STATISTICAL MODELING AND DATA ANALYSIS (II)

Yang Feng

http://www.stat.columbia.edu/~yangfeng

Outline

Statistical Learning

1. Supervised Learning:
   Given a set of N training examples of the form $\{(x_1, y_1), ..., (x_N, y_N)\}$ such that x_i is the feature vector of the i-th example and y_i is its label (i.e., class), a learning algorithm seeks a function $g : X \to Y$, where X is the input space and Y is the output space.
   - Regression
   - Classification
2. Unsupervised Learning:
   Given a set of N training examples of the form $\{x_1, ..., x_N\}$ such that x_i is the feature vector of the i-th example, a learning algorithm seeks to recover the structure of X, where X is the feature space.
   - Clustering
   - Density estimation
   - Dimensional reduction

Examples

1. Example 1: Email Spam
2. Example 2: Prostate Cancer
3. Example 3: Handwritten Digit Recognition
4. Example 4: DNA Expression Microarrays

Table: 4601 email messages, 57 features. Average percentage of words or characters in an email message equal to the indicated word or character.

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

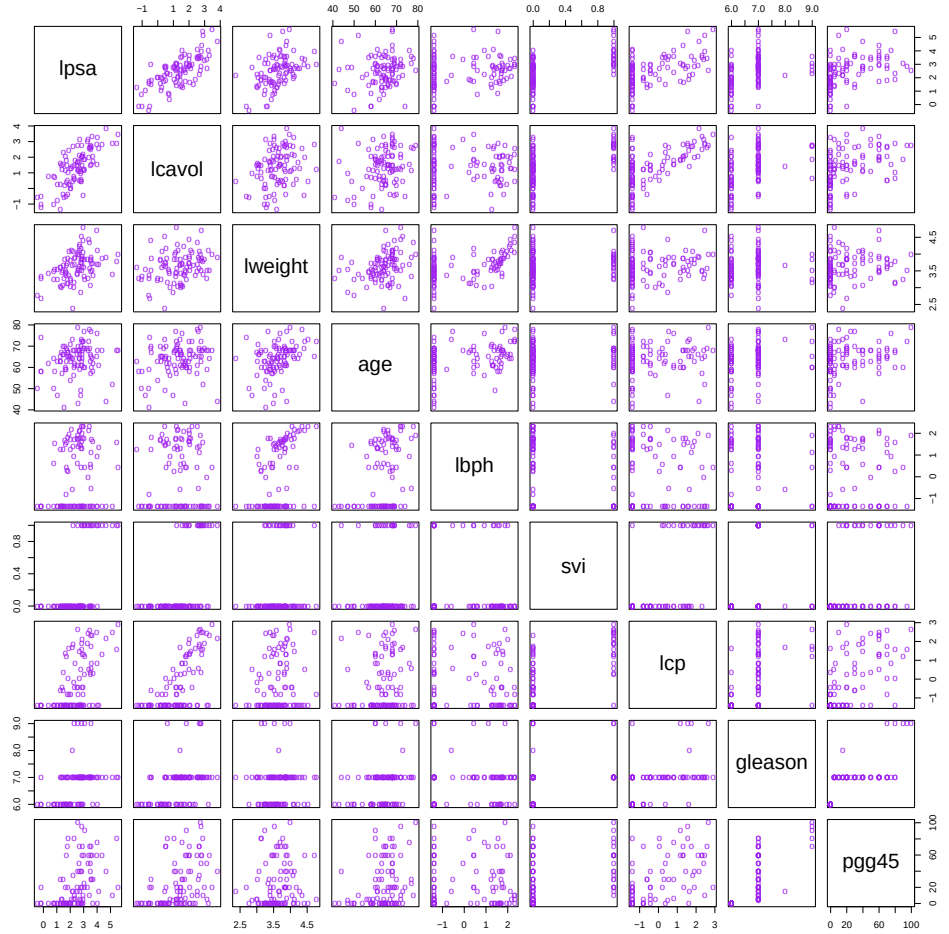


FIGURE 1.1. *Scatterplot matrix of the prostate cancer data. The first row shows the response against each of the predictors in turn. Two of the predictors,* `svi` *and* `gleason`*, are categorical.*

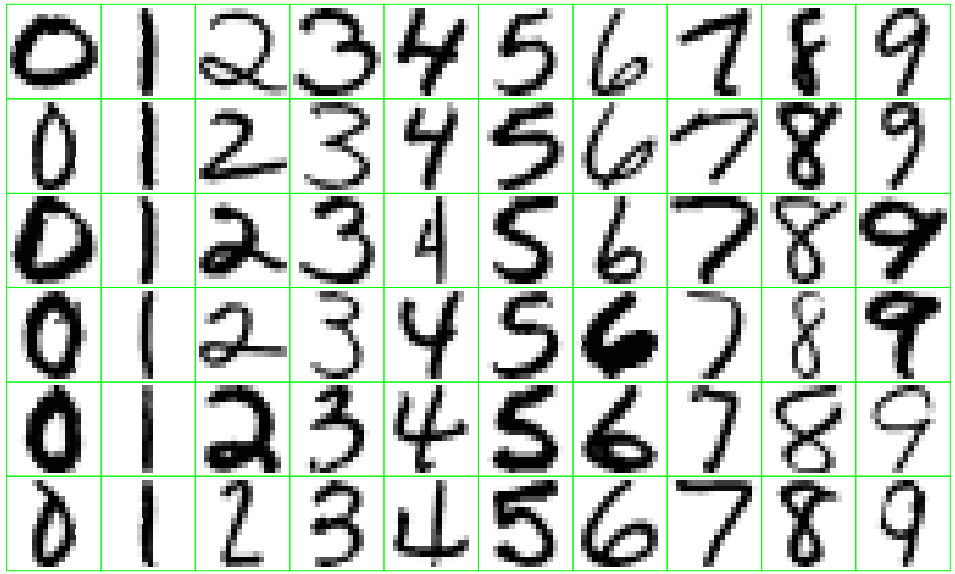


FIGURE 1.2. *Examples of handwritten digits from U.S. postal envelopes.*

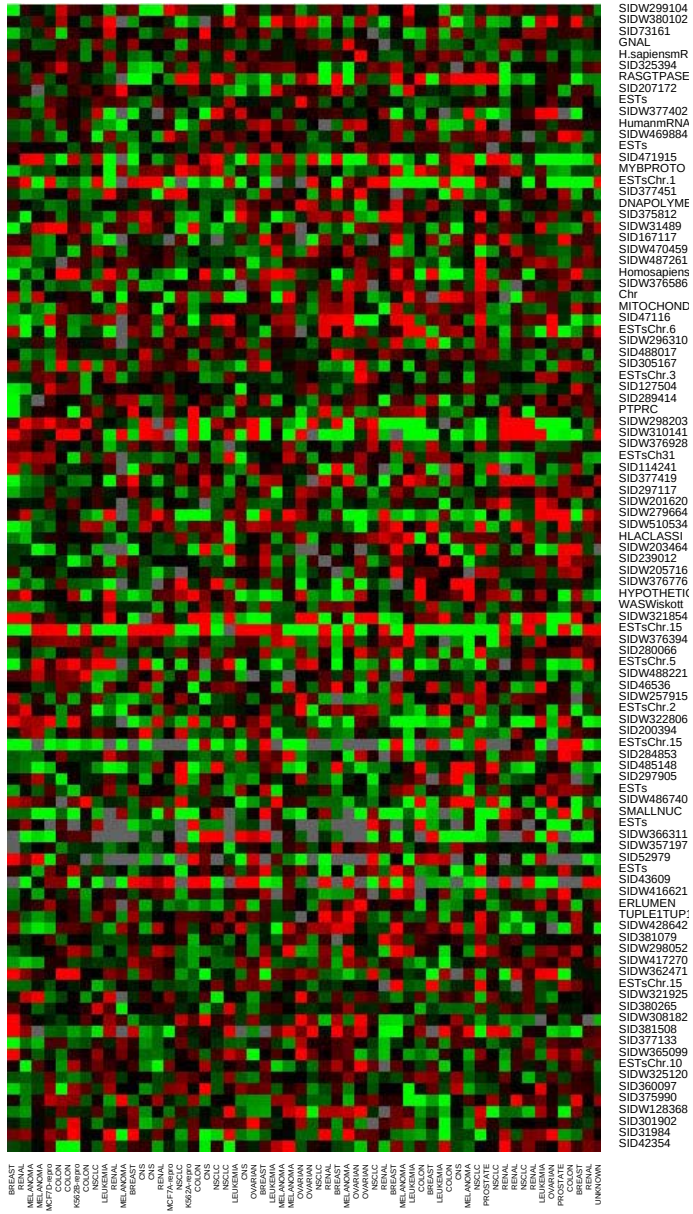


FIGURE 1.3. *DNA microarray data: expression ma-*

Outline

Terminology

- inputs, predictors, independent variables
- outputs, response, dependent variables

- Variable types: quantitative, qualitative, order categorical.

Suppose we have available a set of measurements (x_i, y_i) or (x_i, g_i), $i = 1, ..., N$, known as the training data, with which to construct our prediction rule.

Classification via linear regression

Assume $Y = X\beta + \epsilon$, we have $\hat{\beta} = (X^T X)^{-1} X^T Y$.

- As a result, $\hat{y}_i = x_i^T \hat{\beta}$.
- If $y_i = 0$ or 1, then we would want $\hat{y}_i$ to be 0 and 1.
- Convert $\hat{y}_i$ to $\hat{g}_i$ by $g_i = 1$ if $\hat{y}_i > 0.5$, otherwise, $g_i = 0$.
- *Decision boundary:* $x^T \hat{\beta} = 0.5$.

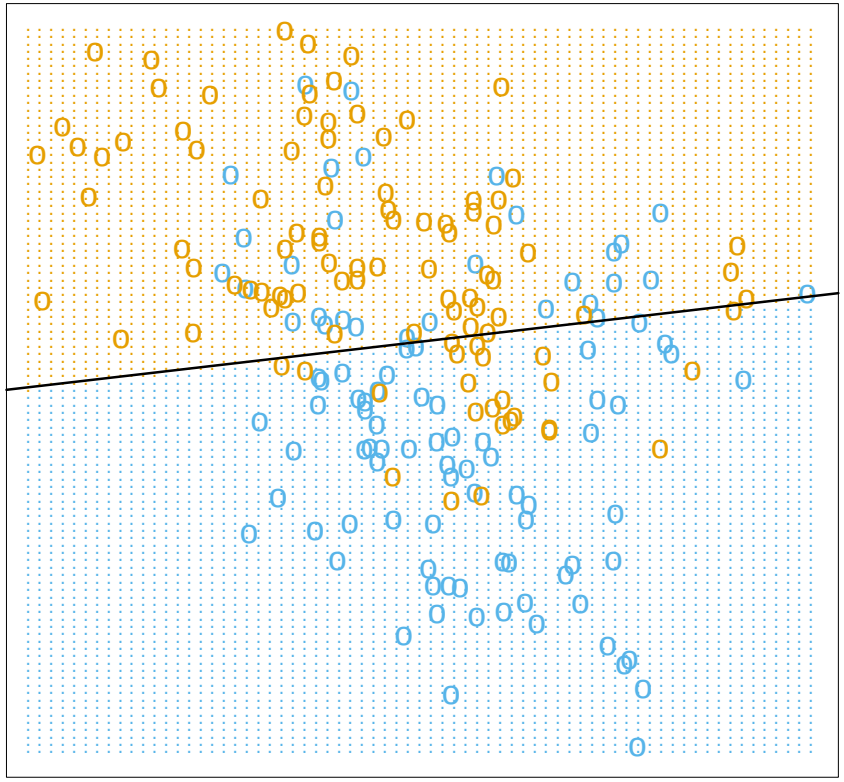


FIGURE 2.1. *A classification example in two dimensions. The classes are coded as a binary variable (*BLUE $= 0$, ORANGE $= 1$*), and then fit by linear regression. The line is the decision boundary defined by* $x^T\hat{\beta} = 0.5$. *The orange shaded region denotes that part of input space classified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

Is it a good classifier?

- 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.

Nearest-Neighbor Methods

- Use the average of neighboring values to provide predictions.
- $$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i,$$

  where $N_k(x)$ is the neighborhood of x defined by the k closest points x_i in the training sample.

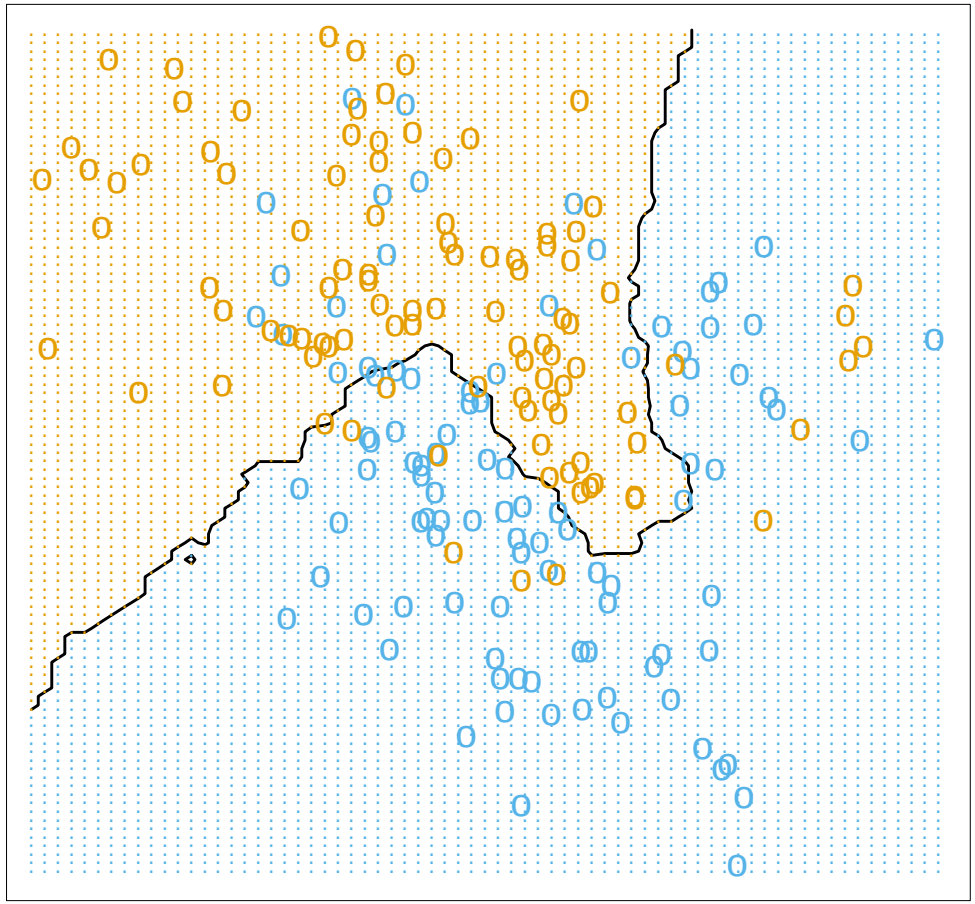


FIGURE 2.2. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$) *and then fit by* 15*-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the* 15*-nearest neighbors.*

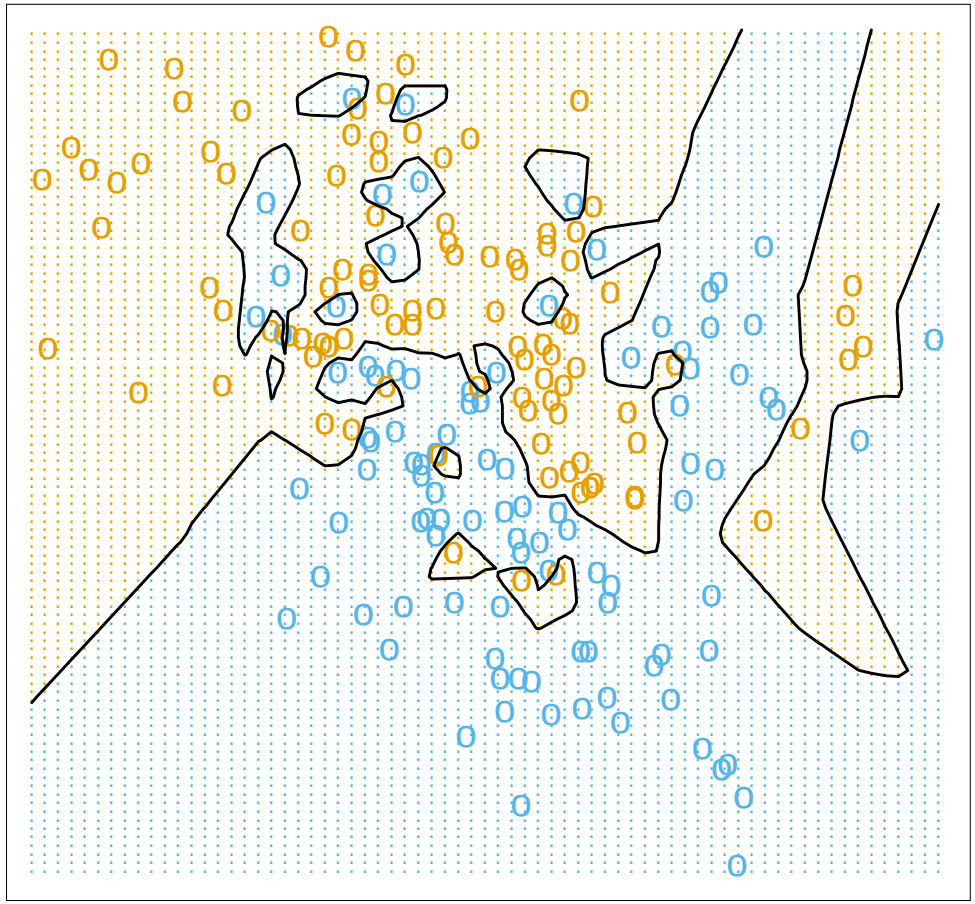


FIGURE 2.3. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$)*, and then predicted by* 1*-nearest-neighbor classification.*

Comparison of least square to nearest neighbor

- least square: low variance, potentially high bias.
- nearest neighbor: high variance, low bias.

DGP: A mixture of Gaussian clusters for each class.

1. generated 10 means m_k from a bivariate Gaussian distribution $N((1,0)^T, I_2)$ and labeled this class BLUE
2. generated 10 means I_k from a bivariate Gaussian distribution $N((0,1)^T, I_2)$ and labeled this class ORANGE
3. For each class we generated 100 observations as follows: for each observation, we picked an m_k/I_k at random with probability $1/10$, then generated a $N(m_k, I_2/5)$.

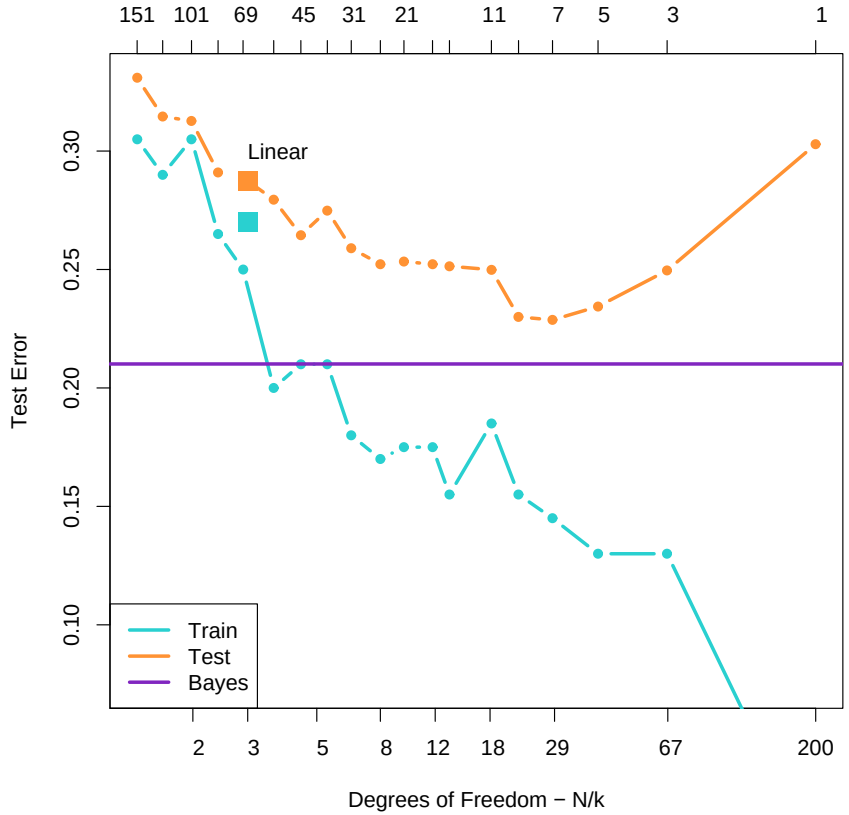


FIGURE 2.4. *Misclassification curves for the simulation example used in Figures 2.1, 2.2 and 2.3. A single training sample of size* 200 *was used, and a test sample of size* $10,000$*. The orange curves are test and the blue are training error for k-nearest-neighbor classification. The results for linear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

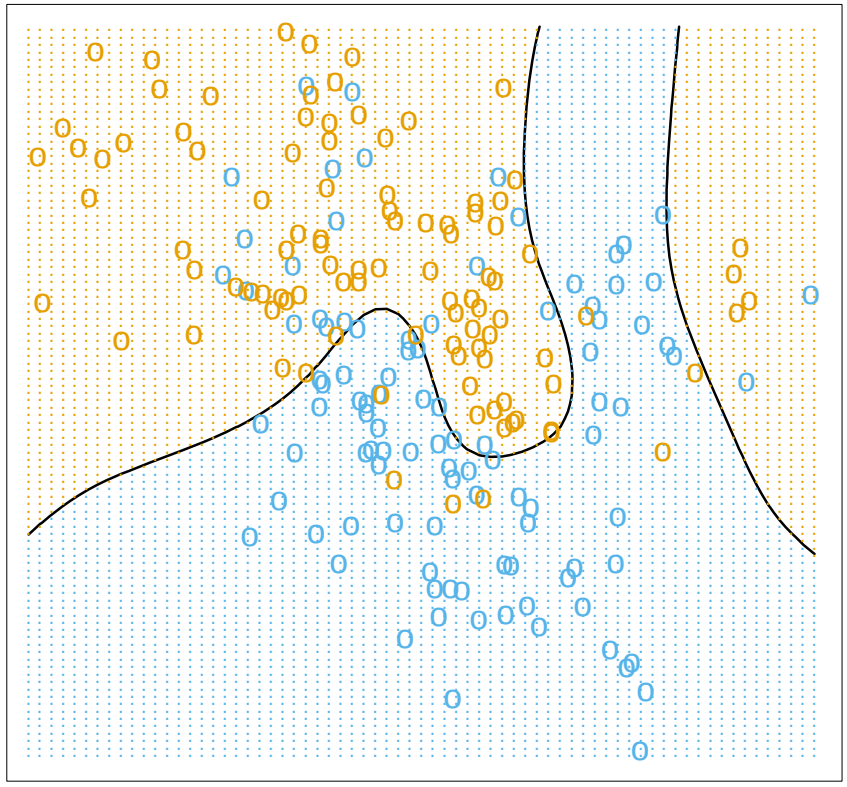


FIGURE 2.5. *The optimal Bayes decision boundary for the simulation example of Figures 2.1, 2.2 and 2.3. Since the generating density is known for each class, this boundary can be calculated exactly (Exercise 2.2).*

Statistical Decision Theory

- $X \in R^p$ denote a real valued random input vector
- $Y \in R$ denote a real valued random output vector
- We seek a function $f(X)$ for predicting Y given values of the input X.
- *Loss function*: $L(Y, f(X))$.
- Common loss function: squared error loss $L(Y, f(X)) = (Y - f(X))^2$.
- expected (squared) prediction error (EPE):

$$\text{EPE}(f) = \int [y - f(x)]^2 Pr(dx, dy) \tag{1}$$

$$= E_X E_{Y|X}([Y - f(X)]^2 | X). \tag{2}$$

Statistical Decision Theory

- Minimize pointwise: $f(x) = \arg\min_c E_{Y|X}([Y - c]^2|X = x)$.
- Solution: $f(x) = E(Y|X = x)$, *regression function*.

