **dispersion parameter** (often set to 1), θ is the natural parameter, A is the partition function, and c is a normalization constant. For example, in the case of logistic regression, θ is the log-odds ratio, $\theta = \log(\frac{\mu}{1-\mu})$, where $\mu = \mathbb{E}[y] = p(y = 1)$ is the mean parameter (see Section 9.2.2.1). To convert from the mean parameter to the natural parameter

Distrib.	Link $g(\mu)$	$\theta = \psi(\mu)$	$\mu = \psi^{-1}(\theta) = \mathbb{E}[y]$
$\mathcal{N}(\mu, \sigma^2)$	identity	$\theta = \mu$	$\mu = \theta$
$\text{Bin}(N, \mu)$	logit	$\theta = \log(\frac{\mu}{1-\mu})$	$\mu = \text{sigm}(\theta)$
$\text{Poi}(\mu)$	log	$\theta = \log(\mu)$	$\mu = e^\theta$

Table 9.1 Canonical link functions ψ and their inverses for some common GLMs.

we can use a function ψ, so $\theta = \Psi(\mu)$. This function is uniquely determined by the form of the exponential family distribution. In fact, this is an invertible mapping, so we have $\mu = \Psi^{-1}(\theta)$. Furthermore, we know from Section 9.2.3 that the mean is given by the derivative of the partition function, so we have $\mu = \Psi^{-1}(\theta) = A'(\theta)$.

Now let us add inputs/ covariates. We first define a linear function of the inputs:

$$\eta_i = \mathbf{w}^T \mathbf{x}_i \tag{9.78}$$

We now make the mean of the distribution be some invertible monotonic function of this linear combination. By convention, this function, known as the **mean function**, is denoted by g^{-1}, so

$$\mu_i = g^{-1}(\eta_i) = g^{-1}(\mathbf{w}^T \mathbf{x}_i) \tag{9.79}$$

See Figure 9.1 for a summary of the basic model.

The inverse of the mean function, namely $g()$, is called the **link function**. We are free to choose almost any function we like for g, so long as it is invertible, and so long as g^{-1} has the appropriate range. For example, in logistic regression, we set $\mu_i = g^{-1}(\eta_i) = \text{sigm}(\eta_i)$.

One particularly simple form of link function is to use $g = \psi$; this is called the **canonical link function**. In this case, $\theta_i = \eta_i = \mathbf{w}^T \mathbf{x}_i$, so the model becomes

$$p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) = \exp\left[\frac{y_i \mathbf{w}^T \mathbf{x}_i - A(\mathbf{w}^T \mathbf{x}_i)}{\sigma^2} + c(y_i, \sigma^2)\right] \tag{9.80}$$

In Table 9.1, we list some distributions and their canonical link functions. We see that for the Bernoulli/ binomial distribution, the canonical link is the logit function, $g(\mu) = \log(\eta/(1-\eta))$, whose inverse is the logistic function, $\mu = \text{sigm}(\eta)$.

Based on the results in Section 9.2.3, we can show that the mean and variance of the response variable are as follows:

$$\mathbb{E}\left[y|\mathbf{x}_i, \mathbf{w}, \sigma^2\right] = \mu_i = A'(\theta_i) \tag{9.81}$$

$$\text{var}\left[y|\mathbf{x}_i, \mathbf{w}, \sigma^2\right] = \sigma_i^2 = A''(\theta_i)\sigma^2 \tag{9.82}$$

To make the notation clearer, let us consider some simple examples.

- For linear regression, we have

$$\log p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) = \frac{y_i \mu_i - \frac{\mu_i^2}{2}}{\sigma^2} - \frac{1}{2}\left(\frac{y_i^2}{\sigma^2} + \log(2\pi\sigma^2)\right) \tag{9.83}$$

  where $y_i \in \mathbb{R}$, and $\theta_i = \mu_i = \mathbf{w}^T \mathbf{x}_i$ Here $A(\theta) = \theta^2/2$, so $\mathbb{E}[y_i] = \mu_i$ and $\text{var}[y_i] = \sigma^2$.

- For binomial regression, we have

$$\log p(y_i|\mathbf{x}_i, \mathbf{w}) = y_i \log(\frac{\pi_i}{1-\pi_i}) + N_i \log(1-\pi_i) + \log \binom{N_i}{y_i} \tag{9.84}$$

  where $y_i \in \{0, 1, \ldots, N_i\}$, $\pi_i = \text{sigm}(\mathbf{w}^T \mathbf{x}_i)$, $\theta_i = \log(\pi_i/(1-\pi_i)) = \mathbf{w}^T\mathbf{x}_i$, and $\sigma^2 = 1$. Here $A(\theta) = N_i \log(1+e^\theta)$, so $\mathbb{E}[y_i] = N_i\pi_i = \mu_i$, $\text{var}[y_i] = N_i\pi_i(1-\pi_i)$.

- For **poisson regression**, we have

$$\log p(y_i|\mathbf{x}_i, \mathbf{w}) = y_i \log \mu_i - \mu_i - \log(y_i!) \tag{9.85}$$

  where $y_i \in \{0, 1, 2, \ldots\}$, $\mu_i = \exp(\mathbf{w}^T\mathbf{x}_i)$, $\theta_i = \log(\mu_i) = \mathbf{w}^T\mathbf{x}_i$, and $\sigma^2 = 1$. Here $A(\theta) = e^\theta$, so $\mathbb{E}[y_i] = \text{var}[y_i] = \mu_i$. Poisson regression is widely used in bio-statistical applications, where y_i might represent the number of diseases of a given person or place, or the number of reads at a genomic location in a high-throughput sequencing context (see e.g., (Kuan et al. 2009)).

9.3.2 ML and MAP estimation

One of the appealing properties of GLMs is that they can be fit using exactly the same methods that we used to fit logistic regression. In particular, the log-likelihood has the following form:

$$\ell(\mathbf{w}) = \log p(\mathcal{D}|\mathbf{w}) = \frac{1}{\sigma^2}\sum_{i=1}^N \ell_i \tag{9.86}$$

$$\ell_i \triangleq \theta_i y_i - A(\theta_i) \tag{9.87}$$

We can compute the gradient vector using the chain rule as follows:

$$\frac{d\ell_i}{dw_j} = \frac{d\ell_i}{d\theta_i}\frac{d\theta_i}{d\mu_i}\frac{d\mu_i}{d\eta_i}\frac{d\eta_i}{dw_j} \tag{9.88}$$

$$= (y_i - A'(\theta_i))\frac{d\theta_i}{d\mu_i}\frac{d\mu_i}{d\eta_i}x_{ij} \tag{9.89}$$

$$= (y_i - \mu_i)\frac{d\theta_i}{d\mu_i}\frac{d\mu_i}{d\eta_i}x_{ij} \tag{9.90}$$

If we use a canonical link, $\theta_i = \eta_i$, this simplifies to

$$\nabla_{\mathbf{w}}\ell(\mathbf{w}) = \frac{1}{\sigma^2}\left[\sum_{i=1}^N (y_i - \mu_i)\mathbf{x}_i\right] \tag{9.91}$$

which is a sum of the input vectors, weighted by the errors. This can be used inside a (stochastic) gradient descent procedure, discussed in Section 8.5.2. However, for improved efficiency, we should use a second-order method. If we use a canonical link, the Hessian is given by

$$\mathbf{H} = -\frac{1}{\sigma^2}\sum_{i=1}^N \frac{d\mu_i}{d\theta_i}\mathbf{x}_i\mathbf{x}_i^T = -\frac{1}{\sigma^2}\mathbf{X}^T\mathbf{S}\mathbf{X} \tag{9.92}$$

Name	Formula
Logistic	$g^{-1}(\eta) = \text{sigm}(\eta) = \frac{e^{\eta}}{1+e^{\eta}}$
Probit	$g^{-1}(\eta) = \Phi(\eta)$
Log-log	$g^{-1}(\eta) = \exp(-\exp(-\eta))$
Complementary log-log	$g^{-1}(\eta) = 1 - \exp(-\exp(\eta))$

Table 9.2 Summary of some possible mean functions for binary regression.

where $\mathbf{S} = \text{diag}(\frac{d\mu_1}{d\theta_1}, \ldots, \frac{d\mu_N}{d\theta_N})$ is a diagonal weighting matrix. This can be used inside the IRLS algorithm (Section 8.3.4). Specifically, we have the following Newton update:

$$\mathbf{w}_{t+1} = (\mathbf{X}^T\mathbf{S}_t\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S}_t\mathbf{z}_t \tag{9.93}$$

$$\mathbf{z}_t = \boldsymbol{\theta}_t + \mathbf{S}_t^{-1}(\mathbf{y} - \boldsymbol{\mu}_t) \tag{9.94}$$

where $\boldsymbol{\theta}_t = \mathbf{X}\mathbf{w}_t$ and $\boldsymbol{\mu}_t = g^{-1}(\boldsymbol{\eta}_t)$.

If we extend the derivation to handle non-canonical links, we find that the Hessian has another term. However, it turns out that the expected Hessian is the same as in Equation 9.92; using the expected Hessian (known as the Fisher information matrix) instead of the actual Hessian is known as the **Fisher scoring method**.

It is straightforward to modify the above procedure to perform MAP estimation with a Gaussian prior: we just modify the objective, gradient and Hessian, just as we added ℓ_2 regularization to logistic regression in Section 8.3.6.

9.3.3 Bayesian inference

Bayesian inference for GLMs is usually conducted using MCMC (Chapter 24). Possible methods include Metropolis Hastings with an IRLS-based proposal (Gamerman 1997), Gibbs sampling using adaptive rejection sampling (ARS) for each full-conditional (Dellaportas and Smith 1993), etc. See e.g., (Dey et al. 2000) for futher information. It is also possible to use the Gaussian approximation (Section 8.4.1) or variational inference (Section 21.8.1.1).

9.4 Probit regression

In (binary) logistic regression, we use a model of the form $p(y = 1|\mathbf{x}_i, \mathbf{w}) = \text{sigm}(\mathbf{w}^T\mathbf{x}_i)$. In general, we can write $p(y = 1|\mathbf{x}_i, \mathbf{w}) = g^{-1}(\mathbf{w}^T\mathbf{x}_i)$, for any function g^{-1} that maps $[-\infty, \infty]$ to $[0, 1]$. Several possible mean functions are listed in Table 9.2.

In this section, we focus on the case where $g^{-1}(\eta) = \Phi(\eta)$, where $\Phi(\eta)$ is the cdf of the standard normal. This is known as **probit regression**. The probit function is very similar to the logistic function, as shown in Figure 8.7(b). However, this model has some advantages over logistic regression, as we will see.

9.4.1 ML/MAP estimation using gradient-based optimization

We can find the MLE for probit regression using standard gradient methods. Let $\mu_i = \mathbf{w}^T\mathbf{x}_i$, and let $\tilde{y}_i \in \{-1, +1\}$. Then the gradient of the log-likelihod for a specific case is given by

$$\mathbf{g}_i \triangleq \frac{d}{d\mathbf{w}} \log p(\tilde{y}_i|\mathbf{w}^T\mathbf{x}_i) = \frac{d\mu_i}{d\mathbf{w}} \frac{d}{d\mu_i} \log p(\tilde{y}_i|\mathbf{w}^T\mathbf{x}_i) = \mathbf{x}_i \frac{\tilde{y}_i\phi(\mu_i)}{\Phi(\tilde{y}_i\mu_i)} \tag{9.95}$$

where ϕ is the standard normal pdf, and Φ is its cdf. Similarly, the Hessian for a single case is given by

$$\mathbf{H}_i = \frac{d}{d\mathbf{w}^2} \log p(\tilde{y}_i|\mathbf{w}^T\mathbf{x}_i) = -\mathbf{x}_i \left(\frac{\phi(\mu_i)^2}{\Phi(\tilde{y}_i\mu_i)^2} + \frac{\tilde{y}_i\mu_i\phi(\mu_i)}{\Phi(\tilde{y}_i\mu_i)} \right) \mathbf{x}_i^T \tag{9.96}$$

We can modify these expressions to compute the MAP estimate in a straightforward manner. In particular, if we use the prior $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \mathbf{V}_0)$, the gradient and Hessian of the penalized log likelihood have the form $\sum_i \mathbf{g}_i + 2\mathbf{V}_0^{-1}\mathbf{w}$ and $\sum_i \mathbf{H}_i + 2\mathbf{V}_0^{-1}$. These expressions can be passed to any gradient-based optimizer. See `probitRegDemo` for a demo.

9.4.2 Latent variable interpretation

We can interpret the probit (and logistic) model as follows. First, let us associate each item $\mathbf{x}_i$ with two latent utilities, u_{0i} and u_{1i}, corresponding to the possible choices of $y_i = 0$ and $y_i = 1$. We then assume that the observed choice is whichever action has larger utility. More precisely, the model is as follows:

$$u_{0i} \triangleq \mathbf{w}_0^T\mathbf{x}_i + \delta_{0i} \tag{9.97}$$
$$u_{1i} \triangleq \mathbf{w}_1^T\mathbf{x}_i + \delta_{1i} \tag{9.98}$$
$$y_i = \mathbb{I}(u_{1i} > u_{10}) \tag{9.99}$$

where δ's are error terms, representing all the other factors that might be relevant in decision making that we have chosen not to (or are unable to) model. This is called a **random utility model** or **RUM** (McFadden 1974; Train 2009).

Since it is only the difference in utilities that matters, let us define $z_i = u_{1i} - u_{0i} + \epsilon_i$, where $\epsilon_i = \delta_{1i} - \delta_{0i}$. If the δ's have a Gaussian distribution, then so does ϵ_i. Thus we can write

$$z_i \triangleq \mathbf{w}^T\mathbf{x}_i + \epsilon_i \tag{9.100}$$
$$\epsilon_i \sim \mathcal{N}(0, 1) \tag{9.101}$$
$$y_i = 1 = \mathbb{I}(z_i \geq 0) \tag{9.102}$$

Following (Fruhwirth-Schnatter and Fruhwirth 2010), we call this the difference RUM or **dRUM** model.

When we marginalize out z_i, we recover the probit model:

$$p(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \int \mathbb{I}(z_i \geq 0)\mathcal{N}(z_i|\mathbf{w}^T\mathbf{x}_i, 1)dz_i \tag{9.103}$$
$$= p(\mathbf{w}^T\mathbf{x}_i + \epsilon \geq 0) = p(\epsilon \geq -\mathbf{w}^T\mathbf{x}_i) \tag{9.104}$$
$$= 1 - \Phi(-\mathbf{w}^T\mathbf{x}_i) = \Phi(\mathbf{w}^T\mathbf{x}_i) \tag{9.105}$$

where we used the symmetry of the Gaussian.[3] This latent variable interpretation provides an alternative way to fit the model, as discussed in Section 11.4.6.

Interestingly, if we use a Gumbel distribution for the δ's, we induce a logistic distibution for ϵ_i, and the model reduces to logistic regression. See Section 24.5.1 for further details.

9.4.3 Ordinal probit regression *

One advantage of the latent variable interpretation of probit regression is that it is easy to extend to the case where the response variable is ordinal, that is, it can take on C discrete values which can be ordered in some way, such as low, medium and high. This is called **ordinal regression**. The basic idea is as follows. We introduce $C+1$ thresholds γ_j and set

$$y_i = j \quad \text{if} \quad \gamma_{j-1} < z_i \leq \gamma_j \tag{9.106}$$

where $\gamma_0 \leq \cdots \leq \gamma_C$. For identifiability reasons, we set $\gamma_0 = -\infty$, $\gamma_1 = 0$ and $\gamma_C = \infty$. For example, if $C = 2$, this reduces to the standard binary probit model, whereby $z_i < 0$ produces $y_i = 0$ and $z_i \geq 0$ produces $y_i = 1$. If $C = 3$, we partition the real line into 3 intervals: $(-\infty, 0]$, $(0, \gamma_2]$, (γ_2, ∞). We can vary the parameter γ_2 to ensure the right relative amount of probability mass falls in each interval, so as to match the empirical frequencies of each class label.

Finding the MLEs for this model is a bit trickier than for binary probit regression, since we need to optimize for $\mathbf{w}$ and $\boldsymbol{\gamma}$, and the latter must obey an ordering constraint. See e.g., (Kawakatsu and Largey 2009) for an approach based on EM. It is also possible to derive a simple Gibbs sampling algorithm for this model (see e.g., (Hoff 2009, p216)).

9.4.4 Multinomial probit models *

Now consider the case where the response variable can take on C unordered categorical values, $y_i \in \{1, \ldots, C\}$. The **multinomial probit** model is defined as follows:

$$z_{ic} = \mathbf{w}^T \mathbf{x}_{ic} + \epsilon_{ic} \tag{9.107}$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \tag{9.108}$$

$$y_i = \arg\max_c z_{ic} \tag{9.109}$$

See e.g., (Dow and Endersby 2004; Scott 2009; Fruhwirth-Schnatter and Fruhwirth 2010) for more details on the model and its connection to multinomial logistic regression. (By defining $\mathbf{w} = [\mathbf{w}_1, \ldots, \mathbf{w}_C]$, and $\mathbf{x}_{ic} = [\mathbf{0}, \ldots, \mathbf{0}, \mathbf{x}_i, \mathbf{0}, \ldots, \mathbf{0}]$, we can recover the more familiar formulation $z_{ic} = \mathbf{x}_i^T \mathbf{w}_c$.) Since only relative utilities matter, we constrain $\mathbf{R}$ to be a correlation matrix. If instead of setting $y_i = \text{argmax}_c\, z_{ic}$ we use $y_{ic} = \mathbb{I}(z_{ic} > 0)$, we get a model known as **multivariate probit**, which is one way to model C correlated binary outcomes (see e.g., (Talhouk et al. 2011)).

3. Note that the assumption that the Gaussian noise term is zero mean and unit variance is made without loss of generality. To see why, suppose we used some other mean μ and variance σ^2. Then we could easily rescale $\mathbf{w}$ and add an offset term without changing the likelihood. since $P(\mathcal{N}(0,1) \geq -\mathbf{w}^T\mathbf{x}) = P(\mathcal{N}(\mu, \sigma^2) \geq -(\mathbf{w}^T\mathbf{x} + \mu)/\sigma)$.

9.5 Multi-task learning

Sometimes we want to fit many related classification or regression models. It is often reasonable to assume the input-output mapping is similar across these different models, so we can get better performance by fitting all the parameters at the same time. In machine learning, this setup is often called **multi-task learning** (Caruana 1998), **transfer learning** (e.g., (Raina et al. 2005)), or **learning to learn** (Thrun and Pratt 1997). In statistics, this is usually tackled using hierarchical Bayesian models (Bakker and Heskes 2003), as we discuss below, although there are other possible methods (see e.g., (Chai 2010)).

9.5.1 Hierarchical Bayes for multi-task learning

Let y_{ij} be the response of the i'th item in group j, for $i = 1 : N_j$ and $j = 1 : J$. For example, j might index schools, i might index students within a school, and y_{ij} might be the test score, as in Section 5.6.2. Or j might index people, and i might index purchaes, and y_{ij} might be the identity of the item that was purchased (this is known as **discrete choice modeling** (Train 2009)). Let $\mathbf{x}_{ij}$ be a feature vector associated with y_{ij}. The goal is to fit the models $p(y_j|\mathbf{x}_j)$ for all j.

Although some groups may have lots of data, there is often a long tail, where the majority of groups have little data. Thus we can't reliably fit each model separately, but we don't want to use the same model for all groups. As a compromise, we can fit a separate model for each group, but encourage the model parameters to be similar across groups. More precisely, suppose $\mathbb{E}[y_{ij}|\mathbf{x}_{ij}] = g(\mathbf{x}_{ij}^T\boldsymbol{\beta}_j)$, where g is the link function for the GLM. Furthermore, suppose $\boldsymbol{\beta}_j \sim \mathcal{N}(\boldsymbol{\beta}_*, \sigma_j^2\mathbf{I})$, and that $\boldsymbol{\beta}_* \sim \mathcal{N}(\boldsymbol{\mu}, \sigma_*^2\mathbf{I})$. In this model, groups with small sample size borrow statistical strength from the groups with larger sample size, because the $\boldsymbol{\beta}_j$'s are correlated via the latent common parents $\boldsymbol{\beta}_*$ (see Section 5.5 for further discussion of this point). The term σ_j^2 controls how much group j depends on the common parents and the σ_*^2 term controls the strength of the overall prior.

Suppose, for simplicity, that $\boldsymbol{\mu} = \mathbf{0}$, and that σ_j^2 and σ_*^2 are all known (e.g., they could be set by cross validation). The overall log probability has the form

$$\log p(\mathcal{D}|\boldsymbol{\beta}) + \log p(\boldsymbol{\beta}) = \sum_j \left[\log p(\mathcal{D}_j|\boldsymbol{\beta}_j) - \frac{||\boldsymbol{\beta}_j - \boldsymbol{\beta}_*||^2}{2\sigma_j^2}\right] - \frac{||\boldsymbol{\beta}_*||^2}{2\sigma_*^2} \tag{9.110}$$

We can perform MAP estimation of $\boldsymbol{\beta} = (\boldsymbol{\beta}_{1:J}, \boldsymbol{\beta}_*)$ using standard gradient methods. Alternatively, we can perform an iterative optimization scheme, alternating between optimizing the $\boldsymbol{\beta}_j$ and the $\boldsymbol{\beta}_*$; since the likelihood and prior are convex, this is guaranteed to converge to the global optimum. Note that once the models are trained, we can discard $\boldsymbol{\beta}_*$, and use each model separately.

9.5.2 Application to personalized email spam filtering

An interesting application of multi-task learning is **personalized spam filtering**. Suppose we want to fit one classifier per user, $\boldsymbol{\beta}_j$. Since most users do not label their email as spam or not, it will be hard to estimate these models independently. So we will let the $\boldsymbol{\beta}_j$ have a common prior $\boldsymbol{\beta}_*$, representing the parameters of a generic user.

In this case, we can emulate the behavior of the above model with a simple trick (Daume 2007b; Attenberg et al. 2009; Weinberger et al. 2009): we make two copies of each feature $\mathbf{x}_i$, one concatenated with the user id, and one not. The effect will be to learn a predictor of the form

$$\mathbb{E}\left[y_i|\mathbf{x}_i, u\right] = (\boldsymbol{\beta}_*, \mathbf{w}_1, \cdots, \mathbf{w}_J)^T [\mathbf{x}_i, \mathbb{I}(u=1)\mathbf{x}_i, \cdots, \mathbb{I}(u=J)\mathbf{x}_i] \tag{9.111}$$

where u is the user id. In other words,

$$\mathbb{E}\left[y_i|\mathbf{x}_i, u=j\right] = (\boldsymbol{\beta}_*^T + \mathbf{w}_j)^T \mathbf{x}_i \tag{9.112}$$

Thus $\boldsymbol{\beta}_*$ will be estimated from everyone's email, whereas $\mathbf{w}_j$ will just be estimated from user j's email.

To see the correspondence with the above hierarchical Bayesian model, define $\mathbf{w}_j = \boldsymbol{\beta}_j - \boldsymbol{\beta}_*$. Then the log probability of the original model can be rewritten as

$$\sum_j \left[\log p(\mathcal{D}_j|\boldsymbol{\beta}_* + \mathbf{w}_j) - \frac{||\mathbf{w}_j||^2}{2\sigma_j^2}\right] - \frac{||\boldsymbol{\beta}_*||^2}{2\sigma_*^2} \tag{9.113}$$

If we assume $\sigma_j^2 = \sigma_*^2$, the effect is the same as using the augmented feature trick, with the same regularizer strength for both $\mathbf{w}_j$ and $\boldsymbol{\beta}_*$. However, one typically gets better performance by not requiring that σ_j^2 be equal to σ_*^2 (Finkel and Manning 2009).

9.5.3 Application to domain adaptation

Domain adaptation is the problem of training a set of classifiers on data drawn from different distributions, such as email and newswire text. This problem is obviously a special case of multi-task learning, where the tasks are the same.

(Finkel and Manning 2009) used the above hierarchical Bayesian model to perform domain adaptation for two NLP tasks, namely named entity recognition and parsing. They report reasonably large improvements over fitting separate models to each dataset, and small improvements over the approach of pooling all the data and fitting a single model.

9.5.4 Other kinds of prior

In multi-task learning, it is common to assume that the prior is Gaussian. However, sometimes other priors are more suitable. For example, consider the task of **conjoint analysis**, which requires figuring out which features of a product customers like best. This can be modelled using the same hierarchical Bayesian setup as above, but where we use a sparsity-promoting prior on $\boldsymbol{\beta}_j$, rather than a Gaussian prior. This is called **multi-task feature selection**. See e.g., (Lenk et al. 1996; Argyriou et al. 2008) for some possible approaches.

It is not always reasonable to assume that all tasks are all equally similar. If we pool the parameters across tasks that are qualitatively different, the performance will be worse than not using pooling, because the inductive bias of our prior is wrong. Indeed, it has been found experimentally that sometimes multi-task learning does worse than solving each task separately (this is called **negative transfer**).

One way around this problem is to use a more flexible prior, such as a mixture of Gaussians. Such flexible priors can provide robustness against prior mis-specification. See e.g., (Xue et al. 2007; Jacob et al. 2008) for details. One can of course combine mixtures with sparsity-promoting priors (Ji et al. 2009). Many other variants are possible.

9.6 Generalized linear mixed models *

Suppose we generalize the multi-task learning scenario to allow the response to include information at the group level, $\mathbf{x}_j$, as well as at the item level, $\mathbf{x}_{ij}$. Similarly, we can allow the parameters to vary across groups, $\boldsymbol{\beta}_j$, or to be tied across groups, $\boldsymbol{\alpha}$. This gives rise to the following model:

$$\mathbb{E}\left[y_{ij}|\mathbf{x}_{ij}, \mathbf{x}_j\right] = g\left(\boldsymbol{\phi}_1(\mathbf{x}_{ij})^T\boldsymbol{\beta}_j + \boldsymbol{\phi}_2(\mathbf{x}_j)^T\boldsymbol{\beta}_j' + \boldsymbol{\phi}_3(\mathbf{x}_{ij})^T\boldsymbol{\alpha} + \boldsymbol{\phi}_4(\mathbf{x}_j)^T\boldsymbol{\alpha}'\right) \tag{9.114}$$

where the $\boldsymbol{\phi}_k$ are basis functions. This model can be represented pictorially as shown in Figure 9.2(a). (Such figures will be explained in Chapter 10.) Note that the number of $\boldsymbol{\beta}_j$ parameters grows with the number of groups, whereas the size of $\boldsymbol{\alpha}$ is fixed.

Frequentists call the terms $\boldsymbol{\beta}_j$ **random effects**, since they vary randomly across groups, but they call $\boldsymbol{\alpha}$ a **fixed effect**, since it is viewed as a fixed but unknown constant. A model with both fixed and random effects is called a **mixed model**. If $p(y|\mathbf{x})$ is a GLM, the overall model is called a **generalized linear mixed effects model** or **GLMM**. Such models are widely used in statistics.

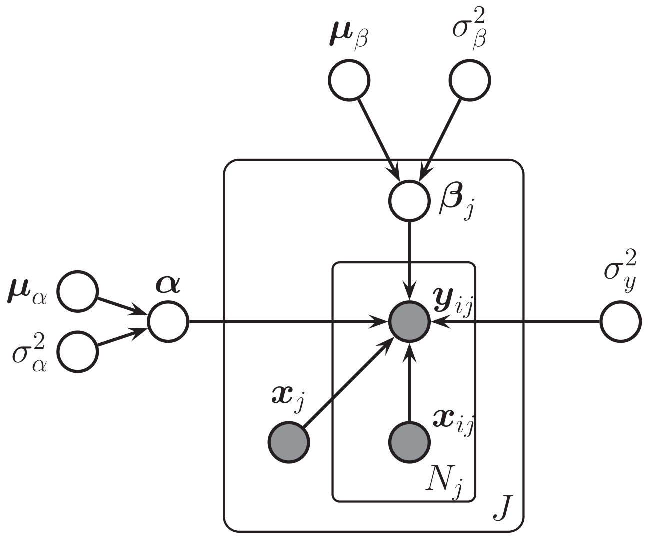
9.6.1 Example: semi-parametric GLMMs for medical data

Consider the following example from (Wand 2009). Suppose y_{ij} is the amount of spinal bone mineral density (SBMD) for person j at measurement i. Let x_{ij} be the age of person, and let x_j be their ethnicity, which can be one of: White, Asian, Black, or Hispanic. The primary goal is to determine if there are significant differences in the mean SBMD among the four ethnic groups, after accounting for age. The data is shown in the light gray lines in Figure 9.2(b). We see that there is a nonlinear effect of SBMD vs age, so we will use a **semi-parametric model** which combines linear regression with non-parametric regression (Ruppert et al. 2003). We also see that there is variation across individuals within each group, so we will use a mixed effects model. Specifically, we will use $\boldsymbol{\phi}_1(\mathbf{x}_{ij}) = 1$ to account for the random effect of each person; $\boldsymbol{\phi}_2(x_{ij}) = 0$ since no other coefficients are person-specific; $\boldsymbol{\phi}_3(x_{ij}) = [b_k(x_{ij})]$, where b_k is the k'th spline basis functions (see Section 15.4.6.2), to account for the nonlinear effect of age; and $\boldsymbol{\phi}_4(x_j) = [\mathbb{I}(x_j = w), \mathbb{I}(x_j = a), \mathbb{I}(x_j = b), \mathbb{I}(x_j = h)]$ to account for the effect of the different ethnicities. Furthermore, we use a linear link function. The overall model is therefore

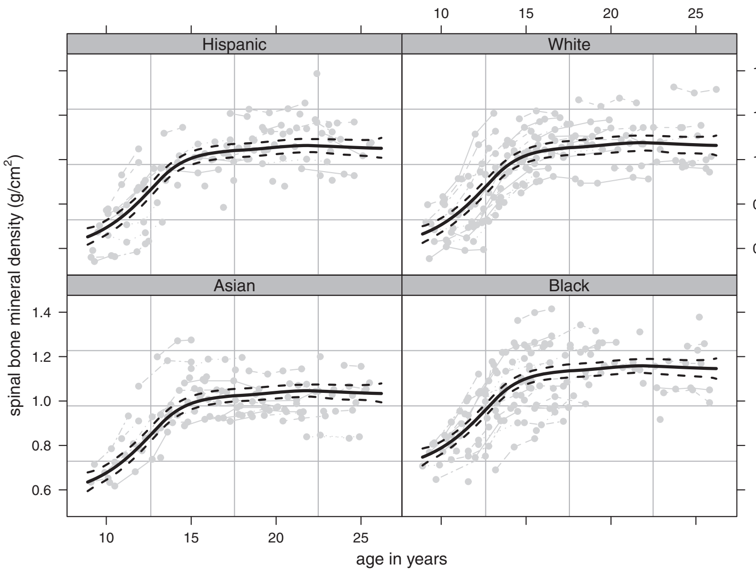
$$\mathbb{E}\left[y_{ij}|x_{ij}, x_j\right] = \beta_j + \boldsymbol{\alpha}^T\mathbf{b}(x_{ij}) + \epsilon_{ij} \tag{9.115}$$

$$+\alpha_w'\mathbb{I}(x_j = w) + \alpha_a'\mathbb{I}(x_j = a) + \alpha_b'\mathbb{I}(x_j = b) + \alpha_h'\mathbb{I}(x_j = h) \tag{9.116}$$

where $\epsilon_{ij} \sim \mathcal{N}(0, \sigma_y^2)$. $\boldsymbol{\alpha}$ contains the non-parametric part of the model related to age, $\boldsymbol{\alpha}'$ contains the parametric part of the model related to ethnicity, and β_j is a random offset for person j. We endow all of these regression coefficients with separate Gaussian priors. We can then perform posterior inference to compute $p(\boldsymbol{\alpha}, \boldsymbol{\alpha}', \boldsymbol{\beta}, \boldsymbol{\sigma}^2|\mathcal{D})$ (see Section 9.6.2 for



(a)



(b)

Figure 9.2 (a) Directed graphical model for generalized linear mixed effects model with J groups. (b) Spinal bone mineral density vs age for four different ethnic groups. Raw data is shown in the light gray lines. Fitted model shown in black (solid is the posterior predicted mean, dotted is the posterior predictive variance). From Figure 9 of (Wand 2009). Used with kind permission of Matt Wand computational details). After fitting the model, we can compute the prediction for each group. See Figure 9.2(b) for the results. We can also perform significance testing, by computing $p(\alpha_g - \alpha_w|\mathcal{D})$ for each ethnic group g relative to some baseline (say, White), as we did in Section 5.2.3.

9.6.2 Computational issues

The principle problem with GLMMs is that they can be difficult to fit, for two reasons. First, $p(y_{ij}|\boldsymbol{\theta})$ may not be conjugate to the prior $p(\boldsymbol{\theta})$ where $\boldsymbol{\theta} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$. Second, there are two levels of unknowns in the model, namely the regression coefficients $\boldsymbol{\theta}$ and the means and variances of the priors $\boldsymbol{\eta} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$.

One approach is to adopt fully Bayesian inference methods, such as variational Bayes (Hall et al. 2011) or MCMC (Gelman and Hill 2007). We discuss VB in Section 21.5, and MCMC in Section 24.1.

An alternative approach is to use empirical Bayes, which we discuss in general terms in Section 5.6. In the context of a GLMM, we can use the EM algorithm (Section 11.4), where in the E step we compute $p(\boldsymbol{\theta}|\boldsymbol{\eta}, \mathcal{D})$, and in the M step we optimize $\boldsymbol{\eta}$. If the linear regression setting, the E step can be performed exactly, but in general we need to use approximations. Traditional methods use numerical quadrature or Monte Carlo (see e.g., (Breslow and Clayton 1993)). A faster approach is to use variational EM; see (Braun and McAuliffe 2010) for an application of variational EM to a multi-level discrete choice modeling problem.

In frequentist statistics, there is a popular method for fitting GLMMs called **generalized estimating equations** or **GEE** (Hardin and Hilbe 2003). However, we do not recommend this approach, since it is not as statistically efficient as likelihood-based methods (see Section 6.4.3). In addition, it can only provide estimates of the population parameters $\boldsymbol{\alpha}$, but not the random effects $\boldsymbol{\beta}_j$, which are sometimes of interest in themselves.

9.7 Learning to rank *

In this section, we discuss the **learning to rank** or **LETOR** problem. That is, we want to learn a function that can rank order a set of items (we will be more precise below). The most common application is to information retrieval. Specifically, suppose we have a query q and a set of documents $d^1, \ldots, d^m$ that might be relevant to q (e.g., all documents that contain the string q). We would like to sort these documents in decreasing order of relevance and show the top k to the user. Similar problems arise in other areas, such as collaborative filtering. (Ranking players in a game or tournament setting is a slightly different kind of problem; see Section 22.5.5.)

Below we summarize some methods for solving this problem, following the presentation of (Liu 2009). This material is not based on GLMs, but we include it in this chapter anyway for lack of a better place.

A standard way to measure the relevance of a document d to a query q is to use a probabilistic language model based on a bag of words model. That is, we define $sim(q, d) \triangleq p(q|d) = \prod_{i=1}^{n} p(q_i|d)$, where q_i is the i'th word or term, and $p(q_i|d)$ is a multinoulli distribution estimated from document d. In practice, we need to smooth the estimated distribution, for example by using a Dirichlet prior, representing the overall frequency of each word. This can be estimated from all documents in the system. More precisely, we can use

$$p(t|d) = (1-\lambda)\frac{\text{TF}(t,d)}{\text{LEN}(d)} + \lambda p(t|\text{background}) \tag{9.117}$$

where TF(t,d) is the frequency of term t in document d, LEN(d) is the number of words in d, and $0 < \lambda < 1$ is a smoothing parameter (see e.g., Zhai and Lafferty (2004) for details).

However, there might be many other signals that we can use to measure relevance. For example, the PageRank of a web document is a measure of its authoritativeness, derived from the web's link structure (see Section 17.2.4 for details). We can also compute how often and where the query occurs in the document. Below we discuss how to learn how to combine all these signals.[4]

9.7.1 The pointwise approach

Suppose we collect some training data representing the relevance of a set of documents for each query. Specifically, for each query q, suppose that we retrieve m possibly relevant documents d_j, for $j = 1:m$. For each query document pair, we define a feature vector, $\mathbf{x}(q,d)$. For example, this might contain the query-document similarity score and the page rank score of the document. Furthermore, suppose we have a set of labels y_j representing the degree of relevance of document d_j to query q. Such labels might be binary (e.g., relevant or irrelevant), or they may represent a degree of relevance (e.g., very relevant, somewhat relevant, irrelevant). Such labels can be obtained from query logs, by thresholding the number of times a document was clicked on for a given query.

