13 Latent Variable Models

13.1 Latent Variable Models

A latent variable model for a random variable Z arises when the probability $P(Z)$ is actually a marginal distribution of $P(Z, X)$, where X is an unobserved random variable.

Example 13.1.1. Imagine we have two coins that appear identical, with probabilities θ_A and θ_B, respectively, of coming up heads. Randomly (with probability $\frac{1}{2}$) choose one of the coins. Let the random variable X be the choice of coin A or B. After choosing the coin, flip it ten times, and let Z be the total number of heads observed. For convenience, we use $\boldsymbol{\theta}$ to denote the pair (θ_A, θ_B). When $\boldsymbol{\theta}$ and X are known, the probability $P(Z|X, \boldsymbol{\theta})$ is easy to compute:

$$P(Z = z \,|\, X = A, \boldsymbol{\theta}) = \binom{10}{z} \theta_A^z (1 - \theta_A)^{10-z}$$
$$\text{and} \tag{13.1}$$
$$P(Z = z \,|\, X = B, \boldsymbol{\theta}) = \binom{10}{z} \theta_B^z (1 - \theta_B)^{10-z}.$$

The DGM for this model is

We leave the node X unfilled to indicate that it is unobserved.

Assuming that the hidden variable X takes only discrete values, and that X and Z depend on parameter θ, we may write

$$\begin{aligned} P(Z \,|\, \theta) &= \sum_x P(Z, X = x \,|\, \theta) \\ &= \sum_x P(Z \,|\, X = x, \theta) P(X = x \,|\, \theta). \end{aligned}$$

Example 13.1.2. Continuing with the same setup as Example 13.1.1, the probability $P(Z = z \,|\, \boldsymbol{\theta})$ is the sum

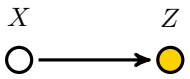
$$\begin{aligned} P(Z = z \,|\, \boldsymbol{\theta}) &= P(Z = z \,|\, X = A, \boldsymbol{\theta}) P(X = A \,|\, \boldsymbol{\theta}) \\ &\quad + P(Z = z \,|\, X = B, \boldsymbol{\theta}) P(X = B \,|\, \boldsymbol{\theta}) \\ &= \frac{1}{2}(P(Z = z \,|\, X = A, \boldsymbol{\theta}) + P(Z = z \,|\, X = B, \boldsymbol{\theta})) \end{aligned}$$

The probability $P(X \,|\, Z, \boldsymbol{\theta})$ can be computed by Bayes' rule

$$P(X \,|\, Z = z, \boldsymbol{\theta}) = \frac{P(Z = z \,|\, X, \boldsymbol{\theta}) P(X \,|\, \boldsymbol{\theta})}{P(Z = z \,|\, \boldsymbol{\theta})}.$$

13.1.1 Mixture Models

The two-coin situation of Examples 13.1.1 and 13.1.2 is an example of a *mixture model*, which is a latent variable model with DGM



where the latent variable X has a categorical distribution $X \sim \text{Cat}(\mathbf{w})$, with $\mathbf{w} = (w_1, \ldots, w_K)$ and $\sum_{k=1}^K w_k = 1$. The conditional distributions $p(Z = \mathbf{z} \,|\, X = k, \boldsymbol{\theta})$, which we write as $p_k(Z = \mathbf{z} \,|\, \boldsymbol{\theta})$ can be arbitrary. The overall model is called a mixture model because it is a convex combination (mixture) of the distributions p_k:

$$P(Z = \mathbf{z} \,|\, \boldsymbol{\theta}) = \sum_{k=1}^K w_k p_k(Z = \mathbf{z} \,|\, \boldsymbol{\theta}). \tag{13.2}$$

To draw from a mixture model, one first draws $X = k$ from $\text{Cat}(\mathbf{w})$ and then draws z from p_k.

Mixture models greatly expand our universe of parametrized distributions, allowing us to construct models that are considerably more intricate than the standard list of named distributions.

A mixture model is often trained (that is, the unknown parameters are estimated) using maximum likelihood estimation and a tool called *expectation maximization*.

Binomial Mixture Models

The two-coin example is a mixture of binomials model with $w_A = w_B = \frac{1}{2}$, where p_A and p_B are binomial, as given in (13.1).

Gaussian Mixture Models

A mixture of Gaussians is called a *Gaussian mixture model (GMM)*. In this case each $p_k(\mathbf{z} \,|\, \boldsymbol{\theta})$ is Gaussian

$$p_k(\mathbf{z} \,|\, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z} \,|\, \boldsymbol{\mu}_k, \Sigma_k) = |2\pi\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu}_k)^\mathsf{T}\Sigma_k^{-1}(\mathbf{z}-\boldsymbol{\mu}_k)}$$

and

$$p(\mathbf{z} \,|\, \boldsymbol{\theta}) = \sum_{k=1}^{K} w_k \mathcal{N}(\mathbf{z} \,|\, \boldsymbol{\mu}_k, \Sigma_k).$$

As in the binomial mixture, to draw from this GMM one first draws $X = k$ from $\text{Cat}(\mathbf{w})$ and then draws z from $\mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k)$.

13.1.2 Mixture of Experts

OLS is a great tool, but sometimes the basic Gaussian linear regression model of the form

$$P(y \,|\, \mathbf{x}) = \mathcal{N}(y \,|\, \mathbf{x}^\mathsf{T}\boldsymbol{\beta}, \sigma^2)$$

only describes the random variable y for certain values of $\mathbf{x}$, but for other values of $\mathbf{x}$ a different model should apply. For example there might be a phase change (such as water turning to ice or steam), and one regression model works well in one phase but a different model is needed to describe a different phase. This can be modeled with what is called a *mixture of experts*, where there is a discrete latent variable Z (the phase, for example), which is partially determined by $\mathbf{x}$, say by

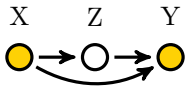
$$P(z \,|\, \mathbf{x}, \boldsymbol{\theta}) = \text{Cat}(\mathcal{S}(V^\mathsf{T}\mathbf{x})),$$

where $\mathcal{S}$ is the softmax function. For each value z of Z there is a different choice of parameters $(\boldsymbol{\beta}_z, \sigma_z^2)$ giving

$$P(y \,|\, \mathbf{x}, z) = \mathcal{N}(y \,|\, \mathbf{x}^\mathsf{T}\boldsymbol{\beta}_z, \sigma_z^2).$$

The idea is that each submodel $(\boldsymbol{\beta}_z, \sigma_z^2)$ is the expert in the region of the input space where $P(z \,|\, \mathbf{x}, \boldsymbol{\theta})$ is maximal; see Figure 13.1.

The directed graphical model for the mixture of experts is



Again, Z is unfilled to indicate that it is unobserved. The arrow from X to Z denotes that X and Z are not independent, and the two arrows into Y denote that Y is not independant from either X or Z.

As with a standard mixture model, a mixture of experts can be trained using expectation maximization.

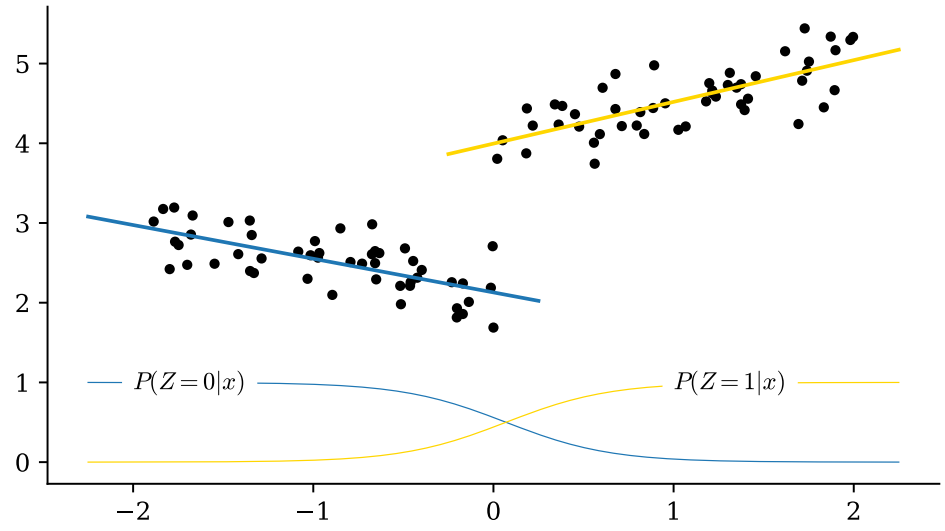
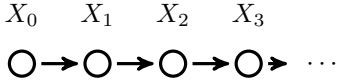


Figure 13.1: An example where a good linear fit to the data in one region is not a good fit in another region. Here the data (black) are modeled well by the heavy blue line when $x < 0$ and are modeled well by the heavy yellow line when $x > 0$. A mixture of experts model involves a latent variable Z (which line to use) influenced by X and influencing Y. The mixture of experts model plotted here uses x (horizontal axis) to choose $z \in \{0, 1\}$ according to $P(z \,|\, x)$, and then predicts y (vertical axis) on the blue line if $Z = 0$ or predicts y on the yellow line if $Z = 1$. The probabilities $P(Z = 0 \,|\, x)$ and $P(Z = 1 \,|\, x)$ are plotted as functions of x along the bottom of this figure as the thin blue and yellow curves, respectively.

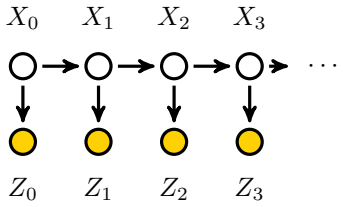
13.1.3 Hidden Markov Models

Consider the following example, due to Mark Stamp [Sta17, Chapter 2]. Imagine you are trying to deduce historical weather information from the rings in an old tree. In years when the weather was wetter, the tree tended to grow more, and when the weather was drier, the tree grew less. We cannot observe the weather directly—we can only observe the ring size (say small, medium, or large). Assuming that the weather is Markov (each year's weather is influenced by the previous year's weather, but is conditionally independent of earlier years' weather), this situation can be modeled with an important latent variable model called a *hidden Markov model (HMM)*.

In an HMM the latent variables form a Markov chain (in our example, this is the weather—dry or wet), denoted graphically as



The model also has observables $Z_0, Z_1, \ldots$ (in our example, these are the size of the tree rings) where each Z_i depends on X_i, but is conditionally independent from all other variables, given X_i. We denote this as



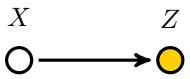
We discuss HMMs in more detail in Section 13.4. As with the mixture models and mixture of experts, one of the main tools for working with HMMs is expectation maximization.

13.2 Expectation Maximization

Recall that the maximum likelihood estimation (MLE) problem consists of finding the parameter $\boldsymbol{\theta}$ that maximizes the likelihood $L(\boldsymbol{\theta}) = P(\mathbf{D} \,|\, \boldsymbol{\theta})$, where $\mathbf{D} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N\}$ represents the data. This is equivalent to maximizing the log-likelihood

$$\ell(\boldsymbol{\theta}) = \log P(\mathbf{D} \,|\, \boldsymbol{\theta}),$$

If the data $\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_N$ are i.i.d., then $P(\mathbf{D} \,|\, \boldsymbol{\theta})$ factors as $P(\mathbf{D} \,|\, \boldsymbol{\theta}) = \prod_{i=1}^{N} P(Z = \mathbf{z}_i \,|\, \boldsymbol{\theta})$, and if this is a latent variable model with DGM



(13.3)

then

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P(\mathbf{z}_i \,|\, \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \sum_{x} P(\mathbf{z}_i, x \,|\, \boldsymbol{\theta}),$$

where x runs over all possible values of the latent variable X. This last sum is usually difficult to compute using standard optimization methods. This difficulty is due, in part, to the fact that the log of the sum makes differentiation messy.

Example 13.2.1. Consider again the two-coin setup of Examples 13.1.1 and 13.1.2. For each coin A or B the number of heads that comes up is binomially distributed with probability θ_A or θ_B, respectively. Imagine that the experiment (that is, a coin has been drawn and flipped ten times, and then replaced) has been repeated five times, giving a sequence of observations $\mathbf{D} = (z_1, \ldots, z_5) = (5, 9, 8, 4, 7)$, but the parameters θ_A, θ_B are unknown.

For each experiment i there is a latent random variable $X_i \in \{A, B\}$. To estimate the values of θ_A and θ_B using maximum likelihood, we must find

$$\begin{aligned}\widehat{\boldsymbol{\theta}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{5} \log P(z_i \,|\, \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{5} \log \left(\frac{1}{2}(P(Z = z_i \,|\, X_i = A, \boldsymbol{\theta}) + P(Z = z_i \,|\, X_i = B, \boldsymbol{\theta})) \right) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{5} \log(\theta_A^{z_i}(1-\theta_A)^{10-z_i} + (\theta_B^{z_i}(1-\theta_B)^{10-z_i})). \end{aligned} \tag{13.4}$$

But the obvious approach to maximization, by differentiating (13.4).

In this section we describe an iterative algorithm called *expectation maximization (EM)* for approximating the MLE $\widehat{\boldsymbol{\theta}}$ in latent variable models.

13.2.1 The Expectation Maximization Algorithm

Definition 13.2.2. *The* expectation maximization (EM) *algorithm is an iterative method for finding the maximum likelihood estimate* $\operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$ *of the parameter* $\boldsymbol{\theta}$ *in a latent variable model* (13.3). *Given an estimate* $\boldsymbol{\theta}^t$ *of* $\widehat{\boldsymbol{\theta}}_{MLE}$*, it constructs a new estimate* $\boldsymbol{\theta}^{t+1}$ *by completing the following two steps. Performed repeatedly, it gives a sequence* $(\boldsymbol{\theta}^k)_{k=0}^{\infty}$.

Expectation: *Compute*

$$q_i^t(x) = P(X_i = x \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t) \tag{13.5}$$

for all possible latent states x *and all* $i \in \{1, \ldots, n\}$. *Note that the superscript* t *here is an index, not an exponent. Use the computed values of* $q_i^t(x)$ *to construct*

$$Q^t(\boldsymbol{\theta}) = \sum_{i=1}^{n} Q_i^t(\boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{x} q_i^t(x) \log P(X_i = x, Z_i = \mathbf{z}_i \,|\, \boldsymbol{\theta}). \tag{13.6}$$

Maximization: *Compute*

$$\boldsymbol{\theta}^{t+1} = \operatorname*{argmax}_{\boldsymbol{\theta}} Q^t(\boldsymbol{\theta}). \tag{13.7}$$

In the next section we prove the following theorem, which suggests, but does not guarantee, that the sequence constructed by the EM algorithm should converge to a critical point of ℓ.

Theorem 13.2.3. *The EM algorithm gives a sequence* $(\boldsymbol{\theta}^t)_{t=0}^{\infty}$ *with the property that* $(\ell(\boldsymbol{\theta}^t))_{t=0}^{\infty}$ *is nondecreasing and bounded above, hence it must converge to some finite value. If, moreover,* $Q^t(\boldsymbol{\theta})$ *is strictly concave, then* $\ell(\boldsymbol{\theta}^{t+1}) > \ell(\boldsymbol{\theta}^t)$*, unless* $\boldsymbol{\theta}^t$ *is a critical point of* ℓ.

Nota Bene 13.2.4. Although Theorem 13.2.3 guarantees that the sequence $(\ell(\boldsymbol{\theta}^t))_{t=0}^{\infty}$ converges, it does *not* guarantee that the sequence of parameters $(\theta^t)_{t=0}^{\infty}$ converges. Moreover, although the sequence $(\ell(\boldsymbol{\theta}^t))_{t=0}^{\infty}$ is nondecreasing and in the strictly concave case can only stop increasing if it reaches a stationary point (where the derivative of ℓ vanishes), it does not guarantee that the limit point $\lim_{t\to\infty} \ell(\theta^t)$ is a stationary point, and thus does not guarantee that the limit point is a local maximum of ℓ.

Example 13.2.5. Returning to the two-coin example (see Examples 13.1.1–13.2.1), we wish to find $\widehat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$, where

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{5} \log P(z_i \,|\, \boldsymbol{\theta}) = \sum_{i=1}^{5} \log(\theta_A^{z_i}(1-\theta_A)^{10-z_i} + (\theta_B^{z_i}(1-\theta_B)^{10-z_i}))$$

and $\mathbf{D} = (z_1, \ldots, z_5) = (5, 9, 8, 4, 7)$.

To approximate $\widehat{\boldsymbol{\theta}} = (\widehat{\theta}_A, \widehat{\theta}_B)$ using the EM algorithm, assume that $\boldsymbol{\theta}^t = (\theta_A^t, \theta_B^t)$ is given (computed or guessed). The expectation step is to compute $q_i^t(x)$ for each $x \in \{A, B\}$ and each $i \in \{1, \ldots, 5\}$. This can be done with Bayes' rule, using the fact that $P(X_i = x \,|\, \boldsymbol{\theta}^t) = \frac{1}{2}$ and

$$P(Z_i = z_i \,|\, X_i = x, \boldsymbol{\theta}^t) = \binom{10}{z_i} (\theta_x^t)^{z_i} (1-\theta_x^t)^{10-z_i},$$

For example, in the case of $x = A$, we have

$$\begin{aligned}
q_i^t(A) &= P(X_i = A \,|\, Z_i = z_i, \boldsymbol{\theta}^t) \\
&= \frac{P(Z_i = z_i \,|\, X_i = A, \boldsymbol{\theta}^t) P(X_i = A \,|\, \boldsymbol{\theta}^t)}{P(Z_i = z_i \,|\, \boldsymbol{\theta}^t)} \\
&= \frac{\binom{10}{z_i}(\theta_A^t)^{z_i}(1-\theta_A^t)^{10-z_i}(\frac{1}{2})}{\binom{10}{z_i}(\theta_A^t)^{z_i}(1-\theta_A^t)^{10-z_i}(\frac{1}{2}) + \binom{10}{z_i}(\theta_B^t)^{z_i}(1-\theta_B^t)^{10-z_i}(\frac{1}{2})} \\
&= \frac{(\theta_A^t)^{z_i}(1-\theta_A^t)^{10-z_i}}{(\theta_A^t)^{z_i}(1-\theta_A^t)^{10-z_i} + (\theta_B^t)^{z_i}(1-\theta_B^t)^{10-z_i}}.
\end{aligned}$$

Computing $q_i^t(B)$ is similar.

The maximization step is to compute

$$\begin{aligned}\boldsymbol{\theta}^{t+1} &= \operatorname*{argmax}_{\boldsymbol{\theta}} Q^t(\boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} Q_i^t(\boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \sum_{x} q_i^t(x) \log P(X_i = x, Z_i = z_i \,|\, \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \sum_{x} q_i^t(x) \log(\binom{10}{z_i} \theta_x^{z_i}(1-\theta_x)^{10-z_i})\end{aligned}$$

Setting the derivatives $\frac{\partial Q}{\partial \theta_A}$ and $\frac{\partial Q}{\partial \theta_B}$ to zero and solving these for θ_A and θ_B gives (see Exercise 13.5)

$$\theta_A^{t+1} = \frac{\sum_{i=1}^n q_i^t(A) z_i}{\sum_{i=1}^n 10 q_i^t(A)} \qquad \text{and} \qquad \theta_B^{t+1} = \frac{\sum_{i=1}^n q_i^t(B) z_i}{\sum_{i=1}^n 10 q_i^t(B)}. \tag{13.8}$$

Repeat the process until the sequence seems to converge.

Remark 13.2.6. As with most iterative optimization routines, a bad initial guess can get you bad results. It's generally best, if possible, to choose an initial value that is relatively close to the true optimizer. It's also wise to avoid initial values with a lot of symmetry (for example, choosing $\theta_A = \theta_B$ in Example 13.2.5), including special values like those lying on the axes or at the origin (choosing θ_A or θ_B in $\{0, 1\}$ in Example 13.2.5 would generally be worse than choosing random values in $(0, 1)$).

Example 13.2.7. Consider a variant of Example 13.2.5 where the probabilities $\pi_A = P(A)$ and $\pi_B = P(B)$ of choosing the coins are unknown. In this case we wish to find $\widehat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$, where $\boldsymbol{\theta} = (\theta_A, \pi_A, \theta_B, \pi_B)$.

Assume that $\boldsymbol{\theta}^t = (\theta_A^t, \pi_A^t, \theta_B^t, \pi_B^t)$ is given (computed or guessed). As before, we compute the expectation step $q_i^t(x)$ using Bayes rule, but now $P(X_i = x \,|\, \boldsymbol{\theta}^t) = \pi_x$. For example, in the case of $x = A$, we have

$$\begin{aligned}q_i^t(A) &= P(X_i = A \,|\, Z_i = z_i, \boldsymbol{\theta}^t) \\ &= \frac{P(Z_i = z_i \,|\, X_i = A, \boldsymbol{\theta}^t) P(X_i = A \,|\, \boldsymbol{\theta}^t)}{P(Z_i = z_i \,|\, \boldsymbol{\theta}^t)} \\ &= \frac{\binom{10}{z_i}(\theta_A^t)^{z_i}(1-\theta_A^t)^{10-z_i}\pi_A}{\binom{10}{z_i}(\theta_A^t)^{z_i}(1-\theta_A^t)^{10-z_i}\pi_A + \binom{10}{z_i}(\theta_B^t)^{z_i}(1-\theta_B^t)^{10-z_i}\pi_B}.\end{aligned}$$

Computing $q_i^t(B)$ is similar.

The maximization step is to compute

$$\begin{aligned}\boldsymbol{\theta}^{t+1} &= \operatorname*{argmax}_{\boldsymbol{\theta}} Q^t(\boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} Q_i^t(\boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \sum_{x} q_i^t(x) \log P(X_i = x, Z_i = z_i \,|\, \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \sum_{x} q_i^t(x) \log(\pi_x \binom{10}{z_i} \theta_x^{z_i}(1-\theta_x)^{10-z_i})\end{aligned}$$

This is now a constrained optimization problem because $\pi_A + \pi_B = 1$. A straightforward computation with Lagrange multipliers shows that the optimizers θ^{t+1} are given by the same formula (13.8) as before, and the πs are given by

$$\pi_A^{t+1} = \frac{\sum_{i=1}^n q_i^t(A)}{\sum_{i=1}^n (q_i^t(A) + q_i^t(B))} \quad \text{and} \quad \pi_B^{t+1} = \frac{\sum_{i=1}^n q_i^t(B)}{\sum_{i=1}^n (q_i^t(A) + q_i^t(B))}. \tag{13.9}$$

Remark 13.2.8. In the coin flip examples, the choice of which coin is called coin A and which is called coin B is arbitrary. Therefore, for any choice of parameters $\boldsymbol{\theta} = (\theta_A, \pi_A, \theta_B, \pi_B)$ there is another choice $\boldsymbol{\theta}' = (\theta_B, \pi_B, \theta_A, \pi_A)$ with $\ell(\boldsymbol{\theta}) = \ell(\boldsymbol{\theta}')$. Except in the very special case when the maximizer $\widehat{\boldsymbol{\theta}}_{MLE}$ is symmetric, with $\boldsymbol{\theta} = \boldsymbol{\theta}'$, this means that ℓ does not have a unique maximizer, and the choice of which one the EM algorithm will find depends on things like the initial guess of $\boldsymbol{\theta}^0$. That is, the EM algorithm and any other form of maximum likelihood estimation will essentially only tell you that one of the coins has parameters θ_A, π_A and the other has parameters θ_B, π_B, but it cannot tell you which coin is which. A similar symmetry and indeterminacy exists in the the parameters for a GMM, which we treat in the next section.

13.2.2 The Idea of the EM Algorithm

In this section we describe the main idea behind the EM algorithm. In some ways it resembles Newton's method for optimization, but instead of finding a quadratic approximation and optimizing that, the EM algorithm finds a different function that approximates ℓ and always lies below the graph of ℓ. Given an estimate $\boldsymbol{\theta}^t$, the idea is to construct a function $\widetilde{Q}^t(\boldsymbol{\theta})$ of $\boldsymbol{\theta}$, satisfying the following properties:

(i) $\widetilde{Q}^t(\boldsymbol{\theta}) \leq \ell(\boldsymbol{\theta})$ for all $\boldsymbol{\theta}$.

(ii) $\widetilde{Q}^t(\boldsymbol{\theta}^t) = \ell(\boldsymbol{\theta}^t)$.

(iii) It is relatively easy to maximize $\widetilde{Q}^t(\boldsymbol{\theta})$.

Given such a $\widetilde{Q}^t$, compute a new estimate $\boldsymbol{\theta}^{t+1}$ as

$$\boldsymbol{\theta}^{t+1} = \operatorname*{argmax}_{\boldsymbol{\theta}} \widetilde{Q}^t(\boldsymbol{\theta}).$$

See Figure 13.2 for an illustration of one step of the EM algorithm.

Exercise 13.3 shows that

$$\ell(\boldsymbol{\theta}^{t+1}) \geq \ell(\boldsymbol{\theta}^t), \tag{13.10}$$

and the sequence $(\ell(\boldsymbol{\theta}^t))_{t=0}^{\infty}$ is bounded above and hence converges to a finite limit. Exercise 13.4 shows that if $\widetilde{Q}^t(\boldsymbol{\theta})$ is strictly concave, then $\ell(\boldsymbol{\theta}^{t+1}) > \ell(\boldsymbol{\theta}^t)$, unless $\boldsymbol{\theta}^t$ is a critical point of ℓ.

13.2.3 Derivation of the EM Algorithm

In this section we complete the proof of Theorem 13.2.3 by doing the following:

(i) Construct a function $\widetilde{Q}^t(\boldsymbol{\theta})$ satisfying the two conditions (i) and (ii) listed at the beginning of Section 13.2.2.

(ii) Show that maximizing the function $Q^t(\boldsymbol{\theta})$ defined in (13.6) is equivalent to maximizing $\widetilde{Q}^t(\boldsymbol{\theta})$.

Given a $\boldsymbol{\theta}^t$, we construct $\widetilde{Q}^t$ as a sum:

$$\widetilde{Q}^t(\boldsymbol{\theta}) = \sum_{i=1}^{n} \widetilde{Q}_i^t(\boldsymbol{\theta}).$$

To construct the function $\widetilde{Q}_i^t(\boldsymbol{\theta})$ first let

$$q_i^t(x) = P(X_i = x \,|\, \mathbf{D}, \boldsymbol{\theta}^t) = P(X_i = x \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t),$$

where the last equality follows from the fact that the samples X_i and Z_i are independent of all the other data X_j and Z_j for all $j \neq i$. The function q_i^t is a distribution (a p.d.f. or p.m.f.) for the possible states of the latent variable X_i. Note that q_i^t is independent of $\boldsymbol{\theta}$ but depends on $\boldsymbol{\theta}^t$.

Now set

$$\widetilde{Q}_i^t(\boldsymbol{\theta}) = \mathbb{E}_{X_i|\boldsymbol{\theta}^t}\left[\log\left(\frac{P(Z_i = \mathbf{z}_i, X_i \,|\, \boldsymbol{\theta})}{q_i^t(X_i)}\right)\right] \tag{13.11}$$

$$= \sum_x q_i^t(x) \log\left(\frac{P(Z_i = \mathbf{z}_i, X_i = x \,|\, \boldsymbol{\theta})}{q_i^t(x)}\right), \tag{13.12}$$

where the expectation is taken with respect to the distribution q_i^t. The computation of $\widetilde{Q}_i^t(\boldsymbol{\theta})$ is called the *expectation (E)* step of the algorithm, because $\widetilde{Q}^t$ is an expected value; whereas, the computation of $\boldsymbol{\theta}^{t+1} = \operatorname{argmax}_{\boldsymbol{\theta}} \widetilde{Q}_i^t(\boldsymbol{\theta})$ is the *maximization (M)* step. The EM algorithm alternates between these two steps.

Lemma 13.2.9. *The function $\widetilde{Q}^t(\boldsymbol{\theta})$, as defined in* (13.11) *satisfies conditions* (i) *and* (ii) *described above.*

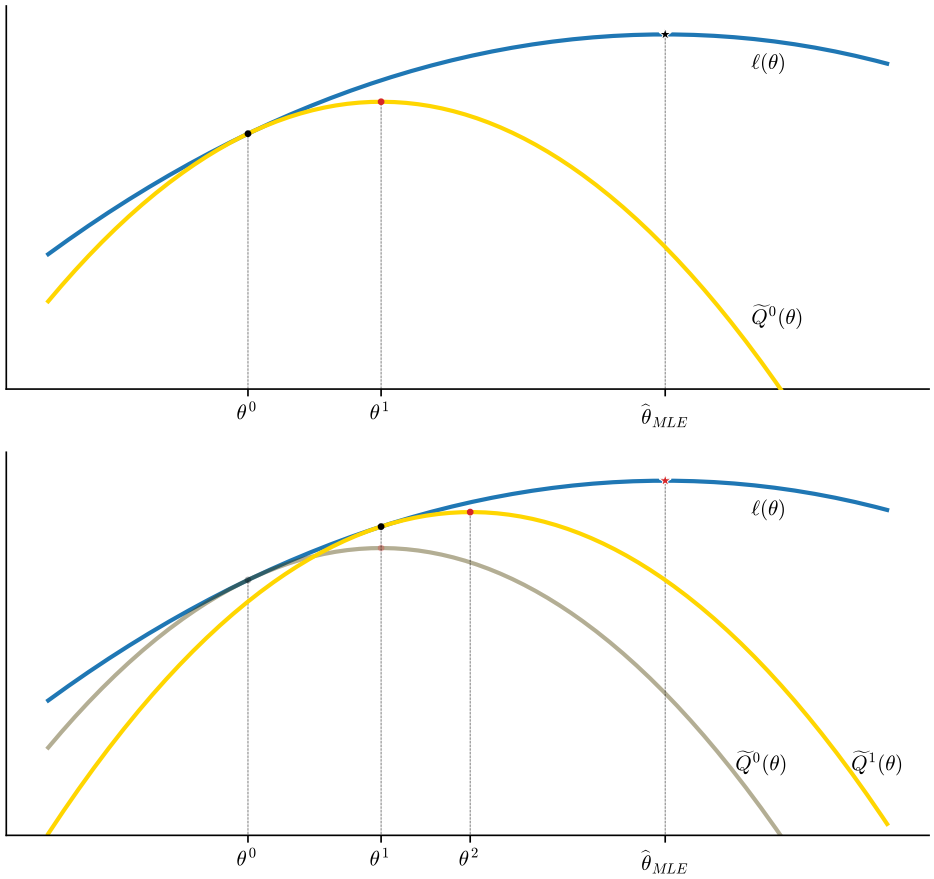


Figure 13.2: An illustration of the first step (top panel) of the EM algorithm for maximizing likelihood. For a given $\boldsymbol{\theta}^0$ find $\widetilde{Q}^0(\boldsymbol{\theta})$ (yellow), which always lies below the log likelihood function (blue), and which agrees (black dot) with the log likelihood function at $\boldsymbol{\theta} = \boldsymbol{\theta}^0$. The maximizer of $\widetilde{Q}^0(\boldsymbol{\theta})$ is the new estimate $\boldsymbol{\theta}^1$. The bottom panel shows the next step, with $\widetilde{Q}^1(\boldsymbol{\theta})$ (yellow), which always lies below the log likelihood function (blue), and which agrees (black dot) with the log likelihood function at $\boldsymbol{\theta} = \boldsymbol{\theta}^1$. The maximizer of $\widetilde{Q}^1(\boldsymbol{\theta})$ is the new estimate $\boldsymbol{\theta}^2$ Repeating the process gives a sequence that must climb upward on the log likelihood function. It often converges to the MLE estimate $\widehat{\boldsymbol{\theta}}_{MLE}$, but that is not always guaranteed.

Proof. Condition (i), that $\widetilde{Q}^t(\boldsymbol{\theta}) \leq \ell(\boldsymbol{\theta})$, follows from Jensen's inequality, since $\sum_x q_i^t(x) = 1$ and log is concave:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^n \log\left(\sum_x P(Z_i = \mathbf{z}_i, X_i = x \,|\, \boldsymbol{\theta})\right) = \sum_{i=1}^n \log\left(\sum_x q_i^t(x) \frac{P(Z_i = \mathbf{z}_i, X_i = x \,|\, \boldsymbol{\theta})}{q_i^t(x)}\right)$$
$$\geq \sum_{i=1}^n \sum_x q_i^t(x) \log\left(\frac{P(Z_i = \mathbf{z}_i, X_i = x \,|\, \boldsymbol{\theta})}{q_i^t(x)}\right) = \widetilde{Q}^t(\boldsymbol{\theta}).$$

To prove Condition (ii), that $\widetilde{Q}^t(\boldsymbol{\theta}^t) = \ell(\boldsymbol{\theta}^t)$, observe that

$$\begin{aligned}
\widetilde{Q}^t(\boldsymbol{\theta}^t) &= \sum_{i=1}^n \sum_x q_i^t(x) \log\left(\frac{P(Z_i = \mathbf{z}_i, X_i = x \,|\, \boldsymbol{\theta}^t)}{P(X_i = x \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t)}\right) \\
&= \sum_{i=1}^n \sum_x q_i^t(x) \log\left(\frac{P(X_i = x \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t) P(Z_i = \mathbf{z}_i \,|\, \boldsymbol{\theta}^t)}{P(X_i = x \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t)}\right) \\
&= \sum_{i=1}^n \log\left(P(Z_i = \mathbf{z}_i \,|\, \boldsymbol{\theta}^t)\right) \sum_x P(X_i = x \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t) \\
&= \sum_{i=1}^n \log P(Z_i = \mathbf{z}_i \,|\, \boldsymbol{\theta}^t) \\
&= \ell(\boldsymbol{\theta}^t).
\end{aligned}$$

Here the third equality follows from the fact that, after cancelling the common factor of $P(X_i = x \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t)$ in the fraction, the expression inside the logarithm is independent of x, and thus the expression with the logarithm can be pulled outside the sum. The fourth equality follows from the fact that $\sum_x P(X_i = x \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t) = 1$. □

Remark 13.2.10. Note that the argument for Condition (i) only used the fact that $\sum_x q_i^t(x) = 1$, so $\ell(\boldsymbol{\theta}) \geq \widetilde{Q}^t(\boldsymbol{\theta})$ regardless of the choice of distribution q, but it can be shown that the optimal choice for q is $q_i^t(x)$.

In the maximization step finding $\operatorname{argmax} \widetilde{Q}^t(\boldsymbol{\theta})$ is equivalent to solving a simpler problem:

$$\begin{aligned}
\boldsymbol{\theta}^{t+1} &= \operatorname*{argmax}_{\boldsymbol{\theta}} \widetilde{Q}^t(\boldsymbol{\theta}) \\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^n \sum_x q_i^t(x) \log\left(\frac{P(x, Z_i = \mathbf{z}_i \,|\, \boldsymbol{\theta})}{q_i^t(x)}\right) \\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^n \sum_x q_i^t(x) \log P(x, Z_i = \mathbf{z}_i \,|\, \boldsymbol{\theta}) - \sum_x q_i^t(x) \log(q_i^t(x)) \\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^n \sum_x q_i^t(x) \log P(x, Z_i = \mathbf{z}_i \,|\, \boldsymbol{\theta})
\end{aligned}$$

The last equality follows from the fact that $q_i^t(x)$ is independent of $\boldsymbol{\theta}$. This means that we can replace $\widetilde{Q}^t(\boldsymbol{\theta})$ by $Q^t(\boldsymbol{\theta}) = \sum_{i=1}^n Q_i^t(\boldsymbol{\theta})$, where

$$Q_i^t(\boldsymbol{\theta}) = \sum_x q_i^t(x) \log P(x, Z_i = \mathbf{z}_i \,|\, \boldsymbol{\theta}).$$

The previous argument shows that the two steps listed in Definition 13.2.2 are equivalent to items (i)–(iii) above, and hence produce a sequence $(\boldsymbol{\theta}^t)_{t=0}^\infty$ with the property that $(\ell(\boldsymbol{\theta}^t))_{t=0}^\infty$ is nondecreasing. This completes the proof of Theorem 13.2.3.

Remark 13.2.11. C.F.J. Wu proved convergence results in 1983.

13.3 EM for Gaussian Mixture Models

Recall that Gaussian mixture model (GMM) is a mixture model of the form

$$P(Z = \mathbf{z} \,|\, \boldsymbol{\theta}) = \sum_{k=1}^{K} w_k p_k(Z = \mathbf{z} \,|\, \boldsymbol{\theta}),$$

where

$$\begin{aligned} p_k(\mathbf{z} \,|\, \boldsymbol{\theta}) &= P(Z = \mathbf{z} \,|\, X = k, \boldsymbol{\theta}) \\ &= \mathcal{N}(\mathbf{z} \,|\, \boldsymbol{\mu}_k, \Sigma_k) = |2\pi\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu}_k)^\mathsf{T} \Sigma_k^{-1} (\mathbf{z}-\boldsymbol{\mu}_k)} \end{aligned}$$

and

$$P(\mathbf{z} \,|\, \boldsymbol{\theta}) = \sum_{k=1}^{K} w_k \mathcal{N}(\mathbf{z} \,|\, \boldsymbol{\mu}_k, \Sigma_k). \tag{13.13}$$

This section describes how to use the EM algorithm to estimate the parameters in a GMM. One of the main applications is clustering. We also discuss how the K-means algorithm can be thought of as a simplified version of EM for GMMs.

13.3.1 GMM for Clustering

One use of Gaussian mixture models is clustering, that is, the unsupervised learning task of identifying clusters in unlabeled data. Assume the data set is of the form $\mathbf{D} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, where each $\mathbf{z}_i$ belongs to some (unobserved) cluster $X = k$, and the distribution associated to the kth cluster is Gaussian, then the data are distributed according to a GMM with p.d.f. as given in (13.13). If $\boldsymbol{\theta} = (w_1, \boldsymbol{\mu}_1, \Sigma_1, \ldots, w_k, \boldsymbol{\mu}_K, \Sigma_K)$ is known, then for each $\mathbf{z}$ we can compute the probability that $\mathbf{z}$ belongs to cluster k using Bayes' rule

$$P(X = k \,|\, \mathbf{z}, \boldsymbol{\theta}) = \frac{P(\mathbf{z} \,|\, X = k, \boldsymbol{\theta}) p(X = k \,|\, \boldsymbol{\theta})}{\sum_{k'} p(\mathbf{z} \,|\, X = k', \boldsymbol{\theta}) p(X = k' \,|\, \boldsymbol{\theta})} = \frac{\mathcal{N}(\mathbf{z} \,|\, \boldsymbol{\mu}_k, \Sigma_k)) w_k}{\sum_{k'} \mathcal{N}(\mathbf{z} \,|\, \boldsymbol{\mu}_{k'}, \Sigma_{k'})) w_{k'}}.$$

Assigning a point $\mathbf{z}$ to the cluster

$$\operatorname*{argmax}_k p(X = k \,|\, \mathbf{z}, \boldsymbol{\theta})$$

gives a clustering of the data. This assignment is identical to the GDA generative classifier described in Section 7.8.1. The difference is in how the model is trained. In GDA, the values of X are given, which makes training relatively easy. In the case of clustering, the labels/clusters are unobserved (latent), but we can still estimate them using EM.

The construction described above is sometimes called *hard clustering*, since every point $\mathbf{z}$ is assigned a unique label/cluster. In contrast, *soft clustering* amounts to assigning to $\mathbf{z}$ the vector of probabilities $(r_1(\mathbf{z}), \ldots, r_K(\mathbf{z}))$, where $r_k(\mathbf{z}) = p(X = k \,|\, \mathbf{z}, \boldsymbol{\theta})$. This allows us a more nuanced view of which clusters are most likely associated to each $\mathbf{z}$.

13.3.2 EM for GMM

Applying the EM algorithm to the case of a GMM with p.d.f. equal to $P(\mathbf{z}) = \sum_{k=1}^{K} w_k f_k(\mathbf{z})$ with $f_k(\mathbf{z}) = \mathcal{N}(\mathbf{z} \,|\, \boldsymbol{\mu}_k, \Sigma_k)$ for each k, we have

$$\begin{aligned} q_i^t(k) &= P(X_i = k \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t) \\ &= \frac{P(Z_i = \mathbf{z}_i \,|\, X_i = k, \boldsymbol{\theta}^t) P(X_i = k \,|\, \boldsymbol{\theta}^t)}{\sum_{k'=1}^{K} P(Z_i = \mathbf{z}_i \,|\, X_i = k', \boldsymbol{\theta}^t) P(X_i = k' \,|\, \boldsymbol{\theta}^t)} \\ &= \frac{|\Sigma_k^t|^{-\frac{1}{2}} \exp(-\frac{1}{2}(\mathbf{z}_i - \boldsymbol{\mu}_k^t)^\mathsf{T}(\Sigma_k^t)^{-1}(\mathbf{z}_i - \boldsymbol{\mu}_k^t)) w_k^t}{\sum_{k'=1}^{K} |\Sigma_{k'}^t|^{-\frac{1}{2}} \exp(-\frac{1}{2}(\mathbf{z}_i - \boldsymbol{\mu}_{k'}^t)^\mathsf{T}(\Sigma_{k'}^t)^{-1}(\mathbf{z}_i - \boldsymbol{\mu}_{k'}^t)) w_{k'}^t} \end{aligned} \tag{13.14}$$

And the function to maximize is

$$\begin{aligned} Q^t(\boldsymbol{\theta}) &= \sum_{i=1}^{n} \sum_{k \in \mathscr{X}} q_i^t(k) \log P(X_i = k, Z_i = \mathbf{z}_i \,|\, \boldsymbol{\theta}) \\ &= \sum_{i=1}^{n} \sum_{k \in \mathscr{X}} q_i^t(k) [\log(w_k) - \frac{1}{2} \log(|2\pi\Sigma_k|) - \frac{1}{2}(\mathbf{z}_i - \boldsymbol{\mu}_k)^\mathsf{T}(\Sigma_k)^{-1}(\mathbf{z}_i - \boldsymbol{\mu}_k)] \end{aligned} \tag{13.15}$$

A straightforward KKT argument (see Volume 2, Section 14.4) to account for the constraints $w_k \geq 0$ and $\sum_k w_k = 1$ shows that the maximizing w_k is

$$w_k^{t+1} = \frac{1}{n} \sum_{i=1}^{n} q_i^t(k). \tag{13.16}$$

Essentially the same argument that gives the usual MLE for a normal distribution shows that in this case we have

$$\begin{aligned} \boldsymbol{\mu}_k^{t+1} &= \frac{\sum_{i=1}^{n} q_i^t(k) \mathbf{z}_i}{\sum_{i=1}^{n} q_i^t(k)} \\ \Sigma_k^{t+1} &= \frac{\sum_{i=1}^{n} q_i^t(k) (\mathbf{z}_i - \boldsymbol{\mu}_k^{t+1})(\mathbf{z}_i - \boldsymbol{\mu}_k^{t+1})^\mathsf{T}}{\sum_{i=1}^{n} q_i^t(k)}. \end{aligned} \tag{13.17}$$

13.3.3 K-means as a Special Case of EM for GMM

The K-means algorithm is a simplified variant of EM for GMM. It assumes that $\Sigma_k = \sigma^2 I$ is the same for all k, and that $w_k = \frac{1}{K}$ for all k. Thus the only parameters that need estimating are the means $\boldsymbol{\mu}_k$.

Given estimates $\boldsymbol{\mu}_k^t$, the algorithm computes the quantity

$$\kappa_i^t = \operatorname*{argmax}_k P(X_i = k \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t), \tag{13.18}$$

which is just the value of k for which $\boldsymbol{\mu}_k^t$ is nearest to $\mathbf{z}_i$, so it is much easier to compute than $P(X_i = k \,|\, Z_i = \mathbf{z}_i, \boldsymbol{\theta}^t)$. The algorithm then substitutes the following rough approximation of (13.14)

$$q_i^t(k) \approx \begin{cases} 1 & \text{if } k = \kappa_i^t \\ 0 & \text{otherwise.} \end{cases} \tag{13.19}$$

Using the approximation (13.19) in (13.17) to compute $\boldsymbol{\mu}_k^{t+1}$ reduces to

$$\boldsymbol{\mu}_k^{t+1} = \frac{1}{N_k} \sum_{i:\kappa_i^t = k} \mathbf{z}_i, \tag{13.20}$$

where N_k is the number of $\mathbf{z}_i$ with $\kappa_i^t = k$.