Categorical Response

- K possible categories for G.
- Loss function can be represented by a $K \times K$ matrix $\mathbf{L}$, where $\mathbf{L}$ is zero on diagonal and nonnegative elsewhere. $L(k, l)$ is the loss for classifying an observation belonging to class $\mathcal{G}_k$ as $\mathcal{G}_l$.
- Usually, we use zero-one loss function, where $\mathbf{L} = \mathbf{J} - \mathbf{I}$.
- Then, we have the expected prediction error

$$\text{EPE} = E[L(G, \hat{G}(X))] \tag{3}$$

$$= E_X \sum_{k=1}^{K} L[\mathcal{G}_k, \hat{G}(X)] Pr(\mathcal{G}_k|X) \tag{4}$$

Categorical Response

- Pointwise minimization

$$\hat{G}(x) = \arg\min_{g \in \mathcal{G}} \sum_{k=1}^{K} L(\mathcal{G}_k, g) Pr(\mathcal{G}_k | X = x). \tag{5}$$

- With 0-1 loss

$$\hat{G}(x) = \arg\min_{g \in \mathcal{G}} [1 - Pr(g | X = x)] \tag{6}$$

or

$$\hat{G}(x) = \mathcal{G}_k \text{ if } Pr(\mathcal{G}_k | X = x) = \max_{g \in \mathcal{G}} Pr(g | X = x). \tag{7}$$

- *Bayes classifier*: most probable class.
- *Bayes rate*: error rate of Bayes classifier.

Curse of dimensionality

- Consider the nearest-neighbor procedure for inputs uniformly distributed in a p-dimensional unit hypercube.
- Consider neighborhood about a target point to capture a fraction r of the observations.
- The expected edge length will be $e_p(r) = r^{1/p}$.
- $e_{10}(0.01) = 0.63$ and $e_{10}(0.1) = 0.8$.

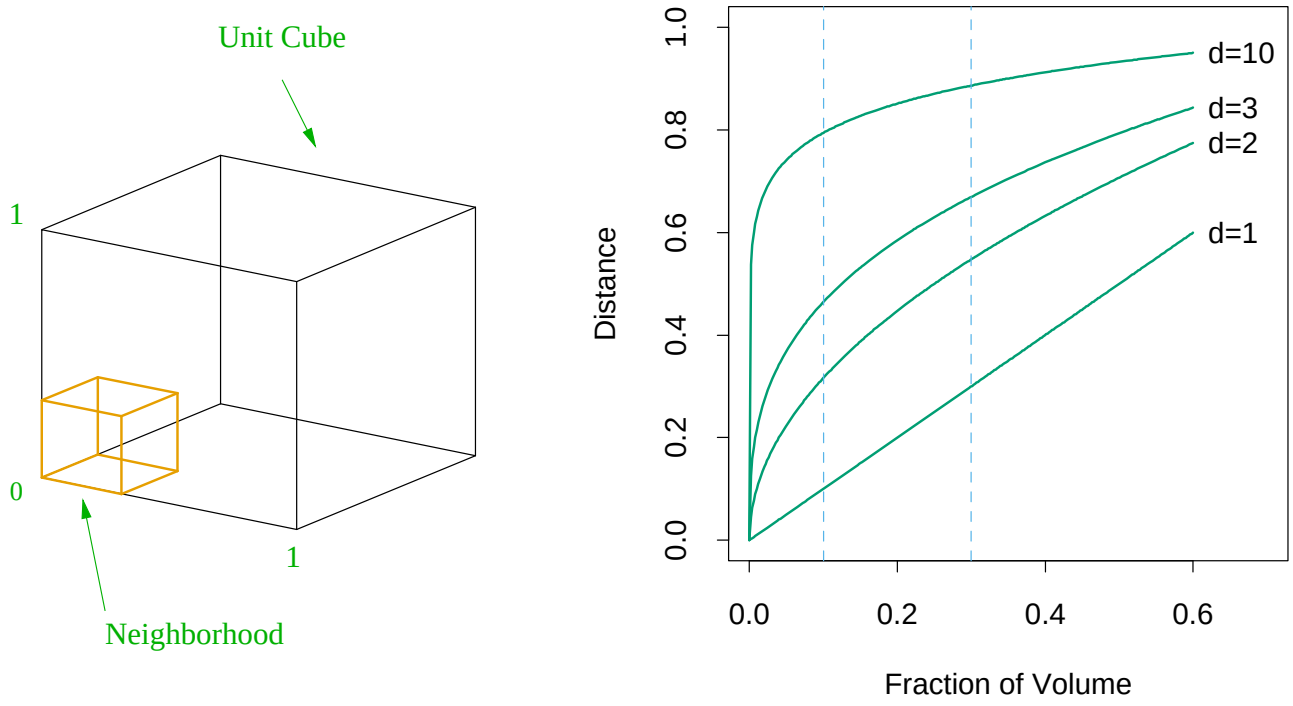


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover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

Another example

- 1000 training examples $x_i \sim Unif([-1,1]^p)$.
- $Y = f(X) = \exp\{-8\|X\|^2\}$.
- Use 1-nearest-neighbor to predict y_0 at $x_0 = 0$.
- Bias-variance decomposition: $MSE(x_0) = Var(\hat{y}_0) + Bias^2(\hat{y}_0)$.
- By $p = 10$, for more than 99% of the samples, the nearest neighbor is a distance greater than 0.5 from the origin.

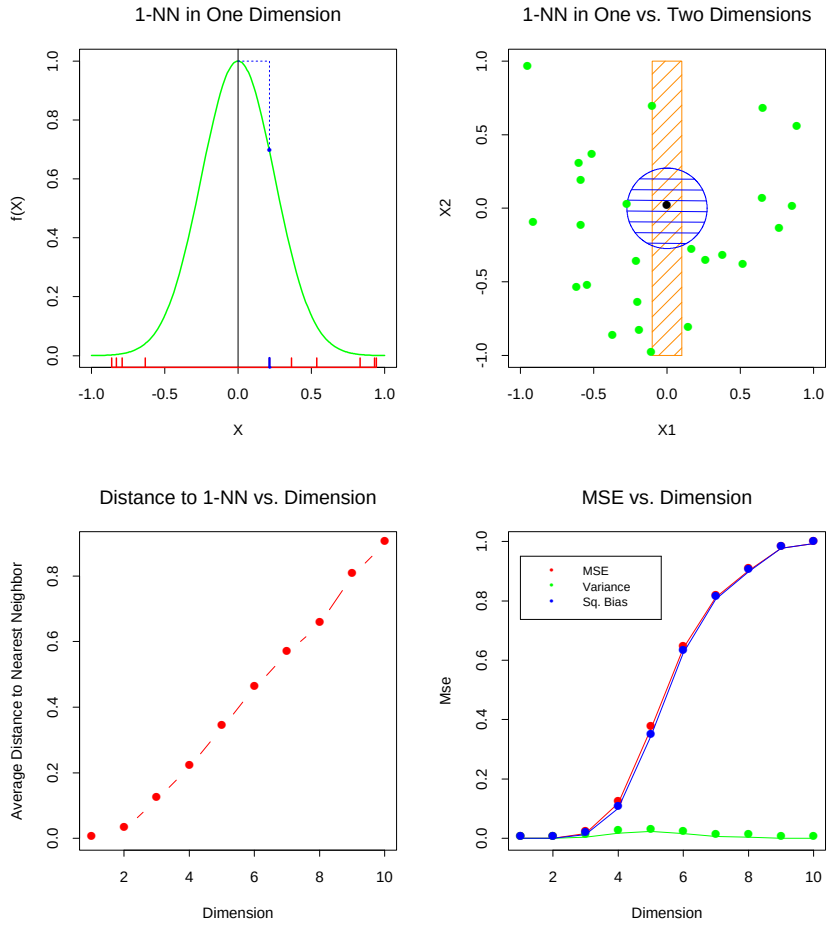


FIGURE 2.7. *A simulation example, demonstrating the curse of dimensionality and its effect on MSE, bias and variance. The input features are uniformly distributed in* $[-1, 1]^p$ *for* $p = 1, \ldots, 10$ *The top left panel shows the target function (no noise) in* $\mathbb{R}$*:* $f(X) = e^{-8||X||^2}$*, and demonstrates the error that* 1*-nearest neighbor makes in estimating* $f(0)$*.*

Test error

- DGP: $Y = f(X) + \epsilon$ with $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma^2$.
- Use k-nearest neighbor regression fit $\hat{f}_k$.
- The expected prediction error at x_0, known as *test* or *generalization* error

$$\begin{aligned} \text{EPE}_k(x_0) &= E[(Y - \hat{f}_k(x_0))^2 | X = x_0] && (8) \\ &= \sigma^2 + [Bias^2(\hat{f}_k(x_0)) + Var(\hat{f}_k(x_0))] && (9) \\ &= \sigma^2 + [f(x_0) - \frac{1}{k}\sum_{l=1}^{k} f(x_{(l)})]^2 + \frac{\sigma^2}{k}, && (10) \end{aligned}$$

The subscripts in parentheses (l) indicate the sequence of nearest neighbors to x_0.

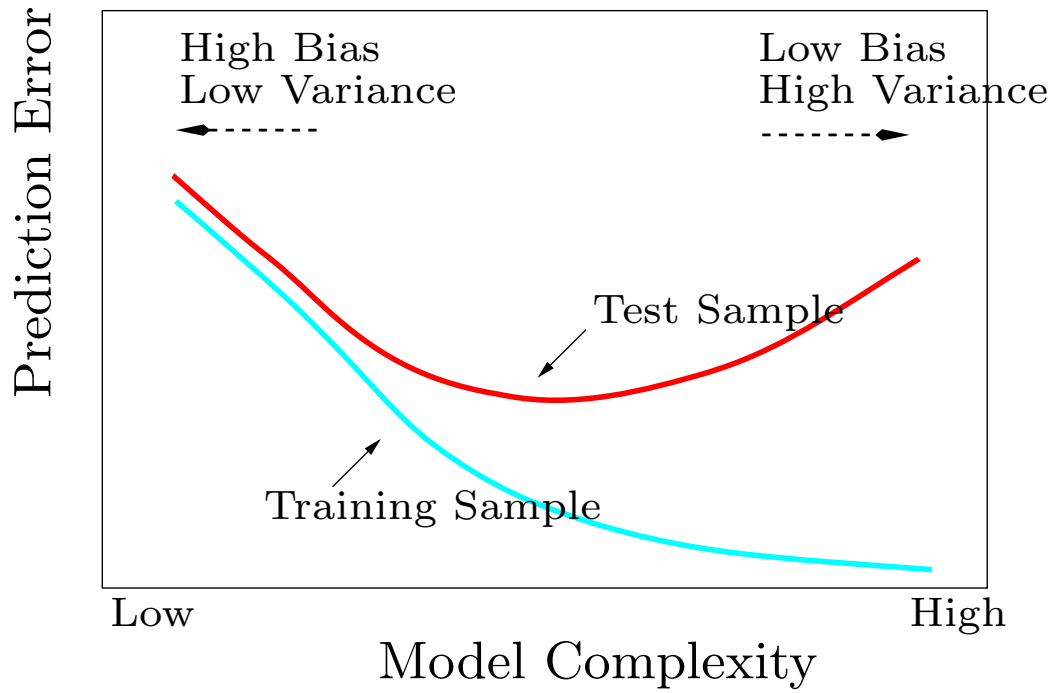


FIGURE 2.11. *Test and training error as a function of model complexity.*

Outline

Least Squares

Suppose we have input $X = (X_1, \cdots, X_p)^T$ and output Y. Assume

$$f(X) = E(Y|X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j. \tag{11}$$

Suppose we have training data $(x_1, y_1), \cdots, (x_N, y_N)$, least square aims to minimize residual sum of squares

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j)^2. \tag{12}$$

Denote $\mathbf{X}$ be the $N \times (p+1)$ *design matrix*. We have $\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$.

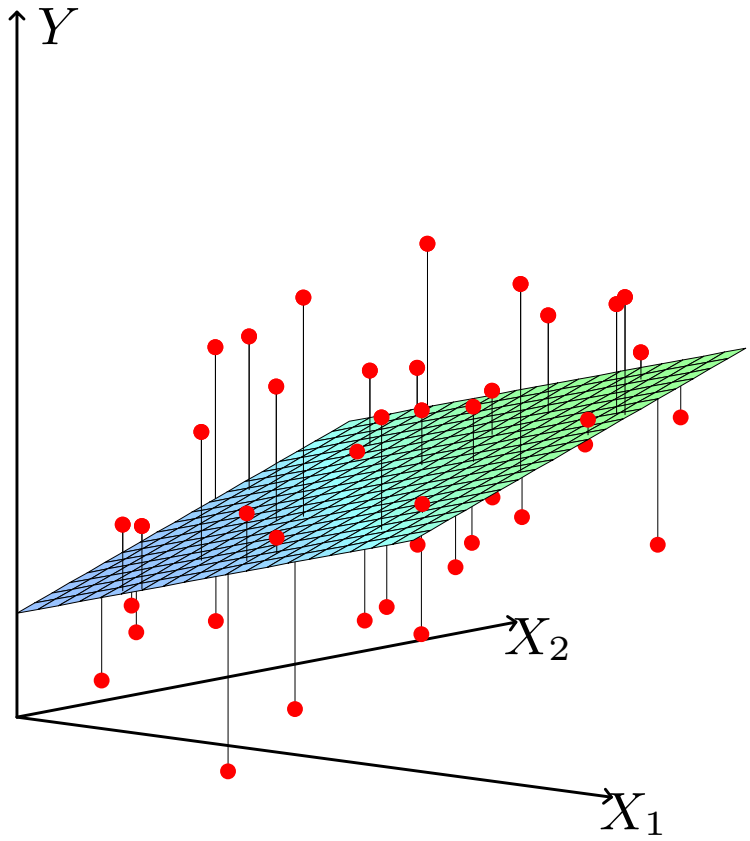


FIGURE 3.1. *Linear least squares fitting with* $X \in \mathbb{R}^2$. *We seek the linear function of* X *that minimizes the sum of squared residuals from* Y.

Least Squares

- Fitted value: $\hat{\mathbf{y}} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \equiv \mathbf{H}\mathbf{y}$.
- $\mathbf{H}$: projection matrix.
- Variance covariance matrix: $Var(\hat{\beta}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$.
- Mean squared error: $\hat{\sigma}^2 = \frac{1}{N-p-1}(\hat{\mathbf{y}} - \mathbf{y})^T(\hat{\mathbf{y}} - \mathbf{y})$.

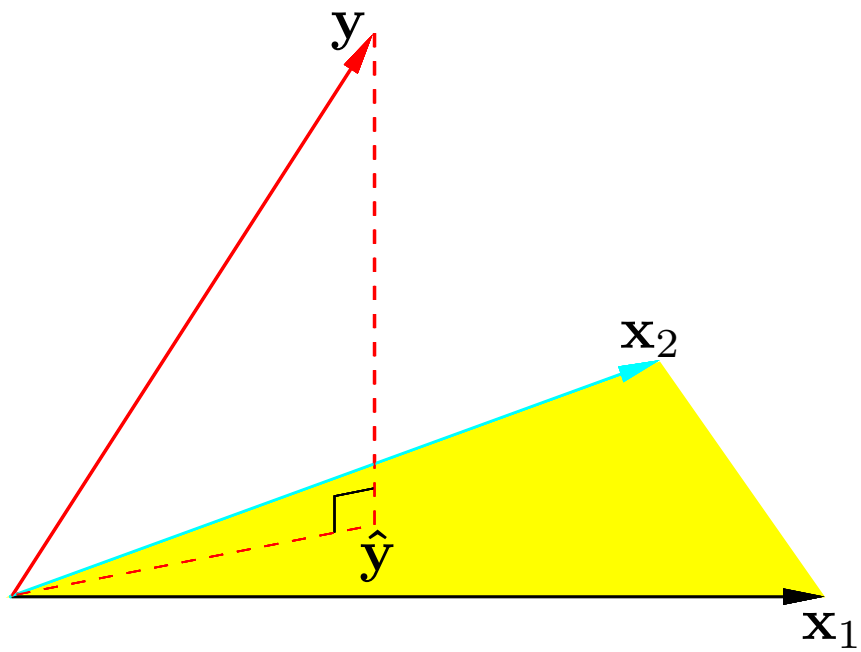


FIGURE 3.2. *The N-dimensional geometry of least squares regression with two predictors. The outcome vector* $\mathbf{y}$ *is orthogonally projected onto the hyperplane spanned by the input vectors* $\mathbf{x}_1$ *and* $\mathbf{x}_2$*. The projection* $\hat{\mathbf{y}}$ *represents the vector of the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

Inference

Now assume $\epsilon_i \overset{i.i.d.}{\sim} N(0, \sigma^2)$.

- $\hat{\boldsymbol{\beta}} \sim N(\boldsymbol{\beta}, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1})$.
- $(N - p - 1)\hat{\sigma}^2 \sim \sigma^2\chi^2_{N-p-1}$.

For testing $H_0 : \beta_j = 0$ v.s. $H_1 : \beta_j \neq 0$. Use t-statistic

$$t_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{v_j}}, \tag{13}$$

where v_j is the j-th diagonal element of $(\mathbf{X}^T\mathbf{X})^{-1}$.

- Under H_0, $t_j \sim t_{N-p-1}$.
- $1 - \alpha$ confidence interval for β_j:

$$(\hat{\beta}_j - t_{1-\alpha/2,N-p-1}v_j^{1/2}\hat{\sigma}, \hat{\beta}_j + t_{1-\alpha/2,N-p-1}v_j^{1/2}\hat{\sigma}) \tag{14}$$

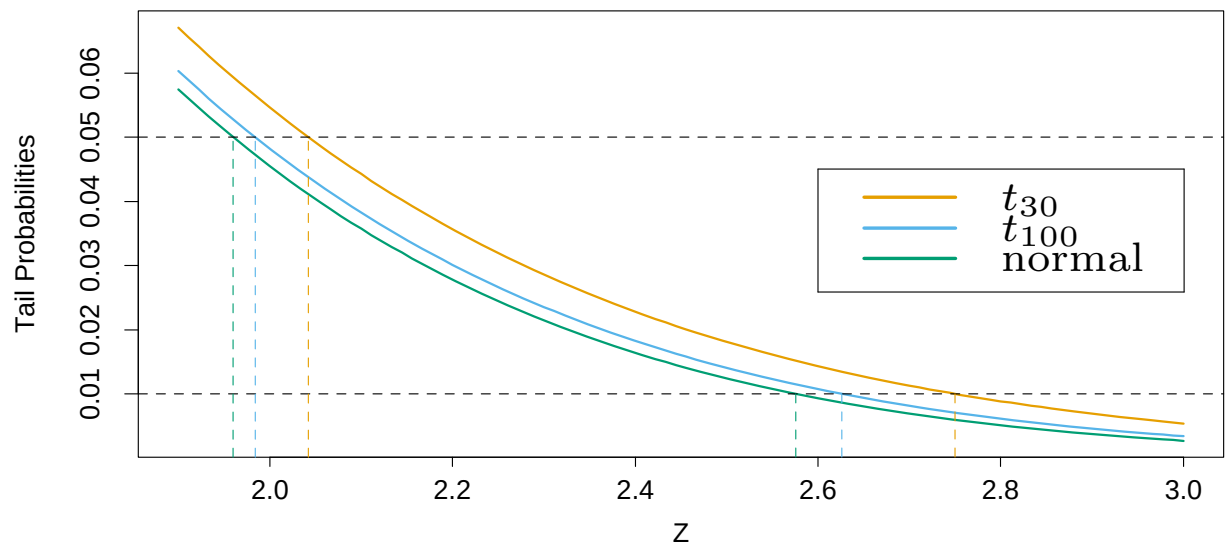


FIGURE 3.3. *The tail probabilities* $\Pr(|Z| > z)$ *for three distributions,* t_{30}*,* t_{100} *and standard normal. Shown are the appropriate quantiles for testing significance at the* $p = 0.05$ *and* 0.01 *levels. The difference between* t *and the standard normal becomes negligible for* N *bigger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

Gauss-Markov Theorem

Theorem

For any parameter $\theta = \mathbf{a}^T\beta$, the best linear unbiased estimator (BLUE) is $\hat{\theta} = \mathbf{a}^T\hat{\beta}$.

For any linear estimator $\tilde{\theta} = \mathbf{c}^T\mathbf{y}$ with $E\tilde{\theta} = \theta$, we have $var(\hat{\theta}) \leq var(\tilde{\theta})$.

Multiple Outputs

- Suppose we have K outputs $Y_1, \cdots, Y_K$.
- $Y_k = \beta_{0k} + \sum_{j=1}^{p} X_j \beta_{jk} + \epsilon_k = f_k(X) + \epsilon_k$.
- With N training cases, we have

$$\mathbf{Y}_{N\times K} = \mathbf{X}_{N\times(p+1)}\mathbf{B}_{(p+1)\times K} + \mathbf{E}_{N\times K}. \tag{15}$$

- New loss function

$$RSS(\mathbf{B}) = tr[(\mathbf{Y} - \mathbf{XB})^T(\mathbf{Y} - \mathbf{XB})]. \tag{16}$$

- LS estimate $\hat{\mathbf{B}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$.
- Define $\boldsymbol{\epsilon} = (\epsilon_1, \cdots, \epsilon_K)^T$.
- If $Cov(\boldsymbol{\epsilon}) = \boldsymbol{\Sigma}$, we can minimize

$$RSS(\mathbf{B}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N}(y_i - f(x_i))^T\boldsymbol{\Sigma}^{-1}(y_i - f(x_i)). \tag{17}$$

Subset selection

- *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.
- *interpretation*: With a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects.

Model selection criteria

Current model M with no. of predictors s. SSR, RSS are the corresponding Sum of Squared Regression and Residual Sum of Squares for M.

- Coefficient of determination

$$R^2(M) = \frac{SSR}{SSTO} \tag{18}$$

- Adjusted R^2

$$R_a^2(M) = 1 - \frac{RSS/(n-s-1)}{SSTO/(n-1)} \tag{19}$$

- Akaike's information criterion (AIC) (Akaike, 1973)

$$AIC(M) = n\log(RSS/n) + 2(s+1) \tag{20}$$

- Bayesian information criterion (BIC) (Schwarz, 1978)

$$BIC(M) = n\log(RSS/n) + \log(n)(s+1) \tag{21}$$

Methods

- Best subset selection.
- Forward and backward stepwise selection.
- Forward stagewise regression.
  1. Set intercept equal to $\bar{y}$, and centered all predictors with coefficients initially all 0.
  2. Identifies the variable most correlated with the current residual.
  3. Computes the simple linear regression coefficient of the residual on this chosen variable, and then adds it to the current coefficient for that variable.
  4. Repeat Steps 2-3 until none of the variables have correlations with the residual.

