13 *Sparse linear models*

13.1 Introduction

We introduced the topic of feature selection in Section 3.5.4, where we discussed methods for finding input variables which had high mutual information with the output. The trouble with this approach is that it is based on a myopic strategy that only looks at one variable at a time. This can fail if there are interaction effects. For example, if $y = \text{xor}(x_1, x_2)$, then neither x_1 nor x_2 on its own can predict the response, but together they perfectly predict the response. For a real-world example of this, consider genetic association studies: sometimes two genes on their own may be harmless, but when present together they cause a recessive disease (Balding 2006).

In this chapter, we focus on selecting sets of variables at a time using a model-based approach. If the model is a generalized linear model, of the form $p(y|\mathbf{x}) = p(y|f(\mathbf{w}^T\mathbf{x}))$ for some link function f, then we can perform feature selection by encouraging the weight vector $\mathbf{w}$ to be **sparse**, i.e., to have lots of zeros. This approach turns out to offer significant computational advantages, as we will see below.

Here are some applications where feature selection/ sparsity is useful:

- In many problems, we have many more dimensions D than training cases N. The corresponding design matrix is short and fat, rather than tall and skinny. This is called the **small N, large D** problem. This is becoming increasingly prevalent as we develop more high throughput measurement devices, For example, with gene microarrays, it is common to measure the expression levels of $D \sim 10,000$ genes, but to only get $N \sim 100$ such examples. (It is perhaps a sign of the times that even our data seems to be getting fatter...) We may want to find the smallest set of features that can accurately predict the response (e.g., growth rate of the cell) in order to prevent overfitting, to reduce the cost of building a diagnostic device, or to help with scientific insight into the problem.

- In Chapter 14, we will use basis functions centered on the training examples, so $\boldsymbol{\phi}(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}, \mathbf{x}_N)]$, where κ is a kernel function. The resulting design matrix has size $N \times N$. Feature selection in this context is equivalent to selecting a subset of the training examples, which can help reduce overfitting and computational cost. This is known as a sparse kernel machine.

- In signal processing, it is common to represent signals (images, speech, etc.) in terms of wavelet basis functions. To save time and space, it is useful to find a sparse representation of the signals, in terms of a small number of such basis functions. This allows us to estimate signals from a small number of measurements, as well as to compress the signal. See Section 13.8.3 for more information.

Note that the topic of feature selection and sparsity is currently one of the most active areas of machine learning/ statistics. In this chapter, we only have space to give an overview of the main results.

13.2 Bayesian variable selection

A natural way to pose the variable selection problem is as follows. Let $\gamma_j = 1$ if feature j is "relevant", and let $\gamma_j = 0$ otherwise. Our goal is to compute the posterior over models

$$p(\boldsymbol{\gamma}|\mathcal{D}) = \frac{e^{-f(\boldsymbol{\gamma})}}{\sum_{\boldsymbol{\gamma}'} e^{-f(\boldsymbol{\gamma}')}} \tag{13.1}$$

where $f(\boldsymbol{\gamma})$ is the cost function:

$$f(\boldsymbol{\gamma}) \triangleq -[\log p(\mathcal{D}|\boldsymbol{\gamma}) + \log p(\boldsymbol{\gamma})] \tag{13.2}$$

For example, suppose we generate $N = 20$ samples from a $D = 10$ dimensional linear regression model, $y_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}_i, \sigma^2)$, in which $K = 5$ elements of $\mathbf{w}$ are non-zero. In particular, we use $\mathbf{w} = (0.00, -1.67, 0.13, 0.00, 0.00, 1.19, 0.00, -0.04, 0.33, 0.00)$ and $\sigma^2 = 1$. We enumerate all $2^{10} = 1024$ models and compute $p(\boldsymbol{\gamma}|\mathcal{D})$ for each one (we give the equations for this below). We order the models in **Gray code** order, which ensures consecutive vectors differ by exactly 1 bit (the reasons for this are computational, and are discussed in Section 13.2.3).

The resulting set of bit patterns is shown in Figure 13.1(a). The cost of each model, $f(\boldsymbol{\gamma})$, is shown in Figure 13.1(b). We see that this objective function is extremely "bumpy". The results are easier to interpret if we compute the posterior distribution over models, $p(\boldsymbol{\gamma}|\mathcal{D})$. This is shown in Figure 13.1(c). The top 8 models are listed below:

model	prob	members
4	0.447	2,
61	0.241	2, 6,
452	0.103	2, 6, 9,
60	0.091	2, 3, 6,
29	0.041	2, 5,
68	0.021	2, 6, 7,
36	0.015	2, 5, 6,
5	0.010	2, 3,

The "true" model is $\{2, 3, 6, 8, 9\}$. However, the coefficients associated with features 3 and 8 are very small (relative to σ^2). so these variables are harder to detect. Given enough data, the method will converge on the true model (assuming the data is generated from a linear model), but for finite data sets, there will usually be considerable posterior uncertainty.

Interpreting the posterior over a large number of models is quite difficult, so we will seek various summary statistics. A natural one is the posterior mode, or MAP estimate

$$\hat{\boldsymbol{\gamma}} = \text{argmax}\, p(\boldsymbol{\gamma}|\mathcal{D}) = \text{argmin}\, f(\boldsymbol{\gamma}) \tag{13.3}$$

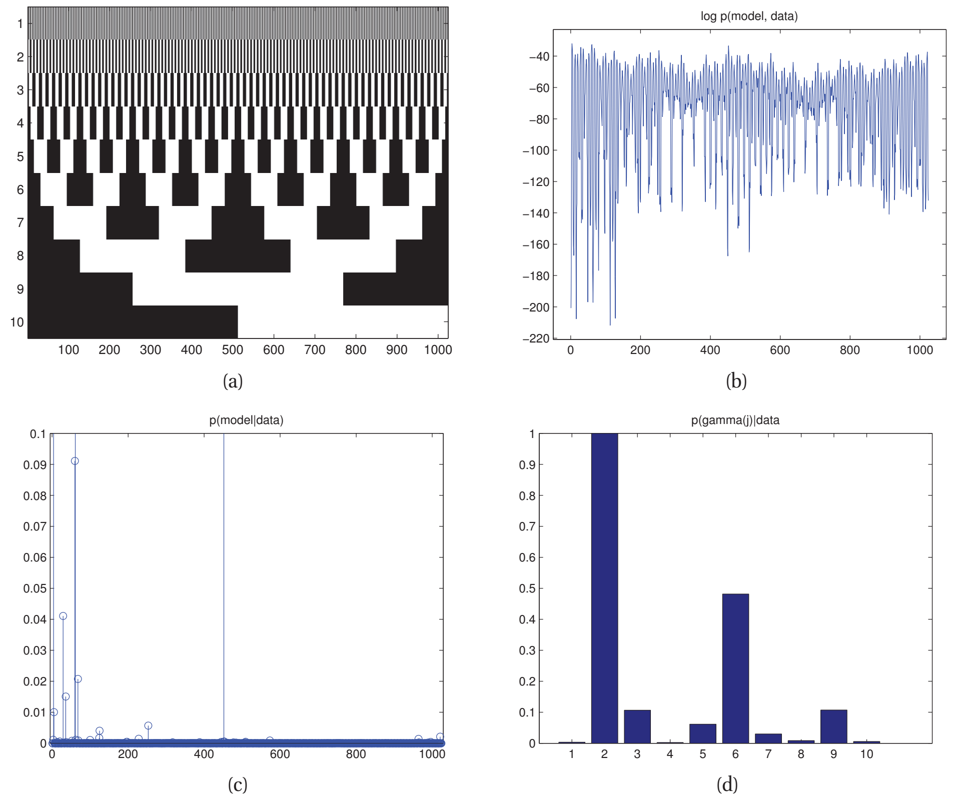


Figure 13.1 (a) All possible bit vectors of length 10 enumerated in Gray code order. (b) Score function for all possible models. (c) Posterior over all 1024 models. Vertical scale has been truncated at 0.1 for clarity. (d) Marginal inclusion probabilities. Figure generated by `linregAllsubsetsGraycodeDemo`.

However, the mode is often not representative of the full posterior mass (see Section 5.2.1.3). A better summary is the **median model** (Barbieri and Berger 2004; Carvahlo and Lawrence 2007), computed using

$$\hat{\gamma} = \{j : p(\gamma_j = 1|\mathcal{D}) > 0.5\} \tag{13.4}$$

This requires computing the posterior marginal **inclusion probabilities**, $p(\gamma_j = 1|\mathcal{D})$. These are shown in Figure 13.1(d). We see that the model is confident that variables 2 and 6 are included; if we lower the decision threshold to 0.1, we would add 3 and 9 as well. However, if we wanted to "capture" variable 8, we would incur two false positives (5 and 7). This tradeoff between false positives and false negatives is discussed in more detail in Section 5.7.2.1.

The above example illustrates the "gold standard" for variable selection: the problem was sufficiently small (only 10 variables) that we were able to compute the full posterior exactly. Of course, variable selection is most useful in the cases where the number of dimensions is large. Since there are 2^D possible models (bit vectors), it will be impossible to compute the full posterior in general, and even finding summaries, such as the MAP estimate or marginal inclusion probabilities, will be intractable. We will therefore spend most of this chapter focussing on algorithmic speedups. But before we do that, we will explain how we computed $p(\boldsymbol{\gamma}|\mathcal{D})$ in the above example.

13.2.1 The spike and slab model

The posterior is given by

$$p(\boldsymbol{\gamma}|\mathcal{D}) \propto p(\boldsymbol{\gamma})p(\mathcal{D}|\boldsymbol{\gamma}) \tag{13.5}$$

We first consider the prior, then the likelihood.

It is common to use the following prior on the bit vector:

$$p(\boldsymbol{\gamma}) = \prod_{j=1}^{D} \text{Ber}(\gamma_j|\pi_0) = \pi_0^{||\boldsymbol{\gamma}||_0}(1-\pi_0)^{D-||\boldsymbol{\gamma}||_0} \tag{13.6}$$

where π_0 is the probability a feature is relevant, and $||\boldsymbol{\gamma}||_0 = \sum_{j=1}^{D} \gamma_j$ is the ℓ_0 **pseudo-norm**, that is, the number of non-zero elements of the vector. For comparison with later models, it is useful to write the log prior as follows:

$$\begin{aligned}
\log p(\boldsymbol{\gamma}|\pi_0) &= ||\boldsymbol{\gamma}||_0 \log \pi_0 + (D - ||\boldsymbol{\gamma}||_0)\log(1-\pi_0) && (13.7)\\
&= ||\boldsymbol{\gamma}||_0(\log \pi_0 - \log(1-\pi_0)) + \text{const} && (13.8)\\
&= -\lambda||\boldsymbol{\gamma}||_0 + \text{const} && (13.9)
\end{aligned}$$

where $\lambda \triangleq \log \frac{1-\pi_0}{\pi_0}$ controls the sparsity of the model.

We can write the likelihood as follows:

$$p(\mathcal{D}|\boldsymbol{\gamma}) = p(\mathbf{y}|\mathbf{X}, \boldsymbol{\gamma}) = \int\int p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \boldsymbol{\gamma})p(\mathbf{w}|\boldsymbol{\gamma}, \sigma^2)p(\sigma^2)d\mathbf{w}d\sigma^2 \tag{13.10}$$

For notational simplicity, we have assumed the response is centered, (i.e., $\overline{y} = 0$), so we can ignore any offset term μ.

We now discuss the prior $p(\mathbf{w}|\boldsymbol{\gamma}, \sigma^2)$. If $\gamma_j = 0$, feature j is irrelevant, so we expect $w_j = 0$. If $\gamma_j = 1$, we expect w_j to be non-zero. If we standardize the inputs, a reasonable prior is $\mathcal{N}(0, \sigma^2\sigma_w^2)$, where σ_w^2 controls how big we expect the coefficients associated with the relevant variables to be (which is scaled by the overall noise level σ^2). We can summarize this prior as follows:

$$p(w_j|\sigma^2, \gamma_j) = \begin{cases} \delta_0(w_j) & \text{if } \gamma_j = 0 \\ \mathcal{N}(w_j|0, \sigma^2\sigma_w^2) & \text{if } \gamma_j = 1 \end{cases} \tag{13.11}$$

The first term is a "spike" at the origin. As $\sigma_w^2 \to \infty$, the distribution $p(w_j|\gamma_j = 1)$ approaches a uniform distribution, which can be thought of as a "slab" of constant height. Hence this is called the **spike and slab** model (Mitchell and Beauchamp 1988).

We can drop the coefficients w_j for which $w_j = 0$ from the model, since they are clamped to zero under the prior. Hence Equation 13.10 becomes the following (assuming a Gaussian likelihood):

$$p(\mathcal{D}|\boldsymbol{\gamma}) = \int\int \mathcal{N}(\mathbf{y}|\mathbf{X}_{\boldsymbol{\gamma}}\mathbf{w}_\gamma, \sigma^2\mathbf{I}_N)\mathcal{N}(\mathbf{w}_\gamma|\mathbf{0}_{D_\gamma}, \sigma^2\sigma_w^2\mathbf{I}_{D_\gamma})p(\sigma^2)d\mathbf{w}_\gamma d\sigma^2 \tag{13.12}$$

where $D_\gamma = ||\boldsymbol{\gamma}||_0$ is the number of non-zero elements in $\boldsymbol{\gamma}$. In what follows, we will generalize this slightly by defining a prior of the form $p(\mathbf{w}|\boldsymbol{\gamma}, \sigma^2) = \mathcal{N}(\mathbf{w}_\gamma|\mathbf{0}_{D_\gamma}, \sigma^2\boldsymbol{\Sigma}_\gamma)$ for any positive definite matrix $\boldsymbol{\Sigma}_\gamma$.[1]

Given these priors, we can now compute the marginal likelihood. If the noise variance is known, we can write down the marginal likelihood (using Equation 13.151) as follows:

$$p(\mathcal{D}|\boldsymbol{\gamma}, \sigma^2) = \int \mathcal{N}(\mathbf{y}|\mathbf{X}_\gamma \mathbf{w}_\gamma, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{w}_\gamma|\mathbf{0}, \sigma^2\boldsymbol{\Sigma}_\gamma)d\mathbf{w}_\gamma = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_\gamma) \tag{13.13}$$

$$\mathbf{C}_\gamma \triangleq \sigma^2\mathbf{X}_\gamma\boldsymbol{\Sigma}_\gamma\mathbf{X}_\gamma^T + \sigma^2\mathbf{I}_N \tag{13.14}$$

If the noise is unknown, we can put a prior on it and integrate it out. It is common to use $p(\sigma^2) = \text{IG}(\sigma^2|a_\sigma, b_\sigma)$. Some guidelines on setting a, b can be found in (Kohn et al. 2001). If we use $a = b = 0$, we recover the Jeffrey's prior, $p(\sigma^2) \propto \sigma^{-2}$. When we integrate out the noise, we get the following more complicated expression for the marginal likelihood (Brown et al. 1998):

$$p(\mathcal{D}|\boldsymbol{\gamma}) = \int\int p(\mathbf{y}|\boldsymbol{\gamma}, \mathbf{w}_\gamma, \sigma^2)p(\mathbf{w}_\gamma|\boldsymbol{\gamma}, \sigma^2)p(\sigma^2)d\mathbf{w}_\gamma d\sigma^2 \tag{13.15}$$

$$\propto |\mathbf{X}_\gamma^T\mathbf{X}_\gamma + \boldsymbol{\Sigma}_\gamma^{-1}|^{-\frac{1}{2}}|\boldsymbol{\Sigma}_\gamma|^{-\frac{1}{2}}(2b_\sigma + S(\boldsymbol{\gamma}))^{-(2a_\sigma+N-1)/2} \tag{13.16}$$

where $S(\boldsymbol{\gamma})$ is the RSS:

$$S(\boldsymbol{\gamma}) \triangleq \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}_\gamma(\mathbf{X}_\gamma^T\mathbf{X}_\gamma + \boldsymbol{\Sigma}_\gamma^{-1})^{-1}\mathbf{X}_\gamma^T\mathbf{y} \tag{13.17}$$

See also Exercise 13.4.

When the marginal likelihood cannot be computed in closed form (e.g., if we are using logistic regression or a nonlinear model), we can approximate it using BIC, which has the form

$$\log p(\mathcal{D}|\boldsymbol{\gamma}) \approx \log p(\mathbf{y}|\mathbf{X}, \hat{\mathbf{w}}_\gamma, \hat{\sigma}^2) - \frac{||\boldsymbol{\gamma}||_0}{2}\log N \tag{13.18}$$

where $\hat{\mathbf{w}}_\gamma$ is the ML or MAP estimate based on $\mathbf{X}_{\boldsymbol{\gamma}}$, and $||\boldsymbol{\gamma}||_0$ is the "degrees of freedom" of the model (Zou et al. 2007). Adding the log prior, the overall objective becomes

$$\log p(\boldsymbol{\gamma}|\mathcal{D}) \approx \log p(\mathbf{y}|\mathbf{X}, \hat{\mathbf{w}}_\gamma, \hat{\sigma}^2) - \frac{||\boldsymbol{\gamma}||_0}{2}\log N - \lambda||\boldsymbol{\gamma}||_0 + \text{const} \tag{13.19}$$

We see that there are two complexity penalties: one arising from the BIC approximation to the marginal likelihood, and the other arising from the prior on $p(\boldsymbol{\gamma})$. Obviously these can be combined into one overall complexity parameter, which we will denote by λ.

13.2.2 From the Bernoulli-Gaussian model to ℓ_0 regularization

Another model that is sometimes used (e.g., (Kuo and Mallick 1998; Zhou et al. 2009; Soussen et al. 2010)) is the following:

$$y_i|\mathbf{x}_i, \mathbf{w}, \boldsymbol{\gamma}, \sigma^2 \sim \mathcal{N}(\sum_j \gamma_j w_j x_{ij}, \sigma^2) \tag{13.20}$$

$$\gamma_j \sim \text{Ber}(\pi_0) \tag{13.21}$$

$$w_j \sim \mathcal{N}(0, \sigma_w^2) \tag{13.22}$$

1. It is common to use a g-prior of the form $\boldsymbol{\Sigma}_\gamma = g(\mathbf{X}_\gamma^T\mathbf{X}_\gamma)^{-1}$ for reasons explained in Section 7.6.3.1 (see also Exercise 13.4). Various approaches have been proposed for setting g, including cross validation, empirical Bayes (Minka 2000b; George and Foster 2000), hierarchical Bayes (Liang et al. 2008), etc.

In the signal processing literature (e.g., (Soussen et al. 2010)), this is called the **Bernoulli-Gaussian** model, although we could also call it the **binary mask** model, since we can think of the γ_j variables as "masking out" the weights w_j.

Unlike the spike and slab model, we do not integrate out the "irrelevant" coefficients; they always exist. In addition, the binary mask model has the form $\gamma_j \rightarrow \mathbf{y} \leftarrow w_j$, whereas the spike and slab model has the form $\gamma_j \rightarrow w_j \rightarrow \mathbf{y}$. In the binary mask model, only the product $\gamma_j w_j$ can be identified from the likelihood.

One interesting aspect of this model is that it can be used to derive an objective function that is widely used in the (non-Bayesian) subset selection literature. First, note that the joint prior has the form

$$p(\boldsymbol{\gamma}, \mathbf{w}) \propto \mathcal{N}(\mathbf{w}|\mathbf{0}, \sigma_w^2 \mathbf{I}) \pi_0^{||\boldsymbol{\gamma}||_0} (1 - \pi_0)^{D - ||\boldsymbol{\gamma}||_0} \tag{13.23}$$

Hence the scaled unnormalized negative log posterior has the form

$$\begin{aligned} f(\boldsymbol{\gamma}, \mathbf{w}) \quad &\triangleq \quad -2\sigma^2 \log p(\boldsymbol{\gamma}, \mathbf{w}, \mathbf{y}|\mathbf{X}) = ||\mathbf{y} - \mathbf{X}(\boldsymbol{\gamma} .* \mathbf{w})||^2 \\ & \quad + \frac{\sigma^2}{\sigma_w^2} ||\mathbf{w}||^2 + \lambda ||\boldsymbol{\gamma}||_0 + \text{const} \end{aligned} \tag{13.24}$$

where

$$\lambda \triangleq 2\sigma^2 \log(\frac{1 - \pi_0}{\pi_0}) \tag{13.25}$$

Let us split $\mathbf{w}$ into two subvectors, $\mathbf{w}_{-\gamma}$ and $\mathbf{w}_\gamma$, indexed by the zero and non-zero entries of $\boldsymbol{\gamma}$ respectively. Since $\mathbf{X}(\boldsymbol{\gamma} .* \mathbf{w}) = \mathbf{X}_\gamma \mathbf{w}_\gamma$, we can just set $\mathbf{w}_{-\gamma} = \mathbf{0}$.

Now consider the case where $\sigma_w^2 \rightarrow \infty$, so we do not regularize the non-zero weights (so there is no complexity penalty coming from the marginal likelihood or its BIC approximation). In this case, the objective becomes

$$f(\boldsymbol{\gamma}, \mathbf{w}) = ||\mathbf{y} - \mathbf{X}_{\boldsymbol{\gamma}} \mathbf{w}_\gamma||_2^2 + \lambda ||\boldsymbol{\gamma}||_0 \tag{13.26}$$

This is similar to the BIC objective above.

Instead of keeping track of the bit vector $\boldsymbol{\gamma}$, we can define the set of relevant variables to be the **support**, or set of non-zero entries, of $\mathbf{w}$. Then we can rewrite the above equation as follows:

$$f(\mathbf{w}) = ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \lambda ||\mathbf{w}||_0 \tag{13.27}$$

This is called ℓ_0 **regularization**. We have converted the discrete optimization problem (over $\boldsymbol{\gamma} \in \{0, 1\}^D$) into a continuous one (over $\mathbf{w} \in \mathbb{R}^D$); however, the ℓ_0 pseudo-norm makes the objective very non smooth, so this is still hard to optimize. We will discuss different solutions to this in the rest of this chapter.

13.2.3 Algorithms

Since there are 2^D models, we cannot explore the full posterior, or find the globally optimal model. Instead we will have to resort to heuristics of one form or another. All of the methods we will discuss involve searching through the space of models, and evaluating the cost $f(\boldsymbol{\gamma})$ at

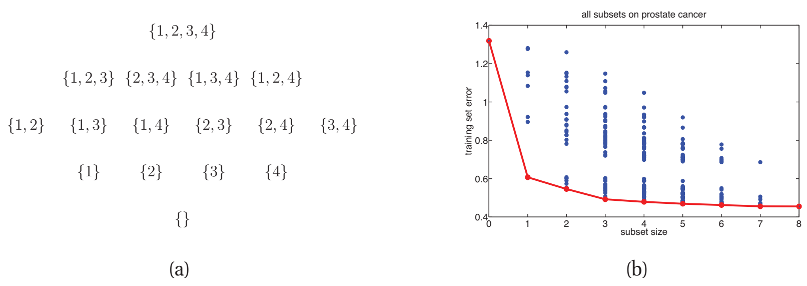


Figure 13.2 (a) A lattice of subsets of $\{1, 2, 3, 4\}$. (b) Residual sum of squares versus subset size, on the prostate cancer data set. The lower envelope is the best RSS achievable for any set of a given size. Based on Figure 3.5 of (Hastie et al. 2001). Figure generated by `prostateSubsets`.

each point. This requires fitting the model (i.e., computing $\operatorname{argmax} p(\mathcal{D}|\mathbf{w})$), or evaluating its marginal likelihood (i.e., computing $\int p(\mathcal{D}|\mathbf{w})p(\mathbf{w})d\mathbf{w}$) at each step. This is sometimes called the **wrapper method**, since we "wrap" our search for the best model (or set of good models) around a generic model-fitting procedure.