Thus the algorithm reduces to the following:
Given K and estimated means $\boldsymbol{\mu}_k^t$

(i) For each i compute κ_i^t to be the k such that $\boldsymbol{\mu}_k^t$ is nearest to $\mathbf{z}_i$.

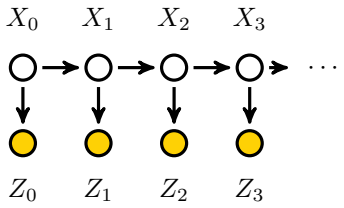
(ii) For each k compute $\boldsymbol{\mu}_k^{t+1}$, using (13.20).

(iii) Repeat until the means converge or the number of iterations reaches some predefined termination point.

K-means is faster to compute than the full EM for GMM algorithm, but if there is reason to believe that the probabilities w_k are not all equal or the covariances Σ_k are not all equal and diagonal, then you may get better clustering results using the full GMM for clustering.

13.4 Discrete Hidden Markov Models

In this section we discuss finite-state, finite-observation, temporally homogeneous hidden Markov models (HMMs). In the next section we discuss how to use EM to estimate the parameters of an HMM. Throughout the section we assume the state space $\mathscr{X}$ and the observation space $\mathscr{Z}$ are finite sets with $|\mathscr{X}| = n$ and $|\mathscr{Z}| = m$. The associated graphical model is as follows.



Because the Markov chain $\mathscr{X}$ is temporally homogeneous, it has constant conditional probability

$$a_{ij} = P(X_t = i \,|\, X_{t-1} = j),$$

giving a transition matrix $A = (a_{ij})$ (this matrix was called Q in Chapter 10, but since Q means something else in EM, we use A here instead). We further assume that Z_i is drawn from a finite set $\mathscr{Z} = \{1, \ldots, m\}$ with constant conditional probability

$$b_{ij} = P(Z_t = i \,|\, X_t = j),$$

giving an observation matrix $B = (b_{ij})$.

Example 13.4.1. Consider the tree-ring-weather problem described in Section 13.1.3. We assume that the average weather in a year takes only one of two states: wet (W) or Dry (D), so that $\mathscr{X} = \{W, D\}$. Assume the rings can only be classified into one of three values: small (S), medium (M), or large (L), so that $\mathscr{Z} = \{S, M, L\}$. Assume that the weather is a Markov chain with transition matrix $A = [a_{xx'}]$ with $a_{xx'} = P(X_t = x \,|\, X_{t-1} = x')$

$X_t \backslash X_{t-1}$	W	D
W	a_{WW}	a_{WD}
D	a_{DW}	a_{DD}.

The probabilities of observing each of the different ring sizes are given by matrix $B = [b_{zx}]$ with $b_{zx} = P(Z_t = z \,|\, X_t = x)$

$Z_t \backslash X_t$	W	D
S	b_{SW}	b_{SD}
M	b_{MW}	b_{MD}
L	b_{LW}	b_{LD}.

Finally, the initial distribution of states in $\mathscr{Z}$ is $\boldsymbol{\pi} = (\pi_W, \pi_D)$, where $\pi_x = P(X_0 = x)$.

Example 13.4.2. The optical character recognition (OCR) problem is the problem of identifying handwritten characters from a sequence of images. If we try to identify each character from its image without taking other images into account, this is just a standard classification problem. But if we also take into account the order of the images, then we can gain additional information (for example if the previous character was the letter q, then, assuming the sequence of images correspond to English words, then it is likely that the current character is the letter u).

This can be modeled simply with an HMM by letting $\mathscr{X}$ be the set of all possible characters and letting $\mathscr{Z}$ be the set of possible images. Of course this simple HMM does not take into account all the information—it does not track meaning, nor whether each resulting word belongs to the English language, nor even that every word has a vowel in it.

Once could estimate the transition probabilities $A_{xx'}$ by taking an English document and finding the probability that letter x' is followed by letter x. Similarly, one could try to estimate the observation probabilities B_{zx} by fitting a large labeled data set of pairs (x, z).

13.4.1 Four Questions about HMMs

Here are four questions that one might ask about an HMM.

(i) Given the parameters $\boldsymbol{\theta} = (\boldsymbol{\pi}, A, B)$, and an observed sequence $\mathbf{z} = (z_0, \ldots, z_{T-1})$ compute $P(\mathbf{z} \,|\, \boldsymbol{\theta})$.

(ii) Given the parameters $\boldsymbol{\theta}$, an observed sequence $\mathbf{z} = (z_0, \ldots, z_T)$, and a non-negative integer $k \leq T$, find the most likely state x_k at time k. That is, find $\operatorname{argmax}_{x_k} P(x_k \,|\, \mathbf{z}, \boldsymbol{\theta})$. In the special case that $k = T$, this problem is called *filtering*. For $k < T$ this problem is called *smoothing*.

(iii) Given the parameters $\boldsymbol{\theta}$ and an observed sequence $\mathbf{z} = (z_0, \ldots, z_{T-1})$, find the most likely sequence $\mathbf{x} = x_0, \ldots, x_{T-1}$. That is, find $\operatorname{argmax}_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta})$. The solution to this problem is sometimes called the *most likely explanation* or the *most likely path*.

(iv) Given $\mathbf{z}$, n, and m, find the most likely model parameters $\boldsymbol{\theta} = (A, B, \boldsymbol{\pi})$. This is sometimes called *parameter estimation* or *learning*.

In this section we focus on questions (i)–(iii). In the next section we discuss how to use EM to solve question (iv).

Nota Bene 13.4.3. Solving question (ii) for all $k \in \{0, \ldots, T-1\}$ might seem like it also solves (iii), but that is not necessarily true. Instead it gives a sequence that maximizes the number of states that are likely to have been correct, but not the sequence that is most likely to to have occurred. In the OCR problem, for example, solving question (ii) might tell you that the individual images in a four-character sequence are most likely *qvid*; that is, the first letter looks most like a q, the second looks most like a v and so forth. But solving question (iii) would tell you that the sequence is more likely either *quid* or *avid*, since v rarely follows q in English.

13.4.2 Probability of an Observation Sequence (Forward Pass)

If the observation sequence $\mathbf{z}$, the parameters $\boldsymbol{\theta}$, and the state sequence $\mathbf{x} = x_0, \ldots, x_{T-1}$ are all known, then the probability of $\mathbf{z}$ is

$$P(\mathbf{z} \,|\, \mathbf{x}, \boldsymbol{\theta}) = \prod_{t=0}^{T-1} P(z_t \,|\, x_t, \boldsymbol{\theta}) = b_{z_0 x_0} \cdots b_{z_{T-1} x_{T-1}}.$$

If, however, the state sequence $\mathbf{x}$ is not known, then the natural thing to try is summing over all the possible choices $\mathbf{x}$, using the law of total probability

$$\begin{aligned} P(\mathbf{z} \,|\, \boldsymbol{\theta}) &= \sum_{\mathbf{x}} P(\mathbf{z}, \mathbf{x} \,|\, \boldsymbol{\theta}) \\ &= \sum_{\mathbf{x}} P(\mathbf{x} \,|\, \boldsymbol{\theta}) P(\mathbf{z} \,|\, \mathbf{x}, \boldsymbol{\theta}) \\ &= \sum_{\mathbf{x}} \pi_{x_0} \prod_{t=1}^{T-1} a_{x_t x_{t-1}} \prod_{t=0}^{T-1} b_{z_t x_t}. \end{aligned}$$

Unfortunately, the number of possible sequences $\mathbf{x}$ is n^T, and each of the summands in the previous equation involves $2T$ multiplications for a temporal complexity of $2Tn^T$. This is, of course, completely intractable if T is large.

Example 13.4.4. Consider again the weather and tree-ring HMM of Section 13.1.3 and Example 13.4.1. Assume the transition matrix, the observation matrix, and the initial state distribution are

$$A = \begin{bmatrix} 0.7 & 0.4 \\ 0.3 & 0.6 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0.1 & 0.7 \\ 0.4 & 0.2 \\ 0.5 & 0.1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\pi} = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$$

respectively. Assume further that the sequence of states is observations is $\mathbf{z} = (S, M, S, L)$. If the state sequence is $\mathbf{x} = (W, W, D, D)$ then we compute

$$\begin{aligned} P(\mathbf{z}, \mathbf{x} \,|\, A, B, \boldsymbol{\pi}) &= \pi_W b_{SW} a_{WW} b_{MW} a_{DW} b_{SD} a_{DD} B_{LD} \\ &= 0.6(0.1)(0.7)(0.4)(0.3)(0.7)(0.6)(0.1) = 0.000212 \end{aligned}$$

One way to compute $P(\mathbf{z} \,|\, A, B, \boldsymbol{\pi})$ without knowing $\mathbf{x}$ is to use the law of total probability and sum over all 2^4 possible state sequences:

$$\begin{aligned} P(\mathbf{z} \,|\, A, B, \boldsymbol{\pi}) &= \sum_{\mathbf{x}} P(\mathbf{z}, \mathbf{x} \,|\, A, B, \boldsymbol{\pi}) \\ &= \sum_{\mathbf{x}} \pi_{x_0} b_{Sx_0} a_{x_1 x_0} b_{Mx_1} a_{x_2 x_1} b_{Sx_2} a_{x_3 x_2} B_{Lx_3}. \end{aligned}$$

This is not hard, because 2^4 is a relatively small number, but if the sequence $\mathbf{z}$ had length 300, the sum would have $2^{300} \approx 2 \times 10^{90}$ terms.

Fortunately, there is a much more efficient way to compute this probability using an algorithm called the *forward pass*, which we describe below. Let

$$\alpha_t(i) = P(z_0, \ldots, z_t, X_t = i \,|\, \boldsymbol{\theta}) \tag{13.21}$$

The law of total probability and (13.21) show that

$$P(\mathbf{z} \,|\, \boldsymbol{\theta}) = \sum_{i \in \mathscr{X}} \alpha_{T-1}(i). \tag{13.22}$$

Thus, if we could compute $\alpha_{T-1}(i)$ for all $i \in \mathscr{X}$, we would also have the desired quantity $P(\mathbf{z} \,|\, \boldsymbol{\theta})$.

Proposition 13.4.5. *The quantities $\alpha_t(i)$ satisfy the recursion relations*

$$\alpha_0(i) = \pi_i b_{z_0 i} \tag{13.23}$$

and

$$\alpha_t(i) = b_{z_t i} \sum_{j \in \mathscr{X}} \alpha_{t-1}(j) a_{ij} \tag{13.24}$$

Proof. Both (13.23) and (13.24) follow directly from writing out the definitions of each term. The details are Exercise 13.10. □

The *forward algorithm* or *forward pass* consists of computing (13.23) for all $i \in \mathscr{X}$ and then for each $t \in \{1, \ldots, T-1\}$ computing (13.24) for all $i \in \mathscr{X}$. For each t and i computing $\alpha_t(i)$ requires no more than $2n$ FLOPs, for an overall temporal complexity of $\sim 2n^2 T$. If this is computed in ascending order of t (forward), then it is an example of dynamic programming (see Volume 2, Section 4.1), which is more efficient than naïve recursion (starting at $t = T - 1$).

Example 13.4.6. To execute the forward pass with the weather-and-tree-ring HMM (Example 13.4.4), first compute

$$\begin{aligned} \alpha_0(W) &= \pi_W b_{SW} = 0.6(0.1) = 0.06 \\ \alpha_0(D) &= \pi_D b_{SD} = 0.4(0.7) = 0.28, \end{aligned}$$

and then iteratively compute

$$\begin{aligned}\alpha_1(W) &= b_{MW}[\alpha_0(W)a_{WW} + \alpha_0(D)a_{WD}] = 0.0616\\ \alpha_1(D) &= b_{MD}[\alpha_0(W)a_{DW} + \alpha_0(D)a_{DD}] = 0.0372,\end{aligned}$$

$$\begin{aligned}\alpha_2(W) &= b_{SW}[\alpha_1(W)a_{WW} + \alpha_1(D)a_{WD}] = 0.0058\\ \alpha_2(D) &= b_{SD}[\alpha_1(W)a_{DW} + \alpha_1(D)a_{DD}] = 0.02856\end{aligned}$$

and

$$\begin{aligned}\alpha_3(W) &= b_{LW}[\alpha_2(W)a_{WW} + \alpha_2(D)a_{WD}] = 0.007742\\ \alpha_3(D) &= b_{LD}[\alpha_2(W)a_{DW} + \alpha_2(D)a_{DD}] = 0.0018876.\end{aligned}$$

13.4.3 Probability of a State (Backward Pass)

Given $\mathbf{z}$ and $\boldsymbol{\theta}$, we now want to compute $P(X_t = i \,|\, \mathbf{z}, \boldsymbol{\theta})$ for each i and t. We denote this quantity $\gamma_t(i)$:

$$\gamma_t(i) = P(X_t = i \,|\, \mathbf{z}, \boldsymbol{\theta}). \tag{13.25}$$

The solution to Question (ii) is given by choosing the most likely state

$$\widehat{x}_t = \operatorname*{argmax}_{i \in \mathscr{X}} \gamma_t(i) \tag{13.26}$$

for each time t.

This can also be computed efficiently, using what is called the *backward pass*. For each $t \in \{0, \ldots, T-1\}$ and $i \in \mathscr{X}$ define

$$\beta_t(j) = P(z_{t+1}, z_{t+2}, \ldots, z_{T-1} \,|\, X_t = j, \boldsymbol{\theta}).$$

The conditional independence assumptions of an HMM guarantee that

$$(z_0, \ldots, z_t) \perp\!\!\!\perp (z_{t+1}, \ldots, z_{T-1}) \,|\, x_t,$$

which implies that

$$\begin{aligned}P(X_t = i, \mathbf{z} \,|\, \boldsymbol{\theta}) &= P(X_t = i, z_0, \ldots, z_t \,|\, \boldsymbol{\theta}) P(z_{t+1}, z_{t+2}, \ldots, z_{T-1} \,|\, X_t = i, z_0, \ldots, z_t, \boldsymbol{\theta})\\ &= P(X_t = i, z_0, \ldots, z_t \,|\, \boldsymbol{\theta}) P(z_{t+1}, z_{t+2}, \ldots, z_{T-1} \,|\, X_t = i, \boldsymbol{\theta})\\ &= \alpha_t(i)\beta_t(i).\end{aligned} \tag{13.27}$$

This gives

$$\gamma_t(i) = P(X_t = i \,|\, \mathbf{z}, \boldsymbol{\theta}) = \frac{P(X_t = i, \mathbf{z} \,|\, \boldsymbol{\theta})}{P(\mathbf{z} \,|\, \boldsymbol{\theta})} = \frac{\alpha_t(i)\beta_t(i)}{P(\mathbf{z} \,|\, \boldsymbol{\theta})}. \tag{13.28}$$

Equation (13.22) gives the denominator from the forward pass, so if we could compute each $\beta_t(i)$, we would have the desired quantity $\gamma_t(i)$. Fortunately, the $\beta_t(i)$ also satisfy a recursion relation that makes them relatively inexpensive to compute.

Proposition 13.4.7. *If $\beta_{T-1}(i) = 1$ for all $i \in \mathscr{X}$, then the $\beta_t(i)$ satisfy the following relation:*

$$\beta_t(j) = \sum_{i \in \mathscr{X}} a_{ij} \beta_{t+1}(i) b_{z_{t+1}, i}. \tag{13.29}$$

Proof. The proof is a direct calculation from the definitions of a_{ij} and $\beta_{t+1}(i)$. The details are Exercise 12.11. □

The *backward pass* sets $\beta_{T-1}(j) = 1$ and for each $t = T-2, T-3, \ldots, 0$ computes $\beta_t(j)$ for all $j \in \mathscr{X}$, using (13.29) for a total cost of $\sim 3n^2T$ FLOPs. Equations (13.25) and (13.26) now give the desired probabilities and states, respectively.

Example 13.4.8. Applying the backward pass to the weather-and-tree-ring example (Examples 13.4.4 and 13.4.6) with observation sequence $\mathbf{z} = (S, M, S, L)$ gives

$$\begin{aligned}
\beta_3(W) &= \beta_3(D) = 1 \\
\beta_2(W) &= \beta_3(W) b_{LW} a_{WW} + \beta_3(D) b_{LD} a_{DW} = 0.38 \\
\beta_2(D) &= \beta_3(W) b_{LW} a_{WD} + \beta_3(D) b_{LD} a_{DD} = 0.26 \\
\beta_1(W) &= \beta_2(W) b_{SW} a_{WW} + \beta_2(D) b_{MD} a_{DW} = 0.0812 \\
\beta_1(D) &= \beta_2(W) b_{SW} a_{WD} + \beta_2(D) b_{MD} a_{DD} = 0.1244 \\
\beta_0(W) &= \beta_1(W) b_{SW} a_{WW} + \beta_1(D) b_{SD} a_{DW} = 0.0302 \\
\beta_0(D) &= \beta_1(W) b_{SW} a_{WD} + \beta_1(D) b_{SD} a_{DD} = 0.02792
\end{aligned}$$

13.4.4 Most Likely Sequence (Viterbi Algorithm)

Now we consider how to answer question (iii), where we wish to find the sequence $\mathbf{x} = (x_0, \ldots, x_{T-1})$ that is most probable, given $\mathbf{z}$ and $\boldsymbol{\theta}$. That is, we wish to find

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}) = \operatorname*{argmax}_{\mathbf{x}} \frac{P(\mathbf{x}, \mathbf{z} \,|\, \boldsymbol{\theta})}{P(\mathbf{z} \,|\, \boldsymbol{\theta})} = \operatorname*{argmax}_{\mathbf{x}} P(\mathbf{x}, \mathbf{z} \,|\, \boldsymbol{\theta}).$$

The algorithm we use is a variant of the forward algorithm called the *Viterbi algorithm*, which is a dynamic optimization algorithm that relies on the Bellman optimality principle (See Volume 2, Sections 4.1.3 and 16.1.3). It is an example of dynamic programming, just like the forward and backward algorithms. Unfortunately some people confusingly call the Viterbi algorithm *the* dynamic programming algorithm for HMMs.

For each $t \in \{0, \ldots, T-1\}$ let

$$\eta_t(i) = \max_{x_0, \ldots, x_{t-1}} P(x_0, \ldots, x_{t-1}, X_t = i, z_0, \ldots, z_t \,|\, \boldsymbol{\theta}). \tag{13.30}$$

When $t = T-1$ this is closely related, but not identical, to the maximal value of $P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta})$ attained by $\mathbf{x}^*$.

It is straightforward to see that

$$\eta_0(i) = \pi_i b_{z_0,i} \qquad \forall i \in \mathscr{X}.$$

The Bellman optimality principle applies in this case and the corresponding Bellman relation is

$$\begin{aligned}\eta_t(i) &= \max_{j\in\mathscr{X}} \eta_{t-1}(j) P(X_t = i, Z_t = z_t \,|\, X_{t-1} = j) \\ &= \max_j \left(\eta_{t-1}(j) a_{ij} b_{z_t,i}\right). \end{aligned} \tag{13.31}$$

This allows us to compute each $\eta_t(i)$ iteratively from the values of $\eta_{t-1}(j)$ for all $j \in \mathscr{X}$. Similarly, the maximal value of $P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta})$ can be realized as

$$\max_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}) = \max_{j\in\mathscr{X}} \eta_{T-1}(j).$$

Of course, the thing we really want is the maximizer $\mathbf{x}^*$, not the maximal value. But, as is typical with dynamic optimization problems, tracking the maximizer with each application of (13.31) allows the reconstruction of the overall maximizer. In the process of computing the $\eta_t(i)$, store the maximizers

$$\widetilde{x}_{t-1}(i) = \operatorname*{argmax}_{j\in\mathscr{X}} \eta_{t-1} a_{ij} b_{z_t,i} \qquad \forall t \in \{1, \ldots, T-1\}, \tag{13.32}$$

and set

$$x^*_{T-1} = \operatorname*{argmax}_j \eta_{T-1}(j).$$

The optimal sequence $\mathbf{x}^* = (x^*_0, \ldots, x^*_{T-1})$ in the state space can be reconstructed by stepping backward from the final optimizer x^*_{T-1} as follows:

$$x^*_{t-1} = \widetilde{x}_{t-1}(x^*_t) \qquad \forall t \leq T-1.$$

Example 13.4.9. To apply the Viterbi algorithm to the weather-and-treering example with observation sequence $\mathbf{z} = (S, M, S, L)$, the initial step is to compute

$$\begin{aligned}\eta_0(W) &= P(X_0 = W, Z_0 = S \,|\, \boldsymbol{\theta}) = \pi_W B_{SW} = 0.6(0.1) = 0.06 \\ \eta_0(D) &= P(X_0 = D, Z_0 = S \,|\, \boldsymbol{\theta}) = \pi_D B_{SD} = 0.4(0.7) = 0.28\end{aligned}$$

The next step computes the probability of each two-step state sequence for the two-step observation sequence (S, M) and identifies the maximizers $\tilde{y}_0(i)$ and maxima $\eta_1(i)$. Below we simplify notation by dropping the symbols X_i and Z_i in the probabilities.

$$\begin{aligned}P(WW, SM \,|\, \boldsymbol{\theta}) &= \eta_0(W) a_{WW} b_{MW} = 0.06(0.7)(0.4) = 0.0168 \\ P(DW, SM \,|\, \boldsymbol{\theta}) &= \eta_0(D) a_{WD} b_{MW} = 0.28(0.4)(0.4) = 0.0448 \\ \eta_1(W) &= \max(0.0168, 0.0448) = 0.0448 \\ \widetilde{x}_1(W) &= \operatorname{argmax}(0.0168, 0.0448) = D.\end{aligned}$$

This shows that the optimal two-step state sequence ending in W has $X_0 = \widetilde{x}_1(W) = D$; hence, the optimal state sequence ending in W is DW. Similarly, we have

$$\begin{aligned} P(WD, SM \,|\, \boldsymbol{\theta}) &= \eta_0(W) a_{DW} b_{MD} = 0.06(0.3)(0.2) = 0.0036 \\ P(DD, SM \,|\, \boldsymbol{\theta}) &= \eta_0(D) a_{DD} b_{MD} = 0.28(0.6)(0.2) = 0.0336 \\ \eta_1(D) &= \max(0.0036, 0.0336) = 0.0336 \\ \widetilde{x}_1(D) &= \operatorname{argmax}(0.0036, 0.0336) = D. \end{aligned}$$

This shows that the optimal two-step state sequence ending in D is has $X_0 = \widetilde{x}_1(D) = D$; hence, the optimal state sequence ending in D is DD.

The next step computes the probability of each candidate for the optimal three-step state sequence for the three-step observation sequence (S, M, S). Because of the Bellman principle, the optimal three-step state sequences must begin with one of the optimal two-step state sequences, so we need only consider the state sequences DWW, DDW, DWD, and DDD:

$$\begin{aligned} P(DWW, SMS \,|\, \boldsymbol{\theta}) &= \eta_1(W) a_{WW} b_{SW} = 0.003136 \\ P(DDW, SMS \,|\, \boldsymbol{\theta}) &= \eta_1(D) a_{WD} b_{SW} = 0.001344 \\ \eta_2(W) &= \max(0.003136, 0.001344) = 0.003136 \\ \widetilde{x}_2(W) &= \operatorname{argmax}(0.003136, 0.001344) = W, \end{aligned}$$

This shows that the optimal three-step state sequence ending in W has $X_1 = W$. Similarly, we compute

$$\begin{aligned} P(DWD, SMS \,|\, \boldsymbol{\theta}) &= \eta_1(W) a_{DW} b_{SD} = 0.009408 \\ P(DDD, SMS \,|\, \boldsymbol{\theta}) &= \eta_1(D) a_{DD} b_{SD} = 0.014112 \\ \eta_2(D) &= \max(0.009408, 0.014112) = 0.014112 \\ \widetilde{x}_2(D) &= \operatorname{argmax}(0.009408, 0.014112) = D, \end{aligned}$$

This shows that the optimal three-step state sequence ending in D has $X_1 = D$. Continuing in this manner, we find

$$\begin{aligned} \eta_3(W) &= 0.0028224 \qquad \widetilde{x}_3(W) = D \\ \eta_3(D) &= 0.00084672 \qquad \widetilde{x}_3(D) = D. \end{aligned}$$

Now we reconstruct the optimal state sequence by first computing the optimal final state

$$x_3^* = \operatorname*{argmax}_{j \in \{W, D\}} \eta_3(j) = W.$$

and from this we compute the optimal penultimate state

$$x_2^* = \widetilde{x}_2(W) = \operatorname*{argmax}_{j \in \{W, D\}} \eta_2(j) a_{Wj} b_{WL} = D.$$

and so on to get

$$x_1^* = \widetilde{x}_1(D) = D \qquad \text{and} \qquad x_0^* = \widetilde{x}_0(D) = D.$$

Vista 13.4.10. The Viterbi algorithm is widely used for decoding error-correcting codes. It allows for fast, accurate transmission of data over noisy channels, and is commonly used in wifi and cellular networks as well as in deep-space communications.

13.5 HMM Parameter Estimation (Baum-Welch)

In this section we use EM to estimate the parameters $\boldsymbol{\theta} = (\boldsymbol{\pi}, A, B)$ of an HMM, given an observation sequence $\mathbf{z}$ (or many such sequences). The resulting algorithm is called the *Baum-Welch* algorithm.

Estimating the parameters $\boldsymbol{\theta} = (\boldsymbol{\pi}, A, B)$ on an HMM amounts to maximizing the log likelihood

$$\ell(\boldsymbol{\theta}) = \log(P(\mathbf{z} \,|\, \boldsymbol{\theta})) = \log\left(\sum_{\mathbf{x}} P(\mathbf{z}, \mathbf{x} \,|\, \boldsymbol{\theta})\right)$$

Here we think of the state sequence $\mathbf{x} = (x_0, \ldots, x_{T-1})$ as a single draw of the latent variable X and the observation sequence $\mathbf{z} = (z_0, \ldots, z_{T-1})$ as a single draw of the observable variable Z. Later we will extend these results to the setting where there are multiple observation sequences.

13.5.1 Baum–Welch

We can use EM to maximize $\ell(\boldsymbol{\theta})$, starting with any $\boldsymbol{\theta}^{old} = (A^\circ, B^\circ, \boldsymbol{\pi}^\circ)$ (the old estimate of $\boldsymbol{\theta}$), constructing

$$q^{old}(\mathbf{x}) = P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}),$$

and setting

$$Q^{old}(\boldsymbol{\theta}) = \sum_{\mathbf{x}} q^{old}(\mathbf{x}) \log\left(P(\mathbf{z}, \mathbf{x} \,|\, \boldsymbol{\theta})\right) = \sum_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \log\left(P(\mathbf{z}, \mathbf{x} \,|\, \boldsymbol{\theta})\right)$$

The probability $P(\mathbf{z}, \mathbf{x} \,|\, \boldsymbol{\theta})$ can be computed as

$$P(\mathbf{z}, \mathbf{x} \,|\, \boldsymbol{\theta}) = \pi_{x_0} \prod_{t=0}^{T-1} b_{z_t, x_t} \prod_{t=1}^{T-1} a_{x_t x_{t-1}}. \tag{13.33}$$

This implies that

$$\begin{aligned} Q^{old}(\boldsymbol{\theta}) = &\sum_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \log(\pi_{x_0}) \\ &+ \sum_{t=0}^{T-1} \sum_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \log(b_{z_t, x_t}) \\ &+ \sum_{t=1}^{T-1} \sum_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \log(a_{x_t x_{t-1}}) \end{aligned} \tag{13.34}$$

Breaking up the first sum of (13.34) into a sum over x_0 and a sum over the rest of $\mathbf{x}$ gives

$$\begin{aligned} \sum_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \log(\pi_{x_0}) &= \sum_{x_0} \log(\pi_{x_0}) \sum_{x_1, \ldots, x_{T-1}} P(x_0, x_1, \ldots, x_{T-1} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \\ &= \sum_{x_0} \log(\pi_{x_0}) P(x_0 \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \\ &= \sum_{x_0} \log(\pi_{x_0}) \gamma_0(x_0) \\ &= \sum_{j} \log(\pi_j) \gamma_0(j), \end{aligned}$$

where the second equality follows from the law of total probability, and each $\gamma_0(j)$ is computed using $\boldsymbol{\theta}^{old}$. Similarly, the second sum can be rewritten as

$$\begin{aligned} \sum_{t=0}^{T-1} \sum_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \log(b_{z_t, x_t}) &= \sum_{t=0}^{T-1} \sum_{x_t} \log(b_{z_t, x_t}) \sum_{x_0, \ldots, \widehat{x}_t, \ldots x_{T-1}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \\ &= \sum_{t=0}^{T-1} \sum_{x_t} \log(b_{z_t, x_t}) P(x_t \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \\ &= \sum_{t=0}^{T-1} \sum_{j \in \mathscr{X}} \log(b_{z_t, j}) \gamma_t(j), \end{aligned}$$

where the hat on $\widehat{x}_t$ indicates that x_t is omitted from the inner sum $\sum_{x_0, \ldots, \widehat{x}_t, \ldots x_{T-1}}$ of the right side of the first line. Again the second line follows from the law of total probability, and the $\gamma_t(j)$s are computed using $\boldsymbol{\theta}^{old}$.

The third sum reduces to

$$\begin{aligned} \sum_{t=1}^{T-1} \sum_{\mathbf{x}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \log(a_{x_t x_{t-1}}) &= \sum_{t=0}^{T-2} \sum_{x_t, x_{t+1}} \log(a_{x_{t+1}, x_t}) \sum_{x_0, \ldots, \widehat{x}_t, \widehat{x}_{t+1}, \ldots x_{T-1}} P(\mathbf{x} \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) \\ &= \sum_{t=0}^{T-2} \sum_{i,j \in \mathscr{X}} \log(a_{ij}) \xi_t(i,j), \end{aligned}$$

where $\xi_t(i,j)$ is defined to be

$$\xi_t(i,j) = P(X_{t+1} = i, X_t = j \,|\, \mathbf{z}, \boldsymbol{\theta}^{old}) = \frac{\alpha_t(j) a_{ij}^{old} b_{z_{t+1}i}^{old} \beta_{t+1}(i)}{P(\mathbf{z} \,|\, \boldsymbol{\theta}^{old})}, \tag{13.35}$$

where the quantities $\alpha_t(i)$ and $\beta_{t+1}(j)$ are all computed using the forward and backward with on $\boldsymbol{\theta}^{old}$. The computation of the α, β and ξ is the expectation step.

Remark 13.5.1. It is useful to note, and straightforward to verify that

$$\gamma_t(j) = \sum_{i \in \mathscr{X}} \xi_t(i,j). \tag{13.36}$$

The maximization step gives the new parameter $\boldsymbol{\theta}^{new} = (A^{new}, B^{new}, \boldsymbol{\pi}^{new})$ as the maximizer of (13.34)

$$\begin{aligned} \boldsymbol{\theta}^{new} = \operatorname*{argmax}_{\boldsymbol{\theta}} & \sum_{j \in \mathscr{X}} \log(\pi_j) \gamma_0(j) \\ & + \sum_{t=0}^{T-1} \sum_{j \in \mathscr{X}} \log(b_{z_t j}) \gamma_t(j) \\ & + \sum_{t=0}^{T-2} \sum_{i,j \in \mathscr{X}} \log(a_{ij}) \xi_t(i,j), \end{aligned} \tag{13.37}$$

subject to the constraints that

$$\begin{aligned} \sum_{j \in \mathscr{X}} \pi_j = 1 \text{ and } \pi_j \in [0,1] \quad \forall j \\ \sum_{i \in \mathscr{X}} a_{ij} = 1 \quad \forall j \text{ and } a_{ij} \in [0,1] \quad \forall i,j \\ \sum_{z \in \mathscr{Z}} b_{zj} = 1 \quad \forall j \text{ and } b_{zj} \in [0,1] \quad \forall z,j \end{aligned} \tag{13.38}$$

The three sums in (13.37) can be maximized separately, which gives

$$\begin{aligned} \boldsymbol{\pi}^{new} &= \operatorname*{argmax}_{\boldsymbol{\pi}} \sum_{j \in \mathscr{X}} \log(\pi_j) \gamma_0(j) \\ B^{new} &= \operatorname*{argmax}_{B} \sum_{t=0}^{T-1} \sum_{j \in \mathscr{X}} \log(b_{z_t,j}) \gamma_t(j) \\ A^{new} &= \operatorname*{argmax}_{A} \sum_{t=0}^{T-2} \sum_{i,j \in \mathscr{X}} \log(a_{ij}) \xi_t(i,j), \end{aligned}$$

subject to the constraints (13.38).

A straightforward optimization using Lagrange multipliers shows that

$$\pi_i^{new} = \gamma_0(i) \tag{13.39}$$

$$B_{ij}^{new} = \frac{\sum_{t=0}^{T-1} \gamma_t(j)\delta_{z_t=i}}{\sum_{t=0}^{T-1} \gamma_t(j)} \tag{13.40}$$

$$A_{ij}^{new} = \frac{\sum_{t=0}^{T-2} \xi_t(i,j)}{\sum_{t=0}^{T-2} \gamma_t(j)} \tag{13.41}$$

Thus, the MLE estimate $\widehat{\boldsymbol{\theta}}$ can be approximated by starting with an arbitrary $\boldsymbol{\theta}^0$, applying the forward and backward pass to compute the terms $\alpha_t(i)$, $\beta_t(i)$, and $\xi_t(i,j)$ (the expectation step), then updating to $\boldsymbol{\theta}^1$ using (13.39–13.41) (the maximization step). The process is then repeated until satisfactory convergence is achieved.

13.5.2 Baum–Welch with Multiple Observations

So far everything we have done used a single observation sequence $\mathbf{z} = (z_0, \ldots, z_{T-1})$. But in many situations we can collect many observations (for example, in the tree-ring-weather model, we could examine the rings of several trees), and we want to use all the data to estimate the parameters.

With multiple sequences $\mathbf{z}^1, \ldots, \mathbf{z}^N$, the approximation $Q^{old}(\boldsymbol{\theta})$ in the EM algorithm is now a sum

$$Q^{old}(\boldsymbol{\theta}) = \sum_{\ell=1}^{N} Q_\ell^{old}(\boldsymbol{\theta})$$

where each $Q_\ell^{old}(\boldsymbol{\theta})$ is given by replacing $\mathbf{z}$ in (13.34) by $\mathbf{z}_\ell$. That changes the optimization problem in (13.37) (but not the constraints) to

$$\begin{aligned}\boldsymbol{\theta}^{new} = \operatorname*{argmax}_{\boldsymbol{\theta}} & \sum_{j \in \mathscr{X}} \log(\pi_j) \sum_{\ell=1}^{N} \gamma_0^\ell(j) \\ & + \sum_{t=0}^{T-1} \sum_{i \in \mathscr{X}} \sum_{j \in \mathscr{Z}} \log(b_{ji}) \sum_{\ell=1}^{N} \gamma_t^\ell(i)\delta_{z_t=j} \\ & + \sum_{t=0}^{T-2} \sum_{i,j \in \mathscr{X}} \log(a_{ji}) \sum_{\ell=1}^{N} \xi_t^\ell(i,j),\end{aligned} \tag{13.42}$$

where $\gamma_t^\ell(i)$ and $\xi_t^\ell(i,j)$ are computed using only the observation sequence $\mathbf{x}^\ell$.

The resulting maximizer is

$$\pi_i^{new} = \frac{1}{N} \sum_{\ell=1}^{N} \gamma_0^\ell(i) \tag{13.43}$$

$$b_{ij}^{new} = \frac{\sum_{t=0}^{T-1} \sum_{\ell=1}^{N} \gamma_t^\ell(j)\delta_{z_t^\ell=i}}{\sum_{t=0}^{T-1} \sum_{\ell=1}^{N} \gamma_t^\ell(j)} \tag{13.44}$$

$$a_{ij}^{new} = \frac{\sum_{t=0}^{T-2} \sum_{\ell=1}^{N} \xi_t^\ell(i,j)}{\sum_{t=0}^{T-2} \sum_{\ell=1}^{N} \gamma_t^\ell(j)}. \tag{13.45}$$

In summary, the full Baum-Welch algorithm for a collection $\mathbf{z}^1, \ldots, \mathbf{z}^N$ of observation sequences is as follows:

(i) Begin with an initial $\boldsymbol{\theta}^{old}$.

(ii) For each $\ell \in \{1, \ldots, N\}$

  (a) Use the forward pass to compute $\alpha_t^\ell(i)$ for every i and t using (13.23) and (13.24).

  (b) Use the backward pass to compute $\beta_t^\ell(i)$ for every i and t, using (13.29).

  (c) Compute $\xi_t^\ell(i,j)$ for each t, i, j using (13.35).

  (d) Compute $\gamma_t^\ell(i)$ for each t, i using (13.36).

(iii) Compute $\boldsymbol{\theta}^{new}$ using (13.43), (13.44), and (13.45).

(iv) Set $\boldsymbol{\theta}^{old} \leftarrow \boldsymbol{\theta}^{new}$ and repeat from (i).

13.5.3 *Rescaling to Avoid Underflow

Most of the quantities discussed in this section become very small as T grows and will underflow if computed naïvely. One solution is to rescale at each step of the forward or backward algorithm, as described below.

Rescaling the Forward Pass

The key idea is to divide the $\alpha_t(j)$s by their sum $\sum_j \alpha_t(j)$ at each step. This prevents the $\alpha_t(j)$s from becoming too small as t get large.

(i) For $t = 0$ compute $\alpha_0(i)$ as usual, and set $\tilde{\alpha}_0(i) = \alpha_0(i)$ for all $i \in \mathscr{X}$. Now let

$$c_0 = \frac{1}{\sum_{j \in \mathscr{X}} \tilde{\alpha}_0(j)}$$

and set

$$\widehat{\alpha}_0(i) = c_0 \tilde{\alpha}_0(i) \quad \forall i \in \mathscr{X}.$$

(ii) For each $t \in \{1, \ldots, T-1\}$ do the following:

  (a) Compute

$$\tilde{\alpha}_t(i) = \sum_{j \in \mathscr{X}} \widehat{\alpha}_{t-1}(j) a_{ij} b_{z_t i}. \quad \forall i \in \mathscr{X} \tag{13.46}$$

  (b) Compute

$$c_t = \frac{1}{\sum_{j \in \mathscr{X}} \tilde{\alpha}_t(j)}. \tag{13.47}$$

  (c) Rescale

$$\widehat{\alpha}_t(i) = c_t \tilde{\alpha}_t(i) \quad \forall i \in \mathscr{X}. \tag{13.48}$$

Proposition 13.5.2. *We have*

$$\widehat{\alpha}_t(i) = \prod_{k=0}^{t} c_k \alpha_t(i) \tag{13.49}$$

and

$$\widehat{\alpha}_t(i) = \frac{\alpha_t(i)}{\sum_{j\in\mathscr{X}} \alpha_t(j)} \tag{13.50}$$

for all $i \in \mathscr{X}$ and all $t \in 0, \ldots, T-1$.

Proof. Equation (13.49) is true by definition for $t = 0$ and all i. The general case of (13.49) follows by induction. If (13.49) holds for some value of $t < T-1$ and all $i \in \mathscr{X}$, then we have

$$\begin{aligned}
\widehat{\alpha}_{t+1}(i) &= c_{t+1}\tilde{\alpha}_{t+1}(i) \\
&= c_{t+1} \sum_{j\in\mathscr{X}} \widehat{\alpha}_t(j) a_{ij} b_{z_{t+1}i} \\
&= c_{t+1} \prod_{k=0}^{t} c_k \sum_{j\in\mathscr{X}} \alpha_t(j) a_{ij} b_{z_{t+1}i} \\
&= \prod_{k=0}^{t+1} c_k \alpha_{t+1}(i).
\end{aligned}$$

Equation 13.48 shows that $\tilde{\alpha}_t(i) = \prod_{k=0}^{t-1} c_k \alpha_t(i)$ for all t and i, which, combined with the definition (13.47) of the c_t gives (13.50). ∎

Corollary 13.5.3. *We have*

$$\sum_{j\in\mathscr{X}} \widehat{\alpha}_t(j) = 1 \tag{13.51}$$

for all $t \in \{0, \ldots, T-1\}$. Moreover, we have

$$P(\mathbf{z} \,|\, \boldsymbol{\theta}) = \frac{1}{\prod_{t=0}^{T-1} c_t}. \tag{13.52}$$

Proof. Equation (13.51) follows immediately from (13.50). Equation (13.52) follows immediately from the combination of (13.22) and (13.49). ∎.

The proposition and the corollary allow us to compute all the quantities of the forward pass from the rescaled forward pass. Note, however, that (13.52) is at risk of underflow for large values of T, so we instead compute its logarithm

$$\log(P(\mathbf{z} \,|\, \boldsymbol{\theta})) = -\sum_{t=0}^{T-1} \log(c_t).$$

Rescaling the Backward Pass

The backward pass is rescaled in the same way as the forward.

(i) For $t = T-1$ let $\widehat{\beta}_{T-1}(i) = c_{T-1}\beta_{T-1}(i) = c_{T-1}$ for all $i \in \mathscr{X}$.

(ii) For each $t \in \{T-2, \ldots, 0\}$ do the following:

(a) Compute

$$\tilde{\beta}_t(j) = \sum_{i \in \mathscr{X}} a_{ij}\widehat{\beta}_{t+1}(i)b_{z_{t+1}i}. \quad \forall j \in \mathscr{X} \tag{13.53}$$

(b) Rescale

$$\widehat{\beta}_t(i) = c_t\tilde{\beta}_t(i) \quad \forall i \in \mathscr{X}. \tag{13.54}$$

Proposition 13.5.4. *The following relations hold among the unscaled and rescaled values for all $i, j \in \mathscr{X}$ and all $t \in 0, \ldots, T-1$:*

$$\widehat{\beta}_t(i) = \prod_{k=t}^{T-1} c_k\beta_t(i) \tag{13.55}$$

$$\gamma_t(i) = \frac{\widehat{\alpha}_t(i)\widehat{\beta}_t(i)}{c_t} \tag{13.56}$$

$$\xi_t(i,j) = \widehat{\alpha}_t(i)a_{ji}b_{z_{t+1}j}\widehat{\beta}_{t+1}(j) \tag{13.57}$$

Proof. The proof is Exercise 12.16. □

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second line are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

13.1. Consider a mixture-of-experts model with $\boldsymbol{\theta} = (V, \boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \sigma_1, \sigma_2)$, where $V = \begin{bmatrix} -20 & 0 \\ 10 & 0 \end{bmatrix}$, $\boldsymbol{\beta}_1 = (1, 3)$, $\boldsymbol{\beta}_2 = (4, -1)$, $\sigma_1 = 0.1$, $\sigma_2 = 0.2$. Code up a way to sample from this model, given an input x. Your code should take as input $x \in \mathbb{R}$, and draw $z \sim \text{Cat}(\mathscr{S}(V^\mathsf{T}\mathbf{x}))$, where $\mathbf{x} = (1, x)$. It should then return a draw from $y \sim \mathscr{N}(\mathbf{x}^\mathsf{T}\beta_z, \sigma_z^2)$.

(i) Draw 10^3 times from $x \sim \text{Uniform}(0, 4)$ and feed the results to your mixture of experts to produce corresponding y values.

(ii) Scatterplot the results (x_i, y_i), and on the same graph plot the line segment $y = \mathbf{x}^\mathsf{T}\beta_1$ for $x \in [0, 2]$ and the line segment $y = \mathbf{x}^\mathsf{T}\beta_2$ for $x \in [2, 4]$.

