LOG-PENALIZED LINEAR REGRESSION

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Joshua A. Sweetkind-Singer

June 2004

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Thomas Cover
(Principal Adviser)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Stephen Boyd

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Benjamin Van Roy

Approved for the University Committee on Graduate Studies.

Abstract

This dissertation discusses *log-penalized* linear regression, given by

$$\hat{\mathbf{b}}_{\text{log penalty}} = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda \sum_j \ln(|b_j| + \delta)$$

and compares it against the more common methods known as *ridge* and *the lasso* whose respective forms are

$$\hat{\mathbf{b}}_{\text{ridge}} = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda \sum_j b_j^2$$
$$\hat{\mathbf{b}}_{\text{lasso}} = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda \sum_j |b_j|$$

In the above minimizations, λ is a complexity parameter. The log penalty form has an additional precision parameter δ. The log penalty yields sparser solutions than the lasso, making it particularly appropriate for overcomplete problems, is tractably solved for large numbers of predictors, and has a nice motivation via the minimum description length principle and the concept of *asymptotic optimality.* Somewhat surprisingly, the ridge and lasso penalties can be reformulated using these principles so that they too can be seen to have log-like penalties, leading to the conjecture that in some sense "all penalties are log penalties". Experiments with the log penalty indicate that it performs better than either ridge or the lasso when the true underlying parameter vector is sparse.

Acknowledgments

From among all the people to whom this thesis owes its existence, I would like to first thank my wife Julie. We met when I was a graduate student and she knew that attaching her life to mine would inescapably include her in my prolonged struggle to obtain a PhD, with its concommitant poverty. She married me anyway. Without her faith in me, without her willingness to help me find the necessary seclusion and focus, I would in all probability be struggling still.

I must also thank my parents, for they nurtured in me the love of learning that brought me here.

I have had many excellent teachers at Stanford and I would like to thank them all: Stephen Boyd, Jerry Friedman, Trevor Hastie, Rob Tibshirani, and Ben Van Roy for their truly exceptional teaching. I feel honored to have learned from you.

I would like to thank professor Abbas El Gamal, for helping me get in to Stanford.

Every day at Stanford has been brighter because of the presence of my colleagues and friends: Max Kamenetsky, Dan O'Neill, Maya Gupta, David Julian, Arak Sutivong, Jon Yard, Young-Han Kim, George Gemelos, and Styrmir Sigurjónsson. All of them have helped me kick around and refine the ideas in this dissertation. David Julian in particular spent hours with me discussing the ideas that eventually became my thesis. It was at his encouragement that I first presented the germ of these ideas at an information theory conference. I would also especially like to thank Young-Han Kim for reading the first draft of the dissertation and for making numerous corrections and suggestions.

None at Stanford has had a greater intellectual influence on me than my adviser, Tom Cover. Our every conversation has been buoyed by his exuberance and fascination for problem-solving. His personality is an unusual mix of both mathematical rigor and jazz-like spontaneity. I recall, when I took his information theory classes, how any question or challenge from a student was met with a rolling-up of the sleeves and an extemporaneous attempt to solve the problem. Usually, once ideas are set in a textbook they fossilize, but Tom kept bringing them back to life, recombining them to make conjectures and answer off-the-cuff questions. It was as though the class were merely a pretext for engaging in true research. He inspired me.

Contents

List of Figures

Chapter 1

Introduction

Regularization penalties are often employed in linear regression as a remedy for the problem of overfitting associated with the ordinary least squares method. Two very common forms of penalized linear regression are *ridge* and *the lasso*, which use the l^2 and l^1 penalties respectively, and whose minimizations have the forms

$$\hat{\mathbf{b}}_{\text{ridge}} = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda \sum_j b_j^2 \tag{1.1}$$

$$\hat{\mathbf{b}}_{\text{lasso}} = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda \sum_j |b_j| \tag{1.2}$$

where λ is a free parameter whose value must be estimated using a method such as cross-validation, or using an information criterion such as Akaike's information criterion (AIC) or the Bayesian information criterion (BIC) [1, 8]. This dissertation motivates the use of another regularization penalty, the *log penalty*, whose form is

$$\hat{\mathbf{b}}_{\text{log penalty}} = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda \sum_j \ln(|b_j| + \delta) \tag{1.3}$$

In all three forms above, λ is a free parameter whose value determines the complexity of the resulting solution, with higher values of λ yielding lower-complexity solutions. Complexity will be discussed in detail in later chapters. For the moment, it is sufficient to think of complexity as more or less synonymous with flexibility. Higher-complexity solutions have more flexibility to fit the data, and hence achieve lower residual error.

In the log-penalized form (1.3) above, there is a second free parameter δ that essentially determines the precision to which we wish to describe the coefficients of $\mathbf{b}$. It naturally arises when we discuss δ-quantized codes later on.

The log penalty is not new, but it does not seem to have received much attention in the statistics literature. Those that have mentioned it or touched upon it [7, 11, 12, 15], have not focused on it directly, but rather as ancillary to some other problem or method, and have not offered a theoretical motivation for its use as a regularization penalty. As it turns out, the log penalty can be strongly motivated by appealing to the minimum description length (MDL) principle [15], which interprets penalties as coding costs, and from which perspective the log penalty emerges as entirely natural.

The analysis makes use of the concept of *asymptotic optimality* and argues that the log penalty is theoretically justified from an MDL perspective because it corresponds to an asymptotically optimal coding cost. This approach also sheds light on the performance of the ridge and lasso penalties, showing that, although they do not appear to be asymptotically optimal, they can be reinterpreted as effectively leading to log-like penalties that are asymptotically optimal when the mechanism for choosing λ is taken into account. While the dissertation proves this only for ridge and the lasso, it seems a reasonable conjecture that a large class of penalties leads to log-like penalties in the same fashion, and that it may be more or less true that, in some sense, "all penalties are log penalties". To this extent, then, the approach offers a unifying perspective on penalized linear regression in general.

The log penalty yields sparse solutions. In fact, its solutions are generally sparser than those of the lasso, which is also known to yield sparse solutions, and therefore, like the lasso, the log penalty can be used as a continuous subset selection method. The problem of choosing the best subset of predictors from which to build a linear model, known as the *subset selection problem*[8], is inherently intractable, since there are 2^p subsets to consider from among p predictors. While there are many theoretically sound information criteria for preferring one subset of predictors over another, including AIC, BIC, and Rissanen's MDL criterion [1, 8, 15], one cannot tractably apply any of these methods to all subsets for large p. One possible solution to the problem is to find a method that somehow considers all possibilities simultaneously and constitutes a kind of descent method among the space of possible model instances. Such methods are called *continuous subset selection* methods. The lasso and the log penalty both fall into this category. The log penalty is particularly apt for *overcomplete* problems, in which the number of predictors p exceeds the number of data points n, since in this case ultra-sparse solutions are sought, and this dissertation discusses a method it calls *lasso-winnowing* for improving the efficiency of the log penalty on overcomplete problems.

Regarding efficiency, one of the great virtues of both ridge and the lasso is that they are convex penalties whose corresponding minimizations can be solved using standard convex optimization techniques. (In the case of ridge, the solution is actually closed-form and no descent method is required.) By contrast, the log penalty is not a convex penalty. In fact, it is concave in the positive orthant, so convex optimization techniques are not directly applicable. However, despite the fact that it is not convex, the log penalty minimization can be solved tractably using a method of *iterative linearization* [7]. Iterative linearization results in a sequence of convex minimization problems whose solutions converge to a local minimum of the original non-convex problem. Each convex minimization is a weighted lasso problem, demonstrating the close relationship between the lasso and log-penalized regression.

It is important to stress that, unlike ridge and the lasso, the log penalty will in general have multiple local minima for a given value of λ. This ineluctable feature should not surprise us, since the subset selection problem is itself non-convex, nor should it distress us too much vis à vis ridge and the lasso, since these methods, while convex for a fixed λ, also become non-convex when the mechanism for choosing λ is taken into account. Still, the non-convexity of the log penalty means that, even for a fixed λ, there will not be a unique solution in general to the log-penalized minimization, and the particular solution yielded by iterative linearization will depend upon the choice of the zeroth iterate. Several obvious choices for the zeroth iterate suggest themselves, leading to variations that this dissertation refers to as the *fixed*, *forward*, and *backward* methods, the latter two being related in spirit to the greedy techniques known as forward and backward stepwise regression.

Later on, we'll apply the log penalty to two different regression problems and compare its performance to the more standard methods such as ridge and the lasso. The first regression problem uses simulated data with a known underlying parameter vector. The experiment indicates that the log penalty does indeed yield sparser solutions than the lasso and that these solutions will in general have somewhat lower prediction error than the lasso when the true underlying parameter vector is sparse. Unsurprisingly, however, the experiment also indicated that, when the true underlying parameter vector is not sparse, the log penalty performs a little worse than the lasso. Of course, the lasso itself performs worse than ordinary least squares in this context. The log penalty method is therefore most apt when one has reason to believe that the true solution is sparse, or when a sparse solution is desired for practical reasons.

The second problem is a real classification problem using gene micro-array data. It is an overcomplete problem, which means that the number of predictors exceeds the number of data points, with $6,088$ predictors and only 34 data points. The experiment illustrates the log penalty's strong drive toward sparsity. Using the log penalty, a classifier with very low classification error emerged using just three out of $6,088$ genes.

Chapter 2

Preliminaries

This chapter describes the assumed underlying model that provides the context for linear regression and explains how penalized linear regression can ameliorate the problem of overfitting. It also describes the standard statistical method of cross-validation, which is often used to estimate a good value of the free parameter λ in a penalized linear regression.

2.1 Linear Regression Context

The underlying context for linear regression is as follows. We are given observations consisting of pairs of data $(\mathbf{x}_i, y_i)$, $\mathbf{x}_i \in \mathcal{R}^p$, $y_i \in \mathcal{R}$, $i = 1, \ldots, n$. This data is conveniently represented as $(\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ is the $n \times p$ matrix whose i^{th} row is $\mathbf{x}_i$, and $\mathbf{y}$ is the vector whose i^{th} component is y_i. We posit a linear relationship between $\mathbf{x}$ and y, governed by an underlying but unknown parameter vector $\mathbf{b}_0 \in \mathcal{R}^p$, which relationship is corrupted by the addition of Gaussian noise. That is, we assume

$$y_i = \mathbf{b}_o^T \mathbf{x}_i + z_i, \quad z_i \overset{\text{iid}}{\sim} N(0, \sigma_Z^2) \tag{2.1}$$

or, equivalently, in matrix form, that

$$\mathbf{y} = \mathbf{X}\mathbf{b}_0 + \mathbf{z} \tag{2.2}$$

where $\mathbf{z} \in \mathcal{R}^p$ is the noise vector whose i^{th} component is z_i. The problem addressed by regression is simply to make a good guess, $\hat{\mathbf{b}}$, for the true but unknown parameter vector $\mathbf{b}_0$, based on the data $(\mathbf{X}, \mathbf{y})$.

The simplest kind of estimate we could make is the ordinary least squares solution, given by

$$\hat{\mathbf{b}}_{ls} \overset{\triangle}{=} \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 \tag{2.3}$$

When $\mathbf{X}$ is full rank, with $p \leq n$, the solution is

$$\hat{\mathbf{b}}_{ls} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{2.4}$$

Many texts, including [8], discuss ordinary least squares regression. However, the ordinary least squares estimate *overfits* the data, which is to say that the residual sum of squares error on the data $(\mathbf{X}, \mathbf{y})$, given by $||\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}||^2$, is not a realistic estimate of how well $\hat{\mathbf{b}}$ will perform on new data $(\mathbf{X}', \mathbf{y}')$. It is too low. To see why, note that, by definition, the ordinary least squares solution chooses the vector $\hat{\mathbf{b}}$ that minimizes the residual sum of squares over all possible candidate solutions, even the true solution $\mathbf{b}_0$. That is, the least squares solution performs better on $(\mathbf{X}, \mathbf{y})$ than even the true solution. One way to understand this phenomenon is to realize that, in seeking to find the absolute minimum residual error, the ordinary least squares solution inevitably fits some of the noise as well as the structure inherent in the data.

A common remedy for the problem of overfitting is to affix a penalty term to (2.3), resulting in what is called *penalized* linear regression [8]. The general form is

$$\hat{\mathbf{b}}(\lambda) \overset{\triangle}{=} \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda g(\mathbf{b}) \tag{2.5}$$

Here $g(\mathbf{b})$ is a penalty function from $\mathcal{R}^p$ into $\mathcal{R}$ and $\lambda \in \mathcal{R}$ is a free parameter that controls the complexity of the resulting solution. This dissertation views complexity from the minimum description length (MDL) perspective. The complexity of a solution is defined to be its description-length with respect to a particular coding scheme.

Complexity will be discussed in detail in chapters 3 and 4. Other treatments of complexity in the regression literature use the concept of *effective degrees of freedom* [8], which is quite similar. Both perspectives are attempts to quantify the notion that higher complexity solutions have more flexibility to conform to the data.

The penalty function g essentially describes our preference for one solution over another when both have the same residual sum of squares. According to (2.5), the minimization prefers the solution with smaller g value. Since in general we prefer solutions that are less complex to those that are more complex—other things being equal—g can be interpreted as assigning a complexity to each candidate solution $\mathbf{b}$, with the resulting minimization preferring solutions that are less complex to those that are more complex. Consistent with this interpretation, the penalty functions g used in practice, such as ridge regression and the lasso, are monotonic in each of the components b_j and take their minimum at $\mathbf{0}$, reflecting our preference for solutions with small component values and our bias to call $\mathbf{0}$ the least complex of all possible solutions. These are desirable, but not necessary, features of a penalty function. In theory, all that is required of g to make (2.5) useful is that it be bounded below.

In (2.5) above, λ is a tuning parameter whose value allows us to control the complexity of the resulting estimate, $\hat{\mathbf{b}}(\lambda)$. When λ is zero, we recover the ordinary least squares solution, because the penalty is given zero weight. As λ increases, the penalty is given more and more weight, causing the minimization to yield less and less complex solutions, which, for reasonable g, results in the components of $\hat{\mathbf{b}}(\lambda)$ becoming smaller in absolute value, a phenomenon called *shrinkage* [8]. Ultimately, a high-enough λ value will drive the solution toward g's minimum, which is typically attained at $\mathbf{b} = \mathbf{0}$. The set of solutions mapped out as λ goes from $\mathbf{0}$ to ∞ will be called the *solution path.*

Remark 2.1. Note that penalized linear regression methods such as ridge and the lasso, when applied to a particular data set $(\mathbf{X}, \mathbf{y})$, do not yield a single solution, but rather a path of solutions. Which solution along the path is chosen depends upon which value of λ is chosen, and the methods themselves offer no advice about how to select λ. Different methods for choosing λ lead to different λ values, and therefore to different points along the solution path. It makes no sense then, to talk about *the* ridge or *the* lasso solution, except with respect to some identified method for choosing λ.

2.2 Cross-Validation

As noted, the parameter λ is free in Equation (2.5). Somehow, a decision must be made as to which λ value will yield a good estimate $\hat{\mathbf{b}}(\lambda)$, for the true underlying parameter vector $\mathbf{b}_0$, given the data at hand $(\mathbf{X}, \mathbf{y})$. *Cross-validation*, described in [8], is a general purpose method for estimating a good value of λ. It is by no means the only method, and [8] describes several others. However, it is robust, considered to be a standard method, and hence is the method used throughout this dissertation to solve for λ in all the experiments described in Chapter 6. In particular, the method used is cross-validation with the one-standard-error rule, as described in this and the following section.

In *m-fold cross-validation*, we partition the data points $(\mathbf{x}_i, y_i)$ into m disjoint sets S_k, $k = 1, \ldots, m$, of as equal size as possible. Let $n_k = |S_k|$ denote the size of the k^{th} set. Thus, $\sum_k n_k = n$. The points $(\mathbf{x}_i, y_i)$ are assigned to their sets arbitrarily. These sets can be designated, in matrix form, as $(\mathbf{X}_k, \mathbf{y}_k)$, $k = 1, \ldots, m$, where $\mathbf{X}_k$ is an $n_k \times p$ matrix whose rows are made up of the $\mathbf{x}_i$ in S_k. The order in which the rows are listed in $\mathbf{X}_k$ is unimportant, so long as the components of $\mathbf{y}_k$, made up of the y_i in S_k, are in the corresponding order.. These will be our m *validation* sets. From these sets, we then generate m *training* sets T_k, $k = 1, \ldots, m$. The k^{th} training set consists of all points *not* in the k^{th} validation set. That is,

$$T_k = \cup_{j \neq k} S_j \tag{2.6}$$

The sets T_k can be designated, in matrix form, as $(\tilde{\mathbf{X}}_k, \tilde{\mathbf{y}}_k)$, $k = 1, \ldots, m$, where $\tilde{\mathbf{X}}_k$ is an $(n - n_k) \times p$ matrix. This results in m training/validation pairs (T_k, S_k). For each λ and k, we then let $\hat{\mathbf{b}}_k(\lambda)$ be the solution when the penalized linear regression is applied to T_k with parameter λ. That is

$$\hat{\mathbf{b}}_k(\lambda) \overset{\triangle}{=} \arg\min_{\mathbf{b}} \|\tilde{\mathbf{y}}_k - \tilde{\mathbf{X}}_k \mathbf{b}\|^2 + \lambda g(\mathbf{b}) \tag{2.7}$$

For a given λ, we can estimate how well $\hat{\mathbf{b}}_k(\lambda)$ is likely to perform on future data by testing it out on S_k, yielding

$$\hat{e}_k(\lambda) \overset{\triangle}{=} \frac{1}{n_k} \|\mathbf{y}_k - \mathbf{X}_k \hat{\mathbf{b}}_k(\lambda)\|^2 \tag{2.8}$$

Since our partition of the data gives us m such training/validation pairs, we can perform this estimate m times and average the results, yielding

$$\hat{e}(\lambda) \overset{\triangle}{=} \frac{1}{m} \sum_{k=1}^{m} \hat{e}_k(\lambda) \tag{2.9}$$

$$= \frac{1}{m} \sum_{k=1}^{m} \frac{1}{n_k} \|\mathbf{y}_k - \mathbf{X}_k \hat{\mathbf{b}}_k(\lambda)\|^2 \tag{2.10}$$

A good λ value to use, then, in the original regression problem, on the full data set $(\mathbf{X}, \mathbf{y})$, is the one whose associated error estimate is minimal:

$$\lambda^* \overset{\triangle}{=} \arg\min_{\lambda} \hat{e}(\lambda) \tag{2.11}$$

As λ increases from 0, the shape of the curve $\hat{e}(\lambda)$ is roughly that of a bowl with a bumpy bottom, which is to say that at first the error will be monotonically decreasing, then, after λ has passed through some range of interest (the bumpy bottom), $\hat{e}(\lambda)$ will be monotonically increasing. It is therefore not too difficult using trial and error to discover a reasonable range of λ values to test under cross-validation in order to come up with a value close to λ^*.

Remark 2.2. Since log-penalized linear regression has two free parameters λ and δ, equations (2.7) through (2.11) above must be modified in obvious ways to accommodate the additional δ parameter, yielding the following new equations

$$\begin{aligned} \hat{\mathbf{b}}_k(\delta,\lambda) &\triangleq \arg\min_{\mathbf{b}} \|\tilde{\mathbf{y}}_k - \tilde{\mathbf{X}}_k\mathbf{b}\|^2 + \lambda\sum_j \ln(|b_j| + \delta) && (2.12)\\ \hat{e}_k(\delta,\lambda) &\triangleq \frac{1}{n_k}\|\mathbf{y}_k - \mathbf{X}_k\hat{\mathbf{b}}_k(\delta,\lambda)\|^2 && (2.13)\\ \hat{e}(\delta,\lambda) &= \frac{1}{m}\sum_{k=1}^{m}\frac{1}{n_k}\|\mathbf{y}_k - \mathbf{X}_k\hat{\mathbf{b}}_k(\delta,\lambda)\|^2 && (2.14)\\ (\delta^*,\lambda^*) &\triangleq \arg\min_{\delta,\lambda}\hat{e}(\delta,\lambda) && (2.15)\end{aligned}$$

2.3 The One-Standard-Error Rule

Rather than using λ^* above, the authors in [8] prefer to use a more conservative method that they call the *one-standard-error* rule. This dissertation also uses the one-standard-error rule to calculate optimal λ (and δ) values in all the experiments described in Chapter 6. Among other things, this leads to sparser solutions when the log penalty is used.

As mentioned in the previous section, the curve $\hat{e}(\lambda)$ has a bumpy bottom in the λ range of interest, which is to say that in general it has several local minima. This feature, combined with the vagaries of randomly partitioning the data, selecting a somewhat arbitrary number of validation sets m ([8] has some advice about choosing m), and the inherent noise in any estimation process, means that λ^* may itself be slightly optimistic and may yield an estimate $\hat{\mathbf{b}}(\lambda^*)$ that still overfits the data somewhat. A more conservative approach is to use instead the highest value $\hat{\lambda}$ whose associated estimated prediction error $\hat{e}(\hat{\lambda})$ falls within one standard error of $\hat{e}(\lambda^*)$. That is,

$$\hat{\lambda} \triangleq \max_{\lambda>0}\lambda \quad \text{subject to} \quad \hat{e}(\lambda) \le \hat{e}(\lambda^*) + s(\lambda^*) \qquad (2.16)$$

where

$$s(\lambda) \triangleq \sqrt{\frac{1}{m(m-1)} \sum_k (\hat{e}_k(\lambda) - \hat{e}(\lambda))^2} \tag{2.17}$$

Since higher values of λ yield less complex solutions, this method essentially selects the simplest solution, $\hat{\mathbf{b}}(\hat{\lambda})$, whose inferiority with respect to the solution $\hat{\mathbf{b}}(\lambda^*)$, in terms of estimated prediction error, is wholly attributable to noise in the cross-validation process.

Remark 2.3. As noted in the previous section, when the log penalty is used there are two free parameters λ and δ, requiring that the one-standard-error rule be modified. In this case, it does not make sense to maximize both δ and λ simultaneously; however, when we reflect that the reason λ is maximized in the single-parameter case is that this yields the least complex solution, then it seems reasonable in the two parameter case to prefer the (δ, λ) pair that yields the sparsest solution. This leads to the following replacements for equations (2.16) and (2.17)

$$(\hat{\delta}, \hat{\lambda}) \triangleq \arg \min_{\delta, \lambda > 0} ||\hat{\mathbf{b}}(\delta, \lambda)||_0 \quad \text{subject to} \quad \hat{e}(\delta, \lambda) \leq \hat{e}(\delta^*, \lambda^*) + s(\delta^*, \lambda^*) \tag{2.18}$$

where $||\hat{\mathbf{b}}(\delta, \lambda)||_0$ is the number of nonzero components in $\hat{\mathbf{b}}(\delta, \lambda)$, and where

$$s(\delta, \lambda) \triangleq \sqrt{\frac{1}{m(m-1)} \sum_k (\hat{e}_k(\delta, \lambda) - \hat{e}(\delta, \lambda))^2} \tag{2.19}$$

The solution to (2.18) need not be unique. One reasonable method of breaking ties is to prefer the $(\hat{\delta}, \hat{\lambda})$ pair with lowest residual sum of squares error $||\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}(\delta, \lambda)||^2$.

Chapter 3

Coding

The log penalty of this dissertation gets its impetus from considerations arising from algorithmic complexity, pioneered by Kolmogorov [10], Chaitin [3], and Solomonoff [16]. The central insight of algorithmic complexity is that the complexity of an object should be defined as the length of the most succinct coding of that object. The objects of interest for our purposes are integers and vectors of integers. Thus, it is important to understand some basics of coding on $\mathcal{Z}$ and $\mathcal{Z}^d$. This chapter introduces these basics. Those familiar with the concepts or who wish not to be bogged down by details may safely skip to Chapter 4, after glancing at Section 3.3 on page 19 and at the definition of asymptotic optimality for δ-quantized codes given in definition 3.10 on page 36.

Ultimately, we will be interested not so much in any particular coding scheme as in the best achievable asymptotic properties of a code: roughly how many bits does it take to code a vector of integers as the vector's components get increasingly large?

3.1 Coding Definitions

This section defines what we mean by a code, and introduces some basic concepts.

Definition 3.1. Let A be a finite or countable set of objects. A *prefix-free code*, (also known as a *prefix code*) $C(\mathbf{m})$, $\mathbf{m} \in A$, is a one-to-one mapping between A and the set of finite-length binary strings $\{0,1\}^*$, such that no string in the range of C is a prefix of any other string in the range of C.

For example, if C is a prefix-free code, then 0110 and 0110101 cannot both be strings in the range of C, since the first is a prefix of the second. See [4] for a longer discussion of prefix-free codes.

In this dissertation, we will typically take A to be either $\mathcal{Z}^d$, the set of d-dimensional vectors with integer components, or a countable subset of $\mathcal{R}^d$. In particular, we will be interested in the countable subset Q^d_δ, defined in Definition 3.8 on page 35.

Remark 3.1. In this dissertation, *all* references to codes are really to prefix-free codes. Sometimes the modifier *prefix-free* is added for emphasis, but even when it is omitted, *code* still means *prefix-free code.*

Definition 3.2. Let $s \in \{0,1\}^*$ be a finite-length binary string. Denote by $|s|$ the length of the string s.

Definition 3.3. Let $s, t \in \{0,1\}^*$ be finite-length binary strings. Denote by st the concatenation of the two strings.

Definition 3.4. Let $C(\mathbf{m})$ be a prefix-free code with domain A. Its associated *code-length function* $L(\mathbf{m})$, also with domain A, is the number of bits required to encode $\mathbf{m}$ using code C. That is

$$L(\mathbf{m}) = |C(\mathbf{m})| \tag{3.1}$$

A basic result from information theory tells us that the code-length function of any prefix-free code must satisfy the following inequality, known as the Kraft Inequality. Conversely, for any function satisfying the Kraft Inequality, there exists a prefix-free code with the corresponding code-lengths.

Theorem 3.1 (Kraft Inequality). *Let $L(\mathbf{m})$ be an integer-valued function on domain A. Then a prefix-free code $C(\mathbf{m})$ on A exists for which $L(\mathbf{m})$ is the associated code-length function if and only if*

$$\sum_{\mathbf{m}\in A} 2^{-L(\mathbf{m})} \leq 1 \tag{3.2}$$

See [4] for a proof.

3.2 A Prefix-Free Code for the Integers

It is a well-known fact from Kolmogorov complexity theory that any integer $m \in \mathcal{Z}$ may be encoded using a prefix-free code requiring no more than $\log(|m|+1) + O(\log\log|m|)$ bits [4, 15, 18]. This section exhibits the construction of such a code, following the standard method.

We define the code $C_Z(m)$ in several stages, creating intermediary codes C_a through C_e, each of whose definitions depends upon the previously defined code. Finally, we define $C_Z(m)$ in terms of C_e, and it will have the asymptotic code-length properties we seek.

We begin with the crudest imaginable code on the non-negative integers:

$$C_a(m) \overset{\triangle}{=} \underbrace{111\ldots111}_{m \text{ times}}0 \tag{3.3}$$

C_a is sometimes called a *unary* code. Figure 3.1 illustrates the C_a encoding of the first few integers. Obviously $C_a(m)$ is prefix-free, since we know the code has terminated as soon as we see a zero. By inspection, one can see that the code-length function $L_a(m)$ corresponding to $C_a(m)$ is given by:

$$L_a(m) = m + 1 \tag{3.4}$$

C_a does not have the asymptotic code-length property we seek. We want a code that

The C_a Code

m	$C_a(m)$
0	0
1	10
2	110
3	1110
4	11110
...	...