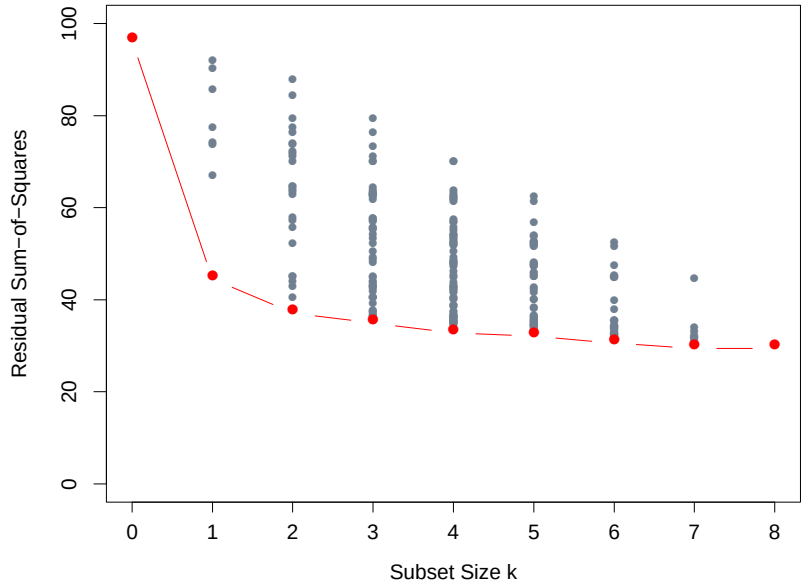


FIGURE 3.5. *All possible subset models for the prostate cancer example. At each subset size is shown the residual sum-of-squares for each model of that size.*

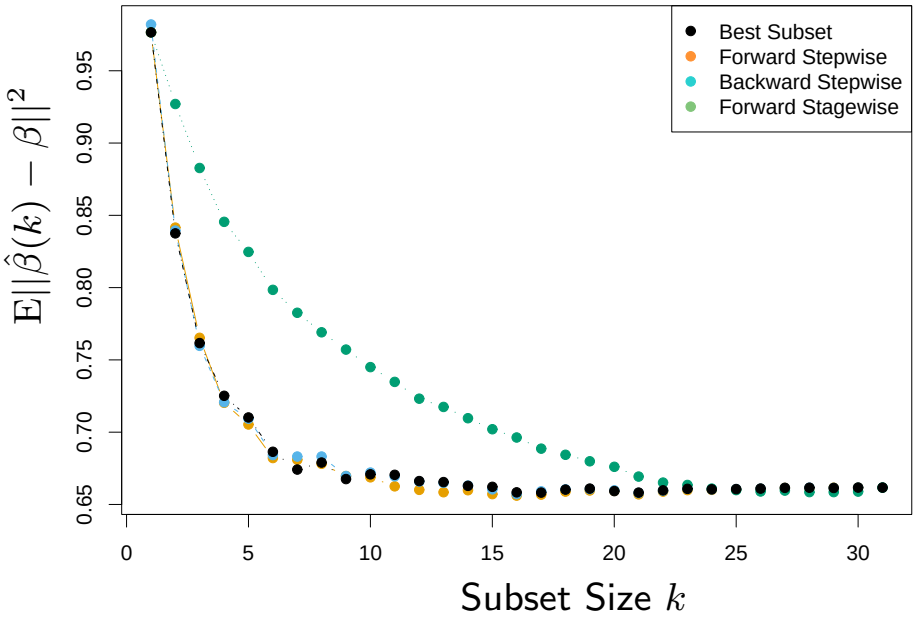


FIGURE 3.6. *Comparison of four subset-selection techniques on a simulated linear regression problem* $Y = X^T\beta + \varepsilon$. *There are* $N = 300$ *observations on* $p = 31$ *standard Gaussian variables, with pairwise correlations all equal to* 0.85. *For* 10 *of the variables, the coefficients 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* β.

Shrinkage methods

- Drawback of subset selection: discontinuous, high variability.
- Ridge Regression
- Lasso

Ridge Regression

First center and standardize each predictor. The intercept estimate would be $\bar{y}$. X would be $N \times p$.

Formulation

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta}\{(\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta)+\lambda\|\beta\|_2^2\}. \tag{22}$$

$$= (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}. \tag{23}$$

Another Formulation

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta}\{(\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta)\}. \tag{24}$$

$$\text{subject to } \|\beta\|_2^2 \le t. \tag{25}$$

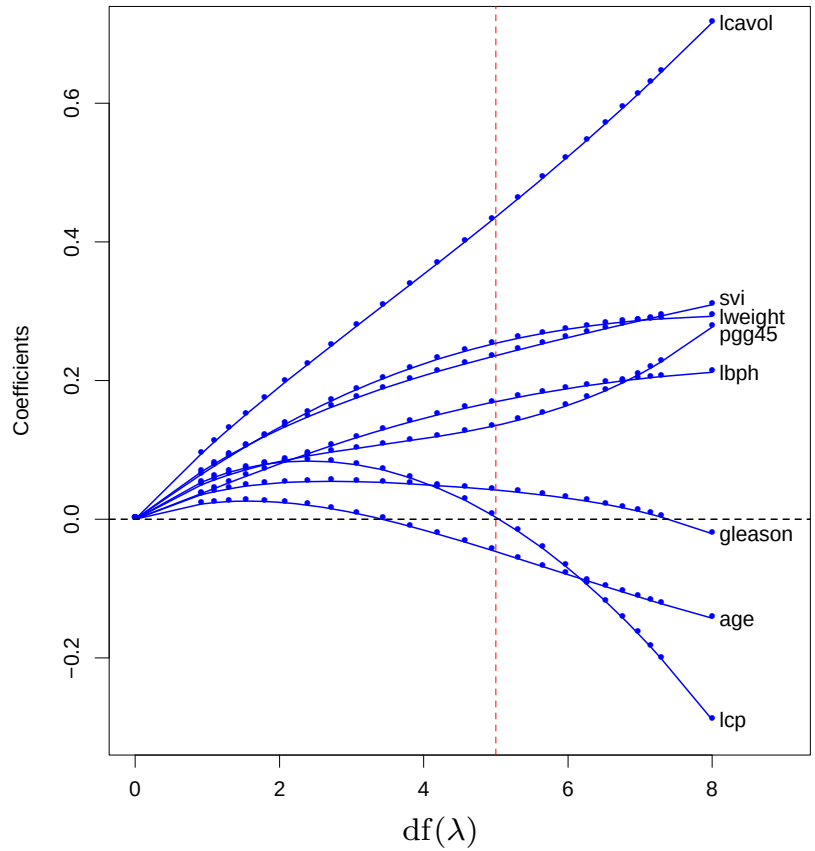


FIGURE 3.8. *Profiles of ridge coefficients for the prostate cancer example, as the tuning parameter* λ *is varied. Coefficients are plotted versus* $\mathrm{df}(\lambda)$*, the effective degrees of freedom. A vertical line is drawn at* $\mathrm{df} = 5.0$*, the value chosen by cross-validation.*

Insights of Ridge Regression

- Perform SVD on $\mathbf{X}$: $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. $\mathbf{D} = \text{diag}(d_1, \cdots, d_p)$.
- Using SVD

$$\mathbf{X}\hat{\beta}^{lm} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{26}$$
$$= \mathbf{U}\mathbf{U}^T\mathbf{y} \tag{27}$$

$$\mathbf{X}\hat{\beta}^{ridge} = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{28}$$
$$= \mathbf{U}\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{y} \tag{29}$$
$$= \sum_{j=1}^{p} \mathbf{u}_j \frac{d_j^2}{d_j^2 + \lambda}\mathbf{u}_j^T\mathbf{y}. \tag{30}$$

Connections to Principle Components

- Consider the Eigen decomposition of $\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{D}^2\mathbf{V}^T$.
- Eigenvectors $\mathbf{v}_j$ (Columns of $\mathbf{V}$) are *principle components*.
- The first principle component $\mathbf{v}_1$ leads to the largest sample variance of $\mathbf{z}_1 = \mathbf{X}\mathbf{v}_1$. $Var(\mathbf{z}_1) = d_1^2/N$.
- As a result, ridge regression tends to shrink the small eigenvalues more.

Degrees of freedom

- For ridge regression, define

$$df(\lambda) = tr[\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T] \tag{31}$$

$$= tr(\mathbf{H}_\lambda) \tag{32}$$

$$= \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda}. \tag{33}$$

- If $\lambda = 0$, $df(\lambda) = p$.
- If $\lambda = \infty$, $df(\lambda) = 0$.

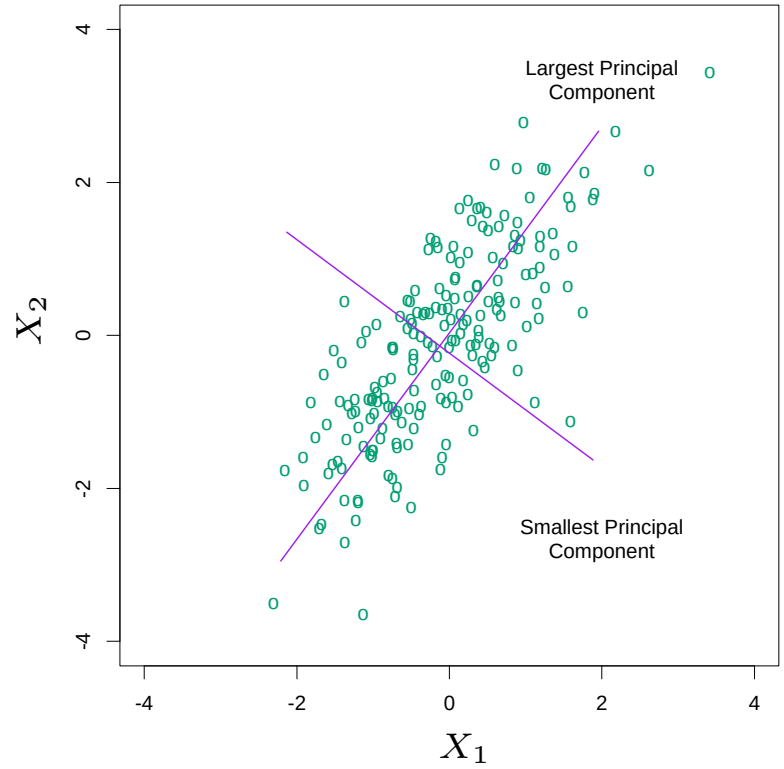


FIGURE 3.9. *Principal components of some input data points. The largest principal component is the direction that maximizes the variance of the projected data, and the smallest principal component minimizes that variance. Ridge regression projects* $\mathbf{y}$ *onto these components, and then shrinks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

Lasso (Tibshirani, 1996, JRSSB; Chen and Donoho, 1994)

Google Scholar Citation: 22K+ as of Jan 2018.

Formulation

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta}\{(\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta)\}. \tag{34}$$

$$\text{subject to } \|\beta\|_1 \leq t. \tag{35}$$

Lagrangian form

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta}\{\frac{1}{2}(\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta) + \lambda\|\beta\|_1\}. \tag{36}$$

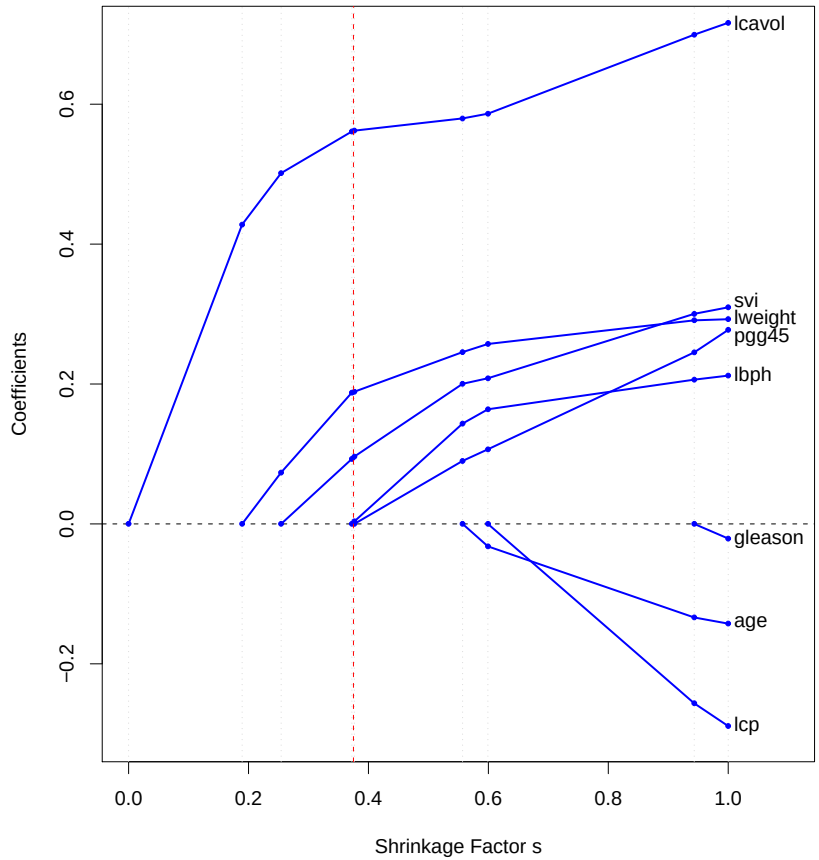
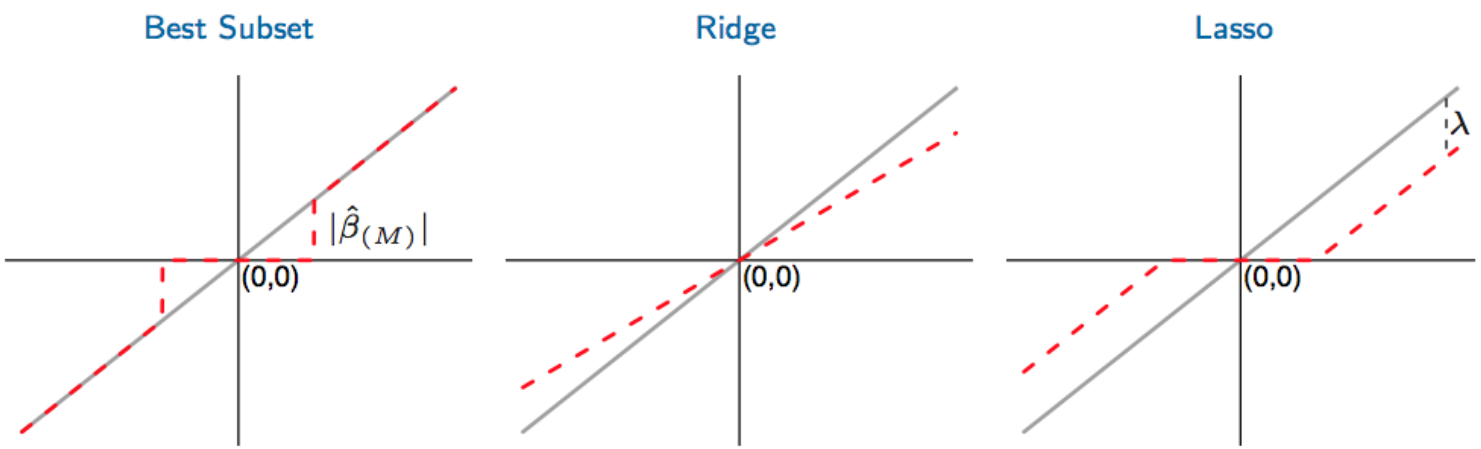


FIGURE 3.10. *Profiles of lasso coefficients, as the tuning parameter t is varied. Coefficients are plotted versus $s = t/\sum_1^p |\hat{\beta}_j|$. A vertical line is drawn at $s = 0.36$, the value chosen by cross-validation. Compare Figure 3.8 on page 9; the lasso profiles hit zero, while those for ridge do not. The profiles are piece-wise linear, and so are computed only at the points displayed;*

Comparisons of best subset, ridge and lasso

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

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



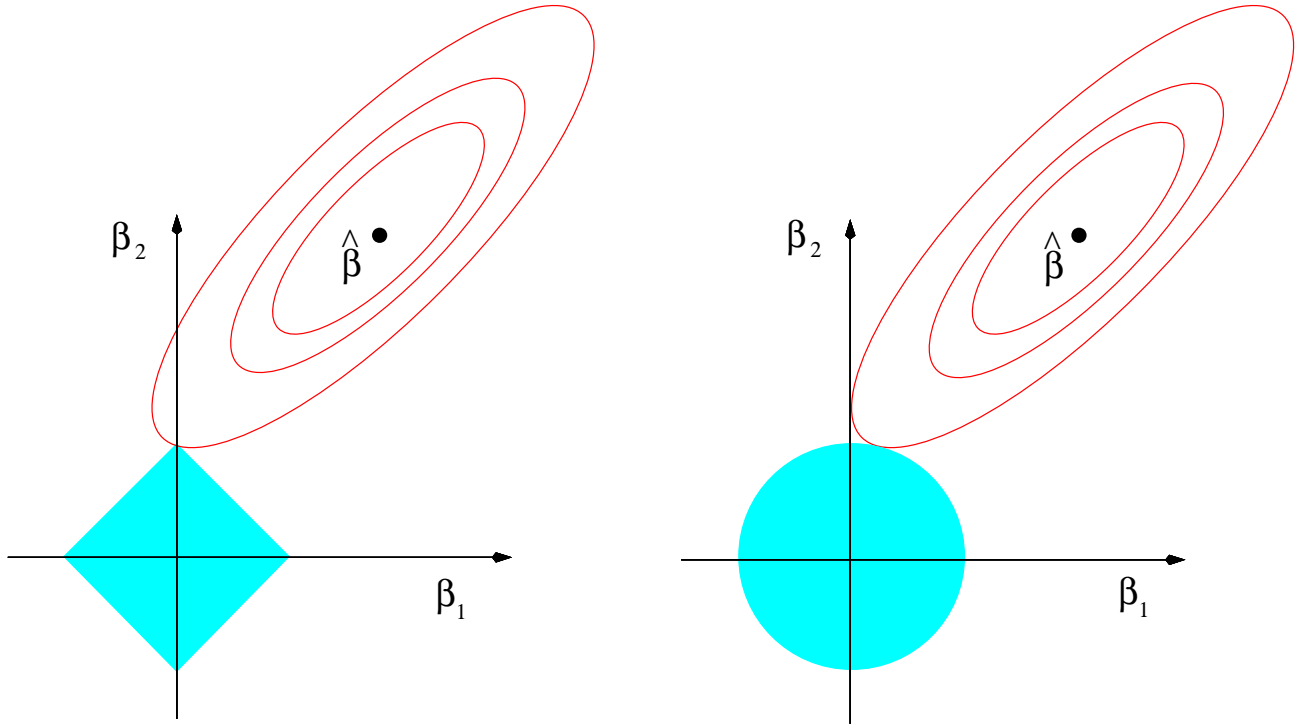


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$, *respectively, while the red ellipses are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

Unified representation

$$\widehat{\boldsymbol{\beta}}_q = \arg\min_{\boldsymbol{\beta}} \{ \frac{1}{2} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} |\beta_j|^q \}. \tag{37}$$

- $q = 0$: best subset
- $q = 1$: Lasso
- $q = 2$: Ridge regression

Bayesian interpretation:

- Lasso: double exponential prior,
- Density $(1/2\tau)\exp(-|\beta|/\tau)$ with $\tau = 1/\lambda$.

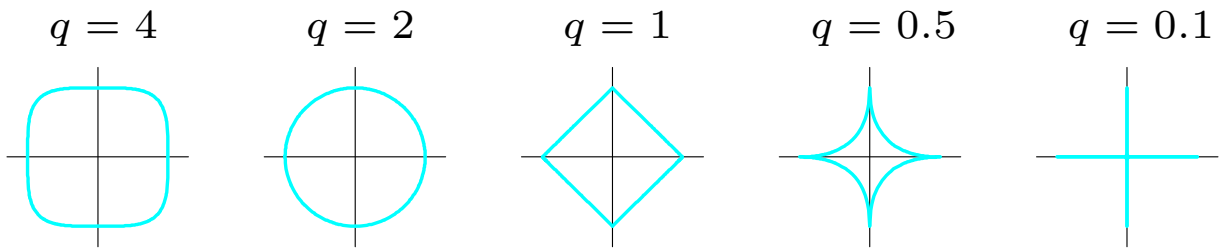


FIGURE 3.12. *Contours of constant value of* $\sum_j |\beta_j|^q$ *for given values of* q.

Elastic net (Zou and Hastie, 2005, JRSSB)

$$\widehat{\boldsymbol{\beta}}^{net} = \arg\min_{\boldsymbol{\beta}}\{(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}) + \lambda\sum_{j=1}^{p}(\alpha\beta_j^2 + (1-\alpha)|\beta_j|)\}. \quad (38)$$

Least Angle Regression (Efron et. al, 2004, AOS)

A related algorithm, *homotopy* algorithm (Osborne et al. 2000).

Algorithm 3.2 *Least Angle Regression.*

1. Standardize the predictors to have mean zero and unit norm. Start with the residual $\mathbf{r} = \mathbf{y} - \bar{\mathbf{y}}$, $\beta_1, \beta_2, \ldots, \beta_p = 0$.

2. Find the predictor $\mathbf{x}_j$ most correlated with $\mathbf{r}$.

3. Move β_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 β_j and β_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.

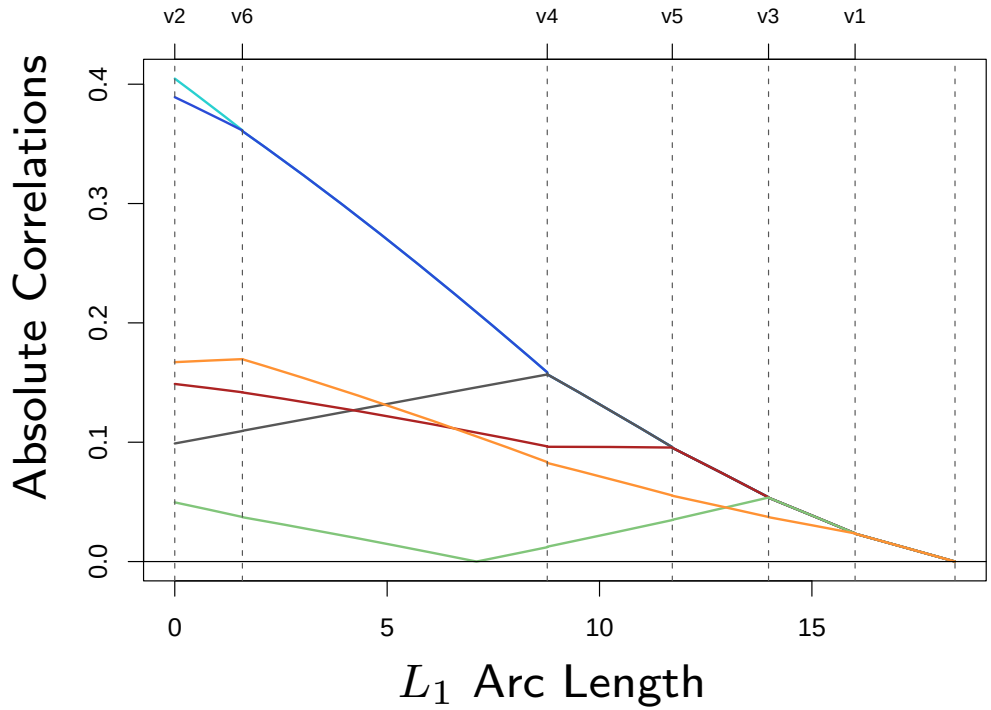


FIGURE 3.14. *Progression of the absolute correlations during each step of the LAR procedure, using a simulated data set with six predictors. The labels at the top of the plot indicate which variables enter the active set at each step. The step length are measured in units of L_1 arc length.*

More on LAR

- Track an active set $\mathcal{A}_k$ at the beginning of the k-th step.
- Denote the coefficient vector as $\boldsymbol{\beta}_{\mathcal{A}_k}$.
- There are $k-1$ nonzero values in $\boldsymbol{\beta}_{\mathcal{A}_k}$.
- Denote the current residual $\mathbf{r}_k = \mathbf{y} - \mathbf{X}_{\mathcal{A}_k}\boldsymbol{\beta}_{\mathcal{A}_k}$.
- The direction for this step is $\delta_k = (\mathbf{X}_{\mathcal{A}_k}^T \mathbf{X}_{\mathcal{A}_k})^{-1}\mathbf{X}_{\mathcal{A}_k}^T \mathbf{r}_k$.
- Update: $\boldsymbol{\beta}_{\mathcal{A}_k}(\alpha) = \boldsymbol{\beta}_{\mathcal{A}_k} + \alpha\delta_k$.
- Implemented in R package lars.