If we have binary relevance labels, we can solve the problem using a standard binary classification scheme to estimate, $p(y=1|\mathbf{x}(q,d))$. If we have ordered relevancy labels, we can use ordinal regression to predict the rating, $p(y=r|\mathbf{x}(q,d))$. In either case, we can then sort the documents by this scoring metric. This is called the **pointwise approach** to LETOR, and is widely used because of its simplicity. However, this method does not take into account the location of each document in the list. Thus it penalizes errors at the end of the list just as much as errors at the beginning, which is often not the desired behavior. In addition, each decision about relevance is made very myopically.

9.7.2 The pairwise approach

There is evidence (e.g., (Carterette et al. 2008)) that people are better at judging the relative relevance of two items rather than absolute relevance. Consequently, the data might tell us that d_j is more relevant than d_k for a given query, or vice versa. We can model this kind of data using a binary classifier of the form $p(y_{jk}|\mathbf{x}(q,d_j),\mathbf{x}(q,d_k))$, where we set $y_{jk}=1$ if rel$(d_j,q) >$ rel(d_k,q) and $y_{jk}=0$ otherwise.

One way to model such a function is as follows:

$$p(y_{jk}=1|\mathbf{x}_j,\mathbf{x}_k) = \text{sigm}(f(\mathbf{x}_j) - f(\mathbf{x}_k)) \tag{9.118}$$

4. Rather surprisingly, Google does not (or at least, did not as of 2008) using such learning methods in its search engine. Source: Peter Norvig, quoted in `http://anand.typepad.com/datawocky/2008/05/are-human-experts-less-prone-to-catastrophic-errors-than-machine-learned-models.html`.

where $f(\mathbf{x})$ is a scoring function, often taken to be linear, $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. This is a special kind of neural network known as **RankNet** (Burges et al. 2005) (see Section 16.5 for a general discussion of neural networks). We can find the MLE of $\mathbf{w}$ by maximizing the log likelihood, or equivalently, by minimizing the cross entropy loss, given by

$$L = \sum_{i=1}^{N}\sum_{j=1}^{m_i}\sum_{k=j+1}^{m_i} L_{ijk} \tag{9.119}$$

$$\begin{aligned}-L_{ijk} &= \mathbb{I}(y_{ijk}=1)\log p(y_{ijk}=1|\mathbf{x}_{ij},\mathbf{x}_{ik},\mathbf{w}) \\ &+\mathbb{I}(y_{ijk}=0)\log p(y_{ijk}=0|\mathbf{x}_{ij},\mathbf{x}_{ik},\mathbf{w})\end{aligned} \tag{9.120}$$

This can be optimized using gradient descent. A variant of RankNet is used by Microsoft's Bing search engine.[5]

9.7.3 The listwise approach

The pairwise approach suffers from the problem that decisions about relevance are made just based on a pair of items (documents), rather than considering the full context. We now consider methods that look at the entire list of items at the same time.

We can define a total order on a list by specifying a permutation of its indices, $\boldsymbol{\pi}$. To model our uncertainty about $\boldsymbol{\pi}$, we can use the **Plackett-Luce** distribution, which derives its name from independent work by (Plackett 1975) and (Luce 1959). This has the following form:

$$p(\boldsymbol{\pi}|\mathbf{s}) = \prod_{j=1}^{m}\frac{s_j}{\sum_{u=j}^{m} s_u} \tag{9.121}$$

where $s_j = s(\pi^{-1}(j))$ is the score of the document ranked at the j'th position.

To understand Equation 9.121, let us consider a simple example. Suppose $\boldsymbol{\pi} = (A, B, C)$. Then we have that $p(\boldsymbol{\pi})$ is the probability of A being ranked first, times the probability of B being ranked second given that A is ranked first, times the probabilty of C being ranked third given that A and B are ranked first and second. In other words,

$$p(\boldsymbol{\pi}|\mathbf{s}) = \frac{s_A}{s_A+s_B+s_C}\times\frac{s_B}{s_B+s_C}\times\frac{s_C}{s_C} \tag{9.122}$$

To incorporate features, we can define $s(d) = f(\mathbf{x}(q,d))$, where we often take f to be a linear function, $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. This is known as the **ListNet** model (Cao et al. 2007). To train this model, let $\mathbf{y}_i$ be the relevance scores of the documents for query i. We then minimize the cross entropy term

$$-\sum_i\sum_{\boldsymbol{\pi}} p(\boldsymbol{\pi}|\mathbf{y}_i)\log p(\boldsymbol{\pi}|\mathbf{s}_i) \tag{9.123}$$

Of course, as stated, this is intractable, since the i'th term needs to sum over $m_i!$ permutations. To make this tractable, we can consider permutations over the top k positions only:

$$p(\boldsymbol{\pi}_{1:k}|\mathbf{s}_{1:m}) = \prod_{j=1}^{k}\frac{s_j}{\sum_{u=1}^{m} s_u} \tag{9.124}$$

5. Source: `http://www.bing.com/community/site_blogs/b/search/archive/2009/06/01/user-needs-features-and-the-science-behind-bing.aspx`.

There are only $m!/(m-k)!$ such permutations. If we set $k=1$, we can evaluate each cross entropy term (and its derivative) in $O(m)$ time.

In the special case where only one document from the presented list is deemed relevant, say $y_i = c$, we can instead use multinomial logistic regression:

$$p(y_i = c|\mathbf{x}) = \frac{\exp(s_c)}{\sum_{c'=1}^{m} \exp(s_{c'})} \tag{9.125}$$

This often performs at least as well as ranking methods, at least in the context of collaborative filtering (Yang et al. 2011).

9.7.4 Loss functions for ranking

There are a variety of ways to measure the performance of a ranking system, which we summarize below.

- **Mean reciprocal rank (MRR)**. For a query q, let the rank position of its first relevant document be denoted by $r(q)$. Then we define the **mean reciprocal rank** to be $1/r(q)$. This is a very simple performance measure.
- **Mean average precision (MAP)**. In the case of binary relevance labels, we can define the **precision at k** of some ordering as follows:

  $$\text{P@k}(\boldsymbol{\pi}) \triangleq \frac{\text{num. relevant documents in the top } k \text{ positions of } \boldsymbol{\pi}}{k} \tag{9.126}$$

  We then define the average precision as follows:

  $$\text{AP}(\boldsymbol{\pi}) \triangleq \frac{\sum_k \text{P@k}(\boldsymbol{\pi}) \cdot I_k}{\text{num. relevant documents}} \tag{9.127}$$

  where I_k is 1 iff document k is relevant. For example, if we have the relevancy labels $\mathbf{y} = (1,0,1,0,1)$, then the AP is $\frac{1}{3}(\frac{1}{1}+\frac{2}{3}+\frac{3}{5}) \approx 0.76$. Finally, we define the **mean average precision** as the AP averaged over all queries.
- **Normalized discounted cumulative gain (NDCG)**. Suppose the relevance labels have multiple levels. We can define the **discounted cumulative gain** of the first k items in an ordering as follows:

  $$\text{DCG@k}(\mathbf{r}) = r_1 + \sum_{i=2}^{k} \frac{r_i}{\log_2 i} \tag{9.128}$$

  where r_i is the relevance of item i and the $\log_2$ term is used to discount items later in the list. Table 9.3 gives a simple numerical example. An alternative definition, that places stronger emphasis on retrieving relevant documents, uses

  $$\text{DCG@k}(\mathbf{r}) = \sum_{i=1}^{k} \frac{2^{r_i}-1}{\log_2(1+i)} \tag{9.129}$$

  The trouble with DCG is that it varies in magnitude just because the length of a returned list may vary. It is therefore common to normalize this measure by the ideal DCG, which is

i	1	2	3	4	5	6
r_i	3	2	3	0	1	2
$\log_2 i$	0	1	1.59	2.0	2.32	2.59
$\frac{r_i}{\log_2 i}$	N/A	2	1.887	0	0.431	0.772

Table 9.3 Illustration of how to compute NDCG, from `http://en.wikipedia.org/wiki/Discounted_cumulative_gain`. The value r_i is the relevance score of the item in position i. From this, we see that DCG@6 $= 3 + (2 + 1.887 + 0 + 0.431 + 0.772) = 8.09$. The maximum DCG is obtained using the ordering with scores 3, 3, 2, 2, 1, 0. Hence the ideal DCG is 8.693, and so the normalized DCG is 8.09 / 8.693 = 0.9306.

the DCG obtained by using the optimal ordering: IDCG@k$(\mathbf{r}) = \text{argmax}_{\boldsymbol{\pi}}$ DCG@k$(\mathbf{r})$. This can be easily computed by sorting $r_{1:m}$ and then computing DCG@k. Finally, we define the **normalized discounted cumulative gain** or **NDCG** as DCG/IDCG. Table 9.3 gives a simple numerical example. The NDCG can be averaged over queries to give a measure of performance.

- **Rank correlation**. We can measure the correlation between the ranked list, $\boldsymbol{\pi}$, and the relevance judgement, $\boldsymbol{\pi}^*$, using a variety of methods. One approach, known as the (weighted) **Kendall's** τ statistics, is defined in terms of the weighted pairwise inconsistency between the two lists:

$$\tau(\boldsymbol{\pi}, \boldsymbol{\pi}^*) = \frac{\sum_{u<v} w_{uv} \left[1 + \text{sgn}(\pi_u - \pi_v)\text{sgn}(\pi_u^* - \pi_v^*)\right]}{2\sum_{u<v} w_{uv}} \tag{9.130}$$

A variety of other measures are commonly used.

These loss functions can be used in different ways. In the Bayesian approach, we first fit the model using posterior inference; this depends on the likelihood and prior, but not the loss. We then choose our actions at test time to minimize the expected future loss. One way to do this is to sample parameters from the posterior, $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathcal{D})$, and then evaluate, say, the precision@k for different thresholds, averaging over $\boldsymbol{\theta}^s$. See (Zhang et al. 2010) for an example of such an approach.

In the frequentist approach, we try to minimize the empirical loss on the training set. The problem is that these loss functions are not differentiable functions of the model parameters. We can either use gradient-free optimization methods, or we can minimize a surrogate loss function instead. Cross entropy loss (i.e., negative log likelihood) is an example of a widely used surrogate loss function.

Another loss, known as **weighted approximate-rank pairwise** or **WARP** loss, proposed in (Usunier et al. 2009) and extended in (Weston et al. 2010), provides a better approximation to the precision@k loss. WARP is defined as follows:

$$\text{WARP}(\mathbf{f}(\mathbf{x}, :), y) \triangleq L(\text{rank}(\mathbf{f}(\mathbf{x}, :), y)) \tag{9.131}$$

$$\text{rank}(\mathbf{f}(\mathbf{x}, :), y) = \sum_{y' \neq y} \mathbb{I}(f(\mathbf{x}, y') \geq f(\mathbf{x}, y)) \tag{9.132}$$

$$L(k) \triangleq \sum_{j=1}^{k} \alpha_j, \quad \text{with } \alpha_1 \geq \alpha_2 \geq \cdots \geq 0 \tag{9.133}$$

Here $\mathbf{f}(x, :) = [f(\mathbf{x}, 1), \ldots, f(\mathbf{x}, |y|)]$ is the vector of scores for each possible output label, or, in IR terms, for each possible document corresponding to input query $\mathbf{x}$. The expression rank$(\mathbf{f}(\mathbf{x}, :), y)$ measures the rank of the true label y assigned by this scoring function. Finally, L transforms the integer rank into a real-valued penalty. Using $\alpha_1 = 1$ and $\alpha_{j>1} = 0$ would optimize the proportion of top-ranked correct labels. Setting $\alpha_{1:k}$ to be non-zero values would optimize the top k in the ranked list, which will induce good performance as measured by MAP or precision@k. As it stands, WARP loss is still hard to optimize, but it can be further approximated by Monte Carlo sampling, and then optimized by gradient descent, as described in (Weston et al. 2010).

Exercises

Exercise 9.1 Conjugate prior for univariate Gaussian in exponential family form

Derive the conjugate prior for μ and $\lambda = 1/\sigma^2$ for a univariate Gaussian using the exponential family, by analogy to Section 9.2.5.5. By suitable reparameterization, show that the prior has the form $p(\mu, \lambda) = \mathcal{N}(\mu|\gamma, \lambda(2\alpha - 1))\text{Ga}(\lambda|\alpha, \beta)$, and thus only has 3 free parameters.

Exercise 9.2 The MVN is in the exponential family

Show that we can write the MVN in exponential family form. Hint: use the information form defined in Section 4.3.3.

10 *Directed graphical models (Bayes nets)*

10.1 Introduction

I basically know of two principles for treating complicated systems in simple ways: the first is the principle of modularity and the second is the principle of abstraction. I am an apologist for computational probability in machine learning because I believe that probability theory implements these two principles in deep and intriguing ways — namely through factorization and through averaging. Exploiting these two mechanisms as fully as possible seems to me to be the way forward in machine learning. — Michael Jordan, 1997 (quoted in (Frey 1998)).

Suppose we observe multiple correlated variables, such as words in a document, pixels in an image, or genes in a microarray. How can we compactly *represent* the **joint distribution** $p(\mathbf{x}|\boldsymbol{\theta})$? How can we use this distribution to *infer* one set of variables given another in a reasonable amount of computation time? And how can we *learn* the parameters of this distribution with a reasonable amount of data? These questions are at the core of probabilistic modeling, inference and learning, and form the topic of this chapter.

10.1.1 Chain rule

By the **chain rule** of probability, we can always represent a joint distribution as follows, using any ordering of the variables:

$$p(x_{1:V}) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)p(x_4|x_1, x_2, x_3) \ldots p(x_V|x_{1:V-1}) \tag{10.1}$$

where V is the number of variables, the Matlab-like notation $1 : V$ denotes the set $\{1, 2, \ldots, V\}$, and where we have dropped the conditioning on the fixed parameters $\boldsymbol{\theta}$ for brevity. The problem with this expression is that it becomes more and more complicated to represent the conditional distributions $p(x_t|\mathbf{x}_{1:t-1})$ as t gets large.

For example, suppose all the variables have K states. We can represent $p(x_1)$ as a table of $O(K)$ numbers, representing a discrete distribution (there are actually only $K - 1$ free parameters, due to the sum-to-one constraint, but we write $O(K)$ for simplicity). Similarly, we can represent $p(x_2|x_1)$ as a table of $O(K^2)$ numbers by writing $p(x_2 = j|x_1 = i) = T_{ij}$; we say that $\mathbf{T}$ is a **stochastic matrix**, since it satisfies the constraint $\sum_j T_{ij} = 1$ for all rows i, and $0 \leq T_{ij} \leq 1$ for all entries. Similarly, we can represent $p(x_3|x_1, x_2)$ as a 3d table with

$O(K^3)$ numbers. These are called **conditional probability tables** or **CPTs**. We see that there are $O(K^V)$ parameters in the model. We would need an awful lot of data to learn so many parameters.

One solution is to replace each CPT with a more parsimonius **conditional probability distribution** or **CPD**, such as multinomial logistic regression, i.e., $p(x_t = k|\mathbf{x}_{1:t-1}) = \mathcal{S}(\mathbf{W}_t\mathbf{x}_{1:t-1})_k$. The total number of parameters is now only $O(K^2V^2)$, making this a compact density model (Neal 1992; Frey 1998). This is adequate if all we want to do is evaluate the probability of a fully observed vector $\mathbf{x}_{1:T}$. For example, we can use this model to define a class-conditional density, $p(\mathbf{x}|y = c)$, thus making a generative classifier (Bengio and Bengio 2000). However, this model is not useful for other kinds of prediction tasks, since each variable depends on all the previous variables. So we need another approach.

10.1.2 Conditional independence

The key to efficiently representing large joint distributions is to make some assumptions about conditional independence (**CI**). Recall from Section 2.2.4 that X and Y are conditionally independent given Z, denoted $X \perp Y|Z$, if and only if (iff) the conditional joint can be written as a product of conditional marginals, i.e.,

$$X \perp Y|Z \iff p(X, Y|Z) = p(X|Z)p(Y|Z) \tag{10.2}$$

Let us see why this might help. Suppose we assume that $x_{t+1} \perp \mathbf{x}_{1:t-1}|x_t$, or in words, “the future is independent of the past given the present”. This is called the (first order) **Markov assumption**. Using this assumption, plus the chain rule, we can write the joint distribution as follows:

$$p(\mathbf{x}_{1:V}) = p(x_1)\prod_{t=1}^{V} p(x_t|x_{t-1}) \tag{10.3}$$

This is called a (first-order) **Markov chain**. They can be characterized by an initial distribution over states, $p(x_1 = i)$, plus a **state transition matrix** $p(x_t = j|x_{t-1} = i)$. See Section 17.2 for more information.

10.1.3 Graphical models

Although the first-order Markov assumption is useful for defining distributions on 1d sequences, how can we define distributions on 2d images, or 3d videos, or, in general, arbitrary collections of variables (such as genes belonging to some biological pathway)? This is where graphical models come in.

A **graphical model** (**GM**) is a way to represent a joint distribution by making CI assumptions. In particular, the nodes in the graph represent random variables, and the (lack of) edges represent CI assumptions. (A better name for these models would in fact be “independence diagrams”, but the term “graphical models” is now entrenched.) There are several kinds of graphical model, depending on whether the graph is directed, undirected, or some combination of directed and undirected. In this chapter, we just study directed graphs. We consider undirected graphs in Chapter 19.

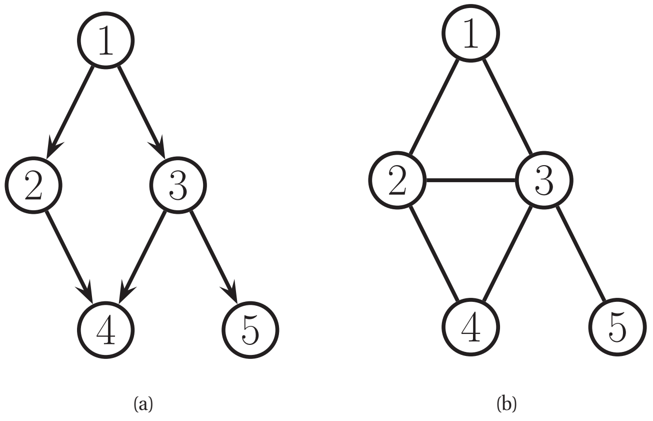


Figure 10.1 (a) A simple DAG on 5 nodes, numbered in topological order. Node 1 is the root, nodes 4 and 5 are the leaves. (b) A simple undirected graph, with the following maximal cliques: $\{1, 2, 3\}$, $\{2, 3, 4\}$, $\{3, 5\}$.

10.1.4 Graph terminology

Before we continue, we must define a few basic terms, most of which are very intuitive.

A **graph** $G = (\mathcal{V}, \mathcal{E})$ consists of a set of **nodes** or **vertices**, $\mathcal{V} = \{1, \ldots, V\}$, and a set of **edges**, $\mathcal{E} = \{(s, t) : s, t \in \mathcal{V}\}$. We can represent the graph by its **adjacency matrix**, in which we write $G(s, t) = 1$ to denote $(s, t) \in \mathcal{E}$, that is, if $s \to t$ is an edge in the graph. If $G(s, t) = 1$ iff $G(t, s) = 1$, we say the graph is **undirected**, otherwise it is **directed**. We usually assume $G(s, s) = 0$, which means there are no **self loops**.

Here are some other terms we will commonly use:

- **Parent** For a directed graph, the **parents** of a node is the set of all nodes that feed into it: $\text{pa}(s) \triangleq \{t : G(t, s) = 1\}$.
- **Child** For a directed graph, the **children** of a node is the set of all nodes that feed out of it: $\text{ch}(s) \triangleq \{t : G(s, t) = 1\}$.
- **Family** For a directed graph, the **family** of a node is the node and its parents, $\text{fam}(s) = \{s\} \cup \text{pa}(s)$.
- **Root** For a directed graph, a **root** is a node with no parents.
- **Leaf** For a directed graph, a **leaf** is a node with no children.
- **Ancestors** For a directed graph, the **ancestors** are the parents, grand-parents, etc of a node. That is, the ancestors of t is the set of nodes that connect to t via a trail: $\text{anc}(t) \triangleq \{s : s \rightsquigarrow t\}$.
- **Descendants** For a directed graph, the **descendants** are the children, grand-children, etc of a node. That is, the descendants of s is the set of nodes that can be reached via trails from s: $\text{desc}(s) \triangleq \{t : s \rightsquigarrow t\}$.
- **Neighbors** For any graph, we define the **neighbors** of a node as the set of all immediately connected nodes, $\text{nbr}(s) \triangleq \{t : G(s, t) = 1 \vee G(t, s) = 1\}$. For an undirected graph, we

write $s \sim t$ to indicate that s and t are neighbors (so $(s, t) \in \mathcal{E}$ is an edge in the graph).

- **Degree** The **degree** of a node is the number of neighbors. For directed graphs, we speak of the **in-degree** and **out-degree**, which count the number of parents and children.
- **Cycle or loop** For any graph, we define a **cycle** or **loop** to be a series of nodes such that we can get back to where we started by following edges, $s_1 - s_2 \cdots - s_n - s_1$, $n \geq 2$. If the graph is directed, we may speak of a directed cycle. For example, in Figure 10.1(a), there are no directed cycles, but $1 \to 2 \to 4 \to 3 \to 1$ is an undirected cycle.
- **DAG** A **directed acyclic graph** or **DAG** is a directed graph with no directed cycles. See Figure 10.1(a) for an example.
- **Topological ordering** For a DAG, a **topological ordering** or **total ordering** is a numbering of the nodes such that parents have lower numbers than their children. For example, in Figure 10.1(a), we can use $(1, 2, 3, 4, 5)$, or $(1, 3, 2, 5, 4)$, etc.
- **Path or trail** A **path** or **trail** $s \rightsquigarrow t$ is a series of directed edges leading from s to t.
- **Tree** An undirected **tree** is an undirectecd graph with no cycles. A directed tree is a DAG in which there are no directed cycles. If we allow a node to have multiple parents, we call it a **polytree**, otherwise we call it a moral directed tree.
- **Forest** A **forest** is a set of trees.
- **Subgraph** A (node-induced) **subgraph** G_A is the graph created by using the nodes in A and their corresponding edges, $G_A = (\mathcal{V}_A, \mathcal{E}_A)$.
- **Clique** For an undirected graph, a **clique** is a set of nodes that are all neighbors of each other. A **maximal clique** is a clique which cannot be made any larger without losing the clique property. For example, in Figure 10.1(b), $\{1, 2\}$ is a clique but it is not maximal, since we can add 3 and still maintain the clique property. In fact, the maximal cliques are as follows: $\{1, 2, 3\}$, $\{2, 3, 4\}$, $\{3, 5\}$.

10.1.5 Directed graphical models

A **directed graphical model** or **DGM** is a GM whose graph is a DAG. These are more commonly known as **Bayesian networks**. However, there is nothing inherently "Bayesian" about Bayesian networks: they are just a way of defining probability distributions. These models are also called **belief networks**. The term "belief" here refers to subjective probability. Once again, there is nothing inherently subjective about the kinds of probability distributions represented by DGMs. Finally, these models are sometimes called **causal networks**, because the directed arrows are sometimes interpreted as representing causal relations. However, there is nothing inherently causal about DGMs. (See Section 26.6.1 for a discussion of causal DGMs.) For these reasons, we use the more neutral (but less glamorous) term DGM.

The key property of DAGs is that the nodes can be ordered such that parents come before children. This is called a topological ordering, and it can be constructed from any DAG. Given such an order, we define the **ordered Markov property** to be the assumption that a node only depends on its immediate parents, not on all predecessors in the ordering, i.e.,

$$x_s \perp \mathbf{x}_{\text{pred}(s) \setminus \text{pa}(s)} | \mathbf{x}_{\text{pa}(s)} \tag{10.4}$$

where $\text{pa}(s)$ are the parents of node s, and $\text{pred}(s)$ are the predecessors of node s in the ordering. This is a natural generalization of the first-order Markov property to from chains to general DAGs.

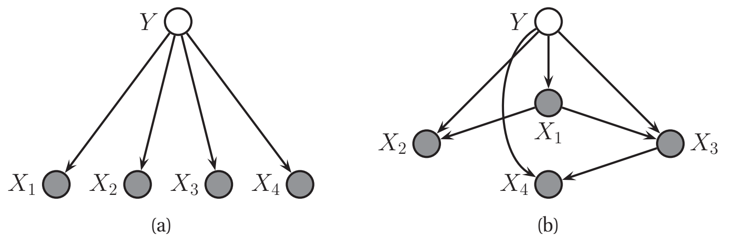


Figure 10.2 (a) A naive Bayes classifier represented as a DGM. We assume there are $D = 4$ features, for simplicity. Shaded nodes are observed, unshaded nodes are hidden. (b) Tree-augmented naive Bayes classifier for $D = 4$ features. In general, the tree topology can change depending on the value of y.

For example, the DAG in Figure 10.1(a) encodes the following joint distribution:

$$\begin{aligned} p(\mathbf{x}_{1:5}) &= p(x_1)p(x_2|x_1)p(x_3|x_1, \cancel{x_2})p(x_4|\cancel{x_1}, x_2, x_3)p(x_5|\cancel{x_1}, \cancel{x_2}, x_3, \cancel{x_4}) && (10.5) \\ &= p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2, x_3)p(x_5|x_3) && (10.6) \end{aligned}$$

In general, we have

$$p(\mathbf{x}_{1:V}|G) = \prod_{t=1}^{V} p(x_t|\mathbf{x}_{\text{pa}(t)}) \tag{10.7}$$

where each term $p(x_t|\mathbf{x}_{\text{pa}(t)})$ is a CPD. We have written the distribution as $p(\mathbf{x}|G)$ to emphasize that this equation only holds if the CI assumptions encoded in DAG G are correct. However, we will usual drop this explicit conditioning for brevity. If each node has $O(F)$ parents and K states, the number of parameters in the model is $O(VK^F)$, which is much less than the $O(K^V)$ needed by a model which makes no CI assumptions.

10.2 Examples

In this section, we show a wide variety of commonly used probabilistic models can be conveniently represented as DGMs.

10.2.1 Naive Bayes classifiers

In Section 3.5, we introduced the naive Bayes classifier. This assumes the features are conditionally independent given the class label. This assumption is illustrated in Figure 10.2(a). This allows us to write the joint distirbution as follows:

$$p(y, \mathbf{x}) = p(y) \prod_{j=1}^{D} p(x_j|y) \tag{10.8}$$

The naive Bayes assumption is rather naive, since it assumes the features are conditionally independent. One way to capture correlation between the features is to use a graphical model. In particular, if the model is a tree, the method is known as a **tree-augmented naive Bayes**

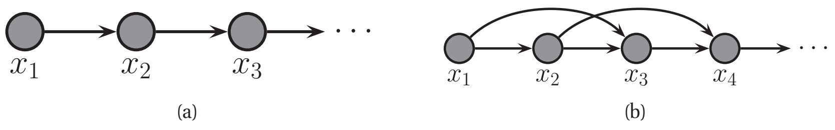


Figure 10.3 A first and second order Markov chain.

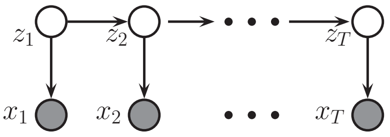


Figure 10.4 A first-order HMM.

classifier or **TAN** model (Friedman et al. 1997). This is illustrated in Figure 10.2(b). The reason to use a tree, as opposed to a generic graph, is two-fold. First, it is easy to find the optimal tree structure using the Chow-Liu algorithm, as explained in Section 26.3. Second, it is easy to handle missing features in a tree-structured model, as we explain in Section 20.2.

10.2.2 Markov and hidden Markov models

Figure 10.3(a) illustrates a first-order Markov chain as a DAG. Of course, the assumption that the immediate past, x_{t-1}, captures everything we need to know about the entire history, $\mathbf{x}_{1:t-2}$, is a bit strong. We can relax it a little by adding a dependence from x_{t-2} to x_t as well; this is called a **second order Markov chain**, and is illustrated in Figure 10.3(b). The corresponding joint has the following form:

$$p(\mathbf{x}_{1:T}) = p(x_1, x_2)p(x_3|x_1, x_2)p(x_4|x_2, x_3)\ldots = p(x_1, x_2)\prod_{t=3}^{T} p(x_t|x_{t-1}, x_{t-2}) \tag{10.9}$$

We can create higher-order Markov models in a similar way. See Section 17.2 for a more detailed discussion of Markov models.

Unfortunately, even the second-order Markov assumption may be inadequate if there are long-range correlations amongst the observations. We can't keep building ever higher order models, since the number of parameters will blow up. An alternative approach is to assume that there is an underlying hidden process, that can be modeled by a first-order Markov chain, but that the data is a noisy observation of this process. The result is known as a **hidden Markov model** or **HMM**, and is illustrated in Figure 10.4. Here z_t is known as a **hidden variable** at "time" t, and x_t is the observed variable. (We put "time" in quotation marks, since these models can be applied to any kind of sequence data, such as genomics or language, where t represents location rather than time.) The CPD $p(z_t|z_{t-1})$ is the **transition model**, and the CPD $p(\mathbf{x}_t|z_t)$ is the **observation model**.

h_0	h_1	h_2	$P(v=0\|h_1,h_2)$	$P(v=1\|h_1,h_2)$
1	0	0	θ_0	$1-\theta_0$
1	1	0	$\theta_0\theta_1$	$1-\theta_0\theta_1$
1	0	1	$\theta_0\theta_2$	$1-\theta_0\theta_2$
1	1	1	$\theta_0\theta_1\theta_2$	$1-\theta_0\theta_1\theta_2$

Table 10.1 Noisy-OR CPD for 2 parents augmented with leak node. We have omitted the t subscript for brevity.

The hidden variables often represent quantities of interest, such as the identity of the word that someone is currently speaking. The observed variables are what we measure, such as the acoustic waveform. What we would like to do is estimate the hidden state given the data, i.e., to compute $p(z_t|\mathbf{x}_{1:t},\boldsymbol{\theta})$. This is called **state estimation**, and is just another form of probabilistic inference. See Chapter 17 for further details on HMMs.

10.2.3 Medical diagnosis

Consider modeling the relationship between various variables that are measured in an intensive care unit (ICU), such as the breathing rate of a patient, their blood pressure, etc. The **alarm network** in Figure 10.5(a) is one way to represent these (in)dependencies (Beinlich et al. 1989). This model has 37 variables and 504 parameters.

Since this model was created by hand, by a process called **knowledge engineering**, it is known as a **probabilistic expert system**. In Section 10.4, we discuss how to learn the parameters of DGMs from data, assuming the graph structure is known, and in Chapter 26, we discuss how to learn the graph structure itself.

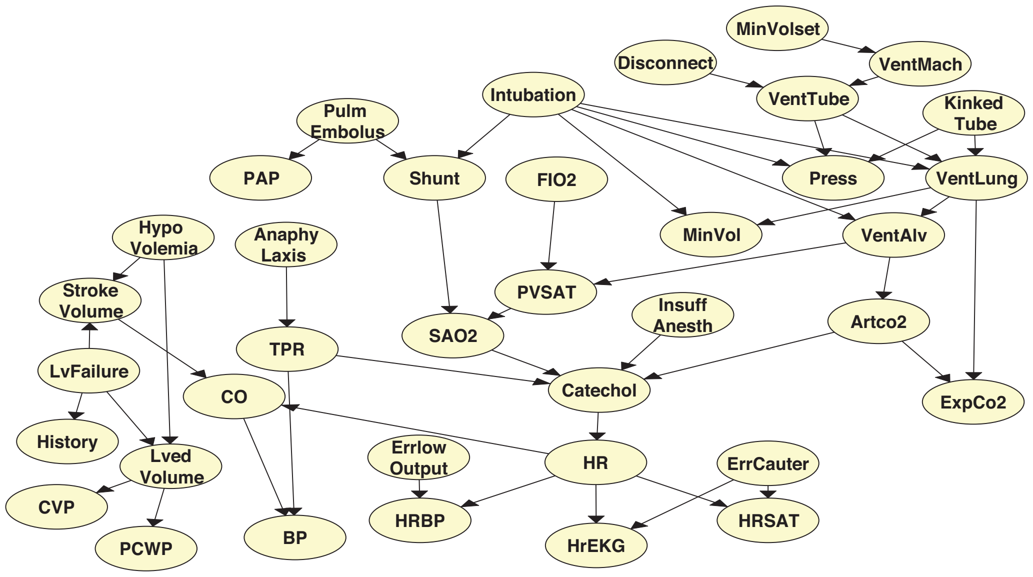
A different kind of medical diagnosis network, known as the **quick medical reference** or **QMR** network (Shwe et al. 1991), is shown in Figure 10.5(b). This was designed to model infectious diseases. The QMR model is a **bipartite graph** structure, with diseases (causes) at the top and symptoms or findings at the bottom. All nodes are binary. We can write the distribution as follows:

$$p(\mathbf{v},\mathbf{h}) = \prod_s p(h_s) \prod_t p(v_t|\mathbf{h}_{\text{pa}(t)}) \tag{10.10}$$

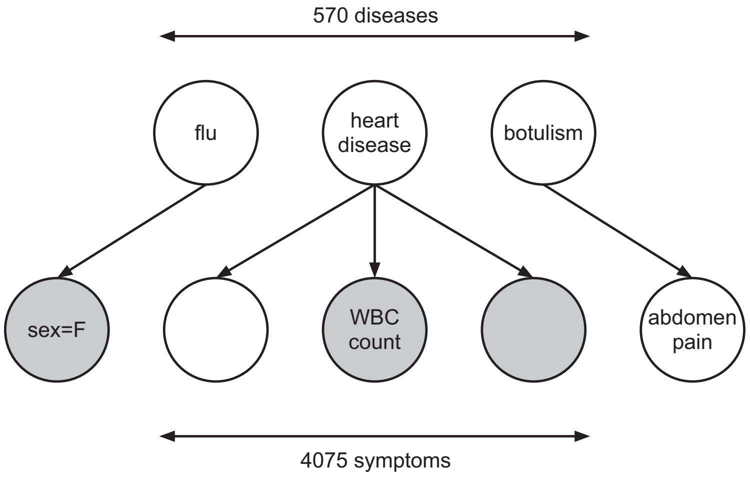
where h_s represent the **hidden nodes** (diseases), and v_t represent the **visible nodes** (symptoms).

The CPD for the root nodes are just Bernoulli distributions, representing the prior probability of that disease. Representing the CPDs for the leaves (symptoms) using CPTs would require too many parameters, because the **fan-in** (number of parents) of many leaf nodes is very high. A natural alternative is to use logistic regression to model the CPD, $p(v_t = 1|\mathbf{h}_{\text{pa}(t)}) = \text{sigm}(\mathbf{w}_t^T\mathbf{h}_{\text{pa}(t)})$. (A DGM in which the CPDs are logistic regression distributions is known as a **sigmoid belief net** (Neal 1992).) However, since the parameters of this model were created by hand, an alternative CPD, known as the **noisy-OR** model, was used.

The noisy-OR model assumes that if a parent is on, then the child will usually also be on (since it is an or-gate), but occasionally the "links" from parents to child may fail, independently at random. In this case, even if the parent is on, the child may be off. To model this more precisely, let $\theta_{st} = 1-q_{st}$ be the probability that the $s\to t$ link fails, so $q_{st} = 1-\theta_{st} = p(v_t =$



(a)



(b)

Figure 10.5 (a) The alarm network. Figure generated by `visualizeAlarmNetwork`. (b) The QMR network.

G^p	G^m	$p(X=a)$	$p(X=b)$	$p(X=o)$	$p(X=ab)$
a	a	1	0	0	0
a	b	0	0	0	1
a	o	1	0	0	0
b	a	0	0	0	1
b	b	0	1	0	0
b	o	0	1	0	0
o	a	1	0	0	0
o	b	0	1	0	0
o	o	0	0	1	0

Table 10.2 CPT which encodes a mapping from genotype to phenotype (bloodtype). This is a deterministic, but many-to-one, mapping.

$1|h_s = 1, \mathbf{h}_{-s} = 0)$ is the probability that s can activate t on its own (its "causal power"). The only way for the child to be off is if all the links from all parents that are on fail independently at random. Thus

$$p(v_t = 0|\mathbf{h}) = \prod_{s \in \text{pa}(t)} \theta_{st}^{\mathbb{I}(h_s=1)} \tag{10.11}$$

Obviously, $p(v_t = 1|\mathbf{h}) = 1 - p(v_t = 0|\mathbf{h})$.

If we observe that $v_t = 1$ but all its parents are off, then this contradicts the model. Such a data case would get probability zero under the model, which is problematic, because it is possible that someone exhibits a symptom but does not have any of the specified diseases. To handle this, we add a dummy **leak node** h_0, which is always on; this represents "all other causes". The parameter q_{0t} represents the probability that the background leak can cause the effect on its own. The modified CPD becomes $p(v_t = 0|\mathbf{h}) = \theta_{0t} \prod_{s \in \text{pa}(t)} \theta_{st}^{h_s}$. See Table 10.1 for a numerical example.

