

Springer Series in Statistics

Trevor Hastie  
Robert Tibshirani  
Jerome Friedman

# The Elements of Statistical Learning

Data Mining, Inference, and Prediction

Second Edition

 Springer

Corrected 12th printing - Jan 13, 2017

*To our parents:*

*Valerie and Patrick Hastie*

*Vera and Sami Tibshirani*

*Florence and Harry Friedman*

*and to our families:*

*Samantha, Timothy, and Lynda*

*Charlie, Ryan, Julie, and Cheryl*

*Melanie, Dora, Monika, and Ildiko*



# Preface to the Second Edition

*In God we trust, all others bring data.*

—William Edwards Deming (1900-1993)<sup>1</sup>

We have been gratified by the popularity of the first edition of *The Elements of Statistical Learning*. This, along with the fast pace of research in the statistical learning field, motivated us to update our book with a second edition.

We have added four new chapters and updated some of the existing chapters. Because many readers are familiar with the layout of the first edition, we have tried to change it as little as possible. Here is a summary of the main changes:

---

<sup>1</sup>On the Web, this quote has been widely attributed to both Deming and Robert W. Hayden; however Professor Hayden told us that he can claim no credit for this quote, and ironically we could find no “data” confirming that Deming actually said this.

Chapter	What's new
<b>1.</b> Introduction	
<b>2.</b> Overview of Supervised Learning	
<b>3.</b> Linear Methods for Regression	LAR algorithm and generalizations of the lasso
<b>4.</b> Linear Methods for Classification	Lasso path for logistic regression
<b>5.</b> Basis Expansions and Regularization	Additional illustrations of RKHS
<b>6.</b> Kernel Smoothing Methods	
<b>7.</b> Model Assessment and Selection	Strengths and pitfalls of cross-validation
<b>8.</b> Model Inference and Averaging	
<b>9.</b> Additive Models, Trees, and Related Methods	
<b>10.</b> Boosting and Additive Trees	New example from ecology; some material split off to Chapter 16.
<b>11.</b> Neural Networks	Bayesian neural nets and the NIPS 2003 challenge
<b>12.</b> Support Vector Machines and Flexible Discriminants	Path algorithm for SVM classifier
<b>13.</b> Prototype Methods and Nearest-Neighbors	
<b>14.</b> Unsupervised Learning	Spectral clustering, kernel PCA, sparse PCA, non-negative matrix factorization, archetypal analysis, nonlinear dimension reduction, Google page rank algorithm, a direct approach to ICA
<b>15.</b> Random Forests	New
<b>16.</b> Ensemble Learning	New
<b>17.</b> Undirected Graphical Models	New
<b>18.</b> High-Dimensional Problems	New

Some further notes:

- Our first edition was unfriendly to colorblind readers; in particular, we tended to favor red/green contrasts which are particularly troublesome. We have changed the color palette in this edition to a large extent, replacing the above with an orange/blue contrast.
- We have changed the name of Chapter 6 from “Kernel Methods” to “Kernel Smoothing Methods”, to avoid confusion with the machine-learning kernel method that is discussed in the context of support vector machines (Chapter 12) and more generally in Chapters 5 and 14.
- In the first edition, the discussion of error-rate estimation in Chapter 7 was sloppy, as we did not clearly differentiate the notions of conditional error rates (conditional on the training set) and unconditional rates. We have fixed this in the new edition.

- Chapters 15 and 16 follow naturally from Chapter 10, and the chapters are probably best read in that order.
- In Chapter 17, we have not attempted a comprehensive treatment of graphical models, and discuss only undirected models and some new methods for their estimation. Due to a lack of space, we have specifically omitted coverage of directed graphical models.
- Chapter 18 explores the “ $p \gg N$ ” problem, which is learning in high-dimensional feature spaces. These problems arise in many areas, including genomic and proteomic studies, and document classification.

We thank the many readers who have found the (too numerous) errors in the first edition. We apologize for those and have done our best to avoid errors in this new edition. We thank Mark Segal, Bala Rajaratnam, and Larry Wasserman for comments on some of the new chapters, and many Stanford graduate and post-doctoral students who offered comments, in particular Mohammed AlQuraishi, John Boik, Holger Hoeffling, Arian Maleki, Donal McMahon, Saharon Rosset, Babak Shababa, Daniela Witten, Ji Zhu and Hui Zou. We thank John Kimmel for his patience in guiding us through this new edition. RT dedicates this edition to the memory of Anna McPhee.

*Trevor Hastie*  
*Robert Tibshirani*  
*Jerome Friedman*

Stanford, California  
 August 2008



# Preface to the First Edition

*We are drowning in information and starving for knowledge.*

—Rutherford D. Roger

The field of Statistics is constantly challenged by the problems that science and industry brings to its door. In the early days, these problems often came from agricultural and industrial experiments and were relatively small in scope. With the advent of computers and the information age, statistical problems have exploded both in size and complexity. Challenges in the areas of data storage, organization and searching have led to the new field of “data mining”; statistical and computational problems in biology and medicine have created “bioinformatics.” Vast amounts of data are being generated in many fields, and the statistician’s job is to make sense of it all: to extract important patterns and trends, and understand “what the data says.” We call this *learning from data*.

The challenges in learning from data have led to a revolution in the statistical sciences. Since computation plays such a key role, it is not surprising that much of this new development has been done by researchers in other fields such as computer science and engineering.

The learning problems that we consider can be roughly categorized as either *supervised* or *unsupervised*. In supervised learning, the goal is to predict the value of an outcome measure based on a number of input measures; in unsupervised learning, there is no outcome measure, and the goal is to describe the associations and patterns among a set of input measures.



This book is our attempt to bring together many of the important new ideas in learning, and explain them in a statistical framework. While some mathematical details are needed, we emphasize the methods and their conceptual underpinnings rather than their theoretical properties. As a result, we hope that this book will appeal not just to statisticians but also to researchers and practitioners in a wide variety of fields.

Just as we have learned a great deal from researchers outside of the field of statistics, our statistical viewpoint may help others to better understand different aspects of learning:

*There is no true interpretation of anything; interpretation is a vehicle in the service of human comprehension. The value of interpretation is in enabling others to fruitfully think about an idea.*

—Andreas Buja

We would like to acknowledge the contribution of many people to the conception and completion of this book. David Andrews, Leo Breiman, Andreas Buja, John Chambers, Bradley Efron, Geoffrey Hinton, Werner Stuetzle, and John Tukey have greatly influenced our careers. Balasubramanian Narasimhan gave us advice and help on many computational problems, and maintained an excellent computing environment. Shin-Ho Bang helped in the production of a number of the figures. Lee Wilkinson gave valuable tips on color production. Ilana Belitskaya, Eva Cantoni, Maya Gupta, Michael Jordan, Shanti Gopatam, Radford Neal, Jorge Picazo, Bogdan Popescu, Olivier Renaud, Saharon Rosset, John Storey, Ji Zhu, Mu Zhu, two reviewers and many students read parts of the manuscript and offered helpful suggestions. John Kimmel was supportive, patient and helpful at every phase; MaryAnn Brickner and Frank Ganz headed a superb production team at Springer. Trevor Hastie would like to thank the statistics department at the University of Cape Town for their hospitality during the final stages of this book. We gratefully acknowledge NSF and NIH for their support of this work. Finally, we would like to thank our families and our parents for their love and support.

*Trevor Hastie  
Robert Tibshirani  
Jerome Friedman*

Stanford, California  
May 2001

*The quiet statisticians have changed our world; not by discovering new facts or technical developments, but by changing the ways that we reason, experiment and form our opinions ....*

—Ian Hacking

# Contents

Preface to the Second Edition	vii
Preface to the First Edition	xi
1 Introduction	1
2 Overview of Supervised Learning	9
2.1 Introduction . . . . .	9
2.2 Variable Types and Terminology . . . . .	9
2.3 Two Simple Approaches to Prediction: Least Squares and Nearest Neighbors . . . . .	11
2.3.1 Linear Models and Least Squares . . . . .	11
2.3.2 Nearest-Neighbor Methods . . . . .	14
2.3.3 From Least Squares to Nearest Neighbors . . . . .	16
2.4 Statistical Decision Theory . . . . .	18
2.5 Local Methods in High Dimensions . . . . .	22
2.6 Statistical Models, Supervised Learning and Function Approximation . . . . .	28
2.6.1 A Statistical Model for the Joint Distribution $\Pr(X, Y)$ . . . . .	28
2.6.2 Supervised Learning . . . . .	29
2.6.3 Function Approximation . . . . .	29
2.7 Structured Regression Models . . . . .	32
2.7.1 Difficulty of the Problem . . . . .	32

2.8	Classes of Restricted Estimators . . . . .	33
2.8.1	Roughness Penalty and Bayesian Methods . . .	34
2.8.2	Kernel Methods and Local Regression . . . . .	34
2.8.3	Basis Functions and Dictionary Methods . . . .	35
2.9	Model Selection and the Bias–Variance Tradeoff . . . . .	37
	Bibliographic Notes . . . . .	39
	Exercises . . . . .	39
<b>3</b>	<b>Linear Methods for Regression</b>	<b>43</b>
3.1	Introduction . . . . .	43
3.2	Linear Regression Models and Least Squares . . . . .	44
3.2.1	Example: Prostate Cancer . . . . .	49
3.2.2	The Gauss–Markov Theorem . . . . .	51
3.2.3	Multiple Regression from Simple Univariate Regression . . . . .	52
3.2.4	Multiple Outputs . . . . .	56
3.3	Subset Selection . . . . .	57
3.3.1	Best-Subset Selection . . . . .	57
3.3.2	Forward- and Backward-Stepwise Selection . . .	58
3.3.3	Forward-Stagewise Regression . . . . .	60
3.3.4	Prostate Cancer Data Example (Continued) . .	61
3.4	Shrinkage Methods . . . . .	61
3.4.1	Ridge Regression . . . . .	61
3.4.2	The Lasso . . . . .	68
3.4.3	Discussion: Subset Selection, Ridge Regression and the Lasso . . . . .	69
3.4.4	Least Angle Regression . . . . .	73
3.5	Methods Using Derived Input Directions . . . . .	79
3.5.1	Principal Components Regression . . . . .	79
3.5.2	Partial Least Squares . . . . .	80
3.6	Discussion: A Comparison of the Selection and Shrinkage Methods . . . . .	82
3.7	Multiple Outcome Shrinkage and Selection . . . . .	84
3.8	More on the Lasso and Related Path Algorithms . . . . .	86
3.8.1	Incremental Forward Stagewise Regression . . .	86
3.8.2	Piecewise-Linear Path Algorithms . . . . .	89
3.8.3	The Dantzig Selector . . . . .	89
3.8.4	The Grouped Lasso . . . . .	90
3.8.5	Further Properties of the Lasso . . . . .	91
3.8.6	Pathwise Coordinate Optimization . . . . .	92
3.9	Computational Considerations . . . . .	93
	Bibliographic Notes . . . . .	94
	Exercises . . . . .	94

<b>4</b>	<b>Linear Methods for Classification</b>	<b>101</b>
4.1	Introduction . . . . .	101
4.2	Linear Regression of an Indicator Matrix . . . . .	103
4.3	Linear Discriminant Analysis . . . . .	106
4.3.1	Regularized Discriminant Analysis . . . . .	112
4.3.2	Computations for LDA . . . . .	113
4.3.3	Reduced-Rank Linear Discriminant Analysis . . . . .	113
4.4	Logistic Regression . . . . .	119
4.4.1	Fitting Logistic Regression Models . . . . .	120
4.4.2	Example: South African Heart Disease . . . . .	122
4.4.3	Quadratic Approximations and Inference . . . . .	124
4.4.4	$L_1$ Regularized Logistic Regression . . . . .	125
4.4.5	Logistic Regression or LDA? . . . . .	127
4.5	Separating Hyperplanes . . . . .	129
4.5.1	Rosenblatt's Perceptron Learning Algorithm . . . . .	130
4.5.2	Optimal Separating Hyperplanes . . . . .	132
	Bibliographic Notes . . . . .	135
	Exercises . . . . .	135
<b>5</b>	<b>Basis Expansions and Regularization</b>	<b>139</b>
5.1	Introduction . . . . .	139
5.2	Piecewise Polynomials and Splines . . . . .	141
5.2.1	Natural Cubic Splines . . . . .	144
5.2.2	Example: South African Heart Disease (Continued) . . . . .	146
5.2.3	Example: Phoneme Recognition . . . . .	148
5.3	Filtering and Feature Extraction . . . . .	150
5.4	Smoothing Splines . . . . .	151
5.4.1	Degrees of Freedom and Smoother Matrices . . . . .	153
5.5	Automatic Selection of the Smoothing Parameters . . . . .	156
5.5.1	Fixing the Degrees of Freedom . . . . .	158
5.5.2	The Bias–Variance Tradeoff . . . . .	158
5.6	Nonparametric Logistic Regression . . . . .	161
5.7	Multidimensional Splines . . . . .	162
5.8	Regularization and Reproducing Kernel Hilbert Spaces . . . . .	167
5.8.1	Spaces of Functions Generated by Kernels . . . . .	168
5.8.2	Examples of RKHS . . . . .	170
5.9	Wavelet Smoothing . . . . .	174
5.9.1	Wavelet Bases and the Wavelet Transform . . . . .	176
5.9.2	Adaptive Wavelet Filtering . . . . .	179
	Bibliographic Notes . . . . .	181
	Exercises . . . . .	181
	Appendix: Computational Considerations for Splines . . . . .	186
	Appendix: $B$ -splines . . . . .	186
	Appendix: Computations for Smoothing Splines . . . . .	189

<b>6</b>	<b>Kernel Smoothing Methods</b>	<b>191</b>
6.1	One-Dimensional Kernel Smoothers . . . . .	192
6.1.1	Local Linear Regression . . . . .	194
6.1.2	Local Polynomial Regression . . . . .	197
6.2	Selecting the Width of the Kernel . . . . .	198
6.3	Local Regression in $\mathbb{R}^p$ . . . . .	200
6.4	Structured Local Regression Models in $\mathbb{R}^p$ . . . . .	201
6.4.1	Structured Kernels . . . . .	203
6.4.2	Structured Regression Functions . . . . .	203
6.5	Local Likelihood and Other Models . . . . .	205
6.6	Kernel Density Estimation and Classification . . . . .	208
6.6.1	Kernel Density Estimation . . . . .	208
6.6.2	Kernel Density Classification . . . . .	210
6.6.3	The Naive Bayes Classifier . . . . .	210
6.7	Radial Basis Functions and Kernels . . . . .	212
6.8	Mixture Models for Density Estimation and Classification . . . . .	214
6.9	Computational Considerations . . . . .	216
	Bibliographic Notes . . . . .	216
	Exercises . . . . .	216
<b>7</b>	<b>Model Assessment and Selection</b>	<b>219</b>
7.1	Introduction . . . . .	219
7.2	Bias, Variance and Model Complexity . . . . .	219
7.3	The Bias–Variance Decomposition . . . . .	223
7.3.1	Example: Bias–Variance Tradeoff . . . . .	226
7.4	Optimism of the Training Error Rate . . . . .	228
7.5	Estimates of In-Sample Prediction Error . . . . .	230
7.6	The Effective Number of Parameters . . . . .	232
7.7	The Bayesian Approach and BIC . . . . .	233
7.8	Minimum Description Length . . . . .	235
7.9	Vapnik–Chervonenkis Dimension . . . . .	237
7.9.1	Example (Continued) . . . . .	239
7.10	Cross-Validation . . . . .	241
7.10.1	$K$ -Fold Cross-Validation . . . . .	241
7.10.2	The Wrong and Right Way to Do Cross-validation . . . . .	245
7.10.3	Does Cross-Validation Really Work? . . . . .	247
7.11	Bootstrap Methods . . . . .	249
7.11.1	Example (Continued) . . . . .	252
7.12	Conditional or Expected Test Error? . . . . .	254
	Bibliographic Notes . . . . .	257
	Exercises . . . . .	257
<b>8</b>	<b>Model Inference and Averaging</b>	<b>261</b>
8.1	Introduction . . . . .	261

8.2	The Bootstrap and Maximum Likelihood Methods . . . .	261
8.2.1	A Smoothing Example . . . . .	261
8.2.2	Maximum Likelihood Inference . . . . .	265
8.2.3	Bootstrap versus Maximum Likelihood . . . . .	267
8.3	Bayesian Methods . . . . .	267
8.4	Relationship Between the Bootstrap and Bayesian Inference . . . . .	271
8.5	The EM Algorithm . . . . .	272
8.5.1	Two-Component Mixture Model . . . . .	272
8.5.2	The EM Algorithm in General . . . . .	276
8.5.3	EM as a Maximization–Maximization Procedure . . . . .	277
8.6	MCMC for Sampling from the Posterior . . . . .	279
8.7	Bagging . . . . .	282
8.7.1	Example: Trees with Simulated Data . . . . .	283
8.8	Model Averaging and Stacking . . . . .	288
8.9	Stochastic Search: Bumping . . . . .	290
	Bibliographic Notes . . . . .	292
	Exercises . . . . .	293
<b>9</b>	<b>Additive Models, Trees, and Related Methods</b>	<b>295</b>
9.1	Generalized Additive Models . . . . .	295
9.1.1	Fitting Additive Models . . . . .	297
9.1.2	Example: Additive Logistic Regression . . . . .	299
9.1.3	Summary . . . . .	304
9.2	Tree-Based Methods . . . . .	305
9.2.1	Background . . . . .	305
9.2.2	Regression Trees . . . . .	307
9.2.3	Classification Trees . . . . .	308
9.2.4	Other Issues . . . . .	310
9.2.5	Spam Example (Continued) . . . . .	313
9.3	PRIM: Bump Hunting . . . . .	317
9.3.1	Spam Example (Continued) . . . . .	320
9.4	MARS: Multivariate Adaptive Regression Splines . . . . .	321
9.4.1	Spam Example (Continued) . . . . .	326
9.4.2	Example (Simulated Data) . . . . .	327
9.4.3	Other Issues . . . . .	328
9.5	Hierarchical Mixtures of Experts . . . . .	329
9.6	Missing Data . . . . .	332
9.7	Computational Considerations . . . . .	334
	Bibliographic Notes . . . . .	334
	Exercises . . . . .	335
<b>10</b>	<b>Boosting and Additive Trees</b>	<b>337</b>
10.1	Boosting Methods . . . . .	337
10.1.1	Outline of This Chapter . . . . .	340

10.2	Boosting Fits an Additive Model . . . . .	341
10.3	Forward Stagewise Additive Modeling . . . . .	342
10.4	Exponential Loss and AdaBoost . . . . .	343
10.5	Why Exponential Loss? . . . . .	345
10.6	Loss Functions and Robustness . . . . .	346
10.7	“Off-the-Shelf” Procedures for Data Mining . . . . .	350
10.8	Example: Spam Data . . . . .	352
10.9	Boosting Trees . . . . .	353
10.10	Numerical Optimization via Gradient Boosting . . . . .	358
10.10.1	Steepest Descent . . . . .	358
10.10.2	Gradient Boosting . . . . .	359
10.10.3	Implementations of Gradient Boosting . . . . .	360
10.11	Right-Sized Trees for Boosting . . . . .	361
10.12	Regularization . . . . .	364
10.12.1	Shrinkage . . . . .	364
10.12.2	Subsampling . . . . .	365
10.13	Interpretation . . . . .	367
10.13.1	Relative Importance of Predictor Variables . . . . .	367
10.13.2	Partial Dependence Plots . . . . .	369
10.14	Illustrations . . . . .	371
10.14.1	California Housing . . . . .	371
10.14.2	New Zealand Fish . . . . .	375
10.14.3	Demographics Data . . . . .	379
	Bibliographic Notes . . . . .	380
	Exercises . . . . .	384

## **11 Neural Networks 389**

11.1	Introduction . . . . .	389
11.2	Projection Pursuit Regression . . . . .	389
11.3	Neural Networks . . . . .	392
11.4	Fitting Neural Networks . . . . .	395
11.5	Some Issues in Training Neural Networks . . . . .	397
11.5.1	Starting Values . . . . .	397
11.5.2	Overfitting . . . . .	398
11.5.3	Scaling of the Inputs . . . . .	398
11.5.4	Number of Hidden Units and Layers . . . . .	400
11.5.5	Multiple Minima . . . . .	400
11.6	Example: Simulated Data . . . . .	401
11.7	Example: ZIP Code Data . . . . .	404
11.8	Discussion . . . . .	408
11.9	Bayesian Neural Nets and the NIPS 2003 Challenge . . . . .	409
11.9.1	Bayes, Boosting and Bagging . . . . .	410
11.9.2	Performance Comparisons . . . . .	412
11.10	Computational Considerations . . . . .	414
	Bibliographic Notes . . . . .	415

Exercises . . . . .	415
<b>12 Support Vector Machines and Flexible Discriminants</b>	<b>417</b>
12.1 Introduction . . . . .	417
12.2 The Support Vector Classifier . . . . .	417
12.2.1 Computing the Support Vector Classifier . . . . .	420
12.2.2 Mixture Example (Continued) . . . . .	421
12.3 Support Vector Machines and Kernels . . . . .	423
12.3.1 Computing the SVM for Classification . . . . .	423
12.3.2 The SVM as a Penalization Method . . . . .	426
12.3.3 Function Estimation and Reproducing Kernels . . . . .	428
12.3.4 SVMs and the Curse of Dimensionality . . . . .	431
12.3.5 A Path Algorithm for the SVM Classifier . . . . .	432
12.3.6 Support Vector Machines for Regression . . . . .	434
12.3.7 Regression and Kernels . . . . .	436
12.3.8 Discussion . . . . .	438
12.4 Generalizing Linear Discriminant Analysis . . . . .	438
12.5 Flexible Discriminant Analysis . . . . .	440
12.5.1 Computing the FDA Estimates . . . . .	444
12.6 Penalized Discriminant Analysis . . . . .	446
12.7 Mixture Discriminant Analysis . . . . .	449
12.7.1 Example: Waveform Data . . . . .	451
Bibliographic Notes . . . . .	455
Exercises . . . . .	455
<b>13 Prototype Methods and Nearest-Neighbors</b>	<b>459</b>
13.1 Introduction . . . . .	459
13.2 Prototype Methods . . . . .	459
13.2.1 $K$ -means Clustering . . . . .	460
13.2.2 Learning Vector Quantization . . . . .	462
13.2.3 Gaussian Mixtures . . . . .	463
13.3 $k$ -Nearest-Neighbor Classifiers . . . . .	463
13.3.1 Example: A Comparative Study . . . . .	468
13.3.2 Example: $k$ -Nearest-Neighbors and Image Scene Classification . . . . .	470
13.3.3 Invariant Metrics and Tangent Distance . . . . .	471
13.4 Adaptive Nearest-Neighbor Methods . . . . .	475
13.4.1 Example . . . . .	478
13.4.2 Global Dimension Reduction for Nearest-Neighbors . . . . .	479
13.5 Computational Considerations . . . . .	480
Bibliographic Notes . . . . .	481
Exercises . . . . .	481



<b>14 Unsupervised Learning</b>	<b>485</b>
14.1 Introduction	485
14.2 Association Rules	487
14.2.1 Market Basket Analysis	488
14.2.2 The Apriori Algorithm	489
14.2.3 Example: Market Basket Analysis	492
14.2.4 Unsupervised as Supervised Learning	495
14.2.5 Generalized Association Rules	497
14.2.6 Choice of Supervised Learning Method	499
14.2.7 Example: Market Basket Analysis (Continued)	499
14.3 Cluster Analysis	501
14.3.1 Proximity Matrices	503
14.3.2 Dissimilarities Based on Attributes	503
14.3.3 Object Dissimilarity	505
14.3.4 Clustering Algorithms	507
14.3.5 Combinatorial Algorithms	507
14.3.6 $K$ -means	509
14.3.7 Gaussian Mixtures as Soft $K$ -means Clustering	510
14.3.8 Example: Human Tumor Microarray Data	512
14.3.9 Vector Quantization	514
14.3.10 $K$ -medoids	515
14.3.11 Practical Issues	518
14.3.12 Hierarchical Clustering	520
14.4 Self-Organizing Maps	528
14.5 Principal Components, Curves and Surfaces	534
14.5.1 Principal Components	534
14.5.2 Principal Curves and Surfaces	541
14.5.3 Spectral Clustering	544
14.5.4 Kernel Principal Components	547
14.5.5 Sparse Principal Components	550
14.6 Non-negative Matrix Factorization	553
14.6.1 Archetypal Analysis	554
14.7 Independent Component Analysis and Exploratory Projection Pursuit	557
14.7.1 Latent Variables and Factor Analysis	558
14.7.2 Independent Component Analysis	560
14.7.3 Exploratory Projection Pursuit	565
14.7.4 A Direct Approach to ICA	565
14.8 Multidimensional Scaling	570
14.9 Nonlinear Dimension Reduction and Local Multidimensional Scaling	572
14.10 The Google PageRank Algorithm	576
Bibliographic Notes	578
Exercises	579

<b>15 Random Forests</b>	<b>587</b>
15.1 Introduction . . . . .	587
15.2 Definition of Random Forests . . . . .	587
15.3 Details of Random Forests . . . . .	592
15.3.1 Out of Bag Samples . . . . .	592
15.3.2 Variable Importance . . . . .	593
15.3.3 Proximity Plots . . . . .	595
15.3.4 Random Forests and Overfitting . . . . .	596
15.4 Analysis of Random Forests . . . . .	597
15.4.1 Variance and the De-Correlation Effect . . . . .	597
15.4.2 Bias . . . . .	600
15.4.3 Adaptive Nearest Neighbors . . . . .	601
Bibliographic Notes . . . . .	602
Exercises . . . . .	603
 <b>16 Ensemble Learning</b>	 <b>605</b>
16.1 Introduction . . . . .	605
16.2 Boosting and Regularization Paths . . . . .	607
16.2.1 Penalized Regression . . . . .	607
16.2.2 The “Bet on Sparsity” Principle . . . . .	610
16.2.3 Regularization Paths, Over-fitting and Margins . . . . .	613
16.3 Learning Ensembles . . . . .	616
16.3.1 Learning a Good Ensemble . . . . .	617
16.3.2 Rule Ensembles . . . . .	622
Bibliographic Notes . . . . .	623
Exercises . . . . .	624
 <b>17 Undirected Graphical Models</b>	 <b>625</b>
17.1 Introduction . . . . .	625
17.2 Markov Graphs and Their Properties . . . . .	627
17.3 Undirected Graphical Models for Continuous Variables . . . . .	630
17.3.1 Estimation of the Parameters when the Graph Structure is Known . . . . .	631
17.3.2 Estimation of the Graph Structure . . . . .	635
17.4 Undirected Graphical Models for Discrete Variables . . . . .	638
17.4.1 Estimation of the Parameters when the Graph Structure is Known . . . . .	639
17.4.2 Hidden Nodes . . . . .	641
17.4.3 Estimation of the Graph Structure . . . . .	642
17.4.4 Restricted Boltzmann Machines . . . . .	643
Exercises . . . . .	645
 <b>18 High-Dimensional Problems: <math>p \gg N</math></b>	 <b>649</b>
18.1 When $p$ is Much Bigger than $N$ . . . . .	649

18.2	Diagonal Linear Discriminant Analysis and Nearest Shrunk Centroids . . . . .	651
18.3	Linear Classifiers with Quadratic Regularization . . . . .	654
18.3.1	Regularized Discriminant Analysis . . . . .	656
18.3.2	Logistic Regression with Quadratic Regularization . . . . .	657
18.3.3	The Support Vector Classifier . . . . .	657
18.3.4	Feature Selection . . . . .	658
18.3.5	Computational Shortcuts When $p \gg N$ . . . . .	659
18.4	Linear Classifiers with $L_1$ Regularization . . . . .	661
18.4.1	Application of Lasso to Protein Mass Spectroscopy . . . . .	664
18.4.2	The Fused Lasso for Functional Data . . . . .	666
18.5	Classification When Features are Unavailable . . . . .	668
18.5.1	Example: String Kernels and Protein Classification . . . . .	668
18.5.2	Classification and Other Models Using Inner-Product Kernels and Pairwise Distances . . . . .	670
18.5.3	Example: Abstracts Classification . . . . .	672
18.6	High-Dimensional Regression: Supervised Principal Components . . . . .	674
18.6.1	Connection to Latent-Variable Modeling . . . . .	678
18.6.2	Relationship with Partial Least Squares . . . . .	680
18.6.3	Pre-Conditioning for Feature Selection . . . . .	681
18.7	Feature Assessment and the Multiple-Testing Problem . . . . .	683
18.7.1	The False Discovery Rate . . . . .	687
18.7.2	Asymmetric Cutpoints and the SAM Procedure . . . . .	690
18.7.3	A Bayesian Interpretation of the FDR . . . . .	692
18.8	Bibliographic Notes . . . . .	693
	Exercises . . . . .	694
	<b>References</b>	<b>699</b>
	<b>Author Index</b>	<b>729</b>
	<b>Index</b>	<b>737</b>

# 1

## Introduction

*Statistical learning* plays a key role in many areas of science, finance and industry. Here are some examples of learning problems:

- Predict whether a patient, hospitalized due to a heart attack, will have a second heart attack. The prediction is to be based on demographic, diet and clinical measurements for that patient.
- Predict the price of a stock in 6 months from now, on the basis of company performance measures and economic data.
- Identify the numbers in a handwritten ZIP code, from a digitized image.
- Estimate the amount of glucose in the blood of a diabetic person, from the infrared absorption spectrum of that person's blood.
- Identify the risk factors for prostate cancer, based on clinical and demographic variables.

The science of learning plays a key role in the fields of statistics, data mining and artificial intelligence, intersecting with areas of engineering and other disciplines.

This book is about learning from data. In a typical scenario, we have an outcome measurement, usually quantitative (such as a stock price) or categorical (such as heart attack/no heart attack), that we wish to predict based on a set of *features* (such as diet and clinical measurements). We have a *training set* of data, in which we observe the outcome and feature

**TABLE 1.1.** *Average percentage of words or characters in an email message equal to the indicated word or character. We have chosen the words and characters showing the largest difference between spam and email.*

	george	you	your	hp	free	hpl	!	our	re	edu	remove
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

measurements for a set of objects (such as people). Using this data we build a prediction model, or *learner*, which will enable us to predict the outcome for new unseen objects. A good learner is one that accurately predicts such an outcome.

The examples above describe what is called the *supervised learning* problem. It is called “supervised” because of the presence of the outcome variable to guide the learning process. In the *unsupervised learning problem*, we observe only the features and have no measurements of the outcome. Our task is rather to describe how the data are organized or clustered. We devote most of this book to supervised learning; the unsupervised problem is less developed in the literature, and is the focus of Chapter 14.

Here are some examples of real learning problems that are discussed in this book.

### *Example 1: Email Spam*

The data for this example consists of information from 4601 email messages, in a study to try to predict whether the email was junk email, or “spam.” The objective was to design an automatic spam detector that could filter out spam before clogging the users’ mailboxes. For all 4601 email messages, the true outcome (email type) **email** or **spam** is available, along with the relative frequencies of 57 of the most commonly occurring words and punctuation marks in the email message. This is a supervised learning problem, with the outcome the class variable **email/spam**. It is also called a *classification* problem.

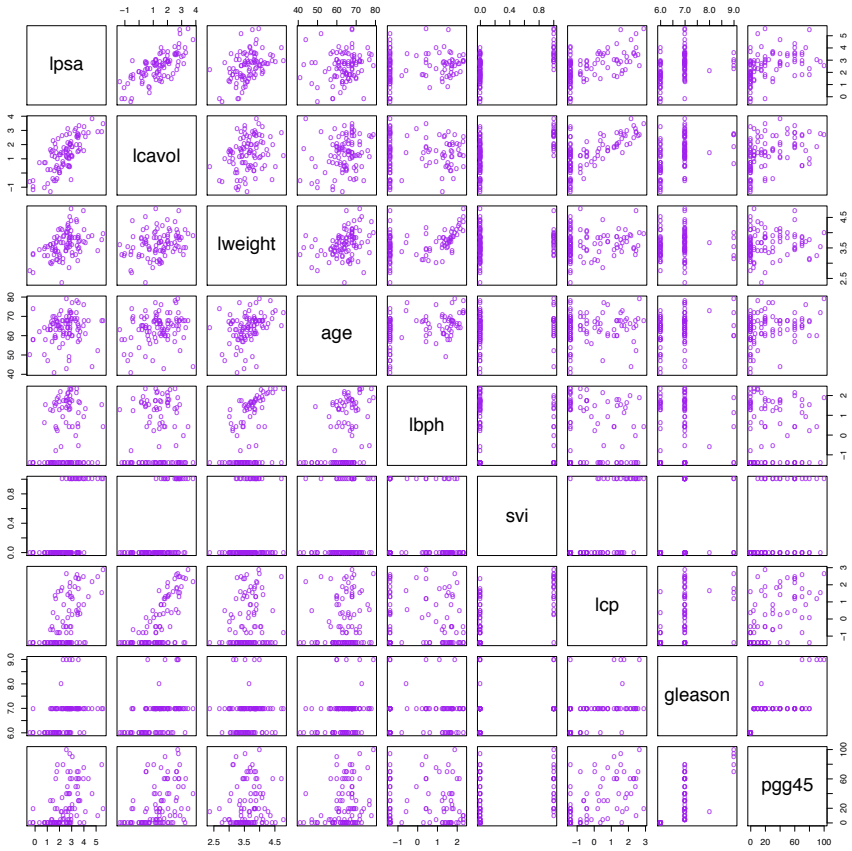
Table 1.1 lists the words and characters showing the largest average difference between **spam** and **email**.

Our learning method has to decide which features to use and how: for example, we might use a rule such as

```
if (%george < 0.6) & (%you > 1.5) then spam
                                else email.
```

Another form of a rule might be:

```
if (0.2 · %you − 0.3 · %george) > 0 then spam
                                else email.
```



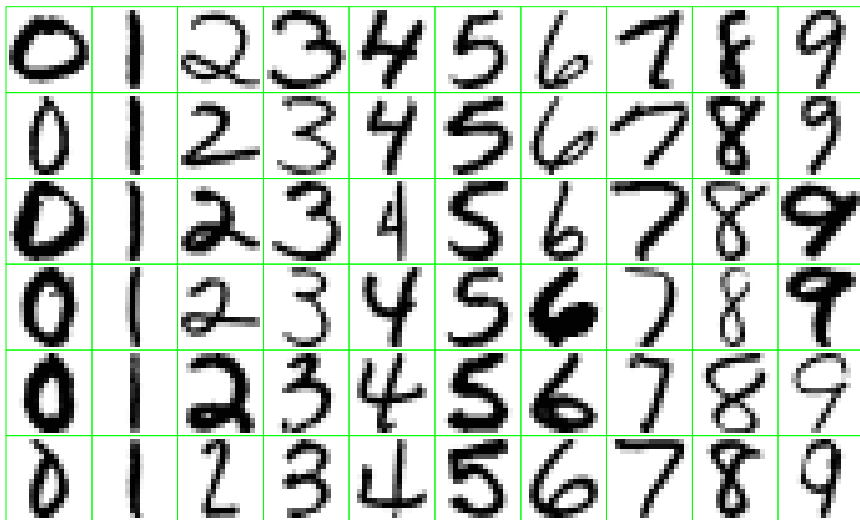
**FIGURE 1.1.** Scatterplot matrix of the prostate cancer data. The first row shows the response against each of the predictors in turn. Two of the predictors, *svi* and *gleason*, are categorical.