In order to make wrapper methods efficient, it is important that we can quickly evaluate the score function for some new model, $\boldsymbol{\gamma}'$, given the score of a previous model, $\boldsymbol{\gamma}$. This can be done provided we can efficiently update the sufficient statistics needed to compute $f(\boldsymbol{\gamma})$. This is possible provided $\boldsymbol{\gamma}'$ only differs from $\boldsymbol{\gamma}$ in one bit (corresponding to adding or removing a single variable), and provided $f(\boldsymbol{\gamma})$ only depends on the data via $\mathbf{X}_{\boldsymbol{\gamma}}$. In this case, we can use rank-one matrix updates/ downdates to efficiently compute $\mathbf{X}_{\boldsymbol{\gamma}'}^T\mathbf{X}_{\boldsymbol{\gamma}'}$ from $\mathbf{X}_{\boldsymbol{\gamma}}^T\mathbf{X}_{\boldsymbol{\gamma}}$. These updates are usually applied to the QR decomposition of $\mathbf{X}$. See e.g., (Miller 2002; Schniter et al. 2008) for details.

13.2.3.1 Greedy search

Suppose we want to find the MAP model. If we use the ℓ_0-regularized objective in Equation 13.27, we can exploit properties of least squares to derive various efficient greedy forwards search methods, some of which we summarize below. For further details, see (Miller 2002; Soussen et al. 2010).

- **Single best replacement** The simplest method is to use greedy hill climbing, where at each step, we define the neighborhood of the current model to be all models than can be reached by flipping a single bit of $\boldsymbol{\gamma}$, i.e., for each variable, if it is currently out of the model, we consider adding it, and if it is currently in the model, we consider removing it. In (Soussen et al. 2010), they call this the **single best replacement** (SBR). Since we are expecting a sparse solution, we can start with the empty set, $\boldsymbol{\gamma} = \mathbf{0}$. We are essentially moving through the lattice of subsets, shown in Figure 13.2(a). We continue adding or removing until no improvement is possible.
- **Orthogonal least squares** If we set $\lambda = 0$ in Equation 13.27, so there is no complexity penalty, there will be no reason to perform deletion steps. In this case, the SBR algorithm is equivalent to **orthogonal least squares** (Chen and Wigger 1995), which in turn is equivalent

to greedy **forwards selection**. In this algorithm, we start with the empty set and add the best feature at each step. The error will go down monotonically with $||\boldsymbol{\gamma}||_0$, as shown in Figure 13.2(b). We can pick the next best feature j^* to add to the current set $\boldsymbol{\gamma}_t$ by solving

$$j^* = \arg\min_{j \notin \boldsymbol{\gamma}_t} \min_{\mathbf{w}} ||\mathbf{y} - (\mathbf{X}_{\boldsymbol{\gamma}_t \cup j})\mathbf{w}||^2 \tag{13.28}$$

We then update the active set by setting $\boldsymbol{\gamma}^{(t+1)} = \boldsymbol{\gamma}^{(t)} \cup \{j^*\}$. To choose the next feature to add at step t, we need to solve $D - D_t$ least squares problems at step t, where $D_t = |\boldsymbol{\gamma}_t|$ is the cardinality of the current active set. Having chosen the best feature to add, we need to solve an additional least squares problem to compute $\mathbf{w}_{t+1}$).

- **Orthogonal matching pursuits** Orthogonal least squares is somewhat expensive. A simplification is to "freeze" the current weights at their current value, and then to pick the next feature to add by solving

$$j^* = \arg\min_{j \notin \boldsymbol{\gamma}_t} \min_{\beta} ||\mathbf{y} - \mathbf{X}\mathbf{w}_t - \beta\mathbf{x}_{:,j}||^2 \tag{13.29}$$

This inner optimization is easy to solve: we simply set $\beta = \mathbf{x}_{:,j}^T\mathbf{r}_t/||\mathbf{x}_{:,j}||^2$, where $\mathbf{r}_t = \mathbf{y} - \mathbf{X}\mathbf{w}_t$ is the current residual vector. If the columns are unit norm, we have

$$j^* = \arg\max \mathbf{x}_{:,j}^T\mathbf{r}_t \tag{13.30}$$

so we are just looking for the column that is most correlated with the current residual. We then update the active set, and compute the new least squares estimate $\mathbf{w}_{t+1}$ using $\mathbf{X}_{\boldsymbol{\gamma}_{t+1}}$. This method is called **orthogonal matching pursuits** or **OMP** (Mallat et al. 1994). This only requires one least squares calculation per iteration and so is faster than orthogonal least squares, but is not quite as accurate (Blumensath and Davies 2007).

- **Matching pursuits** An even more aggressive approximation is to just greedily add the feature that is most correlated with the current residual. This is called **matching pursuits** (Mallat and Zhang 1993). This is also equivalent to a method known as least squares boosting (Section 16.4.6).
- **Backwards selection Backwards selection** starts with all variables in the model (the so-called **saturated model**), and then deletes the worst one at each step. This is equivalent to performing a greedy search from the top of the lattice downwards. This can give better results than a bottom-up search, since the decision about whether to keep a variable or not is made in the context of all the other variables that might depende on it. However, this method is typically infeasible for large problems, since the saturated model will be too expensive to fit.
- **FoBa** The **forwards-backwards algorithm** of (Zhang 2008) is similar to the single best replacement algorithm presented above, except it uses an OMP-like approximation when choosing the next move to make. A similar "dual-pass" algorithm was described in (Moghaddam et al. 2008).
- **Bayesian Matching pursuit** The algorithm of (Schniter et al. 2008) is similiar to OMP except it uses a Bayesian marginal likelihood scoring criterion (under a spike and slab model) instead of a least squares objective. In addition, it uses a form of beam search to explore multiple paths through the lattice at once.

13.2.3.2 Stochastic search

If we want to approximate the posterior, rather than just computing a mode (e.g. because we want to compute marginal inclusion probabilities), one option is to use MCMC. The standard approach is to use Metropolis Hastings, where the proposal distribution just flips single bits. This enables us to efficiently compute $p(\boldsymbol{\gamma}'|\mathcal{D})$ given $p(\boldsymbol{\gamma}|\mathcal{D})$. The probability of a state (bit configuration) is estimated by counting how many times the random walk visits this state. See (O'Hara and Sillanpaa 2009) for a review of such methods, and (Bottolo and Richardson 2010) for a very recent method based on evolutionary MCMC.

However, in a discrete state space, MCMC is needlessly inefficient, since we can compute the (unnormalized) probability of a state directly using $p(\boldsymbol{\gamma}, \mathcal{D}) = \exp(-f(\boldsymbol{\gamma}))$; thus *there is no need to ever revisit a state.* A much more efficient alternative is to use some kind of stochastic search algorithm, to generate a set $\mathcal{S}$ of high scoring models, and then to make the following approximation

$$p(\boldsymbol{\gamma}|\mathcal{D}) \approx \frac{e^{-f(\boldsymbol{\gamma})}}{\sum_{\boldsymbol{\gamma}' \in \mathcal{S}} e^{-f(\boldsymbol{\gamma}')}} \tag{13.31}$$

See (Heaton and Scott 2009) for a review of recent methods of this kind.

13.2.3.3 EM and variational inference *

It is tempting to apply EM to the spike and slab model, which has the form $\gamma_j \to w_j \to \mathbf{y}$. We can compute $p(\gamma_j = 1|w_j)$ in the E step, and optimize $\mathbf{w}$ in the M step. However, this will not work, because when we compute $p(\gamma_j = 1|w_j)$, we are comparing a delta-function, $\delta_0(w_j)$, with a Gaussian pdf, $\mathcal{N}(w_j|0, \sigma_w^2)$. We can replace the delta function with a narrow Gaussian, and then the E step amounts to classifying w_j under the two possible Gaussian models. However, this is likely to suffer from severe local minima.

An alternative is to apply EM to the Bernoulli-Gaussian model, which has the form $\gamma_j \to \mathbf{y} \leftarrow w_j$. In this case, the posterior $p(\boldsymbol{\gamma}|\mathcal{D}, \mathbf{w})$ is intractable to compute because all the bits become correlated due to explaining away. However, it is possible to derive a mean field approximation of the form $\prod_j q(\gamma_j)q(w_j)$ (Huang et al. 2007; Rattray et al. 2009).

13.3 ℓ_1 regularization: basics

When we have many variables, it is computationally difficult to find the posterior mode of $p(\boldsymbol{\gamma}|\mathcal{D})$. And although greedy algorithms often work well (see e.g., (Zhang 2008) for a theoretical analysis), they can of course get stuck in local optima.

Part of the problem is due to the fact that the γ_j variables are discrete, $\gamma_j \in \{0, 1\}$. In the optimization community, it is common to relax hard constraints of this form by replacing discrete variables with continuous variables. We can do this by replacing the spike-and-slab style prior, that assigns finite probability mass to the event that $w_j = 0$, to continuous priors that "encourage" $w_j = 0$ by putting a lot of probability density near the origin, such as a zero-mean Laplace distribution. This was first introduced in Section 7.4 in the context of robust linear regression. There we exploited the fact that the Laplace has heavy tails. Here we exploit the fact

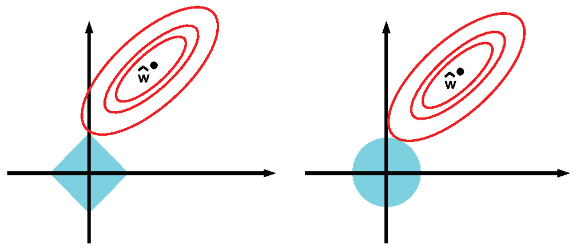


Figure 13.3 Illustration of ℓ_1 (left) vs ℓ_2 (right) regularization of a least squares problem. Based on Figure 3.12 of (Hastie et al. 2001).

that it has a spike near $\mu = 0$. More precisely, consider a prior of the form

$$p(\mathbf{w}|\lambda) = \prod_{j=1}^{D} \text{Lap}(w_j|0, 1/\lambda) \propto \prod_{j=1}^{D} e^{-\lambda|w_j|} \tag{13.32}$$

We will use a uniform prior on the offset term, $p(w_0) \propto 1$. Let us perform MAP estimation with this prior. The penalized negative log likelihood has the form

$$f(\mathbf{w}) = -\log p(\mathcal{D}|\mathbf{w}) - \log p(\mathbf{w}|\lambda) = \text{NLL}(\mathbf{w}) + \lambda||\mathbf{w}||_1 \tag{13.33}$$

where $||\mathbf{w}||_1 = \sum_{j=1}^{D} |w_j|$ is the ℓ_1 norm of $\mathbf{w}$. For suitably large λ, the estimate $\hat{\mathbf{w}}$ will be sparse, for reasons we explain below. Indeed, this can be thought of as a convex approximation to the non-convex ℓ_0 objective

$$\underset{\mathbf{w}}{\text{argmin}} \, \text{NLL}(\mathbf{w}) + \lambda||\mathbf{w}||_0 \tag{13.34}$$

In the case of linear regression, the ℓ_1 objective becomes

$$\begin{aligned} f(\mathbf{w}) &= \sum_{i=1}^{N} -\frac{1}{2\sigma^2}(y_i - (w_0 + \mathbf{w}^T\mathbf{x}_i))^2 + \lambda||\mathbf{w}||_1 && (13.35) \\ &= \text{RSS}(\mathbf{w}) + \lambda'||\mathbf{w}||_1 && (13.36) \end{aligned}$$

where $\lambda' = 2\lambda\sigma^2$. This method is known as **basis pursuit denoising** or **BPDN** (Chen et al. 1998). The reason for this term will become clear later. In general, the technique of putting a zero-mean Laplace prior on the parameters and performing MAP estimation is called ℓ_1 **regularization**. It can be combined with any convex or non-convex NLL term. Many different algorithms have been devised for solving such problems, some of which we review in Section 13.4.

13.3.1 Why does ℓ_1 regularization yield sparse solutions?

We now explain why ℓ_1 regularization results in sparse solutions, whereas ℓ_2 regularization does not. We focus on the case of linear regression, although similar arguments hold for logistic regression and other GLMs.

The objective is the following non-smooth objective function:

$$\min_{\mathbf{w}} \mathrm{RSS}(\mathbf{w}) + \lambda||\mathbf{w}||_1 \tag{13.37}$$

We can rewrite this as a constrained but smooth objective (a quadratic function with linear constraints):

$$\min_{\mathbf{w}} \mathrm{RSS}(\mathbf{w}) \quad \text{s.t.} \quad ||\mathbf{w}||_1 \leq B \tag{13.38}$$

where B is an upper bound on the ℓ_1-norm of the weights: a small (tight) bound B corresponds to a large penalty λ, and vice versa.[2] Equation 13.38 is known as **lasso**, which stands for "least absolute shrinkage and selection operator" (Tibshirani 1996). We will see why it has this name later.

Similarly, we can write ridge regression

$$\min_{\mathbf{w}} \mathrm{RSS}(\mathbf{w}) + \lambda||\mathbf{w}||_2^2 \tag{13.39}$$

or as a bound constrained form:

$$\min_{\mathbf{w}} \mathrm{RSS}(\mathbf{w}) \quad \text{s.t.} \quad ||\mathbf{w}||_2^2 \leq B \tag{13.40}$$

In Figure 13.3, we plot the contours of the RSS objective function, as well as the contours of the ℓ_2 and ℓ_1 constraint surfaces. From the theory of constrained optimization, we know that the optimal solution occurs at the point where the lowest level set of the objective function intersects the constraint surface (assuming the constraint is active). It should be geometrically clear that as we relax the constraint B, we "grow" the ℓ_1 "ball" until it meets the objective; the corners of the ball are more likely to intersect the ellipse than one of the sides, especially in high dimensions, because the corners "stick out" more. The corners correspond to sparse solutions, which lie on the coordinate axes. By contrast, when we grow the ℓ_2 ball, it can intersect the objective at any point; there are no "corners", so there is no preference for sparsity.

To see this another away, notice that, with ridge regression, the prior cost of a sparse solution, such as $\mathbf{w} = (1, 0)$, is the same as the cost of a dense solution, such as $\mathbf{w} = (1/\sqrt{2}, 1/\sqrt{2})$, as long as they have the same ℓ_2 norm:

$$||(1,0)||_2 = ||(1/\sqrt{2}, 1/\sqrt{2}||_2 = 1 \tag{13.41}$$

However, for lasso, setting $\mathbf{w} = (1, 0)$ is cheaper than setting $\mathbf{w} = (1/\sqrt{2}, 1/\sqrt{2})$, since

$$||(1,0)||_1 = 1 < ||(1/\sqrt{2}, 1/\sqrt{2}||_1 = \sqrt{2} \tag{13.42}$$

The most rigorous way to see that ℓ_1 regularization results in sparse solutions is to examine conditions that hold at the optimum. We do this in Section 13.3.2.

13.3.2 Optimality conditions for lasso

The lasso objective has the form

$$f(\boldsymbol{\theta}) = \mathrm{RSS}(\boldsymbol{\theta}) + \lambda||\mathbf{w}||_1 \tag{13.43}$$

2. Equation 13.38 is an example of a **quadratic program** or **QP**, since we have a quadratic objective subject to linear inequality constraints. Its Lagrangian is given by Equation 13.37.

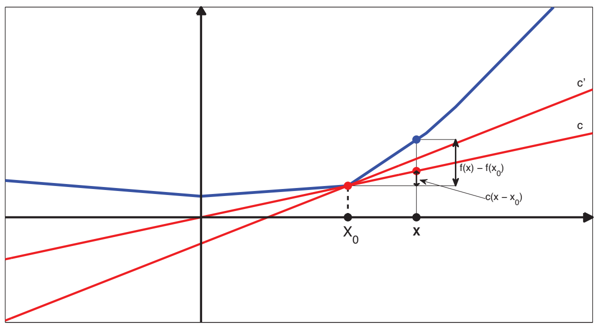


Figure 13.4 Illustration of some sub-derivatives of a function at point x_0. Based on a figure at `http://en.wikipedia.org/wiki/Subderivative`. Figure generated by `subgradientPlot`.

Unfortunately, the $||\mathbf{w}||_1$ term is not differentiable whenever $w_j = 0$. This is an example of a **non-smooth** optimization problem.

To handle non-smooth functions, we need to extend the notion of a derivative. We define a **subderivative** or **subgradient** of a (convex) function $f : \mathcal{I} \rightarrow \mathbb{R}$ at a point θ_0 to be a scalar g such that

$$f(\theta) - f(\theta_0) \geq g(\theta - \theta_0) \quad \forall \theta \in \mathcal{I} \tag{13.44}$$

where $\mathcal{I}$ is some interval containing θ_0. See Figure 13.4 for an illustration.[3] We define the *set* of subderivatives as the interval $[a, b]$ where a and b are the one-sided limits

$$a = \lim_{\theta \rightarrow \theta_0^-} \frac{f(\theta) - f(\theta_0)}{\theta - \theta_0}, \quad b = \lim_{\theta \rightarrow \theta_0^+} \frac{f(\theta) - f(\theta_0)}{\theta - \theta_0} \tag{13.46}$$

The set $[a, b]$ of all subderivatives is called the **subdifferential** of the function f at θ_0 and is denoted $\partial f(\theta)|_{\theta_0}$. For example, in the case of the absolute value function $f(\theta) = |\theta|$, the subderivative is given by

$$\partial f(\theta) = \begin{cases} \{-1\} & \text{if } \theta < 0 \\ [-1, 1] & \text{if } \theta = 0 \\ \{+1\} & \text{if } \theta > 0 \end{cases} \tag{13.47}$$

If the function is everywhere differentiable, then $\partial f(\theta) = \{\frac{df(\theta)}{d\theta}\}$. By analogy to the standard calculus result, one can show that the point $\hat{\theta}$ is a local minimum of f iff $0 \in \partial f(\theta)|\hat{\theta}$.

3. In general, for a vector valued function, we say that $\mathbf{g}$ is a subgradient of f at $\boldsymbol{\theta}_0$ if for all vectors $\boldsymbol{\theta}$,

$$f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}_0) \geq (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{g} \tag{13.45}$$

so $\mathbf{g}$ is a linear lower bound to the function at $\boldsymbol{\theta}_0$.

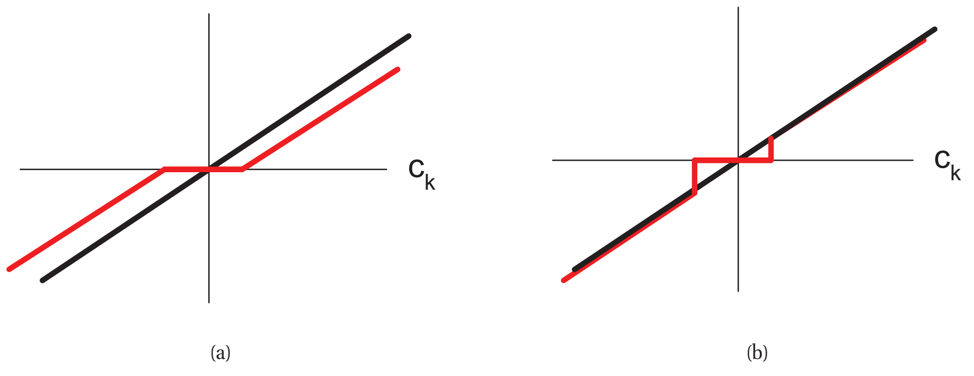


Figure 13.5 Left: soft thresholding. The flat region is the interval $[-\lambda, +\lambda]$. Right: hard thresholding.

Let us apply these concepts to the lasso problem. Let us initially ignore the non-smooth penalty term. One can show (Exercise 13.1) that

$$\frac{\partial}{\partial w_j}\text{RSS}(\mathbf{w}) = a_j w_j - c_j \tag{13.48}$$

$$a_j = 2\sum_{i=1}^{n} x_{ij}^2 \tag{13.49}$$

$$c_j = 2\sum_{i=1}^{n} x_{ij}(y_i - \mathbf{w}_{-j}^T \mathbf{x}_{i,-j}) \tag{13.50}$$

where $\mathbf{w}_{-j}$ is $\mathbf{w}$ without component j, and similarly for $\mathbf{x}_{i,-j}$. We see that c_j is (proportional to) the correlation between the j'th feature $\mathbf{x}_{:,j}$ and the residual due to the other features, $\mathbf{r}_{-j} = \mathbf{y} - \mathbf{X}_{:,-j}\mathbf{w}_{-j}$. Hence the magnitude of c_j is an indication of how relevant feature j is for predicting $\mathbf{y}$ (relative to the other features and the current parameters).

Adding in the penalty term, we find that the subderivative is given by

$$\partial_{w_j} f(\mathbf{w}) = (a_j w_j - c_j) + \lambda \partial_{w_j} ||\mathbf{w}||_1 \tag{13.51}$$

$$= \begin{cases} \{a_j w_j - c_j - \lambda\} & \text{if } w_j < 0 \\ [-c_j - \lambda, -c_j + \lambda] & \text{if } w_j = 0 \\ \{a_j w_j - c_j + \lambda\} & \text{if } w_j > 0 \end{cases} \tag{13.52}$$

We can write this in a more compact fashion as follows:

$$\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})_j \in \begin{cases} \{-\lambda\} & \text{if } w_j < 0 \\ [-\lambda, \lambda] & \text{if } w_j = 0 \\ \{\lambda\} & \text{if } w_j > 0 \end{cases} \tag{13.53}$$

Depending on the value of c_j, the solution to $\partial_{w_j} f(\mathbf{w}) = 0$ can occur at 3 different values of w_j, as follows:

1. If $c_j < -\lambda$, so the feature is strongly negatively correlated with the residual, then the subgradient is zero at $\hat{w}_j = \frac{c_j+\lambda}{a_j} < 0$.
2. If $c_j \in [-\lambda, \lambda]$, so the feature is only weakly correlated with the residual, then the subgradient is zero at $\hat{w}_j = 0$.
3. If $c_j > \lambda$, so the feature is strongly positively correlated with the residual, then the subgradient is zero at $\hat{w}_j = \frac{c_j-\lambda}{a_j} > 0$.