If we define $w_{st} \triangleq \log(\theta_{st})$, we can rewrite the CPD as

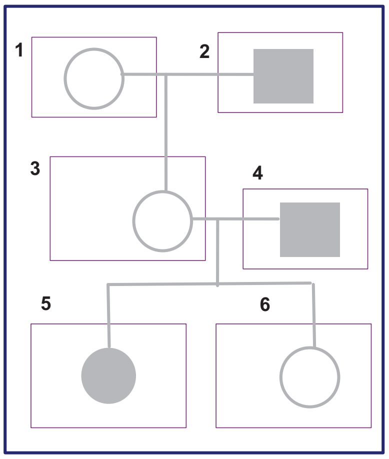
$$p(v_t = 1|\mathbf{h}) = 1 - \exp\left(w_{0t} + \sum_s h_s w_{st}\right) \tag{10.12}$$

We see that this is similar to a logistic regression model.

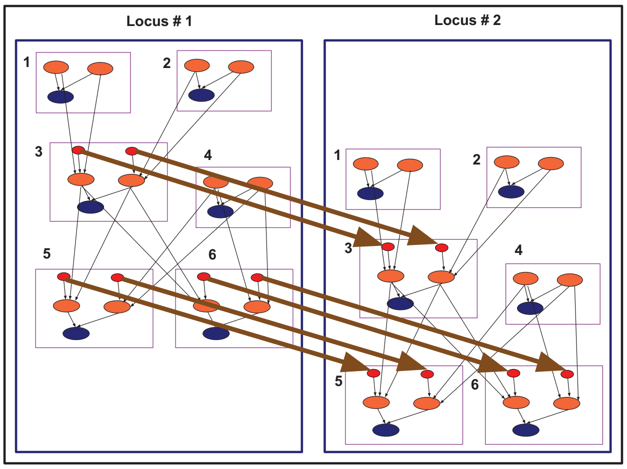
Bipartite models with noisy-OR CPDs are called **BN2O** models. It is relatively easy to set the θ_{st} parameters by hand, based on domain expertise. However, it is also possible to learn them from data (see e.g, (Neal 1992; Meek and Heckerman 1997)). Noisy-OR CPDs have also proved useful in modeling human causal learning (Griffiths and Tenenbaum 2005), as well as general binary classification settings (Yuille and Zheng 2009).

10.2.4 Genetic linkage analysis *

Another important (and historically very early) application of DGMs is to the problem of **genetic linkage analysis**. We start with a **pedigree graph**, which is a DAG that representing the relationship between parents and children, as shown in Figure 10.6(a). We then convert this to a DGM, as we explain below. Finally we perform probabilistic inference in the resulting model.



(a)



(b)

Figure 10.6 Left: family tree, circles are females, squares are males. Individuals with the disease of interest are highlighted. Right: DGM for two loci. Blue nodes X_{ij} is the observed phenotype for individual i at locus j. All other nodes are hidden. Orange nodes $G_{ij}^{p/m}$ is the paternal/ maternal allele. Small red nodes $z_{ijl}^{p/m}$ are the paternal/ maternal selection switching variables. These are linked across loci, $z_{ij}^m \rightarrow z_{i,j+1}^m$ and $z_{ij}^p \rightarrow z_{i,j+1}^p$. The founder (root) nodes do not have any parents, and hence do no need switching variables. Based on Figure 3 from (Friedman et al. 2000).

In more detail, for each person (or animal) i and location or locus j along the genome, we create three nodes: the observed **marker** X_{ij} (which can be a property such as blood type, or just a fragment of DNA that can be measured), and two hidden **alleles**, G^m_{ij} and G^p_{ij}, one inherited from i's mother (maternal allele) and the other from i's father (paternal allele). Together, the ordered pair $\mathbf{G}_{ij} = (G^m_{ij}, G^p_{ij})$ constitutes i's hidden **genotype** at locus j.

Obviously we must add $G^m_{ij} \to X_{ij}$ and $G^p_{ij} \to X_{ij}$ arcs representing the fact that genotypes cause phenotypes (observed manifestations of genotypes). The CPD $p(X_{ij}|G^m_{ij}, G^p_{ij})$ is called the **penetrance model**. As a very simple example, suppose $X_{ij} \in \{A, B, O, AB\}$ represents person i's observed bloodtype, and $G^m_{ij}, G^p_{ij} \in \{A, B, O\}$ is their genotype. We can represent the penetrance model using the deterministic CPD shown in Table 10.2. For example, A dominates O, so if a person has genotype AO or OA, their phenotype will be A.

In addition, we add arcs from i's mother and father into $\mathbf{G}_{ij}$, reflecting the **Mendelian inheritance** of genetic material from one's parents. More precisely, let $m_i = k$ be i's mother. Then G^m_{ij} could either be equal to G^m_{kj} or G^p_{kj}, that is, i's maternal allele is a copy of one of its mother's two alleles. Let Z^m_{ij} be a hidden variable than specifies the choice. We can model this using the following CPD, known as the **inheritance model**:

$$p(G^m_{ij}|G^m_{kj}, G^p_{kj}, Z^m_{ij}) = \begin{cases} \mathbb{I}(G^m_{ij} = G^m_{kj}) & \text{if } Z^m_{ij} = m \\ \mathbb{I}(G^m_{ij} = G^p_{kj}) & \text{if } Z^m_{ij} = p \end{cases} \tag{10.13}$$

We can define $p(G^p_{ij}|G^m_{kj}, G^p_{kj}, Z^p_{ij})$ similarly, where $k = p_i$ is i's father. The values of the Z_{ij} are said to specify the **phase** of the genotype. The values of $G^p_{i,j}$, $G^m_{i,j}$, $Z^p_{i,j}$ and $Z^m_{i,j}$ constitute the **haplotype** of person i at locus j.[1]

Next, we need to specify the prior for the root nodes, $p(G^m_{ij})$ and $p(G^p_{ij})$. This is called the **founder model**, and represents the overall prevalence of difference kinds of alleles in the population. We usually assume independence between the loci for these founder alleles.

Finally, we need to specify priors for the switch variables that control the inheritance process. These variables are spatially correlated, since adjacent sites on the genome are typically inherited together (recombination events are rare). We can model this by imposing a two-state Markov chain on the Z's, where the probability of switching state at locus j is given by $\theta_j = \frac{1}{2}(1 - e^{-2d_j})$, where d_j is the distance between loci j and $j+1$. This is called the **recombination model**.

The resulting DGM is shown in Figure 10.6(b): it is a series of replicated pedigree DAGs, augmented with switching Z variables, which are linked using Markov chains. (There is a related model known as **phylogenetic HMM** (Siepel and Haussler 2003), which is used to model evolution amongst phylogenies.)

As a simplified example of how this model can be used, suppose we only have one locus, corresponding to blood type. For brevity, we will drop the j index. Suppose we observe $x_i = A$. Then there are 3 possible genotypes: $\mathbf{G}_i$ is (A, A), (A, O) or (O, A). There is ambiguity because the genotype to phenotype mapping is many-to-one. We want to reverse this mapping. This is known as an **inverse problem**. Fortunately, we can use the blood types of relatives to help disambiguate the evidence. Information will "flow" from the other $x_{i'}$'s up to their $\mathbf{G}_{i'}$'s, then across to i's $\mathbf{G}_i$ via the pedigree DAG. Thus we can combine our **local evidence** $p(x_i|\mathbf{G}_i)$

1. Sometimes the observed marker is equal to the unphased genotype, which is the unordered set $\{G^p_{ij}, G^m_{ij}\}$; however, the phased or hidden genotype is not directly measurable.

with an informative prior, $p(\mathbf{G}_i|\mathbf{x}_{-i})$, conditioned on the other data, to get a less entropic local posterior, $p(G_i|\mathbf{x}) \propto p(x_i|G_i)p(G_i|\mathbf{x}_{-i})$.

In practice, the model is used to try to determine where along the genome a given disease-causing gene is assumed to lie — this is the genetic linkage analysis task. The method works as follows. First, suppose all the parameters of the model, including the distance between all the marker loci, are known. The only unknown is the location of the disease-causing gene. If there are L marker loci, we construct $L+1$ models: in model ℓ, we postulate that the disease gene comes after marker ℓ, for $0 < \ell < L+1$. We can estimate the Markov switching parameter $\hat{\theta}_\ell$, and hence the distance d_ℓ between the disease gene and its nearest known locus. We measure the quality of that model using its likelihood, $p(\mathcal{D}|\hat{\theta}_\ell)$. We then can then pick the model with highest likelihood (which is equivalent to the MAP model under a uniform prior).

Note, however, that computing the likelihood requires marginalizing out all the hidden Z and G variables. See (Fishelson and Geiger 2002) and the references therein for some exact methods for this task; these are based on the variable elimination algorithm, which we discuss in Section 20.3. Unfortunately, for reasons we explain in Section 20.5, exact methods can be computationally intractable if the number of individuals and/or loci is large. See (Albers et al. 2006) for an approximate method for computing the likelihood; this is based on a form of variational inference, which we will discuss in Section 22.4.1.

10.2.5 Directed Gaussian graphical models *

Consider a DGM where all the variables are real-valued, and all the CPDs have the following form:

$$p(x_t|\mathbf{x}_{\text{pa}(t)}) = \mathcal{N}(x_t|\mu_t + \mathbf{w}_t^T \mathbf{x}_{\text{pa}(t)}, \sigma_t^2) \tag{10.14}$$

This is called a **linear Gaussian** CPD. As we show below, multiplying all these CPDs together results in a large joint Gaussian distribution of the form $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. This is called a directed GGM, or a **Gaussian Bayes net**.

We now explain how to derive $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ from the CPD parameters, following (Shachter and Kenley 1989, App. B). For convenience, we will rewrite the CPDs in the following form:

$$x_t = \mu_t + \sum_{s \in \text{pa}(t)} w_{ts}(x_s - \mu_s) + \sigma_t z_t \tag{10.15}$$

where $z_t \sim \mathcal{N}(0,1)$, σ_t is the conditional standard deviation of x_t given its parents, w_{ts} is the strength of the $s \to t$ edge, and μ_t is the local mean.[2]

It is easy to see that the global mean is just the concatenation of the local means, $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_D)$. We now derive the global covariance, $\boldsymbol{\Sigma}$. Let $\mathbf{S} \triangleq \text{diag}(\boldsymbol{\sigma})$ be a diagonal matrix containing the standard deviations. We can rewrite Equation 10.15 in matrix-vector form as follows:

$$(\mathbf{x} - \boldsymbol{\mu}) = \mathbf{W}(\mathbf{x} - \boldsymbol{\mu}) + \mathbf{S}\mathbf{z} \tag{10.16}$$

2. If we do not subtract off the parent's mean (i.e., if we use $x_t = \mu_t + \sum_{s \in \text{pa}(t)} w_{ts} x_s + \sigma_t z_t$), the derivation of $\boldsymbol{\Sigma}$ is much messier, as can be seen by looking at (Bishop 2006b, p370).

Now let $\mathbf{e}$ be a vector of noise terms:

$$\mathbf{e} \triangleq \mathbf{S}\mathbf{z} \tag{10.17}$$

We can rearrange this to get

$$\mathbf{e} = (\mathbf{I} - \mathbf{W})(\mathbf{x} - \boldsymbol{\mu}) \tag{10.18}$$

Since $\mathbf{W}$ is lower triangular (because $w_{ts} = 0$ if $t > s$ in the topological ordering), we have that $\mathbf{I} - \mathbf{W}$ is lower triangular with 1s on the diagonal. Hence

$$\begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_d \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ -w_{21} & 1 & & & \\ -w_{32} & -w_{31} & 1 & & \\ \vdots & & & \ddots & \\ -w_{d1} & -w_{d2} & \dots & -w_{d,d-1} & 1 \end{pmatrix} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \\ \vdots \\ x_d - \mu_d \end{pmatrix} \tag{10.19}$$

Since $\mathbf{I} - \mathbf{W}$ is always invertible, we can write

$$\mathbf{x} - \boldsymbol{\mu} = (\mathbf{I} - \mathbf{W})^{-1}\mathbf{e} \triangleq \mathbf{U}\mathbf{e} = \mathbf{U}\mathbf{S}\mathbf{z} \tag{10.20}$$

where we defined $\mathbf{U} = (\mathbf{I} - \mathbf{W})^{-1}$. Thus the regression weights correspond to a Cholesky decomposition of $\boldsymbol{\Sigma}$, as we now show:

$$\begin{aligned} \boldsymbol{\Sigma} &= \text{cov}[\mathbf{x}] = \text{cov}[\mathbf{x} - \boldsymbol{\mu}] & (10.21) \\ &= \text{cov}[\mathbf{U}\mathbf{S}\mathbf{z}] = \mathbf{U}\mathbf{S}\,\text{cov}[\mathbf{z}]\;\mathbf{S}\mathbf{U}^T = \mathbf{U}\mathbf{S}^2\mathbf{U}^T & (10.22) \end{aligned}$$

10.3 Inference

We have seen that graphical models provide a compact way to define joint probability distributions. Given such a joint distribution, what can we do with it? The main use for such a joint distribution is to perform **probabilistic inference**. This refers to the task of estimating unknown quantities from known quantities. For example, in Section 10.2.2, we introduced HMMs, and said that one of the goals is to estimate the hidden states (e.g., words) from the observations (e.g., speech signal). And in Section 10.2.4, we discussed genetic linkage analysis, and said that one of the goals is to estimate the likelihood of the data under various DAGs, corresponding to different hypotheses about the location of the disease-causing gene.

In general, we can pose the inference problem as follows. Suppose we have a set of correlated random variables with joint distribution $p(\mathbf{x}_{1:V}|\boldsymbol{\theta})$. (In this section, we are assuming the parameters $\boldsymbol{\theta}$ of the model are known. We discuss how to learn the parameters in Section 10.4.) Let us partition this vector into the **visible variables** $\mathbf{x}_v$, which are observed, and the **hidden variables**, $\mathbf{x}_h$, which are unobserved. Inference refers to computing the posterior distribution of the unknowns given the knowns:

$$p(\mathbf{x}_h|\mathbf{x}_v, \boldsymbol{\theta}) = \frac{p(\mathbf{x}_h, \mathbf{x}_v|\boldsymbol{\theta})}{p(\mathbf{x}_v|\boldsymbol{\theta})} = \frac{p(\mathbf{x}_h, \mathbf{x}_v|\boldsymbol{\theta})}{\sum_{\mathbf{x}'_h} p(\mathbf{x}'_h, \mathbf{x}_v|\boldsymbol{\theta})} \tag{10.23}$$

Essentially we are **conditioning** on the data by **clamping** the visible variables to their observed values, $\mathbf{x}_v$, and then normalizing, to go from $p(\mathbf{x}_h, \mathbf{x}_v)$ to $p(\mathbf{x}_h|\mathbf{x}_v)$. The normalization constant $p(\mathbf{x}_v|\boldsymbol{\theta})$ is the likelihood of the data, also called the **probability of the evidence**.

Sometimes only some of the hidden variables are of interest to us. So let us partition the hidden variables into **query variables**, $\mathbf{x}_q$, whose value we wish to know, and the remaining **nuisance variables**, $\mathbf{x}_n$, which we are not interested in. We can compute what we are interested in by **marginalizing out** the nuisance variables:

$$p(\mathbf{x}_q|\mathbf{x}_v,\boldsymbol{\theta}) = \sum_{\mathbf{x}_n} p(\mathbf{x}_q,\mathbf{x}_n|\mathbf{x}_v,\boldsymbol{\theta}) \tag{10.24}$$

In Section 4.3.1, we saw how to perform all these operations for a multivariate Gaussian in $O(V^3)$ time, where V is the number of variables. What if we have discrete random variables, with say K states each? If the joint distribution is represented as a multi-dimensional table, we can always perform these operations exactly, but this will take $O(K^V)$ time. In Chapter 20, we explain how to exploit the factorization encoded by the GM to perform these operations in $O(VK^{w+1})$ time, where w is a quantity known as the treewidth of the graph. This measures how "tree-like" the graph is. If the graph is a tree (or a chain), we have $w = 1$, so for these models, inference takes time linear in the number of nodes. Unfortunately, for more general graphs, exact inference can take time exponential in the number of nodes, as we explain in Section 20.5. We will therefore examine various approximate inference schemes later in the book.

10.4 Learning

In the graphical models literature, it is common to distinguish between inference and learning. Inference means computing (functions of) $p(\mathbf{x}_h|\mathbf{x}_v,\boldsymbol{\theta})$, where v are the visible nodes, h are the hidden nodes, and $\boldsymbol{\theta}$ are the parameters of the model, assumed to be known. Learning usually means computing a MAP estimate of the parameters given data:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(\mathbf{x}_{i,v}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \tag{10.25}$$

where $\mathbf{x}_{i,v}$ are the visible variables in case i. If we have a uniform prior, $p(\boldsymbol{\theta}) \propto 1$, this reduces to the MLE, as usual.

If we adopt a Bayesian view, the parameters are unknown variables and should also be inferred. Thus to a Bayesian, there is no distinction between inference and learning. In fact, we can just add the parameters as nodes to the graph, condition on $\mathcal{D}$, and then infer the values of all the nodes. (We discuss this in more detail below.)

In this view, the main difference between hidden variables and parameters is that the number of hidden variables grows with the amount of training data (since there is usually a set of hidden variables for each observed data case), whereas the number of parameters in usually fixed (at least in a parametric model). This means that we must integrate out the hidden variables to avoid overfitting, but we may be able to get away with point estimation techniques for parameters, which are fewer in number.

10.4.1 Plate notation

When inferring parameters from data, we often assume the data is iid. We can represent this assumption explicitly using a graphical model, as shown in Figure 10.7(a). This illustrates the

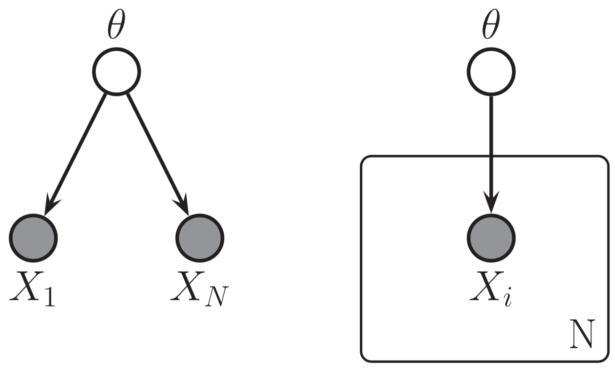


Figure 10.7 Left: data points x_i are conditionally independent given θ. Right: Plate notation. This represents the same model as the one on the left, except the repeated x_i nodes are inside a box, known as a plate; the number in the lower right hand corner, N, specifies the number of repetitions of the X_i node.

assumption that each data case was generated independently but from the same distribution. Notice that the data cases are only independent conditional on the parameters $\boldsymbol{\theta}$; marginally, the data cases are dependent. Nevertheless, we can see that, in this example, the order in which the data cases arrive makes no difference to our beliefs about $\boldsymbol{\theta}$, since all orderings will have the same sufficient statistics. Hence we say the data is **exchangeable**.

To avoid visual clutter, it is common to use a form of **syntactic sugar** called **plates**: we simply draw a little box around the repeated variables, with the convention that nodes within the box will get repeated when the model is **unrolled**. We often write the number of copies or repetitions in the bottom right corner of the box. See Figure 10.7(b) for a simple example. The corresponding joint distribution has the form

$$p(\boldsymbol{\theta}, \mathcal{D}) = p(\boldsymbol{\theta}) \left[\prod_{i=1}^{N} p(\mathbf{x}_i | \boldsymbol{\theta}) \right] \tag{10.26}$$

This DGM represents the CI assumptions behind the models we considered in Chapter 5.

A slightly more complex example is shown in Figure 10.8. On the left we show a naive Bayes classifier that has been "unrolled" for D features, but uses a plate to represent repetition over cases $i = 1 : N$. The version on the right shows the same model using **nested plate** notation. When a variable is inside two plates, it will have two sub-indices. For example, we write θ_{jc} to represent the parameter for feature j in class-conditional density c. Note that plates can be nested or crossing. Notational devices for modeling more complex parameter tying patterns can be devised (e.g., (Heckerman et al. 2004)), but these are not widely used. What is not clear from the figure is that θ_{jc} is used to generate x_{ij} iff $y_i = c$, otherwise it is ignored. This is an example of **context specific independence**, since the CI relationship $x_{ij} \perp \theta_{jc}$ only holds if $y_i \neq c$.

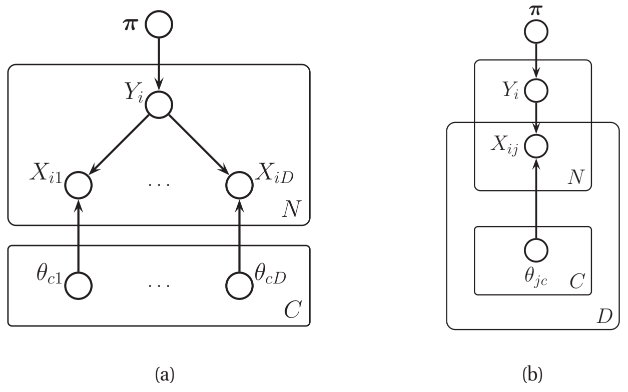


Figure 10.8 Naive Bayes classifier as a DGM. (a) With single plates. (b) WIth nested plates.

10.4.2 Learning from complete data

If all the variables are fully observed in each case, so there is no missing data and there are no hidden variables, we say the data is **complete**. For a DGM with complete data, the likelihood is given by

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{N} p(\mathbf{x}_i|\boldsymbol{\theta}) = \prod_{i=1}^{N}\prod_{t=1}^{V} p(x_{it}|\mathbf{x}_{i,\text{pa}(t)}, \boldsymbol{\theta}_t) = \prod_{t=1}^{V} p(\mathcal{D}_t|\boldsymbol{\theta}_t) \tag{10.27}$$

where $\mathcal{D}_t$ is the data associated with node t and its parents, i.e., the t'th family. This is a product of terms, one per CPD. We say that the likelihood **decomposes** according to the graph structure.

Now suppose that the prior factorizes as well:

$$p(\boldsymbol{\theta}) = \prod_{t=1}^{V} p(\boldsymbol{\theta}_t) \tag{10.28}$$

Then clearly the posterior also factorizes:

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}) = \prod_{t=1}^{V} p(\mathcal{D}_t|\boldsymbol{\theta}_t)p(\boldsymbol{\theta}_t) \tag{10.29}$$

This means we can compute the posterior of each CPD independently. In other words,

factored prior plus factored likelihood implies factored posterior (10.30)

Let us consider an example, where all CPDs are tabular, thus extending the earlier results of Secion 3.5.1.2, where discussed Bayesian naive Bayes. We have a separate row (i.e., a separate multinoulli distribution) for each **conditioning case**, i.e., for each combination of parent values, as in Table 10.2. Formally, we can write the t'th CPT as $x_t|\mathbf{x}_{\text{pa}(t)} = c \sim \text{Cat}(\boldsymbol{\theta}_{tc})$, where $\theta_{tck} \triangleq p(x_t = k|\mathbf{x}_{\text{pa}(t)} = c)$, for $k = 1 : K_t$, $c = 1 : C_t$ and $t = 1 : T$. Here K_t is the number of states for node t, $C_t \triangleq \prod_{s \in \text{pa}(t)} K_s$ is the number of parent combinations, and T is the number of nodes. Obviously $\sum_k \theta_{tck} = 1$ for each row of each CPT.

Let us put a separate Dirichlet prior on each row of each CPT, i.e., $\boldsymbol{\theta}_{tc} \sim \text{Dir}(\boldsymbol{\alpha}_{tc})$. Then we can compute the posterior by simply adding the pseudo counts to the empirical counts to get $\boldsymbol{\theta}_{tc}|\mathcal{D} \sim \text{Dir}(\mathbf{N}_{tc} + \boldsymbol{\alpha}_{tc})$, where N_{tck} is the number of times that node t is in state k while its parents are in state c:

$$N_{tck} \triangleq \sum_{i=1}^{N} \mathbb{I}(x_{i,t} = k, x_{i,\text{pa}(t)} = c) \tag{10.31}$$

From Equation 2.77, the mean of this distribution is given by the following:

$$\overline{\theta}_{tck} = \frac{N_{tck} + \alpha_{tck}}{\sum_{k'}(N_{tck'} + \alpha_{tck'})} \tag{10.32}$$

For example, consider the DGM in Figure 10.1(a). Suppose the training data consists of the following 5 cases:

x_1	x_2	x_3	x_4	x_5
0	0	1	0	0
0	1	1	1	1
1	1	0	1	0
0	1	1	0	0
0	1	1	1	0

Below we list all the sufficient statistics N_{tck}, and the posterior mean parameters $\overline{\theta}_{ick}$ under a Dirichlet prior with $\alpha_{ick} = 1$ (corresponding to add-one smoothing) for the $t = 4$ node:

x_2	x_3	$N_{tck=1}$	$N_{tck=0}$	$\overline{\theta}_{tck=1}$	$\overline{\theta}_{tck=0}$
0	0	0	0	1/2	1/2
1	0	1	0	2/3	1/3
0	1	0	1	1/3	2/3
1	1	2	1	3/5	2/5

It is easy to show that the MLE has the same form as Equation 10.32, except without the α_{tck} terms, i.e.,

$$\hat{\theta}_{tck} = \frac{N_{tck}}{\sum_{k'} N_{tck'}} \tag{10.33}$$

Of course, the MLE suffers from the zero-count problem discussed in Section 3.3.4.1, so it is important to use a prior to regularize the estimation problem.

10.4.3 Learning with missing and/or latent variables

If we have missing data and/or hidden variables, the likelihood no longer factorizes, and indeed it is no longer convex, as we explain in detail in Section 11.3. This means we will usually can only compute a locally optimal ML or MAP estimate. Bayesian inference of the parameters is even harder. We discuss suitable approximate inference techniques in later chapters.

10.5 Conditional independence properties of DGMs

At the heart of any graphical model is a set of conditional indepence (CI) assumptions. We write $\mathbf{x}_A \perp_G \mathbf{x}_B | \mathbf{x}_C$ if A is independent of B given C in the graph G, using the semantics to be defined below. Let $I(G)$ be the set of all such CI statements encoded by the graph.

We say that G is an **I-map** (independence map) for p, or that p is **Markov** wrt G, iff $I(G) \subseteq I(p)$, where $I(p)$ is the set of all CI statements that hold for distribution p. In other words, the graph is an I-map if it does not make any assertions of CI that are not true of the distribution. This allows us to use the graph as a safe proxy for p when reasoning about p's CI properties. This is helpful for designing algorithms that work for large classes of distributions, regardless of their specific numerical parameters $\boldsymbol{\theta}$.

Note that the fully connected graph is an I-map of all distributions, since it makes no CI assertions at all (since it is not missing any edges). We therefore say G is a **minimal I-map** of p if G is an I-map of p, and if there is no $G' \subseteq G$ which is an I-map of p.

It remains to specify how to determine if $\mathbf{x}_A \perp_G \mathbf{x}_B | \mathbf{x}_C$. Deriving these independencies for undirected graphs is easy (see Section 19.2), but the DAG situation is somewhat complicated, because of the need to respect the orientation of the directed edges. We give the details below.

10.5.1 d-separation and the Bayes Ball algorithm (global Markov properties)

First, we introduce some definitions. We say an *undirected path* P is **d-separated** by a set of nodes E (containing the evidence) iff at least one of the following conditions hold:

1. P contains a chain, $s \rightarrow m \rightarrow t$ or $s \leftarrow m \leftarrow t$, where $m \in E$
2. P contains a tent or fork, $s \swarrow^{m} \searrow t$, where $m \in E$
3. P contains a **collider** or **v-structure**, $s \searrow_{m} \swarrow t$, where m is not in E and nor is any descendant of m.

Next, we say that a *set of nodes* A is d-separated from a different set of nodes B given a third observed set E iff each undirected path from every node $a \in A$ to every node $b \in B$ is d-separated by E. Finally, we define the CI properties of a DAG as follows:

$$\mathbf{x}_A \perp_G \mathbf{x}_B | \mathbf{x}_E \iff \text{A is d-separated from B given E} \tag{10.34}$$

The **Bayes ball algorithm** (Shachter 1998) is a simple way to see if A is d-separated from B given E, based on the above definition. The idea is this. We "shade" all nodes in E, indicating that they are observed. We then place "balls" at each node in A, let them "bounce around" according to some rules, and then ask if any of the balls reach any of the nodes in B. The three main rules are shown in Figure 10.9. Notice that balls can travel opposite to edge directions. We see that a ball can pass through a chain, but not if it is shaded in the middle. Similarly, a ball can pass through a fork, but not if it is shaded in the middle. However, a ball cannot pass through a v-structure, unless it is shaded in the middle.

We can justify the 3 rules of Bayes ball as follows. First consider a chain structure $X \rightarrow Y \rightarrow Z$, which encodes

$$p(x, y, z) = p(x)p(y|x)p(z|y) \tag{10.35}$$

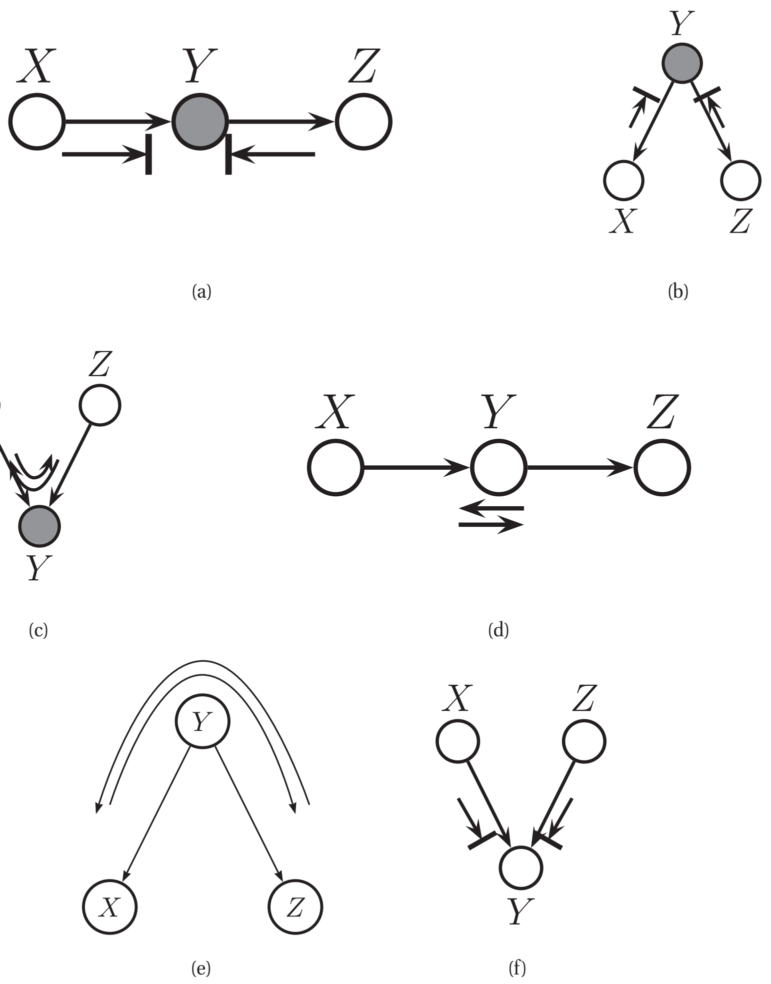


Figure 10.9 Bayes ball rules. A shaded node is one we condition on. If there is an arrow hitting a bar, it means the ball cannot pass through; otherwise the ball can pass through. Based on (Jordan 2007).

When we condition on y, are x and z independent? We have

$$p(x,z|y) = \frac{p(x)p(y|x)p(z|y)}{p(y)} = \frac{p(x,y)p(z|y)}{p(y)} = p(x|y)p(z|y) \tag{10.36}$$

and therefore $x \perp z|y$. So observing the middle node of chain breaks it in two (as in a Markov chain).

Now consider the tent structure $X \leftarrow Y \rightarrow Z$. The joint is

$$p(x,y,z) = p(y)p(x|y)p(z|y) \tag{10.37}$$

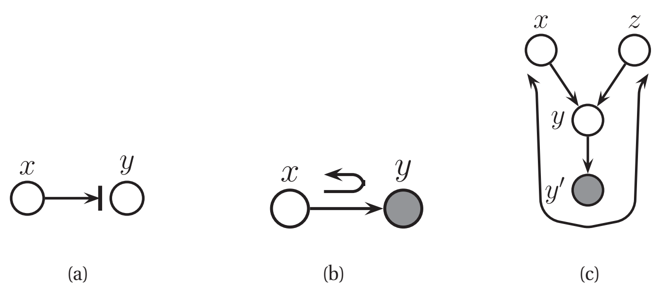


Figure 10.10 (a-b) Bayes ball boundary conditions. (c) Example of why we need boundary conditions. y' is an observed child of y, rendering y "effectively observed", so the ball bounces back up on its way from x to z.

When we condition on y, are x and z independent? We have

$$p(x,z|y) = \frac{p(x,y,z)}{p(y)} = \frac{p(y)p(x|y)p(z|y)}{p(y)} = p(x|y)p(z|y) \tag{10.38}$$

and therefore $x \perp z|y$. So observing a root node separates its children (as in a naive Bayes classifier: see Section 3.5).

Finally consider a v-structure $X \to Y \leftarrow Z$. The joint is

$$p(x,y,z) = p(x)p(z)p(y|x,z) \tag{10.39}$$

When we condition on y, are x and z independent? We have

$$p(x,z|y) = \frac{p(x)p(z)p(y|x,z)}{p(y)} \tag{10.40}$$

so $x \not\perp z|y$. However, in the unconditional distribution, we have

$$p(x,z) = p(x)p(z) \tag{10.41}$$

so we see that x and z are marginally independent. So we see that conditioning on a common child at the bottom of a v-structure makes its parents become dependent. This important effect is called **explaining away**, **inter-causal reasoning**, or **Berkson's paradox**. As an example of explaining away, suppose we toss two coins, representing the binary numbers 0 and 1, and we observe the "sum" of their values. A priori, the coins are independent, but once we observe their sum, they become coupled (e.g., if the sum is 1, and the first coin is 0, then we know the second coin is 1).

Finally, Bayes Ball also needs the "boundary conditions" shown in Figure 10.10(a-b). To understand where these rules come from, consider Figure 10.10(c). Suppose Y' is a noise-free copy of Y. Then if we observe Y', we effectively observe Y as well, so the parents X and Z have to compete to explain this. So if we send a ball down $X \to Y \to Y'$, it should "bounce back" up along $Y' \to Y \to Z$. However, if Y and all its children are hidden, the ball does not bounce back.

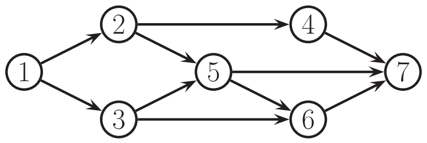


Figure 10.11 A DGM.

For example, in Figure 10.11, we see that $x_2 \perp x_6|x_5$, since the $2 \rightarrow 5 \rightarrow 6$ path is blocked by x_5 (which is observed), the $2 \rightarrow 4 \rightarrow 7 \rightarrow 6$ path is blocked by x_7 (which is hidden), and the $2 \rightarrow 1 \rightarrow 3 \rightarrow 6$ path is blocked by x_1 (which is hidden). However, we also see that $x_2 \not\perp x_6|x_5, x_7$, since now the $2 \rightarrow 4 \rightarrow 7 \rightarrow 6$ path is no longer blocked by x_7 (which is observed). Exercise 10.2 gives you some more practice in determining CI relationships for DGMs.

10.5.2 Other Markov properties of DGMs

From the d-separation criterion, one can conclude that

$$t \perp \text{nd}(t) \setminus \text{pa}(t)|\text{pa}(t) \tag{10.42}$$

where the **non-descendants** of a node $\text{nd}(t)$ are all the nodes except for its descendants, $\text{nd}(t) = \mathcal{V} \setminus \{t \cup \text{desc}(t)\}$. Equation 10.42 is called the **directed local Markov property**. For example, in Figure 10.11, we have $\text{nd}(3) = \{2, 4\}$, and $\text{pa}(3) = 1$, so $3 \perp 2, 4|1$.

A special case of this property is when we only look at predecessors of a node according to some topological ordering. We have

$$t \perp \text{pred}(t) \setminus \text{pa}(t)|\text{pa}(t) \tag{10.43}$$

which follows since $\text{pred}(t) \subseteq \text{nd}(t)$. This is called the **ordered Markov property**, which justifies Equation 10.7. For example, in Figure 10.11, if we use the ordering $1, 2, \ldots, 7$. we find $\text{pred}(3) = \{1, 2\}$ and $\text{pa}(3) = 1$, so $3 \perp 2|1$.