For this problem not all errors are equal; we want to avoid filtering out good email, while letting spam get through is not desirable but less serious in its consequences. We discuss a number of different methods for tackling this learning problem in the book.

### Example 2: Prostate Cancer

The data for this example, displayed in Figure 1.1<sup>1</sup>, come from a study by Stamey et al. (1989) that examined the correlation between the level of

<sup>1</sup>There was an error in these data in the first edition of this book. Subject 32 had a value of 6.1 for *lweight*, which translates to a 449 gm prostate! The correct value is 44.9 gm. We are grateful to Prof. Stephen W. Link for alerting us to this error.



**FIGURE 1.2.** Examples of handwritten digits from U.S. postal envelopes.

prostate specific antigen (PSA) and a number of clinical measures, in 97 men who were about to receive a radical prostatectomy.

The goal is to predict the log of PSA (`lpsa`) from a number of measurements including log cancer volume (`lcavol`), log prostate weight `lweight`, age, log of benign prostatic hyperplasia amount `lbph`, seminal vesicle invasion `svi`, log of capsular penetration `lcp`, Gleason score `gleason`, and percent of Gleason scores 4 or 5 `pgg45`. Figure 1.1 is a scatterplot matrix of the variables. Some correlations with `lpsa` are evident, but a good predictive model is difficult to construct by eye.

This is a supervised learning problem, known as a *regression problem*, because the outcome measurement is quantitative.

### Example 3: Handwritten Digit Recognition

The data from this example come from the handwritten ZIP codes on envelopes from U.S. postal mail. Each image is a segment from a five digit ZIP code, isolating a single digit. The images are  $16 \times 16$  eight-bit grayscale maps, with each pixel ranging in intensity from 0 to 255. Some sample images are shown in Figure 1.2.

The images have been normalized to have approximately the same size and orientation. The task is to predict, from the  $16 \times 16$  matrix of pixel intensities, the identity of each image (0, 1, ..., 9) quickly and accurately. If it is accurate enough, the resulting algorithm would be used as part of an automatic sorting procedure for envelopes. This is a classification problem for which the error rate needs to be kept very low to avoid misdirection of

mail. In order to achieve this low error rate, some objects can be assigned to a “don’t know” category, and sorted instead by hand.

### *Example 4: DNA Expression Microarrays*

DNA stands for deoxyribonucleic acid, and is the basic material that makes up human chromosomes. DNA microarrays measure the expression of a gene in a cell by measuring the amount of mRNA (messenger ribonucleic acid) present for that gene. Microarrays are considered a breakthrough technology in biology, facilitating the quantitative study of thousands of genes simultaneously from a single sample of cells.

Here is how a DNA microarray works. The nucleotide sequences for a few thousand genes are printed on a glass slide. A target sample and a reference sample are labeled with red and green dyes, and each are hybridized with the DNA on the slide. Through fluoroscopy, the log (red/green) intensities of RNA hybridizing at each site is measured. The result is a few thousand numbers, typically ranging from say  $-6$  to  $6$ , measuring the expression level of each gene in the target relative to the reference sample. Positive values indicate higher expression in the target versus the reference, and vice versa for negative values.

A gene expression dataset collects together the expression values from a series of DNA microarray experiments, with each column representing an experiment. There are therefore several thousand rows representing individual genes, and tens of columns representing samples: in the particular example of Figure 1.3 there are 6830 genes (rows) and 64 samples (columns), although for clarity only a random sample of 100 rows are shown. The figure displays the data set as a heat map, ranging from green (negative) to red (positive). The samples are 64 cancer tumors from different patients.

The challenge here is to understand how the genes and samples are organized. Typical questions include the following:

- (a) which samples are most similar to each other, in terms of their expression profiles across genes?
- (b) which genes are most similar to each other, in terms of their expression profiles across samples?
- (c) do certain genes show very high (or low) expression for certain cancer samples?

We could view this task as a regression problem, with two categorical predictor variables—genes and samples—with the response variable being the level of expression. However, it is probably more useful to view it as *unsupervised learning* problem. For example, for question (a) above, we think of the samples as points in 6830-dimensional space, which we want to *cluster* together in some way.





**FIGURE 1.3.** DNA microarray data: expression matrix of 6830 genes (rows) and 64 samples (columns), for the human tumor data. Only a random sample of 100 rows are shown. The display is a heat map, ranging from bright green (negative, under expressed) to bright red (positive, over expressed). Missing values are gray. The rows and columns are displayed in a randomly chosen order.

## *Who Should Read this Book*

This book is designed for researchers and students in a broad variety of fields: statistics, artificial intelligence, engineering, finance and others. We expect that the reader will have had at least one elementary course in statistics, covering basic topics including linear regression.

We have not attempted to write a comprehensive catalog of learning methods, but rather to describe some of the most important techniques. Equally notable, we describe the underlying concepts and considerations by which a researcher can judge a learning method. We have tried to write this book in an intuitive fashion, emphasizing concepts rather than mathematical details.

As statisticians, our exposition will naturally reflect our backgrounds and areas of expertise. However in the past eight years we have been attending conferences in neural networks, data mining and machine learning, and our thinking has been heavily influenced by these exciting fields. This influence is evident in our current research, and in this book.


## *How This Book is Organized*

Our view is that one must understand simple methods before trying to grasp more complex ones. Hence, after giving an overview of the supervising learning problem in [Chapter 2](#), we discuss linear methods for regression and classification in [Chapters 3 and 4](#). In [Chapter 5](#) we describe splines, wavelets and regularization/penalization methods for a single predictor, while [Chapter 6](#) covers kernel methods and local regression. Both of these sets of methods are important building blocks for high-dimensional learning techniques. Model assessment and selection is the topic of [Chapter 7](#), covering the concepts of bias and variance, overfitting and methods such as cross-validation for choosing models. [Chapter 8](#) discusses model inference and averaging, including an overview of maximum likelihood, Bayesian inference and the bootstrap, the EM algorithm, Gibbs sampling and bagging. A related procedure called boosting is the focus of [Chapter 10](#).

In [Chapters 9–13](#) we describe a series of structured methods for supervised learning, with [Chapters 9 and 11](#) covering regression and [Chapters 12 and 13](#) focusing on classification. [Chapter 14](#) describes methods for unsupervised learning. Two recently proposed techniques, random forests and ensemble learning, are discussed in [Chapters 15 and 16](#). We describe undirected graphical models in [Chapter 17](#) and finally we study high-dimensional problems in [Chapter 18](#).

At the end of each chapter we discuss [computational considerations](#) important for data mining applications, including how the computations scale with the number of observations and predictors. Each chapter ends with [Bibliographic Notes](#) giving background references for the material.

We recommend that Chapters 1–4 be first read in sequence. Chapter 7 should also be considered mandatory, as it covers central concepts that pertain to all learning methods. With this in mind, the rest of the book can be read sequentially, or sampled, depending on the reader’s interest.

The symbol  indicates a technically difficult section, one that can be skipped without interrupting the flow of the discussion.

### *Book Website*

The website for this book is located at

`http://www-stat.stanford.edu/ElemStatLearn`

It contains a number of resources, including many of the datasets used in this book.

### *Note for Instructors*

We have successively used the first edition of this book as the basis for a two-quarter course, and with the additional materials in this second edition, it could even be used for a three-quarter sequence. Exercises are provided at the end of each chapter. It is important for students to have access to good software tools for these topics. We used the R and S-PLUS programming languages in our courses.

# 2

## Overview of Supervised Learning

### 2.1 Introduction

The first three examples described in Chapter 1 have several components in common. For each there is a set of variables that might be denoted as *inputs*, which are measured or preset. These have some influence on one or more *outputs*. For each example the goal is to use the inputs to predict the values of the outputs. This exercise is called *supervised learning*.

We have used the more modern language of machine learning. In the statistical literature the inputs are often called the *predictors*, a term we will use interchangeably with inputs, and more classically the *independent variables*. In the pattern recognition literature the term *features* is preferred, which we use as well. The outputs are called the *responses*, or classically the *dependent variables*.

### 2.2 Variable Types and Terminology

The outputs vary in nature among the examples. In the glucose prediction example, the output is a *quantitative* measurement, where some measurements are bigger than others, and measurements close in value are close in nature. In the famous Iris discrimination example due to R. A. Fisher, the output is *qualitative* (species of Iris) and assumes values in a finite set  $\mathcal{G} = \{Virginica, Setosa \text{ and } Versicolor\}$ . In the handwritten digit example the output is one of 10 different digit *classes*:  $\mathcal{G} = \{0, 1, \dots, 9\}$ . In both of

these there is no explicit ordering in the classes, and in fact often descriptive labels rather than numbers are used to denote the classes. Qualitative variables are also referred to as *categorical* or *discrete* variables as well as *factors*.

For both types of outputs it makes sense to think of using the inputs to predict the output. Given some specific atmospheric measurements today and yesterday, we want to predict the ozone level tomorrow. Given the grayscale values for the pixels of the digitized image of the handwritten digit, we want to predict its class label.

This distinction in output type has led to a naming convention for the prediction tasks: *regression* when we predict quantitative outputs, and *classification* when we predict qualitative outputs. We will see that these two tasks have a lot in common, and in particular both can be viewed as a task in function approximation.

Inputs also vary in measurement type; we can have some of each of qualitative and quantitative input variables. These have also led to distinctions in the types of methods that are used for prediction: some methods are defined most naturally for quantitative inputs, some most naturally for qualitative and some for both.

A third variable type is *ordered categorical*, such as *small*, *medium* and *large*, where there is an ordering between the values, but no metric notion is appropriate (the difference between medium and small need not be the same as that between large and medium). These are discussed further in Chapter 4.

Qualitative variables are typically represented numerically by codes. The easiest case is when there are only two classes or categories, such as “success” or “failure,” “survived” or “died.” These are often represented by a single binary digit or bit as 0 or 1, or else by  $-1$  and  $1$ . For reasons that will become apparent, such numeric codes are sometimes referred to as *targets*. When there are more than two categories, several alternatives are available. The most useful and commonly used coding is via *dummy variables*. Here a  $K$ -level qualitative variable is represented by a vector of  $K$  binary variables or bits, only one of which is “on” at a time. Although more compact coding schemes are possible, dummy variables are symmetric in the levels of the factor.

We will typically denote an input variable by the symbol  $X$ . If  $X$  is a vector, its components can be accessed by subscripts  $X_j$ . Quantitative outputs will be denoted by  $Y$ , and qualitative outputs by  $G$  (for group). We use uppercase letters such as  $X$ ,  $Y$  or  $G$  when referring to the generic aspects of a variable. Observed values are written in lowercase; hence the  $i$ th observed value of  $X$  is written as  $x_i$  (where  $x_i$  is again a scalar or vector). Matrices are represented by bold uppercase letters; for example, a set of  $N$  input  $p$ -vectors  $x_i$ ,  $i = 1, \dots, N$  would be represented by the  $N \times p$  matrix  $\mathbf{X}$ . In general, vectors will not be bold, except when they have  $N$  components; this convention distinguishes a  $p$ -vector of inputs  $x_i$  for the

$i$ th observation from the  $N$ -vector  $\mathbf{x}_j$  consisting of all the observations on variable  $X_j$ . Since all vectors are assumed to be column vectors, the  $i$ th row of  $\mathbf{X}$  is  $x_i^T$ , the vector transpose of  $x_i$ .

For the moment we can loosely state the learning task as follows: given the value of an input vector  $X$ , make a good prediction of the output  $Y$ , denoted by  $\hat{Y}$  (pronounced “y-hat”). If  $Y$  takes values in  $\mathbb{R}$  then so should  $\hat{Y}$ ; likewise for categorical outputs,  $\hat{G}$  should take values in the same set  $\mathcal{G}$  associated with  $G$ .

For a two-class  $G$ , one approach is to denote the binary coded target as  $Y$ , and then treat it as a quantitative output. The predictions  $\hat{Y}$  will typically lie in  $[0, 1]$ , and we can assign to  $\hat{G}$  the class label according to whether  $\hat{y} > 0.5$ . This approach generalizes to  $K$ -level qualitative outputs as well.

We need data to construct prediction rules, often a lot of it. We thus suppose we have available a set of measurements  $(x_i, y_i)$  or  $(x_i, g_i)$ ,  $i = 1, \dots, N$ , known as the *training data*, with which to construct our prediction rule.

## 2.3 Two Simple Approaches to Prediction: Least Squares and Nearest Neighbors

In this section we develop two simple but powerful prediction methods: the linear model fit by least squares and the  $k$ -nearest-neighbor prediction rule. The linear model makes huge assumptions about structure and yields stable but possibly inaccurate predictions. The method of  $k$ -nearest neighbors makes very mild structural assumptions: its predictions are often accurate but can be unstable.

### 2.3.1 Linear Models and Least Squares

The linear model has been a mainstay of statistics for the past 30 years and remains one of our most important tools. Given a vector of inputs  $X^T = (X_1, X_2, \dots, X_p)$ , we predict the output  $Y$  via the model

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^p X_j \hat{\beta}_j. \quad (2.1)$$

The term  $\hat{\beta}_0$  is the intercept, also known as the *bias* in machine learning. Often it is convenient to include the constant variable 1 in  $X$ , include  $\hat{\beta}_0$  in the vector of coefficients  $\hat{\beta}$ , and then write the linear model in vector form as an inner product

$$\hat{Y} = X^T \hat{\beta}, \quad (2.2)$$

where  $X^T$  denotes vector or matrix transpose ( $X$  being a column vector). Here we are modeling a single output, so  $\hat{Y}$  is a scalar; in general  $\hat{Y}$  can be a  $K$ -vector, in which case  $\beta$  would be a  $p \times K$  matrix of coefficients. In the  $(p+1)$ -dimensional input-output space,  $(X, \hat{Y})$  represents a hyperplane. If the constant is included in  $X$ , then the hyperplane includes the origin and is a subspace; if not, it is an affine set cutting the  $Y$ -axis at the point  $(0, \hat{\beta}_0)$ . From now on we assume that the intercept is included in  $\hat{\beta}$ .

Viewed as a function over the  $p$ -dimensional input space,  $f(X) = X^T \beta$  is linear, and the gradient  $f'(X) = \beta$  is a vector in input space that points in the steepest uphill direction.

How do we fit the linear model to a set of training data? There are many different methods, but by far the most popular is the method of *least squares*. In this approach, we pick the coefficients  $\beta$  to minimize the residual sum of squares

$$\text{RSS}(\beta) = \sum_{i=1}^N (y_i - x_i^T \beta)^2. \quad (2.3)$$

$\text{RSS}(\beta)$  is a quadratic function of the parameters, and hence its minimum always exists, but may not be unique. The solution is easiest to characterize in matrix notation. We can write

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta), \quad (2.4)$$

where  $\mathbf{X}$  is an  $N \times p$  matrix with each row an input vector, and  $\mathbf{y}$  is an  $N$ -vector of the outputs in the training set. Differentiating w.r.t.  $\beta$  we get the *normal equations*

$$\mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta) = 0. \quad (2.5)$$

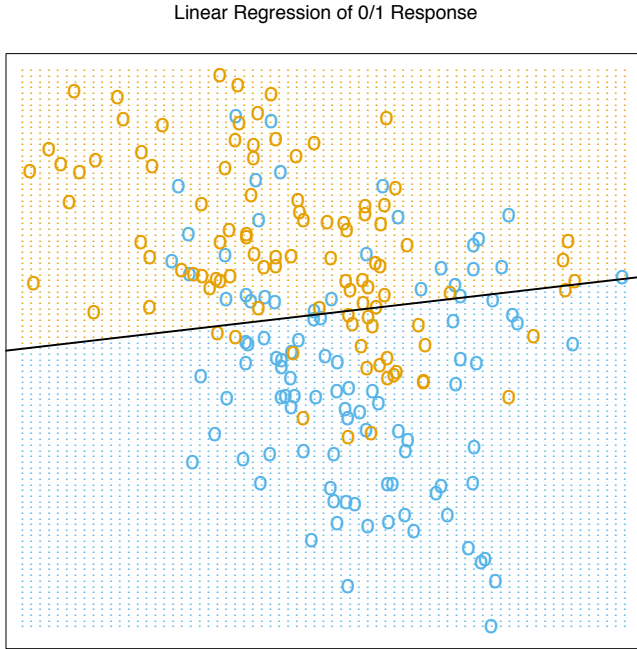
If  $\mathbf{X}^T \mathbf{X}$  is nonsingular, then the unique solution is given by

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}, \quad (2.6)$$

and the fitted value at the  $i$ th input  $x_i$  is  $\hat{y}_i = \hat{y}(x_i) = x_i^T \hat{\beta}$ . At an arbitrary input  $x_0$  the prediction is  $\hat{y}(x_0) = x_0^T \hat{\beta}$ . The entire fitted surface is characterized by the  $p$  parameters  $\hat{\beta}$ . Intuitively, it seems that we do not need a very large data set to fit such a model.

Let's look at an example of the linear model in a classification context. Figure 2.1 shows a scatterplot of training data on a pair of inputs  $X_1$  and  $X_2$ . The data are simulated, and for the present the simulation model is not important. The output class variable  $G$  has the values **BLUE** or **ORANGE**, and is represented as such in the scatterplot. There are 100 points in each of the two classes. The linear regression model was fit to these data, with the response  $Y$  coded as 0 for **BLUE** and 1 for **ORANGE**. The fitted values  $\hat{Y}$  are converted to a fitted class variable  $\hat{G}$  according to the rule

$$\hat{G} = \begin{cases} \text{ORANGE} & \text{if } \hat{Y} > 0.5, \\ \text{BLUE} & \text{if } \hat{Y} \leq 0.5. \end{cases} \quad (2.7)$$



**FIGURE 2.1.** A classification example in two dimensions. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1), and then fit by linear regression. The line is the decision boundary defined by  $x^T \hat{\beta} = 0.5$ . The orange shaded region denotes that part of input space classified as ORANGE, while the blue region is classified as BLUE.

The set of points in  $\mathbb{R}^2$  classified as ORANGE corresponds to  $\{x : x^T \hat{\beta} > 0.5\}$ , indicated in Figure 2.1, and the two predicted classes are separated by the decision boundary  $\{x : x^T \hat{\beta} = 0.5\}$ , which is linear in this case. We see that for these data there are several misclassifications on both sides of the decision boundary. Perhaps our linear model is too rigid—or are such errors unavoidable? Remember that these are errors on the training data itself, and we have not said where the constructed data came from. Consider the two possible scenarios:

**Scenario 1:** The training data in each class were generated from bivariate Gaussian distributions with uncorrelated components and different means.

**Scenario 2:** The training data in each class came from a mixture of 10 low-variance Gaussian distributions, with individual means themselves distributed as Gaussian.

A mixture of Gaussians is best described in terms of the generative model. One first generates a discrete variable that determines which of



the component Gaussians to use, and then generates an observation from the chosen density. In the case of one Gaussian per class, we will see in Chapter 4 that a linear decision boundary is the best one can do, and that our estimate is almost optimal. The region of overlap is inevitable, and future data to be predicted will be plagued by this overlap as well.

In the case of mixtures of tightly clustered Gaussians the story is different. A linear decision boundary is unlikely to be optimal, and in fact is not. The optimal decision boundary is nonlinear and disjoint, and as such will be much more difficult to obtain.

We now look at another classification and regression procedure that is in some sense at the opposite end of the spectrum to the linear model, and far better suited to the second scenario.

### 2.3.2 Nearest-Neighbor Methods

Nearest-neighbor methods use those observations in the training set  $\mathcal{T}$  closest in input space to  $x$  to form  $\hat{Y}$ . Specifically, the  $k$ -nearest neighbor fit for  $\hat{Y}$  is defined as follows:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \quad (2.8)$$

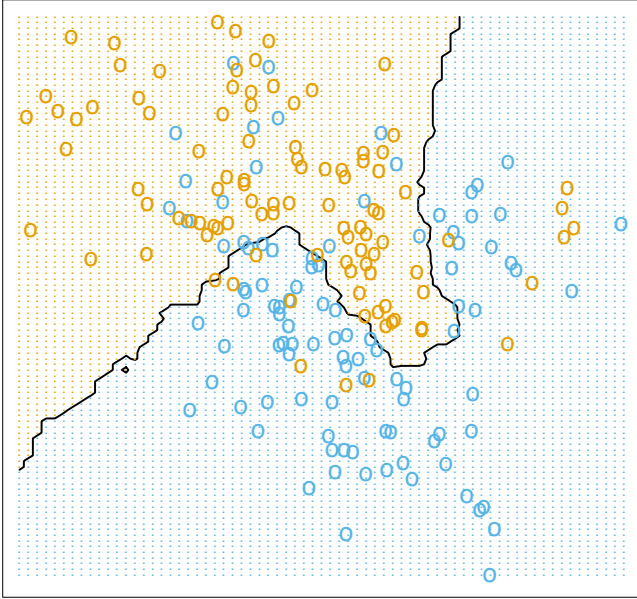
where  $N_k(x)$  is the neighborhood of  $x$  defined by the  $k$  closest points  $x_i$  in the training sample. Closeness implies a metric, which for the moment we assume is Euclidean distance. So, in words, we find the  $k$  observations with  $x_i$  closest to  $x$  in input space, and average their responses.

In Figure 2.2 we use the same training data as in Figure 2.1, and use 15-nearest-neighbor averaging of the binary coded response as the method of fitting. Thus  $\hat{Y}$  is the proportion of **ORANGE**'s in the neighborhood, and so assigning class **ORANGE** to  $\hat{G}$  if  $\hat{Y} > 0.5$  amounts to a majority vote in the neighborhood. The colored regions indicate all those points in input space classified as **BLUE** or **ORANGE** by such a rule, in this case found by evaluating the procedure on a fine grid in input space. We see that the decision boundaries that separate the **BLUE** from the **ORANGE** regions are far more irregular, and respond to local clusters where one class dominates.

Figure 2.3 shows the results for 1-nearest-neighbor classification:  $\hat{Y}$  is assigned the value  $y_\ell$  of the closest point  $x_\ell$  to  $x$  in the training data. In this case the regions of classification can be computed relatively easily, and correspond to a *Voronoi tessellation* of the training data. Each point  $x_i$  has an associated tile bounding the region for which it is the closest input point. For all points  $x$  in the tile,  $\hat{G}(x) = g_i$ . The decision boundary is even more irregular than before.

The method of  $k$ -nearest-neighbor averaging is defined in exactly the same way for regression of a quantitative output  $Y$ , although  $k = 1$  would be an unlikely choice.

## 15-Nearest Neighbor Classifier

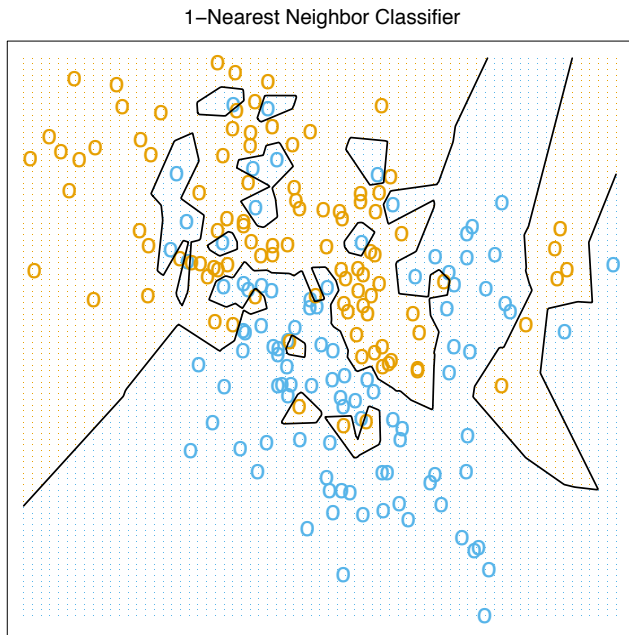


**FIGURE 2.2.** The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1) and then fit by 15-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the 15-nearest neighbors.

In Figure 2.2 we see that far fewer training observations are misclassified than in Figure 2.1. This should not give us too much comfort, though, since in Figure 2.3 *none* of the training data are misclassified. A little thought suggests that for  $k$ -nearest-neighbor fits, the error on the training data should be approximately an increasing function of  $k$ , and will always be 0 for  $k = 1$ . An independent test set would give us a more satisfactory means for comparing the different methods.

It appears that  $k$ -nearest-neighbor fits have a single parameter, the number of neighbors  $k$ , compared to the  $p$  parameters in least-squares fits. Although this is the case, we will see that the *effective* number of parameters of  $k$ -nearest neighbors is  $N/k$  and is generally bigger than  $p$ , and decreases with increasing  $k$ . To get an idea of why, note that if the neighborhoods were nonoverlapping, there would be  $N/k$  neighborhoods and we would fit one parameter (a mean) in each neighborhood.

It is also clear that we cannot use sum-of-squared errors on the training set as a criterion for picking  $k$ , since we would always pick  $k = 1$ ! It would seem that  $k$ -nearest-neighbor methods would be more appropriate for the mixture Scenario 2 described above, while for Gaussian data the decision boundaries of  $k$ -nearest neighbors would be unnecessarily noisy.



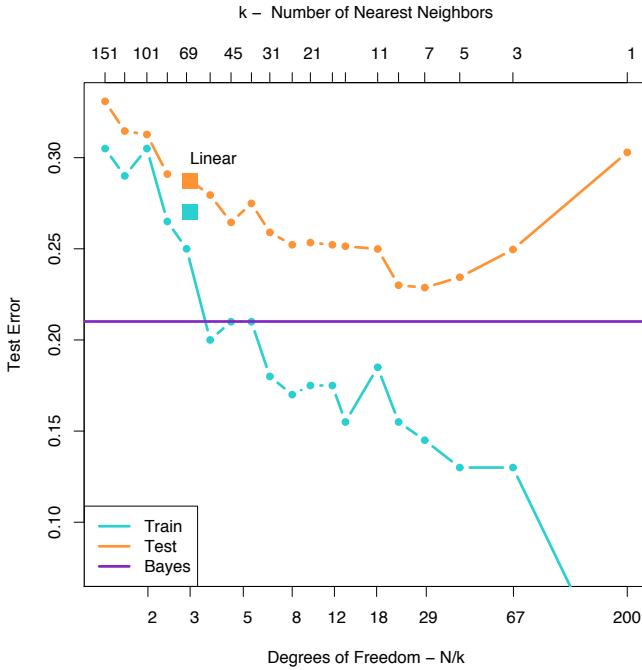
**FIGURE 2.3.** The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1), and then predicted by 1-nearest-neighbor classification.

### 2.3.3 From Least Squares to Nearest Neighbors

The linear decision boundary from least squares is very smooth, and apparently stable to fit. It does appear to rely heavily on the assumption that a linear decision boundary is appropriate. In language we will develop shortly, it has low variance and potentially high bias.

On the other hand, the  $k$ -nearest-neighbor procedures do not appear to rely on any stringent assumptions about the underlying data, and can adapt to any situation. However, any particular subregion of the decision boundary depends on a handful of input points and their particular positions, and is thus wiggly and unstable—high variance and low bias.

Each method has its own situations for which it works best; in particular linear regression is more appropriate for Scenario 1 above, while nearest neighbors are more suitable for Scenario 2. The time has come to expose the oracle! The data in fact were simulated from a model somewhere between the two, but closer to Scenario 2. First we generated 10 means  $m_k$  from a bivariate Gaussian distribution  $N((1, 0)^T, \mathbf{I})$  and labeled this class BLUE. Similarly, 10 more were drawn from  $N((0, 1)^T, \mathbf{I})$  and labeled class ORANGE. Then for each class we generated 100 observations as follows: for each observation, we picked an  $m_k$  at random with probability 1/10, and



**FIGURE 2.4.** Misclassification curves for the simulation example used in Figures 2.1, 2.2 and 2.3. A single training sample of size 200 was used, and a test sample of size 10,000. The orange curve is test and the blue are training error for  $k$ -nearest-neighbor classification. The results for linear regression are the bigger orange and blue squares at three degrees of freedom. The purple line is the optimal Bayes error rate.

then generated a  $N(m_k, \mathbf{I}/5)$ , thus leading to a mixture of Gaussian clusters for each class. Figure 2.4 shows the results of classifying 10,000 new observations generated from the model. We compare the results for least squares and those for  $k$ -nearest neighbors for a range of values of  $k$ .

A large subset of the most popular techniques in use today are variants of these two simple procedures. In fact 1-nearest-neighbor, the simplest of all, captures a large percentage of the market for low-dimensional problems. The following list describes some ways in which these simple procedures have been enhanced:

- Kernel methods use weights that decrease smoothly to zero with distance from the target point, rather than the effective 0/1 weights used by  $k$ -nearest neighbors.
- In high-dimensional spaces the distance kernels are modified to emphasize some variable more than others.

- Local regression fits linear models by locally weighted least squares, rather than fitting constants locally.
- Linear models fit to a basis expansion of the original inputs allow arbitrarily complex models.
- Projection pursuit and neural network models consist of sums of non-linearly transformed linear models.

## 2.4 Statistical Decision Theory

In this section we develop a small amount of theory that provides a framework for developing models such as those discussed informally so far. We first consider the case of a quantitative output, and place ourselves in the world of random variables and probability spaces. Let  $X \in \mathbb{R}^p$  denote a real valued random input vector, and  $Y \in \mathbb{R}$  a real valued random output variable, with joint distribution  $\Pr(X, Y)$ . We seek a function  $f(X)$  for predicting  $Y$  given values of the input  $X$ . This theory requires a *loss function*  $L(Y, f(X))$  for penalizing errors in prediction, and by far the most common and convenient is *squared error loss*:  $L(Y, f(X)) = (Y - f(X))^2$ . This leads us to a criterion for choosing  $f$ ,

$$\text{EPE}(f) = \mathbb{E}(Y - f(X))^2 \quad (2.9)$$

$$= \int [y - f(x)]^2 \Pr(dx, dy), \quad (2.10)$$

the expected (squared) prediction error. By conditioning<sup>1</sup> on  $X$ , we can write EPE as

$$\text{EPE}(f) = \mathbb{E}_X \mathbb{E}_{Y|X} ([Y - f(X)]^2 | X) \quad (2.11)$$

and we see that it suffices to minimize EPE pointwise:

$$f(x) = \operatorname{argmin}_c \mathbb{E}_{Y|X} ([Y - c]^2 | X = x). \quad (2.12)$$

The solution is

$$f(x) = \mathbb{E}(Y | X = x), \quad (2.13)$$

the conditional expectation, also known as the *regression* function. Thus the best prediction of  $Y$  at any point  $X = x$  is the conditional mean, when best is measured by average squared error.

The nearest-neighbor methods attempt to directly implement this recipe using the training data. At each point  $x$ , we might ask for the average of all

---

<sup>1</sup>Conditioning here amounts to factoring the joint density  $\Pr(X, Y) = \Pr(Y|X)\Pr(X)$  where  $\Pr(Y|X) = \Pr(Y, X)/\Pr(X)$ , and splitting up the bivariate integral accordingly.

those  $y_i$ s with input  $x_i = x$ . Since there is typically at most one observation at any point  $x$ , we settle for

$$\hat{f}(x) = \text{Ave}(y_i | x_i \in N_k(x)), \quad (2.14)$$

where “Ave” denotes average, and  $N_k(x)$  is the neighborhood containing the  $k$  points in  $\mathcal{T}$  closest to  $x$ . Two approximations are happening here:

- expectation is approximated by averaging over sample data;
- conditioning at a point is relaxed to conditioning on some region “close” to the target point.

For large training sample size  $N$ , the points in the neighborhood are likely to be close to  $x$ , and as  $k$  gets large the average will get more stable. In fact, under mild regularity conditions on the joint probability distribution  $\Pr(X, Y)$ , one can show that as  $N, k \rightarrow \infty$  such that  $k/N \rightarrow 0$ ,  $\hat{f}(x) \rightarrow \mathbb{E}(Y | X = x)$ . In light of this, why look further, since it seems we have a universal approximator? We often do not have very large samples. If the linear or some more structured model is appropriate, then we can usually get a more stable estimate than  $k$ -nearest neighbors, although such knowledge has to be learned from the data as well. There are other problems though, sometimes disastrous. In Section 2.5 we see that as the dimension  $p$  gets large, so does the metric size of the  $k$ -nearest neighborhood. So settling for nearest neighborhood as a surrogate for conditioning will fail us miserably. The convergence above still holds, but the *rate* of convergence decreases as the dimension increases.

How does linear regression fit into this framework? The simplest explanation is that one assumes that the regression function  $f(x)$  is approximately linear in its arguments:

$$f(x) \approx x^T \beta. \quad (2.15)$$

This is a model-based approach—we specify a model for the regression function. Plugging this linear model for  $f(x)$  into EPE (2.9) and differentiating we can solve for  $\beta$  theoretically:

$$\beta = [\mathbb{E}(XX^T)]^{-1} \mathbb{E}(XY). \quad (2.16)$$

Note we have *not* conditioned on  $X$ ; rather we have used our knowledge of the functional relationship to *pool* over values of  $X$ . The least squares solution (2.6) amounts to replacing the expectation in (2.16) by averages over the training data.

So both  $k$ -nearest neighbors and least squares end up approximating conditional expectations by averages. But they differ dramatically in terms of model assumptions:

- Least squares assumes  $f(x)$  is well approximated by a globally linear function.

- $k$ -nearest neighbors assumes  $f(x)$  is well approximated by a locally constant function.

Although the latter seems more palatable, we have already seen that we may pay a price for this flexibility.

Many of the more modern techniques described in this book are model based, although far more flexible than the rigid linear model. For example, additive models assume that

$$f(X) = \sum_{j=1}^p f_j(X_j). \quad (2.17)$$

This retains the additivity of the linear model, but each coordinate function  $f_j$  is arbitrary. It turns out that the optimal estimate for the additive model uses techniques such as  $k$ -nearest neighbors to approximate *univariate* conditional expectations *simultaneously* for each of the coordinate functions. Thus the problems of estimating a conditional expectation in high dimensions are swept away in this case by imposing some (often unrealistic) model assumptions, in this case additivity.