Figure 3.1: *The C_a encoding of the first few integers*

takes about $\log |m|$ bits to describe m, whereas C_a requires $m + 1$ bits to describe m. But we will leverage this simple beginning to create a code that has the property we seek.

The next stage in the code $C_b(m)$ is a code on the positive integers, as opposed to the non-negative integers, and we build it using C_a to help us. Let $s(m)$ denote the standard binary representation of the positive integer m, less its leading 1. For example, the standard binary representation of $m = 5$ is 101. Dropping the leading 1 gives us $s(5) = 01$. At the moment we are building a code on just the positive integers, and their binary representations always begin with the symbol 1. Thus, for any $m > 0$, it is sufficient to describe just the symbols following the 1 in its binary representation. Notice that $s(m)$ by itself does not constitute a prefix-free code, since it maps onto the entire set $\{0, 1\}^*$. However, if we precede the string $s(m)$ with a prefix-free encoding of the *length* of the string $s(m)$, then we achieve a prefix-free code, because the initial encoding tells us when the remainder of the codeword terminates. This will be our strategy, and we will use C_a to encode the length of the string $s(m)$. Our new code is

$$C_b(m) \triangleq C_a(|s(m)|)s(m) \tag{3.5}$$

Note that C_b must be prefix-free. The codeword $C_b(m) = C_a(|s(m)|)s(m)$ cannot be the prefix of any longer codeword because all codewords beginning with the bit sequence $C_a(|s(m)|)$ have precisely the same length. Note also that $|s(m)|$ could be as small as zero, when $m = 1$, but that is okay, because C_a is a code on the non-negative

The C_b Code

m	$C_a(m)$	$s(m)$	$C_b(m)$
0	0	[undefined]	[no code exists]
1	10	[empty string]	0
2	110	0	100
3	1110	1	101
4	11110	00	11000
...	...	...	...

Figure 3.2: *The C_b encoding of the first few integers.*

integers, so it is capable of encoding zero. Table 3.2 illustrates the C_b encoding of the first few integers. Now let's calculate the associated code-length function $L_b(m)$.

$$\begin{aligned} L_b(m) &= L_a(|s(m)|) + |s(m)| && (3.6)\\ &\overset{(a)}{=} L_a(\lfloor \log m \rfloor) + \lfloor \log m \rfloor && (3.7)\\ &\overset{(b)}{=} \lfloor \log m \rfloor + 1 + \lfloor \log m \rfloor && (3.8)\\ &= 2\lfloor \log m \rfloor + 1 && (3.9) \end{aligned}$$

where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x. It is easy to verify that the length of $s(m)$ is $\lfloor \log m \rfloor$, (note: all logs in this section are base two), which is used in (a) above. (b) follows from (3.4).

Although $C_b(m)$ just codes the positive integers, it is easy to extend this to a code on the non-negative integers by defining the new code

$$C_c(m) \triangleq C_b(m+1) \tag{3.10}$$

with associated code-length function

$$\begin{aligned} L_c(m) &= L_b(m+1) && (3.11)\\ &= 2\lfloor \log(m+1) \rfloor + 1 && (3.12) \end{aligned}$$

Figure 3.3 illustrates the C_c encoding of the first few integers. Already we are in the

The C_c Code

m	$C_a(m)$	$C_b(m)$	C_c
0	0	[no code exists]	0
1	10	0	100
2	110	100	101
3	1110	101	11000
4	11110	11000	11001
...	...	...	...

Figure 3.3: *The C_c encoding of the first few integers.*

ballpark of what we hope to achieve. C_c takes about $2\log m$ bits to encode m. We would like to cut that in half. One more application of our inductive code construction will do the trick. Let us define:

$$C_d(m) \triangleq C_c(|s(m)|)s(m) \tag{3.13}$$

with associated code-length function

$$\begin{aligned} L_d(m) &= L_c(|s(m)|) + |s(m)| && (3.14)\\ &= L_c(\lfloor \log m \rfloor) + \lfloor \log m \rfloor && (3.15)\\ &= 2\lfloor \log(\lfloor \log m \rfloor + 1)\rfloor + 1 + \lfloor \log m \rfloor && (3.16)\\ &= \lfloor \log m \rfloor + 2\lfloor \log(\lfloor \log m \rfloor + 1)\rfloor + 1 && (3.17) \end{aligned}$$

As before, we extend this to a code on the non-negative integers by defining the new code

$$C_e(m) \triangleq C_d(m+1) \tag{3.18}$$

with associated code-length function

$$\begin{aligned} L_e(m) &= L_d(m+1) && (3.19)\\ &= \lfloor \log(m+1)\rfloor + 2\lfloor \log(\lfloor \log(m+1)\rfloor + 1)\rfloor + 1 && (3.20) \end{aligned}$$

The C_e Code

m	$C_a(m)$	$C_b(m)$	C_c	C_d	C_e
0	0	[no code exists]	0	[no code exists]	0
1	10	0	100	0	1000
2	110	100	101	1000	1001
3	1110	101	11000	1001	10100
4	11110	11000	11001	10100	10101
...	...	...	...	...	...

Figure 3.4: *The C_e encoding of the first few integers.*

Figure 3.4 illustrates the C_e encoding of the first few integers.

Finally, C_e can be extended to the negative integers by affixing a sign bit to the code:

$$C_Z(m) \triangleq \begin{cases} 0C_e(m) & m \geq 0 \\ 1C_e(-m) & m < 0 \end{cases} \tag{3.21}$$

with associated code-length function

$$L_Z(m) = \lfloor \log(|m|+1) \rfloor + 2\lfloor \log(\lfloor \log(|m|+1) \rfloor + 1) \rfloor + 2 \tag{3.22}$$

which has the desired asymptotic code-length property:

$$L_Z(m) = \log(|m|+1) + O(\log\log|m|) \tag{3.23}$$

The reader may wonder why the form $\log(|m|+1)$ was chosen to express the asymptotic property above, rather than the more compact form $\log|m|$, for, surely, if $L_Z(m) = \log(|m|+1) + O(\log\log|m|)$, then also $L_Z(m) = \log|m| + O(\log\log|m|)$. The answer is that not only does the term $\log(|m|+1)$ convey the principal asymptotic order, it also represents a good approximation of the code-length for all m. Importantly, at $m = 0$, it yields an approximate code-length of 0, rather than $-\infty$. Later, we will drop the lower-order $\log\log$ term and use $\log(|m|+1)$ as a reasonably good measure of the complexity of any integer.

3.3 Asymptotic Optimality

Important for the justification of the log penalty is the notion of *asymptotic optimality*. We wish to make precise the idea that a code achieves—in the limit as $\mathbf{m}$ grows large—the smallest code-lengths that it could hope to achieve. This differs from the standard analysis of coding optimality in information theory, in which the goodness of a code is measured by how the code-length grows as the number of symbols to be encoded goes to infinity. In our case, the number of integers to be encoded d stays fixed, and we measure the goodness of the code instead by how well it codes ever larger integers.

In the previous section, we exhibited a code $C_Z(m)$ on the integers that grows asymptotically like $\log |m|$. That is

$$\lim_{|m|\to\infty} \frac{L_Z(m)}{\log |m|} = 1 \tag{3.24}$$

It is natural to ask whether $L(m) \sim \log |m|$ is the best one can do, or whether there possibly exists a code with significantly shorter code-lengths. It turns out that there is a well-defined sense in which this is the best that can be done. While it is possible to design a code whose code-length function $L(m)$ is significantly smaller than $\log |m|$ on a particular set of values of m (infinitely often, in fact), there cannot be an integer M such that $\forall m, |m| > M \Rightarrow L(m) \leq \log |m|$. To see why not, suppose such a code-length function $L(m)$ and integer M existed. Then we derive a contradiction as follows:

$$1 \overset{(a)}{\geq} \sum_{m\in\mathcal{Z}} 2^{-L(m)} \tag{3.25}$$

$$\geq \sum_{m:|m|>M} 2^{-L(m)} \tag{3.26}$$

$$\overset{(b)}{\geq} \sum_{m:|m|>M} 2^{-\log |m|)} \tag{3.27}$$

$$= \sum_{m:|m|>M} \frac{1}{|m|} \tag{3.28}$$

$$= \infty \tag{3.29}$$

where (a) is the Kraft Inequality, Theorem (3.1) on page 14, and (b) follows by hypothesis. We conclude that $L(m)$ must exceed $\log |m|$ infinitely often, and therefore $L(m)/\log |m|$ must exceed 1 infinitely often. If $\lim_{|m|\to\infty} \frac{L(m)}{\log |m|}$ exists, it must be greater than or equal to 1.

Of course, for some codes, the limit might not exist at all, even though the code were one we'd like to characterize as optimal. For example, the code-length function $L(m)$ might not be monotonic in $|m|$, attaining very short code-lengths infinitely often, but not so often as to make the worst case code-length grow faster than $\log |m|$ asymptotically. For example, the Kolmogorov complexity function $K(m)$ of the integers [4] is a code-length function with this property. Any definition of asymptotic optimality must account for this possibility, and there are a few approaches, one of which, taken by Rissanen in [15], is simply to exclude from consideration code-length functions that are not monotonic. Focusing on worst-case code-lengths also suffices, leading to the following definition:

Definition 3.5. A code $C(\mathbf{m})$ on $\mathcal{Z}^d$, with associated code-length function $L(\mathbf{m})$, is *asymptotically optimal* in case

$$\lim_{r\to\infty} \max_{||\mathbf{m}||\leq r} \frac{L(\mathbf{m})}{d \log r} = 1 \tag{3.30}$$

where $||\mathbf{m}|| = \sqrt{\sum m_j^2}$ is the standard Euclidean norm.

Note that for some codes the above limit still might not exist. But, if so, it will be because the worst case code-length is greater than $(d+\epsilon)\log r$ infinitely often and hence we would not want it to qualify as an asymptotically optimal code.

The following theorem justifies the above definition by showing that $d \log r$ is the best asymptotic code-length one could hope to achieve.

Theorem 3.2. *Let $L(\mathbf{m})$ be the code-length function associated with a prefix-free code on $\mathcal{Z}^d$. Then*

$$\liminf_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L(\mathbf{m})}{d\log r} \ge 1 \tag{3.31}$$

Proof: The essence of the proof is that the number of vectors $\mathbf{m}$ in $\mathcal{Z}^d$ satisfying $\|\mathbf{m}\| \le r$ is approximately $c(d)r^d$, where $c(d)$ is a constant depending only on d. Thus, the number of bits required to describe each vector in this set must be approximately $\log c(d)r^d = d\log r + O(1)$. More precisely, let $S(r) \overset{\triangle}{=} \{\mathbf{m} \in \mathcal{Z}^d : \|\mathbf{m}\| \le r\}$. Then

$$\begin{aligned} 1 &\ge \sum_{\mathbf{m}\in\mathcal{Z}^d} 2^{-L(\mathbf{m})} && (3.32)\\ &\ge \sum_{\|\mathbf{m}\|\le r} 2^{-L(\mathbf{m})} && (3.33)\\ &\ge |S(r)| \min_{\|\mathbf{m}\|\le r} 2^{-L(\mathbf{m})} && (3.34)\end{aligned}$$

Taking logs of both sides yields

$$\begin{aligned} 0 &\ge \log|S(r)| + \min_{\|\mathbf{m}\|\le r} -L(\mathbf{m}) \quad \text{which implies} && (3.35)\\ 0 &\ge \log|S(r)| - \max_{\|\mathbf{m}\|\le r} L(\mathbf{m}) \quad \text{leading to} && (3.36)\\ \max_{\|\mathbf{m}\|\le r} L(\mathbf{m}) &\ge \log|S(r)| && (3.37)\end{aligned}$$

Furthermore, there clearly exists a constant c and a threshold R such that $\forall r, r > R \Rightarrow |S(r)| \ge c \cdot r^d$. Therefore, for $r > R$,

$$\begin{aligned} \max_{\|\mathbf{m}\|\le r} L(\mathbf{m}) &\ge \log|S(r)| && (3.38)\\ &\ge d\log r + \log c \quad \text{which implies} && (3.39)\\ \max_{\|\mathbf{m}\|\le r} \frac{L(\mathbf{m})}{d\log r} &\ge \frac{d\log r + \log c}{d\log r} \quad \text{leading to} && (3.40)\\ \liminf_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L(\mathbf{m})}{d\log r} &\ge \liminf_{r\to\infty} \frac{d\log r + \log c}{d\log r} = 1 && (3.41)\end{aligned}$$

Remark 3.2. It should be obvious from (3.24) that the natural code is asymptotically optimal according to Definition 3.5. A full proof for the generalized natural code on $\mathcal{Z}^d$ is given in Section 3.4. However, there is also a sense in which the natural code as we've developed it is not asymptotically optimal. The recursive method used to create the code C_Z can be iterated to create another code C^* with code-length function L^* that is strictly less than L_Z for large m. That is, there exists an integer M, such that

$$\forall m, |m| > M \Rightarrow L^*(m) \leq L_Z(m) \tag{3.42}$$

The code-length function L^* is given by

$$L^*(m) \overset{\triangle}{=} c + \log|m| + \log\log|m| + \log\log\log|m| + \dots \tag{3.43}$$

down to the last iterated logarithm of $|m|$ that is still positive, and where c is a constant. The sum of successive iterated logarithms is known as the $\log^*$ function and is discussed in [4, 15] along with the derivation of the code C^*.

Note that L_Z and L^* have the same first-order term $\log|m|$, but that L^* is more efficient in its lower-order terms for large m. The first order term is all that will be of interest to us in later chapters, so the simpler code C_Z suffices for our purposes.

3.4 The Natural Code On $\mathcal{Z}^d$

Having exhibited an asymptotically optimal code C_Z on the integers, we can now leverage it to create asymptotically optimal codes on $\mathcal{Z}^d$. This section constructs a natural extension of C_Z to $\mathcal{Z}^d$, simply by separately coding each of the components of $\mathbf{m} \in \mathcal{Z}^d$ using C_Z. For this reason, we call the resultant code the *natural code* on $\mathcal{Z}^d$.

Definition 3.6. The *natural code* on $\mathcal{Z}^d$ is the code given by

$$C_{Z^d}(\mathbf{m}) \overset{\triangle}{=} C_Z(m_1)C_Z(m_2)\dots C_Z(m_d) \tag{3.44}$$

Since C_{Z^d} is the concatenation of a fixed number d of prefix-free codes, it too is a prefix-free code. The code-length of $\mathbf{m} \in \mathcal{Z}^d$ is just the sum of the code-lengths of its components:

$$L_{Z^d}(\mathbf{m}) = \sum_{j=1}^{d} L_Z(m_j) \tag{3.45}$$

$$= \sum_{j=1}^{d} \lfloor \log(|m_j| + 1) \rfloor + 2\lfloor \log(\lfloor \log(|m_j| + 1) \rfloor + 1) \rfloor + 2 \tag{3.46}$$

$$\leq \left(\sum_{j=1}^{d} \log(|m_j| + 1) \right) + 2d \log(\log(\|\mathbf{m}\| + 1) + 1) + 2d \tag{3.47}$$

leading to

$$L_{Z^d}(\mathbf{m}) = \sum_{j=1}^{d} \log(|m_j| + 1) + O(\log\log \|\mathbf{m}\|) \tag{3.48}$$

As in the previous section, the principal asymptotic component of L_{Z^d} is described as $\sum_{j=1}^{d} \log(|m_j| + 1)$, rather than as the more compact $\sum_{j=1}^{d} \log(|m_j|)$, in order to emphasize that this expression is a good approximation of the code-length for all $\mathbf{m}$, including $\mathbf{m} = \mathbf{0}$. When $\mathbf{m} = \mathbf{0}$, the more compact form yields the unreasonable value of $-\infty$.

Notation 3.1. Given the close affinity of C_{Z^d} to the original code C_Z on the integers, we will slightly abuse notation. Thus, for $\mathbf{m} \in \mathcal{Z}^d$, we will use the less cumbersome forms $C_Z(\mathbf{m})$ and $L_Z(\mathbf{m})$ to refer to $C_{Z^d}(\mathbf{m})$ and $L_{Z^d}(\mathbf{m})$ respectively.

Theorem 3.3. *The natural code C_Z on $\mathcal{Z}^d$ is asymptotically optimal.*

Proof: We must show that

$$\lim_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L_Z(\mathbf{m})}{d\log r} = 1 \tag{3.49}$$

First, we note that if $\|\mathbf{m}\| \le r$, then

$$|m_j| \le r \quad \text{for all } j \tag{3.50}$$

which implies that

$$\sum_j \log(|m_j|+1) \le d\log(r+1) \tag{3.51}$$

leading to

$$L_Z(\mathbf{m}) \le d\log(r+1) + O(\log\log r) \tag{3.52}$$

The final line above follows from $L_Z(\mathbf{m}) = \sum_j \log(|m_j|+1) + O(\log\log\|\mathbf{m}\|)$, derived at Equation (3.48). Continuing, since Equation (3.52) holds for all $\mathbf{m}$ such that $\|\mathbf{m}\| \le r$, it must hold for the max over this set, yielding

$$\max_{\|\mathbf{m}\|\le r} L_Z(\mathbf{m}) \le d\log(r+1) + O(\log\log r) \tag{3.53}$$

which implies

$$\max_{\|\mathbf{m}\|\le r} \frac{L_Z(\mathbf{m})}{d\log r} \le \frac{d\log(r+1) + O(\log\log r)}{d\log r} \tag{3.54}$$

which implies

$$\limsup_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L_Z(\mathbf{m})}{d\log r} \le \limsup_{r\to\infty} \frac{d\log(r+1) + O(\log\log r)}{d\log r} = 1 \tag{3.55}$$

leading to

$$\lim_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L_Z(\mathbf{m})}{d\log r} = 1 \tag{3.56}$$

where (3.56) follows because we know

$$\liminf_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L_Z(\mathbf{m})}{d\log r} \ge 1 \tag{3.57}$$

from Theorem 3.2 on page 21.

3.5 Indexed Codes

Suppose we have several coding schemes "on hand", any of which can be used to encode $\mathbf{m} \in \mathcal{Z}^d$. Then we can describe $\mathbf{m}$ in two parts. The first part describes the index of a particular code and the second part describes $\mathbf{m}$ using that code. In effect, the set of coding schemes must itself be coded using a coding scheme. This is the idea behind indexed codes. Since in general we are interested in minimal code lengths, we are free to select from among the available coding schemes the one that yields the shortest two-part code-length for $\mathbf{m}$. This is the central notion behind the theory of minimum description-length estimation, covered in Chapter 4.

Definition 3.7. Let $\mathcal{D}$ be a finite or countable set of prefix-free codes. Let A be the union of the domains of the codes in $\mathcal{D}$. Let C_0 be a prefix-free code on $\mathcal{D}$ with associated code-length function L_0. Also, denote by $|C(\mathbf{m})|$ the code-length function associated with $C \in \mathcal{D}$. Then the *indexed code* or *two-part code* C_I on A, constructed from $\mathcal{D}$ and C_0, is

$$C_I(\mathbf{m}) \triangleq C_0(C^*)C^*(\mathbf{m}) \tag{3.58}$$

where C^* satisfies

$$C^* = \arg\min_{C \in \mathcal{D}} L_0(C) + |C(\mathbf{m})| \tag{3.59}$$

Here we take $|C(\mathbf{m})|$ to be ∞ when $\mathbf{m}$ is not in the domain of C. (In case there are more than one C attaining the minimum above, we can arbitrarily break ties by preferring the code whose codeword precedes the others in dictionary order.)

By construction, the code-length function of C_I is

$$L_I(\mathbf{m}) = \arg\min_{C \in \mathcal{D}} L_0(C) + |C(\mathbf{m})| \tag{3.60}$$

Remark 3.3. Note that C^* is the code that attains the shortest total description length for $\mathbf{m}$ from among all codes in $\mathcal{D}$, taking into account the fact that C^* itself must be described using $L_0(C^*)$ bits. Therefore it is not necessarily the case that $C^* = \arg\min_C |C(\mathbf{m})|$. This important fact is at the heart of the MDL principle. When in Chapter 4 we interpret codes as models and code-lengths as their associated complexity, $L_0(C)$ acts as a regularizer on the estimation process, forcing us to pay a price for choosing more complex models.

3.6 The l^2 Code On $\mathcal{Z}^d$

This section describes a code $C_2(\mathbf{m})$ whose associated code-length function depends upon the l^2 norm of $\mathbf{m}$, so we'll call it *the l^2 code.*

We begin by describing a code $C_a^{(n)}(\mathbf{m})$ just on those vectors $\mathbf{m} \in \mathcal{Z}^d$ in the positive orthant whose norm is less than the integer n. To that end, let $S(n) \triangleq \{\mathbf{m} \in \mathcal{Z}^d : \|\mathbf{m}\| \leq n;\ \forall j, m_j > 0\}$. Note that, at the moment, we are excluding from consideration any vectors $\mathbf{m}$ with zero components. Hence, $S(n) = \emptyset$ for $n < \sqrt{d}$. Further, we'll impose an order on the elements of $S(n)$, numbering them from 0 to $|S(n)|-1$. Any ordering will do, but for the sake of specificity, let's impose dictionary order: we compare two vectors $\mathbf{m}_1$ and $\mathbf{m}_2$ componentwise from left to right until

$S(4)$ and the $C_a^{(4)}$ Code

$S(4)$	$C_a^{(4)}(\mathbf{m})$
$(1,1)$	000
$(1,2)$	001
$(1,3)$	010
$(2,1)$	011
$(2,2)$	100
$(2,3)$	101
$(3,1)$	110
$(3,2)$	111

Figure 3.5: *The above table shows the first few elements of $S(n)$ in increasing dictionary order and their $C_a^{(n)}$ encoding for $n = 4$, $d = 2$.*

the first component j in which they disagree. Then $\mathbf{m}_1$ precedes $\mathbf{m}_2$ just in case $\mathbf{m}_1(j) < \mathbf{m}_2(j)$. $C_a^{(n)}$ uses the same number of bits $k = \lceil \log |S(n)| \rceil$ (where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x) to describe all $\mathbf{m} \in S(n)$, assigning to $\mathbf{m}$ the k-bit binary representation of its index in the set, using leading zeros if necessary. Obviously, the associated code-length function is $L_a^{(n)}(\mathbf{m}) = k$. Figure 3.5 illustrates the first few elements of $S(n)$, in increasing dictionary order, and their $C_a^{(n)}$ encoding, for $n = 4$, $d = 2$.

We then leverage $C_a^{(n)}$ to create a code on the entire positive orthant by recognizing that any $\mathbf{m}$ in the positive orthant belongs to $S(n)$ for some n. (In fact, it belongs to $S(n)$ for infinitely many n: those satisfying $n > ||\mathbf{m}||$.) We can therefore describe $\mathbf{m}$ using a two-part code: first we describe some $n > ||\mathbf{m}||$ using the natural code $C_Z(n)$ and then we describe $\mathbf{m} \in S(n)$ using $C_a^{(n)}(\mathbf{m})$. We'll call this code $C_b(\mathbf{m})$.

C_b has not yet been precisely defined, since we haven't specified which of the infinitely many n values compatible with $\mathbf{m}$ to encode. It might seem as though we should choose the smallest compatible value $n = \lceil ||\mathbf{m}|| \rceil$ in order to obtain the smallest code-length for $\mathbf{m}$, but this turns out to be incorrect, because it requires about $\log ||\mathbf{m}||$ bits to describe this number, and it will turn out to take about $d \log ||\mathbf{m}||$ bits to describe $\mathbf{m} \in S(||\mathbf{m}||)$, leading to a total code-length of about $(d+1) \log ||\mathbf{m}||$, which is not asymptotically optimal. Had we defined $S(n)$ to be the set of vectors in the positive orthant whose norms were within a tolerance of 1/2 of n, then this would have worked, but, as it stands, the choice of $n = \lceil \|\mathbf{m}\| \rceil$ describes $\|\mathbf{m}\|$ more accurately than we really need.

Interestingly, we only need to describe $\|\mathbf{m}\|$ to within a factor of 2. That is, we need only describe a number n such that $\|\mathbf{m}\| \leq n \leq 2\|\mathbf{m}\|$. This means that we can choose n to be a power of 2, i.e. $n = 2^q$ for some $q \in \mathcal{Z}$, which in turn means that we can describe n merely by describing q, and that takes only about $\log\log\|\mathbf{m}\|$ bits, which is asymptotically negligible. In so doing, we may be describing a set $S(n)$ that is up to 2^d times as large as it ideally would need to be, but this potential enlarging of the set size induces a cost of at most d bits over the ideal code-length, which is also asymptotically negligible.

As mentioned above, we seek a number $n = 2^q$, $q \in \mathcal{Z}$, such that $\|\mathbf{m}\| \leq n = 2^q \leq 2\|\mathbf{m}\|$. Taking logs, we have $\log\|\mathbf{m}\| \leq q \leq \log\|\mathbf{m}\| + 1$, which means that $q = \lceil \log\|\mathbf{m}\| \rceil$. If we define $q(\mathbf{m}) \stackrel{\triangle}{=} \lceil \log\|\mathbf{m}\| \rceil$ and $n(\mathbf{m}) \stackrel{\triangle}{=} 2^{q(\mathbf{m})}$, then we can define $C_b(\mathbf{m})$ as

$$C_b(\mathbf{m}) \stackrel{\triangle}{=} C_Z(q(\mathbf{m}))C_a^{(n(\mathbf{m}))}(\mathbf{m}) \tag{3.61}$$

whose associated code-length function is

$$L_b(m) = L_Z(\lceil \log\|\mathbf{m}\| \rceil) + \log|S(2^{\lceil \log\|\mathbf{m}\| \rceil})| \tag{3.62}$$

Let $\mathbf{1}$ be the vector of all ones. It is easy to extend C_b from a code on the positive orthant to a code on the non-negative orthant by defining the new code

$$C_c(\mathbf{m}) = C_b(\mathbf{m} + \mathbf{1}) \tag{3.63}$$

with associated code-length function

$$L_c(\mathbf{m}) = L_Z(\lceil \log\|\mathbf{m} + \mathbf{1}\| \rceil) + \log|S(2^{\lceil \log\|\mathbf{m}+\mathbf{1}\| \rceil})| \tag{3.64}$$

It is also easy to extend this code to the entire space $\mathcal{Z}^d$ by affixing a d-bit sign string to the code. Let $\mathbf{s}(\mathbf{m}) \in \{0,1\}^d$ be the d-bit string satisfying

$$s_j = \begin{cases} 0 & m_j > 0 \\ 1 & m_j \leq 0 \end{cases} \tag{3.65}$$

In effect, s_j tells us whether the j^{th} component of $\mathbf{m}$ should be interpreted as positive or negative. Now let $\mathbf{m} \in \mathcal{Z}^d$ and let $|\mathbf{m}|$ denote the vector in the non-negative orthant whose j^{th} component is $|m_j|$. With this addition, the final code is given by

$$C_2(\mathbf{m}) = \mathbf{s}(\mathbf{m})C_c(|\mathbf{m}|) \tag{3.66}$$

with associated code-length function

$$\begin{aligned} L_2(\mathbf{m}) &= L_c(|\mathbf{m}|) + |\mathbf{s}(\mathbf{m})| && (3.67) \\ &= L_Z(\lceil \log \|\ |\mathbf{m}| + \mathbf{1}\| \rceil) + \log |S(2^{\lceil \log \|\ |\mathbf{m}| + \mathbf{1}\| \rceil})| + d && (3.68) \end{aligned}$$

We wish to understand the asymptotic properties of L_2 as expressed in Equation (3.68) above. The first and third terms are asymptotically negligible, since $L_Z(\lceil \log \||\mathbf{m}| + \mathbf{1}\| \rceil) = O(\log\log \|\mathbf{m}\|)$, and $d = O(1)$. To get a handle on the middle term, notice that the set $S(n)$ is completely contained in the cube of side n in the positive orthant of $\mathcal{Z}^d$. To be precise, let $T(n) \triangleq \{\mathbf{m} \in \mathcal{Z}^d : \forall j, 0 < m_j \leq n\}$. Then clearly

$$\begin{aligned} S(n) \subset T(n) \text{ which implies } |S(n)| \leq |T(n)| = n^d && (3.69) \\ \text{which implies } \log |S(n)| \leq d \log n \quad \text{leading to} && (3.70) \\ L_2(\mathbf{m}) = d \log \|\ |\mathbf{m}| + \mathbf{1}\| + O(\log\log \|\mathbf{m}\|) && (3.71) \end{aligned}$$

where the last line above follows from (3.70) and (3.68).