We have now described three Markov properties for DAGs: the directed global Markov property G in Equation 10.34, the ordered Markov property O in Equation 10.43, and the directed local Markov property L in Equation 10.42. It is obvious that $G \implies L \implies O$. What is less obvious, but nevertheless true, is that $O \implies L \implies G$ (see e.g., (Koller and Friedman 2009) for the proof). Hence all these properties are equivalent.

Furthermore, any distribution p that is Markov wrt G can be factorized as in Equation 10.7; this is called the factorization property F. It is obvious that $O \implies F$, but one can show that the converse also holds (see e.g., (Koller and Friedman 2009) for the proof).

10.5.3 Markov blanket and full conditionals

The set of nodes that renders a node t conditionally independent of all the other nodes in the graph is called t's **Markov blanket**; we will denote this by $\text{mb}(t)$. One can show that the Markov blanket of a node in a DGM is equal to the parents, the children, and the **co-parents**, i.e., other nodes who are also parents of its children:

$$\text{mb}(t) \triangleq \text{ch}(t) \cup \text{pa}(t) \cup \text{copa}(t) \tag{10.44}$$

For example, in Figure 10.11, we have

$$\text{mb}(5) = \{6, 7\} \cup \{2, 3\} \cup \{4\} = \{2, 3, 4, 6, 7\} \tag{10.45}$$

where 4 is a co-parent of 5 because they share a common child, namely 7.

To see why the co-parents are in the Markov blanket, note that when we derive $p(x_t|\mathbf{x}_{-t}) = p(x_t, \mathbf{x}_{-t})/p(\mathbf{x}_{-t})$, all the terms that do not involve x_t will cancel out between numerator and denominator, so we are left with a product of CPDs which contain x_t in their **scope**. Hence

$$p(x_t|\mathbf{x}_{-t}) \quad \propto \quad p(x_t|\mathbf{x}_{\text{pa}(t)}) \prod_{s \in \text{ch}(t)} p(x_s|\mathbf{x}_{\text{pa}(s)}) \tag{10.46}$$

For example, in Figure 10.11 we have

$$p(x_5|\mathbf{x}_{-5}) \propto p(x_5|x_2, x_3)p(x_6|x_3, x_5)p(x_7|x_4, x_5, x_6) \tag{10.47}$$

The resulting expression is called t's **full conditional**, and will prove to be important when we study Gibbs sampling (Section 24.2).

10.6 Influence (decision) diagrams *

We can represent multi-stage (Bayesian) decision problems by using a graphical notation known as a **decision diagram** or an **influence diagram** (Howard and Matheson 1981; Kjaerulff and Madsen 2008). This extends directed graphical models by adding **decision nodes** (also called **action nodes**), represented by rectangles, and **utility nodes** (also called **value nodes**), represented by diamonds. The original random variables are called **chance nodes**, and are represented by ovals, as usual.

Figure 10.12(a) gives a simple example, illustrating the famous **oil wild-catter** problem.[3] In this problem, you have to decide whether to drill an oil well or not. You have two possible actions: $d = 1$ means drill, $d = 0$ means don't drill. You assume there are 3 states of nature: $o = 0$ means the well is dry, $o = 1$ means it is wet (has some oil), and $o = 2$ means it is soaking (has a lot of oil). Suppose your prior beliefs are $p(o) = [0.5, 0.3, 0.2]$. Finally, you must specify the utility function $U(d, o)$. Since the states and actions are discrete, we can represent it as a table (analogous to a CPT in a DGM). Suppose we use the following numbers, in dollars:

	$o = 0$	$o = 1$	$o = 2$
$d = 0$	0	0	0
$d = 1$	-70	50	200

We see that if you don't drill, you incur no costs, but also make no money. If you drill a dry well, you lose $70; if you drill a wet well, you gain $50; and if you drill a soaking well, you gain $200. Your prior expected utility if you drill is given by

$$EU(d = 1) = \sum_{o=0}^{2} p(o)U(d, o) = 0.5 \cdot (-70) + 0.3 \cdot 50 + 0.2 \cdot 200 = 20 \tag{10.48}$$

3. This example is originally from (Raiffa 1968). Our presentation is based on some notes by Daphne Koller.

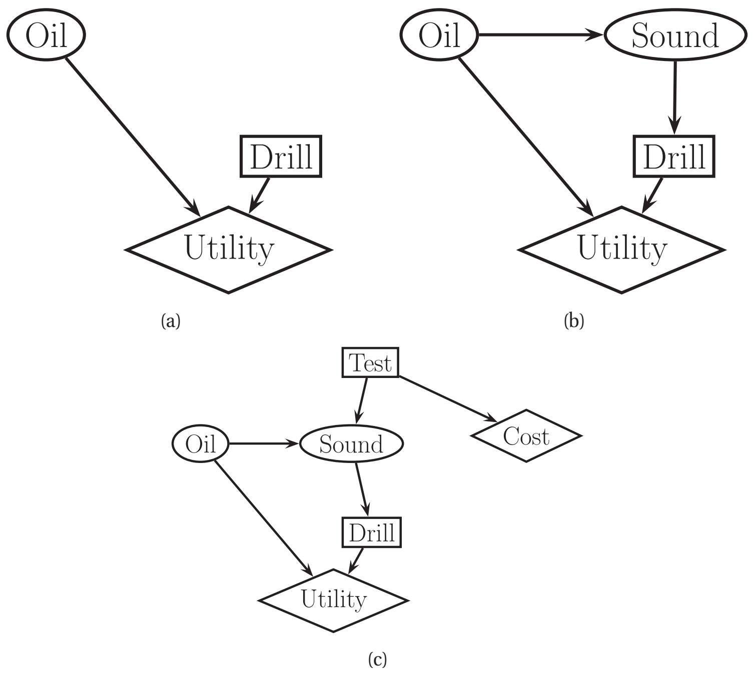


Figure 10.12 (a) Influence diagram for basic oil wild catter problem. (b) An extension in which we have an information arc from the Sound chance node to the Drill decision node. (c) An extension in which we get to decide whether to perform the test or not.

Your expected utility if you don't drill is 0. So your maximum expected utility is

$$MEU = \max\{EU(d = 0), EU(d = 1)\} = \max\{0, 20\} = 20 \tag{10.49}$$

and therefore the optimal action is to drill:

$$d^* = \arg\max\{EU(d = 0), EU(d = 1)\} = 1 \tag{10.50}$$

Now let us consider a slight extension to the model. Suppose you perform a sounding to estimate the state of the well. The sounding observation can be in one of 3 states: $s = 0$ is a diffuse reflection pattern, suggesting no oil; $s = 1$ is an open reflection pattern, suggesting some oil; and $s = 2$ is a closed reflection pattern, indicating lots of oil. Since S is caused by O, we add an $O \to S$ arc to our model. In addition, we assume that the outcome of the sounding test will be available before we decide whether to drill or not; hence we add an **information arc** from S to D. This is illustrated in Figure 10.12(b).

Let us model the reliability of our sensor using the following conditional distribution for $p(s|o)$:

	$s = 0$	$s = 1$	$s = 2$
$o = 0$	0.6	0.3	0.1
$o = 1$	0.3	0.4	0.3
$o = 2$	0.1	0.4	0.5

Suppose we do the sounding test and we observe $s = 0$. The posterior over the oil state is

$$p(o|s = 0) = [0.732, 0.219, 0.049] \tag{10.51}$$

Now your posterior expected utility of performing action d is

$$EU(d|s = 0) = \sum_{o=0}^{2} p(o|s = 0)U(o, d) \tag{10.52}$$

If $d = 1$, this gives

$$EU(d = 1|s = 0) = 0.732 \times (-70) + 0.219 \times 50 + 0.049 \times 200 = -30.5 \tag{10.53}$$

However, if $d = 0$, then $EU(d = 0|s = 0) = 0$, since not drilling incurs no cost. So if we observe $s = 0$, we are better off not drilling, which makes sense.

Now suppose we do the sounding test and we observe $s = 1$. By similar reasoning, one can show that $EU(d = 1|s = 1) = 32.9$, which is higher than $EU(d = 0|s = 1) = 0$. Similarly, if we observe $s = 2$, we have $EU(d = 1|s = 2) = 87.5$ which is much higher than $EU(d = 0|s = 2) = 0$. Hence the optimal policy $d^*(s)$ is as follows: if $s = 0$, choose $d^*(0) = 0$ and get \$0; if $s = 1$, choose $d^*(1) = 1$ and get \$32.9; and if $s = 2$, choose $d^*(2) = 1$ and get \$87.5.

You can compute your **expected profit** or maximum expected utility as follows:

$$MEU = \sum_{s} p(s)EU(d^*(s)|s) \tag{10.54}$$

This is the expected utility given possible outcomes of the sounding test, assuming you act optimally given the outcome. The prior marginal on the outcome of the test is

$$p(s) = \sum_{o} p(o)p(s|o) = [0.41, 0.35, 0.24] \tag{10.55}$$

Hence your maximum expected utility is

$$MEU = 0.41 \times 0 + 0.35 \times 32.9 + 0.24 \times 87.5 = 32.2 \tag{10.56}$$

Now suppose you can choose whether to do the test or not. This can be modelled as shown in Figure 10.12(c), where we add a new test node T. If $T = 1$, we do the test, and S can enter 1 of 3 states, determined by O, exactly as above. If $T = 0$, we don't do the test, and S enters a special unknown state. There is also some cost associated with performing the test.

Is it worth doing the test? This depends on how much our MEU changes if we know the outcome of the test (namely the state of S). If you don't do the test, we have $MEU = 20$ from Equation 10.49. If you do the test, you have $MEU = 32.2$ from Equation 10.56. So the improvement in utility if you do the test (and act optimally on its outcome) is \$12.2. This is

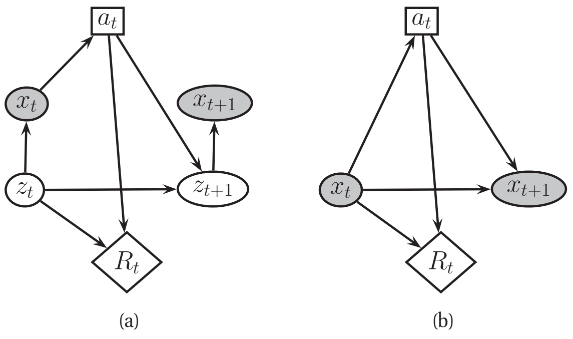


Figure 10.13 (a) A POMDP, shown as an influence diagram. z_t are hidden world states. We implicitly make the **no forgetting** assumption, which effectively means that a_t has arrows coming into it from all previous observations, $x_{1:t}$. (b) An MDP, shown as an influence diagram.

called the **value of perfect information** (VPI). So we should do the test as long as it costs less than $12.2.

In terms of graphical models, the VPI of a variable T can be determined by computing the MEU for the base influence diagram, I, and then computing the MEU for the same influence diagram where we add information arcs from T to the action nodes, and then computing the difference. In other words,

$$\text{VPI} = \text{MEU}(I + T \to D) - \text{MEU}(I) \tag{10.57}$$

where D is the decision node and T is the variable we are measuring.

It is possible to modify the variable elimination algorithm (Section 20.3) so that it computes the optimal policy given an influence diagram. These methods essentially work backwards from the final time-step, computing the optimal decision at each step assuming all following actions are chosen optimally. See e.g., (Lauritzen and Nilsson 2001; Kjaerulff and Madsen 2008) for details.

We could continue to extend the model in various ways. For example, we could imagine a dynamical system in which we test, observe outcomes, perform actions, move on to the next oil well, and continue drilling (and polluting) in this way. In fact, many problems in robotics, business, medicine, public policy, etc. can be usefully formulated as influence diagrams unrolled over time (Raiffa 1968; Lauritzen and Nilsson 2001; Kjaerulff and Madsen 2008).

A generic model of this form is shown in Figure 10.13(a). This is known as a **partially observed Markov decision process** or **POMDP** (pronounced "pom-d-p"). This is basically a hidden Markov model (Section 17.3) augmented with action and reward nodes. This can be used to model the **perception-action** cycle that all intelligent agents use (see e.g., (Kaelbling et al. 1998) for details).

A special case of a POMDP, in which the states are fully observed, is called a **Markov decision process** or **MDP**, shown in Figure 10.13(b). This is much easier to solve, since we only have to compute a mapping from observed states to actions. This can be solved using dynamic programming (see e.g., (Sutton and Barto 1998) for details).

In the POMDP case, the information arc from x_t to a_t is not sufficient to uniquely determine

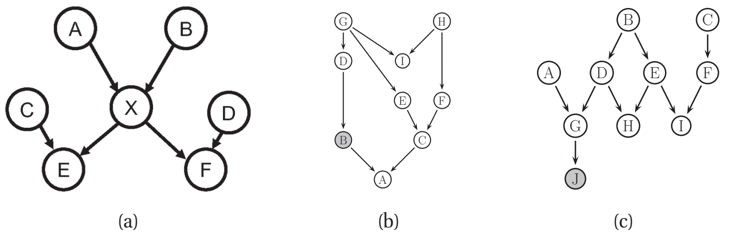


Figure 10.14 Some DGMs.

the best action, since the state is not fully observed. Instead, we need to choose actions based on our **belief state**, $p(z_t|\mathbf{x}_{1:t}, a_{1:t})$. Since the belief updating process is deterministic (see Section 17.4.2), we can compute a **belief state MDP**. For details on to compute the policies for such models, see e.g., (Kaelbling et al. 1998; Spaan and Vlassis 2005).

Exercises

Exercise 10.1 Marginalizing a node in a DGM

(Source: Koller.)

Consider the DAG G in Figure 10.14(a). Assume it is a minimal I-map for $p(A, B, C, D, E, F, X)$. Now consider marginalizing out X. Construct a new DAG G' which is a minimal I-map for $p(A, B, C, D, E, F)$. Specify (and justify) which extra edges need to be added.

Exercise 10.2 Bayes Ball

(Source: Jordan.)

Here we compute some global independence statements from some directed graphical models. You can use the "Bayes ball" algorithm, the d-separation criterion, or the method of converting to an undirected graph (all should give the same results).

a. Consider the DAG in Figure 10.14(b). List all variables that are independent of A given evidence on B.
b. Consider the DAG in Figure 10.14(c). List all variables that are independent of A given evidence on J.

Exercise 10.3 Markov blanket for a DGM

Prove that the full conditional for node i in a DGM is given by

$$p(X_i|X_{-i}) \quad \propto \quad p(X_i|Pa(X_i)) \prod_{Y_j \in ch(X_i)} p(Y_j|Pa(Y_j)) \tag{10.58}$$

where $ch(X_i)$ are the children of X_i and $Pa(Y_j)$ are the parents of Y_j.

Exercise 10.4 Hidden variables in DGMs

Consider the DGMs in Figure 11.1 which both define $p(X_{1:6})$, where we number empty nodes left to right, top to bottom. The graph on the left defines the joint as

$$p(X_{1:6}) = \sum_h p(X_1)p(X_2)p(X_3)p(H = h|X_{1:3})p(X_4|H = h)p(X_5|H = h)p(X_6|H = h) \tag{10.59}$$

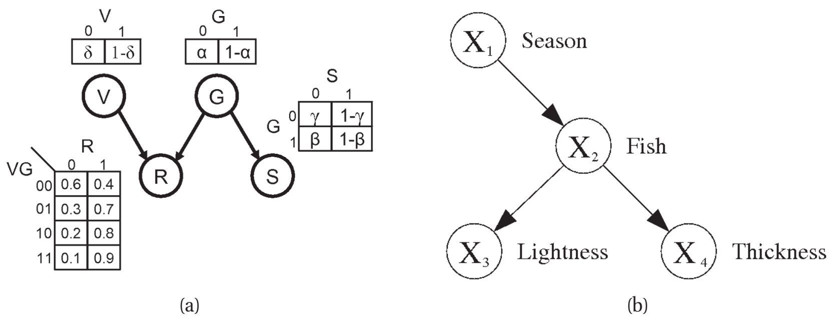


Figure 10.15 (a) Weather BN. (b) Fishing BN.

where we have marginalized over the hidden variable H. The graph on the right defines the joint as

$$p(X_{1:6}) = p(X_1)p(X_2)p(X_3)p(X_4|X_{1:3})p(X_5|X_{1:4})p(X_6|X_{1:5}) \tag{10.60}$$

a. (5 points) Assuming all nodes (including H) are binary and all CPDs are tabular, prove that the model on the left has 17 free parameters.
b. (5 points) Assuming all nodes are binary and all CPDs are tabular, prove that the model on the right has 59 free parameters.
c. (5 points) Suppose we have a data set $\mathcal{D} = X^n_{1:6}$ for $n = 1 : N$, where we observe the Xs but not H, and we want to estimate the parameters of the CPDs using maximum likelihood. For which model is this easier? Explain your answer.

Exercise 10.5 Bayes nets for a rainy day

(Source: Nando de Freitas.). In this question you must model a problem with 4 binary variables: $G =$"gray", $V =$"Vancouver", $R =$"rain" and $S =$"sad". Consider the directed graphical model describing the relationship between these variables shown in Figure 10.15(a).

a. Write down an expression for $P(S = 1|V = 1)$ in terms of $\alpha, \beta, \gamma, \delta$.
b. Write down an expression for $P(S = 1|V = 0)$. Is this the same or different to $P(S = 1|V = 1)$? Explain why.
c. Find maximum likelihood estimates of α, β, γ using the following data set, where each row is a training case. (You may state your answers without proof.)

V	G	R	S
1	1	1	1
1	1	0	1
1	0	0	0

(10.61)

Exercise 10.6 Fishing nets

(Source: (Duda et al. 2001)..) Consider the Bayes net shown in Figure 10.15(b). Here, the nodes represent the following variables

$$X_1 \in \{\text{winter, spring, summer, autumn}\},\ X_2 \in \{\text{salmon, sea bass}\} \tag{10.62}$$

$$X_3 \in \{\text{light, medium, dark}\},\ X_4 \in \{\text{wide, thin}\} \tag{10.63}$$

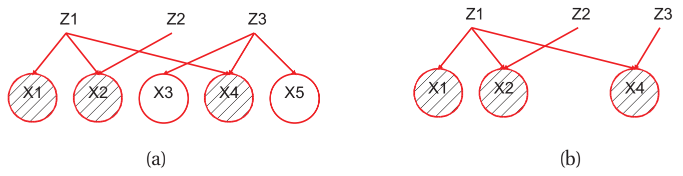


Figure 10.16 (a) A QMR-style network with some hidden leaves. (b) Removing the barren nodes.

The corresponding conditional probability tables are

$$p(x_1) = \begin{pmatrix} .25 & .25 & .25 & .25 \end{pmatrix}, \; p(x_2|x_1) = \begin{pmatrix} .9 & .1 \\ .3 & .7 \\ .4 & .6 \\ .8 & .2 \end{pmatrix} \tag{10.64}$$

$$p(x_3|x_2) = \begin{pmatrix} .33 & .33 & .34 \\ .8 & .1 & .1 \end{pmatrix}, \; p(x_4|x_2) = \begin{pmatrix} .4 & .6 \\ .95 & .05 \end{pmatrix} \tag{10.65}$$

Note that in $p(x_4|x_2)$, the rows represent x_2 and the columns x_4 (so each row sums to one and represents the child of the CPD). Thus $p(x_4 = \text{thin}|x_2 = \text{sea bass}) = 0.05$, $p(x_4 = \text{thin}|x_2 = \text{salmon}) = 0.6$, etc.

Answer the following queries. You may use matlab or do it by hand. In either case, show your work.

a. Suppose the fish was caught on December 20 — the end of autumn and the beginning of winter — and thus let $p(x_1) = (.5, \ 0, \ 0, \ .5)$ instead of the above prior. (This is called **soft evidence**, since we do not know the exact value of X_1, but we have a distribution over it.) Suppose the lightness has not been measured but it is known that the fish is thin. Classify the fish as salmon or sea bass.

b. Suppose all we know is that the fish is thin and medium lightness. What season is it now, most likely? Use $p(x_1) = \begin{pmatrix} .25 & .25 & .25 & .25 \end{pmatrix}$

Exercise 10.7 Removing leaves in BN20 networks

a. Consider the QMR network, where only some of the symtpoms are observed. For example, in Figure 10.16(a), X_4 and X_5 are hidden. Show that we can safely remove all the hidden leaf nodes without affecting the posterior over the disease nodes, i.e., prove that we can compute $p(\mathbf{z}_{1:3}|x_1, x_2, x_4)$ using the network in Figure 10.16(b). This is called barren node removal, and can be applied to any DGM.

b. Now suppose we partition the leaves into three groups: on, off and unknown. Clearly we can remove the unknown leaves, since they are hidden and do not affect their parents. Show that we can analytically remove the leaves that are in the "off state", by absorbing their effect into the prior of the parents. (This trick only works for noisy-OR CPDs.)

Exercise 10.8 Handling negative findings in the QMR network

Consider the QMR network. Let $\mathbf{d}$ be the hidden diseases, $\mathbf{f}^-$ be the negative findings (leaf nodes that are off), and $\mathbf{f}^-$ be the positive findings (leaf nodes that are on). We can compute the posterior $p(\mathbf{d}|\mathbf{f}\text{'}\mathbf{f}^+)$ in two steps: first absorb the negative findings, $p(\mathbf{d}|\mathbf{f}^-) \propto p(\mathbf{d})p(\mathbf{f}^-|\mathbf{d})$, then absorb the positive findings, $p(\mathbf{d}|\mathbf{f}^-, \mathbf{f}^+) \propto p(\mathbf{d}|\mathbf{f}^-)p(\mathbf{f}^+|\mathbf{d})$. Show that the first step can be done in $O(|\mathbf{d}||\mathbf{f}^-|)$ time, where $|\mathbf{d}|$ is the number of dieases and $|\mathbf{f}^-|$ is the number of negative findings. For simplicity, you can ignore leak nodes. (Intuitively, the reason for this is that there is no correlation induced amongst the parents when the finding is off, since there is no explaining away.)

Exercise 10.9 Moralization does not introduce new independence statements

Recall that the process of moralizing a DAG means connecting together all “unmarried” parents that share a common child, and then dropping all the arrows. Let M be the moralization of DAG G. Show that $CI(M) \subseteq CI(G)$, where CI are the set of conditional independence statements implied by the model.

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 $p(\mathbf{x}_i|z_i = k) = p_k(\mathbf{x}_i)$,

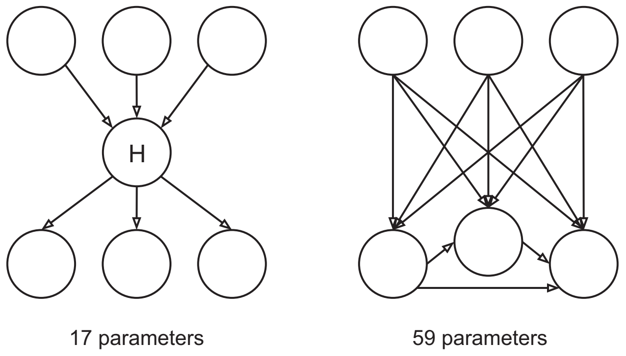


Figure 11.1 A DGM with and without hidden variables. The leaves represent medical symptoms. The roots represent primary causes, such as smoking, diet and exercise. The hidden variable can represent mediating factors, such as heart disease, which might not be directly visible.

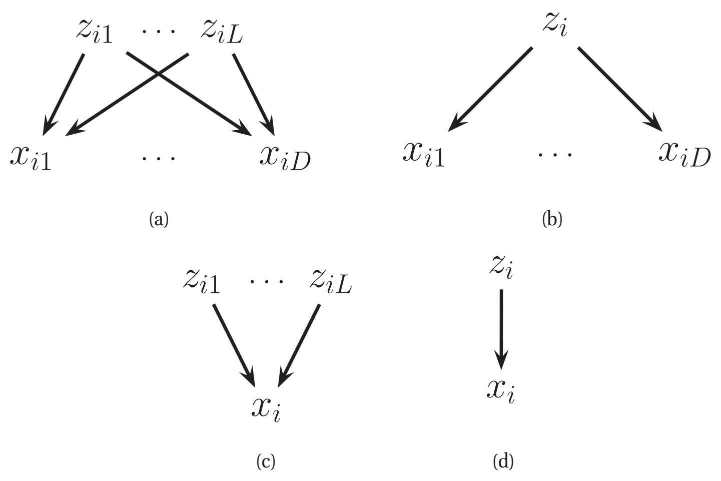


Figure 11.2 A latent variable model represented as a DGM. (a) Many-to-many. (b) One-to-many. (c) Many-to-one. (d) One-to-one.

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".

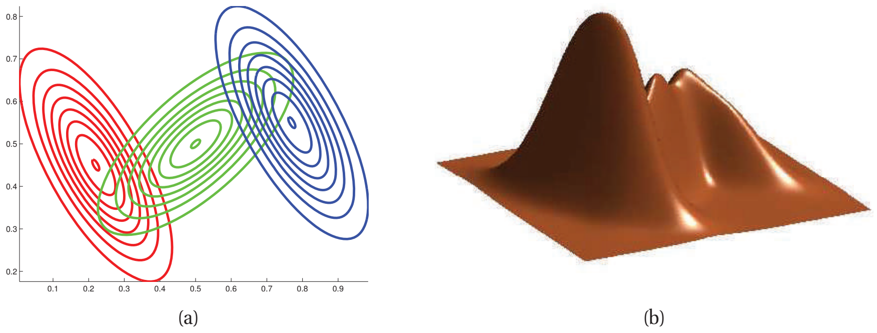


Figure 11.3 A mixture of 3 Gaussians in 2d. (a) We show the contours of constant probability for each component in the mixture. (b) A surface plot of the overall density. Based on Figure 2.23 of (Bishop 2006a). Figure generated by `mixGaussPlotDemo`.

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

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

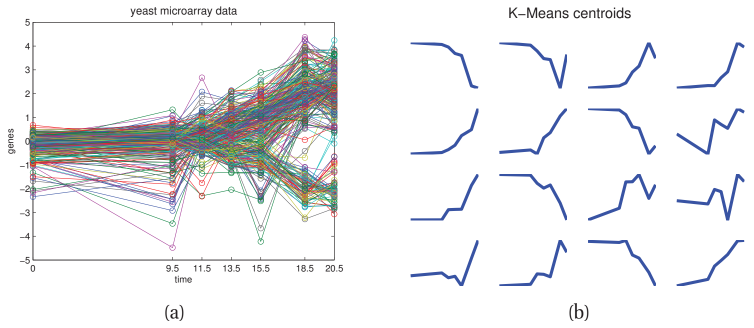


Figure 11.4 (a) Some yeast gene expression data plotted as a time series. (c) Visualizing the 16 cluster centers produced by K-means. Figure generated by `kmeansYeastDemo`.

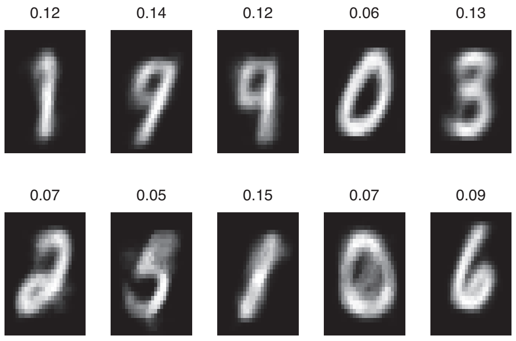


Figure 11.5 We fit a mixture of 10 Bernoullis to the binarized MNIST digit data. We show the MLE for the corresponding cluster means, $\boldsymbol{\mu}_k$. The numbers on top of each image represent the mixing weights $\hat{\pi}_k$. No labels were used when training the model. Figure generated by `mixBerMnistEM`.

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:

$$\begin{aligned} p(y_i|\mathbf{x}_i, z_i = k, \boldsymbol{\theta}) &= \mathcal{N}(y_i|\mathbf{w}_k^T\mathbf{x}_i, \sigma_k^2) && (11.8) \\ p(z_i|\mathbf{x}_i, \boldsymbol{\theta}) &= \text{Cat}(z_i|\mathcal{S}(\mathbf{V}^T\mathbf{x}_i)) && (11.9) \end{aligned}$$

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

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

We discuss how to fit this model in Section 11.4.3.

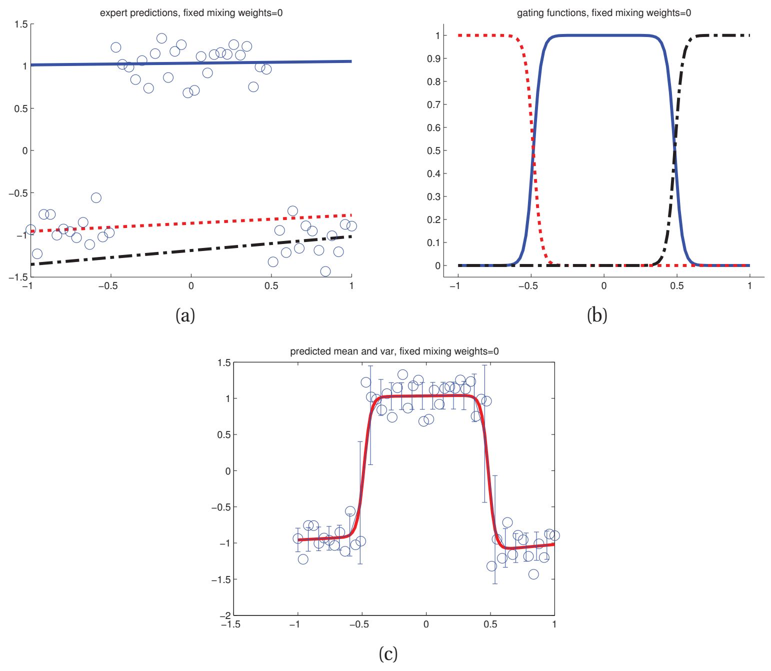


Figure 11.6 (a) Some data fit with three separate regression lines. (b) Gating functions for three different "experts". (c) The conditionally weighted average of the three expert predictions. Figure generated by `mixexpDemo`.

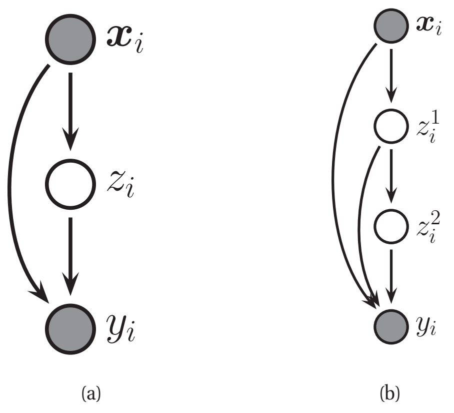


Figure 11.7 (a) A mixture of experts. (b) A hierarchical mixture of experts.

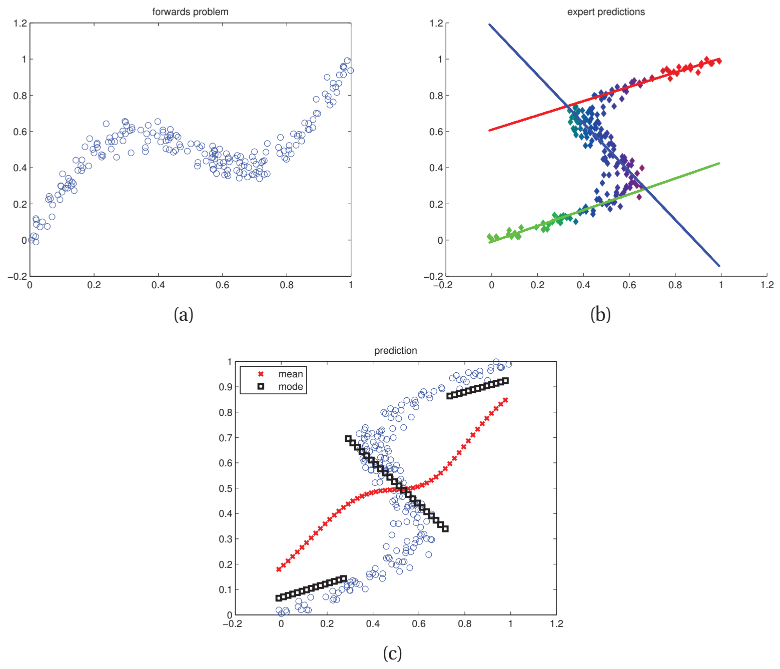


Figure 11.8 (a) Some data from a simple forwards model. (b) Some data from the inverse model, fit with a mixture of 3 linear regressions. Training points are color coded by their responsibilities. (c) The predictive mean (red cross) and mode (black square). Based on Figures 5.20 and 5.21 of (Bishop 2006b).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

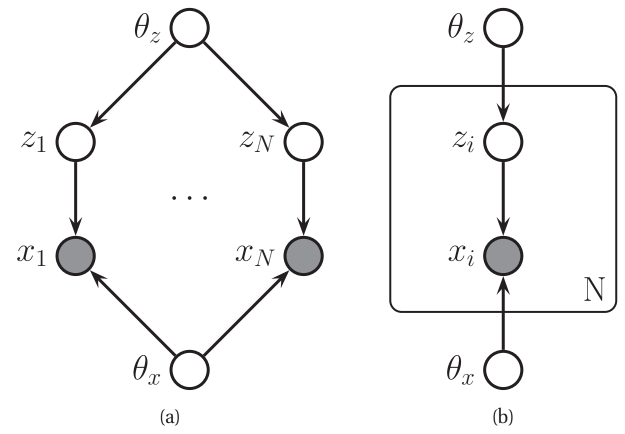


Figure 11.9 A LVM represented as a DGM. Left: Model is unrolled for N examples. Right: same model using plate notation.

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

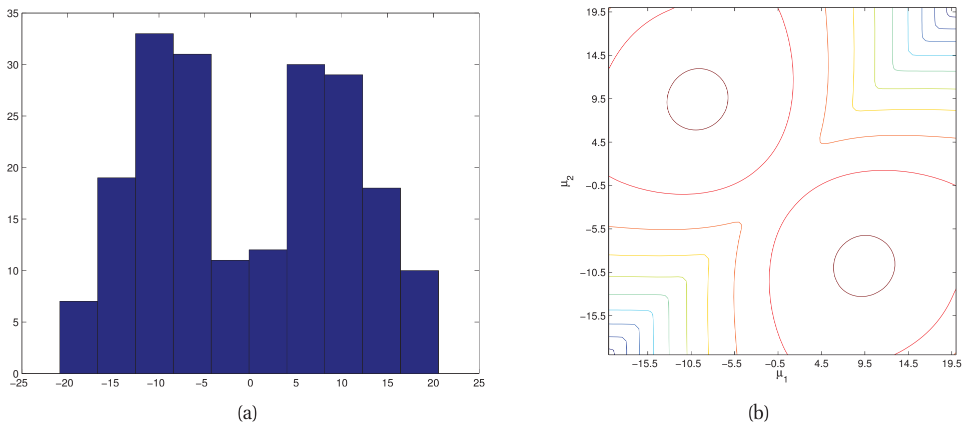


Figure 11.10 Left: $N = 200$ data points sampled from a mixture of 2 Gaussians in 1d, with $\pi_k = 0.5$, $\sigma_k = 5$, $\mu_1 = -10$ and $\mu_2 = 10$. Right: Likelihood surface $p(\mathcal{D}|\mu_1, \mu_2)$, with all other parameters set to their true values. We see the two symmetric modes, reflecting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

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

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:

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

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

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t-1)}) \triangleq \mathbb{E}\left[\sum_i \log p(\mathbf{x}_i, z_i|\boldsymbol{\theta})\right] \tag{11.22}$$

$$= \sum_i \mathbb{E}\left[\log\left[\prod_{k=1}^K (\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k))^{\mathbb{I}(z_i=k)}\right]\right] \tag{11.23}$$

$$= \sum_i \sum_k \mathbb{E}\left[\mathbb{I}(z_i = k)\right] \log[\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k)] \tag{11.24}$$

$$= \sum_i \sum_k p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}^{t-1}) \log[\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k)] \tag{11.25}$$

$$= \sum_i \sum_k r_{ik} \log \pi_k + \sum_i \sum_k r_{ik} \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) \tag{11.26}$$

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

$$\ell(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \sum_k \sum_i r_{ik} \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) \tag{11.29}$$

$$= -\frac{1}{2}\sum_i r_{ik}\left[\log|\boldsymbol{\Sigma}_k| + (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)\right] \tag{11.30}$$

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

$$\boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\mathbf{x}_i}{r_k} \tag{11.31}$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{r_k} = \frac{\sum_i r_{ik}\mathbf{x}_i\mathbf{x}_i^T}{r_k} - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T \tag{11.32}$$

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.