13.2. Consider the weather-and-tree-ring HMM where the Markov chain of the unobserved variables X_i has state space $\{W, D\}$ with transition matrix $A = \begin{bmatrix} 0.7 & 0.4 \\ 0.3 & 0.6 \end{bmatrix}$ and initial probability $\boldsymbol{\pi} = (0.1, 0.9)$. Assume, further, that the observations Z_i take values in $\{S, M, L\}$ and the probabilities of observing Z_i given X_i are given by the matrix $B = \begin{bmatrix} 0.1 & 0.7 \\ 0.4 & 0.2 \\ 0.5 & 0.1 \end{bmatrix}$ (so, for example, $P(Z = M \,|\, X = D) = 0.2$). We denote all these parameters by $\boldsymbol{\lambda} = (A, B, \boldsymbol{\pi})$. Assume that we have observed the sequence $z_0, z_1, z_2 = M, S, L$.

(i) Compute the probability $P((M, S, L) \,|\, (W, W, W), \boldsymbol{\lambda}) = P(Z_0 = M, Z_1 = S, Z_2 = L \,|\, \boldsymbol{\lambda}, X_0 = X_1 = X_2 = W)$ by hand.

(ii) For every possible sequence of hidden states x_0, x_1, x_2, compute the probability $P((M, S, L) \,|\, \boldsymbol{\lambda}, (x_0, x_1, x_2))$. Hint: Computers are useful for doing tedious chores, and you can often learn an idea as well or better from programming those chores as from doing them by hand.

(iii) The values of X_i are hidden (unobserved), but we can calculate $P((M, S, L) \,|\, \boldsymbol{\lambda})$ by marginalizing (summing over all possible values of the hidden variables). Use the results of the previous step to compute $P((M, S, L) \,|\, \boldsymbol{\lambda}) = \sum_{x_0, x_1, x_2} P((M, S, L) \,|\, \boldsymbol{\lambda}, (x_0, x_1, x_2)) P((x_0, x_1, x_2))$.

(iv) For an observed sequence $(z_0, \ldots, z_{n-1})$ of length n, how many individual terms will need to be computed to find the sum $P((z_0, \ldots, z_{n-1}) \,|\, \boldsymbol{\lambda}) = \sum_{x_0, \ldots, x_{n-1}} P((z_0, \ldots, z_{n-1}) \,|\, \boldsymbol{\lambda}, (x_0, \ldots, x_{n-1})) P((x_0, \ldots, x_{n-1}))$? Later in the chapter we give an alternative algorithm to compute $P((z_0, \ldots, z_{n-1}) \,|\, \boldsymbol{\lambda})$ much more efficiently.

13.3. Using only the definitions of ℓ and θ^{t+1} and the two conditions (i) and (ii) listed at the beginning of Section 13.2.2, show the following properties of the sequence $(\ell(\theta^t))_{t=0}^\infty$ constructed by the EM algorithm:

(i) The sequence $(\ell(\theta^t))_{t=0}^\infty$ is nondecreasing.

(ii) The log likelihood $\ell(\theta)$ is bounded above.

(iii) The sequence $(\ell(\theta^t))_{t=0}^\infty$ converges to some finite value (Hint: see Lemma 5.4.28 in Volume 1).

13.4. Assume that $\widetilde{Q}^t(\theta)$ and ℓ are both continuously differentiable in θ and that $\widetilde{Q}^t(\theta)$ is strictly concave and satisfies the conditions (i) and (ii) described at the beginning of Section 13.2.2. Prove that if $\ell(\theta^{t+1}) = \ell(\theta^t)$ then $D\ell(\theta^t) = \mathbf{0}$ as follows:

(i) Show that $\widetilde{Q}^t(\theta^t) = \widetilde{Q}^t(\theta^{t+1})$.

(ii) Show that $\theta^t = \theta^{t+1}$.

(iii) Show that $D\widetilde{Q}^t(\theta^t) = \mathbf{0}$

(iv) Show (by way of contradiction) that if $D\ell(\theta^t) \neq \mathbf{0}$, then there exists $\mathbf{v}$ such that the directional derivative $D_{\mathbf{v}}\ell(\theta^t)$ satisfies $D_{\mathbf{v}}\ell(\theta^t) < 0$.

(v) Use the limit definition of the directional derivative and the two properties (i) and (ii) from the beginning of Section 13.2.2 to show that $D_{\mathbf{v}}\widetilde{Q}^t(\theta^t) \leq D_{\mathbf{v}}\ell(\theta^t)$

(vi) Show that the previous step contradicts something that you have already proved, and hence the assumption in (iv) above is false; that is, $D\ell(\theta^t)$ vanishes, as required.

13.5. Prove that the update equations (13.8) and (13.9) for EM applied to the coin flips in Example 13.2.5 and 13.2.7 are correct; that is, show that these do indeed give the correct values of $\operatorname{argmax}_{\boldsymbol{\theta}} Q^t(\boldsymbol{\theta})$.

13.6. Code up the EM algorithm for the Binomial mixture model of Example 13.2.5. Your code should accept an array $\mathtt{z}$ (the sample $z_1, \ldots, z_n$), and, optionally, an initial guess of the parameter $\boldsymbol{\theta}^0 = (\theta_A^0, \theta_B^0)$. If no initial $\boldsymbol{\theta}^0$ is provided, construct one randomly (What's a good way to do this?). With each iteration and existing estimate $\boldsymbol{\theta}^t$, your code should compute $q_i^t(x)$ for each $i \in \{1, \ldots, n\}$ and each $x \in \{A, B\}$ and the updates θ_A^{t+1} and θ_B^{t+1}. Include a stopping criterion (a maximum number of iterations and some measure of how well the estimates are converging). Return the final value of $\boldsymbol{\theta}^t$.

(i) Apply your code to the data from Example 13.2.5. What are its predictions for the values of θ_A and θ_B?

(ii) Let $\theta_A = 0.3$ and $\theta_B = 0.8$, and take a draw of length 20 from a Bernoulli($\frac{1}{2}$) random variable to get values $x_1, \ldots, x_{20}$ for the latent variable $X \in \{A, B\}$. For each of the twenty values of x_i, draw a corresponding z_i from a Binomial$(10, \theta_{x_i})$. The sequence $\mathbf{D} = \{z_1, \ldots, z_{20}\}$ is your data set.

(iii) Apply your code to the data $\mathbf{D}$ from the previous step. What are its predictions for the values of θ_A and θ_B?

13.7. Prove that the (13.16) gives the value of $\mathbf{w}$ that maximizes (13.15), and that (13.17) gives the maximizing $\boldsymbol{\mu}$ and Σ.
Hint: For the computation of Σ you may want to differentiate with respect to Σ^{-1}. Also feel free to assume the following properties of matrices and derivatives:

(i) $D_A \log(\det(A)) = (A^{-1})^{\mathsf{T}}$ (If you don't want to assume this fact, you can verify it directly with a tedious computation, first by expressing both $\det(A)$ and A^{-1} in terms of their cofactor expansions and then taking the partial derivatives $\frac{\partial}{\partial A_{ij}}$ of the expression $\log(\det(A))$ for each i, j).

(ii) $\operatorname{tr}(ABC) = \operatorname{tr}(CAB) = \operatorname{tr}(BCA)$.

(iii) $(\mathbf{z} - \boldsymbol{\mu})^\mathsf{T} A(\mathbf{z} - \boldsymbol{\mu}) = \operatorname{tr}((\mathbf{z} - \boldsymbol{\mu})^\mathsf{T} A(\mathbf{z} - \boldsymbol{\mu}))$

(iv) $D_A \operatorname{tr}(AB) = B^\mathsf{T}$

13.8. In your own words explain what the EM algorithm is for and why one might want to use it instead of something like Newton's method or gradient descent.

13.9. K-means vs. EM for GMMs:

(i) In your own words, explain how the k-means algorithm could be thought of as a simplified variant of EM for GMMs.

(ii) Explain the relative benefits and disadvantages of doing clustering using EM for GMMs as compared to the k-means algorithm.

(iii) Draw or plot a dataset where clustering using k-means would perform poorly compared to clustering with a GMM learned using EM.

13.10. Prove Proposition 13.4.5.

13.11. Prove Proposition 13.4.7.

13.12. Prove the Bellman relation (13.31) holds.

13.13. Explain in your own words the difference between finding the most likely path (Question (iii) of Section 13.4.1) versus filtering (or smoothing) to find each of the most likely states (Question (ii) of Section 13.4.1) and assembling them all into the sequence of most likely states.

13.14. Prove that (13.39), (13.40), and (13.41) do indeed give the maximizers they claim.

13.15. Prove (13.36).

13.16.* Prove Proposition 13.5.4.

Notes

14 Continuous State-Space HMMs

L'etat, c'est moi.
—Louis XIV

The Kalman filter is among the most notable innovations of the 20th Century. This algorithm recursively estimates the state variables in a noisy linear dynamical system as new observations are measured and as the system evolves in time. This method optimally updates the estimates of the system variables, for example the position and velocity of a projectile, by minimizing the mean-squared estimation error of the current state as noisy measurements are received. Each update provides the latest unbiased estimate of the system variables together with a measure on the uncertainty of those estimates in the form of a covariance matrix. Since the updating process is fairly general and relatively easy to compute, the Kalman filter can often be implemented in real-time.

The Kalman filter is used widely in virtually every technical or quantitative field. In engineering, for example, the Kalman filter is pervasive in the areas of navigation and global positioning, tracking, guidance, robotics, radar, fault detection and computer vision. It is also utilized in applications involving signal processing, voice recognition, video stabilization, and automotive control systems. In purely quantitative fields, the Kalman filter also plays an important role in time-series analysis, econometrics, mathematical finance, system identification, and neural networks.

It has been just over 50 years since Rudolf Kalman's seminal paper on state estimation [Kal60], which launched a major shift toward state-space dynamic modeling. This *tour de force* in mathematical systems theory, together with his two other groundbreaking papers [Kal63, KB61], helped secure a number of major awards for Kalman, including the IEEE Medal of Honor in 1974, the Kyoto Prize in 1985, the Steele Prize in 1986, the Charles Stark Draper Prize in 2008, and the U.S. National Medal of Science in 2009.

As is sometimes the case with revolutionary advances, some people were initially slow to accept Kalman's work. According to Grewal and Andrews [GA15, Chapter 1], Kalman's second paper [KB61] was actually rejected by an electrical engineering journal, since, as one of the referees put it, "it cannot possibly be true." However, with the help of Stanley F. Schmidt at the NASA Ames Research Center, the Kalman filter ultimately gained acceptance, as it was used successfully in the navigation systems for the Apollo missions, as well as several subsequent NASA projects and a number of military defense systems.

Today the *Kalman family* of state estimation methods, which includes the Kalman filter and its many variations, are the *de facto* standard for state estimation. There have been thousands patents awarded in the U.S. on applications or processes involving the Kalman filter. In academia, its influence is no less noteworthy. There have been over one-hundred thousand academic papers using the Kalman Filter.

14.1 Continuous State-Space HMMs

14.1.1 State Space Models

An HMM with a continuous state space is often called a *state space model*.

Example 14.1.1. Consider the situation of a water tank with constant water level. This corresponds to a very simple state-space model of the form

$$x_k = x_{k-1}$$

where the water level at time k is x_k.

If there is any chance of sloshing or other movement of the water in the tank, then it might make sense to add noise to the state equation, giving

$$x_k = x_{k-1} + w_k$$

with $w_k \sim \mathcal{N}(0, q)$ for some variance $q > 0$.

In this chapter we are interested in a specific type of state-space model called a *stochastic discrete-time linear dynamical system* of the form

$$\mathbf{x}_{k+1} = F\mathbf{x}_k + G\mathbf{u}_k + \mathbf{w}_k, \tag{14.1}$$

where the variable $\mathbf{x}_k \in \mathbb{R}^p$ represents the state at time k, and the variable $\mathbf{u}_k \in \mathbb{R}^q$ represents an input (or control). The term $\mathbf{w}_k \in \mathbb{R}^p$ is noise process, which is assumed to have mean zero with known covariance $Q_k > 0$. We assume that the system has a stochastic initial state $\mathbf{x}_0 = \boldsymbol{\mu}_0 + \mathbf{w}_{-1}$, where $\boldsymbol{\mu}_0$ is the expected value of $\mathbf{x}_0$ and $\mathbf{w}_{-1}$ is a zero-mean random variable with covariance $\mathbb{E}[\mathbf{w}_{-1}\mathbf{w}_{-1}^\mathsf{T}] = Q_{-1} > 0$.

Nota Bene 14.1.2. Essentially everything in the chapter extends easily to the case where the linear transformations F and G change at each step, with the model

$$\mathbf{x}_{k+1} = F_k\mathbf{x}_k + G_k\mathbf{u}_k + \mathbf{w}_k,$$

but for simplicity, we focus mostly on the time-invariant case here.

Example 14.1.3. Assume that a water tank is being filled at some fixed rate, so that the depth is also changing at a fixed rate c; that is, $x_k = x_{k-1} + c$ for all k. One way to model this situation is to treat the inflow as a control

$$x_k = x_{k-1} + c + w_k$$

For this model we have $F = G = 1$, $u_k = c$, and $w_k \sim \mathcal{N}(0, q)$ for some $q > 0$ and for all $k \in \mathbb{Z}^+$.

Alternatively, we could model this without a control variable by making the state space two dimensional, with $\mathbf{x}_k = (x_k^0, x_k^1)$ tracking both the depth x_k^1 and the inflow rate $x_k^0 = c$. In this model, we have

$$\mathbf{x}_k = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \mathbf{x}_{k-1} + 0\mathbf{u}_k + \mathbf{w}_k$$

For this model $F = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$, $G = 0$, and $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, Q)$, where $Q \in M_{2\times 2}$ is positive definite. One advantage of this second model is that it allows for some random fluctuation in the inflow rate, where the earlier model did not.

Example 14.1.4. Assume that a mass m is suspended by a spring with Hooke constant κ. Its height y is given by the differential equation

$$\ddot{y} + \omega^2 y = 0, \tag{14.2}$$

where $\omega = \sqrt{\frac{\kappa}{m}}$ is the angular frequency. The general solution to the differential equation is

$$y(t) = c_1 \cos(\omega t) + c_2 \sin(\omega t).$$

We can discretize (14.2) in time by breaking the domain $[0, T]$ into equally spaced subintervals defined by $0 = t_0 < t_1 < \cdots < t_n = T$, where $t_i = i\Delta t$ and $\Delta t = \frac{T}{n}$. The second derivative is approximated as

$$\ddot{y}(t_k) \approx \frac{y(t_{k+1}) - 2y(t_k) + y(t_{k-1})}{(\Delta t)^2},$$

which for $y(t_k) = y_k$ implies that

$$y_{k+1} = (2 - \omega^2(\Delta t)^2)y_k - y_{k-1}. \tag{14.3}$$

We can write this as a first-order system

$$\begin{bmatrix} y_{k+1} \\ y_k \end{bmatrix} = \begin{bmatrix} (2 - \omega^2(\Delta t)^2) & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_k \\ y_{k-1} \end{bmatrix}.$$

14.1.2 Filtering

In many settings the states are hidden, or unknown, but we have some observations that reveal information about the state. In this case we have the system

$$\mathbf{x}_k = F\mathbf{x}_{k-1} + G\mathbf{u}_{k-1} + \mathbf{w}_{k-1}, \tag{14.4a}$$
$$\mathbf{z}_k = H\mathbf{x}_k + \mathbf{v}_k, \tag{14.4b}$$

where the variable $\mathbf{x}_k \in \mathbb{R}^p$ represents the state at time k, the variable $\mathbf{u}_k \in \mathbb{R}^q$ the input, and the variable $\mathbf{z}_k \in \mathbb{R}^r$ the observation. The terms $\mathbf{w}_k \in \mathbb{R}^p$ and $\mathbf{v}_k \in \mathbb{R}^r$ are noise processes, which are assumed to have mean zero and be mutually uncorrelated, with known covariances $Q_k > 0$ and $R_k > 0$, respectively. We assume that the system has the stochastic initial state $\mathbf{x}_0 = \boldsymbol{\mu}_0 + \mathbf{w}_{-1}$, where $\boldsymbol{\mu}_0$ is the mean and $\mathbf{w}_{-1}$ is a zero-mean random variable with covariance $\mathbb{E}[\mathbf{w}_{-1}\mathbf{w}_{-1}^\mathsf{T}] = Q_{-1} > 0$.

Given m known observations $\mathbf{z}_1, \ldots, \mathbf{z}_m$ and k known inputs $\mathbf{u}_1, \ldots, \mathbf{u}_k$, where $k \geq m$, we wish to estimate the states $\mathbf{x}_0, \ldots, \mathbf{x}_k$. This is the *filtering problem*.

Example 14.1.5. Consider the situation of Example 14.1.1 of a water tank with constant water level. Assume the level is unknown but we take depth measurements at discrete time intervals with a water-level meter that is noisy but has no bias. Expressed mathematically, if the water level is x_k and the meter reading is z_k at time k, then

$$x_k = x_{k-1} + w_k$$
$$z_k = x_k + v_k,$$

where v_k is a noise process. Thus $H = 1$, and all other variables are as before. The filtering problem is to try to estimate $X_k = x_k$, given the noisy measurement $z_1, \ldots, z_m$.

Example 14.1.6. Consider the situation of Example 14.1.3, where a water tank is being filled at some fixed (known) rate. Assume the actual states x_k are unknown, but we take measurements z_k with a noisy depth-meter.

If the system is modeled with the inflow as a control variable, then we have

$$x_k = x_{k-1} + c + w_k$$
$$z_k = x_k + v_k,$$

so $F = G = H = 1$, $u_k = c$.

If the system is modeled without control but a two-dimensional state space, then

$$z_k = \begin{bmatrix} 0 & 1 \end{bmatrix} \mathbf{x}_k + v_k.$$

and $H = \begin{bmatrix} 0 & 1 \end{bmatrix}$.

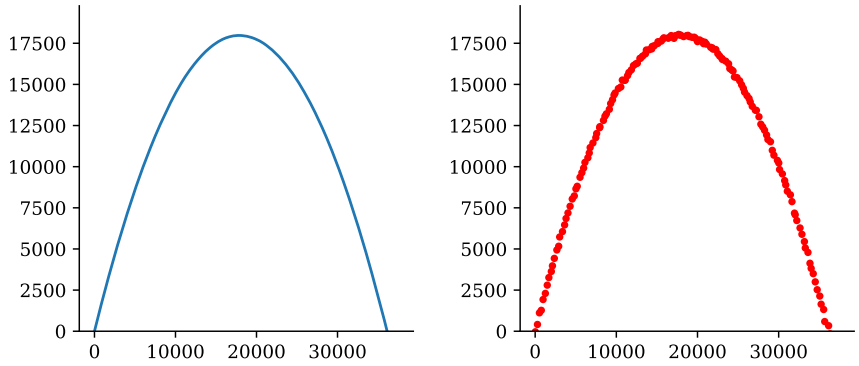


Figure 14.1: State sequence (left) and observation sequence $\mathbf{z}_k$ (right) for a projectile in the plane, as described in Example 14.1.7

Example 14.1.7. Assume that the state of a projectile in $\mathbb{R}^2$ is given by the vector $\mathbf{x} = (s_x, s_y, v_x, v_y)$, where s_x and s_y are the horizontal and vertical components of position, respectively, with corresponding velocity components $v_x = \dot{s}_x$ and $v_y = \dot{s}_y$.

Assuming no air resistance, there is only a vertical force due to gravity. As a continuous-time system, we can write the evolution of the system as

$$\frac{dv_x}{dt} = 0$$
$$\frac{dv_y}{dt} = -g,$$

where $g = -9.8m/s^2$ is acceleration due to gravity. We discretize the system by approximating to first order as

$$\begin{aligned}
\frac{s_x(k+1) - s_x(k)}{\Delta t} &= v_x(k) \\
\frac{s_y(k+1) - s_y(k)}{\Delta t} &= v_y(k) \\
\frac{v_x(k+1) - v_x(k)}{\Delta t} &= 0 \\
\frac{v_y(k+1) - v_y(k)}{\Delta t} &= -g.
\end{aligned}$$

where Δt is the interval of time between measurements. We may write this in the form of (14.4a) as

$$\mathbf{x}_{k+1} = F\mathbf{x}_k + \mathbf{u} + \mathbf{w}_k, \tag{14.5}$$

where

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g\Delta t \end{bmatrix}.$$

Assume the noise process $\mathbf{w}_k$ (for example, wind gusts that slightly perturb the projectile at each step) has zero mean with covariance $Q_k > 0$. Assuming that there are no significant wind gusts, it seems reasonable to assume that the model (14.5) is fairly accurate, so we could take the variance Q_k of the model error to be a small constant. If all distances are measured in meters and the time step is $\Delta_t = 0.1s$, then the covariance Q of the model error should not be large, so it seems reasonable to take $Q_k = Q = 0.1I_4$ for all $k \in \mathbb{N}$.

Assume the projectile is tracked by a radar device that is only able to measure the position of the projectile—not the velocity. We write the observation equation (14.4b) as

$$\mathbf{z}_k = H\mathbf{x}_k + \mathbf{v}_k, \quad \text{where} \quad H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

where $\mathbf{v}_k$ is the measurement noise (the radar device is imperfect), which we assume has zero mean with covariance $R_k > 0$. The tracking radar is likely to make much larger errors than the model, so let's take $R_k = R = 5000I_2$.

Figure 14.1 shows the true state sequence (or, rather, the positions (s_x, s_y), corresponding to the first two coordinates of the state) on the left and the (noisy) observation sequence $\mathbf{z}_k$ on the right.

In this situation the filtering problem is to estimate the state $\mathbf{x}_k$ at time k, given k observations $\mathbf{z}_1, \ldots, \mathbf{z}_k$ and assuming constant $\mathbf{u}$.

14.2 The Kalman Filter

Measured observations are often prone to significant noise, due to restrictions on measurement accuracy. For example, most commercial GPS devices can provide a good estimate of geolocation, but only within a dozen meters or so. The Kalman filter is an algorithm that takes a sequence of noisy observations made over time and attempts to get rid of the noise, producing more accurate estimates than the original observations.

14.2.1 Setup and Algorithm

The Kalman filter is an efficient, recursive method for solving the filtering problem for the system (14.4) (which we repeat here for the reader's convenience):

$$\begin{aligned} \mathbf{x}_k &= F\mathbf{x}_{k-1} + G\mathbf{u}_k + \mathbf{w}_k, \\ \mathbf{z}_k &= H\mathbf{x}_k + \mathbf{v}_k \\ \mathbb{E}[\mathbf{w}_k] &= \mathbf{0} \text{ and } \operatorname{Cov}(\mathbf{w}_k) = Q_k \\ \mathbb{E}[\mathbf{v}_k] &= \mathbf{0} \text{ and } \operatorname{Cov}(\mathbf{v}_k) = R_k. \end{aligned} \tag{14.4 revisited}$$

The Kalman filter gives the minimum-variance linear unbiased estimator (MVLUE) $\widehat{\mathbf{x}}_k$ of the unknown state variable $\mathbf{x}_k$, and it also estimates the covariance $P_k = \mathbb{E}[(\mathbf{x}_k - \widehat{\mathbf{x}}_k)(\mathbf{x}_k - \widehat{\mathbf{x}}_k)^\mathsf{T}]$ of the estimator. It does this in terms of only the previous estimate $\widehat{\mathbf{x}}_{k-1}$ and P_{k-1} and the latest data $\mathbf{u}_k$ and $\mathbf{z}_k$. The Kalman filter has the advantage that once $\widehat{\mathbf{x}}_k$ and P_k are computed, we no longer need $\widehat{\mathbf{x}}_{k-1}$, $\mathbf{z}_k$, or $\mathbf{u}_k$ to compute future estimates.

We assume that the system has the stochastic initial state $\mathbf{x}_0 = \boldsymbol{\mu}_0 + \mathbf{w}_{-1}$, where $\boldsymbol{\mu}_0$ is the mean and $\mathbf{w}_{-1}$ is a zero-mean random variable with covariance $\mathbb{E}[\mathbf{w}_{-1}\mathbf{w}_{-1}^\mathsf{T}] = Q_{-1} > 0$. Given k noisy observations $\mathbf{z}_1, \dots, \mathbf{z}_k$ and k known inputs $\mathbf{u}_0, \dots, \mathbf{u}_{k-1}$, the Kalman filter is initialized with $\widehat{\mathbf{x}}_0 = \boldsymbol{\mu}_0$ and $P_0 = Q_{-1}$. It then proceeds in two steps:

Predict: The predictive step gives a prediction $\widehat{\mathbf{x}}_{k|k-1}$ for $\mathbf{x}_k$ and covariance matrix $P_{k|k-1}$ using only the previous estimate $\widehat{\mathbf{x}}_{k-1}$, P_{k-1} and the input $\mathbf{u}_k$. The subscript notation $k|k-1$ indicates that this is a prediction for the kth state or covariance matrix, but it is only using the observations up to $\mathbf{z}_{k-1}$. The prediction is computed as

$$\widehat{\mathbf{x}}_{k|k-1} = F\widehat{\mathbf{x}}_{k-1} + G\mathbf{u}_k. \tag{14.6a}$$

$$P_{k|k-1} = FP_{k-1}F^\mathsf{T} + Q_k. \tag{14.6b}$$

Update: The update step uses the observation $\mathbf{z}_k$ to correct the prediction-covariance pair $(\widehat{\mathbf{x}}_{k|k-1}, P_{k|k-1})$ from the first step, giving a better estimate-covariance pair $(\widehat{\mathbf{x}}_k, P_k) = (\widehat{\mathbf{x}}_{k|k}, P_{k|k})$, as

$$P_k = (P_{k|k-1}^{-1} + H^\mathsf{T}R_k^{-1}H)^{-1}, \tag{14.7a}$$

$$\widehat{\mathbf{x}}_k = \widehat{\mathbf{x}}_{k|k-1} - P_kH^\mathsf{T}R_k^{-1}(H\widehat{\mathbf{x}}_{k|k-1} - \mathbf{z}_k). \tag{14.7b}$$

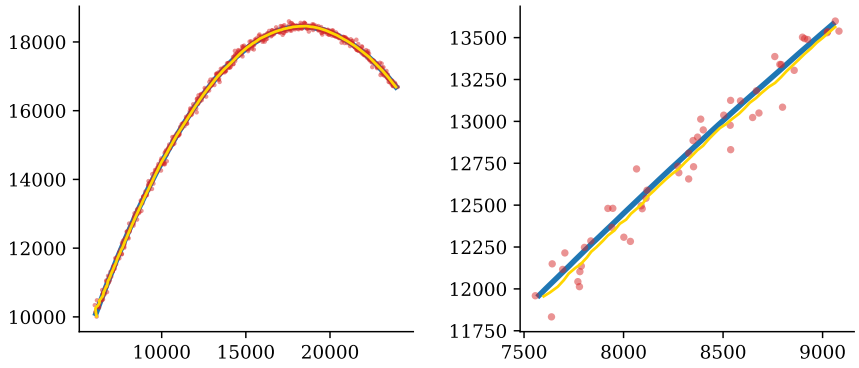


Figure 14.2: State estimates (yellow) produced by the Kalman filter (detailed view on the right), together with observations (red) and true state sequence (blue) for the projectile in Example 14.2.1.

Example 14.2.1. Consider again the projectile from Example 14.1.7, and assume that the radar sensor has taken observations from time steps 200 through 800. Using these observations, we seek to estimate the corresponding true states of the projectile.

To begin using the Kalman filter, we must start with an initial state estimate $\widehat{\mathbf{x}}_{200}$ and an estimate of the accuracy (the covariance P_{200} of the estimate $\widehat{\mathbf{x}}_{200}$). It is not unreasonable to take $\mathbf{z}_{200}$ as the initial position of the projectile. We can approximate velocity at time 200 by $\frac{\mathbf{z}_{201}-\mathbf{z}_{200}}{\Delta_t}$, but if the observation error is large, this might not be a good estimate.

Instead, take the average $\overline{\mathbf{v}}$ of the approximate velocities

$$\frac{\mathbf{z}_1-\mathbf{z}_0}{\Delta_t}, \frac{\mathbf{z}_2-\mathbf{z}_1}{\Delta_t}, \ldots, \frac{\mathbf{z}_9-\mathbf{z}_8}{\Delta_t}$$

for the first 9 steps. Because the covariance of the observation error is 50,000 times the size of the model error, it does not seem unreasonable to assume that this initial estimate $\widehat{\mathbf{x}}_{200}$ has covariance less than $10^6 Q$. Thus we take $P_{200} = 10^6 Q$.

To estimate the state $\mathbf{x}_{201}$, we first compute the prediction

$$P_{201|200} = FP_{200}F^{\mathsf{T}} + Q \qquad \text{and} \qquad \widehat{\mathbf{x}}_{201|200} = F\widehat{\mathbf{x}}_{200} + u.$$

The update step now gives

$$P_{201} = (P_{201|200}^{-1} + H^{\mathsf{T}}R^{-1}H)$$

and

$$\widehat{\mathbf{x}}_{201} = \widehat{\mathbf{x}}_{201|200} - P_{201}H^{\mathsf{T}}R^{-1}(H\mathbf{x}_{201|200} - \mathbf{z}_{201}).$$

The process can now be repeated, using the resulting P_{201} and $\widehat{\mathbf{x}}_{201}$ in (14.6) and (14.7) to compute $\widehat{\mathbf{x}}_{202}$ and P_{202}, and so forth, until we have computed estimates up to $\widehat{\mathbf{x}}_{800}$ and P_{800}—one for each observation.

14.2.2 Effects of Initial Conditions

If the model is correctly specified, meaning that F, G, H, Q, and R are correct, and the initial guess $\widehat{x}_0$ and covariance P_0 are reasonable, then the Kalman filter does very well. Consider, for example, the situation of the constant water level model of Examples 14.1.1 and 14.1.5, where the initial state value is $x_0 = 7$. The plots in Figure 14.3 show the true state in blue, the measurements in gray, and the Kalman estimates in yellow, assuming that the model is correctly specified, and the initial starting estimate is $\widehat{x}_0 = 6.5$. The initial covariance P_0 tells the model how much to trust the predictions, as compared to the observations, with larger covariance indicating that the true state could be farther from the predictions (so trust the predictions less), and smaller covariance indicating that the true value should be close to the prediction. Over time, the updates to the estimates not only improve the estimates $\widehat{x}_k$ of the state, but also the covariance P_k of the Kalman estimator.

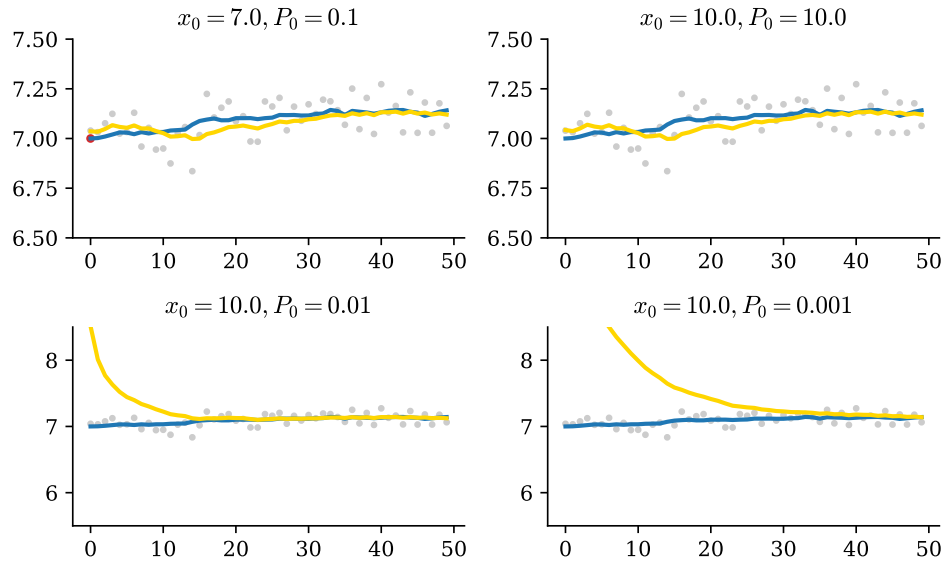


Figure 14.3: The Kalman filter applied to the system of Examples 14.1.1 and 14.1.5, where the initial state value is $x_0 = 7$. The true state in shown in blue, the measurements in gray, and the Kalman estimates in yellow, assuming that the model is correctly specified. In the upper left panel the initial estimate $\widehat{x}_0$ (red) is exactly correct and the Kalman filter does a good job of approximating the correct sequence of states. In the upper right panel the initial estimate $\widehat{x}_0 = 10$ is not very close to the true value $x_0 = 7.0$, but the initial covariance is large, so the update step moves even the first estimate very far toward the observations, and the estimates $\widehat{x}_k$ quickly approach the true values and thereafter stay relatively close. The bottom panels both start at $\widehat{x}_0 = 10$ (as in the upper right panel), but now the initial covariance P_0 is small, indicating that the model should give much more weight to the predictions. The sequence $\widehat{x}_k$ still eventually gets close to the true state values; but the smaller P_0 is, the longer that takes.

14.2.3 Effects of Model Misspecification

If the model is misspecified in the Kalman filter, it can be difficult for the filter to do very well, as shown in Figures 14.4, 14.5, and 14.6.

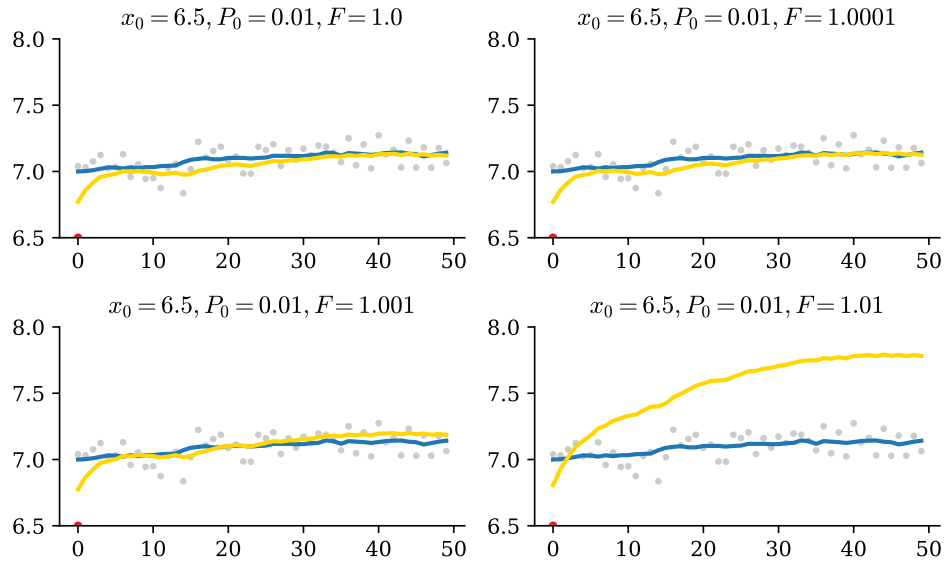


Figure 14.4: The Kalman filter applied to the system of Examples 14.1.1 and 14.1.5, where the initial state value is $x_0 = 7$. The true state is shown in blue, the noisy measurements in gray, and the Kalman estimates in yellow. When the model is not correct it can severely affect the accuracy of the Kalman filter. In the upper left panel the model is correct ($F = 1$), but in the other panels the choice of F in the Kalman filter is larger than the true value, corresponding to a system where each prediction is a little larger than the previous state. When the discrepancy between the model and the true value is small (upper right panel), the filter can still correct for the model misspecification. But for larger discrepancies, the filter begins to struggle: some error is visible in the lower left panel for $F = 1.001$, and a lot of error in the lower right, where $F = 1.01$.

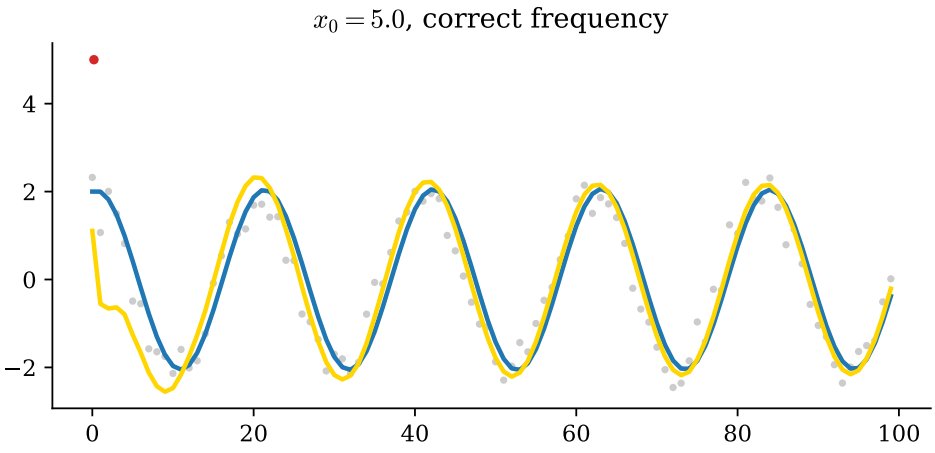


Figure 14.5: The Kalman filter applied to the system of Example 14.1.4, where the initial state value is $\widehat{\mathbf{x}}_0 = (5, 5)$. The second coordinate of the true state in shown in blue, the measurements in gray, and the second coordinate of the Kalman estimates in yellow. The model is correctly specified; in particular, it has the right frequency, which allows it to approach the correct state sequence eventually.

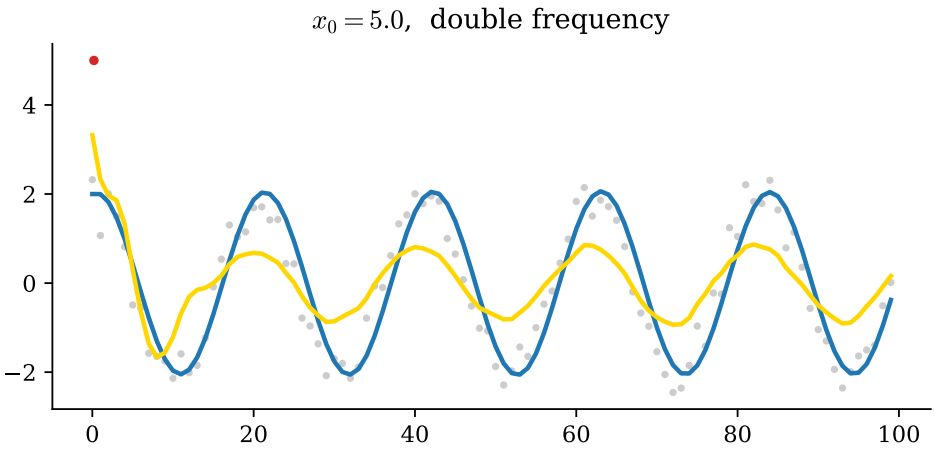


Figure 14.6: The Kalman filter applied to the model of Example 14.1.4, where the initial state value is $\widehat{\mathbf{x}}_0 = (5, 5)$. The second coordinate of the true state in shown in blue, the measurements in gray, and the Kalman estimates in yellow. Here the model is incorrectly specified—the model has a frequency that is slightly more than twice that of the true system. This prevents the filter from accurately estimating the true amplitude, but after a bit of initial trouble, it does seem to start to approximate the true frequency.

14.2.4 Predicting and Rewinding Without Observations

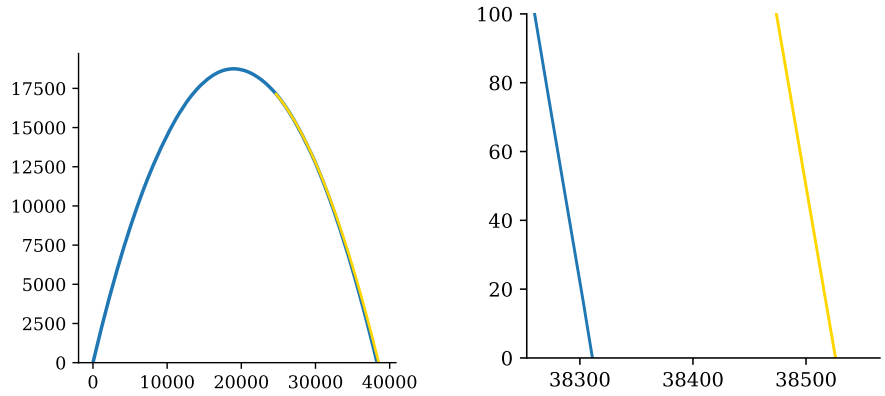


Figure 14.7: Predicted and actual point of impact (detailed view on right) for the projectile of Example 14.2.1. The predictions past time step 800 have no corresponding observations, so the correction step of the Kalman filter cannot be used.

To estimate $\mathbf{x}_t$ beyond the available observations (for $t > m$), one can still use the predictor step (14.6) without the correction (14.7). This is justified by the fact that the expected state noise has mean zero ($\mathbb{E}[\mathbf{w}_k] = 0$) at any time k. Thus, given a current state estimate $\widehat{\mathbf{x}}_{n|m}$ using only measurements up through time m, the expected state at time $n+1$ is

$$\widehat{\mathbf{x}}_{n+1|m} = F_{n+1}\widehat{\mathbf{x}}_{n|m} + \mathbf{u}_{n+1}$$

These estimates will, of course, be less accurate because they don't have observations to correct the predictions, but they are the best we can do with the available information. In particular, they are still the minimum-variance unbiased estimates. See Figure 14.7 for an example of this applied to the projectile problem.

If the transition matrix F is invertible, then we can also reverse the system and iterate backward in time to infer information about states of the system prior to measured observations. The system is reversed by

$$\mathbf{x}_k = F^{-1}(\mathbf{x}_{k+1} - G\mathbf{u} - \mathbf{w}_{k+1}).$$

Again $\mathbb{E}[\mathbf{w}_{k+1}] = 0$ at any time k, so the best estimate for $\mathbf{x}_k$ simplifies to

$$\widehat{\mathbf{x}}_k = F^{-1}(\mathbf{x}_{k+1} - G\mathbf{u}).$$

See Figure 14.8 for an example of this applied to the projectile problem.

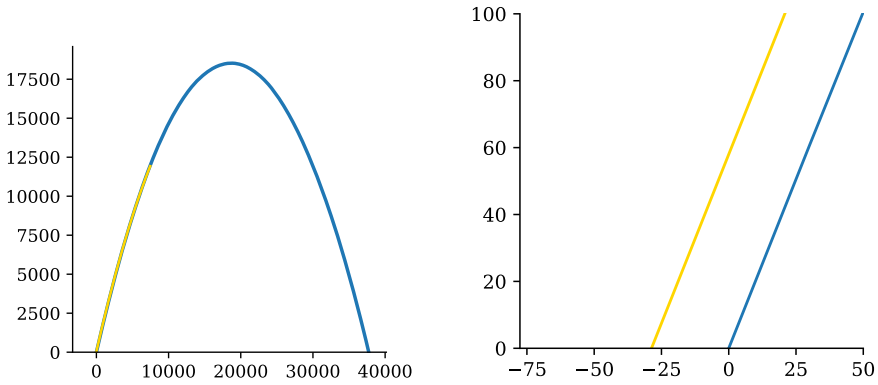


Figure 14.8: Predicted (yellow) versus actual (blue) point of origin (detailed view on right) for the projectile of Example 14.2.1.

14.2.5 Additional Extensions to the Kalman Filter

There are many other extensions of the basic Kalman filter. The *extended Kalman filter (EKF)* can be applied to nonlinear state space models by locally linearizing, using the derivative. Unfortunately, the EKF is somewhat finicky, and some would even say it stinks. The most popular alternative to the EKF is called the *unscented Kalman filter*. Alternatively, *particle filters* use Monte Carlo methods for their updates rather than linear projections.