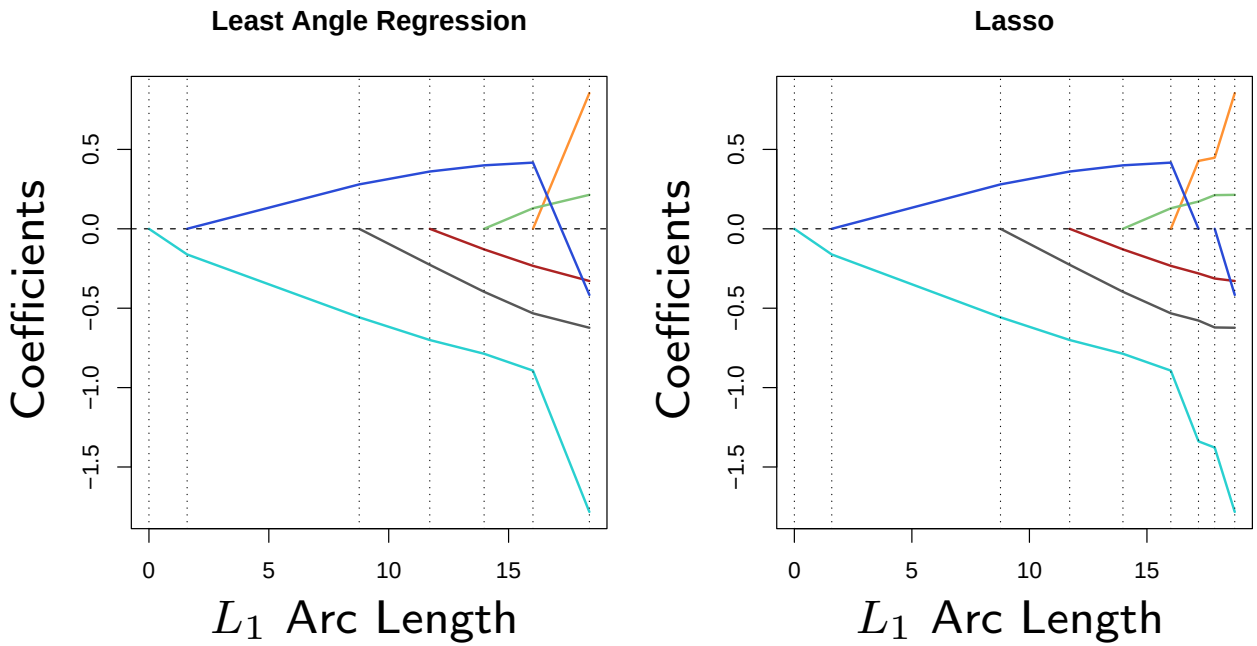


FIGURE 3.15. *Left panel shows the LAR coefficient profiles on the simulated data, as a function of the* L_1 *arc length. The right panel shows the Lasso profile. They are identical until the dark-blue coefficient crosses zero at an arc length of about* 18.

LAR vs LASSO

Suppose all input features are standardized, we can work with inner-products.

LAR

$$\mathbf{x}_j^T(\mathbf{y} - \mathbf{X}\beta) = \gamma s_j, \forall j \in \mathcal{A}, \tag{39}$$

where $s_j \in \{-1, 1\}$. And $|\mathbf{x}_j^T(\mathbf{y} - \mathbf{X}\beta)| \leq \gamma$ for all $j \notin \mathcal{A}$.

LASSO

Suppose $\mathcal{B}$ is the active set.

$$\mathbf{x}_j^T(\mathbf{y} - \mathbf{X}\beta) = \lambda \text{sign}(\beta_j), \forall j \in \mathcal{B} \tag{40}$$

$$|\mathbf{x}_j^T(\mathbf{y} - \mathbf{X}\beta)| \leq \lambda, \forall j \notin \mathcal{B} \tag{41}$$

LAR-LASSO modification

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.

Degrees of Freedom

A general definition with a fitted vector $\hat{\mathbf{y}} = (\hat{y}_1, \cdots, \hat{y}_N)$.

$$df(\hat{\mathbf{y}}) = \frac{1}{\sigma^2} \sum_{i=1}^{N} Cov(\hat{y}_i, y_i). \tag{42}$$

- Linear regression with k predictors: $df(\hat{\mathbf{y}}) = k$
- Ridge regression with penalty λ: $df(\hat{\mathbf{y}}) = tr(\mathbf{H}_\lambda)$.
- LAR at k-th step, $df(\hat{\mathbf{y}}) = k$.
- LASSO, $df(\hat{\mathbf{y}}) \approx |\mathcal{B}|_0$, the number of active variables. More at Zou et al. (2007), AOS.

Principal Components Regression (pcr)

- Derived input columns $\mathbf{z}_m = \mathbf{X}\mathbf{v}_m$, then regression $\mathbf{y}$ on $\mathbf{z}_1, \cdots, \mathbf{z}_M$ for some $M \leq p$.
-

$$\hat{\mathbf{y}}_{(M)}^{pcr} = \bar{y}\mathbf{1} + \sum_{m=1}^{M} \hat{\theta}_m \mathbf{z}_m, \tag{43}$$

where $\hat{\theta}_m =< \mathbf{z}_m, \mathbf{y} > / < \mathbf{z}_m, \mathbf{z}_m >$.

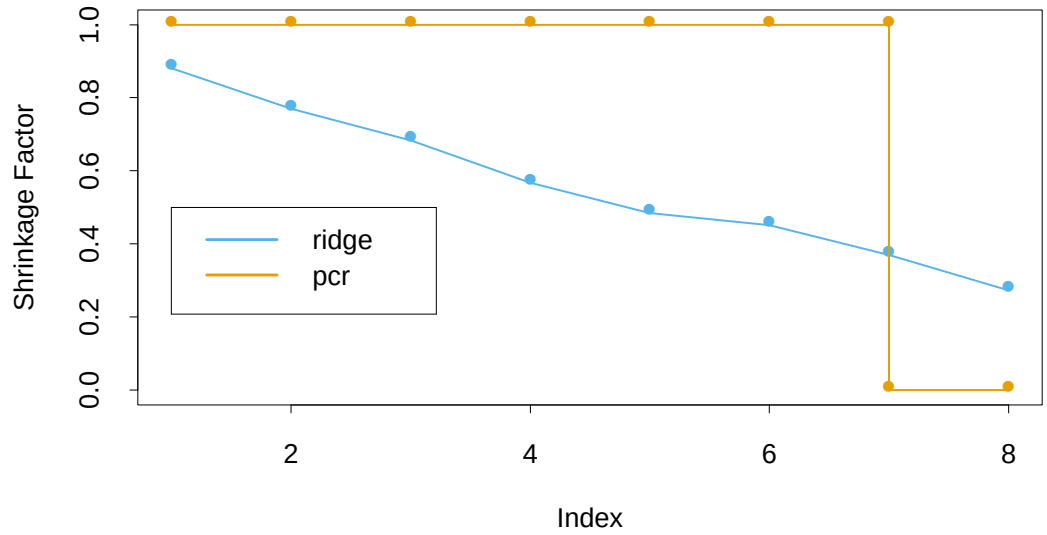


FIGURE 3.17. *Ridge regression shrinks the regression coefficients of the principal components, using shrinkage factors* $d_j^2/(d_j^2 + \lambda)$ *as in (3.47). Principal component regression truncates them. Shown are the shrinkage and truncation patterns corresponding to Figure 3.7, as a function of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

Partial Least Squares (PLS)

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 $\mathbf{x}_j^{(0)} = \mathbf{x}_j,\ j = 1, \ldots, p$.

2. For $m = 1, 2, \ldots, p$

   (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 $\mathbf{z}_m$: $\mathbf{x}_j^{(m)} = \mathbf{x}_j^{(m-1)} - [\langle \mathbf{z}_m, \mathbf{x}_j^{(m-1)} \rangle / \langle \mathbf{z}_m, \mathbf{z}_m \rangle]\mathbf{z}_m,\ j = 1, 2, \ldots, p$.

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.

PLS vs. PCR

PCR

The m-th PC direction $\mathbf{v}_m$ solves

$$\max_{\boldsymbol{\alpha}} \ Var(\mathbf{X}\boldsymbol{\alpha}) \tag{44}$$

$$s.t. \ \|\boldsymbol{\alpha}\| = 1, \boldsymbol{\alpha}^T \mathbf{S}\mathbf{v}_\ell = 0, \ell = 1, \cdots, m-1, \tag{45}$$

PLS

The m-th PC direction $\boldsymbol{\varphi}_m$ solves

$$\max_{\boldsymbol{\alpha}} \ Corr^2(\mathbf{y}, \mathbf{X}\boldsymbol{\alpha}) Var(\mathbf{X}\boldsymbol{\alpha}) \tag{46}$$

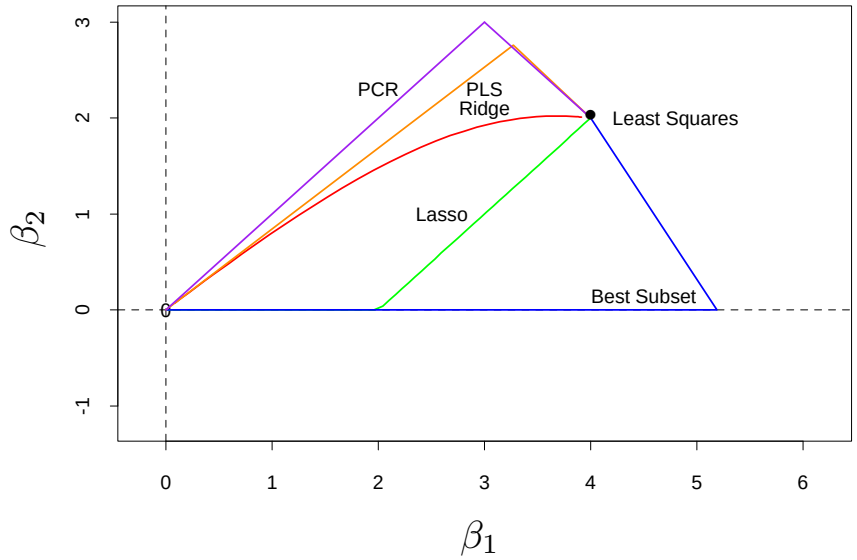
$$s.t. \ \|\boldsymbol{\alpha}\| = 1, \boldsymbol{\alpha}^T \mathbf{S}\boldsymbol{\varphi}_\ell = 0, \ell = 1, \cdots, m-1, \tag{47}$$

Comparison of selection and shrinkage methods

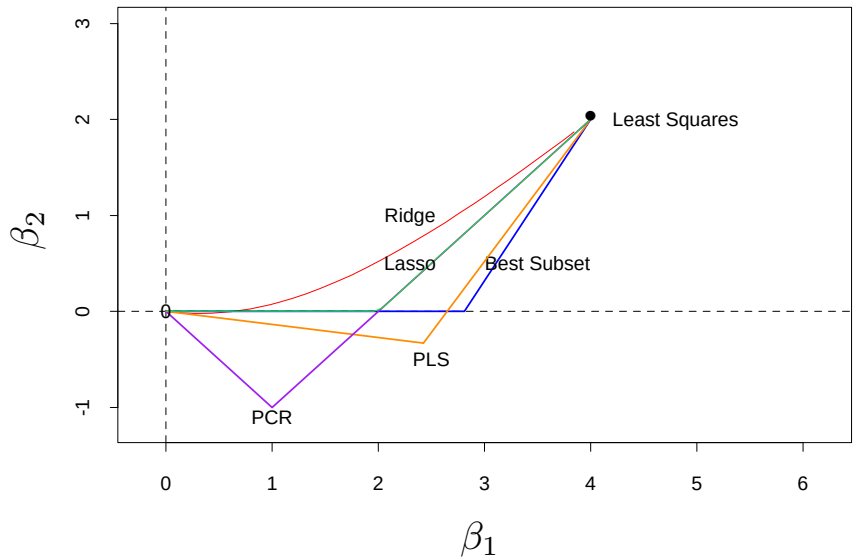
Two inputs with correlation ± 0.5, and the true regression coefficients $\beta = (4, 2)^T$.

- Least Square
- Best subset
- Ridge
- Lasso
- PCR
- PLS

$\rho = 0.5$


PCR
PLS
Ridge
Least Squares
Lasso
Best Subset
0
β_2
β_1

$\rho = -0.5$


Least Squares
Ridge
Lasso
Best Subset
PLS
PCR
0
β_2
β_1

Incremental Forward Stagewise Regression

Algorithm 3.4 *Incremental Forward Stagewise Regression—FS_ϵ.*

1. Start with the residual $\mathbf{r}$ equal to $\mathbf{y}$ and $\beta_1, \beta_2, \ldots, \beta_p = 0$. 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 $\delta_j = \epsilon \cdot \text{sign}[\langle \mathbf{x}_j, \mathbf{r} \rangle]$ 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.

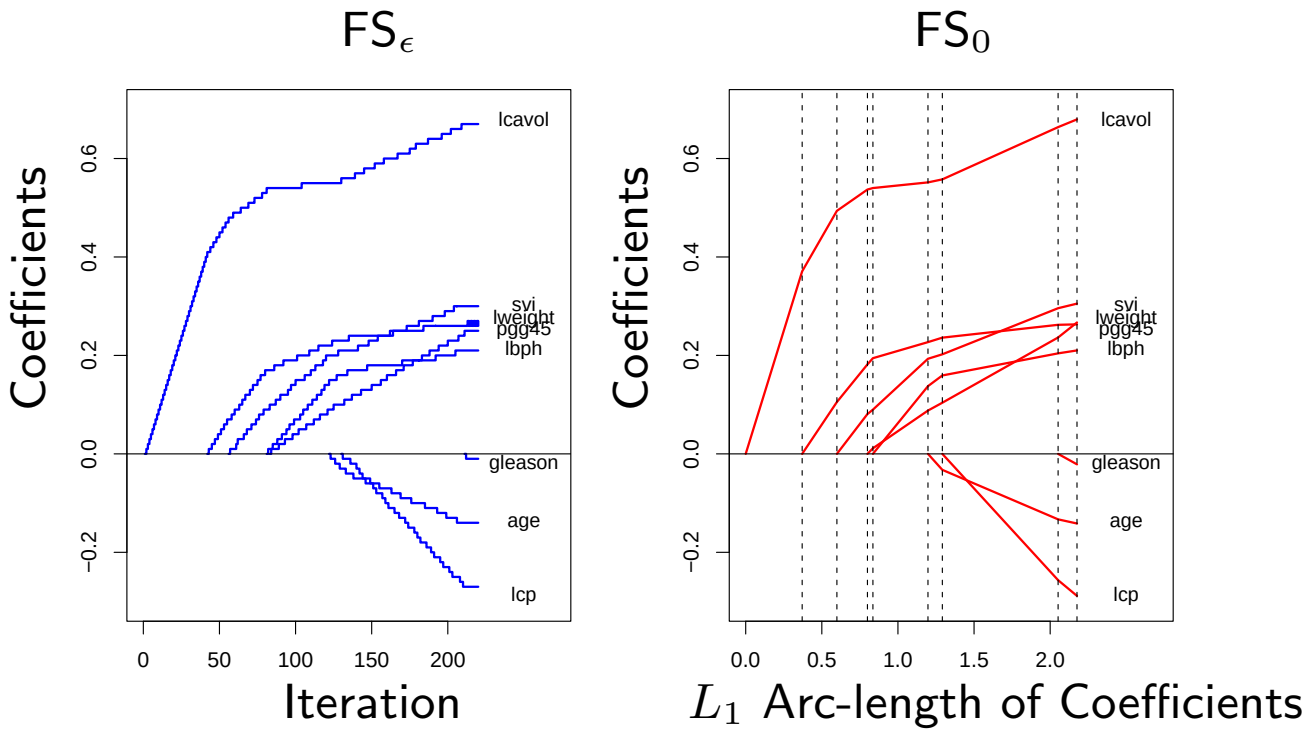


FIGURE 3.19. *Coefficient profiles for the prostate data. The left panel shows incremental forward stagewise regression with step size* $\epsilon = 0.01$. *The right panel shows the infinitesimal version* FS_0 *obtained letting* $\epsilon \to 0$. *This profile was fit by the modification 3.2b to the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

Connection to LAR and Boosting

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 $\langle \mathbf{x}_j, \mathbf{r} \rangle$.

Bühlmann and Hothorn (2007) refer to Incremental Forward Stagewise Regression as "L2boost".

Dantzig Selector (Candes and Tao, 2007, AOS)

In honor of George Dantzig (Inventor of the simplex method for linear programming).

DS

$$\min_{\beta} \|\beta\|_1 \text{ s.t. } \|\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)\|_\infty \leq s. \tag{48}$$

Equivalently,

$$\min_{\beta} \|\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)\|_\infty \text{ s.t. } \|\beta\|_1 \leq t. \tag{49}$$

Equivalent to LASSO (Bickel et al., 2008, AOS).

Group Lasso (Bakin, 1999; Lin and Zhang, 2006; Yuan and Lin, 2007)

- Predictors belong to pre-defined groups, e.g., factors.
- p predictors, divide into L groups, p_ℓ the number of in group ℓ.
- Formulation:

$$\min_{\boldsymbol{\beta}} \|\mathbf{y} - \beta_0 \mathbf{1} - \sum_{\ell=1}^{L} \mathbf{X}_\ell \boldsymbol{\beta}_\ell\|^2 + \lambda \sum_{\ell=1}^{L} \sqrt{p_l} \|\boldsymbol{\beta}_\ell\|_2 \tag{50}$$

Folded Concave Penalties

- Why shrink all the coefficients in the same amount?
- Smoothly Clipped Absolute Deviation (Fan and Li, 2001, JASA)

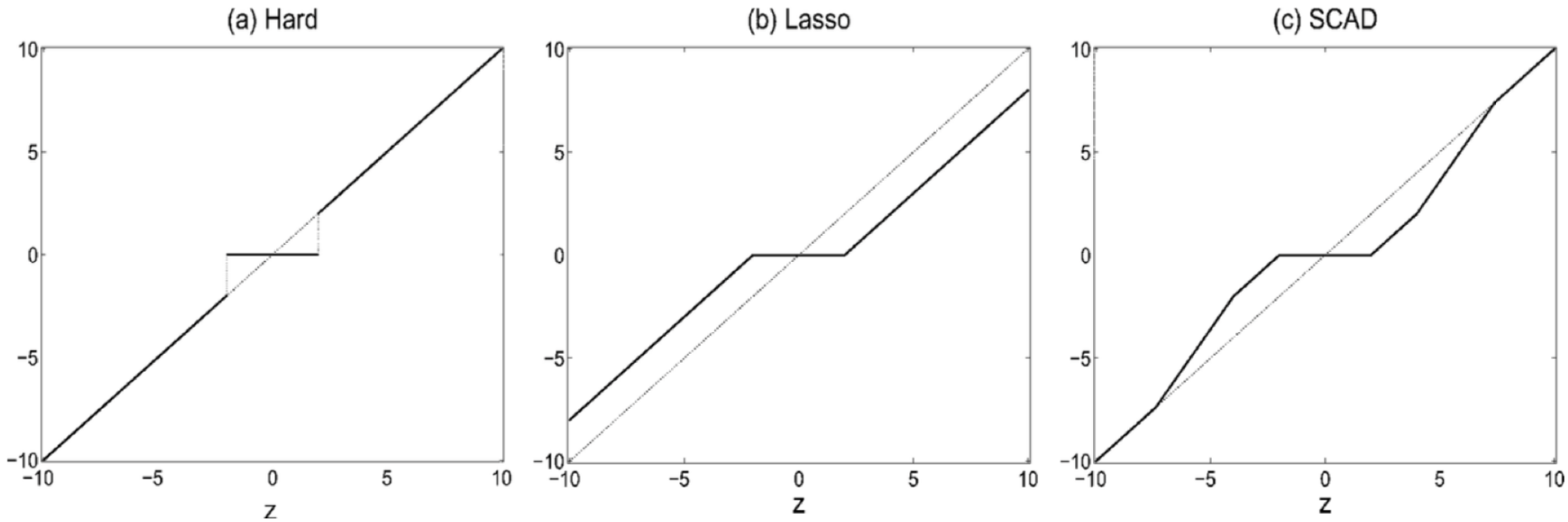


Figure 2. Plot of Thresholding Functions for (a) the Hard, (b) the Soft, and (c) the SCAD Thresholding Functions With $\lambda = 2$ and $a = 3.7$ for SCAD.

Penalized Least Square

A generic penalized least square problem

$$\arg\min_{\beta}\left\{\frac{1}{2}(\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta)+\sum_{j=1}^{p}p_\lambda(|\beta_j|)\right\} \tag{51}$$

SCAD

$$p'_\lambda(\beta)=\lambda\left\{I(\beta\le\lambda)+\frac{(a\lambda-\beta)_+}{(a-1)\lambda}I(\theta>\beta)\right\} \tag{52}$$

Three properties

Consider the penalized least square problem

$$\frac{1}{2}(z-\beta)^2 + p_\lambda(|\beta|). \tag{53}$$

From Fan and Li (2001), a good penalty function should result in an estimator with three properties.

- Unbiasedness: The resulting estimator is nearly unbiased when the true unknown parameter is large to avoid unnecessary modeling bias.
- Sparsity: The resulting estimator is a thresholding rule, which automatically sets small estimated coefficients to zero to reduce model complexity.
- Continuity: The resulting estimator is continuous in data to avoid instability in model prediction.

Minimum Concavity Penalty (Zhang, 2010, AOS)

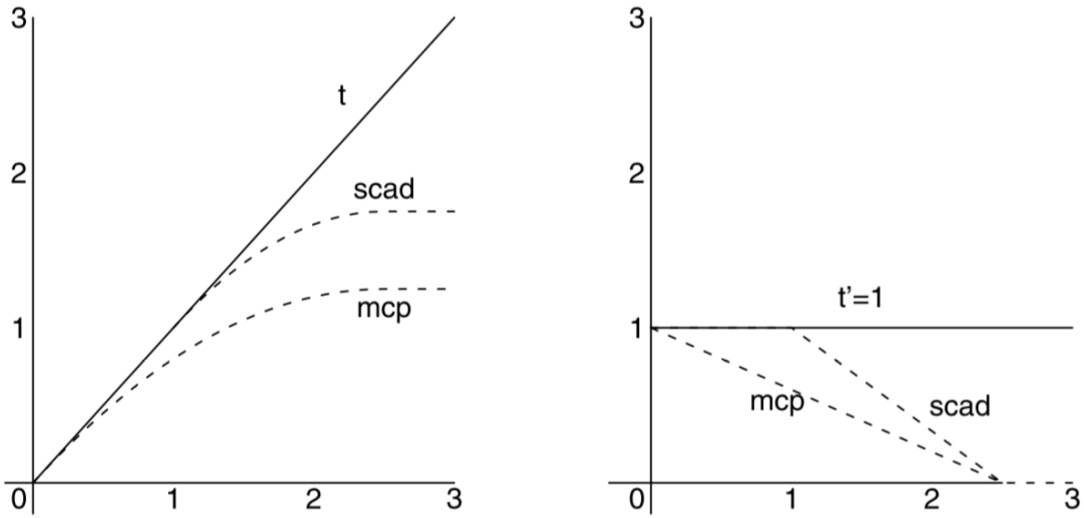


FIG. 1. *The ℓ_1 penalty $\rho_1(t) = t$ for the LASSO along with the MCP $\rho_2(t)$ and the SCAD penalty $\rho_3(t)$, $t > 0$, $\gamma = 5/2$. Left: penalties $\rho_m(t)$. Right: their derivatives $\dot{\rho}_m(t)$.*

Adaptive Lasso (Zou, 2006, JASA)

- Penalize large coefficients less and small coefficients more.
-

$$\arg\min_{\beta} \left\{ \frac{1}{2}(\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta) + \sum_{j=1}^{p} \frac{1}{|\hat{\beta}_j^{LS}|^{\nu}} |\beta_j| \right\}, \tag{54}$$

for some $0 < \nu \leq 1$.