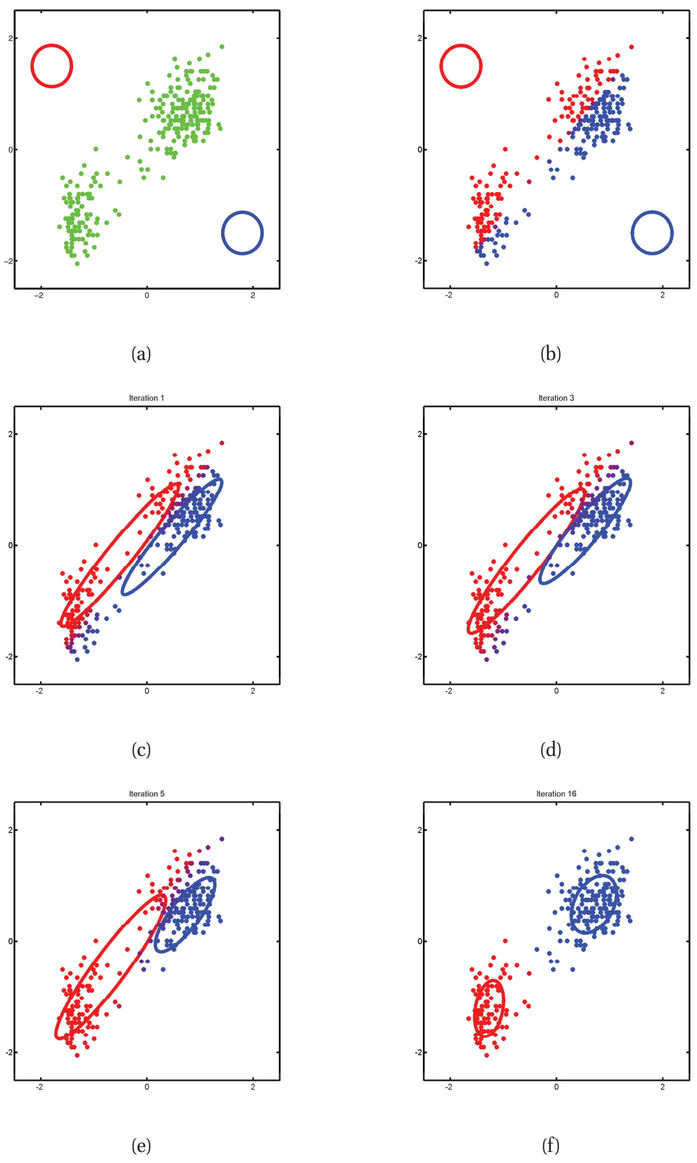
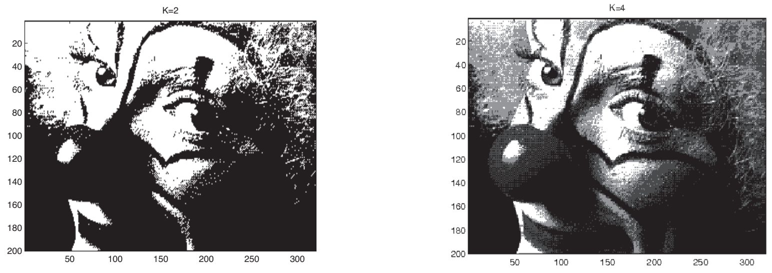


Figure 11.11 Illustration of the EM for a GMM applied to the Old Faithful data. (a) Initial (random) values of the parameters. (b) Posterior responsibility of each point computed in the first E step. The degree of redness indicates the degree to which the point belongs to the red cluster, and similarly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

```
1 initialize $\mathbf{m}_k$;
2 repeat
3     Assign each data point to its closest cluster center: $z_i = \arg\min_k ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$;
4     Update each cluster center by computing the mean of all points assigned to it:
      $\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_{i:z_i=k} \mathbf{x}_i$;
5 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 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:

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

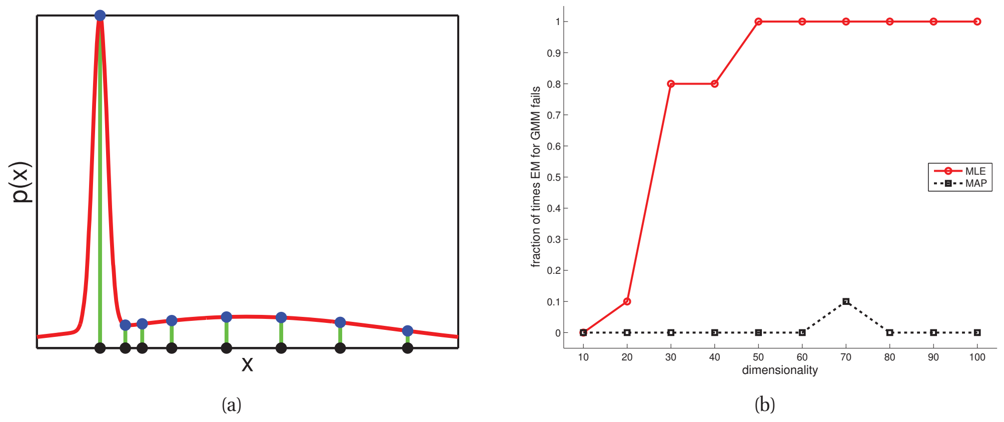


Figure 11.13 (a) Illustration of how singularities can arise in the likelihood function of GMMs. Based on (Bishop 2006a) Figure 9.7. Figure generated by `mixGaussSingularity`. (b) Illustration of the benefit of MAP estimation vs ML estimation when fitting a Gaussian mixture model. We plot the fraction 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

$$\begin{aligned} \ell_c(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \left[\log \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma}/z_i) + \log \text{Ga}(z_i|\nu/2, \nu/2)\right] & (11.62) \\ &= \sum_{i=1}^{N} \Big[-\frac{D}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{z_i}{2}\delta_i + \frac{\nu}{2}\log\frac{\nu}{2} - \log\Gamma(\frac{\nu}{2}) & (11.63) \\ &\quad + \frac{\nu}{2}(\log z_i - z_i) + (\frac{D}{2} - 1)\log z_i \Big] & (11.64) \end{aligned}$$

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

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

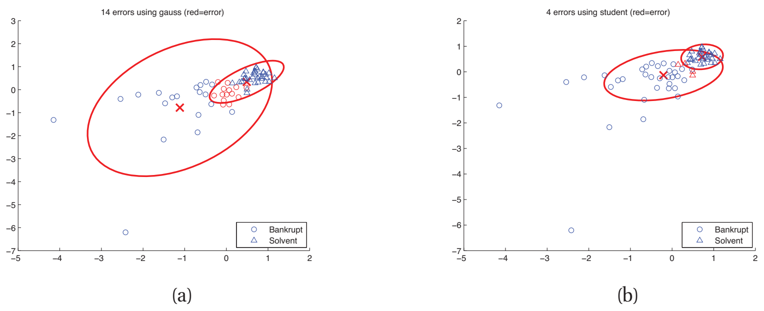


Figure 11.14 Mixture modeling on the bankruptcy data set. Left: Gaussian class conditional densities. Right: Student class conditional densities. Points that belong to class 1 are shown as triangles, points that belong to class 2 are shown as circles The estimated labels, based on the posterior probability 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

$$\begin{aligned} \overline{\ell}_i^{(t)} &= \Psi(\frac{\nu^{(t)} + D}{2}) - \log(\frac{\nu^{(t)} + \delta_i^{(t)}}{2}) & (11.75) \\ &= \log(\overline{z}_i^{(t)}) + \Psi(\frac{\nu^{(t)} + D}{2}) - \log(\frac{\nu^{(t)} + D}{2}) & (11.76) \end{aligned}$$

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**:

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

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

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

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

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

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.

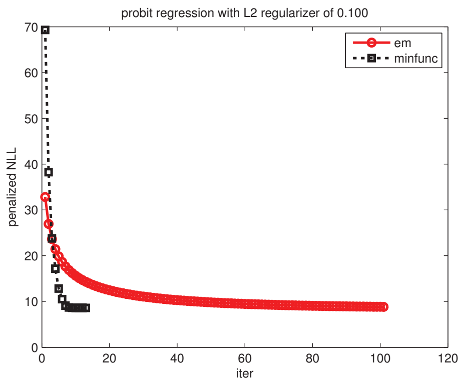


Figure 11.15 Fitting a probit regression model in 2d using a quasi-Newton method or EM. Figure generated by `probitRegDemo`.

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

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

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

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

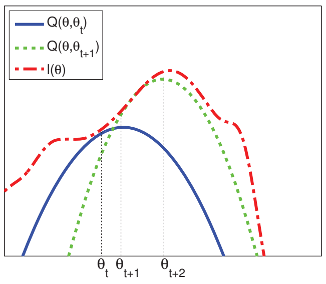


Figure 11.16 Illustration of EM as a bound optimization algorithm. Based on Figure 9.14 of (Bishop 2006a). Figure generated by `emLogLikelihoodMax`.

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

```
1 initialize $\boldsymbol{\mu}$;
2 repeat
3     $\boldsymbol{\mu}^{new} = \mathbf{0}$ ;
4     for each example $i = 1 : N$ do
5         $\mathbf{s}_i := \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}_i, \boldsymbol{\theta}(\boldsymbol{\mu}))\boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z})$ ;
6         $\boldsymbol{\mu}^{new} := \boldsymbol{\mu}^{new} + \mathbf{s}_i$; ;
7     $\boldsymbol{\mu} := \boldsymbol{\mu}^{new}$;
8 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

```
1 initialize $\mathbf{s}_i$ for $i = 1 : N$;
2 $\boldsymbol{\mu} = \sum_i \mathbf{s}_i$;
3 repeat
4     for each example $i = 1 : N$ in a random order do
5         $\mathbf{s}_i^{new} := \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}_i, \boldsymbol{\theta}(\boldsymbol{\mu}))\boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z})$ ;
6         $\boldsymbol{\mu} := \boldsymbol{\mu} + \mathbf{s}_i^{new} - \mathbf{s}_i$;
7         $\mathbf{s}_i := \mathbf{s}_i^{new}$;
8 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.

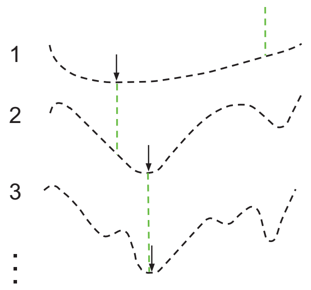


Figure 11.17 Illustration of deterministic annealing. Based on `http://en.wikipedia.org/wiki/Graduated_optimization`.

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

Algorithm 11.4: Stepwise EM algorithm

```
1 initialize $\boldsymbol{\mu}$; $k = 0$ ;
2 repeat
3     for each example $i = 1 : N$ in a random order do
4         $\mathbf{s}_i := \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}_i, \boldsymbol{\theta}(\boldsymbol{\mu}))\boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z})$ ;
5         $\boldsymbol{\mu} := (1 - \eta_k)\boldsymbol{\mu} + \eta_k \mathbf{s}_i$;
6         $k := k + 1$
7 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

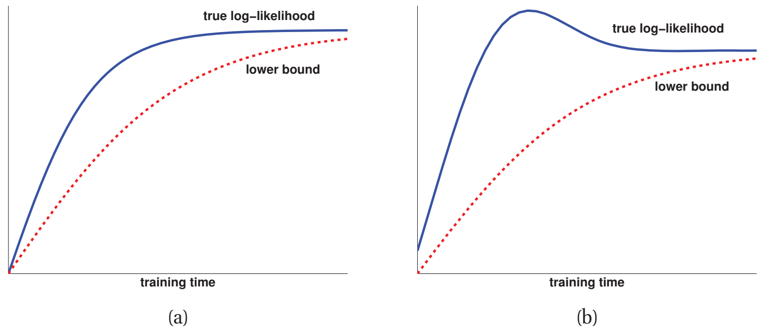


Figure 11.18 Illustration of possible behaviors of variational EM. (a) The lower bound increases at each iteration, and so does the likelihood. (b) The lower bound increases but the likelihood decreases. In this case, the algorithm is closing the gap between the approximate and true posterior. This can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

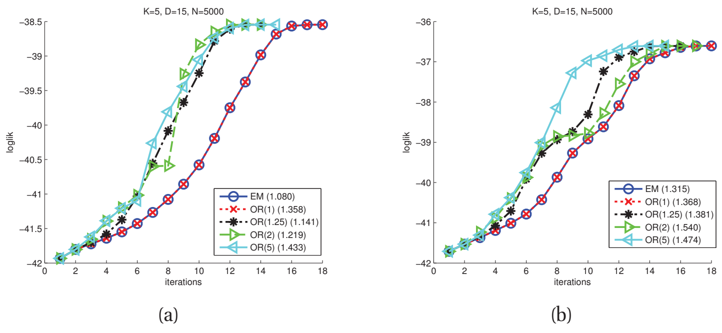


Figure 11.19 Illustration of adaptive over-relaxed EM applied to a mixture of 5 Gaussians in 15 dimensions. We show the algorithm applied to two different datasets, randomly sampled from a mixture of 10 Gaussians. We plot the convergence for different update rates η. Using $\eta = 1$ gives the same results 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

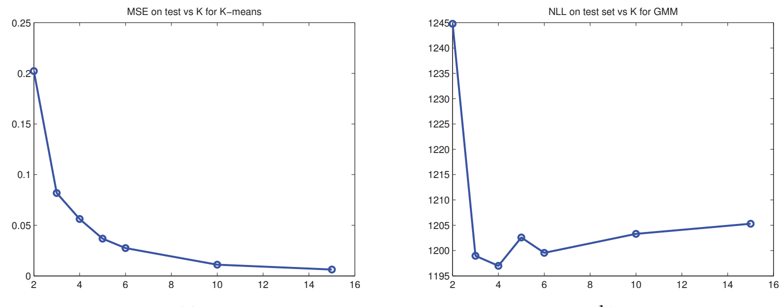
What if we are not using a probabilistic model? For example, how do we choose K for the K-means algorithm? Since this does not correspond to a probability model, there is no likelihood, so none of the methods described above can be used.

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:

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

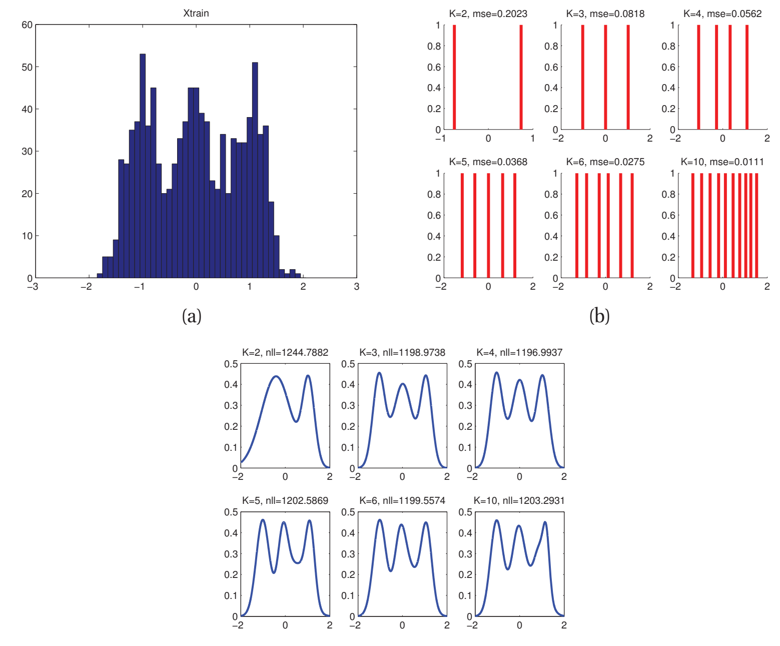
In the case of K-means, the reconstruction is given by $\hat{\mathbf{x}}_i = \boldsymbol{\mu}_{z_i}$, where $z_i = \text{argmin}_k \, ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$, as explained in Section 11.4.2.6.

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:

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

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

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

Hence the expected sufficient statistics are

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

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 $\text{cov}[\mathbf{x}] = \mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right] - \mathbb{E}[\mathbf{x}]\,\mathbb{E}\left[\mathbf{x}^T\right]$. Hence

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

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

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

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.

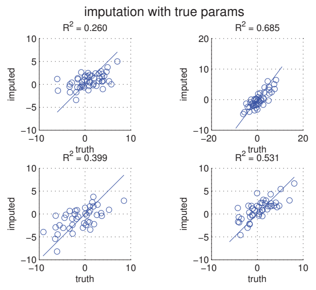
11.6.1.5 Extension to the GMM case

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)

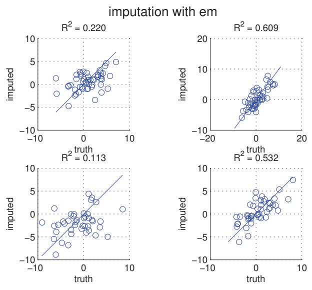

(b)

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

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

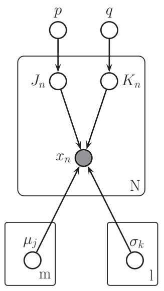


Figure 11.23 A mixture of Gaussians with two discrete latent indicators. J_n specifies which mean to use, and K_n specifies which variance to use.

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

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

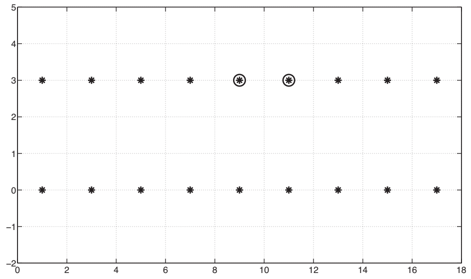


Figure 11.24 Some data points in 2d. Circles represent the initial guesses for $\mathbf{m}_1$ and $\mathbf{m}_2$.

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,

$$\sum_i (x_i - \mu)^2 = \sum_i [(x_i - \overline{x}) - (\mu - \overline{x})]^2 \tag{11.132}$$

$$= \sum_i (x_i - \overline{x})^2 + \sum_i (\overline{x} - \mu)^2 - 2\sum_i (x_i - \overline{x})(\mu - \overline{x}) \tag{11.133}$$

$$= ns^2 + n(\overline{x} - \mu)^2 \tag{11.134}$$

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.

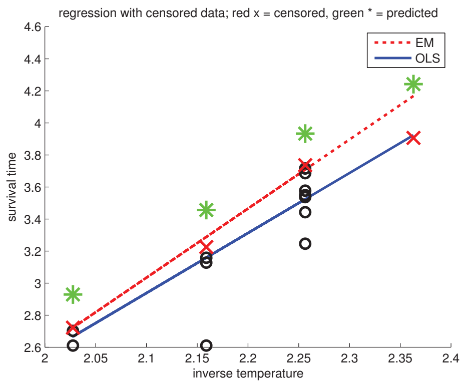


Figure 11.25 Example of censored linear regression. Black circles are observed training points, red crosses are observed but censored training points. Green stars are predicted values of the censored training points. We also show the lines fit by least squares (ignoring censoring) and by EM. Based on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

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

and

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

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

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

12 *Latent linear models*

12.1 Factor analysis

One problem with mixture models is that they only use a single latent variable to generate the observations. In particular, each observation can only come from one of K prototypes. One can think of a mixture model as using K hidden binary variables, representing a one-hot encoding of the cluster identity. But because these variables are mutually exclusive, the model is still limited in its representational power.

An alternative is to use a vector of real-valued latent variables, $\mathbf{z}_i \in \mathbb{R}^L$. The simplest prior to use is a Gaussian (we will consider other choices later):

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \tag{12.1}$$

If the observations are also continuous, so $\mathbf{x}_i \in \mathbb{R}^D$, we may use a Gaussian for the likelihood. Just as in linear regression, we will assume the mean is a linear function of the (hidden) inputs, thus yielding

$$p(\mathbf{x}_i|\mathbf{z}_i, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi}) \tag{12.2}$$

where $\mathbf{W}$ is a $D \times L$ matrix, known as the **factor loading matrix**, and $\boldsymbol{\Psi}$ is a $D \times D$ covariance matrix. We take $\boldsymbol{\Psi}$ to be diagonal, since the whole point of the model is to "force" $\mathbf{z}_i$ to explain the correlation, rather than "baking it in" to the observation's covariance. This overall model is called **factor analysis** or **FA**. The special case in which $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$ is called **probabilistic principal components analysis** or **PPCA**. The reason for this name will become apparent later.

The generative process, where $L = 1$, $D = 2$ and $\boldsymbol{\Psi}$ is diagonal, is illustrated in Figure 12.1. We take an isotropic Gaussian "spray can" and slide it along the 1d line defined by $\mathbf{w}z_i + \mu$. This induces an ellongated (and hence correlated) Gaussian in 2d.

12.1.1 FA is a low rank parameterization of an MVN

FA can be thought of as a way of specifying a joint density model on $\mathbf{x}$ using a small number of parameters. To see this, note that from Equation 4.126, the induced marginal distribution $p(\mathbf{x}_i|\boldsymbol{\theta})$ is a Gaussian:

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{x}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi})\mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)d\mathbf{z}_i \tag{12.3}$$

$$= \mathcal{N}(\mathbf{x}_i|\mathbf{W}\boldsymbol{\mu}_0 + \boldsymbol{\mu}, \boldsymbol{\Psi} + \mathbf{W}\boldsymbol{\Sigma}_0\mathbf{W}^T) \tag{12.4}$$

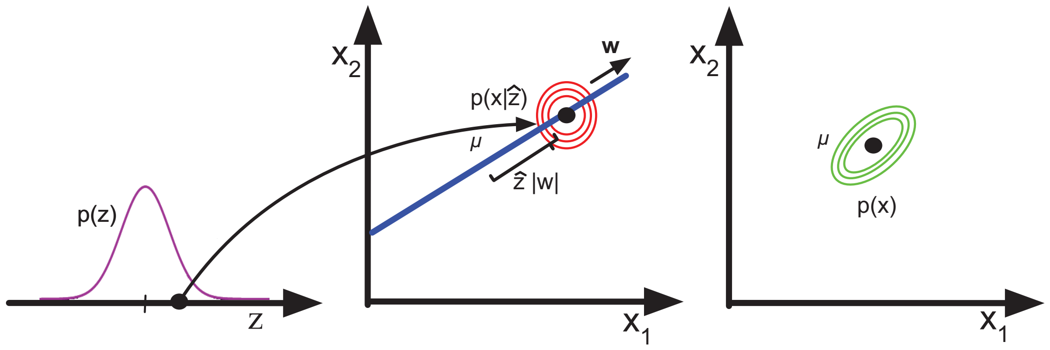


Figure 12.1 Illustration of the PPCA generative process, where we have $L = 1$ latent dimension generating $D = 2$ observed dimensions. Based on Figure 12.9 of (Bishop 2006b).

From this, we see that we can set $\boldsymbol{\mu}_0 = \mathbf{0}$ without loss of generality, since we can always absorb $\mathbf{W}\boldsymbol{\mu}_0$ into $\boldsymbol{\mu}$. Similarly, we can set $\boldsymbol{\Sigma}_0 = \mathbf{I}$ without loss of generality, because we can always "emulate" a correlated prior by using defining a new weight matrix, $\tilde{\mathbf{W}} = \mathbf{W}\boldsymbol{\Sigma}_0^{-\frac{1}{2}}$. Then we find

$$\text{cov}\left[\mathbf{x}|\boldsymbol{\theta}\right] = \tilde{\mathbf{W}}^T + \mathbb{E}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\right] = (\mathbf{W}\boldsymbol{\Sigma}_0^{-\frac{1}{2}})\boldsymbol{\Sigma}_0(\mathbf{W}\boldsymbol{\Sigma}_0^{-\frac{1}{2}})^T + \boldsymbol{\Psi} = \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi} \tag{12.5}$$

We thus see that FA approximates the covariance matrix of the visible vector using a low-rank decomposition:

$$\mathbf{C} \triangleq \text{cov}\left[\mathbf{x}\right] = \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi} \tag{12.6}$$

This only uses $O(LD)$ parameters, which allows a flexible compromise between a full covariance Gaussian, with $O(D^2)$ parameters, and a diagonal covariance, with $O(D)$ parameters. Note that if we did not restrict $\boldsymbol{\Psi}$ to be diagonal, we could trivially set $\boldsymbol{\Psi}$ to a full covariance matrix; then we could set $\mathbf{W} = \mathbf{0}$, in which case the latent factors would not be required.

12.1.2 Inference of the latent factors

Although FA can be thought of as just a way to define a density on $\mathbf{x}$, it is often used because we hope that the latent factors $\mathbf{z}$ will reveal something interesting about the data. To do this, we need to compute the posterior over the latent factors. We can use Bayes rule for Gaussians to give

$$p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_i|\mathbf{m}_i, \boldsymbol{\Sigma}_i) \tag{12.7}$$

$$\boldsymbol{\Sigma}_i \triangleq (\boldsymbol{\Sigma}_0^{-1} + \mathbf{W}^T\boldsymbol{\Psi}^{-1}\mathbf{W})^{-1} \tag{12.8}$$

$$\mathbf{m}_i \triangleq \boldsymbol{\Sigma}_i(\mathbf{W}^T\boldsymbol{\Psi}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}) + \boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0) \tag{12.9}$$

Note that in the FA model, $\boldsymbol{\Sigma}_i$ is actually independent of i, so we can denote it by $\boldsymbol{\Sigma}$. Computing this matrix takes $O(L^3 + L^2 D)$ time, and computing each $\mathbf{m}_i = \mathbb{E}\left[\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta}\right]$ takes $O(L^2 + LD)$ time. The $\mathbf{m}_i$ are sometimes called the latent **scores**, or latent **factors**.

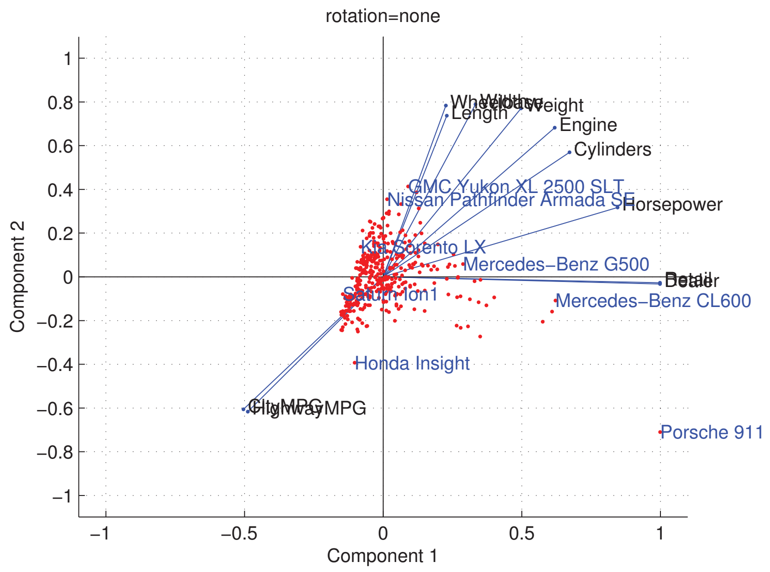


Figure 12.2 2D projection of 2004 cars data based on factor analysis. The blue text are the names of cars corresponding to certain chosen points. Figure generated by `faBiplotDemo`.

Let us give a simple example, based (Shalizi 2009). We consider a dataset of $D = 11$ variables and $N = 387$ cases describing various aspects of cars, such as the engine size, the number of cylinders, the miles per gallon (MPG), the price, etc. We first fit a $L = 2$ dimensional model. We can plot the $\mathbf{m}_i$ scores as points in $\mathbb{R}^2$, to visualize the data, as shown in Figure 12.2.

To get a better understanding of the "meaning" of the latent factors, we can project unit vectors corresponding to each of the feature dimensions, $\mathbf{e}_1 = (1, 0, \ldots, 0)$, $\mathbf{e}_2 = (0, 1, 0, \ldots, 0)$, etc. into the low dimensional space. These are shown as blue lines in Figure 12.2; this is known as a **biplot**. We see that the horizontal axis represents price, corresponding to the features labeled "dealer" and "retail", with expensive cars on the right. The vertical axis represents fuel efficiency (measured in terms of MPG) versus size: heavy vehicles are less efficient and are higher up, whereas light vehicles are more efficient and are lower down. We can "verify" this interpretation by clicking on some points, and finding the closest exemplars in the training set, and printing their names, as in Figure 12.2. However, in general, interpreting latent variable models is fraught with difficulties, as we discuss in Section 12.1.3.

12.1.3 Unidentifiability

Just like with mixture models, FA is also unidentifiable. To see this, suppose $\mathbf{R}$ is an arbitrary orthogonal rotation matrix, satisfying $\mathbf{R}\mathbf{R}^T = \mathbf{I}$. Let us define $\tilde{\mathbf{W}} = \mathbf{W}\mathbf{R}$; then the likelihood function of this modified matrix is the same as for the unmodified matrix, since

$$\begin{aligned} \text{cov}[\mathbf{x}] &= \tilde{\mathbf{W}}\mathbb{E}\left[\mathbf{z}\mathbf{z}^T\right]\tilde{\mathbf{W}}^T + \mathbb{E}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\right] && (12.10)\\ &= \mathbf{W}\mathbf{R}\mathbf{R}^T\mathbf{W}^T + \boldsymbol{\Psi} = \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi} && (12.11)\end{aligned}$$

Geometrically, multiplying $\mathbf{W}$ by an orthogonal matrix is like rotating $\mathbf{z}$ before generating $\mathbf{x}$; but since $\mathbf{z}$ is drawn from an isotropic Gaussian, this makes no difference to the likelihood. Consequently, we cannot unique identify $\mathbf{W}$, and therefore cannot uniquely identify the latent factors, either.

To ensure a unique solution, we need to remove $L(L-1)/2$ degrees of freedom, since that is the number of orthonormal matrices of size $L \times L$.[1] In total, the FA model has $D + LD - L(L-1)/2$ free parameters (excluding the mean), where the first term arises from $\boldsymbol{\Psi}$. Obviously we require this to be less than or equal to $D(D+1)/2$, which is the number of parameters in an unconstrained (but symmetric) covariance matrix. This gives us an upper bound on L, as follows:

$$L_{max} = \lfloor D + 0.5(1 - \sqrt{1+8D})\rfloor \qquad (12.12)$$

For example, $D = 6$ implies $L \leq 3$. But we usually never choose this upper bound, since it would result in overfitting (see discussion in Section 12.3 on how to choose L).

Unfortunately, even if we set $L < L_{max}$, we still cannot uniquely identify the parameters, since the rotational ambiguity still exists. Non-identifiability does not affect the predictive performance of the model. However, it does affect the loading matrix, and hence the interpretation of the latent factors. Since factor analysis is often used to uncover structure in the data, this problem needs to be addressed. Here are some commonly used solutions:

- **Forcing $\mathbf{W}$ to be orthonormal** Perhaps the cleanest solution to the identifiability problem is to force $\mathbf{W}$ to be orthonormal, and to order the columns by decreasing variance of the corresponding latent factors. This is the approach adopted by PCA, which we will discuss in Section 12.2. The result is not necessarily more interpretable, but at least it is unique.
- **Forcing $\mathbf{W}$ to be lower triangular** One way to achieve identifiability, which is popular in the Bayesian community (e.g., (Lopes and West 2004)), is to ensure that the first visible feature is only generated by the first latent factor, the second visible feature is only generated by the first two latent factors, and so on. For example, if $L = 3$ and $D = 4$, the correspond factor loading matrix is given by

  $$\mathbf{W} = \begin{pmatrix} w_{11} & 0 & 0 \\ w_{21} & w_{22} & 0 \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix} \qquad (12.13)$$

  We also require that $w_{jj} > 0$ for $j = 1 : L$. The total number of parameters in this constrained matrix is $D + DL - L(L-1)/2$, which is equal to the number of uniquely identifiable parameters. The disadvantage of this method is that the first L visible variables,

1. To see this, note that there are $L-1$ free parameters in $\mathbf{R}$ in the first column (since the column vector must be normalized to unit length), there are $L-2$ free parameters in the second column (which must be orthogonal to the first), and so on.

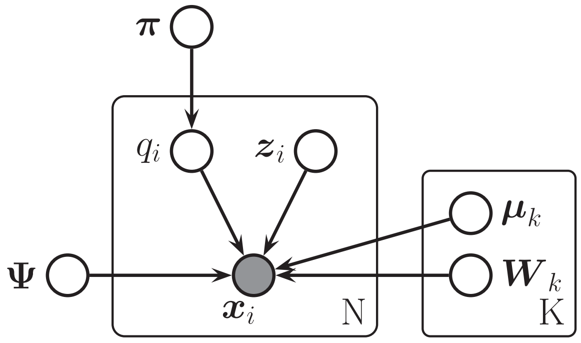


Figure 12.3 Mixture of factor analysers as a DGM.

known as the **founder variables**, affect the interpretation of the latent factors, and so must be chosen carefully.

- **Sparsity promoting priors on the weights** Instead of pre-specifying which entries in $\mathbf{W}$ are zero, we can encourage the entries to be zero, using ℓ_1 regularization (Zou et al. 2006), ARD (Bishop 1999; Archambeau and Bach 2008), or spike-and-slab priors (Rattray et al. 2009). This is called sparse factor analysis. This does not necessarily ensure a unique MAP estimate, but it does encourage interpretable solutions. See Section 13.8.
- **Choosing an informative rotation matrix** There are a variety of heuristic methods that try to find rotation matrices $\mathbf{R}$ which can be used to modify $\mathbf{W}$ (and hence the latent factors) so as to try to increase the interpretability, typically by encouraging them to be (approximately) sparse. One popular method is known as **varimax** (Kaiser 1958).
- **Use of non-Gaussian priors for the latent factors** In Section 12.6, we will dicuss how replacing $p(\mathbf{z}_i)$ with a non-Gaussian distribution can enable us to sometimes uniquely identify $\mathbf{W}$ as well as the latent factors. This technique is known as ICA.

12.1.4 Mixtures of factor analysers

The FA model assumes that the data lives on a low dimensional linear manifold. In reality, most data is better modeled by some form of low dimensional *curved* manifold. We can approximate a curved manifold by a piecewise linear manifold. This suggests the following model: let the k'th linear subspace of dimensionality L_k be represented by $\mathbf{W}_k$, for $k = 1 : K$. Suppose we have a latent indicator $q_i \in \{1, \ldots, K\}$ specifying which subspace we should use to generate the data. We then sample $\mathbf{z}_i$ from a Gaussian prior and pass it through the $\mathbf{W}_k$ matrix (where $k = q_i$), and add noise. More precisely, the model is as follows:

$$p(\mathbf{x}_i|\mathbf{z}_i, q_i = k, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k + \mathbf{W}_k\mathbf{z}_i, \boldsymbol{\Psi}) \quad (12.14)$$

$$p(\mathbf{z}_i|\boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I}) \quad (12.15)$$

$$p(q_i|\boldsymbol{\theta}) = \mathrm{Cat}(q_i|\boldsymbol{\pi}) \quad (12.16)$$

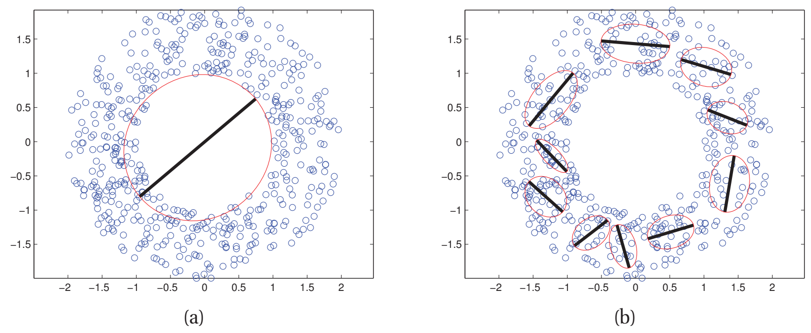


Figure 12.4 Mixture of 1d PPCAs fit to a dataset, for $K = 1, 10$. Figure generated by `mixPpcaDemoNetlab`.

This is called a **mixture of factor analysers** (MFA) (Hinton et al. 1997). The CI assumptions are represented in Figure 12.3.

Another way to think about this model is as a low-rank version of a mixture of Gaussians. In particular, this model needs $O(KLD)$ parameters instead of the $O(KD^2)$ parameters needed for a mixture of full covariance Gaussians. This can reduce overfitting. In fact, MFA is a good generic density model for high-dimensional real-valued data.

12.1.5 EM for factor analysis models

Using the results from Chapter 4, it is straightforward to derive an EM algorithm to fit an FA model. With just a little more work, we can fit a mixture of FAs. Below we state the results without proof. The derivation can be found in (Ghahramani and Hinton 1996a); however, deriving these equations yourself is a useful exercise if you want to become proficient at the math.

To obtain the results for a single factor analyser, just set $r_{ic} = 1$ and $c = 1$ in the equations below. In Section 12.2.5 we will see a further simplification of these equations that arises when fitting a PPCA model, where the results will turn out to have a particularly simple and elegant intepretation.