Are we happy with the criterion (2.11)? What happens if we replace the  $L_2$  loss function with the  $L_1$ :  $E|Y - f(X)|$ ? The solution in this case is the conditional median,

$$\hat{f}(x) = \text{median}(Y|X = x), \quad (2.18)$$

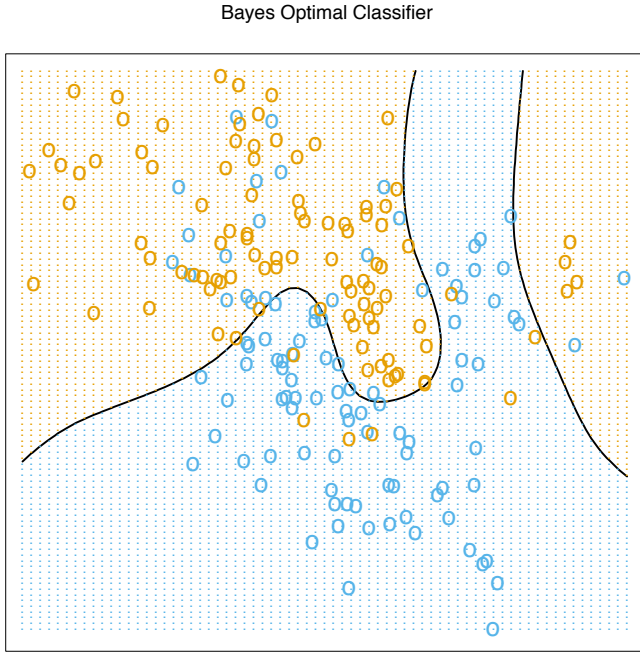
which is a different measure of location, and its estimates are more robust than those for the conditional mean.  $L_1$  criteria have discontinuities in their derivatives, which have hindered their widespread use. Other more resistant loss functions will be mentioned in later chapters, but squared error is analytically convenient and the most popular.

What do we do when the output is a categorical variable  $G$ ? The same paradigm works here, except we need a different loss function for penalizing prediction errors. An estimate  $\hat{G}$  will assume values in  $\mathcal{G}$ , the set of possible classes. Our loss function can be represented by a  $K \times K$  matrix  $\mathbf{L}$ , where  $K = \text{card}(\mathcal{G})$ .  $\mathbf{L}$  will be zero on the diagonal and nonnegative elsewhere, where  $L(k, \ell)$  is the price paid for classifying an observation belonging to class  $\mathcal{G}_k$  as  $\mathcal{G}_\ell$ . Most often we use the *zero-one* loss function, where all misclassifications are charged a single unit. The expected prediction error is

$$\text{EPE} = E[L(G, \hat{G}(X))], \quad (2.19)$$

where again the expectation is taken with respect to the joint distribution  $\text{Pr}(G, X)$ . Again we condition, and can write EPE as

$$\text{EPE} = E_X \sum_{k=1}^K L[\mathcal{G}_k, \hat{G}(X)] \text{Pr}(\mathcal{G}_k|X) \quad (2.20)$$



**FIGURE 2.5.** The optimal Bayes decision boundary for the simulation example of Figures 2.1, 2.2 and 2.3. Since the generating density is known for each class, this boundary can be calculated exactly (Exercise 2.2).

and again it suffices to minimize EPE pointwise:

$$\hat{G}(x) = \operatorname{argmin}_{g \in \mathcal{G}} \sum_{k=1}^K L(\mathcal{G}_k, g) \Pr(\mathcal{G}_k | X = x). \quad (2.21)$$

With the 0–1 loss function this simplifies to

$$\hat{G}(x) = \operatorname{argmin}_{g \in \mathcal{G}} [1 - \Pr(g | X = x)] \quad (2.22)$$

or simply

$$\hat{G}(x) = \mathcal{G}_k \text{ if } \Pr(\mathcal{G}_k | X = x) = \max_{g \in \mathcal{G}} \Pr(g | X = x). \quad (2.23)$$

This reasonable solution is known as the *Bayes classifier*, and says that we classify to the most probable class, using the conditional (discrete) distribution  $\Pr(G|X)$ . Figure 2.5 shows the Bayes-optimal decision boundary for our simulation example. The error rate of the Bayes classifier is called the *Bayes rate*.



Again we see that the  $k$ -nearest neighbor classifier directly approximates this solution—a majority vote in a nearest neighborhood amounts to exactly this, except that conditional probability at a point is relaxed to conditional probability within a neighborhood of a point, and probabilities are estimated by training-sample proportions.

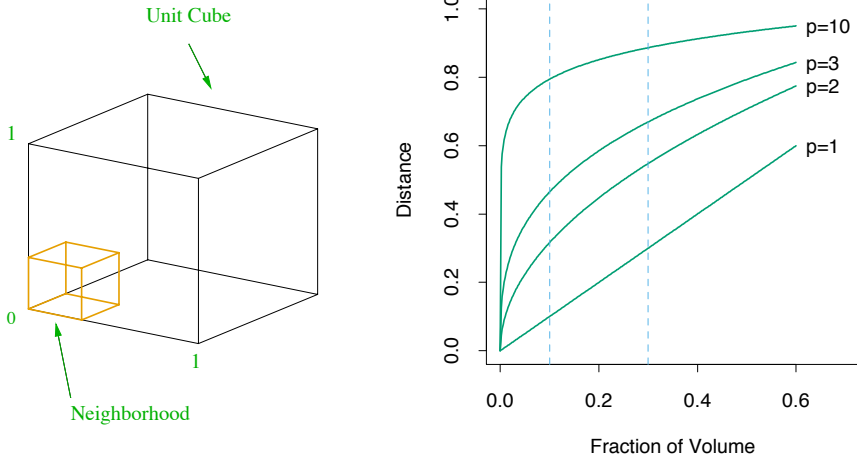
Suppose for a two-class problem we had taken the dummy-variable approach and coded  $G$  via a binary  $Y$ , followed by squared error loss estimation. Then  $\hat{f}(X) = E(Y|X) = \Pr(G = \mathcal{G}_1|X)$  if  $\mathcal{G}_1$  corresponded to  $Y = 1$ . Likewise for a  $K$ -class problem,  $E(Y_k|X) = \Pr(G = \mathcal{G}_k|X)$ . This shows that our dummy-variable regression procedure, followed by classification to the largest fitted value, is another way of representing the Bayes classifier. Although this theory is exact, in practice problems can occur, depending on the regression model used. For example, when linear regression is used,  $\hat{f}(X)$  need not be positive, and we might be suspicious about using it as an estimate of a probability. We will discuss a variety of approaches to modeling  $\Pr(G|X)$  in Chapter 4.

## 2.5 Local Methods in High Dimensions

We have examined two learning techniques for prediction so far: the stable but biased linear model and the less stable but apparently less biased class of  $k$ -nearest-neighbor estimates. It would seem that with a reasonably large set of training data, we could always approximate the theoretically optimal conditional expectation by  $k$ -nearest-neighbor averaging, since we should be able to find a fairly large neighborhood of observations close to any  $x$  and average them. This approach and our intuition breaks down in high dimensions, and the phenomenon is commonly referred to as the *curse of dimensionality* (Bellman, 1961). There are many manifestations of this problem, and we will examine a few here.

Consider the nearest-neighbor procedure for inputs uniformly distributed in a  $p$ -dimensional unit hypercube, as in Figure 2.6. Suppose we send out a hypercubical neighborhood about a target point to capture a fraction  $r$  of the observations. Since this corresponds to a fraction  $r$  of the unit volume, the expected edge length will be  $e_p(r) = r^{1/p}$ . In ten dimensions  $e_{10}(0.01) = 0.63$  and  $e_{10}(0.1) = 0.80$ , while the entire range for each input is only 1.0. So to capture 1% or 10% of the data to form a local average, we must cover 63% or 80% of the range of each input variable. Such neighborhoods are no longer “local.” Reducing  $r$  dramatically does not help much either, since the fewer observations we average, the higher is the variance of our fit.

Another consequence of the sparse sampling in high dimensions is that all sample points are close to an edge of the sample. Consider  $N$  data points uniformly distributed in a  $p$ -dimensional unit ball centered at the origin. Suppose we consider a nearest-neighbor estimate at the origin. The median



**FIGURE 2.6.** The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction  $r$  of the volume of the data, for different dimensions  $p$ . In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.

distance from the origin to the closest data point is given by the expression

$$d(p, N) = \left(1 - \frac{1}{2}\right)^{1/p} \quad (2.24)$$

(Exercise 2.3). A more complicated expression exists for the mean distance to the closest point. For  $N = 500$ ,  $p = 10$ ,  $d(p, N) \approx 0.52$ , more than halfway to the boundary. Hence most data points are closer to the boundary of the sample space than to any other data point. The reason that this presents a problem is that prediction is much more difficult near the edges of the training sample. One must extrapolate from neighboring sample points rather than interpolate between them.

Another manifestation of the curse is that the sampling density is proportional to  $N^{1/p}$ , where  $p$  is the dimension of the input space and  $N$  is the sample size. Thus, if  $N_1 = 100$  represents a dense sample for a single input problem, then  $N_{10} = 100^{10}$  is the sample size required for the same sampling density with 10 inputs. Thus in high dimensions all feasible training samples sparsely populate the input space.

Let us construct another uniform example. Suppose we have 1000 training examples  $x_i$  generated uniformly on  $[-1, 1]^p$ . Assume that the true relationship between  $X$  and  $Y$  is

$$Y = f(X) = e^{-8\|X\|^2},$$

without any measurement error. We use the 1-nearest-neighbor rule to predict  $y_0$  at the test-point  $x_0 = 0$ . Denote the training set by  $\mathcal{T}$ . We can

compute the expected prediction error at  $x_0$  for our procedure, averaging over all such samples of size 1000. Since the problem is deterministic, this is the mean squared error (MSE) for estimating  $f(0)$ :

$$\begin{aligned}\text{MSE}(x_0) &= \mathbb{E}_{\mathcal{T}}[f(x_0) - \hat{y}_0]^2 \\ &= \mathbb{E}_{\mathcal{T}}[\hat{y}_0 - \mathbb{E}_{\mathcal{T}}(\hat{y}_0)]^2 + [\mathbb{E}_{\mathcal{T}}(\hat{y}_0) - f(x_0)]^2 \\ &= \text{Var}_{\mathcal{T}}(\hat{y}_0) + \text{Bias}^2(\hat{y}_0).\end{aligned}\tag{2.25}$$

Figure 2.7 illustrates the setup. We have broken down the MSE into two components that will become familiar as we proceed: variance and squared bias. Such a decomposition is always possible and often useful, and is known as the *bias-variance decomposition*. Unless the nearest neighbor is at 0,  $\hat{y}_0$  will be smaller than  $f(0)$  in this example, and so the average estimate will be biased downward. The variance is due to the sampling variance of the 1-nearest neighbor. In low dimensions and with  $N = 1000$ , the nearest neighbor is very close to 0, and so both the bias and variance are small. As the dimension increases, the nearest neighbor tends to stray further from the target point, and both bias and variance are incurred. By  $p = 10$ , for more than 99% of the samples the nearest neighbor is a distance greater than 0.5 from the origin. Thus as  $p$  increases, the estimate tends to be 0 more often than not, and hence the MSE levels off at 1.0, as does the bias, and the variance starts dropping (an artifact of this example).

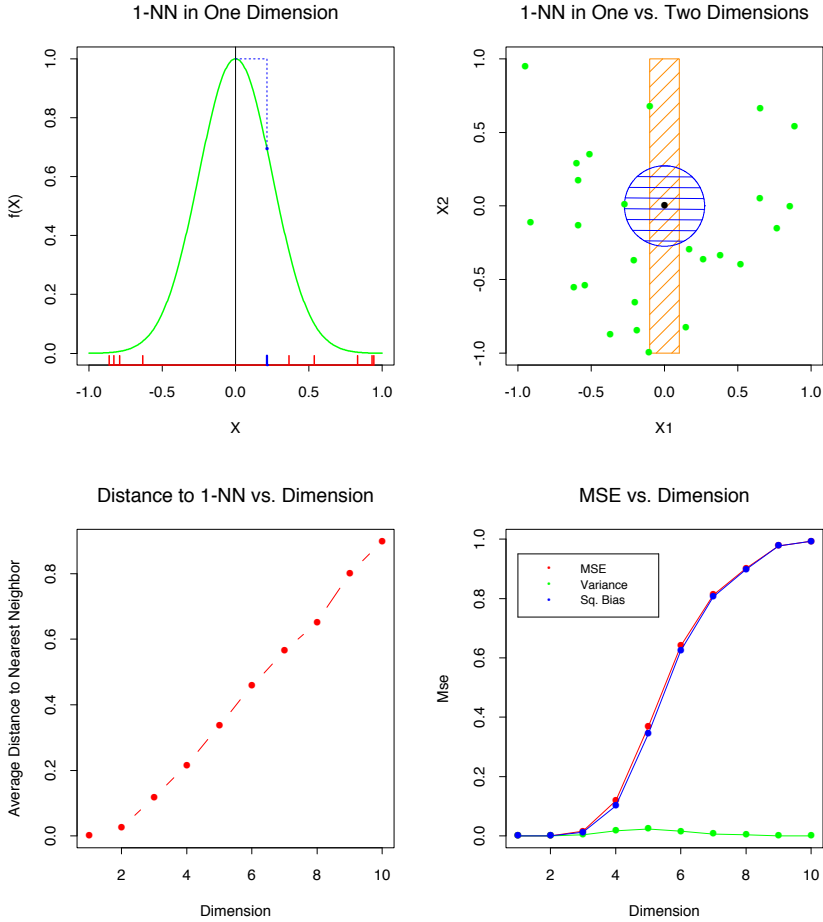
Although this is a highly contrived example, similar phenomena occur more generally. The complexity of functions of many variables can grow exponentially with the dimension, and if we wish to be able to estimate such functions with the same accuracy as function in low dimensions, then we need the size of our training set to grow exponentially as well. In this example, the function is a complex interaction of all  $p$  variables involved.

The dependence of the bias term on distance depends on the truth, and it need not always dominate with 1-nearest neighbor. For example, if the function always involves only a few dimensions as in Figure 2.8, then the variance can dominate instead.

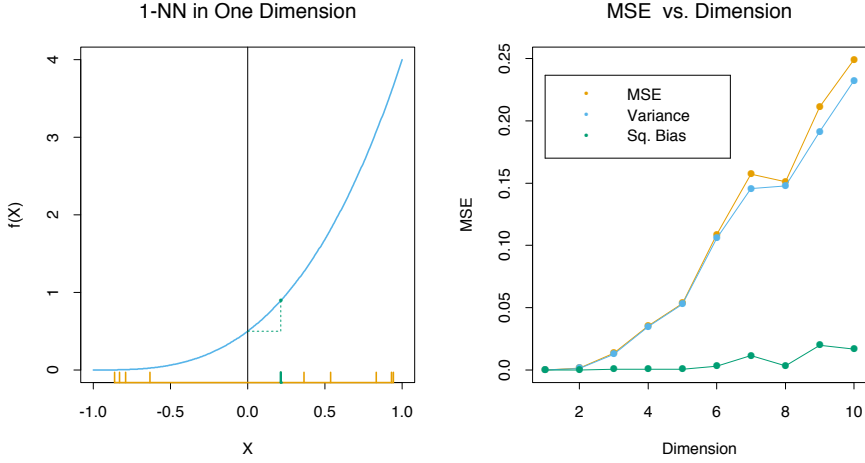
Suppose, on the other hand, that we know that the relationship between  $Y$  and  $X$  is linear,

$$Y = X^T \beta + \varepsilon,\tag{2.26}$$

where  $\varepsilon \sim N(0, \sigma^2)$  and we fit the model by least squares to the training data. For an arbitrary test point  $x_0$ , we have  $\hat{y}_0 = x_0^T \hat{\beta}$ , which can be written as  $\hat{y}_0 = x_0^T \beta + \sum_{i=1}^N \ell_i(x_0) \varepsilon_i$ , where  $\ell_i(x_0)$  is the  $i$ th element of  $\mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} x_0$ . Since under this model the least squares estimates are



**FIGURE 2.7.** A simulation example, demonstrating the curse of dimensionality and its effect on MSE, bias and variance. The input features are uniformly distributed in  $[-1, 1]^p$  for  $p = 1, \dots, 10$ . The top left panel shows the target function (no noise) in  $\mathbb{R}$ :  $f(X) = e^{-8\|X\|^2}$ , and demonstrates the error that 1-nearest neighbor makes in estimating  $f(0)$ . The training point is indicated by the blue tick mark. The top right panel illustrates why the radius of the 1-nearest neighborhood increases with dimension  $p$ . The lower left panel shows the average radius of the 1-nearest neighborhoods. The lower-right panel shows the MSE, squared bias and variance curves as a function of dimension  $p$ .



**FIGURE 2.8.** A simulation example with the same setup as in Figure 2.7. Here the function is constant in all but one dimension:  $f(X) = \frac{1}{2}(X_1 + 1)^3$ . The variance dominates.

unbiased, we find that

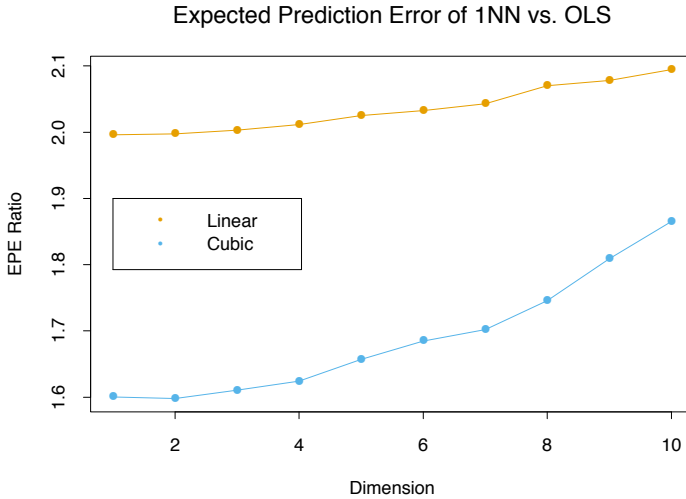
$$\begin{aligned}
 \text{EPE}(x_0) &= \mathbb{E}_{y_0|x_0} \mathbb{E}_{\mathcal{T}} (y_0 - \hat{y}_0)^2 \\
 &= \text{Var}(y_0|x_0) + \mathbb{E}_{\mathcal{T}} [\hat{y}_0 - \mathbb{E}_{\mathcal{T}} \hat{y}_0]^2 + [\mathbb{E}_{\mathcal{T}} \hat{y}_0 - x_0^T \beta]^2 \\
 &= \text{Var}(y_0|x_0) + \text{Var}_{\mathcal{T}}(\hat{y}_0) + \text{Bias}^2(\hat{y}_0) \\
 &= \sigma^2 + \mathbb{E}_{\mathcal{T}} x_0^T (\mathbf{X}^T \mathbf{X})^{-1} x_0 \sigma^2 + 0^2.
 \end{aligned} \tag{2.27}$$

Here we have incurred an additional variance  $\sigma^2$  in the prediction error, since our target is not deterministic. There is no bias, and the variance depends on  $x_0$ . If  $N$  is large and  $\mathcal{T}$  were selected at random, and assuming  $\mathbb{E}(X) = 0$ , then  $\mathbf{X}^T \mathbf{X} \rightarrow N \text{Cov}(X)$  and

$$\begin{aligned}
 \mathbb{E}_{x_0} \text{EPE}(x_0) &\sim \mathbb{E}_{x_0} x_0^T \text{Cov}(X)^{-1} x_0 \sigma^2 / N + \sigma^2 \\
 &= \text{trace}[\text{Cov}(X)^{-1} \text{Cov}(x_0)] \sigma^2 / N + \sigma^2 \\
 &= \sigma^2(p/N) + \sigma^2.
 \end{aligned} \tag{2.28}$$

Here we see that the expected EPE increases linearly as a function of  $p$ , with slope  $\sigma^2/N$ . If  $N$  is large and/or  $\sigma^2$  is small, this growth in variance is negligible (0 in the deterministic case). By imposing some heavy restrictions on the class of models being fitted, we have avoided the curse of dimensionality. Some of the technical details in (2.27) and (2.28) are derived in Exercise 2.5.

Figure 2.9 compares 1-nearest neighbor vs. least squares in two situations, both of which have the form  $Y = f(X) + \varepsilon$ ,  $X$  uniform as before, and  $\varepsilon \sim N(0, 1)$ . The sample size is  $N = 500$ . For the orange curve,  $f(x)$



**FIGURE 2.9.** The curves show the expected prediction error (at  $x_0 = 0$ ) for 1-nearest neighbor relative to least squares for the model  $Y = f(X) + \varepsilon$ . For the orange curve,  $f(x) = x_1$ , while for the blue curve  $f(x) = \frac{1}{2}(x_1 + 1)^3$ .

is linear in the first coordinate, for the blue curve, cubic as in Figure 2.8. Shown is the relative EPE of 1-nearest neighbor to least squares, which appears to start at around 2 for the linear case. Least squares is unbiased in this case, and as discussed above the EPE is slightly above  $\sigma^2 = 1$ . The EPE for 1-nearest neighbor is always above 2, since the variance of  $\hat{f}(x_0)$  in this case is at least  $\sigma^2$ , and the ratio increases with dimension as the nearest neighbor strays from the target point. For the cubic case, least squares is biased, which moderates the ratio. Clearly we could manufacture examples where the bias of least squares would dominate the variance, and the 1-nearest neighbor would come out the winner.

By relying on rigid assumptions, the linear model has no bias at all and negligible variance, while the error in 1-nearest neighbor is substantially larger. However, if the assumptions are wrong, all bets are off and the 1-nearest neighbor may dominate. We will see that there is a whole spectrum of models between the rigid linear models and the extremely flexible 1-nearest-neighbor models, each with their own assumptions and biases, which have been proposed specifically to avoid the exponential growth in complexity of functions in high dimensions by drawing heavily on these assumptions.

Before we delve more deeply, let us elaborate a bit on the concept of *statistical models* and see how they fit into the prediction framework.

## 2.6 Statistical Models, Supervised Learning and Function Approximation

Our goal is to find a useful approximation  $\hat{f}(x)$  to the function  $f(x)$  that underlies the predictive relationship between the inputs and outputs. In the theoretical setting of Section 2.4, we saw that squared error loss lead us to the regression function  $f(x) = E(Y|X = x)$  for a quantitative response. The class of nearest-neighbor methods can be viewed as direct estimates of this conditional expectation, but we have seen that they can fail in at least two ways:

- if the dimension of the input space is high, the nearest neighbors need not be close to the target point, and can result in large errors;
- if special structure is known to exist, this can be used to reduce both the bias and the variance of the estimates.

We anticipate using other classes of models for  $f(x)$ , in many cases specifically designed to overcome the dimensionality problems, and here we discuss a framework for incorporating them into the prediction problem.

### 2.6.1 A Statistical Model for the Joint Distribution $\Pr(X, Y)$

Suppose in fact that our data arose from a statistical model

$$Y = f(X) + \varepsilon, \quad (2.29)$$

where the random error  $\varepsilon$  has  $E(\varepsilon) = 0$  and is independent of  $X$ . Note that for this model,  $f(x) = E(Y|X = x)$ , and in fact the conditional distribution  $\Pr(Y|X)$  depends on  $X$  *only* through the conditional mean  $f(x)$ .

The additive error model is a useful approximation to the truth. For most systems the input–output pairs  $(X, Y)$  will not have a deterministic relationship  $Y = f(X)$ . Generally there will be other unmeasured variables that also contribute to  $Y$ , including measurement error. The additive model assumes that we can capture all these departures from a deterministic relationship via the error  $\varepsilon$ .

For some problems a deterministic relationship does hold. Many of the classification problems studied in machine learning are of this form, where the response surface can be thought of as a colored map defined in  $\mathbb{R}^p$ . The training data consist of colored examples from the map  $\{x_i, g_i\}$ , and the goal is to be able to color any point. Here the function is deterministic, and the randomness enters through the  $x$  location of the training points. For the moment we will not pursue such problems, but will see that they can be handled by techniques appropriate for the error-based models.

The assumption in (2.29) that the errors are independent and identically distributed is not strictly necessary, but seems to be at the back of our mind

when we average squared errors uniformly in our EPE criterion. With such a model it becomes natural to use least squares as a data criterion for model estimation as in (2.1). Simple modifications can be made to avoid the independence assumption; for example, we can have  $\text{Var}(Y|X = x) = \sigma(x)$ , and now both the mean and variance depend on  $X$ . In general the conditional distribution  $\text{Pr}(Y|X)$  can depend on  $X$  in complicated ways, but the additive error model precludes these.

So far we have concentrated on the quantitative response. Additive error models are typically not used for qualitative outputs  $G$ ; in this case the target function  $p(X)$  is the conditional density  $\text{Pr}(G|X)$ , and this is modeled directly. For example, for two-class data, it is often reasonable to assume that the data arise from independent binary trials, with the probability of one particular outcome being  $p(X)$ , and the other  $1 - p(X)$ . Thus if  $Y$  is the 0–1 coded version of  $G$ , then  $E(Y|X = x) = p(x)$ , but the variance depends on  $x$  as well:  $\text{Var}(Y|X = x) = p(x)[1 - p(x)]$ .

### 2.6.2 Supervised Learning

Before we launch into more statistically oriented jargon, we present the function-fitting paradigm from a machine learning point of view. Suppose for simplicity that the errors are additive and that the model  $Y = f(X) + \varepsilon$  is a reasonable assumption. Supervised learning attempts to learn  $f$  by example through a *teacher*. One observes the system under study, both the inputs and outputs, and assembles a *training* set of observations  $\mathcal{T} = (x_i, y_i)$ ,  $i = 1, \dots, N$ . The observed input values to the system  $x_i$  are also fed into an artificial system, known as a learning algorithm (usually a computer program), which also produces outputs  $\hat{f}(x_i)$  in response to the inputs. The learning algorithm has the property that it can modify its input/output relationship  $\hat{f}$  in response to differences  $y_i - \hat{f}(x_i)$  between the original and generated outputs. This process is known as *learning by example*. Upon completion of the learning process the hope is that the artificial and real outputs will be close enough to be useful for all sets of inputs likely to be encountered in practice.

### 2.6.3 Function Approximation

The learning paradigm of the previous section has been the motivation for research into the supervised learning problem in the fields of machine learning (with analogies to human reasoning) and neural networks (with biological analogies to the brain). The approach taken in applied mathematics and statistics has been from the perspective of function approximation and estimation. Here the data pairs  $\{x_i, y_i\}$  are viewed as points in a  $(p + 1)$ -dimensional Euclidean space. The function  $f(x)$  has domain equal to the  $p$ -dimensional input subspace, and is related to the data via a model



such as  $y_i = f(x_i) + \varepsilon_i$ . For convenience in this chapter we will assume the domain is  $\mathbb{R}^p$ , a  $p$ -dimensional Euclidean space, although in general the inputs can be of mixed type. The goal is to obtain a useful approximation to  $f(x)$  for all  $x$  in some region of  $\mathbb{R}^p$ , given the representations in  $\mathcal{T}$ . Although somewhat less glamorous than the learning paradigm, treating supervised learning as a problem in function approximation encourages the geometrical concepts of Euclidean spaces and mathematical concepts of probabilistic inference to be applied to the problem. This is the approach taken in this book.

Many of the approximations we will encounter have associated a set of parameters  $\theta$  that can be modified to suit the data at hand. For example, the linear model  $f(x) = x^T \beta$  has  $\theta = \beta$ . Another class of useful approximators can be expressed as *linear basis expansions*

$$f_\theta(x) = \sum_{k=1}^K h_k(x) \theta_k, \quad (2.30)$$

where the  $h_k$  are a suitable set of functions or transformations of the input vector  $x$ . Traditional examples are polynomial and trigonometric expansions, where for example  $h_k$  might be  $x_1^2$ ,  $x_1 x_2^2$ ,  $\cos(x_1)$  and so on. We also encounter nonlinear expansions, such as the sigmoid transformation common to neural network models,

$$h_k(x) = \frac{1}{1 + \exp(-x^T \beta_k)}. \quad (2.31)$$

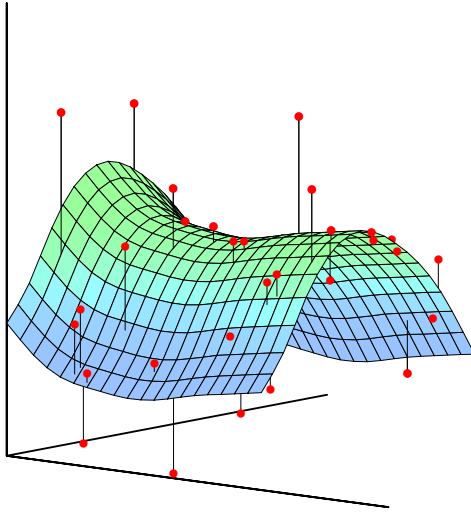
We can use least squares to estimate the parameters  $\theta$  in  $f_\theta$  as we did for the linear model, by minimizing the residual sum-of-squares

$$\text{RSS}(\theta) = \sum_{i=1}^N (y_i - f_\theta(x_i))^2 \quad (2.32)$$

as a function of  $\theta$ . This seems a reasonable criterion for an additive error model. In terms of function approximation, we imagine our parameterized function as a surface in  $p + 1$  space, and what we observe are noisy realizations from it. This is easy to visualize when  $p = 2$  and the vertical coordinate is the output  $y$ , as in Figure 2.10. The noise is in the output coordinate, so we find the set of parameters such that the fitted surface gets as close to the observed points as possible, where close is measured by the sum of squared vertical errors in  $\text{RSS}(\theta)$ .

For the linear model we get a simple closed form solution to the minimization problem. This is also true for the basis function methods, if the basis functions themselves do not have any hidden parameters. Otherwise the solution requires either iterative methods or numerical optimization.

While least squares is generally very convenient, it is not the only criterion used and in some cases would not make much sense. A more general



**FIGURE 2.10.** *Least squares fitting of a function of two inputs. The parameters of  $f_\theta(x)$  are chosen so as to minimize the sum-of-squared vertical errors.*

principle for estimation is *maximum likelihood estimation*. Suppose we have a random sample  $y_i$ ,  $i = 1, \dots, N$  from a density  $\text{Pr}_\theta(y)$  indexed by some parameters  $\theta$ . The log-probability of the observed sample is

$$L(\theta) = \sum_{i=1}^N \log \text{Pr}_\theta(y_i). \quad (2.33)$$

The principle of maximum likelihood assumes that the most reasonable values for  $\theta$  are those for which the probability of the observed sample is largest. Least squares for the additive error model  $Y = f_\theta(X) + \varepsilon$ , with  $\varepsilon \sim N(0, \sigma^2)$ , is equivalent to maximum likelihood using the conditional likelihood

$$\text{Pr}(Y|X, \theta) = N(f_\theta(X), \sigma^2). \quad (2.34)$$

So although the additional assumption of normality seems more restrictive, the results are the same. The log-likelihood of the data is

$$L(\theta) = -\frac{N}{2} \log(2\pi) - N \log \sigma - \frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - f_\theta(x_i))^2, \quad (2.35)$$

and the only term involving  $\theta$  is the last, which is  $\text{RSS}(\theta)$  up to a scalar negative multiplier.

A more interesting example is the multinomial likelihood for the regression function  $\text{Pr}(G|X)$  for a qualitative output  $G$ . Suppose we have a model  $\text{Pr}(G = \mathcal{G}_k|X = x) = p_{k,\theta}(x)$ ,  $k = 1, \dots, K$  for the conditional probability of each class given  $X$ , indexed by the parameter vector  $\theta$ . Then the

log-likelihood (also referred to as the cross-entropy) is

$$L(\theta) = \sum_{i=1}^N \log p_{g_i, \theta}(x_i), \quad (2.36)$$

and when maximized it delivers values of  $\theta$  that best conform with the data in this likelihood sense.

## 2.7 Structured Regression Models

We have seen that although nearest-neighbor and other local methods focus directly on estimating the function at a point, they face problems in high dimensions. They may also be inappropriate even in low dimensions in cases where more structured approaches can make more efficient use of the data. This section introduces classes of such structured approaches. Before we proceed, though, we discuss further the need for such classes.

### 2.7.1 *Difficulty of the Problem*

Consider the RSS criterion for an arbitrary function  $f$ ,

$$\text{RSS}(f) = \sum_{i=1}^N (y_i - f(x_i))^2. \quad (2.37)$$

Minimizing (2.37) leads to infinitely many solutions: any function  $\hat{f}$  passing through the training points  $(x_i, y_i)$  is a solution. Any particular solution chosen might be a poor predictor at test points different from the training points. If there are multiple observation pairs  $x_i, y_{i\ell}$ ,  $\ell = 1, \dots, N_i$  at each value of  $x_i$ , the risk is limited. In this case, the solutions pass through the average values of the  $y_{i\ell}$  at each  $x_i$ ; see Exercise 2.6. The situation is similar to the one we have already visited in Section 2.4; indeed, (2.37) is the finite sample version of (2.11) on page 18. If the sample size  $N$  were sufficiently large such that repeats were guaranteed and densely arranged, it would seem that these solutions might all tend to the limiting conditional expectation.

In order to obtain useful results for finite  $N$ , we must restrict the eligible solutions to (2.37) to a smaller set of functions. How to decide on the nature of the restrictions is based on considerations outside of the data. These restrictions are sometimes encoded via the parametric representation of  $f_\theta$ , or may be built into the learning method itself, either implicitly or explicitly. These restricted classes of solutions are the major topic of this book. One thing should be clear, though. Any restrictions imposed on  $f$  that lead to a unique solution to (2.37) do not really remove the ambiguity

caused by the multiplicity of solutions. There are infinitely many possible restrictions, each leading to a unique solution, so the ambiguity has simply been transferred to the choice of constraint.

In general the constraints imposed by most learning methods can be described as *complexity* restrictions of one kind or another. This usually means some kind of regular behavior in small neighborhoods of the input space. That is, for all input points  $x$  sufficiently close to each other in some metric,  $\hat{f}$  exhibits some special structure such as nearly constant, linear or low-order polynomial behavior. The estimator is then obtained by averaging or polynomial fitting in that neighborhood.

The strength of the constraint is dictated by the neighborhood size. The larger the size of the neighborhood, the stronger the constraint, and the more sensitive the solution is to the particular choice of constraint. For example, local constant fits in infinitesimally small neighborhoods is no constraint at all; local linear fits in very large neighborhoods is almost a globally linear model, and is very restrictive.

The nature of the constraint depends on the metric used. Some methods, such as kernel and local regression and tree-based methods, directly specify the metric and size of the neighborhood. The nearest-neighbor methods discussed so far are based on the assumption that locally the function is constant; close to a target input  $x_0$ , the function does not change much, and so close outputs can be averaged to produce  $\hat{f}(x_0)$ . Other methods such as splines, neural networks and basis-function methods implicitly define neighborhoods of local behavior. In Section 5.4.1 we discuss the concept of an *equivalent kernel* (see Figure 5.8 on page 157), which describes this local dependence for any method linear in the outputs. These equivalent kernels in many cases look just like the explicitly defined weighting kernels discussed above—peaked at the target point and falling away smoothly away from it.

One fact should be clear by now. Any method that attempts to produce locally varying functions in small isotropic neighborhoods will run into problems in high dimensions—again the curse of dimensionality. And conversely, all methods that overcome the dimensionality problems have an associated—and often implicit or adaptive—metric for measuring neighborhoods, which basically does not allow the neighborhood to be simultaneously small in all directions.