Theorem 3.4. *The code C_2 on $\mathcal{Z}^d$ is asymptotically optimal.*

Proof: The proof follows identical lines to the proof of Theorem 3.3. We must show that

$$\lim_{r \to \infty} \max_{\|\mathbf{m}\| \leq r} \frac{L_2(\mathbf{m})}{d \log r} = 1 \tag{3.72}$$

First, we note that if $\|\mathbf{m}\| \leq r$ then

$$\| \, |\mathbf{m}| + \mathbf{1}\| \leq r + \sqrt{d} \tag{3.73}$$

which implies

$$\log \| \, |\mathbf{m}| + \mathbf{1}\| \leq \log(r + \sqrt{d}) \tag{3.74}$$

leading to

$$L_2(\mathbf{m}) \leq d \log(r + \sqrt{d}) + O(\log \log r) \tag{3.75}$$

where the line above follows from $L_2(\mathbf{m}) = d \log \| \, |\mathbf{m}| + \mathbf{1}\| + O(\log \log \|\mathbf{m}\|)$, derived at Equation (3.71).

Continuing, since Equation (3.75) holds for all $\mathbf{m}$ such that $\|\mathbf{m}\| \leq r$, it must hold for the max over this set, yielding

$$\max_{\|\mathbf{m}\| \leq r} L_2(\mathbf{m}) \leq d \log(r + \sqrt{d}) + O(\log \log r) \tag{3.76}$$

which implies

$$\max_{\|\mathbf{m}\| \leq r} \frac{L_2(\mathbf{m})}{d \log r} \leq \frac{d \log(r + \sqrt{d}) + O(\log \log r)}{d \log r} \tag{3.77}$$

leading to

$$\limsup_{r \to \infty} \max_{\|\mathbf{m}\| \leq r} \frac{L_2(\mathbf{m})}{d \log r} \leq \limsup_{r \to \infty} \frac{d \log(r + \sqrt{d}) + O(\log \log r)}{d \log r} = 1 \tag{3.78}$$

from which we conclude

$$\lim_{r \to \infty} \max_{\|\mathbf{m}\| \leq r} \frac{L_2(\mathbf{m})}{d \log r} = 1 \tag{3.79}$$

since we know that, for all L,

$$\liminf_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L(\mathbf{m})}{d\log r} \ge 1 \tag{3.80}$$

from Theorem 3.2 on page 21.

3.7 The l^1 Code On $\mathcal{Z}^d$

This section describes a code $C_1(\mathbf{m})$, whose associated code-length function depends upon the l^1 norm of $\mathbf{m}$, so we'll call it *the l^1code.* Its development parallels exactly the construction of the l^2 code of the previous section.

As before, we begin by describing a code $C_a^{(n)}(\mathbf{m})$, just on those vectors $\mathbf{m} \in \mathcal{Z}^d$ in the positive orthant whose l^1 norm is less than the integer n, and we let $S(n) \triangleq \{\mathbf{m} \in \mathcal{Z}^d : \|\mathbf{m}\|_1 \le n;\ \forall j, m_j > 0\}$, where $\|\mathbf{m}\|_1 \triangleq \sum_j |m_j|$. As before, note that, at the moment, we are excluding from consideration any vectors $\mathbf{m}$ with zero components, and, hence, that $S(n) = \emptyset$ for $n < d$. As before, we impose dictionary order on the elements of $S(n)$, numbering them from 0 to $|S(n)| - 1$. $C_a^{(n)}$ uses the same number of bits, $k = \lceil \log |S(n)| \rceil$, to describe all $\mathbf{m} \in S(n)$, assigning to $\mathbf{m}$ the k-bit binary representation of its index in the set, using leading zeros if necessary. Obviously, the associated code-length function is $L_a^{(n)}(\mathbf{m}) = k$. Figure 3.6 illustrates $S(n)$, in increasing dictionary order, and its $C_a^{(n)}$ encoding, for $n = 4$, $d = 2$.

As before, we then leverage $C_a^{(n)}$ to create a code on the entire positive orthant via a two-part code that first describes n satisfying $\|\mathbf{m}\|_1 \le n \le 2\|\mathbf{m}\|_1$ and then describes $\mathbf{m}$ using $C_a^{(n)}$, where $n = 2^q$ and $q = \lceil \log \|\mathbf{m}\|_1 \rceil$. If we define $q(\mathbf{m}) \triangleq \lceil \log \|\mathbf{m}\|_1 \rceil$ and $n(\mathbf{m}) \triangleq 2^{q(\mathbf{m})}$), then we can define $C_b(\mathbf{m})$ as

$$C_b(\mathbf{m}) \triangleq C_Z(q(\mathbf{m}))C_a^{(n(\mathbf{m}))}(\mathbf{m}) \tag{3.81}$$

whose associated code-length function is

$$L_b(m) = L_Z(\lceil \log \|\mathbf{m}\|_1 \rceil) + \log |S(2^{\lceil \log \|\mathbf{m}\|_1 \rceil})| \tag{3.82}$$

$S(4)$ and the $C_a^{(4)}$ Code

$S(4)$	$C_a^{(4)}(\mathbf{m})$
$(1,1)$	000
$(1,2)$	001
$(1,3)$	010
$(2,1)$	011
$(2,2)$	100
$(3,1)$	110

Figure 3.6: *The above table shows the first few elements of $S(n)$ in increasing dictionary order and their $C_a^{(n)}$ encoding for $n = 4$, $d = 2$.*

We then extend C_b to a code on the non-negative orthant by defining the new code

$$C_c(\mathbf{m}) = C_b(\mathbf{m} + \mathbf{1}) \tag{3.83}$$

with associated code-length function

$$L_c(\mathbf{m}) = L_Z(\lceil \log \|\mathbf{m} + \mathbf{1}\|_1 \rceil) + \log |S(2^{\lceil \log \|\mathbf{m}+\mathbf{1}\|_1 \rceil})| \tag{3.84}$$

and then further extend the code to the entire space $\mathcal{Z}^d$ by affixing a d-bit sign string to the code. As before, let $\mathbf{s}(\mathbf{m}) \in \{0,1\}^d$ be the d-bit string satisfying

$$s_j = \begin{cases} 0 & m_j > 0 \\ 1 & m_j \leq 0 \end{cases} \tag{3.85}$$

and let $|\mathbf{m}|$ denote the vector in the non-negative orthant whose j^{th} component is $|m_j|$. The final code is then given by

$$C_1(\mathbf{m}) = \mathbf{s}(\mathbf{m}) C_c(|\mathbf{m}|) \tag{3.86}$$

with associated code-length function

$$L_1(\mathbf{m}) \quad = \quad L_c(|\mathbf{m}|) + |\mathbf{s}(\mathbf{m})| \tag{3.87}$$

$$= \quad L_Z(\lceil \log \| \, |\mathbf{m}| + \mathbf{1} \|_1 \rceil) + \log |S(2^{\lceil \log \| \, |\mathbf{m}| + \mathbf{1} \|_1 \rceil})| + d \qquad (3.88)$$

In order to understand the asymptotic properties of L_1 as expressed in Equation (3.88) above, we note, as before, that the first and third terms are asymptotically negligible, since $L_Z(\lceil \log \|\mathbf{m} + \mathbf{1}\|_1 \rceil) = O(\log \log \|\mathbf{m}\|_1)$, and $d = O(1)$. For the middle term, we notice, as before, that $S(n)$ is completely contained in the cube of side n in the positive orthant of $\mathcal{Z}^d$. To be precise, let $T(n) \overset{\triangle}{=} \{\mathbf{m} \in \mathcal{Z}^d : \forall j, 0 < m_j \leq n\}$. Then clearly

$$S(n) \subset T(n) \text{ which implies } |S(n)| \leq |T(n)| = n^d \qquad (3.89)$$

$$\text{which implies } \log |S(n)| \leq d \log n \quad \text{leading to} \qquad (3.90)$$

$$L_1(\mathbf{m}) = d \log \| \, |\mathbf{m}| + \mathbf{1} \|_1 + O(\log \log \|\mathbf{m}\|_1) \qquad (3.91)$$

where the last line above follows from (3.90) and (3.88).

Theorem 3.5. *The code C_1 on $\mathcal{Z}^d$ is asymptotically optimal.*

Proof: The proof follows identical lines to the proof of Theorem 3.3. We must show that

$$\lim_{r \to \infty} \max_{\|\mathbf{m}\| \leq r} \frac{L_1(\mathbf{m})}{d \log r} = 1 \qquad (3.92)$$

First, we note that if $\|\mathbf{m}\| \leq r$ then

$$\| \, |\mathbf{m}| + \mathbf{1} \|_1 \leq r\sqrt{d} + d \qquad (3.93)$$

which implies

$$\log \| \, |\mathbf{m}| + \mathbf{1} \|_1 \leq \log(r\sqrt{d} + d) \qquad (3.94)$$

leading to

$$L_1(\mathbf{m}) \leq d \log(r\sqrt{d} + d) + O(\log \log r) \qquad (3.95)$$

where the line above follows from $L_1(\mathbf{m}) = d \log \|\ |\mathbf{m}| + \mathbf{1}\|_1 + O(\log\log\|\mathbf{m}\|_1)$, derived at Equation (3.91).

Continuing, since Equation (3.95) holds for all $\mathbf{m}$ such that $\|\mathbf{m}\| \le r$, it must hold for the max over this set, yielding

$$\max_{\|\mathbf{m}\|\le r} L_1(\mathbf{m}) \le d \log(r\sqrt{d} + d) + O(\log\log r) \tag{3.96}$$

which implies

$$\max_{\|\mathbf{m}\|\le r} \frac{L_1(\mathbf{m})}{d \log r} \le \frac{d \log(r\sqrt{d} + d) + O(\log\log r)}{d \log r} \tag{3.97}$$

Noting that $\log r\sqrt{d}$ above grows asymptotically in r like $\log r$, since $\log r\sqrt{d} = \log r + 1/2 \log d$, this leads to

$$\limsup_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L_1(\mathbf{m})}{d \log r} \le \limsup_{r\to\infty} \frac{d \log(r\sqrt{d} + d) + O(\log\log r)}{d \log r} = 1 \tag{3.98}$$

from which we conclude

$$\lim_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L_1(\mathbf{m})}{d \log r} = 1 \tag{3.99}$$

since we know that, for all L,

$$\liminf_{r\to\infty} \max_{\|\mathbf{m}\|\le r} \frac{L(\mathbf{m})}{d \log r} \ge 1 \tag{3.100}$$

from Theorem 3.2 on page 21.

3.8 δ-Quantized Codes

We now turn to the problem of describing an arbitrary $\mathbf{x} \in \mathcal{R}^d$ to some specified precision $\delta/2$. Since $\mathcal{R}^d$ is uncountable, no prefix-free code can exist on it. However, by quantizing it into δ-sized cubes whose centers we take as reproduction points, we create a countable subset of $\mathcal{R}^d$ that is δ-close to any arbitrary $\mathbf{x} \in \mathcal{R}^d$ we care to name. Since this quantized subset is countable, a prefix-free code exists on it, and, in fact, we can easily extend any of the prefix-free codes on $\mathcal{Z}^d$ constructed in the previous sections to a code on the reproduction points.

Definition 3.8. Let $\delta > 0$ be given. Then the set of *reproduction points* in $\mathcal{R}^d$ induced by quantization width δ is

$$Q_\delta^d \triangleq \{\delta\mathbf{m} : \mathbf{m} \in \mathcal{Z}^d\} \tag{3.101}$$

Further, let $\mathbf{x} \in \mathcal{R}^d$ be an arbitrary vector. Then the reproduction point associated with the quantization cube into which $\mathbf{x}$ falls is

$$Q_\delta^d(\mathbf{x}) \triangleq \hat{\mathbf{x}}, \text{ where } \hat{x}_j = \delta\lfloor x_j/\delta + 1/2\rfloor \tag{3.102}$$

Remark 3.4. Note that if $\hat{\mathbf{x}} \in Q_\delta^d$, then the index of the quantization cube for which $\hat{\mathbf{x}}$ is the reproduction point is the integer vector $\mathbf{m} = \hat{\mathbf{x}}/\delta$.

Since every reproduction point $\hat{\mathbf{x}} \in Q_\delta^d$ is indexed by an integer vector $\mathbf{m} = \hat{\mathbf{x}}/\delta$, any code C on $\mathcal{Z}^d$ can be used to induce a code C_δ on the reproduction points, leading to the following definition:

Definition 3.9. A δ-*quantized* code C_δ is a code on Q_δ^d. If C is a code on $\mathcal{Z}^d$, then *the δ-quantized code on Q_δ^d induced by C* is

$$C_\delta(\hat{\mathbf{x}}) \triangleq C(\hat{\mathbf{x}}/\delta) \tag{3.103}$$

and, conversely, if C_δ is a δ-quantized code on Q_δ^d, then the *integer code on $\mathcal{Z}^d$ induced by C_δ* is

$$C(\mathbf{m}) \triangleq C_\delta(\delta\mathbf{m}) \tag{3.104}$$

Definition 3.10. A δ-quantized code-length function, L_δ on Q_δ^d, is *asymptotically optimal* in case

$$\lim_{r\to\infty} \max_{\hat{\mathbf{x}}\in Q_\delta^d:\ \|\hat{\mathbf{x}}\|\le r} \frac{L_\delta(\hat{\mathbf{x}})}{d\log r} = 1 \tag{3.105}$$

The justification for the above definition is the same as for the integer case. Asymptotically, $d\log\|\hat{\mathbf{x}}\|$ is the best one can hope to achieve in encoding $\hat{\mathbf{x}}$. The proof follows identical lines as in the integer case and will be omitted. When dealing with δ-quantized codes, there is no essential difference than when dealing with integers: we have a countable set of reproduction points, evenly spaced on $\mathcal{R}^d$. The fact that they happen to be spaced δ apart instead of 1 apart does not substantively change the coding properties.

A δ-quantized code-length function L_δ is defined only on Q_δ^d, rather than on $\mathcal{R}^d$. In Chapter 4, it will become important to find appropriate extensions of such code-length functions to $\mathcal{R}^d$. That is, given L_δ on Q_δ^d, we will want to find an appropriate relaxed function l_δ on $\mathcal{R}^d$, so that descent methods can be used to solve the minimization. To that end, we make the following definition.

Definition 3.11. Let $l_\delta(\mathbf{x})$ be a function from $\mathcal{R}^d$ to $\mathcal{R}$. l_δ is a *δ-approximate code-length function* in case there exists a code C_δ on Q_δ^d with associated code-length function L_δ, and a constant α, satisfying

$$|l_\delta(\hat{\mathbf{x}}) - L_\delta(\hat{\mathbf{x}})| \le \alpha, \quad \forall\hat{\mathbf{x}} \in Q_\delta^d \tag{3.106}$$

and

$$\min_{\hat{\mathbf{x}}\in Q_\delta^d:\ \|\hat{\mathbf{x}}-\mathbf{x}\|_\infty\le\delta} L_\delta(\hat{\mathbf{x}}) \le l_\delta(\mathbf{x}) \le \max_{\hat{\mathbf{x}}\in Q_\delta^d:\ \|\hat{\mathbf{x}}-\mathbf{x}\|_\infty\le\delta} L_\delta(\hat{\mathbf{x}}) \tag{3.107}$$

Formally, we extend the notion of asymptotic optimality to include δ-approximate code-length functions.

Definition 3.12. A δ-approximate code-length function, l_δ on $\mathcal{R}^d$, is *asymptotically optimal* in case

$$\lim_{r\to\infty} \max_{\|\mathbf{x}\|\le r} \frac{l_\delta(\mathbf{x})}{d\log r} = 1 \tag{3.108}$$

The δ-quantized code on Q_δ^d induced by the natural code C_Z on $\mathcal{Z}^d$ is *the natural code on* Q_δ^d, which we'll denote $C_{\delta,Z}(\hat{\mathbf{x}})$. Its code-length function $L_{\delta,Z}(\hat{\mathbf{x}})$, according to equations (3.45), (3.46) on page 23, and remark 3.4 on page 35, is given by

$$\begin{aligned} L_{\delta,Z}(\hat{\mathbf{x}}) &= \sum_{j=1}^{d} L_Z(\hat{x}_j/\delta) && (3.109)\\ &= \sum_{j=1}^{d} \lfloor \log(|\hat{x}_j|/\delta+1)\rfloor + 2\lfloor \log(\lfloor \log(|\hat{x}_j|/\delta+1)\rfloor+1)\rfloor + 2 && (3.110)\end{aligned}$$

That $L_{\delta,Z}(\hat{\mathbf{x}})$ is asymptotically optimal according to Definition 3.12 follows directly from the asymptotic optimality of L_Z on $\mathcal{Z}^d$.

Similarly, the δ-quantized codes on Q_δ^d induced by the l^2 and l^1 codes described in sections 3.6 and 3.7, have respectively the code-length functions

$$\begin{aligned} L_{\delta,2}(\hat{\mathbf{x}}) &= (\lceil \log \| \, |\hat{\mathbf{x}}|/\delta + \mathbf{1}\|\rceil) + \log|S(2^{\lceil \log \| \, |\hat{\mathbf{x}}|/\delta+\mathbf{1}\|\rceil})| + d \quad \text{and} && (3.111)\\ L_{\delta,1}(\hat{\mathbf{x}}) &= (\lceil \log \| \, |\hat{\mathbf{x}}|/\delta + \mathbf{1}\|_1\rceil) + \log|S(2^{\lceil \log \| \, |\hat{\mathbf{x}}|/\delta+\mathbf{1}\|_1\rceil})| + d && (3.112)\end{aligned}$$

according to equations (3.68) on page 29 and (3.88) on page 33, as well as remark 3.4 on page 35. That these code-length functions are asymptotically optimal according to Definition 3.12 follows directly from the asymptotic optimality of L_2 and L_1 on $\mathcal{Z}^d$.

The development of δ-quantized codes assumed that $\delta > 0$ was an arbitrary real number, specified in advance, and available, "for free", as it were, to both the encoding and decoding process. By specifying δ as part of the coding process, we can obtain a code on a dense countable subset of $\mathcal{R}^d$, allowing us to approximately encode any real vector to any desired degree of precision. We do this using the indexed coding scheme described in Section 3.5. We begin with a definition.

Definition 3.13. Let Q^d denote the set of *diadic* reproduction points:

$$Q^d \triangleq \{2^{-q}\mathbf{m} : q \in \mathcal{Z}, \mathbf{m} \in \mathcal{Z}^d\} \tag{3.113}$$

$$= \bigcup_{\delta:\ \delta=2^{-q},\ q\in\mathcal{Z}} Q^d_\delta \tag{3.114}$$

Q^d is countable and contains reproduction points to any arbitrary finite precision. That is, given an arbitrary vector $\mathbf{x} \in \mathcal{R}^d$, Q^d contains points arbitrarily close to it.

Now we proceed with the construction of an indexed code on Q^d. Let C be a fixed code on $\mathcal{Z}^d$, and, for each $\delta > 0$, let C_δ denote the δ-quantized code on Q^d_δ induced by C as described in Definition 3.9. Let $\mathcal{D} \triangleq \{C_\delta : \delta = 2^{-q}, q \in \mathcal{Z}\}$ be the countable subset of such codes corresponding to quantization widths $\delta = 2^{-q}$ for some integer q, and, for all $C_\delta \in \mathcal{D}$, let $C_0(C_\delta) \triangleq C_Z(q)$. We observe that the union of the domains of the codes in $\mathcal{D}$ is Q^d. Let C_I be the resulting indexed code on Q^d constructed from $\mathcal{D}$ and C_0.

If $\hat{\mathbf{x}} \in Q^d$, then it has more than one candidate representation of the form $\hat{\mathbf{x}} = 2^{-q}\mathbf{m}$. In fact, it has infinitely many, for if $\hat{\mathbf{x}} = 2^{-q}\mathbf{m}$, then also $\hat{\mathbf{x}} = 2^{-q-1}(2\mathbf{m})$. The indexed code chooses the representation with the shortest code-length, so, if $\hat{\mathbf{x}} \in Q^d$, its codeword will be

$$C_I(\hat{\mathbf{x}}) = C_Z(q^*)C(\mathbf{m}^*) \tag{3.115}$$

where q^* and $\mathbf{m}^*$ satisfy

$$(q^*, \mathbf{m}^*) = \arg \min_{q,\mathbf{m}:2^{-q}\mathbf{m}=\hat{\mathbf{x}}} L_Z(q) + L(\mathbf{m}) \tag{3.116}$$

By construction, the code-length function of C_I is

$$L_I(\hat{\mathbf{x}}) = \arg \min_{q,\mathbf{m}:2^{-q}\mathbf{m}=\hat{\mathbf{x}}} L_Z(q) + L(\mathbf{m}) \tag{3.117}$$

Definition 3.14. When the code C above is the natural code C_Z on $\mathcal{Z}^d$, then we'll call the code C_I constructed from it the *natural code* on Q^d, writing $C_Z(\hat{\mathbf{x}})$, and writing $L_Z(\hat{\mathbf{x}})$ to indicate its associated code-length function.

Remark 3.5. If $\hat{\mathbf{x}}$ is a reproduction point in Q^d, then it codeword under the natural code is

$$C_Z(\hat{\mathbf{x}}) = C_Z(q^*)C_Z(\mathbf{m}^*) \tag{3.118}$$

where q^* and $\mathbf{m}^*$ satisfy

$$(q^*, \mathbf{m}^*) = \arg \min_{q,\mathbf{m}:2^{-q}\mathbf{m}=\hat{\mathbf{x}}} L_Z(q) + L_Z(\mathbf{m}) \tag{3.119}$$

In other words, the codeword for $\hat{\mathbf{x}}$ under C_Z really *is* the natural code on an integer vector $(q^*, \mathbf{m}^*) \in \mathcal{Z}^{d+1}$ that determines it, and its code-length function is

$$L_Z(\hat{\mathbf{x}}) = \arg \min_{q,\mathbf{m}:2^{-q}\mathbf{m}=\hat{\mathbf{x}}} L_Z(q) + L_Z(\mathbf{m}) \tag{3.120}$$

3.9 Codes Derived From Probability Mass Functions

The previous sections exhibited the explicit construction of some codes on $\mathcal{Z}^d$. It is also possible to derive a code from a probability distribution, and, more importantly, from a parametric family of probability distributions.

Definition 3.15. If $p(\mathbf{m})$ is a probability mass function on $\mathcal{Z}^n$, then the Kraft Inequality, Theorem 3.1 on page 14, guarantees the existence of a code C_p with code-lengths $L_p(\mathbf{m}) = \lceil -\log p(\mathbf{m}) \rceil$. C_p is referred to as a *Shannon code* associated with $p(\mathbf{m})$.

Remark 3.6. We will often write about *the* Shannon code C_p associated with $p(\mathbf{m})$. Although such a code is never unique, the code-lengths L_p are, which is all we really care about.

Now suppose $\mathcal{P} = \{p_{\mathbf{b}}(\mathbf{m})\}$ is a family of probability mass functions on $\mathcal{Z}^n$, indexed by a real-vector $\mathbf{b} \in \mathcal{R}^d$, and, for each $p_{\mathbf{b}} \in \mathcal{P}$, let $C_{\mathbf{b}}$ and $L_{\mathbf{b}}$ be the associated Shannon code and code-length function respectively. Let $\mathcal{D} = \{C_{\mathbf{b}} : \mathbf{b} \in Q^d\}$, where Q^d is the set of diadic reproduction points defined in Definition 3.13. Then $\mathcal{D}$ is a countable collection of codes indexed by the elements of Q^d, so we can build from it an indexed code $C(\mathbf{m})$ on $\mathcal{Z}^n$. Following the construction in Section 3.5, let $C_0(\mathbf{b})$ be a code on $\mathbf{b} \in Q^d$. Then the indexed code is

$$C(\mathbf{m}) \triangleq C_0(\hat{\mathbf{b}})C_{\hat{\mathbf{b}}}(\mathbf{m}) \tag{3.121}$$

where $\hat{\mathbf{b}}$ satisfies

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b} \in Q^d} L_0(\mathbf{b}) + L_{\mathbf{b}}(\mathbf{m}) \tag{3.122}$$

The preceding construction plays a crucial role in the theory of minimum description-length (MDL) estimation, which will be discussed in Chapter 4. In fact, for the given parametric family, $\mathcal{P}$, and the particular quantization set Q^d and coding scheme C_0, $\hat{\mathbf{b}}$ above is the MDL estimator, because it yields the shortest two-part description (code-length) of the data $\mathbf{m}$.

3.10 δ-Quantized Codes Derived From Probability Density Functions

Let $\mathcal{F} = \{f_{\mathbf{b}}(\mathbf{x})\}$ be a family of probability density functions on $\mathcal{R}^n$, indexed by a real-valued vector $\mathbf{b} \in \mathcal{R}^d$. We can combine the methods of the last several sections to create δ-quantized codes on $\mathcal{R}^n$ based on $\mathcal{F}$. Let $\delta > 0$ be fixed. We begin by quantizing $\mathcal{R}^n$ into δ-sized cubes whose centers are reproduction points in Q_δ^n. For a given $\mathbf{b}$, each of these cubes has an associated probability mass, which is the cube's probability under $f_{\mathbf{b}}$. The quantization therefore allows us to convert the family $\mathcal{F} = \{f_{\mathbf{b}}(\mathbf{x})\}$ of pdfs on $\mathcal{R}^n$ into a family $\mathcal{P}_\delta = \{p_{\delta,\mathbf{b}}(\hat{\mathbf{x}})\}$ of pmfs on Q_δ^n. In more detail, if $V_\delta(\hat{\mathbf{x}})$ denotes the quantization cube whose reproduction point is $\hat{\mathbf{x}} \in Q_\delta^n$, then $p_{\delta,\mathbf{b}}(\hat{\mathbf{x}})$ is given by

$$p_{\delta,\mathbf{b}}(\hat{\mathbf{x}}) \stackrel{\triangle}{=} \int_{V_\delta(\hat{\mathbf{x}})} f_{\mathbf{b}}(\mathbf{x}) d\mathbf{x} \tag{3.123}$$

Now that we have a family of pmfs, we can follow the methods of the previous section to create an indexed code C_δ on Q_δ^n. For each $p_{\delta,\mathbf{b}} \in \mathcal{P}_\delta$, let $C_{\delta,\mathbf{b}}$ and $L_{\delta,\mathbf{b}}$ be the associated Shannon code and code-length function respectively. Let $\mathcal{D} = \{C_{\delta,\mathbf{b}} : \mathbf{b} \in Q^d\}$, where Q^d is the set of diadic reproduction points defined in Definition 3.13. Then $\mathcal{D}$ is a countable collection of codes indexed by the elements of Q^d, so we can build from it an indexed code $C_\delta(\hat{\mathbf{x}})$ on Q_δ^n. Following the construction in Section 3.5 on page 25, let $C_0(\mathbf{b})$ be a code on $\mathbf{b} \in Q^d$. Then the indexed code is

$$C_\delta(\hat{\mathbf{x}}) \stackrel{\triangle}{=} C_0(\hat{\mathbf{b}}) C_{\delta,\hat{\mathbf{b}}}(\hat{\mathbf{x}}) \tag{3.124}$$

where $\hat{\mathbf{b}}$ satisfies

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b} \in Q^d} L_0(\mathbf{b}) + L_{\delta,\mathbf{b}}(\hat{\mathbf{x}}) \tag{3.125}$$

Chapter 4

Log-Penalized Linear Regression

This chapter discusses the theoretical motivation for the log penalty as well as practical methods for solving the log-penalized linear regression minimization. When one takes a complexity-based approach to estimation, as embodied by the minimum description length principle, the log penalty emerges quite naturally.