In summary, we have

$$\hat{w}_j(c_j) = \begin{cases} (c_j+\lambda)/a_j & \text{if } c_j < -\lambda \\ 0 & \text{if } c_j \in [-\lambda, \lambda] \\ (c_j-\lambda)/a_j & \text{if } c_j > \lambda \end{cases} \tag{13.54}$$

We can write this as follows:

$$\hat{w}_j = \text{soft}(\frac{c_j}{a_j}; \frac{\lambda}{a_j}) \tag{13.55}$$

where

$$\text{soft}\,(a;\delta) \triangleq \text{sign}(a)\,(|a|-\delta)_+ \tag{13.56}$$

and $x_+ = \max(x, 0)$ is the positive part of x. This is called **soft thresholding**. This is illustrated in Figure 13.5(a), where we plot $\hat{w}_j$ vs c_j. The dotted line is the line $w_j = c_j/a_j$ corresponding to the least squares fit. The solid line, which represents the regularized estimate $\hat{w}_j(c_j)$, shifts the dotted line down (or up) by λ, except when $-\lambda \le c_j \le \lambda$, in which case it sets $w_j = 0$.

By contrast, in Figure 13.5(b), we illustrate **hard thresholding**. This sets values of w_j to 0 if $-\lambda \le c_j \le \lambda$, but it does not shrink the values of w_j outside of this interval. The slope of the soft thresholding line does not coincide with the diagonal, which means that even large coefficients are shrunk towards zero; consequently lasso is a biased estimator. This is undesirable, since if the likelihood indicates (via c_j) that the coefficient w_j should be large, we do not want to shrink it. We will discuss this issue in more detail in Section 13.6.2.

Now we finally can understand why Tibshirani invented the term "lasso" in (Tibshirani 1996): it stands for "least absolute selection and shrinkage operator", since it selects a subset of the variables, and shrinks all the coefficients by penalizing the absolute values. If $\lambda = 0$, we get the OLS solution (of minimal ℓ_1 norm). If $\lambda \ge \lambda_{max}$, we get $\hat{\mathbf{w}} = \mathbf{0}$, where

$$\lambda_{max} = ||\mathbf{X}^T\mathbf{y}||_\infty = \max_j |\mathbf{y}^T\mathbf{x}_{:,j}| \tag{13.57}$$

This value is computed using the fact that **0** is optimal if $(\mathbf{X}^T\mathbf{y})_j \in [-\lambda, \lambda]$ for all j. In general, the maximum penalty for an ℓ_1 regularized objective is

$$\lambda_{max} = \max_j |\nabla_j NLL(\mathbf{0})| \tag{13.58}$$

13.3.3 Comparison of least squares, lasso, ridge and subset selection

We can gain further insight into ℓ_1 regularization by comparing it to least squares, and ℓ_2 and ℓ_0 regularized least squares. For simplicity, assume all the features of $\mathbf{X}$ are orthonormal, so $\mathbf{X}^T\mathbf{X} = \mathbf{I}$. In this case, the RSS is given by

$$\begin{aligned} \text{RSS}(\mathbf{w}) &= ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 = \mathbf{y}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} &\quad (13.59)\\ &= \text{const} + \sum_k w_k^2 - 2\sum_k\sum_i w_k x_{ik} y_i &\quad (13.60)\end{aligned}$$

so we see this factorizes into a sum of terms, one per dimension. Hence we can write down the MAP and ML estimates analytically, as follows:

- **MLE** The OLS solution is given by

  $$\hat{w}_k^{OLS} = \mathbf{x}_{:k}^T\mathbf{y} \quad (13.61)$$

  where $\mathbf{x}_{:k}$ is the k'th column of $\mathbf{X}$. This follows trivially from Equation 13.60. We see that $\hat{w}_k^{OLS}$ is just the orthogonal projection of feature k onto the response vector (see Section 7.3.2).
- **Ridge** One can show that the ridge estimate is given by

  $$\hat{w}_k^{ridge} = \frac{\hat{w}_k^{OLS}}{1+\lambda} \quad (13.62)$$

- **Lasso** From Equation 13.55, and using the fact that $a_k = 2$ and $\hat{w}_k^{OLS} = c_k/2$, we have

  $$\hat{w}_k^{lasso} = \text{sign}(\hat{w}_k^{OLS})\left(|\hat{w}_k^{OLS}| - \frac{\lambda}{2}\right)_+ \quad (13.63)$$

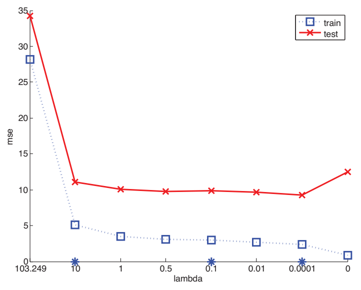
  This corresponds to soft thresholding, shown in Figure 13.5(a).
- **Subset selection** If we pick the best K features using subset selection, the parameter estimate is as follows

  $$\hat{w}_k^{SS} = \begin{cases} \hat{w}_k^{OLS} & \text{if rank}(|w_k^{OLS}|) \le K \\ 0 & \text{otherwise} \end{cases} \quad (13.64)$$

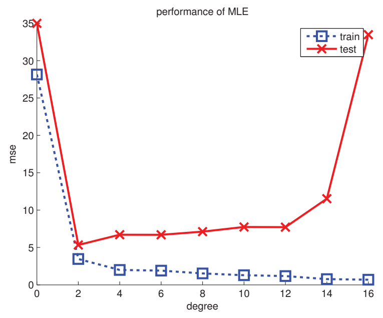
  where rank refers to the location in the sorted list of weight magnitudes. This corresponds to hard thresholding, shown in Figure 13.5(b).

Figure 13.6(a) plots the MSE vs λ for lasso for a degree 14 polynomial, and Figure 13.6(b) plots the MSE vs polynomial order. We see that lasso gives similar results to the subset selection method.

As another example, consider a data set concerning prostate cancer. We have $D = 8$ features and $N = 67$ training cases; the goal is to predict the log prostate-specific antigen levels (see (Hastie et al. 2009, p4) for more biological details). Table 13.1 shows that lasso gives better prediction accuracy (at least on this particular data set) than least squares, ridge, and best subset regression. (In each case, the strength of the regularizer was chosen by cross validation.) Lasso also gives rise to a sparse solution. Of course, for other problems, ridge may give better predictive accuracy. In practice, a combination of lasso and ridge, known as the elastic net, often performs best, since it provides a good combination of sparsity and regularization (see Section 13.5.3).



(a)



(b)

Figure 13.6 (a) MSE vs λ for lasso for a degree 14 polynomial. Note that λ decreases as we move to the right. Figure generated by `linregPolyLassoDemo`. (b) MSE versus polynomial degree. Note that the model order increases as we move to the right. See Figure 1.18 for a plot of some of these polynomial regression models. Figure generated by `linregPolyVsDegree`.

Term	LS	Best Subset	Ridge	Lasso
Intercept	2.452	2.481	2.479	2.480
lcavol	0.716	0.651	0.656	0.653
lweight	0.293	0.380	0.300	0.297
age	-0.143	-0.000	-0.129	-0.119
lbph	0.212	-0.000	0.208	0.200
svi	0.310	-0.000	0.301	0.289
lcp	-0.289	-0.000	-0.260	-0.236
gleason	-0.021	-0.000	-0.019	0.000
pgg45	0.277	0.178	0.256	0.226
Test Error	0.586	0.572	0.580	0.564

Table 13.1 Results of different methods on the prostate cancer data, which has 8 features and 67 training cases. Methods are: LS = least squares, Subset = best subset regression, Ridge, Lasso. Rows represent the coefficients; we see that subset regression and lasso give sparse solutions. Bottom row is the mean squared error on the test set (30 cases). Based on Table 3.3. of (Hastie et al. 2009). Figure generated by `prostateComparison`.

13.3.4 Regularization path

As we increase λ, the solution vector $\hat{\mathbf{w}}(\lambda)$ will tend to get sparser, although not necessarily monotonically. We can plot the values $\hat{w}_j(\lambda)$ vs λ for each feature j; this is known as the **regularization path**.

This is illustrated for ridge regression in Figure 13.7(a), where we plot $\hat{w}_j(\lambda)$ as the regularizer λ decreases. We see that when $\lambda = \infty$, all the coefficients are zero. But for any finite value of λ, all coefficients are non-zero; furthermore, they increase in magnitude as λ is decreased.

In Figure 13.7(b), we plot the analogous result for lasso. As we move to the right, the upper bound on the ℓ_1 penalty, B, increases. When $B = 0$, all the coefficients are zero. As we increase

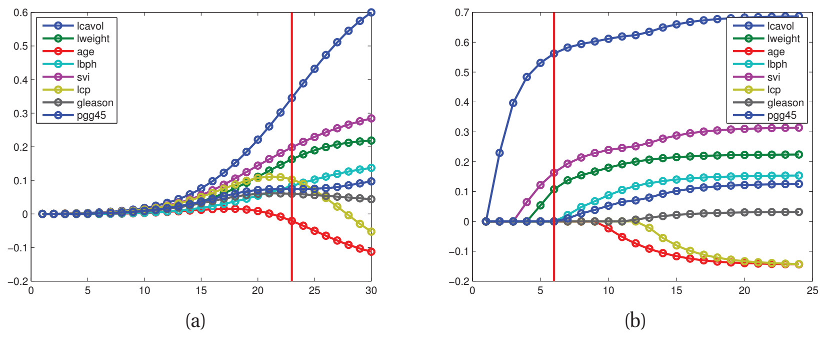


Figure 13.7 (a) Profiles of ridge coefficients for the prostate cancer example vs bound on ℓ_2 norm of $\mathbf{w}$, so small t (large λ) is on the left. The vertical line is the value chosen by 5-fold CV using the 1SE rule. Based on Figure 3.8 of (Hastie et al. 2009). Figure generated by `ridgePathProstate`. (b) Profiles of lasso coefficients for the prostate cancer example vs bound on ℓ_1 norm of $\mathbf{w}$, so small t (large λ) is on the left. Based on Figure 3.10 of (Hastie et al. 2009). Figure generated by `lassoPathProstate`.

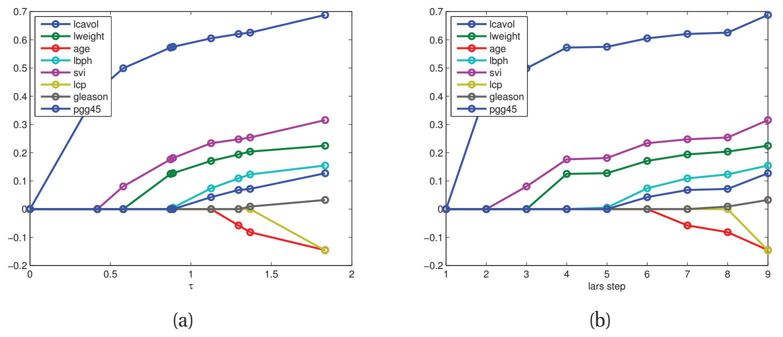


Figure 13.8 Illustration of piecewise linearity of regularization path for lasso on the prostate cancer example. (a) We plot $\hat{w}_j(B)$ vs B for the critical values of B. (b) We plot vs steps of the LARS algorithm. Figure generated by `lassoPathProstate`.

B, the coefficients gradually "turn on". But for any value between 0 and $B_{max} = ||\hat{\mathbf{w}}_{OLS}||_1$, the solution is sparse.[4]

Remarkably, it can be shown that the solution path is a piecewise linear function of B (Efron et al. 2004). That is, there are a set of critical values of B where the active set of non-zero coefficients changes. For values of B between these critical values, each non-zero coefficient increases or decreases in a linear fashion. This is illustrated in Figure 13.8(a). Furthermore, one can solve for these critical values analytically. This is the basis of the **LARS** algorithm (Efron et al. 2004), which stands for "least angle regression and shrinkage" (see Section 13.4.2 for details). Remarkably, LARS can compute the entire regularization path for roughly the same

4. It is common to plot the solution versus the **shrinkage factor**, defined as $s(B) = B/B_{max}$, rather than against B. This merely affects the scale of the horizontal axis, not the shape of the curves.

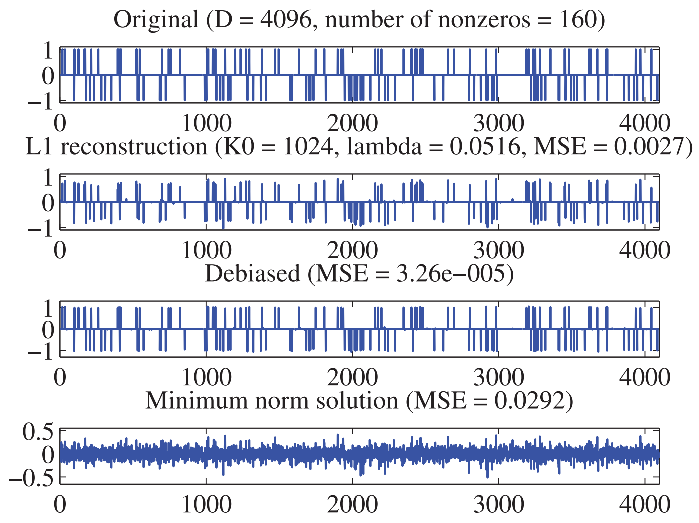


Figure 13.9 Example of recovering a sparse signal using lasso. See text for details. Based on Figure 1 of (Figueiredo et al. 2007). Figure generated by `sparseSensingDemo`, written by Mario Figueiredo.

computational cost as a single least squares fit (namely $O(\min(ND^2, DN^2))$.

In Figure 13.8(b), we plot the coefficients computed at each critical value of B. Now the piecewise linearity is more evident. Below we display the actual coefficient values at each step along the regularization path (the last line is the least squares solution):

Listing 13.1 Output of `lassoPathProstate`

```
        0         0         0         0         0         0         0         0
   0.4279         0         0         0         0         0         0         0
   0.5015    0.0735         0         0         0         0         0         0
   0.5610    0.1878         0         0    0.0930         0         0         0
   0.5622    0.1890         0    0.0036    0.0963         0         0         0
   0.5797    0.2456         0    0.1435    0.2003         0         0    0.0901
   0.5864    0.2572   -0.0321    0.1639    0.2082         0         0    0.1066
   0.6994    0.2910   -0.1337    0.2062    0.3003   -0.2565         0    0.2452
   0.7164    0.2926   -0.1425    0.2120    0.3096   -0.2890   -0.0209    0.2773
```

By changing B from 0 to B_{max}, we can go from a solution in which all the weights are zero to a solution in which all weights are non-zero. Unfortunately, not all subset sizes are achievable using lasso. One can show that, if $D > N$, the optimal solution can have at most N variables in it, before reaching the complete set corresponding to the OLS solution of minimal ℓ_1 norm. In Section 13.5.3, we will see that by using an ℓ_2 regularizer as well as an ℓ_1 regularizer (a method known as the elastic net), we can achieve sparse solutions which contain more variables than training cases. This lets us explore model sizes between N and D.

13.3.5 Model selection

It is tempting to use ℓ_1 regularization to estimate the set of relevant variables. In some cases, we can recover the true sparsity pattern of $\mathbf{w}^*$, the parameter vector that generated the data. A method that can recover the true model in the $N \to \infty$ limit is called **model selection consistent**. The details on which methods enjoy this property, and when, are beyond the scope of this book; see e.g., (Buhlmann and van de Geer 2011) for details.

Instead of going into a theoretical discussion, we will just show a small example. We first generate a sparse signal $\mathbf{w}^*$ of size $D = 4096$, consisting of 160 randomly placed ± 1 spikes. Next we generate a random design matrix $\mathbf{X}$ of size $N \times D$, where $N = 1024$. Finally we generate a noisy observation $\mathbf{y} = \mathbf{X}\mathbf{w}^* + \boldsymbol{\epsilon}$, where $\epsilon_i \sim \mathcal{N}(0, 0.01^2)$. We then estimate $\mathbf{w}$ from $\mathbf{y}$ and $\mathbf{X}$.

The original $\mathbf{w}^*$ is shown in the first row of Figure 13.9. The second row is the ℓ_1 estimate $\hat{\mathbf{w}}_{L1}$ using $\lambda = 0.1\lambda_{max}$. We see that this has "spikes" in the right places, but they are too small. The third row is the least squares estimate of the coefficients which are estimated to be non-zero based on $\text{supp}(\hat{\mathbf{w}}_{L1})$. This is called **debiasing**, and is necessary because lasso shrinks the relevant coefficients as well as the irrelevant ones. The last row is the least squares estimate for all the coefficients jointly, ignoring sparsity. We see that the (debiased) sparse estimate is an excellent estimate of the original signal. By contrast, least squares without the sparsity assumption performs very poorly.

Of course, to perform model selection, we have to pick λ. It is common to use cross validation. However, it is important to note that cross validation is picking a value of λ that results in good predictive accuracy. This is not usually the same value as the one that is likely to recover the "true" model. To see why, recall that ℓ_1 regularization performs selection *and* shrinkage, that is, the chosen coefficients are brought closer to 0. In order to prevent relevant coefficients from being shrunk in this way, cross validation will tend to pick a value of λ that is not too large. Of course, this will result in a less sparse model which contains irrelevant variables (false positives). Indeed, it was proved in (Meinshausen and Buhlmann 2006) that the prediction-optimal value of λ does not result in model selection consistency. In Section 13.6.2, we will discuss some adaptive mechanisms for automatically tuning λ on a per-dimension basis that does result in model selection consistency.

A downside of using ℓ_1 regularization to select variables is that it can give quite different results if the data is perturbed slightly. The Bayesian approach, which estimates posterior marginal inclusion probabilities, $p(\gamma_j = 1|\mathcal{D})$, is much more robust. A frequentist solution to this is to use bootstrap resampling (see Section 6.2.1), and to rerun the estimator on different versions of the data. By computing how often each variable is selected across different trials, we can approximate the posterior inclusion probabilities. This method is known as **stability selection** (Meinshausen and BÃijhlmann 2010).

We can threshold the stability selection (bootstrap) inclusion probabilities at some level, say 90%, and thus derive a sparse estimator. This is known as **bootstrap lasso** or **bolasso** (Bach 2008). It will include a variable if it occurs in at least 90% of sets returned by lasso (for a fixed λ). This process of intersecting the sets is a way of eliminating the false positives that vanilla lasso produces. The theoretical results in (Bach 2008) prove that bolasso is model selection consistent under a wider range of conditions than vanilla lasso.

As an illustration, we reproduced the experiments in (Bach 2008). In particular, we created

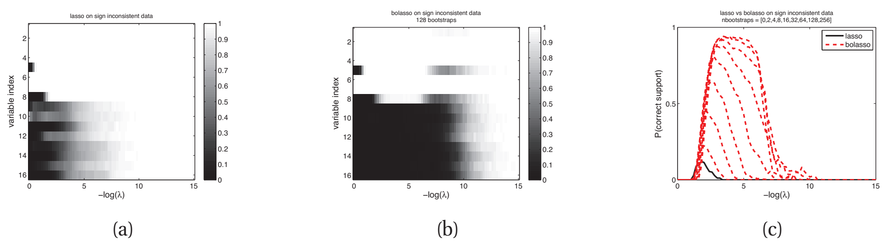


Figure 13.10 (a) Probability of selection of each variable (white = large probabilities, black = small probabilities) vs. regularization parameter for Lasso. As we move from left to right, we decrease the amount of regularization, and therefore select more variables. (b) Same as (a) but for bolasso. (c) Probability of correct sign estimation vs. regularization parameter. Bolasso (red, dashed) and Lasso (black, plain): The number of bootstrap replications is in $\{2, 4, 8, 16, 32, 64, 128, 256\}$. Based on Figures 1-3 of (Bach 2008). Figure generated by `bolassoDemo`.

256 datasets of size $N = 1000$ with $D = 16$ variables, of which 8 are relevant. See (Bach 2008) for more detail on the experimental setup. For dataset n, variable j, and sparsity level k, define $S(j, k, n) = \mathbb{I}(\hat{w}_j(\lambda_k, \mathcal{D}_n) \neq 0)$. Now define$P(j, k)$ be the average of $S(j, k, n)$ over the 256 datasets. In Figure 13.10(a-b), we plot P vs $-\log(\lambda)$ for lasso and bolasso. We see that for bolasso, there is a large range of λ where the true variables are selected, but this is not the case for lasso. This is emphasized in Figure 13.10(c), where we plot the empirical probability that the correct set of variables is recovered, for lasso and for bolasso with an increasing number of bootstrap samples. Of course, using more samples takes longer. In practice, 32 bootstraps seems to be a good compromise between speed and accuracy.

With bolasso, there is the usual issue of picking λ. Obviously we could use cross validation, but plots such as Figure 13.10(b) suggest another heuristic: shuffle the rows to create a large black block, and then pick λ to be in the middle of this region. Of course, operationalizing this intuition may be tricky, and will require various ad-hoc thresholds (it is reminiscent of the "find the knee in the curve" heuristic discussed in Section 11.5.2 when discussing how to pick K for mixture models). A Bayesian approach provides a more principled method for selecting λ.

13.3.6 Bayesian inference for linear models with Laplace priors

We have been focusing on MAP estimation in sparse linear models. It is also possible to perform Bayesian inference (see e.g., (Park and Casella 2008; Seeger 2008)). However, the posterior mean and median, as well as samples from the posterior, are not sparse; only the mode is sparse. This is another example of the phenomenon discussed in Section 5.2.1, where we said that the MAP estimate is often untypical of the bulk of the posterior.

Another argument in favor of using the posterior mean comes from Equation 5.108, which showed that that plugging in the posterior mean, rather than the posterior mode, is the optimal thing to do if we want to minimize squared prediction error. (Schniter et al. 2008) shows experimentally, and (Elad and Yavnch 2009) shows theoretically, that using the posterior mean with a spike-and-slab prior results in better prediction accuracy than using the posterior mode with a Laplace prior, albeit at slightly higher computational cost.

13.4 ℓ_1 regularization: algorithms

In this section, we give a brief review of some algorithms that can be used to solve ℓ_1 regularized estimation problems. We focus on the lasso case, where we have a quadratic loss. However, most of the algorithms can be extended to more general settings, such as logistic regression (see (Yaun et al. 2010) for a comprehensive review of ℓ_1 regularized logistic regression). Note that this area of machine learning is advancing very rapidly, so the methods below may not be state of the art by the time you read this chapter. (See (Schmidt et al. 2009; Yaun et al. 2010; Yang et al. 2010) for some recent surveys.)

13.4.1 Coordinate descent

Sometimes it is hard to optimize all the variables simultaneously, but it easy to optimize them one by one. In particular, we can solve for the j'th coefficient with all the others held fixed:

$$w_j^* = \operatorname*{argmin}_z f(\mathbf{w} + z\mathbf{e}_j) - f(\mathbf{w}) \tag{13.65}$$

where $\mathbf{e}_j$ is the j'th unit vector. We can either cycle through the coordinates in a deterministic fashion, or we can sample them at random, or we can choose to update the coordinate for which the gradient is steepest.

The coordinate descent method is particularly appealing if each one-dimensional optimization problem can be solved analytically For example, the **shooting** algorithm (Fu 1998; Wu and Lange 2008) for lasso uses Equation 13.54 to compute the optimal value of w_j given all the other coefficients. See Algorithm 7 for the pseudo code (and `LassoShooting` for some Matlab code).

See (Yaun et al. 2010) for some extensions of this method to the logistic regression case. The resulting algorithm was the fastest method in their experimental comparison, which concerned document classification with large sparse feature vectors (representing bags of words). Other types of data (e.g., dense features and/or regression problems) might call for different algorithms.