## 2.8 Classes of Restricted Estimators

The variety of nonparametric regression techniques or learning methods fall into a number of different classes depending on the nature of the restrictions imposed. These classes are not distinct, and indeed some methods fall in several classes. Here we give a brief summary, since detailed descriptions

are given in later chapters. Each of the classes has associated with it one or more parameters, sometimes appropriately called *smoothing* parameters, that control the effective size of the local neighborhood. Here we describe three broad classes.

### 2.8.1 Roughness Penalty and Bayesian Methods

Here the class of functions is controlled by explicitly penalizing  $\text{RSS}(f)$  with a roughness penalty

$$\text{PRSS}(f; \lambda) = \text{RSS}(f) + \lambda J(f). \quad (2.38)$$

The user-selected functional  $J(f)$  will be large for functions  $f$  that vary too rapidly over small regions of input space. For example, the popular *cubic smoothing spline* for one-dimensional inputs is the solution to the penalized least-squares criterion

$$\text{PRSS}(f; \lambda) = \sum_{i=1}^N (y_i - f(x_i))^2 + \lambda \int [f''(x)]^2 dx. \quad (2.39)$$

The roughness penalty here controls large values of the second derivative of  $f$ , and the amount of penalty is dictated by  $\lambda \geq 0$ . For  $\lambda = 0$  no penalty is imposed, and any interpolating function will do, while for  $\lambda = \infty$  only functions linear in  $x$  are permitted.

Penalty functionals  $J$  can be constructed for functions in any dimension, and special versions can be created to impose special structure. For example, additive penalties  $J(f) = \sum_{j=1}^p J(f_j)$  are used in conjunction with additive functions  $f(X) = \sum_{j=1}^p f_j(X_j)$  to create additive models with smooth coordinate functions. Similarly, *projection pursuit regression* models have  $f(X) = \sum_{m=1}^M g_m(\alpha_m^T X)$  for adaptively chosen directions  $\alpha_m$ , and the functions  $g_m$  can each have an associated roughness penalty.

Penalty function, or *regularization* methods, express our prior belief that the type of functions we seek exhibit a certain type of smooth behavior, and indeed can usually be cast in a Bayesian framework. The penalty  $J$  corresponds to a log-prior, and  $\text{PRSS}(f; \lambda)$  the log-posterior distribution, and minimizing  $\text{PRSS}(f; \lambda)$  amounts to finding the posterior mode. We discuss roughness-penalty approaches in Chapter 5 and the Bayesian paradigm in Chapter 8.

### 2.8.2 Kernel Methods and Local Regression

These methods can be thought of as explicitly providing estimates of the regression function or conditional expectation by specifying the nature of the local neighborhood, and of the class of regular functions fitted locally. The local neighborhood is specified by a *kernel function*  $K_\lambda(x_0, x)$  which assigns

weights to points  $x$  in a region around  $x_0$  (see Figure 6.1 on page 192). For example, the Gaussian kernel has a weight function based on the Gaussian density function

$$K_\lambda(x_0, x) = \frac{1}{\lambda} \exp \left[ -\frac{\|x - x_0\|^2}{2\lambda} \right] \quad (2.40)$$

and assigns weights to points that die exponentially with their squared Euclidean distance from  $x_0$ . The parameter  $\lambda$  corresponds to the variance of the Gaussian density, and controls the width of the neighborhood. The simplest form of kernel estimate is the Nadaraya–Watson weighted average

$$\hat{f}(x_0) = \frac{\sum_{i=1}^N K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^N K_\lambda(x_0, x_i)}. \quad (2.41)$$

In general we can define a local regression estimate of  $f(x_0)$  as  $f_{\hat{\theta}}(x_0)$ , where  $\hat{\theta}$  minimizes

$$\text{RSS}(f_\theta, x_0) = \sum_{i=1}^N K_\lambda(x_0, x_i) (y_i - f_\theta(x_i))^2, \quad (2.42)$$

and  $f_\theta$  is some parameterized function, such as a low-order polynomial. Some examples are:

- $f_\theta(x) = \theta_0$ , the constant function; this results in the Nadaraya–Watson estimate in (2.41) above.
- $f_\theta(x) = \theta_0 + \theta_1 x$  gives the popular local linear regression model.

Nearest-neighbor methods can be thought of as kernel methods having a more data-dependent metric. Indeed, the metric for  $k$ -nearest neighbors is

$$K_k(x, x_0) = I(\|x - x_0\| \leq \|x_{(k)} - x_0\|),$$

where  $x_{(k)}$  is the training observation ranked  $k$ th in distance from  $x_0$ , and  $I(S)$  is the indicator of the set  $S$ .

These methods of course need to be modified in high dimensions, to avoid the curse of dimensionality. Various adaptations are discussed in Chapter 6.

### 2.8.3 Basis Functions and Dictionary Methods

This class of methods includes the familiar linear and polynomial expansions, but more importantly a wide variety of more flexible models. The model for  $f$  is a linear expansion of basis functions

$$f_\theta(x) = \sum_{m=1}^M \theta_m h_m(x), \quad (2.43)$$

where each of the  $h_m$  is a function of the input  $x$ , and the term linear here refers to the action of the parameters  $\theta$ . This class covers a wide variety of methods. In some cases the sequence of basis functions is prescribed, such as a basis for polynomials in  $x$  of total degree  $M$ .

For one-dimensional  $x$ , polynomial splines of degree  $K$  can be represented by an appropriate sequence of  $M$  spline basis functions, determined in turn by  $M - K - 1$  *knots*. These produce functions that are piecewise polynomials of degree  $K$  between the knots, and joined up with continuity of degree  $K - 1$  at the knots. As an example consider linear splines, or piecewise linear functions. One intuitively satisfying basis consists of the functions  $b_1(x) = 1$ ,  $b_2(x) = x$ , and  $b_{m+2}(x) = (x - t_m)_+$ ,  $m = 1, \dots, M - 2$ , where  $t_m$  is the  $m$ th knot, and  $z_+$  denotes positive part. Tensor products of spline bases can be used for inputs with dimensions larger than one (see Section 5.2, and the CART and MARS models in Chapter 9.) The parameter  $M$  controls the degree of the polynomial or the number of knots in the case of splines.

*Radial basis functions* are symmetric  $p$ -dimensional kernels located at particular centroids,

$$f_\theta(x) = \sum_{m=1}^M K_{\lambda_m}(\mu_m, x) \theta_m; \quad (2.44)$$

for example, the Gaussian kernel  $K_\lambda(\mu, x) = e^{-\|x - \mu\|^2 / 2\lambda}$  is popular.

Radial basis functions have centroids  $\mu_m$  and scales  $\lambda_m$  that have to be determined. The spline basis functions have knots. In general we would like the data to dictate them as well. Including these as parameters changes the regression problem from a straightforward linear problem to a combinatorially hard nonlinear problem. In practice, shortcuts such as greedy algorithms or two stage processes are used. Section 6.7 describes some such approaches.

A single-layer feed-forward neural network model with linear output weights can be thought of as an adaptive basis function method. The model has the form

$$f_\theta(x) = \sum_{m=1}^M \beta_m \sigma(\alpha_m^T x + b_m), \quad (2.45)$$

where  $\sigma(x) = 1/(1 + e^{-x})$  is known as the *activation* function. Here, as in the projection pursuit model, the directions  $\alpha_m$  and the *bias* terms  $b_m$  have to be determined, and their estimation is the meat of the computation. Details are given in Chapter 11.

These adaptively chosen basis function methods are also known as *dictionary* methods, where one has available a possibly infinite set or dictionary  $\mathcal{D}$  of candidate basis functions from which to choose, and models are built up by employing some kind of search mechanism.

## 2.9 Model Selection and the Bias–Variance Tradeoff

All the models described above and many others discussed in later chapters have a *smoothing* or *complexity* parameter that has to be determined:

- the multiplier of the penalty term;
- the width of the kernel;
- or the number of basis functions.

In the case of the smoothing spline, the parameter  $\lambda$  indexes models ranging from a straight line fit to the interpolating model. Similarly a local degree- $m$  polynomial model ranges between a degree- $m$  global polynomial when the window size is infinitely large, to an interpolating fit when the window size shrinks to zero. This means that we cannot use residual sum-of-squares on the training data to determine these parameters as well, since we would always pick those that gave interpolating fits and hence zero residuals. Such a model is unlikely to predict future data well at all.

The  $k$ -nearest-neighbor regression fit  $\hat{f}_k(x_0)$  usefully illustrates the competing forces that affect the predictive ability of such approximations. Suppose the data arise from a model  $Y = f(X) + \varepsilon$ , with  $E(\varepsilon) = 0$  and  $\text{Var}(\varepsilon) = \sigma^2$ . For simplicity here we assume that the values of  $x_i$  in the sample are fixed in advance (nonrandom). The expected prediction error at  $x_0$ , also known as *test* or *generalization* error, can be decomposed:

$$\begin{aligned} \text{EPE}_k(x_0) &= E[(Y - \hat{f}_k(x_0))^2 | X = x_0] \\ &= \sigma^2 + [\text{Bias}^2(\hat{f}_k(x_0)) + \text{Var}_{\mathcal{T}}(\hat{f}_k(x_0))] \end{aligned} \quad (2.46)$$

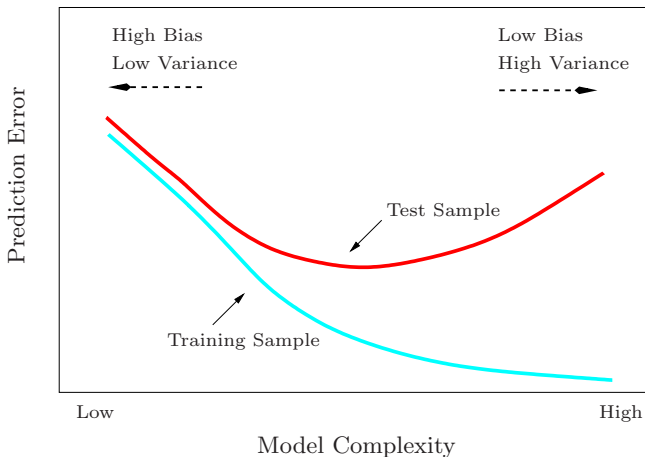
$$= \sigma^2 + \left[ f(x_0) - \frac{1}{k} \sum_{\ell=1}^k f(x_{(\ell)}) \right]^2 + \frac{\sigma^2}{k}. \quad (2.47)$$

The subscripts in parentheses ( $\ell$ ) indicate the sequence of nearest neighbors to  $x_0$ .

There are three terms in this expression. The first term  $\sigma^2$  is the *irreducible* error—the variance of the new test target—and is beyond our control, even if we know the true  $f(x_0)$ .

The second and third terms are under our control, and make up the *mean squared error* of  $\hat{f}_k(x_0)$  in estimating  $f(x_0)$ , which is broken down into a bias component and a variance component. The bias term is the squared difference between the true mean  $f(x_0)$  and the expected value of the estimate— $[E_{\mathcal{T}}(\hat{f}_k(x_0)) - f(x_0)]^2$ —where the expectation averages the randomness in the training data. This term will most likely increase with  $k$ , if the true function is reasonably smooth. For small  $k$  the few closest neighbors will have values  $f(x_{(\ell)})$  close to  $f(x_0)$ , so their average should





**FIGURE 2.11.** Test and training error as a function of model complexity.

be close to  $f(x_0)$ . As  $k$  grows, the neighbors are further away, and then anything can happen.

The variance term is simply the variance of an average here, and decreases as the inverse of  $k$ . So as  $k$  varies, there is a *bias–variance tradeoff*.

More generally, as the *model complexity* of our procedure is increased, the variance tends to increase and the squared bias tends to decrease. The opposite behavior occurs as the model complexity is decreased. For  $k$ -nearest neighbors, the model complexity is controlled by  $k$ .

Typically we would like to choose our model complexity to trade bias off with variance in such a way as to minimize the test error. An obvious estimate of test error is the *training error*  $\frac{1}{N} \sum_i (y_i - \hat{y}_i)^2$ . Unfortunately training error is not a good estimate of test error, as it does not properly account for model complexity.

Figure 2.11 shows the typical behavior of the test and training error, as model complexity is varied. The training error tends to decrease whenever we increase the model complexity, that is, whenever we fit the data harder. However with too much fitting, the model adapts itself too closely to the training data, and will not generalize well (i.e., have large test error). In that case the predictions  $\hat{f}(x_0)$  will have large variance, as reflected in the last term of expression (2.46). In contrast, if the model is not complex enough, it will *underfit* and may have large bias, again resulting in poor generalization. In Chapter 7 we discuss methods for estimating the test error of a prediction method, and hence estimating the optimal amount of model complexity for a given prediction method and training set.

## Bibliographic Notes

Some good general books on the learning problem are Duda et al. (2000), Bishop (1995), (Bishop, 2006), Ripley (1996), Cherkassky and Mulier (2007) and Vapnik (1996). Parts of this chapter are based on Friedman (1994b).

## Exercises

**Ex. 2.1** Suppose each of  $K$ -classes has an associated target  $t_k$ , which is a vector of all zeros, except a one in the  $k$ th position. Show that classifying to the largest element of  $\hat{y}$  amounts to choosing the closest target,  $\min_k \|t_k - \hat{y}\|$ , if the elements of  $\hat{y}$  sum to one.

**Ex. 2.2** Show how to compute the Bayes decision boundary for the simulation example in Figure 2.5.

**Ex. 2.3** Derive equation (2.24).

**Ex. 2.4** The edge effect problem discussed on page 23 is not peculiar to uniform sampling from bounded domains. Consider inputs drawn from a spherical multinormal distribution  $X \sim N(0, \mathbf{I}_p)$ . The squared distance from any sample point to the origin has a  $\chi_p^2$  distribution with mean  $p$ . Consider a prediction point  $x_0$  drawn from this distribution, and let  $a = x_0/\|x_0\|$  be an associated unit vector. Let  $z_i = a^T x_i$  be the projection of each of the training points on this direction.

Show that the  $z_i$  are distributed  $N(0, 1)$  with expected squared distance from the origin 1, while the target point has expected squared distance  $p$  from the origin.

Hence for  $p = 10$ , a randomly drawn test point is about 3.1 standard deviations from the origin, while all the training points are on average one standard deviation along direction  $a$ . So most prediction points see themselves as lying on the edge of the training set.

**Ex. 2.5**

- (a) Derive equation (2.27). The last line makes use of (3.8) through a conditioning argument.
- (b) Derive equation (2.28), making use of the *cyclic* property of the trace operator [ $\text{trace}(AB) = \text{trace}(BA)$ ], and its linearity (which allows us to interchange the order of trace and expectation).

**Ex. 2.6** Consider a regression problem with inputs  $x_i$  and outputs  $y_i$ , and a parameterized model  $f_\theta(x)$  to be fit by least squares. Show that if there are observations with *tied* or *identical* values of  $x$ , then the fit can be obtained from a reduced weighted least squares problem.

**Ex. 2.7** Suppose we have a sample of  $N$  pairs  $x_i, y_i$  drawn i.i.d. from the distribution characterized as follows:

$$\begin{aligned} x_i &\sim h(x), \text{ the design density} \\ y_i &= f(x_i) + \varepsilon_i, \text{ } f \text{ is the regression function} \\ \varepsilon_i &\sim (0, \sigma^2) \text{ (mean zero, variance } \sigma^2) \end{aligned}$$

We construct an estimator for  $f$  linear in the  $y_i$ ,

$$\hat{f}(x_0) = \sum_{i=1}^N \ell_i(x_0; \mathcal{X}) y_i,$$

where the weights  $\ell_i(x_0; \mathcal{X})$  do not depend on the  $y_i$ , but do depend on the entire training sequence of  $x_i$ , denoted here by  $\mathcal{X}$ .

- (a) Show that linear regression and  $k$ -nearest-neighbor regression are members of this class of estimators. Describe explicitly the weights  $\ell_i(x_0; \mathcal{X})$  in each of these cases.
- (b) Decompose the conditional mean-squared error

$$E_{\mathcal{Y}|\mathcal{X}}(f(x_0) - \hat{f}(x_0))^2$$

into a conditional squared bias and a conditional variance component. Like  $\mathcal{X}$ ,  $\mathcal{Y}$  represents the entire training sequence of  $y_i$ .

- (c) Decompose the (unconditional) mean-squared error

$$E_{\mathcal{Y}, \mathcal{X}}(f(x_0) - \hat{f}(x_0))^2$$

into a squared bias and a variance component.

- (d) Establish a relationship between the squared biases and variances in the above two cases.

**Ex. 2.8** Compare the classification performance of linear regression and  $k$ -nearest neighbor classification on the `zipcode` data. In particular, consider only the 2's and 3's, and  $k = 1, 3, 5, 7$  and 15. Show both the training and test error for each choice. The `zipcode` data are available from the book website [www-stat.stanford.edu/ElemStatLearn](http://www-stat.stanford.edu/ElemStatLearn).

**Ex. 2.9** Consider a linear regression model with  $p$  parameters, fit by least squares to a set of training data  $(x_1, y_1), \dots, (x_N, y_N)$  drawn at random from a population. Let  $\hat{\beta}$  be the least squares estimate. Suppose we have some test data  $(\tilde{x}_1, \tilde{y}_1), \dots, (\tilde{x}_M, \tilde{y}_M)$  drawn at random from the same population as the training data. If  $R_{tr}(\beta) = \frac{1}{N} \sum_{i=1}^N (y_i - \beta^T x_i)^2$  and  $R_{te}(\beta) = \frac{1}{M} \sum_{i=1}^M (\tilde{y}_i - \beta^T \tilde{x}_i)^2$ , prove that

$$E[R_{tr}(\hat{\beta})] \leq E[R_{te}(\hat{\beta})],$$

where the expectations are over all that is random in each expression. [This exercise was brought to our attention by Ryan Tibshirani, from a homework assignment given by Andrew Ng.]



# 3

## Linear Methods for Regression

### 3.1 Introduction

A linear regression model assumes that the regression function  $E(Y|X)$  is linear in the inputs  $X_1, \dots, X_p$ . Linear models were largely developed in the precomputer age of statistics, but even in today's computer era there are still good reasons to study and use them. They are simple and often provide an adequate and interpretable description of how the inputs affect the output. For prediction purposes they can sometimes outperform fancier nonlinear models, especially in situations with small numbers of training cases, low signal-to-noise ratio or sparse data. Finally, linear methods can be applied to transformations of the inputs and this considerably expands their scope. These generalizations are sometimes called basis-function methods, and are discussed in Chapter 5.

In this chapter we describe linear methods for regression, while in the next chapter we discuss linear methods for classification. On some topics we go into considerable detail, as it is our firm belief that an understanding of linear methods is essential for understanding nonlinear ones. In fact, many nonlinear techniques are direct generalizations of the linear methods discussed here.

## 3.2 Linear Regression Models and Least Squares

As introduced in Chapter 2, we have an input vector  $X^T = (X_1, X_2, \dots, X_p)$ , and want to predict a real-valued output  $Y$ . The linear regression model has the form

$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j. \quad (3.1)$$

The linear model either assumes that the regression function  $E(Y|X)$  is linear, or that the linear model is a reasonable approximation. Here the  $\beta_j$ 's are unknown parameters or coefficients, and the variables  $X_j$  can come from different sources:

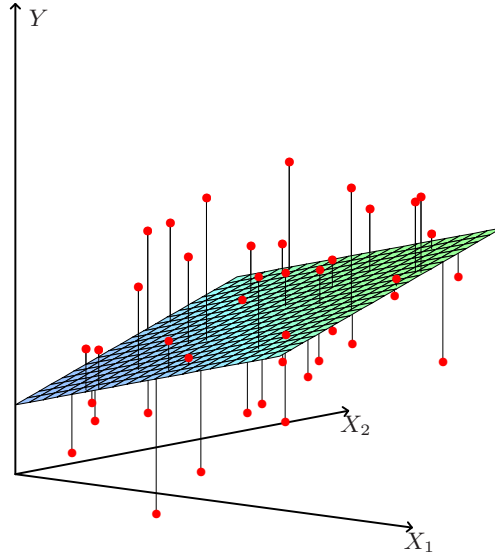
- quantitative inputs;
- transformations of quantitative inputs, such as log, square-root or square;
- basis expansions, such as  $X_2 = X_1^2$ ,  $X_3 = X_1^3$ , leading to a polynomial representation;
- numeric or “dummy” coding of the levels of qualitative inputs. For example, if  $G$  is a five-level factor input, we might create  $X_j$ ,  $j = 1, \dots, 5$ , such that  $X_j = I(G = j)$ . Together this group of  $X_j$  represents the effect of  $G$  by a set of level-dependent constants, since in  $\sum_{j=1}^5 X_j \beta_j$ , one of the  $X_j$ s is one, and the others are zero.
- interactions between variables, for example,  $X_3 = X_1 \cdot X_2$ .

No matter the source of the  $X_j$ , the model is linear in the parameters.

Typically we have a set of training data  $(x_1, y_1) \dots (x_N, y_N)$  from which to estimate the parameters  $\beta$ . Each  $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T$  is a vector of feature measurements for the  $i$ th case. The most popular estimation method is *least squares*, in which we pick the coefficients  $\beta = (\beta_0, \beta_1, \dots, \beta_p)^T$  to minimize the residual sum of squares

$$\begin{aligned} \text{RSS}(\beta) &= \sum_{i=1}^N (y_i - f(x_i))^2 \\ &= \sum_{i=1}^N \left( y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2. \end{aligned} \quad (3.2)$$

From a statistical point of view, this criterion is reasonable if the training observations  $(x_i, y_i)$  represent independent random draws from their population. Even if the  $x_i$ 's were not drawn randomly, the criterion is still valid if the  $y_i$ 's are conditionally independent given the inputs  $x_i$ . Figure 3.1 illustrates the geometry of least-squares fitting in the  $\mathbb{R}^{p+1}$ -dimensional



**FIGURE 3.1.** Linear least squares fitting with  $X \in \mathbb{R}^2$ . We seek the linear function of  $X$  that minimizes the sum of squared residuals from  $Y$ .

space occupied by the pairs  $(X, Y)$ . Note that (3.2) makes no assumptions about the validity of model (3.1); it simply finds the best linear fit to the data. Least squares fitting is intuitively satisfying no matter how the data arise; the criterion measures the average lack of fit.

How do we minimize (3.2)? Denote by  $\mathbf{X}$  the  $N \times (p + 1)$  matrix with each row an input vector (with a 1 in the first position), and similarly let  $\mathbf{y}$  be the  $N$ -vector of outputs in the training set. Then we can write the residual sum-of-squares as

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta). \quad (3.3)$$

This is a quadratic function in the  $p + 1$  parameters. Differentiating with respect to  $\beta$  we obtain

$$\begin{aligned} \frac{\partial \text{RSS}}{\partial \beta} &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) \\ \frac{\partial^2 \text{RSS}}{\partial \beta \partial \beta^T} &= 2\mathbf{X}^T\mathbf{X}. \end{aligned} \quad (3.4)$$

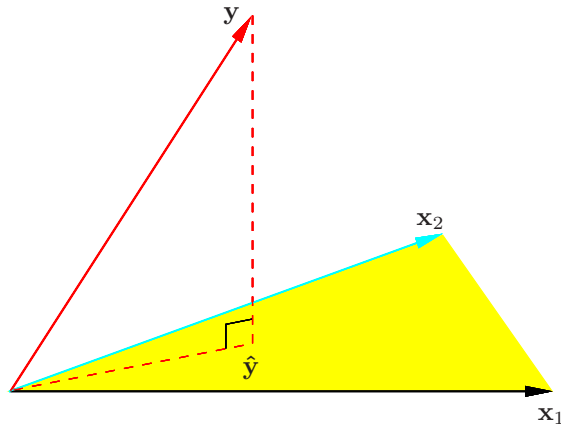
Assuming (for the moment) that  $\mathbf{X}$  has full column rank, and hence  $\mathbf{X}^T\mathbf{X}$  is positive definite, we set the first derivative to zero

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \quad (3.5)$$

to obtain the unique solution

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \quad (3.6)$$





**FIGURE 3.2.** The  $N$ -dimensional geometry of least squares regression with two predictors. The outcome vector  $\mathbf{y}$  is orthogonally projected onto the hyperplane spanned by the input vectors  $\mathbf{x}_1$  and  $\mathbf{x}_2$ . The projection  $\hat{\mathbf{y}}$  represents the vector of the least squares predictions

The predicted values at an input vector  $x_0$  are given by  $\hat{f}(x_0) = (1 : x_0)^T \hat{\beta}$ ; the fitted values at the training inputs are

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \quad (3.7)$$

where  $\hat{y}_i = \hat{f}(x_i)$ . The matrix  $\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$  appearing in equation (3.7) is sometimes called the “hat” matrix because it puts the hat on  $\mathbf{y}$ .

Figure 3.2 shows a different geometrical representation of the least squares estimate, this time in  $\mathbb{R}^N$ . We denote the column vectors of  $\mathbf{X}$  by  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_p$ , with  $\mathbf{x}_0 \equiv 1$ . For much of what follows, this first column is treated like any other. These vectors span a subspace of  $\mathbb{R}^N$ , also referred to as the column space of  $\mathbf{X}$ . We minimize  $\text{RSS}(\beta) = \|\mathbf{y} - \mathbf{X}\beta\|^2$  by choosing  $\hat{\beta}$  so that the residual vector  $\mathbf{y} - \hat{\mathbf{y}}$  is orthogonal to this subspace. This orthogonality is expressed in (3.5), and the resulting estimate  $\hat{\mathbf{y}}$  is hence the *orthogonal projection* of  $\mathbf{y}$  onto this subspace. The hat matrix  $\mathbf{H}$  computes the orthogonal projection, and hence it is also known as a projection matrix.

It might happen that the columns of  $\mathbf{X}$  are not linearly independent, so that  $\mathbf{X}$  is not of full rank. This would occur, for example, if two of the inputs were perfectly correlated, (e.g.,  $\mathbf{x}_2 = 3\mathbf{x}_1$ ). Then  $\mathbf{X}^T\mathbf{X}$  is singular and the least squares coefficients  $\hat{\beta}$  are not uniquely defined. However, the fitted values  $\hat{\mathbf{y}} = \mathbf{X}\hat{\beta}$  are still the projection of  $\mathbf{y}$  onto the column space of  $\mathbf{X}$ ; there is just more than one way to express that projection in terms of the column vectors of  $\mathbf{X}$ . The non-full-rank case occurs most often when one or more qualitative inputs are coded in a redundant fashion. There is usually a natural way to resolve the non-unique representation, by recoding and/or dropping redundant columns in  $\mathbf{X}$ . Most regression software packages detect these redundancies and automatically implement

some strategy for removing them. Rank deficiencies can also occur in signal and image analysis, where the number of inputs  $p$  can exceed the number of training cases  $N$ . In this case, the features are typically reduced by filtering or else the fitting is controlled by regularization (Section 5.2.3 and Chapter 18).

Up to now we have made minimal assumptions about the true distribution of the data. In order to pin down the sampling properties of  $\hat{\beta}$ , we now assume that the observations  $y_i$  are uncorrelated and have constant variance  $\sigma^2$ , and that the  $x_i$  are fixed (non random). The variance-covariance matrix of the least squares parameter estimates is easily derived from (3.6) and is given by

$$\text{Var}(\hat{\beta}) = (\mathbf{X}^T \mathbf{X})^{-1} \sigma^2. \quad (3.8)$$

Typically one estimates the variance  $\sigma^2$  by

$$\hat{\sigma}^2 = \frac{1}{N - p - 1} \sum_{i=1}^N (y_i - \hat{y}_i)^2.$$

The  $N - p - 1$  rather than  $N$  in the denominator makes  $\hat{\sigma}^2$  an unbiased estimate of  $\sigma^2$ :  $E(\hat{\sigma}^2) = \sigma^2$ .

To draw inferences about the parameters and the model, additional assumptions are needed. We now assume that (3.1) is the correct model for the mean; that is, the conditional expectation of  $Y$  is linear in  $X_1, \dots, X_p$ . We also assume that the deviations of  $Y$  around its expectation are additive and Gaussian. Hence

$$\begin{aligned} Y &= E(Y|X_1, \dots, X_p) + \varepsilon \\ &= \beta_0 + \sum_{j=1}^p X_j \beta_j + \varepsilon, \end{aligned} \quad (3.9)$$

where the error  $\varepsilon$  is a Gaussian random variable with expectation zero and variance  $\sigma^2$ , written  $\varepsilon \sim N(0, \sigma^2)$ .

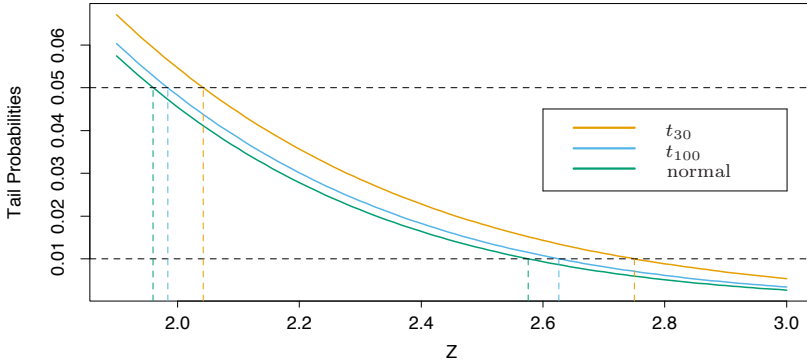
Under (3.9), it is easy to show that

$$\hat{\beta} \sim N(\beta, (\mathbf{X}^T \mathbf{X})^{-1} \sigma^2). \quad (3.10)$$

This is a multivariate normal distribution with mean vector and variance-covariance matrix as shown. Also

$$(N - p - 1) \hat{\sigma}^2 \sim \sigma^2 \chi_{N-p-1}^2, \quad (3.11)$$

a chi-squared distribution with  $N - p - 1$  degrees of freedom. In addition  $\hat{\beta}$  and  $\hat{\sigma}^2$  are statistically independent. We use these distributional properties to form tests of hypothesis and confidence intervals for the parameters  $\beta_j$ .



**FIGURE 3.3.** The tail probabilities  $\Pr(|Z| > z)$  for three distributions,  $t_{30}$ ,  $t_{100}$  and standard normal. Shown are the appropriate quantiles for testing significance at the  $p = 0.05$  and  $0.01$  levels. The difference between  $t$  and the standard normal becomes negligible for  $N$  bigger than about 100.

To test the hypothesis that a particular coefficient  $\beta_j = 0$ , we form the standardized coefficient or *Z-score*

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{v_j}}, \quad (3.12)$$

where  $v_j$  is the  $j$ th diagonal element of  $(\mathbf{X}^T \mathbf{X})^{-1}$ . Under the null hypothesis that  $\beta_j = 0$ ,  $z_j$  is distributed as  $t_{N-p-1}$  (a  $t$  distribution with  $N - p - 1$  degrees of freedom), and hence a large (absolute) value of  $z_j$  will lead to rejection of this null hypothesis. If  $\hat{\sigma}$  is replaced by a known value  $\sigma$ , then  $z_j$  would have a standard normal distribution. The difference between the tail quantiles of a  $t$ -distribution and a standard normal become negligible as the sample size increases, and so we typically use the normal quantiles (see Figure 3.3).

Often we need to test for the significance of groups of coefficients simultaneously. For example, to test if a categorical variable with  $k$  levels can be excluded from a model, we need to test whether the coefficients of the dummy variables used to represent the levels can all be set to zero. Here we use the  $F$  statistic,

$$F = \frac{(\text{RSS}_0 - \text{RSS}_1)/(p_1 - p_0)}{\text{RSS}_1/(N - p_1 - 1)}, \quad (3.13)$$

where  $\text{RSS}_1$  is the residual sum-of-squares for the least squares fit of the bigger model with  $p_1 + 1$  parameters, and  $\text{RSS}_0$  the same for the nested smaller model with  $p_0 + 1$  parameters, having  $p_1 - p_0$  parameters constrained to be

zero. The  $F$  statistic measures the change in residual sum-of-squares per additional parameter in the bigger model, and it is normalized by an estimate of  $\sigma^2$ . Under the Gaussian assumptions, and the null hypothesis that the smaller model is correct, the  $F$  statistic will have a  $F_{p_1-p_0, N-p_1-1}$  distribution. It can be shown (Exercise 3.1) that the  $z_j$  in (3.12) are equivalent to the  $F$  statistic for dropping the single coefficient  $\beta_j$  from the model. For large  $N$ , the quantiles of  $F_{p_1-p_0, N-p_1-1}$  approach those of  $\chi^2_{p_1-p_0}/(p_1-p_0)$ .

Similarly, we can isolate  $\beta_j$  in (3.10) to obtain a  $1-2\alpha$  confidence interval for  $\beta_j$ :

$$(\hat{\beta}_j - z^{(1-\alpha)} v_j^{\frac{1}{2}} \hat{\sigma}, \hat{\beta}_j + z^{(1-\alpha)} v_j^{\frac{1}{2}} \hat{\sigma}). \quad (3.14)$$

Here  $z^{(1-\alpha)}$  is the  $1-\alpha$  percentile of the normal distribution:

$$\begin{aligned} z^{(1-0.025)} &= 1.96, \\ z^{(1-.05)} &= 1.645, \text{ etc.} \end{aligned}$$

Hence the standard practice of reporting  $\hat{\beta} \pm 2 \cdot \text{se}(\hat{\beta})$  amounts to an approximate 95% confidence interval. Even if the Gaussian error assumption does not hold, this interval will be approximately correct, with its coverage approaching  $1-2\alpha$  as the sample size  $N \rightarrow \infty$ .

In a similar fashion we can obtain an approximate confidence set for the entire parameter vector  $\beta$ , namely

$$C_\beta = \{\beta | (\hat{\beta} - \beta)^T \mathbf{X}^T \mathbf{X} (\hat{\beta} - \beta) \leq \hat{\sigma}^2 \chi^2_{p+1}^{(1-\alpha)}\}, \quad (3.15)$$

where  $\chi^2_\ell^{(1-\alpha)}$  is the  $1-\alpha$  percentile of the chi-squared distribution on  $\ell$  degrees of freedom: for example,  $\chi^2_5^{(1-0.05)} = 11.1$ ,  $\chi^2_5^{(1-0.1)} = 9.2$ . This confidence set for  $\beta$  generates a corresponding confidence set for the true function  $f(x) = x^T \beta$ , namely  $\{x^T \beta | \beta \in C_\beta\}$  (Exercise 3.2; see also Figure 5.4 in Section 5.2.2 for examples of confidence bands for functions).