14.3 Kalman Filter: Derivation and Improvements

14.3.1 Bayesian Derivation of the Kalman Filter

In the special case that the noise processes are all Gaussian, the Kalman filter can be derived by looking at the posterior $P(\mathbf{x}_k \,|\, \widehat{\mathbf{x}}_{k-1}, \mathbf{z}_k)$, which itself is Gaussian. The mean of that posterior is $\widehat{\mathbf{x}}_k$, which is also the MAP, and the covariance is P_k.

Remark 14.3.1. We recall that for a normal random variable $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with support in $\mathbb{R}^n$, if we define a random variable $Y = AX + \mathbf{b}$ for a constant matrix $A \in M_{m,n}$ and a constant vector $\mathbf{b} \in \mathbb{R}^m$, then Y is distributed as

$$Y = AX + \mathbf{b} \sim \mathcal{N}(A\boldsymbol{\mu} + \mathbf{b}, A\Sigma A^{\mathsf{T}}). \tag{14.8}$$

Moreover, if $Z \sim \mathcal{N}(\boldsymbol{\nu}, \Lambda)$ is another normal random variable that is independent of X, then $X + Z$ is also normal and

$$X + Z \sim \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{\nu}, \Sigma + \Lambda). \tag{14.9}$$

We set $P_0 = Q_{-1}$ and assume the system has the form of (14.4) and the following random variables are normal:

$$X_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, P_0) \tag{14.10}$$
$$\mathbf{w}_k \sim \mathcal{N}(0, Q_k) \qquad \forall k \geq 0 \tag{14.11}$$
$$\mathbf{v}_k \sim \mathcal{N}(0, R_k) \qquad \forall k \geq 0. \tag{14.12}$$

We want to choose $\mathbf{x}_0, \ldots, \mathbf{x}_k$ to maximize the probability $P(\mathbf{x}_0, \ldots, \mathbf{x}_k \,|\, \mathbf{z}_1, \ldots \mathbf{z}_k)$. The independence properties of the system (the state sequence is Markov and, after conditioning on X_k, the variable Z_k is independent of all other variables in the system) combined with Bayes' rule give

$$\begin{aligned} P(\mathbf{x}_0, \ldots, \mathbf{x}_k \,|\, \mathbf{z}_1, \ldots \mathbf{z}_k) &\propto P(\mathbf{z}_1, \ldots \mathbf{z}_k \,|\, \mathbf{x}_0, \ldots, \mathbf{x}_k) P(\mathbf{x}_0, \ldots, \mathbf{x}_k) \\ &= \prod_{i=1}^{k} P(\mathbf{z}_i \,|\, \mathbf{x}_i)\, P(\mathbf{x}_0) \prod_{j=1}^{k} P(\mathbf{x}_j \,|\, \mathbf{x}_{j-1}) \\ &= P(\mathbf{x}_0) \prod_{i=1}^{k} P(\mathbf{z}_i \,|\, \mathbf{x}_i) P(\mathbf{x}_i \,|\, \mathbf{x}_{i-1}) \end{aligned}$$

To approximate the maximizer, we proceed inductively. The maximizer

$$\operatorname*{argmax}_{\mathbf{x}_0} P(\mathbf{x}_0) = \operatorname*{argmax}_{\mathbf{x}_0} \mathcal{N}(\mathbf{x}_0 \,|\, \boldsymbol{\mu}_0, P_0)$$

is the mean $\boldsymbol{\mu}_0$, which we denote by $\widehat{\mathbf{x}}_0$. For each $k > 0$, assuming that $X_{k-1} \sim \mathcal{N}(\widehat{\mathbf{x}}_{k-1}, P_{k-1})$, we wish to compute (and maximize)

$$P(\mathbf{x}_k \,|\, \mathbf{z}_k, \mathbf{x}_{k-1}) \propto P(\mathbf{z}_k \,|\, \mathbf{x}_k) P(\mathbf{x}_k \,|\, \mathbf{x}_{k-1}).$$

Applying the result of Remark 14.3.1 to $X_k = FX_{k-1} + G\mathbf{u}_k + \mathbf{w}_k$ gives

$$X_k \,|\, \mathbf{x}_{k-1} \sim \mathcal{N}(F\mathbf{x}_{k-1} + G\mathbf{u}_k,\ F^{\mathsf{T}} P_{k-1} F + Q_k).$$

The mean of $X_k \,|\, \mathbf{x}_{k-1}$ is $F\mathbf{x}_{k-1} + G\mathbf{u}_k$, which is exactly the prediction term $\widehat{\mathbf{x}}_{k|k-1}$ in (14.6a). And the covariance is $F^\mathsf{T} P_{k-1} F + Q_k$, which is exactly the predicted covariance $P_{k|k-1}$ in (14.6b).

Thus we have

$$\begin{aligned}
P(\mathbf{x}_k \,|\, \mathbf{z}_k, \mathbf{x}_{k-1}) &\propto P(\mathbf{z}_k \,|\, \mathbf{x}_k) \exp\left(-\frac{1}{2}(\mathbf{x}_k - \widehat{\mathbf{x}}_{k|k-1})^\mathsf{T} P_{k|k-1}(\mathbf{x}_k - \widehat{\mathbf{x}}_{k|k-1})\right) \\
&\propto \exp\Big(-\frac{1}{2}\big((\mathbf{z}_k - H\mathbf{x}_k)^\mathsf{T} R_k^{-1}(\mathbf{z}_k - H\mathbf{x}_k) \\
&\qquad + (\mathbf{x}_k - \widehat{\mathbf{x}}_{k|k-1})^\mathsf{T} P_{k-1}^{-1}(\mathbf{x}_k - \mathbf{x}_{k-1})\big)\Big) \\
&\propto \exp\Big(-\frac{1}{2}\big(\mathbf{x}_k^\mathsf{T}(H^\mathsf{T} R_k^{-1} H + P_{k|k-1}^{-1})\mathbf{x}_k \\
&\qquad - 2\mathbf{x}_k^\mathsf{T}(H^\mathsf{T} R_k^{-1}\mathbf{z}_k + P_{k|k-1}^{-1}\widehat{\mathbf{x}}_{k|k-1})\big)\Big) \\
&= -\exp\Big(-\frac{1}{2}\big(\mathbf{x}_k^\mathsf{T}(P_k)^{-1}\mathbf{x}_k \\
&\qquad - 2\mathbf{x}_k^\mathsf{T} P_k^{-1} P_k(H^\mathsf{T} R_k^{-1}\mathbf{z}_k + P_{k|k-1}^{-1}\widehat{\mathbf{x}}_{k|k-1})\big)\Big) \\
&\propto -\exp\left(-\frac{1}{2}(\mathbf{x}_k - \boldsymbol{\nu})^\mathsf{T}(P_k)^{-1}(\mathbf{x}_k - \boldsymbol{\nu})\right),
\end{aligned}$$

where P_k is given by the update equation (14.7a) and $\boldsymbol{\nu} = P_k(H^\mathsf{T} R_k^{-1} z_k + P_{k|k-1}^{-1}\widehat{\mathbf{x}}_{k|k-1})$. Thus we have

$$P(\mathbf{x}_k \,|\, \mathbf{z}_k, \mathbf{x}_{k-1}) \sim \mathcal{N}(\boldsymbol{\nu}, P_k).$$

All that remains is to show that $\boldsymbol{\nu} = \widehat{\mathbf{x}}_k$, as given by the update equation (14.7b). This is equivalent to showing that

$$\begin{aligned}
(H^\mathsf{T} R_k^{-1} z_k + P_{k|k-1}^{-1}\widehat{\mathbf{x}}_{k|k-1}) &= P_k^{-1}\left(\widehat{\mathbf{x}}_{k|k-1} - P_k H^\mathsf{T} R_k^{-1}(H\widehat{\mathbf{x}}_{k|k-1} - \mathbf{z}_k)\right) \\
&= P_k^{-1}\widehat{\mathbf{x}}_{k|k-1} - H^\mathsf{T} R_k^{-1}(H\widehat{\mathbf{x}}_{k|k-1} - \mathbf{z}_k) \\
&= (H^\mathsf{T} R_k^{-1} H + P_{k|k-1}^{-1})\widehat{\mathbf{x}}_{k|k-1} - H^\mathsf{T} R_k^{-1}(H\widehat{\mathbf{x}}_{k|k-1} - \mathbf{z}_k) \\
&= P_{k|k-1}^{-1}\widehat{\mathbf{x}}_{k|k-1} + H^\mathsf{T} R_k^{-1}\mathbf{z}_k.
\end{aligned}$$

Thus we have shown that

$$P(\mathbf{x}_k \,|\, \mathbf{z}_k, \mathbf{x}_{k-1}) \sim \mathcal{N}(\widehat{\mathbf{x}}_k, P_k),$$

14.3.2 Efficient Form of the Kalman Filter

With some matrix algebra, the Kalman filter can be expressed in an equivalent, but more efficient form. Using the same prediction step as the previous section

$$\widehat{\mathbf{x}}_{k|k-1} = F\widehat{\mathbf{x}}_{k-1} + G\mathbf{u}_k \tag{14.13a}$$

$$P_{k|k-1} = FP_{k-1}F^\mathsf{T} + Q_k, \tag{14.13b}$$

an equivalent but more efficient form of the update step (14.7) is to compute

$$K_k = P_{k|k-1}H^\mathsf{T}(HP_{k|k-1}H^\mathsf{T} + R_k)^{-1}, \tag{14.14a}$$

$$\widehat{\mathbf{x}}_k = \widehat{\mathbf{x}}_{k|k-1} + K_k(\mathbf{z}_k - H\widehat{\mathbf{x}}_{k|k-1}), \tag{14.14b}$$

$$P_k = (I - K_k H)P_{k|k-1}. \tag{14.14c}$$

The quantity K_k is called the *Kalman Gain*. One can show (see Exercise 14.7) that Equations (14.7) and (14.14) are equivalent by Lemma 14.3.2, below.

The difference $\mathbf{y}_k = \mathbf{z}_k - H\widehat{\mathbf{x}_k}$ between the actual measurement $\mathbf{z}_k$ and the predicted output $H\widehat{\mathbf{x}}_{k|k-1}$ is an indicator of how much the prediction and observation disagree. The Kalman gain K_k determines how that discrepancy affects the update $\widehat{\mathbf{x}}_k$. One could also weight K_k

Lemma 14.3.2. *Assume that $A, C > 0$ and for notational convenience, denote*

$$S = BAB^{\mathsf{T}} + C^{-1} \quad \textit{and} \quad G = AB^{\mathsf{T}}S^{-1}.$$

Then

$$(A^{-1} + B^{\mathsf{T}}CB)^{-1} = (I - GB)A \tag{14.15}$$

$$= (I - GB)A(I - B^{\mathsf{T}}G^{\mathsf{T}}) + GC^{-1}G^{\mathsf{T}} \tag{14.16}$$

and

$$G = (A^{-1} + B^{\mathsf{T}}CB)^{-1}B^{\mathsf{T}}C. \tag{14.17}$$

Proof. The proof is Exercise 14.6 □

14.3.3 Minimum Variance Linear Unbiased Estimator

An important property of the Kalman filter is that it computes the minimum-variance linear unbiased estimator (MVLUE) for all the hidden states.

We consider a slightly more general form of the filtering problem. Given m known observations $\mathbf{z}_0, \ldots, \mathbf{z}_m$ and k known inputs $\mathbf{u}_0, \ldots, \mathbf{u}_{k-1}$, where $k \geq m$, we wish to estimate the states $\mathbf{x}_0, \ldots, \mathbf{x}_k$ in the system (14.4a)–(14.4b).

The Kalman filter gives the MVLUE of all the states $\mathbf{x}_0, \ldots, \mathbf{x}_k$. We begin by writing (14.4a)–(14.4b) as a large linear estimation problem with all the known measurements (the inputs and outputs) on the left and the unknown states on the right, with the noise terms.

$$\begin{array}{rcll}
\boldsymbol{\mu}_0 & = & \mathbf{x}_0 & - \quad \mathbf{w}_{-1}, \\
G_0\mathbf{u}_0 & = & \mathbf{x}_1 - F_0\mathbf{x}_0 & - \quad \mathbf{w}_0, \\
\mathbf{z}_1 & = & H_1\mathbf{x}_1 & + \quad \mathbf{v}_1, \\
 & \vdots & & \vdots \\
G_{m-1}\mathbf{u}_{m-1} & = & \mathbf{x}_m - F_{m-1}\mathbf{x}_{m-1} & - \quad \mathbf{w}_{m-1}, \\
\mathbf{z}_m & = & H_m\mathbf{x}_m & + \quad \mathbf{v}_m, \\
G_m\mathbf{u}_m & = & \mathbf{x}_{m+1} - F_m\mathbf{x}_m & - \quad \mathbf{w}_m, \\
 & \vdots & & \vdots \\
G_{k-1}\mathbf{u}_{k-1} & = & \mathbf{x}_k - F_{k-1}\mathbf{x}_{k-1} & - \quad \mathbf{w}_{k-1}.
\end{array} \tag{14.18}$$

The parameters to be estimated in this linear estimation problem are the states $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k$, which we write as

$$\mathbf{y}_k = (\mathbf{x}_0, \cdots, \mathbf{x}_k) \in \mathbb{R}^{(k+1)n}.$$

Denote the vector on the left side of (14.18) by

$$\mathbf{b}_{k|m} = (\boldsymbol{\mu}_0, G_0\mathbf{u}_0, \mathbf{z}_1, \ldots, G_{k-1}\mathbf{u}_{k-1}).$$

And denote by

$$A_{k|m} = \begin{bmatrix} I & 0 & 0 & 0 & \ldots \\ -F_0 & I & 0 & 0 & \ldots \\ 0 & H_1 & 0 & 0 & \ldots \\ \vdots & \ddots & \ddots & & \\ & & & -F_{k-1} & I \end{bmatrix}$$

the matrix describing the middle column of (14.18), so that the linear system takes the form

$$\mathbf{b}_{k|m} = A_{k|m}\mathbf{y}_k + \boldsymbol{\varepsilon}_{k|m},$$

where

$$\boldsymbol{\varepsilon}_{k|m} = (\mathbf{w}_{-1}, \mathbf{w}_0, \mathbf{v}_1, \cdots, \mathbf{w}_{m-1}, \mathbf{v}_m, \mathbf{w}_m, \cdots, \mathbf{w}_{k-1})$$

is a zero-mean random variable whose inverse covariance is the positive-definite block-diagonal matrix

$$W_{k|m} = \operatorname{diag}(Q_{-1}^{-1}, Q_0^{-1}, R_1^{-1}, \ldots, Q_{m-1}^{-1}, R_m^{-1}, Q_m^{-1} \ldots, Q_{k-1}^{-1}).$$

Each column of $A_{k|m}$ is a pivot column, so $A_{k|m}$ has full column rank and the weighted Grammian $A_{k|m}^{\mathsf{T}} W_{k|m} A_{k|m}$ is nonsingular.

By Gauss-Markov (see Section 7.2.2), the minimum-variance linear unbiased estimator $\widehat{\mathbf{y}}_{k|m}$ of $\mathbf{y}_k$ is given by the weighted least-squares solution

$$\widehat{\mathbf{y}}_{k|m} = (A_{k|m}^{\mathsf{T}} W_{k|m} A_{k|m})^{-1} A_{k|m}^{\mathsf{T}} W_{k|m} \mathbf{b}_{k|m}.$$

Computed naïvely, this would be a very expensive calculation because $A_{k|m}^{\mathsf{T}} W_{k|m} A_{k|m}$ is a very large matrix to try to invert. The Kalman filter algorithm allows us to compute this solution very cheaply.

The first step is to recognize that since the problem is cast as a linear model, it can also be solved by minimizing the following positive-definite objective function

$$J_{k|m}(\mathbf{y}) = \frac{1}{2}\|A_{k|m}\mathbf{y} - \mathbf{b}_{k|m}\|_{W_{k|m}}^2. \tag{14.19}$$

Since Newton's method on a quadratic objective converges in a single step (see Volume 2, Section 12.4.2), we can find its minimizer $\widehat{\mathbf{y}}_{k|m}$ by performing a single iteration of Newton's method given by

$$\widehat{\mathbf{y}}_{k|m} = \mathbf{y} - (A_{k|m}^{\mathsf{T}} W_{k|m} A_{k|m})^{-1} A_{k|m}^{\mathsf{T}} W_{k|m} (A_{k|m}\mathbf{y} - \mathbf{b}_{k|m}), \tag{14.20}$$

where $\mathbf{y} \in \mathbb{R}^{(k+1)n}$ is arbitrary. The Kalman filter boils down to (14.20) with a certain, well-chosen initial point $\mathbf{y}$. This is described Section 14.5

14.3.4 *Stable Computation of the Kalman Filter

One problem with the Kalman filter is that it is not very stable when Q_k is ill conditioned. One common problem is that the computed value of P_k may fail to be positive definite. The *square root algorithm* is a variant of the Kalman filter that corrects this problem.

To begin, we note that for any positive definite matrix $A \in M_n(\mathbb{R})$ there is a straightforward algorithm, similar to Gaussian elimination, with complexity $\sim n^3/3$, to construct the unique *Cholesky decomposition* of A, which gives a lower-triangular matrix $S \in M_n(\mathbb{R})$ such that $A = SS^\mathsf{T}$. Moreover, since S is lower triangular, its inverse is easy to compute.

The square root algorithm begins with a Cholesky decomposition $P_0 = S_0 S_0^\mathsf{T}$ of P_0 and a square root (not necessarily lower triangular) $Q_0 = T_0 T_0^\mathsf{T}$ of Q_0, and it proceeds inductively, like the Kalman filter. For any $k > 0$, let $M_{k-1} = \begin{bmatrix} F_{k-1}S_{k-1} & T_{k-1} \end{bmatrix}$ be the $n \times 2n$ matrix constructed by concatenating $F_{k-1}S_{k-1}$ and T_{k-1}. Note that $P_{k|k-1} = M_{k-1}M_{k-1}^\mathsf{T}$. Let $S_{k|k-1} \in M_n(\mathbb{R})$ be any square root (not necessarily lower triangular) of $M_{k-1}M_{k-1}^\mathsf{T}$

$$S_{k|k-1}S_{k|k-1}^\mathsf{T} = P_{k|k-1} = M_{k-1}M_{k-1}^\mathsf{T} \tag{14.21}$$

Now compute a Cholesky decomposition of $HP_{k|k-1}H^\mathsf{T} + R_k$

$$L_k L_k^\mathsf{T} = HP_{k|k-1}H^\mathsf{T} + R_k$$

and a Cholesky decomposition $V_k V_k^\mathsf{T} = R_k$. Since L_k and V_k are lower triangular, it is easy to compute $(L_k + V_k)^{-1}$. Define

$$S_k = S_{k|k-1}\left(I - S_{k|k-1}^\mathsf{T}H^\mathsf{T}(L_k^{-1})^\mathsf{T}(L_k + V_k)^{-1}HS_{k|k-1}\right)$$

Proposition 14.3.3. *Using the notation defined above, we have*

$$P_{k|k-1} = S_{k|k-1}S_{k|k-1}^\mathsf{T} \tag{14.22a}$$

$$P_k = S_k S_k^\mathsf{T} \tag{14.22b}$$

for all $k \in \mathbb{Z}^+$.

Proof. The proof proceeds by induction on k. Equation (14.22b) holds for $k = 0$ by definition, and (14.22a) holds by definition (14.24) for each $k > 0$, provided (14.22b) holds for $k - 1$.

It remains to show (14.22b), assuming (14.22a) holds for k. Define $W = (L_k^{-1})^\mathsf{T}(L_k + V_k)^{-1}$. The desired result follows by a straightforward, but tedious, manipulation of

$$\begin{aligned} S_k S_k^\mathsf{T} &= S_{k|k-1}\left(I - S_{k|k-1}^\mathsf{T}H^\mathsf{T}WHS_{k|k-1}\right)\left(I - S_{k|k-1}^\mathsf{T}H^\mathsf{T}WHS_{k|k-1}\right)^\mathsf{T} S_{k|k-1}^\mathsf{T} \\ &= S_{k|k-1}\Big(I - S_{k|k-1}^\mathsf{T}H^\mathsf{T}WHS_{k|k-1} - S_{k|k-1}^\mathsf{T}H^\mathsf{T}W^\mathsf{T}HS_{k|k-1} \\ &\quad + S_{k|k-1}^\mathsf{T}H^\mathsf{T}WHP_{k|k-1}H^\mathsf{T}W^\mathsf{T}HS_{k|k-1}\Big)S_{k|k-1} \end{aligned} \tag{14.23}$$

See [CC17, Section 7.2] for more details. □

The result of Proposition 14.3.3 and the preceding paragraphs is that the following algorithm computes the Kalman filter

(i) Set $\widehat{\mathbf{x}}_{0|0} = \mathbb{E}[\mathbf{x}_0] = \boldsymbol{\mu}_0$.

(ii) Compute the Cholesky decomposition S_0 of P_0

(iii) Set $k = 1$

(iv) Compute a square root (not necessarily lower triangular) $Q_{k-1} = T_{k-1}T_{k-1}^{\mathsf{T}}$ of Q_{k-1} and set $M_{k-1} = \begin{bmatrix} F_{k-1}S_{k-1} & T_{k-1} \end{bmatrix}$

(v) Compute a Cholesky decomposition[43] $S_{k|k-1}$ of $M_{k-1}M_{k-1}^{\mathsf{T}}$

(vi) The predict step of the Kalman filter can now be computed as

$$P_{k|k-1} = S_{k|k-1}S_{k|k-1}^{\mathsf{T}} \tag{14.24a}$$

$$\widehat{\mathbf{x}}_{k|k-1} = F\mathbf{x}_{k-1|k-1} + G\mathbf{u}_k \tag{14.24b}$$

(vii) Compute a Cholesky decomposition L_k of $HP_{k|k-1}H^{\mathsf{T}} + R_k$ and a Cholesky decomposition V_k of R_k.

(viii) Compute

$$S_k = S_{k|k-1}\left(I - S_{k|k-1}^{\mathsf{T}}H^{\mathsf{T}}(L_k^{-1})^{\mathsf{T}}(L_k + V_k)^{-1}HS_{k|k-1}\right)$$

(ix) Compute

$$K_k = P_{k|k-1}H^{\mathsf{T}}(L_k^{-1})^{\mathsf{T}}L_k^{-1}.$$

(x) The update step of the Kalman filter is now given by

$$\widehat{\mathbf{x}}_{k|k} = \widehat{\mathbf{x}}_{k|k-1} - K_k(\widehat{\mathbf{x}}_k - H\widehat{\mathbf{x}}_{k|k-1}) \tag{14.25a}$$

$$P_k = S_kS_k^{\mathsf{T}} \tag{14.25b}$$

(xi) Increment k and return to (iv).

[43] Since M_{k-1} is already partly lower triangular, this computation is faster than starting with the product $P_{k|k-1} = M_{k-1}M_{k-1}^{\mathsf{T}}$.

14.4 System Identification and Forecasting

Let $(Z_t)_{t\in\mathbb{N}}$ be the observation sequence of a state-space model

$$\begin{aligned} X_{t+1} &= FX_t + W_t \\ Z_t &= HX_t + \boldsymbol{\mu} + V_t \end{aligned}$$

for fixed $\boldsymbol{\mu}$ and with Gaussian noise processes $W_t \sim \mathcal{N}(\mathbf{0}, Q)$ and $V_t \sim \mathcal{N}(\mathbf{0}, R)$. For simplicity of notation, denote the parameters of the system by $\Theta = (F, H, \boldsymbol{\mu}, Q, R)$.

Suppose we have one observation sequence $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_m$, denoted collectively by $(\mathbf{z}_t)_{t=1}^m$. Although the system is a hidden Markov model, the Baum-Welch algorithm cannot be used because the state space is continuous. But we can use the Kalman filter to compute the likelihood of any given Θ, and then either use the EM algorithm to find the MLE for Θ or just use automatic differentiation of the likelihood combined with a gradient descent method or BFGS to find the MLE. This section treats the process in more detail.

14.4.1 Likelihood of Θ

Using the chain rule, we can factor the log likelihood of the model under these data as

$$\begin{aligned} \ell(\Theta) = \log L(\Theta) &= \log(f((\mathbf{z}_t)_{t=1}^m \,|\, \Theta)) \\ &= \sum_{t=1}^m \log f(\mathbf{z}_t | \mathbf{z}_{t-1}, \ldots, \mathbf{z}_1, \Theta) \end{aligned} \tag{14.26}$$

where f is the p.d.f. of the corresponding distribution. Assuming that the error terms are Gaussian, each conditional distribution $f(\mathbf{z}_t|\mathbf{z}_{t-1}, \ldots, \mathbf{z}_1, \Theta)$ in 14.26 is also Gaussian and is completely characterized by its mean and variance. Specifically, given Θ and observations $\mathbf{z}_1, \ldots, \mathbf{z}_{t-1}$, the theory of the Kalman filter guarantees that $\mathbf{x}_t$ is distributed as $\mathcal{N}(\widehat{\mathbf{x}}_{t|t-1}, P_{t|t-1})$, where $\widehat{\mathbf{x}}_{t|t-1}$ and $P_{t|t-1}$ are found during the predict step of the Kalman filter. Therefore, given Θ and observations $\mathbf{z}_1, \ldots, \mathbf{z}_{t-1}$, we have

$$\mathbf{z}_t = H\mathbf{x}_t + \boldsymbol{\mu} \sim \mathcal{N}(H\widehat{\mathbf{x}}_{t|t-1} + \boldsymbol{\mu}, HP_{t|t-1}H^\mathsf{T} + R).$$

The log likelihood thus satisfies

$$\ell(\Theta) = \sum_{t=1}^m \log \mathcal{N}(\mathbf{z}_t \,|\, H\widehat{\mathbf{x}}_{t|t-1} + \boldsymbol{\mu}, HP_{t|t-1}H^\mathsf{T} + R) \tag{14.27}$$

But the very first step of this process is not determined, because the $\widehat{\mathbf{x}}_{1|0}$ and $P_{1|0}$ are not known. Treat those initial values as additional parameters (in Θ) to be learned, the likelihood of $\Theta = (F, H, Q, R, \boldsymbol{\mu}, \mathbf{x}_{0|1}, P_{0|1})$ can now be computed from (14.27).

14.4.2 Identification and Fitting

Given a sequence of observations $(\mathbf{z}_t)_{t=1}^m$ and a choice of $\Theta = (F, H, Q, R, \boldsymbol{\mu}, \mathbf{x}_{1|0}, P_{1|0})$, running the Kalman filter with the initial values of $\mathbf{x}_{1|0}$ and $P_{1|0}$ and using (14.27) gives the likelihood of Θ. Although the likelihood is a complicated function of Θ computed using the Kalman filter, it can be maximized with a numerical optimizer using automatic differentiation. Specifically, the likelihood can be automatically differentiated at any Θ, which permits the use of (stochastic) gradient descent, BFGS, or another numerical optimizer, provided we have a good initial guess Θ^0.

Example 14.4.1. Sometimes it is known that F or H has a certain structure. For example, in the case that z_t is a discretization of the ODE describing Hooke's law (see Example 14.1.4) with unknown angular frequency ω, then taking $\mathbf{x}_t = (z_t, z_{t-1})$ gives

$$F = \begin{bmatrix} (2 - \omega^2(\Delta t)^2) & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} a & -1 \\ 1 & 0 \end{bmatrix}$$

for some unknown a. And the observation matrix is completely determined as

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

In this case forcing the optimizer to search only for the missing entry a rather than all the entries of F and H usually enables it to run much faster (the search space is much smaller) and ensures the final answer has the correct form. One way natural way to limit the search to the allowable values is to express the log likelihood (14.27) as a function of a rather than of F and of H and then optimize the resulting function of a.

Initialization

Given the dimensions p and r of X_t and Z_t, respectively, we could choose F^0 and H^0 randomly (say, with each entry of F^0 and H^0 having a standard normal distribution), but whenever possible, the choice should incorporate as much prior knowledge as possible about their structure and values. The covariance matrices Q^0 and R^0 must be positive definite, so they could be chosen by selecting a random square matrix A of the same size and then setting $Q = AA^\mathsf{T}$ (and similarly for Q), but a common choice is just $Q^0 = \sigma^2 I$ and $R^0 = \tau^2 I$ for some $\sigma^2 > 0$ and $\tau^2 > 0$ appropriately reflecting what we know about the variance in the system updates and observation, respectively.

It is natural to take $\boldsymbol{\mu}$ to be the mean of the observations $\mathbf{z}_1, \ldots, \mathbf{z}_m$ and

$$\widehat{\mathbf{x}}_{1|0} = \mathbb{E}(\mathbf{x}_1) = 0 \tag{14.28}$$

The choice of what the initial $P_{1|0}$ should be is harder. One way to approach it is to consider the situation where there are no observations at all, so for a given k and covariance matrix P_{k-1}, the update is simply

$$P_k = P_{k|k-1} = FP_{k-1}F^\mathsf{T} + Q.$$

It seems reasonable to choose a $P_{1|0} = P$ that is unchanged by such an update, that is

$$P = FPF^\mathsf{T} + Q. \tag{14.29}$$

This could be thought of as an analogue of the the stationary distribution for Markov chains—the thing we expect P_k to converge to as $k \to \infty$, assuming there are no observations.[44]

One way to construct such a P is to compute

$$\operatorname{vec}(P) = \left[I_{p^2} - (F \otimes F)\right]^{-1} \operatorname{vec}(Q), \tag{14.30}$$

where vec flattens a matrix and $\otimes$ is the Kronecker product. Here p is the dimension of the state space, so F has shape $p \times p$. The vec operator takes a $p \times q$ matrix $A = \begin{bmatrix}\mathbf{a}_{:1} & \cdots & \mathbf{a}_{:q}\end{bmatrix}$ whose jth column is $\mathbf{a}_{:j}$ and returns the pq-dimensional vector

$$\operatorname{vec}(A) = \begin{bmatrix}\mathbf{a}_{:1}\\ \mathbf{a}_{:2}\\ \vdots \\ \mathbf{a}_{:q}\end{bmatrix}.$$

This is also invertible, with vec^{-1} taking a vector of dimension pq to a $p \times q$ matrix.[45] Thus the initial P could be constructed by applying vec^{-1} to (14.30). The Kronecker product of a $p \times q$ matrix A with an $m \times n$ matrix B is the $pm \times qn$ matrix with block form

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1q}B \\ a_{21}B & a_{22}B & \cdots & a_{2q}B \\ \vdots & \vdots & & \vdots \\ a_{p1}B & a_{p2}B & \cdots & a_{pq}B \end{bmatrix}$$

for

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ a_{21} & a_{22} & \cdots & a_{2q} \\ \vdots & \vdots & & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pq} \end{bmatrix}.$$

Proposition 14.4.2. *The P defined in* (14.30) *is preserved by updates without observation, that is, it satisfies* (14.29)*.*

Proof. The proof relies on the following relation involving $\otimes$ and vec:

$$(F \otimes F)\operatorname{vec}(P) = \operatorname{vec}(FPF^\mathsf{T}), \tag{14.31}$$

for square matrices P and F.

[44] In the case that $\|F\| < 1$, it can be shown that for any initial P_0 and any Q, the sequence $\{P_k\}_{k\in\mathbb{N}}$ converges to such a P as $k \to \infty$.

[45] In Numpy the operator `ravel` flattens row by row instead of column by column, so the operator vec corresponds to `ravel` of the transpose. The `reshape` operator undoes the ravel operator, also row by row. Thus $\operatorname{vec}(A) =$ `ravel(A.T)` and $\operatorname{vec}^{-1}(\mathbf{v}) = (v.reshape(q,p).T)$.

Multiplying (14.30) by $\left[I_{p^2} - (F \otimes F)\right]$ on the left gives

$$\operatorname{vec}(P) - (F \otimes F)\operatorname{vec}(P) = \operatorname{vec}(Q)$$

Applying (14.31) gives

$$\operatorname{vec}(P) - \operatorname{vec}(FPF^{\mathsf{T}}) = \operatorname{vec}(Q).$$

Unflattening this relation (reshaping into matrices) gives

$$P - FPF^{\mathsf{T}} = Q,$$

from which (14.29) follows. □

Of course, one should use whatever domain knowledge is available to choose the initial values Θ^0. For example, in the case of Hooke's law with unknown angular frequency (see Example 14.4.1), the the observation matrix H is completely known, and the state transition matrix F is known except for the one value of a. Moreover, one might have a good initial guess from looking at the data of what that frequency is close to.

Dimensions

Although the dimension of the observation space is known (we observe it), in many cases we know very little about the hidden states and even the dimension of the state space may not be known. It might seem like the best way to estimate that dimension is just to do a grid search through various choices of p and maximize the likelihood for each choice p of dimension. Unfortunately this will often overfit because, just as with OLS, the maximum likelihood always increases as the dimension of the state space increases. To discourage overfitting instead choose the dimension to minimize the (adjusted) AIC, for example

$$2k\left(1 + \frac{k+1}{n-k}\right) - 2\ell(\Theta)$$

where n is the sample size, k is the number of parameters in the model, and $\ell(\Theta)$ is the maximum log-likelihood for the model class.

14.4.3 Expectation Maximization

Instead of gradient-based optimizers by differentiating the log likelihood $\ell(\Theta)$, one can also use expectation maximization. The expectation step involves using a tool called the *Kalman smoother* to compute $P(\mathbf{x}_t \mid \mathbf{z}_1, \ldots, \mathbf{z}_m, \Theta^{old})$ for each $t \in \{0, \ldots m\}$, expressed as a mean $\widehat{\mathbf{x}}^{old}_{t|m}$ and covariance $P^{old}_{t|m}$. Then the maximization step involves computing the maximizer

$$\begin{aligned}
\Theta^{new} &= \operatorname*{argmax}_{\Theta} \log(P(\widehat{\mathbf{x}}^{old}_{1|m}, \ldots, \widehat{\mathbf{x}}^{old}_{m|m}, \mathbf{z}_1, \ldots, \mathbf{z}_m \,|\, \Theta)) \\
&= \operatorname*{argmax}_{\Theta} \sum_{t=1}^{m} \log(P(\mathbf{z}_t \,|\, \widehat{\mathbf{x}}^{old}_{t|m} \Theta))
\end{aligned}$$

The EM algorithm guarantees that the resulting sequence of Θs produces a bounded nondecreasing sequence of log likelihoods, which must, therefore converge to a stationary point of the log likelihood. The computation of the Kalman smoother is similar to that of the Kalman filter, but we won't give the details here.

The EM algorithm applied to state-space models typically moves rapidly in the early stages, but is known to slow down significantly as it approaches the maximizer [**?**, pg. 183]. In cases where fast convergence is important, one can begin with EM and switch to BFGS or gradient descent once the EM algorithm slows down.

14.4.4 Forecasting State-Space Models

Predicting future values of state-space models is straightforward, once the parameters Θ are known. Given observations $\mathbf{z}_1, \ldots, \mathbf{z}_t$, the distribution of a future observation $\mathbf{z}_{t+k}$ is given by

$$\mathbf{z}_{t+k} \,\big|\, \mathbf{z}_1, \ldots, \mathbf{z}_t \sim \mathcal{N}(\mathbf{z}_{t+k} \,\big|\, H\widehat{\mathbf{x}}_{t+k|t} + \mu, HP_{t+k|t}H^\mathsf{T} + R), \tag{14.32}$$

with mean $H\widehat{\mathbf{x}}_{t+k|t} + \mu$ and covariance $HP_{t+k|t}H^\mathsf{T} + R$. The future unobserved states can be computed recursively via the relations

$$\widehat{\mathbf{x}}_{t+k|t} = F\widehat{\mathbf{x}}_{t+k-1|t} \tag{14.33}$$

$$P_{t+k|t} = FP_{t+k-1|t}F^\mathsf{T} + Q. \tag{14.34}$$

Of course, we need the values of $\widehat{\mathbf{x}}_{t|t}$ and $P_{t|t}$ to get this started. These can be found using the usual predict and update recursions from the Kalman filter.

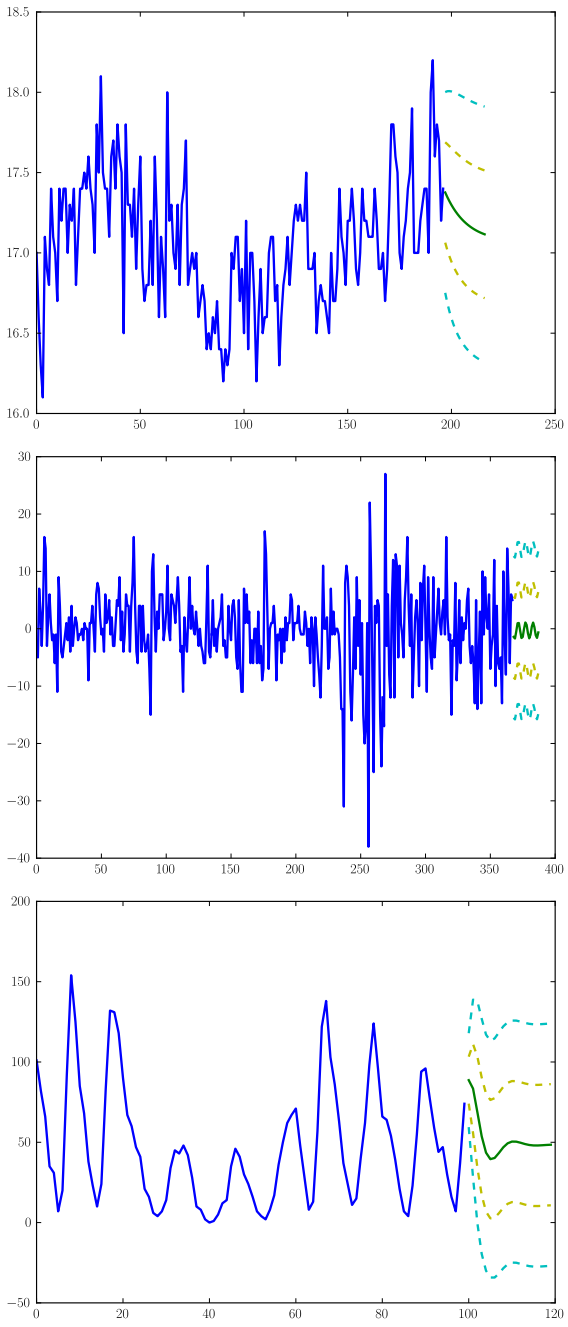


Figure 14.9: Three time series (blue) along with expected future values (green), $\pm\sigma_a$ confidence interval (yellow), and $\pm 2\sigma_a$ confidence interval (cyan) of twenty forecast values. In each case the parameters Θ were estimated from the blue data and then used to forcast future states and observations.

14.5 *Kalman Filter Gives the MVLUE

We consider the filtering problem in the case that $m = k$, that is, when the number of observations equals the number of inputs. It is of particular interest in applications to estimate the current state $\mathbf{x}_k$ given the observations $\mathbf{z}_1, \dots, \mathbf{z}_k$ and inputs $\mathbf{u}_1, \dots, \mathbf{u}_k$. The Kalman filter gives a recursive algorithm to efficiently compute the minimum-variance linear unbiased estimate $\widehat{\mathbf{x}}_{k|k}$ of $\mathbf{x}_k$ (and its covariance $P_{k|k} = \mathbb{E}[(\mathbf{x}_k - \widehat{\mathbf{x}}_{k|k})(\mathbf{x}_k - \widehat{\mathbf{x}}_{k|k})^\mathsf{T}]$) in terms of only the previous estimate $\widehat{\mathbf{x}}_{k-1|k-1}$ (and $P_{k-1|k-1}$) and the latest data $\mathbf{u}_k$ and $\mathbf{z}_k$. The Kalman filter has the advantage that once $\widehat{\mathbf{x}}_{k|k}$ and $P_{k|k}$ are computed, we no longer need $\mathbf{x}_{k-1}$,$\mathbf{z}_k$, or $\mathbf{u}_k$ to compute future estimates.

The Kalman filter is initialized with $\widehat{\mathbf{x}}_0 = \boldsymbol{\mu}_0$ and $P_0 = Q_{-1}$. It then proceeds in two steps. The first step, called the *predictive step*, determines a prediction $\widehat{\mathbf{x}}_{k|k-1}$ for $\mathbf{x}_k$ and covariance matrix $P_{k|k-1}$, and it does this using only the previous estimate $\widehat{\mathbf{x}}_{k-1|k-1}$, $P_{k-1|k-1}$ and the input $\mathbf{u}_k$.

The second step, called the *update* step, uses the output (or observation) $\mathbf{z}_k$ to correct the prediction-covariance pair $(\widehat{\mathbf{x}}_{k|k-1}, P_{k|k-1})$ from the first step, giving a better estimate-covariance pair $(\widehat{\mathbf{x}}_{k|k}, P_{k|k})$.

Predictive Step

For notational convenience let $\widehat{\mathbf{x}}_k = \widehat{\mathbf{x}}_{k|k}$, $\widehat{\mathbf{y}}_k = \widehat{\mathbf{y}}_{k|k}$, $P_k = P_{k|k}$ and $\widehat{J}_k = \widehat{J}_{k|k}$. Also define $\mathscr{F}_k = \begin{bmatrix} 0 & \cdots & 0 & F_k \end{bmatrix} \in \mathbb{R}^{n \times kn}$, with the entry F lying in the block corresponding to $\mathbf{x}_{k-1}$ so that $\mathscr{F}_k \mathbf{y}_{k-1} = F\mathbf{x}_{k-1}$.

Using this notation, we may write $J_{k|k-1}$ recursively as

$$J_{k|k-1}(\mathbf{y}_k) = J_{k-1}(\mathbf{y}_{k-1}) + \frac{1}{2}\|\mathbf{x}_k - \mathscr{F}_k \mathbf{y}_{k-1} - G\mathbf{u}_k\|^2_{Q_k^{-1}}. \tag{14.35}$$

The gradient and Hessian are given by

$$DJ_{k|k-1}(\mathbf{y}_k) = \begin{bmatrix} DJ_{k-1}(\mathbf{y}_{k-1}) + \mathscr{F}_k^\mathsf{T} Q_k^{-1}(\mathscr{F}_k \mathbf{y}_{k-1} - \mathbf{x}_k + G\mathbf{u}_k) \\ -Q_k^{-1}(\mathscr{F}_k \mathbf{y}_{k-1} - \mathbf{x}_k + G\mathbf{u}_k) \end{bmatrix}$$

and

$$D^2 J_{k|k-1}(\mathbf{y}_k) = \begin{bmatrix} D^2 J_{k-1}(\mathbf{y}_{k-1}) + \mathscr{F}_k Q_k^{-1} \mathscr{F}_k & -\mathscr{F}_k^\mathsf{T} Q_k^{-1} \\ -Q_k^{-1}\mathscr{F}_k & Q_k^{-1} \end{bmatrix}, \tag{14.36}$$

respectively.

Lemma 14.5.1. *Let $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times n}$, $C \in \mathbb{R}^{m\times m}$, and $D \in \mathbb{R}^{m\times m}$, with $A, C > 0$ and $D \geq 0$. Then*

$$\begin{bmatrix} A + B^\mathsf{T} C B & -B^\mathsf{T} C \\ -CB & C + D \end{bmatrix} > 0. \tag{14.37}$$

Proof. Note that

$$\begin{bmatrix} x^\mathsf{T} & y^\mathsf{T} \end{bmatrix} \begin{bmatrix} A + B^\mathsf{T} C B & -B^\mathsf{T} C \\ -CB & C + D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x^\mathsf{T} A x + y^\mathsf{T} D y + \|Bx - y\|^2_C \geq 0,$$

with equality only if each term on the right is zero. Since $A > 0$, then we get equality only if $x = 0$, which also implies that $y = 0$. Thus (14.5.1) is positive definite. ∎

This lemma immediately gives the following corollary.