In the E step, we compute the posterior responsibility of cluster c for data point i using

$$r_{ic} \triangleq p(q_i = c|\mathbf{x}_i, \boldsymbol{\theta}) \propto \pi_c \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \mathbf{W}_c\mathbf{W}_c^T + \boldsymbol{\Psi}) \tag{12.17}$$

The conditional posterior for $\mathbf{z}_i$ is given by

$$\begin{aligned} p(\mathbf{z}_i|\mathbf{x}_i, q_i = c, \boldsymbol{\theta}) &= \mathcal{N}(\mathbf{z}_i|\mathbf{m}_{ic}, \boldsymbol{\Sigma}_{ic}) && (12.18)\\ \boldsymbol{\Sigma}_{ic} &\triangleq (\mathbf{I}_L + \mathbf{W}_c^T\boldsymbol{\Psi}_c^{-1}\mathbf{W}_c)^{-1} && (12.19)\\ \mathbf{m}_{ic} &\triangleq \boldsymbol{\Sigma}_{ic}(\mathbf{W}_c^T\boldsymbol{\Psi}_c^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_c)) && (12.20) \end{aligned}$$

In the M step, it is easiest to estimate $\boldsymbol{\mu}_c$ and $\mathbf{W}_c$ at the same time, by defining $\tilde{\mathbf{W}}_c =$

$(\mathbf{W}_c, \boldsymbol{\mu}_c)$, $\tilde{\mathbf{z}} = (\mathbf{z}, 1)$, Also, define

$$\mathbf{b}_{ic} \triangleq \mathbb{E}\left[\tilde{\mathbf{z}}|\mathbf{x}_i, q_i = c\right] = [\mathbf{m}_{ic}; 1] \tag{12.21}$$

$$\mathbf{C}_{ic} \triangleq \mathbb{E}\left[\tilde{\mathbf{z}}\tilde{\mathbf{z}}^T|\mathbf{x}_i, q_i = c\right] = \begin{pmatrix} \mathbb{E}\left[\mathbf{z}\mathbf{z}^T|\mathbf{x}_i, q_i = c\right] & \mathbb{E}\left[\mathbf{z}|\mathbf{x}_i, q_i = c\right] \\ \mathbb{E}\left[\mathbf{z}|\mathbf{x}_i, q_i = c\right]^T & 1 \end{pmatrix} \tag{12.22}$$

Then the M step is as follows:

$$\hat{\tilde{\mathbf{W}}}_c = \left[\sum_i r_{ic}\mathbf{x}_i\mathbf{b}_{ic}^T\right]\left[\sum_i r_{ic}\mathbf{C}_{ic}\right]^{-1} \tag{12.23}$$

$$\hat{\boldsymbol{\Psi}} = \frac{1}{N}\text{diag}\left\{\sum_{ic} r_{ic}\left(\mathbf{x}_i - \hat{\tilde{\mathbf{W}}}_c\mathbf{b}_{ic}\right)\mathbf{x}_i^T\right\} \tag{12.24}$$

$$\hat{\pi}_c = \frac{1}{N}\sum_{i=1}^N r_{ic} \tag{12.25}$$

Note that these updates are for "vanilla" EM. A much faster version of this algorithm, based on ECM, is described in (Zhao and Yu 2008).

12.1.6 Fitting FA models with missing data

In many applications, such as collaborative filtering, we have missing data. One virtue of the EM approach to fitting an FA/PPCA model is that it is easy to extend to this case. However, overfitting can be a problem if there is a lot of missing data. Consequently it is important to perform MAP estimation or to use Bayesian inference. See e.g., (Ilin and Raiko 2010) for details.

12.2 Principal components analysis (PCA)

Consider the FA model where we constrain $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$, and $\mathbf{W}$ to be orthonormal. It can be shown (Tipping and Bishop 1999) that, as $\sigma^2 \to 0$, this model reduces to classical (non-probabilistic) **principal components analysis** (**PCA**), also known as the **Karhunen Loeve** transform. The version where $\sigma^2 > 0$ is known as **probabilistic PCA** (**PPCA**) (Tipping and Bishop 1999), or **sensible PCA** (Roweis 1997). (An equivalent result was derived independently, from a different perspective, in (Moghaddam and Pentland 1995).)

To make sense of this result, we first have to learn about classical PCA. We then connect PCA to the SVD. And finally we return to discuss PPCA.

12.2.1 Classical PCA: statement of the theorem

The **synthesis view** of classical PCA is summarized in the forllowing theorem.

Theorem 12.2.1. *Suppose we want to find an orthogonal set of L linear basis vectors $\mathbf{w}_j \in \mathbb{R}^D$, and the corresponding scores $\mathbf{z}_i \in \mathbb{R}^L$, such that we minimize the average* **reconstruction error**

$$J(\mathbf{W}, \mathbf{Z}) = \frac{1}{N}\sum_{i=1}^N ||\mathbf{x}_i - \hat{\mathbf{x}}_i||^2 \tag{12.26}$$

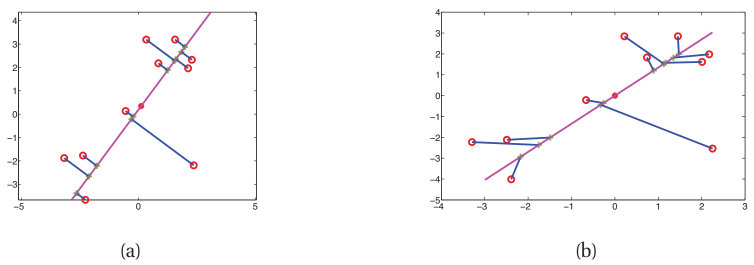


Figure 12.5 An illustration of PCA and PPCA where $D = 2$ and $L = 1$. Circles are the original data points, crosses are the reconstructions. The red star is the data mean. (a) PCA. The points are orthogonally projected onto the line. Figure generated by `pcaDemo2d`. (b) PPCA. The projection is no longer orthogonal: the reconstructions are shrunk towards the data mean (red star). Based on Figure 7.6 of (Nabney 2001). Figure generated by `ppcaDemo2d`.

where $\hat{\mathbf{x}}_i = \mathbf{W}\mathbf{z}_i$, subject to the constraint that $\mathbf{W}$ is orthonormal. Equivalently, we can write this objective as follows:

$$J(\mathbf{W}, \mathbf{Z}) = ||\mathbf{X} - \mathbf{W}\mathbf{Z}^T||_F^2 \tag{12.27}$$

where $\mathbf{Z}$ is an $N \times L$ matrix with the $\mathbf{z}_i$ in its rows, and $||\mathbf{A}||_F$ is the **Frobenius norm** *of matrix $\mathbf{A}$, defined by*

$$||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} = \sqrt{\text{tr}(\mathbf{A}^T\mathbf{A})} = ||\mathbf{A}(:)||_2 \tag{12.28}$$

The optimal solution is obtained by setting $\hat{\mathbf{W}} = \mathbf{V}_L$, where $\mathbf{V}_L$ contains the L eigenvectors with largest eigenvalues of the empirical covariance matrix, $\hat{\mathbf{\Sigma}} = \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i\mathbf{x}_i^T$. (We assume the $\mathbf{x}_i$ have zero mean, for notational simplicity.) Furthermore, the optimal low-dimensional encoding of the data is given by $\hat{\mathbf{z}}_i = \mathbf{W}^T\mathbf{x}_i$, which is an orthogonal projection of the data onto the column space spanned by the eigenvectors.

An example of this is shown in Figure 12.5(a) for $D = 2$ and $L = 1$. The diagonal line is the vector $\mathbf{w}_1$; this is called the first principal component or principal direction. The data points $\mathbf{x}_i \in \mathbb{R}^2$ are orthogonally projected onto this line to get $z_i \in \mathbb{R}$. This is the best 1-dimensional approximation to the data. (We will discuss Figure 12.5(b) later.)

In general, it is hard to visualize higher dimensional data, but if the data happens to be a set of images, it is easy to do so. Figure 12.6 shows the first three principal vectors, reshaped as images, as well as the reconstruction of a specific image using a varying number of basis vectors. (We discuss how to choose L in Section 11.5.)

Below we will show that the principal directions are the ones along which the data shows maximal variance. This means that PCA can be "misled" by directions in which the variance is high merely because of the measurement scale. Figure 12.7(a) shows an example, where the vertical axis (weight) uses a large range than the horizontal axis (height), resulting in a line that looks somewhat "unnatural". It is therefore standard practice to standardize the data first, or

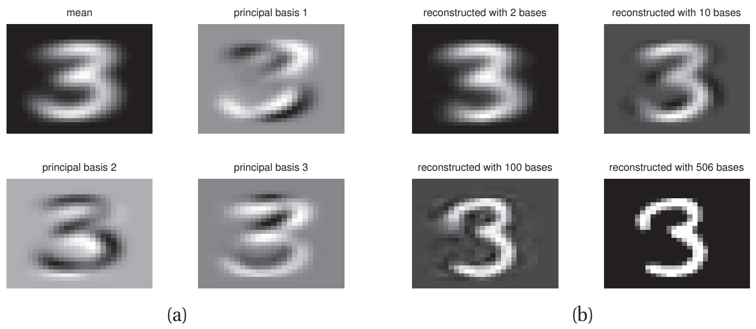

mean
principal basis 1
principal basis 2
principal basis 3
(a)
reconstructed with 2 bases
reconstructed with 10 bases
reconstructed with 100 bases
reconstructed with 506 bases
(b)

Figure 12.6 (a) The mean and the first three PC basis vectors (eigendigits) based on 25 images of the digit 3 (from the MNIST dataset). (b) Reconstruction of an image based on 2, 10, 100 and all the basis vectors. Figure generated by `pcaImageDemo`.

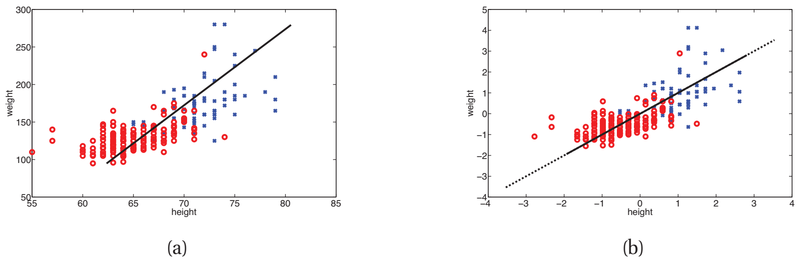


Figure 12.7 Effect of standardization on PCA applied to the height/ weight dataset. Left: PCA of raw data. Right: PCA of standardized data. Figure generated by `pcaDemoHeightWeight`.

equivalently, to work with correlation matrices instead of covariance matrices. The benefits of this are apparent from Figure 12.7(b).

12.2.2 Proof *

Proof. We use $\mathbf{w}_j \in \mathbb{R}^D$ to denote the j'th principal direction, $\mathbf{x}_i \in \mathbb{R}^D$ to denote the i'th high-dimensional observation, $\mathbf{z}_i \in \mathbb{R}^L$ to denote the i'th low-dimensional representation, and $\tilde{\mathbf{z}}_j \in \mathbb{R}^N$ to denote the $[z_{1j}, \ldots, z_{Nj}]$, which is the j'th component of all the low-dimensional vectors.

Let us start by estimating the best 1d solution, $\mathbf{w}_1 \in \mathbb{R}^D$, and the corresponding projected points $\tilde{\mathbf{z}}_1 \in \mathbb{R}^N$. We will find the remaining bases $\mathbf{w}_2$, $\mathbf{w}_3$, etc. later. The reconstruction error is given by

$$J(\mathbf{w}_1, \mathbf{z}_1) = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{x}_i - z_{i1}\mathbf{w}_1||^2 = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - z_{i1}\mathbf{w}_1)^T(\mathbf{x}_i - z_{i1}\mathbf{w}_1) \tag{12.29}$$

$$= \frac{1}{N}\sum_{i=1}^{N}[\mathbf{x}_i^T\mathbf{x}_i - 2z_{i1}\mathbf{w}_1^T\mathbf{x}_i + z_{i1}^2\mathbf{w}_1^T\mathbf{w}_1] \tag{12.30}$$

$$= \frac{1}{N}\sum_{i=1}^{N}[\mathbf{x}_i^T\mathbf{x}_i - 2z_{i1}\mathbf{w}_1^T\mathbf{x}_i + z_{i1}^2] \tag{12.31}$$

since $\mathbf{w}_1^T\mathbf{w}_1 = 1$ (by the orthonormality assumption). Taking derivatives wrt z_{i1} and equating to zero gives

$$\frac{\partial}{\partial z_{i1}} J(\mathbf{w}_1, \mathbf{z}_1) = \frac{1}{N}[-2\mathbf{w}_1^T\mathbf{x}_i + 2z_{i1}] = 0 \Rightarrow z_{i1} = \mathbf{w}_1^T\mathbf{x}_i \tag{12.32}$$

So the optimal reconstruction weights are obtained by orthogonally projecting the data onto the first principal direction, $\mathbf{w}_1$ (see Figure 12.5(a)). Plugging back in gives

$$J(\mathbf{w}_1) = \frac{1}{N}\sum_{i=1}^{N}[\mathbf{x}_i^T\mathbf{x}_i - z_{i1}^2] = \text{const} - \frac{1}{N}\sum_{i=1}^{N} z_{i1}^2 \tag{12.33}$$

Now the variance of the projected coordinates is given by

$$\text{var}[\tilde{\mathbf{z}}_1] = \mathbb{E}\left[\tilde{\mathbf{z}}_1^2\right] - (\mathbb{E}[\tilde{\mathbf{z}}_1])^2 = \frac{1}{N}\sum_{i=1}^{N} z_{i1}^2 - 0 \tag{12.34}$$

since

$$\mathbb{E}[z_{i1}] = \mathbb{E}\left[\mathbf{x}_i^T\mathbf{w}_1\right] = \mathbb{E}[\mathbf{x}_i]^T\,\mathbf{w}_1 = 0 \tag{12.35}$$

because the data has been centered. From this, we see that *minimizing* the reconstruction error is equivalent to *maximizing* the variance of the projected data, i.e.,

$$\arg\min_{\mathbf{w}_1} J(\mathbf{w}_1) = \arg\max_{\mathbf{w}_1} \text{var}[\tilde{\mathbf{z}}_1] \tag{12.36}$$

This is why it is often said that PCA finds the directions of maximal variance. This is called the **analysis view** of PCA.

The variance of the projected data can be written as

$$\frac{1}{N}\sum_{i=1}^{N} z_{i1}^2 = \frac{1}{N}\sum_{i=1}^{N}\mathbf{w}_1^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{w}_1 = \mathbf{w}_1^T\hat{\mathbf{\Sigma}}\mathbf{w}_1 \tag{12.37}$$

where $\hat{\mathbf{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N}\sum_i \mathbf{x}_i\mathbf{x}_i^T$ is the empirical covariance matrix (or correlation matrix if the data is standardized).

We can trivially maximize the variance of the projection (and hence minimize the reconstruction error) by letting $||\mathbf{w}_1|| \to \infty$, so we impose the constraint $||\mathbf{w}_1|| = 1$ and instead maximize

$$\tilde{J}(\mathbf{w}_1) = \mathbf{w}_1^T \hat{\mathbf{\Sigma}} \mathbf{w}_1 + \lambda_1(\mathbf{w}_1^T \mathbf{w}_1 - 1) \quad (12.38)$$

where λ_1 is the Lagrange multiplier. Taking derivatives and equating to zero we have

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}_1} \tilde{J}(\mathbf{w}_1) &= 2\hat{\mathbf{\Sigma}}\mathbf{w}_1 - 2\lambda_1 \mathbf{w}_1 = 0 && (12.39) \\ \hat{\mathbf{\Sigma}}\mathbf{w}_1 &= \lambda_1 \mathbf{w}_1 && (12.40) \end{aligned}$$

Hence the direction that maximizes the variance is an eigenvector of the covariance matrix. Left multiplying by $\mathbf{w}_1$ (and using $\mathbf{w}_1^T \mathbf{w}_1 = 1$) we find that the variance of the projected data is

$$\mathbf{w}_1^T \hat{\mathbf{\Sigma}} \mathbf{w}_1 = \lambda_1 \quad (12.41)$$

Since we want to maximize the variance, we pick the eigenvector which corresponds to the largest eigenvalue.

Now let us find another direction $\mathbf{w}_2$ to further minimize the reconstruction error, subject to $\mathbf{w}_1^T \mathbf{w}_2 = 0$ and $\mathbf{w}_2^T \mathbf{w}_2 = 1$. The error is

$$J(\mathbf{w}_1, \mathbf{z}_1, \mathbf{w}_2, \mathbf{z}_2) = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{x}_i - z_{i1}\mathbf{w}_1 - z_{i2}\mathbf{w}_2||^2 \quad (12.42)$$

Optimizing wrt $\mathbf{w}_1$ and $\mathbf{z}_1$ gives the same solution as before. Exercise 12.4 asks you to show that $\frac{\partial J}{\partial \mathbf{z}_2} = 0$ yields $z_{i2} = \mathbf{w}_2^T \mathbf{x}_i$. In other words, the second principal encoding is gotten by projecting onto the second principal direction. Substituting in yields

$$J(\mathbf{w}_2) = \frac{1}{n} \sum_{i=1}^{N} [\mathbf{x}_i^T \mathbf{x}_i - \mathbf{w}_1^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{w}_1 - \mathbf{w}_2^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{w}_2] = \text{const} - \mathbf{w}_2^T \hat{\mathbf{\Sigma}} \mathbf{w}_2 \quad (12.43)$$

Dropping the constant term and adding the constraints yields

$$\tilde{J}(\mathbf{w}_2) = -\mathbf{w}_2^T \hat{\mathbf{\Sigma}} \mathbf{w}_2 + \lambda_2(\mathbf{w}_2^T \mathbf{w}_2 - 1) + \lambda_{12}(\mathbf{w}_2^T \mathbf{w}_1 - 0) \quad (12.44)$$

Exercise 12.4 asks you to show that the solution is given by the eigenvector with the second largest eigenvalue:

$$\hat{\mathbf{\Sigma}} \mathbf{w}_2 = \lambda_2 \mathbf{w}_2 \quad (12.45)$$

The proof continues in this way. (Formally one can use induction.) □

12.2.3 Singular value decomposition (SVD)

We have defined the solution to PCA in terms of eigenvectors of the covariance matrix. However, there is another way to obtain the solution, based on the **singular value decomposition**, or **SVD**. This basically generalizes the notion of eigenvectors from square matrices to any kind of matrix.

In particular, any (real) $N \times D$ matrix $\mathbf{X}$ can be decomposed as follows

$$\underbrace{\mathbf{X}}_{N\times D} = \underbrace{\mathbf{U}}_{N\times N}\underbrace{\mathbf{S}}_{N\times D}\underbrace{\mathbf{V}^T}_{D\times D} \tag{12.46}$$

where $\mathbf{U}$ is an $N \times N$ matrix whose columns are orthornormal (so $\mathbf{U}^T\mathbf{U} = \mathbf{I}_N$), $\mathbf{V}$ is $D \times D$ matrix whose rows and columns are orthonormal (so $\mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{V}^T = \mathbf{I}_D$), and $\mathbf{S}$ is a $N \times D$ matrix containing the $r = \min(N, D)$ **singular values** $\sigma_i \geq 0$ on the main diagonal, with 0s filling the rest of the matrix. The columns of $\mathbf{U}$ are the left singular vectors, and the columns of $\mathbf{V}$ are the right singular vectors. See Figure 12.8(a) for an example.

Since there are at most D singular values (assuming $N > D$), the last $N - D$ columns of $\mathbf{U}$ are irrelevant, since they will be multiplied by 0. The **economy sized SVD**, or **thin SVD**, avoids computing these unnecessary elements. Let us denote this decomposition by $\hat{\mathbf{U}}\hat{\mathbf{S}}\hat{\mathbf{V}}$. If $N > D$, we have

$$\underbrace{\mathbf{X}}_{N\times D} = \underbrace{\hat{\mathbf{U}}}_{N\times D}\underbrace{\hat{\mathbf{S}}}_{D\times D}\underbrace{\hat{\mathbf{V}}^T}_{D\times D} \tag{12.47}$$

as in Figure 12.8(a). If $N < D$, we have

$$\underbrace{\mathbf{X}}_{N\times D} = \underbrace{\hat{\mathbf{U}}}_{N\times N}\underbrace{\hat{\mathbf{S}}}_{N\times N}\underbrace{\hat{\mathbf{V}}^T}_{N\times D} \tag{12.48}$$

Computing the economy-sized SVD takes $O(ND\min(N, D))$ time (Golub and van Loan 1996, p254).

The connection between eigenvectors and singular vectors is the following. For an arbitrary real matrix $\mathbf{X}$, if $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$, we have

$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{S}^T\mathbf{U}^T\,\mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{V}(\mathbf{S}^T\mathbf{S})\mathbf{V}^T = \mathbf{V}\mathbf{D}\mathbf{V}^T \tag{12.49}$$

where $\mathbf{D} = \mathbf{S}^2$ is a diagonal matrix containing the squares singular values. Hence

$$(\mathbf{X}^T\mathbf{X})\mathbf{V} = \mathbf{V}\mathbf{D} \tag{12.50}$$

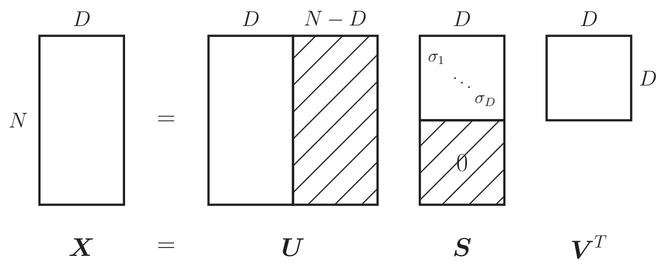
so the eigenvectors of $\mathbf{X}^T\mathbf{X}$ are equal to $\mathbf{V}$, the right singular vectors of $\mathbf{X}$, and the eigenvalues of $\mathbf{X}^T\mathbf{X}$ are equal to $\mathbf{D}$, the squared singular values. Similarly

$$\mathbf{X}\mathbf{X}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T\,\mathbf{V}\mathbf{S}^T\mathbf{U}^T = \mathbf{U}(\mathbf{S}\mathbf{S}^T)\mathbf{U}^T \tag{12.51}$$

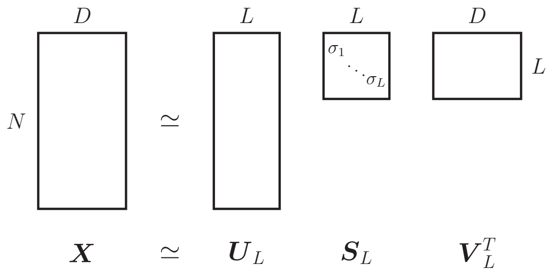
$$(\mathbf{X}\mathbf{X}^T)\mathbf{U} = \mathbf{U}(\mathbf{S}\mathbf{S}^T) = \mathbf{U}\mathbf{D} \tag{12.52}$$

so the eigenvectors of $\mathbf{X}\mathbf{X}^T$ are equal to $\mathbf{U}$, the left singular vectors of $\mathbf{X}$. Also, the eigenvalues of $\mathbf{X}\mathbf{X}^T$ are equal to the squared singular values. We can summarize all this as follows:

$$\mathbf{U} = \text{evec}(\mathbf{X}\mathbf{X}^T),\ \mathbf{V} = \text{evec}(\mathbf{X}^T\mathbf{X}),\ \mathbf{S}^2 = \text{eval}(\mathbf{X}\mathbf{X}^T) = \text{eval}(\mathbf{X}^T\mathbf{X}) \tag{12.53}$$



(a)



(b)

Figure 12.8 (a) SVD decomposition of non-square matrices $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$. The shaded parts of $\mathbf{S}$, and all the off-diagonal terms, are zero. The shaded entries in $\mathbf{U}$ and $\mathbf{S}$ are not computed in the economy-sized version, since they are not needed. (b) Truncated SVD approximation of rank L.

Since the eigenvectors are unaffected by linear scaling of a matrix, we see that the right singular vectors of $\mathbf{X}$ are equal to the eigenvectors of the empirical covariance $\hat{\mathbf{\Sigma}}$. Furthermore, the eigenvalues of $\hat{\mathbf{\Sigma}}$ are a scaled version of the squared singular values. This means we can perform PCA using just a few lines of code (see `pcaPmtk`).

However, the connection between PCA and SVD goes deeper. From Equation 12.46, we can represent a rank r matrix as follows:

$$\mathbf{X} = \sigma_1 \begin{pmatrix} | \\ \mathbf{u}_1 \\ | \end{pmatrix} \begin{pmatrix} - & \mathbf{v}_1^T & - \end{pmatrix} + \cdots + \sigma_r \begin{pmatrix} | \\ \mathbf{u}_r \\ | \end{pmatrix} \begin{pmatrix} - & \mathbf{v}_r^T & - \end{pmatrix} \tag{12.54}$$

If the singular values die off quickly as in Figure 12.10, we can produce a rank L approximation to the matrix as follows:

$$\mathbf{X} \approx \mathbf{U}_{:,1:L}\ \mathbf{S}_{1:L,1:L}\ \mathbf{V}^T_{:,1:L} \tag{12.55}$$

This is called a **truncated SVD** (see Figure 12.8(b)). The total number of parameters needed to represent an $N \times D$ matrix using a rank L approximation is

$$NL + LD + L = L(N + D + 1) \tag{12.56}$$

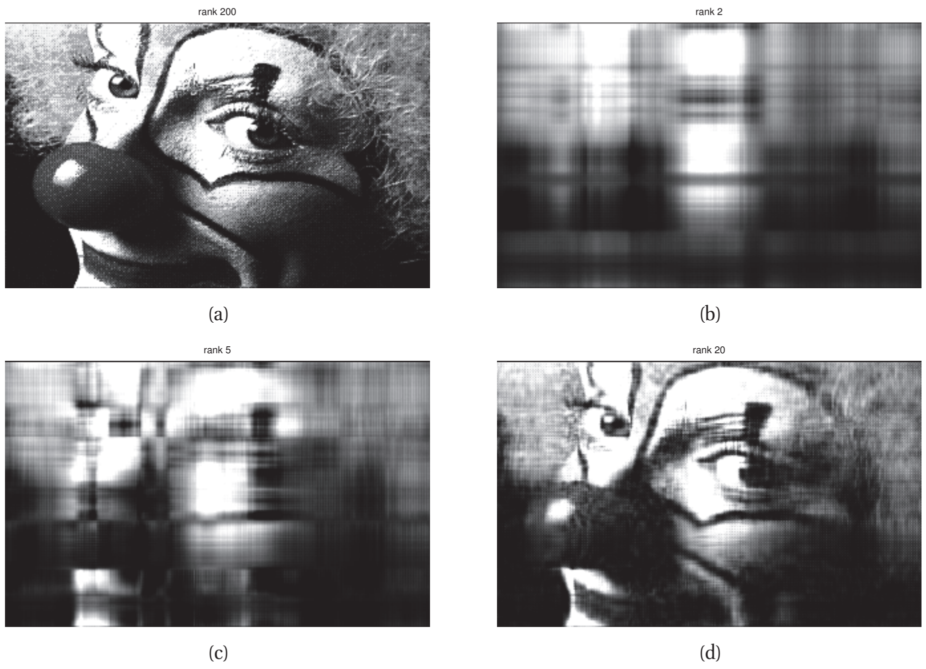


Figure 12.9 Low rank approximations to an image. Top left: The original image is of size 200×320, so has rank 200. Subsequent images have ranks 2, 5, and 20. Figure generated by `svdImageDemo`.

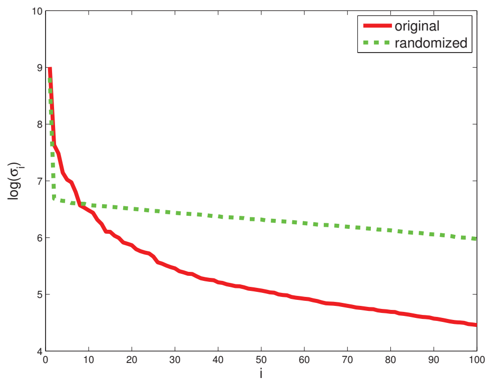


Figure 12.10 First 50 log singular values for the clown image (solid red line), and for a data matrix obtained by randomly shuffling the pixels (dotted green line). Figure generated by `svdImageDemo`.

As an example, consider the 200×320 pixel image in Figure 12.9(top left). This has 64,000 numbers in it. We see that a rank 20 approximation, with only $(200 + 320 + 1) \times 20 = 10,420$ numbers is a very good approximation.

One can show that the error in this approximation is given by

$$||\mathbf{X} - \mathbf{X}_L||_F \approx \sigma_{L+1} \tag{12.57}$$

Furthermore, one can show that the SVD offers the best rank L approximation to a matrix (best in the sense of minimizing the above Frobenius norm).

Let us connect this back to PCA. Let $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ be a truncated SVD of $\mathbf{X}$. We know that $\hat{\mathbf{W}} = \mathbf{V}$, and that $\hat{\mathbf{Z}} = \mathbf{X}\hat{\mathbf{W}}$, so

$$\hat{\mathbf{Z}} = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V} = \mathbf{U}\mathbf{S} \tag{12.58}$$

Furthermore, the optimal reconstruction is given by $\hat{\mathbf{X}} = \mathbf{Z}\hat{\mathbf{W}}^T$, so we find

$$\hat{\mathbf{X}} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{12.59}$$

This is precisely the same as a truncated SVD approximation! This is another illustration of the fact that PCA is the best low rank approximation to the data.

12.2.4 Probabilistic PCA

We are now ready to revisit PPCA. One can show the following remarkable result.

Theorem 12.2.2 ((Tipping and Bishop 1999))**.** *Consider a factor analysis model in which $\mathbf{\Psi} = \sigma^2\mathbf{I}$ and $\mathbf{W}$ is orthogonal. The observed data log likelihood is given by*

$$\log p(\mathbf{X}|\mathbf{W}, \sigma^2) = -\frac{N}{2}\ln|\mathbf{C}| - \frac{1}{2}\sum_{i=1}^{N} \mathbf{x}_i^T\mathbf{C}^{-1}\mathbf{x}_i = -\frac{N}{2}\ln|\mathbf{C}| + \mathrm{tr}(\mathbf{C}^{-1}\hat{\mathbf{\Sigma}}) \tag{12.60}$$

where $\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}$ and $\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^T = (1/N)\mathbf{X}^T\mathbf{X}$. (We are assuming centered data, for notational simplicity.) The maxima of the log-likelihood are given by

$$\hat{\mathbf{W}} = \mathbf{V}(\mathbf{\Lambda} - \sigma^2\mathbf{I})^{\frac{1}{2}}\mathbf{R} \tag{12.61}$$

where $\mathbf{R}$ is an arbitrary $L \times L$ orthogonal matrix, $\mathbf{V}$ is the $D \times L$ matrix whose columns are the first L eigenvectors of $\mathbf{S}$, and $\mathbf{\Lambda}$ is the corresponding diagonal matrix of eigenvalues. Without loss of generality, we can set $\mathbf{R} = \mathbf{I}$. Furthermore, the MLE of the noise variance is given by

$$\hat{\sigma}^2 = \frac{1}{D-L}\sum_{j=L+1}^{D}\lambda_j \tag{12.62}$$

which is the average variance associated with the discarded dimensions.

Thus, as $\sigma^2 \to 0$, we have $\hat{\mathbf{W}} \to \mathbf{V}$, as in classical PCA. What about $\hat{\mathbf{Z}}$? It is easy to see that the posterior over the latent factors is given by

$$p(\mathbf{z}_i|\mathbf{x}_i, \hat{\boldsymbol{\theta}}) = \mathcal{N}(\mathbf{z}_i|\hat{\mathbf{F}}^{-1}\hat{\mathbf{W}}^T\mathbf{x}_i, \sigma^2\hat{\mathbf{F}}^{-1}) \tag{12.63}$$

$$\hat{\mathbf{F}} \triangleq \hat{\mathbf{W}}^T\hat{\mathbf{W}} + \hat{\sigma}^2\mathbf{I} \tag{12.64}$$

(Do not confuse $\mathbf{F} = \mathbf{W}^T\mathbf{W} + \sigma^2\mathbf{I}$ with $\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}$.) Hence, as $\sigma^2 \to 0$, we find $\hat{\mathbf{W}} \to \mathbf{V}$, $\hat{\mathbf{F}} \to \mathbf{I}$ and $\hat{\mathbf{z}}_i \to \mathbf{V}^T\mathbf{x}_i$. Thus the posterior mean is obtained by an orthogonal projection of the data onto the column space of $\mathbf{V}$, as in classical PCA.

Note, however, that if $\sigma^2 >$, the posterior mean is not an orthogonal projection, since it is shrunk somewhat towards the prior mean, as illustrated in Figure 12.5(b). This sounds like an undesirable property, but it means that the reconstructions will be closer to the overall data mean, $\hat{\boldsymbol{\mu}} = \overline{\mathbf{x}}$.

12.2.5 EM algorithm for PCA

Although the usual way to fit a PCA model uses eigenvector methods, or the SVD, we can also use EM, which will turn out to have some advantages that we discuss below. EM for PCA relies on the probabilistic formulation of PCA. However the algorithm continues to work in the zero noise limit, $\sigma^2 = 0$, as shown by (Roweis 1997).

Let $\tilde{\mathbf{Z}}$ be a $L \times N$ matrix storing the posterior means (low-dimensional representations) along its columns. Similarly, let $\tilde{\mathbf{X}} = \mathbf{X}^T$ store the original data along its columns. From Equation 12.63, when $\sigma^2 = 0$, we have

$$\tilde{\mathbf{Z}} = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\tilde{\mathbf{X}} \tag{12.65}$$

This constitutes the E step. Notice that this is just an orthogonal projection of the data.

From Equation 12.23, the M step is given by

$$\hat{\mathbf{W}} = \left[\sum_i \mathbf{x}_i \mathbb{E}\left[\mathbf{z}_i\right]^T\right]\left[\sum_i \mathbb{E}\left[\mathbf{z}_i\right]\mathbb{E}\left[\mathbf{z}_i\right]^T\right]^{-1} \tag{12.66}$$

where we exploited the fact that $\boldsymbol{\Sigma} = \text{cov}\left[\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta}\right] = 0\mathbf{I}$ when $\sigma^2 = 0$. It is worth comparing this expression to the MLE for multi-output linear regression (Equation 7.89), which has the form $\mathbf{W} = (\sum_i \mathbf{y}_i\mathbf{x}_i^T)(\sum_i \mathbf{x}_i\mathbf{x}_i^T)^{-1}$. Thus we see that the M step is like linear regression where we replace the observed inputs by the expected values of the latent variables.

In summary, here is the entire algorithm:

- **E step** $\tilde{\mathbf{Z}} = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\tilde{\mathbf{X}}$
- **M step** $\mathbf{W} = \tilde{\mathbf{X}}\tilde{\mathbf{Z}}^T(\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T)^{-1}$

(Tipping and Bishop 1999) showed that the only stable fixed point of the EM algorithm is the globally optimal solution. That is, the EM algorithm converges to a solution where $\mathbf{W}$ spans the same linear subspace as that defined by the first L eigenvectors. However, if we want $\mathbf{W}$ to be orthogonal, and to contain the eigenvectors in descending order of eigenvalue, we have to orthogonalize the resulting matrix (which can be done quite cheaply). Alternatively, we can modify EM to give the principal basis directly (Ahn and Oh 2003).