### 3.2.1 Example: Prostate Cancer

The data for this example come from a study by Stamey et al. (1989). They examined the correlation between the level of prostate-specific antigen and a number of clinical measures in men who were about to receive a radical prostatectomy. The variables are log cancer volume (`lcavol`), log prostate weight (`lweight`), `age`, log of the amount of benign prostatic hyperplasia (`lbph`), seminal vesicle invasion (`svi`), log of capsular penetration (`lcp`), Gleason score (`gleason`), and percent of Gleason scores 4 or 5 (`pgg45`). The correlation matrix of the predictors given in Table 3.1 shows many strong correlations. Figure 1.1 (page 3) of Chapter 1 is a scatterplot matrix showing every pairwise plot between the variables. We see that `svi` is a binary variable, and `gleason` is an ordered categorical variable. We see, for

**TABLE 3.1.** *Correlations of predictors in the prostate cancer data.*

	lcavol	lweight	age	lbph	svi	lcp	gleason
lweight	0.300						
age	0.286	0.317					
lbph	0.063	0.437	0.287				
svi	0.593	0.181	0.129	−0.139			
lcp	0.692	0.157	0.173	−0.089	0.671		
gleason	0.426	0.024	0.366	0.033	0.307	0.476	
pgg45	0.483	0.074	0.276	−0.030	0.481	0.663	0.757

**TABLE 3.2.** *Linear model fit to the prostate cancer data. The  $Z$  score is the coefficient divided by its standard error (3.12). Roughly a  $Z$  score larger than two in absolute value is significantly nonzero at the  $p = 0.05$  level.*

Term	Coefficient	Std. Error	$Z$ Score
Intercept	2.46	0.09	27.60
lcavol	0.68	0.13	5.37
lweight	0.26	0.10	2.75
age	−0.14	0.10	−1.40
lbph	0.21	0.10	2.06
svi	0.31	0.12	2.47
lcp	−0.29	0.15	−1.87
gleason	−0.02	0.15	−0.15
pgg45	0.27	0.15	1.74

example, that both `lcavol` and `lcp` show a strong relationship with the response `lpsa`, and with each other. We need to fit the effects jointly to untangle the relationships between the predictors and the response.

We fit a linear model to the log of prostate-specific antigen, `lpsa`, after first standardizing the predictors to have unit variance. We randomly split the dataset into a training set of size 67 and a test set of size 30. We applied least squares estimation to the training set, producing the estimates, standard errors and  $Z$ -scores shown in Table 3.2. The  $Z$ -scores are defined in (3.12), and measure the effect of dropping that variable from the model. A  $Z$ -score greater than 2 in absolute value is approximately significant at the 5% level. (For our example, we have nine parameters, and the 0.025 tail quantiles of the  $t_{67-9}$  distribution are  $\pm 2.002$ !) The predictor `lcavol` shows the strongest effect, with `lweight` and `svi` also strong. Notice that `lcp` is not significant, once `lcavol` is in the model (when used in a model without `lcavol`, `lcp` is strongly significant). We can also test for the exclusion of a number of terms at once, using the  $F$ -statistic (3.13). For example, we consider dropping all the non-significant terms in Table 3.2, namely `age`,

lcp, gleason, and pgg45. We get

$$F = \frac{(32.81 - 29.43)/(9 - 5)}{29.43/(67 - 9)} = 1.67, \quad (3.16)$$

which has a  $p$ -value of 0.17 ( $\Pr(F_{4,58} > 1.67) = 0.17$ ), and hence is not significant.

The mean prediction error on the test data is 0.521. In contrast, prediction using the mean training value of `lpsa` has a test error of 1.057, which is called the “base error rate.” Hence the linear model reduces the base error rate by about 50%. We will return to this example later to compare various selection and shrinkage methods.

### 3.2.2 The Gauss–Markov Theorem

One of the most famous results in statistics asserts that the least squares estimates of the parameters  $\beta$  have the smallest variance among all linear unbiased estimates. We will make this precise here, and also make clear that the restriction to unbiased estimates is not necessarily a wise one. This observation will lead us to consider biased estimates such as ridge regression later in the chapter. We focus on estimation of any linear combination of the parameters  $\theta = a^T \beta$ ; for example, predictions  $f(x_0) = x_0^T \beta$  are of this form. The least squares estimate of  $a^T \beta$  is

$$\hat{\theta} = a^T \hat{\beta} = a^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}. \quad (3.17)$$

Considering  $\mathbf{X}$  to be fixed, this is a linear function  $\mathbf{c}_0^T \mathbf{y}$  of the response vector  $\mathbf{y}$ . If we assume that the linear model is correct,  $a^T \hat{\beta}$  is unbiased since

$$\begin{aligned} E(a^T \hat{\beta}) &= E(a^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}) \\ &= a^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{X} \beta \\ &= a^T \beta. \end{aligned} \quad (3.18)$$

The Gauss–Markov theorem states that if we have any other linear estimator  $\tilde{\theta} = \mathbf{c}^T \mathbf{y}$  that is unbiased for  $a^T \beta$ , that is,  $E(\mathbf{c}^T \mathbf{y}) = a^T \beta$ , then

$$\text{Var}(a^T \hat{\beta}) \leq \text{Var}(\mathbf{c}^T \mathbf{y}). \quad (3.19)$$

The proof (Exercise 3.3) uses the triangle inequality. For simplicity we have stated the result in terms of estimation of a single parameter  $a^T \beta$ , but with a few more definitions one can state it in terms of the entire parameter vector  $\beta$  (Exercise 3.3).

Consider the mean squared error of an estimator  $\tilde{\theta}$  in estimating  $\theta$ :

$$\begin{aligned} \text{MSE}(\tilde{\theta}) &= E(\tilde{\theta} - \theta)^2 \\ &= \text{Var}(\tilde{\theta}) + [E(\tilde{\theta}) - \theta]^2. \end{aligned} \quad (3.20)$$

The first term is the variance, while the second term is the squared bias. The Gauss-Markov theorem implies that the least squares estimator has the smallest mean squared error of all linear estimators with no bias. However, there may well exist a biased estimator with smaller mean squared error. Such an estimator would trade a little bias for a larger reduction in variance. Biased estimates are commonly used. Any method that shrinks or sets to zero some of the least squares coefficients may result in a biased estimate. We discuss many examples, including variable subset selection and ridge regression, later in this chapter. From a more pragmatic point of view, most models are distortions of the truth, and hence are biased; picking the right model amounts to creating the right balance between bias and variance. We go into these issues in more detail in Chapter 7.

Mean squared error is intimately related to prediction accuracy, as discussed in Chapter 2. Consider the prediction of the new response at input  $x_0$ ,

$$Y_0 = f(x_0) + \varepsilon_0. \quad (3.21)$$

Then the expected prediction error of an estimate  $\tilde{f}(x_0) = x_0^T \tilde{\beta}$  is

$$\begin{aligned} \mathbb{E}(Y_0 - \tilde{f}(x_0))^2 &= \sigma^2 + \mathbb{E}(x_0^T \tilde{\beta} - f(x_0))^2 \\ &= \sigma^2 + \text{MSE}(\tilde{f}(x_0)). \end{aligned} \quad (3.22)$$

Therefore, expected prediction error and mean squared error differ only by the constant  $\sigma^2$ , representing the variance of the new observation  $y_0$ .

### 3.2.3 Multiple Regression from Simple Univariate Regression

The linear model (3.1) with  $p > 1$  inputs is called the *multiple linear regression model*. The least squares estimates (3.6) for this model are best understood in terms of the estimates for the *univariate* ( $p = 1$ ) linear model, as we indicate in this section.

Suppose first that we have a univariate model with no intercept, that is,

$$Y = X\beta + \varepsilon. \quad (3.23)$$

The least squares estimate and residuals are

$$\begin{aligned} \hat{\beta} &= \frac{\sum_1^N x_i y_i}{\sum_1^N x_i^2}, \\ r_i &= y_i - x_i \hat{\beta}. \end{aligned} \quad (3.24)$$

In convenient vector notation, we let  $\mathbf{y} = (y_1, \dots, y_N)^T$ ,  $\mathbf{x} = (x_1, \dots, x_N)^T$  and define

$$\begin{aligned} \langle \mathbf{x}, \mathbf{y} \rangle &= \sum_{i=1}^N x_i y_i, \\ &= \mathbf{x}^T \mathbf{y}, \end{aligned} \quad (3.25)$$

the *inner product* between  $\mathbf{x}$  and  $\mathbf{y}$ <sup>1</sup>. Then we can write

$$\begin{aligned}\hat{\beta} &= \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\langle \mathbf{x}, \mathbf{x} \rangle}, \\ \mathbf{r} &= \mathbf{y} - \mathbf{x}\hat{\beta}.\end{aligned}\tag{3.26}$$

As we will see, this simple univariate regression provides the building block for multiple linear regression. Suppose next that the inputs  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p$  (the columns of the data matrix  $\mathbf{X}$ ) are orthogonal; that is  $\langle \mathbf{x}_j, \mathbf{x}_k \rangle = 0$  for all  $j \neq k$ . Then it is easy to check that the multiple least squares estimates  $\hat{\beta}_j$  are equal to  $\langle \mathbf{x}_j, \mathbf{y} \rangle / \langle \mathbf{x}_j, \mathbf{x}_j \rangle$ —the univariate estimates. In other words, when the inputs are orthogonal, they have no effect on each other's parameter estimates in the model.

Orthogonal inputs occur most often with balanced, designed experiments (where orthogonality is enforced), but almost never with observational data. Hence we will have to orthogonalize them in order to carry this idea further. Suppose next that we have an intercept and a single input  $\mathbf{x}$ . Then the least squares coefficient of  $\mathbf{x}$  has the form

$$\hat{\beta}_1 = \frac{\langle \mathbf{x} - \bar{x}\mathbf{1}, \mathbf{y} \rangle}{\langle \mathbf{x} - \bar{x}\mathbf{1}, \mathbf{x} - \bar{x}\mathbf{1} \rangle},\tag{3.27}$$

where  $\bar{x} = \sum_i x_i / N$ , and  $\mathbf{1} = \mathbf{x}_0$ , the vector of  $N$  ones. We can view the estimate (3.27) as the result of two applications of the simple regression (3.26). The steps are:

1. regress  $\mathbf{x}$  on  $\mathbf{1}$  to produce the residual  $\mathbf{z} = \mathbf{x} - \bar{x}\mathbf{1}$ ;
2. regress  $\mathbf{y}$  on the residual  $\mathbf{z}$  to give the coefficient  $\hat{\beta}_1$ .

In this procedure, “regress  $\mathbf{b}$  on  $\mathbf{a}$ ” means a simple univariate regression of  $\mathbf{b}$  on  $\mathbf{a}$  with no intercept, producing coefficient  $\hat{\gamma} = \langle \mathbf{a}, \mathbf{b} \rangle / \langle \mathbf{a}, \mathbf{a} \rangle$  and residual vector  $\mathbf{b} - \hat{\gamma}\mathbf{a}$ . We say that  $\mathbf{b}$  is adjusted for  $\mathbf{a}$ , or is “orthogonalized” with respect to  $\mathbf{a}$ .

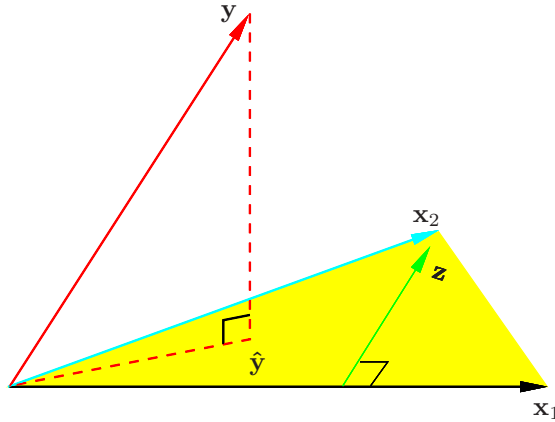
Step 1 orthogonalizes  $\mathbf{x}$  with respect to  $\mathbf{x}_0 = \mathbf{1}$ . Step 2 is just a simple univariate regression, using the orthogonal predictors  $\mathbf{1}$  and  $\mathbf{z}$ . Figure 3.4 shows this process for two general inputs  $\mathbf{x}_1$  and  $\mathbf{x}_2$ . The orthogonalization does not change the subspace spanned by  $\mathbf{x}_1$  and  $\mathbf{x}_2$ , it simply produces an orthogonal basis for representing it.

This recipe generalizes to the case of  $p$  inputs, as shown in Algorithm 3.1. Note that the inputs  $\mathbf{z}_0, \dots, \mathbf{z}_{j-1}$  in step 2 are orthogonal, hence the simple regression coefficients computed there are in fact also the multiple regression coefficients.

---

<sup>1</sup>The inner-product notation is suggestive of generalizations of linear regression to different metric spaces, as well as to probability spaces.





**FIGURE 3.4.** Least squares regression by orthogonalization of the inputs. The vector  $\mathbf{x}_2$  is regressed on the vector  $\mathbf{x}_1$ , leaving the residual vector  $\mathbf{z}$ . The regression of  $\mathbf{y}$  on  $\mathbf{z}$  gives the multiple regression coefficient of  $\mathbf{x}_2$ . Adding together the projections of  $\mathbf{y}$  on each of  $\mathbf{x}_1$  and  $\mathbf{z}$  gives the least squares fit  $\hat{\mathbf{y}}$ .

---

**Algorithm 3.1** Regression by Successive Orthogonalization.

---

1. Initialize  $\mathbf{z}_0 = \mathbf{x}_0 = \mathbf{1}$ .

2. For  $j = 1, 2, \dots, p$

Regress  $\mathbf{x}_j$  on  $\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_{j-1}$  to produce coefficients  $\hat{\gamma}_{\ell j} = \langle \mathbf{z}_\ell, \mathbf{x}_j \rangle / \langle \mathbf{z}_\ell, \mathbf{z}_\ell \rangle$ ,  $\ell = 0, \dots, j-1$  and residual vector  $\mathbf{z}_j = \mathbf{x}_j - \sum_{k=0}^{j-1} \hat{\gamma}_{kj} \mathbf{z}_k$ .

3. Regress  $\mathbf{y}$  on the residual  $\mathbf{z}_p$  to give the estimate  $\hat{\beta}_p$ .

---

The result of this algorithm is

$$\hat{\beta}_p = \frac{\langle \mathbf{z}_p, \mathbf{y} \rangle}{\langle \mathbf{z}_p, \mathbf{z}_p \rangle}. \quad (3.28)$$

Re-arranging the residual in step 2, we can see that each of the  $\mathbf{x}_j$  is a linear combination of the  $\mathbf{z}_k$ ,  $k \leq j$ . Since the  $\mathbf{z}_j$  are all orthogonal, they form a basis for the column space of  $\mathbf{X}$ , and hence the least squares projection onto this subspace is  $\hat{\mathbf{y}}$ . Since  $\mathbf{z}_p$  alone involves  $\mathbf{x}_p$  (with coefficient 1), we see that the coefficient (3.28) is indeed the multiple regression coefficient of  $\mathbf{y}$  on  $\mathbf{x}_p$ . This key result exposes the effect of correlated inputs in multiple regression. Note also that by rearranging the  $\mathbf{x}_j$ , any one of them could be in the last position, and a similar result holds. Hence stated more generally, we have shown that the  $j$ th multiple regression coefficient is the univariate regression coefficient of  $\mathbf{y}$  on  $\mathbf{x}_{j-012\dots(j-1)(j+1)\dots p}$ , the residual after regressing  $\mathbf{x}_j$  on  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{j-1}, \mathbf{x}_{j+1}, \dots, \mathbf{x}_p$ :

*The multiple regression coefficient  $\hat{\beta}_j$  represents the additional contribution of  $\mathbf{x}_j$  on  $\mathbf{y}$ , after  $\mathbf{x}_j$  has been adjusted for  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{j-1}, \mathbf{x}_{j+1}, \dots, \mathbf{x}_p$ .*

If  $\mathbf{x}_p$  is highly correlated with some of the other  $\mathbf{x}_k$ 's, the residual vector  $\mathbf{z}_p$  will be close to zero, and from (3.28) the coefficient  $\hat{\beta}_p$  will be very unstable. This will be true for all the variables in the correlated set. In such situations, we might have all the Z-scores (as in Table 3.2) be small—any one of the set can be deleted—yet we cannot delete them all. From (3.28) we also obtain an alternate formula for the variance estimates (3.8),

$$\text{Var}(\hat{\beta}_p) = \frac{\sigma^2}{\langle \mathbf{z}_p, \mathbf{z}_p \rangle} = \frac{\sigma^2}{\|\mathbf{z}_p\|^2}. \quad (3.29)$$

In other words, the precision with which we can estimate  $\hat{\beta}_p$  depends on the length of the residual vector  $\mathbf{z}_p$ ; this represents how much of  $\mathbf{x}_p$  is unexplained by the other  $\mathbf{x}_k$ 's.

Algorithm 3.1 is known as the *Gram–Schmidt* procedure for multiple regression, and is also a useful numerical strategy for computing the estimates. We can obtain from it not just  $\hat{\beta}_p$ , but also the entire multiple least squares fit, as shown in Exercise 3.4.

We can represent step 2 of Algorithm 3.1 in matrix form:

$$\mathbf{X} = \mathbf{Z}\mathbf{\Gamma}, \quad (3.30)$$

where  $\mathbf{Z}$  has as columns the  $\mathbf{z}_j$  (in order), and  $\mathbf{\Gamma}$  is the upper triangular matrix with entries  $\hat{\gamma}_{kj}$ . Introducing the diagonal matrix  $\mathbf{D}$  with  $j$ th diagonal entry  $D_{jj} = \|\mathbf{z}_j\|$ , we get

$$\begin{aligned} \mathbf{X} &= \mathbf{Z}\mathbf{D}^{-1}\mathbf{D}\mathbf{\Gamma} \\ &= \mathbf{Q}\mathbf{R}, \end{aligned} \quad (3.31)$$

the so-called *QR* decomposition of  $\mathbf{X}$ . Here  $\mathbf{Q}$  is an  $N \times (p+1)$  orthogonal matrix,  $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ , and  $\mathbf{R}$  is a  $(p+1) \times (p+1)$  upper triangular matrix.

The  $\mathbf{QR}$  decomposition represents a convenient orthogonal basis for the column space of  $\mathbf{X}$ . It is easy to see, for example, that the least squares solution is given by

$$\hat{\beta} = \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{y}, \quad (3.32)$$

$$\hat{\mathbf{y}} = \mathbf{Q}\mathbf{Q}^T\mathbf{y}. \quad (3.33)$$

Equation (3.32) is easy to solve because  $\mathbf{R}$  is upper triangular (Exercise 3.4).

### 3.2.4 Multiple Outputs

Suppose we have multiple outputs  $Y_1, Y_2, \dots, Y_K$  that we wish to predict from our inputs  $X_0, X_1, X_2, \dots, X_p$ . We assume a linear model for each output

$$Y_k = \beta_{0k} + \sum_{j=1}^p X_j \beta_{jk} + \varepsilon_k \quad (3.34)$$

$$= f_k(X) + \varepsilon_k. \quad (3.35)$$

With  $N$  training cases we can write the model in matrix notation

$$\mathbf{Y} = \mathbf{X}\mathbf{B} + \mathbf{E}. \quad (3.36)$$

Here  $\mathbf{Y}$  is the  $N \times K$  response matrix, with  $ik$  entry  $y_{ik}$ ,  $\mathbf{X}$  is the  $N \times (p+1)$  input matrix,  $\mathbf{B}$  is the  $(p+1) \times K$  matrix of parameters and  $\mathbf{E}$  is the  $N \times K$  matrix of errors. A straightforward generalization of the univariate loss function (3.2) is

$$\text{RSS}(\mathbf{B}) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2 \quad (3.37)$$

$$= \text{tr}[(\mathbf{Y} - \mathbf{X}\mathbf{B})^T (\mathbf{Y} - \mathbf{X}\mathbf{B})]. \quad (3.38)$$

The least squares estimates have exactly the same form as before

$$\hat{\mathbf{B}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}. \quad (3.39)$$

Hence the coefficients for the  $k$ th outcome are just the least squares estimates in the regression of  $\mathbf{y}_k$  on  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_p$ . Multiple outputs do not affect one another's least squares estimates.

If the errors  $\varepsilon = (\varepsilon_1, \dots, \varepsilon_K)$  in (3.34) are correlated, then it might seem appropriate to modify (3.37) in favor of a multivariate version. Specifically, suppose  $\text{Cov}(\varepsilon) = \mathbf{\Sigma}$ , then the multivariate weighted criterion

$$\text{RSS}(\mathbf{B}; \mathbf{\Sigma}) = \sum_{i=1}^N (y_i - f(x_i))^T \mathbf{\Sigma}^{-1} (y_i - f(x_i)) \quad (3.40)$$

arises naturally from multivariate Gaussian theory. Here  $f(x)$  is the vector function  $(f_1(x), \dots, f_K(x))^T$ , and  $y_i$  the vector of  $K$  responses for observation  $i$ . However, it can be shown that again the solution is given by (3.39);  $K$  separate regressions that ignore the correlations (Exercise 3.11). If the  $\mathbf{\Sigma}_i$  vary among observations, then this is no longer the case, and the solution for  $\mathbf{B}$  no longer decouples.

In Section 3.7 we pursue the multiple outcome problem, and consider situations where it does pay to combine the regressions.

## 3.3 Subset Selection

There are two reasons why we are often not satisfied with the least squares estimates (3.6).

- The first is *prediction accuracy*: the least squares estimates often have low bias but large variance. Prediction accuracy can sometimes be improved by shrinking or setting some coefficients to zero. By doing so we sacrifice a little bit of bias to reduce the variance of the predicted values, and hence may improve the overall prediction accuracy.
- The second reason is *interpretation*. With a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects. In order to get the “big picture,” we are willing to sacrifice some of the small details.

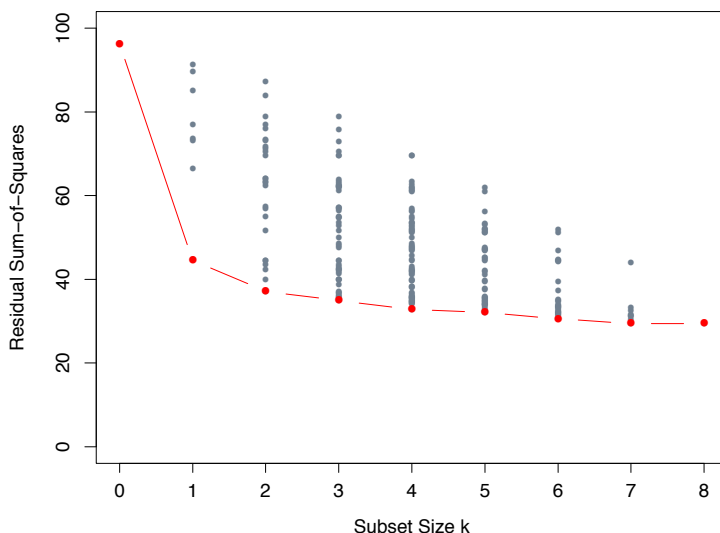
In this section we describe a number of approaches to variable subset selection with linear regression. In later sections we discuss shrinkage and hybrid approaches for controlling variance, as well as other dimension-reduction strategies. These all fall under the general heading *model selection*. Model selection is not restricted to linear models; Chapter 7 covers this topic in some detail.

With subset selection we retain only a subset of the variables, and eliminate the rest from the model. Least squares regression is used to estimate the coefficients of the inputs that are retained. There are a number of different strategies for choosing the subset.

### 3.3.1 Best-Subset Selection

Best subset regression finds for each  $k \in \{0, 1, 2, \dots, p\}$  the subset of size  $k$  that gives smallest residual sum of squares (3.2). An efficient algorithm—the *leaps and bounds* procedure (Furnival and Wilson, 1974)—makes this feasible for  $p$  as large as 30 or 40. Figure 3.5 shows all the subset models for the prostate cancer example. The lower boundary represents the models that are eligible for selection by the best-subsets approach. Note that the best subset of size 2, for example, need not include the variable that was in the best subset of size 1 (for this example all the subsets are nested). The best-subset curve (red lower boundary in Figure 3.5) is necessarily decreasing, so cannot be used to select the subset size  $k$ . The question of how to choose  $k$  involves the tradeoff between bias and variance, along with the more subjective desire for parsimony. There are a number of criteria that one may use; typically we choose the smallest model that minimizes an estimate of the expected prediction error.

Many of the other approaches that we discuss in this chapter are similar, in that they use the training data to produce a sequence of models varying in complexity and indexed by a single parameter. In the next section we use



**FIGURE 3.5.** All possible subset models for the prostate cancer example. At each subset size is shown the residual sum-of-squares for each model of that size.

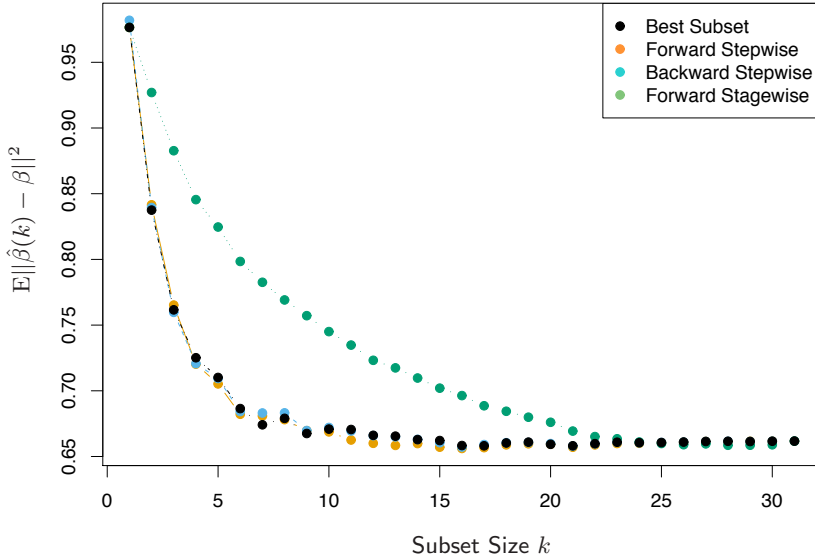
cross-validation to estimate prediction error and select  $k$ ; the AIC criterion is a popular alternative. We defer more detailed discussion of these and other approaches to Chapter 7.

### 3.3.2 Forward- and Backward-Stepwise Selection

Rather than search through all possible subsets (which becomes infeasible for  $p$  much larger than 40), we can seek a good path through them. *Forward-stepwise selection* starts with the intercept, and then sequentially adds into the model the predictor that most improves the fit. With many candidate predictors, this might seem like a lot of computation; however, clever updating algorithms can exploit the QR decomposition for the current fit to rapidly establish the next candidate (Exercise 3.9). Like best-subset regression, forward stepwise produces a sequence of models indexed by  $k$ , the subset size, which must be determined.

Forward-stepwise selection is a *greedy algorithm*, producing a nested sequence of models. In this sense it might seem sub-optimal compared to best-subset selection. However, there are several reasons why it might be preferred:

- *Computational*; for large  $p$  we cannot compute the best subset sequence, but we can always compute the forward stepwise sequence (even when  $p \gg N$ ).
- *Statistical*; a price is paid in variance for selecting the best subset of each size; forward stepwise is a more constrained search, and will have lower variance, but perhaps more bias.



**FIGURE 3.6.** Comparison of four subset-selection techniques on a simulated linear regression problem  $Y = X^T \beta + \varepsilon$ . There are  $N = 300$  observations on  $p = 31$  standard Gaussian variables, with pairwise correlations all equal to 0.85. For 10 of the variables, the coefficients are drawn at random from a  $N(0, 0.4)$  distribution; the rest are zero. The noise  $\varepsilon \sim N(0, 6.25)$ , resulting in a signal-to-noise ratio of 0.64. Results are averaged over 50 simulations. Shown is the mean-squared error of the estimated coefficient  $\hat{\beta}(k)$  at each step from the true  $\beta$ .

*Backward-stepwise selection* starts with the full model, and sequentially deletes the predictor that has the least impact on the fit. The candidate for dropping is the variable with the smallest Z-score (Exercise 3.10). Backward selection can only be used when  $N > p$ , while forward stepwise can always be used.

Figure 3.6 shows the results of a small simulation study to compare best-subset regression with the simpler alternatives forward and backward selection. Their performance is very similar, as is often the case. Included in the figure is forward stagewise regression (next section), which takes longer to reach minimum error.

On the prostate cancer example, best-subset, forward and backward selection all gave exactly the same sequence of terms.

Some software packages implement hybrid stepwise-selection strategies that consider both forward and backward moves at each step, and select the “best” of the two. For example in the **R** package the `step` function uses the AIC criterion for weighing the choices, which takes proper account of the number of parameters fit; at each step an add or drop will be performed that minimizes the AIC score.

Other more traditional packages base the selection on  $F$ -statistics, adding “significant” terms, and dropping “non-significant” terms. These are out of fashion, since they do not take proper account of the multiple testing issues. It is also tempting after a model search to print out a summary of the chosen model, such as in Table 3.2; however, the standard errors are not valid, since they do not account for the search process. The bootstrap (Section 8.2) can be useful in such settings.

Finally, we note that often variables come in groups (such as the dummy variables that code a multi-level categorical predictor). Smart stepwise procedures (such as `step` in **R**) will add or drop whole groups at a time, taking proper account of their degrees-of-freedom.

### 3.3.3 *Forward-Stagewise Regression*

Forward-stagewise regression (FS) is even more constrained than forward-stepwise regression. It starts like forward-stepwise regression, with an intercept equal to  $\bar{y}$ , and centered predictors with coefficients initially all 0. At each step the algorithm identifies the variable most correlated with the current residual. It then computes the simple linear regression coefficient of the residual on this chosen variable, and then adds it to the current coefficient for that variable. This is continued till none of the variables have correlation with the residuals—i.e. the least-squares fit when  $N > p$ .

Unlike forward-stepwise regression, none of the other variables are adjusted when a term is added to the model. As a consequence, forward stagewise can take many more than  $p$  steps to reach the least squares fit, and historically has been dismissed as being inefficient. It turns out that this “slow fitting” can pay dividends in high-dimensional problems. We see in Section 3.8.1 that both forward stagewise and a variant which is slowed down even further are quite competitive, especially in very high-dimensional problems.

Forward-stagewise regression is included in Figure 3.6. In this example it takes over 1000 steps to get all the correlations below  $10^{-4}$ . For subset size  $k$ , we plotted the error for the last step for which there were  $k$  nonzero coefficients. Although it catches up with the best fit, it takes longer to do so.

### 3.3.4 Prostate Cancer Data Example (Continued)

Table 3.3 shows the coefficients from a number of different selection and shrinkage methods. They are *best-subset selection* using an all-subsets search, *ridge regression*, the *lasso*, *principal components regression* and *partial least squares*. Each method has a complexity parameter, and this was chosen to minimize an estimate of prediction error based on tenfold cross-validation; full details are given in Section 7.10. Briefly, cross-validation works by dividing the training data randomly into ten equal parts. The learning method is fit—for a range of values of the complexity parameter—to nine-tenths of the data, and the prediction error is computed on the remaining one-tenth. This is done in turn for each one-tenth of the data, and the ten prediction error estimates are averaged. From this we obtain an estimated prediction error curve as a function of the complexity parameter.

Note that we have already divided these data into a training set of size 67 and a test set of size 30. Cross-validation is applied to the training set, since selecting the shrinkage parameter is part of the training process. The test set is there to judge the performance of the selected model.

The estimated prediction error curves are shown in Figure 3.7. Many of the curves are very flat over large ranges near their minimum. Included are estimated standard error bands for each estimated error rate, based on the ten error estimates computed by cross-validation. We have used the “one-standard-error” rule—we pick the most parsimonious model within one standard error of the minimum (Section 7.10, page 244). Such a rule acknowledges the fact that the tradeoff curve is estimated with error, and hence takes a conservative approach.

Best-subset selection chose to use the two predictors `lcvol` and `lweight`. The last two lines of the table give the average prediction error (and its estimated standard error) over the test set.

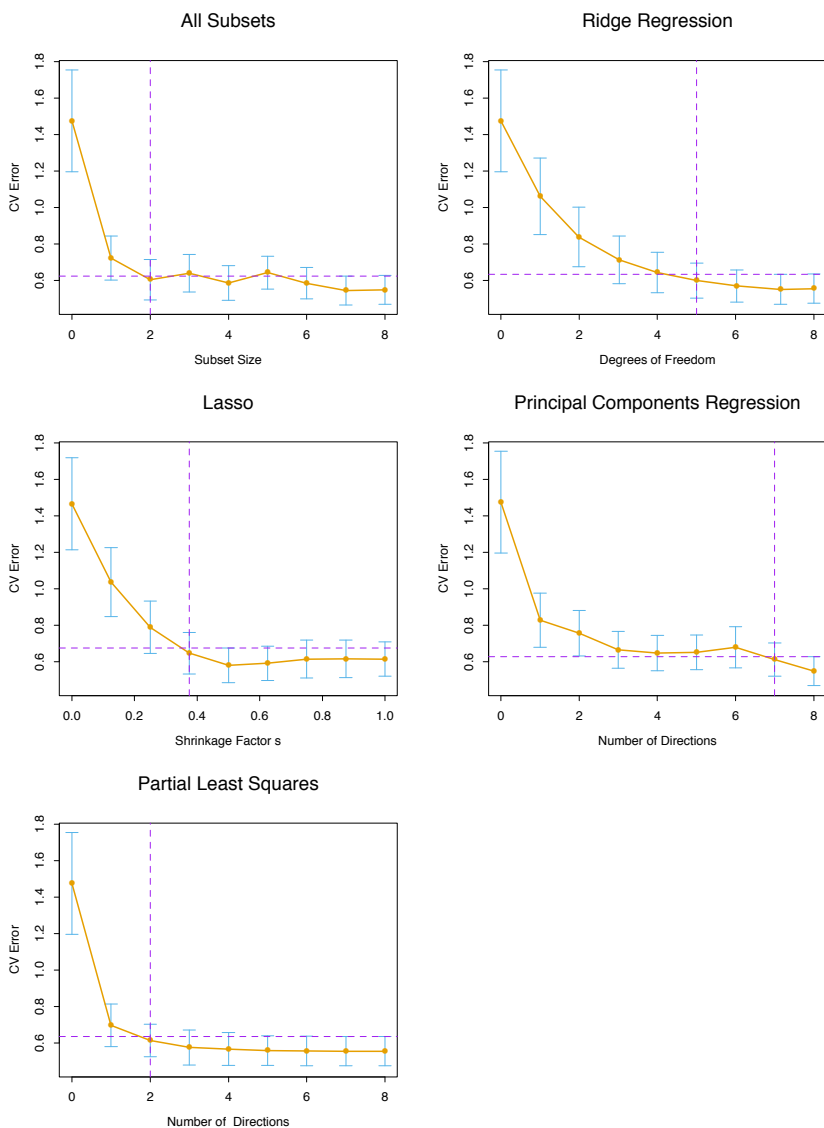
## 3.4 Shrinkage Methods

By retaining a subset of the predictors and discarding the rest, subset selection produces a model that is interpretable and has possibly lower prediction error than the full model. However, because it is a discrete process—variables are either retained or discarded—it often exhibits high variance, and so doesn’t reduce the prediction error of the full model. Shrinkage methods are more continuous, and don’t suffer as much from high variability.