Algorithm 13.1: Coordinate descent for lasso (aka shooting algorithm)

```
1 Initialize $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$;
2 repeat
3     for $j = 1, \ldots, D$ do
4         $a_j = 2\sum_{i=1}^n x_{ij}^2$;
5         $c_j = 2\sum_{i=1}^n x_{ij}(y_i - \mathbf{w}^T\mathbf{x}_i + w_j x_{ij})$ ;
6         $w_j = \text{soft}(\frac{c_j}{a_j}, \frac{\lambda}{a_j})$;
7 until converged;
```

13.4.2 LARS and other homotopy methods

The problem with coordinate descent is that it only updates one variable at a time, so can be slow to converge. **Active set** methods update many variables at a time. Unfortunately, they are more complicated, because of the need to identify which variables are constrained to be zero, and which are free to be updated.

Active set methods typically only add or remove a few variables at a time, so they can take a long if they are started far from the solution. But they are ideally suited for generating a set of solutions for different values of λ, starting with the empty set, i.e., for generating regularization path. These algorithms exploit the fact that one can quickly compute $\hat{\mathbf{w}}(\lambda_k)$ from $\hat{\mathbf{w}}(\lambda_{k-1})$ if $\lambda_k \approx \lambda_{k-1}$; this is known as **warm starting**. In fact, even if we only want the solution for a single value of λ, call it λ_*, it can sometimes be computationally more efficient to compute a set of solutions, from λ_{max} down to λ_*, using warm-starting; this is called a **continuation method** or **homotopy** method. This is often much faster than directly "cold-starting" at λ_*; this is particularly true if λ_* is small.

Perhaps the most well-known example of a homotopy method in machine learning is the **LARS** algorithm, which stands for "least angle regression and shrinkage" (Efron et al. 2004) (a similar algorithm was independently invented in (Osborne et al. 2000b,a)). This can compute $\hat{\mathbf{w}}(\lambda)$ for all possible values of λ in an efficient manner.

LARS works as follows. It starts with a large value of λ, such that only the variable that is most correlated with the response vector $\mathbf{y}$ is chosen. Then λ is decreased until a second variable is found which has the same correlation (in terms of magnitude) with the current residual as the first variable, where the residual at step k is defined as $\mathbf{r}_k = \mathbf{y} - \mathbf{X}_{:,F_k}\mathbf{w}_k$, where F_k is the current **active set** (c.f., Equation 13.50). Remarkably, one can solve for this new value of λ analytically, by using a geometric argument (hence the term "least angle"). This allows the algorithm to quickly "jump" to the next point on the regularization path where the active set changes. This repeats until all the variables are added.

It is necessary to allow variables to be removed from the active set if we want the sequence of solutions to correspond to the regularization path of lasso. If we disallow variable removal, we get a slightly different algorithm called **LAR**, which tends to be faster. In particular, LAR costs the same as a single ordinary least squares fit, namely $O(ND\min(N, D))$, which is $O(ND^2)$ if $N > D$, and $O(N^2D)$ if $D > N$. LAR is very similar to greedy forward selection, and a method known as least squares boosting (see Section 16.4.6).

There have been many attempts to extend the LARS algorithm to compute the full regularization path for ℓ_1 regularized GLMs, such as logistic regression. In general, one cannot analytically solve for the critical values of λ. Instead, the standard approach is to start at $\lambda_{\max}$, and then slowly decrease λ, tracking the solution as we go; this is called a **continuation method** or **homotopy** method. These methods exploit the fact that we can quickly compute $\hat{\mathbf{w}}(\lambda_k)$ from $\hat{\mathbf{w}}(\lambda_{k-1})$ if $\lambda_k \approx \lambda_{k-1}$; this is known as **warm starting**. Even if we don't want the full path, this method is often much faster than directly "cold-starting" at the desired value of λ (this is particularly true if λ is small).

The method described in (Friedman et al. 2010) combines coordinate descent with this warm-starting strategy, and computes the full regularization path for any ℓ_1 regularized GLM. This has been implemented in the glmnet package, which is bundled with PMTK.

13.4.3 Proximal and gradient projection methods

In this section, we consider some methods that are suitable for very large scale problems, where homotopy methods made be too slow. These methods will also be easy to extend to other kinds of regularizers, beyond ℓ_1, as we will see later. Our presentation in this section is based on (Vandenberghe 2011; Yang et al. 2010).

Consider a convex objective of the form

$$f(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + R(\boldsymbol{\theta}) \tag{13.66}$$

where $L(\boldsymbol{\theta})$ (representing the loss) is convex and differentiable, and $R(\boldsymbol{\theta})$ (representing the regularizer) is convex but not necessarily differentiable. For example, $L(\boldsymbol{\theta}) = \text{RSS}(\boldsymbol{\theta})$ and $R(\boldsymbol{\theta}) = \lambda||\boldsymbol{\theta}||_1$ corresponds to the BPDN problem. As another example, the lasso problem can be formulated as follows: $L(\boldsymbol{\theta}) = \text{RSS}(\boldsymbol{\theta})$ and $R(\boldsymbol{\theta}) = I_C(\boldsymbol{\theta})$, where $C = \{\boldsymbol{\theta} : ||\boldsymbol{\theta}||_1 \leq B\}$, and $I_C(\boldsymbol{\theta})$ is the indicator function of a convex set C, defined as

$$I_C(\boldsymbol{\theta}) \triangleq \begin{cases} 0 & \boldsymbol{\theta} \in C \\ +\infty & \text{otherwise} \end{cases} \tag{13.67}$$

In some cases, it is easy to optimize functions of the form in Equation 13.66. For example, suppose $L(\boldsymbol{\theta}) = \text{RSS}(\boldsymbol{\theta})$, and the design matrix is simply $\mathbf{X} = \mathbf{I}$. Then the obective becomes $f(\boldsymbol{\theta}) = R(\boldsymbol{\theta}) + \frac{1}{2}||\boldsymbol{\theta} - \mathbf{y}|_2^2$. The minimizer of this is given by $\text{prox}_R(\mathbf{y})$, which is the **proximal operator** for the convex function R, defined by

$$\text{prox}_R(\mathbf{y}) = \underset{\mathbf{z}}{\text{argmin}} \left(R(\mathbf{z}) + \frac{1}{2}||\mathbf{z} - \mathbf{y}||_2^2 \right) \tag{13.68}$$

Intuitively, we are returning a point that minimizes R but which is also close (proximal) to $\mathbf{y}$. In general, we will use this operator inside an iterative optimizer, in which case we want to stay close to the previous iterate. In this case, we use

$$\text{prox}_R(\boldsymbol{\theta}_k) = \underset{\mathbf{z}}{\text{argmin}} \left(R(\mathbf{z}) + \frac{1}{2}||\mathbf{z} - \boldsymbol{\theta}_k||_2^2 \right) \tag{13.69}$$

The key issues are: how do we efficiently compute the proximal operator for different regularizers R, and how do we extend this technique to more general loss functions L? We discuss these issues below.

13.4.3.1 Proximal operators

If $R(\boldsymbol{\theta}) = \lambda||\boldsymbol{\theta}||_1$, the proximal operator is given by componentwise soft-thresholding:

$$\text{prox}_R(\boldsymbol{\theta}) = \text{soft}(\boldsymbol{\theta}, \lambda) \tag{13.70}$$

as we showed in Section 13.3.2. If $R(\boldsymbol{\theta}) = \lambda||\boldsymbol{\theta}||_0$, the proximal operator is given by componentwise hard-thresholding:

$$\text{prox}_R(\boldsymbol{\theta}) = \text{hard}(\boldsymbol{\theta}, \sqrt{2\lambda}) \tag{13.71}$$

where $\text{hard}(u, a) \triangleq u\mathbb{I}(|u| > a)$.

If $R(\boldsymbol{\theta}) = I_C(\boldsymbol{\theta})$, the proximal operator is given by the projection onto the set C:

$$\text{prox}_R(\boldsymbol{\theta}) = \underset{\mathbf{z} \in C}{\text{argmin}} \, ||\mathbf{z} - \boldsymbol{\theta}||_2^2 = \text{proj}_C(\boldsymbol{\theta}) \tag{13.72}$$

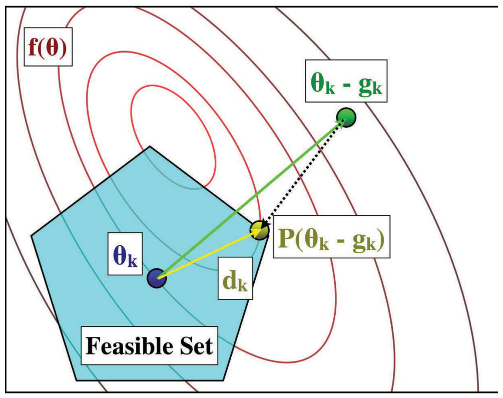


Figure 13.11 Illustration of projected gradient descent. The step along the negative gradient, to $\boldsymbol{\theta}_k - \mathbf{g}_k$, takes us outside the feasible set. If we project that point onto the closest point in the set we get $\boldsymbol{\theta}_{k+1} = \text{proj}_\Theta(\boldsymbol{\theta}_k - \mathbf{g}_k)$. We can then derive the implicit update direction using $\mathbf{d}_k = \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k$. Used with kind permission of Mark Schmidt.

For some convex sets, it is easy to compute the projection operator. For example, to project onto the rectangular set defined by the box constraints $C = \{\boldsymbol{\theta} : \ell_j \le \theta_j \le u_j\}$ we can use

$$\text{proj}_C(\boldsymbol{\theta})_j = \begin{cases} \ell_j & \theta_j \le \ell_j \\ \theta_j & \ell_j \le \theta_j \le u_j \\ u_j & \theta_j \ge u_j \end{cases} \tag{13.73}$$

To project onto the Euclidean ball $C = \{\boldsymbol{\theta} : ||\boldsymbol{\theta}||_2 \le 1\}$ we can use

$$\text{proj}_C(\boldsymbol{\theta}) = \begin{cases} \frac{\boldsymbol{\theta}}{||\boldsymbol{\theta}||_2} & ||\boldsymbol{\theta}||_2 > 1 \\ \boldsymbol{\theta} & ||\boldsymbol{\theta}||_2 \le 1 \end{cases} \tag{13.74}$$

To project onto the 1-norm ball $C = \{\boldsymbol{\theta} : ||\boldsymbol{\theta}||_1 \le 1\}$ we can use

$$\text{proj}_C(\boldsymbol{\theta}) = \text{soft}(\boldsymbol{\theta}, \lambda) \tag{13.75}$$

where $\lambda = 0$ if $||\boldsymbol{\theta}||_1 \le 1$, and otherwise λ is the solution to the equation

$$\sum_{j=1}^{D} \max(|\theta_j| - \lambda, 0) = 1 \tag{13.76}$$

We can implement the whole procedure in $O(D)$ time, as explained in (Duchi et al. 2008).

We will see an application of these different projection methods in Section 13.5.1.2.

13.4.3.2 Proximal gradient method

We now discuss how to use the proximal operator inside of a gradient descent routine. The basic idea is to minimize a simple quadratic approximation to the loss function, centered on the

$\boldsymbol{\theta}_k$:

$$\boldsymbol{\theta}_{k+1} = \operatorname*{argmin}_{\mathbf{z}} R(\mathbf{z}) + L(\boldsymbol{\theta}_k) + \mathbf{g}_k^T(\mathbf{z} - \boldsymbol{\theta}_k) + \frac{1}{2t_k}||\mathbf{z} - \boldsymbol{\theta}_k||_2^2 \tag{13.77}$$

where $\mathbf{g}_k = \nabla L(\boldsymbol{\theta}_k)$ is the gradient of the loss, t_k is a constant discussed below, and the last term arises from a simple approximation to the Hessian of the loss of the form $\nabla^2 L(\boldsymbol{\theta}_k) \approx \frac{1}{t_k}\mathbf{I}$.

Dropping terms that are independent of $\mathbf{z}$, and multiplying by t_k, we can rewrite the above expression in terms of a proximal operator as follows:

$$\begin{aligned}
\boldsymbol{\theta}_{k+1} &= \operatorname*{argmin}_{\mathbf{z}} \left[t_k R(\mathbf{z}) + \frac{1}{2}||\mathbf{z} - \mathbf{u}_k||_2^2\right] = \text{prox}_{t_k R}(\mathbf{u}_k) && (13.78)\\
\mathbf{u}_k &= \boldsymbol{\theta}_k - t_k \mathbf{g}_k && (13.79)\\
\mathbf{g}_k &= \nabla L(\boldsymbol{\theta}_k) && (13.80)
\end{aligned}$$

If $R(\boldsymbol{\theta}) = 0$, this is equivalent to gradient descent. If $R(\boldsymbol{\theta}) = I_C(\boldsymbol{\theta})$, the method is equivalent to **projected gradient descent**, sketched in Figure 13.11. If $R(\boldsymbol{\theta}) = \lambda||\boldsymbol{\theta}||_1$, the method is known as **iterative soft thresholding**.

There are several ways to pick t_k, or equivalently, $\alpha_k = 1/t_k$. Given that $\alpha_k \mathbf{I}$ is an approximation to the Hessian $\nabla^2 L$, we require that

$$\alpha_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}) \approx \mathbf{g}_k - \mathbf{g}_{k-1} \tag{13.81}$$

in the least squares sense. Hence

$$\alpha_k = \operatorname*{argmin}_{\alpha} ||\alpha(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}) - (\mathbf{g}_k - \mathbf{g}_{k-1})||_2^2 = \frac{(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})^T(\mathbf{g}_k - \mathbf{g}_{k-1})}{(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})^T(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})} \tag{13.82}$$

This is known as the **Barzilai-Borwein** (BB) or **spectral** stepsize (Barzilai and Borwein 1988; Fletcher 2005; Raydan 1997). This stepsize can be used with any gradient method, whether proximal or not. It does not lead to monotonic decrease of the objective, but it is much faster than standard line search techniques. (To ensure convergence, we require that the objective decrease "on average", where the average is computed over a sliding window of size $M + 1$.)

When we combine the BB stepsize with the iterative soft thresholding technique (for $R(\boldsymbol{\theta}) = \lambda||\boldsymbol{\theta}||_1$), plus a continuation method that gradually reduces λ, we get a fast method for the BPDN problem known as the SpaRSA algorithm, which stands for "sparse reconstruction by separable approximation" (Wright et al. 2009). However, we will call it the iterative shrinkage and thresholding algorithm. See Algorithm 12 for some pseudocode, and `SpaRSA` for some Matlab code. See also Exercise 13.11 for a related approach based on projected gradient descent.

13.4.3.3 Nesterov's method

A faster version of proximal gradient descent can be obtained by epxanding the quadratic approximation around a point other than the most recent parameter value. In particular, consider performing updates of the form

$$\begin{aligned}
\boldsymbol{\theta}_{k+1} &= \text{prox}_{t_k R}(\boldsymbol{\phi}_k - t_k \mathbf{g}_k) && (13.83)\\
\mathbf{g}_k &= \nabla L(\boldsymbol{\phi}_k) && (13.84)\\
\boldsymbol{\phi}_k &= \boldsymbol{\theta}_k + \frac{k-1}{k+2}(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}) && (13.85)
\end{aligned}$$

Algorithm 13.2: Iterative Shrinkage-Thresholding Algorithm (ISTA)

```
1  Input: $\mathbf{X} \in \mathbb{R}^{N \times D}$, $\mathbf{y} \in \mathbb{R}^N$, parameters $\lambda \geq 0$, $M \geq 1$, $0 < s < 1$ ;
2  Initialize $\boldsymbol{\theta}_0 = \mathbf{0}$, $\alpha = 1$, $\mathbf{r} = \mathbf{y}$, $\lambda_0 = \infty$;
3  repeat
4      $\lambda_t = \max(s||\mathbf{X}^T\mathbf{r}||_\infty, \lambda)$ // Adapt the regularizer ;
5      repeat
6          $\mathbf{g} = \nabla L(\boldsymbol{\theta})$;
7          $\mathbf{u} = \boldsymbol{\theta} - \frac{1}{\alpha}\mathbf{g}$;
8          $\boldsymbol{\theta} = \text{soft}(\mathbf{u}, \frac{\lambda_t}{\alpha})$;
9          Update $\alpha$ using BB stepsize in Equation 13.82 ;
10     until $f(\boldsymbol{\theta})$ increased too much within the past $M$ steps;
11     $\mathbf{r} = \mathbf{y} - \mathbf{X}\boldsymbol{\theta}$ // Update residual ;
12 until $\lambda_t = \lambda$;
```

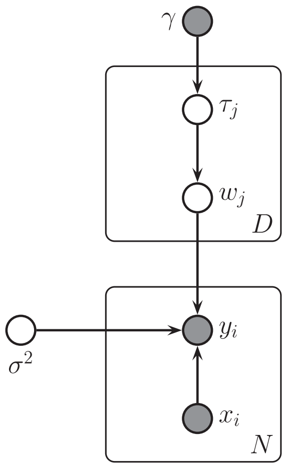


Figure 13.12 Representing lasso using a Gaussian scale mixture prior.

This is known as **Nesterov's method** (Nesterov 2004; Tseng 2008). As before, there are a variety of ways of setting t_k; typically one uses line search.

When this method is combined with the iterative soft thresholding technique (for $R(\boldsymbol{\theta}) = \lambda||\boldsymbol{\theta}||_1$), plus a continuation method that gradually reduces λ, we get a fast method for the BPDN problem known as the **fast iterative shrinkage thesholding algorithm** or **FISTA** (Beck and Teboulle 2009).

13.4.4 EM for lasso

In this section, we show how to solve the lasso problem using lasso. At first sight, this might seem odd, since there are no hidden variables. The key insight is that we can represent the Laplace distribution as a **Gaussian scale mixture** (GSM) (Andrews and Mallows 1974; West 1987) as follows:

$$\text{Lap}(w_j|0, 1/\gamma) = \frac{\gamma}{2}e^{-\gamma|w_j|} = \int \mathcal{N}(w_j|0, \tau_j^2)\text{Ga}(\tau_j^2|1, \frac{\gamma^2}{2})d\tau_j^2 \tag{13.86}$$

Thus the Laplace is a GSM where the mixing distibution on the variances is the exponential distribution, $\text{Expon}(\tau_j^2|\frac{\gamma^2}{2} = \text{Ga}(\tau_j^2|1, \frac{\gamma^2}{2})$. Using this decomposition, we can represent the lasso model as shown in Figure 13.12. The corresponding joint distribution has the form

$$\begin{aligned} p(\mathbf{y}, \mathbf{w}, \boldsymbol{\tau}, \sigma^2|\mathbf{X}) &= \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I}_N)\ \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{D}_\tau) \\ & \quad \text{IG}(\sigma^2|a_\sigma, b_\sigma)\ \left[\prod_j \text{Ga}(\tau_j^2|1, \gamma^2/2)\right] \end{aligned} \tag{13.87}$$

where $\mathbf{D}_\tau = \text{diag}(\tau_j^2)$, and where we have assumed for notational simplicity that $\mathbf{X}$ is standardized and that $\mathbf{y}$ is centered (so we can ignore the offset term μ). Expanding out, we get

$$\begin{aligned} p(\mathbf{y}, \mathbf{w}, \boldsymbol{\tau}, \sigma^2|\mathbf{X}) &\propto \left(\sigma^2\right)^{-N/2} \exp\left(-\frac{1}{2\sigma^2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2\right)\ |\mathbf{D}_\tau|^{-\frac{1}{2}} \\ & \quad \exp\left(-\frac{1}{2}\mathbf{w}^T\mathbf{D}_\tau\mathbf{w}\right)(\sigma^2)^{-(a_\sigma+1)} \\ & \quad \exp(-b_\sigma/\sigma^2)\prod_j \exp(-\frac{\gamma^2}{2}\tau_j^2) \end{aligned} \tag{13.88}$$

Below we describe how to apply the EM algorithm to the model in Figure 13.12.[5] In brief, in the E step we infer τ_j^2 and σ^2, and in the M step we estimate $\mathbf{w}$. The resulting estimate $\hat{\mathbf{w}}$ is the same as the lasso estimator. This approach was first proposed in (Figueiredo 2003) (see also (Griffin and Brown 2007; Caron and Doucet 2008; Ding and Harrison 2010) for some extensions).

13.4.4.1 Why EM?

Before going into the details of EM, it is worthwhile asking why we are presenting this approach at all, given that there are a variety of other (often much faster) algorithms that directly solve the ℓ_1 MAP estimation problem (see `linregFitL1Test` for an empirical comparison). The reason is that the latent variable perspective brings several advantages, such as the following:

- It provides an easy way to derive an algorithm to find ℓ_1-regularized parameter estimates for a variety of other models, such as robust linear regression (Exercise 11.12) or probit regression (Exercise 13.9).

5. To ensure the posterior is unimodal, one can follow (Park and Casella 2008) and slightly modify the model by making the prior variance for the weights depend on the observation noise: $p(w_j|\tau_j^2, \sigma^2) = \mathcal{N}(w_j|0, \sigma^2\tau_j^2)$. The EM algorithm is easy to modify.

- It suggests trying other priors on the variances besides $\mathrm{Ga}(\tau_j^2|1, \gamma^2/2)$. We will consider various extensions below.
- It makes it clear how we can compute the full posterior, $p(\mathbf{w}|\mathcal{D})$, rather than just a MAP estimate. This technique is known as the **Bayesian lasso** (Park and Casella 2008; Hans 2009).

13.4.4.2 The objective function

From Equation 13.88, the complete data penalized log likelihood is as follows (dropping terms that do not depend on $\mathbf{w}$)

$$\ell_c(\mathbf{w}) = -\frac{1}{2\sigma^2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 - \frac{1}{2}\mathbf{w}^T\mathbf{\Lambda}\mathbf{w} + \text{const} \tag{13.89}$$

where $\mathbf{\Lambda} = \text{diag}(\frac{1}{\tau_j^2})$ is the precision matrix for $\mathbf{w}$.

13.4.4.3 The E step

The key is to compute $\mathbb{E}\left[\frac{1}{\tau_j^2}|w_j\right]$. We can derive this directly (see Exercise 13.8). Alternatively, we can derive the full posterior, which is given by the following (Park and Casella 2008):

$$p(1/\tau_j^2|\mathbf{w}, \mathcal{D}) = \text{InverseGaussian}\left(\sqrt{\frac{\gamma^2}{w_j^2}}, \gamma^2\right) \tag{13.90}$$

(Note that the **inverse Gaussian** distribution is also known as the Wald distribution.) Hence

$$\mathbb{E}\left[\frac{1}{\tau_j^2}|w_j\right] = \frac{\gamma}{|w_j|} \tag{13.91}$$

Let $\overline{\mathbf{\Lambda}} = \text{diag}(\mathbb{E}\left[1/\tau_1^2\right], \ldots, \mathbb{E}\left[1/\tau_D^2\right])$ denote the result of this E step.