4.1 The Minimum Description-Length Principle

The minimum description-length (MDL) principle was developed by Rissanen [15], based on the pioneering work in algorithmic complexity by Kolmogorov [10], Chaitin [3], and Solomonoff [16]. The principle relates parametric estimation theory to coding theory by asserting that the estimate be associated with a minimum-length code on the data. Using ideas that parallel those of algorithmic complexity, Rissanen defines the complexity of a data set to be the code-length of the data with respect to an indexed code induced by a parametric family of probability distributions. From this perspective, *complexity* is just a synonym for code-length with respect to an identified code.

Let $\{p_{\mathbf{b}}(\mathbf{m})\}$ be a family of pmfs on $\mathcal{Z}^n$, indexed by a real vector $\mathbf{b} \in \mathcal{R}^d$. Suppose that we are given data $\mathbf{m}$ generated from the probability distribution whose index is $\mathbf{b}_0$—that is, $\mathbf{m} \sim p_{\mathbf{b}_0}(\mathbf{m})$—and that $\mathbf{b}_0$ is unknown. We would like to find a good estimate $\hat{\mathbf{b}}$ of the true but unknown underlying parameter vector $\mathbf{b}_0$.

To that end, let $C_{\mathbf{b}}$ and $L_{\mathbf{b}}$ be the Shannon code and code-length function associated with $p_{\mathbf{b}}$, let Q^d be the set of diadic reproduction points on $\mathcal{R}^d$ as defined in Definition 3.13, and let $C(\mathbf{b})$ be a code (any code, for the moment) on Q^d with associated code-length function $L(\mathbf{b})$. As illustrated in Equation (3.121) on page 40, $\mathbf{m} \in \mathcal{Z}^n$ can then be encoded with an indexed code C_I as

$$C_I(\mathbf{m}) = C(\hat{\mathbf{b}})C_{\hat{\mathbf{b}}}(\mathbf{m}) \tag{4.1}$$

where $\hat{\mathbf{b}}$ satisfies

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}\in Q^d} L(\mathbf{b}) + L_{\mathbf{b}}(\mathbf{m}) \tag{4.2}$$

$$= \arg\min_{\mathbf{b}\in Q^d} L(\mathbf{b}) + \lceil -\log p_{\mathbf{b}}(\mathbf{m})\rceil \tag{4.3}$$

In Equation (4.2) above, $\hat{\mathbf{b}}$ is the MDL estimate. Note that, for any $\mathbf{b} \in Q^d$ whatever, $\mathbf{m}$ can be described as $C(\mathbf{b})C_{\mathbf{b}}(\mathbf{m})$, but that the $\hat{\mathbf{b}}$ selected above is the one that attains the minimum code-length from among all possible choices of $\mathbf{b}$. The *complexity* of $\mathbf{m}$ is defined to be its code-length under this coding scheme, $L(\hat{\mathbf{b}}) + \lceil -\log p_{\hat{\mathbf{b}}}(\mathbf{m})\rceil$, and the complexity of $\hat{\mathbf{b}}$ is defined to be $L(\hat{\mathbf{b}})$. (Rissanen actually uses a more involved definition to define what he calls the *stochastic complexity* of $\mathbf{m}$, but our definition captures the spirit and is sufficient for our purposes.) The structure of the two-part code reflects our competing desires to both choose a $\hat{\mathbf{b}}$ that fits the data $\mathbf{m}$ well and to choose a $\hat{\mathbf{b}}$ of low complexity.

Note also that there is nothing particular about Q^d. Any dense countable subset of $\mathcal{R}^d$ would do. We focus on Q^d because it is convenient. Finally, note that $\hat{\mathbf{b}}$ depends upon the code C chosen.

Definition 4.1. Let $\mathcal{P} = \{p_{\mathbf{b}}(\mathbf{m})\}$ be a family of pmfs on $\mathcal{Z}^n$, indexed by a real vector $\mathbf{b} \in \mathcal{R}^d$, and let Q be a dense countable subset of $\mathcal{R}^d$ on which a prefix-free code C with code-length function L is defined. Then the *MDL estimator* determined by $\mathcal{P}, Q,$ and L is

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}\in Q} L(\mathbf{b}) + \lceil -\log p_{\mathbf{b}}(\mathbf{m})\rceil$$

As noted above, the MDL estimator depends upon the choice of C. The code C embodies our inherent preference for certain kinds of models. Different choices lead to different MDL estimates. Rissanen comments on this in [15]. He notes that while this may be formally equivalent to a Bayesian approach, and therefore just as ad hoc, the coding perspective naturally causes one to focus on succinct codes rather than on prior probability distributions, which can be a liberating viewpoint. From the Bayesian perspective, one may question whether the chosen prior distribution truly reflects an underlying generation mechanism for the data. However, from the coding perspective, any code that yields a succinct coding of the data is perfectly acceptable. Further, when the coding perspective is adopted, this dissertation observes that it makes sense to consider only asymptotically optimal codes. The naturalness with which this conclusion follows from the coding perspective also reinforces Rissanen's observation that the coding perspective can lead to a different choice of equivalent prior, since it leads to what we have been calling the natural code, which does not belong to any standardly recognized parametric family of probability densities.

4.2 MDL On Continuous Data

The MDL method can be extended to handle continuous rather than discrete data. Let $\mathcal{F} = \{f_{\mathbf{b}}(\mathbf{x})\}$ be a parametric family of probability density functions on $\mathcal{R}^n$ indexed by $\mathbf{b} \in \mathcal{R}^d$ and let $C(\mathbf{b})$ be a code on Q^d with code-length function $L(\mathbf{b})$. We assume that the data $\mathbf{x}$ is generated from the pdf whose index is $\mathbf{b}_0$—that is, $\mathbf{x} \sim f_{\mathbf{b}_0}(\mathbf{x})$—and we seek a good estimate $\hat{\mathbf{b}}$ for the true underlying parameter vector $\mathbf{b}_0$.

We begin by observing that the problem can be reduced to that of discrete estimation by quantizing the domain. Let $\epsilon > 0$ be given. We use the construction detailed in Section 3.10 on page 41 to convert $\mathcal{F}$ into a family $\mathcal{P} = \{p_{\epsilon,\mathbf{b}}\}$ of pmfs defined on Q_ϵ^n. We can then use the MDL method described in the previous section to find an estimate $\hat{\mathbf{b}}$ of the quantized data $Q_\epsilon^n(\mathbf{x})$ under the parametric family $\mathcal{P}$, yielding

$$\hat{\mathbf{b}}_\epsilon = \arg\min_{\mathbf{b}\in Q^d} L(\mathbf{b}) + \lceil -\log p_{\epsilon,\mathbf{b}}(Q_\epsilon^n(\mathbf{x}))\rceil \tag{4.4}$$

We can go a step further. In theory, we'd like ϵ to be arbitrarily small. For increasingly smaller ϵ, the approximations $Q_\epsilon^n(\mathbf{x}) \approx \mathbf{x}$ and $p_{\epsilon,\mathbf{b}}(\hat{\mathbf{x}}) \approx f_\mathbf{b}(\hat{\mathbf{x}})\epsilon^n$ become increasingly accurate, which means that the approximation

$$-\log p_{\epsilon,\mathbf{b}}(\hat{\mathbf{x}}) \approx -\log f_\mathbf{b}(\hat{\mathbf{x}}) - n\log\epsilon \tag{4.5}$$

becomes increasingly accurate, leading to the following chain of approximations

$$\begin{aligned}
\hat{\mathbf{b}}_\epsilon &= \arg\min_{\mathbf{b}\in Q^d} L(\mathbf{b}) + \lceil -\log p_{\epsilon,\mathbf{b}}(Q_\epsilon^n(\mathbf{x}))\rceil && (4.6)\\
&\approx \arg\min_{\mathbf{b}\in Q^d} L(\mathbf{b}) + \lceil -\log f_\mathbf{b}(\mathbf{x}) - n\log\epsilon\rceil && (4.7)\\
&\approx \arg\min_{\mathbf{b}\in Q^d} L(\mathbf{b}) + -\log f_\mathbf{b}(\mathbf{x}) - n\log\epsilon && (4.8)\\
&= \arg\min_{\mathbf{b}\in Q^d} L(\mathbf{b}) - \log f_\mathbf{b}(\mathbf{x}) && (4.9)
\end{aligned}$$

and to the following definition.

Definition 4.2. Let $\mathcal{F} = \{f_\mathbf{b}(\mathbf{x})\}$ be a family of pmfs on $\mathcal{R}^n$, indexed by a real vector $\mathbf{b} \in \mathcal{R}^d$, and let Q be a dense countable subset of $\mathcal{R}^d$ on which a prefix-free code C with code-length function L is defined. Then the *MDL estimator* determined by $\mathcal{F}, Q,$ and L is

$$\hat{\mathbf{b}}(\mathbf{x}) = \arg\min_{\mathbf{b}\in Q} L(\mathbf{b}) - \log f_\mathbf{b}(\mathbf{x})$$

4.3 MDL And Regression

In the regression setting, we are given n pairs of data $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \mathcal{R}^p$, $y_i \in \mathcal{R}$, and we assume a relationship between $\mathbf{x}$ and y governed by a deterministic function

$h_{\mathbf{b}_0} : \mathcal{R}^p \to \mathcal{R}$ belonging to a parametric family $\mathcal{H} = \{h_{\mathbf{b}}(\mathbf{x})\}$ indexed by $\mathbf{b} \in \mathcal{R}^d$ and then corrupted by zero-mean additive Gaussian noise $\mathbf{z}$ with known variance σ_Z^2. That is,

$$y_i = h_{\mathbf{b}_0}(\mathbf{x}_i) + z_i, \quad z_i \overset{iid}{\sim} N(0, \sigma_Z^2) \tag{4.10}$$

Let $\mathbf{y} \in \mathcal{R}^n$ be the vector whose i^{th} component is y_i. Let $\mathbf{X}$ be the $n \times p$ matrix whose i^{th} row is $\mathbf{x}_i$. Let $h_{\mathbf{b}}(\mathbf{X})$ denote the vector whose i^{th} component is $h_{\mathbf{b}}(\mathbf{x}_i)$. Then an alternate formulation of the above, assuming $\mathbf{X}$ to be fixed and given, is that $\mathbf{y}$ is a random variable drawn according to a Gaussian distribution with mean $h_{\mathbf{b}_0}(\mathbf{X})$ and variance $\sigma_Z^2 I_n$. If we let $\mathcal{F} = \{f_{\mathbf{b}}(\mathbf{y})\}$ be the parametric family of such distributions, indexed by $\mathbf{b} \in \mathcal{R}^p$, and given by

$$f_{\mathbf{b}}(\mathbf{y}) \triangleq \left(\frac{1}{\sqrt{2\pi\sigma_Z^2}} \right)^n e^{-\frac{1}{2\sigma_Z^2}\|\mathbf{y} - h_{\mathbf{b}}(\mathbf{X})\|^2} \tag{4.11}$$

then we can apply the methods of the previous section to obtain an MDL estimate $\hat{\mathbf{b}}$ of the true underlying parameter vector $\mathbf{b}_0$.

Remark 4.1. It will be convenient in the following derivation, and in many of the subsequent derivations, to use the natural logarithm, rather than the log base 2. When the natural logarithm is used, the code-length function L must be thought of as expressing code-lengths in nats, rather than in bits, so that the units remain comparable. Formally, a nat is $\log_2 e$ bits. So if $\tilde{L}(\mathbf{b})$ is a code-length function expressed in bits, then $L(\mathbf{b}) = \tilde{L}(\mathbf{b}) / \log_2 e$ is the equivalent code-length function in nats.

Proceeding with the derivation of the MDL estimate:

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b} \in Q^d} L(\mathbf{b}) - \log f_{\mathbf{b}}(\mathbf{y}) \tag{4.12}$$

$$= \arg\min_{\mathbf{b} \in Q^d} L(\mathbf{b}) + n \log \sqrt{2\pi\sigma_Z^2} + \frac{1}{2\sigma_Z^2}\|\mathbf{y} - h_{\mathbf{b}}(\mathbf{X})\|^2 \tag{4.13}$$

$$= \arg\min_{\mathbf{b} \in Q^d} L(\mathbf{b}) + \frac{1}{2\sigma_Z^2}\|\mathbf{y} - h_{\mathbf{b}}(\mathbf{X})\|^2 \tag{4.14}$$

$$= \arg\min_{\mathbf{b}\in Q^d} \|\mathbf{y} - h_{\mathbf{b}}(\mathbf{X})\|^2 + 2\sigma_Z^2 L(\mathbf{b}) \tag{4.15}$$

The linear regression form of MDL estimation occurs when the family $\mathcal{H} = \{h_{\mathbf{b}}(\mathbf{x})\}$ is the family of linear functions on $\mathbf{x}$ given by $h_{\mathbf{b}}(\mathbf{x}) = \mathbf{b}^T\mathbf{x}$, yielding

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}\in Q^d} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 L(\mathbf{b}) \tag{4.16}$$

At this stage, there are a couple of impediments to solving (4.15) or its linear form (4.16) to actually yield an estimate for a particular data set $(\mathbf{X}, \mathbf{y})$. One problem is philosophical; the other is practical. The philosophical problem is that we don't necessarily know what constitutes a good code-length function L to use in the above minimization. The practical problem is that, given a particular code-length function L there may not exist efficient computational methods to solve the minimization.

Regarding the first problem, this dissertation argues that the natural code-length function on Q^d is an obvious choice for L, both because it is natural and because it is asymptotically optimal. The philosophical reasons behind this choice are discussed in greater length in Section 4.4.

Regarding the problem of computational efficiency, various relaxations can be made to yield a computationally tractable solution in the linear case. While the natural code-length function represents a theoretically sound choice regarding the MDL solution to the general regression problem (4.15), this dissertation focuses only on tractable methods for the linear case (4.16).

4.4 Motivation For Asymptotically Optimal Codes

Using the MDL method is nearly equivalent to performing Bayesian maximum a posteriori (MAP) estimation, although there are some differences induced by the integer constraints of coding. The MDL method uses code-lengths to assess whether one candidate model instance is less complex than another, and therefore inherently preferable, while the Bayesian method uses a prior probability distribution to assess whether one candidate model instance is more probable than another, and therefore inherently preferable. In the discrete case, there is a direct correspondence between probability distributions and code-length functions, and the equivalence between MDL and Bayesian MAP is quite tight. Moreover, MAP estimation with a Bayesian prior is, in turn, mathematically equivalent to penalized regression, as demonstrated in Section 5.2 on page 75. All three methods require the pulling out of thin air, as it were, of a function—be it a code-length function, a probability distribution, or a penalty function—whose selection determines a preference for some model instances over some others, with seemingly no rationale for so doing. Why use this code-length function, this probability distribution, this penalty, and not some other?

The situation is reminiscent of that in algorithmic complexity, in which the complexity of a bit string depends upon the choice of general-purpose computer used to describe it. There is seemingly no objective way to compare the complexity of two objects without first making a subjective selection of the machinery used to describe them. In algorithmic complexity, we can take heart from a kind of weak universality: any two specific general-purpose machines differ by at most a constant in their complexity assignments [4, 18]. Unfortunately, this constant can be arbitrarily large, depending on the two machines. In statistics, we can take heart from a similar kind of universality: as the amount of data n goes to infinity, the regression method used, be it MDL, Bayesian, or penalized, converges to the correct estimate, regardless of the code-length function, prior probability distribution, or penalty function chosen. As in the case of algorithmic complexity, however, the amount of data N required to achieve a certain level of estimation accuracy can be arbitrarily large, depending on the choice made.

Although MDL, Bayesian MAP, and penalized regression are roughly equivalent mathematically, they offer different perspectives on the regression problem, which may lead to different insights. Taking the MDL approach and thinking about the problem in terms of code-lengths, it is not too difficult to see why asymptotically optimal codes might be preferable. In a well-defined sense, an asymptotically optimal code achieves in the limit the shortest possible code-lengths. This is not so much a matter of being efficient as it is a matter of allocating short codewords judiciously. In fact, asymptotically optimal codes need not be efficient and efficient codes need not be asymptotically optimal (loosely speaking, a code is *efficient* if no other code exists that assigns as short or shorter code-lengths to all objects). While all codes must make arbitrary decisions about which objects get shorter codewords than others, an asymptotically optimal code is not allowed to lavish exceedingly short code-lengths on a particular subset of objects to the detriment of achieving asymptotic optimality. There must always be enough short codes to go around such that the worst case code-length length is optimal to first order. Put another way, if a code puts so much emphasis on its short codewords that the rest have to become much longer, then this code is biasing the estimation too much.

In the absence of any a priori knowledge about the true underlying parameter vector $\mathbf{b}_0$, it seems wise to choose a robust code that distributes the code-length pain as evenly as possible over the domain. Such codes are in some sense universal, because they do as well in the limit as any code possibly could. Conversely, to use instead a code that forsakes asymptotic optimality in order to favor a particular subset of the parameter domain with particularly short code-lengths would be to implicitly assert that one knew something a priori about where in the domain the true underlying parameter vector $\mathbf{b}_0$ fell. Unless such is known to be the case, it is imprudent to use such a code. This fact is intuitively understood by those applying Bayesian methods. No one would seriously propose a specific prior, such as i.i.d. standard normal, on $\mathbf{b}$, since it would hopelessly bias the regression toward solutions with $||\hat{\mathbf{b}}||^2 = p$. Instead, if an i.i.d. zero-mean Gaussian prior were desired, the entire family of zero-mean Gaussian distributions on $\mathbf{b}$ would be considered, with the unknown variance $\sigma_{\mathbf{b}}$ treated as a hyper-parameter whose value must be estimated. But this method effectively leads to an asymptotically optimal code, as demonstrated in sections 5.2 and 5.3 beginning on page 75.

4.5 The Log Penalty

Having motivated above the use of an asymptotically optimal code-length function for MDL regression, there are several we might use, for example, the l^2 or l^1 codes, or the natural code. Using the natural code L_Z on Q^d, as the code-length function in

Equation (4.15) leads to what this dissertation calls log-penalized regression. While the log penalty could be used in any kind of penalized regression, this dissertation focuses on the linear regression form of Equation (4.16), showing how it can be relaxed to yield an approximate solution in a computationally efficient manner.

Remark 4.2. In the linear regression setting, the dimensionality of $\mathbf{b}$ must be the same as the dimensionality of $\mathbf{x}_i \in \mathcal{R}^p$. Henceforth, the dimensionality of $\mathbf{b}$ will be denoted by p rather than d. Note also that code-lengths in this section will be expressed in nats, rather than bits.

We begin with the linear regression form of MDL estimation expressed in Equation (4.16), using the natural code-length function L_Z,

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}\in Q^p} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 L_Z(\mathbf{b}) \tag{4.17}$$

Formally, this is an integer programming problem over the $(q, \mathbf{m})$ pairs that represent the points of the form $2^{-q}\mathbf{m}$ in Q^p. We'd like to relax it to a problem on $\mathcal{R}^p$ so that descent methods can be used on it. If we could find a function $l(\mathbf{b})$, $\mathbf{b} \in \mathcal{R}^p$, that were a good approximation of $L_Z(\mathbf{b})$, $\mathbf{b} \in Q^p$, then we could substitute the relaxed problem

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}\in \mathcal{R}^p} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 l(\mathbf{b}) \tag{4.18}$$

This is the approach we'll take. Unfortunately, L_Z is not continuous on Q^p, so there is no immediate continuous extension to $\mathcal{R}^p$. In fact, for any $\mathbf{b}_0 \in \mathcal{R}^p$, we have

$$\lim_{\mathbf{b}\to\mathbf{b}_0, \mathbf{b}\in Q^p} L_Z(\mathbf{b}) = \infty \tag{4.19}$$

But what if we separate Equation (4.17) as follows:

$$\hat{\mathbf{b}} = \arg\min_{\delta=2^{-q},\ \mathbf{b}\in Q^p_\delta} L_Z(q) + \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 L_{\delta,Z}(\mathbf{b}) \tag{4.20}$$

where $L_{\delta,Z}$ is defined in definition 3.9 on page 35, and where, by the admittedly awkward notation $\arg\min_{\delta=2^{-q},\ \mathbf{b}\in Q^p_\delta}$ above, we mean that if $(\delta^*, \mathbf{b}^*)$ is a minimizing pair of (4.20), then $\hat{\mathbf{b}} = \mathbf{b}^*$. Now for fixed δ the minimization can be relaxed to $\mathcal{R}^p$ easily, because the natural code $L_{\delta,Z}$ on Q_δ^p has a natural extension to $\mathcal{R}^p$. According to Equation (3.110) on page 37, $L_{\delta,Z}$ is given by

$$L_{\delta,Z}(\hat{\mathbf{b}}) = \sum_{j=1}^{d} \lfloor \log(|\hat{b}_j|/\delta + 1) \rfloor + 2\lfloor \log(\lfloor \log(|\hat{b}_j|/\delta + 1)\rfloor + 1)\rfloor + 2 \tag{4.21}$$

for $\hat{\mathbf{b}} \in Q_\delta^d$. Defining a relaxed function $\tilde{l}_\delta(\mathbf{b})$ on $\mathbf{b} \in \mathcal{R}^d$ by

$$\tilde{l}_\delta(\mathbf{b}) \triangleq \sum_{j=1}^{d} \log(|b_j|/\delta + 1) + 2\log(\log(|b_j|/\delta + 1) + 1) + 2 \tag{4.22}$$

we obtain an asymptotically optimal δ-approximate code-length function, as defined by definitions 3.11 and 3.12 on page 36. However, we don't really require all the baggage of the lower-order terms, since they won't much affect minimizations involving $\tilde{l}_\delta$. By dropping the lower order terms, we arrive at the further relaxed approximation

$$l_\delta(\mathbf{b}) \triangleq \sum_{j=1}^{p} \ln(|b_j|/\delta + 1) \tag{4.23}$$

referred to in this dissertation as the *log penalty.* l_δ retains the asymptotic properties of $\tilde{l}_\delta$, so, in a loose sense, it can still be thought of as an asymptotically optimal approximate coding-length function, to first order.

Note also that $\sum_{j=1}^{p} \ln(|b_j|/\delta + 1)$ can be expressed as

$$\sum_{j=1}^{p} \ln(|b_j|/\delta + 1) = \sum_{j=1}^{p} \ln\left(\frac{|b_j| + \delta}{\delta}\right) \tag{4.24}$$

$$= \sum_{j=1}^{p} \ln(|b_j| + \delta) + p\ln(1/\delta) \tag{4.25}$$

so, for minimizations in which δ is a constant, we can drop the term $p\ln(1/\delta)$ and use the more succinct form

$$l'_\delta(\mathbf{b}) = \sum_{j=1}^{p} \ln(|b_j| + \delta) \tag{4.26}$$

Plugging our relaxed version l_δ of $L_{\delta,Z}$ in to Equation (4.20) leads to

$$\begin{aligned}
\hat{\mathbf{b}} &= \arg \min_{\delta=2^{-q},\ \mathbf{b}\in Q_\delta^p} L_Z(q) + \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 L_{\delta,Z}(\mathbf{b}) && (4.27)\\
&\approx \arg \min_{\delta\in\mathcal{R},\ \mathbf{b}\in\mathcal{R}^p} L_Z(\ln 1/\delta) + \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 l_\delta(\mathbf{b}) && (4.28)\\
&= \arg \min_{\delta\in\mathcal{R},\ \mathbf{b}\in\mathcal{R}^p} L_Z(\ln 1/\delta) + \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 \sum_{j=1}^{p} \ln(|b_j|/\delta + 1) && (4.29)\\
&= \arg \min_{\delta\in\mathcal{R},\ \mathbf{b}\in\mathcal{R}^p} L_Z(\ln 1/\delta) + 2p\sigma_Z^2 \ln(1/\delta) + \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 \sum_{j=1}^{p} \ln(|b_j| + \delta) && (4.30)\\
&\approx \arg \min_{\delta\in\mathcal{R},\ \mathbf{b}\in\mathcal{R}^p} 2p\sigma_Z^2 \ln 1/\delta + \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 \sum_{j=1}^{p} \ln(|b_j| + \delta) && (4.31)
\end{aligned}$$

In (4.28), we relax the domain from quantized to continuous. In (4.31), we drop the term $L_Z(\ln 1/\delta) = O(\ln\ln 1/\delta)$, since it is dominated by the higher-order term $2p\sigma_Z^2 \ln 1/\delta$. The relaxed minimization problem (4.31) is now in a form where it can be solved efficiently. Note that for fixed δ the first term is a constant and the minimization reduces to

$$\hat{\mathbf{b}} = \arg \min_{\mathbf{b}\in\mathcal{R}^p} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2\sigma_Z^2 \sum_{j=1}^{p} \ln(|b_j| + \delta) \tag{4.32}$$

Section 4.8 discusses a method for solving the above minimization. With such a method in hand, it is then straightforward to plug a handful of reasonable δ values into (4.31), say for $q = 1, \ldots, 10$, solving (4.32) in each case, and to select the $\hat{\mathbf{b}}$ value yielding the minimum over all. Since (4.31) is an approximation to the total code-length in nats required to express the data $\mathbf{y}$, the value $\hat{\mathbf{b}}$ that minimizes (4.31) is approximately the MDL estimator.

But this is not the only way to solve the problem. Strange as it may seem, having used the MDL principle to motivate the penalty $\sum_{j=1}^{p} \ln(|b_j|+\delta)$, we can now abandon that view and give the problem a conventional statistical treatment as a penalized linear regression problem. That is, rather than using minimum code-length as the criterion for establishing an optimal guess $\hat{\mathbf{b}}$, we can treat (4.32) simply as a penalized linear regression problem and consider it from the perspective outlined in Chapter 2 on page 5. To that end, we'll replace $2\sigma_Z^2$ in (4.31) with the free variable λ (in practice we would have needed to estimate σ_Z^2 anyway). Then, for each (δ, λ) pair, we have the following minimization problem

$$\hat{\mathbf{b}}(\delta, \lambda) = \arg\min_{\mathbf{b}\in\mathcal{R}^p} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda \sum_{j=1}^{p} \ln(|b_j| + \delta) \tag{4.33}$$

and we can use standard methods such as cross-validation to estimate the (δ, λ) pair that will yield the estimator $\hat{\mathbf{b}}$ with lowest expected prediction error. Note that (4.33) is just a generalization of (4.32) with σ_Z^2 replaced by the free parameter λ, so any computational method that solves the latter also solves the former.