### 3.4.1 Ridge Regression

Ridge regression shrinks the regression coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized residual sum of





**FIGURE 3.7.** Estimated prediction error curves and their standard errors for the various selection and shrinkage methods. Each curve is plotted as a function of the corresponding complexity parameter for that method. The horizontal axis has been chosen so that the model complexity increases as we move from left to right. The estimates of prediction error and their standard errors were obtained by tenfold cross-validation; full details are given in Section 7.10. The least complex model within one standard error of the best is chosen, indicated by the purple vertical broken lines.

**TABLE 3.3.** *Estimated coefficients and test error results, for different subset and shrinkage methods applied to the prostate data. The blank entries correspond to variables omitted.*

Term	LS	Best Subset	Ridge	Lasso	PCR	PLS
Intercept	2.465	2.477	2.452	2.468	2.497	2.452
lcavol	0.680	0.740	0.420	0.533	0.543	0.419
lweight	0.263	0.316	0.238	0.169	0.289	0.344
age	-0.141		-0.046		-0.152	-0.026
lbph	0.210		0.162	0.002	0.214	0.220
svi	0.305		0.227	0.094	0.315	0.243
lcp	-0.288		0.000		-0.051	0.079
gleason	-0.021		0.040		0.232	0.011
pgg45	0.267		0.133		-0.056	0.084
Test Error	0.521	0.492	0.492	0.479	0.449	0.528
Std Error	0.179	0.143	0.165	0.164	0.105	0.152

squares,

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right\}. \quad (3.41)$$

Here  $\lambda \geq 0$  is a complexity parameter that controls the amount of shrinkage: the larger the value of  $\lambda$ , the greater the amount of shrinkage. The coefficients are shrunk toward zero (and each other). The idea of penalizing by the sum-of-squares of the parameters is also used in neural networks, where it is known as *weight decay* (Chapter 11).

An equivalent way to write the ridge problem is

$$\begin{aligned} \hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^N \left( y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2, \\ \text{subject to } \sum_{j=1}^p \beta_j^2 \leq t, \end{aligned} \quad (3.42)$$

which makes explicit the size constraint on the parameters. There is a one-to-one correspondence between the parameters  $\lambda$  in (3.41) and  $t$  in (3.42). When there are many correlated variables in a linear regression model, their coefficients can become poorly determined and exhibit high variance. A wildly large positive coefficient on one variable can be canceled by a similarly large negative coefficient on its correlated cousin. By imposing a size constraint on the coefficients, as in (3.42), this problem is alleviated.

The ridge solutions are not equivariant under scaling of the inputs, and so one normally standardizes the inputs before solving (3.41). In addition,

notice that the intercept  $\beta_0$  has been left out of the penalty term. Penalization of the intercept would make the procedure depend on the origin chosen for  $Y$ ; that is, adding a constant  $c$  to each of the targets  $y_i$  would not simply result in a shift of the predictions by the same amount  $c$ . It can be shown (Exercise 3.5) that the solution to (3.41) can be separated into two parts, after reparametrization using *centered* inputs: each  $x_{ij}$  gets replaced by  $x_{ij} - \bar{x}_j$ . We estimate  $\beta_0$  by  $\bar{y} = \frac{1}{N} \sum_1^N y_i$ . The remaining coefficients get estimated by a ridge regression without intercept, using the centered  $x_{ij}$ . Henceforth we assume that this centering has been done, so that the input matrix  $\mathbf{X}$  has  $p$  (rather than  $p + 1$ ) columns.

Writing the criterion in (3.41) in matrix form,

$$\text{RSS}(\lambda) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^T\beta, \quad (3.43)$$

the ridge regression solutions are easily seen to be

$$\hat{\beta}^{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \quad (3.44)$$

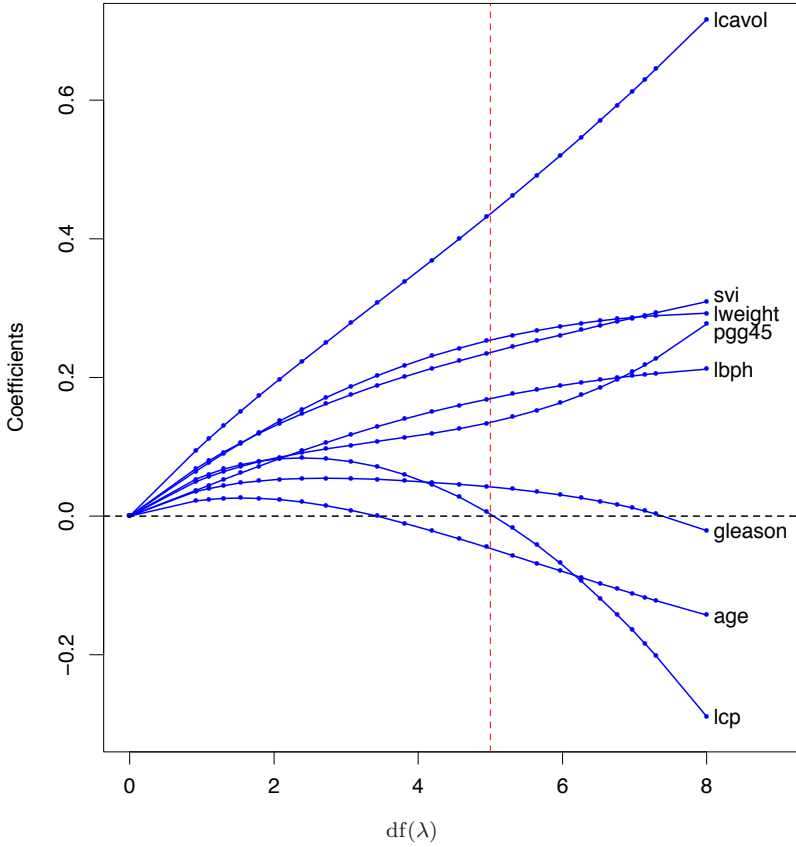
where  $\mathbf{I}$  is the  $p \times p$  identity matrix. Notice that with the choice of quadratic penalty  $\beta^T\beta$ , the ridge regression solution is again a linear function of  $\mathbf{y}$ . The solution adds a positive constant to the diagonal of  $\mathbf{X}^T\mathbf{X}$  before inversion. This makes the problem nonsingular, even if  $\mathbf{X}^T\mathbf{X}$  is not of full rank, and was the main motivation for ridge regression when it was first introduced in statistics (Hoerl and Kennard, 1970). Traditional descriptions of ridge regression start with definition (3.44). We choose to motivate it via (3.41) and (3.42), as these provide insight into how it works.

Figure 3.8 shows the ridge coefficient estimates for the prostate cancer example, plotted as functions of  $\text{df}(\lambda)$ , the *effective degrees of freedom* implied by the penalty  $\lambda$  (defined in (3.50) on page 68). In the case of orthonormal inputs, the ridge estimates are just a scaled version of the least squares estimates, that is,  $\hat{\beta}^{\text{ridge}} = \hat{\beta}/(1 + \lambda)$ .

Ridge regression can also be derived as the mean or mode of a posterior distribution, with a suitably chosen prior distribution. In detail, suppose  $y_i \sim N(\beta_0 + x_i^T\beta, \sigma^2)$ , and the parameters  $\beta_j$  are each distributed as  $N(0, \tau^2)$ , independently of one another. Then the (negative) log-posterior density of  $\beta$ , with  $\tau^2$  and  $\sigma^2$  assumed known, is equal to the expression in curly braces in (3.41), with  $\lambda = \sigma^2/\tau^2$  (Exercise 3.6). Thus the ridge estimate is the mode of the posterior distribution; since the distribution is Gaussian, it is also the posterior mean.

The *singular value decomposition* (SVD) of the centered input matrix  $\mathbf{X}$  gives us some additional insight into the nature of ridge regression. This decomposition is extremely useful in the analysis of many statistical methods. The SVD of the  $N \times p$  matrix  $\mathbf{X}$  has the form

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T. \quad (3.45)$$



**FIGURE 3.8.** Profiles of ridge coefficients for the prostate cancer example, as the tuning parameter  $\lambda$  is varied. Coefficients are plotted versus  $df(\lambda)$ , the effective degrees of freedom. A vertical line is drawn at  $df = 5.0$ , the value chosen by cross-validation.

Here  $\mathbf{U}$  and  $\mathbf{V}$  are  $N \times p$  and  $p \times p$  orthogonal matrices, with the columns of  $\mathbf{U}$  spanning the column space of  $\mathbf{X}$ , and the columns of  $\mathbf{V}$  spanning the row space.  $\mathbf{D}$  is a  $p \times p$  diagonal matrix, with diagonal entries  $d_1 \geq d_2 \geq \dots \geq d_p \geq 0$  called the singular values of  $\mathbf{X}$ . If one or more values  $d_j = 0$ ,  $\mathbf{X}$  is singular.

Using the singular value decomposition we can write the least squares fitted vector as

$$\begin{aligned}\mathbf{X}\hat{\beta}^{\text{ls}} &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{U}\mathbf{U}^T\mathbf{y},\end{aligned}\tag{3.46}$$

after some simplification. Note that  $\mathbf{U}^T\mathbf{y}$  are the coordinates of  $\mathbf{y}$  with respect to the orthonormal basis  $\mathbf{U}$ . Note also the similarity with (3.33);  $\mathbf{Q}$  and  $\mathbf{U}$  are generally different orthogonal bases for the column space of  $\mathbf{X}$  (Exercise 3.8).

Now the ridge solutions are

$$\begin{aligned}\mathbf{X}\hat{\beta}^{\text{ridge}} &= \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{U}\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{y} \\ &= \sum_{j=1}^p \mathbf{u}_j \frac{d_j^2}{d_j^2 + \lambda} \mathbf{u}_j^T \mathbf{y},\end{aligned}\tag{3.47}$$

where the  $\mathbf{u}_j$  are the columns of  $\mathbf{U}$ . Note that since  $\lambda \geq 0$ , we have  $d_j^2/(d_j^2 + \lambda) \leq 1$ . Like linear regression, ridge regression computes the coordinates of  $\mathbf{y}$  with respect to the orthonormal basis  $\mathbf{U}$ . It then shrinks these coordinates by the factors  $d_j^2/(d_j^2 + \lambda)$ . This means that a greater amount of shrinkage is applied to the coordinates of basis vectors with smaller  $d_j^2$ .

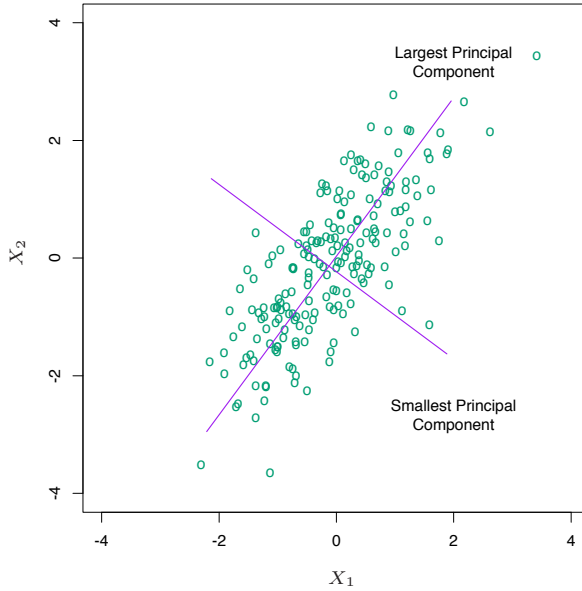
What does a small value of  $d_j^2$  mean? The SVD of the centered matrix  $\mathbf{X}$  is another way of expressing the *principal components* of the variables in  $\mathbf{X}$ . The sample covariance matrix is given by  $\mathbf{S} = \mathbf{X}^T\mathbf{X}/N$ , and from (3.45) we have

$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{D}^2\mathbf{V}^T,\tag{3.48}$$

which is the *eigen decomposition* of  $\mathbf{X}^T\mathbf{X}$  (and of  $\mathbf{S}$ , up to a factor  $N$ ). The eigenvectors  $v_j$  (columns of  $\mathbf{V}$ ) are also called the *principal components* (or Karhunen–Loeve) directions of  $\mathbf{X}$ . The first principal component direction  $v_1$  has the property that  $\mathbf{z}_1 = \mathbf{X}v_1$  has the largest sample variance amongst all normalized linear combinations of the columns of  $\mathbf{X}$ . This sample variance is easily seen to be

$$\text{Var}(\mathbf{z}_1) = \text{Var}(\mathbf{X}v_1) = \frac{d_1^2}{N},\tag{3.49}$$

and in fact  $\mathbf{z}_1 = \mathbf{X}v_1 = \mathbf{u}_1d_1$ . The derived variable  $\mathbf{z}_1$  is called the first principal component of  $\mathbf{X}$ , and hence  $\mathbf{u}_1$  is the normalized first principal



**FIGURE 3.9.** *Principal components of some input data points. The largest principal component is the direction that maximizes the variance of the projected data, and the smallest principal component minimizes that variance. Ridge regression projects  $\mathbf{y}$  onto these components, and then shrinks the coefficients of the low-variance components more than the high-variance components.*

component. Subsequent principal components  $\mathbf{z}_j$  have maximum variance  $d_j^2/N$ , subject to being orthogonal to the earlier ones. Conversely the last principal component has *minimum* variance. Hence the small singular values  $d_j$  correspond to directions in the column space of  $\mathbf{X}$  having small variance, and ridge regression shrinks these directions the most.

Figure 3.9 illustrates the principal components of some data points in two dimensions. If we consider fitting a linear surface over this domain (the  $Y$ -axis is sticking out of the page), the configuration of the data allow us to determine its gradient more accurately in the long direction than the short. Ridge regression protects against the potentially high variance of gradients estimated in the short directions. The implicit assumption is that the response will tend to vary most in the directions of high variance of the inputs. This is often a reasonable assumption, since predictors are often chosen for study because they vary with the response variable, but need not hold in general.

In Figure 3.7 we have plotted the estimated prediction error versus the quantity

$$\begin{aligned} \text{df}(\lambda) &= \text{tr}[\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T], \\ &= \text{tr}(\mathbf{H}_\lambda) \\ &= \sum_{j=1}^p \frac{d_j^2}{d_j^2 + \lambda}. \end{aligned} \quad (3.50)$$

This monotone decreasing function of  $\lambda$  is the *effective degrees of freedom* of the ridge regression fit. Usually in a linear-regression fit with  $p$  variables, the degrees-of-freedom of the fit is  $p$ , the number of free parameters. The idea is that although all  $p$  coefficients in a ridge fit will be non-zero, they are fit in a restricted fashion controlled by  $\lambda$ . Note that  $\text{df}(\lambda) = p$  when  $\lambda = 0$  (no regularization) and  $\text{df}(\lambda) \rightarrow 0$  as  $\lambda \rightarrow \infty$ . Of course there is always an additional one degree of freedom for the intercept, which was removed *a priori*. This definition is motivated in more detail in Section 3.4.4 and Sections 7.4–7.6. In Figure 3.7 the minimum occurs at  $\text{df}(\lambda) = 5.0$ . Table 3.3 shows that ridge regression reduces the test error of the full least squares estimates by a small amount.

### 3.4.2 The Lasso

The lasso is a shrinkage method like ridge, with subtle but important differences. The lasso estimate is defined by

$$\begin{aligned} \hat{\beta}^{\text{lasso}} &= \underset{\beta}{\text{argmin}} \sum_{i=1}^N \left( y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j \right)^2 \\ &\text{subject to } \sum_{j=1}^p |\beta_j| \leq t. \end{aligned} \quad (3.51)$$

Just as in ridge regression, we can re-parametrize the constant  $\beta_0$  by standardizing the predictors; the solution for  $\hat{\beta}_0$  is  $\bar{y}$ , and thereafter we fit a model without an intercept (Exercise 3.5). In the signal processing literature, the lasso is also known as *basis pursuit* (Chen et al., 1998).

We can also write the lasso problem in the equivalent *Lagrangian form*

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\text{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^N \left( y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^p |\beta_j| \right\}. \quad (3.52)$$

Notice the similarity to the ridge regression problem (3.42) or (3.41): the  $L_2$  ridge penalty  $\sum_1^p \beta_j^2$  is replaced by the  $L_1$  lasso penalty  $\sum_1^p |\beta_j|$ . This latter constraint makes the solutions nonlinear in the  $y_i$ , and there is no closed form expression as in ridge regression. Computing the lasso solution

is a quadratic programming problem, although we see in Section 3.4.4 that efficient algorithms are available for computing the entire path of solutions as  $\lambda$  is varied, with the same computational cost as for ridge regression. Because of the nature of the constraint, making  $t$  sufficiently small will cause some of the coefficients to be exactly zero. Thus the lasso does a kind of continuous subset selection. If  $t$  is chosen larger than  $t_0 = \sum_1^p |\hat{\beta}_j|$  (where  $\hat{\beta}_j = \hat{\beta}_j^{\text{ls}}$ , the least squares estimates), then the lasso estimates are the  $\hat{\beta}_j$ 's. On the other hand, for  $t = t_0/2$  say, then the least squares coefficients are shrunk by about 50% on average. However, the nature of the shrinkage is not obvious, and we investigate it further in Section 3.4.4 below. Like the subset size in variable subset selection, or the penalty parameter in ridge regression,  $t$  should be adaptively chosen to minimize an estimate of expected prediction error.

In Figure 3.7, for ease of interpretation, we have plotted the lasso prediction error estimates versus the standardized parameter  $s = t / \sum_1^p |\hat{\beta}_j|$ . A value  $\hat{s} \approx 0.36$  was chosen by 10-fold cross-validation; this caused four coefficients to be set to zero (fifth column of Table 3.3). The resulting model has the second lowest test error, slightly lower than the full least squares model, but the standard errors of the test error estimates (last line of Table 3.3) are fairly large.

Figure 3.10 shows the lasso coefficients as the standardized tuning parameter  $s = t / \sum_1^p |\hat{\beta}_j|$  is varied. At  $s = 1.0$  these are the least squares estimates; they decrease to 0 as  $s \rightarrow 0$ . This decrease is not always strictly monotonic, although it is in this example. A vertical line is drawn at  $s = 0.36$ , the value chosen by cross-validation.

### 3.4.3 Discussion: Subset Selection, Ridge Regression and the Lasso

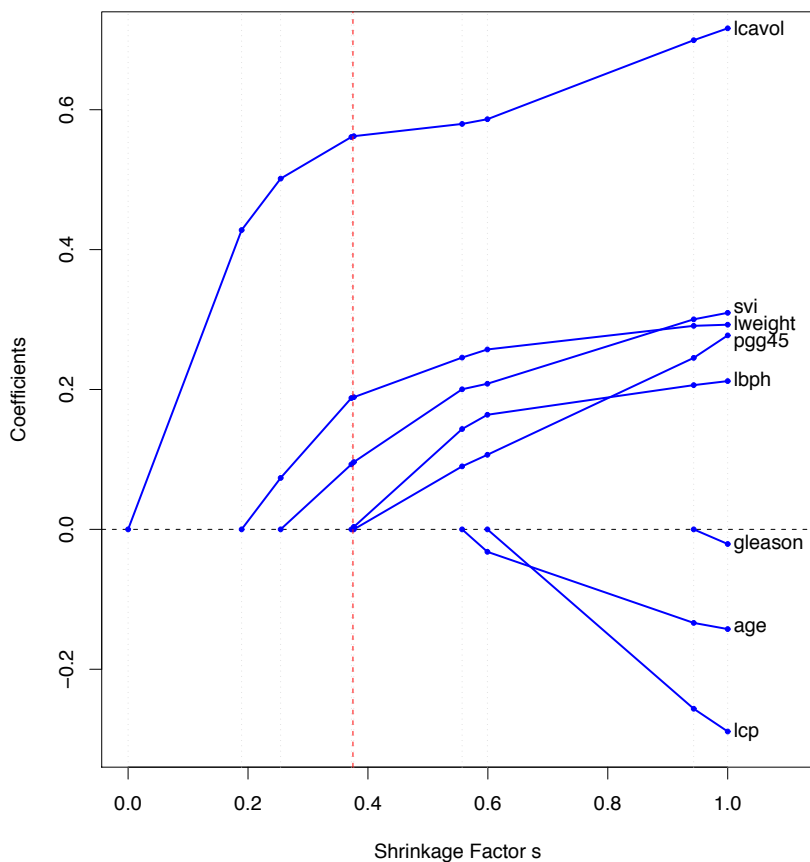
In this section we discuss and compare the three approaches discussed so far for restricting the linear regression model: subset selection, ridge regression and the lasso.

In the case of an orthonormal input matrix  $\mathbf{X}$  the three procedures have explicit solutions. Each method applies a simple transformation to the least squares estimate  $\hat{\beta}_j$ , as detailed in Table 3.4.

Ridge regression does a proportional shrinkage. Lasso translates each coefficient by a constant factor  $\lambda$ , truncating at zero. This is called “soft thresholding,” and is used in the context of wavelet-based smoothing in Section 5.9. Best-subset selection drops all variables with coefficients smaller than the  $M$ th largest; this is a form of “hard-thresholding.”

Back to the nonorthogonal case; some pictures help understand their relationship. Figure 3.11 depicts the lasso (left) and ridge regression (right) when there are only two parameters. The residual sum of squares has elliptical contours, centered at the full least squares estimate. The constraint

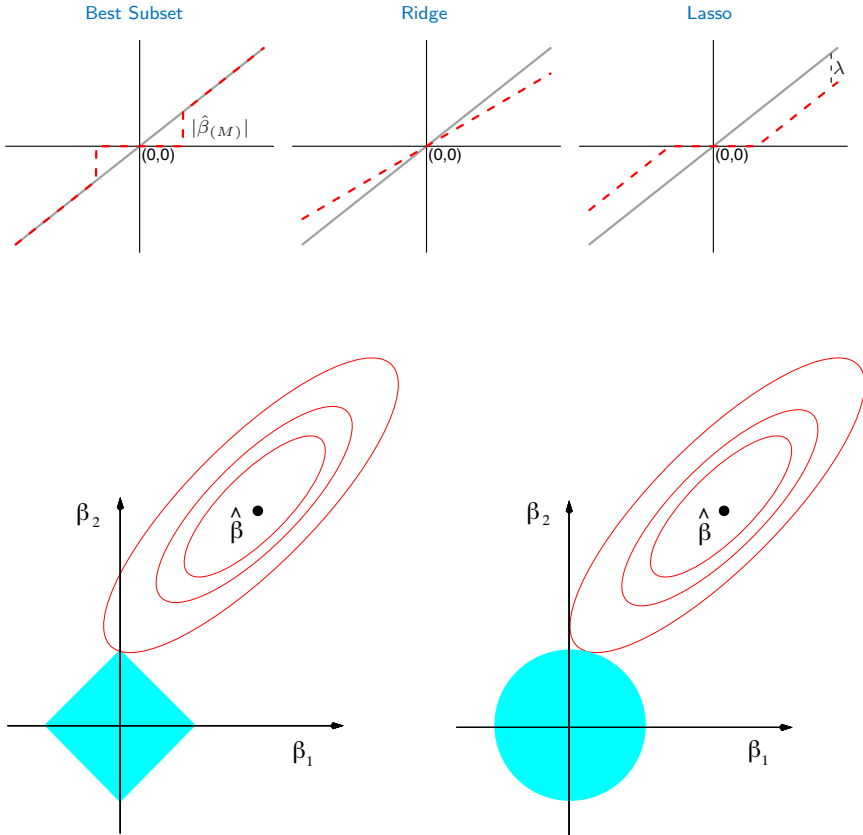




**FIGURE 3.10.** Profiles of lasso coefficients, as the tuning parameter  $t$  is varied. Coefficients are plotted versus  $s = t / \sum_1^p |\hat{\beta}_j|$ . A vertical line is drawn at  $s = 0.36$ , the value chosen by cross-validation. Compare Figure 3.8 on page 65; the lasso profiles hit zero, while those for ridge do not. The profiles are piece-wise linear, and so are computed only at the points displayed; see Section 3.4.4 for details.

**TABLE 3.4.** Estimators of  $\beta_j$  in the case of orthonormal columns of  $\mathbf{X}$ .  $M$  and  $\lambda$  are constants chosen by the corresponding techniques;  $\text{sign}$  denotes the sign of its argument ( $\pm 1$ ), and  $x_+$  denotes “positive part” of  $x$ . Below the table, estimators are shown by broken red lines. The  $45^\circ$  line in gray shows the unrestricted estimate for reference.

Estimator	Formula
Best subset (size $M$ )	$\hat{\beta}_j \cdot I( \hat{\beta}_j  \geq  \hat{\beta}_{(M)} )$
Ridge	$\hat{\beta}_j / (1 + \lambda)$
Lasso	$\text{sign}(\hat{\beta}_j)( \hat{\beta}_j  - \lambda)_+$



**FIGURE 3.11.** Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions  $|\beta_1| + |\beta_2| \leq t$  and  $\beta_1^2 + \beta_2^2 \leq t^2$ , respectively, while the red ellipses are the contours of the least squares error function.

region for ridge regression is the disk  $\beta_1^2 + \beta_2^2 \leq t$ , while that for lasso is the diamond  $|\beta_1| + |\beta_2| \leq t$ . Both methods find the first point where the elliptical contours hit the constraint region. Unlike the disk, the diamond has corners; if the solution occurs at a corner, then it has one parameter  $\beta_j$  equal to zero. When  $p > 2$ , the diamond becomes a rhomboid, and has many corners, flat edges and faces; there are many more opportunities for the estimated parameters to be zero.

We can generalize ridge regression and the lasso, and view them as Bayes estimates. Consider the criterion

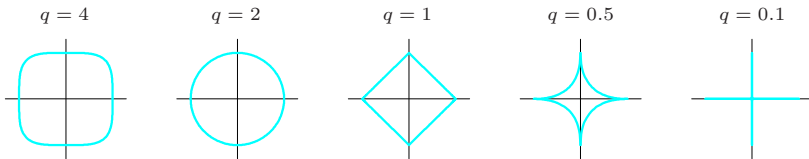
$$\tilde{\beta} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j|^q \right\} \quad (3.53)$$

for  $q \geq 0$ . The contours of constant value of  $\sum_j |\beta_j|^q$  are shown in Figure 3.12, for the case of two inputs.

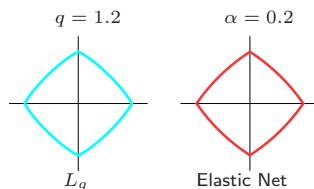
Thinking of  $|\beta_j|^q$  as the log-prior density for  $\beta_j$ , these are also the equi-contours of the prior distribution of the parameters. The value  $q = 0$  corresponds to variable subset selection, as the penalty simply counts the number of nonzero parameters;  $q = 1$  corresponds to the lasso, while  $q = 2$  to ridge regression. Notice that for  $q \leq 1$ , the prior is not uniform in direction, but concentrates more mass in the coordinate directions. The prior corresponding to the  $q = 1$  case is an independent double exponential (or Laplace) distribution for each input, with density  $(1/2\tau) \exp(-|\beta|/\tau)$  and  $\tau = 1/\lambda$ . The case  $q = 1$  (lasso) is the smallest  $q$  such that the constraint region is convex; non-convex constraint regions make the optimization problem more difficult.

In this view, the lasso, ridge regression and best subset selection are Bayes estimates with different priors. Note, however, that they are derived as posterior modes, that is, maximizers of the posterior. It is more common to use the mean of the posterior as the Bayes estimate. Ridge regression is also the posterior mean, but the lasso and best subset selection are not.

Looking again at the criterion (3.53), we might try using other values of  $q$  besides 0, 1, or 2. Although one might consider estimating  $q$  from the data, our experience is that it is not worth the effort for the extra variance incurred. Values of  $q \in (1, 2)$  suggest a compromise between the lasso and ridge regression. Although this is the case, with  $q > 1$ ,  $|\beta_j|^q$  is differentiable at 0, and so does not share the ability of lasso ( $q = 1$ ) for



**FIGURE 3.12.** Contours of constant value of  $\sum_j |\beta_j|^q$  for given values of  $q$ .



**FIGURE 3.13.** Contours of constant value of  $\sum_j |\beta_j|^q$  for  $q = 1.2$  (left plot), and the elastic-net penalty  $\sum_j (\alpha\beta_j^2 + (1-\alpha)|\beta_j|)$  for  $\alpha = 0.2$  (right plot). Although visually very similar, the elastic-net has sharp (non-differentiable) corners, while the  $q = 1.2$  penalty does not.

setting coefficients exactly to zero. Partly for this reason as well as for computational tractability, Zou and Hastie (2005) introduced the *elastic-net* penalty

$$\lambda \sum_{j=1}^p (\alpha\beta_j^2 + (1-\alpha)|\beta_j|), \quad (3.54)$$

a different compromise between ridge and lasso. Figure 3.13 compares the  $L_q$  penalty with  $q = 1.2$  and the elastic-net penalty with  $\alpha = 0.2$ ; it is hard to detect the difference by eye. The elastic-net selects variables like the lasso, and shrinks together the coefficients of correlated predictors like ridge. It also has considerable computational advantages over the  $L_q$  penalties. We discuss the elastic-net further in Section 18.4.

### 3.4.4 Least Angle Regression

Least angle regression (LAR) is a relative newcomer (Efron et al., 2004), and can be viewed as a kind of “democratic” version of forward stepwise regression (Section 3.3.2). As we will see, LAR is intimately connected with the lasso, and in fact provides an extremely efficient algorithm for computing the entire lasso path as in Figure 3.10.

Forward stepwise regression builds a model sequentially, adding one variable at a time. At each step, it identifies the best variable to include in the *active set*, and then updates the least squares fit to include all the active variables.

Least angle regression uses a similar strategy, but only enters “as much” of a predictor as it deserves. At the first step it identifies the variable most correlated with the response. Rather than fit this variable completely, LAR moves the coefficient of this variable continuously toward its least-squares value (causing its correlation with the evolving residual to decrease in absolute value). As soon as another variable “catches up” in terms of correlation with the residual, the process is paused. The second variable then joins the active set, and their coefficients are moved together in a way that keeps their correlations tied and decreasing. This process is continued

until all the variables are in the model, and ends at the full least-squares fit. Algorithm 3.2 provides the details. The termination condition in step 5 requires some explanation. If  $p > N - 1$ , the LAR algorithm reaches a zero residual solution after  $N - 1$  steps (the  $-1$  is because we have centered the data).

---

**Algorithm 3.2** *Least Angle Regression.*

---

1. Standardize the predictors to have mean zero and unit norm. Start with the residual  $\mathbf{r} = \mathbf{y} - \bar{\mathbf{y}}$ ,  $\beta_1, \beta_2, \dots, \beta_p = 0$ .
  2. Find the predictor  $\mathbf{x}_j$  most correlated with  $\mathbf{r}$ .
  3. Move  $\beta_j$  from 0 towards its least-squares coefficient  $\langle \mathbf{x}_j, \mathbf{r} \rangle$ , until some other competitor  $\mathbf{x}_k$  has as much correlation with the current residual as does  $\mathbf{x}_j$ .
  4. Move  $\beta_j$  and  $\beta_k$  in the direction defined by their joint least squares coefficient of the current residual on  $(\mathbf{x}_j, \mathbf{x}_k)$ , until some other competitor  $\mathbf{x}_l$  has as much correlation with the current residual.
  5. Continue in this way until all  $p$  predictors have been entered. After  $\min(N - 1, p)$  steps, we arrive at the full least-squares solution.
- 

Suppose  $\mathcal{A}_k$  is the active set of variables at the beginning of the  $k$ th step, and let  $\beta_{\mathcal{A}_k}$  be the coefficient vector for these variables at this step; there will be  $k - 1$  nonzero values, and the one just entered will be zero. If  $\mathbf{r}_k = \mathbf{y} - \mathbf{X}_{\mathcal{A}_k} \beta_{\mathcal{A}_k}$  is the current residual, then the direction for this step is

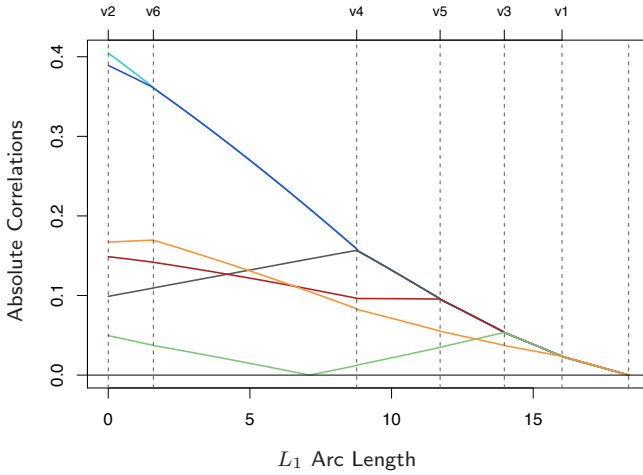
$$\delta_k = (\mathbf{X}_{\mathcal{A}_k}^T \mathbf{X}_{\mathcal{A}_k})^{-1} \mathbf{X}_{\mathcal{A}_k}^T \mathbf{r}_k. \quad (3.55)$$

The coefficient profile then evolves as  $\beta_{\mathcal{A}_k}(\alpha) = \beta_{\mathcal{A}_k} + \alpha \cdot \delta_k$ . Exercise 3.23 verifies that the directions chosen in this fashion do what is claimed: keep the correlations tied and decreasing. If the fit vector at the beginning of this step is  $\hat{\mathbf{f}}_k$ , then it evolves as  $\hat{\mathbf{f}}_k(\alpha) = \hat{\mathbf{f}}_k + \alpha \cdot \mathbf{u}_k$ , where  $\mathbf{u}_k = \mathbf{X}_{\mathcal{A}_k} \delta_k$  is the new fit direction. The name “least angle” arises from a geometrical interpretation of this process;  $\mathbf{u}_k$  makes the smallest (and equal) angle with each of the predictors in  $\mathcal{A}_k$  (Exercise 3.24). Figure 3.14 shows the absolute correlations decreasing and joining ranks with each step of the LAR algorithm, using simulated data.

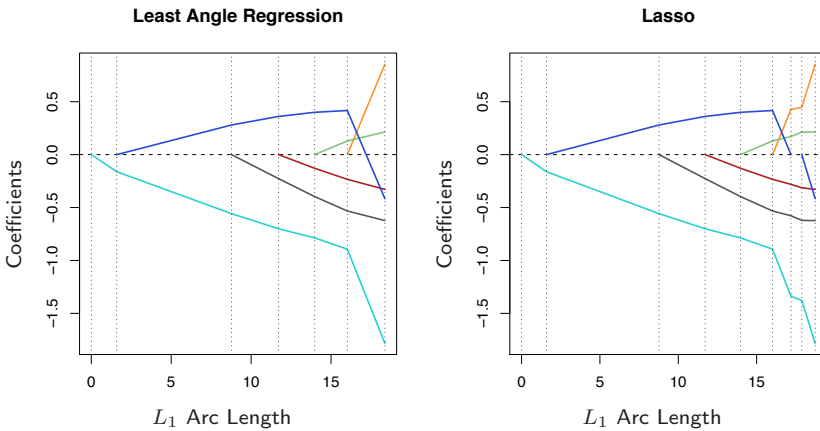
By construction the coefficients in LAR change in a piecewise linear fashion. Figure 3.15 [left panel] shows the LAR coefficient profile evolving as a function of their  $L_1$  arc length<sup>2</sup>. Note that we do not need to take small

---

<sup>2</sup>The  $L_1$  arc-length of a differentiable curve  $\beta(s)$  for  $s \in [0, S]$  is given by  $\text{TV}(\beta, S) = \int_0^S \|\dot{\beta}(s)\|_1 ds$ , where  $\dot{\beta}(s) = \partial\beta(s)/\partial s$ . For the piecewise-linear LAR coefficient profile, this amounts to summing the  $L_1$  norms of the changes in coefficients from step to step.