We also need to infer σ^2. It is easy to show that that the posterior is

$$p(\sigma^2|\mathcal{D}, \mathbf{w}) = \text{IG}(a_\sigma + (N)/2, b_\sigma + \frac{1}{2}(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})) = \text{IG}(a_N, b_N) \tag{13.92}$$

Hence

$$\mathbb{E}\left[1/\sigma^2\right] = \frac{a_N}{b_N} \triangleq \overline{\omega} \tag{13.93}$$

13.4.4.4 The M step

The M step consists of computing

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\text{argmax}} -\frac{1}{2}\overline{\omega}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 - \frac{1}{2}\mathbf{w}^T\mathbf{\Lambda}\mathbf{w} \tag{13.94}$$

This is just MAP estimation under a Gaussian prior:

$$\hat{\mathbf{w}} = (\sigma^2\overline{\mathbf{\Lambda}} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{13.95}$$

However, since we expect many $w_j = 0$, we will have $\tau_j^2 = 0$ for many j, making inverting $\overline{\mathbf{\Lambda}}$ numerically unstable. Fortunately, we can use the SVD of $\mathbf{X}$, given by $\mathbf{X} = \mathbf{UDV}^T$, as follows:

$$\hat{\mathbf{w}} \quad = \quad \mathbf{\Psi V}(\mathbf{V}^T \mathbf{\Psi V} + \frac{1}{\overline{\omega}}\mathbf{D}^{-2})^{-1}\mathbf{D}^{-1}\mathbf{U}^T\mathbf{y} \tag{13.96}$$

where

$$\mathbf{\Psi} = \overline{\mathbf{\Lambda}}^{-1} = \text{diag}(\frac{1}{\mathbb{E}\left[1/\tau_j^2\right]}) = \text{diag}(\frac{|w_j|}{\pi'(w_j)}) \tag{13.97}$$

13.4.4.5 Caveat

Since the lasso objective is convex, this method should always find the global optimum. Unfortunately, this sometimes does not happen, for numerical reasons. In particular, suppose that in the true solution, $w_j^* \neq 0$. Further, suppose that we set $\hat{w}_j = 0$ in an M step. In the following E step we infer that $\tau_j^2 = 0$, so then we set $\hat{w}_j = 0$ again; thus we can never "undo" our mistake. Fortunately, in practice, this situation seems to be rare. See (Hunter and Li 2005) for further discussion.

13.5 ℓ_1 regularization: extensions

In this section, we discuss various extensions of "vanilla" ℓ_1 regularization.

13.5.1 Group Lasso

In standard ℓ_1 regularization, we assume that there is a 1:1 correspondence between parameters and variables, so that if $\hat{w}_j = 0$, we interpret this to mean that variable j is excluded. But in more complex models, there may be many parameters associated with a given variable. In particular, we may have a vector of weights for each input, $\mathbf{w}_j$. Here are some examples:

- **Multinomial logistic regression** Each feature is associated with C different weights, one per class.
- **Linear regression with categorical inputs** Each scalar input is one-hot encoded into a vector of length C.
- **Multi-task learning** In multi-task learning, we have multiple related prediction problems. For example, we might have C separate regression or binary classification problems. Thus each feature is associated with C different weights. We may want to use a feature for all of the tasks or none of the tasks, and thus select weights at the group level (Obozinski et al. 2007).

If we use an ℓ_1 regularizer of the form $||\mathbf{w}|| = \sum_j \sum_c |w_{jc}|$, we may end up with with some elements of $\mathbf{w}_{j,:}$ being zero and some not. To prevent this kind of situation, we partition the parameter vector into G groups. We now minimize the following objective

$$J(\mathbf{w}) = \text{NLL}(\mathbf{w}) + \sum_{g=1}^{G} \lambda_g ||\mathbf{w}_g||_2 \tag{13.98}$$

where

$$||\mathbf{w}_g||_2 = \sqrt{\sum_{j \in g} w_j^2} \tag{13.99}$$

is the 2-norm of the group weight vector. If the NLL is least squares, this method is called **group lasso** (Yuan and Lin 2006).

We often use a larger penalty for larger groups, by setting $\lambda_g = \lambda\sqrt{d_g}$, where d_g is the number of elements in group g. For example, if we have groups $\{1, 2\}$ and $\{3, 4, 5\}$, the objective becomes

$$J(\mathbf{w}) = \text{NLL}(\mathbf{w}) + \lambda \left[\sqrt{2}\sqrt{(w_1^2 + w_2^2|)} + \sqrt{3}\sqrt{(w_3^2 + w_4^2 + w_5^2)} \right] \tag{13.100}$$

Note that if we had used the square of the 2-norms, the model would become equivalent to ridge regression, since

$$\sum_{g=1}^{G} ||\mathbf{w}_g||_2^2 = \sum_g \sum_{j \in g} w_j^2 = ||\mathbf{w}||_2^2 \tag{13.101}$$

By using the square root, we are penalizing the radius of a ball containing the group's weight vector: the only way for the radius to be small is if all elements are small. Thus the square root results in group sparsity.

A variant of this technique replaces the 2-norm with the infinity-norm (Turlach et al. 2005; Zhao et al. 2005):

$$||\mathbf{w}_g||_\infty = \max_{j \in g} |w_j| \tag{13.102}$$

It is clear that this will also result in group sparsity.

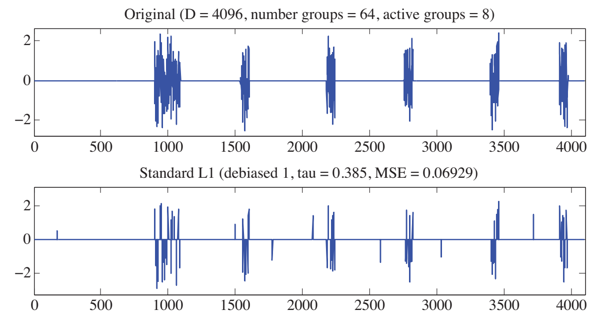
An illustration of the difference is shown in Figures 13.13 and 13.14. In both cases, we have a true signal $\mathbf{w}$ of size $D = 2^{12} = 4096$, divided into 64 groups each of size 64. We randomly choose 8 groups of $\mathbf{w}$ and assign them non-zero values. In the first example, the values are drawn from a $\mathcal{N}(0, 1)$. In the second example, the values are all set to 1. We then pick a random design matrix $\mathbf{X}$ of size $N \times D$, where $N = 2^{10} = 1024$. Finally, we generate $\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, 10^{-4}\mathbf{I}_N)$. Given this data, we estimate the support of $\mathbf{w}$ using ℓ_1 or group ℓ_1, and then estimate the non-zero values using least squares. We see that group lasso does a much better job than vanilla lasso, since it respects the known group structure.[6] We also see that the ℓ_∞ norm has a tendency to make all the elements within a block to have similar magnitude. This is appropriate in the second example, but not the first. (The value of λ was the same in all examples, and was chosen by hand.)

13.5.1.1 GSM interpretation of group lasso

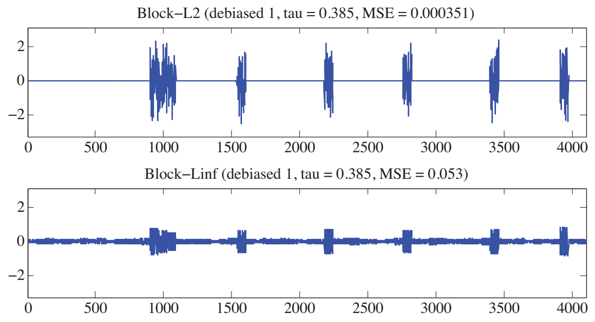
Group lasso is equivalent to MAP estimation using the following prior

$$p(\mathbf{w}|\gamma, \sigma^2) \propto \exp\left(-\frac{\gamma}{\sigma} \sum_{g=1}^{G} ||\mathbf{w}_g||_2 \right) \tag{13.103}$$

6. The slight non-zero "noise" in the ℓ_∞ group lasso results is presumably due to numerical errors.



(a)



(b)

Figure 13.13 Illustration of group lasso where the original signal is piecewise Gaussian. Top left: original signal. Bottom left:: vanilla lasso estimate. Top right: group lasso estimate using a ℓ_2 norm on the blocks. Bottom right: group lasso estimate using an ℓ_∞ norm on the blocks. Based on Figures 3-4 of (Wright et al. 2009). Figure generated by `groupLassoDemo`, based on code by Mario Figueiredo.

Now one can show (Exercise 13.10) that this prior can be written as a GSM, as follows:

$$\mathbf{w}_g|\sigma^2, \tau_g^2 \sim \mathcal{N}(\mathbf{0}, \sigma^2\tau_g^2\mathbf{I}_{d_g}) \tag{13.104}$$

$$\tau_g^2|\gamma \sim \text{Ga}(\frac{d_g+1}{2}, \frac{\gamma}{2}) \tag{13.105}$$

where d_g is the size of group g. So we see that there is one variance term per group, each of which comes from a Gamma prior, whose shape parameter depends on the group size, and whose rate parameter is controlled by γ. Figure 13.15 gives an example, where we have 2 groups, one of size 2 and one of size 3.

This picture also makes it clearer why there should be a grouping effect. Suppose $w_{1,1}$ is small; then τ_1^2 will be estimated to be small, which will force $w_{1,2}$ to be small. Converseley, suppose $w_{1,1}$ is large; then τ_1^2 will be estimated to be large, which will allow $w_{1,2}$ to be become large as well.

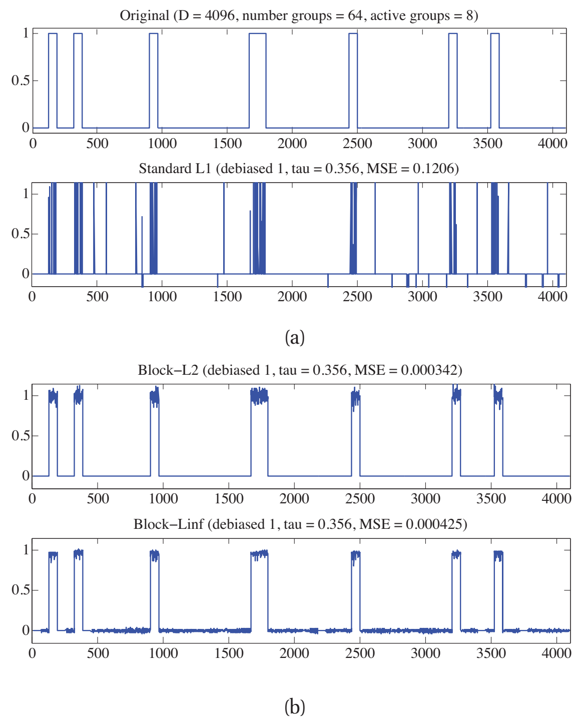


Figure 13.14 Same as Figure 13.13, except the original signal is piecewise constant.

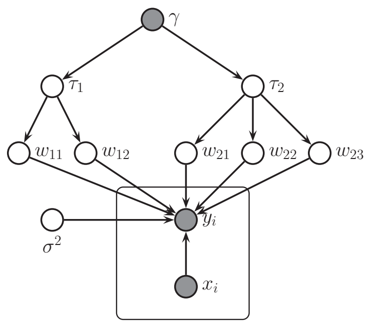


Figure 13.15 Graphical model for group lasso with 2 groups, the first has size $G_1 = 2$, the second has size $G_2 = 3$.

13.5.1.2 Algorithms for group lasso

There are a variety of algorithms for group lasso. Here we briefly mention two. The first approach is based on proximal gradient descent, discussed in Section 13.4.3. Since the regularizer is separable, $R(\mathbf{w}) = \sum_g ||\mathbf{w}_g||_p$, the proximal operator decomposes into G separate operators of the form

$$\text{prox}_R(\mathbf{b}) = \underset{\mathbf{z} \in \mathbb{R}^{D_g}}{\text{argmin}} ||\mathbf{z} - \mathbf{b}||_2^2 + \lambda ||\mathbf{z}||_p \tag{13.106}$$

where $\mathbf{b} = \boldsymbol{\theta}_{kg} - t_k \mathbf{g}_{kg}$. If $p = 2$, one can show (Combettes and Wajs 2005) that this can be implemented as follows

$$\text{prox}_R(\mathbf{b}) = \mathbf{b} - \text{proj}_{\lambda C}(\mathbf{b}) \tag{13.107}$$

where $C = \{\mathbf{z} : ||\mathbf{z}||_2 \leq 1\}$ is the ℓ_2 ball. Using Equation 13.74, if $||\mathbf{b}||_2 < \lambda$, we have

$$\text{prox}_R(\mathbf{b}) = \mathbf{b} - \mathbf{b} = \mathbf{0} \tag{13.108}$$

otherwise we have

$$\text{prox}_R(\mathbf{b}) = \mathbf{b} - \lambda \frac{\mathbf{b}}{||\mathbf{b}||_2} = \mathbf{b} \frac{||\mathbf{b}||_2 - \lambda}{||\mathbf{b}||_2} \tag{13.109}$$

We can combine these into a vectorial soft-threshold function as follows (Wright et al. 2009):

$$\text{prox}_R(\mathbf{b}) = \mathbf{b} \frac{\max(||\mathbf{b}||_2 - \lambda, 0)}{\max(||\mathbf{b}||_2 - \lambda, 0) + \lambda} \tag{13.110}$$

If $p = \infty$, we use $C = \{\mathbf{z} : ||\mathbf{z}||_1 \leq 1\}$, which is the ℓ_1 ball. We can project onto this in $O(d_g)$ time using an algorithm described in (Duchi et al. 2008).

Another approach is to modify the EM algorithm. The method is almost the same as for vanilla lasso. If we define $\tau_j^2 = \tau_{g(j)}^2$, where $g(j)$ is the group to which dimension j belongs, we can use the same full conditionals for σ^2 and $\mathbf{w}$ as before. The only changes are as follows:

- We must modify the full conditional for the weight precisions, which are estimated based on a shared set of weights:

$$\frac{1}{\tau_g^2} | \gamma, \mathbf{w}, \sigma^2, \mathbf{y}, \mathbf{X} \sim \text{InverseGaussian}(\sqrt{\frac{\gamma^2 \sigma^2}{||\mathbf{w}_g||_2^2}}, \gamma^2) \tag{13.111}$$

where $||\mathbf{w}_g||_2^2 = \sum_{j \in g} w_{jg}^2$. For the E step, we can use

$$\mathbb{E}\left[\frac{1}{\tau_g^2}\right] = \frac{\gamma \sigma}{||\mathbf{w}_g||_2} \tag{13.112}$$

- We must modify the full conditional for the tuning parameter, which is now only estimated based on G values of τ_g^2:

$$p(\gamma^2 | \boldsymbol{\tau}) = \text{Ga}(a_\gamma + G/2, b_\gamma + \frac{1}{2} \sum_g^G \tau_g^2) \tag{13.113}$$

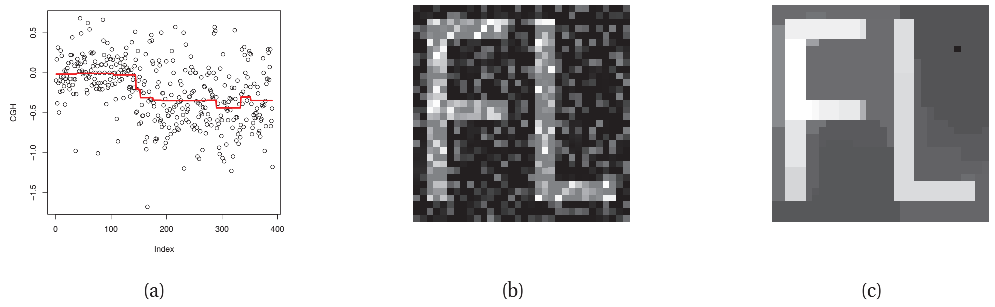


Figure 13.16 (a) Example of the fused lasso. The vertical axis represents array CGH (chromosomal genome hybridization) intensity, and the horizontal axis represents location along a genome. Source: Figure 1 of (Hoefling 2010). (b) Noisy image. (c) Fused lasso estimate using 2d lattice prior. Source: Figure 2 of (Hoefling 2010). Used with kind permission of Holger Hoefling.

13.5.2 Fused lasso

In some problem settings (e.g., functional data analysis), we want neighboring coefficients to be similar to each other, in addition to being sparse. An example is given in Figure 13.16(a), where we want to fit a signal that is mostly "off", but in addition has the property that neighboring locations are typically similar in value. We can model this by using a prior of the form

$$p(\mathbf{w}|\sigma^2) \propto \exp\left(-\frac{\lambda_1}{\sigma}\sum_{j=1}^{D}|w_j| - \frac{\lambda_2}{\sigma}\sum_{j=1}^{D-1}|w_{j+1}-w_j|\right) \tag{13.114}$$

This is known as the **fused lasso** penalty. In the context of functional data analysis, we often use $\mathbf{X} = \mathbf{I}$, so there is one coefficient for each location in the signal (see Section 4.4.2.3). In this case, the overall objective has the form

$$J(\mathbf{w}, \lambda_1, \lambda_2) = \sum_{i=1}^{N}(y_i - w_i)^2 + \lambda_1 \sum_{i=1}^{N}|w_i| + \lambda_2 \sum_{i=1}^{N-1}|w_{i+1} - w_i| \tag{13.115}$$

This is a sparse version of Equation 4.148.

It is possible to generalize this idea beyond chains, and to consider other graph structures, using a penalty of the form

$$J(\mathbf{w}, \lambda_1, \lambda_2) = \sum_{s\in V}(y_s - w_s)^2 + \lambda_1 \sum_{s\in V}|w_s| + \lambda_2 \sum_{(s,t)\in E}|w_s - w_t| \tag{13.116}$$

This is called **graph-guided fused lasso** (see e.g., (Chen et al. 2010)). The graph might come from some prior knowledge, e.g., from a database of known biological pathways. Another example is shown in Figure 13.16(b-c), where the graph structure is a 2d lattice.

13.5.2.1 GSM interpretation of fused lasso

One can show (Kyung et al. 2010) that the fused lasso model is equivalent to the following hierarchical model

$$\mathbf{w}|\sigma^2, \boldsymbol{\tau}, \boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{\Sigma}(\boldsymbol{\tau}, \boldsymbol{\omega})) \tag{13.117}$$

$$\tau_j^2|\gamma_1 \sim \text{Expon}(\frac{\gamma_1^2}{2}), \quad j = 1 : D \tag{13.118}$$

$$\omega_j^2|\gamma_2 \sim \text{Expon}(\frac{\gamma_2^2}{2}), \quad j = 1 : D-1 \tag{13.119}$$

where $\boldsymbol{\Sigma} = \boldsymbol{\Omega}^{-1}$, and $\boldsymbol{\Omega}$ is a tridiagonal precision matrix with

$$\text{main diagonal} = \{\frac{1}{\tau_j^2} + \frac{1}{\omega_{j-1}^2} + \frac{1}{\omega_j^2}\} \tag{13.120}$$

$$\text{off diagonal} = \{-\frac{1}{\omega_j^2}\} \tag{13.121}$$

where we have defined $\omega_0^{-2} = \omega_D^{-2} = 0$. This is very similar to the model in Section 4.4.2.3, where we used a chain-structured Gaussian Markov random field as the prior, with fixed variance. Here we just let the variance be random. In the case of graph-guided lasso, the structure of the graph is reflected in the zero pattern of the Gaussian precision matrix (see Section 19.4.4).

13.5.2.2 Algorithms for fused lasso

It is possible to generalize the EM algorithm to fit the fused lasso model, by exploiting the Markov structure of the Gaussian prior for efficiency. Direct solvers (which don't use the latent variable trick) can also be derived (see e.g., (Hoefling 2010)). However, this model is undeniably more expensive to fit than the other variants we have considered.

13.5.3 Elastic net (ridge and lasso combined)

Although lasso has proved to be effective as a variable selection technique, it has several problems (Zou and Hastie 2005), such as the following:

- If there is a group of variables that are highly correlated (e.g., genes that are in the same pathway), then the lasso tends to select only one of them, chosen rather arbitrarily. (This is evident from the LARS algorithm: once one member of the group has been chosen, the remaining members of the group will not be very correlated with the new residual and hence will not be chosen.) It is usually better to select all the relevant variables in a group. If we know the grouping structure, we can use group lasso, but often we don't know the grouping structure.
- In the $D > N$ case, lasso can select at most N variables before it saturates.
- If $N > D$, but the variables are correlated, it has been empirically observed that the prediction performance of ridge is better than that of lasso.

Zou and Hastie (Zou and Hastie 2005) proposed an approach called the **elastic net**, which is a hybrid between lasso and ridge regression, which solves all of these problems. It is apparently called the "elastic net" because it is "like a stretchable fishing net that retains 'all the big fish"' (Zou and Hastie 2005).

13.5.3.1 Vanilla version

The vanilla version of the model defines the following objective function:

$$J(\mathbf{w}, \lambda_1, \lambda_2) = ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 + \lambda_2 ||\mathbf{w}||_2^2 + \lambda_1 ||\mathbf{w}||_1 \tag{13.122}$$

Notice that this penalty function is *strictly convex* (assuming $\lambda_2 > 0$) so there is a unique global minimum, even if $\mathbf{X}$ is not full rank.

It can be shown (Zou and Hastie 2005) that any strictly convex penalty on $\mathbf{w}$ will exhibit a **grouping effect**, which means that the regression coefficients of highly correlated variables tend to be equal (up to a change of sign if they are negatively correlated). For example, if two features are equal, so $\mathbf{X}_{:j} = \mathbf{X}_{:k}$, one can show that their estimates are also equal, $\hat{w}_j = \hat{w}_k$. By contrast, with lasso, we may have that $\hat{w}_j = 0$ and $\hat{w}_k \neq 0$ or vice versa.

13.5.3.2 Algorithms for vanilla elastic net

It is simple to show (Exercise 13.5) that the elastic net problem can be reduced to a lasso problem on modified data. In particular, define

$$\tilde{\mathbf{X}} = c \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda_2}\mathbf{I}_D \end{pmatrix}, \quad \tilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y} \\ \mathbf{0}_{D \times 1} \end{pmatrix} \tag{13.123}$$

where $c = (1 + \lambda_2)^{-\frac{1}{2}}$. Then we solve

$$\tilde{\mathbf{w}} = \arg\min_{\tilde{\mathbf{w}}} ||\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\tilde{\mathbf{w}}||^2 + c\lambda_1 ||\tilde{\mathbf{w}}||_1 \tag{13.124}$$

and set $\mathbf{w} = c\tilde{\mathbf{w}}$.

We can use LARS to solve this subproblem; this is known as the LARS-EN algorithm. If we stop the algorithm after m variables have been included, the cost is $O(m^3 + Dm^2)$. Note that we can use $m = D$ if we wish, since $\tilde{\mathbf{X}}$ has rank D. This is in contrast to lasso, which cannot select more than N variables (before jumping to the OLS solution) if $N < D$.

When using LARS-EN (or other ℓ_1 solvers), one typically uses cross-validation to select λ_1 and λ_2.

13.5.3.3 Improved version

Unfortunately it turns out that the "vanilla" elastic net does not produce functions that predict very accurately, unless it is very close to either pure ridge or pure lasso. Intuitively the reason is that it performs shrinkage twice: once due to the ℓ_2 penalty and again due to the ℓ_1 penalty. The solution is simple: undo the ℓ_2 shrinkage by scaling up the estimates from the vanilla version. In other words, if $\mathbf{w}^*$ is the solution of Equation 13.124, then a better estimate is

$$\hat{\mathbf{w}} = \sqrt{1 + \lambda_2}\tilde{\mathbf{w}} \tag{13.125}$$

We will call this a corrected estimate.