Pathwise Coordinate Optimization

- Usually called "Coordinate Descent" algorithm.
- Very simple idea: optimize over one parameter at a time, keeping all the others fixed.
- Proposed by Fu (1998) (Tibshirani's student), shooting algorithm. Tibshirani doesn't fully appreciate it.
- In 2002, Ingrid Daubechies gives a talk at Stanford, describes a one-at-a-time algorithm for the lasso. Hastie implements it, makes an error, and Hastie +Tibshirani conclude that the method doesn't work.
- Friedman et al. (2007), Pathwise Coordinate Optimization, AoAS.

Coordinate wise update

- Suppose all predictors are standardized.
- At penalty λ, denote the estimate as $\tilde{\boldsymbol{\beta}}(\lambda)$.
- Fix all coordinates except β_j, find β_j to minimize

$$R(\beta_j) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \sum_{k\neq j} x_{ik}\tilde{\beta}_k(\lambda) - x_{ij}\beta_j\right)^2 + \lambda\sum_{k\neq j}|\tilde{\beta}_k(\lambda)| + \lambda|\beta_j|. \tag{55}$$

- We have the update for $\tilde{\beta}_j$ as follows.

$$\tilde{\beta}_j(\lambda) \leftarrow S\left(\sum_{i=1}^{N} x_{ij}(y_i - \sum_{k\neq j} x_{ik}\tilde{\beta}_k(\lambda)), \lambda\right), \tag{56}$$

where $S(t,\lambda) = \text{sign}(t)(|t| - \lambda)_+$ is the soft-thresholding operator.

Warm start strategy

- Start with the smallest value $\lambda_{\max}$ such that $\hat{\boldsymbol{\beta}}(\lambda_{\max}) = \mathbf{0}$.
- Q: what value?
- Choose an exponentially decaying sequence $\lambda_{\max} = \lambda_1 > \lambda_2 > \cdots > \lambda_K = \epsilon\lambda_{\max} = \lambda_{\min}$, where K is the total number of penalty parameters (typical value 100), ϵ is a small value controlling $\lambda_{\min}$ (typical value 1e-2 or 1e-4).
- The solution $\hat{\boldsymbol{\beta}}(\lambda_k)$ for $\lambda = \lambda_k$ can be used as the starting solution for $\lambda = \lambda_{k+1}$.
- R package glmnet.

Computation Complexity

- N observations, p predictors.
- Least Square:
  - Cholesky decomposition $\rightarrow p^3 + Np^2/2$.
  - QR decomposition $\rightarrow Np^2$.
- LAR: Same computation cost.

Outline

Linear regression of an indicator matrix

- $\mathcal{G}$ has K classes, K indicators $Y_k, k = 1, \cdots, K$. $Y_k = 1$ if $G = k$.
- $Y = (Y_1, \cdots, Y_K)^T$ representing the class.
- $N \times K$ dimensional indicator response matrix $\mathbf{Y}$
- Fit $\mathbf{Y}$ on $\mathbf{X}$.
- $\hat{\mathbf{Y}} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$.
- Coefficient matrix $\hat{\mathbf{B}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$.
- For a new observation with input $\mathbf{x}$,
  1. Computed fitted output $\hat{f}(x)^T = (1, \mathbf{x}^T)^T\hat{\mathbf{B}}$.
  2. Set $\hat{G}(x) = \arg\max_{k\mathcal{G}} \hat{f}_k(x)$.

Masking Issue

- When $K > 2$, some classes can be *masked* by others.
- Polynomial terms up to degree $K - 1$ might be needed to separate the classes.

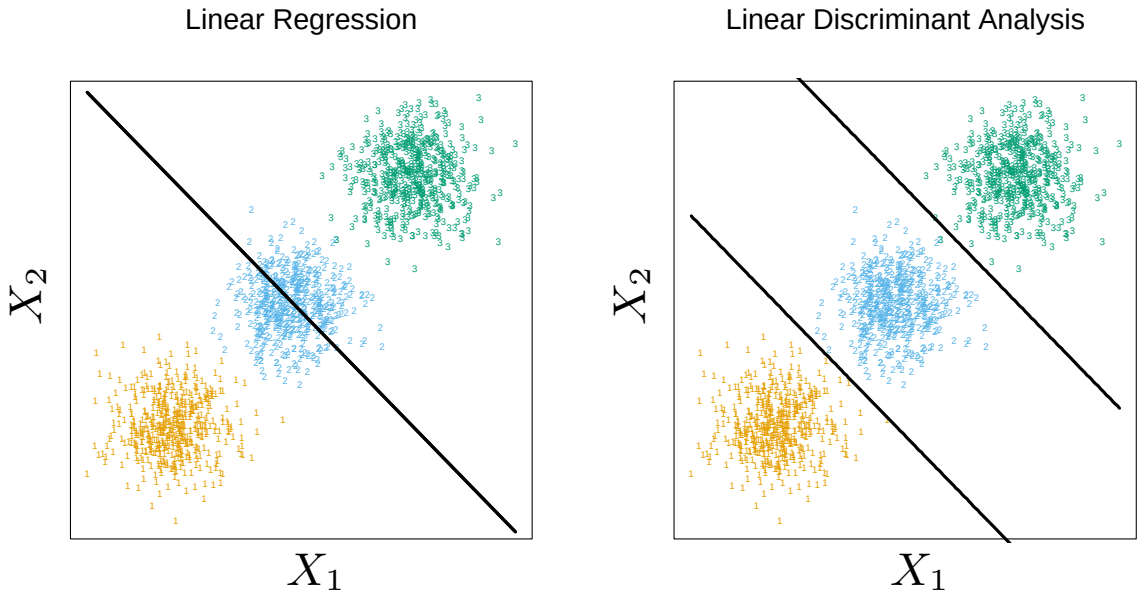


FIGURE 4.2. *The data come from three classes in* $\mathbb{R}^2$ *and are easily separated by linear decision boundaries. The right plot shows the boundaries found by linear discriminant analysis. The left plot shows the boundaries found by linear regression of the indicator response variables. The middle class is completely masked (never dominates).*

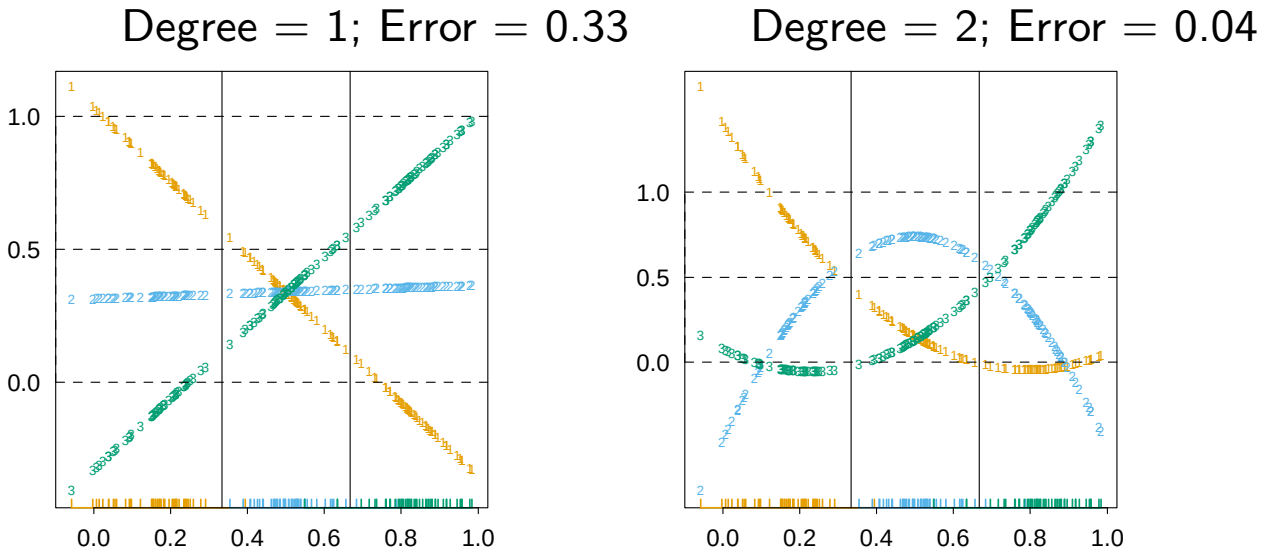


FIGURE 4.3. *The effects of masking on linear regression in* $\mathbb{R}$ *for a three-class problem. The* rug plot *at the base indicates the positions and class membership of each observation. The three curves in each panel are the fitted regressions to the three-class indicator variables; for example, for the blue class,* y_{blue} *is* 1 *for the blue observations, and* 0 *for the green and orange. The fits are linear and quadratic polynomials. Above each plot is the training error rate. The Bayes error rate is* 0.025 *for this problem, as is the LDA error rate.*

General Setup

- From decision theory, we need $Pr(G|X)$ for optimal classification.
- Suppose $f_k(x)$ is the class-conditional density of X in class $G = k$.
- π_k be the prior probability of class k. $\sum_{k=1}^{K} \pi_k = 1$
- From Bayes theorem

$$Pr(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^{K} f_\ell(x)\pi_\ell}. \tag{57}$$

Possible densities $f_k(x)$

- Gaussian densities: LDA, QDA
- mixtures of Gaussian: nonlinear decision boundaries
- general nonparametric density estimates
- independent conditional on class: Naive Bayes

Linear Discriminant Analysis

- Suppose each class is from Gaussian

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \mathbf{\Sigma}_k^{-1}(x-\mu_k)}. \tag{58}$$

- LDA: $\mathbf{\Sigma}_k = \mathbf{\Sigma}$ for any k.
-

$$\log \frac{Pr(G=k|X=x)}{Pr(G=\ell|X=x)} \tag{59}$$

$$= \log \frac{f_k(x)}{f_\ell(x)} + \log \frac{\pi_k}{\pi_\ell} \tag{60}$$

$$= \log \frac{\pi_k}{\pi_\ell} - \frac{1}{2}(\mu_k + \mu_\ell)^T \mathbf{\Sigma}^{-1}(\mu_k - \mu_\ell) + x^T \mathbf{\Sigma}^{-1}(\mu_k - \mu_\ell). \tag{61}$$

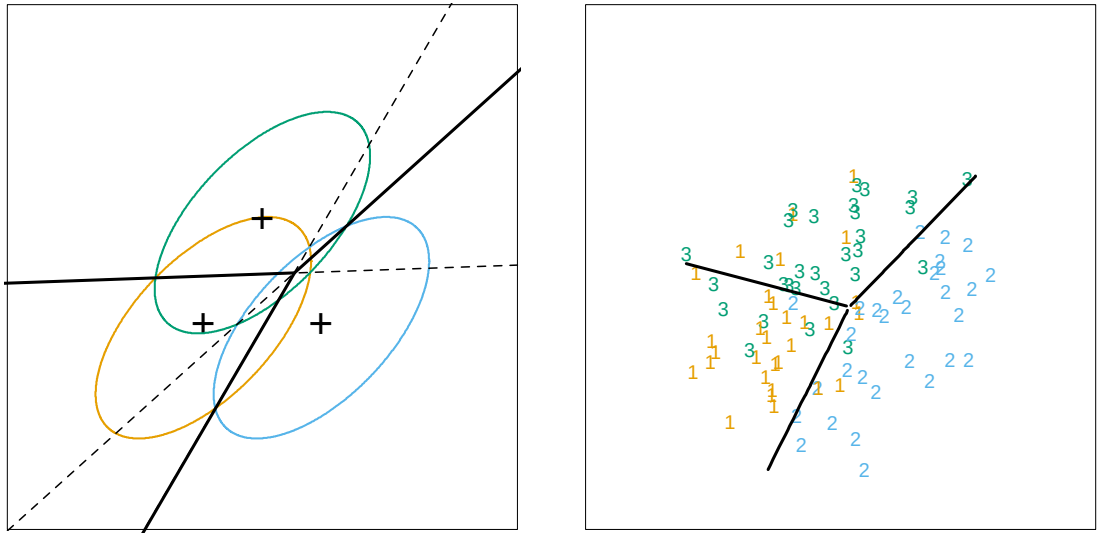


FIGURE 4.5. *The left panel shows three Gaussian distributions, with the same covariance and different means. Included are the contours of constant density enclosing* 95% *of the probability in each case. The Bayes decision boundaries between each pair of classes are shown (broken straight lines), and the Bayes decision boundaries separating all three classes are the thicker solid lines (a subset of the former). On the right we see a sample of* 30 *drawn from each Gaussian distribution, and the fitted LDA decision boundaries.*

Linear discriminant functions

- $\delta_k(x) = x^T \mathbf{\Sigma}^{-1} \mu_k - \frac{1}{2} \mu_k^T \mathbf{\Sigma}^{-1} \mu_k + \log \pi_k$.
- Set $G(x) = \arg\max_k \delta_k(x)$.
- In practice, use samples to estimate $\pi_k, \mu_k, \mathbf{\Sigma}$.
- LDA classifier to class 2 if

$$x^T \hat{\mathbf{\Sigma}}^{-1} (\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{2} (\hat{\mu}_2 + \hat{\mu}_1)^T \hat{\mathbf{\Sigma}}^{-1} (\hat{\mu}_2 - \hat{\mu}_1) - \log(N_2/N_1). \quad (62)$$

- Direction is the same to solution of least square. Intercept different unless $N_1 = N_2$.
- In practice, use training data to choose intercept.

Quadratic Discriminant Analysis

- Quadratic discriminant functions:
- $\delta_k(x) = -\frac{1}{2}\log|\mathbf{\Sigma}_k| - \frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}_k^{-1}(x-\mu_k) + \log\pi_k$.
- Set $G(x) = \arg\max_k \delta_k(x)$.
- In practice, use samples to estimate $\pi_k, \mu_k, \mathbf{\Sigma}_k$.
- No. of parameters:
  - LDA: $(K-1)\times(p+1)$.
  - QDA:
    $(K-1)\times\{p(p-1)/2+p+p+1\} = (K-1)\times\{p(p+3)/2+1\}$.
  - Off-diagonal of Σ_k, diagonal of Σ_k, μ_k, π_k.

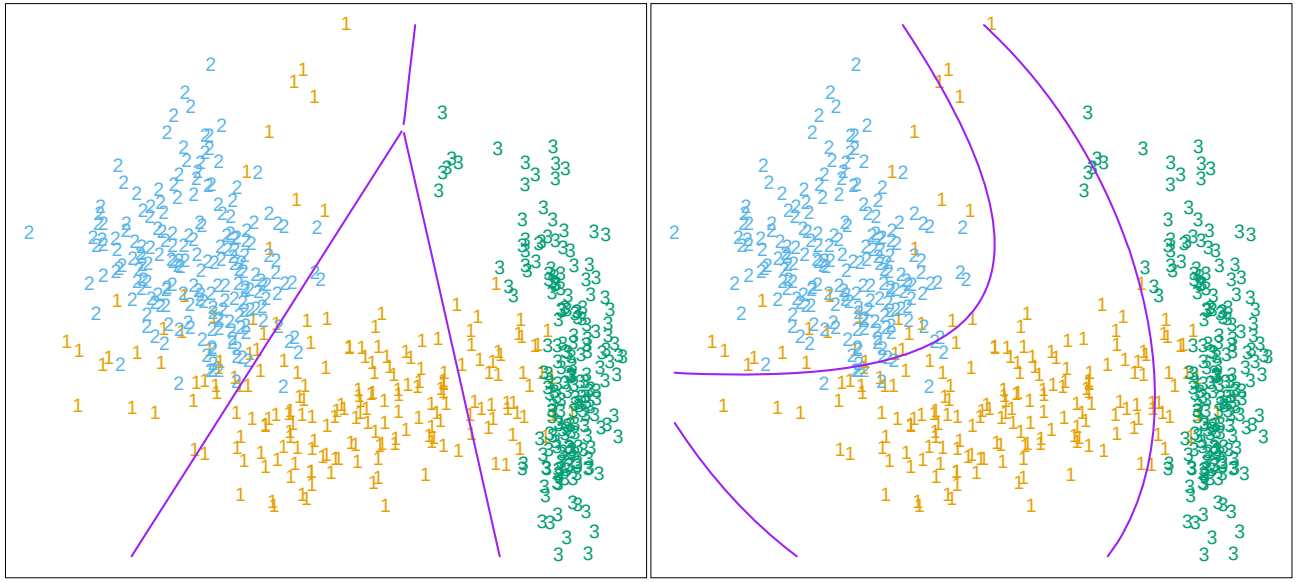


FIGURE 4.1. *The left plot shows some data from three classes, with linear decision boundaries found by linear discriminant analysis. The right plot shows quadratic decision boundaries. These were obtained by finding linear boundaries in the five-dimensional space* $X_1, X_2, X_1X_2, X_1^2, X_2^2$. *Linear inequalities in this space are quadratic inequalities in the original space.*

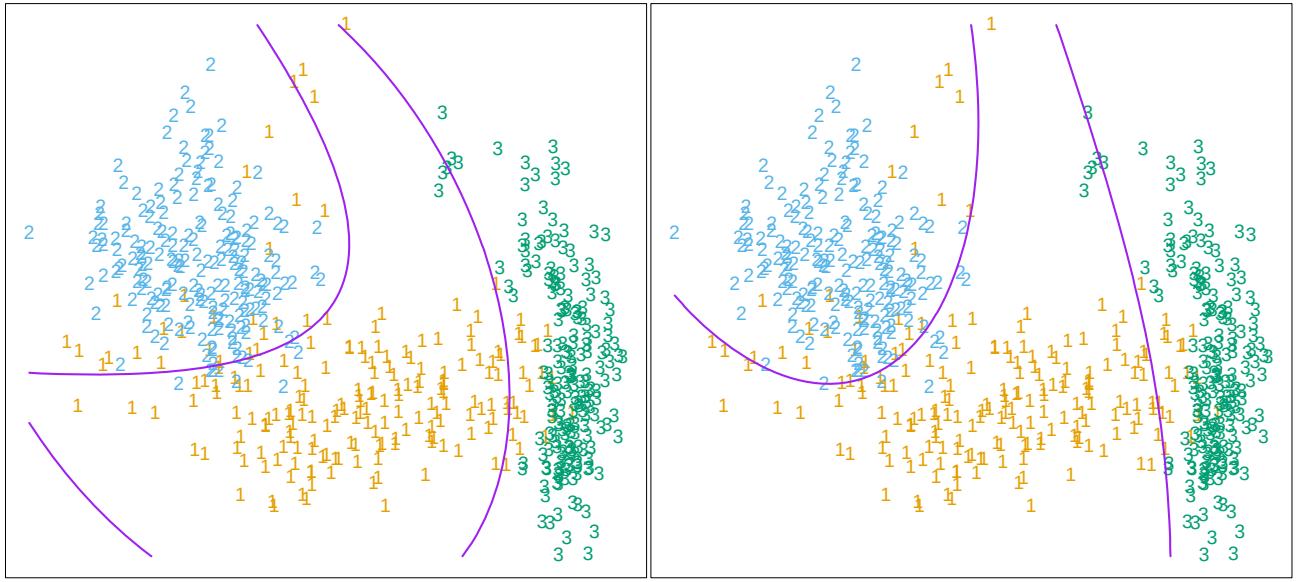


FIGURE 4.6. *Two methods for fitting quadratic boundaries. The left plot shows the quadratic decision boundaries for the data in Figure 4.1 (obtained using LDA in the five-dimensional space* $X_1, X_2, X_1X_2, X_1^2, X_2^2$*). The right plot shows the quadratic decision boundaries found by QDA. The differences are small, as is usually the case.*

Strong performance of LDA and QDA

- LDA and QDA perform well on a large and diverse set of classification tasks.
- They should serve as benchmarks.
- Reason: not because the data are approximately Gaussian, but data can only support simple decision boundaries such as linear or quadratic.
- Bias and variance tradeoff!

Regularized Discriminant Analysis (RDA), Friedman, 1989

- Tradeoff between LDA and QDA
- $\hat{\mathbf{\Sigma}}_k(\alpha) = \alpha\hat{\mathbf{\Sigma}}_k + (1-\alpha)\hat{\mathbf{\Sigma}}$.
- $\alpha \in [0,1]$ controls the amount of shrinkage to LDA from QDA, can be chosen by CV.

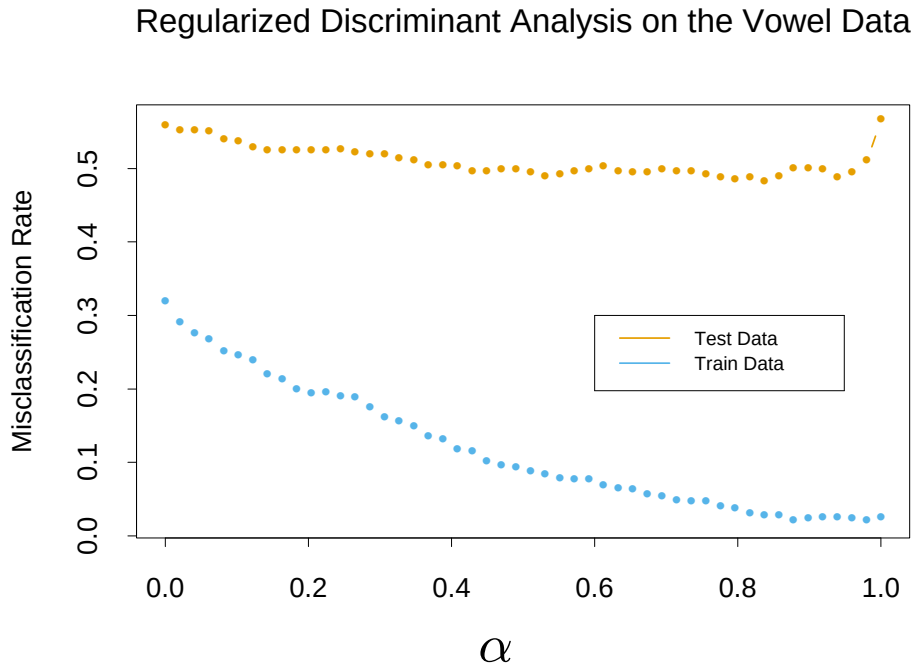


FIGURE 4.7. *Test and training errors for the vowel data, using regularized discriminant analysis with a series of values of* $\alpha \in [0, 1]$. *The optimum for the test data occurs around* $\alpha = 0.9$, *close to quadratic discriminant analysis.*

Projection view of LDA

- K centroids in p-dimensional input space lies in an affine subspace of dimension $\leq K-1$. Call it H_{K-1}
- We can project X onto this centroid-spanning subspace H_{K-1}. Then make distance comparisons.
- No information loss in the LDA process.

Reduced-Rank LDA

- If $K = 3$, project the data into two-dimensional plot.
- If $K > 3$, may want a $L < K - 1$ dimensional subspace $H_L \subset H_{K-1}$ optimal for LDA.
- Fisher defined optimal: the projected centroids were spread out as much as possible in terms of variance.
- In other words, finding principal component subspaces of the centroids.