4.6 Relaxations of the l^2 and l^1 Codes

We'll digress for a moment to re-emphasize that the log penalty, while natural, is not the only asymptotically optimal code. For example, the δ-quantized code-length functions, $L_{\delta,1}$ and $L_{\delta,1}$, described in Section 3.8 on page 34, are also asymptotically optimal, and they can be similarly relaxed to yield approximate code-length functions on $\mathcal{R}^p$. We'll illustrate using the $L_{\delta,2}$ code, with the derivation for the $L_{\delta,1}$ code being virtually identical.

We'll take the inner minimization of Equation (4.20) as our starting point, replacing $L_{\delta,Z}$ with $L_{\delta,2}$

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}\in Q_\delta^p} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + 2\sigma_Z^2 L_{\delta,2}(\mathbf{b}) \tag{4.34}$$

According to Equation (3.111) on page 37, $L_{\delta,2}$ is given by

$$L_{\delta,2}(\hat{\mathbf{b}}) = L_Z(\lceil \log \| \, |\hat{\mathbf{b}}|/\delta + \mathbf{1}\| \rceil) + \log |S(2^{\lceil \log \| \, |\hat{\mathbf{b}}|/\delta + \mathbf{1}\| \rceil})| + d \tag{4.35}$$

Defining a relaxed function, $\tilde{l}_\delta(\mathbf{b})$ on $\mathcal{R}^p$ by

$$\tilde{l}_{\delta,2}(\mathbf{b}) \stackrel{\triangle}{=} l_Z(\log \| \, |\mathbf{b}|/\delta + \mathbf{1}\|) + \log |S(2^{\log \| \, |\mathbf{b}|/\delta + \mathbf{1}\|})| + d \tag{4.36}$$

where l_Z is the natural relaxation of L_Z to $\mathcal{R}$, given by

$$l_Z(x) \stackrel{\triangle}{=} \log(|x| + 1) + 2\log(\log(|x| + 1) + 1) + 2 \tag{4.37}$$

yields an asymptotically optimal δ-approximate code-length function, as defined by definitions 3.11 and 3.12 on page 36. As with the log penalty, we don't really require all the baggage of the lower-order terms. Their presence or absence will not much affect the resulting minimization, so we can simplify things by removing them. According to Equation (3.71) on page 29, the principal order of (4.36) above is simply $p \ln \| \, |\mathbf{b}|/\delta + \mathbf{1}\|$, allowing for a relaxation to principal order of

$$l_{\delta,2}(\mathbf{b}) \stackrel{\triangle}{=} p \ln \| \, |\mathbf{b}|/\delta + \mathbf{1}\| \tag{4.38}$$

As with the log penalty, note also that $p \ln \| \, |\mathbf{b}|/\delta + \mathbf{1}\|$ can be expressed as

$$p \ln \| \, |\mathbf{b}|/\delta + \mathbf{1}\| = p \ln \sqrt{\sum_j \left(\frac{|b_j|}{\delta} + 1\right)^2} \tag{4.39}$$

$$= \frac{p}{2} \ln \sum_j \left(\frac{|b_j| + \delta}{\delta}\right)^2 \tag{4.40}$$

$$= \frac{p}{2} \ln \frac{1}{\delta^2} \sum_j (|b_j| + \delta)^2 \tag{4.41}$$

$$= p \ln \| \, |\mathbf{b}| + \delta \mathbf{1}\| + p \ln \frac{1}{\delta} \tag{4.42}$$

so, for minimizations in which δ is constant, we can drop the term $p \ln 1/\delta$ and use the more succinct form

$$l'_{\delta,2}(\mathbf{b}) = p \ln \| \, |\mathbf{b}| + \delta \mathbf{1} \| \tag{4.43}$$

A similar process applied to the $L_{\delta,1}$ code yields the respective relaxed forms

$$l_{\delta,1}(\mathbf{b}) \triangleq p \ln \| \, |\mathbf{b}|/\delta + \mathbf{1} \|_1 \quad \text{and} \tag{4.44}$$

$$l'_{\delta,1}(\mathbf{b}) \triangleq p \ln \| \, |\mathbf{b}| + \delta \mathbf{1} \|_1 \tag{4.45}$$

Substituting the relaxed forms $l'_{\delta,2}$ and $l'_{\delta,1}$ for the log penalty in Equation (4.32) yields what might be called the MDL forms of ridge and lasso regression

$$\hat{\mathbf{b}}_{\text{ridge,MDL}} = \arg\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2p\sigma_Z^2 \ln \| \, |\mathbf{b}| + \delta \mathbf{1} \| \quad \text{and} \tag{4.46}$$

$$\hat{\mathbf{b}}_{\text{lasso,MDL}} = \arg\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2p\sigma_Z^2 \ln \| \, |\mathbf{b}| + \delta \mathbf{1} \|_1 \tag{4.47}$$

For non-zero δ, the solutions to the above forms are not precisely ridge and lasso solutions. For example, there will be no λ such that $\hat{\mathbf{b}}_{\text{ridge}}(\lambda)$ is the solution to (4.46) above. However, as $\delta \to 0$, the solution converges to a ridge solution. At $\delta = 0$, the minimizations above have spurious non-local minima at $\mathbf{b} = 0$ (which stems directly from using a quantization width of 0), but their local minima are still well-defined and will lie precisely on the ridge and lasso solution paths respectively. This is due primarily to the fact that (4.46) and (4.47) have the same level curves when $\delta = 0$ and is straightforward to show by taking derivatives.

For example, consider (4.46). At $\delta = 0$, it reduces to the simpler, unquantized form

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2p\sigma_Z^2 \ln \|\mathbf{b}\| \tag{4.48}$$

If $\hat{\mathbf{b}}$ is a local minimum of the above equation, then, taking derivatives, it must satisfy

$$\mathbf{0} = 2\mathbf{X}^T\mathbf{X}\hat{\mathbf{b}} - 2\mathbf{X}^T\mathbf{y} + \frac{2p\sigma_Z^2}{\|\mathbf{b}\|^2}\mathbf{b}, \quad \text{which implies} \tag{4.49}$$

$$(\mathbf{X}^T\mathbf{X} + \frac{2p\sigma_Z^2}{\|\mathbf{b}\|^2}\mathbf{I})\hat{\mathbf{b}} = \mathbf{X}^T\mathbf{y}, \quad \text{leading to} \tag{4.50}$$

$$\hat{\mathbf{b}} = (\mathbf{X}^T\mathbf{X} + \frac{2p\sigma_Z^2}{||\mathbf{b}||^2}\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{4.51}$$

The ridge solution path is shown in [8] to be $\mathbf{b}(\lambda) = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$, so, if $\hat{\mathbf{b}}$ is a local minimum of (4.48) then, via comparison with (4.51), it obviously lies on the ridge solution path for $\lambda = 2p\sigma_Z^2/||\mathbf{b}||^2$.

More generally now, suppose f is a convex objective function and g is a convex penalty function. Suppose further that $\tilde{g}$ has the same level curves as g. That is, suppose $\tilde{g}(\mathbf{b}) = h(g(\mathbf{b}))$ for some monotonic increasing function h from $\mathcal{R}$ to $\mathcal{R}$. (If h were not monotonic, it would map two level curves of g onto the same level curve of $\tilde{g}$. If h were monotonic decreasing, it would invert the nesting order of the level curves.) Then any local minimum to the penalized regression

$$f(\mathbf{b}) + \alpha h(g(\mathbf{b})) \tag{4.52}$$

for a particular $\alpha > 0$, is also a local minimum to

$$f(\mathbf{b}) + \lambda g(\mathbf{b}) \tag{4.53}$$

for some $\lambda > 0$. To see this, suppose that $\mathbf{b}$ were a local minimum of both (4.52) and (4.53) above for some $\alpha > 0$ and $\lambda > 0$. Then, taking derivatives, we have

$$\mathbf{0} = \nabla f(\hat{\mathbf{b}}) + \lambda \nabla g(\hat{\mathbf{b}}) \quad \text{and} \tag{4.54}$$

$$\mathbf{0} = \nabla f(\hat{\mathbf{b}}) + \alpha \dot{h}(g(\hat{\mathbf{b}}))\nabla g(\hat{\mathbf{b}}) \tag{4.55}$$

from which we conclude, comparing the above two forms, that

$$\lambda = \alpha \dot{h}(g(\hat{\mathbf{b}})) \tag{4.56}$$

Obviously (4.56) is a necessary condition for $\hat{\mathbf{b}}$ to be a local minimum of both (4.52) and (4.53), but it is easy to see that it is also a sufficient condition. If $\hat{\mathbf{b}}$ is a local minimum of (4.52) for a particular $\alpha > 0$, then it is also a local minimum of (4.53) for $\lambda = \alpha \dot{h}(g(\hat{\mathbf{b}}))$. The converse does not quite hold. If $\hat{\mathbf{b}}$ is a local minimum of (4.53), then it is a critical point of (4.52) for $\alpha = \lambda / \dot{h}(g(\hat{\mathbf{b}})$, but not necessarily a local minimum. The implication works in the first direction because all the critical points of (4.53) must be local minima, under the assumption that f and g are convex.

Note also that, since h is monotonic increasing, $\dot{h}(g(\mathbf{b}))$ is positive for all $\mathbf{b}$ and hence $\lambda = \alpha \dot{h}(g(\hat{\mathbf{b}})$ is positive whenever α is positive.

4.7 Sparsity

It has been mentioned that the log penalty leads to sparse solutions, an assertion that is corroborated by the experiments in Chapter 6. From a philosophical point of view, it should be unsurprising that a penalty based upon an asymptotically optimal coding cost and the MDL principle should yield sparse solutions, since the MDL principle is primarily about finding low complexity solutions, and a sparse solution is one kind of low complexity solution. We can get a more solid grasp on why the log penalty in particular should yield sparse solutions when we consider the shape of its level curves. Figure 4.1 illustrates one of the level curves of the log penalty in two dimensions. The curve is quite pointy, and it is essentially this pointiness that leads to sparse solutions. A local minimum of the log-penalized linear regression must occur at a point $\hat{\mathbf{b}}$ at which the level curve of the log penalty going through $\hat{\mathbf{b}}$ is tangent to the level curve of the residual sum of squares term going through $\hat{\mathbf{b}}$. In particular, at $\mathbf{b} = \hat{\mathbf{b}}$, we have

$$\nabla_{\mathbf{b}}\{\|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2\} = -\lambda \nabla_{\mathbf{b}}\{\sum_j \ln(|b_j| + \delta)\} \tag{4.57}$$

Other things being equal, the point of tangency is more likely to occur at the pointy end of the log penalty's level curve, which represents a sparse solution, than at the interior of the curve, which represents a non-sparse solution. In p dimensions, the log penalty's level surfaces have k-dimensional pointy edges for all $k < p$.

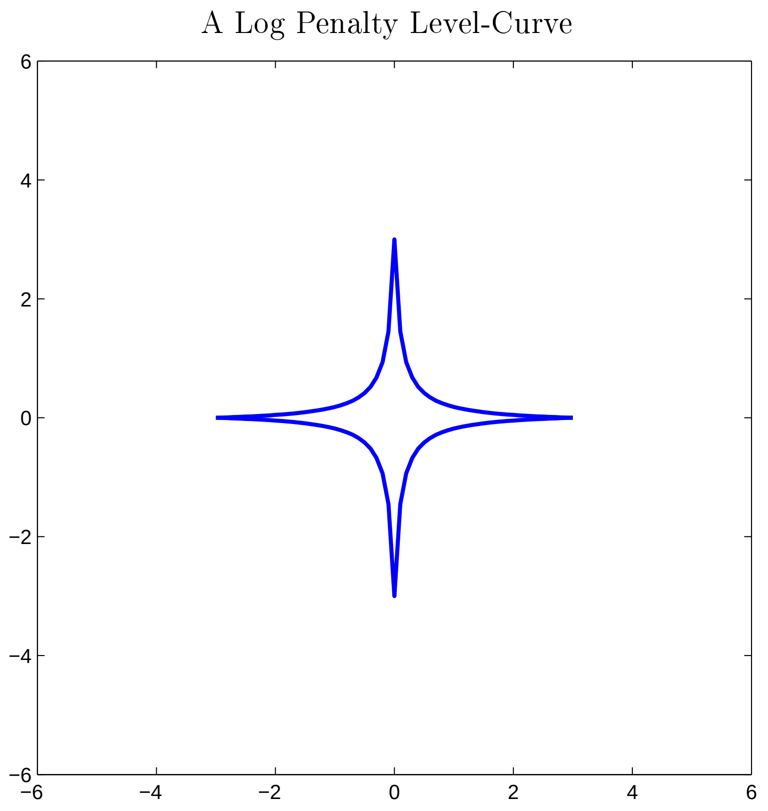

Figure 4.1: *This graph shows a level curve in two dimensions of the log penalty. The pointiness of the curve is essentially what causes log-penalized regression to yield sparse solutions*

4.8 Iterative Linearization

The main impediment to efficiently solving the log penalty minimization (4.33) is that the objective is not convex; specifically, the log penalty term is not convex, and, in fact, it is concave on the positive orthant. This means that, in general, there will be multiple local minima, and the machinery of convex optimization cannot be directly applied. However, a method called *iterative linearization*, described in [7], allows one to efficiently find a local minimum of the log penalty minimization through solving a sequence of convex minimization problems, each of which locally approximates the log penalized minimization about the current iterate.

(*Current iterate* is a concept from descent method techniques, all of which are iterative. One begins at some arbitrary point in the solution space, the *zeroth iterate*, calculates a descent direction, takes a small step in that direction, leading to a new point, the *first iterate*. One then repeats the process, calculating a new descent direction, takes a small step, leading to the next iterate, and so on. This results in a sequence of iterates that converges to a minimum.)

The linearization trick is to take a first-order Taylor series expansion of the log penalty about the current iterate and to solve the resulting linearized minimization problem to get the next iterate, leading to the following iterative method:

$$\mathbf{b}^{(k+1)} = \arg\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + \lambda \sum_j \ln(|b_j^{(k)}| + \delta) + \frac{1}{|b_j^{(k)}| + \delta}(|b_j| - |b_j^{(k)}|) \tag{4.58}$$

$$= \arg\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + \lambda \sum_j \frac{|b_j|}{|b_j^{(k)}| + \delta} + \lambda \sum_j \ln(|b_j^{(k)}| + \delta) - \frac{|b_j^{(k)}|}{|b_j^{(k)}| + \delta} \tag{4.59}$$

$$= \arg\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + \lambda \sum_j \frac{|b_j|}{|b_j^{(k)}| + \delta} \tag{4.60}$$

Equation (4.58) is derived from Equation (4.33) by substituting for the log penalty its first-order Taylor series expansion about the k^{th} iterate. (Actually, the substitution made is not quite a first-order Taylor series expansion, because of the absolute-value signs. This makes the substituted penalty function a cone with hyperplanar faces rather than simply a hyperplane, while still corresponding locally to the first-order Taylor series expansion in the region of the k^{th} iterate. It also makes the substituted penalty have a finite minimum and mirrors the symmetry of the original log penalty.) Equation (4.59) merely re-arranges the terms in (4.59) so that constants not affecting the minimization are collected together. Equation (4.60) then drops those constants, since they don't affect the result of the minimization.

The method works as follows: we begin at $k = 0$ with an arbitrary zeroth iterate, say $\mathbf{b}^{(0)} = \mathbf{0}$. Plugging this in to Equation (4.60), we solve the resulting minimization, which is convex, leading to the first iterate $\mathbf{b}^{(1)}$. We then plug $\mathbf{b}^{(1)}$ into (4.60), leading to a new convex minimization problem whose solution gives us $\mathbf{b}^{(2)}$, and so on. This results in a sequence of convex minimization problems whose solutions lead to a sequence $\{\mathbf{b}^{(k)}\}$ of iterates that converges to a local minimum of the original log penalized regression problem (4.33).

Note that, for each $\mathbf{b}^{(k)}$, the minimization (4.60) is a weighted lasso minimization. That is, it has the form

$$\hat{\mathbf{b}} = \arg\min \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + \lambda \sum_j \frac{|b_j|}{w_j} \tag{4.61}$$

where $w_j > 0$ are arbitrary weights. Thus, the relationship between log-penalized linear regression and the lasso is a close one, with the log penalty solution being the limit of a succession of iteratively re-weighted lasso minimizations. Equation (4.60) can be solved efficiently using standard quadratic programming methods or by the least-angle-regression (LARS) method of [6]. The LARS method is straightforward but somewhat intricate and will not be discussed here. An excellent reference on convex optimization methods in general is [2].

One thing that needs to be mentioned, though, is a *stopping criterion* for the above iterative linearization method. The stopping criterion tells us when the current iterate $\mathbf{b}^{(k)}$ is sufficiently close to the actual local minimum for us to stop the iteration. To obtain a stopping criterion, we note that if $\mathbf{b}$ is a local minimum of (4.33), then the gradient of the objective function at $\mathbf{b}$ must be the p-dimensional zero-vector, leading to

$$\mathbf{0} = \nabla_{\mathbf{b}} \left[\|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + \lambda \sum_j \frac{|b_j|}{|b_j^{(k)}| + \delta} \right] \tag{4.62}$$

$$= 2\mathbf{X}^T\mathbf{X}\mathbf{b} - 2\mathbf{X}^T\mathbf{y} + \lambda \frac{\text{sign}(\mathbf{b})}{|\mathbf{b}| + \delta\mathbf{1}} \tag{4.63}$$

where

$$\frac{\text{sign}(\mathbf{b})}{|\mathbf{b}| + \delta\mathbf{1}}$$

denotes the vector whose j^{th} component is

$$\frac{\text{sign}(b_j)}{|b_j| + \delta}$$

The iteration can then be stopped when (4.63) becomes sufficiently small, say when

$$\|2\mathbf{X}^T\mathbf{X}\mathbf{b}^{(k)} - 2\mathbf{X}^T\mathbf{y} + \lambda \frac{\text{sign}(\mathbf{b}^{(k)})}{|\mathbf{b}^{(k)}| + \delta\mathbf{1}}\| \leq \gamma \tag{4.64}$$

where γ is some small positive threshold, say $\gamma = .001$.

Putting together all the pieces just described, we have the following iterative solution method for obtaining a local minimum to (4.33)

Algorithm 4.1 (Iterative Solution To The Log Penalty Minimization). Let data $(\mathbf{X}, \mathbf{y})$ and parameters (δ, λ) be given, as well as a starting iterate $\mathbf{b}^{(0)}$ and a tolerance $\gamma > 0$. Then a local minimum of (4.33), to within a tolerance determined by γ, is given by the following algorithm:

1. $\mathbf{b} \leftarrow \mathbf{b}^{(0)}$.

2. Repeat

   (a) if

   $$\|2\mathbf{X}^T\mathbf{X}\mathbf{b} - 2\mathbf{X}^T\mathbf{y} + \lambda \frac{\text{sign}(\mathbf{b})}{|\mathbf{b}| + \delta\mathbf{1}}\| \leq \gamma$$

   then stop, returning $\mathbf{b}$ as the solution.

(b) Otherwise, assign

$$\mathbf{w} \leftarrow \mathbf{b} + \delta \mathbf{1}$$

(c) and assign

$$\mathbf{b} \leftarrow \arg\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + \lambda \sum_j \frac{|b_j|}{w_j}$$

Line 2c above can be computed using the LARS algorithm [6] or any standard convex problem solver.

Remark 4.3. As is standard with any linear regression method, we assume in algorithm 4.1 above that the $\mathbf{X}$ matrix includes a column of all 1s as its initial column, to capture the intercept of the model. If it does not, the $\mathbf{X}$ matrix should be so augmented and the dimension of the $\mathbf{b}$ vector increased by one to accommodate this.

As with the lasso and ridge models, it is also important to standardize the data before performing the above algorithm. This means at the very least that each column of the $\mathbf{X}$ matrix should have squared norm equal to n. The reason for normalization is simple. Consider scaling all the values in the j^{th} column by some large number α. Then the corresponding coefficient values b_j could reasonably take on would be reduced by a factor of α, meaning that its contribution to the penalty $\ln(|b_j|+\delta)$ would be very small, essentially encouraging introduction of this predictor into the model. In order not to bias the process toward one predictor over the other, we normalize the columns. It is less crucial to normalize $\mathbf{y}$, though this is also standard practice. It is also less crucial to subtract out the mean μ_j of each column of $\mathbf{X}$ (except the first column of all ones) from each element in the column, resulting in a matrix each of whose columns has mean zero, though this is also standard practice and results in a solution that is translation independent.

Remark 4.4. It is important to remember that (4.33) is non-convex and therefore has multiple local minima in general, and that algorithm 4.1 only leads to one of these local minima, not necessarily the best. The local minimum found depends entirely upon the initial choice for the zeroth iterate $\mathbf{b}^{(0)}$. (In practice the solution will also depend upon the tolerance γ.)

Neither ridge nor the lasso has this deficiency. For a particular λ value, the ridge and lasso minimizations are convex, leading to a global minimum. However, it should be pointed out that finding the solution for a particular λ value is not the end of the problem. One must still use some method, such as cross-validation, the Akaike criterion [1], etc., to estimate which λ value leads to a model with minimum prediction error, and when this aspect is factored into the equation, the problem again becomes non-convex, even for ridge and the lasso.

Finally, the log penalty leads to sparser models than the lasso, and Chapter 6 demonstrates examples in which these sparser models have comparable or better prediction error than the lasso. So, in cases in which sparsity is desired, or believed to be a property of the solution, the log penalty may be preferable to the lasso. After all, there is no reason a priori why the local minimum of one method might not be sparser and even have lower prediction error than the global minimum of another method. This is seen readily when one compares the performance of ridge, whose solutions are global minima, against the log penalty in the simulated example whose results are tabulated in Figure 6.1 on page 89.

4.9 Obtaining Different Solution Paths

A good way to think about the problem of solving for $\hat{\mathbf{b}}(\delta, \lambda)$ in (4.33) is to think of δ as being fixed first, and then to think of the resulting minimization as having just the single free parameter λ, for which we desire to obtain the solution path $\mathbf{b}_\delta(\lambda)$ as λ progresses through a finite set of values $\lambda_1, \ldots, \lambda_N$, ranging from $\lambda_1 = 0$ to $\lambda_N = \lambda_{\max}$. At $\lambda_1 = 0$, the solution $\hat{\mathbf{b}}_\delta(0)$ is the ordinary least squares solution. As λ grows, increasing from 0 to $\lambda_{\max}$, the solution path $\hat{\mathbf{b}}_\delta(\lambda)$ shrinks toward the origin. Finally, at $\lambda_N = \lambda_{\max}$, $\hat{\mathbf{b}}_\delta(\lambda_{\max}) = \mathbf{0}$.

For a fixed δ and particular range of λ values, it is possible to obtain different solution paths, depending on whether one increases λ from zero or decreases it from

$\lambda_{\max}$, and depending on how one selects the zeroth iterate to begin the iterative solution process. One reasonable approach to selecting the zeroth iterate for λ_k is to use the solution value $\hat{\mathbf{b}}(\delta, \lambda_{k-1})$. That is, the starting point is taken to be the solution for the previous λ-value. If successive λ-values $\lambda_1, \dots, \lambda_N$ are close to each other, the solution to the current λ-value should be close to the solution for the previous λ-value, so starting there makes sense and is more likely to ensure continuity in the solution path. Another reasonable approach is simply to choose $\mathbf{b}^{(0)} = \mathbf{0}$ for all λ_k, in which case the solution path is independent of whether one increases or decreases λ, since the zeroth iterate in each case does not depend on the solution to the previous λ-value.

Assigning the zeroth iterate to be $\hat{\mathbf{b}}_\delta(\lambda_{k-1})$ as λ increases from $\lambda_1 = 0$ to $\lambda_N = \lambda_{\max}$ results in what we'll call the *backward method.* Assigning the zeroth iterate to be $\hat{\mathbf{b}}_\delta(\lambda_{k+1})$ as λ decreases from $\lambda_N = \lambda_{\max}$ down to $\lambda_1 = \mathbf{0}$ results in what we'll call the *forward method.* These methods are so-named because they have something in common with the traditional greedy statistical techniques for subset selection termed *forward* and *backward* stepwise regression respectively. Descriptions of forward and backward stepwise regression abound, and can be found in [5, 6, 8].

Assigning the zeroth iterate to be $\mathbf{0}$ for all λ results in what we'll call the *fixed method.* The fixed method is obviously less sensitive to initial conditions, and also has the nice theoretical property that its first iterate is always the lasso solution. The forward and backward methods are greedier, but exploit continuity assumptions.

Algorithm 4.2 (Forward Method). Let standardized data $(\mathbf{X}, \mathbf{y})$, a fixed $\delta > 0$, a sequence-length $N > 1$, and a tolerance $\gamma > 0$ be given. Then the forward-method solution path $\hat{\mathbf{b}}_\delta(\lambda_k), k = 1, \dots, N$, is given by the following algorithm:

```
1. $\mathbf{b}^{(0)} \leftarrow \mathbf{0}$

2. $\lambda_{\max} \leftarrow 2\delta \|\mathbf{X}^T\mathbf{y}\|_\infty$

3. $d\lambda \leftarrow \lambda_{\max}/(N-1)$

4. $\lambda_N \leftarrow \lambda_{\max}$
```

5. Repeat for $k \leftarrow N$ down to 1
   (a) $\hat{\mathbf{b}}_\delta(\lambda_k) \leftarrow \text{Alg4.1}(\mathbf{X}, \mathbf{y}, \delta, \lambda_k, \mathbf{b}^{(0)}, \gamma)$
   (b) $\mathbf{b}^{(0)} \leftarrow \hat{\mathbf{b}}_\delta(\lambda_k)$
   (c) $\lambda_{k-1} \leftarrow \lambda_k - d\lambda$

$\lambda_{\max}$ in 2 is calculated to be the smallest λ value that will yield a solution of $\mathbf{0}$ as a local minimum of (4.33). (Therefore the first iteration of the above loop is irrelevant, since its solution is known to be $\mathbf{0}$, and could be optimized out.) The value is obtained by considering that at a local minimum $\mathbf{b}$ the gradient must be $\mathbf{0}$. This leads to

$$\mathbf{0} = \nabla_{\mathbf{b}} \left[\|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + \lambda \sum_j \frac{|b_j|}{|b_j^{(k)}| + \delta} \right] \tag{4.65}$$

$$= 2\mathbf{X}^T\mathbf{X}\mathbf{b} - 2\mathbf{X}^T\mathbf{y} + \lambda \frac{\text{sign}(\mathbf{b})}{|\mathbf{b}| + \delta\mathbf{1}} \tag{4.66}$$

where $|\mathbf{b}|$ and $\text{sign}(\mathbf{b})$ are the vectors whose j^{th} components are $\text{sign}(b_j)$ and $|b_j|$ respectively. We run into a slight problem when we try to solve (4.66) for λ by plugging in $\mathbf{b} = \mathbf{0}$, because the sign function is discontinuous at 0, or, in other words, $|b_j|$ is non-differentiable at 0. A non-differentiable point is a local minimum if $\mathbf{0}$ belongs to the subgradient at that point. Let us extend the definition of $\text{sign}(\mathbf{b})$ so that it represents the subgradient of $|\mathbf{b}|$. To be precise, we'll define

$$\text{sign}(\mathbf{b}) = \{\mathbf{v} : v_j \in \begin{cases} \{1\} & b_j > 0 \\ \{-1\} & b_j < 0 \\ [-1, 1] & b_j = 0 \end{cases} \tag{4.67}$$

Generalizing(4.66) and using our extended definition of sign, we have that $\mathbf{b}$ is a local minimum in case

$$\mathbf{0} \in 2\mathbf{X}^T\mathbf{X}\mathbf{b} - 2\mathbf{X}^T\mathbf{y} + \lambda \frac{\text{sign}(\mathbf{b})}{|\mathbf{b}| + \delta\mathbf{1}} \tag{4.68}$$

which at $\mathbf{b} = 0$ implies that

$$2\mathbf{X}^T\mathbf{y} \in \frac{\lambda}{\delta}\text{sign}(\mathbf{b}) \tag{4.69}$$

leading to

$$\lambda = 2\delta\|\mathbf{X}^T\mathbf{y}\|_\infty \tag{4.70}$$

Algorithm 4.3 (Backward Method). Let standardized data $(\mathbf{X}, \mathbf{y})$, a fixed $\delta > 0$, a finite, monotonically increasing sequence $\{\lambda_k\}_{k=1}^{k=N}$ satisfying $\lambda_1 = 0$, and a tolerance $\gamma > 0$ be given. Then the backward-method solution path $\hat{\mathbf{b}}_\delta(\lambda_k), k = 1, \ldots, N$, is given by the following algorithm:

```
1. $\mathbf{b}^{(0)} \leftarrow \mathbf{0}$
2. Repeat for $k \leftarrow 1$ to $N$
    (a) $\hat{\mathbf{b}}_\delta(\lambda_k) \leftarrow \text{Alg4.1}(\mathbf{X}, \mathbf{y}, \delta, \lambda_k, \mathbf{b}^{(0)}, \gamma)$
    (b) $\mathbf{b}^{(0)} \leftarrow \hat{\mathbf{b}}_\delta(\lambda_k)$
```

Remark 4.5. Note that the value $\lambda_{\max}$ as calculated in algorithm 4.2 cannot be used for the backward method. Because of the non-convexity of the problem, nonzero minima with associated λ values higher than the $\mathbf{0}$-solution's $\lambda_{\max}$ can and do occur. Some ad hoc method must be used to determine a reasonable value for λ_N in the backward method. The value should be high enough that $\hat{\mathbf{b}}_\delta(\lambda_N) = \mathbf{0}$.