Corollary 14.5.2. *The Hessian $D^2 J_{k|k-1}$, as given in* (14.36)*, is positive definite.*

Since $D^2 J_{k|k-1}$ is positive definite, a single iteration of Newton's method yields the minimizer

$$\widehat{\mathbf{y}}_{k|k-1} = \mathbf{y} - D^2 J_{k|k-1}(\mathbf{y})^{-1} D J_k(\mathbf{y}) \tag{14.38}$$

for any $\mathbf{y} \in \mathbb{R}^{(k+1)n}$. Now, $DJ_{k-1}(\widehat{\mathbf{y}}_{k-1}) = 0$ and $\mathscr{F}_k \widehat{\mathbf{y}}_{k-1} = F\widehat{\mathbf{x}}_{k-1}$, so a judicious initial guess is

$$\mathbf{y} = \begin{bmatrix} \widehat{\mathbf{y}}_{k-1} \\ F\widehat{\mathbf{x}}_{k-1} + G\mathbf{u}_k \end{bmatrix}.$$

With this starting point, the gradient reduces to $DJ_{k|k-1}(\mathbf{y}) = 0$. That is, the optimal estimate of $\mathbf{x}_k$, given the measurements $\mathbf{z}_1, \ldots, \mathbf{z}_{k-1}$, and inputs $\mathbf{u}_1, \ldots, \mathbf{u}_{k-1}$ is the bottom row of $\widehat{\mathbf{y}}_{k|k-1}$, and the covariance, $P_{k|k-1}$ is the bottom right block of the inverse Hessian $D^2 J^{-1}_{k|k-1}$, which by Schur's Lemma (Lemma 14.5.3, below) is

$$\widehat{\mathbf{x}}_{k|k-1} = F\widehat{\mathbf{x}}_{k-1} + G\mathbf{u}_k. \tag{14.39a}$$

$$P_{k|k-1} = FP_{k-1}F^{\mathsf{T}} + Q_k, \tag{14.39b}$$

Lemma 14.5.3 (Schur). *Let M be a square matrix with block form*

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}.$$

If A, D, $A - BD^{-1}C$, and $D - CA^{-1}B$ are nonsingular, then the inverse can be written in the following two ways.

$$M^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix} \tag{14.40}$$

and

$$M^{-1} = \begin{bmatrix} (A - BD^{-1}C)^{-1} & -(A - BD^{-1}C)^{-1}BD^{-1} \\ -D^{-1}C(A - BD^{-1}C)^{-1} & D^{-1} + D^{-1}C(A - BD^{-1}C)^{-1}BD^{-1} \end{bmatrix}. \tag{14.41}$$

The matrices $A - BD^{-1}C$ and $D - CA^{-1}B$ are the Schur complements *of A and D, respectively.*

Proof. These identities can be verified by inspection; see Exercise 14.**??**. ∎

Update Step

In this step we use the observation $\mathbf{z}_k$ to update (or correct) the pair $\widehat{\mathbf{x}}_{k|k-1}, P_{k|k-1}$.

In analogy to $\mathscr{F}_k$, we introduce the notation $\mathscr{H}_k = \begin{bmatrix} 0 & \cdots & H \end{bmatrix} \in \mathbb{R}^{q \times n(k+1)}$, with the entry H lying in the block corresponding to $\mathbf{x}_k$ so that $\mathscr{H}_k \mathbf{y}_k = H\mathbf{x}_k$. With this notation, the objective function can be written as

$$J_k(\mathbf{y}_k) = J_{k|k-1}(\mathbf{y}_k) + \frac{1}{2}\|\mathbf{z}_k - \mathscr{H}_k \mathbf{y}_k\|^2_{R_k^{-1}} \tag{14.42}$$

and the gradient and Hessian are, respectively,

$$\begin{aligned} DJ_k(\mathbf{y}_k) &= DJ_{k|k-1}(\mathbf{y}_k) + \mathscr{H}_k^{\mathsf{T}} R_k^{-1}(\mathscr{H}_k \mathbf{y}_k - \mathbf{z}_k) \\ &= DJ_{k|k-1}(\mathbf{y}_k) + \mathscr{H}_k^{\mathsf{T}} R_k^{-1}(H\mathbf{x}_k - \mathbf{z}_k) \end{aligned}$$

and

$$D^2 J_k(\mathbf{y}_k) = D^2 J_{k|k-1} + \mathscr{H}_k^{\mathsf{T}} R_k^{-1} \mathscr{H}_k.$$

The Hessian is clearly positive definite. Again, a single iteration of Newton's method yields the minimzer. If we choose the initial guess $\mathbf{y} = \widehat{\mathbf{y}}_{k|k-1}$ then, since $DJ_{k|k-1}(\widehat{\mathbf{y}}_{k|k-1}) = 0$, the gradient becomes

$$DJ_k(\widehat{\mathbf{y}}_{k|k-1}) = \mathscr{H}_K^{\mathsf{T}} R_k^{-1}(H\widehat{\mathbf{x}}_{k|k-1} - \mathbf{z}_k).$$

The estimate $\widehat{\mathbf{x}}_k$ of $\mathbf{x}_k$, together with its covariance P_k are then obtained from the bottom row of the Newton update and the bottom right block of the covariance matrix $D^2 J_k(\mathbf{y}_k)^{-1}$. Again, using Lemma 14.5.3, this is

$$P_k = (P_{k|k-1}^{-1} + H^{\mathsf{T}} R_k^{-1} H)^{-1}, \tag{14.43a}$$

$$\widehat{\mathbf{x}}_k = \widehat{\mathbf{x}}_{k|k-1} - P_k H^{\mathsf{T}} R_k^{-1}(H\widehat{\mathbf{x}}_{k|k-1} - \mathbf{z}_k). \tag{14.43b}$$

The upshot of all our work in this section is that, given a system of the form (14.4), an inital approximate value $\boldsymbol{\mu}_0$, and observations $\mathbf{z}_0, \ldots, \mathbf{z}_m$, the Equations (14.6) and (14.7), give a two-step recursive method for finding the minimum-variance unbiased linear estimate $\widehat{\mathbf{x}}_t$ for the state $\mathbf{x}_t$ at each time $t = 1, 2, \ldots, m$ and for computing the covariance P_t of each estimator. If the model is well designed (a good fit), and the observations are not too noisy, then the norm $\|P_t\|$ of the covariance should converge toward 0 as t grows.

Remark 14.5.4. This is not the derivation that Kalman gave. His approach assumed that the noise is normally distributed, and he showed that the Kalman filter gives the optimal (minimum-variance unbiased) estimate by tracking the evolution of the mean and covariance of the distribution of $\mathbf{x}_k$ under iteration of the dynamical system. Our approach is more general and shows that the result of the Kalman filter is the optimal estimate even when the noise is not normally distributed.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second line are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

14.1. Consider a particle moving in a straight line with initial velocity v_0 and varying acceleration a_k at each discrete point t_k in time.

(i) Construct a discrete-time linear state-space model with a two-dimensional state space to describe the motion of the particle. What are states and the matrices F and G?

(ii) If only the position of the particle is observed in this model, what are the observation matrices H?

(iii) Assume that the acceleration can be noisy, so the true acceleration at time t_k is $a_k + w_k$ for $w_k \sim \mathcal{N}(0, q)$. Construct a discrete-time linear state space model with a three-dimensional state space to describe the motion of the particle. What are states and the transition matrices F and G? Hint: Beware that the acceleration at each time is independent of the previous acceleration and only depends on the new value of a_k and the noise w_k.

(iv) If only the position of the particle is observed in the three-dimensional model, what are the observation matrices H?

14.2. Consider a general state space model with hidden states as described by (14.4), and let Θ denote the collection $(F, G, H, Q_k, R_k)_k$ of all the parameters for all k. Let $(\mathbf{x}_k)_{k=0}^T$ be a sequence of states, $(\mathbf{u}_k)_{k=0}^{T-1}$ be a sequence of controls, and $(\mathbf{z}_k)_{k=0}^T$ be a sequence of observations. Assume that the noise terms satisfy $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, Q_k)$ and $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, R_k)$.

(i) What is the joint p.d.f. of $(\mathbf{z}_k)_{k=0}^T$ given $(\mathbf{x}_k)_{k=0}^T, \Theta$?

(ii) Write the p.d.f. of $(\mathbf{z}_k)_{k=0}^T$ given Θ in terms of an integral over all sequences $(\mathbf{x}_k)_{k=0}^T$.

14.3. Install FilterPy on your computer (or in CoLab) with `pip install filterpy`

(i) For each of the state space examples 14.1.1, 14.1.3, and 14.1.4, do the following:

(a) Choose an initial state value $\mathbf{x}_0$ and sample from the system (compute both the states `x` and the observations `z`) for at least $n = 100$ steps. Make sure that your states have shape `(n,dim_x)` and your observations have shape `(n,dim_z)`.

(b) Instantiate a Kalman filter for the system, using FilterPy. The following is an example of how you might do this for the 2-dimensional water-tank filling system:

```
from filterpy.kalman import KalmanFilter

n = 100               # Number of samples
c = 0.1               # Initial inflow rate
x0hat = np.array([c,7.]) # Initial guess of the state
u = np.zeros((n,2))   # No control
P = np.eye(2)         # Initial estimator covariance

# instantiate the filter
kf = KalmanFilter(dim_x=2,dim_z=1)

#define the system
kf.F = np.array([[1,0],[1,1]]) # State transition
kf.G = np.eye(2)          # Control matrix
kf.Q = 0.001*np.eye(2)    # Covar of state transition
kf.H = np.array([0.,1.]) # Observation matrix
kf.R = 0.1*np.eye(1)      # Covar of observation

# Set the starting values
kf.x = x0hat.reshape(-1,1)  # shape matters
kf.P = P
```

(c) Using the following helper code, run the Kalman filter using your observations `z` to compute estimates `xhat` of the sequence of states.

```
def filter(f,zs):      # filter using each observation
    n = np.size(zs,0)

    # initialize
    xhat = np.zeros((n,f.dim_x,1))
    Ps = np.zeros((n,f.dim_x,f.dim_x))

    for i,z in enumerate(zs):
        f.predict()
        f.update(z)
        xhat[i] = f.x.copy()
        Ps[i] = f.P.copy()

    return xhat, Ps
```

Warning: the `KalmanFilter` class remembers the current estimate (`self.x`) and covariance (`self.P`). So, if you want to start again at `x0`, then you need to reset `kf.x` and `kf.P` to their initial values.

(d) Plot the actual states (or the most natural one-dimensional part of the states), the observations, and the estimates `xhat`, all together on one plot, and compare them.

(e) Compute the norms of the covariances `Ps` and plot them. Do you see any connection between the size of the norm and how well `xhat` tracks the observations versus the true state values?

(ii) For the last system (Example 14.1.4) choose two significantly different values for the initial guess `x0hat`, three significantly different values for `P`, and three different perturbations of `F`, and for each of the 18 resulting combinations, run the Kalman filter to compute estimates `xhat` and `Ps` Plot the results `xhat`, the true states, and the observations all together. (So you should have 18 plots). What conclusions do you draw from all these plots?

(iii) Plot the 18 different covariances from the previous step, identify any trends, and compare the size of the covariance matrix to the how well `xhat` tracks the observations and the true state values.

14.4. Given a variable $\mathbf{x} \in \mathbb{R}^n$ and constants $\boldsymbol{\mu} \in \mathbb{R}^n$ and $\Sigma \in M_n$, prove that

$$\exp(-\frac{1}{2}(\mathbf{x}^\mathsf{T}\Sigma^{-1}\mathbf{x} - 2\mathbf{x}^\mathsf{T}\Sigma^{-1}\boldsymbol{\mu})) \propto \mathscr{N}(\mathbf{x}\,|\,\boldsymbol{\mu}, \Sigma).$$

Use this fact to prove that If $f_X(\mathbf{x})$ is the p.d.f. of $\mathscr{N}(\mathbf{x}\,|\,\mathbf{a}, A)$ and $f_Y(\mathbf{y})$ is the p.d.f. of $\mathscr{N}(\mathbf{y}\,|\,\mathbf{b}, B)$, then the product $f_X(\mathbf{z})f_Y(\mathbf{z})$ is proportional to the following normal p.d.f.:

$$f_X(\mathbf{z})f_Y(\mathbf{z}) \propto \exp\left(-\frac{1}{2}\left((\mathbf{z}-\boldsymbol{\mu})^\mathsf{T}\Sigma^{-1}(\mathbf{z}-\boldsymbol{\mu})\right)\right) \propto \mathscr{N}(\mathbf{z}\,|\,\boldsymbol{\mu}, \Sigma) \qquad (14.44)$$

where

$$\Sigma = (A^{-1} + B^{-1})^{-1}$$
$$\boldsymbol{\mu} = \Sigma(A^{-1}\mathbf{a} + B^{-1}\mathbf{b})$$

Note: The product XY of Gaussian random variables X and Y is **NOT a Gaussian random variable**, but the product $f_X(\mathbf{z})f_Y(\mathbf{z})$ of their p.d.f.s is proportional to the p.d.f. of a normal distribution, as this exercise shows. The product of two normal p.d.f.s often shows up in Bayesian computations (like the one we did for the Kalman filter).

14.5. Determine the (leading order) temporal complexity of computing the naïve Kalman update steps (14.7) as well as of the alternative update steps (14.14). Assume that matrix inversion costs $\sim 2n^3$ FLOPs, that matrix multiplication of an $m \times n$ matrix and a $n \times p$ matrix costs $\sim 2mnp$ FLOPs, that LU decomposition costs $\sim 2/3n^3$ FLOPS, and that backsolving $LU\mathbf{x} = \mathbf{b}$ for a single column vector $\mathbf{b} \in \mathbb{R}^n$ (and LU is $n \times n$) costs $\sim 2n^2$ FLOPs. Hint: Remember that wherever possible it is faster ($\sim (2/3)n^3$ vs $\sim 2n^3$) and more accurate to solve a linear system than it is to invert a matrix.

14.6. Prove Lemma 14.3.2. Hint: To prove that $M^{-1} = N$, it suffices to show that $MN = I$, and to show that $PQ^{-1} = R$ it suffices to show that $P = RQ$.

14.7. Prove that Equations (14.7) and (14.14) are equivalent.

14.8. Equation (14.26) and (14.27) give formulas for the log likelihood of Θ given one observation sequence $(\mathbf{z}_t)_{t=1}^m$. Suppose that we have multiple independent observation sequences $(\mathbf{z}_t^1)_{t=1}^m, (\mathbf{z}_t^2)_{t=1}^m, \ldots (\mathbf{z}_t^n)_{t=1}^m)$ from the same state-space model with unknown parameters Θ. Give the analogue of both (14.26) and (14.27) for the log likelihood of Θ given all n of the observation sequences.

14.9. Prove the relation (14.31). Hint: Write V as columns like $V = \begin{bmatrix}\mathbf{v}_{:1} & \ldots & \mathbf{v}_{:p}\end{bmatrix}$ and write F^T as rows (corresponding to columns of F).

14.10. Prove that if $\|F\| < 1$, then for any initial P_0 and any Q, the sequence defined by $P_k = FP_{k-1}F^\mathsf{T} + Q$ converges as $k \to \infty$, and its limit P satisfies $P = FPF^\mathsf{T} + Q$.

14.11. Assume you are given a sequence of data $\mathbf{z}_1, \ldots, \mathbf{z}_m$ which you believe is the observation sequence of an unknown state-space model.

(i) Describe in detail, in your own words, how you would estimate the parameters $\Theta = (F, H, Q, R, \boldsymbol{\mu}, \mathbf{x}_{1|0}, P_{1|0})$ and the dimension of the hidden state space.

(ii) Explain how, once you had the parameters in Θ, you could use them to forecast future hidden states and future observations $\mathbf{z}_{m+1}, \ldots, \mathbf{z}_{m+n}$.

(iii) How could you estimate a confidence interval around your forecasts?

(iv) What assumptions would need to hold for the parameter estimation and the forecasts to be valid?

Notes

For historical notes on the Kalman Filter, see [**?**, GA15].

The Kalman filter is used widely in virtually every technical or quantitative field. In engineering, for example, the Kalman filter is pervasive in the areas of navigation and global positioning [**?**, **?**], tracking [**?**], guidance [**?**], robotics [**?**], radar [**?**], fault detection [**?**] and computer vision [**?**]. It is also utilized in applications involving signal processing [**?**], voice recognition [**?**], video stabilization [**?**], and automotive control systems [**?**]. In purely quantitative fields, the Kalman filter also plays an important role in time-series analysis [**?**], econometrics [**?**], mathematical finance [**?**], system identification [**?**], and neural networks [**?**].

15 Time Series

Prediction is very difficult, especially about the future.
—Neils Bohr

15.1 Introduction to Time Series Models

A *time series* is any stochastic process indexed by time; that is, it is a sequence of random variables $(Z_t)_{t \in T}$, where $t \in T \subset \mathbb{R}$. We focus here on the special case of discrete time ($T \subset \mathbb{Z}$) where T is an interval in $\mathbb{Z}$. The random variables Z_t need not be real numbers or even numerical; they could be elements of any set, for example musical chords, or names of people in a class. The musical notes in a tune, the value of the Dow Jones stock market index each minute, the maximum temperature in your home town each day, your blood glucose level each second, and the number of widgets sold on your webpage each hour are all examples of such time series. One of the most common ways to model time series is as a hidden Markov model (HMM); see Section 13.1.3. Hidden Markov models are discussed in much more detail later in the text.

For the rest of this section we focus on time series that take values in $\mathbb{R}^n$ for some $n > 0$, but later when we discuss HMMs in more detail we'll be able to say many useful things about time series that do not necessarily take their values in $\mathbb{R}^n$. Of course it is usually easy to create an embedding of the support of Z_t into $\mathbb{R}^n$, for example, we could define a map $\varphi : \{\text{musical note}\} \to \mathbb{N}$ taking musical notes to their position on a scale by $C \mapsto 0$, $C^\sharp \mapsto 1$, $D \mapsto 2$, and so on. But for many such maps it does not make sense to use all of the standard arithmetic operations on the resulting real number: for example adding the notes $C + D$ does not really makes sense, even though it is possible to add the mapped values $\varphi(C) + \varphi(D) = 0 + 2 = 2 = \varphi(D)$. Thus you should always be careful when using the methods of this section that the arithmetic operations you are performing are meaningful and valid.

15.1.1 Moments

We denote a specific draw, or realization, of the time series $(Z_t)_{t=0}^N$ by $(\mathbf{z}_t)_{t=0}^N$. Assuming that Z_t takes its values in $\mathbb{R}^n$ for some $n > 0$ and that the following arithmetic operations are meaningful for the time series,the first and second moments of the series give valuable information. They are not necessarily constant, but rather are functions of t.

Definition 15.1.1. *The* mean *of the series at time t is*

$$\boldsymbol{\mu}(t) = \mathbb{E}[Z_t].$$

The autocovariance of order k *of the series at time t is the covariance of Z_t and Z_{t-k}*

$$\boldsymbol{\gamma}_k(t) = \mathrm{Cov}(Z_t, Z_{t-k})$$

In the case that $Z_t \in \mathbb{R}$ for all t, then $\gamma_k(t) \in \mathbb{R}$ (we won't write it bold), and we define the autocorrelation of order k *of the series at time t as*

$$\rho_k(t) = \frac{\gamma_k(t)}{\sqrt{\gamma_0(t)\gamma_0(t-k)}}.$$

If $\boldsymbol{\mu}$ and $\boldsymbol{\gamma}_k$ are independent of t for all $k \in \mathbb{N}$, and if $\mathbb{E}[\|Z_t\|_2^2]$ is finite, then the time series is called covariance stationary.[46]

Example 15.1.2. If the Z_t are all i.i.d.with finite covariance Σ, then the time series is covariance stationary. Moreover, we have $\boldsymbol{\gamma}_0(t) = \Sigma$, and $\boldsymbol{\gamma}_k(t) = \mathbf{0}$ for all $k > 0$.

If $(Z_t)_{t=0}^\infty$ is a univariate covariance-stationary time series, then $\rho_j = \gamma_j/\gamma_0$ for all $j \in \mathbb{N}$.

Proposition 15.1.3. *If Z_t is a covariance-stationary time series, then $\|\boldsymbol{\mu}\|$ and the trace $\mathrm{tr}(\boldsymbol{\gamma}_k)$ are finite for all $k \in \mathbb{N}$.*

Proof. Jensen's inequality gives

$$\|\boldsymbol{\mu}\|_2^2 = \|\mathbb{E}[Z_t]\|_2^2 \leq \mathbb{E}[\|Z_t\|_2^2] < \infty.$$

Recall that

$$\mathrm{tr}(\mathbf{v}\mathbf{w}^\mathsf{T}) = \mathrm{tr}(\mathbf{v}^\mathsf{T}\mathbf{w}) = \mathbf{v}^\mathsf{T}\mathbf{w} \tag{15.1}$$

for any vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$, and thus $\mathrm{tr}(\mathbf{v}\mathbf{v}^\mathsf{T}) = \|\mathbf{v}\|_2^2$. A straightforward computation shows that the pairing $\langle X, Y\rangle = \mathrm{tr}(\mathbb{E}[XY^\mathsf{T}])$ is an inner product on the space of mean-zero random variables with support in $\mathbb{R}^n$, and the corresponding norm satisfies

$$\|X\|^2 = \langle X, X\rangle = \mathrm{tr}\,\mathbb{E}[XX^\mathsf{T}] = \mathbb{E}[\|X\|_2^2].$$

[46] Another commonly used name for covariance stationary is *weakly stationary*. The related concept of *strongly stationary* requires that for any finite set $\{t_1, \ldots, t_s\} \subset T$ of times and any $k \in \mathbb{N}$ the joint c.d.f.s should satisfy $F_{Z_{t_1},\ldots,Z_{t_s}} = F_{Z_{t_1+k},\ldots,Z_{t_s+k}}$. This is an important concept, but we do not treat it in this text.

For any $t \in T$ and $k \in \mathbb{N}$, let $X = Z_t - \boldsymbol{\mu}$ and $Y = Z_{t+k} - \boldsymbol{\mu}$. The Cauchy-Schwarz inequality gives

$$\begin{aligned}(\operatorname{tr}(\boldsymbol{\gamma}_k))^2 &= (\operatorname{tr}(\mathbb{E}[XY^{\mathsf{T}}]))^2 \\ &= \langle X, Y\rangle^2 \\ &\leq \|X\|^2\|Y\|^2 \\ &= \mathbb{E}[\operatorname{tr}(XX^{\mathsf{T}})]\mathbb{E}[\operatorname{tr}(YY^{\mathsf{T}})] \\ &= (\operatorname{tr}(\boldsymbol{\gamma}_0))^2 \\ &= (\mathbb{E}[\operatorname{tr}(Z_t Z_t^{\mathsf{T}})] - \operatorname{tr}(\boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}}))^2 \\ &= (\mathbb{E}[\|Z_t\|_2^2] - \|\boldsymbol{\mu}\|_2^2)^2 \\ &< \infty. \quad \square\end{aligned}$$

Proposition 15.1.4. *Let Z_t be a covariance-stationary time series taking values in $\mathbb{R}^m$. If $f : \mathbb{R}^m \to \mathbb{R}^n$ is an affine function of the form $f(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$ for constant A and $\mathbf{b}$, then $\operatorname{tr}\mathbb{E}[(f(Z_t))(f(Z_t))^{\mathsf{T}}] < \infty$ and $Y_t = f(Z_t)$ is a covariance-stationary time series.*

Proof. The proof is Exercise 15.2 $\square$

Unexample 15.1.5. Of course many time series are not covariance stationary. An important example is the *random walk*, which is of the form

$$Z_0 = \mathbf{0} \qquad \text{and} \qquad Z_t = Z_{t-1} + \boldsymbol{\varepsilon}_t, \quad \forall t \in \mathbb{Z}^+,$$

where the $\boldsymbol{\varepsilon}_t$ are i.i.d. with mean $\mathbf{0}$ and constant covariance Σ. The mean $\boldsymbol{\mu}(t) = \mathbf{0}$ is constant, but

$$\begin{aligned}\boldsymbol{\gamma}_0(0) &= \mathbb{E}[Z_0 Z_0^{\mathsf{T}}] = 0 \\ \boldsymbol{\gamma}_0(1) &= \mathbb{E}[Z_1 Z_1^{\mathsf{T}}] = \mathbb{E}[\boldsymbol{\varepsilon}_1\boldsymbol{\varepsilon}_1^{\mathsf{T}}] = \Sigma \\ \boldsymbol{\gamma}_0(2) &= \mathbb{E}[(\boldsymbol{\varepsilon}_1 + \boldsymbol{\varepsilon}_2)(\boldsymbol{\varepsilon}_1 + \boldsymbol{\varepsilon}_2)^{\mathsf{T}}] = 2\Sigma \\ &\vdots \\ \boldsymbol{\gamma}_0(t) &= t\Sigma\end{aligned}$$

is not constant in t.

15.1.2 Classical Decomposition

A time series Z_t is traditionally decomposed into a sum of three parts $Z_t = T_t + S_t + R_t$:

(i) *Trend* $T_t = \boldsymbol{\mu}(t)$: Often this is a linear function of t, but it need not always be. An example of this is given in the second (from the top) panel of Figure 15.1.

(ii) *Seasonal* S_t: A seasonal pattern occurs when a time series is affected by seasonal factors such as the time of the year, time of day, or the day of the week. Seasonality is always of a fixed and known frequency. We require S_t to be a periodic function of t with a fixed period $m > 0$ and $\sum_{i=0}^{m-1} S_i = 0$. An example of this is given in the third panel of Figure 15.1.

(iii) *Remainder* R_t. This is what remains after subtracting out the trend and the seasonal component. It satisfies $\mathbb{E}[R_t] = 0$, and, if we are lucky, it might also be covariance stationary or otherwise reasonable to work with. An example of this is given in the bottom panel of Figure 15.1.

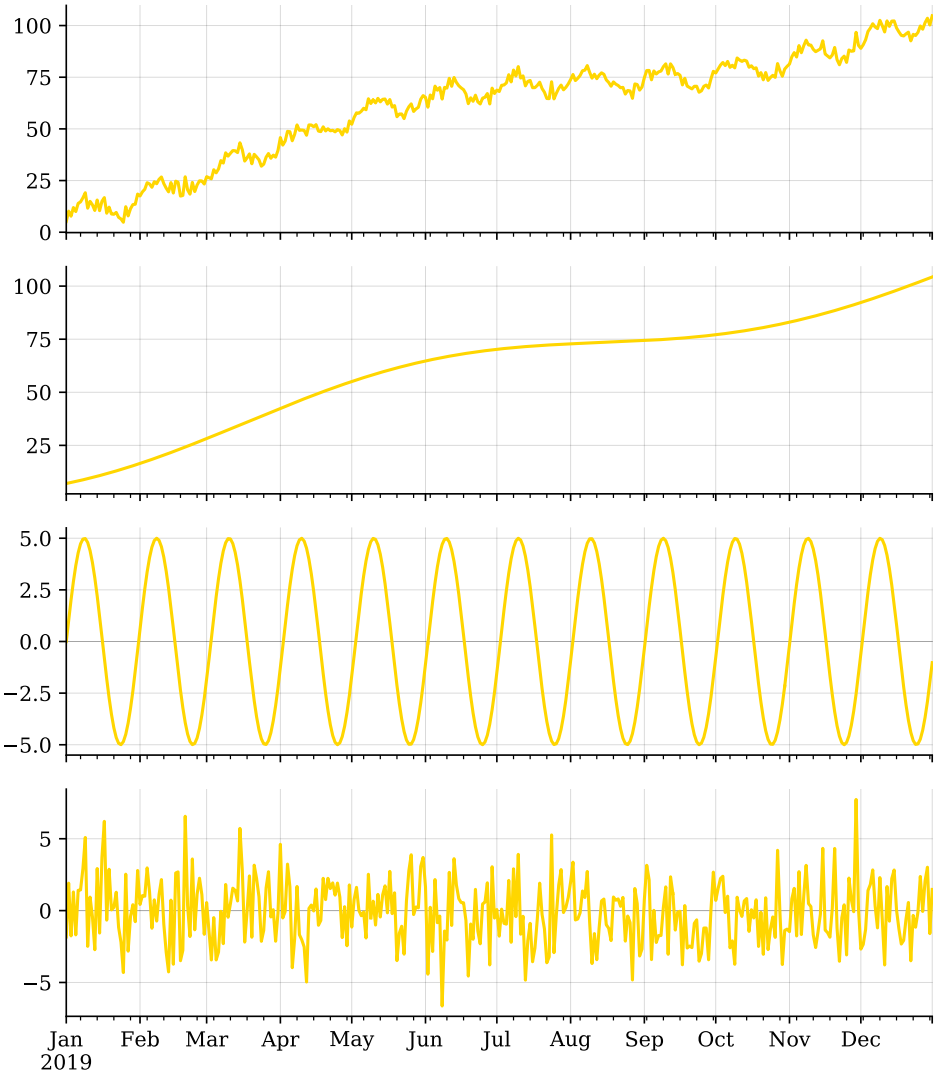


Figure 15.1: As described in Subsection 15.1.2, classical decomposition of a time series (top panel) decomposes the time series into the sum of a trend T_t (second panel), which is often approximately linear; a seasonal component S_t (third panel), which has a fixed period (in this case monthly); and the rest R_t (bottom panel), which we hope is covariance stationary or otherwise reasonable to work with.
.

For a time series $(Z_t)_{t=0}^N$ where the seasonal component has a known period of length m, a fundamental tool for estimating the classical decomposition is the *(symmetric) moving average of length m.*

Definition 15.1.6. *For an odd integer $m = 2\ell + 1 > 0$ the* length-m moving average *of Z_t at t is*

$$T_t^{(m)} = \frac{1}{m} \sum_{j=-\ell}^{\ell} Z_{t+j} = \frac{1}{m} \left(Z_{t-\ell} + \cdots + Z_{t+\ell} \right).$$

for $\ell \leq t \leq N - \ell$. For an even integer $m = 2\ell$, the length-m symmetric moving average *is*

$$T_t^{(m)} = \frac{1}{m} \left(\frac{1}{2} Z_{t-\ell} + Z_{t-\ell+1} + \cdots + Z_t + \cdots + Z_{t+\ell-1} + \frac{1}{2} Z_{t+\ell} \right).$$

We denote the corresponding estimates by $\widehat{T}_t^{(m)}$. That is, $\widehat{T}_t^{(m)}$ is the realization of the estimator $T_t^{(m)}$ at a specific realization $(\mathbf{z}_t)_{t=0}^N$ of $(Z_t)_{t=0}^N$.

Proposition 15.1.7. *Assume that $Z_t = T_t + S_t + R_t$, where S_t is a periodic function of t with a fixed period of length m, and $\sum_{i=0}^{m-1} S_{t+i} = 0$ for all t. If $T_t = \mathbf{v}t + c$ is an affine function of t, then $\mathbb{E}[T_t^{(m)}] = T_t$.*

Proof. The proof is Exercise 15.3. □

Remark 15.1.8. Both the even and odd moving averages are symmetric around t, but in the case that m is even, the naïve moving average $\frac{1}{m}\left(Z_{t-\ell} + \cdots + Z_{t+\ell}\right)$ has length $m + 1$ and Proposition 15.1.7 does not hold for it; see Exercise 15.3.

Proposition 15.1.7 motivates the following method for estimating the classical decomposition of a time series. Given a realization $(\mathbf{z}_t)_{t=0}^N$ of a time series with classical decomposition $Z_t = T_t + S_t + R_t$, where S_t has period m, we estimate the trend T_t with the length-m (symmetric) moving average $\widehat{T}_t^m$. The difference $\mathbf{y}_t = \mathbf{z}_t - \widehat{T}_t^m$ is called the *detrended series* associated to $\mathbf{z}_t$.

The seasonal component S_t can be estimated by averaging the detrended series $\mathbf{y}_t$ over all values in one period:

$$S_k \approx \frac{1}{\lfloor (N-k)/m \rfloor} \sum_{t \equiv k \bmod m} \mathbf{y}_t \qquad \forall k \in \{0, 1, \ldots, m-1\},$$

and

$$\widehat{S}_t = \widehat{S}_k \qquad \forall\, t \equiv k \bmod m.$$

The estimated remainder is $\widehat{R}_t = \mathbf{z}_t - \widehat{T}_t^m - \widehat{S}_t$.

Remark 15.1.9. Of course the previous method for estimating the classical decomposition only works if m is known and is large enough that the moving average gives a reasonable estimate for the mean $\boldsymbol{\mu}(t)$. In many situations m can be guessed: for example, time series that depend on weather are likely to have a seasonal component with period equal to the length of a year, while time series that depend on human sleep habits should have a seasonal component with period equal to the length of a day.

Remark 15.1.10. It is common to separate the trend into the sum of a linear function (or other simple function) and another component that is often called the *cyclic* component. This is an unfortunate name because the cyclic component is usually not periodic (the periodic part of the decomposition should be incorporated into the seasonal component). The standard example of a cyclic component is the multi-year "business cycle."

A linear trend can be estimated either by fitting a linear function to the data (using OLS), or by computing the forward difference $\Delta\mathbf{z}_t = \mathbf{z}_t - \mathbf{z}_{t-1}$. In particular, if $\mathbf{z}_t = T_t + \mathbf{y}_t$ decomposes into the sum of a mean-zero series $\mathbf{y}_t$ (that is, $\mathbb{E}[\mathbf{y}_t] = \mathbf{0}$ for all t) and a linear trend of the form

$$T_t = t\mathbf{v} + \mathbf{c}$$

for constants $\mathbf{v}, \mathbf{c} \in \mathbb{R}^n$, then

$$\Delta\mathbf{z}_t = \mathbf{v} + \Delta\mathbf{y}_t.$$

Since the mean of $\Delta\mathbf{y}_t$ is zero, we have

$$\mathbf{v} = \mathbb{E}[\Delta\mathbf{z}_t] \approx \frac{1}{N}\sum_{t=0}^{N-1} \Delta\mathbf{z}_t = \widehat{\mathbf{v}}.$$

Similarly, the constant $\mathbf{c}$ can be approximated by

$$\mathbf{c} = \mathbb{E}[\mathbf{c} + \mathbf{y}_t] = \mathbb{E}[\mathbf{z}_t - t\mathbf{v}] \approx \frac{1}{N}\sum_{t=0}^{N-1} (\mathbf{z}_t - t\mathbf{v}) = \widehat{\mathbf{c}}.$$

If the trend is quadratic instead of linear, with $\mathbf{z}_t = Q_t + \mathbf{y}_t$ and $Q_t = \mathbf{c} + t\mathbf{v} + t^2\mathbf{w}$ for fixed $\mathbf{c}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^n$, then the constants can be estimated using the first and second forward differences of $\mathbf{z}_t$.

15.1.3 Multiplicative and other Models

Not all time series decompose additively. If the trend of the time series appears to be exponential, rather than linear or quadratic, then it might be more natural to use a multiplicative version of the classical decomposition.

$$Z_t = T_t \cdot S_t \cdot R_t.$$

Another hint that a multiplicative model might be better than an additive one is if the amplitude of the seasonality tends to grow or shrink over time. Economic time series are often naturally multiplicative.

A multiplicative decomposition is easily converted into an additive model by taking the logarithm

$$\log(Z_t) = \log(T_t) + \log(S_t) + \log(R_t)$$

In some circumstances other transformations of the data may also be useful. The square root transformation $\sqrt{Z_t}$ and the *Box-Cox* transformation

$$\frac{Z_t^\lambda - 1}{\lambda}$$

for a fixed $\lambda \neq 0$ are both popular choices.

Nota Bene 15.1.11. Beware that these transformations fundamentally change the associated distributions, so if your forecasts or analyses rely on R_t to be distributed in a certain way (normal for example), then applying one of these transforms could invalidate your analysis.

15.2 ARIMA Models

Another commonly-used model for time series is the *autoregressive integrated moving-averages (ARIMA)* model. This is a combination of three separate models: *autoregressive (AR)* models, *moving averages (MA)* models, and *integrated (I) models*.

15.2.1 AR Models

In an *autoregressive* time series the next value is a linear combination of several previous values and an error term. This linear combination is called an *autoregression*. If there is a strong correlation between current and recent previous values of a time series, for example with the temperature in a given location, then an autoregressive model might be appropriate.

Definition 15.2.1. *An* AR(p) *model, or* autoregressive model of order p *is a discrete-time stochastic process* $(Z_t)_{t\in\mathbb{Z}}$ *that satisfies*

$$Z_t = \mathbf{c} + \sum_{i=1}^{p} \Phi_i Z_{t-i} + \boldsymbol{\varepsilon}_t, \tag{15.2}$$

where $\mathbf{c} \in \mathbb{R}^n$ *and all* $\Phi_i \in M_n(\mathbb{R})$ *are constant, and the errors* $\boldsymbol{\varepsilon}_t$ *are i.i.d. with mean zero and constant covariance* Σ.

Remark 15.2.2. If $n > 1$ an AR(p) process is often called a *vector autoregressive process* or VAR(p) process.

Example 15.2.3.

(i) An AR(0) model has no dependence on previous terms and is simply a sequence of i.i.d. random variables drawn from a distribution with mean $\mathbf{c}$ and covariance Σ.

(ii) An AR(1) process with $\Phi = 0$ is an AR(0) process.

(iii) An AR(1) process with $\Phi = I$ satisfies

$$Z_t = \mathbf{c} + Z_{t-1} + \boldsymbol{\varepsilon}_t.$$

If $\mathbf{c} = \mathbf{0}$ this is a *random walk*. If $\mathbf{c} \neq \mathbf{0}$ it is a random walk *with drift*. If $(Z_t)_{t=0}^{\infty}$ is a random walk, with or without drift, then the first difference $\Delta Z_t = Z_t - Z_{t-1}$ defines a covariance-stationary process.

Example 15.2.4. Consider a univariate $AR(1)$ process

$$Z_t = c + \varphi Z_{t-1} + \varepsilon_t$$

with constant initial value $Z_0 = a$ and constant variance $\mathrm{Var}(\varepsilon_t) = \sigma^2$ for the error ε_t. The autovariance γ_0 satisfies

$$\begin{aligned}
\gamma_0(0) &= \mathrm{Var}(Z_0) = 0 \\
\gamma_0(1) &= \mathrm{Var}(c + \varphi Z_0 + \varepsilon_1) = \sigma^2 \\
\gamma_0(2) &= \mathrm{Var}(c + \varphi Z_1 + \varepsilon_2) = \varphi^2\sigma^2 + \sigma^2 \\
&\vdots \\
\gamma_0(t) &= \mathrm{Var}(c + \varphi^2 Z_{t-1} + \varepsilon_t) = \sigma^2 \sum_{i=0}^{t-1} \varphi^{2i} \\
&= \sigma^2 \frac{\varphi^{2t} - 1}{\varphi^2 - 1}.
\end{aligned}$$

If $\varphi^2 < 1$, then the autovariance approaches $\sigma^2 \frac{1}{1-\varphi^2}$, but if $\varphi^2 \geq 1$, then $\gamma_0(t) \to \infty$ as $t \to \infty$.

Theorem 15.2.5. *Let $(Z_t)_{t\in\mathbb{Z}}$ be an* AR(1) *process with parameters Φ and $\mathbf{c}$ defined for all $t \in \mathbb{Z}$. If the spectral radius*[47] *$r(\Phi)$ of the linear operator Φ is less than 1 and $\lim_{\ell\to-\infty} \|Z_\ell\| < \infty$, then the series is covariance stationary and*

$$\boldsymbol{\mu}(t) = \mathbb{E}[Z_t] = (I - \Phi)^{-1}\mathbf{c} \tag{15.3}$$

$$\boldsymbol{\gamma}_k(t) = \operatorname{Cov}(Z_t, Z_{t-k}) = \Phi^k \sum_{j=0}^{\infty} \Phi^j \Sigma (\Phi^j)^{\mathsf{T}} \tag{15.4}$$

for all $k \in \mathbb{Z}$.

Proof. We have

$$\begin{aligned}
Z_t &= \mathbf{c} + \Phi Z_{t-1} + \varepsilon_t \\
&= \mathbf{c} + \Phi(\mathbf{c} + \Phi Z_{t-2} + \varepsilon_{t-1}) + \varepsilon_t \\
&= \mathbf{c} + \Phi(\mathbf{c} + \Phi(\mathbf{c} + \Phi Z_{t-3} + \varepsilon_{t-2}) + \varepsilon_{t-1}) + \varepsilon_t \\
&= (\mathbf{c} + \varepsilon_t) + \Phi(\mathbf{c} + \varepsilon_{t-1}) + \Phi^2(\mathbf{c} + \varepsilon_{t-2}) + \cdots + \Phi^k(\mathbf{c} + \Phi Z_{t-k-1} + \varepsilon_{t-k}) \\
&= \sum_{i=0}^{k-1} \Phi^i \mathbf{c} + \sum_{i=0}^{k-1} \Phi^i \varepsilon_{t-i} + \Phi^k Z_{t-k}.
\end{aligned}$$

for all $k \in \mathbb{Z}^+$. Since $r(\Phi) < 1$ the operator $I - \Phi$ is invertible, and thus

$$Z_t = (I - \Phi)^{-1}(I - \Phi^k)\mathbf{c} + \sum_{i=0}^{k-1} \Phi^i \varepsilon_{t-i} + \Phi^k Z_{t-k}.$$

Taking the limit as $k \to \infty$ gives

$$Z_t = (I - \Phi)^{-1}\mathbf{c} + \sum_{i=0}^{\infty} \Phi^i \varepsilon_{t-i}. \tag{15.5}$$

Equation (15.3) follows immediately from (15.5).

To see that (15.4) holds, note that by (15.5) we have

$$\begin{aligned}
\boldsymbol{\gamma}_k(t) &= \operatorname{Cov}(Z_t, Z_{t-k}) \\
&= \mathbb{E}\left[\sum_{i=0}^{\infty} \Phi^i \varepsilon_{t-i} \sum_{j=0}^{\infty} (\Phi^j \varepsilon_{t-j-k})^{\mathsf{T}}\right] \\
&= \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \Phi^i \mathbb{E}\left[\varepsilon_{t-i}\varepsilon_{t-j-k}^{\mathsf{T}}\right] (\Phi^j)^{\mathsf{T}} \\
&= \sum_{j=0}^{\infty} \Phi^{j+k} \Sigma\, (\Phi^j)^{\mathsf{T}}
\end{aligned} \tag{15.6}$$

Hence, (15.4) holds for all k, which implies the series is covariance stationary. □

[47] See Lemma 12.3.13 of Volume 1 for more on the spectral radius.

Remark 15.2.6. A univariate AR(p) process $(Z_t)_{t\in\mathbb{Z}}$ satisfies

$$Z_t = \varphi_1 Z_{t-1} + \cdots + \varphi_p Z_{t-p} + c + \varepsilon_t.$$

This process has *characteristic polynomial*

$$1 - \varphi_1 x - \varphi_2 x^2 - \cdots - \varphi_p x^p.$$

It can be shown that if all of the roots of the characteristic polynomial have modulus greater than 1, then the series is covariance stationary.

If the root of smallest modulus is $x = 1$, then the series is said to have a *unit root*. In this case the series is not stationary.

15.2.2 MA Models

Definition 15.2.7. *A* moving averages model of order q (or MA(q) model) *is a discrete-time stochastic process* $(Z_t)_{t\in\mathbb{Z}}$ *that satisfies*

$$Z_t = \mathbf{c} + \boldsymbol{\varepsilon}_t + \Theta_1 \boldsymbol{\varepsilon}_{t-1} + \cdots + \Theta_q \boldsymbol{\varepsilon}_{t-q}, \tag{15.7}$$

where $\mathbf{c} \in \mathbb{R}^n$ *and the* $\Theta_k \in M_n(\mathbb{R})$ *are constant, and the* $\boldsymbol{\varepsilon}_k$ *are i.i.d. with mean zero and fixed covariance* Σ*. It is convenient to take* $\Theta_0 = I \in M_n(\mathbb{R})$*.*

In the case that $Z_t \in \mathbb{R}$ is real valued, the moving averages model is formally similar to a moving average of the variables ε_{t-i}, but in an MA(q) model, the Θ_i need not sum to 1 and need not be positive.