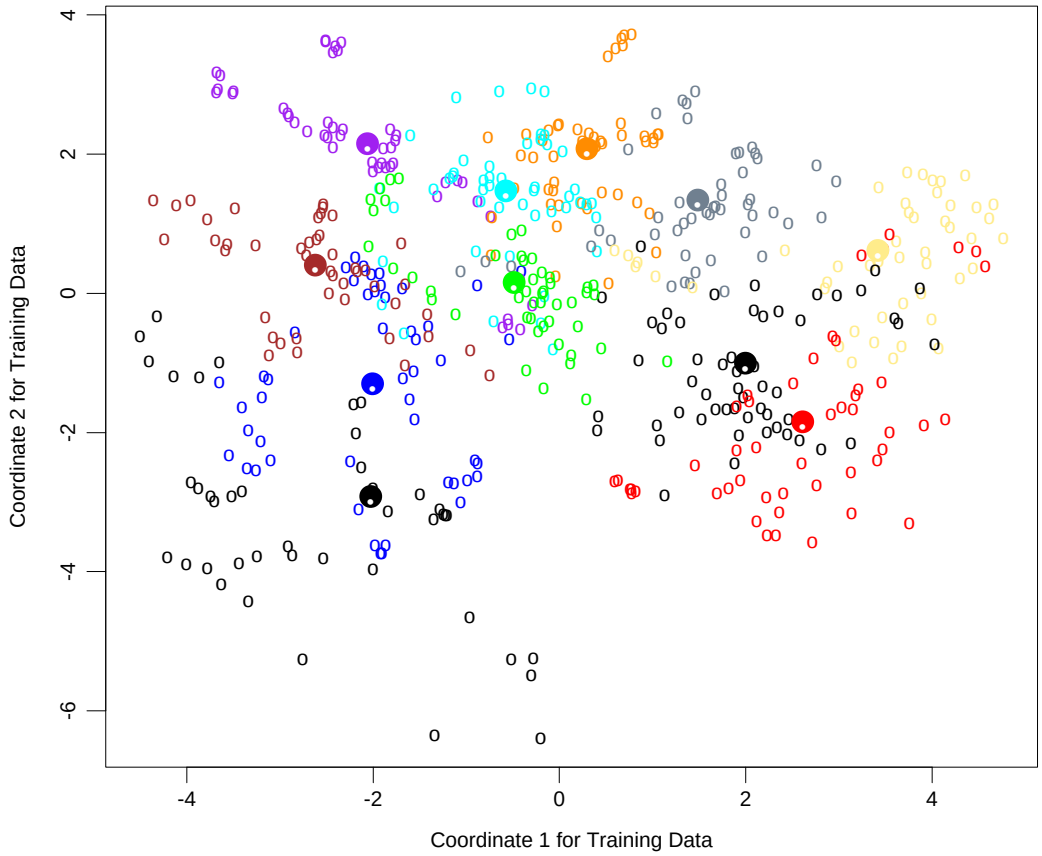


FIGURE 4.4. *A two-dimensional plot of the vowel training data. There are eleven classes with* $X \in \mathbb{R}^{10}$*, and this is the best view in terms of a LDA model (Section 4.3.3). The heavy circles are the projected mean vectors for each class. The class overlap is considerable.*

Finding the optimal subspace

- compute the $K \times p$ matrix of class centroids M and the common covariance matrix W (for within-class covariance)
- compute $M^* = MW^{-1/2}$ using the eigen-decomposition of W.
- compute B^*, the covariance matrix of M^* (B for between-class covariance), and its eigen-decomposition $B^* = V^* D_B (V^*)^T$. The columns v_ℓ^* of V^* in sequence from first to last define the coordinates of the optimal subspaces.
- The ℓ-th *discriminant variable* is $Z_\ell = v_\ell^T X$ with $v_\ell = W^{-1/2} v_\ell^*$.

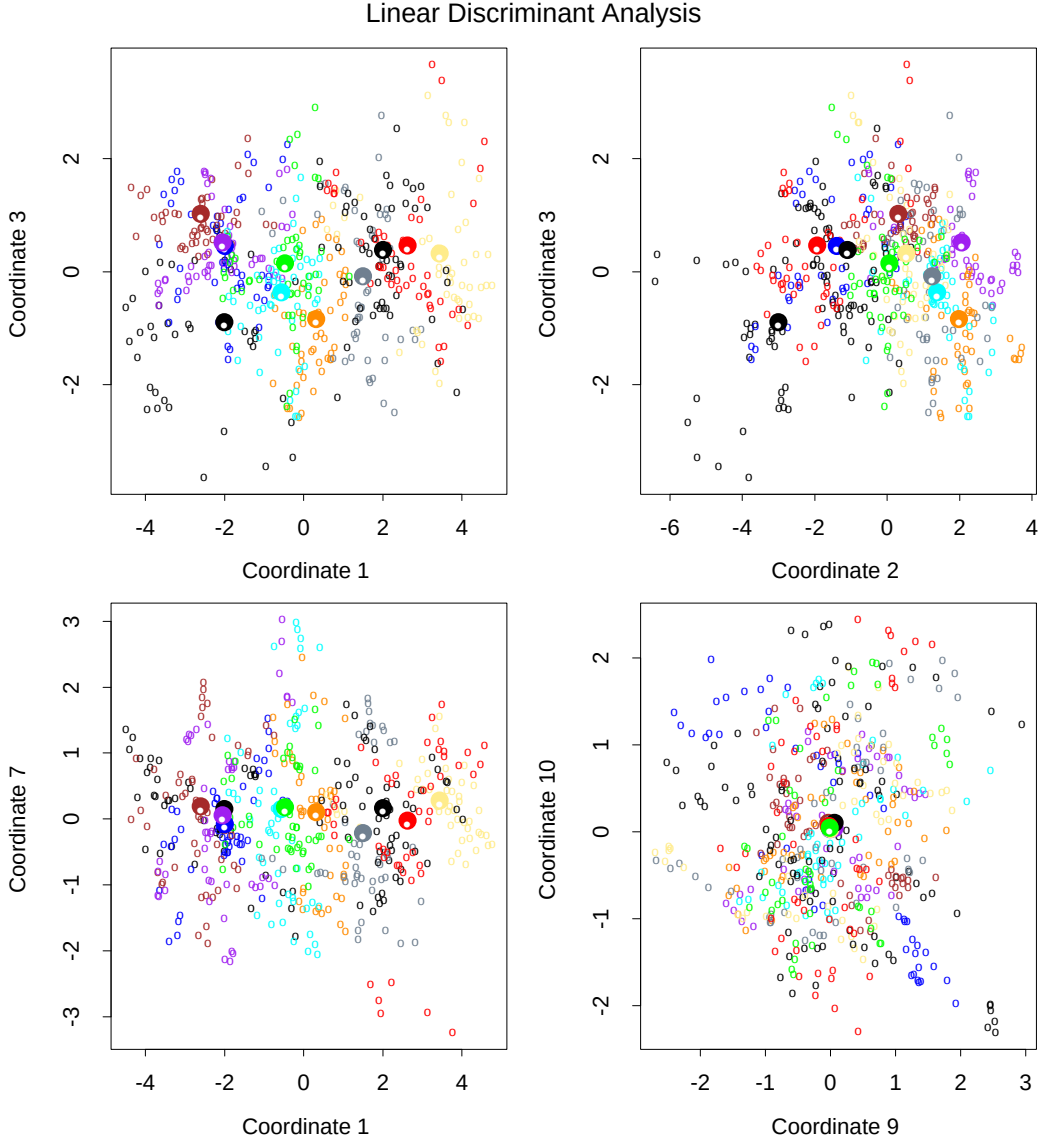


FIGURE 4.8. *Four projections onto pairs of canonical variates. Notice that as the rank of the canonical variates increases, the centroids become less spread out. In the lower right panel they appear to be superimposed, and the classes most confused.*

An equivalent formulation

Fisher's question

Find the linear combination $Z = a^T X$ such that the between-class variance is maximized relative to the within-class variance.

- Between-class variance: $a^T B a$, where B is the covariance matrix of the class centroid matrix M. Here, $M = (\mu_1, \mu_2, \cdots, \mu_K)^T$. And $B = \frac{1}{K-1}(M - \bar{M})^T(M - \bar{M})$, where $\bar{M} = (\bar{\mu}, \bar{\mu}, \cdots, \bar{\mu})^T$.
- Within-class variance: $a^T W a$. Here, $W = \Sigma$.
- $T = B + W$ is the *total covariance matrix* of X, ignoring class information.

Optimization problem

Rayleigh quotient

Maximize *Rayleigh quotient*,

$$\max_a \frac{a^T Ba}{a^T Wa}, \tag{63}$$

or equivalently,

$$\max_a a^T Ba, \text{ subject to } a^T Wa = 1. \tag{64}$$

- Generalized Eigenvalue Problem

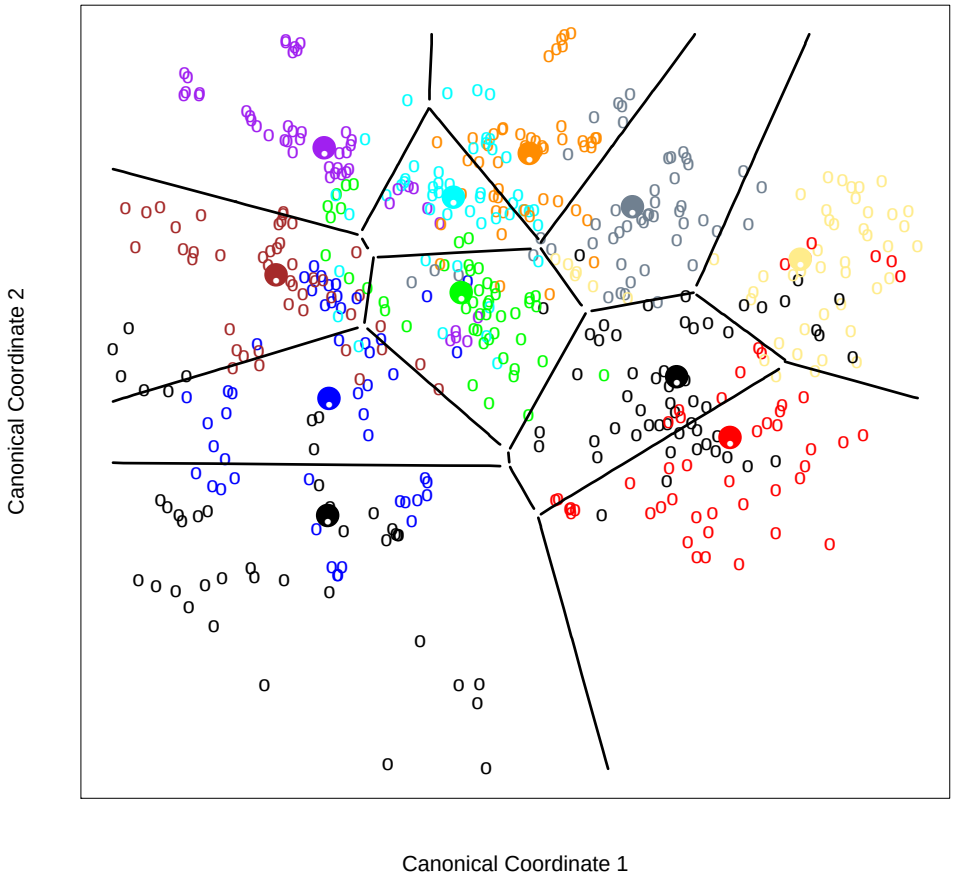


FIGURE 4.11. *Decision boundaries for the vowel training data, in the two-dimensional subspace spanned by the first two canonical variates. Note that in any higher-dimensional subspace, the decision boundaries are higher-dimensional affine planes, and could not be represented as lines.*

Logistic regression

- Setup

$$\log \frac{Pr(G=1|X=x)}{Pr(G=K|X=x)} = \beta_{10} + \beta_1^T x \tag{65}$$

$$\log \frac{Pr(G=2|X=x)}{Pr(G=K|X=x)} = \beta_{20} + \beta_2^T x \tag{66}$$

$$\cdots \tag{67}$$

$$\log \frac{Pr(G=K-1|X=x)}{Pr(G=K|X=x)} = \beta_{K-1,0} + \beta_{K-1}^T x \tag{68}$$

-

$$Pr(G=k|X=x) = \frac{\exp(\beta_{k0} + \beta_k^T x)}{1 + \sum_{\ell=1}^{K-1} \exp(\beta_{\ell 0} + \beta_\ell^T x)}, k = 1, \cdots, K-1 \tag{69}$$

$$Pr(G=K|X=x) = \frac{1}{1 + \sum_{\ell=1}^{K-1} \exp(\beta_{\ell 0} + \beta_\ell^T x)} \tag{70}$$

Fitting logistic regression model

- Parameter: $\theta = (\beta_{10}, \beta_1^T, \cdots, \beta_{(K-1)0}, \beta_{K-1}^T)$.
- Likelihood function

$$\ell(\theta) = \sum_{i=1}^{N} \log Pr(G = k | X = x_i; \theta). \tag{71}$$

- MLE estimates $\hat{\theta} = \arg\max_\theta \ell(\theta)$.
- Tools: NewtonRaphson algorithm, IRLS.
- L_1-regularzied MLE estimates

$$\hat{\theta}^{Lasso} = \arg\max_\theta \ell(\theta) - \lambda \sum_{k=1}^{K-1} \sum_{j=1}^{p} |\beta_{kj}|. \tag{72}$$

LDA or logistic regression

- Logistic regression is more general than LDA.
- $$Pr(X, G = k) = Pr(X)Pr(G = k|X), \tag{73}$$

  where $Pr(X)$ is the marginal density of X.
- Logistic regression: no assumption on $Pr(X)$, directly model $Pr(G = k|X)$ (*conditional likelihood*)
- LDA: model $Pr(X, G = k)$ for all k.
- For Gaussian data, Logistic regression lose about 30% efficiency. (Efron, 1975)
- Logistic regression: more robust to outliers.

Separating hyperplanes

- Motivation: both LDA and logistic regression are search for linear classifiers.
- How about we directly target on the decision boundary?
- For two dimensional x, search for $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ for

$$\{x : \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0\} \tag{74}$$

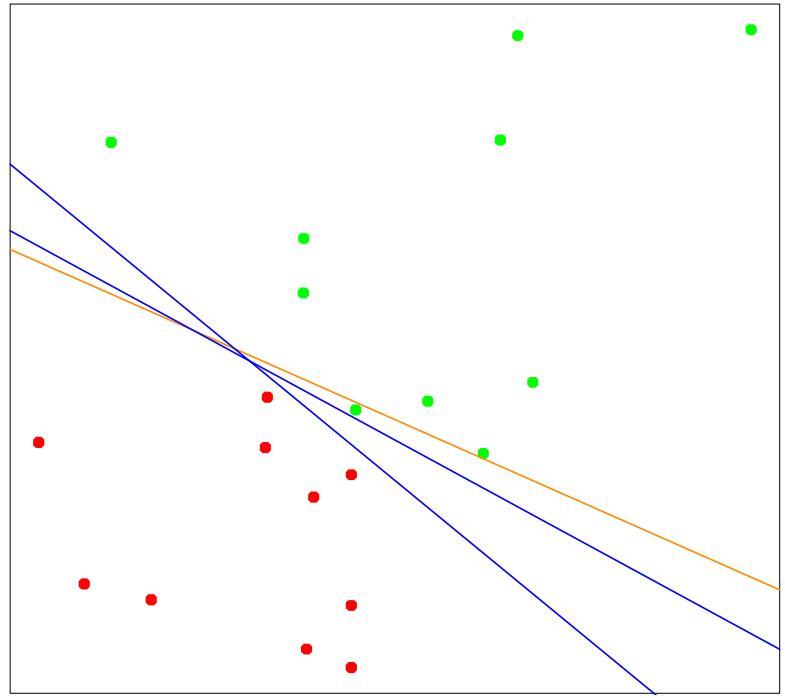


FIGURE 4.14. *A toy example with two classes separable by a hyperplane. The orange line is the least squares solution, which misclassifies one of the training points. Also shown are two blue separating hyperplanes found by the* perceptron learning algorithm *with different random starts.*

Perceptron Learning Algorithm

- Idea: find a separating hyperplane by minimizing the distance of misclassified points to the decision boundary.
- perceptron learning algorithm (Rosenblatt, 1958).
- Hyperplane L: $f(x) = \beta_0 + \beta^T x = 0$.

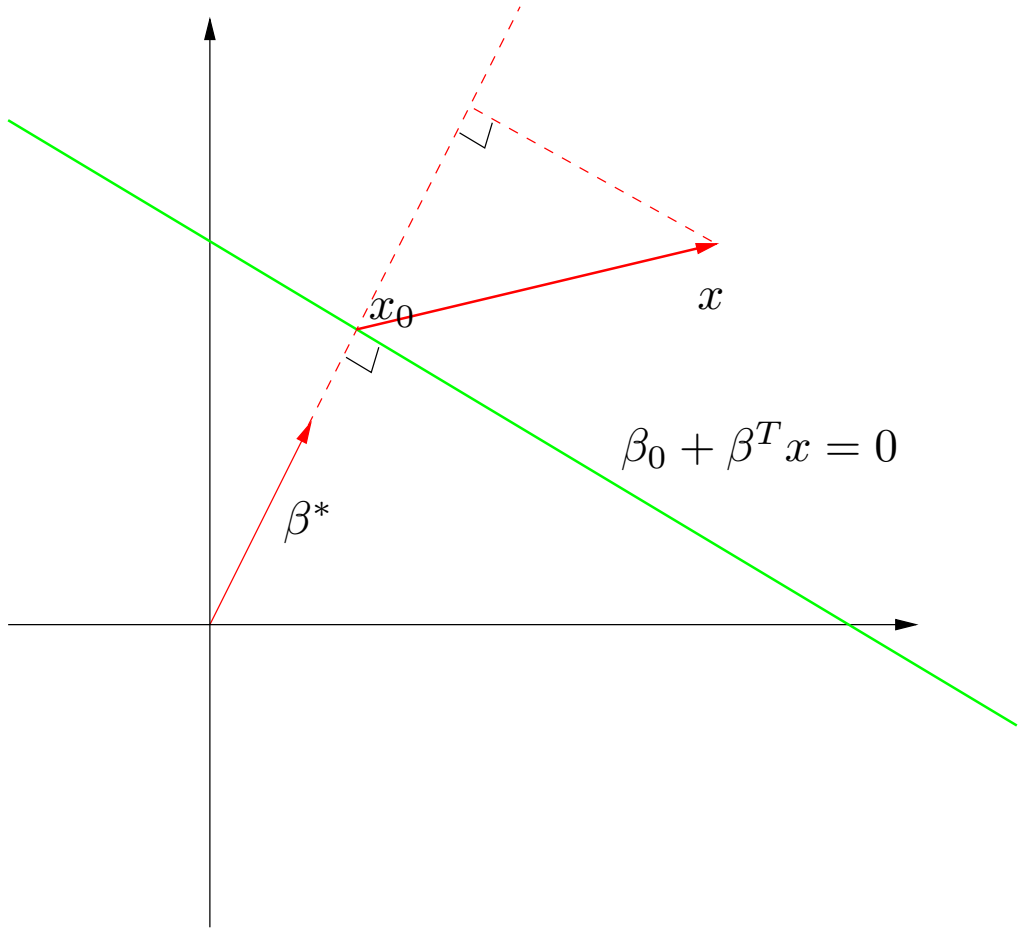


FIGURE 4.15. *The linear algebra of a hyperplane (affine set).*

Perceptron Learning Algorithm

- For any two points x_1, x_2 on L, $\beta^T(x_1 - x_2) = 0$. $\beta^* = \beta/\|\beta\|$ is the normalized vector orthogonal to L.
- The signed distance of x to L is

$$\beta^*(x - x_0) = \frac{1}{\|\beta\|}(\beta^T x + \beta_0). \tag{75}$$

- If $f(x) > 0$ set $y = 1$. Otherwise, set $y = -1$
- Minimize $D(\beta, \beta_0) = -\sum_{i \in \mathcal{M}} y_i(x_i^T \beta + \beta_0)$, where $\mathcal{M}$ is the misclassified point set.

Stochastic gradient descent

- Gradient is

$$\frac{\partial D(\beta, \beta_0)}{\partial \beta} = -\sum_{i \in \mathcal{M}} y_i x_i \tag{76}$$

$$\frac{\partial D(\beta, \beta_0)}{\partial \beta_0} = -\sum_{i \in \mathcal{M}} y_i \tag{77}$$

- Update (β, β_0) after observing more data, *online algorithm*.

$$\begin{bmatrix} \beta \\ \beta_0 \end{bmatrix} \leftarrow \begin{bmatrix} \beta \\ \beta_0 \end{bmatrix} + \rho \begin{bmatrix} y_i x_i \\ y_i, \end{bmatrix} \tag{78}$$

  where ρ is the learning rate. Set to 1 WLOG.
- Converge in finite number of steps if linearly separable.

Problem with the algorithm (Ripley, 1996)

- When the data are separable, there are many solutions, and which one is found depends on the starting values.
- The "finite" number of steps can be very large. The smaller the gap, the longer the time to find it.
- When the data are not separable, the algorithm will not converge, and cycles develop. The cycles can be long and therefore hard to detect.

Support Vector Machines (Vapnik and Chervonenkis, 1963)

- Idea: maximize the separation between two classes! i.e., maximize the distance to the closest point from either class.
- Benefit: leads to a unique solution, (usually) better performance on the test data.
- Formulation

$$\max_{\beta,\beta_0,\|\beta\|=1} M \tag{79}$$

$$\text{s. t. } y_i(x_i^T\beta + \beta_0) \geq M, i = 1, \cdots, N. \tag{80}$$

- M: margin/slab.

Equivalent formulation

- Get rid of the $\|\beta\| = 1$ constraint

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 \tag{81}$$

$$\text{s. t. } y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, \cdots, N. \tag{82}$$

- Lagrange function

$$L_P = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{N} \alpha_i[y_i(x_i^T\beta + \beta_0) - 1]. \tag{83}$$

- Lagrange dual problem:

$$\max_{\alpha} \min_{\beta,\beta_0} L_P \tag{84}$$

$$\text{s. t. } \alpha_i \geq 0, i = 1, \cdots, N \tag{85}$$

Support vectors

- The minimum of L_p can be explicitly calculated by taking derivatives with β and β_0.
- Setting derivatives to 0

$$\sum_{i=1}^{N} \alpha_i y_i x_i = \beta, \sum_{i=1}^{N} \alpha_i y_i = 0, \tag{86}$$

- Then we have the *Wolfe dual* L_D

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k \tag{87}$$

$$\text{s. t. } \alpha_i \geq 0 \text{ and } \sum_{i=1}^{N} \alpha_i y_i = 0. \tag{88}$$

Support points

From the Lagrange dual problem, we have $\alpha_i[y_i(x_i^T\beta+\beta_0)-1]=0$ for any i.

- If $\alpha>0$, $y_i(x_i^T\beta+\beta_0)-1=0$, x_i is on the boundary of the slab. Support vectors/points.
- If $y_i(x_i^T\beta+\beta_0)-1>0$, x_i is not on the boundary of the slab. $\alpha_i=0$.

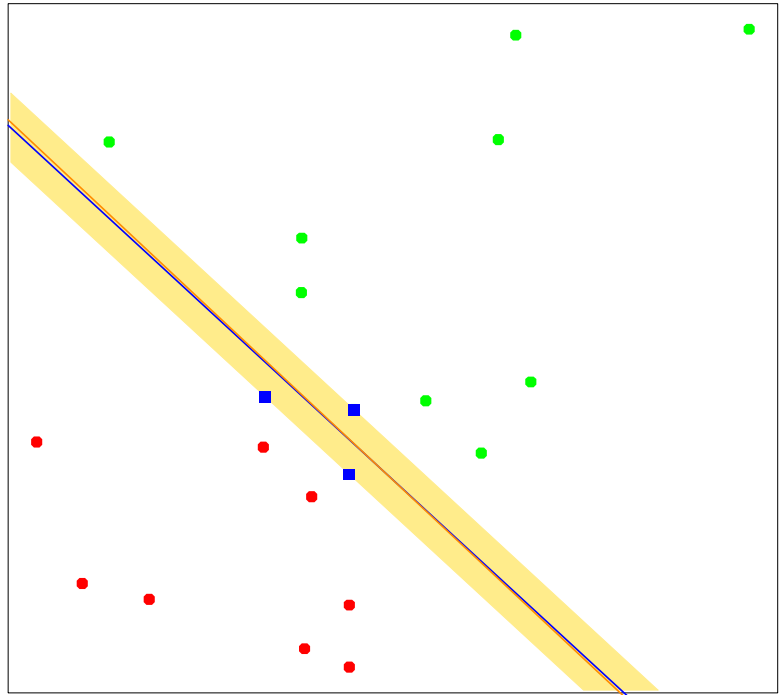


FIGURE 4.16. *The same data as in Figure 4.14. The shaded region delineates the maximum margin separating the two classes. There are three support points indicated, which lie on the boundary of the margin, and the optimal separating hyperplane (blue line) bisects the slab. Included in the figure is the boundary found using logistic regression (red line), which is very close to the optimal separating hyperplane (see Section 12.3.3).*

Properties

- Decision boundary focuses more on support vectors, more robust to model misspecification/outliers than say LDA.
- Identification of the support vectors requires the use of all data.
- The data needs to be separable, called *hard-margin* SVM.
- Extension to non-separable case later, *soft-margin* SVM.
- More on SVM in Chapter 12.