Algorithm 4.4 (Fixed Method). Let standardized data $(\mathbf{X}, \mathbf{y})$, a fixed $\delta > 0$, a sequence-length $N > 1$, and a tolerance $\gamma > 0$ be given. Then the fixed-method solution path $\hat{\mathbf{b}}_\delta(\lambda_k), k = 1, \ldots, N$, is given by the following algorithm:

```
1. $\mathbf{b}^{(0)} \leftarrow \mathbf{0}$
2. $\lambda_{\max} \leftarrow 2\delta\|\mathbf{X}^T\mathbf{y}\|_\infty$
3. $d\lambda \leftarrow \lambda_{\max}/(N-1)$
4. $\lambda_N \leftarrow \lambda_{\max}$
5. Repeat for $k \leftarrow 1$ to $N$
    (a) $\hat{\mathbf{b}}_\delta(\lambda_k) \leftarrow \text{Alg4.1}(\mathbf{X}, \mathbf{y}, \delta, \lambda_k, \mathbf{b}^{(0)}, \gamma)$
    (b) $\lambda_{k-1} \leftarrow \lambda_k - d\lambda$
```

Remark 4.6. The fixed method as defined above is more like the forward method, since for every λ its zeroth iterate starts at $\mathbf{b}^{(0)} = \mathbf{0}$. One could just as easily define a fixed method in which for every λ the zeroth iterate started at $\mathbf{b}^{(0)} = \mathbf{b}_{LS}$, the least-squares solution. Such a method would be more like the backward method, and, obviously, would not yield the same solution path.

As mentioned earlier in this section, the forward and backward methods are related to forward and backward stepwise regression. Forward regression adds to the current model the predictor that most reduces the norm of the current residual $\|\mathbf{y} - \mathbf{X}\mathbf{b}\|$. Backward regression subtracts from the current model the predictor that least increases the norm of the current residual. These methods do not in general yield the same sequence of predictor subsets. Continuous subset selection methods are able to add "a little bit" of a predictor at a time, yielding continuous solution paths. This phenomenon is discussed in [6, 17], and comparisons to forward and backward stepwise regression are drawn. Because the lasso is convex, its minima are global, and forward/backward methods yield precisely the same solution paths. While the log penalty provides for continuous subset selection via continuous solution paths, the non-convexity of the log penalty means that forward and backward methods will not in general yield the same solution paths. Nevertheless, because its solutions are based on iteratively solving weighted-lasso problems, many of the associations between lasso and stepwise regression still apply. For example, when the log penalty forward method adds a predictor to the active set (the *active set* is the set of predictors whose coefficients are currently non-zero; see [6]), it will be the one most highly correlated with the residual, as in lasso. When the log penalty backward method drops a predictor from the active set, it will be one whose coefficient value has been driven to 0, as in lasso.

Stepwise regression and lasso can be seen as opposite ends of a spectrum, with the log penalty sitting somewhere in between. On the greediest end, we have stepwise regression, which, when it adds a predictor to the model, adds the entire predictor, assigning its associated coefficient to have full magnitude, that corresponding to the least-squares solution for the current set of predictors. On the least greedy end, we have the lasso, which, when it adds a predictor to the model, increases the magnitude of that predictor's coefficient only until another predictor becomes equally worthy of addition to the model, based on correlation with the residual. Somewhere in the middle is the log penalty. When it adds a predictor, it doesn't add the full predictor, but neither does it stop increasing that predictor's coefficient when another predictor becomes equally worthy of addition, based on correlation with the residual. The log penalty will prefer the existing active predictor to the inactive predictor, continuing to increase the active predictor's coefficient, and not adding in a new predictor until its correlation with the residual actually exceeds the correlations of the active predictors. The exact relationship can be determined when we consider that a local minimum solution when restricted to the active predictors must have gradient zero. Accordingly, for a particular λ and δ, let $\mathbf{b}_A$ be a local minimum of the log-penalized linear regression, restricted to just the active (non-zero) components. Then $\mathbf{b}_A$ satisfies

$$\mathbf{0} = \nabla_{\mathbf{b}_A}\left[\|\mathbf{y}-\mathbf{X}_A\mathbf{b}_A\|^2 + \lambda\sum_{j\in A}\frac{|b_A(j)|}{|b_j^{(k)}|+\delta}\right] \tag{4.71}$$

$$= 2\mathbf{X}_A^T\mathbf{X}_A\mathbf{b}_A - 2\mathbf{X}_A^T\mathbf{y} + \lambda\frac{\operatorname{sign}(\mathbf{b}_A)}{|\mathbf{b}_A|+\delta\mathbf{1}} \tag{4.72}$$

which implies

$$\mathbf{X}_A^T(\mathbf{y} - \mathbf{X}_A\mathbf{b}_A) = \frac{\lambda}{2}\frac{\text{sign}(\mathbf{b}_A)}{|\mathbf{b}_A| + \delta\mathbf{1}} \tag{4.73}$$

Now, if we define $\mathbf{c}_A \triangleq \mathbf{X}_A^T(\mathbf{y} - \mathbf{X}_A\mathbf{b}_A)$ to be the vector of correlations of the active predictors with the current residual, then

$$c_j = \frac{\lambda}{2}\frac{\text{sign}(b_A(j))}{|b_A(j)| + \delta}, \quad \forall j \in A \tag{4.74}$$

which implies

$$\left(\frac{c_j}{\text{sign}(b_A(j))}\right)(|b_A(j)| + \delta) = \frac{\lambda}{2}, \quad \forall j \in A \tag{4.75}$$

From considerations discussed in [6, 13, 14], we know that $c_j/\text{sign}(b_A(j)) = |c_j|$, leading to

$$|c_j|(|b_A(j)| + \delta) = \frac{\lambda}{2}, \quad \forall j \in A \tag{4.76}$$

From (4.76) we deduce that if k is the index of a predictor that has just become active (meaning its coefficient b_k has been 0 up until now), and if j is the index of any already active predictor, then

$$\delta|c_k| = |c_j|(|b_A(j)| + \delta) \quad \text{whence} \tag{4.77}$$

$$|c_k| = (1 + \frac{|b_A(j)|}{\delta})|c_j| \tag{4.78}$$

This raises the bar for new predictors to enter the model, effectively creating sparser models.

4.10 Overcomplete Systems

Since the log penalty aggressively seeks sparse solutions, it is an apt method whenever there is reason to believe that the true underlying solution is sparse, or simply whenever a sparse solution is desired on practical grounds. If one is faced with a regression problem in which the number of predictors is large, say in the thousands, but one suspects that only a handful of predictors actually play a role in the phenomenon being investigated, then the log penalty's preference for sparse solutions will be an advantage.

A particular kind of problem in which this often applies is that of an overcomplete system, in which the number of predictors p exceeds the number of data points n. If $p >> n$, then a sparse solution (sparse in the sense that the number of predictors selected to be in the model is far fewer than the total number of available predictors) can be returned by virtually any method, even ordinary least squares, but this will be an artifact of the problem set-up only, as one can always fit the data exactly using no more than n predictors, and, in fact, any independent set of n predictors would do the job! In such a case, we are unlikely to place too much faith in the particular n predictors returned by a regression method. What we need in this case is a solution that is sparse even relative to n. If we get a solution that uses, say, only $n/2$ predictors, then there is reason to believe that the solution is significant rather than artifactual. Once again, the log penalty may be of use.

When dealing with an overcomplete system, one can use the LARS/lasso method [6] to winnow the set of predictors to a much smaller set—though it will still be an overcomplete set—before applying a log penalty algorithm to the data. An example will illustrate the technique. Suppose our standardized data matrix $\mathbf{X}$ were of dimension $50 \times 10,000$. Rather than running the log penalty directly on all $10,000$ predictors, we first apply to $\mathbf{X}$ the LARS algorithm for obtaining the entire lasso solution path. The computation cost is merely one log penalty iteration. We then examine the solution path and mark as *useful* any predictor that at any time is active (has a nonzero coefficient) along the path. The number of useful predictors so marked will be greater than 50 but far less than $10,000$. We then create a new data matrix

$\tilde{\mathbf{X}}$ containing only the columns of the useful predictors, and perform log-penalized regression on $\tilde{\mathbf{X}}$. The intuition behind such a procedure is that, if a predictor truly plays a role in the relationship between $\mathbf{x}$ and y, it will probably become active at least somewhere along the lasso solution path. Further, since the log penalty is even more stringent about sparsity than lasso, if the predictor is never active under lasso, it's unlikely ever to be active under the log penalty. Both the standard log penalty algorithm and the LARS-winnowed pre-processing method were applied to the Golub gene classification experiment described in Section 6.2, yielding identical results.

Remark 4.7. Note that the useful predictors, as defined in the preceding, are those that become active at least once along the lasso solution path, which is not the same thing as those that are active at the end of the solution path ($\lambda = 0$), corresponding to the minimum l^1 norm solution. It is clear from the analysis given in [6] that, in an overcomplete system of full rank, no solution can have more than n predictors in it, and exactly n predictors will participate in the solution for $\lambda = 0$. (The same must be true of the log penalty's solutions, since, ultimately, each solution is the result of a weighted-lasso minimization.) Nonetheless, the number of predictors that are *ever* active along the solution path can easily exceed n.

Chapter 5

Comparing The Penalties

We have seen that the MDL form of linear regression, when relaxed to $\mathcal{R}^p$, leads to a penalized linear regression in which the penalty is interpreted as a coding cost. In particular, the log penalty corresponds to an asymptotically optimal coding cost, which this dissertation has argued to be a desirable property in a penalty function. However, when we consider ridge and the lasso from this viewpoint, interpreting their penalties as coding costs, the penalties are not asymptotically optimal. Does this mean that they are biased toward a particular area of the solution space, as this dissertation has argued? If so, toward what portion of the space are they biased? Further, since ridge and lasso are standard methods that have been proven to work well in practice, is asymptotic optimality really that important? Perhaps surprisingly, it turns out that it is possible to view ridge and lasso from a perspective in which their penalties can be re-interpreted as log-like penalties that are asymptotically optimal, providing further support for the notion that asymptotic optimality is actually an important concept.

5.1 Asymptotic Optimality Vis-à-Vis Ridge and the Lasso

It is clear that ridge regression,

$$\hat{\mathbf{b}}_{\text{ridge}}(\lambda) = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda||\mathbf{b}||^2 \tag{5.1}$$

and the lasso,

$$\hat{\mathbf{b}}_{\text{lasso}}(\lambda) = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda||\mathbf{b}||_1 \tag{5.2}$$

have penalties that are not asymptotically optimal when interpreted as coding costs. In the case of ridge, the penalty is $g(\mathbf{b}) = ||\mathbf{b}||^2$; in the case of the lasso, the penalty is $g(\mathbf{b}) = ||\mathbf{b}||_1$; and for both these cases, we have

$$\lim_{r\to\infty} \frac{\max_{||\mathbf{b}||\leq r} g(\mathbf{b})}{p \ln r} = \infty \tag{5.3}$$

In point of fact, a penalty need not be asymptotically optimal in order to perform well on a particular set of data $(\mathbf{X}, \mathbf{y})$. For one thing, the least squares solution $\mathbf{b}_{ls}$, corresponding to $\lambda = 0$, is the "biggest" possible solution we can get. That is, regardless of the value of λ, all solutions $\mathbf{b}(\lambda)$ will lie in the set $\{\mathbf{b} : g(\mathbf{b}) \leq g(\mathbf{b}_{ls})\}$. So asymptotics of the penalty function as $\mathbf{b} \to \infty$ never come in to play. All we care about is that the penalty have certain desirable properties (whatever those may be) in a region near $\mathbf{b}_0$, the true underlying solution. Further, we have a free parameter λ to play with, whose value can be selected as a function of the data $(\mathbf{X}, \mathbf{y})$ in the hope of conferring such desirable properties upon the penalty function.

Note that if we were not allowed to choose λ as a function of the data, but were, instead, required to use some fixed value of λ, say $\lambda = 1$, regardless of the data, then the ridge and lasso methods would not perform well at all. In fact, given the equivalency between penalized linear regression and MAP estimation demonstrated in Section 5.2, the methods would be biased to favor estimates $\hat{\mathbf{b}}$ with particular l^2

and l^1 norms respectively, just as predicted by the theory outlined in Section 4.4 on page 47. Note also that the MDL linear regression formulation does not rely on λ in the same way as do ridge and lasso. Equation (4.18) implies that $\lambda = 2\sigma_Z^2$ is always a reasonable assignment, independent of the data $(\mathbf{X}, \mathbf{y})$.

So, for ridge and the lasso, the ability to scale λ based on the data is essential. It allows us to tune the penalty so that it is neither too big nor too small in a crucial area near the true solution $\mathbf{b}_0$. Since λ is chosen only after looking at the data, the possibility arises that the choice of λ may create an "effective penalty" on $\mathbf{b}$ that is quite different from the apparent l^2 or l^1 penalties being used. This dissertation conjectures that, when methods such as cross-validation for determining λ are taken into account, the λ value chosen effectively scales the penalty such that it approximates, to first order up to an additive constant, an asymptotically optimal coding cost, in the neighborhood of $\mathbf{b}_0$. It is not easy to formalize cross-validation mathematically. However, when other reasonable methods for selecting λ are used, we can show that this is what is going on, at least in the cases of ridge and lasso, though one might easily speculate that the phenomenon is quite general.

To be more precise, let $\lambda(\mathbf{X}, \mathbf{y})$ denote the λ value selected, as a function of the data, by some method such as cross-validation. Let $g(\mathbf{b})$ denote the penalty, not necessarily an asymptotically optimal one, being used in the penalized linear regression. Then the conjecture is that there exists an asymptotically optimal penalty function $\tilde{g}(\mathbf{b})$, such that

$$\lambda(\mathbf{X}, \mathbf{y}) \nabla_{\mathbf{b}} g(\mathbf{b}_0) \approx 2\sigma_Z^2 \nabla_{\mathbf{b}} \tilde{g}(\mathbf{b}_0) \tag{5.4}$$

where the exact nature of the approximation will be made clear in the derivations that follow, and where the term $2\sigma_Z^2$ comes from (4.18) on page 50. Note that two different penalties $\lambda_1 g_1$ and $\lambda_2 g_2$ will yield the same penalized regression solutions if and only if they have the same gradients. To the extent that $\lambda_1 g_1$ and $\lambda_2 g_2$ have approximately equal gradients in a neighborhood, their local minima in that neighborhood, if any, will be approximately the same. The penalties are free to differ by an additive constant, since constants don't affect the minimization over $\mathbf{b}$. This is what is meant by saying that $\lambda_1 g_1$ and $\lambda_2 g_2$ are approximately the same to first order up to an additive constant. Thus, when (5.4) holds in a neighborhood of the true underlying parameter vector, any solution in this neighborhood attained by using the penalty g with $\lambda(\mathbf{X}, \mathbf{y})$ will be close to a solution attained by performing MDL regression with asymptotically optimal penalty $\tilde{g}$.

5.2 MAP Solutions to Ridge and the Lasso

This section describes a method of solving for λ based on the equivalence between penalized linear regression and maximum a posteriori (MAP) estimation. When applied to ridge and the lasso, this method yields log-like equivalent penalties.

It is well-known that, for a fixed λ, and known noise σ_Z^2, penalized linear regression is equivalent to MAP estimation under the assumption that $\mathbf{b}$ is drawn from a prior distribution whose exact form is determined by λ and the penalty function $g(\mathbf{b})$. The derivation is as follows:

$$\hat{\mathbf{b}}(\lambda) = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \lambda g(\mathbf{b}) \tag{5.5}$$

$$= \arg\min_{\mathbf{b}} \frac{1}{2\sigma_Z^2} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \frac{\lambda}{2\sigma_Z^2} g(\mathbf{b}) \tag{5.6}$$

$$= \arg\min_{\mathbf{b}} \frac{1}{2\sigma_Z^2} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \frac{n}{2} \ln 2\pi\sigma_Z^2 + \frac{\lambda}{2\sigma_Z^2} g(\mathbf{b}) \tag{5.7}$$

now, letting $\theta = \lambda / 2\sigma_Z^2$,

$$= \arg\min_{\mathbf{b}} \frac{1}{2\sigma_Z^2} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \frac{n}{2} \ln 2\pi\sigma_Z^2 + \theta g(\mathbf{b}) \tag{5.8}$$

$$= \arg\min_{\mathbf{b}} \frac{1}{2\sigma_Z^2} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \frac{n}{2} \ln 2\pi\sigma_Z^2 + \theta g(\mathbf{b}) - \ln c(\theta) \tag{5.9}$$

where $c(\theta)$ is any arbitrary function of θ,

$$= \arg\min_{\mathbf{b}} - \ln \left(\frac{1}{\sqrt{2\pi\sigma_Z^2}} \right)^n e^{-\frac{1}{2\sigma_Z^2}||\mathbf{y} - \mathbf{X}\mathbf{b}||^2} - \ln \left\{ \frac{1}{c(\theta)} e^{-\theta g(\mathbf{b})} \right\} \tag{5.10}$$

where now $c(\theta)$ is chosen specifically to be $c(\theta) = \int e^{-\theta g(\mathbf{b})} d\mathbf{b}$ so that the expression within the curly braces integrates to 1,

$$= \arg\max_{\mathbf{b}} \left(\frac{1}{\sqrt{2\pi\sigma_Z^2}} \right)^n e^{-\frac{1}{2\sigma_Z^2}||\mathbf{y}-\mathbf{X}\mathbf{b}||^2} \left\{ \frac{1}{c(\theta)} e^{-\theta g(\mathbf{b})} \right\} \tag{5.11}$$

Equation (5.11) now has the form of a MAP estimation, where $\mathbf{b}$ is drawn according to a distribution belonging to an exponential family parameterized by θ, and where, conditional on $\mathbf{b}$, $\mathbf{y}$ is drawn Gaussian with mean $\mathbf{X}\mathbf{b}$ and variance $\sigma_Z \mathbf{I}_n$, like so:

$$f(\mathbf{b}) = \frac{1}{c(\theta)} e^{-\theta g(\mathbf{b})}, \quad \text{and} \quad f(\mathbf{y}|\mathbf{b}) = N(\mathbf{X}\mathbf{b}, \sigma_Z^2 I_n) \tag{5.12}$$

Thus, for every λ, the solution to the penalized regression in (5.5) is the same as the solution to the MAP estimation in (5.11), with $\theta = \lambda/2\sigma_Z^2$. Of course, in the penalized regression form of (5.5), the value of λ is not known, and hence in the MAP form of (5.11) the value of θ is not known. For the penalized regression form, we could use cross-validation or some similar technique to come up with a good value for λ. For the MAP form, another method suggests itself. We could treat θ as a hyperparameter and try to estimate it using maximum likelihood. That is, we could solve

$$(\hat{\mathbf{b}}, \hat{\theta}) = \arg\max_{\mathbf{b},\theta} \left(\frac{1}{\sqrt{2\pi\sigma_Z^2}} \right)^n e^{-\frac{1}{2\sigma_Z^2}||\mathbf{y}-\mathbf{X}\mathbf{b}||^2} \left\{ \frac{1}{c(\theta)} e^{-\theta g(\mathbf{b})} \right\} \tag{5.13}$$

or, equivalently,

$$\hat{\mathbf{b}} = \arg\max_{\mathbf{b}} \left(\frac{1}{\sqrt{2\pi\sigma_Z^2}} \right)^n e^{-\frac{1}{2\sigma_Z^2}||\mathbf{y}-\mathbf{X}\mathbf{b}||^2} \max_{\theta} \left\{ \frac{1}{c(\theta)} e^{-\theta g(\mathbf{b})} \right\} \tag{5.14}$$

Taking the negative log-likelihood form of (5.14) above, we have

$$\hat{\mathbf{b}} = \arg\max_{\mathbf{b}} \left(\frac{1}{\sqrt{2\pi\sigma_Z^2}} \right)^n e^{-\frac{1}{2\sigma_Z^2}||\mathbf{y}-\mathbf{X}\mathbf{b}||^2} \max_{\theta} \left\{ \frac{1}{c(\theta)} e^{-\theta g(\mathbf{b})} \right\} \tag{5.15}$$

$$= \arg\min_{\mathbf{b}} \frac{1}{2\sigma_Z^2}||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \frac{n}{2}\ln 2\pi\sigma_Z^2 + \min_{\theta}\{\theta g(\mathbf{b}) + \ln c(\theta)\} \tag{5.16}$$

$$= \arg\min_{\mathbf{b}} \frac{1}{2\sigma_Z^2}||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + \min_{\theta}\{\theta g(\mathbf{b}) + \ln c(\theta)\} \tag{5.17}$$

$$= \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + 2\sigma_Z^2\min_{\theta}\{\theta g(\mathbf{b}) + \ln c(\theta)\} \tag{5.18}$$

$$= \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + 2\sigma_Z^2 l(\mathbf{b}) \tag{5.19}$$

where $l(\mathbf{b})$ is given by

$$l(\mathbf{b}) \triangleq \min_{\theta}\{\theta g(\mathbf{b}) + \ln c(\theta)\} \tag{5.20}$$

When we solve (5.20) above explicitly for $l(\mathbf{b})$ in the cases of the ridge and lasso penalties, (the derivations are given in sections 5.3 and 5.4), $l(\mathbf{b})$ is seen to have the following log-like forms

$$l_{\text{ridge}}(\mathbf{b}) = p\ln||\mathbf{b}|| + \alpha_1, \quad \text{and} \tag{5.21}$$

$$l_{\text{lasso}}(\mathbf{b}) = p\ln||\mathbf{b}||_1 + \alpha_2 \tag{5.22}$$

respectively, where α_1 and α_2 are constants. Since the constants can be ignored when performing a minimization over $\mathbf{b}$, the MAP method of solving for λ in a penalized linear regression yields ridge and lasso solutions given by

$$\hat{\mathbf{b}}_{\text{ridge,MAP}} = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + 2p\sigma_Z^2\ln||\mathbf{b}||, \quad \text{and} \tag{5.23}$$

$$\hat{\mathbf{b}}_{\text{lasso,MAP}} = \arg\min_{\mathbf{b}} ||\mathbf{y} - \mathbf{X}\mathbf{b}||^2 + 2p\sigma_Z^2\ln||\mathbf{b}||_1 \tag{5.24}$$

respectively. Note that the forms above have spurious non-local minima at $\mathbf{b} = 0$. This occurs because we are using maximum likelihood on pdfs instead of pmfs, yielding an infinite maximum likelihood value of the density when $\mathbf{b} = 0$. If we had used an MDL approach and quantized the domain to create true pmfs, this would not have occurred, although the derivation would have only yielded an approximate correspondence with penalized linear regression. Still, the form of the solutions above indicate that the effective penalties are the relaxed $l'_{\delta,2}$ and $l'_{\delta,1}$ codes described in

Section 4.6 on page 53, corresponding to the code-length functions associated with the l^2 and l^1 codes, and that the proper way to remove the spurious infima is to add a δ term corresponding to some quantization of the domain. This yields the δ-quantized forms described in Section 4.6 as the MDL versions of ridge and lasso

$$\hat{\mathbf{b}}_{\text{ridge,MDL}} = \arg\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2p\sigma_Z^2 \ln \| \, |\mathbf{b}| + \delta \mathbf{1}\|, \quad \text{and} \tag{5.25}$$

$$\hat{\mathbf{b}}_{\text{lasso,MDL}} = \arg\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 + 2p\sigma_Z^2 \ln \| \, |\mathbf{b}| + \delta \mathbf{1}\|_1 \tag{5.26}$$

where $|\mathbf{b}|$ is the vector in the positive orthant whose j^{th} component is $|b_j|$. Recall that the above penalties are asymptotically optimal. Thus, somewhat surprisingly, although ridge and the lasso do not seem to have asymptotically optimal penalties, the MAP method of choosing λ leads to effective penalties that *are* asymptotically optimal, and which have a log-like form reminiscent of the log penalty itself.

Remark 5.1. Note that the penalties derived above are truly asymptotically optimal, which is a stronger assertion than the conjecture of Section 5.1, which merely stated that λ scaled itself such that the effective penalty approximated an asymptotically optimal penalty to first order near $\mathbf{b}_0$. This situation occurs because λ is not really chosen based on the data $(\mathbf{X}, \mathbf{y})$, but, rather, based on the choice of $\mathbf{b}$. This is seen in Equation (5.20), where the minimization over θ depends only on $\mathbf{b}$, not on $(\mathbf{X}, \mathbf{y})$.

Remark 5.2. It is natural to ask what parametric family of distributions arises when the log penalty itself is plugged in to (5.12). One obtains a distribution of the form

$$f_\theta(\mathbf{b}) = \frac{1}{c(\theta)} \cdot \frac{1}{\prod_j (|b_j| + \delta)^\theta} \tag{5.27}$$

which is finitely integrable for $\theta > 1$. It is also natural to wonder what happens when we solve for $l(\mathbf{b})$ in (5.20). Since we are starting out with a log penalty, do we get an effective penalty that is an iterated log? That is, do we get an effective penalty like

$O(\ln \sum(\ln |b_j| + \delta))$? No, the log penalty remains essentially unchanged, albeit with some messy lower-order terms.