**FIGURE 3.14.** Progression of the absolute correlations during each step of the LAR procedure, using a simulated data set with six predictors. The labels at the top of the plot indicate which variables enter the active set at each step. The step length are measured in units of  $L_1$  arc length.



**FIGURE 3.15.** Left panel shows the LAR coefficient profiles on the simulated data, as a function of the  $L_1$  arc length. The right panel shows the Lasso profile. They are identical until the dark-blue coefficient crosses zero at an arc length of about 18.

steps and recheck the correlations in step 3; using knowledge of the covariance of the predictors and the piecewise linearity of the algorithm, we can work out the exact step length at the beginning of each step (Exercise 3.25).

The right panel of Figure 3.15 shows the lasso coefficient profiles on the same data. They are almost identical to those in the left panel, and differ for the first time when the blue coefficient passes back through zero. For the prostate data, the LAR coefficient profile turns out to be identical to the lasso profile in Figure 3.10, which never crosses zero. These observations lead to a simple modification of the LAR algorithm that gives the entire lasso path, which is also piecewise-linear.

---

**Algorithm 3.2a** *Least Angle Regression: Lasso Modification.*

---

- 4a. If a non-zero coefficient hits zero, drop its variable from the active set of variables and recompute the current joint least squares direction.
- 

The LAR(lasso) algorithm is extremely efficient, requiring the same order of computation as that of a single least squares fit using the  $p$  predictors. Least angle regression always takes  $p$  steps to get to the full least squares estimates. The lasso path can have more than  $p$  steps, although the two are often quite similar. Algorithm 3.2 with the lasso modification 3.2a is an efficient way of computing the solution to any lasso problem, especially when  $p \gg N$ . Osborne et al. (2000a) also discovered a piecewise-linear path for computing the lasso, which they called a *homotopy* algorithm.

We now give a heuristic argument for why these procedures are so similar. Although the LAR algorithm is stated in terms of correlations, if the input features are standardized, it is equivalent and easier to work with inner-products. Suppose  $\mathcal{A}$  is the active set of variables at some stage in the algorithm, tied in their absolute inner-product with the current residuals  $\mathbf{y} - \mathbf{X}\beta$ . We can express this as

$$\mathbf{x}_j^T (\mathbf{y} - \mathbf{X}\beta) = \gamma \cdot s_j, \quad \forall j \in \mathcal{A} \quad (3.56)$$

where  $s_j \in \{-1, 1\}$  indicates the sign of the inner-product, and  $\gamma$  is the common value. Also  $|\mathbf{x}_k^T (\mathbf{y} - \mathbf{X}\beta)| \leq \gamma \quad \forall k \notin \mathcal{A}$ . Now consider the lasso criterion (3.52), which we write in vector form

$$R(\beta) = \frac{1}{2} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \|\beta\|_1. \quad (3.57)$$

Let  $\mathcal{B}$  be the active set of variables in the solution for a given value of  $\lambda$ . For these variables  $R(\beta)$  is differentiable, and the stationarity conditions give

$$\mathbf{x}_j^T (\mathbf{y} - \mathbf{X}\beta) = \lambda \cdot \text{sign}(\beta_j), \quad \forall j \in \mathcal{B} \quad (3.58)$$

Comparing (3.58) with (3.56), we see that they are identical only if the sign of  $\beta_j$  matches the sign of the inner product. That is why the LAR

algorithm and lasso start to differ when an active coefficient passes through zero; condition (3.58) is violated for that variable, and it is kicked out of the active set  $\mathcal{B}$ . Exercise 3.23 shows that these equations imply a piecewise-linear coefficient profile as  $\lambda$  decreases. The stationarity conditions for the non-active variables require that

$$|\mathbf{x}_k^T(\mathbf{y} - \mathbf{X}\beta)| \leq \lambda, \quad \forall k \notin \mathcal{B}, \quad (3.59)$$

which again agrees with the LAR algorithm.

Figure 3.16 compares LAR and lasso to forward stepwise and stagewise regression. The setup is the same as in Figure 3.6 on page 59, except here  $N = 100$  here rather than 300, so the problem is more difficult. We see that the more aggressive forward stepwise starts to overfit quite early (well before the 10 true variables can enter the model), and ultimately performs worse than the slower forward stagewise regression. The behavior of LAR and lasso is similar to that of forward stagewise regression. Incremental forward stagewise is similar to LAR and lasso, and is described in Section 3.8.1.

### Degrees-of-Freedom Formula for LAR and Lasso

Suppose that we fit a linear model via the least angle regression procedure, stopping at some number of steps  $k < p$ , or equivalently using a lasso bound  $t$  that produces a constrained version of the full least squares fit. How many parameters, or “degrees of freedom” have we used?

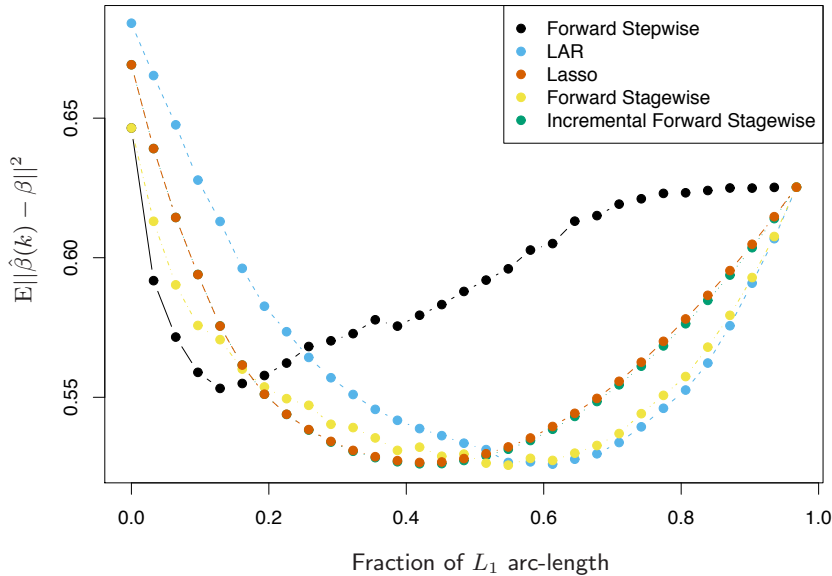
Consider first a linear regression using a subset of  $k$  features. If this subset is prespecified in advance without reference to the training data, then the degrees of freedom used in the fitted model is defined to be  $k$ . Indeed, in classical statistics, the number of linearly independent parameters is what is meant by “degrees of freedom.” Alternatively, suppose that we carry out a best subset selection to determine the “optimal” set of  $k$  predictors. Then the resulting model has  $k$  parameters, but in some sense we have used up more than  $k$  degrees of freedom.

We need a more general definition for the effective degrees of freedom of an adaptively fitted model. We define the degrees of freedom of the fitted vector  $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_N)$  as

$$\text{df}(\hat{\mathbf{y}}) = \frac{1}{\sigma^2} \sum_{i=1}^N \text{Cov}(\hat{y}_i, y_i). \quad (3.60)$$

Here  $\text{Cov}(\hat{y}_i, y_i)$  refers to the sampling covariance between the predicted value  $\hat{y}_i$  and its corresponding outcome value  $y_i$ . This makes intuitive sense: the harder that we fit to the data, the larger this covariance and hence  $\text{df}(\hat{\mathbf{y}})$ . Expression (3.60) is a useful notion of degrees of freedom, one that can be applied to any model prediction  $\hat{\mathbf{y}}$ . This includes models that are





**FIGURE 3.16.** Comparison of LAR and lasso with forward stepwise, forward stagewise (FS) and incremental forward stagewise ( $FS_0$ ) regression. The setup is the same as in Figure 3.6, except  $N = 100$  here rather than 300. Here the slower FS regression ultimately outperforms forward stepwise. LAR and lasso show similar behavior to FS and  $FS_0$ . Since the procedures take different numbers of steps (across simulation replicates and methods), we plot the MSE as a function of the fraction of total  $L_1$  arc-length toward the least-squares fit.

adaptively fitted to the training data. This definition is motivated and discussed further in Sections 7.4–7.6.

Now for a linear regression with  $k$  fixed predictors, it is easy to show that  $\text{df}(\hat{\mathbf{y}}) = k$ . Likewise for ridge regression, this definition leads to the closed-form expression (3.50) on page 68:  $\text{df}(\hat{\mathbf{y}}) = \text{tr}(\mathbf{S}_\lambda)$ . In both these cases, (3.60) is simple to evaluate because the fit  $\hat{\mathbf{y}} = \mathbf{H}_\lambda \mathbf{y}$  is linear in  $\mathbf{y}$ . If we think about definition (3.60) in the context of a best subset selection of size  $k$ , it seems clear that  $\text{df}(\hat{\mathbf{y}})$  will be larger than  $k$ , and this can be verified by estimating  $\text{Cov}(\hat{y}_i, y_i)/\sigma^2$  directly by simulation. However there is no closed form method for estimating  $\text{df}(\hat{\mathbf{y}})$  for best subset selection.

For LAR and lasso, something magical happens. These techniques are adaptive in a smoother way than best subset selection, and hence estimation of degrees of freedom is more tractable. Specifically it can be shown that after the  $k$ th step of the LAR procedure, the effective degrees of freedom of the fit vector is exactly  $k$ . Now for the lasso, the (modified) LAR procedure

often takes more than  $p$  steps, since predictors can drop out. Hence the definition is a little different; for the lasso, at any stage  $\text{df}(\hat{\mathbf{y}})$  approximately equals the number of predictors in the model. While this approximation works reasonably well anywhere in the lasso path, for each  $k$  it works best at the *last* model in the sequence that contains  $k$  predictors. A detailed study of the degrees of freedom for the lasso may be found in Zou et al. (2007).

## 3.5 Methods Using Derived Input Directions

In many situations we have a large number of inputs, often very correlated. The methods in this section produce a small number of linear combinations  $Z_m$ ,  $m = 1, \dots, M$  of the original inputs  $X_j$ , and the  $Z_m$  are then used in place of the  $X_j$  as inputs in the regression. The methods differ in how the linear combinations are constructed.

### 3.5.1 Principal Components Regression

In this approach the linear combinations  $Z_m$  used are the principal components as defined in Section 3.4.1 above.

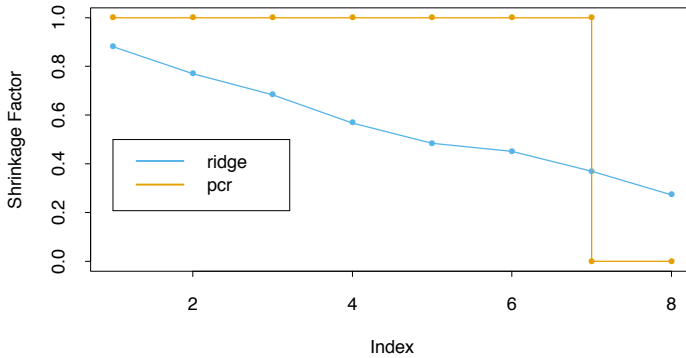
Principal component regression forms the derived input columns  $\mathbf{z}_m = \mathbf{X}v_m$ , and then regresses  $\mathbf{y}$  on  $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_M$  for some  $M \leq p$ . Since the  $\mathbf{z}_m$  are orthogonal, this regression is just a sum of univariate regressions:

$$\hat{\mathbf{y}}_{(M)}^{\text{pcr}} = \bar{y}\mathbf{1} + \sum_{m=1}^M \hat{\theta}_m \mathbf{z}_m, \quad (3.61)$$

where  $\hat{\theta}_m = \langle \mathbf{z}_m, \mathbf{y} \rangle / \langle \mathbf{z}_m, \mathbf{z}_m \rangle$ . Since the  $\mathbf{z}_m$  are each linear combinations of the original  $\mathbf{x}_j$ , we can express the solution (3.61) in terms of coefficients of the  $\mathbf{x}_j$  (Exercise 3.13):

$$\hat{\beta}^{\text{pcr}}(M) = \sum_{m=1}^M \hat{\theta}_m v_m. \quad (3.62)$$

As with ridge regression, principal components depend on the scaling of the inputs, so typically we first standardize them. Note that if  $M = p$ , we would just get back the usual least squares estimates, since the columns of  $\mathbf{Z} = \mathbf{U}\mathbf{D}$  span the column space of  $\mathbf{X}$ . For  $M < p$  we get a reduced regression. We see that principal components regression is very similar to ridge regression: both operate via the principal components of the input matrix. Ridge regression shrinks the coefficients of the principal components (Figure 3.17), shrinking more depending on the size of the corresponding eigenvalue; principal components regression discards the  $p - M$  smallest eigenvalue components. Figure 3.17 illustrates this.



**FIGURE 3.17.** Ridge regression shrinks the regression coefficients of the principal components, using shrinkage factors  $d_j^2/(d_j^2 + \lambda)$  as in (3.47). Principal component regression truncates them. Shown are the shrinkage and truncation patterns corresponding to Figure 3.7, as a function of the principal component index.

In Figure 3.7 we see that cross-validation suggests seven terms; the resulting model has the lowest test error in Table 3.3.

### 3.5.2 Partial Least Squares

This technique also constructs a set of linear combinations of the inputs for regression, but unlike principal components regression it uses  $\mathbf{y}$  (in addition to  $\mathbf{X}$ ) for this construction. Like principal component regression, partial least squares (PLS) is not scale invariant, so we assume that each  $\mathbf{x}_j$  is standardized to have mean 0 and variance 1. PLS begins by computing  $\hat{\varphi}_{1j} = \langle \mathbf{x}_j, \mathbf{y} \rangle$  for each  $j$ . From this we construct the derived input  $\mathbf{z}_1 = \sum_j \hat{\varphi}_{1j} \mathbf{x}_j$ , which is the first partial least squares direction. Hence in the construction of each  $\mathbf{z}_m$ , the inputs are weighted by the strength of their univariate effect on  $\mathbf{y}$ <sup>3</sup>. The outcome  $\mathbf{y}$  is regressed on  $\mathbf{z}_1$  giving coefficient  $\hat{\theta}_1$ , and then we orthogonalize  $\mathbf{x}_1, \dots, \mathbf{x}_p$  with respect to  $\mathbf{z}_1$ . We continue this process, until  $M \leq p$  directions have been obtained. In this manner, partial least squares produces a sequence of derived, orthogonal inputs or directions  $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_M$ . As with principal-component regression, if we were to construct all  $M = p$  directions, we would get back a solution equivalent to the usual least squares estimates; using  $M < p$  directions produces a reduced regression. The procedure is described fully in Algorithm 3.3.

<sup>3</sup>Since the  $\mathbf{x}_j$  are standardized, the first directions  $\hat{\varphi}_{1j}$  are the univariate regression coefficients (up to an irrelevant constant); this is not the case for subsequent directions.

**Algorithm 3.3** *Partial Least Squares.*

1. Standardize each  $\mathbf{x}_j$  to have mean zero and variance one. Set  $\hat{\mathbf{y}}^{(0)} = \bar{y}\mathbf{1}$ , and  $\mathbf{x}_j^{(0)} = \mathbf{x}_j$ ,  $j = 1, \dots, p$ .
2. For  $m = 1, 2, \dots, p$ 
  - (a)  $\mathbf{z}_m = \sum_{j=1}^p \hat{\varphi}_{mj} \mathbf{x}_j^{(m-1)}$ , where  $\hat{\varphi}_{mj} = \langle \mathbf{x}_j^{(m-1)}, \mathbf{y} \rangle$ .
  - (b)  $\hat{\theta}_m = \langle \mathbf{z}_m, \mathbf{y} \rangle / \langle \mathbf{z}_m, \mathbf{z}_m \rangle$ .
  - (c)  $\hat{\mathbf{y}}^{(m)} = \hat{\mathbf{y}}^{(m-1)} + \hat{\theta}_m \mathbf{z}_m$ .
  - (d) Orthogonalize each  $\mathbf{x}_j^{(m-1)}$  with respect to  $\mathbf{z}_m$ :  $\mathbf{x}_j^{(m)} = \mathbf{x}_j^{(m-1)} - [\langle \mathbf{z}_m, \mathbf{x}_j^{(m-1)} \rangle / \langle \mathbf{z}_m, \mathbf{z}_m \rangle] \mathbf{z}_m$ ,  $j = 1, 2, \dots, p$ .
3. Output the sequence of fitted vectors  $\{\hat{\mathbf{y}}^{(m)}\}_1^p$ . Since the  $\{\mathbf{z}_\ell\}_1^m$  are linear in the original  $\mathbf{x}_j$ , so is  $\hat{\mathbf{y}}^{(m)} = \mathbf{X} \hat{\beta}^{\text{pls}}(m)$ . These linear coefficients can be recovered from the sequence of PLS transformations.

In the prostate cancer example, cross-validation chose  $M = 2$  PLS directions in Figure 3.7. This produced the model given in the rightmost column of Table 3.3.

What optimization problem is partial least squares solving? Since it uses the response  $\mathbf{y}$  to construct its directions, its solution path is a nonlinear function of  $\mathbf{y}$ . It can be shown (Exercise 3.15) that partial least squares seeks directions that have high variance *and* have high correlation with the response, in contrast to principal components regression which keys only on high variance (Stone and Brooks, 1990; Frank and Friedman, 1993). In particular, the  $m$ th principal component direction  $v_m$  solves:

$$\begin{aligned} & \max_{\alpha} \text{Var}(\mathbf{X}\alpha) \\ & \text{subject to } \|\alpha\| = 1, \alpha^T \mathbf{S} v_\ell = 0, \ell = 1, \dots, m-1, \end{aligned} \quad (3.63)$$

where  $\mathbf{S}$  is the sample covariance matrix of the  $\mathbf{x}_j$ . The conditions  $\alpha^T \mathbf{S} v_\ell = 0$  ensures that  $\mathbf{z}_m = \mathbf{X}\alpha$  is uncorrelated with all the previous linear combinations  $\mathbf{z}_\ell = \mathbf{X}v_\ell$ . The  $m$ th PLS direction  $\hat{\varphi}_m$  solves:

$$\begin{aligned} & \max_{\alpha} \text{Corr}^2(\mathbf{y}, \mathbf{X}\alpha) \text{Var}(\mathbf{X}\alpha) \\ & \text{subject to } \|\alpha\| = 1, \alpha^T \mathbf{S} \hat{\varphi}_\ell = 0, \ell = 1, \dots, m-1. \end{aligned} \quad (3.64)$$

Further analysis reveals that the variance aspect tends to dominate, and so partial least squares behaves much like ridge regression and principal components regression. We discuss this further in the next section.

If the input matrix  $\mathbf{X}$  is orthogonal, then partial least squares finds the least squares estimates after  $m = 1$  steps. Subsequent steps have no effect

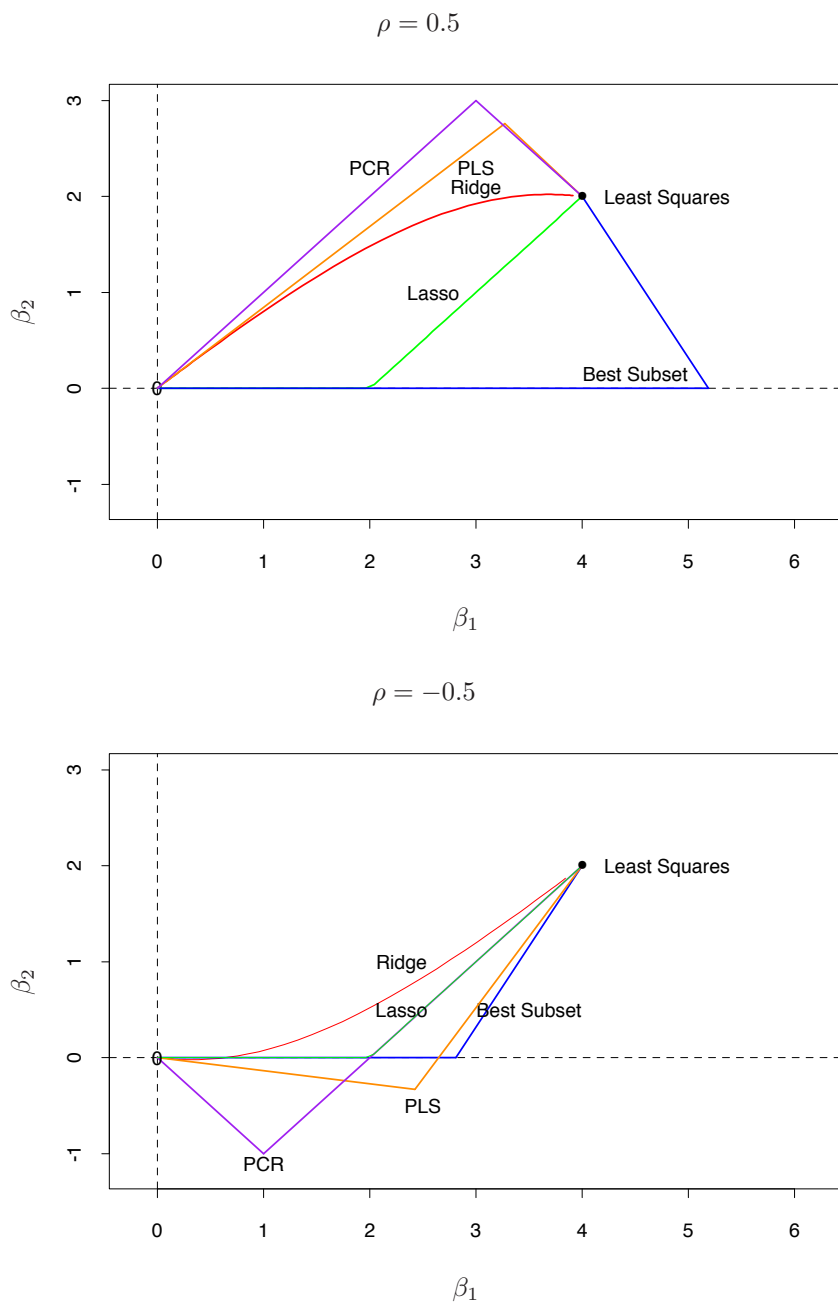
since the  $\hat{\varphi}_{mj}$  are zero for  $m > 1$  (Exercise 3.14). It can also be shown that the sequence of PLS coefficients for  $m = 1, 2, \dots, p$  represents the conjugate gradient sequence for computing the least squares solutions (Exercise 3.18).

## 3.6 Discussion: A Comparison of the Selection and Shrinkage Methods

There are some simple settings where we can understand better the relationship between the different methods described above. Consider an example with two correlated inputs  $X_1$  and  $X_2$ , with correlation  $\rho$ . We assume that the true regression coefficients are  $\beta_1 = 4$  and  $\beta_2 = 2$ . Figure 3.18 shows the coefficient profiles for the different methods, as their tuning parameters are varied. The top panel has  $\rho = 0.5$ , the bottom panel  $\rho = -0.5$ . The tuning parameters for ridge and lasso vary over a continuous range, while best subset, PLS and PCR take just two discrete steps to the least squares solution. In the top panel, starting at the origin, ridge regression shrinks the coefficients together until it finally converges to least squares. PLS and PCR show similar behavior to ridge, although are discrete and more extreme. Best subset overshoots the solution and then backtracks. The behavior of the lasso is intermediate to the other methods. When the correlation is negative (lower panel), again PLS and PCR roughly track the ridge path, while all of the methods are more similar to one another.

It is interesting to compare the shrinkage behavior of these different methods. Recall that ridge regression shrinks all directions, but shrinks low-variance directions more. Principal components regression leaves  $M$  high-variance directions alone, and discards the rest. Interestingly, it can be shown that partial least squares also tends to shrink the low-variance directions, but can actually inflate some of the higher variance directions. This can make PLS a little unstable, and cause it to have slightly higher prediction error compared to ridge regression. A full study is given in Frank and Friedman (1993). These authors conclude that for minimizing prediction error, ridge regression is generally preferable to variable subset selection, principal components regression and partial least squares. However the improvement over the latter two methods was only slight.

To summarize, PLS, PCR and ridge regression tend to behave similarly. Ridge regression may be preferred because it shrinks smoothly, rather than in discrete steps. Lasso falls somewhere between ridge regression and best subset regression, and enjoys some of the properties of each.



**FIGURE 3.18.** Coefficient profiles from different methods for a simple problem: two inputs with correlation  $\pm 0.5$ , and the true regression coefficients  $\beta = (4, 2)$ .

## 3.7 Multiple Outcome Shrinkage and Selection



As noted in Section 3.2.4, the least squares estimates in a multiple-output linear model are simply the individual least squares estimates for each of the outputs.

To apply selection and shrinkage methods in the multiple output case, one could apply a univariate technique individually to each outcome or simultaneously to all outcomes. With ridge regression, for example, we could apply formula (3.44) to each of the  $K$  columns of the outcome matrix  $\mathbf{Y}$ , using possibly different parameters  $\lambda$ , or apply it to all columns using the same value of  $\lambda$ . The former strategy would allow different amounts of regularization to be applied to different outcomes but require estimation of  $k$  separate regularization parameters  $\lambda_1, \dots, \lambda_k$ , while the latter would permit all  $k$  outputs to be used in estimating the sole regularization parameter  $\lambda$ .

Other more sophisticated shrinkage and selection strategies that exploit correlations in the different responses can be helpful in the multiple output case. Suppose for example that among the outputs we have

$$Y_k = f(X) + \varepsilon_k \quad (3.65)$$

$$Y_\ell = f(X) + \varepsilon_\ell; \quad (3.66)$$

i.e., (3.65) and (3.66) share the same structural part  $f(X)$  in their models. It is clear in this case that we should pool our observations on  $Y_k$  and  $Y_\ell$  to estimate the common  $f$ .

Combining responses is at the heart of *canonical correlation analysis* (CCA), a data reduction technique developed for the multiple output case. Similar to PCA, CCA finds a sequence of uncorrelated linear combinations  $\mathbf{X}v_m$ ,  $m = 1, \dots, M$  of the  $\mathbf{x}_j$ , and a corresponding sequence of uncorrelated linear combinations  $\mathbf{Y}u_m$  of the responses  $\mathbf{y}_k$ , such that the correlations

$$\text{Corr}^2(\mathbf{Y}u_m, \mathbf{X}v_m) \quad (3.67)$$

are successively maximized. Note that at most  $M = \min(K, p)$  directions can be found. The leading canonical response variates are those linear combinations (derived responses) best predicted by the  $\mathbf{x}_j$ ; in contrast, the trailing canonical variates can be poorly predicted by the  $\mathbf{x}_j$ , and are candidates for being dropped. The CCA solution is computed using a generalized SVD of the sample cross-covariance matrix  $\mathbf{Y}^T \mathbf{X} / N$  (assuming  $\mathbf{Y}$  and  $\mathbf{X}$  are centered; Exercise 3.20).

*Reduced-rank regression* (Izenman, 1975; van der Merwe and Zidek, 1980) formalizes this approach in terms of a regression model that explicitly pools information. Given an error covariance  $\text{Cov}(\varepsilon) = \mathbf{\Sigma}$ , we solve the following

restricted multivariate regression problem:

$$\hat{\mathbf{B}}^{\text{rr}}(m) = \underset{\text{rank}(\mathbf{B})=m}{\text{argmin}} \sum_{i=1}^N (y_i - \mathbf{B}^T x_i)^T \boldsymbol{\Sigma}^{-1} (y_i - \mathbf{B}^T x_i). \quad (3.68)$$

With  $\boldsymbol{\Sigma}$  replaced by the estimate  $\mathbf{Y}^T \mathbf{Y} / N$ , one can show (Exercise 3.21) that the solution is given by a CCA of  $\mathbf{Y}$  and  $\mathbf{X}$ :

$$\hat{\mathbf{B}}^{\text{rr}}(m) = \hat{\mathbf{B}} \mathbf{U}_m \mathbf{U}_m^{-}, \quad (3.69)$$

where  $\mathbf{U}_m$  is the  $K \times m$  sub-matrix of  $\mathbf{U}$  consisting of the first  $m$  columns, and  $\mathbf{U}$  is the  $K \times M$  matrix of *left* canonical vectors  $u_1, u_2, \dots, u_M$ .  $\mathbf{U}_m^{-}$  is its generalized inverse. Writing the solution as

$$\hat{\mathbf{B}}^{\text{rr}}(M) = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{Y} \mathbf{U}_m) \mathbf{U}_m^{-}, \quad (3.70)$$

we see that reduced-rank regression performs a linear regression on the pooled response matrix  $\mathbf{Y} \mathbf{U}_m$ , and then maps the coefficients (and hence the fits as well) back to the original response space. The reduced-rank fits are given by

$$\begin{aligned} \hat{\mathbf{Y}}^{\text{rr}}(m) &= \mathbf{X} (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \mathbf{U}_m \mathbf{U}_m^{-} \\ &= \mathbf{HYP}_m, \end{aligned} \quad (3.71)$$

where  $\mathbf{H}$  is the usual linear regression projection operator, and  $\mathbf{P}_m$  is the rank- $m$  CCA response projection operator. Although a better estimate of  $\boldsymbol{\Sigma}$  would be  $(\mathbf{Y} - \mathbf{X} \hat{\mathbf{B}})^T (\mathbf{Y} - \mathbf{X} \hat{\mathbf{B}}) / (N - pK)$ , one can show that the solution remains the same (Exercise 3.22).

Reduced-rank regression borrows strength among responses by truncating the CCA. Breiman and Friedman (1997) explored with some success shrinkage of the canonical variates between  $\mathbf{X}$  and  $\mathbf{Y}$ , a smooth version of *reduced rank* regression. Their proposal has the form (compare (3.69))

$$\hat{\mathbf{B}}^{\text{c+w}} = \hat{\mathbf{B}} \mathbf{U} \boldsymbol{\Lambda} \mathbf{U}^{-1}, \quad (3.72)$$

where  $\boldsymbol{\Lambda}$  is a diagonal shrinkage matrix (the “c+w” stands for “Curds and Whey,” the name they gave to their procedure). Based on optimal prediction in the population setting, they show that  $\boldsymbol{\Lambda}$  has diagonal entries

$$\lambda_m = \frac{c_m^2}{c_m^2 + \frac{p}{N}(1 - c_m^2)}, \quad m = 1, \dots, M, \quad (3.73)$$

where  $c_m$  is the  $m$ th canonical correlation coefficient. Note that as the ratio of the number of input variables to sample size  $p/N$  gets small, the shrinkage factors approach 1. Breiman and Friedman (1997) proposed modified versions of  $\boldsymbol{\Lambda}$  based on training data and cross-validation, but the general form is the same. Here the fitted response has the form

$$\hat{\mathbf{Y}}^{\text{c+w}} = \mathbf{HYS}^{\text{c+w}}, \quad (3.74)$$



where  $\mathbf{S}^{c+w} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}$  is the response shrinkage operator.

Breiman and Friedman (1997) also suggested shrinking in both the  $Y$  space and  $X$  space. This leads to hybrid shrinkage models of the form

$$\hat{\mathbf{Y}}^{\text{ridge},c+w} = \mathbf{A}_\lambda \mathbf{Y} \mathbf{S}^{c+w}, \quad (3.75)$$

where  $\mathbf{A}_\lambda = \mathbf{X}(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T$  is the ridge regression shrinkage operator, as in (3.46) on page 66. Their paper and the discussions thereof contain many more details.

## 3.8 More on the Lasso and Related Path Algorithms

Since the publication of the LAR algorithm (Efron et al., 2004) there has been a lot of activity in developing algorithms for fitting regularization paths for a variety of different problems. In addition,  $L_1$  regularization has taken on a life of its own, leading to the development of the field *compressed sensing* in the signal-processing literature. (Donoho, 2006a; Candes, 2006). In this section we discuss some related proposals and other path algorithms, starting off with a precursor to the LAR algorithm.

### 3.8.1 Incremental Forward Stagewise Regression

Here we present another LAR-like algorithm, this time focused on forward stagewise regression. Interestingly, efforts to understand a flexible nonlinear regression procedure (boosting) led to a new algorithm for linear models (LAR). In reading the first edition of this book and the forward stagewise

---

**Algorithm 3.4** *Incremental Forward Stagewise Regression— $FS_\epsilon$ .*

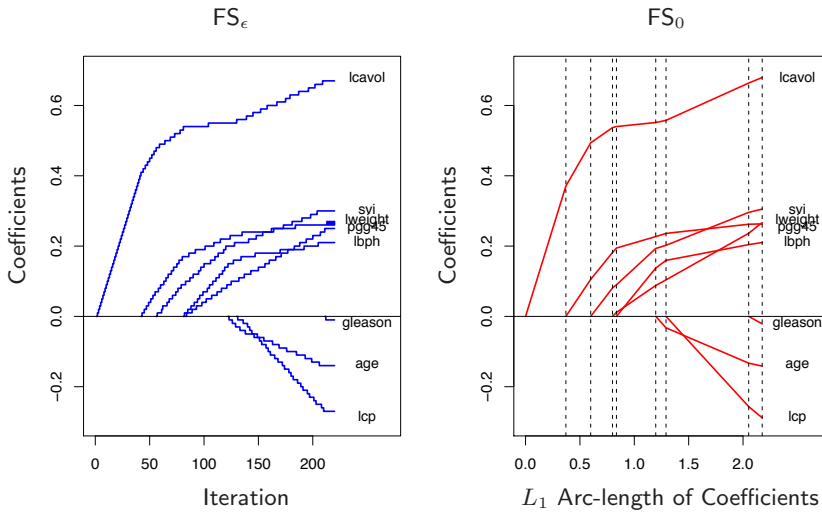
---

1. Start with the residual  $\mathbf{r}$  equal to  $\mathbf{y}$  and  $\beta_1, \beta_2, \dots, \beta_p = 0$ . All the predictors are standardized to have mean zero and unit norm.
  2. Find the predictor  $\mathbf{x}_j$  most correlated with  $\mathbf{r}$
  3. Update  $\beta_j \leftarrow \beta_j + \delta_j$ , where  $\delta_j = \epsilon \cdot \text{sign}[(\mathbf{x}_j, \mathbf{r})]$  and  $\epsilon > 0$  is a small step size, and set  $\mathbf{r} \leftarrow \mathbf{r} - \delta_j \mathbf{x}_j$ .
  4. Repeat steps 2 and 3 many times, until the residuals are uncorrelated with all the predictors.
- 

Algorithm 16.1 of Chapter 16<sup>4</sup>, our colleague Brad Efron realized that with

---

<sup>4</sup>In the first edition, this was Algorithm 10.4 in Chapter 10.