Theorem 15.2.8. *Every* MA(q) *process is covariance stationary, and*

$$\boldsymbol{\mu}(t) = \mathbf{c} \tag{15.8}$$

$$\boldsymbol{\gamma}_k(t) = \sum_{j=0}^{q-k} \Theta_{j+k}\, \Sigma\, \Theta_j^{\mathsf{T}} \qquad \text{for } 0 \le k \le q \tag{15.9}$$

$$\boldsymbol{\gamma}_k(t) = 0 \qquad \text{for } q < k, \tag{15.10}$$

where $\Theta_0 = 1$*.*

Proof. The computation of $\boldsymbol{\mu}(t)$ and $\boldsymbol{\gamma}_k(t)$ is straightforward. The details are Exercise 15.9. Since these are independent of t, the series is covariance stationary. □

Moving averages models are important in part because of the following proposition.

Proposition 15.2.9. *Given a covariance-stationary time series* $(Z_t)_{t\in\mathbb{Z}}$*, if there exists* $q > 0$ *such that the autocovariance* $\boldsymbol{\gamma}_k = 0$ *vanishes for all* $k > q$*, then* $(Z_t)_{t\in\mathbb{Z}}$ *is an* MA(q) *process.*

15.2.3 ARMA and ARIMA Models

An ARMA(p, q) model is a combination of an AR(p) model and an MA(q), model.

Definition 15.2.10. *An* ARMA(p, q) *model or* autoregressive moving-average model of order p, q *is a discrete-time stochastic process* $(Z_t)_{t\in\mathbb{Z}}$ *that satisfies*

$$Z_t = \mathbf{c} + \left(\sum_{i=1}^{p} \Phi_i Z_{t-i}\right) + \left(\sum_{j=0}^{q} \Theta_j \boldsymbol{\varepsilon}_{t-j}\right), \tag{15.11}$$

where the $\boldsymbol{\varepsilon}_t$ *are i.i.d. mean-zero random variables with constant covariance* Σ*.*

An important theorem in time series analysis is the *Wold theorem*, which essentially says that every covariance-stationary time series can be well approximated by some ARMA model. This justifies limiting our study of covariance-stationary time series to ARMA models.

15.2.4 Integrated Models

Not all ARMA models are covariance stationary, but many nonstationary time series cannot be expressed as ARMA models without some sort of transformation.

Fortunately, many time series encountered in economics and other applications can be made stationary by *differencing*. Let ΔZ_t be the times series $Y_t = Z_t - Z_{t-1}$ obtained from Z_t by taking the difference of the terms. Often the first difference ΔZ_t will already be stationary (if the trend is linear). In some cases a second difference $\Delta^2 Z_t = \Delta(\Delta Z_t)$ may be necessary (if the trend is quadratic), or, even more rarely, a third difference.

Definition 15.2.11. *An* ARIMA(p, d, q) *model or* autoregressive, integrated, moving averages model of order (p, d, q) *is a discrete-time stochastic process* $(Z_t)_{t\in\mathbb{Z}}$ *satisfying*

$$\Delta^d Z_t = \mathbf{c} + \left(\sum_{i=1}^{p} \Phi_i \mathbf{y}_{t-i}\right) + \left(\sum_{j=0}^{q} \Theta_j \boldsymbol{\varepsilon}_{t-j}\right).$$

15.3 ARIMA as a State Space Model

Let $\Theta = \{\varphi_i, \theta_j, \mu, \sigma_a^2\}$ be the set of parameters for a univariate ARMA(p, q) model. Suppose we have a collection of observations $(z_t)_{t=1}^m$. Our goal is to find the values of p, q, and Θ that maximize the likelihood of the ARMA model given the data. Using the chain rule, we can factor the likelihood of the model given this data as

$$P((\mathbf{z}_t)_{t=1}^m|\Theta) = \prod_{t=1}^{m} P(\mathbf{z}_t \,\big|\, \mathbf{z}_{t-1}, \ldots, \mathbf{z}_1, \Theta) \tag{15.12}$$

In a general ARMA(p, q) model, the likelihood is a function of the unobserved error terms a_t and is not trivial to compute. Simple approximations can be made, but these may be inaccurate under many circumstances. Explicit derivations of the likelihood are possible, but tedious. However, the ARMA model can be written as a state-space model, and then the Kalman filter affords a straightforward, recursive way to compute the likelihood.

15.3.1 ARMA as a State Space Model

Here we focus only on the univariate case, but the multivariate case can also be written as a state space model and computed using the Kalman filter.

Here we give one possible state-space representation of a univariate ARMA(p, q) model $(z_t)_{t\in\mathbb{N}}$ satisfying equation (15.11), with parameters $\{\varphi_i, \theta_j, c, \sigma_a^2\}$, where we take $\theta_0 = 1$. Let $r = \max(p, q + 1)$, and set $\varphi_k = 0$ for $k > p$ and $\theta_j = 0$ for $j > q$.

Let

$$\mu = \frac{c}{1 - \sum_{i=1}^{p} \varphi_i} \qquad \text{and} \qquad y_t = 0 \qquad \forall t < 0. \tag{15.13}$$

Inductively define $y_0, \ldots, y_{r-1}$ by

$$y_0 = z_0 - \mu \qquad \text{and} \qquad y_t = z_t - \mu - \sum_{j=1}^{r} \theta_j y_{t-j}, \tag{15.14}$$

taking $y_{-k} = 0$ for all $k \in \mathbb{N}$, and for $t \geq r$ define

$$y_t = \sum_{i=1}^{r} \varphi_i y_{t-i} + a_t, \tag{15.15}$$

where a_t has mean zero and variance σ_a^2. Let

$$\mathbf{x}_t = (y_t, y_{t-1}, \ldots, y_{t-(r-1)}) \tag{15.16a}$$

$$F = \begin{bmatrix} \varphi_1 & \varphi_2 & \cdots & \varphi_{r-1} & \varphi_r \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \tag{15.16b}$$

$$H = \begin{bmatrix} 1 & \theta_1 & \theta_2 & \cdots & \theta_{r-1} \end{bmatrix} \tag{15.16c}$$

$$Q = \begin{bmatrix} \sigma_a^2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{15.16d}$$

$$\mathbf{w}_t = (a_t, 0, 0, \ldots, 0) \tag{15.16e}$$

For $t \geq r$ Equation 15.15 gives

$$F\mathbf{x}_{t-1} + \mathbf{w}_t = \begin{bmatrix} \sum_{i=1}^{r} \varphi_i y_{t-i} \\ y_{t-1} \\ y_{t-2} \\ \vdots \\ y_{t-(r-1)} \end{bmatrix} + \begin{bmatrix} \varepsilon_t \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{15.17}$$

$$= \begin{bmatrix} y_t & y_{t-1} & \cdots & y_{t-(r-1)} \end{bmatrix}^{\mathsf{T}} \tag{15.18}$$

$$= \mathbf{x}_t \tag{15.19}$$

Proposition 15.3.1. *Using the notation defined above, we have $z_t = H\mathbf{x}_t + \mu$ and the stochastic linear dynamical system*

$$\mathbf{x}_{t+1} = F\mathbf{x}_t + \mathbf{w}_t \tag{15.20}$$

$$z_t = H\mathbf{x}_t + \mu \tag{15.21}$$

describes the same process as the original ARMA *model.*

Proof. The proof is a direct check that z_t satisfies (15.11). The details are Exercise 15.11. □

15.3.2 Integrated Models

The previous subsection shows that every ARMA model can be written as a linear state space model. But state space models describe many other models as well, including ARIMA models. The next proposition shows that every ARIMA model can be written as a linear state space model.

Proposition 15.3.2. *If Y_t can be written as a linear state space model in the form*

$$\mathbf{x}_t = F\mathbf{x}_{t-1} + \mathbf{w}_t$$
$$Y_t = H\mathbf{x}_t + \boldsymbol{\mu}$$

and we construct a new series Z_t as the following linear state space model

$$\widetilde{\mathbf{x}}_t = \widetilde{F}\widetilde{\mathbf{x}}_{t-1} + \widetilde{\mathbf{w}}_t$$
$$\widetilde{H} = \begin{bmatrix} \mathbf{0} & I & \mathbf{0} \end{bmatrix}$$
$$Z_t = \widetilde{H}\widetilde{\mathbf{x}}_t + \mathbf{c},$$

where $\mathbf{c}$ is a constant and

$$\widetilde{\mathbf{x}}_t = \begin{bmatrix} \mathbf{x}_{t+1} \\ \zeta_t \\ \boldsymbol{\mu} \end{bmatrix} \quad \text{and} \quad \widetilde{F} = \begin{bmatrix} F & \mathbf{0} & \mathbf{0} \\ H & I & I \\ \mathbf{0} & \mathbf{0} & I \end{bmatrix} \quad \text{and} \quad \widetilde{\mathbf{w}}_t = \begin{bmatrix} \mathbf{w}_t \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \tag{15.22}$$

then $\Delta Z_t = Y_t$ (the initial value of ζ_0 is unimportant, but later values of ζ_t are determined by the state equations above).

Proof. The proof is Exercise 15.12. □

Proposition 15.3.3. *For any time series Z'_t whose first difference $\Delta Z'_t$ can be written as a state space model, the original time series Z'_t can also be written as a state space model.*

Proof. The proof follows by showing that any two time series Z_t and Z'_t with the same difference $\Delta Z'_t = \Delta Z_t$ differ only by a random variable C that is independent of time $Z'_t = Z_t + C$ and then using the result of Proposition 15.3.2. The details are Exercise 15.13. □

Many other types of time series models can be expressed as state space models, making them amenable to Kalman-based techniques for fitting. See [DK12] for more details.

15.3.3 Learning and Forecasting an ARIMA Model

Once an ARIMA model is expressed as a state space model, the parameters of the model can be estimated from the data using the techniques of Section 14.4. For a given choice of p, q and d, the model has a specific form, as given in (15.16) or (15.22). Expressing the log likelihood in terms of the free parameters in the models (15.16) and (15.22) improves efficiency when maximizing the log likelihood. A grid search over various choices of p, q, d using the corrected AIC (AICc) yields a good estimate for all the parameters.

Given a good estimate of the parameters p, q, d and $\Theta = (F, H, Q, R, \boldsymbol{\mu}, \widehat{x}_{1|0}, P_{1|0})$, future states and observations can be forecast in the usual way (see Section 14.4.4).

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second line are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

15.1. Prove the following claims made in the proof of Proposition 15.1.3:

(i) For any vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ we have $\operatorname{tr}(\mathbf{v}\mathbf{w}^\mathsf{T}) = \mathbf{v}^\mathsf{T}\mathbf{w}$

(ii) $\operatorname{tr}(\mathbf{v}\mathbf{v}^\mathsf{T}) = \|\mathbf{v}\|_2^2$.

(iii) The pairing $\langle X, Y\rangle = \operatorname{tr}(\mathbb{E}[XY^\mathsf{T}])$ is an inner product on the space of mean-zero random variables with support in $\mathbb{R}^n$.

(iv) The norm for the inner product given in the previous claim satisfies

$$\|X\|^2 = \langle X, X\rangle = \operatorname{tr}\mathbb{E}[XX^\mathsf{T}] = \mathbb{E}[\|X\|_2^2].$$

15.2. Prove Proposition 15.1.4.

15.3. Prove Proposition 15.1.7. Explain why the proposition is false for even integers m if the naïve moving average $\frac{1}{m}\left(Z_{t-\ell} + \cdots + Z_{t+\ell}\right)$ is used instead of the symmetric moving average.

15.4. Let $Z_t = T_t + Y_t$ be a sum of time series where Y_t has $\mathbb{E}[Y_t] = \mathbf{0}$ for all t and $T_t = \sum_{i=0}^{k} t^i \mathbf{c}_i$ for constants $\mathbf{c}_i \in \mathbb{R}^n$.

(i) Prove that the expected value of the kth forward difference $\Delta^k Z_t$ is $k!\mathbf{c}_k$.

(ii) Prove that if m is large enough that $\frac{1}{2m+1}\sum_{j=-m}^{m} \mathbf{y}_{t+j} \approx \mathbf{0}$ and $\mathbf{c}_i = \mathbf{0}$ for all even indices i, then $T_t \approx \frac{1}{2m+1}\sum_{j=-m}^{m} \mathbf{z}_{t+j}$ for all $m \le t \le N-m$. But if one of the $\mathbf{c}_i$ is nonzero for an even index of i, then T_t could differ substantially from the moving average $\frac{1}{2m+1}\sum_{j=-m}^{m} \mathbf{z}_{t+j}$.

15.5. Download the hourly report of Utah air quality (PM2.5 levels) for 2015 at `http://www.airmonitoring.utah.gov/dataarchive/2015-PM2.5.csv` We will focus on the north Provo data (station NP). Values reported are micrograms per cubic meter.

(i) Clean up the data: among other things, there should be no negative values. Why not? What is a good strategy for imputing the missing values?

(ii) Construct a classical decomposition of the time series as follows:

(a) Estimate the trend with a symmetric, length-24 moving average $\widehat{T}_t^{24}$ corresponding to averaging over all the hours in a day and plot $\widehat{T}_t$.

(b) Detrend the original series to $y_t = z_t - \widehat{T}_t$ (drop a day on each end where $\widehat{T}_t$ is not defined).

(c) Compute the hour-of-the-day seasonal component by finding the mean of y_t for each hour of the day—that is, find

$$\widehat{S}_k = \operatorname{mean}(y_k, y_{k+24}, y_{k+48}, \ldots)$$

for each $k \in 0, \ldots, 23$.

(d) Plot the series $\widehat{S}$. Interpret.

(e) Remove the hourly seasonal component from the detrended data to get $\widehat{R}_t = y_t - \widehat{S}_t$ (remember that $\widehat{S}_{t+24}$ should equal $\widehat{S}_t$ for all t). Plot the result.

(f) If the resulting time series is stationary, then the autocorrelation should be constant for each lag k. Compute the sample mean and the sample autocovariance γ_k for each $k \in 0, 1, \ldots, 20$ (treat each time t and its corresponding x_{t-k}) as one sample pair). Use these to compute the autocorrelation for each k. Plot the autocorrelation as a function of k.

(g) For each $j \in 1, 2, 3$ compute the jth difference $\Delta_j \widehat{R}_t$ time series and compute and plot the the autocorrelation as a function of k (as in part (ii)f) for each of these three time series. Hint: The first different operator is defined in Remark 15.1.10. The jth difference operator is computed by applying the first difference operator j times.

15.6. Consider univariate AR(1) models of the form $Z_t = c + \varphi Z_{t-1} + \varepsilon_t$, where $\varepsilon_t \sim \mathcal{N}(0, \frac{1}{4})$. Simulate (and plot) such time series for $t \in \{0, \ldots, 50\}$ with the following parameter values:

(i) $c = 0, \varphi = 0.5$

(ii) $c = 0, \varphi = 1.5$

(iii) $c = 0, \varphi = 1.0$

(iv) $c = 1, \varphi = 0.5$

(v) $c = 1, \varphi = 1.5$

(vi) $c = 1, \varphi = 1.0$

(vii) $c = -1, \varphi = 0.5$

(viii) $c = -1, \varphi = 1.5$

(ix) $c = -1, \varphi = 1.0$

15.7. Consider univariate AR(2) models of the form $Z_t = c + \varphi_1 Z_{t-1} + \varphi_2 Z_{t-2} + \varepsilon_t$, where $\varepsilon_t \sim \mathcal{N}(0, \frac{1}{4})$. Simulate (and plot) such time series for $t \in \{0, \ldots, 50\}$ with the following parameter values:

(i) $c = 0, \varphi_1 = 0, \varphi_2 = 4$

(ii) $c = 0, \varphi_1 = 1, \varphi_2 = 4$

(iii) $c = 0, \varphi_1 = 1, \varphi_2 = 1$

(iv) $c = 0, \varphi_1 = 1, \varphi_2 = 0.25$

(v) $c = 1, \varphi_1 = 0, \varphi_2 = 4$

(vi) $c = 1, \varphi_1 = 1, \varphi_2 = 4$

(vii) $c = 1, \varphi_1 = 1, \varphi_2 = 1$

(viii) $c = 1, \varphi_1 = 1, \varphi_2 = 0.25$

(ix) $c = -1, \varphi_1 = 0, \varphi_2 = 4$

(x) $c = -1, \varphi_1 = 1, \varphi_2 = 4$

(xi) $c = -1, \varphi_1 = 1, \varphi_2 = 1$

(xii) $c = -1, \varphi_1 = 1, \varphi_2 = 0.25$

15.8. Consider univariate MA(1) models of the form $Z_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1}$, where $\varepsilon_t \sim \mathcal{N}(0, \frac{1}{4})$. Simulate (and plot) such time series for $t \in \{0, \ldots, 50\}$ with the parameter values given below. Also compute the sample autocovariance $\widehat{\gamma}_k$ for $k \in \{0, \ldots, 10\}$ for each of these series and compare your answer to the values given in Theorem 15.2.8.

(i) $c = 0, \theta_1 = 1$

(ii) $c = 0, \theta_1 = 2$

(iii) $c = 0, \theta_1 = \frac{1}{2}$

(iv) $c = 1, \theta_1 = 1$

(v) $c = 1, \theta_1 = 2$

(vi) $c = 1, \theta_1 = \frac{1}{2}$

15.9. Prove Theorem 15.2.8. Is the theorem still true if the time series starts at time $t = 0$ instead of running over all $t \in \mathbb{Z}$? Why or why not?

15.10. Do the following for the Provo air-quality time series z_t of Exercise 15.5.

(i) Check to see if (z_t) is easily identifiable as an
2) model by regressing z_t on the lags z_{t-1} and z_{t-2} (and a constant term!). The R^2 should give a reasonable sense of how good the fit is.

(ii) Repeat part (i) for $\Delta_j z_t$ for each $j \in \{1, 2, 3\}$. What do you conclude about this time series?

15.11. Prove Proposition 15.3.1.

15.12. Prove Proposition 15.3.2.

15.13. Prove Proposition 15.3.3.

Notes

Part IV

Large Models

16 High-Dimensional Space

When discussing complex systems like brains and other societies, it is easy to over-simplify: I call this Occam's lobotomy.
—I.J. Good

Details are all that matters: God dwells there, and you will never get to see Him if you don't struggle to get them right.
—Stephen J. Gould

This chapter treats the mathematics of higher-dimensional geometry. When working with data it is common to see high-dimensional data sets. For example many cameras have over 10 million pixels, and each pixel corresponds to a red, green, and blue value in $\mathbb{R}^3$, so a single photograph corresponds to a point in $(\mathbb{R}^3)^{1,000,000} = \mathbb{R}^{3,000,000}$. As another example, in most streaming services like AmazonPrime or Netflix, each user can rate a show on some scale. To build a recommender system (to recommend to users what to watch next), it's common to store all of a user's ratings as a giant vector, with one dimension for each show—so if there are 3000 shows, that's a point in $\mathbb{R}^{3000}$. Many of these entries are initially blank, but the job of the recommender system is to identify a predicted rating for each show, that is, to identify the point in $\mathbb{R}^{3500}$ corresponding to each user.

Aside from the obvious effects of the curse of dimensionality, many of the abstract tools introduced in this volume handle high dimensional data sets without too much difficulty. But intuition about the geometry of high dimensional spaces does not abstract well from the two- and three-dimensional geometry we are used to working in. As dimension increases, some of our intuition about what "should" be true falls short. As an example of how geometry differs in high dimensional spaces, we consider the volume of a unit ball. Recall (Volume 1, Exercise 8.35) that the volume of an n-dimensional unit ball is given by the formula

$$V(n) = \frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)}. \tag{16.1}$$

And the volume of the n-dimensional ball of radius r is

$$V(n,r) = \frac{\pi^{n/2} r^n}{\Gamma\left(\frac{n}{2}+1\right)}. \tag{16.2}$$

Our intuition may lead us to believe that as dimension increases, the volume of a unit ball should also increase. This assumption seems validated by considering Table 16.1.

Dimension	Unit Sphere	Volume
1	$[-1,1]$	2
2	$\{(x,y)\|x^2+y^2 \leq 1\}$	$\pi \approx 3.14$
3	$\{(x,y,z)\|x^2+y^2+z^2 \leq 1\}$	$\frac{4}{3}\pi \approx 4.19$
4	$\{\mathbf{x} \in \mathbb{R}^4 \mid \|\mathbf{x}\| \leq 1\}$	$\frac{1}{2}\pi^2 \approx 4.93$
5	$\{\mathbf{x} \in \mathbb{R}^5 \mid \|\mathbf{x}\| \leq 1\}$	$\frac{8}{15}\pi^2 \approx 5.26$

Table 16.1: The volume of a unit ball grows with dimension up to dimension five, but beyond that it starts to decrease with dimension (see also Figure 16.1.)

Surprisingly this trend does not continue as n increases: after $n = 5$ the volume of the n-dimensional unit ball decreases as n increases. This can be seen by noting that the denominator $\Gamma(n/2+1)$ of (16.2) grows as a factorial (greater than $\left(\sqrt{\frac{n}{2e}}\right)^n$ by Volume 2, Equation (2.1)), whereas the numerator only grows exponentially (namely, as $(r\sqrt{\pi})^n$).

16.1 Properties of the unit ball in higher dimensional space

In this section, we focus on the properties of the unit ball in n-dimensional space. As shown in the introduction, the volume of these balls goes to zero as the dimension n increases. We take inspiration from the text *Foundations of Data Science* by Blum, Hopcroft, and Kannan.

16.1.1 Volume near the boundaries

In high dimensions most of the volume of objects is concentrated near the surface. This is true for any (measurable) high dimensional object A, but we begin with case of a ball of radius r centered at the origin. Shrinking the ball by a factor of $(1-\varepsilon)$ produces a smaller ball $\hat{A} = \{(1-\varepsilon)x | x \in A\}$. Equation (16.2) gives

$$\text{volume}(\hat{A}) = V(n, r(1-\varepsilon)) = (1-\varepsilon)^n V(n,r) = (1-\varepsilon)^n \text{volume}(A).$$

Thus the ratio of volumes of $\hat{A}$ and A shrinks to 0

$$\frac{\text{volume}(\hat{A})}{\text{volume}(A)} = (1-\varepsilon)^n \to 0 \tag{16.3}$$

as $n \to \infty$. This means that the vast majority of the volume of a high-dimensional ball must be concentrated near the boundary. In fact, for a n-dimensional ball of radius r most of the volume is concentrated in a shell of thickness $O(r/n)$ near the boundary.

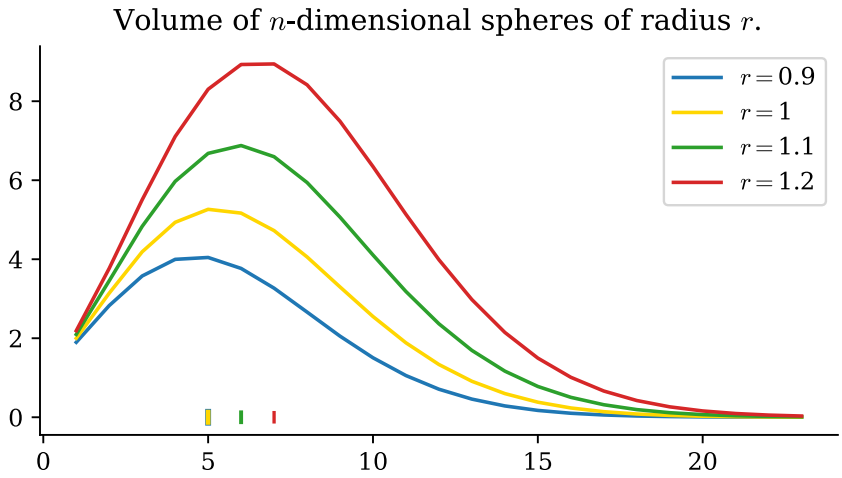


Figure 16.1: Plot of the volume of unit n-balls of radius r as n grows. Initially the volume grows as the dimension increases, but after a certain point (marked on the x-axis with a vertical line of the appropriate color) the volume only decreases as n continues to grow. Note also that for large n, a small change in radius produces a large change in volume, showing that a significant proportion of the volume of any n-ball lies very near outer edge.

Similarly, an n-cube of side length r has volume r^n, and an n-cube of side length $r(1-\varepsilon)$ has volume $r^n(1-\varepsilon)^n$. Thus the ratio of the two volumes is $(1-\varepsilon)^n \to 0$ as $n \to \infty$. Again, this means that the vast majority of the volume of the n-cube is concentrated near its outer boundary. Since the volume of any measurable set in n-space can be computed by successive approximations by n-cubes, the same result (16.3) applies for any measurable sets A and $\hat{A} = \{(1-\varepsilon)\mathbf{x} \mid \mathbf{x} \in A\}$.

16.1.2 Volume Near the Equator

In addition to the volume of high dimensional objects being concentrated near the outer shell, the volume is also concentrated in a small slice through the center of the ball. Sometimes this is shortened to say that the volume is concentrated near any "equator" of the ball, although that language may be a little confusing.

The combination of the two observations (volume concentrated near the boundary and concentrated in any equator) leads to the somewhat surprising result that in high dimensions when two points are chosen at random in the unit ball they are very likely to be nearly orthogonal.

We start by showing that the distribution of volume in a n-dimensional ball is concentrated in a small slice through the center of the ball.

Theorem 16.1.1. *Let S be the n-dimensional unit ball $\{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\|_2 \leq 1\}$ centered about the origin in $\mathbb{R}^n$. For any $c \geq 1$ and any $n \geq 3$, at least a $1 - \frac{2}{c}e^{-c^2/2}$ fraction of the volume of S has $|x_1| < \frac{c}{\sqrt{n-1}}$, where x_1 denotes the first coordinate of $\mathbf{x} \in \mathbb{R}^n$.*

Proof. By symmetry we need only show that at most a $\frac{2}{c}e^{-c^2/2}$ fraction of the hemisphere with $x_1 \geq 0$ has $x_1 \geq \frac{c}{\sqrt{n-1}}$. Let $A = \left\{\mathbf{x} \in \mathbb{R} \mid x_1 \geq \frac{c}{\sqrt{n-1}}\right\}$ be the part of the ball with $x_1 \geq \frac{c}{\sqrt{n-1}}$ and $T = \{\mathbf{x} \in \mathbb{R} \mid x_1 \geq 0\}$ be the upper hemisphere. The ratio of the volume of A to the volume of T can be bounded by considering an upper bound on the volume of A and a lower bound on the volume of T. More specifically we claim that

$$\frac{\text{volume}(A)}{\text{volume}(T)} \leq \frac{\text{upper bound volume}(A)}{\text{lower bound volume}(T)} = \frac{2}{c}e^{-c^2/2}.$$

To calculate the volume of A consider a disk of width dx_1 whose face is a ball of dimension $n-1$ and radius $\sqrt{1-x_1^2}$. The surface area of the disk is $(1-x_1^2)^{(n-1)/2}V(n-1)$ and the volume above the slice is

$$\text{volume}(A) = \int_{\frac{c}{\sqrt{n-1}}}^{1} (1-x_1^2)^{(n-1)/2}V(n-1)\,dx_1.$$

This integral is not easy to compute, but we can bound it from above by first integrating to infinity instead of to 1 and then noting that $1 - x \leq e^{-x}$ when $x \geq 0$. In the range of integration ($x_1 > \frac{c}{n-1}$) we also have $\frac{x_1\sqrt{n-1}}{c} \geq 1$. This gives

$$\begin{aligned}
\text{volume}(A) &= \int_{\frac{c}{\sqrt{n-1}}}^{1} (1-x_1^2)^{(n-1)/2}V(n-1)dx_1 \\
&\leq \int_{\frac{c}{\sqrt{n-1}}}^{\infty} \frac{x_1\sqrt{n-1}}{c}e^{-\frac{n-1}{2}x_1^2}V(n-1)dx_1 \\
&= V(n-1)\frac{\sqrt{n-1}}{c}\int_{\frac{c}{\sqrt{n-1}}}^{\infty} x_1 e^{-\frac{n-1}{2}x_1^2}\,dx_1 \\
&= \frac{V(n-1)}{c\sqrt{n-1}}e^{-c^2/2}.
\end{aligned}$$

Although we have the formula (16.1) for the volume of the ball, and hence could easily compute the volume of T, we want, instead, to bound that volume (below) in terms of $V(n-1)$. Observe that the volume of T is certainly greater than the volume of the part of the unit ball lying below the hyperplane $x_1 = \frac{1}{\sqrt{n-1}}$ and above the hyperplane $x_1 = 0$. That volume is bounded from below by the volume of a cylinder of height $\frac{1}{\sqrt{n-1}}$ and radius $r = \sqrt{1 - \frac{1}{(n-1)}}$ (draw a picture in three dimensions). The base of that cylinder is an a $n-1$-ball of radius r, so the volume of the cylinder is

$$V(n-1)r^{n-1}\frac{1}{\sqrt{n-1}} = V(n-1)\left(1 - \frac{1}{(n-1)}\right)^{(n-1)/2}\frac{1}{\sqrt{n-1}}.$$

Figure 16.2: The volume of a high dimensional ball is mostly concentrated in a small slice through the center of the ball. In three dimensions, a great circle (black) is the intersection of a plane H through the origin with the outer surface of the ball. We are interested in a general hyperplane H through the origin. Taking all the points in the ball that are at most $\frac{c}{\sqrt{n-1}}$ away from the hyperplane H, gives a slice of the ball (bounded in red) with much of the volume. Specifically, Corollary 16.1.2 guarantees that the volume of the slice is at least $1 - \frac{2}{c}e^{-c^2/2}$ times the volume of the ball.

The fact that $(1-x)^a \geq 1 - ax$ for $a \geq 1$ implies that the volume of the cylinder is at least $\dfrac{V(n-1)}{2\sqrt{n-1}}$ for $n \geq 3$. Thus

$$\frac{\text{volume}(A)}{\text{volume}(T)} \leq \frac{\text{upper bound volume}(A)}{\text{lower bound volume}(T)} = \frac{\frac{V(n-1)}{c\sqrt{n-1}}e^{-c^2/2}}{\frac{V(n-1)}{2\sqrt{n-1}}} = \frac{2}{c}e^{\frac{-c^2}{2}}. \quad \square$$

Corollary 16.1.2. *Let S be the n-dimensional unit ball centered about the origin in $\mathbb{R}^n$. For any hyperplane H through the origin, and for any $c \geq 1$ and $n \geq 3$, at least a $1 - \frac{2}{c}e^{-c^2/2}$ fraction of the volume of S lies no more than $\frac{c}{\sqrt{n-1}}$ away from the hyperplane H.*

Proof. This follows immediately by applying a rigid rotation (orthonormal transformation) R to space to put the hyperplane H in the position of $\{\mathbf{x} \in \mathbb{R}^n \mid x_1 = 0\}$. Any rigid rotation maps the unit ball to itself and volumes are unchanged because $\det(R) = 1$. To see that such a rotation exists, note first that every hyperplane through the origin can be written as $H = \{\mathbf{x} \mid \langle \mathbf{x}, \mathbf{w} \rangle = 0\}$ for some $\mathbf{w} \in \mathbb{R}^n$. Without loss of generality, we may take $\|\mathbf{w}\| = 1$ (rescale $\mathbf{w}$ if necessary). Any orthonormal matrix with $\mathbf{w}^\mathsf{T}$ as its first row will rotate $\mathbf{w}$ to $\mathbf{e}_1 = (1, 0, \ldots, 0)$ and thus transform H to $\{\mathbf{x} \mid \langle \mathbf{x}, \mathbf{e}_1 \rangle = 0\} = \{\mathbf{x} \mid x_1 = 0\}$. $\square$

This result, combined with our observation that most of the volume is located near the boundary of the ball, leads us to the conclusion that if two points are drawn at random from a unit ball, then they will probably be nearly orthogonal to each other. More specifically, we know with high probability that the first point drawn will be close to the surface. If we define that point to be the "north pole" we know that with high probability any other point drawn will lie in the equatorial region of the ball with the first point as the north pole. This idea is formalized and strengthened in the following theorem.

Theorem 16.1.3. *Consider drawing d points $\mathbf{x}_1, \ldots, \mathbf{x}_d$ uniformly from the unit ball in $\mathbb{R}^n$. With probability at least $1 - \frac{1}{d}$*

(i) $|\mathbf{x}_i| \geq 1 - \frac{2 \ln d}{n}$ *for all i, and*

(ii) $|\langle \mathbf{x}_i, \mathbf{x}_j \rangle| \leq \frac{\sqrt{6 \ln d}}{\sqrt{n-1}}$ *for all $i \neq j$.*

Proof. For (i), for any fixed i the analysis of 16.1.1 shows that $|\mathbf{x}_i| < 1 - \varepsilon$ with probability less than $e^{-\varepsilon n}$. This implies

$$P\left(|\mathbf{x}_i| < 1 - \frac{2 \ln d}{n}\right) \leq e^{-\left(\frac{2 \ln d}{n}\right) n} = \frac{1}{d^2},$$

which gives

$$P(\exists i \text{ s.t. } |\mathbf{x}_i| < 1 - \frac{2 \ln d}{n}) = P\left(\bigvee_{i=1}^{d} \left(|\mathbf{x}_i| < 1 - \frac{2 \ln d}{n}\right)\right) \leq \sum_{i=1}^{d} \frac{1}{d^2} = \frac{1}{d}.$$

For part (ii) consider any pair $\mathbf{x}_i, \mathbf{x}_j$, and let H be the hyperplane $\{\mathbf{x} \mid \langle \mathbf{x}_i, \mathbf{x} \rangle = 0\}$. The inner product $\left\langle \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|}, \mathbf{x}_j \right\rangle$ is exactly the distance δ from $\mathbf{x}_j$ to H; and thus $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \delta \|\mathbf{x}_i\| \leq \delta$, since $\mathbf{x}_i$ is drawn from the unit ball. By Corollary 16.1.2, for any $c \geq 1$ the probability that the distance δ is more than $\frac{c}{\sqrt{n-1}}$ from H is at most $\frac{2}{c} e^{-c^2/2}$; therefore the probability that $\langle \mathbf{x}_i, \mathbf{x}_j \rangle > \frac{\sqrt{6 \log d}}{\sqrt{n-1}}$ is at most $\frac{2}{\sqrt{6 \log d}} e^{-3 \log d} = \frac{2}{d^3 \sqrt{6 \log d}}$. There are $\binom{d}{2}$ pairs i and j, thus the inner product condition is violated with probability at most

$$\frac{d(d-1)}{2} \frac{2}{d^3 \sqrt{6 \log d}} = \frac{(d-1)}{d^2 \sqrt{6 \log d}} < \frac{1}{d}. \quad \square$$

16.1.3 Drawing points uniformly at random from a ball

To draw point uniformly from the unit ball, we can use rejection sampling, drawing from a bounding box placed around the ball. In lower dimensions this works well. As the dimension increases, however, this method falls short because as the dimension increases more of the volume is concentrated near the boundary. Thus a smaller and smaller fraction of the points drawn from the cube will lie inside the ball, resulting in many of the points being rejected. Here is an alternative method for drawing uniformly from an n-dimensional ball.

Consider drawing n i.i.d. points $x_1, \ldots x_n$ from a one-dimensional standard normal (zero-mean unit-variance) distribution. The probability density of $\mathbf{x} = (x_1, \ldots, x_n)$ is the product of the standard normal one-dimensional Gaussian p.d.f.s, namely $\mathcal{N}(\mathbf{0}, I)$

$$f_X(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}} \exp(-(x_1^2 + x_2^2 + \cdots x_n^2)/2), \tag{16.4}$$

and is spherically symmetric. Normalizing this vector to a unit vector gives a distribution that is uniform over the surface of the n-dimensional ball. It is important to note that once the vector is normalized the individual coordinates are no longer statistically independent.

This algorithm gives a uniform distribution of points on the surface of a unit ball in n dimensions, but we'd like to generate points uniformly in both the surface and the interior of the ball. A naïve approach would be to simply multiply the point on the surface $\mathbf{x}/\|\mathbf{x}\|$ by a scalar ρ uniformly drawn from the interval $[0, 1]$. But this does not yield a uniform distribution of points in the interior of the ball—too many of the points constructed this way will be near the origin. Instead the density of ρ should depend on the distance from the origin. In the case of the two-dimensional ball, the density of ρ at a point of radius r should be proportional to r. And, since the p.d.f. $f_\rho(r)$ must integrate to 1, it must be $f_\rho(r) = 2r$; that is, $\rho \sim \text{Beta}(2, 1)$. This corresponds to the fact that in polar coordinates the uniform distribution on the unit ball is computed by integrating the density

$$\frac{1}{\pi} r \, dr \, d\theta = (2r \, dr) \left(\frac{1}{2\pi} \, d\theta \right) \tag{16.5}$$

That is, select a point on the circumference with a draw from Uniform($[0, 2\pi)$) with density $\frac{1}{2\pi} \, d\theta$, and select a radius with density $2r \, dr$. These are independent draws, so the resulting density is the product (16.5).

For the three-dimensional ball the density of ρ should be proportional to r^2, and to integrate to 1 it must be $3r^2$; that is $\rho \sim \text{Beta}(3, 1)$. This corresponds to the fact that in spherical coordinates, the density of the uniform distribution on the three-ball is

$$\frac{3}{4\pi} r^2 \sin(\phi) \, dr \, d\phi \, d\theta = (3r^2 \, dr) \left(\frac{1}{4\pi} \sin(\phi) \, d\phi \, d\theta \right). \tag{16.6}$$

That is, draw a point on the surface of the sphere from the uniform distribution with density $\frac{1}{4\pi} \sin(\phi) \, d\phi \, d\theta$, and draw a radius with density $3r^2 \, dr$. These are independent draws, so the resulting density is the product (16.6). In n dimensions, the correct distribution to draw ρ from is proportional to r^{n-1} and thus is given by the probability density function $f_\rho(r) = nr^{n-1}$, that is, $\rho \sim \text{Beta}(n, 1)$.

Nota Bene 16.1.4. In summary, to draw a point uniformly from the interior of the unit ball in n dimensions, first draw n values $x_1, \ldots, x_n$ from a one-dimensional standard normal distribution. Putting those together as $\mathbf{x} = (x_1, \ldots, x_n)$ gives one point from the Gaussian $\mathcal{N}(\mathbf{0}, I)$. Divide by the norm of $\mathbf{x}$ to get $\mathbf{u} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$ of unit length (on the surface of the n-sphere). Now draw $\rho \sim \beta(n, 1)$, and the product $\rho\mathbf{u} = \rho\frac{\mathbf{x}}{\|\mathbf{x}\|}$ is the desired point on the interior of the unit ball.

16.1.4 Gaussians in high dimensions

Recall that in a single dimension (and even in two dimensions) a Gaussian distribution has its mass concentrated near the origin (see Figures 4.3 and 4.5). In arbitrary dimensions the p.d.f. of the Gaussian is given by

$$f_X(x) = (2\pi)^{-n/2} \det(\Sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\intercal \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right).$$

The density is maximized at the origin, but in high dimensions, there is very little volume there, so there is little mass near the origin. One must increase the radius of a ball to nearly $\sqrt{n}$ before there is significant volume and hence significant probability mass. However, increasing the radius of the ball much beyond $\sqrt{n}$ does not increase the probability mass significantly because density shrinks faster than the volume increases. This is formalized by the following theorem.

Theorem 16.1.5 (Gaussian Annulus Theorem). *For an n-dimensional spherical Gaussian with unit variance in each direction and for any $\beta \leq \sqrt{n}$, all but at most $3e^{-c\beta^2}$ of the probability mass lies within the annulus $\sqrt{n} - \beta \leq |\mathbf{x}| \leq \sqrt{n} + \beta$, where c is a positive constant.*

The proof of this theorem requires the use of the Master Tail Theorem (a generalization of the concentration inequalities found in Chapter 8) and is beyond the scope of this section. The Gaussian Annulus Theorem has multiple uses in data science, including giving bounds for separable Gaussian mixture models. It also provides an alternate proof of the Johnson-Lindenstrauss Theorem (Theorem 16.3.1).

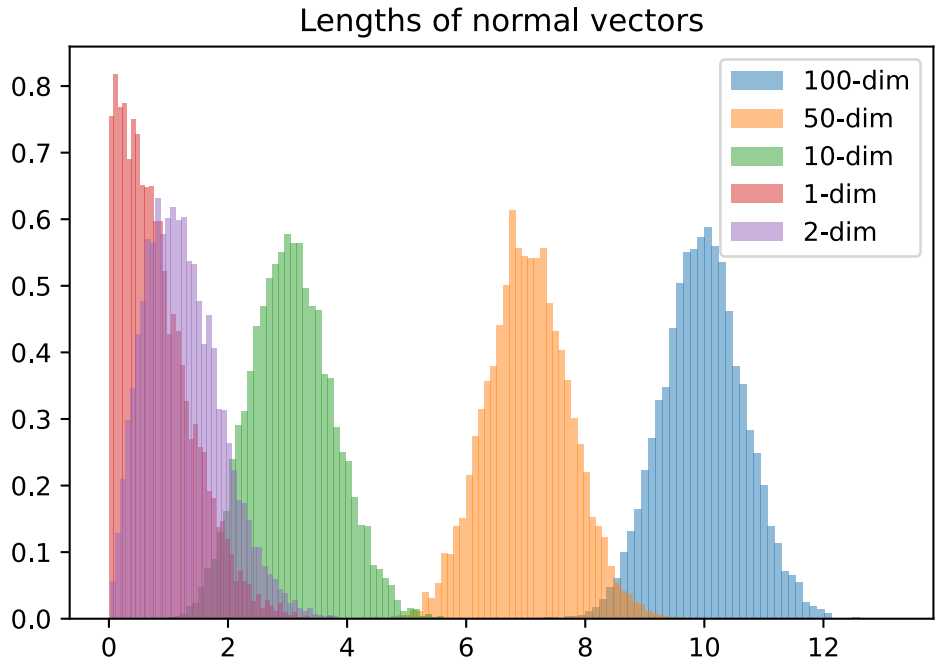


Figure 16.3: Histograms of the norms of a large number of points drawn from a multivariate normal with mean zero and covariance I. In dimension 1, most of the points have norm near zero. As the dimension n grows, the norms of most of the points have their norm in a relatively narrow band around $\sqrt{n}$.

16.2 Linear Dimension Reduction

Data points may may have a large number of features, but often there are good reasons to believe that the underlying process generating the data is low dimensional. In such a situation, it makes sense to try to identify the a low-dimensional space which contains most of the variability of the data and project the data to this subspace. This reduces the complexity (both spatial and temporal) of any analysis done, reduces noise, and tends to reduce variance (overfitting) of the models developed.

Many mathematical models are linear, and if the features are all measured in the same units, then we can use linear techniques for dimension reduction. Some of the most important of these are *principle component analysis*, which is just a simple application of the SVD, and *nonnegative matrix factorization*, which we discuss in Section 16.5.

16.2.1 Principle Component Analysis

Assume that the data consists of n samples from $\mathbb{R}^d$, stored in the form of an $n \times d$ matrix $\mathbb{X}$, where the ith row $\mathbf{x}_i^\mathsf{T} = \begin{bmatrix} x_{i1} & \dots & x_{id} \end{bmatrix}$ corresponds to the ith data point.

Nota Bene 16.2.1. We always center the data by subtracting the sample mean $\hat{\boldsymbol{\mu}} = (\hat{\mu}_1, \dots, \hat{\mu}_d) = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i$; that is, replace each $\mathbf{x}_i$ by $\mathbf{x}_i - \hat{\boldsymbol{\mu}}$ to get $\mathbb{X} - \mathbb{1}_n \hat{\boldsymbol{\mu}}^\mathsf{T}$, where $\mathbb{1}_n = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}^\mathsf{T}$. We assume throughout the rest of this section that the data has been centered in this way so that $\hat{\boldsymbol{\mu}} = \mathbf{0}$.