This algorithm has a simple physical analogy in the case $D = 2$ and $L = 1$ (Roweis 1997). Consider some points in $\mathbb{R}^2$ attached by springs to a rigid rod, whose orientation is defined by a vector $\mathbf{w}$. Let z_i be the location where the i'th spring attaches to the rod. In the E step, we hold the rod fixed, and let the attachment points slide around so as to minimize the spring energy (which is proportional to the sum of squared residuals). In the M step, we hold the attachment

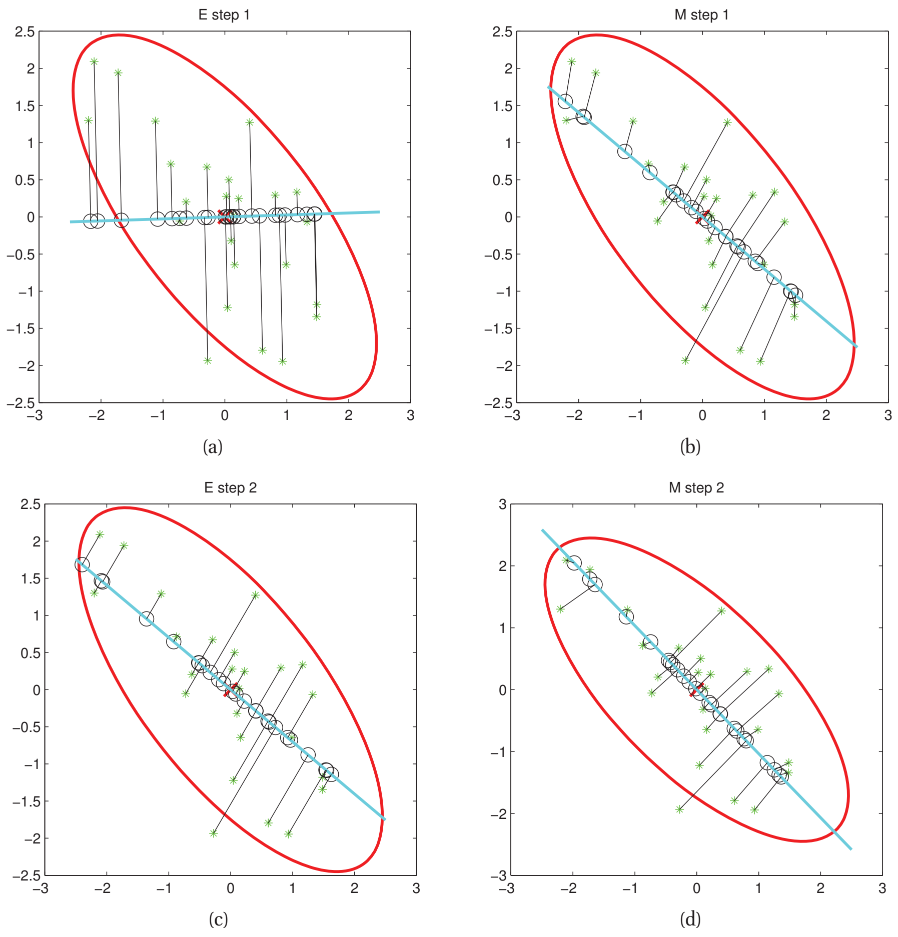


Figure 12.11 Illustration of EM for PCA when $D = 2$ and $L = 1$. Green stars are the original data points, black circles are their reconstructions. The weight vector $\mathbf{w}$ is represented by blue line. (a) We start with a random initial guess of $\mathbf{w}$. The E step is represented by the orthogonal projections. (b) We update the rod $\mathbf{w}$ in the M step, keeping the projections onto the rod (black circles) fixed. (c) Another E step. The black circles can 'slide' along the rod, but the rod stays fixed. (d) Another M step. Based on Figure 12.12 of (Bishop 2006b). Figure generated by `pcaEmStepByStep`.

points fixed and let the rod rotate so as to minimize the spring energy. See Figure 12.11 for an illustration.

Apart from this pleasing intuitive interpretation, EM for PCA has the following advantages over eigenvector methods:

- EM can be faster. In particular, assuming $N, D \gg L$, the dominant cost of EM is the projection operation in the E step, so the overall time is $O(TLND)$, where T is the number of

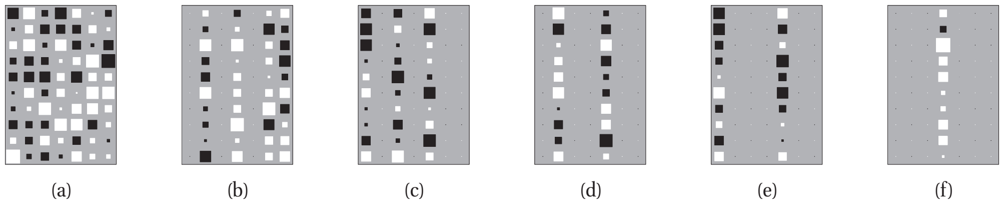


Figure 12.12 Illustration of estimating the effective dimensionalities in a mixture of factor analysers using VBEM. The blank columns have been forced to 0 via the ARD mechanism. The data was generated from 6 clusters with intrinsic dimensionalities of $7, 4, 3, 2, 2, 1$, which the method has successfully estimated. Source: Figure 4.4 of (Beal 2003). Used with kind permission of Matt Beal.

iterations. (Roweis 1997) showed experimentally that the number of iterations is usually very small (the mean was 3.6), regardless of N or D. (This results depends on the ratio of eigenvalues of the empirical covariance matrix.) This is much faster than the $O(\min(ND^2, DN^2))$ time required by straightforward eigenvector methods, although more sophisticated eigenvector methods, such as the Lanczos algorithm, have running times comparable to EM.

- EM can be implemented in an online fashion, i.e., we can update our estimate of $\mathbf{W}$ as the data streams in.
- EM can handle missing data in a simple way (see Section 12.1.6).
- EM can be extended to handle mixtures of PPCA/ FA models.
- EM can be modified to variational EM or to variational Bayes EM to fit more complex models.

12.3 Choosing the number of latent dimensions

In Section 11.5, we discussed how to choose the number of components K in a mixture model. In this section, we discuss how to choose the number of latent dimensions L in a FA/PCA model.

12.3.1 Model selection for FA/PPCA

If we use a probabilistic model, we can in principle compute $L^* = \text{argmax}_L \, p(L|\mathcal{D})$. However, there are two problems with this. First, evaluating the marginal likelihood for LVMs is quite difficult. In practice, simple approximations, such as BIC or variational lower bounds (see Section 21.5), can be used (see also (Minka 2000a)). 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 L. However, sometimes we can set the model to its maximal size, and then use a technique called automatic relevancy determination (Section 13.7), combined with EM, to automatically prune out irrelevant weights.

number of points per cluster	intrinsic dimensionalities					
	1	7	4	3	2	2
8	2				1	
8	1		2			
16	1	4				2
32	1	6	3	3	2	2
64	1	7	4	3	2	2
128	1	7	4	3	2	2

Figure 12.13 We show the estimated number of clusters, and their estimated dimensionalities, as a function of sample size. The VBEM algorithm found two different solutions when $N = 8$. Note that more clusters, with larger effective dimensionalities, are discovered as the sample sizes increases. Source: Table 4.1 of (Beal 2003). Used with kind permission of Matt Beal.

This technique will be described in a supervised context in Chapter 13, but can be adapted to the (M)FA context as shown in (Bishop 1999; Ghahramani and Beal 2000).

Figure 12.12 illustrates this approach applied to a mixture of FAs fit to a small synthetic dataset. The figures visualize the weight matrices for each cluster, using **Hinton diagrams**, where where the size of the square is proportional to the value of the entry in the matrix.[2] We see that many of them are sparse. Figure 12.13 shows that the degree of sparsity depends on the amount of training data, in accord with the Bayesian Occam's razor. In particular, when the sample size is small, the method automatically prefers simpler models, but as the sample size gets sufficiently large, the method converges on the "correct" solution, which is one with 6 subspaces of dimensionality 1, 2, 2, 3, 4 and 7.

Although the ARD/ EM method is elegant, it still needs to perform search over K. This is done using "birth" and "death" moves (Ghahramani and Beal 2000). An alternative approach is to perform stochastic sampling in the space of models. Traditional approaches, such as (Lopes and West 2004), are based on reversible jump MCMC, and also use birth and death moves. However, this can be slow and difficult to implement. More recent approaches use non-parametric priors, combined with Gibbs sampling, see e.g., (Paisley and Carin 2009).

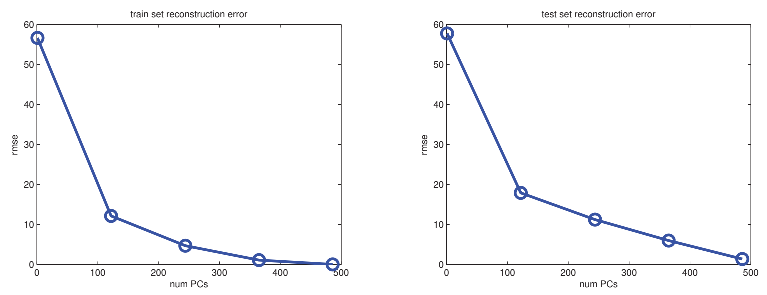
12.3.2 Model selection for PCA

Since PCA is not a probabilistic model, we cannot use any of the methods described above. An obvious proxy for the likelihood is the reconstruction error:

$$E(\mathcal{D}, L) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} ||\mathbf{x}_i - \hat{\mathbf{x}}_i||^2 \tag{12.67}$$

In the case of PCA, the reconstruction is given by by $\hat{\mathbf{x}}_i = \mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}$, where $\mathbf{z}_i = \mathbf{W}^T(\mathbf{x}_i - \boldsymbol{\mu})$ and $\mathbf{W}$ and $\boldsymbol{\mu}$ are estimated from $\mathcal{D}_{\text{train}}$.

2. Geoff Hinton is an English professor of computer science at the University of Toronto.



(a) (b)

Figure 12.14 Reconstruction error on MNIST vs number of latent dimensions used by PCA. (a) Training set. (b) Test set. Figure generated by `pcaOverfitDemo`.

Figure 12.14(a) plots $E(\mathcal{D}_{\text{train}}, L)$ vs L on the MNIST training data in Figure 12.6. We see that it drops off quite quickly, indicating that we can capture most of the empirical correlation of the pixels with a small number of factors, as illustrated qualitatively in Figure 12.6.

Exercise 12.5 asks you to prove that the residual error from only using L terms is given by the sum of the discarded eigenvalues:

$$E(\mathcal{D}_{\text{train}}, L) = \sum_{j=L+1}^{D} \lambda_j \tag{12.68}$$

Therefore an alternative to plotting the error is to plot the retained eigenvalues, in decreasing order. This is called a **scree plot**, because "the plot looks like the side of a mountain, and 'scree' refers to the debris fallen from a mountain and lying at its base".[3] This will have the same shape as the residual error plot.

A related quantity is the **fraction of variance explained**, defined as

$$F(\mathcal{D}_{\text{train}}, L) = \frac{\sum_{j=1}^{L} \lambda_j}{\sum_{j'=1}^{L_{max}} \lambda_{j'}} \tag{12.69}$$

This captures the same information as the scree plot.

Of course, if we use $L = \text{rank}(\mathbf{X})$, we get zero reconstruction error on the training set. To avoid overfitting, it is natural to plot reconstruction error on the test set. This is shown in Figure 12.14(b). Here we see that the error continues to go down even as the model becomes more complex! Thus we do not get the usual U-shaped curve that we typically expect to see.

What is going on? The problem is that PCA is not a proper generative model of the data. It is merely a compression technique. If you give it more latent dimensions, it will be able to approximate the test data more accurately. By contrast, a probabilistic model enjoys a Bayesian Occam's razor effect (Section 5.3.1), in that it gets "punished" if it wastes probability mass on parts of the space where there is little data. This is illustrated in Figure 12.15, which plots the

3. Quotation from `http://janda.org/workshop/factoranalysis/SPSSrun/SPSS08.htm`.

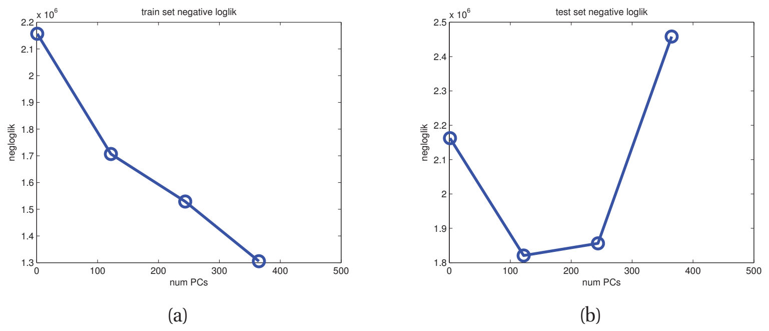


Figure 12.15 Negative log likelihood on MNIST vs number of latent dimensions used by PPCA. (a) Training set. (b) Test set. Figure generated by `pcaOverfitDemo`.

negative log likelihood, computed using PPCA, vs L. Here, on the test set, we see the usual U-shaped curve.

These results are analogous to those in Section 11.5.2, where we discussed the issue of choosing K in the K-means algorithm vs using a GMM.

12.3.2.1 Profile likelihood

Although there is no U-shape, there is sometimes a "regime change" in the plots, from relatively large errors to relatively small. One way to automate the detection of this is described in (Zhu and Ghodsi 2006). The idea is this. Let λ_k be some measure of the error incurred by a model of size k, such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{L_{max}}$. In PCA, these are the eigenvalues, but the method can also be applied to K-means. Now consider partitioning these values into two groups, depending on whether $k < L$ or $k > L$, where L is some threshold which we will determine. To measure the quality of L, we will use a simple change-point model, where $\lambda_k \sim \mathcal{N}(\mu_1, \sigma^2)$ if $k \leq L$, and $\lambda_k \sim \mathcal{N}(\mu_2, \sigma^2)$ if $k > L$. (It is important that σ^2 be the same in both models, to prevent overfitting in the case where one regime has less data than the other.) Within each of the two regimes, we assume the λ_k are iid, which is obviously incorrect, but is adequate for our present purposes. We can fit this model for each $L = 1 : L_{max}$ by partitioning the data and computing the MLEs, using a pooled estimate of the variance:

$$\mu_1(L) = \frac{\sum_{k \leq L} \lambda_k}{L}, \quad \mu_2(L) = \frac{\sum_{k > L} \lambda_k}{N - L} \tag{12.70}$$

$$\sigma^2(L) = \frac{\sum_{k \leq L}(\lambda_k - \mu_1(L))^2 + \sum_{k > L}(\lambda_k - \mu_2(L))^2}{N} \tag{12.71}$$

We can then evaluate the **profile log likelihood**

$$\ell(L) = \sum_{k=1}^{L} \log \mathcal{N}(\lambda_k|\mu_1(L), \sigma^2(L)) + \sum_{k=L+1}^{K} \log \mathcal{N}(\lambda_k|\mu_2(L), \sigma^2(L)) \tag{12.72}$$

Finally, we choose $L^* = \arg\max \ell(L)$. This is illustrated in Figure 12.16. On the left, we plot the scree plot, which has the same shape as in Figure 12.14(a). On the right, we plot the profile

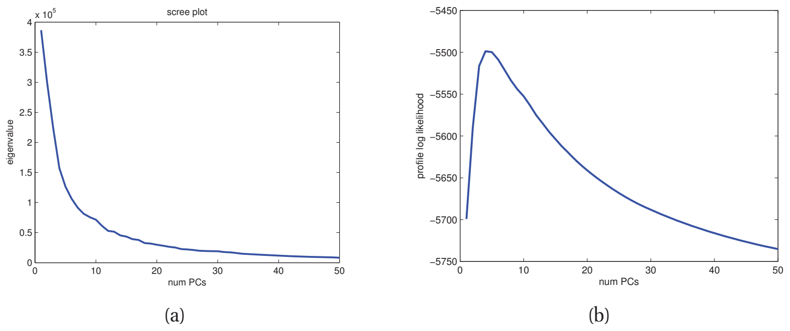


Figure 12.16 (a) Scree plot for training set, corresponding to Figure 12.14(a). (b) Profile likelihood. Figure generated by `pcaOverfitDemo`.

likelihood. Rather miraculously, we see a fairly well-determined peak.

12.4 PCA for categorical data

In this section, we consider extending the factor analysis model to the case where the observed data is categorical rather than real-valued. That is, the data has the form $y_{ij} \in \{1, \ldots, C\}$, where $j = 1 : R$ is the number of observed response variables. We assume each y_{ij} is generated from a latent variable $\mathbf{z}_i \in \mathbb{R}^L$, with a Gaussian prior, which is passed through the softmax function as follows:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{12.73}$$

$$p(\mathbf{y}_i|\mathbf{z}_i, \boldsymbol{\theta}) = \prod_{r=1}^{R} \text{Cat}(y_{ir}|\mathcal{S}(\mathbf{W}_r^T \mathbf{z}_i + \mathbf{w}_{0r})) \tag{12.74}$$

where $\mathbf{W}_r \in \mathbb{R}^{L \times M}$ is the factor loading matrix for response j, and $\mathbf{w}_{0r} \in \mathbb{R}^M$ is the offset term for response r, and $\boldsymbol{\theta} = (\mathbf{W}_r, \mathbf{w}_{0r})_{r=1}^R$. (We need an explicit offset term, since clamping one element of $\mathbf{z}_i$ to 1 can cause problems when computing the posterior covariance.) As in factor analysis, we have defined the prior mean to be $\mathbf{m}_0 = \mathbf{0}$ and the prior covariance $\mathbf{V}_0 = \mathbf{I}$, since we can capture non-zero mean by changing $\mathbf{w}_{0j}$ and non-identity covariance by changing $\mathbf{W}_r$. We will call this **categorical PCA**. See Chapter 27 for a discussion of related models.

It is interesting to study what kinds of distributions we can induce on the observed variables by varying the parameters. For simplicity, we assume there is a single ternary response variable, so y_i lives in the 3d probability simplex. Figure 12.17 shows what happens when we vary the parameters of the prior, $\mathbf{m}_0$ and $\mathbf{V}_0$, which is equivalent to varying the parameters of the likelihood, $\mathbf{W}_1$ and $\mathbf{w}_{01}$. We see that this can define fairly complex distributions over the simplex. This induced distribution is known as the **logistic normal** distribution (Aitchison 1982).

We can fit this model to data using a modified version of EM. The basic idea is to infer a Gaussian approximation to the posterior $p(\mathbf{z}_i|\mathbf{y}_i, \boldsymbol{\theta})$ in the E step, and then to maximize $\boldsymbol{\theta}$ in the M step. The details for the multiclass case, can be found in (Khan et al. 2010) (see

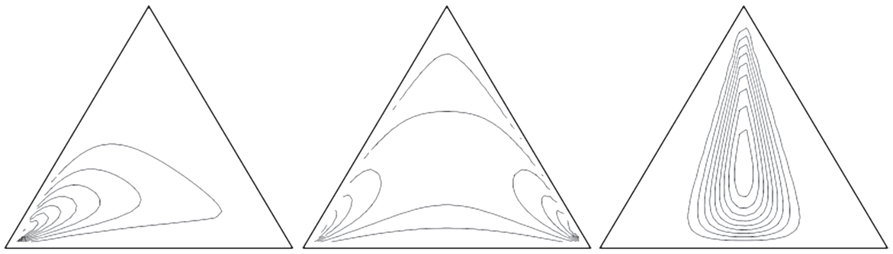


Figure 12.17 Some examples of the logistic normal distribution defined on the 3d simplex. (a) Diagonal covariance and non-zero mean. (b) Negative correlation between states 1 and 2. (c) Positive correlation between states 1 and 2. Source: Figure 1 of (Blei and Lafferty 2007). Used with kind permission of David Blei.

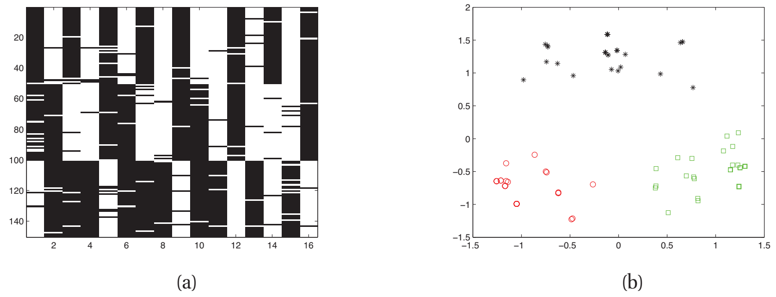


Figure 12.18 Left: 150 synthetic 16 dimensional bit vectors. Right: the 2d embedding learned by binary PCA, using variational EM. We have color coded points by the identity of the true "prototype" that generated them. Figure generated by `binaryFaDemoTipping`.

also Section 21.8.1.1). The details for the binary case for the the sigmoid link can be found in Exercise 21.9, and for the probit link in Exercise 21.10.

One application of such a model is to visualize high dimensional categorical data. Figure 12.18(a) shows a simple example where we have 150 6-dimensional bit vectors. It is clear that each sample is just a noisy copy of one of three binary prototypes. We fit a 2d catFA to this model, yielding approximate MLEs $\hat{\boldsymbol{\theta}}$. In Figure 12.18(b), we plot $\mathbb{E}\left[\mathbf{z}_i|\mathbf{x}_i, \hat{\boldsymbol{\theta}}\right]$. We see that there are three distinct clusters, as is to be expected.

In (Khan et al. 2010), we show that this model outperforms finite mixture models on the task of imputing missing entries in design matrices consisting of real and categorical data. This is useful for analysing social science survey data, which often has missing data and variables of mixed type.

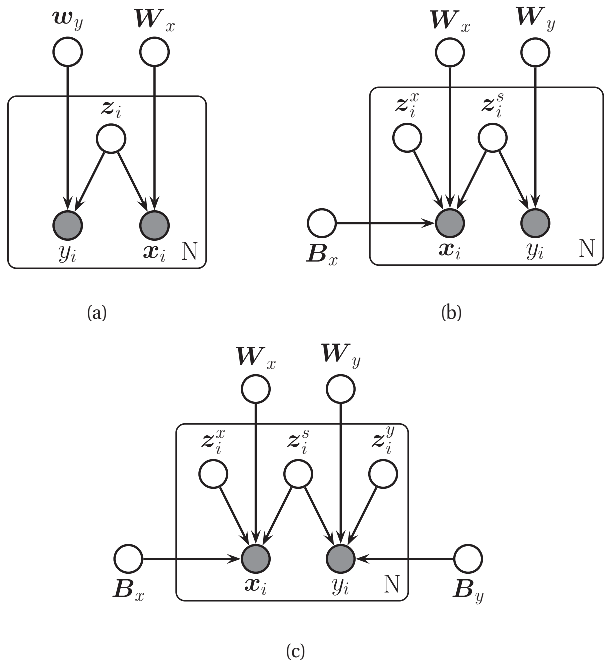


Figure 12.19 Gaussian latent factor models for paired data. (a) Supervised PCA. (b) Partial least squares. (c) Canonical correlation analysis.

12.5 PCA for paired and multi-view data

It is common to have a pair of related datasets, e.g., gene expression and gene copy number, or movie ratings by users and movie reviews. It is natural to want to combine these together into a low-dimensional embedding. This is an example of **data fusion**. In some cases, we might want to predict one element of the pair, say $\mathbf{x}_{i1}$, from the other one, $\mathbf{x}_{i2}$, via the low-dimensional "bottleneck".

Below we discuss various latent Gaussian models for these tasks, following the presentation of (Virtanen 2010). The models easily generalize from pairs to sets of data, $\mathbf{x}_{im}$, for $m = 1 : M$. We focus on the case where $\mathbf{x}_{im} \in \mathbb{R}^{D_m}$. In this case, the joint distribution is multivariate Gaussian, so we can easily fit the models using EM, or Gibbs sampling.

We can generalize the models to handle discrete and count data by using the exponential family as a response distribution instead of the Gaussian, as we explain in Section 27.2.2. However, this will require the use of approximate inference in the E step (or an analogous modification to MCMC).

12.5.1 Supervised PCA (latent factor regression)

Consider the following model, illustrated in Figure 12.19(a):

$$\begin{aligned} p(\mathbf{z}_i) &= \mathcal{N}(\mathbf{0}, \mathbf{I}_L) && (12.75) \\ p(y_i|\mathbf{z}_i) &= \mathcal{N}(\mathbf{w}_y^T \mathbf{z}_i + \mu_y, \sigma_y^2) && (12.76) \\ p(\mathbf{x}_i|\mathbf{z}_i) &= \mathcal{N}(\mathbf{W}_x \mathbf{z}_i + \boldsymbol{\mu}_x, \sigma_x^2 \mathbf{I}_D) && (12.77) \end{aligned}$$

In (Yu et al. 2006), this is called **supervised PCA**. In (West 2003), this is called **Bayesian factor regression**. This model is like PCA, except that the target variable y_i is taken into account when learning the low dimensional embedding. Since the model is jointly Gaussian, we have

$$y_i|\mathbf{x}_i \sim \mathcal{N}(\mathbf{x}_i^T \mathbf{w}, \sigma_y^2 + \mathbf{w}_y^T \mathbf{C} \mathbf{w}_y) \quad (12.78)$$

where $\mathbf{w} = \boldsymbol{\Psi}^{-1} \mathbf{W}_x \mathbf{C} \mathbf{w}_y$, $\boldsymbol{\Psi} = \sigma_x^2 \mathbf{I}_D$, and $\mathbf{C}^{-1} = \mathbf{I} + \mathbf{W}_x^T \boldsymbol{\Psi}^{-1} \mathbf{W}_x$. So although this is a joint density model of $(y_i, \mathbf{x}_i)$, we can infer the implied conditional distribution.

We now show an interesting connection to Zellner's g-prior. Suppose $p(\mathbf{w}_y) = \mathcal{N}(\mathbf{0}, \frac{1}{g}\boldsymbol{\Sigma}^2)$, and let $\mathbf{X} = \mathbf{R}\mathbf{V}^T$ be the SVD of $\mathbf{X}$, where $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ and $\mathbf{R}^T\mathbf{R} = \boldsymbol{\Sigma}^2 = \mathrm{diag}(\sigma_j^2)$ contains the squared singular values. Then one can show (West 2003) that

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, g\mathbf{V}^{-T}\boldsymbol{\Sigma}^{-2}\mathbf{V}^{-1}) = \mathcal{N}(\mathbf{0}, g(\mathbf{X}^T\mathbf{X})^{-1}) \quad (12.79)$$

So the dependence of the prior for $\mathbf{w}$ on $\mathbf{X}$ arises from the fact that $\mathbf{w}$ is derived indirectly by a joint model of $\mathbf{X}$ and $\mathbf{y}$.

The above discussion focussed on regression. (Guo 2009) generalizes CCA to the exponential family, which is more appropriate if $\mathbf{x}_i$ and/or y_i are discrete. Although we can no longer compute the conditional $p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$ in closed form, the model has a similar interpretation to the regression case, namely that we are predicting the response via a latent "bottleneck".

The basic idea of compressing $\mathbf{x}_i$ to predict $\mathbf{y}_i$ can be formulated using information theory. In particular, we might want to find an encoding distribution $p(\mathbf{z}|\mathbf{x})$ such that we minimize

$$\mathbb{I}(X; Z) - \beta \mathbb{I}(X; Y) \quad (12.80)$$

where $\beta \geq 0$ is some parameter controlling the tradeoff between compression and predictive accuracy. This is known as the **information bottleneck** (Tishby et al. 1999). Often Z is taken to be discrete, as in clustering. However, in the Gaussian case, IB is closely related to CCA (Chechik et al. 2005).

We can easily generalize CCA to the case where y_i is a vector of responses to be predicted, as in multi-label classification. (Ma et al. 2008; Williamson and Ghahramani 2008) used this model to perform collaborative filtering, where the goal is to predict $y_{ij} \in \{1, \ldots, 5\}$, the rating person i gives to movie j, where the "side information" $\mathbf{x}_i$ takes the form of a list of i's friends. The intuition behind this approach is that knowledge of who your friends are, as well as the ratings of all other users, should help predict which movies you will like. In general, any setting where the tasks are correlated could benefit from CCA. Once we adopt a probabilistic view, various extensions are straightforward. For example, we can easily generalize to the semi-supervised case, where we do not observe $\mathbf{y}_i$ for all i (Yu et al. 2006).

12.5.1.1 Discriminative supervised PCA

One problem with this model is that it puts as much weight on predicting the inputs $\mathbf{x}_i$ as the outputs $\mathbf{y}_i$. This can be partially alleviated by using a weighted objective of the following form (Rish et al. 2008):

$$\ell(\boldsymbol{\theta}) = \prod_i p(\mathbf{y}_i|\boldsymbol{\eta}_{iy})^{\alpha_y} p(\mathbf{x}_i|\boldsymbol{\eta}_{ix})^{\alpha_x} \tag{12.81}$$

where the α_m control the relative importance of the data sources, and $\boldsymbol{\eta}_{im} = \mathbf{W}_m \mathbf{z}_i$. For Gaussian data, we can see that α_m just controls the noise variance:

$$\ell(\boldsymbol{\theta}) \propto \prod_i \exp(-\frac{1}{2}\alpha_x ||\mathbf{x}_i^T - \boldsymbol{\eta}_{ix}||^2) \exp(-\frac{1}{2}\alpha_y ||\mathbf{y}_i^T - \boldsymbol{\eta}_{iy}||^2) \tag{12.82}$$

This interpretation holds more generally for the exponential family. Note, however, that it is hard to estimate the α_m parameters, because changing them changes the normalization constant of the likelihood. We give an alternative approach to weighting $\mathbf{y}$ more heavily below.

12.5.2 Partial least squares

The technique of **partial least squares** (**PLS**) (Gustafsson 2001; Sun et al. 2009) is an asymmetric or more "discriminative" form of supervised PCA. The key idea is to allow some of the (co)variance in the input features to be explained by its own subspace, $\mathbf{z}_i^x$, and to let the rest of the subspace, $\mathbf{z}_i^s$, be shared between input and output. The model has the form

$$\begin{aligned} p(\mathbf{z}_i) &= \mathcal{N}(\mathbf{z}_i^s|\mathbf{0}, \mathbf{I}_{L_s})\mathcal{N}(\mathbf{z}_i^x|\mathbf{0}, \mathbf{I}_{L_x}) & (12.83)\\ p(\mathbf{y}_i|\mathbf{z}_i) &= \mathcal{N}(\mathbf{W}_y \mathbf{z}_i^s + \boldsymbol{\mu}_y, \sigma^2 \mathbf{I}_{D_y}) & (12.84)\\ p(\mathbf{x}_i|\mathbf{z}_i) &= \mathcal{N}(\mathbf{W}_x \mathbf{z}_i^s + \mathbf{B}_x \mathbf{z}_i^x + \boldsymbol{\mu}_x, \sigma^2 \mathbf{I}_{D_x}) & (12.85) \end{aligned}$$

See Figure 12.19(b). The corresponding induced distribution on the visible variables has the form

$$p(\mathbf{v}_i|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{v}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I})d\mathbf{z}_i = \mathcal{N}(\mathbf{v}_i|\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}) \tag{12.86}$$

where $\mathbf{v}_i = (\mathbf{x}_i; \mathbf{y}_i)$, $\boldsymbol{\mu} = (\boldsymbol{\mu}_y; \boldsymbol{\mu}_x)$ and

$$\begin{aligned} \mathbf{W} &= \begin{pmatrix} \mathbf{W}_y & \mathbf{0} \\ \mathbf{W}_x & \mathbf{B}_x \end{pmatrix} & (12.87)\\ \mathbf{W}\mathbf{W}^T &= \begin{pmatrix} \mathbf{W}_y \mathbf{W}_y^T & \mathbf{W}_x \mathbf{W}_x^T \\ \mathbf{W}_x \mathbf{W}_x^T & \mathbf{W}_x \mathbf{W}_x^T + \mathbf{B}_x \mathbf{B}_x^T \end{pmatrix} & (12.88) \end{aligned}$$

We should choose L large enough so that the shared subspace does not capture covariate-specific variation.

This model can be easily generalized to discrete data using the exponential family (Virtanen 2010).

12.5.3 Canonical correlation analysis

Canonical correlation analysis or **CCA** is like a symmetric unsupervised version of PLS: it allows each view to have its own "private" subspace, but there is also a shared subspace. If we have two observed variables, $\mathbf{x}_i$ and $\mathbf{y}_i$, then we have three latent variables, $\mathbf{z}_i^s \in \mathbb{R}^{L_0}$ which is shared, $\mathbf{z}_i^x \in \mathbb{R}^{L_x}$ and $\mathbf{z}_i^y \in \mathbb{R}^{L_y}$ which are private. We can write the model as follows (Bach and Jordan 2005):

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i^s|\mathbf{0}, \mathbf{I}_{L_s})\mathcal{N}(\mathbf{z}_i^x|\mathbf{0}, \mathbf{I}_{L_x})\mathcal{N}(\mathbf{z}_i^y|\mathbf{0}, \mathbf{I}_{L_y}) \tag{12.89}$$

$$p(\mathbf{x}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{x}_i|\mathbf{B}_x\mathbf{z}_i^x + \mathbf{W}_x\mathbf{z}_i^s + \boldsymbol{\mu}_x, \sigma^2\mathbf{I}_{D_x}) \tag{12.90}$$

$$p(\mathbf{y}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{y}_i|\mathbf{B}_y\mathbf{z}_i^y + \mathbf{W}_y\mathbf{z}_i^s + \boldsymbol{\mu}_y, \sigma^2\mathbf{I}_{D_y}) \tag{12.91}$$

See Figure 12.19(c). The corresponding observed joint distribution has the form

$$p(\mathbf{v}_i|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{v}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I})d\mathbf{z}_i = \mathcal{N}(\mathbf{v}_i|\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}_D) \tag{12.92}$$

where

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_x & \mathbf{B}_x & \mathbf{0} \\ \mathbf{W}_y & \mathbf{0} & \mathbf{B}_y \end{pmatrix} \tag{12.93}$$

$$\mathbf{W}\mathbf{W}^T = \begin{pmatrix} \mathbf{W}_x\mathbf{W}_x^T + \mathbf{B}_x\mathbf{B}_x^T & \mathbf{W}_x\mathbf{W}_y^T \\ \mathbf{W}_y\mathbf{W}_y^T & \mathbf{W}_y\mathbf{W}_y^T + \mathbf{B}_y\mathbf{B}_y^T \end{pmatrix} \tag{12.94}$$

One can compute the MLE for this model using EM. (Bach and Jordan 2005) show that the resulting MLE is equivalent (up to rotation and scaling) to the classical, non-probabilistic view. However, the advantages of the probabilistic view are many: we can trivially generalize to $M > 2$ observed variables; we can create mixtures of CCA (Viinikanoja et al. 2010); we can create sparse versions of CCA using ARD (Archambeau and Bach 2008); we can generalize to the exponential family (Klami et al. 2010); we can perform Bayesian inference of the parameters (Wang 2007; Klami and Kaski 2008); we can handle non-parametric sparsity-promoting priors for $\mathbf{W}$ and $\mathbf{B}$ (Rai and Daume 2009); and so on.

12.6 Independent Component Analysis (ICA)

Consider the following situation. You are in a crowded room and many people are speaking. Your ears essentially act as two microphones, which are listening to a linear combination of the different speech signals in the room. Your goal is to deconvolve the mixed signals into their constituent parts. This is known as the **cocktail party problem**, and is an example of **blind signal separation** (BSS), or **blind source separation**, where "blind" means we know "nothing" about the source of the signals. Besides the obvious applications to acoustic signal processing, this problem also arises when analysing EEG and MEG signals, financial data, and any other dataset (not necessarily temporal) where latent sources or factors get mixed together in a linear way.

We can formalize the problem as follows. Let $\mathbf{x}_t \in \mathbb{R}^D$ be the observed signal at the sensors at "time" t, and $\mathbf{z}_t \in \mathbb{R}^L$ be the vector of source signals. We assume that

$$\mathbf{x}_t = \mathbf{W}\mathbf{z}_t + \boldsymbol{\epsilon}_t \tag{12.95}$$

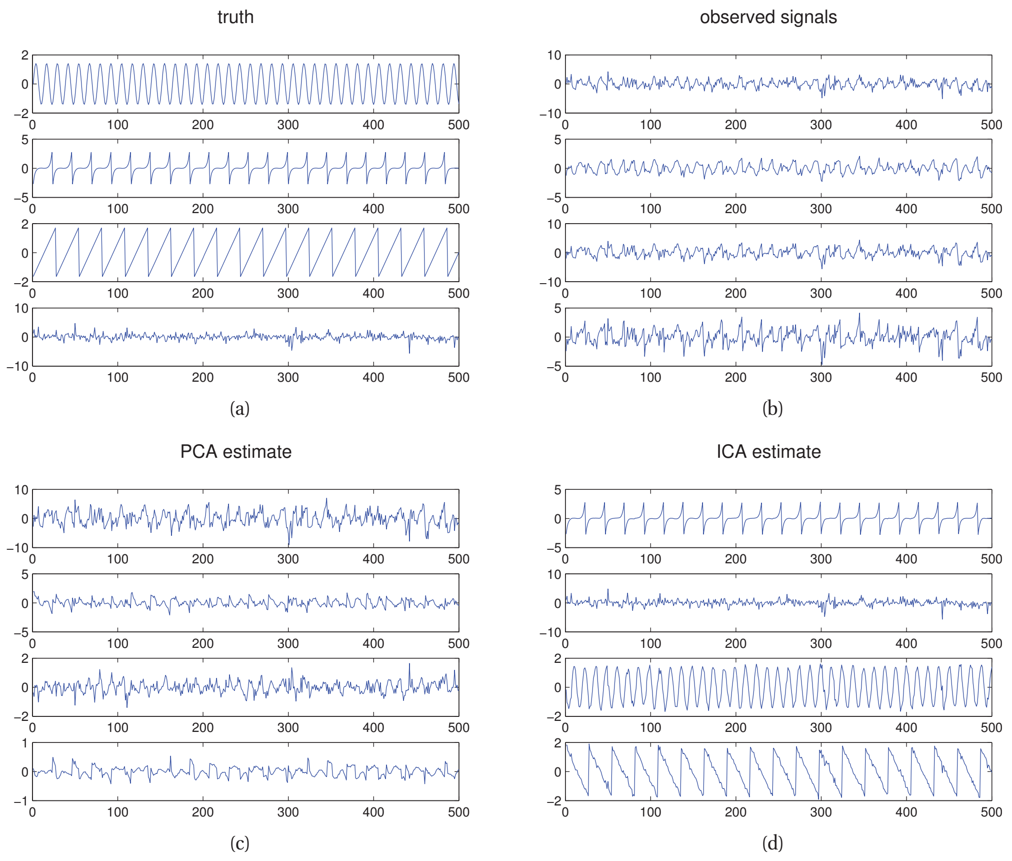


Figure 12.20 Illustration of ICA applied to 500 iid samples of a 4d source signal. (a) Latent signals. (b) Observations. (c) PCA estimate. (d) ICA estimate. Figure generated by `icaDemo`, written by Aapo Hyvarinen.

where $\mathbf{W}$ is an $D \times L$ matrix, and $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Psi})$. In this section, we treat each time point as an independent observation, i.e., we do not model temporal correlation (so we could replace the t index with i, but we stick with t to be consistent with much of the ICA literature). The goal is to infer the source signals, $p(\mathbf{z}_t|\mathbf{x}_t, \boldsymbol{\theta})$, as illustrated in Figure 12.20. In this context, $\mathbf{W}$ is called the **mixing matrix**. If $L = D$ (number of sources = number of sensors), it will be a square matrix. Often we will assume the noise level, $|\boldsymbol{\Psi}|$, is zero, for simplicity.