**FIGURE 3.19.** Coefficient profiles for the prostate data. The left panel shows incremental forward stagewise regression with step size  $\epsilon = 0.01$ . The right panel shows the infinitesimal version  $FS_0$  obtained letting  $\epsilon \rightarrow 0$ . This profile was fit by the modification 3.2b to the LAR Algorithm 3.2. In this example the  $FS_0$  profiles are monotone, and hence identical to those of lasso and LAR.

linear models, one could explicitly construct the piecewise-linear lasso paths of Figure 3.10. This led him to propose the LAR procedure of Section 3.4.4, as well as the incremental version of forward-stagewise regression presented here.

Consider the linear-regression version of the forward-stagewise boosting algorithm 16.1 proposed in Section 16.1 (page 608). It generates a coefficient profile by repeatedly updating (by a small amount  $\epsilon$ ) the coefficient of the variable most correlated with the current residuals. Algorithm 3.4 gives the details. Figure 3.19 (left panel) shows the progress of the algorithm on the prostate data with step size  $\epsilon = 0.01$ . If  $\delta_j = \langle \mathbf{x}_j, \mathbf{r} \rangle$  (the least-squares coefficient of the residual on  $j$ th predictor), then this is exactly the usual forward stagewise procedure (FS) outlined in Section 3.3.3.

Here we are mainly interested in small values of  $\epsilon$ . Letting  $\epsilon \rightarrow 0$  gives the right panel of Figure 3.19, which in this case is identical to the lasso path in Figure 3.10. We call this limiting procedure *infinitesimal forward stagewise regression* or  $FS_0$ . This procedure plays an important role in non-linear, adaptive methods like boosting (Chapters 10 and 16) and is the version of incremental forward stagewise regression that is most amenable to theoretical analysis. Bühlmann and Hothorn (2007) refer to the same procedure as “L2boost”, because of its connections to boosting.

Efron originally thought that the LAR Algorithm 3.2 was an implementation of  $FS_0$ , allowing each tied predictor a chance to update their coefficients in a balanced way, while remaining tied in correlation. However, he then realized that the LAR least-squares fit amongst the tied predictors can result in coefficients moving in the *opposite* direction to their correlation, which cannot happen in Algorithm 3.4. The following modification of the LAR algorithm implements  $FS_0$ :

---

**Algorithm 3.2b** *Least Angle Regression:  $FS_0$  Modification.*

---

4. Find the new direction by solving the constrained least squares problem

$$\min_b \|\mathbf{r} - \mathbf{X}_{\mathcal{A}}b\|_2^2 \text{ subject to } b_j s_j \geq 0, j \in \mathcal{A},$$

where  $s_j$  is the sign of  $\langle \mathbf{x}_j, \mathbf{r} \rangle$ .

---

The modification amounts to a non-negative least squares fit, keeping the signs of the coefficients the same as those of the correlations. One can show that this achieves the optimal balancing of infinitesimal “update turns” for the variables tied for maximal correlation (Hastie et al., 2007). Like lasso, the entire  $FS_0$  path can be computed very efficiently via the LAR algorithm.

As a consequence of these results, if the LAR profiles are monotone non-increasing or non-decreasing, as they are in Figure 3.19, then all three methods—LAR, lasso, and  $FS_0$ —give identical profiles. If the profiles are not monotone but do not cross the zero axis, then LAR and lasso are identical.

Since  $FS_0$  is different from the lasso, it is natural to ask if it optimizes a criterion. The answer is more complex than for lasso; the  $FS_0$  coefficient profile is the solution to a differential equation. While the lasso makes optimal progress in terms of reducing the residual sum-of-squares per unit increase in  $L_1$ -norm of the coefficient vector  $\beta$ ,  $FS_0$  is optimal per unit increase in  $L_1$  arc-length traveled along the coefficient path. Hence its coefficient path is discouraged from changing directions too often.

$FS_0$  is more constrained than lasso, and in fact can be viewed as a monotone version of the lasso; see Figure 16.3 on page 614 for a dramatic example.  $FS_0$  may be useful in  $p \gg N$  situations, where its coefficient profiles are much smoother and hence have less variance than those of lasso. More details on  $FS_0$  are given in Section 16.2.3 and Hastie et al. (2007). Figure 3.16 includes  $FS_0$  where its performance is very similar to that of the lasso.

### 3.8.2 Piecewise-Linear Path Algorithms

The least angle regression procedure exploits the piecewise linear nature of the lasso solution paths. It has led to similar “path algorithms” for other regularized problems. Suppose we solve

$$\hat{\beta}(\lambda) = \operatorname{argmin}_{\beta} [R(\beta) + \lambda J(\beta)], \quad (3.76)$$

with

$$R(\beta) = \sum_{i=1}^N L(y_i, \beta_0 + \sum_{j=1}^p x_{ij} \beta_j), \quad (3.77)$$

where both the loss function  $L$  and the penalty function  $J$  are convex. Then the following are sufficient conditions for the solution path  $\hat{\beta}(\lambda)$  to be piecewise linear (Rosset and Zhu, 2007):

1.  $R$  is quadratic or piecewise-quadratic as a function of  $\beta$ , and
2.  $J$  is piecewise linear in  $\beta$ .

This also implies (in principle) that the solution path can be efficiently computed. Examples include squared- and absolute-error loss, “Huberized” losses, and the  $L_1, L_\infty$  penalties on  $\beta$ . Another example is the “hinge loss” function used in the support vector machine. There the loss is piecewise linear, and the penalty is quadratic. Interestingly, this leads to a piecewise-linear path algorithm in the *dual space*; more details are given in Section 12.3.5.

### 3.8.3 The Dantzig Selector

Candes and Tao (2007) proposed the following criterion:

$$\min_{\beta} \|\beta\|_1 \text{ subject to } \|\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)\|_\infty \leq s. \quad (3.78)$$

They call the solution the *Dantzig selector* (DS). It can be written equivalently as

$$\min_{\beta} \|\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)\|_\infty \text{ subject to } \|\beta\|_1 \leq t. \quad (3.79)$$

Here  $\|\cdot\|_\infty$  denotes the  $L_\infty$  norm, the maximum absolute value of the components of the vector. In this form it resembles the lasso, replacing squared error loss by the maximum absolute value of its gradient. Note that as  $t$  gets large, both procedures yield the least squares solution if  $N < p$ . If  $p \geq N$ , they both yield the least squares solution with minimum  $L_1$  norm. However for smaller values of  $t$ , the DS procedure produces a different path of solutions than the lasso.

Candes and Tao (2007) show that the solution to DS is a linear programming problem; hence the name Dantzig selector, in honor of the late

George Dantzig, the inventor of the simplex method for linear programming. They also prove a number of interesting mathematical properties for the method, related to its ability to recover an underlying sparse coefficient vector. These same properties also hold for the lasso, as shown later by Bickel et al. (2008).

Unfortunately the operating properties of the DS method are somewhat unsatisfactory. The method seems similar in spirit to the lasso, especially when we look at the lasso's stationary conditions (3.58). Like the LAR algorithm, the lasso maintains the same inner product (and correlation) with the current residual for all variables in the active set, and moves their coefficients to optimally decrease the residual sum of squares. In the process, this common correlation is decreased monotonically (Exercise 3.23), and at all times this correlation is larger than that for non-active variables. The Dantzig selector instead tries to minimize the maximum inner product of the current residual with all the predictors. Hence it can achieve a smaller maximum than the lasso, but in the process a curious phenomenon can occur. If the size of the active set is  $m$ , there will be  $m$  variables tied with maximum correlation. However, these need not coincide with the active set! Hence it can include a variable in the model that has smaller correlation with the current residual than some of the excluded variables (Efron et al., 2007). This seems unreasonable and may be responsible for its sometimes inferior prediction accuracy. Efron et al. (2007) also show that DS can yield extremely erratic coefficient paths as the regularization parameter  $s$  is varied.

### 3.8.4 The Grouped Lasso

In some problems, the predictors belong to pre-defined groups; for example genes that belong to the same biological pathway, or collections of indicator (dummy) variables for representing the levels of a categorical predictor. In this situation it may be desirable to shrink and select the members of a group together. The *grouped lasso* is one way to achieve this. Suppose that the  $p$  predictors are divided into  $L$  groups, with  $p_\ell$  the number in group  $\ell$ . For ease of notation, we use a matrix  $\mathbf{X}_\ell$  to represent the predictors corresponding to the  $\ell$ th group, with corresponding coefficient vector  $\beta_\ell$ . The grouped-lasso minimizes the convex criterion

$$\min_{\beta \in \mathbb{R}^p} \left( \|\mathbf{y} - \beta_0 \mathbf{1} - \sum_{\ell=1}^L \mathbf{X}_\ell \beta_\ell\|_2^2 + \lambda \sum_{\ell=1}^L \sqrt{p_\ell} \|\beta_\ell\|_2 \right), \quad (3.80)$$

where the  $\sqrt{p_\ell}$  terms accounts for the varying group sizes, and  $\|\cdot\|_2$  is the Euclidean norm (not squared). Since the Euclidean norm of a vector  $\beta_\ell$  is zero only if all of its components are zero, this procedure encourages sparsity at both the group and individual levels. That is, for some values of  $\lambda$ , an entire group of predictors may drop out of the model. This procedure

was proposed by Bakin (1999) and Lin and Zhang (2006), and studied and generalized by Yuan and Lin (2007). Generalizations include more general  $L_2$  norms  $\|\eta\|_K = (\eta^T K \eta)^{1/2}$ , as well as allowing overlapping groups of predictors (Zhao et al., 2008). There are also connections to methods for fitting sparse additive models (Lin and Zhang, 2006; Ravikumar et al., 2008).

### 3.8.5 Further Properties of the Lasso

A number of authors have studied the ability of the lasso and related procedures to recover the correct model, as  $N$  and  $p$  grow. Examples of this work include Knight and Fu (2000), Greenshtein and Ritov (2004), Tropp (2004), Donoho (2006b), Meinshausen (2007), Meinshausen and Bühlmann (2006), Tropp (2006), Zhao and Yu (2006), Wainwright (2006), and Bunea et al. (2007). For example Donoho (2006b) focuses on the  $p > N$  case and considers the lasso solution as the bound  $t$  gets large. In the limit this gives the solution with minimum  $L_1$  norm among all models with zero training error. He shows that under certain assumptions on the model matrix  $\mathbf{X}$ , if the true model is sparse, this solution identifies the correct predictors with high probability.

Many of the results in this area assume a condition on the model matrix of the form

$$\max_{j \in \mathcal{S}^c} \|\mathbf{x}_j^T \mathbf{X}_{\mathcal{S}} (\mathbf{X}_{\mathcal{S}}^T \mathbf{X}_{\mathcal{S}})^{-1}\|_1 \leq (1 - \epsilon) \text{ for some } \epsilon \in (0, 1]. \quad (3.81)$$

Here  $\mathcal{S}$  indexes the subset of features with non-zero coefficients in the true underlying model, and  $\mathbf{X}_{\mathcal{S}}$  are the columns of  $\mathbf{X}$  corresponding to those features. Similarly  $\mathcal{S}^c$  are the features with true coefficients equal to zero, and  $\mathbf{X}_{\mathcal{S}^c}$  the corresponding columns. This says that the least squares coefficients for the columns of  $\mathbf{X}_{\mathcal{S}^c}$  on  $\mathbf{X}_{\mathcal{S}}$  are not too large, that is, the “good” variables  $\mathcal{S}$  are not too highly correlated with the nuisance variables  $\mathcal{S}^c$ .

Regarding the coefficients themselves, the lasso shrinkage causes the estimates of the non-zero coefficients to be biased towards zero, and in general they are not consistent<sup>5</sup>. One approach for reducing this bias is to run the lasso to identify the set of non-zero coefficients, and then fit an unrestricted linear model to the selected set of features. This is not always feasible, if the selected set is large. Alternatively, one can use the lasso to select the set of non-zero predictors, and then apply the lasso again, but using only the selected predictors from the first step. This is known as the *relaxed lasso* (Meinshausen, 2007). The idea is to use cross-validation to estimate the initial penalty parameter for the lasso, and then again for a second penalty parameter applied to the selected set of predictors. Since

---

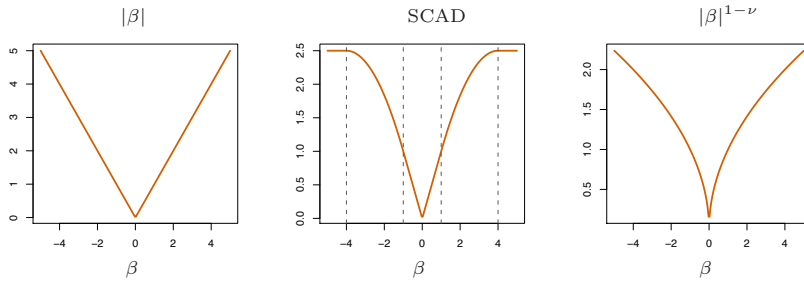
<sup>5</sup>Statistical consistency means as the sample size grows, the estimates converge to the true values.

the variables in the second step have less “competition” from noise variables, cross-validation will tend to pick a smaller value for  $\lambda$ , and hence their coefficients will be shrunk less than those in the initial estimate.

Alternatively, one can modify the lasso penalty function so that larger coefficients are shrunk less severely; the *smoothly clipped absolute deviation* (SCAD) penalty of Fan and Li (2005) replaces  $\lambda|\beta|$  by  $J_a(\beta, \lambda)$ , where

$$\frac{dJ_a(\beta, \lambda)}{d\beta} = \lambda \cdot \text{sign}(\beta) \left[ I(|\beta| \leq \lambda) + \frac{(a\lambda - |\beta|)_+}{(a-1)\lambda} I(|\beta| > \lambda) \right] \quad (3.82)$$

for some  $a \geq 2$ . The second term in square-braces reduces the amount of shrinkage in the lasso for larger values of  $\beta$ , with ultimately no shrinkage as  $a \rightarrow \infty$ . Figure 3.20 shows the SCAD penalty, along with the lasso and



**FIGURE 3.20.** The lasso and two alternative non-convex penalties designed to penalize large coefficients less. For SCAD we use  $\lambda = 1$  and  $a = 4$ , and  $\nu = \frac{1}{2}$  in the last panel.

$|\beta|^{1-\nu}$ . However this criterion is non-convex, which is a drawback since it makes the computation much more difficult. The *adaptive lasso* (Zou, 2006) uses a weighted penalty of the form  $\sum_{j=1}^p w_j |\beta_j|$  where  $w_j = 1/|\hat{\beta}_j|^\nu$ ,  $\hat{\beta}_j$  is the ordinary least squares estimate and  $\nu > 0$ . This is a practical approximation to the  $|\beta|^q$  penalties ( $q = 1 - \nu$  here) discussed in Section 3.4.3. The adaptive lasso yields consistent estimates of the parameters while retaining the attractive convexity property of the lasso.

### 3.8.6 Pathwise Coordinate Optimization

An alternate approach to the LARS algorithm for computing the lasso solution is simple coordinate descent. This idea was proposed by Fu (1998) and Daubechies et al. (2004), and later studied and generalized by Friedman et al. (2007), Wu and Lange (2008) and others. The idea is to fix the penalty parameter  $\lambda$  in the Lagrangian form (3.52) and optimize successively over each parameter, holding the other parameters fixed at their current values.

Suppose the predictors are all standardized to have mean zero and unit norm. Denote by  $\tilde{\beta}_k(\lambda)$  the current estimate for  $\beta_k$  at penalty parameter

$\lambda$ . We can rearrange (3.52) to isolate  $\beta_j$ ,

$$R(\tilde{\beta}(\lambda), \beta_j) = \frac{1}{2} \sum_{i=1}^N \left( y_i - \sum_{k \neq j} x_{ik} \tilde{\beta}_k(\lambda) - x_{ij} \beta_j \right)^2 + \lambda \sum_{k \neq j} |\tilde{\beta}_k(\lambda)| + \lambda |\beta_j|, \quad (3.83)$$

where we have suppressed the intercept and introduced a factor  $\frac{1}{2}$  for convenience. This can be viewed as a univariate lasso problem with response variable the partial residual  $y_i - \tilde{y}_i^{(j)} = y_i - \sum_{k \neq j} x_{ik} \tilde{\beta}_k(\lambda)$ . This has an explicit solution, resulting in the update

$$\tilde{\beta}_j(\lambda) \leftarrow S \left( \sum_{i=1}^N x_{ij} (y_i - \tilde{y}_i^{(j)}), \lambda \right). \quad (3.84)$$

Here  $S(t, \lambda) = \text{sign}(t)(|t| - \lambda)_+$  is the soft-thresholding operator in Table 3.4 on page 71. The first argument to  $S(\cdot)$  is the simple least-squares coefficient of the partial residual on the standardized variable  $x_{ij}$ . Repeated iteration of (3.84)—cycling through each variable in turn until convergence—yields the lasso estimate  $\hat{\beta}(\lambda)$ .

We can also use this simple algorithm to efficiently compute the lasso solutions at a grid of values of  $\lambda$ . We start with the smallest value  $\lambda_{\max}$  for which  $\hat{\beta}(\lambda_{\max}) = 0$ , decrease it a little and cycle through the variables until convergence. Then  $\lambda$  is decreased again and the process is repeated, using the previous solution as a “warm start” for the new value of  $\lambda$ . This can be faster than the LARS algorithm, especially in large problems. A key to its speed is the fact that the quantities in (3.84) can be updated quickly as  $j$  varies, and often the update is to leave  $\tilde{\beta}_j = 0$ . On the other hand, it delivers solutions over a grid of  $\lambda$  values, rather than the entire solution path. The same kind of algorithm can be applied to the elastic net, the grouped lasso and many other models in which the penalty is a sum of functions of the individual parameters (Friedman et al., 2010). It can also be applied, with some substantial modifications, to the fused lasso (Section 18.4.2); details are in Friedman et al. (2007).

## 3.9 Computational Considerations

Least squares fitting is usually done via the Cholesky decomposition of the matrix  $\mathbf{X}^T \mathbf{X}$  or a QR decomposition of  $\mathbf{X}$ . With  $N$  observations and  $p$  features, the Cholesky decomposition requires  $p^3 + Np^2/2$  operations, while the QR decomposition requires  $Np^2$  operations. Depending on the relative size of  $N$  and  $p$ , the Cholesky can sometimes be faster; on the other hand, it can be less numerically stable (Lawson and Hansen, 1974). Computation of the lasso via the LAR algorithm has the same order of computation as a least squares fit.



## Bibliographic Notes

Linear regression is discussed in many statistics books, for example, Seber (1984), Weisberg (1980) and Mardia et al. (1979). Ridge regression was introduced by Hoerl and Kennard (1970), while the lasso was proposed by Tibshirani (1996). Around the same time, lasso-type penalties were proposed in the *basis pursuit* method for signal processing (Chen et al., 1998). The least angle regression procedure was proposed in Efron et al. (2004); related to this is the earlier homotopy procedure of Osborne et al. (2000a) and Osborne et al. (2000b). Their algorithm also exploits the piecewise linearity used in the LAR/lasso algorithm, but lacks its transparency. The criterion for the forward stagewise criterion is discussed in Hastie et al. (2007). Park and Hastie (2007) develop a path algorithm similar to least angle regression for generalized regression models. Partial least squares was introduced by Wold (1975). Comparisons of shrinkage methods may be found in Copas (1983) and Frank and Friedman (1993).

## Exercises

**Ex. 3.1** Show that the  $F$  statistic (3.13) for dropping a single coefficient from a model is equal to the square of the corresponding  $z$ -score (3.12).

**Ex. 3.2** Given data on two variables  $X$  and  $Y$ , consider fitting a cubic polynomial regression model  $f(X) = \sum_{j=0}^3 \beta_j X^j$ . In addition to plotting the fitted curve, you would like a 95% confidence band about the curve. Consider the following two approaches:

1. At each point  $x_0$ , form a 95% confidence interval for the linear function  $a^T \beta = \sum_{j=0}^3 \beta_j x_0^j$ .
2. Form a 95% confidence set for  $\beta$  as in (3.15), which in turn generates confidence intervals for  $f(x_0)$ .

How do these approaches differ? Which band is likely to be wider? Conduct a small simulation experiment to compare the two methods.

**Ex. 3.3** Gauss–Markov theorem:

- (a) Prove the Gauss–Markov theorem: the least squares estimate of a parameter  $a^T \beta$  has variance no bigger than that of any other linear unbiased estimate of  $a^T \beta$  (Section 3.2.2).
- (b) The matrix inequality  $\mathbf{B} \preceq \mathbf{A}$  holds if  $\mathbf{A} - \mathbf{B}$  is positive semidefinite. Show that if  $\hat{\mathbf{V}}$  is the variance-covariance matrix of the least squares estimate of  $\beta$  and  $\tilde{\mathbf{V}}$  is the variance-covariance matrix of any other linear unbiased estimate, then  $\hat{\mathbf{V}} \preceq \tilde{\mathbf{V}}$ .

**Ex. 3.4** Show how the vector of least squares coefficients can be obtained from a single pass of the Gram–Schmidt procedure (Algorithm 3.1). Represent your solution in terms of the QR decomposition of  $\mathbf{X}$ .

**Ex. 3.5** Consider the ridge regression problem (3.41). Show that this problem is equivalent to the problem

$$\hat{\beta}^c = \operatorname{argmin}_{\beta^c} \left\{ \sum_{i=1}^N [y_i - \beta_0^c - \sum_{j=1}^p (x_{ij} - \bar{x}_j) \beta_j^c]^2 + \lambda \sum_{j=1}^p \beta_j^{c2} \right\}. \quad (3.85)$$

Give the correspondence between  $\beta^c$  and the original  $\beta$  in (3.41). Characterize the solution to this modified criterion. Show that a similar result holds for the lasso.

**Ex. 3.6** Show that the ridge regression estimate is the mean (and mode) of the posterior distribution, under a Gaussian prior  $\beta \sim N(0, \tau \mathbf{I})$ , and Gaussian sampling model  $\mathbf{y} \sim N(\mathbf{X}\beta, \sigma^2 \mathbf{I})$ . Find the relationship between the regularization parameter  $\lambda$  in the ridge formula, and the variances  $\tau$  and  $\sigma^2$ .

**Ex. 3.7** Assume  $y_i \sim N(\beta_0 + x_i^T \beta, \sigma^2)$ ,  $i = 1, 2, \dots, N$ , and the parameters  $\beta_j$ ,  $j = 1, \dots, p$  are each distributed as  $N(0, \tau^2)$ , independently of one another. Assuming  $\sigma^2$  and  $\tau^2$  are known, and  $\beta_0$  is not governed by a prior (or has a flat improper prior), show that the (minus) log-posterior density of  $\beta$  is proportional to  $\sum_{i=1}^N (y_i - \beta_0 - \sum_j x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2$  where  $\lambda = \sigma^2 / \tau^2$ .

**Ex. 3.8** Consider the QR decomposition of the uncentered  $N \times (p+1)$  matrix  $\mathbf{X}$  (whose first column is all ones), and the SVD of the  $N \times p$  centered matrix  $\tilde{\mathbf{X}}$ . Show that  $\mathbf{Q}_2$  and  $\mathbf{U}$  span the same subspace, where  $\mathbf{Q}_2$  is the sub-matrix of  $\mathbf{Q}$  with the first column removed. Under what circumstances will they be the same, up to sign flips?

**Ex. 3.9** *Forward stepwise regression.* Suppose we have the QR decomposition for the  $N \times q$  matrix  $\mathbf{X}_1$  in a multiple regression problem with response  $\mathbf{y}$ , and we have an additional  $p - q$  predictors in the matrix  $\mathbf{X}_2$ . Denote the current residual by  $\mathbf{r}$ . We wish to establish which one of these additional variables will reduce the residual-sum-of-squares the most when included with those in  $\mathbf{X}_1$ . Describe an efficient procedure for doing this.

**Ex. 3.10** *Backward stepwise regression.* Suppose we have the multiple regression fit of  $\mathbf{y}$  on  $\mathbf{X}$ , along with the standard errors and Z-scores as in Table 3.2. We wish to establish which variable, when dropped, will increase the residual sum-of-squares the least. How would you do this?

**Ex. 3.11** Show that the solution to the multivariate linear regression problem (3.40) is given by (3.39). What happens if the covariance matrices  $\Sigma_i$  are different for each observation?

**Ex. 3.12** Show that the ridge regression estimates can be obtained by ordinary least squares regression on an augmented data set. We augment the centered matrix  $\mathbf{X}$  with  $p$  additional rows  $\sqrt{\lambda}\mathbf{I}$ , and augment  $\mathbf{y}$  with  $p$  zeros. By introducing artificial data having response value zero, the fitting procedure is forced to shrink the coefficients toward zero. This is related to the idea of *hints* due to Abu-Mostafa (1995), where model constraints are implemented by adding artificial data examples that satisfy them.

**Ex. 3.13** Derive the expression (3.62), and show that  $\hat{\beta}^{\text{PCR}}(p) = \hat{\beta}^{\text{ls}}$ .

**Ex. 3.14** Show that in the orthogonal case, PLS stops after  $m = 1$  steps, because subsequent  $\hat{\varphi}_{mj}$  in step 2 in Algorithm 3.3 are zero.

**Ex. 3.15** Verify expression (3.64), and hence show that the partial least squares directions are a compromise between the ordinary regression coefficient and the principal component directions.

**Ex. 3.16** Derive the entries in Table 3.4, the explicit forms for estimators in the orthogonal case.

**Ex. 3.17** Repeat the analysis of Table 3.3 on the spam data discussed in Chapter 1.

**Ex. 3.18** Read about conjugate gradient algorithms (Murray et al., 1981, for example), and establish a connection between these algorithms and partial least squares.

**Ex. 3.19** Show that  $\|\hat{\beta}^{\text{ridge}}\|$  increases as its tuning parameter  $\lambda \rightarrow 0$ . Does the same property hold for the lasso and partial least squares estimates? For the latter, consider the “tuning parameter” to be the successive steps in the algorithm.

**Ex. 3.20** Consider the canonical-correlation problem (3.67). Show that the leading pair of canonical variates  $u_1$  and  $v_1$  solve the problem

$$\max_{\substack{u^T(\mathbf{Y}^T\mathbf{Y})u=1 \\ v^T(\mathbf{X}^T\mathbf{X})v=1}} u^T(\mathbf{Y}^T\mathbf{X})v, \quad (3.86)$$

a generalized SVD problem. Show that the solution is given by  $u_1 = (\mathbf{Y}^T\mathbf{Y})^{-\frac{1}{2}}u_1^*$ , and  $v_1 = (\mathbf{X}^T\mathbf{X})^{-\frac{1}{2}}v_1^*$ , where  $u_1^*$  and  $v_1^*$  are the leading left and right singular vectors in

$$(\mathbf{Y}^T\mathbf{Y})^{-\frac{1}{2}}(\mathbf{Y}^T\mathbf{X})(\mathbf{X}^T\mathbf{X})^{-\frac{1}{2}} = \mathbf{U}^*\mathbf{D}^*\mathbf{V}^{*T}. \quad (3.87)$$

Show that the entire sequence  $u_m, v_m, m = 1, \dots, \min(K, p)$  is also given by (3.87).

**Ex. 3.21** Show that the solution to the reduced-rank regression problem (3.68), with  $\Sigma$  estimated by  $\mathbf{Y}^T\mathbf{Y}/N$ , is given by (3.69). *Hint:* Transform

$\mathbf{Y}$  to  $\mathbf{Y}^* = \mathbf{Y}\Sigma^{-\frac{1}{2}}$ , and solved in terms of the canonical vectors  $u_m^*$ . Show that  $\mathbf{U}_m = \Sigma^{-\frac{1}{2}}\mathbf{U}_m^*$ , and a generalized inverse is  $\mathbf{U}_m^- = \mathbf{U}_m^{*T}\Sigma^{\frac{1}{2}}$ .

**Ex. 3.22** Show that the solution in Exercise 3.21 does not change if  $\Sigma$  is estimated by the more natural quantity  $(\mathbf{Y} - \mathbf{X}\hat{\mathbf{B}})^T(\mathbf{Y} - \mathbf{X}\hat{\mathbf{B}})/(N - pK)$ .

**Ex. 3.23** Consider a regression problem with all variables and response having mean zero and standard deviation one. Suppose also that each variable has identical absolute correlation with the response:

$$\frac{1}{N}|\langle \mathbf{x}_j, \mathbf{y} \rangle| = \lambda, \quad j = 1, \dots, p.$$

Let  $\hat{\beta}$  be the least-squares coefficient of  $\mathbf{y}$  on  $\mathbf{X}$ , and let  $\mathbf{u}(\alpha) = \alpha\mathbf{X}\hat{\beta}$  for  $\alpha \in [0, 1]$  be the vector that moves a fraction  $\alpha$  toward the least squares fit  $\mathbf{u}$ . Let  $RSS$  be the residual sum-of-squares from the full least squares fit.

(a) Show that

$$\frac{1}{N}|\langle \mathbf{x}_j, \mathbf{y} - \mathbf{u}(\alpha) \rangle| = (1 - \alpha)\lambda, \quad j = 1, \dots, p,$$

and hence the correlations of each  $\mathbf{x}_j$  with the residuals remain equal in magnitude as we progress toward  $\mathbf{u}$ .

(b) Show that these correlations are all equal to

$$\lambda(\alpha) = \frac{(1 - \alpha)}{\sqrt{(1 - \alpha)^2 + \frac{\alpha(2 - \alpha)}{N} \cdot RSS}} \cdot \lambda,$$

and hence they decrease monotonically to zero.

(c) Use these results to show that the LAR algorithm in Section 3.4.4 keeps the correlations tied and monotonically decreasing, as claimed in (3.55).

**Ex. 3.24** *LAR directions.* Using the notation around equation (3.55) on page 74, show that the LAR direction makes an equal angle with each of the predictors in  $\mathcal{A}_k$ .

**Ex. 3.25** *LAR look-ahead* (Efron et al., 2004, Sec. 2). Starting at the beginning of the  $k$ th step of the LAR algorithm, derive expressions to identify the next variable to enter the active set at step  $k + 1$ , and the value of  $\alpha$  at which this occurs (using the notation around equation (3.55) on page 74).

**Ex. 3.26** Forward stepwise regression enters the variable at each step that most reduces the residual sum-of-squares. LAR adjusts variables that have the most (absolute) correlation with the current residuals. Show that these two entry criteria are not necessarily the same. [Hint: let  $\mathbf{x}_{j,\mathcal{A}}$  be the  $j$ th

variable, linearly adjusted for all the variables currently in the model. Show that the first criterion amounts to identifying the  $j$  for which  $\text{Cor}(\mathbf{x}_{j \cdot \mathcal{A}}, \mathbf{r})$  is largest in magnitude.

**Ex. 3.27 Lasso and LAR:** Consider the lasso problem in Lagrange multiplier form: with  $L(\beta) = \frac{1}{2} \sum_i (y_i - \sum_j x_{ij} \beta_j)^2$ , we minimize

$$L(\beta) + \lambda \sum_j |\beta_j| \quad (3.88)$$

for fixed  $\lambda > 0$ .

- (a) Setting  $\beta_j = \beta_j^+ - \beta_j^-$  with  $\beta_j^+, \beta_j^- \geq 0$ , expression (3.88) becomes  $L(\beta) + \lambda \sum_j (\beta_j^+ + \beta_j^-)$ . Show that the Lagrange dual function is

$$L(\beta) + \lambda \sum_j (\beta_j^+ + \beta_j^-) - \sum_j \lambda_j^+ \beta_j^+ - \sum_j \lambda_j^- \beta_j^- \quad (3.89)$$

and the Karush–Kuhn–Tucker optimality conditions are

$$\begin{aligned} \nabla L(\beta)_j + \lambda - \lambda_j^+ &= 0 \\ -\nabla L(\beta)_j + \lambda - \lambda_j^- &= 0 \\ \lambda_j^+ \beta_j^+ &= 0 \\ \lambda_j^- \beta_j^- &= 0, \end{aligned}$$

along with the non-negativity constraints on the parameters and all the Lagrange multipliers.

- (b) Show that  $|\nabla L(\beta)_j| \leq \lambda \forall j$ , and that the KKT conditions imply one of the following three scenarios:

$$\begin{aligned} \lambda = 0 &\Rightarrow \nabla L(\beta)_j = 0 \forall j \\ \beta_j^+ > 0, \lambda > 0 &\Rightarrow \lambda_j^+ = 0, \nabla L(\beta)_j = -\lambda < 0, \beta_j^- = 0 \\ \beta_j^- > 0, \lambda > 0 &\Rightarrow \lambda_j^- = 0, \nabla L(\beta)_j = \lambda > 0, \beta_j^+ = 0. \end{aligned}$$

Hence show that for any “active” predictor having  $\beta_j \neq 0$ , we must have  $\nabla L(\beta)_j = -\lambda$  if  $\beta_j > 0$ , and  $\nabla L(\beta)_j = \lambda$  if  $\beta_j < 0$ . Assuming the predictors are standardized, relate  $\lambda$  to the correlation between the  $j$ th predictor and the current residuals.

- (c) Suppose that the set of active predictors is unchanged for  $\lambda_0 \geq \lambda \geq \lambda_1$ . Show that there is a vector  $\gamma_0$  such that

$$\hat{\beta}(\lambda) = \hat{\beta}(\lambda_0) - (\lambda - \lambda_0)\gamma_0 \quad (3.90)$$

Thus the lasso solution path is linear as  $\lambda$  ranges from  $\lambda_0$  to  $\lambda_1$  (Efron et al., 2004; Rosset and Zhu, 2007).

**Ex. 3.28** Suppose for a given  $t$  in (3.51), the fitted lasso coefficient for variable  $X_j$  is  $\hat{\beta}_j = a$ . Suppose we augment our set of variables with an identical copy  $X_j^* = X_j$ . Characterize the effect of this exact collinearity by describing the set of solutions for  $\hat{\beta}_j$  and  $\hat{\beta}_j^*$ , using the same value of  $t$ .

**Ex. 3.29** Suppose we run a ridge regression with parameter  $\lambda$  on a single variable  $X$ , and get coefficient  $a$ . We now include an exact copy  $X^* = X$ , and refit our ridge regression. Show that both coefficients are identical, and derive their value. Show in general that if  $m$  copies of a variable  $X_j$  are included in a ridge regression, their coefficients are all the same.

**Ex. 3.30** Consider the elastic-net optimization problem:

$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda[\alpha\|\beta\|_2^2 + (1 - \alpha)\|\beta\|_1]. \quad (3.91)$$

Show how one can turn this into a lasso problem, using an augmented version of  $\mathbf{X}$  and  $\mathbf{y}$ .