Recall that the sample covariance of the data is the matrix

$$\mathrm{Cov} = \frac{1}{n-1}\sum_{i=1}^n (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^\mathsf{T} = \frac{1}{n-1}\mathbb{X}^\mathsf{T}\mathbb{X}.$$

Applying the SVD to $\mathbb{X}$ gives $\mathbb{X} = U\Sigma V^\mathsf{T}$, with Σ a diagonal $n \times d$ matrix and U and V orthonormal matrices of dimension $n \times n$ and $d \times d$, respectively. This gives

$$\mathrm{Cov} = \frac{1}{n-1}\mathbb{X}^\mathsf{T}\mathbb{X} = \frac{1}{n-1} V\Sigma^\mathsf{T}\Sigma V^\mathsf{T}.$$

Changing the basis of $\mathbb{R}^d$ from the standard basis to the columns of V, sends each $\mathbf{x}_i$ to $V^\mathsf{T}\mathbf{x}_i$; and this diagonalizes the sample covariance

$$\mathrm{Cov} \mapsto V^\mathsf{T}\,\mathrm{Cov}\,V = \frac{1}{n-1} V^\mathsf{T}\mathbb{X}^\mathsf{T}\mathbb{X}V = \frac{1}{n-1}\Sigma^\mathsf{T}\Sigma.$$

Recall that the diagonal matrix Σ is ordered so that if $\Sigma = \mathrm{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0)$, then $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, where r is the rank of Σ. This means that in this orthonormal basis, the covariance $\mathrm{Cov}(\mathbf{v}_i, \mathbf{v}_j) = 0$ if $i \neq j$, and the variance $\mathrm{Var}(\mathbf{v}_i)$ is equal to σ_i^2. Therefore, the direction of greatest variance is $\mathbf{v}_1$, and the direction of least variance is $\mathbf{v}_d$. The Schmidt, Mirsky, Eckart–Young theorem (Volume 1, Thm. 4.6.3) guarantees that the approximation $\mathbb{X}_s = \sum_{i=1}^s \mathbf{u}_i\sigma_i\mathbf{v}_i^\mathsf{T}$ is the best rank-s approximation of $\mathbb{X}$ (measured either in terms of the 2-norm or the Frobenius norm).

If all the features in the data are measured in the same units, then we can reduce the dimension of the feature space by discarding some of the directions of lowest variance, while keeping the directions of greatest variance; that is, for some rank $s < r$, project each data point $\mathbf{x}$ onto the subspace of $\mathbb{R}^d$ spanned by the first s columns $\mathbf{v}_1, \dots, \mathbf{v}_s$ of V by mapping $\mathbf{x} \mapsto V_s^\mathsf{T}\mathbf{x}$, where

$$V_s^\mathsf{T} = \begin{bmatrix} \mathbf{v}_1^\mathsf{T} \\ \vdots \\ \mathbf{v}_s^\mathsf{T} \end{bmatrix}$$

This gives an s-dimensional approximation of $V^\mathsf{T}\mathbf{x}$ that preserves the directions of greatest variance and discards those of least variance. This is called *principal component analysis*. The orthonormal vectors $\mathbf{v}_1, \dots, \mathbf{v}_s$ are called the *principal axes* of the data, and the *principal components* of a data point $\mathbf{x}$ are the coefficients $(a_1 \dots, a_s)$ of the projection $\mathbf{x} \approx \mathrm{proj}_{\mathrm{span}(\mathbf{v}_1,\dots,\mathbf{v}_s)}\,\mathbf{x} = \sum_{i=1}^s a_i\mathbf{v}_i$.

Nota Bene 16.2.2. PCA and the decision of which features to keep must only be calculated using the training set. Once it has been computed, that same transformation must be used on the test set as trained—not recomputed on the test set. Recomputing the PCA on the test set is a form of data leakage and will give grossly misleading estimates of the generalization error of your methods.

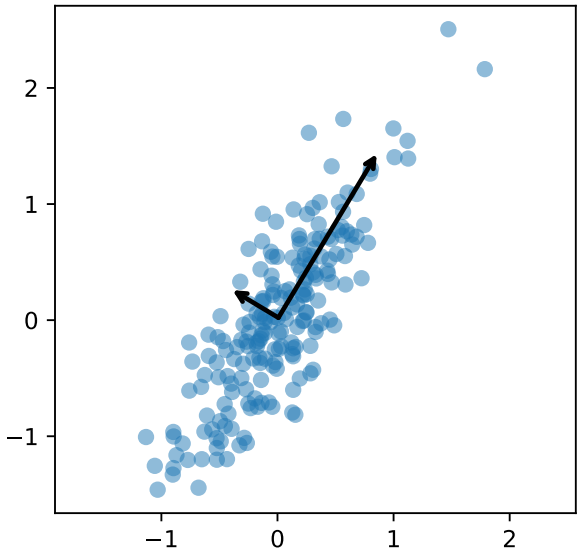


Figure 16.4: Principal component analysis of some two-dimensional data. PCA identifies an orthonormal basis of *principal axes* of the data. These are arranged in order of the amount of variance they account for, so the most variance occurs in the direction of the first principal axis (the long arrow), and the least variance occurs in the the direction of the last principal axis (the short arrow).

16.2.2 Application of PCA

PCA has multiple benefits and applications, including

(i) Removing linear dependence relations

(ii) Compressing the data.

(iii) Reducing temporal complexity of algorithms that use the data.

(iv) Noise reduction

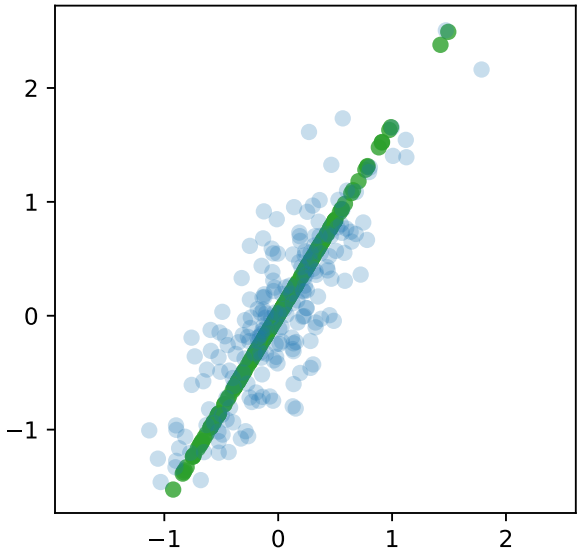


Figure 16.5: Orthogonally projecting (green) the data onto the first principal axis puts the data into a one-dimensional subspace while maintaining much of the variance of the original data (blue).

Keeping only the principal axes that have relatively large singular values removes any linear dependence that otherwise would cause problems in linear regression.

Reducing the dimension from d to s allows the data to be compressed because it requires only s dimensions to store each data point instead of the original d dimensions. It can also substantially reduce computational complexity of other machine learning methods, since for many algorithms the temporal complexity is exponential in the dimension of the data (the curse of dimensionality).

PCA is also useful for noise reduction if the noise in the data is roughly the same in all directions. Especially when the variance in the noise is small compared to the variance in the principal axes of the data, the noise should not have much effect on the principal components. We can see this mathematically as follows. If $\mathbb{X}$ is corrupted to $\widetilde{\mathbb{X}} = \mathbb{X} + E$, where the noise $E \sim \mathscr{N}(0, \varepsilon^2 I)$, and if $\mathbb{X} = U\Sigma V^\mathsf{T}$ is the SVD of $\mathbb{X}$, then

$$\begin{aligned}
\mathbb{E}[V^\mathsf{T}\widetilde{\mathbb{X}}^\mathsf{T}\widetilde{\mathbb{X}}V] &= \mathbb{E}[V^\mathsf{T}(\mathbb{X}+E)^\mathsf{T}(\mathbb{X}+E)V] \\
&= \mathbb{E}[V^\mathsf{T}\mathbb{X}^\mathsf{T}\mathbb{X}V] + \mathbb{E}[E^\mathsf{T}E] \\
&= \Sigma^\mathsf{T}\Sigma + \varepsilon^2 I.
\end{aligned}$$

The transformation V orthonormally diagonalizes the covariance matrix of $\widetilde{\mathbb{X}}$, and, therefore, the SVD of $\widetilde{\mathbb{X}}$ is (expected to be) of the form $\widetilde{U}\widetilde{\Sigma}V^{\mathsf{T}}$, where

$$\widetilde{\Sigma} = \operatorname{diag}(\sqrt{\sigma_1^2 + \varepsilon^2}, \ldots, \sqrt{\sigma_d^2 + \varepsilon^2}),$$

and where we set $\sigma_i = 0$ if i is greater than the rank of $\mathbb{X}$. This means that the expected values of the principal axes of $\widetilde{\mathbb{X}}$ are the same as those of $\mathbb{X}$. When we say *signal* here we mean the variance of the original uncorrupted data, that is the sum $\sum_{i=1}^{r} \sigma_i^2$. The noise can be measured as the sum of all the squared errors $\sum_{i=1}^{d} \varepsilon^2$ (It's important to square the ε_i because error needs to be measured in the same units as the signal). The expected value of the signal-to-noise ratio of $\widetilde{\mathbb{X}}$ is

$$\operatorname{SNR}(\widetilde{\mathbb{X}}) = \frac{\sum_{i=1}^{d} \sigma_i^2}{\sum_{i=1}^{d} \varepsilon^2} = \frac{1}{d\varepsilon^2} \sum_{i=1}^{d} \sigma_i^2. \tag{16.7}$$

An orthonormal projection of the data to the first s principal axes gives a signal-to-noise ratio of

$$\operatorname{SNR}(\widetilde{\mathbb{X}}_s) = \frac{\sum_{i=1}^{s} \sigma_i^2}{\sum_{i=1}^{s} \varepsilon^2} = \frac{1}{s\varepsilon^2} \sum_{i=1}^{s} \sigma_i^2. \tag{16.8}$$

It is straightforward to verify that (16.8) is greater than or equal to (16.7), and the inequality is strict unless all the singular values of $\mathbb{X}$ are equal (see Exercise 16.7).

16.2.3 Pitfall: Inconsistent Units

If some of the features are measured in different units than others, then changing the units for one feature can change the size of the singular values of $\mathbb{X}$, which changes the directions with greatest covariance and singular values.

Example 16.2.3. Consider the situation where the data consist of pairs $\mathbf{x}_i = (x_{i1}, x_{i2}) \in \mathbb{R}^2$, where x_{i1} is measured in meters and x_{i2} is measured in minutes, and these measurements are (empirically) independent, so that the sample covariance $\operatorname{Cov} = \frac{1}{n-1}\mathbb{X}^{\mathsf{T}}\mathbb{X}$ is diagonal of the form $\operatorname{Cov} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$. Changing the units of time for each x_{i2} from minutes to seconds changes each x_{i2} to $60x_{i2}$ and changes σ_2^2 to $3600\sigma_2^2$. Similarly, changing the units of x_{i1} to centimeters changes σ_1^2 to $10^4\sigma_1^2$. Since the choice of units is arbitrary, there should be no significance to the ordering $\sigma_1 \geq \sigma_2$.

Since the arbitrary choice of units changes which directions have the greater singular values, but discarding some directions based on an arbitrary choice of units is not a good idea. One way to try to address the problem of differing units is to normalize by dividing each feature by the corresponding sample standard deviation (remember that we have already centered the data to have its mean at the origin). Thus we map the kth coordinate x_k of each $\mathbf{x}$ to $\frac{x_k}{s_k}$ where $s_k = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} x_{ik}^2}$ is the sample standard deviation. Since each s_k is measured in the same units as x_k, the rescaled features are unitless. The covariance matrix of the normalized data is called the *correlation matrix*.

But normalizing is not necessarily a good idea if the features are already all measured in the same units and are naturally comparable. For example, when the features are all independent, normalizing just makes the covariance matrix into the identity matrix; and all the components have the same variance, which means all the components have equal claim to being "principal."

Example 16.2.4. Here are some examples of data sets and a discussion of which ones normalization with PCA might be useful for:

(i) Photographs and other images. Normalization is not usually needed or wanted for images, since pixel values are all naturally comparable and measured in the same units. It can be useful to rescale all the pixels by a common value (for example the total variance of all the pixels), but dividing each pixel by its individual variance destroys much of the benefit of using PCA.

(ii) The *Pima diabetes* dataset. The features include skin thickness, Body Mass Index (BMI), and number of pregnancies. These have different units of measurement: skin thickness could be measured in millimeters or centimeters), whereas BMI is measured in kilograms per square meter. And they are clearly not comparable in any natural way. Therefore, PCA is not meaningful on this data set without normalization.

(iii) The *California housing* database. Features like latitude, median income in the neighborhood, and age of house are all measured in different units, so this requires normalization.

(iv) The *Iris* dataset. This is more subtle. The measurements are all taken in the same units, but length of a petal is naturally greater than the width, and hence we should expect more variance in the length than the in width of a petal. And yet, we have no clear reason to believe that the length measurement is more meaningful or more important for classification than the width is. So, if you are doing PCA, normalization here may still be a good idea.

Remark 16.2.5. If the data are not approximately normal, then the mean and covariance probably do not describe the distribution very well, so it may be useful to apply a transformation to the data to make it more approximately normal.

16.2.4 How Many Components?

When doing PCA, how many components should be kept? This depends on a variety of factors, including the needs of the user and the computational resources available. One approach to this question is to find the minimum number of components necessary to account for a given percentage of the total variability. That means for a percentage p, we look for the smallest s such that $\sum_{i=1}^{s} \sigma_i^2 \geq p \sum_{i=1}^{d} \sigma_i^2$.

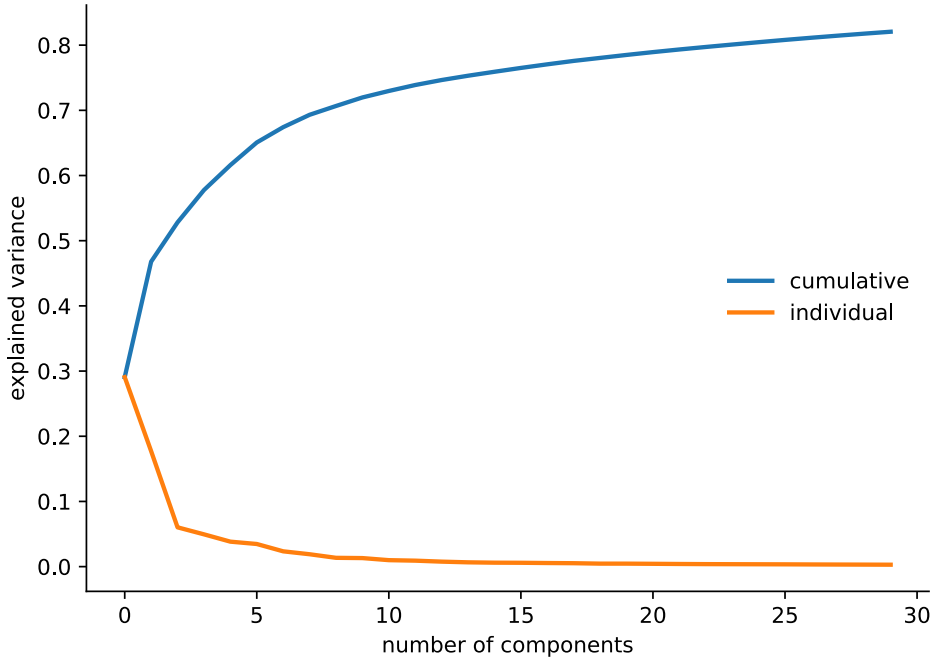


Figure 16.6: A plot of the percentage of variance explained in the Fashion-MNIST dataset as a function of the principal components. The orange plot is a plot of the amount of variance explained by each individual principal component (a scree plot). This shows that a lot of the variance can be explained with fewer than 4 dimensions, but thereafter no additional dimension contributes much. The blue is a plot of the cumulative variance explained by all the principal components up to the one being plotted.

Another approach is to plot the percentage of variability explained for each principle component (for example, the orange plot in Figure 16.6). In some situations the plot starts high and at some point drops off quickly, forming a sort of cliff, after which the remaining components contribute very little. This is sometimes called a *scree plot* because the remaining components at the bottom of the cliff are sometimes reminiscent of scree. It is common to keep the principal components that describe everything until the bottom of the cliff, but to discard the scree. This might not bring the total variance to a particular level, but none of the the subsequent principal components contribute much to the total, so the marginal benefit of adding more components is low.

16.2.5 Multiple Correspondence Analysis

If a feature is categorical rather than continuous, then there is no reason to expect that PCA would work at all, since there is no interpretation of the categorical variables in terms of size. Assume that the data are binary and are given as a matrix Z with the ith row $\mathbf{z}_i$ corresponding to the ith individual data point, with $z_{ik} \in \{0, 1\}$ for all i and k. Here each column (feature) takes the value 1 or 0, depending on whether the data point has the corresponding feature or not.

In this situation, we can transform the binary data as follows. For each k, let $p_k = \frac{1}{n}\sum_{i=1}^{n} z_{ik}$ be the relative frequency of feature k. The idea of multiple correspondence analysis (MCA) is to divide each z_{ik} by the weight p_k; that is, replace Z by the matrix with ik-entry $\frac{z_{ik}}{p_k}$. The effect of this is to give more weight to feature k if few individuals have feature k, and less weight if many individuals have that feature. It is easy to check that the sample mean of the rescaled data is $(1, 1, \ldots, 1)$. Recentering to put the mean at the origin gives a new data matrix $\mathbb{X} = [x_{ik}]$ with $x_{ik} = \frac{z_{ik}}{p_k} - 1$ for each i and k. Applying the usual PCA (not normalized) to the data $\mathbb{X}$ gives what is called *multiple correspondence analysis*.

16.3 Randomized Dimension Reduction

The SVD computes a linear projection to lower-dimensional subspaces in a way that preserves as much of the covariance as possible, but it is not cheap to compute when the original dimensions are large. For a matrix of shape n, d the SVD has a temporal complexity of roughly $O(dn \min(d, n))$. But, remarkably, we can often get excellent behavior from projections chosen completely at random, and these are much cheaper to compute than the SVD.

In this section we prove the Johnson–Lindenstrauss theorem, which can be used to show that random projections to a subspace have a high probability of approximately preserving the distance between pairs of points in the data set if the dimension of the subspace is not too small. Moreover, the minimum dimension of the target subspace depends only on the number of points in the data set and how much error you can tolerate. It is independent of the dimension of the space in which the points were originally located. For many applications this gives a very fast way to reduce the dimension of high-dimensional data—just choose a random projection to a lower-dimensional subspace.

Randomized dimension reduction can be very useful in many contexts, including latent semantic indexing, combinatorial optimization, and computing nearest neighbors. It can even be used to construct an inexpensive approximation to the SVD, as we discuss in Section 16.3.4.

16.3.1 Johnson-Lindenstrauss

Given a desired dimension k, the SVD (or PCA) gives what might be called an ideal linear projection to a subspace of dimension k, at least in terms of average behavior—maximizing the variance explained by the k dimensions.

But in many settings we want a map to a lower dimension that will not just have good average behavior, but will approximately preserve the distance between any two points in a given data set. This is important for applications such as clustering, finding nearest neighbors, and learning mixtures of Gaussians, among others. But, of course, if the target dimension is too small, this usually doesn't work.

Therefore, an important question is what is the smallest k for which this can be done? Moreover, given such a k, can we find the promised map? The *Johnson-Lindenstrauss* Theorem answers these two questions and it does so in a wonderful way—by showing that the probability is not zero that a random projection has the desired properties, and thus there must be at least one such projection. We can further leverage this result to find bounds that make the probability as large as we want that a random projection will have the desired properties; see Proposition 16.3.8.

Theorem 16.3.1 (Johnson-Lindenstrauss). *For any collection*

$$\mathbf{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^d,$$

for any $\varepsilon > 0$*, and for any*

$$k \geq \frac{24}{3\varepsilon^2 - 2\varepsilon^3} \log(N) \tag{16.9}$$

there exists a linear transformation $f : \mathbb{R}^d \to \mathbb{R}^k$ *such that*

$$(1-\varepsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2 \leq \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|^2 \leq (1+\varepsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2 \tag{16.10}$$

for every pair of points $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}$.

The dimension k of the target space depends only on the amount ε of allowed distortion and on the number N of the points. It does not depend on the dimension d of the original space.

Example 16.3.2. Unfortunately, the bound (16.9) is not nearly as small as we might like. For example, if we want the distortion in the distance to be no more than $\varepsilon = 10\%$, then we need

$$k \geq= \frac{12000}{14} \log(N) = 857.143 \log(N).$$

Thus, if $N = 3$, we need 942 dimensions to guarantee that the distance between any two of the three points is not distorted by more than a factor of 0.1. And if $N = 10$, we need $k \geq 1974$.

This is, of course, not helpful if the points are already embedded in a space of smaller dimension. In that case don't project at all—leave the points where they are.

But this bound is independent of the original dimension. Thus, if 10 points are embedded in a space of dimension 10^6, 10^{10}, or even 10^{100}, we can project down to a space of dimension only 1,974 with at most a 10% distortion in the distances between every pair of points. Moreover, the minimal dimension grows logarithmically in the number of points, so squaring the number of points to 100 only doubles the minimal dimension to 3,948.

If you are willing to put up with a greater distortion, then the Johnson-Lindenstrauss bound is much better. For example, with an error of $\varepsilon \geq 50\%$, the bound is only $48\log(N)$.

The idea of the proof of the Johnson-Lindenstrauss theorem (Theorem 16.3.1) is to show that the probability that a random linear transformation fails to satisfy the conclusion of the theorem is strictly less than one, which means there must be at least one random linear transformation that does satisfy the conclusion of the theorem. This proof technique is called the *probabilistic method*.

The proof depends on several lemmata.

Lemma 16.3.3. *If $w \sim \mathcal{N}(0,1)$, then the following holds:*

$$\mathbb{E}\left[e^{\lambda w^2}\right] = (1-2\lambda)^{-1/2}$$

for all $0 \leq \lambda < \frac{1}{2}$.

Proof. This is the moment generating function[48] of a χ_1^2 random variable (χ_ν^2 is another name for the distribution Gamma$(\frac{\nu}{2}, \frac{1}{2})$). The proof is Exercise 16.12. □

Lemma 16.3.4. *Assume $\mathbf{u} \in \mathbb{R}^d$ and the entries of $A \in M_{k\times d}$ are i.i.d. drawn from $\mathcal{N}(0, 1/k)$, let $\mathbf{w} = \frac{\sqrt{k}}{\|\mathbf{u}\|} A\mathbf{u}$, so $\|\mathbf{w}\|_2^2 = \frac{k\|A\mathbf{u}\|_2^2}{\|\mathbf{u}\|_2^2}$. The coordinates w_i of $\mathbf{w}$ are i.i.d. and distributed as $\mathcal{N}(0,1)$.*

Proof. Let B be the matrix $\sqrt{k}A$ and let $\mathbf{v} = \frac{\mathbf{u}}{\|\mathbf{u}\|}$, so that $\mathbf{w} = B\mathbf{v}$. Denote the ith column of B as $\mathbf{b}_i$. The $\mathbf{b}_i$s are i.i.d. and distributed as $\mathcal{N}(\mathbf{0}, I)$. A linear combination of normal random variables is again normal, so $\mathbf{w}$ is normal, and we need only compute the expected value and the covariance of $\mathbf{w}$. Prop **??** shows the w_is are independent, so the covariance matrix of $\mathbf{w}$ is diagonal. The computation of the mean and variance of each w_i is Exercise 16.13. □

Lemma 16.3.5. *Given the same assumptions as the previous lemma, if* (16.9) *also holds, then the following bound on the probability of distortion holds:*

$$P\left(\|A\mathbf{u}\|_2^2 \geq (1+\varepsilon)\|\mathbf{u}\|_2^2\right) < N^{-2}. \tag{16.11}$$

[48] See Remark 6.2.4 about moment generating functions.

Proof. Let $\mathbf{w} = \frac{\sqrt{k}}{\|\mathbf{u}\|} A\mathbf{u}$, so $\|\mathbf{w}\|_2^2 = \frac{k\|A\mathbf{u}\|_2^2}{\|\mathbf{u}\|_2^2}$. Lemma 16.3.4 shows that the coordinates w_i of $\mathbf{w}$ are i.i.d. and distributed as $\mathscr{N}(0,1)$. We have

$$\begin{aligned} P\left(\|A\mathbf{u}\|_2^2 \geq (1+\varepsilon)\|\mathbf{u}\|_2^2\right) &= P\left(\frac{1}{k}\|\mathbf{u}\|_2^2\|\mathbf{w}\|_2^2 \geq (1+\varepsilon)\|\mathbf{u}\|_2^2\right) \\ &= P\left(\|\mathbf{w}\|_2^2 \geq (1+\varepsilon)k\right) \\ &= P\left(e^{\lambda\|\mathbf{w}\|_2^2} \geq e^{\lambda(1+\varepsilon)k}\right) \qquad \forall \lambda > 0 \end{aligned}$$

Markov's inequality applied to the last line implies that

$$\begin{aligned} P\left(\|A\mathbf{u}\|_2^2 \geq (1+\varepsilon)\|\mathbf{u}\|_2^2\right) &= P\left(e^{\lambda\|\mathbf{w}\|_2^2} \geq e^{\lambda(1+\varepsilon)k}\right) \\ &\leq \frac{\mathbb{E}\left[e^{\lambda\|\mathbf{w}\|_2^2}\right]}{e^{\lambda(1+\varepsilon)k}} \\ &= \frac{\prod_{i=1}^k \mathbb{E}\left[e^{\lambda w_i^2}\right]}{e^{\lambda(1+\varepsilon)k}} \qquad \text{(Because the } w_i \text{ are i.i.d..)} \\ &= \left(\frac{\mathbb{E}\left[e^{\lambda w^2}\right]}{e^{\lambda(1+\varepsilon)}}\right)^k \qquad \text{(For } w \sim \mathscr{N}(0,1).) \\ &= \left(\frac{1}{\sqrt{1-2\lambda}\, e^{\lambda(1+\varepsilon)}}\right)^k \qquad \forall\ 0 < \lambda < \frac{1}{2}. \end{aligned}$$

Here the last line follows from Lemma 16.3.3.

Setting $\lambda = \frac{\varepsilon}{2(1+\varepsilon)}$ in the previous computation gives

$$P(\|A\mathbf{u}\|_2^2 \geq (1+\varepsilon)\|\mathbf{u}\|_2^2) \leq \left(\frac{1}{\sqrt{1-\frac{\varepsilon}{1+\varepsilon}}\, e^{\frac{\varepsilon}{2}}}\right)^k = \left((1+\varepsilon)e^{-\varepsilon}\right)^{k/2}. \tag{16.12}$$

We would like to bound the term $1+\varepsilon$ on the right side of (16.12) in a way that combines nicely with the expression $e^{-\varepsilon}$. To do this, consider the Taylor expansion of $\log(1+\varepsilon)$ around $\varepsilon = 0$, which is an alternating series for $\varepsilon \in (0,1]$. Truncating the series to a degree-three polynomial gives a strict upper bound

$$\log(1+\varepsilon) < \varepsilon - \frac{\varepsilon^2}{2} + \frac{\varepsilon^3}{3}.$$

Combining this inequality with the expression $(1+\varepsilon) = e^{\log(1+\varepsilon)}$ and substituting into (16.12) gives

$$P(\|A\mathbf{u}\|_2^2 \geq (1+\varepsilon)\|\mathbf{u}\|_2^2) \leq e^{-\left(\frac{\varepsilon^2}{2} - \frac{\varepsilon^3}{3}\right)k/2}, \tag{16.13}$$

and the bound (16.9) on k gives

$$P(\|A\mathbf{u}\|_2^2 \geq (1+\varepsilon)\|\mathbf{u}\|_2^2) \leq e^{-2\log(N)} = N^{-2}. \quad \square$$

An argument similar to that in the previous proof shows that

$$P\left(\|A\mathbf{u}\|_2^2 \leq (1-\varepsilon)\|\mathbf{u}\|_2^2\right) < N^{-2}. \tag{16.14}$$

Proof. (of Johnson-Lindenstrauss) Combining (16.11) and (16.14) shows that for any $\mathbf{u}$, if the entries of A are i.i.d. $\sim \mathcal{N}\left(0, \frac{1}{k}\right)$, then the probability of a distortion by A of more than $1 \pm \varepsilon$ is

$$P\left(\|A\mathbf{u}\|_2^2 \notin ((1-\varepsilon)\|\mathbf{u}\|_2^2, (1+\varepsilon)\|\mathbf{u}\|_2^2)\right) < \frac{2}{N^2} \tag{16.15}$$

Let E_{ij} denote the event that $\mathbf{u} = \mathbf{x}_i - \mathbf{x}_j$ is distorted by more than $1 \pm \varepsilon$, that is, the event that $\|A\mathbf{u}\|_2^2 \notin ((1-\varepsilon)\|\mathbf{u}\|_2^2, (1+\varepsilon)\|\mathbf{u}\|_2^2)$. Applying (16.15) to each pair we have

$$\begin{aligned} P\left(\bigcup_{i\neq j} E_{ij}\right) &\leq \sum_{i\neq j} P(E_{ij}) \\ &\leq \frac{N(N-1)}{2}\frac{2}{N^2} \\ &= 1 - \frac{1}{N} < 1. \end{aligned}$$

Since the probability of failure is not 1, there must be at least one A which does not fail. This is the desired linear transformation. □

Remark 16.3.6. This proof of Johnson-Lindenstrauss uses random linear transformations, but the original proof used orthogonal projections. Since random orthogonal projections are more costly to construct than random matrices, most people nowadays work with random matrices, but because the original maps were projections, it is still common to call the maps defined by random matrices *random projections*, despite the fact that they don't satisfy the usual property $P = P^2$ of a linear projection.

Remark 16.3.7. Computing the transformation $A\mathbf{x}_i$ for each $\mathbf{x}_i$ has a temporal cost of $O(d \log N)$ for fixed ε. Alon [**?**] shows that k cannot be further reduced in size by any significant amount. But Achlioptas [**?**] has shown that we can use a sparse matrix A, with the density reduced by a constant factor and the Gaussian distribution can be replaced by the simple distribution

$$A_{ij} = \sqrt{3}\begin{cases} +1 & \text{with probability } \frac{1}{6} \\ -1 & \text{with probability } \frac{1}{6} \\ 0 & \text{with probability } \frac{2}{3}. \end{cases} \tag{16.16}$$

This has the advantage of being sparse, which reduces the cost of the matrix multiplication, and also of allowing integer arithmetic if the data $\mathbf{x}_i \in \mathbf{D}$ all have integer coordinates.

16.3.2 Improving the Probability

The basic Johnson-Lindenstrauss theorem shows that there exists some random matrix that satisfies the condition (16.10) and the proof showed that probability of failure is no more than $1 - 1/N$. So if we try T different random matrices, the probability that they all fail is $(1 - \frac{1}{N})^T < e^{-T/N}$ (this last bound follows from the Taylor expansion of $\log(1 - x)$ around $x = 1$). This means that for any probability $\delta > 0$ we can ensure the probability of failure is less than δ by taking $T > -N \log(\delta)$.

But to verify that a given matrix A has the desired property requires computing the distances for all pairs, both before and after multiplying by A, this has a temporal cost of $O\left(\binom{N}{2}(d+k)\right) = O(N^2(d+k))$, which can be computationally expensive. Alternatively, we can increase the probability that one randomly chosen A satisfies the condition (16.10) by increasing k.

Proposition 16.3.8. *For any collection* $\mathbf{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^d$*, for any* $\varepsilon > 0$*, and for any* $\delta \in (0, 1)$ *if*

$$k \geq (2\log(N) - \log(\delta))\frac{12}{3\varepsilon^2 - 2\varepsilon^3}, \tag{16.17}$$

then the probability is at least $1 - \delta$ *that a random linear transformation* $f : \mathbb{R}^d \to \mathbb{R}^k$*, with i.i.d. entries drawn from* $\mathcal{N}(0, \frac{1}{k})$ *satisfies* (16.10) *for every pair of points* $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}$.

Proof. This is a computation using (16.13) and (16.17), similar to the proof of the Johnson-Lindenstrauss theorem. See Exercise 16.11. □

Proposition 16.3.8 is a powerful tool because for a given number N of points, the bound on k grows like $O(-\log(\delta))$, which makes it relatively easy to achieve a high probability that a random linear transformation will satisfy the condition (16.10).

Example 16.3.9. Proposition 16.3.8 shows that given $N = 20$ points in $\mathbb{R}^d$ for any d, to get a greater than 90% chance that a random $d \times k$ matrix maps those points into $\mathbb{R}^k$ with no more distortion than 1 ± 0.3, we need $k \geq 461$. To take that probability to 99%, we need $k \geq 589$, and to achieve 99.9%, we only need $k \geq 717$.

16.3.3 Applications of Random Projection Methods

Data in linguistic applications like latent semantic indexing (LSI) often live in extremely high dimensional spaces. The previous results can be immediately applied to reduce the dimension substantially with little loss of information.

In this section we briefly review a few other applications.

Rounding in Combinatorial Optimization

A standard approach to dealing with NP-hard combinatorial optimization problems is to construct an *approximation algorithm* that is not guaranteed to give the absolute optimal solution, but rather gives a solution that is only approximately optimal. In many settings an approximate optimum is completely satisfactory.

One approach to constructing approximation algorithms is to first relax the problem to one that is easier to solve, often with some random choices, and then find a *rounding*, which is a map from the set of feasible solutions of the easier problem to the set of solutions to the harder problem.

Example 16.3.10. The *max cut* problem on a graph $G = (V, E)$ is to color each vertex one of two colors (say red or blue) so as to maximize the number of edges that have red on one side and blue on the other (*cut* edges). This problem is NP hard. But coloring the vertices randomly gives a good approximation algorithm, with the expected number of cut edges being $\frac{1}{2}$ of all the edges. Thus this algorithm will, on average, do no worse than half as well as the absolute optimal result.

Goemans and Williamson constructed a much better approximation algorithms (0.87856 times the optimum) that uses random projections.

KD Trees and Nearest Neighbors

KD trees are binary rooted trees whose leaves contain points of $\mathbb{R}^d$ and whose other nodes correspond to a splitting of $\mathbb{R}^d$ into two half spaces, separated by a hyperplane of the form $x_i = c$. Just as a binary search tree (BST) makes searching easy in $\mathbb{R}$, a KD tree provides a way to efficiently search $\mathbb{R}^d$. For more on KD trees, see the *Nearest Neighbor Lab* in the Volume 2 Lab manual.

KD trees are useful in low dimensions for things like a nearest neighbors classifier because a KD tree can be built to partition space into cells that allow for faster searching for neighbors. But there are data sets in $\mathbb{R}^d$ which are difficult to split into an efficient search KD tree. Specifically, these data sets have the bad property that the corresponding KD tree has $O(2^d)$ nodes. Thus KD trees are subject to the curse of dimensionality. Even when the underlying data have much lower intrinsic dimension d', meaning that they lie on some manifold in $\mathbb{R}^d$ of dimension $d' \ll d$, KD trees still suffer from this factor of 2^d, rather than a smaller factor like $2^{d'}$.

In a KD tree, when splitting, the algorithm chooses the coordinate direction with the greatest distance between points, and then splits at the median point. Dasgupta and Freund show that if, instead, one chooses a random direction to split and then splits a random distance from the median point, the resulting *random projection tree* grows in complexity like $2^{d'}$, where d' is the instrinsic dimension, instead of growing like 2^d.

16.3.4 Randomized SVD

The cost of computing the SVD of an $n \times d$ matrix X using typical methods like those in the most common numerical linear algebra library LAPACK, is $O(dn \min(d, n))$, which may be prohibitive if d and n are both large.

The *randomized SVD (RSVD)* constructs a Gaussian random matrix G of size $m \times n$ for some $m < \min(n, d)$. We call $Y = GX$ the *sampling matrix*. In many cases the rows of the matrix Y form a good approximate basis for the row space of X and we can identify approximations to the singular vectors of X by orthonormalizing the columns of Y to find a basis for its column space and then project X to this subspace and use deterministic methods to compute a full SVD of the resulting small matrix.

Specifically, when seeking the best possible approximation to X of rank k we choose $m = k + p$ for some small p ($p = 5$ or $p = 10$ are recommended by the originators of the algorithm). The steps of the algorithm are

(i) Choose a Gaussian random matrix G of size $m \times n$ and compute the $m \times d$ matrix $Y = GX$.

(ii) Compute a QR decomposition $Y^\mathsf{T} = QR$, with Q of size $d \times m$.

(iii) Project X to the row space by computing the $n \times m$ matrix $B = XQ$.

(iv) Compute the compact SVD $B = U_1 \Sigma_1 \tilde{V}_1^\mathsf{T}$ and define the $d \times m$ matrix $V_1 = Q\tilde{V}_1$.

We now have an approximate factorization

$$X \approx XQQ^\mathsf{T} = U_1 \Sigma_1 V_1^\mathsf{T}.$$

Here Σ_1 gives aproximate singluar values of X.

The cost of computing the RSVD is $O(dnm)$ and uses the (potentially very large) matrix X only twice—once for computing Y and once for computing B. It can be shown that the RSVD is accurate when the singular values of X decay reasonably rapidly.

16.4 Nonlinear Dimension Reduction

All of the dimension reduction techniques discussed so far have been linear, meaning that we use a linear transformation to map the data to a lower-dimensional space. But sometimes the data lie on a set, often a manifold, of lower dimension embedded in a high dimensional space.

Example 16.4.1. The data set called the *swiss roll* depicted in Figure 16.7 is an example of data that lie on a two-dimensional manifold embedded in three space. Points on different layers of the roll could be close together in three-space but probably should not be thought of as being close together—really we should measure distance along the roll, instead.

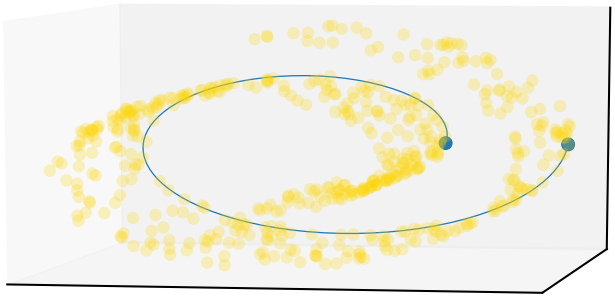


Figure 16.7: A swiss roll data set. Points (blue) that are near in three space are far apart if we measure distance along the roll. A linear dimension reduction method like PCA cannot capture this subtlety.

There are many methods for dimension reduction that try to identify a lower-dimensional manifold on which the data lie. In this section we discuss the idea of probabilistic dimension reduction and describe one algorithm called *t-Distributed Stochastic Neighbor Embedding (t-SNE)*. In Section 16.4.6 we describe another, more efficient, dimension reduction algorithm called *Uniform Manifold Approximation and Projection (UMAP)*.

16.4.1 Probabilistic Dimension Reduction

Given data $\mathbf{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ in $\mathbb{R}^d$ for large d, the idea is to start with a guess of a map $f_0 : \mathbf{D} \to \mathbb{R}^s$ for some $s \ll d$ (it is common to use $s = 2$, which makes it very easy to visualize the resulting data set). Set $\mathbf{y}_i = f(\mathbf{x}_i)$ for all $i \in \{1, \ldots, N\}$. Now put a distribution p_{ij} on the set of pairs $\{(i,j) \,|\, 1 \leq i \neq j \leq N\}$ with a pair (i,j) being more likely than another pair (i',j') if $\|\mathbf{x}_i - \mathbf{x}_j\|_2 < \|\mathbf{x}_{i'} - \mathbf{x}_{j'}\|_2$ in $\mathbb{R}^d$. One can think of p_{ij} as being the probability that $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbors.

Now put another distribution q_{ij} on the same set of pairs, but now make (i,j) more likely than (i',j') if $\|\mathbf{y}_i - \mathbf{y}_j\|_2 < \|\mathbf{y}_{i'} - \mathbf{y}_{j'}\|_2$ in $\mathbb{R}^s$. One can think of q_{ij} as the probability that $\mathbf{y}_i$ and $\mathbf{y}_j$ are neighbors in the target space $\mathbb{R}^s$.

The similarity between the two distributions can be measured by the KL-divergence

$$\mathscr{D}_{\mathrm{KL}}(p\|q) = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right).$$

The desired embedding of $\mathbf{D}$ in $\mathbb{R}^s$ is given by

$$\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_N = \underset{\mathbf{y}_1,\ldots,\mathbf{y}_N}{\operatorname{argmin}} \mathscr{D}_{\mathrm{KL}}(p\|q). \tag{16.18}$$

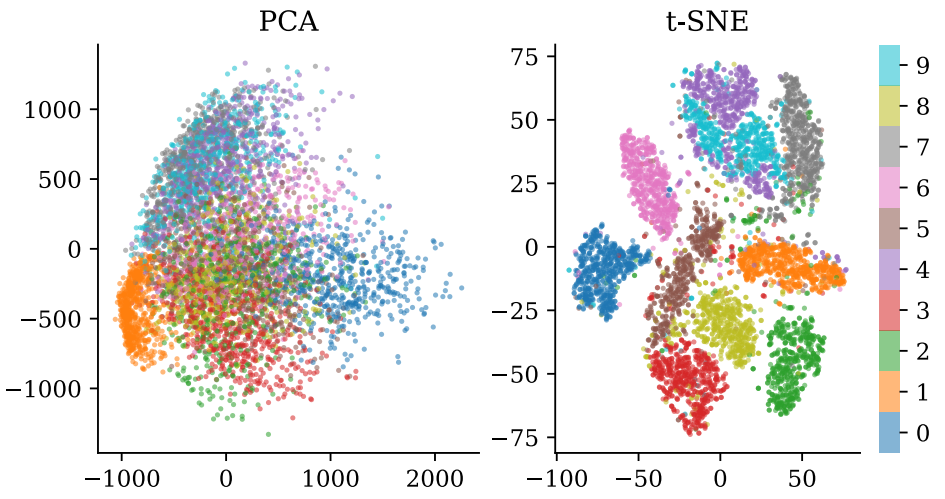


Figure 16.8: t-SNE and PCA mappings of a subset of the MNIST data set into $\mathbb{R}^2$. Note that the two-dimensional PCA projection does not separate the different digits very well, while t-SNE separates them better.

We can now use standard numerical optimization methods (usually a variant of gradient descent) to find the optimal values of the $\mathbf{y}_i$.

16.4.2 t-SNE

The t-SNE algorithm uses the probabilistic model described above, with the following distributions.

For the distribution p_{ij}, let $p_{j\,|\,i}$ be the conditional probability density that $\mathbf{x}_i$ would pick $\mathbf{x}_j$ as its neighbor from among the points in $\mathbf{D}$ if neighbors were picked in proportion to their probability density under a Gaussian centered at $\mathbf{x}_i$ with covariance $\Sigma_i = \sigma_i^2 I$ for some choice of σ_i^2. That is, set

$$\begin{aligned} p_{j\,|\,i} &= \frac{P(\mathbf{x}_j \,|\, \boldsymbol{\mu} = \mathbf{x}_i, \Sigma_i)}{\sum_{k \neq i} P(\mathbf{x}_k \,|\, \boldsymbol{\mu} = \mathbf{x}_i, \Sigma_i)} \qquad \forall j \neq i \\ &= \frac{\exp\left(-\frac{\|\mathbf{x}_j - \mathbf{x}_i\|_2^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|\mathbf{x}_k - \mathbf{x}_i\|_2^2}{2\sigma_i^2}\right)} \qquad \forall j \neq i. \end{aligned} \tag{16.19}$$

For nearby datapoints $p_{j\,|\,i}$ is relatively high, whereas for widely separated data points $p_{j\,|\,i}$ is very small. The choice of σ^2 determines just how far apart two points must be to be "widely separated." Note that these conditional probabilities are not symmetric: $p_{i\,|\,j} \neq p_{j\,|\,i}$. We make a symmetric distribution on the set of all pairs by setting

$$p_{ij} = \frac{p_{i\,|\,j} + p_{j\,|\,i}}{2N}.$$

The distribution q_{ij} corresponding to the lower-dimensional points $\mathbf{y}_i, \mathbf{y}_j$ is similar to the previous distribution except that we use a Student t-distribution[49] with one degree of freedom (the Cauchy distribution) on $\mathbb{R}^s$ instead of a normal distribution:

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq i}(1 + \|\mathbf{y}_i - \mathbf{y}_k\|^2)^{-1}}.$$

The t in *t-SNE* refers to this t-distribution.