One can show that the corrected estimates are given by

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \mathbf{w}^T \left(\frac{\mathbf{X}^T\mathbf{X} + \lambda_2 \mathbf{I}}{1 + \lambda_2} \right) \mathbf{w} - 2\mathbf{y}^T \mathbf{X}\mathbf{w} + \lambda_1 ||\mathbf{w}||_1 \tag{13.126}$$

Now

$$\frac{\mathbf{X}^T\mathbf{X} + \lambda_2 \mathbf{I}}{1 + \lambda_2} = (1 - \rho)\hat{\mathbf{\Sigma}} + \rho \mathbf{I} \tag{13.127}$$

where $\rho = \lambda_2/(1 + \lambda_2)$. So the the elastic net is like lasso but where we use a version of $\hat{\mathbf{\Sigma}}$ that is shrunk towards $\mathbf{I}$. (See Section 4.2.6 for more discussion of regularized estimates of covariance matrices.)

13.5.3.4 GSM interpretation of elastic net

The implicit prior being used by the elastic net obviously has the form

$$p(\mathbf{w}|\sigma^2) \propto \exp\left(-\frac{\gamma_1}{\sigma} \sum_{j=1}^{D} |w_j| - \frac{\gamma_2}{2\sigma^2} \sum_{j=1}^{D} w_j^2 \right) \tag{13.128}$$

which is just a product of Gaussian and Laplace distributions.

This can be written as a hierarchical prior as follows (Kyung et al. 2010; Chen et al. 2011):

$$w_j|\sigma^2, \tau_j^2 \sim \mathcal{N}(0, \sigma^2(\tau_j^{-2} + \gamma_2)^{-1}) \tag{13.129}$$

$$\tau_j^2|\gamma_1 \sim \text{Expon}(\frac{\gamma_1^2}{2}) \tag{13.130}$$

Clearly if $\gamma_2 = 0$, this reduces to the regular lasso.

It is possible to perform MAP estimation in this model using EM, or Bayesian inference using MCMC (Kyung et al. 2010) or variational Bayes (Chen et al. 2011).

13.6 Non-convex regularizers

Although the Laplace prior results in a convex optimization problem, from a statistical point of view this prior is not ideal. There are two main problems with it. First, it does not put enough probability mass near 0, so it does not sufficiently suppress noise. Second, it does not put enough probability mass on large values, so it causes shrinkage of relevant coefficients, corresponding to “signal”. (This can be seen in Figure 13.5(a): we see that ℓ_1 estimates of large coefficients are significantly smaller than their ML estimates, a phenomenon known as bias.)

Both problems can be solved by going to more flexible kinds of priors which have a larger spike at 0 and heavier tails. Even though we cannot find the global optimum anymore, these non-convex methods often outperform ℓ_1 regularization, both in terms of predictive accuracy and in detecting relevant variables (Fan and Li 2001; Schniter et al. 2008). We give some examples below.

13.6.1 Bridge regression

A natural generalization of ℓ_1 regularization, known as **bridge regression** (Frank and Friedman 1993), has the form

$$\hat{\mathbf{w}} = \text{NLL}(\mathbf{w}) + \lambda \sum_j |w_j|^b \tag{13.131}$$

for $b \geq 0$. This corresponds to MAP estimation using a **exponential power distribution** given by

$$\text{ExpPower}(w|\mu, a, b) \triangleq \frac{b}{2a\Gamma(1+1/b)} \exp\left(-\frac{|x-\mu|}{a}^b\right) \tag{13.132}$$

If $b = 2$, we get the Gaussian distribution (with $a = \sigma\sqrt{2}$), corresonding to ridge regression; if we set $b = 1$, we get the Laplace distribution, corresponding to lasso; if we set $b = 0$, we get ℓ_0 regression, which is equivalent to best subset selection. Unfortunately, the objective is not convex for $b < 1$, and is not sparsity promoting for $b > 1$. So the ℓ_1 norm is the tightest convex approximation to the ℓ_0 norm.

The effect of changing b is illustrated in Figure 13.17, where we plot the prior for $b = 2$, $b = 1$ and $b = 0.4$; we assume $p(\mathbf{w}) = p(w_1)p(w_2)$. We also plot the posterior after seeing a single observation, $(\mathbf{x}, y)$, which imposes a single linear constraint of the form, $y = \mathbf{w}^T\mathbf{x}$, with a certain tolerance controlled by the observation noise (compare to Figure 7.11). We see see that the mode of the Laplace is on the vertical axis, corresponding to $w_1 = 0$. By contrast, there are two modes when using $b = 0.4$, corresponding to two different sparse solutions. When using the Gaussian, the MAP estimate is not sparse (the mode does not lie on either of the coordinate axes).

13.6.2 Hierarchical adaptive lasso

Recall that one of the principal problems with lasso is that it results in biased estimates. This is because it needs to use a large value of λ to "squash" the irrelevant parameters, but this then over-penalizes the relevant parameters. It would be better if we could associate a different penalty parameter with each parameter. Of course, it is completely infeasible to tune D parameters by cross validation, but this poses no problem to the Bayesian: we simply make each τ_j^2 have its own private tuning parameter, γ_j, which are now treated as random variables coming from the conjugate prior $\gamma_j \sim \text{IG}(a, b)$. The full model is as follows:

$$\gamma_j \sim \text{IG}(a, b) \tag{13.133}$$

$$\tau_j^2|\gamma_j \sim \text{Ga}(1, \gamma_j^2/2) \tag{13.134}$$

$$w_j|\tau_j^2 \sim \mathcal{N}(0, \tau_j^2) \tag{13.135}$$

See Figure 13.18(a). This has been called the **hierarchical adaptive lasso** (HAL) (Lee et al. 2010) (see also (Lee et al. 2011; Cevher 2009; Armagan et al. 2011)). We can integrate out τ_j^2, which induces a $\text{Lap}(w_j|0, 1/\gamma_j)$ distribution on w_j as before. The result is that $p(w_j)$ is now a scaled mixture of Laplacians. It turns out that we can fit this model (i.e., compute a *local* posterior mode) using EM, as we explain below. The resulting estimate, $\hat{\mathbf{w}}_{HAL}$, often works

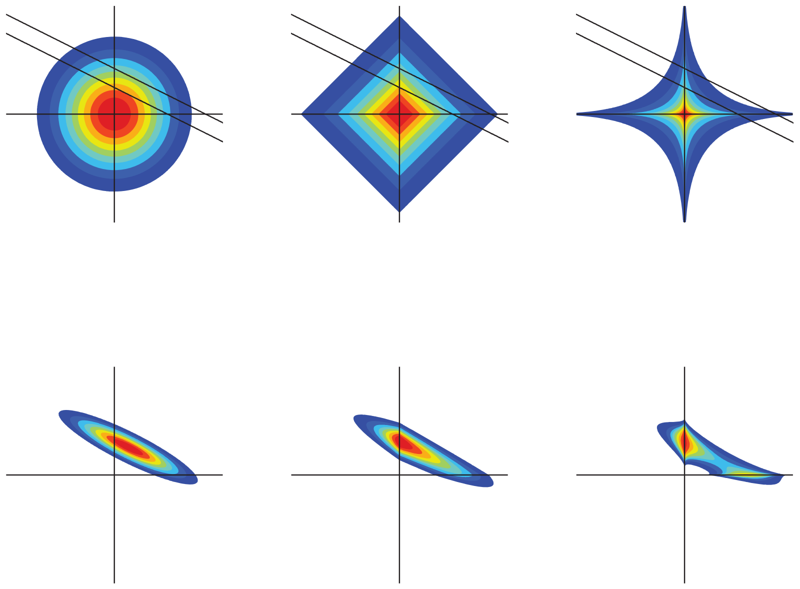


Figure 13.17 Top: plot of log *prior* for three different distributions with unit variance: Gaussian, Laplace and exponential power. Bottom: plot of log *posterior* after observing a single observation, corresponding to a single linear constraint. The precision of this observation is shown by the diagonal lines in the top figure. In the case of the Gaussian prior, the posterior is unimodal and symmetric. In the case of the Laplace prior, the posterior is unimodal and asymmetric (skewed). In the case of the exponential prior, the posterior is bimodal. Based on Figure 1 of (Seeger 2008). Figure generated by `sparsePostPlot`, written by Florian Steinke.

much better than the estimate returned by lasso, $\hat{\mathbf{w}}_{L1}$, in the sense that it is more likely to contain zeros in the right places (model selection consistency) and more likely to result in good predictions (prediction consistency) (Lee et al. 2010). We give an explanation for this behavior in Section 13.6.2.2.

13.6.2.1 EM for HAL

Since the inverse Gamma is conjugate to the Laplace, we find that the E step for γ_j is given by

$$p(\gamma_j|w_j) = \text{IG}(a+1, b+|w_j|) \tag{13.136}$$

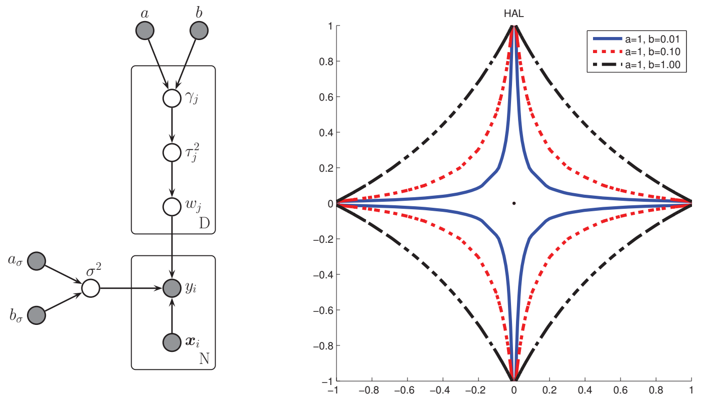
The E step for σ^2 is the same as for vanilla lasso.

The prior for $\mathbf{w}$ has the following form:

$$p(\mathbf{w}|\boldsymbol{\gamma}) = \prod_j \frac{1}{2\gamma_j} \exp(-|w_j|/\gamma_j) \tag{13.137}$$

Hence the M step must optimize

$$\hat{\mathbf{w}}^{(t+1)} = \underset{\mathbf{w}}{\text{argmax}} \log \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \sigma^2) - \sum_j |w_j| \mathbb{E}\left[1/\gamma_j\right] \tag{13.138}$$



(a) (b)

Figure 13.18 (a) DGM for hierarchical adaptive lasso. (b) Contours of Hierarchical adpative Laplace. Based on Figure 1 of (Lee et al. 2010). Figure generated by `normalGammaPenaltyPlotDemo`.

The expectation is given by

$$\mathbb{E}[1/\gamma_j] = \frac{a+1}{b+|w_j^{(t)}|} \triangleq s_j^{(t)} \tag{13.139}$$

Thus the M step becomes a weighted lasso problem:

$$\hat{\mathbf{w}}^{(t+1)} = \underset{\mathbf{w}}{\operatorname{argmin}} ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \sum_j s_j^{(t)}|w_j| \tag{13.140}$$

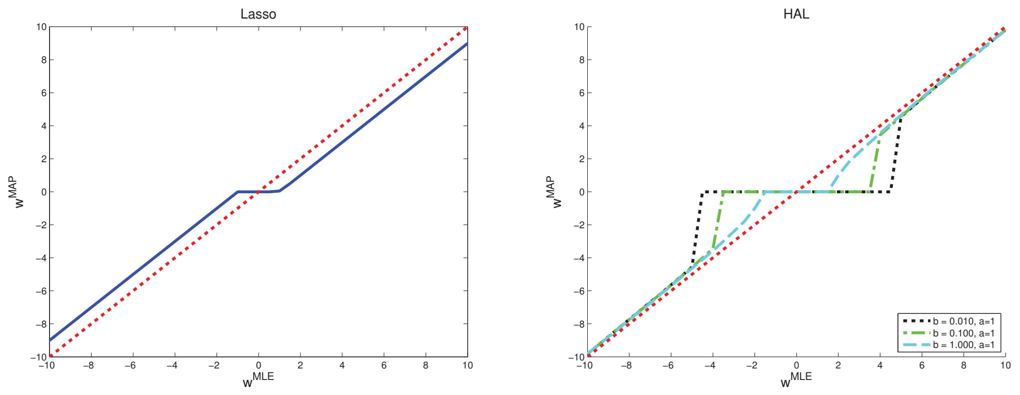
This is easily solved using standard methods (e.g., LARS). Note that if the coefficient was estimated to be large in the previous iteration (so $w_j^{(t)}$ is large), then the scaling factor $s_j^{(t)}$ will be small, so large coefficients are not penalized heavily. Conversely, small coefficients *do* get penalized heavily. This is the way that the algorithm adapts the penalization strength of each coefficient. The result is an estimate that is often much sparser than returned by lasso, but also less biased.

Note that if we set $a = b = 0$, and we only perform 1 iteration of EM, we get a method that is closely related to the **adaptive lasso** of (Zou 2006; Zou and Li 2008). This EM algorithm is also closely related to some iteratively reweighted ℓ_1 methods proposed in the signal processing community (Chartrand and Yin 2008; Candes et al. 2008).

13.6.2.2 Understanding the behavior of HAL

We can get a better understanding of HAL by integrating out γ_j to get the following marginal distribution,

$$p(w_j|a,b) = \frac{a}{2b}\left(\frac{|w_j|}{b} + 1\right)^{-(a+1)} \tag{13.141}$$



(a) (b)

Figure 13.19 Thresholding behavior of two penalty functions (negative log priors). (a) Laplace. (b) Hierarchical adaptive Laplace. Based on Figure 2 of (Lee et al. 2010). Figure generated by `normalGammaThresholdPlotDemo`.

This is an instance of the **generalized t distribution** (McDonald and Newey 1988) (in (Cevher 2009; Armagan et al. 2011), this is called the double Pareto distribution) defined as

$$\text{GT}(w|\mu, a, c, q) \triangleq \frac{q}{2ca^{1/q}B(1/q, a)} \left(1 + \frac{|w-\mu|^q}{ac^q}\right)^{-(a+1/q)} \tag{13.142}$$

where c is the scale parameter (which controls the degree of sparsity), and a is related to the degrees of freedom. When $q = 2$ and $c = \sqrt{2}$ we recover the standard t distribution; when $a \to \infty$, we recover the exponential power distribution; and when $q = 1$ and $a = \infty$ we get the Laplace distribution. In the context of the current model, we see that $p(w_j|a, b) = \text{GT}(w_j|0, a, b/a, 1)$.

The resulting penalty term has the form

$$\pi_{\boldsymbol{\lambda}}(w_j) \triangleq -\log p(w_j) = (a+1)\log(1 + \frac{|w_j|}{b}) + \text{const} \tag{13.143}$$

where $\boldsymbol{\lambda} = (a, b)$ are the tuning parameters. We plot this penalty in 2d (i.e., we plot $\pi_{\boldsymbol{\lambda}}(w_1) + \pi_{\boldsymbol{\lambda}}(w_2)$) in Figure 13.18(b) for various values of b. Compared to the diamond-shaped Laplace penalty, shown in Figure 13.3(a), we see that the HAL penalty looks more like a "star fish": it puts much more density along the "spines", thus enforcing sparsity more aggressively. Note that this penalty is clearly not convex.

We can gain further understanding into the behavior of this penalty function by considering applying it to the problem of linear regression with an orthogonal design matrix. In this case,

$p(\tau_j^2)$	$p(\gamma_j)$	$p(w_j)$	Ref
$\text{Ga}(1, \frac{\gamma^2}{2})$	Fixed	$\text{Lap}(0, 1/\gamma)$	(Andrews and Mallows 1974; West 1987)
$\text{Ga}(1, \frac{\gamma^2}{2})$	$\text{IG}(a, b)$	$\text{GT}(0, a, b/a, 1)$	(Lee et al. 2010, 2011; Cevher 2009; Armagan et al. 2011)
$\text{Ga}(1, \frac{\gamma^2}{2})$	$\text{Ga}(a, b)$	$\text{NEG}(a, b)$	(Griffin and Brown 2007, 2010; Chen et al. 2011)
$\text{Ga}(\delta, \frac{\gamma^2}{2})$	Fixed	$\text{NG}(\delta, \gamma)$	(Griffin and Brown 2007, 2010)
$\text{Ga}(\tau_j^2 \vert 0, 0)$	-	$\text{NJ}(w_j)$	(Figueiredo 2003)
$\text{IG}(\frac{\delta}{2}, \frac{\delta\gamma^2}{2})$	Fixed	$\mathcal{T}(0, \delta, \gamma)$	(Andrews and Mallows 1974; West 1987)
$C^+(0, \gamma)$	$C^+(0, b)$	horseshoe(b)	(Carvahlo et al. 2010)

Table 13.2 Some scale mixtures of Gaussians. Abbreviations: C^+ = half-rectified Cauchy; Ga = Gamma (shape and rate parameterization); GT = generalized t; IG = inverse Gamma; NEG = Normal-Exponential-Gamma; NG = Normal-Gamma; NJ = Normal-Jeffreys. The horseshoe distribution is the name we give to the distribution induced on w_j by the prior described in (Carvahlo et al. 2010); this has no simple analytic form. The definitions of the NEG and NG densities are a bit complicated, but can be found in the references. The other distributions are defined in the text.

one can show that the objective becomes

$$J(\mathbf{w}) = \frac{1}{2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \sum_{j=1}^{D} \pi_{\boldsymbol{\lambda}}(|w_j|) \tag{13.144}$$

$$= \frac{1}{2}||\mathbf{y} - \hat{\mathbf{y}}||^2 + \frac{1}{2}\sum_{j=1}^{D}(\hat{w}_j^{mle} - w_j)^2 + \sum_{j=1}^{D} \pi_{\boldsymbol{\lambda}}(|w_j|) \tag{13.145}$$

where $\hat{\mathbf{w}}^{mle} = \mathbf{X}^T\mathbf{y}$ is the MLE and $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}^{mle}$. Thus we can compute the MAP estimate one dimension at a time by solving the following 1d optimization problem:

$$\hat{w}_j = \underset{w_j}{\text{argmin}} \frac{1}{2}(\hat{w}_j^{mle} - w_j)^2 + \pi_{\boldsymbol{\lambda}}(w_j) \tag{13.146}$$

In Figure 13.19(a) we plot the lasso estimate, $\hat{w}^{L1}$, vs the ML estimate, $\hat{w}^{mle}$. We see that the ℓ_1 estimator has the usual soft-thresholding behavior seen earlier in Figure 13.5(a). However, this behavior is undesirable since the large magnitude coefficients are also shrunk towards 0, whereas we would like them to be equal to their unshrunken ML estimates.

In Figure 13.19(b) we plot the HAL estimate, $\hat{w}^{HAL}$, vs the ML estimate $\hat{w}^{mle}$. We see that this approximates the more desirable hard thresholding behavior seen earlier in Figure 13.5(b) much more closely.

13.6.3 Other hierarchical priors

Many other hierarchical sparsity-promoting priors have been proposed; see Table 13.2 for a brief summary. In some cases, we can analytically derive the form of the marginal prior for w_j. Generally speaking, this prior is not concave.

A particularly interesting prior is the improper Normal-Jeffreys prior, which has been used in (Figueiredo 2003). This puts a non-informative Jeffreys prior on the variance, $\text{Ga}(\tau_j^2|0, 0) \propto$

$1/\tau_j^2$; the resulting marginal has the form $p(w_j) = \text{NJ}(w_j) \propto 1/|w_j|$. This gives rise to a thresholding rule that looks very similar to HAL in Figure 13.19(b), which in turn is very similar to hard thresholding. However, this prior has no free parameters, which is both a good thing (nothing to tune) and a bad thing (no ability to adapt the level of sparsity).

13.7 Automatic relevance determination (ARD)/sparse Bayesian learning (SBL)

All the methods we have considered so far (except for the spike-and-slab methods in Section 13.2.1) have used a **factorial prior** of the form $p(\mathbf{w}) = \prod_j p(w_j)$. We have seen how these priors can be represented in terms of Gaussian scale mixtures of the form $w_j \sim \mathcal{N}(0, \tau_j^2)$, where τ_j^2 has one of the priors listed in Table 13.2. Using these latent variances, we can represent the model in the form $\tau_j^2 \to w_j \to \mathbf{y} \leftarrow \mathbf{X}$. We can then use EM to perform MAP estimation, where in the E step we infer $p(\tau_j^2|w_j)$, and in the M step we estimate $\mathbf{w}$ from $\mathbf{y}$, $\mathbf{X}$ and $\boldsymbol{\tau}$. This M step either involves a closed-form weighted ℓ_2 optimization (in the case of Gaussian scale mixtures), or a weighted ℓ_1 optimization (in the case of Laplacian scale mixtures). We also discussed how to perform Bayesian inference in such models, rather than just computing MAP estimates.

In this section, we discuss an alternative approach based on type II ML estimation (empirical Bayes), whereby we integrate out $\mathbf{w}$ and maximize the marginal likelihood wrt $\boldsymbol{\tau}$. This EB procedure can be implemented via EM, or via a reweighted ℓ_1 scheme, as we will explain below. Having estimated the variances, we plug them in to compute the posterior mean of the weights, $\mathbb{E}[\mathbf{w}|\hat{\boldsymbol{\tau}}, \mathcal{D}]$; rather surprisingly (in view of the Gaussian prior), the result is an (approximately) sparse estimate, for reasons we explain below.

In the context of neural networks, this this method is called called **automatic relevance determination** or **ARD** (MacKay 1995b; Neal 1996): see Section 16.5.7.5. In the context of the linear models we are considering in this chapter, this method is called **sparse Bayesian learning** or **SBL** (Tipping 2001). Combining ARD/SBL with basis function expansion in a linear model gives rise to a technique called the relevance vector machine (RVM), which we will discuss in Section 14.3.2.