One should also remember that L_Z is a code-length function and therefore has an equivalent prior with respect to which it is the Shannon code, given by

$$f(\mathbf{b}) = e^{-L_Z(\mathbf{b})} \tag{5.28}$$

If we use the relaxed function $l_\delta(\mathbf{b}) = \sum(|b_j| + \delta)$ as an approximation of L_Z, we get the improper prior

$$f(\mathbf{b}) = \frac{1}{\prod_j (|b_j| + \delta)} \tag{5.29}$$

The prior is improper because we have thrown away the lower-order terms. If we add back in the $O(\ln \ln ||\mathbf{b}||)$ term, we get something akin to

$$f(\mathbf{b}) = \frac{1}{\prod_j (|b_j| + \delta)(\ln(|b_j| + \delta)^2} \tag{5.30}$$

which is finitely integrable.

5.3 MAP Ridge Solution

Recalling Equation (5.20) on page 77,

$$l(\mathbf{b}) \stackrel{\triangle}{=} \min_\theta \{\theta g(\mathbf{b}) + \ln c(\theta)\}$$

we solve for the explicit form of $l(\mathbf{b})$ when the penalty function is the ridge penalty $g(\mathbf{b}) = ||\mathbf{b}||^2$. Recall also that $c(\theta) = \int e^{-\theta g(\mathbf{b})} d\mathbf{b}$ is a normalizing constant chosen to make the associated exponential family integrate to 1. When $g(\mathbf{b}) = ||\mathbf{b}||^2$, the associated exponential family is a Gaussian family with mean 0 and variance $1/(2\theta)$, given by

$$f_\theta(\mathbf{b}) = \left(\frac{\theta}{\pi}\right)^{\frac{p}{2}} e^{-\theta ||\mathbf{b}||^2} \tag{5.31}$$

Thus, we have

$$c(\theta) = \left(\frac{\pi}{\theta}\right)^{\frac{p}{2}} \tag{5.32}$$

and

$$\begin{aligned} l(\mathbf{b}) &= \arg\min_\theta \theta\|\mathbf{b}\|^2 + \ln c(\theta) &(5.33)\\ &= \arg\min_\theta \theta\|\mathbf{b}\|^2 + \frac{p}{2}\ln\frac{\pi}{\theta} &(5.34)\\ &= \arg\min_\theta \theta\|\mathbf{b}\|^2 - \frac{p}{2}\ln\theta + \frac{p}{2}\ln\pi &(5.35)\end{aligned}$$

Taking derivatives to solve for the minimizing θ yields

$$\begin{aligned} 0 &= \|\mathbf{b}\|^2 - \frac{p}{2\theta} \quad \text{whence} &(5.36)\\ \theta &= \frac{p}{2\|\mathbf{b}\|^2} &(5.37)\end{aligned}$$

Plugging this back in to (5.35) yields

$$\begin{aligned} l(\mathbf{b}) &= \frac{p}{2\|\mathbf{b}\|^2}\cdot\|\mathbf{b}\|^2 - \frac{p}{2}\ln\frac{p}{2\|\mathbf{b}\|^2} + \frac{p}{2}\ln\pi &(5.38)\\ &= \frac{p}{2} + p\ln\|\mathbf{b}\| - \frac{p}{2}\ln\frac{p}{2} + \frac{p}{2}\ln\pi &(5.39)\\ &= p\ln\|\mathbf{b}\| + \alpha &(5.40)\end{aligned}$$

where α is just a constant. Thus, we see that when we solve for λ using the MAP method the resulting effective penalty for ridge regression is $p\ln\|\mathbf{b}\|$, which is asymptotically optimal, rather than $\|\mathbf{b}\|^2$.

5.4 MAP Lasso Solution

Again recalling Equation (5.20) on page 77,

$$l(\mathbf{b}) \overset{\triangle}{=} \min_\theta\{\theta g(\mathbf{b}) + \ln c(\theta)\}$$

we solve for the explicit form of $l(\mathbf{b})$ when the penalty function is the lasso penalty $g(\mathbf{b}) = ||\mathbf{b}||_1$. When $g(\mathbf{b}) = ||\mathbf{b}||_1$, the associated exponential family is a Laplacian family, given by

$$f_\theta(\mathbf{b}) = \left(\frac{\theta}{2}\right)^p e^{-\theta||\mathbf{b}||_1} \tag{5.41}$$

Thus, we have

$$c(\theta) = \left(\frac{2}{\theta}\right)^p \tag{5.42}$$

and

$$\begin{aligned} l(\mathbf{b}) &= \arg\min_\theta \theta||\mathbf{b}||_1 + \ln c(\theta) & (5.43)\\ &= \arg\min_\theta \theta||\mathbf{b}||_1 + p\ln\frac{2}{\theta} & (5.44)\\ &= \arg\min_\theta \theta||\mathbf{b}||_1 - p\ln\theta + p\ln 2 & (5.45)\end{aligned}$$

Taking derivatives to solve for the minimizing θ yields

$$\begin{aligned} 0 &= ||\mathbf{b}||_1 - \frac{p}{\theta} \quad \text{whence} & (5.46)\\ \theta &= \frac{p}{||\mathbf{b}||_1} & (5.47)\end{aligned}$$

Plugging this back in to (5.45) yields

$$\begin{aligned} l(\mathbf{b}) &= \frac{p}{||\mathbf{b}||_1}\cdot||\mathbf{b}||_1 - p\ln\frac{p}{||\mathbf{b}||_1} + p\ln 2 & (5.48)\\ &= p\ln||\mathbf{b}||_1 + \alpha & (5.49)\end{aligned}$$

where α is just a constant. Thus, we see that, when λ is solved for using the MAP method, the resulting effective penalty for lasso regression is $p\ln||\mathbf{b}||_1$, which is asymptotically optimal, rather than $||\mathbf{b}||_1$.

5.5 Generalized C_p Solution To Ridge

In the case of ridge regression, there is another method that can be used to estimate a good value of λ. The method is based on a concept known as *effective degrees of freedom*, which is a generalization of the C_p statistic. Both are discussed in [8]. We will only outline the method here, without giving insights into its derivation.

Let $\hat{\mathbf{y}}_\lambda$ be the *hat vector* associated with the data $\mathbf{y}$ and the parameter λ, given by $\hat{\mathbf{y}}_\lambda \overset{\triangle}{=} \mathbf{X}\hat{\mathbf{b}}(\lambda)$. With certain regression methods, for a given λ, $\hat{\mathbf{y}}_\lambda$ is a linear function of $\mathbf{y}$. That is, $\hat{\mathbf{y}}_\lambda = \mathbf{S}_\lambda \mathbf{y}$ for some matrix $\mathbf{S}_\lambda$ whose value depends only on $\mathbf{X}$ and λ but not on $\mathbf{y}$. This is the case with ridge regression. For ridge, we have

$$\hat{\mathbf{y}}_\lambda = \left\{\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\right\}\mathbf{y} \tag{5.50}$$

$$= \mathbf{S}_\lambda \mathbf{y} \tag{5.51}$$

In those cases, such as ridge, where $\hat{\mathbf{y}}_\lambda = \mathbf{S}_\lambda \mathbf{y}$, the generalized C_p method dictates that the λ value to use is the one satisfying

$$\lambda^* = \arg\min_\lambda \|\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}(\lambda)\|^2 + 2\sigma_Z^2 \operatorname{Tr} \mathbf{S}_\lambda \tag{5.52}$$

We can solve the above equation explicitly for ridge regression in the special case in which $\mathbf{X}$ is orthogonal, i.e., where $\mathbf{X}^T\mathbf{X} = \mathbf{I}$, and show that λ^* scales the original penalty $\|\mathbf{b}\|^2$ such that it effectively agrees with the MDL ridge log penalty (4.46) to first order in the region near $\mathbf{b}_0$.

In the case of ridge, we have

$$\hat{\mathbf{b}}(\lambda) = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \quad \text{and} \tag{5.53}$$

$$\mathbf{S}_\lambda = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T \tag{5.54}$$

The above expressions are derived in [8]. In the case in which $\mathbf{X}^T\mathbf{X} = \mathbf{I}$, they simplify. In (5.53), $\hat{\mathbf{b}}(\lambda)$ simplifies to

$$\hat{\mathbf{b}}(\lambda) = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{5.55}$$

$$= (\mathbf{I} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{5.56}$$

$$= \frac{1}{1+\lambda}\mathbf{X}^T\mathbf{y} \tag{5.57}$$

$$= \frac{\mathbf{b}_{ls}}{1+\lambda} \tag{5.58}$$

where $\mathbf{b}_{ls}$ is the ordinary least squares solution, obtained by substituting $\lambda = 0$ into (5.57). In (5.54), $\mathbf{S}_\lambda$ simplifies to

$$\mathbf{S}_\lambda = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T \tag{5.59}$$

$$= \mathbf{X}(\mathbf{I} + \lambda\mathbf{I})^{-1}\mathbf{X}^T \tag{5.60}$$

$$= \frac{1}{1+\lambda}\mathbf{X}\mathbf{X}^T \tag{5.61}$$

and hence $\operatorname{Tr}\mathbf{S}_\lambda$ equals

$$\operatorname{Tr}\mathbf{S}_\lambda = \operatorname{Tr}\frac{1}{1+\lambda}\mathbf{X}\mathbf{X}^T \tag{5.62}$$

$$= \operatorname{Tr}\frac{1}{1+\lambda}\mathbf{X}^T\mathbf{X} \tag{5.63}$$

$$= \frac{p}{1+\lambda} \tag{5.64}$$

Substituting Equation (5.64) back into (5.52) yields

$$\lambda^* = \arg\min_\lambda \|\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}(\lambda)\|^2 + 2\sigma_Z^2 \operatorname{Tr}\mathbf{S}_\lambda \tag{5.65}$$

$$= \arg\min_\lambda \|\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}(\lambda)\|^2 + \frac{2p\sigma_Z^2}{1+\lambda} \tag{5.66}$$

If λ^* above is a local minimum, then, by taking the derivative of (5.66) with respect to λ and setting it equal to $\mathbf{0}$, we see that λ^* must satisfy

$$0 = (2\mathbf{X}^T\mathbf{X}\hat{\mathbf{b}}(\lambda^*) - 2\mathbf{X}^T\mathbf{y})^T\frac{d\hat{\mathbf{b}}(\lambda^*)}{d\lambda} - \frac{2p\sigma_Z^2}{(1+\lambda^*)^2} \quad \text{which implies} \tag{5.67}$$

$$0 = (\frac{\mathbf{b}_{ls}}{1+\lambda^*} - \mathbf{b}_{ls})^T\frac{d\hat{\mathbf{b}}(\lambda^*)}{d\lambda} - \frac{p\sigma_Z^2}{(1+\lambda^*)^2} \quad \text{leading to} \tag{5.68}$$

$$0 = \frac{-\lambda^*}{1+\lambda^*}\mathbf{b}_{ls}^T \frac{d\hat{\mathbf{b}}(\lambda^*)}{d\lambda} - \frac{p\sigma_Z^2}{(1+\lambda^*)^2} \tag{5.69}$$

In going from (5.67) to (5.68), we have used the substitutions $\mathbf{X}^T\mathbf{X} = \mathbf{I}$ (assumed), $\mathbf{X}^T\mathbf{y} = \mathbf{b}_{ls}$ (5.57 with $\lambda = 0$), and $\hat{\mathbf{b}}(\lambda) = \mathbf{b}_{ls}/(1+\lambda)$ (5.58). From Equation (5.58), we can also calculate $d\hat{\mathbf{b}}(\lambda^*)/d\lambda$. Substituting that back into (5.69) yields

$$\begin{aligned}
0 &= \frac{-\lambda^*}{1+\lambda^*}\mathbf{b}_{ls}^T \frac{-\mathbf{b}_{ls}}{(1+\lambda^*)^2} - \frac{p\sigma_Z^2}{(1+\lambda^*)^2} && (5.70)\\
&= \frac{\lambda^*}{1+\lambda^*}||\mathbf{b}_{ls}||^2 - p\sigma_Z^2 && (5.71)\\
&= \lambda^*||\mathbf{b}_{ls}||^2 - (1+\lambda^*)p\sigma_Z^2 && (5.72)\\
&= \lambda^*||\mathbf{b}_{ls}||^2 - p\sigma_Z^2 - \lambda^* p\sigma_Z^2 \quad \text{which implies} && (5.73)\\
p\sigma_Z^2 &= \lambda^*(||\mathbf{b}_{ls}||^2 - p\sigma_Z^2) \quad \text{leading to} && (5.74)\\
\lambda^* &= \frac{p\sigma_Z^2}{||\mathbf{b}_{ls}||^2 - p\sigma_Z^2} && (5.75)
\end{aligned}$$

Let us now consider the conjecture (5.4) and see whether $\lambda^*||\mathbf{b}||^2$ approximately agrees to first order up to an additive constant with the asymptotically optimal penalty associated with ridge regression in the neighborhood of $\mathbf{b}_0$. We'll use the non-δ quantized form of (5.23) $2p\sigma_Z^2 \ln ||\mathbf{b}||$ for the comparison. According to (5.4), we only need to compare their gradients at $\mathbf{b}_0$. The gradient of $\lambda^*||\mathbf{b}||^2$ is given by

$$\begin{aligned}
\nabla_{\mathbf{b}}\{\lambda^*||\mathbf{b}||^2\} &= \frac{p\sigma_Z^2}{||\mathbf{b}_{ls}||^2 - p\sigma_Z^2}\nabla_{\mathbf{b}}\{||\mathbf{b}||^2\} && (5.76)\\
&= \frac{2p\sigma_Z^2}{||\mathbf{b}_{ls}||^2 - p\sigma_Z^2}\mathbf{b} && (5.77)
\end{aligned}$$

The gradient of $2p\sigma_Z^2 \ln ||\mathbf{b}||$ is

$$\begin{aligned}
\nabla_{\mathbf{b}}\{2p\sigma_Z^2 \ln ||\mathbf{b}||\} &= p\sigma_Z^2 \nabla_{\mathbf{b}}\{\ln ||\mathbf{b}||^2\} && (5.78)\\
&= \frac{2p\sigma_Z^2}{||\mathbf{b}||^2}\mathbf{b} && (5.79)
\end{aligned}$$

We wonder whether equations (5.77) and (5.79) evaluated at $\mathbf{b} = \mathbf{b}_0$ are approximately equal. That is, we wonder whether

$$\frac{2p\sigma_Z^2}{||\mathbf{b}_{ls}||^2 - p\sigma_Z^2}\mathbf{b}_0 \overset{?}{\approx} \frac{2p\sigma_Z^2}{||\mathbf{b}_0||^2}\mathbf{b}_0 \tag{5.80}$$

The terms on the left and right hand sides of (5.80) are now the same, except for the denominators of the fractions, $||\mathbf{b}_{ls}||^2 - p\sigma_Z^2$, on the left hand side, and $||\mathbf{b}_0||^2$ on the right hand side. Note that $\mathbf{b}_{ls}$ on the left hand side is a random variable whose value depends on the data, so we could never expect perfectly equality with the deterministic value on the right hand side. However, the expected value of $\mathbf{b}_{ls}$ (treating $\mathbf{X}$ as fixed and the only randomness as coming from the additive noise $\mathbf{z}$), is $||\mathbf{b}_0||^2 + p\sigma_Z^2$. So (5.80) is approximately true in the sense that

$$\frac{2p\sigma_Z^2}{\mathbf{E}||\mathbf{b}_{ls}||^2 - p\sigma_Z^2}\mathbf{b}_0 = \frac{2p\sigma_Z^2}{||\mathbf{b}_0||^2}\mathbf{b}_0 \tag{5.81}$$

Chapter 6

Experimental Results

This chapter shows the results of two experiments comparing the performance of log-penalized linear regression with the performance of the ridge and lasso methods. The first experiment is on simulated data, where the true underlying model is known. The second experiment is a classification problem on micro-array data.

A general conclusion from these experiments is that the log penalty finds sparser solutions than the lasso and that, when the true underlying model is sparse, the log penalty generally finds a more favorable solution than lasso. However, when the true underlying model is not sparse, the log penalty generally performs worse than lasso, which performs about the same as the ordinary least squares solution.

The experiments all used cross-validation with the one-standard-error rule, as described in sections 2.2 and 2.3, to estimate the optimal values of the free parameter λ, and, in the case of the log penalty, of the free parameters (δ, λ).

6.1 A Large Test-Suite

This test-suite was designed to measure how well the log penalty does against other standard methods as a function of the sparsity-level of the underlying parameter vector. A moderately large predictor set is used, consisting of 50 predictors.

The best way to describe the test-suite is to focus on just one test, at a specific sparsity-level k which means that the underlying parameter vector has only k non-zero components. In order to see how well a particular regression method such as the log penalty does at sparsity-level k, we generate $T = 10$ data sets $(\mathbf{X}_{k,t}, \mathbf{y}_{k,t})$, $t = 1, \ldots, T$ at this sparsity level and average the resulting prediction errors to obtain an estimate of the prediction error that the estimation method incurs. For each t, the data set $(\mathbf{X}_{k,t}, \mathbf{y}_{k,t})$ is generated by first generating a vector $\mathbf{b}_{k,t} \in \mathcal{R}^{50}$ with exactly k non-zero components. Each of the non-zero components is drawn standard normal. Then a 100×50 data matrix $\mathbf{X}_{k,t}$ is generated, with each of its entries also drawn standard normal, and the associated $\mathbf{y}_{k,t} \in \mathcal{R}^{100}$ vector is generated, according to the equation

$$\mathbf{y}_{k,t} = \mathbf{X}_{k,t}\mathbf{b}_{k,t} + \mathbf{z}_{k,t} \tag{6.1}$$

where $\mathbf{z}_{k,t} \in \mathcal{R}^{100}$ is a noise vector, each of whose components is drawn i.i.d. $N(0, 4)$. The regression method in question is then applied individually to each of the T data sets, resulting in estimates $\hat{\mathbf{b}}_{k,t}$, $t = 1, \ldots, T$. The error associated with each estimate is defined to be

$$\begin{aligned}
e_{k,t} &\overset{\triangle}{=} \mathbf{E}_{\mathbf{x}}[(\mathbf{b}_{k,t}^T\mathbf{x} - \hat{\mathbf{b}}_{k,t}^T\mathbf{x}])^2] && (6.2)\\
&= \mathbf{E}_{\mathbf{x}}[(\mathbf{b}_{k,t} - \hat{\mathbf{b}}_{k,t})^T\mathbf{x}\mathbf{x}^T(\mathbf{b}_{k,t} - \hat{\mathbf{b}}_{k,t})] && (6.3)\\
&= (\mathbf{b}_{k,t} - \hat{\mathbf{b}}_{k,t})^T\mathbf{E}_{\mathbf{x}}[\mathbf{x}\mathbf{x}^T](\mathbf{b}_{k,t} - \hat{\mathbf{b}}_{k,t}) && (6.4)\\
&= (\mathbf{b}_{k,t} - \hat{\mathbf{b}}_{k,t})^T I_p(\mathbf{b}_{k,t} - \hat{\mathbf{b}}_{k,t}) && (6.5)\\
&= \|\mathbf{b}_{k,t} - \hat{\mathbf{b}}_{k,t}\|^2 && (6.6)
\end{aligned}$$

where (6.5) follows because, by assumption, the components of $\mathbf{x}$ are drawn i.i.d. standard normal. The estimated error is then just

$$\hat{e}_k \overset{\triangle}{=} \frac{1}{T}\sum_{t=1}^{T} e_{k,t} \tag{6.7}$$

and its standard error is given by

$$s_k = \sqrt{\frac{\sum_{t=1}^{T}(e_{k,t} - e_k)^2}{T(T-1)}} \tag{6.8}$$

The above set-up was tried for the log penalty, ridge, the lasso, and ordinary least squares. For each sparsity-level k, we get an estimate of how well the log penalty performs relative to the other methods. As Figure 6.1 shows, the log penalty does quite well for $k = 1, \ldots, 10$. It had the best performance over the entire range, with the lasso coming in a close second, supporting the conclusion that, when the true underlying model is sparse, the log penalty is good to use.

However, as Figure 6.2 shows, when the underlying parameter vector is not sparse, the log penalty does not do so well. In fact, it clearly has the worst performance over the entire range. Nevertheless, its performance in these non-sparse cases is not grossly worse than that of ordinary least squares. By contrast, the log penalty performs significantly better when the true underlying model is sparse.

Figure 6.3 shows the average sparsity of each method, as a function of k, for $k = 1, \ldots, 10$. The *sparsity* of a vector is the number of non-zero components in the vector. Recall that, for each k, there are ten data sets, leading for each method to ten different estimates with different sparsities. For each method, these ten different sparsities are averaged, which can lead to a non-integral number. The table reveals that the log penalty definitely leads to sparser solutions than the other methods.

In fact, it seems that the log penalty leads to solutions that are too sparse in that they are less sparse than the true underlying parameter vector, while the lasso seems to lead to solutions that are closer in sparsity to that of the underlying parameter vector. However, as Figure 6.4 shows, that is somewhat misleading. Although the lasso seems to lead to solutions with roughly the same sparsity as the true underlying parameter vector, it is not finding the correct non-zero components. Instead, it is positing predictors to be non-zero that are actually zero in the true underlying parameter vector, and vice versa. What we are really interested in is that a sparse solution find the *right* predictors: those that are actually non-zero in the true underlying parameter vectors, as opposed to simply finding the right *number* of predictors. With this in mind, a better measure of the accuracy of each method as regards finding predictors that are truly involved in the underlying model is *sparsity distance*, which we define

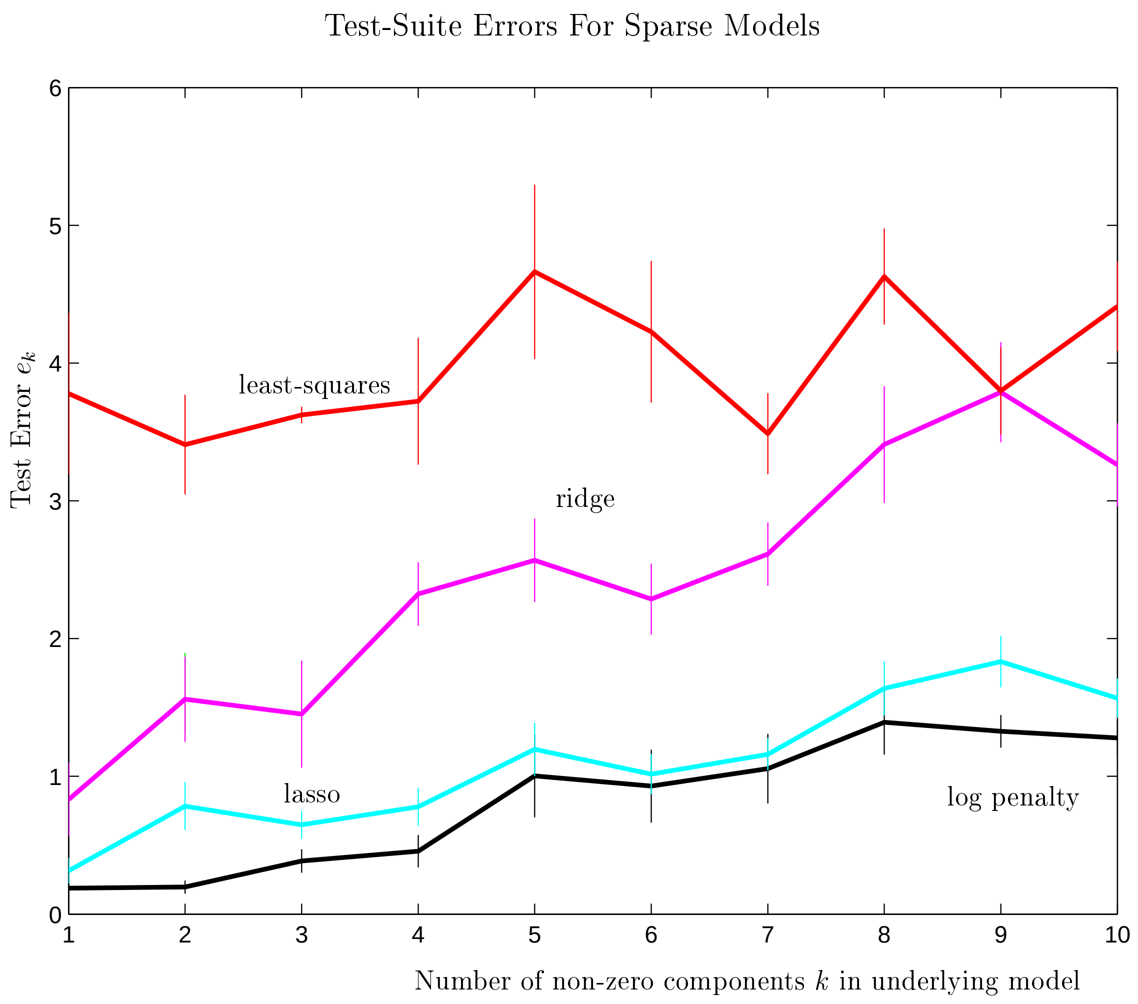

Figure 6.1: *This graph shows the errors associated with the four different regression methods on sparsity levels* $1, \ldots, 10$. *The vertical bars at each data point are standard error bars, showing plus or minus one standard deviation of the estimated error* e_k. *The log penalty does best.*

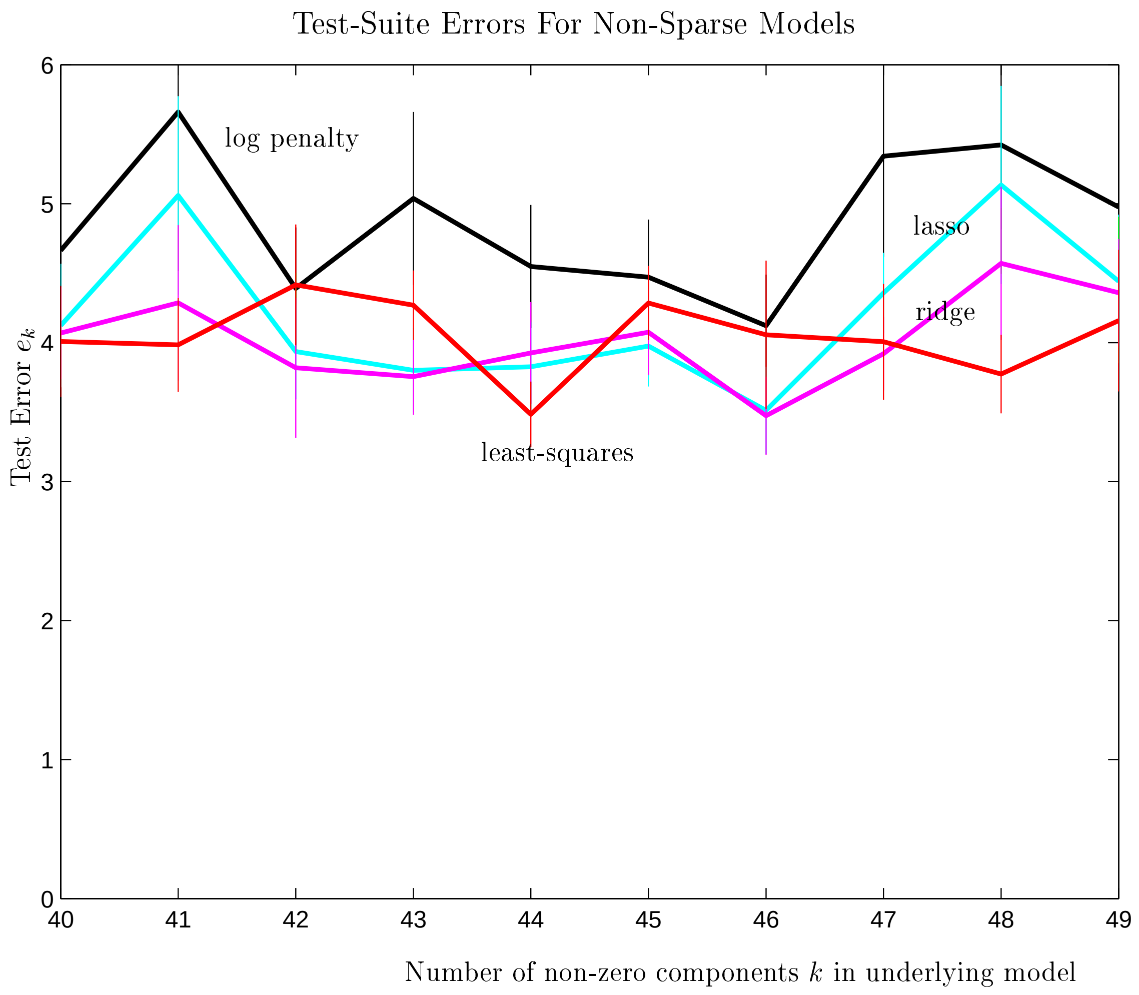

Figure 6.2: *This graph shows the errors associated with the four different regression methods on sparsity levels* $40, \ldots, 49$*. The vertical bars at each data point are standard error bars, showing plus or minus one standard deviation of the estimated error* e_k*. The log penalty does worst.*

Test-Suite Sparsity

True Sparsity	Log Penalty	Lasso	Ridge	Ordinary Least Squares
1	0.4	0.5	50	50
2	1.3	1.3	49.9	50
3	1.4	1.5	50	50
4	2.6	3.5	50	50
5	2.5	5.5	50	50
6	3.3	7.5	50	50
7	2.7	6.7	50	50
8	3.6	6.8	50	50
9	4.2	7.3	50	50
10	6.3	11.5	49.9	50

Figure 6.3: *This table shows the average sparsity of the estimates for each method as a function of k, the sparsity of the true underlying parameter vector, for $k = 1, \ldots, 10$. The* sparsity *of a vector is the number of non-zero components in the vector.*

to be

$$d(\mathbf{b}_1, \mathbf{b}_2) \triangleq \sum_{j=1}^{p} |1(b_{1j} \neq 0) - 1(b_{2j} \neq 0)| \tag{6.9}$$

In terms of sparsity distance, we see from Figure 6.4 that the log penalty generally does a better job at finding the right predictors than do the other methods.