One reason for using the fat-tailed t-distribution instead of a normal distribution for q_{ij} is computational. The computation using the t-distribution is less expensive than a Gaussian in two ways. First, the evaluation of the probability q_{ij} is less expensive than a Gaussian would be because it involves only rational functions and no exponentials. And second, the optimization problem turns out to be easier to solve.

But a more significant reason for using the t-distribution is that it makes it easier for points that are somewhat separated in $\mathbb{R}^d$ to be mapped farther apart in low-dimensional space. To see that this is necessary, consider again the Swiss roll data set. A good mapping from $\mathbb{R}^3$ to $\mathbb{R}^1$ would unroll the data into a line, and thus points that are not too far apart but lie in different layers in the Swiss roll in $\mathbb{R}^3$ would need to map to points that are very far apart in $\mathbb{R}^1$.

When the KL-divergence of the distributions is minimized, the p_{ij} are, on average, close to their counterparts q_{ij}. The Student t-distribution has much fatter tails than the normal distribution, meaning that for $x, y \geq 2$ if $f_{\mathcal{N}(0,1)}(x) = f_{\mathcal{T}(1)}(y)$, then $y \gg x$; see Figure 16.9. Using a fatter-tailed distribution for the low-dimensional points $\mathbf{y}_i$ than we use for the high-dimensional points $\mathbf{x}_i$, means that the distance $\|\mathbf{y}_i - \mathbf{y}_j\|^2$ is generally much greater than the distance $\|\mathbf{x}_i - \mathbf{x}_j\|^2$.

The result of running t-SNE is a representation of d-dimensional data in $\mathbb{R}^s$, where points that are very close in $\mathbb{R}^d$ map to points that are close in $\mathbb{R}^s$, but points that are mid-distance and farther away in $\mathbb{R}^d$ may be mapped very far from each other in $\mathbb{R}^s$. This can be very useful as a preconditioner for a variety of other machine-learning tasks and also useful for visualization in the special case that $s = 2$.

16.4.3 t-SNE Variance and Perplexity

Constructing the distribution p_{ij} requires the choice of variance σ_i^2 for each i. Each choice of σ_i^2 gives a distribution $P_i(\mathbf{x}_j) = p_{i|j}$ on the set of all the points in $\mathbf{D}$.

Recall that the entropy $H(P) = -\sum_j P(\mathbf{x}_j) \log_2 P(\mathbf{x}_j)$ measures the average number of bits required to encode the information scheme of the the distribution P on the points of $\mathbf{D}$. It is also the expected information gain from learning that an event $\mathbf{x} \in \mathbf{D}$ has occurred. For the distribution P_i the *perplexity* of the distribution is

$$\text{perplexity}(P_i) = 2^{H(P_i)},$$

[49] See the desciption of Student's t-distribution and the Cauchy distribution in Section **??**.

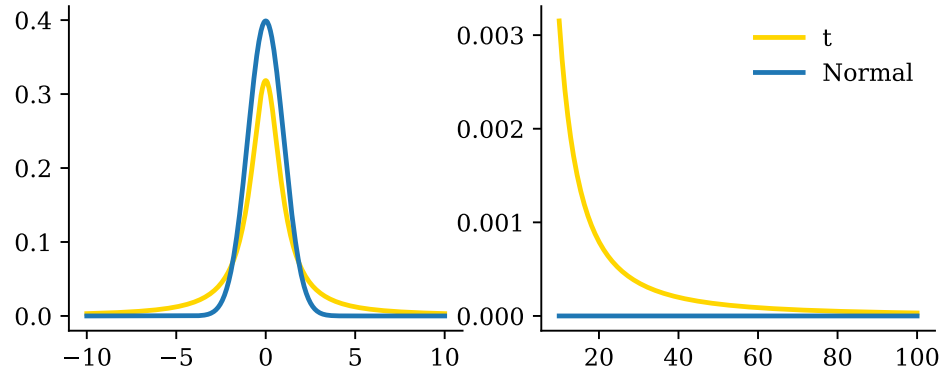


Figure 16.9: The t-distribution (yellow) with one degree of freedom (also called the Cauchy distribution) has much fatter tails than the normal distribution (blue). This means that once $x, y \geq 2$ if $f_{\mathcal{N}(0,1)}(x) = f_{\mathcal{T}(1)}(y)$, then $y \gg x$. This means that t-SNE takes points $\mathbf{x}_i$ and $\mathbf{x}_j$ that are only a moderate distance apart in the high-dimensional space $\mathbb{R}^d$ and maps them to lower-dimensional points $\mathbf{y}_i$ and $\mathbf{y}_j$ that are much farther apart in $\mathbb{R}^s$.

and it can be interpreted as a smooth approximation to the number of neighbors of $\mathbf{x}_i$. Roughly speaking, if we want t-SNE to map approximately the k nearest neighbors of $\mathbf{x}_i$ to near neighbors of $\mathbf{y}_i$, then we should choose σ_i^2 to make the perplexity equal to k. For a given perplexity k, it is not easy to find a closed form expression for the desired value of σ_i^2, but it can be solved as a straightforward one-dimensional numerical rootfinding problem.

The choice k for the perplexity should increase as the number N of data points increases, but it should usually be much smalthan N. The authors of t-SNE recommend choosing the perplexity in the range $[5, 50]$, but you will need to experiment with different choices for your particular data sets. Changing the perplexity can have a big impact on the result of t-SNE.

16.4.4 t-SNE Computation

The standard way to find the minimizer (16.18) is by gradient descent with *momentum*, meaning that each step $\mathbf{d}_k$ is a combination of the negative gradient $-D^\mathsf{T}C(Y_{k-1})$ and the previous step $\mathbf{d}_{k-1}$:

$$\begin{aligned} Y_k &= Y_{k-1} + \mathbf{d}_k \\ \mathbf{d}_k &= -(1-\beta)D^\mathsf{T}C(Y_{k-1}) + \beta\mathbf{d}_{k-1} \end{aligned}$$

for some choice of $\beta \in [0, 1)$. This implies

$$\mathbf{d}_k = -(1-\beta)(D^\mathsf{T}C(Y_{k-1}) - \beta D^\mathsf{T}C(Y_{k-2}) - \beta^2 D^\mathsf{T}C(Y_{k-3}) - \cdots).$$

Other iterative methods can also be used. Of course this is not necessarily a convex optimization problem, so numerical optimization methods generally converge to a local minimizer that is not a global minimizer. Nevertheless, this local minimizer is often good enough.

Proposition 16.4.2. *The derivative of the cost function*

$$C(\mathbf{y}_1, \dots, \mathbf{y}_N) = \mathscr{D}_{\mathrm{KL}}(p \| q)$$

is

$$D_{\mathbf{y}_i} C = 4 \sum_{j \neq i} \frac{p_{ij} - q_{ij}}{1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2} (\mathbf{y}_i - \mathbf{y}_j). \tag{16.20}$$

Proof. The proof is a tedious exercise in multivariate differentiation requiring no special tricks but a lot of unpleasant bookkeeping. For details see [vdMH08, Appendix A]. □

Note that the terms p_{ij} remain the same for all values of $\mathbf{y}_i, \mathbf{y}_j$, so they need only be computed once. The naïve computation of t-SNE has computational complexity $O(N^2)$, but additional improvements like the *Barnes-Hut* algorithm allow it to be computed in much less time.

Remark 16.4.3. Note that the gradient (16.20) is a sum of terms of the form $\mathbf{y}_i - \mathbf{y}_j$. Each of those terms can be thought of as applying a force between $\mathbf{y}_i$ and $\mathbf{y}_j$ that either pulls them closer or pushes they farther away. The strength of that force is small when p_{ij} and q_{ij} are similar in size or when $\mathbf{y}_i$ and $\mathbf{y}_j$ are very far apart, and it is greater when p_{ij} and q_{ij} are dissimilar. With each iteration, the system moves in the direction determined by the sum of all these forces. Sometimes this is visualized as putting a spring between every pair of points $\mathbf{y}_i, \mathbf{y}_j$, with the system moving to respond to the total of all the spring forces pulling or pushing.

16.4.5 Using t-SNE

t-SNE is very useful for visualizing data. Mapping the data to $\mathbb{R}^2$ using t-SNE gives easy-to-parse plots of high-dimensional data. t-SNE can also be used for clustering by first applying t-SNE and then applying standard clustering algorithms on the output of t-SNE. This has the advantage of significanly reducing the dimension of the data, which may be essential to run the clustering algorithms. But, of course, the computation of the t-SNE embedding itself may be expensive. Moreover, t-SNE might split some clusters as well as push points together that should not be clustered (just as the clustering algorithm itself might do).

One problem with t-SNE is that it is not well suited to handling new data points. Unlike the linear dimension reduction techniques we have used so far, t-SNE does not define a map from all of $\mathbb{R}^d$ to $\mathbb{R}^s$. It only gives a map from $\mathbf{D}$ to $\mathbb{R}^s$. This means that new data aren't naturally mapped to the target space $\mathbb{R}^s$. For large data sets the naïve approach of applying t-SNE to the new data set $\mathbf{D} \cup \{\mathbf{x}_{new}\}$ is generally not workable, since the algorithm is relatively expensive to run even once.

One way to try to use t-SNE in a way that allows new data to be incorporated is to fit a multivariate regression to predict the map from the pairs $(\mathbf{x}_i, \mathbf{y}_i)$, once the original t-SNE has been run on $\mathbf{D}$ to produce the $\mathbf{y}_i$.

16.4.6 UMAP

A more recent popular alternative to t-SNE is the *uniform manifold approximation and projection (UMAP)* algorithm. UMAP is usually explained and justified in terms of fancy algebraic topology, fuzzy sets, and category theory, but that is not at all necessary to understand it. In fact, it is very similar to t-SNE and does not depend on any of the fancy stuff. The algorithm constructs a weighted (undirected) graph where each vertex is an $\mathbf{x}_i \in \mathbf{D}$ and each edge from $\mathbf{x}_i$ to $\mathbf{x}_j$ is weighted with the probability that $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbors. This step is essentially the same as t-SNE, except that the distribution p_{ij} is built in a different manner.

Similarly, UMAP searches for an embedding of $\mathbf{D}$ in to $\mathbb{R}^s$ by assigning to each embedding $\{\mathbf{y}_1, \dots, \mathbf{y}_N\} \subset \mathbb{R}^s$ a weight q_{ij} to the pairs and then trying to minimize a cost function that measures how different the two distributions are.

UMAP begins with a user-chosen integer parameter k, corresponding to the number of neighbors to consider at each step. This is analogous to the choice of perplexity for t-SNE.

Assume we have a metric d on the set $\mathbf{D}$ giving a distance $d(\mathbf{x}_i, \mathbf{x}_j) > 0$ for every pair $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}$. For each i find $\mathbf{x}_i$'s k nearest neighbors $N_i = \{\mathbf{x}_{j_1}, \dots, \mathbf{x}_{j_k}\}$ and compute the distance to $\mathbf{x}_i$'s nearest neighbor

$$\rho_i = \min\{d(\mathbf{x}_i, \mathbf{x}_j) \mid \mathbf{x}_j \in N_i\}.$$

For each i and j we choose weights $p_{j|i}$ as follows. Set $p_{j|i} = 0$ if $j \notin N_i$, and for each $j \in N_i$ compute $p_{j|i}$ as

$$p_{j|i} = \exp\left(-\frac{d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i}{\sigma_i}\right),$$

where σ_i is chosen to make

$$\sum_{j \in N_i} \exp\left(-\frac{d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i}{\sigma_i}\right) = \log_2 k.$$

This does not make $p_{j|i}$ into a distribution, but that won't matter. As with t-SNE symmetrize the weights, but do it in the following (different) way

$$p_{ij} = p_{i|j} + p_{j|i} - (p_{i|j} p_{j|i}). \tag{16.21}$$

UMAP also chooses weights for pairs of embedded points $\mathbf{y}_i, \mathbf{y}_j \in \mathbb{R}^s$ in a way that is similar to t-SNE, without rescaling and with two extra hyperparameters a and b:

$$q_{ij} = \frac{1}{1 + a\|\mathbf{y}_i - \mathbf{y}_j\|^{2b}}.$$

Since p_{ij} and q_{ij} are not distributions, we can't compare them using KL-divergence, but we can use cross entropy

$$C_{\text{UMAP}}(\mathbf{y}_1, \dots, \mathbf{y}_N) = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right) + (1 - p_{ij}) \log\left(\frac{1 - p_{ij}}{1 - q_{ij}}\right).$$

The desired embedding is given by finding $\mathbf{y}_1, \dots, \mathbf{y}_N$ to minimize C_{UMAP}. The typical method to do this is again a variant of gradient descent.

There are several speed advantages of UMAP over t-SNE. First, there is a fast algorithm called *nearest neighbor descent* for finding the k nearest neighbors of every point. Empirically nearest neighbor descent has aymptotic complexity of about $O(N^{1.14})$. Once the nearest neighbors have been identified, we need only compute p_{ij} if i is among the k nearest neighbors of j or vice versa. UMAP also initializes using something called *spectral layout* that provides faster convergence and greater stability. A number of other careful implementation adjustements have also been made to UMAP that make the most common implementation of UMAP significantly faster than the default implementation of t-SNE.

16.5 Nonnegative Matrix Factorization

PCA corresponds to approximating the data matrix X by a low-rank product $WH = (U_s\Sigma_s)V_s^\mathsf{T}$, where the rows of $H = V_s^\mathsf{T}$ are a basis for an s-dimensional subspace of $\mathbb{R}^d$ in which the rows of X are almost contained, so projecting the rows of X onto that subspace loses very little information. The rows of $W = U_s\Sigma_s$ give the coefficients to express the rows of X in terms of this basis (formed from the rows of H).

If we have no restrictions on the forms of W and H other than a bound on the dimensions to be $n \times r$ and $r \times d$, respectively, then the Schmidt, Mirsky, Eckart–Young theorem (Volume 1, Thm. 4.6.3) guarantees that the best rank-r approximation of X is the one obtained by PCA. But in many cases we need more constraints on W and H. A common choice of constraint is that W and H should have no negative entries. This is called *nonnegative matrix factorization (NMF).*

16.5.1 Applications of Nonnegative Matrix Factorization

One big advantage of having W and H nonnegative is that it is often much easier to interpret the meaning of the factorization.

Example 16.5.1. In hyperspectral imaging instead of just measuring the intensity of red, green, and blue light at each pixel, the camera records the intensity of many (often 100 to 200) different wavelengths. In the final image each pixel of the image has many channels (features), each corresponding to the intensity of a different wavelength of light. If X is the data matrix where each row corresponds to one pixel and each column corresponds to a specific wavelength, then for a nonnegative matrix factorization $X \approx WH$ with $W \in M_{n\times r}$ and $H \in M_{r\times d}$, we can interpret each of the r rows of H as corresponding to some basic material occuring in the image, such as grass, asphalt, sand, seawater, or a certain species of plant. The rows of W correspond to pixels and the entries in each pixel's row are the (nonnegative) coefficients defining the pixel as a linear combination of the different materials. This allows one to draw the image in terms of each of the basic materials; so the columns of W correspond to (flattened) images showing only the relative quantities of the various materials, rather than the intensities of specific wavelengths. See Figure 16.10 for an example of this.

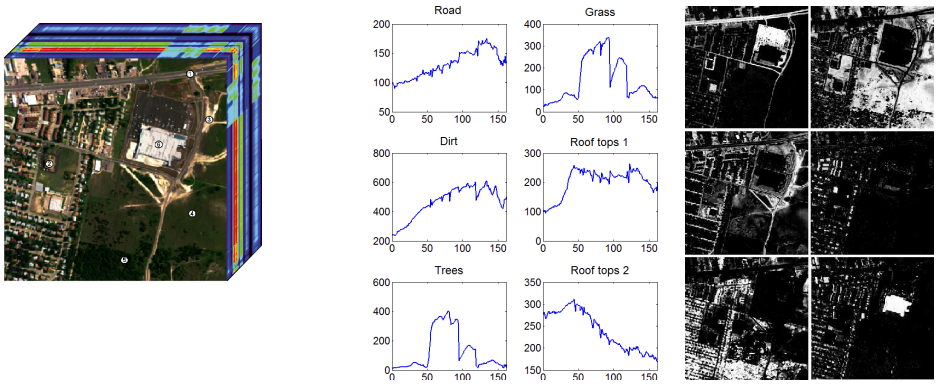


Figure 16.10: Decomposition of a hyperspectral image from `http://www.agc.army.mil/`, as described in Example 16.5.1. This decomposition consists of six materials ($r = 6$), where each row of the matrix H is the spectral signature of a material, while each row of the matrix W corresponds to one pixel and indicates how much of each material is present in that pixel. (Note that to obtain this decomposition, a sparse prior on the matrix W was used.) The images on the right are made by plotting how much of each material (white is more, black less) shows up in each pixel—so these images correspond to the columns of W. A careful look at these images suggests that we could label these materials as road, dirt, trees, grass, rooftop1, and rooftop2, respectively, but that labeling is done by humans, after the NMF is computed. It is not part of the NMF in any way, but once we have the NMF, we can make these identifications. If the rank r is chosen to be too small or too large, it might not be so easy to interpret the rows of W as materials in a meaningful way. This image and the decomposition are taken from [Gil14].

Example 16.5.2. NMF can be used for analysis and identification of faces. If each image corresponds to a $j \times k$ array of pixel intensities, then the data matrix is constructed by flattening the image into a vector of length jk and using these vectors as the rows of X, so X has dimensions $n \times jk$, where n is the number of images to analyze.

As with PCA, constructing the nonnegative matrix factorization $X \approx WH$ of rank r gives r rows of H which span a subspace that approximately contains the rows of X, and the rows of W give the coefficients that define the rows of X as positive linear combinations of the rows of H. Because of nonnegativity, the images consist only of nonnegative weighted combinations of these rows, so the rows of H tend to correspond to parts of faces (like eyes, noses, and mustaches) that can be assembled together to make faces. One advantage of this approach over the eigenfaces arising from PCA is that it is more robust to partial occlusion.

16.5.2 Computation of NMF

To construct a basic NMF for X, we solve the optimization problem

$$\begin{array}{ll} \underset{W,H}{\text{minimize}} & \|X - WH\| \\ \text{subject to} & W \in M_{n\times r}, \quad H \in M_{r\times d} \\ & W \succeq \mathbf{0}, \quad H \succeq \mathbf{0} \end{array} \tag{16.22}$$

A common choice of norm is the Frobenius norm $\|\cdot\|_F$, but other norms are also useful, including the earthmover distance, the L^1-norm, or even the relative entropy $\mathscr{D}_{\mathrm{KL}}$ (which, of course, is not a norm at all).

It has been shown that solving (16.22) exactly is NP-hard, but there are some numerical optimization methods that generally perform well.

The strategy is to use the fact that for a fixed W, finding the optimal H is a convex optimization problem (see Exercise 16.18), and similarly, for a fixed H finding the optimal W is convex. Therefore, we may start with initial values of H_0 and solve for the optimal W—call this W_1. Then fixing W_1, solve for the optimal H—call this H_1, and repeat the process until a satisfactory stopping criterion has been satisfied. The resulting sequence of objective values is always nonincreasing and is bounded below, so it must converge to some value (but not necessarily the absolute minimum, nor even a local minimim).

There are several implementations in Python that compute the NMF. A popular choice is the one built into `sklearn`.

There are a few issues to keep in mind when constructing an NMF, including the following:

(i) The problem is ill posed because there is not necessarily a unique global optimizer, and different optimizers can potentially give very different factorizations and hence different interpretations. The choice of initialization for the numerical optimization methods also plays a role in which factorization is found.

(ii) The choice of the rank r is not determined by the theory, but rather by the needs of the user, by consultation with experts in the domain of application, or by a search with cross validation, as described below in Section 16.5.5.

Unexample 16.5.3. The NMF is not necessarily unique. Here are two distinct nonnegative factorizations of one matrix

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

16.5.3 Regularization

It is usually a good idea to include a regularization term, penalizing the Frobenius norm or the L^1 norm (or both) of the factorization. To do this fix a regularization parameter $\alpha \geq 0$ and an "L^1 ratio" $\lambda \in [0,1]$ and solve the optimization problem

$$\begin{array}{ll} \underset{W,H}{\text{minimize}} & \|X - WH\| + \alpha[\lambda(\|(W)\|_{L^1} + \|(H)\|_{L^1}) + (1-\lambda)(\|W\|_F + \|H\|_F)] \\ \text{subject to} & W \in M_{n\times r}, \quad H \in M_{r\times d} \\ & W \succeq \mathbf{0}, \quad H \succeq \mathbf{0}. \end{array} \tag{16.23}$$

When $\lambda = 1$, the L^1 norm is penalized, but the Frobenius norm is not; this encourages sparsity in the factorization but permits some entries to be large. When $\lambda = 0$, the Frobenius norm is penalized, but the L^1 norm is not. This encourages smaller entries in the the factorization, but not necessarily zero entries.

A sparse factorization usually lends itself to interpretation more than a more dense factorization does, thus if interpretation is important, making the L^1 regularization penalty $\alpha\lambda$ large can be helpful.

Another benefit of the regularization term is that without it, the NMF problem is not convex, but including a regularization term can make it convex. The L^1 and L^2 regularization terms are both convex; so, when α is large enough, then the problem becomes a convex one, which not only makes it easier to solve, but also makes the minimizer unique.

16.5.4 Missing Data

In many applications the data matrix X has some entries missing. In this case the naïve NMF doesn't work, but this is easy to fix, as long as we are using the right norm in the objective in (16.22) or (16.23).

Example 16.5.4. For a movie recommender system one can let the rows of X correspond to users of a movie database, let the columns correspond to movies, and let the entry in row i and column j be the rating that user i gave to movie j.

If WH is an NMF of X, then the rows of H correspond to types of movies (perhaps, "Action," "Jane Austen adaptations," and "movies starring Brie Larsen") but most of the types are unlikely to be easily interpretable. The entries in each row of H correspond to the amount that each movie could be classified as belonging to that type. For each row of W corresponding to a given user, the entries in the row indicate how much each user's ratings are determined by the corresponding movie types.

But most users have not seen or rated every movie in the database, and therefore many rows have missing values. If we can make an NMF $X \approx WH$ then WH would have no missing values, even if entries are missing in X. The entry in the ij position of WH is could be considered a reasonable prediction of the rating that user i will give to movie j.

To predict a given person's rating of a given movie, simply find the entry of WH corresponding to the person and the movie of interest. But to do this we need to construct the NMF without knowing all the values of X.

Assume for now that we are using the Frobenius norm, so that

$$\|X - WH\|_F = \sum_{ij}((X - WH)_{ij})^2 \tag{16.24}$$

is the sum of the squares of each of the coordinates of the matrix $X - WH$. If the ij-entry X_{ij} is missing from X, then we simply drop the ij entry from the sum (16.24) in the objective function.

Example 16.5.5. To write the matrix $\left[\begin{smallmatrix} 1 & \text{NaN} \\ 2 & 3 \end{smallmatrix}\right]$ as $WH = \begin{bmatrix} w_1 & w_2 \end{bmatrix}\begin{bmatrix} h_1 & h_2 \end{bmatrix}^\mathsf{T}$ using the Frobenius norm, we would solve the optimization problem

$$\begin{aligned} \underset{W,H}{\text{minimize}} \quad & (1 - w_1h_1)^2 + (2 - w_2h_1)^2 + (3 - w_2h_2)^2 \\ \text{subject to} \quad & w_1, w_2, h_1, h_2 \geq 0. \end{aligned}$$

Of course, the final product WH is not equal to X, because it has no missing values.

Equivalently, if X has dimension $n \times d$, then we can make another matrix M of the same dimension with entries consisting only of 0 or 1, with a 1 in every position where the corresponding entry of X is known and a 0 in every position where an entry of X is unknown. In the objective in (16.22) or (16.23) we can then replace the term $\|X - WH\|$ with

$$\|M \odot (X - WH)\|,$$

where $\odot$ is the Hadamard (entrywise) product. This gives an objective function that does not depend on the missing information.

16.5.5 Cross Validation for Parameter Selection

The rank r and the regularization parameters α and λ in (16.23) (which determine how much each regularization term contributes to the objective) must be chosen before constructing an NMF. One way to evaluate the quality of a choice of these parameters is to use cross validation, where in each fold some known entries of X are dropped, an NMF is constructed of the form WH for the matrix with those missing entries. The score for the choice of the parameters in one fold is the average of the errors $|X_{ij} - (WH)_{ij}|$ or the norm $\|(\mathbb{1} - M) \odot (X - WH)\|$, where $\mathbb{1}$ is the all-ones matrix. Ideally every entry of X should be left out of exactly one fold and included in all the others. Also, there should not be too many omitted entires in each fold, and those entries should not have a special pattern in the matrix. For example, if every element of a given row is omitted, then the entire corresponding row of W would be undetermined, which would be undesirable.

Averaging the error of all the folds for a given choice of parameters should give a good measure of how well the particular parameters perform on the given matrix.

16.5.6 *Application: Latent Semantic Indexing

Latent semantic indexing is an application of PCA or NMF to linguistic data. Starting with a collection (called a *corpus*) of text documents, we first remove all so-called *stop words*, which are common words with little discriminative power (like *the*, *a*, *is*, *it*, and so forth). We then build a matrix X where each row corresponds to a document (one data point) and each column corresponds to a word in the corpus. The i,k-entry of X is a value determined by the number of times work w occurs in document i. Taking the SVD or NMF to approximately factor X as $X \approx WH$ identifies directions (corresponding to certain weighted combinations of words) that seem to most strongly distinguish different documents. These are usually thought of as "topics."

The most naïve approach for populating the entries of X is to use the raw count r_{ik}, where r_{ik} is the integer number of times that word k occurs in document i. Alternatively, it is sometimes more useful to restrict r_{ik} to lie in $\{0, 1\}$, where 1 indicates that word k occurs at least once in document i.

One problem with the raw count is that longer documents will have more instances of each word than shorter documents, but the topic of the document should not really depend on the length of the document. To address this, it is common to use, instead, the *term frequency*, which is the raw count divided by the row sum (the total of all the raw counts of all the words in a given document):

$$\mathrm{tf}_{i,k} = \frac{r_{i,k}}{\sum_{k' \in d_i} r_{i,k'}},$$

where the sum is taken over all words k' in the ith document d_i.

The term frequency also has a problem, namely, common words that show up in many documents have little discriminative power, but are weighted heavily, while rare words that show up in only a few documents and thus are more likely to discriminate between documents and topics, have a lower weight. To address this, it is common to multiply the term frequency by the *inverse document frequency*

$$\mathrm{idf}_k = \log\left(1 + \frac{n}{n_k}\right),$$

where n is the total number of documents in the corpus and n_k is the number of documents that contain the word k. There are other variants of the idf that may be more suitable for certain situations, but what is important is that they all give a smaller value to the words that occur in more documents and a larger value to words that occur in fewer documents. One of the most common choices for populating the data matrix X is called tf-idf, which is just the product of tf with idf:

$$x_{ik} = \mathrm{tf}_{i,k}\,\mathrm{idf}_k\,.$$

Once the data matrix is populated, we can analyze the corpus by reducing the dimension via PCA (centering the data, taking the SVD, and finding the principal components), by uncentered application of the SVD (not centering leaves X sparse, so that the SVD can be computed more efficiently, but not centering also makes the result not generalizable to new data) or by NMF. The lower-dimensional representation can be used to identify similar documents or to identify synonymous words.

PCA gives a factorization $X \approx WH$ without a nonnegativity constraint. This corresponds to a collection of basis vectors (the rows of H) that we would like to interpret as topics. These vectors are measured relative to the sample mean $\hat{\boldsymbol{\mu}}$, which is always nonnegative, since the entries of X (before centering) are nonnegative. A basis vector $\mathbf{b}_i$ with a negative entry $b_{i,k} < 0$ corresponds to a "topic" that has fewer instances of the kth word than the mean does. But it is also possible that the vector $\boldsymbol{\mu} + \mathbf{b}_i$ has negative entries, which doesn't seem to have a good interpretation, since it would mean that for the ith topic, the corresponding word occurs a negative number of times.

But if we use NMF, that is, if we leave X uncentered and require $W \succeq \mathbf{0}$ and $H \succeq \mathbf{0}$ in the factorization $X \approx WH$, then the rows of H never have negative entries and words only appear with positive weights in each topic. Moreover, requiring the weights H to be positive guarantees that any document is a positive linear combination of topics, so each topic can contribute to a document, but (unlike with PCA) you would never see a negative amount of a topic in a document.

One obvious limitation of LSI, regardless of whether we use PCA or NMF, is the fact that it completely ignores word order and word proximity. But both order and proximity are important for understanding the meaning of a phrase in English. In LSI each element of the data matrix X is determined only by the words that occur in each document, but not by the which words were close to each other nor the order in which they came. Models that depend only on which words occur but ignore order and proximity are called *bag-of-words* models. A more sophisticated way to model text documents is to track *n-grams*, which are ordered n-tuples of consecutive words. For example, the previous sentence contains the 3-grams (more, sophisticated, way) and (model, text, documents) among others.

Of course, if there are N total words in the corpus, the number of potential 2-grams to search for is $\binom{N}{2} \sim \frac{1}{2}N^2$, and so if a corpus contains 20,000 unique words, then the number of 2-grams to consider is on the order of 2×10^8. Moreover, the complexity of an n-gram model grows exponentially in n, so it is generally impractical to make computations with longer n-grams.

Exercises

Note to the student: Each section of this chapter has several corresponding exercises, all collected here at the end of the chapter. The exercises between the first and second line are for Section 1, the exercises between the second and third lines are for Section 2, and so forth.

You should **work every exercise** (your instructor may choose to let you skip some of the advanced exercises marked with *). We have carefully selected them, and each is important for your ability to understand subsequent material. Many of the examples and results proved in the exercises are used again later in the text. Exercises marked with ⚠ are especially important and are likely to be used later in this book and beyond. Those marked with † are harder than average, but should still be done.

Although they are gathered together at the end of the chapter, we strongly recommend you do the exercises for each section as soon as you have completed the section, rather than saving them until you have finished the entire chapter.

16.1. Generate 500 points uniformly *on the surface* of a unit-radius sphere in 50 dimensions.

(i) Randomly generate five additional points on the surface of the sphere. For each of the five new points, calculate a narrow band of width $2/\sqrt{50}$ at the equator, assuming the point was the north pole. How many of the 500 points are in each band corresponding to one of the five equators? How many of the points are in all five bands? How wide do the bands need to be for all points to be in all bands?

(ii) Create several random lines through the origin and project the points onto each line. Plot the distribution of points on each line. What does your result say about the surface area of the sphere in relation to the lines, i.e., where is the surface area concentrated relative to each line?

16.2. Implement the algorithm described in Section 16.1.3 to generate 500 points uniformly *in the interior* of a unit-radius ball in 50 dimensions.

(i) Randomly generate five additional points on the surface of the ball. For each of the five new points, calculate a narrow band of width $2/\sqrt{50}$ at the equator, assuming the point was the north pole. How many of the 500 points are in each band corresponding to one of the five equators? How many of the points are in all five bands? How wide do the bands need to be for all points to be in all bands?

(ii) Create several random lines through the origin and project the points onto each line. Plot the distribution of points on each line. What does your result say about the volume of the sphere in relation to the lines, i.e., where is the interior concentrated relative to each line?

16.3. Assume that 100 points are chosen uniformly inside an n-dimensional unit-radius ball, for some large n. Given a random vector, it defines two parallel hyperplanes on opposite sides of the origin that are equal distance from the origin. How close can the hyperplanes be moved and still have at least 99% probability that all 100 points land between them?

16.4. The n-dimensional unit cube is the set $\{\mathbf{x} \in \mathbb{R}^n \mid 0 \leq x_i \leq 1 \forall i\}$, so it has one vertex at the origin and all other vertices correspond to vectors with every entry either 0 or 1. The vertices adjacent to the origin are the the standard unit basis vectors $\mathbf{e}_i = (0, \ldots, 0, 1, 0 \ldots, 0)$ with a 1 only in the ith position. Define the *equator* of the unit hypercube to be the hyperplane

$$\left\{ \mathbf{x} \middle| \sum_{i=1}^{n} x_i = \frac{n}{2} \right\}.$$

(i) Are the vertices of a unit cube concentrated close to the equator? Justify your answer.

(ii) Is the volume concentrated close to the equator? Justify your answer.

16.5. Code up the PCA algorithm (you may use a pre-built SVD solver). Your code should

(i) Accept a data matrix X where each row is a data point, and each column is a feature, and accept an optional integer s number of principal components to compute. If $s = \texttt{None}$, then assume $s = d$ (the number of features of the data).

(ii) Compute the variance $(\sigma_1^2, \ldots, \sigma_s^2)$ for the first s principal directions.

(iii) Provide a method `.transform` that accepts new data points $\mathbf{x} \in \mathbb{R}^d$ and returns the principal components $(a_1, \ldots, a_s) = (\mathbf{v}_1^\mathsf{T}\mathbf{x}, \ldots, \mathbf{v}_s^\mathsf{T}\mathbf{x})$. Note that the $\mathbf{v}_i$ here are determined by the first (training) matrix X, and not by subsequent data points $\mathbf{x}$.

(iv) Provide a method `.project` that accepts data points $\mathbf{x} \in \mathbb{R}^d$ and returns the projection $\text{proj}_{\text{span}(\mathbf{v}_1,\ldots,\mathbf{v}_s)} \mathbf{x} = \sum_{i=1}^s a_i \mathbf{v}_i$ of $\mathbf{x}$ to the subspace spanned by the principal axes $(\mathbf{v}_1, \ldots, \mathbf{v}_s)$.

16.6. For the Fashion-MNIST dataset and one other large dataset of your choice do the following:

(i) Remove index and identification numbers and any other irrelevant data. Separate the data into X (the features for each data point) and y (a classification, diagnosis, or dependent variable).

(ii) Identify whether the data X is suitable for PCA as is, or whether it needs centering, normalization, conversion of categorical data to continuous (as in MCA), or other adjustments. Make the necessary transformations.

(iii) Using your code from the previous problem (or a standard implementation of PCA), find the variance for each principal axis and plot the amount of variance explained as a function of the number of principal components (cumulative variance explained) and the variance explained by each of the principal components (a scree plot). Is there a value of s that is much less than d but such that the first s principal components still explain most of the variance? Explain.

(iv) Compute the first two principal components of X. Plot this as a scatterplot where the color of each point is determined by the corresponding label y. Does this 2-variable PCA do a good job of separating the different classes? Explain this in terms of the plots you made for the previous part of this problem.

(v) For the Fashion-MNIST dataset only: Find the minimum number of principal components necessary to account for 90% of the variance. Orthogonally project the data onto the subspace spanned by those principal axes ($\text{proj}_{\text{span}(\mathbf{v}_1,\ldots,\mathbf{v}_s)} \mathbf{x} = \sum_{i=1}^s a_i \mathbf{v}_i$) and plot a random selection of 20 of the resulting images, along with the the corresponding 20 images of the original dataset.

16.7. Prove that applying principal component analysis (projecting data to the first s principal axes) can never decrease the signal-to-noise ratio and will strictly improve that ratio unless all of the singular values $\sigma_1, \ldots, \sigma_d$ are equal. Hint: The relevant signal-to-noise ratios are written out in (16.7) and (16.8).

16.8.* Code up the MCA method. Your code should accept a binary data matrix Y (the output of the one-hot encoder) and convert it to a matrix X of continuous data whose entries x_{ik} are as described in Section 16.2.5.

16.9.* Add Gaussian noise $\sim \mathcal{N}(\mathbf{0}, I)$ to each image in the Fashion-MNIST dataset, apply PCA, and orthogonally project to the subspace spanned by the fewest principal axes needed to account for 90% of the variance. Plot 20 randomly selected images from the clean, original dataset, the corresponding 20 of the noisy images, and the corresponding 20 of the filtered images.

16.10. Experiment with the JL bound in the following way:

(i) Write a method called `JL_bound(n,eps)` to compute the Johnson-Lindenstrauss bound k using (16.9), where `n` is the number of points and `eps` is the maximal allowed distortion of the distance.

(ii) Write a method called `preserves_distance(X,A,eps)` that returns a Boolean value indicating whether the matrix `A` preserserves the relative distance between all pairs of points (rows of `X`) with a distortion of no more than `eps`. Hint: The rows of `X` are transposed, so `A` and the maps defined by `A` need to be adjusted slightly from what is written in the book.

(iii) For each value of `n` in $\{3, 10, 100\}$ randomly generate `n` points in $\mathbb{R}^{10^5}$ and use them to construct an array `X` of shape $n \times 10^5$.

(iv) For each `n` in $3, 10, 100$ and each `eps` in $0.7, 0.5, 0.1$ let k be the corresponding JL bound and construct a random projection `A` from $\mathbb{R}^{10^5}$ to $\mathbb{R}^k$, where the entries of A are drawn from $\mathcal{N}(0, 1/k)$.

(v) Run `preserves_distance(X,A,eps)` for each of the previously constructed `X`, `A`, and corresponding `eps`. If a given projection `A` fails its test, choose a new random projection and repeat the test. Repeat as needed until it passes. For each choice of `n` and `eps` record the number of failures before the test is passed.

16.11. Prove Proposition 16.3.8.

16.12. Prove Lemma 16.3.3 and explain why the condition $\lambda < \frac{1}{2}$ is needed. Hint: When computing the required integral, consider using the substitution $y = w\sqrt{1-2\lambda}$.

16.13. Finish the proof of Lemma 16.3.4 by computing the expectation and covariance matrix of $\mathbf{w}$. Hint: Write the ij term of the covariance matrix explicitly as a (weighted) sum of products of the form $A_{i\ell}u_\ell u_m A_{jm}$ and use the fact that all the entries of A are i.i.d.

16.14. Often a very simple random projection to a very low-dimensional space—much smaller than the bounds computed in Proposition 16.3.8—can still be very useful. Here is one example with the FashionMNIST data set:

(i) Construct a random projection that (randomly) forgets all but 78 (10%) of the 784 FashionMNIST features and randomly changes the sign of half of those remaining features. Apply your projection to both the training data and the test data to get x values in 78-dimensional space instead of 784-dimensional space.

(ii) Time how long it takes to train the sklearn `GradientBoostingClassifier` on one tenth (6,000 randomly selected points) of the randomly projected (78-dimensional) training data. Compare this to how long it takes to train on the same collection of points without the random projection (784-dimensional) with the same choice of hyperparameters.

(iii) Using the same random projection on the test set, compute the accuracy of the classifier trained on the randomly projected training set. Compare that to the accuracy of the classifier trained on the unprojected training set.

(iv) Time how long it takes to compute the SVD `u,s,vh = SVD(X)` of the design matrix `X` for the (6000-point) unprojected training set. Projecting the data to the first 20 principal components corresponds to multiplying the design matrix `X` on the right by the first 20 columns of `v` to get `X @ vh[:20,:].T`. Compute that principle-component projection for both the training data and the test data.

(v) Time how long it takes to train the sklearn `GradientBoostingClassifier` on the principle-component-projected training data and add that training time to the time it took to compute the SVD. Compare that sum to the training time required for the unprojected data and for the randomly projected data. One reason that training the classifier on the principal-component-projected data takes longer than you might expect is that the principal-component projected data are not sparse while the unprojected and randomly projected data are relatively sparse.

(vi) Use the principle-component-projected test data to evaluate the accuracy of the classifier trained on the principle-component-projected training data. Compare to the accuracy of the unprojected and randomly-projected classifiers.

16.15. As described in Section 16.4.3, perplexity controls the parameter σ_i in the distribution p_{ij}, and it corresponds roughly to the number of neighbors for each point in the original data set. Explore how the choice of perplexity value changes the output of t-SNE in the following examples. For each collection of clusters do the following: For each value of `perplexity` in $\{2, 5, 10, 30, 50\}$, run t-SNE with `n_components=2` on the data set and plot the result using different colors to distinguish the different clusters.

(i) Two disjoint clusters in $\mathbb{R}^3$ created by sampling 50 points from a multivariate normal distribution with mean $(0, 0, 0)$ and covariance I (give these the label 0) and sampling 50 points from a multivariate normal with mean $(3, 3, 3)$ and covariance I (give these the label 1).

(ii) Three clusters in $\mathbb{R}^3$ created by sampling 50 points from a multivariate normal distribution with mean $(0, 0, 0)$, 50 more points with mean $(2, 2, 2)$ and 50 more points with covariance $(-1, 2, 1)$, all with covariance I.

(iii) Two line-like clusters in $\mathbb{R}^2$ created by sampling 50 points from a multivariate normal distribution with mean $(0, 0)$ and covariance $\Sigma = \left[\begin{smallmatrix} 10 & 0 \\ 0 & 0.1 \end{smallmatrix}\right]$ and sampling 50 points from a multivariate normal with mean $(2, 2)$ and covariance Σ.

(iv) A single cluster in $\mathbb{R}^3$ consisting of 100 points drawn from the multivariate normal distribution with mean $(0, 0, 0)$ and covariance I.

(v) A single cluster consisting of all the points in $[0, 10] \times [0, 10]$ with integer coordinates: $(0, 0), (0, 1), (1, 0), \ldots, (9, 10), (10, 9), (10, 10)$.

16.16. Explain in your own words (200 words or less) the mathematical idea behind t-SNE and UMAP. Are there any circumstances that t-SNE might perform better than UMAP?

16.17. For the Fashion-MNIST dataset and one other large dataset of your own choice, do the following:
Run (and time) t-SNE and UMAP with their default setting to get projections of the data set down to 2 dimension. Plot your results as a scatterplot where the color of each point is determined by the corresponding label. Compare your results to the PCA results you got for Exercise 16.6. Some clusters in the Fashion-MNIST scatterplots may have multiple colors. For each of these clusters, find a pair of points with different colors that lie near each other in the cluster and plot the corresponding images. Does it look reasonable for these images to be mapped to similar places in $\mathbb{R}^2$? Hint: Use the default implementation of UMAP and the sklearn implementation of t-SNE. Note: use `pip install umap-learn`. Do not use the package called `umap`, which is something else entirely. If the computations do not complete in 10 minutes or less, you may sample randomly from the data set to get a smaller data set with at least 4000 points and rerun the computation on the smaller set.

16.18. Assume that $\alpha \geq 0$, that $\lambda \in [0, 1]$, and that $\|\cdot\|$ is a convex norm (like the Frobenius norm, or the p-norm for any $p \geq 1$). Prove that for any fixed W, the objective in (16.23) is a convex function of H. This shows that for a fixed W, the optimization problem of finding H to minimize (16.23) is a convex optimization problem. Also prove that for any fixed H the objective is a convex function of W.

16.19. Many examples of recommender systems (including this year's NMF lab) compute the NMF of a target matrix X that represents the number of each item that each person bought. Of course there are often zero entries, since not everyone buys everything, but there are no missing values.

(i) In your own words explain why, for a movie recommender system with many different movies, using the NMF of a matrix with zeros in the entries corresponding to items that a person did not rate might result in a misleading or suboptimal result.

(ii) What mathematical alternatives could be used to avoid the issues described in the previous step?

(iii) Describe in detail, with equations, the optimization problem corresponding to your solution in the previous step.

16.20. Cross validation to choose parameters for an NMF is different from traditional cross validation for a supervised learning problem with a data set of the form $\{(\mathbf{x}_1, y_1), \ldots, \mathbf{x}_N, y_N)\}$. In your own words, describe how the data set in an NMF problem differs from the data sets of a supervised learning problem and how cross validation differs for NMF.

Notes