13.7.1 ARD for linear regression

We will explain the procedure in the context of linear regression; ARD for GLMs requires the use of the Laplace (or some other) approximation. case can be It is conventional, when discussing ARD / SBL, to denote the weight precisions by $\alpha_j = 1/\tau_j^2$, and the measurement precision by $\beta = 1/\sigma^2$ (do not confuse this with the use of $\boldsymbol{\beta}$ in statistics to represent the regression coefficients!). In particular, we will assume the following model:

$$p(y|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x}, 1/\beta) \tag{13.147}$$

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{A}^{-1}) \tag{13.148}$$

where $\mathbf{A} = \text{diag}(\boldsymbol{\alpha})$. The marginal likelihood can be computed analytically as follows:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha}, \beta) = \int \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \beta\mathbf{I}_N)\mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{A})d\mathbf{w} \tag{13.149}$$

$$= \mathcal{N}(\mathbf{y}|\mathbf{0}, \beta\mathbf{I}_N + \mathbf{X}\mathbf{A}^{-1}\mathbf{X}^T) \tag{13.150}$$

$$= (2\pi)^{-N/2}|\mathbf{C}_\alpha|^{-\frac{1}{2}} \exp(-\frac{1}{2}\mathbf{y}^T\mathbf{C}_\alpha^{-1}\mathbf{y}) \tag{13.151}$$

where

$$\mathbf{C}_{\boldsymbol{\alpha}} \triangleq \beta^{-1}\mathbf{I}_N + \mathbf{X}\mathbf{A}^{-1}\mathbf{X}^T \tag{13.152}$$

Compare this to the marginal likelihood in Equation 13.13 in the spike and slab model; modulo the $\beta = 1/\sigma^2$ factor missing from the second term, the equations are the same, except we have replaced the binary $\gamma_j \in \{0, 1\}$ with continuous $\alpha_j \in \mathbb{R}^+$. In log form, the objective becomes

$$\ell(\boldsymbol{\alpha}, \beta) \triangleq -\frac{1}{2}\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha}, \beta) = \log|\mathbf{C}_\alpha| + \mathbf{y}^T\mathbf{C}_{\boldsymbol{\alpha}}^{-1}\mathbf{y} \tag{13.153}$$

To regularize the problem, we may put a conjugate prior on each precision, $\alpha_j \sim \text{Ga}(a, b)$ and $\beta \sim \text{Ga}(c, d)$. The modified objective becomes

$$\ell(\boldsymbol{\alpha}, \beta) \triangleq -\frac{1}{2}\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha}, \beta) + \sum_j \log \text{Ga}(\alpha_j|a, b) + \log \text{Ga}(\beta|c, d) \tag{13.154}$$

$$= \log|\mathbf{C}_\alpha| + \mathbf{y}^T\mathbf{C}_{\boldsymbol{\alpha}}^{-1}\mathbf{y} + \sum_j (a \log \alpha_j - b\alpha_j) + c \log \beta - d\beta \tag{13.155}$$

This is useful when performing Bayesian inference for $\boldsymbol{\alpha}$ and β (Bishop and Tipping 2000). However, when performing (type II) point estimation, we will use the improper prior $a = b = c = d = 0$, which results in maximal sparsity.

Below we describe how to optimize $\ell(\boldsymbol{\alpha}, \beta)$ wrt the precision terms $\boldsymbol{\alpha}$ and β.[7] This is a proxy for finding the most probable model setting of $\boldsymbol{\gamma}$ in the spike and slab model, which in turn is closely related to ℓ_0 regularization. In particular, it can be shown (Wipf et al. 2010) that the objective in Equation 13.153 has many fewer local optima than the ℓ_0 objective, and hence is much easier to optimize.

Once we have estimated $\boldsymbol{\alpha}$ and β, we can compute the posterior over the parameters using

$$p(\mathbf{w}|\mathcal{D}, \hat{\boldsymbol{\alpha}}, \hat{\beta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{13.156}$$

$$\boldsymbol{\Sigma}^{-1} = \hat{\beta}\mathbf{X}^T\mathbf{X} + \mathbf{A} \tag{13.157}$$

$$\boldsymbol{\mu} = \hat{\beta}\boldsymbol{\Sigma}\mathbf{X}^T\mathbf{y} \tag{13.158}$$

The fact that we compute a posterior over $\mathbf{w}$, while simultaneously encouraging sparsity, is why the method is called "sparse Bayesian learning". Nevertheless, since there are many ways to be sparse and Bayesian, we will use the "ARD" term instead, even in the linear model context. (In addition, SBL is only "being Bayesian" about the values of the coefficients, rather than reflecting uncertainty about the set of relevant variables, which is typically of more interest.)

7. An alternative approach to optimizing β is to put a Gamma prior on β and to integrate it out to get a Student posterior for $\mathbf{w}$ (Buntine and Weigend 1991). However, it turns out that this results in a less accurate estimate for $\boldsymbol{\alpha}$ (MacKay 1999). In addition, working with Gaussians is easier than working with the Student distribution, and the Gaussian case generalizes more easily to other cases such as logistic regression.

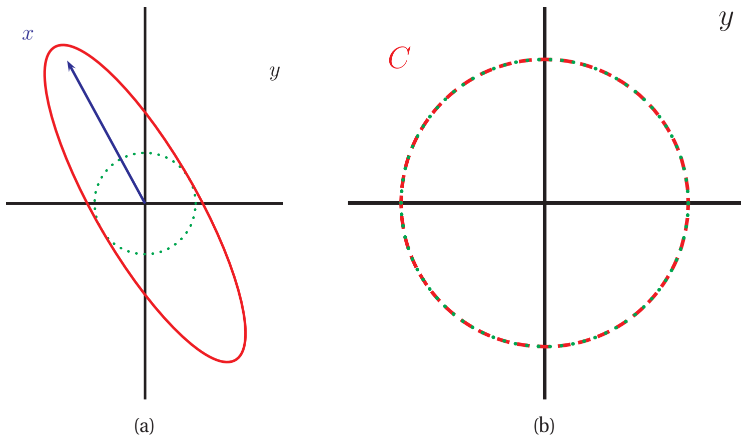


Figure 13.20 Illustration of why ARD results in sparsity. The vector of inputs $\mathbf{x}$ does not point towards the vector of outputs $\mathbf{y}$, so the feature should be removed. (a) For finite α, the probability density is spread in directions away from $\mathbf{y}$. (b) When $\alpha = \infty$, the probability density at $\mathbf{y}$ is maximized. Based on Figure 8 of (Tipping 2001).

13.7.2 Whence sparsity?

If $\hat{\alpha}_j \approx 0$, we find $\hat{w}_j \approx \hat{w}_j^{mle}$, since the Gaussian prior shrinking w_j towards 0 has zero precision. However, if we find that $\hat{\alpha}_j \approx \infty$, then the prior is very confident that $w_j = 0$, and hence that feature j is "irrelevant". Hence the posterior mean will have $\hat{w}_j \approx 0$. Thus irrelevant features automatically have their weights "turned off" or "pruned out".

We now give an intuitive argument, based on (Tipping 2001), about why ML-II should encourage $\alpha_j \to \infty$ for irrelevant features. Consider a 1d linear regression with 2 training examples, so $\mathbf{X} = \mathbf{x} = (x_1, x_2)$, and $\mathbf{y} = (y_1, y_2)$. We can plot $\mathbf{x}$ and $\mathbf{y}$ as vectors in the plane, as shown in Figure 13.20. Suppose the feature is irrelevant for predicting the response, so $\mathbf{x}$ points in a nearly orthogonal direction to $\mathbf{y}$. Let us see what happens to the marginal likelihood as we change α. The marginal likelihood is given by $p(\mathbf{y}|\mathbf{x}, \alpha, \beta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C})$, where

$$\mathbf{C} = \frac{1}{\beta}\mathbf{I} + \frac{1}{\alpha}\mathbf{x}\mathbf{x}^T \tag{13.159}$$

If α is finite, the posterior will be elongated along the direction of $\mathbf{x}$, as in Figure 13.20(a). However, if $\alpha = \infty$, we find $\mathbf{C} = \frac{1}{\beta}\mathbf{I}$, so $\mathbf{C}$ is spherical, as in Figure 13.20(b). If $|\mathbf{C}|$ is held constant, the latter assigns higher probability density to the observed response vector $\mathbf{y}$, so this is the preferred solution. In other words, the marginal likelihood "punishes" solutions where α_j is small but $\mathbf{X}_{:,j}$ is irrelevant, since these waste probability mass. It is more parsimonious (from the point of view of Bayesian Occam's razor) to eliminate redundant dimensions.

13.7.3 Connection to MAP estimation

ARD seems quite different from the MAP estimation methods we have been considering earlier in this chapter. In particular, in ARD, we are not integrating out $\boldsymbol{\alpha}$ and optimizing $\mathbf{w}$, but vice versa. Because the parameters w_j become correlated in the posterior (due to explaining away), when we estimate α_j we are borrowing information from all the features, not just feature j. Consequently, the effective prior $p(\mathbf{w}|\hat{\boldsymbol{\alpha}})$ is **non-factorial**, and furthermore it depends on the data $\mathcal{D}$ (and σ^2). However, in (Wipf and Nagarajan 2007), it was shown that ARD can be viewed as the following MAP estimation problem:

$$\hat{\mathbf{w}}^{ARD} = \arg\min_{\mathbf{w}} \beta||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + g_{ARD}(\mathbf{w}) \tag{13.160}$$

$$g_{ARD}(\mathbf{w}) \triangleq \min_{\boldsymbol{\alpha}\geq 0} \sum_j \alpha_j w_j^2 + \log|\mathbf{C}_{\boldsymbol{\alpha}}| \tag{13.161}$$

The proof, which is based on convex analysis, is a little complicated and hence is omitted.

Furthermore, (Wipf and Nagarajan 2007; Wipf et al. 2010) prove that MAP estimation with non-factorial priors is strictly better than MAP estimation with any possible factorial prior in the following sense: the non-factorial objective always has fewer local minima than factorial objectives, while still satisfying the property that the global optimum of the non-factorial objective corresponds to the global optimum of the ℓ_0 objective — a property that ℓ_1 regularization, which has no local minima, does not enjoy.

13.7.4 Algorithms for ARD *

In this section, we review several different algorithms for implementing ARD.

13.7.4.1 EM algorithm

The easiest way to implement SBL/ARD is to use EM. The expected complete data log likelihood is given by

$$\begin{aligned} Q(\boldsymbol{\alpha}, \beta) &= \mathbb{E}\left[\log \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I}) + \log \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{A}^{-1})\right] & (13.162)\\ &= \frac{1}{2}\mathbb{E}\left[N\log\beta - \beta||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 + \sum_j \log\alpha_j - \text{tr}(\mathbf{A}\mathbf{w}\mathbf{w}^T)\right] + \text{const} & (13.163)\\ &= \frac{1}{2}N\log\beta - \frac{\beta}{2}\left(||\mathbf{y} - \mathbf{X}\boldsymbol{\mu}||^2 + \text{tr}(\mathbf{X}^T\mathbf{X}\boldsymbol{\Sigma})\right) \\ &\quad + \frac{1}{2}\sum_j \log\alpha_j - \frac{1}{2}\text{tr}[\mathbf{A}(\boldsymbol{\mu}\boldsymbol{\mu}^T + \boldsymbol{\Sigma})] + \text{const} & (13.164) \end{aligned}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are computed in the E step using Equation 13.158.

Suppose we put a $\text{Ga}(a, b)$ prior on α_j and a $\text{Ga}(c, d)$ prior on β. The penalized objective becomes

$$Q'(\boldsymbol{\alpha}, \beta) = Q(\boldsymbol{\alpha}, \beta) + \sum_j (a\log\alpha_j - b\alpha_j) + c\log\beta - d\beta \tag{13.165}$$

Setting $\frac{dQ'}{d\alpha_j} = 0$ we get the following M step:

$$\alpha_j = \frac{1 + 2a}{\mathbb{E}\left[w_j^2\right] + 2b} = \frac{1 + 2a}{m_j^2 + \Sigma_{jj} + 2b} \tag{13.166}$$

If $\alpha_j = \alpha$, and $a = b = 0$, the update becomes

$$\alpha = \frac{D}{\mathbb{E}\left[\mathbf{w}^T\mathbf{w}\right]} = \frac{D}{\boldsymbol{\mu}^T\boldsymbol{\mu} + \text{tr}(\boldsymbol{\Sigma})} \tag{13.167}$$

The update for β is given by

$$\beta_{new}^{-1} = \frac{||\mathbf{y} - \mathbf{X}\boldsymbol{\mu}||^2 + \beta^{-1}\sum_j(1 - \alpha_j\Sigma_{jj}) + 2d}{N + 2c} \tag{13.168}$$

(Deriving this is Exercise 13.2.)

13.7.4.2 Fixed-point algorithm

A faster and more direct approach is to directly optimize the objective in Equation 13.155. One can show (Exercise 13.3) that the equations $\frac{d\ell}{d\alpha_j} = 0$ and $\frac{d\ell}{d\beta} = 0$ lead to the following fixed point updates:

$$\alpha_j \leftarrow \frac{\gamma_j + 2a}{m_j^2 + 2b} \tag{13.169}$$

$$\beta^{-1} \leftarrow \frac{||\mathbf{y} - \mathbf{X}\boldsymbol{\mu}||^2 + 2d}{N - \sum_j \gamma_j + 2c} \tag{13.170}$$

$$\gamma_j \triangleq 1 - \alpha_j\Sigma_{jj} \tag{13.171}$$

The quantity γ_j is a measure of how well-determined w_j is by the data (MacKay 1992). Hence $\gamma = \sum_j \gamma_j$ is the effective degrees of freedom of the model. See Section 7.5.3 for further discussion.

Since $\boldsymbol{\alpha}$ and β both depend on $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ (which can be computed using Equation 13.158 or the Laplace approximation), we need to re-estimate these equations until convergence. (Convergence properties of this algorithm have been studied in (Wipf and Nagarajan 2007).) At convergence, the results are formally identical to those obtained by EM, but since the objective is non-convex, the results can depend on the initial values.

13.7.4.3 Iteratively reweighted ℓ_1 algorithm

Another approach to solving the ARD problem is based on the view that it is a MAP estimation problem. Although the log prior $g(\mathbf{w})$ is rather complex in form, it can be shown to be a non-decreasing, concave function of $|w_j|$. This means that it can be solved by an iteratively reweighted ℓ_1 problem of the form

$$\mathbf{w}^{t+1} = \arg\min_{\mathbf{w}} \text{NLL}(\mathbf{w}) + \sum_j \lambda_j^{(t)}|w_j| \tag{13.172}$$

In (Wipf and Nagarajan 2010), the following procedure for setting the penalty terms is suggested (based on a convex bound to the penalty function). We initialize with $\lambda_j^{(0)} = 1$, and then at iteration $t+1$, compute $\lambda_j^{(t+1)}$ by iterating the following equation a few times:[8]

$$\lambda_j \leftarrow \left[\mathbf{X}_{:,j} \left(\sigma^2 \mathbf{I} + \mathbf{X} \mathrm{diag}(1/\lambda_j) \mathrm{diag}(|w_j^{(t+1)}|) \right)^{-1} \mathbf{X}^T)^{-1} \mathbf{X}_{:,j} \right]^{\frac{1}{2}} \tag{13.173}$$

We see that the new penalty λ_j depends on *all* the old weights. This is quite different from the adaptive lasso method of Section 13.6.2.

To understand this difference, consider the noiseless case where $\sigma^2 = 0$, and assume $D \gg N$. In this case, there are $\binom{D}{N}$ solutions which perfectly reconstruct the data, $\mathbf{X}\mathbf{w} = \mathbf{y}$, and which have sparsity $||\mathbf{w}||_0 = N$; these are called basic feasible solutions or BFS. What we want are solutions that satsify $\mathbf{X}\mathbf{w} = \mathbf{y}$ but which are much sparser than this. Suppose the method has found a BFS. We do not want to increase the penalty on a weight just because it is small (as in adaptive lasso), since that will just reinforce our current local optimum. Instead, we want to increase the penalty on a weight if it is small and if we have $||\mathbf{w}^{(t+1)}|| < N$. The covariance term $(\mathbf{X}\mathrm{diag}(1/\lambda_j)\mathrm{diag}(|w_j^{(t+1)}|))^{-1}$ has this effect: if $\mathbf{w}$ is a BFS, this matrix will be full rank, so the penalty will not increase much, but if $\mathbf{w}$ is sparser than N, the matrix will not be full rank, so the penalties associated with zero-valued coefficients will increase, thus reinforcing this solution (Wipf and Nagarajan 2010).

13.7.5 ARD for logistic regression

Now consider binary logistic regression, $p(y|\mathbf{x}, \mathbf{w}) = \mathrm{Ber}(y|\mathrm{sigm}(\mathbf{w}^T\mathbf{x}))$, using the same Gaussian prior, $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{A}^{-1})$. We can no longer use EM to estimate $\boldsymbol{\alpha}$, since the Gaussian prior is not conjugate to the logistic likelihood, so the E step cannot be done exactly. One approach is to use a variational approximation to the E step, as discussed in Section 21.8.1.1. A simpler approach is to use a Laplace approximation (see Section 8.4.1) in the E step. We can then use this approximation inside the same EM procedure as before, except we no longer need to update β. Note, however, that this is not guaranteed to converge.

An alternative is to use the techniques from Section 13.7.4.3. In this case, we can use exact methods to compute the inner weighted ℓ_1 regularized logistic regression problem, and no approximations are required.

13.8 Sparse coding *

So far, we have been concentrating on sparse priors for supervised learning. In this section, we discuss how to use them for unsupervised learning.

In Section 12.6, we discussed ICA, which is like PCA except it uses a non-Gaussian prior for the latent factors $\mathbf{z}_i$. If we make the non-Gaussian prior be sparsity promoting, such as a Laplace distribution, we will be approximating each observed vector $\mathbf{x}_i$ as a sparse combination of basis vectors (columns of $\mathbf{W}$); note that the sparsity pattern (controlled by $\mathbf{z}_i$) changes from data case to data case. If we relax the constraint that $\mathbf{W}$ is orthogonal, we get a method called

8. The algorithm in (Wipf and Nagarajan 2007) is equivalent to a single iteration of Equation 13.173. However, since the equation is cheap to compute (only $O(ND||\mathbf{w}^{(t+1)}||_0)$ time), it is worth iterating a few times before solving the more expensive ℓ_1 problem.

Method	$p(\mathbf{z}_i)$	$p(\mathbf{W})$	$\mathbf{W}$ orthogonal
PCA	Gauss	-	yes
FA	Gauss	-	no
ICA	Non-Gauss	-	yes
Sparse coding	Laplace	-	no
Sparse PCA	Gauss	Laplace	maybe
Sparse MF	Laplace	Laplace	no

Table 13.3 Summary of various latent factor models. A dash "-" in the $p(\mathbf{W})$ column means we are performing ML parameter estimation rather than MAP parameter estimation. Summary of abbreviations: PCA = principal components analysis; FA = factor analysis; ICA = independent components analysis; MF = matrix factorization.

sparse coding. In this context, we call the factor loading matrix $\mathbf{W}$ a **dictionary**; each column is referred to as an **atom**.[9] In view of the sparse representation, it is common for $L > D$, in which case we call the representation **overcomplete**.

In sparse coding, the dictionary can be fixed or learned. If it is fixed, it is common to use a wavelet or DCT basis, since many natural signals can be well approximated by a small number of such basis functions. However, it is also possible to learn the dictionary, by maximizing the likelihood

$$\log p(\mathcal{D}|\mathbf{W}) = \sum_{i=1}^{N} \log \int_{\mathbf{z}_i} \mathcal{N}(\mathbf{x}_i|\mathbf{W}\mathbf{z}_i, \sigma^2\mathbf{I})p(\mathbf{z}_i)d\mathbf{z}_i \tag{13.174}$$

We discuss ways to optimize this below, and then we present several interesting applications.

Do not confuse sparse coding with **sparse PCA** (see e.g., (Witten et al. 2009; Journee et al. 2010)): this puts a sparsity promoting prior on the regression weights $\mathbf{W}$, whereas in sparse coding, we put a sparsity promoting prior on the latent factors $\mathbf{z}_i$. Of course, the two techniques can be combined; we call the result **sparse matrix factorization**, although this term is non-standard. See Table 13.3 for a summary of our terminology.

13.8.1 Learning a sparse coding dictionary

Since Equation 13.174 is a hard objective to maximize, it is common to make the following approximation:

$$\log p(\mathcal{D}|\mathbf{W}) \approx \sum_{i=1}^{N} \max_{\mathbf{z}_i} \left[\log \mathcal{N}(\mathbf{x}_i|\mathbf{W}\mathbf{z}_i, \sigma^2\mathbf{I}) + \log p(\mathbf{z}_i)\right] \tag{13.175}$$

If $p(\mathbf{z}_i)$ is Laplace, we can rewrite the NLL as

$$\text{NLL}(\mathbf{W}, \mathbf{Z}) = \sum_{i=1}^{N} \frac{1}{2}||\mathbf{x}_i - \mathbf{W}\mathbf{z}_i||_2^2 + \lambda||\mathbf{z}_i||_1 \tag{13.176}$$

9. It is common to denote the dictionary by $\mathbf{D}$, and to denote the latent factors by $\boldsymbol{\alpha}_i$. However, we will stick with the $\mathbf{W}$ and $\mathbf{z}_i$ notation.

To prevent $\mathbf{W}$ from becoming arbitrarily large, it is common to constrain the ℓ_2 norm of its columns to be less than or equal to 1. Let us denote this constraint set by

$$\mathcal{C} = \{\mathbf{W} \in \mathbb{R}^{D \times L} \quad \text{s.t.} \quad \mathbf{w}_j^T \mathbf{w}_j \leq 1\} \tag{13.177}$$

Then we want to solve $\min_{\mathbf{W} \in \mathcal{C}, \mathbf{Z} \in \mathbb{R}^{N \times L}} \text{NLL}(\mathbf{W}, \mathbf{Z})$. For a fixed $\mathbf{z}_i$, the optimization over $\mathbf{W}$ is a simple least squares problem. And for a fixed dictionary $\mathbf{W}$, the optimization problem over $\mathbf{Z}$ is identical to the lasso problem, for which many fast algorithms exist. This suggests an obvious iterative optimization scheme, in which we alternate between optimizing $\mathbf{W}$ and $\mathbf{Z}$. (Mumford 1994) called this kind of approach an **analysis-synthesis** loop, where estimating the basis $\mathbf{W}$ is the analysis phase, and estimating the coefficients $\mathbf{Z}$ is the synthesis phase. In cases where this is too slow, more sophisticated algorithms can be used, see e.g., (Mairal et al. 2010).

A variety of other models result in an optimization problem that is similar to Equation 13.176. For example, **non-negative matrix factorization** or **NMF** (Paatero and Tapper 1994; Lee and Seung 2001) requires solving an objective of the form

$$\min_{\mathbf{W} \in \mathcal{C}, \mathbf{Z} \in \mathbb{R}^{L \times N}} \frac{1}{2} \sum_{i=1}^{N} ||\mathbf{x}_i - \mathbf{W}\mathbf{z}_i||_2^2 \quad \text{s.t.} \quad \mathbf{W} \geq 0, \mathbf{z}_i \geq 0 \tag{13.178}$$

(Note that this has no hyper-parameters to tune.) The intuition behind this constraint is that the learned dictionary may be more interpretable if it is a positive sum of positive "parts", rather than a sparse sum of atoms that may be positive or negative. Of course, we can combine NMF with a sparsity promoting prior on the latent factors. This is called **non-negative sparse coding** (Hoyer 2004).

Alternatively, we can drop the positivity constraint, but impose a sparsity constraint on both the factors $\mathbf{z}_i$ and the dictionary $\mathbf{W}$. We call this **sparse matrix factorization**. To ensure strict convexity, we can use an elastic net type penalty on the weights (Mairal et al. 2010) resulting in

$$\min_{\mathbf{W}, \mathbf{Z}} \frac{1}{2} \sum_{i=1}^{N} ||\mathbf{x}_i - \mathbf{W}\mathbf{z}_i||_2^2 + \lambda ||\mathbf{z}_i||_1 \quad \text{s.t.} \quad ||\mathbf{w}_j||_2^2 + \gamma ||\mathbf{w}_j||_1 \leq 1 \tag{13.179}$$

There are several related objectives one can write down. For example, we can replace the lasso NLL with group lasso or fused lasso (Witten et al. 2009).