So far, the model is identical to factor analysis (or PCA if there is no noise, except we don't in general require orthogonality of $\mathbf{W}$). However, we will use a different prior for $p(\mathbf{z}_t)$. In PCA, we assume each source is independent, and has a Gaussian distribution

$$p(\mathbf{z}_t) = \prod_{j=1}^{L} \mathcal{N}(z_{tj}|0, 1) \tag{12.96}$$

We will now relax this Gaussian assumption and let the source distributions be any *non-Gaussian*

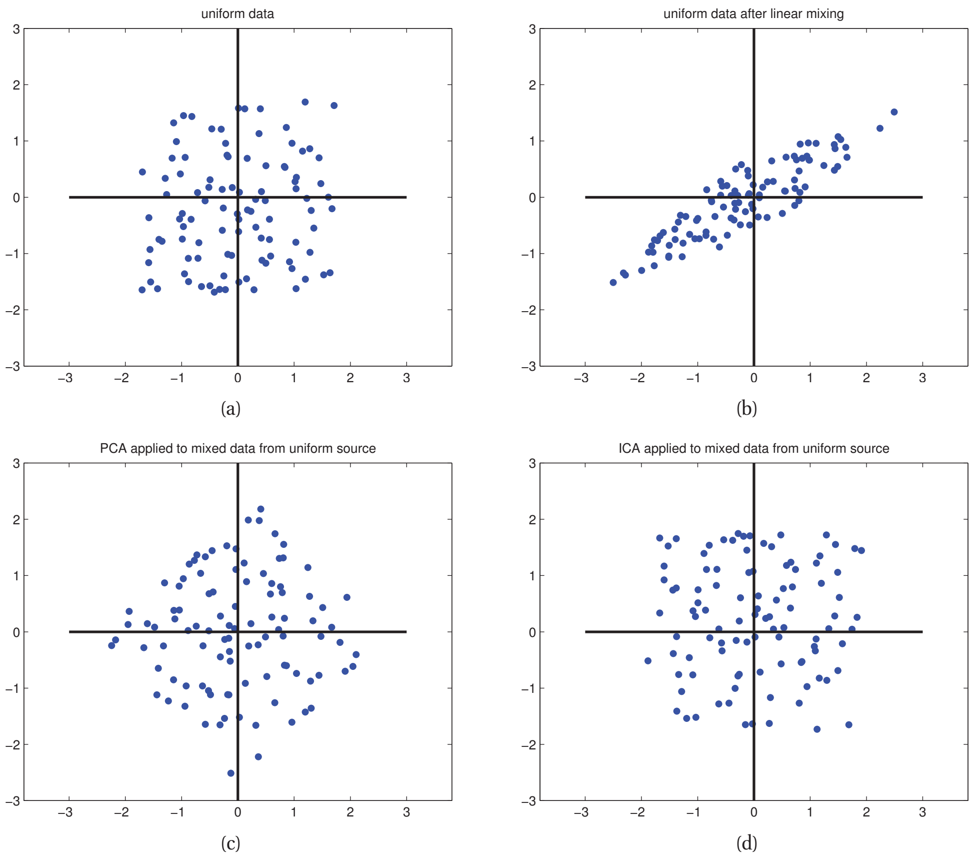


Figure 12.21 Illustration of ICA and PCA applied to 100 iid samples of a 2d source signal with a uniform distribution. (a) Latent signals. (b) Observations. (c) PCA estimate. (d) ICA estimate. Figure generated by `icaDemoUniform`, written by Aapo Hyvarinen.

distribution

$$p(\mathbf{z}_t) = \prod_{j=1}^{L} p_j(z_{tj}) \tag{12.97}$$

Without loss of generality, we can constrain the variance of the source distributions to be 1, because any other variance can be modelled by scaling the rows of $\mathbf{W}$ appropriately. The resulting model is known as **independent component analysis** or **ICA**.

The reason the Gaussian distribution is disallowed as a source prior in ICA is that it does not permit unique recovery of the sources, as illustrated in Figure 12.20(c). This is because the PCA likelihood is invariant to any orthogonal transformation of the sources $\mathbf{z}_t$ and mixing matrix $\mathbf{W}$. PCA can recover the best linear subspace in which the signals lie, but cannot uniquely recover the signals themselves.

To illustrate this, suppose we have two independent sources with uniform distributions, as shown in Figure 12.21(a). Now suppose we have the following mixing matrix

$$\mathbf{W} = \begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \tag{12.98}$$

Then we observe the data shown in Figure 12.21(b) (assuming no noise). If we apply PCA followed by scaling to this, we get the result in Figure 12.21(c). This corresponds to a whitening of the data. To uniquely recover the sources, we need to perform an additional rotation. The trouble is, there is no information in the symmetric Gaussian posterior to tell us which angle to rotate by. In a sense, PCA solves "half" of the problem, since it identifies the linear subspace; all that ICA has to do is then to identify the appropriate rotation. (Hence we see that ICA is not that different from methods such as varimax, which seek good rotations of the latent factors to enhance interpretability.)

Figure 12.21(d) shows that ICA can recover the source, up to a permutation of the indices and possible sign change. ICA requires that $\mathbf{W}$ is square and hence invertible. In the non-square case (e.g., where we have more sources than sensors), we cannot uniquely recover the true signal, but we can compute the posterior $p(\mathbf{z}_t|\mathbf{x}_t, \hat{\mathbf{W}})$, which represents our beliefs about the source. In both cases, we need to estimate $\mathbf{W}$ as well as the source distributions p_j. We discuss how to do this below.

12.6.1 Maximum likelihood estimation

In this section, we discuss ways to estimate square mixing matrices $\mathbf{W}$ for the noise-free ICA model. As usual, we will assume that the observations have been centered; hence we can also assume $\mathbf{z}$ is zero-mean. In addition, we assume the observations have been whitened, which can be done with PCA.

If the data is centered and whitened, we have $\mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right] = \mathbf{I}$. But in the noise free case, we also have

$$\text{cov}\left[\mathbf{x}\right] = \mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right] = \mathbf{W}\mathbb{E}\left[\mathbf{z}\mathbf{z}^T\right]\mathbf{W}^T = \mathbf{W}\mathbf{W}^T \tag{12.99}$$

Hence we see that $\mathbf{W}$ must be orthogonal. This reduces the number of parameters we have to estimate from D^2 to $D(D-1)/2$. It will also simplify the math and the algorithms.

Let $\mathbf{V} = \mathbf{W}^{-1}$; these are often called the **recognition weights**, as opposed to $\mathbf{W}$, which are the **generative weights**.[4]

Since $\mathbf{x} = \mathbf{W}\mathbf{z}$, we have, from Equation 2.89,

$$p_x(\mathbf{W}\mathbf{z}_t) = p_z(\mathbf{z}_t)|\det(\mathbf{W}^{-1})| = p_z(\mathbf{V}\mathbf{x}_t)|\det(\mathbf{V})| \tag{12.100}$$

Hence we can write the log-likelihood, assuming T iid samples, as follows:

$$\frac{1}{T}\log p(\mathcal{D}|\mathbf{V}) = \log|\det(\mathbf{V})| + \frac{1}{T}\sum_{j=1}^{L}\sum_{t=1}^{T}\log p_j(\mathbf{v}_j^T\mathbf{x}_t) \tag{12.101}$$

4. In the literature, it is common to denote the generative weights by $\mathbf{A}$ and the recognition weights by $\mathbf{W}$, but we are trying to be consistent with the notation used earlier in this chapter.

where $\mathbf{v}_j$ is the j'th row of $\mathbf{V}$. Since we are constraining $\mathbf{V}$ to be orthogonal, the first term is a constant, so we can drop it. We can also replace the average over the data with an expectation operator to get the following objective

$$\text{NLL}(\mathbf{V}) = \sum_{j=1}^{L} \mathbb{E}\left[G_j(z_j)\right] \tag{12.102}$$

where $z_j = \mathbf{v}_j^T \mathbf{x}$ and $G_j(z) \triangleq -\log p_j(z)$. We want to minimize this subject to the constraint that the rows of $\mathbf{V}$ are orthogonal. We also want them to be unit norm, since this ensures that the variance of the factors is unity (since, with whitened data, $\mathbb{E}\left[\mathbf{v}_j^T \mathbf{x}\right] = ||\mathbf{v}_j||^2$), which is necessary to fix the scale of the weights. In otherwords, $\mathbf{V}$ should be an orthonormal matrix.

It is straightforward to derive a gradient descent algorithm to fit this model; however, it is rather slow. One can also derive a faster algorithm that follows the natural gradient; see e.g., (MacKay 2003, ch 34) for details. A popular alternative is to use an approximate Newton method, which we discuss in Section 12.6.2. Another approach is to use EM, which we discuss in Section 12.6.3.

12.6.2 The FastICA algorithm

We now describe the **fast ICA** algorithm, based on (Hyvarinen and Oja 2000), which we will show is an approximate Newton method for fitting ICA models.

For simplicity of presentation, we initially assume there is only one latent factor. In addition, we initially assume all source distributions are known and are the same, so we can just write $G(z) = -\log p(z)$. Let $g(z) = \frac{d}{dz} G(z)$. The constrained objective, and its gradient and Hessian, are given by

$$f(\mathbf{v}) = \mathbb{E}\left[G(\mathbf{v}^T \mathbf{x})\right] + \lambda(1 - \mathbf{v}^T \mathbf{v}) \tag{12.103}$$

$$\nabla f(\mathbf{v}) = \mathbb{E}\left[\mathbf{x} g(\mathbf{v}^T \mathbf{x})\right] - \beta \mathbf{v} \tag{12.104}$$

$$\mathbf{H}(\mathbf{v}) = \mathbb{E}\left[\mathbf{x}\mathbf{x}^T g'(\mathbf{v}^T \mathbf{x})\right] - \beta \mathbf{I} \tag{12.105}$$

where $\beta = 2\lambda$ is a Lagrange multiplier. Let us make the approximation

$$\mathbb{E}\left[\mathbf{x}\mathbf{x}^T g'(\mathbf{v}^T \mathbf{x})\right] \approx \mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right] \mathbb{E}\left[g'(\mathbf{v}^T \mathbf{x})\right] = \mathbb{E}\left[g'(\mathbf{v}^T \mathbf{x})\right] \tag{12.106}$$

This makes the Hessian very easy to invert, giving rise to the following Newton update:

$$\mathbf{v}^* \triangleq \mathbf{v} - \frac{\mathbb{E}\left[\mathbf{x} g(\mathbf{v}^T \mathbf{x})\right] - \beta \mathbf{v}}{\mathbb{E}\left[g'(\mathbf{v}^T \mathbf{x})\right] - \beta} \tag{12.107}$$

One can rewrite this in the following way

$$\mathbf{v}^* \triangleq \mathbb{E}\left[\mathbf{x} g(\mathbf{v}^T \mathbf{x})\right] - \mathbb{E}\left[g'(\mathbf{v}^T \mathbf{x})\right] \mathbf{v} \tag{12.108}$$

(In practice, the expectations can be replaced by Monte Carlo estimates from the training set, which gives an efficient online learning algorithm.) After performing this update, one should project back onto the constraint surface using

$$\mathbf{v}^{new} \triangleq \frac{\mathbf{v}^*}{||\mathbf{v}^*||} \tag{12.109}$$

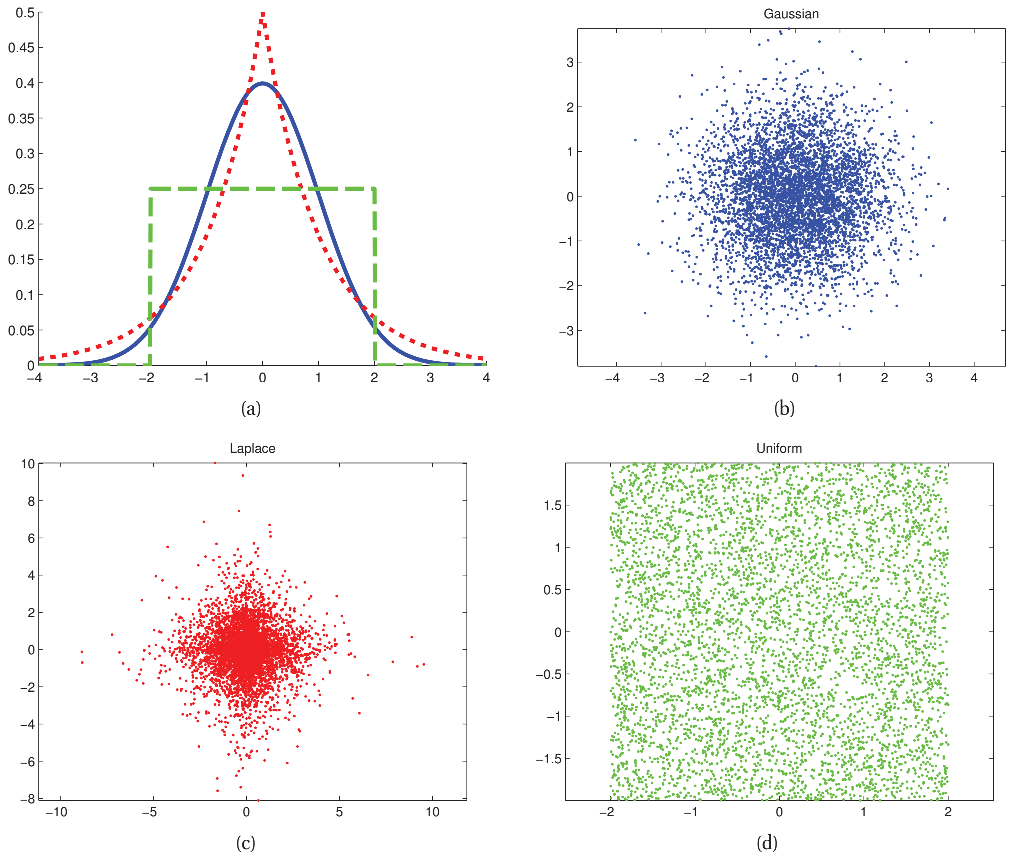


Figure 12.22 Illustration of Gaussian, sub-Gaussian (uniform) and super-Gaussian (Laplace) distributions in 1d and 2d. Figure generated by `subSuperGaussPlot`, written by Kevin Swersky.

One iterates this algorithm until convergence. (Due to the sign ambiguity of $\mathbf{v}$, the values of $\mathbf{v}$ may not converge, but the direction defined by this vector should converge, so one can assess convergence by monitoring $|\mathbf{v}^T\mathbf{v}^{new}|$, which should approach 1.)

Since the objective is not convex, there are multiple local optima. We can use this fact to learn multiple different weight vectors or **features**. We can either learn the features sequentially and then project out the part of $\mathbf{v}_j$ that lies in the subspace defined by earlier features, or we can learn them in parallel, and orthogonalize $\mathbf{V}$ in parallel. This latter approach is usually preferred, since, unlike PCA, the features are not ordered in any way. So the first feature is not "more important" than the second, and hence it is better to treat them symmetrically.

12.6.2.1 Modeling the source densities

So far, we have assumed that $G(z) = -\log p(z)$ is known. What kinds of models might be reasonable as signal priors? We know that using Gaussians (which correspond to quadratic functions for G) won't work. So we want some kind of non-Gaussian distribution. In general, there are several kinds of non-Gaussian distributions, such as the following:

- **Super-Gaussian distributions** These are distributions which have a big spike at the mean, and hence (in order to ensure unit variance) have heavy tails. The Laplace distribution is a classic example. See Figure 12.22. Formally, we say a distribution is **super-Gaussian** or **leptokurtic** ("lepto" coming from the Greek for "thin") if $\text{kurt}(z) > 0$, where $\text{kurt}(z)$ is the **kurtosis** of the distribution, defined by

  $$\text{kurt}(z) \triangleq \frac{\mu_4}{\sigma^4} - 3 \tag{12.110}$$

  where σ is the standard deviation, and μ_k is the k'th **central moment**, or moment about the mean:

  $$\mu_k \triangleq \mathbb{E}\left[(X - \mathbb{E}[X])^k\right] \tag{12.111}$$

  (So $\mu_1 = \mu$ is the mean, and $\mu_2 = \sigma^2$ is the variance.) It is conventional to subtract 3 in the definition of kurtosis to make the kurtosis of a Gaussian variable equal to zero.
- **Sub-Gaussian distributions** A **sub-Gaussian** or **platykurtic** ("platy" coming from the Greek for "broad") distribution has negative kurtosis. These are distributions which are much flatter than a Gaussian. The uniform distribution is a classic example. See Figure 12.22.
- **Skewed distributions** Another way to "be non-Gaussian" is to be asymmetric. One measure of this is **skewness**, defined by

  $$\text{skew}(z) \triangleq \frac{\mu_3}{\sigma^3} \tag{12.112}$$

  An example of a (right) skewed distribution is the gamma distribution (see Figure 2.9).

When one looks at the empirical distribution of many natural signals, such as images and speech, when passed through certain linear filters, they tend to be very super-Gaussian. This result holds both for the kind of linear filters found in certain parts of the brain, such as the simple cells found in the primary visual cortex, as well as for the kinds of linear filters used in signal processing, such as wavelet transforms. One obvious choice for modeling natural signals with ICA is therefore the Laplace distribution. For mean zero and variance 1, this has a log pdf given by

$$\log p(z) = -\sqrt{2}|z| - \log(\sqrt{2}) \tag{12.113}$$

Since the Laplace prior is not differentiable at the origin, it is more common to use other, smoother super-Gaussian distributions. One example is the logistic distribution. The corresponding log pdf, for the case where the mean is zero and the variance is 1 (so $\mu = 0$ and $s = \frac{\sqrt{3}}{\pi}$), is given by the following:

$$\log p(z) = -2 \log \cosh\left(\frac{\pi}{2\sqrt{3}} z\right) - \log \frac{4\sqrt{3}}{\pi} \tag{12.114}$$

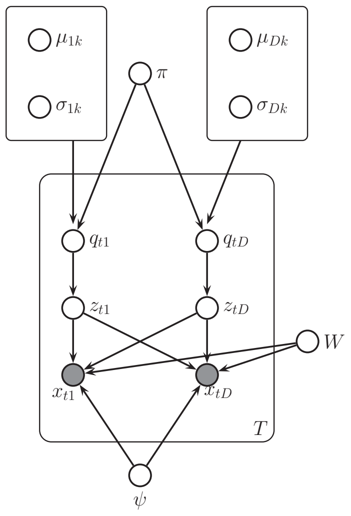


Figure 12.23 Modeling the source distributions using a mixture of univariate Gaussians (the independent factor analysis model of (Moulines et al. 1997; Attias 1999)).

Various ways of estimating $G(Z) = -\log p(z)$ are discussed in the seminal paper (Pham and Garrat 1997). However, when fitting ICA by maximum likelihood, it is not critical that the exact shape of the source distribution be known (although it is important to know whether it is sub or super Gaussian). Consequently, it is common to just use $G(z) = \sqrt{z}$ or $G(z) = \log\cosh(z)$ instead of the more complex expressions above.

12.6.3 Using EM

An alternative to assuming a particular form for $G(z)$, or equivalently for $p(z)$, is to use a flexible non-parametric density estimator, such as a mixture of (uni-variate) Gaussians:

$$p(q_j = k) = \pi_k \quad (12.115)$$

$$p(z_j | q_j = k) = \mathcal{N}(\mu_{j,k}, \sigma^2_{j,k}) \quad (12.116)$$

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{W}\mathbf{z}, \boldsymbol{\Psi}) \quad (12.117)$$

This approach was proposed in (Moulines et al. 1997; Attias 1999), and the corresponding graphical model is shown in Figure 12.23.

It is possible to derive an exact EM algorithm for this model. The key observation is that it is possible to compute $\mathbb{E}[\mathbf{z}_t|\mathbf{x}_t, \boldsymbol{\theta}]$ exactly by summing over all K^L combinations of the $\mathbf{q}_t$ variables, where K is the number of mixture components per source. (If this is too expensive, one can use a variational mean field approximation (Attias 1999).) We can then estimate all the source distributions in parallel by fitting a standard GMM to $\mathbb{E}[\mathbf{z}_t]$. When the source GMMs are known, we can compute the marginals $p_j(z_j)$ very easily, using

$$p_j(z_j) = \sum_{k=1}^{K} \pi_{j,k} \mathcal{N}(z_j | \mu_{j,k}, \sigma_{j,k}^2) \tag{12.118}$$

Given the p_j's, we can then use an ICA algorithm to estimate $\mathbf{W}$. Of course, these steps should be interleaved. The details can be found in (Attias 1999).

12.6.4 Other estimation principles *

It is quite common to estimate the parameters of ICA models using methods that seem different to maximum likelihood. We will review some of these methods below, because they give additional insight into ICA. However, we will also see that these methods in fact are equivalent to maximum likelihood after all. Our presentation is based on (Hyvarinen and Oja 2000).

12.6.4.1 Maximizing non-Gaussianity

An early approach to ICA was to find a matrix $\mathbf{V}$ such that the distribution $\mathbf{z} = \mathbf{V}\mathbf{x}$ is as far from Gaussian as possible. (There is a related approach in statistics called **projection pursuit**.) One measure of non-Gaussianity is kurtosis, but this can be sensitive to outliers. Another measure is the **negentropy**, defined as

$$\text{negentropy}(z) \triangleq \mathbb{H}\left(\mathcal{N}(\mu, \sigma^2)\right) - \mathbb{H}(z) \tag{12.119}$$

where $\mu = \mathbb{E}[z]$ and $\sigma^2 = \text{var}[z]$. Since the Gaussian is the maximum entropy distribution, this measure is always non-negative and becomes large for distributions that are highly non-Gaussian.

We can define our objective as maximizing

$$J(\mathbf{V}) = \sum_j \text{negentropy}(z_j) = \sum_j \mathbb{H}\left(\mathcal{N}(\mu_j, \sigma_j^2)\right) - \mathbb{H}(z_j) \tag{12.120}$$

where $\mathbf{z} = \mathbf{V}\mathbf{x}$. If we fix $\mathbf{V}$ to be orthogonal, and if we whiten the data, the covariance of $\mathbf{z}$ will be $\mathbf{I}$ independently of $\mathbf{V}$, so the first term is a constant. Hence

$$J(\mathbf{V}) = \sum_j -\mathbb{H}(z_j) + \text{const} = \sum_j \mathbb{E}[\log p(z_j)] + \text{const} \tag{12.121}$$

which we see is equal (up to a sign change, and irrelevant constants) to the log-likelihood in Equation 12.102.

12.6.4.2 Minimizing mutual information

One measure of dependence of a set of random variables is the **multi-information**:

$$I(\mathbf{z}) \triangleq \mathbb{KL}\left(p(\mathbf{z}) || \prod_j p(z_j)\right) = \sum_j \mathbb{H}(z_j) - \mathbb{H}(\mathbf{z}) \tag{12.122}$$

We would like to minimize this, since we are trying to find independent components. Put another way, we want the best possible factored approximation to the joint distribution.

Now since $\mathbf{z} = \mathbf{V}\mathbf{x}$, we have

$$I(\mathbf{z}) = \sum_j \mathbb{H}(z_j) - \mathbb{H}(\mathbf{V}\mathbf{x}) \tag{12.123}$$

If we constrain $\mathbf{V}$ to be orthogonal, we can drop the last term, since then $\mathbb{H}(\mathbf{V}\mathbf{x}) = \mathbb{H}(\mathbf{x})$ (since multiplying by $\mathbf{V}$ does not change the shape of the distribution), and $\mathbb{H}(\mathbf{x})$ is a constant which is is solely determined by the empirical distribution. Hence we have $I(\mathbf{z}) = \sum_j \mathbb{H}(z_j)$. Minimizing this is equivalent to maximizing the negentropy, which is equivalent to maximum likelihood.

12.6.4.3 Maximizing mutual information (infomax)

Instead of trying to minimize the mutual information between the components of $\mathbf{z}$, let us imagine a neural network where $\mathbf{x}$ is the input and $y_j = \phi(\mathbf{v}_j^T \mathbf{x}) + \epsilon$ is the noisy output, where ϕ is some nonlinear scalar function, and $\epsilon \sim \mathcal{N}(0, 1)$. It seems reasonable to try to maximize the information flow through this system, a principle known as **infomax**. (Bell and Sejnowski 1995). That is, we want to maximize the mutual information between $\mathbf{y}$ (the internal neural representation) and $\mathbf{x}$ (the observed input signal). We have $\mathbb{I}(\mathbf{x}; \mathbf{y}) = \mathbb{H}(\mathbf{y}) - \mathbb{H}(\mathbf{y}|\mathbf{x})$, where the latter term is constant if we assume the noise has constant variance. One can show that we can approximate the former term as follows

$$\mathbb{H}(\mathbf{y}) = \sum_{j=1}^{L} \mathbb{E}\left[\log \phi'(\mathbf{v}_j^T \mathbf{x})\right] + \log|\det(\mathbf{V})| \tag{12.124}$$

where, as usual, we can drop the last term if $\mathbf{V}$ is orthogonal. If we define $\phi(z)$ to be a cdf, then $\phi'(z)$ is its pdf, and the above expression is equivalent to the log likelihood. In particular, if we use a logistic nonlinearity, $\phi(z) = \text{sigm}(z)$, then the corresponding pdf is the logistic distribution, and $\log \phi'(z) = \log \cosh(z)$ (ignoring irrelevant constants). Thus we see that infomax is equivalent to maximum likelihood.

Exercises

Exercise 12.1 M step for FA

For the FA model, show that the MLE in the M step for $\mathbf{W}$ is given by Equation 12.23.

Exercise 12.2 MAP estimation for the FA model

Derive the M step for the FA model using conjugate priors for the parameters.

Exercise 12.3 Heuristic for assessing applicability of PCA

(Source: (Press 2005, Q9.8).). Let the empirical covariance matrix Σ have eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d > 0$. Explain why the variance of the evalues, $\sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(\lambda_i - \overline{\lambda})^2$ is a good measure of whether or not PCA would be useful for analysing the data (the higher the value of σ^2 the more useful PCA).

Exercise 12.4 Deriving the second principal component

a. Let

$$J(\mathbf{v}_2, \mathbf{z}_2) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - z_{i1}\mathbf{v}_1 - z_{i2}\mathbf{v}_2)^T(\mathbf{x}_i - z_{i1}\mathbf{v}_1 - z_{i2}\mathbf{v}_2) \tag{12.125}$$

Show that $\frac{\partial J}{\partial \mathbf{z}_2} = 0$ yields $z_{i2} = \mathbf{v}_2^T\mathbf{x}_i$.

b. Show that the value of $\mathbf{v}_2$ that minimizes

$$\tilde{J}(\mathbf{v}_2) = -\mathbf{v}_2^T\mathbf{C}\mathbf{v}_2 + \lambda_2(\mathbf{v}_2^T\mathbf{v}_2 - 1) + \lambda_{12}(\mathbf{v}_2^T\mathbf{v}_1 - 0) \tag{12.126}$$

is given by the eigenvector of $\mathbf{C}$ with the second largest eigenvalue. Hint: recall that $\mathbf{C}\mathbf{v}_1 = \lambda_1\mathbf{v}_1$ and $\frac{\partial \mathbf{x}^T\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = (\mathbf{A} + \mathbf{A}^T)\mathbf{x}$.

Exercise 12.5 Deriving the residual error for PCA

a. Prove that

$$||\mathbf{x}_i - \sum_{j=1}^{K} z_{ij}\mathbf{v}_j||^2 = \mathbf{x}_i^T\mathbf{x}_i - \sum_{j=1}^{K}\mathbf{v}_j^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{v}_j \tag{12.127}$$

Hint: first consider the case $K = 2$. Use the fact that $\mathbf{v}_j^T\mathbf{v}_j = 1$ and $\mathbf{v}_j^T\mathbf{v}_k = 0$ for $k \neq j$. Also, recall $z_{ij} = \mathbf{x}_i^T\mathbf{v}_j$.

b. Now show that

$$J_K \triangleq \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^T\mathbf{x}_i - \sum_{j=1}^{K}\mathbf{v}_j^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{v}_j\right) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^T\mathbf{x}_i - \sum_{j=1}^{K}\lambda_j \tag{12.128}$$

Hint: recall $\mathbf{v}_j^T\mathbf{C}\mathbf{v}_j = \lambda_j\mathbf{v}_j^T\mathbf{v}_j = \lambda_j$.

c. If $K = d$ there is no truncation, so $J_d = 0$. Use this to show that the error from only using $K < d$ terms is given by

$$J_K = \sum_{j=K+1}^{d}\lambda_j \tag{12.129}$$

Hint: partition the sum $\sum_{j=1}^{d}\lambda_j$ into $\sum_{j=1}^{K}\lambda_j$ and $\sum_{j=K+1}^{d}\lambda_j$.

Exercise 12.6 Derivation of Fisher's linear discriminant

Show that the maximum of $J(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}}$ is given by $\mathbf{S}_B\mathbf{w} = \lambda\mathbf{S}_W\mathbf{w}$
where $\lambda = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}}$. Hint: recall that the derivative of a ratio of two scalars is given by $\frac{d}{dx}\frac{f(x)}{g(x)} = \frac{f'g - fg'}{g^2}$, where $f' = \frac{d}{dx}f(x)$ and $g' = \frac{d}{dx}g(x)$. Also, recall that $\frac{d}{d\mathbf{x}}\mathbf{x}^T\mathbf{A}\mathbf{x} = (\mathbf{A} + \mathbf{A}^T)\mathbf{x}$.

Exercise 12.7 PCA via successive deflation

Let $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ be the first k eigenvectors with largest eigenvalues of $\mathbf{C} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$, i.e., the principal basis vectors. These satisfy

$$\mathbf{v}_j^T\mathbf{v}_k = \begin{cases} 0 & \text{if } j \neq k \\ 1 & \text{if } j = k \end{cases} \tag{12.130}$$

We will construct a method for finding the $\mathbf{v}_j$ sequentially.

As we showed in class, $\mathbf{v}_1$ is the first principal eigenvector of $\mathbf{C}$, and satisfies $\mathbf{C}\mathbf{v}_1 = \lambda_1 \mathbf{v}_1$. Now define $\tilde{\mathbf{x}}_i$ as the orthogonal projection of $\mathbf{x}_i$ onto the space orthogonal to $\mathbf{v}_1$:

$$\tilde{\mathbf{x}}_i = \mathbf{P}_{\perp \mathbf{v}_1} \, \mathbf{x}_i = (\mathbf{I} - \mathbf{v}_1 \mathbf{v}_1^T)\mathbf{x}_i \tag{12.131}$$

Define $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1; ...; \tilde{\mathbf{x}}_n]$ as the **deflated matrix** of rank $d-1$, which is obtained by removing from the d dimensional data the component that lies in the direction of the first principal direction:

$$\tilde{\mathbf{X}} = (\mathbf{I} - \mathbf{v}_1 \mathbf{v}_1^T)^T \mathbf{X} = (\mathbf{I} - \mathbf{v}_1 \mathbf{v}_1^T)\mathbf{X} \tag{12.132}$$

a. Using the facts that $\mathbf{X}^T\mathbf{X}\mathbf{v}_1 = n\lambda_1\mathbf{v}_1$ (and hence $\mathbf{v}_1^T\mathbf{X}^T\mathbf{X} = n\lambda_1\mathbf{v}_1^T$) and $\mathbf{v}_1^T\mathbf{v}_1 = 1$, show that the covariance of the deflated matrix is given by

$$\tilde{\mathbf{C}} \triangleq \frac{1}{n}\tilde{\mathbf{X}}^T\tilde{\mathbf{X}} = \frac{1}{n}\mathbf{X}^T\mathbf{X} - \lambda_1\mathbf{v}_1\mathbf{v}_1^T \tag{12.133}$$

b. Let $\mathbf{u}$ be the principal eigenvector of $\tilde{\mathbf{C}}$. Explain why $\mathbf{u} = \mathbf{v}_2$. (You may assume $\mathbf{u}$ is unit norm.)

c. Suppose we have a simple method for finding the leading eigenvector and eigenvalue of a pd matrix, denoted by $[\lambda, \mathbf{u}] = f(\mathbf{C})$. Write some pseudo code for finding the first K principal basis vectors of $\mathbf{X}$ that only uses the special f function and simple vector arithmetic, i.e., your code should not use SVD or the `eig` function. Hint: this should be a simple iterative routine that takes 2–3 lines to write. The input is $\mathbf{C}$, K and the function f, the output should be $\mathbf{v}_j$ and λ_j for $j = 1 : K$. Do not worry about being syntactically correct.

Exercise 12.8 Latent semantic indexing

(Source: de Freitas.). In this exercise, we study a technique called **latent semantic indexing**, which applies SVD to a document by term matrix, to create a low-dimensional embedding of the data that is designed to capture semantic similarity of words.

The file `lsiDocuments.pdf` contains 9 documents on various topics. A list of all the 460 unique words/terms that occur in these documents is in `lsiWords.txt`. A document by term matrix is in `lsiMatrix.txt`.

a. Let X be the transpose of `lsiMatrix`, so each column represents a document. Compute the SVD of X and make an approximation to it $\hat{X}$ using the first 2 singular values/ vectors. Plot the low dimensional representation of the 9 documents in 2D. You should get something like Figure 12.24.

b. Consider finding documents that are about alien abductions. If If you look at `lsiWords.txt`, there are 3 versions of this word, term 23 ("abducted"), term 24 ("abduction") and term 25 ("abductions"). Suppose we want to find documents containing the word "abducted". Documents 2 and 3 contain it, but document 1 does not. However, document 1 is clearly related to this topic. Thus LSI should also find document 1. Create a test document q containing the one word "abducted", and project it into the 2D subspace to make $\hat{q}$. Now compute the cosine similarity between $\hat{q}$ and the low dimensional representation of all the documents. What are the top 3 closest matches?

Exercise 12.9 Imputation in a FA model

Derive an expression for $p(\mathbf{x}_h|\mathbf{x}_v, \boldsymbol{\theta})$ for a FA model.

Exercise 12.10 Efficiently evaluating the PPCA density

Derive an expression for $p(\mathbf{x}|\hat{\mathbf{W}}, \hat{\sigma}^2)$ for the PPCA model based on plugging in the MLEs and using the matrix inversion lemma.

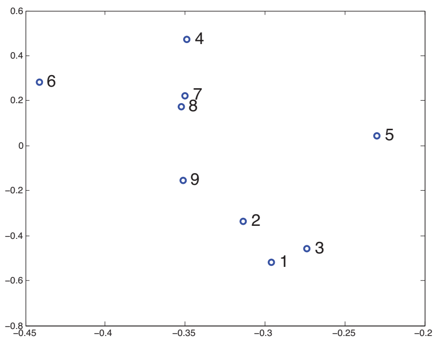


Figure 12.24 Projection of 9 documents into 2 dimensions. Figure generated by `lsiCode`.

Exercise 12.11 PPCA vs FA

(Source: Exercise 14.15 of (Hastie et al. 2009), due to Hinton.). Generate 200 observations from the following model, where $\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$: $x_{i1} = z_{i1}$, $x_{i2} = z_{i1} + 0.001 z_{i2}$, $x_{i3} = 10 z_{i3}$. Fit a FA and PCA model with 1 latent factor. Hence show that the corresponding weight vector $\mathbf{w}$ aligns with the maximal variance direction (dimension 3) in the PCA case, but with the maximal correlation direction (dimensions 1+2) in the case of FA.