Outline

Beyond Linearity

- Main idea: Augment/replace the input X with additional variables, then use linear models.
- Denote by $h_m(X) : R^p \to R$ the mth transformation of X, $m = 1, \cdots, M$. Then

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X), \tag{89}$$

a *linear basis expansion* in X.

Common choices of h_m

- $h_m(X) = X_m$ for $m = 1, \cdots, p$.
- $h_m(X) = X_j^2$ or $X_j X_k$.
- $h_m(X) = \log(X_j), \sqrt{X_j}, \cdots,$
- $h_m(X) = I(L_m \leq X_k < U_m)$, indicator function for region $[L_m, U_m)$. Piecewise constant function.
- piecewise-polynomials
- splines
- wavelets

Control the complexity of the model

Usually, dictionary $\mathcal{D}$ contains a large number of basis functions.

- Restriction methods. Limit the class of functions. Say, impose additivity.
- Selection methods. lasso, CART, MARS, boosting.
- Regularization methods. Ridge regression, lasso.

Piecewise Polynomials and Splines

Assume X is one-dimensional.

- Piecewise constant basis functions: $h_1(X) = I(X < \xi_1)$, $h_2(X) = I(\xi_1 \leq X < \xi_2)$, $h_3(X) = I(\xi_2 \leq X)$. $f(X) = \sum_{m=1}^{3} \beta_m h_m(X)$ with estimates $\hat{\beta}_m = \bar{Y}_m$, the mean of Y in the mth region.
- Piecewise linear. Three additional basis functions: $h_{m+3}(X) = h_m(X)X$, for $m = 1, 2, 3$.
- Piecewise linear with continuity constraints at the two knots. $f(\xi_1^-) = f(\xi_1^+)$. four free parameters. More direct way: $h_1(X) = 1$, $h_2(X) = X$, $h_3(X) = (X - \xi_1)_+$, $h_4(X) = (X - \xi_2)_+$.

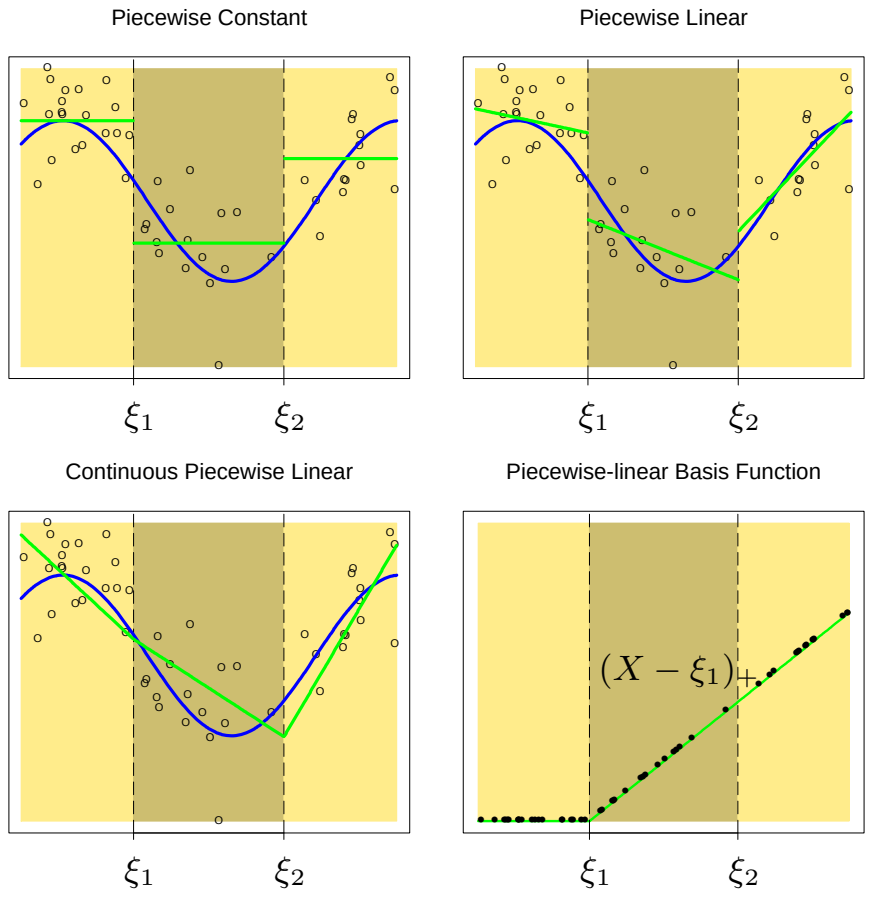


FIGURE 5.1. *The top left panel shows a piecewise constant function fit to some artificial data. The broken vertical lines indicate the positions of the two* knots *ξ_1 and ξ_2. The blue curve represents the true function, from which the data were generated with Gaussian noise. The remaining two panels show piecewise linear functions fit to the same data—the top right unrestricted, and the lower left restricted to be continuous at the knots. The lower right panel shows a piecewise–*

Piecewise cubic polynomials

Cubic spline: continuous, and has continuous first and second derivatives at the knots.

- $h_1(X) = 1$, $h_2(X) = X$, $h_3(X) = X^2$, $h_4(X) = X^3$, $h_5(X) = (X - \xi_1)^3_+$, $h_6(X) = (X - \xi_2)^3_+$.
- More generally, an order-M spline with knots $\xi_j, j = 1, \cdots, K$ is a piecewise-polynomial of order M, and has continuous derivatives up to order $M - 2$.
- Cubic spline: $M = 4$, piecewise constant: $M = 1$, continuous piecewise linear: $M = 2$. (Degree = order - 1).
- General truncated-power basis set is $h_j(X) = X^{j-1}, j = 1, \cdots, M$ and $h_{M+\ell} = (X - \xi_\ell)^{M-1}_+, \ell = 1, \cdots, K$.
- *regression splines*: select order of the spline, the number of the knots and the locations.

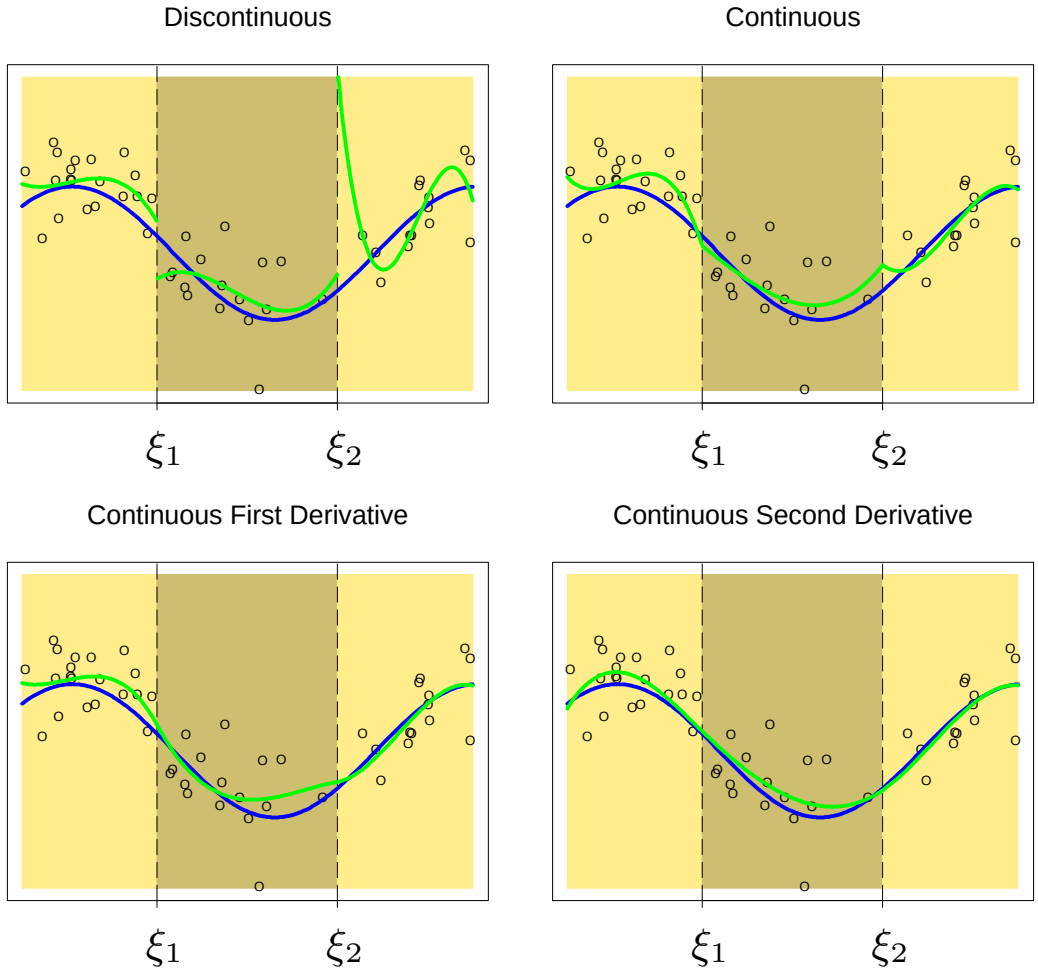


FIGURE 5.2. *A series of piecewise-cubic polynomials, with increasing orders of continuity.*

Natural Cubic Splines

- Regression splines: large variance near the boundaries as they are polynomials.
- Solution: *natural cubic spline*. Make the function linear beyond the boundary knots.
- Start with a basis for cubic splines and impose the boundary constraints.

$$N_1(X) = 1, N_2(X) = X, N_{k+2}(X) = d_k(X) - d_{k-1}(X), \qquad (90)$$

where $d_k(X) = \frac{(X-\xi_k)_+^3 - (X-\xi_K)_+^3}{\xi_K - \xi_k}$.

- Each basis function has zero second and third derivative for $X > \xi_K$.

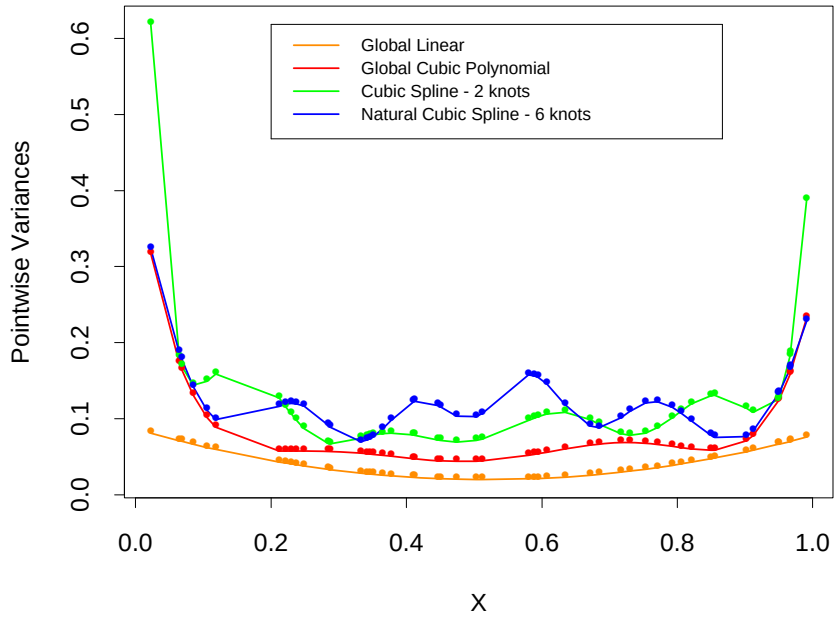


FIGURE 5.3. *Pointwise variance curves for four different models, with X consisting of 50 points drawn at random from $U[0, 1]$, and an assumed error model with constant variance. The linear and cubic polynomial fits have two and four degrees of freedom, respectively, while the cubic spline and natural cubic spline each have six degrees of freedom. The cubic spline has two knots at 0.33 and 0.66, while the natural spline has boundary knots at 0.1 and 0.9, and four interior knots uniformly spaced between them.*

Smoothing Splines

Avoid knot selection by using a maximal set of knots. Control the complexity by regularization.

$$RSS(f, \lambda) = \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2 dt, \tag{91}$$

where λ is the smoothing parameter.

- $\lambda = 0$: interpolation.
- $\lambda = \infty$: least square fit.

Solution

- The criterion is defined on an infinite-dimensional function space in fact, a Sobolev space of functions for which the second term is defined.
- Exists an explicit, finite-dimensional, unique minimizer which is a natural cubic spline with knots at the unique values of the x_i, $i = 1, ..., N$ (Exercise 5.7) (Could be less than N if there are multiple observations at the same x values).
- Write it as $f(x) = \sum_{j=1}^{N} N_j(x)\theta_j$, where $N_j(x)$ are an N-dimensional set of basis functions.
- Then we have

$$RSS(\theta, \lambda) = (\mathbf{y} - \mathbf{N}\theta)^T(\mathbf{y} - \mathbf{N}\theta) + \lambda\theta^T\Omega_N\theta, \tag{92}$$

with $\{\mathbf{N}\}_{ij} = N_j(x_i)$ and $\{\Omega_N\}_{jk} = \int N_j''(t)N_k''(t)df$.

Solution

- The solution to the minimization problem is

$$\hat{\theta} = (\mathbf{N}^T\mathbf{N} + \lambda\Omega_N)^{-1}\mathbf{N}^T\mathbf{y}, \tag{93}$$

  a generalized ridge regression.
- The fitted smoothing spline is

$$\hat{f}(x) = \sum_{i=1}^{N} N_j(x)\hat{\theta}_j. \tag{94}$$

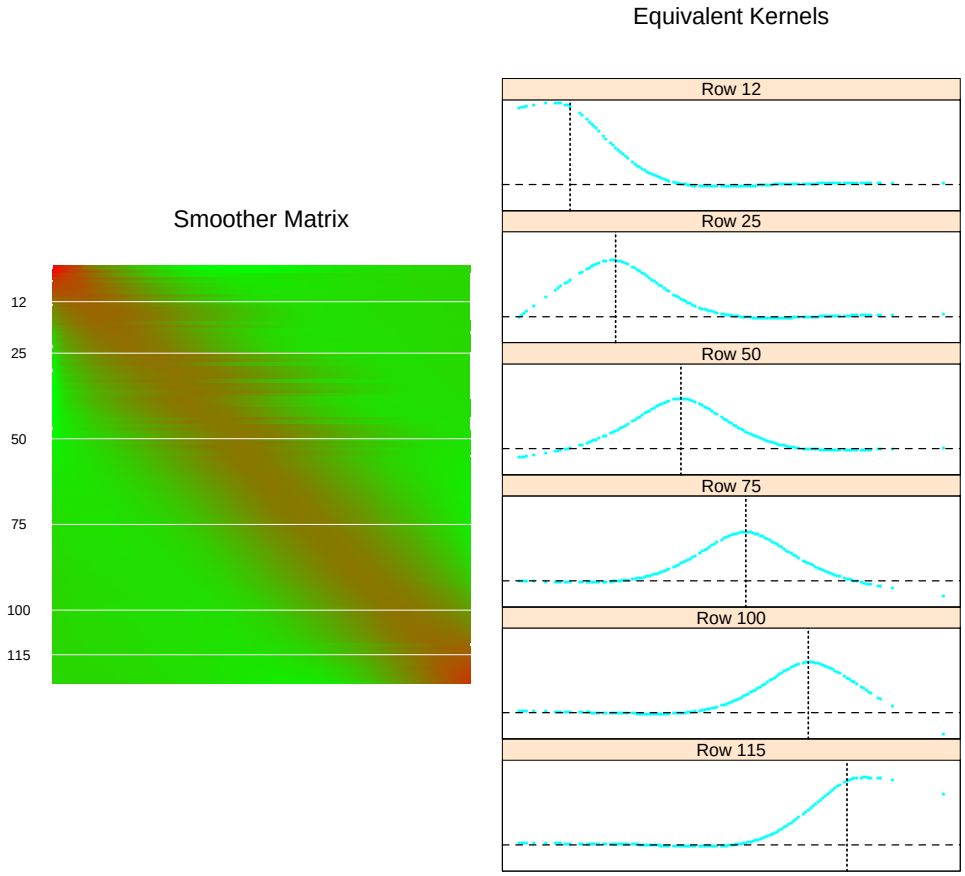


FIGURE 5.8. *The smoother matrix for a smoothing spline is nearly banded, indicating an equivalent kernel with local support. The left panel represents the elements of* $\mathbf{S}$ *as an image. The right panel shows the equivalent kernel or weighting function in detail for the indicated rows.*

Degrees of Freedom

A smoothing spline with a prespecified λ is an example of a linear smoother (as in linear operator).

$$\hat{\mathbf{f}} = \mathbf{N}(\mathbf{N}^T\mathbf{N} + \lambda\Omega_N)^{-1}\mathbf{N}^T\mathbf{y} =: \mathbf{S}_\lambda\mathbf{y}. \tag{95}$$

- Smoother matrix: $\mathbf{S}_\lambda$.
- Effective degrees of freedom $df_\lambda = Trace(\mathbf{S}_\lambda)$.
- $\mathbf{S}_\lambda$ symmetric and positive semidefinite. Write it in *Reinsch* form:

$$\mathbf{S}_\lambda = (\mathbf{I} + \lambda\mathbf{K})^{-1}. \tag{96}$$

where $\mathbf{K}$ does not depend on λ.

Equivalent form

- As a result, $\mathbf{f}$ will solve

$$\min_{\mathbf{f}} (\mathbf{y} - \mathbf{f})^T (\mathbf{y} - \mathbf{f}) + \lambda \mathbf{f}^T \mathbf{K} \mathbf{f}. \tag{97}$$

- Eigen-decomposition of $\mathbf{S}_\lambda$:

$$\mathbf{S}_\lambda = \sum_{k=1}^{N} \rho_k(\lambda) \mathbf{u}_k \mathbf{u}_k^T, \tag{98}$$

where $\rho_k(\lambda) = 1/(1 + \lambda d_k)$.

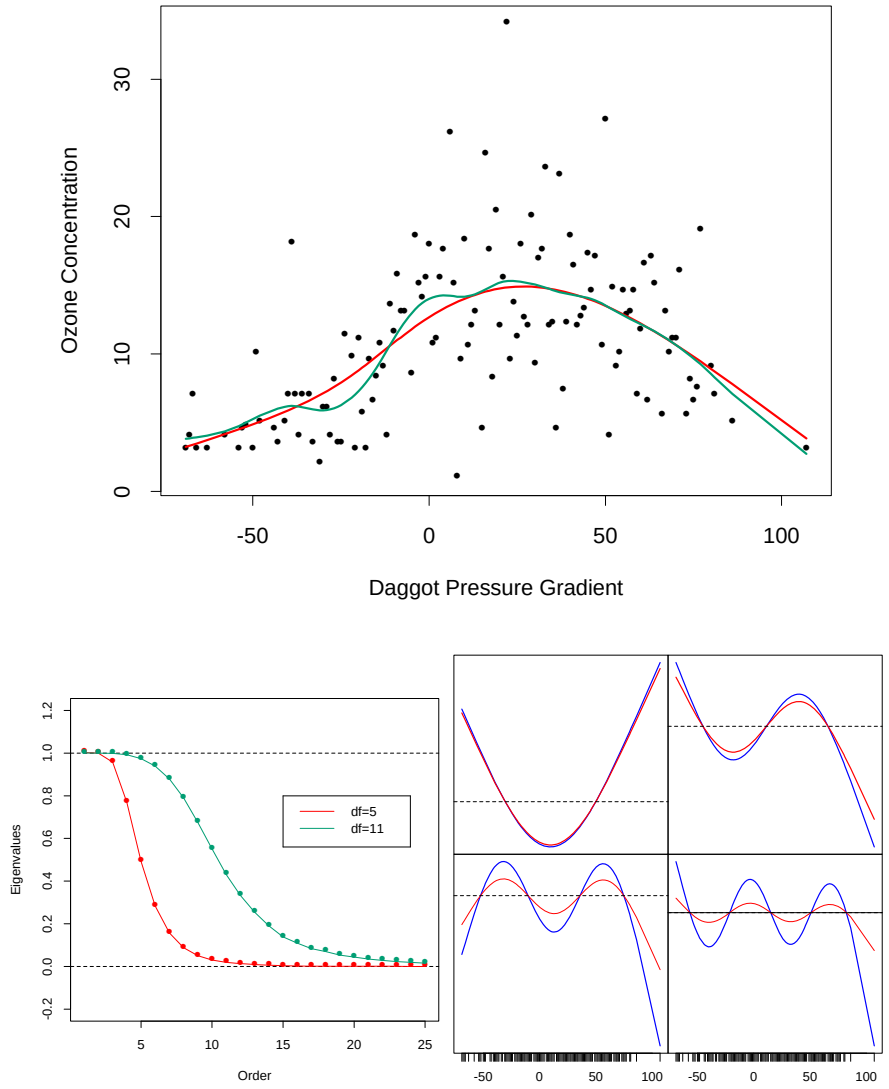


FIGURE 5.7. *(Top:) Smoothing spline fit of ozone concentration versus Daggot pressure gradient. The two fits correspond to different values of the smoothing parameter, chosen to achieve five and eleven effective degrees of freedom, defined by* $df_\lambda = trace(\mathbf{S}_\lambda)$. *(Lower*

Automatic Selection of the Smoothing Parameters

- Fixing the Degrees of Freedom. In R: smooth.spline(x,y,df=6).
- Bias-Variance Tradeoff.

Example:

$$Y = f(X) + \epsilon, \tag{99}$$

$$f(X) = \frac{\sin(12(X + 0.2))}{X + 0.2}, \tag{100}$$

with $X \sim U[0, 1]$ and $\epsilon \sim N(0, 1)$. Training sample size $N = 100$.

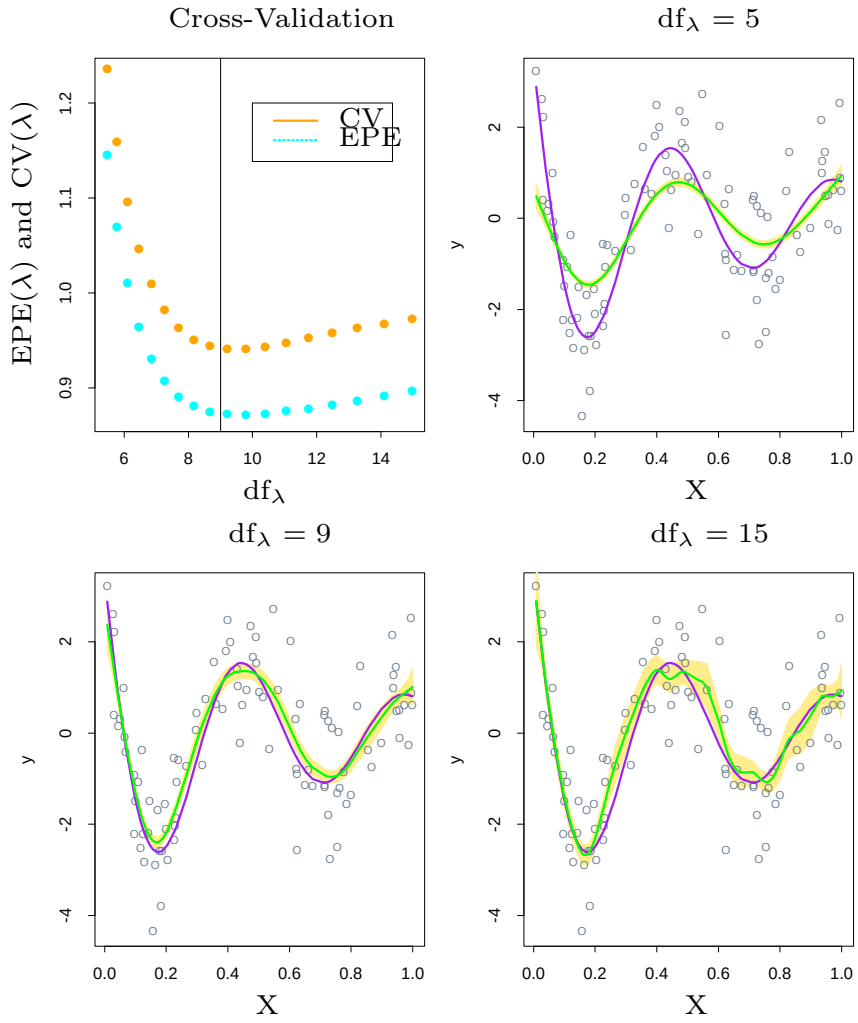


FIGURE 5.9. *The top left panel shows the* $\mathrm{EPE}(\lambda)$ *and* $\mathrm{CV}(\lambda)$ *curves for a realization from a nonlinear additive error model (5.22). The remaining panels show the data, the true functions (in purple), and the fitted curves (in green) with yellow shaded* $\pm 2\times$ *standard error bands, for three different values of* df_λ.

integrated squared prediction error (EPE)

combines both bias and variance in a single summary:

$$EPE(\hat{f}_\lambda) = E(Y - \hat{f}_\lambda(X))^2 \tag{101}$$

$$= Var(Y) + E[Bias^2(\hat{f}_\lambda(X)) + Var(\hat{f}_\lambda(X))] \tag{102}$$

$$= \sigma^2 + MSE(\hat{f}_\lambda) \tag{103}$$

- EPE can not be calculated, but we can use K-fold CV, GCV and C_p to estimate the value.