We can also use other sparsity-promoting priors besides the Laplace. For example, (Zhou et al. 2009) propose a model in which the latent factors $\mathbf{z}_i$ are made sparse using the binary mask model of Section 13.2.2. Each bit of the mask can be generated from a Bernoulli distribution with parameter π, which can be drawn from a beta distribution. Alternatively, we can use a non-parametric prior, such as the beta process. This allows the model to use dictionaries of unbounded size, rather than having to specify L in advance. One can perform Bayesian inference in this model using e.g., Gibbs sampling or variational Bayes. One finds that the effective size of the dictionary goes down as the noise level goes up, due to the Bayesian Occam's razor. This can prevent overfitting. See (Zhou et al. 2009) for details.

13.8.2 Results of dictionary learning from image patches

One reason that sparse coding has generated so much interest recently is because it explains an interesting phenomenon in neuroscience. In particular, the dictionary that is learned by applying

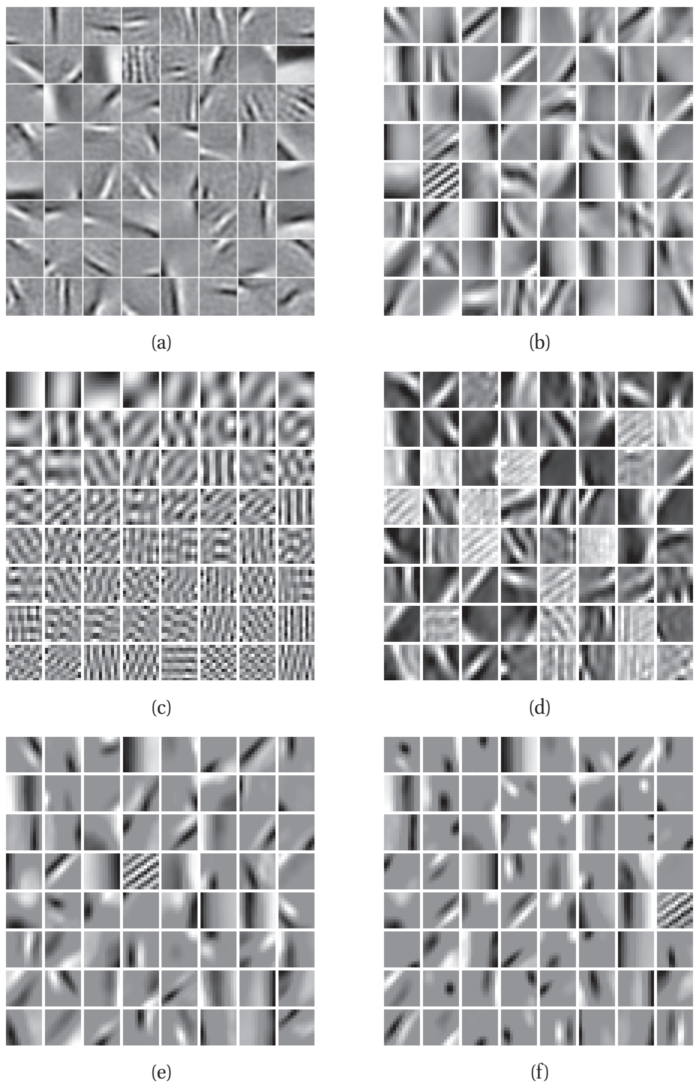


Figure 13.21 Illustration of the filters learned by various methods when applied to natural image patches. (Each patch is first centered and normalized to unit norm.) (a) ICA. Figure generated by `icaBasisDemo`, kindly provided by Aapo Hyvarinen. (b) sparse coding. (c) PCA. (d) non-negative matrix factorization. (e) sparse PCA with low sparsity on weight matrix. (f) sparse PCA with high sparsity on weight matrix. Figure generated by `sparseDictDemo`, written by Julien Mairal.

sparse coding to patches of natural images consists of basis vectors that look like the filters that are found in simple cells in the primary visual cortex of the mammalian brain (Olshausen and Field 1996). In particular, the filters look like bar and edge detectors, as shown in Figure 13.21(b). (In this example, the parameter λ was chosen so that the number of active basis functions (non-zero components of $\mathbf{z}_i$) is about 10.) Interestingly, using ICA gives visually similar results, as shown in Figure 13.21(a). By contrast, applying PCA to the same data results in sinusoidal gratings, as shown in Figure 13.21(c); these do not look like cortical cell response patterns.[10] It has therefore been conjectured that parts of the cortex may be performing sparse coding of the sensory input; the resulting latent representation is then further processed by higher levels of the brain.

Figure 13.21(d) shows the result of using NMF, and Figure 13.21(e-f) show the results of sparse PCA, as we increase the sparsity of the basis vectors.

13.8.3 Compressed sensing

Although it is interesting to look at the dictionaries learned by sparse coding, it is not necessarily very useful. However, there are some practical applications of sparse coding, which we discuss below.

Imagine that, instead of observing the data $\mathbf{x} \in \mathbb{R}^D$, we observe a low-dimensional projection of it, $\mathbf{y} = \mathbf{R}\mathbf{x} + \boldsymbol{\epsilon}$ where $\mathbf{y} \in \mathbb{R}^M$, $\mathbf{R}$ is a $M \times D$ matrix, $M \ll D$, and $\boldsymbol{\epsilon}$ is a noise term (usually Gaussian). We assume $\mathbf{R}$ is a known sensing matrix, corresponding to different linear projections of $\mathbf{x}$. For example, consider an MRI scanner: each beam direction corresponds to a vector, encoded as a row in $\mathbf{R}$. Figure 13.22 illustrates the modeling assumptions.

Our goal is to infer $p(\mathbf{x}|\mathbf{y}, \mathbf{R})$. How can we hope to recover all of $\mathbf{x}$ if we do not measure all of $\mathbf{x}$? The answer is: we can use Bayesian inference with an appropriate prior, that exploits the fact that natural signals can be expressed as a weighted combination of a small number of suitably chosen basis functions. That is, we assume $\mathbf{x} = \mathbf{W}\mathbf{z}$, where $\mathbf{z}$ has a sparse prior, and $\mathbf{W}$ is suitable dictionary. This is called **compressed sensing** or **compressive sensing** (Candes et al. 2006; Baruniak 2007; Candes and Wakin 2008; Bruckstein et al. 2009).

For CS to work, it is important to represent the signal in the right basis, otherwise it will not be sparse. In traditional CS applications, the dictionary is fixed to be a standard form, such as wavelets. However, one can get much better performance by learning a domain-specific dictionary using sparse coding (Zhou et al. 2009). As for the sensing matrix $\mathbf{R}$, it is often chosen to be a random matrix, for reasons explained in (Candes and Wakin 2008). However, one can get better performance by adapting the projection matrix to the dictionary (Seeger and Nickish 2008; Chang et al. 2009).

13.8.4 Image inpainting and denoising

Suppose we have an image which is corrupted in some way, e.g., by having text or scratches sparsely superimposed on top of it, as in Figure 13.23. We might want to estimate the underlying

10. The reason PCA discovers sinusoidal grating patterns is because it is trying to model the covariance of the data, which, in the case of image patches, is translation invariant. This means cov $[I(x, y), I(x', y')] = f\left[(x - x')^2 + (y - y')^2\right]$ for some function f, where $I(x, y)$ is the image intensity at location (x, y). One can show (Hyvarinen et al. 2009, p125) that the eigenvectors of a matrix of this kind are always sinusoids of different phases, i.e., PCA discovers a Fourier basis.

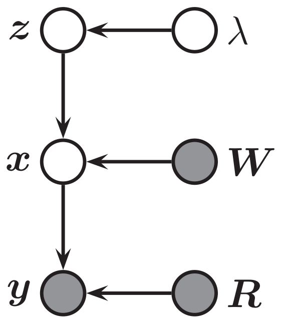
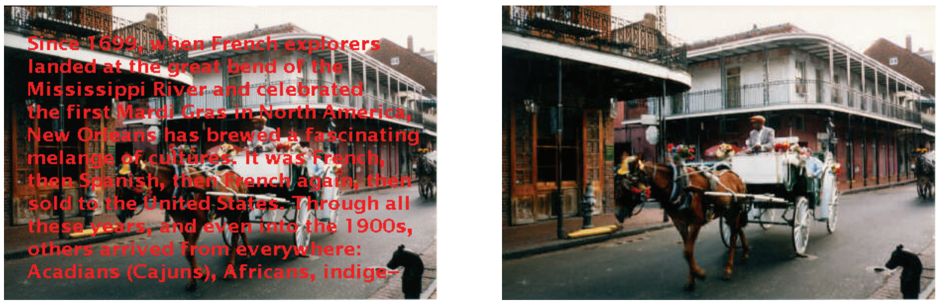


Figure 13.22 Schematic DGM for compressed sensing. We observe a low dimensional measurement $\mathbf{y}$ generated by passing $\mathbf{x}$ through a measurement matrix $\mathbf{R}$, and possibly subject to observation noise with variance σ^2. We assume that $\mathbf{x}$ has a sparse decomposition in terms of the dictionary $\mathbf{W}$ and the latent variables $\mathbf{z}$. the parameter λ controlls the sparsity level.


Since 1699, when French explorers landed at the great bend of the Mississippi River and celebrated the first Mardi Gras in North America, New Orleans has brewed a fascinating melange of cultures. It was French, then Spanish, then French again, then sold to the United States. Through all these years, and even into the 1900s, others arrived from everywhere: Acadians (Cajuns), Africans, indige-

(a) (b)

Figure 13.23 An example of image inpainting using sparse coding. Left: original image. Right: reconstruction. Source: Figure 13 of (Mairal et al. 2008). Used with kind permission of Julien Mairal.

"clean" image. This is called **image inpainting**. One can use similar techniques for **image denoising**.

We can model this as a special kind of compressed sensing problem. The basic idea is as follows. We partition the image into overlapping patches, $\mathbf{y}_i$, and concatenate them to form $\mathbf{y}$. We define $\mathbf{R}$ so that the i'th row selects out patch i. Now define $\mathcal{V}$ to be the visible (uncorrupted) components of $\mathbf{y}$, and $\mathcal{H}$ to be the hidden components. To perform image inpainting, we just compute $p(\mathbf{y}_{\mathcal{H}}|\mathbf{y}_{\mathcal{V}}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the model parameters, which specify the dictionary $\mathbf{W}$ and the sparsity level λ of $\mathbf{z}$. We can either learn a dictionary offline from a database of images, or we can learn a dictionary just for this image, based on the non-corrupted patches.

Figure 13.23 shows this technique in action. The dictionary (of size 256 atoms) was learned from 7×10^6 undamaged 12×12 color patches in the 12 mega-pixel image.

An alternative approach is to use a graphical model (e.g., the **fields of experts** model (S.

and Black 2009)) which directly encodes correlations between neighboring image patches, rather than using a latent variable model. Unfortunately such models tend to be computationally more expensive.

Exercises

Exercise 13.1 Partial derivative of the RSS

Define

$$RSS(\mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \tag{13.180}$$

a. Show that

$$\begin{aligned}\frac{\partial}{\partial w_k} RSS(\mathbf{w}) &= a_k w_k - c_k && (13.181)\\ a_k &= 2\sum_{i=1}^{n} x_{ik}^2 = 2||\mathbf{x}_{:,k}||^2 && (13.182)\\ c_k &= 2\sum_{i=1}^{n} x_{ik}(y_i - \mathbf{w}_{-k}^T \mathbf{x}_{i,-k}) = 2\mathbf{x}_{:,k}^T \mathbf{r}_k && (13.183)\end{aligned}$$

where $\mathbf{w}_{-k} = \mathbf{w}$ without component k, $\mathbf{x}_{i,-k}$ is $\mathbf{x}_i$ without component k, and $\mathbf{r}_k = \mathbf{y} - \mathbf{w}_{-k}^T \mathbf{x}_{:,-k}$ is the residual due to using all the features except feature k. Hint: Partition the weights into those involving k and those not involving k.

b. Show that if $\frac{\partial}{\partial w_k} RSS(\mathbf{w}) = 0$, then

$$\hat{w}_k = \frac{\mathbf{x}_{:,k}^T \mathbf{r}_k}{||\mathbf{x}_{:,k}||^2} \tag{13.184}$$

Hence when we sequentially add features, the optimal weight for feature k is computed by computing orthogonally projecting $\mathbf{x}_{:,k}$ onto the current residual.

Exercise 13.2 Derivation of M step for EB for linear regression

Derive Equations 13.166 and 13.168. Hint: the following identity should be useful

$$\begin{aligned}\mathbf{\Sigma}\mathbf{X}^T\mathbf{X} &= \mathbf{\Sigma}\mathbf{X}^T\mathbf{X} + \beta^{-1}\mathbf{\Sigma}\mathbf{A} - \beta^{-1}\mathbf{\Sigma}\mathbf{A} && (13.185)\\ &= \mathbf{\Sigma}(\mathbf{X}^T\mathbf{X}\beta + \mathbf{A})\beta^{-1} - \beta^{-1}\mathbf{\Sigma}\mathbf{A} && (13.186)\\ &= (\mathbf{A} + \beta\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X}\beta + \mathbf{A})\beta^{-1} - \beta^{-1}\mathbf{\Sigma}\mathbf{A} && (13.187)\\ &= (\mathbf{I} - \mathbf{A}\mathbf{\Sigma})\beta^{-1} && (13.188)\end{aligned}$$

Exercise 13.3 Derivation of fixed point updates for EB for linear regression

Derive Equations 13.169 and 13.170. Hint: The easiest way to derive this result is to rewrite $\log p(\mathcal{D}|\boldsymbol{\alpha}, \beta)$ as in Equation 8.54. This is exactly equivalent, since in the case of a Gaussian prior and likelihood, the posterior is also Gaussian, so the Laplace "approximation" is exact. In this case, we get

$$\begin{aligned}\log p(\mathcal{D}|\boldsymbol{\alpha}, \beta) &= \frac{N}{2}\log\beta - \frac{\beta}{2}||\mathbf{y} - \mathbf{X}\mathbf{w}||^2\\ &+ \frac{1}{2}\sum_j \log\alpha_j - \frac{1}{2}\mathbf{m}^T\mathbf{A}\mathbf{m} + \frac{1}{2}\log|\mathbf{\Sigma}| - \frac{D}{2}\log(2\pi) && (13.189)\end{aligned}$$

The rest is straightforward algebra.

Exercise 13.4 Marginal likelihood for linear regression

Suppose we use a g-prior of the form $\boldsymbol{\Sigma}_\gamma = g(\mathbf{X}_\gamma^T \mathbf{X}_\gamma)^{-1}$. Show that Equation 13.16 simplifies to

$$p(\mathcal{D}|\boldsymbol{\gamma}) \propto (1+g)^{-D_\gamma/2}(2b_\sigma + S(\boldsymbol{\gamma}))^{-(2a_\sigma+N-1)/2} \quad (13.190)$$

$$S(\boldsymbol{\gamma}) = \mathbf{y}^T\mathbf{y} - \frac{g}{1+g}\mathbf{y}^T\mathbf{X}_\gamma(\mathbf{X}_\gamma^T\mathbf{X}_\gamma)^{-1}\mathbf{X}_\gamma^T\mathbf{y} \quad (13.191)$$

Exercise 13.5 Reducing elastic net to lasso

Define

$$J_1(\mathbf{w}) = |\mathbf{y} - \mathbf{X}\mathbf{w}|^2 + \lambda_2|\mathbf{w}|^2 + \lambda_1|\mathbf{w}|_1 \quad (13.192)$$

and

$$J_2(\mathbf{w}) = |\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\tilde{\mathbf{w}}|^2 + c\lambda_1|\mathbf{w}|_1 \quad (13.193)$$

where $c = (1+\lambda_2)^{-\frac{1}{2}}$ and

$$\tilde{\mathbf{X}} = c\begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda_2}\mathbf{I}_d \end{pmatrix}, \quad \tilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y} \\ \mathbf{0}_{d\times 1} \end{pmatrix} \quad (13.194)$$

Show

$$\arg\min J_1(\mathbf{w}) = c(\arg\min J_2(\mathbf{w})) \quad (13.195)$$

i.e.

$$J_1(c\mathbf{w}) = J_2(\mathbf{w}) \quad (13.196)$$

and hence that one can solve an elastic net problem using a lasso solver on modified data.

Exercise 13.6 Shrinkage in linear regression

(Source: Jaakkola.) Consider performing linear regression with an orthonormal design matrix, so $||\mathbf{x}_{:,k}||_2^2 = 1$ for each column (feature) k, and $\mathbf{x}_{:,k}^T\mathbf{x}_{:,j} = 0$, so we can estimate each parameter w_k separately.

Figure 13.24 plots $\hat{w}_k$ vs $c_k = 2\mathbf{y}^T\mathbf{x}_{:,k}$, the correlation of feature k with the response, for 3 different esimation methods: ordinary least squares (OLS), ridge regression with parameter λ_2, and lasso with parameter λ_1.

a. Unfortunately we forgot to label the plots. Which method does the solid (1), dotted (2) and dashed (3) line correspond to? Hint: see Section 13.3.3.
b. What is the value of λ_1?
c. What is the value of λ_2?

Exercise 13.7 Prior for the Bernoulli rate parameter in the spike and slab model

Consider the model in Section 13.2.1. Suppose we put a prior on the sparsity rates, $\pi_j \sim \text{Beta}(\alpha_1, \alpha_2)$. Derive an expression for $p(\boldsymbol{\gamma}|\boldsymbol{\alpha})$ after integrating out the π_j's. Discuss some advantages and disadvantages of this approach compared to assuming $\pi_j = \pi_0$ for fixed π_0.

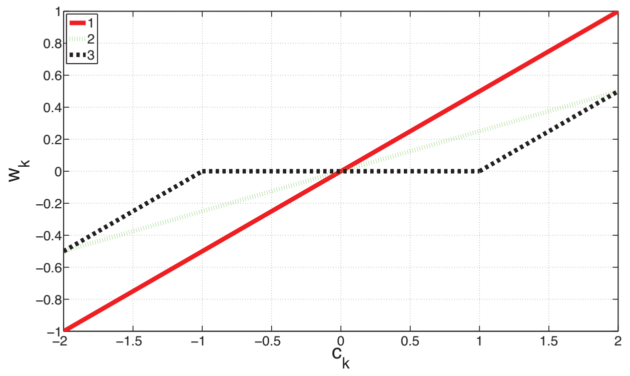


Figure 13.24 Plot of $\hat{w}_k$ vs amount of correlation c_k for three different estimators.

Exercise 13.8 Deriving E step for GSM prior

Show that

$$\mathbb{E}\left[\frac{1}{\tau_j^2}|w_j\right] = \frac{\pi'(w_j)}{|w_j|} \tag{13.197}$$

where $\pi(w_j) = -\log p(w_j)$ and $p(w_j) = int\mathcal{N}(w_j|0,\tau_j^2)p(\tau_j^2)d\tau_j^2$. Hint 1:

$$\frac{1}{\tau_j^2}\mathcal{N}(w_j|0,\tau_j^2) \propto \frac{1}{\tau_j^2}\exp(-\frac{w_j^2}{2\tau_j^2}) \tag{13.198}$$

$$= \frac{-1}{|w_j|}\frac{-2w_j}{2\tau_j^2}\exp(-\frac{w_j^2}{2\tau_j^2}) \tag{13.199}$$

$$= \frac{-1}{|w_j|}\frac{d}{d|w_j|}\mathcal{N}(w_j|0,\tau_j^2) \tag{13.200}$$

Hint 2:

$$\frac{d}{d|w_j|}p(w_j) = \frac{1}{p(w_j)}\frac{d}{d|w_j|}\log p(w_j) \tag{13.201}$$

Exercise 13.9 EM for sparse probit regression with Laplace prior

Derive an EM algorithm for fitting a binary probit classifier (Section 9.4) using a Laplace prior on the weights. (If you get stuck, see (Figueiredo 2003; Ding and Harrison 2010).)

Exercise 13.10 GSM representation of group lasso

Consider the prior $\tau_j^2 \sim \text{Ga}(\delta, \rho^2/2)$, ignoring the grouping issue for now. The marginal distribution induced on the weights by a Gamma mixing distribution is called the **normal Gamma** distribution and is given by

$$\text{NG}(w_j|\delta,\rho) = \int \mathcal{N}(w_j|0,\tau_j^2)\text{Ga}(\tau_j^2|\delta,\rho^2/2)d\tau_j^2 \tag{13.202}$$

$$= \frac{1}{Z}|w_j|^{\delta-1/2}\,\mathcal{K}_{\delta-\frac{1}{2}}(\rho|w_j|) \tag{13.203}$$

$$1/Z = \frac{\rho^{\delta+\frac{1}{2}}}{\sqrt{\pi}\,2^{\delta-1/2}\,\rho(\delta)} \tag{13.204}$$

where $\mathcal{K}_\alpha(x)$ is the modified Bessel function of the second kind (the `besselk` function in Matlab).

Now suppose we have the following prior on the variances

$$p(\boldsymbol{\sigma}_{1:D}^2) = \prod_{g=1}^{G} p(\boldsymbol{\sigma}_{1:d_g}^2),\ p(\boldsymbol{\sigma}_{1:d_g}^2) = \prod_{j\in g}\text{Ga}(\tau_j^2|\delta_g,\rho^2/2) \tag{13.205}$$

The corresponding marginal for each group of weights has the form

$$p(\mathbf{w}_g) \propto |u_g|^{\delta_g-d_g/2}\,\mathcal{K}_{\delta_g-d_g/2}(\rho u_g) \tag{13.206}$$

where

$$u_g \triangleq \sqrt{\sum_{j\in g} w_{g,j}^2} = ||\mathbf{w}_g||_2 \tag{13.207}$$

Now suppose $\delta_g = (d_g+1)/2$, so $\delta_g - d_g/2 = \frac{1}{2}$. Conveniently, we have $\mathcal{K}_{\frac{1}{2}}(z) = \sqrt{\frac{\pi}{2z}}\exp(-z)$. Show that the resulting MAP estimate is equivalent to group lasso.

Exercise 13.11 Projected gradient descent for ℓ_1 regularized least squares

Consider the BPDN problem $\text{argmin}_{\boldsymbol{\theta}}\,\text{RSS}(\boldsymbol{\theta}) + \lambda||\boldsymbol{\theta}||_1$. By using the split variable trick introducted in Section 7.4 (i.e., by defining $\boldsymbol{\theta} = [\boldsymbol{\theta}_+,\boldsymbol{\theta}_-]$), rewrite this as a quadratic program with a simple bound constraint. Then sketch how to use projected gradient descent to solve this problem. (If you get stuck, consult (Figueiredo et al. 2007).)

Exercise 13.12 Subderivative of the hinge loss function

Let $f(x) = (1-x)_+$ be the hinge loss function, where $(z)_+ = \max(0,z)$. What are $\partial f(0)$, $\partial f(1)$, and $\partial f(2)$?

Exercise 13.13 Lower bounds to convex functions

Let f be a convex function. Explain how to find a global affine lower bound to f at an arbitrary point $\mathbf{x} \in \text{dom}(f)$.