Remark 6.1. The data was not standardized prior to running any of the regression methods. By design, the data was generated zero-mean and of equivalent scale among predictors. Also, the specific log penalty algorithm used was the backward method, as described in algorithm 4.3. All methods used 10-fold cross-validation and the one-standard error rule as described in sections 2.2 and 2.3 to determine which model along the solution path to use. The δ values tried by the log penalty were $\{.01, .001, .0001\}$.

Sparsity Error

True Sparsity	LP	Lasso	Ridge	LS
1	0.6	0.5	49	49
2	1.1	1.1	48	48
3	2.0	1.9	47	47
4	1.8	2.5	46	46
5	2.9	4.9	45	45
6	3.5	6.1	44	44
7	4.3	5.3	43	43
8	4.4	5.6	42	42
9	5.2	5.5	41	41
10	5.3	7.7	40	40

Figure 6.4: *This table shows the average sparsity distance of each method's estimate from the true underlying parameter vector. The sparsity distance counts* 1 *for each non-zero component of the estimate that is zero in the true underlying parameter vector, and* 1 *for each zero component of the estimate that is non-zero in the true underlying parameter vector.*
Example:

$$\begin{aligned} \mathbf{b}_{true} &= (1.2, \; -3.3, \; .7, \; 0, \; 0, \; \ldots, \; 0) \\ \hat{\mathbf{b}} &= (0, \; -1.6, \; .3, \; .2, \; 0, \; \ldots, \; 0) \\ d(\hat{\mathbf{b}}, \mathbf{b}_{true}) &= 2 \end{aligned}$$

6.2 Golub Gene Data

The analysis of micro-array data virtually always involves an overcomplete data set. Micro-arrays are a fairly recent tool in the statistical analysis of the role of genes in determining phenotypic traits. A micro-array measures the expression-levels of a set of designated genes. A gene is *expressed* by being converted into mRNA and eventually into protein. The extent to which a gene is “on” is determined by the magnitude of its expression level. Micro-arrays are capable of measuring several thousand gene expression levels at a time. The genes are the predictors, so micro-array data-sets generally contain several thousand predictors and generally fewer than 100 data points.

The Golub data set [9] is a case in point. It consists of a training set containing gene expression levels of 6088 genes from 38 individuals (the actual number of genes was greater, but many of these resulted in duplicate columns in the data matrix), and a test set containing gene expression levels for the same genes on 34 different individuals. Each of the people involved in the study was identified as having one of two types of leukemia: either acute myeloid leukemia (AML), or acute lymphoblastic leukemia (ALL). The goal was to develop a classifier that distinguished between the two based on gene expression levels.

The log penalty, the lasso, and the minimum l^1 and l^2 norm solutions were all tried on the Golub data set. Nothing fancy was done in the way of modifying the methods to accommodate a classification setting as opposed to a regression setting. The class values AML and ALL were assigned real values of 1 and -1 respectively, and, in the case of the log penalty and lasso, the resulting regression problem was solved using cross-validation with the one-standard error rule (the min l^1 and l^2 solutions have no free parameters to solve for). The resulting model was used to create a classifier in the obvious way: if $\mathbf{b}^T\mathbf{x} > 0$, classify as AML, otherwise classify as ALL. This classifier was then applied to the test set to get an estimate of its classification error.

The results are summarized in table 6.5. There are several comments to make. The first is that the log penalty did indeed find a sparser solution than the lasso. Its solution uses only three genes as predictors, while the lasso’s uses 11 genes, yet both have equally good performance on the test set. (All three genes found by the log penalty were among the 11 found by the lasso.) However, neither do as well as the non-sparse solution yielded by the minimum l^1 norm method (a 38-dimensional solution can't be considered sparse when there are only 38 data points), or the truly non-sparse solution yielded by the minimum l^2 norm method, indicating that, in fact, a great number of genes actually participate in the AML/ALL distinction. That is, despite the excellent classification performance of the three-gene classifier, the assumption of sparsity in the underlying model is probably not valid. This does not by any means invalidate the log-penalty results however. Its job, so to speak, is to find sparse solutions even when the true underlying solution is not sparse, a fact that could only make its job harder. Further, even were it known for certain that the true underlying model were not sparse, a sparse solution might be sought on practical, aesthetic, or explanatory grounds. In this case, the log penalty yields an ultra-sparse solution that is nearly as good as the non-sparse solutions. One could argue that it has done precisely what we want it to do.

Classification Error On Golub Data

Method	Sparsity	Classification Error
Min l^2 Norm	6088	1/34
Min l^1 Norm	38	1/34
Lasso	11	2/34
Log Penalty	3	2/34

Figure 6.5: *This graph shows the classification error associated with the four different regression methods.*

Remark 6.2. The minimum l^1 and l^2 norm solutions are given respectively by

$$\hat{\mathbf{b}}_{l^1} \triangleq \min_{\mathbf{b}} \|\mathbf{b}\|_1 \quad \text{subject to} \quad \mathbf{y} = \mathbf{X}\mathbf{b} \tag{6.10}$$

and

$$\hat{\mathbf{b}}_{l^2} \triangleq \min_{\mathbf{b}} ||\mathbf{b}|| \quad \text{subject to} \quad \mathbf{y} = \mathbf{X}\mathbf{b} \tag{6.11}$$

Remark 6.3. For all methods, the $\mathbf{X}$ matrix was first standardized by normalizing each of its columns to have squared norm n, and augmented by a ones column. The columns were *not* standardized to have mean zero. The backward method, as described in algorithm 4.3, was used for the log penalty solutions. As mentioned in Section 4.10, the log penalty solutions were calculated twice: once with LARS-winnowed pre-processing, and once without, yielding identical results. The δ values tried by the log penalty method were $\{1, .5, .25, .1, .05, .01\}$

The upper graph in Figure 6.6 shows the solution path associated with $\delta = .01$ for the log penalty. The lower graph shows the corresponding estimated test error and actual training error for each solution. Note how the upper graph breaks cleanly into fairly stable regimes. Since the graph is robust relative to perturbations in λ, an ultra-precise method of estimating the optimal λ is not needed. Most of the predictors are driven to zero almost immediately, then a five-predictor solution pops out, followed by a three-predictor solution, followed by a two-predictor solution. Even without a fancy method for choosing the optimal λ value, it is easy to eyeball the graph and see that the three-gene and five-gene classifiers will both do quite well, while the two-gene classifier is not as good. The horizontal red line in the lower graph demarcates the one-standard-error boundary, and the vertical magenta line indicates where the blue estimated error line crosses this boundary, establishing the value of the optimal λ value according to the one-standard-error rule. (Since the log penalty has two free parameters, the one-standard-error line may be determined by, and in this case was determined by, a different δ than the one shown here. It's not possible to look at these graphs and figure out why the line is where it is.)

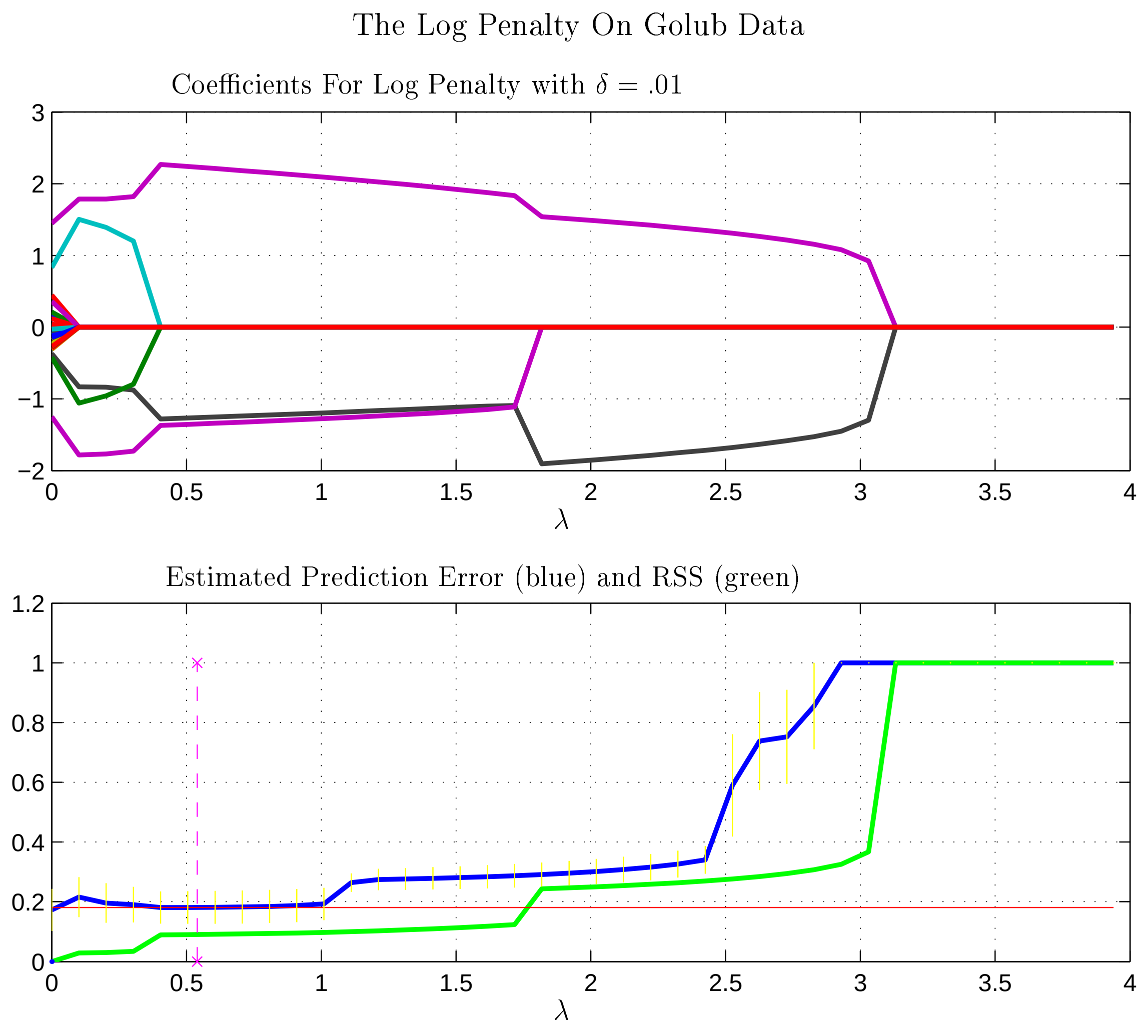

Figure 6.6: *The upper graph is the solution path for log-penalized linear regression applied to the Golub training data for $\delta = .01$. Each vertical slice through a given λ value represents the associated coefficient values of $\hat{\mathbf{b}}_\delta(\lambda)$. All 6088 predictors are represented here, though most of them are zero throughout the entire solution path. The lower graph shows the corresponding training error (RSS) in green and estimated prediction error in blue. The yellow vertical bars at the data points are standard errors associated with the cross-validation estimate of prediction error. The horizontal red line demarcates the one-standard-error boundary, and the vertical magenta line demarcates where the estimated prediction error (blue line) crosses this boundary, indicating the optimal λ value.*

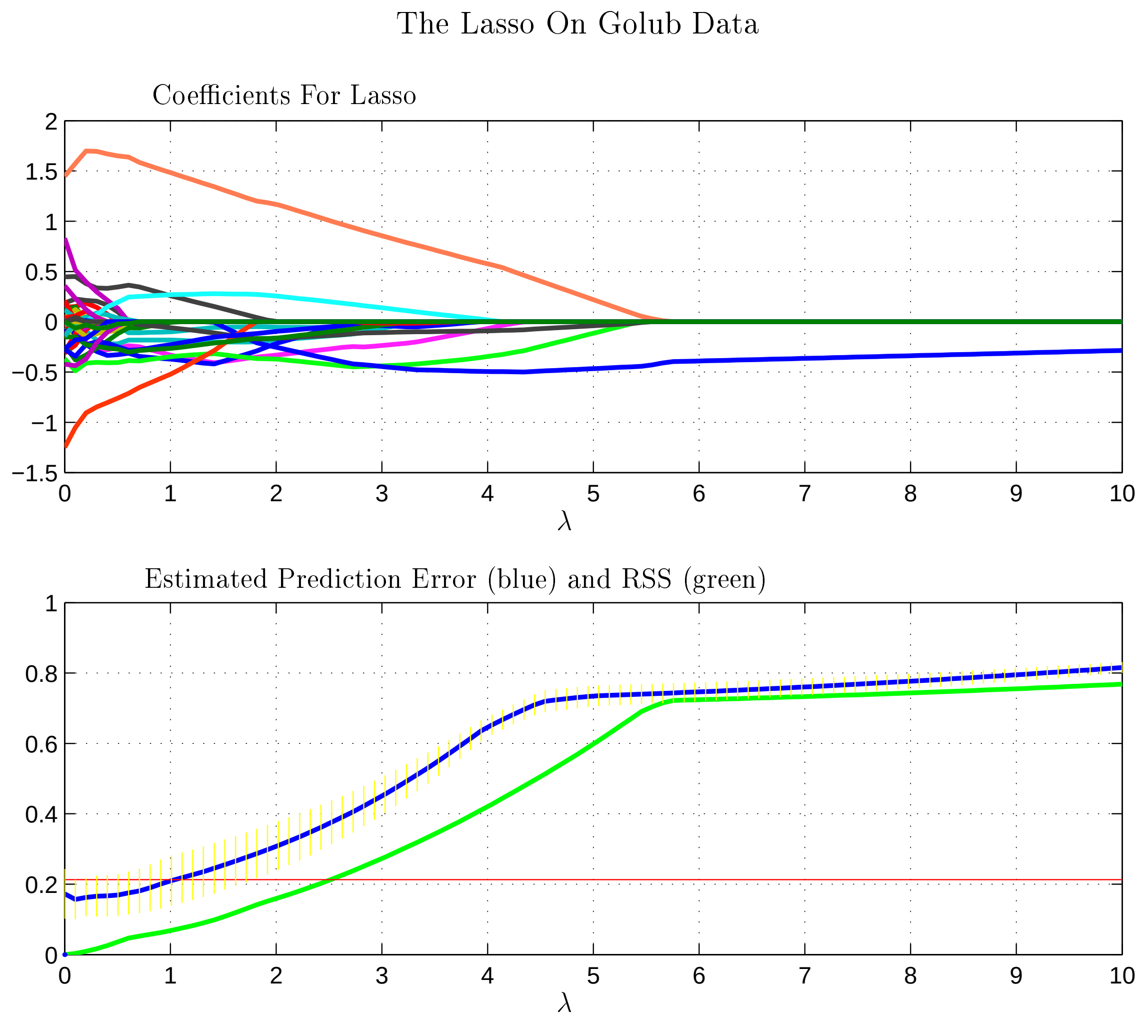

Figure 6.7: *The upper graph is the solution path for the lasso applied to the Golub training data. Each vertical slice through a given λ value represents the associated coefficient values of $\hat{\mathbf{b}}_(\lambda)$. All 6088 predictors are represented here, though most of them are zero throughout the entire solution path. The lower graph shows the corresponding training error (RSS) in green and estimated prediction error in blue. The yellow vertical bars at the data points are standard errors associated with the cross-validation estimate of prediction error. The horizontal red line demarcates the one-standard-error boundary.*

Figure 6.7 shows the lasso solution path. Notice how much smoother it is. Although this a nice property, in some sense, it also makes it much harder to eyeball the results to assess which model might be optimal, and, obviously, it is much more sensitive than the log penalty to the choice of λ.

Figure 6.8 gives a graphical view of the performance of the log penalty's classifier. With the exception of one point, both the training and test data seem quite well separated by this sparse classifier.

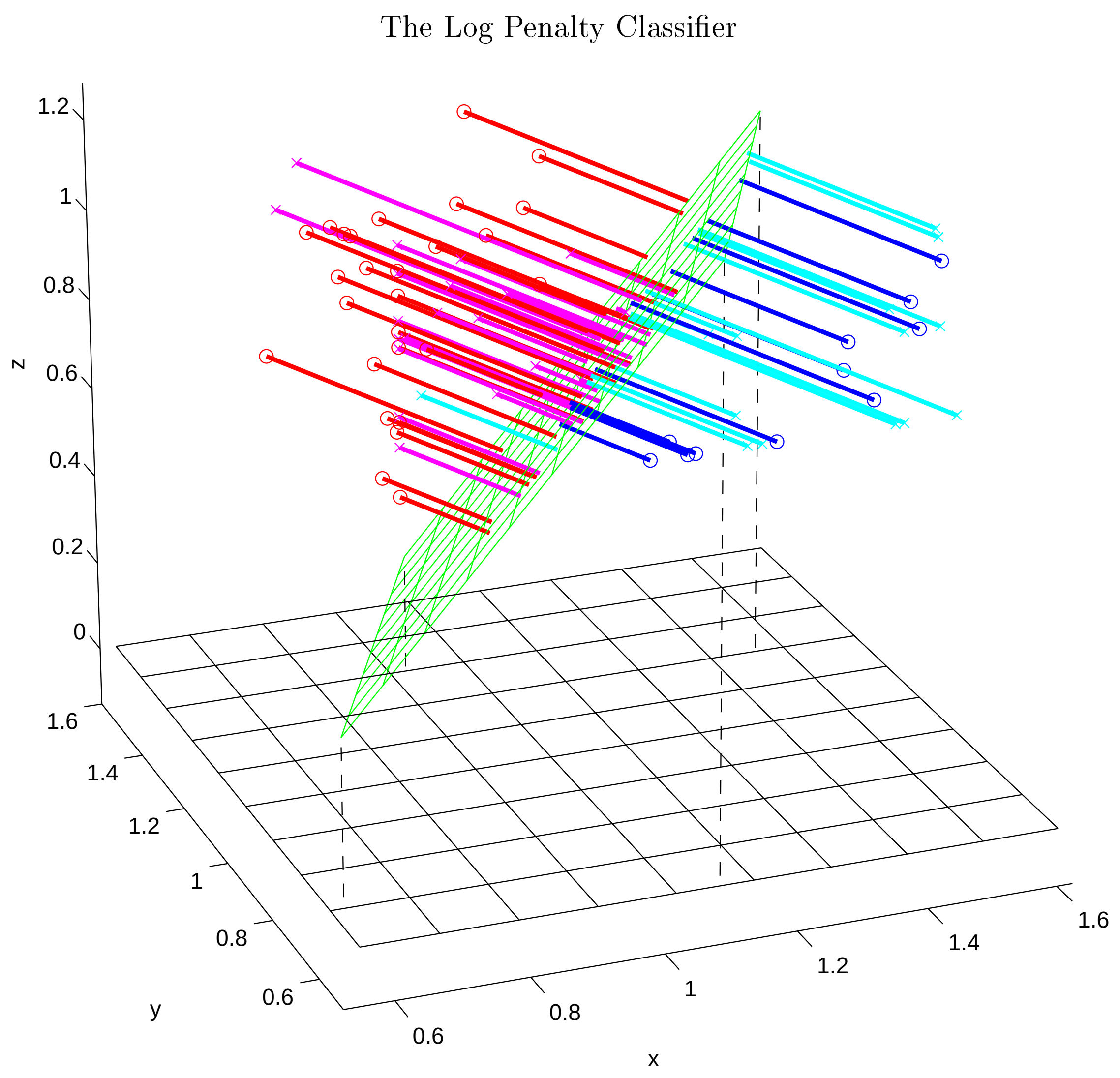

Figure 6.8: *A graph showing the separating plane for the log penalty classifier. Training data from classes AML and ALL are shown as red and blue respectively. Test data from classes AML and ALL are shown as magenta and light blue respectively. One of the two misclassified test points, in light blue, figures prominently on the left side of the plane.*

Chapter 7

Concluding Remarks

As a penalty function, the log penalty at first blush does not seem to have much to recommend itself. Unlike ridge and the lasso, it is not convex, so involving it in a regression minimization leads to multiple minima and to a less tractable optimization problem. Nor does its MAP-equivalent distribution correspond to any well-known parametric family such as the Gaussian and Laplacian families. Undoubtably these are among the reasons it has received little attention in the statistics literature. It is interesting therefore how naturally the log penalty arises when one approaches the regression problem from a complexity perspective. Moreover, when we consider the implications of asymptotic optimality from this perspective, the log penalty attains a kind of primacy, and we find, perhaps surprisingly, that the ridge and lasso penalties themselves can be reinterpreted as having log-like penalties, when the mechanism for estimating λ is taken into account. Indeed, all asymptotically optimal penalties must be log-like penalties in the sense that they must grow no faster than $p \ln ||\mathbf{b}||$ to first order. As such, the theory becomes a kind of unifying principle from which to understand penalized linear regression. If it is the property of asymptotic optimality that makes a penalized regression method impartial enough—in terms of its inherent preference for one kind of model over another— to be of general use, then all reasonable penalties must effectively resolve to log-like penalties.

Despite the interesting theoretical motivation for the log penalty, it would be of no direct practical value if the impediments to efficient solution could not be overcome.

Fortunately, via the proper relaxation from Q^p to $\mathcal{R}^p$, and through application of iterative linearization, we obtain approximate characterizations of log-penalized linear regression that admit tractable solutions. Because the log penalty yields very sparse solutions, sparser than those of the lasso, it may be desirable to use when an ultra-sparse solution is sought for practical or aesthetic reasons, or when it is suspected a priori that the true solution is sparse. In particular, the log penalty may find a niche in the realm of overcomplete regression problems, which is gaining currency as a representational approach. In that regard, the log penalty's performance on the Golub micro-array data is promising, since it finds an excellent three-predictor classifier that performs just as well as the eleven-predictor classifier found by the lasso.

At the theoretical level, some interesting work has been done on overcomplete systems. Donoho's recent paper [5] gives conditions under which the lasso solution is close to the true solution in the presence of noise. The analysis might possibly be leveraged to learn something about the properties of the log penalty in similar circumstances.

Finally, although the complexity approach to estimation as embodied by the MDL principle has stood alongside traditional estimation techniques for awhile, there does not yet seem to be a theoretical treatment that unifies the two approaches. This dissertation has provided perhaps a piece of the puzzle by showing in the special cases of ridge and the lasso how their penalties can be reformulated to reflect asymptotically optimal coding costs, but one might suspect that a stronger relationship exists in general between the standard statistical methods, which judge the goodness of a solution based on its estimated prediction error, and the MDL methods, which judge the goodness of a solution based on its ability to compress data. The exact relationship between complexity and estimation still waits to be found.

Bibliography

[1] H. Akaike, "A new look at the statistical model identification", *IEEE Trans. Aut. Control*, AC-19, pages 716-723, 1974.

[2] Stephen Boyd and Lieven Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[3] G. J. Chaitin, "On the Lengths of Programs for Computing Finite Binary Sequences", *JACM*, volume 13, pages 547-569, 1966.

[4] Thomas M. Cover and Joy A. Thomas, *Elements Of Information Theory*, John Wiley & Sons , New York, 1991.

[5] David Donoho, Michael Elad, and Vladimir Temlyako, *Stable Recovery of Sparse Overcomplete Representations in the Presence of Noise*, Dept. of Statistics Technical Report, Stanford University, 2004.

[6] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani, *Least Angle Regression*, Technical Report 220, Dept. of Statistics, Stanford, 2002.

[7] Maryam Fazel, *Matrix Rank Minimization With Applications*, PhD Dissertation, Stanford University, 2002.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning*, Springer-Verlag, New York, 2001.

[9] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E.

S. Lander, “Classification of cancer: class discovery and class prediction by gene expression monitoring”, *Science Magazine*, Vol 286, Issue 5439, pages 531-537, October 15, 1999.

[10] A. N. Kolmogorov, “Three Approaches to the Quantitative Definition of Information”, *Problems of Information Transmission*, volume 1, pages 4-7, 1965.

[11] Miguel Sousa Lobo, Maryam Fazel, and Stephen Boyd, “Portfolio optimization with linear and fixed transaction costs”, submitted to *Operations Research*, October, 2002.

[12] Irina F. Gorodnitsky, Bhaskar D. Rao, “Sparse Signal Reconstruction from Limited Data Using FOCUSS: A Re-weighted Minimum Norm Algorithm”, *IEEE Transactions On Signal Processing*, volume 45, issue 3, March 1997.

[13] M. R. Osborne, Brett Presnell, and B. A. Turlach, “A new approach to variable selection in least squares problems”, *IMA Journal of Numerical Analysis*, volume 20, pages 389-403, 2000.

[14] M. R. Osborne, Brett Presnell, and B. A. Turlach, “On the lasso and its dual”, *Journal of Computational and Graphical Statistics*, volume 9, issue 2, pages 319-337, 2000.

[15] Jorma Rissanen, *Stochastic Complexity in Statistical Inquiry*, World Scientific Publishing Co. Pte Ltd., New Jersey, 1998.

[16] R. J. Solomonoff, “A Formal Theory of Inductive Inference”, Part I, *Information and Control*, volume 7, pages 1-22; Part II, *Information and Control*, volume 7, pages 224-254, 1964.

[17] Robert Tibshirani, “Regression Shrinkage and Selection via the Lasso”, *Journal of the Royal Statistical Society*, Series B (Methodological), volume 58, issue 1, 267-288, 1996.

[18] Ming Li and Paul Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, New York, 1997.