Nonparametric Logistic Regression

Borrow the idea from smoothing splines. Assume

$$Pr(Y = 1|X = x) = \frac{e^{f(x)}}{1 + e^{f(x)}}. \tag{104}$$

Penalized log-likelihood

$$\ell(f; \lambda) = \sum_{i=1}^{N} [y_i f(x_i) - \log(1 + \exp(f(x_i))] - \frac{1}{2}\lambda \int \{f''(t)\}^2 dt. \tag{105}$$

The maximizer is finite-dimensional natural spline with knots at the unique values of x. We can write $f(x) = \sum_{j=1}^{N} N_j(x)\theta_j$. Use Newton-Raphson to update.

Multidimensional Splines

Suppose $X \in R^2$. We have basis of functions $h_{1k}(X_1), k = 1, \ldots, M_1$, $h_{2k}(X_2), k = 1, \ldots, M_2$. Define the $M_1 \times M_2$ dimensional tensor product basis

$$g_{jk}(X) = h_{1j}(X_1)h_{2k}(X_2), j = 1, \cdots, M_1, k = 1, \cdots, M_2. \tag{106}$$

Two-dimensional function:

$$g(X) = \sum_{j=1}^{M_1} \sum_{k=1}^{M_2} \theta_{jk} g_{jk}(X). \tag{107}$$

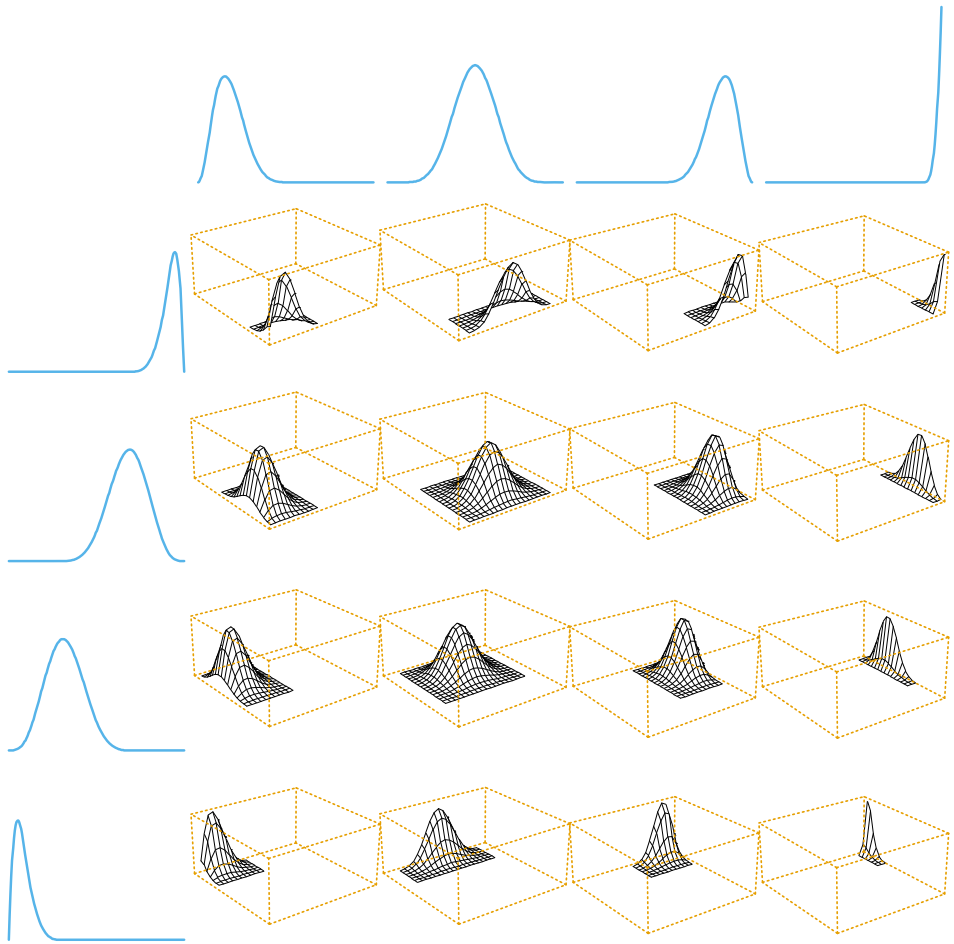
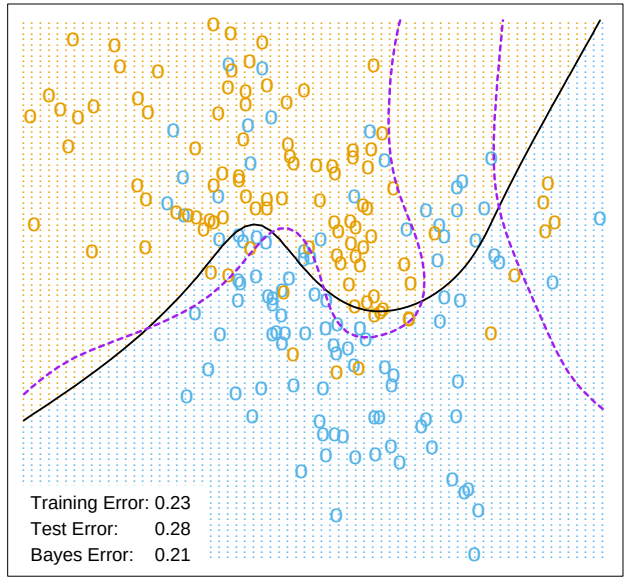


FIGURE 5.10. *A tensor product basis of B-splines, showing some selected pairs. Each two-dimensional function is the tensor product of the corresponding one dimensional marginals.*



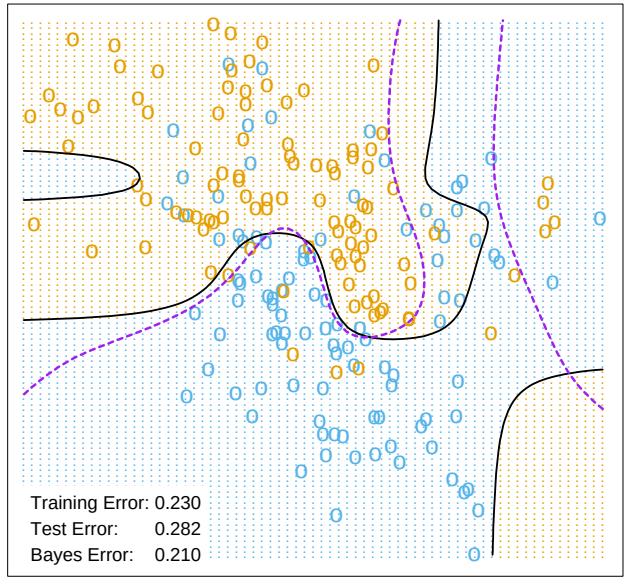


FIGURE 5.11. *The simulation example of Figure 2.1. The upper panel shows the decision bound-*

Penalized Spline

Regularization problem

$$\min_f \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda J[f], \tag{108}$$

J is a penalty functional. Extension of the one-dimensional roughness penalty to two-dimensional surface:

$$J[f] = \int\int_{R^2} \left[\left(\frac{\partial^2 f(x)}{\partial x_1^2}\right)^2 + 2\left(\frac{\partial^2 f(x)}{\partial x_1 \partial x_2}\right)^2 + \left(\frac{\partial^2 f(x)}{\partial x_2^2}\right)^2 \right] dx_1 dx_2. \tag{109}$$

Solution: thin-plate spline.

Properties

- $\lambda \to 0$, the solution approaches an interpolating function.
- $\lambda \to \infty$, the solution approaches the least squares plane.
- For intermediate values of λ, the solution can be represented as a linear expansion of basis functions, whose coefficients are obtained by a form of generalized ridge regression.

$$f(x) = \beta_0 + \beta^T x + \sum_{j=1}^{N} \alpha_j h_j(x), \tag{110}$$

where $h_j(x) = \|x - x_j\|^2 \log \|x - x_j\|$. (Radial Basis Functions).

Hilbert Space

Real or complex inner product space and also a complete metric space with respect to the distance function induced by the inner product.

- Inner product $< \cdot, \cdot >: X \times X \to F$.
  - Conjugate symmetry
  - Linearity
  - Positive-definitiveness.
- Complete metric space: Cauchy sequence of points in X has a limit in X. Distance function: $d(x, y) = \sqrt{< x - y, x - y >}$.

Reproducing Kernel Hilbert Space (RKHS)

- Reproducing Kernel Hilbert Space a Hilbert space of functions in which point evaluation is a continuous linear functional.
- Let X be an arbitrary set and $\mathcal{H}$ a Hilbert space of real-valued functions on X.
- The evaluation functional over the Hilbert space of functions $\mathcal{H}$ is a linear functional that evaluates each function at a point x,

$$L_x : f \mapsto f(x) \; \forall f \in \mathcal{H}. \tag{111}$$

- We say that $\mathcal{H}$ is a reproducing kernel Hilbert space if L_x is continuous at any f in $\mathcal{H}$ or, equivalently, if for all x in X, L_x is a bounded operator on $\mathcal{H}$, i.e. there exists some $M > 0$ such that

$$L_x(f) := f(x) \leq M\|f\|_H \; \forall f \in \mathcal{H}. \tag{112}$$

Reproducing property

- The Riesz representation theorem implies that for all x in X there exists a unique element K_x of $\mathcal{H}$ with the reproducing property,

$$f(x) = L_x(f) = \langle f,\ K_x \rangle \quad \forall f \in \mathcal{H}. \tag{113}$$

- Since K_y is itself a function in $\mathcal{H}$ we have that for each y in X

$$K_y(x) = \langle K_y,\ K_x \rangle_{\mathcal{H}}. \tag{114}$$

- This allows us to define the reproducing kernel of $\mathcal{H}$ as a function $K : X \times X \to \mathbb{R}$ by $K(x, y) = \langle K_x,\ K_y \rangle_{\mathcal{H}}$.
- From this definition it is easy to see that $K : X \times X \to \mathbb{R}$ is both symmetric and positive definite, i.e.

$$\sum_{i,j=1}^{n} c_i c_j K(x_i, x_j) \geq 0 \ \forall n \in \mathbb{N}, x_1, \ldots, x_n \in X, \text{ and } c_1, \ldots, c_n \in \mathbb{R}. \tag{115}$$

Reproducing Kernel Hilbert Space (RKHS)

- The Moore-Aronszajn theorem: every symmetric, positive definite kernel defines a unique reproducing kernel Hilbert space.

Theorem

Suppose K is a symmetric, positive definite kernel on a set X. Then there is a unique Hilbert space of functions on X for which K is a reproducing kernel.

- Positive definite kernel K with eigen-decomposition

$$K(x,y) = \sum_{i=1}^{\infty} \gamma_i \phi_i(x) \phi_i(y), \tag{116}$$

with $\gamma_i \geq 0$ and $\sum_{i=1}^{\infty} \gamma_i^2 < \infty$, $\phi_i(x)$: eigen-functions.

- Then we have the decomposition for functions in $\mathcal{H}_K$:

$$f(x) = \sum_{i=1}^{\infty} c_i \phi_i(x), \|f\|_{\mathcal{H}_k}^2 := \sum_{i=1}^{\infty} c_i^2/\gamma_i < \infty. \tag{117}$$

Check the property

- $$f(x) = \sum_{i=1}^{\infty} c_i \phi_i(x), g(x) = \sum_{i=1}^{\infty} d_i \phi_i(x), \quad (118)$$
- Define inner product
$$< f(x), g(x) >_{\mathcal{H}_K} := \sum_{i=1}^{\infty} c_i d_i / \gamma_i. \quad (119)$$
- Verify the reproducing property
$$< f(y), K(y, x) >_{\mathcal{H}_K} = \sum_{i=1}^{\infty} \frac{c_i \gamma_i \phi_i(x)}{\gamma_i} = \sum_{i=1}^{\infty} c_i \phi_i(x) = f(x). \quad (120)$$

Tikhonov Regularization

$$\min_{f \in \mathcal{H}_k} \left[\sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|f\|^2_{\mathcal{H}_k} \right]. \tag{121}$$

Using the expansion.

$$\min_{\{c_j\}_0^\infty} \left[\sum_{i=1}^{N} L(y_i, \sum_{i=1}^{\infty} c_i \phi_i(x)) + \lambda \sum_{i=1}^{\infty} c_i^2/\gamma_i \right]. \tag{122}$$

Representer Theorem leads to Finite-dimensional solution: (Wahba, 1990, Ex 5.15)

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x, x_i). \tag{123}$$

Penalized Polynomial Regression

Kernel $K(x, y) = (< x, y > +1)^d$: polynomial kernel. For $p = 2, d = 2$,

$$K(x, y) = 1 + 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 \tag{124}$$

$$= \sum_{m=1}^{M} h_m(x)h_m(y), \tag{125}$$

with $h(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)^T$.

Other Basis Functions

- Gaussian Kernel $K(x, y) = \exp\{-\nu\|x - y\|^2\}$ leads to the Gaussian Radial Basis Functions

$$k_m(x) = e^{-\nu\|x - x_m\|^2}, m = 1, \cdots, N, \tag{126}$$

centered at one of the training feature vectors x_m.

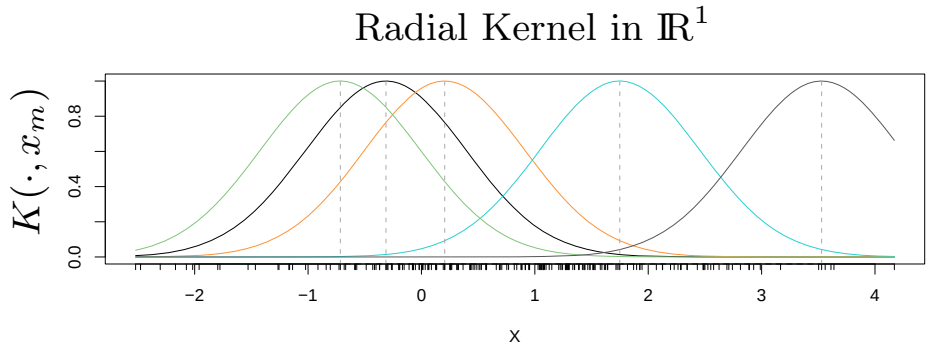


FIGURE 5.13. *Radial kernels $k_k(x)$ for the mixture data, with scale parameter $\nu = 1$. The kernels are centered at five points x_m chosen at random from the 200.*

Wavelet Smoothing

- Splines basis functions: smooth function.
- Wavelets basis functions: can capture both smooth and locally bumpy functions.
- Use orthonormal basis functions, and shrink the coefficients towards a sparse representation.

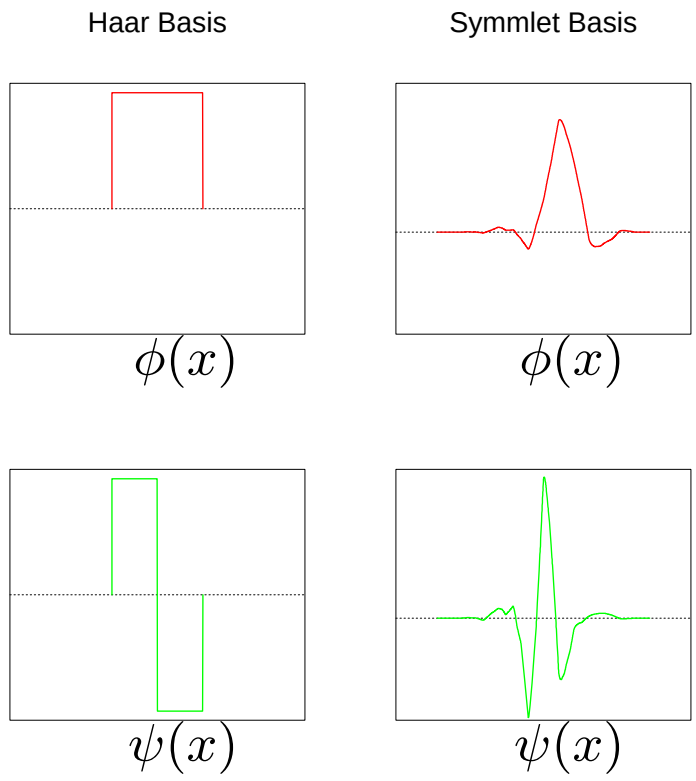


FIGURE 5.18. *The* Haar *and* symmlet *father (scaling) wavelet* $\phi(x)$ *and mother wavelet* $\psi(x)$.

Wavelet Bases and the Wavelet Transform

Wavelet bases are generated by translations and dilations of a single scaling function $\phi(x)$ (also known as the father).

- Haar basis: piecewise-constant representation.
- $\phi(x) = I(x \in [0, 1])$.
- Translation $\phi_{0,k}(x) = \phi(x - k)$. Space spanned by $\phi_{0,k}(x), k \in \mathcal{Z}$: reference space V_0.
- Dilations $\phi_{1,0}(x) = \sqrt{2}\phi(2x)$.
- In general, we have $\phi_{j,k} = 2^{j/2}\phi(2^j x - k)$. Space V_j.
- $\cdots \supset V_1 \supset V_0 \supset V_1 \supset \cdots$.

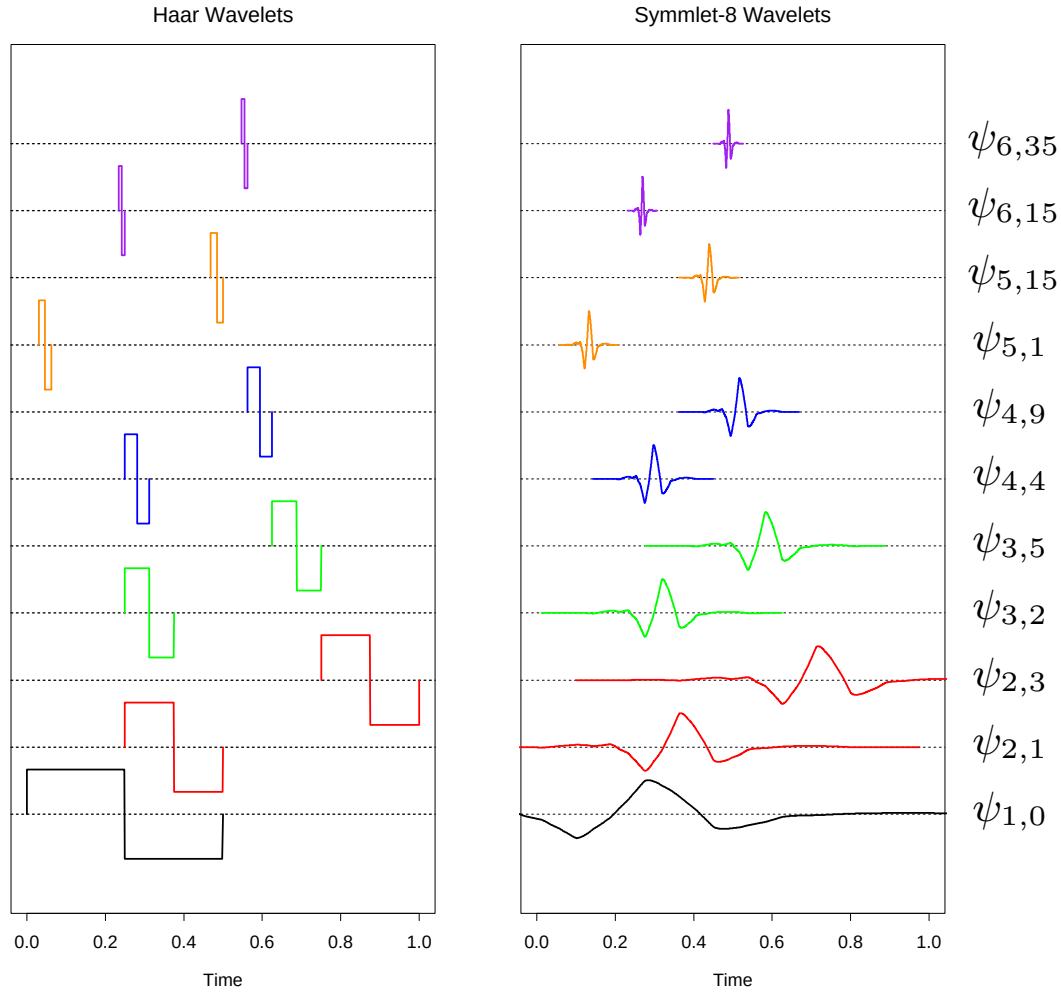


FIGURE 5.16. *Some selected wavelets at different translations and dilations for the Haar and symmlet families. The functions have been scaled to suit the display.*

Wavelet decomposition

We can write $V_{j+1} = V_j \oplus W_j$, where W_j represent the *detail*. Consider function $\psi(x-k)$ generated by the *mother wavelet* $\psi(x) = \phi(2x) - \phi(2x-1)$.

- $\psi_{j,k} = 2^{j/2}\psi(2^j x - k), k \in \mathcal{Z}$ form an orthonormal basis for W_j.
- Since $V_{j+1} = V_j \oplus W_j = V_{j-1} \oplus W_{j-1} \oplus W_j$, we have $V_J = V_0 \oplus W_0 \oplus W_1 \cdots \oplus W_{J-1}$. *multiresolution analysis*.
- Daubechies symmlet-8 basis: support covering 15 consecutive time intervals (Haar basis covers only 1). Smoother than Haar.
- symmlet-p wavelet $\psi(x)$ has p vanishing moments

$$\int \psi(x) x^j dx = 0, j = 0, \cdots, p-1. \tag{127}$$

Adaptive Wavelet Filtering

Suppose $\mathbf{y}$ is the response vector on $N = 2^J$ lattice-points. $\mathbf{W}$ is the $N \times N$ orthonormal wavelet basis matrix evaluated at the N uniformly spaced observations.

- $\mathbf{y}^* = \mathbf{W}^T\mathbf{y}$ is called *wavelet transform* of $\mathbf{y}$.
- SURE Shrinkage (Stein's Unbiased Risk Estimation, Donoho and Johnstone, 1994).

$$\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{W}\boldsymbol{\theta}\|_2^2 + 2\lambda\|\boldsymbol{\theta}\|_1. \tag{128}$$

- Since $\mathbf{W}$ is orthonormal, $\hat{\theta}_j = \text{sign}(y_j^*)(|y_j^*| - \lambda)_+$.
- Then, the fitted function is given by *inverse wavelet transform* $\hat{\mathbf{f}} = \mathbf{W}\hat{\boldsymbol{\theta}}$.

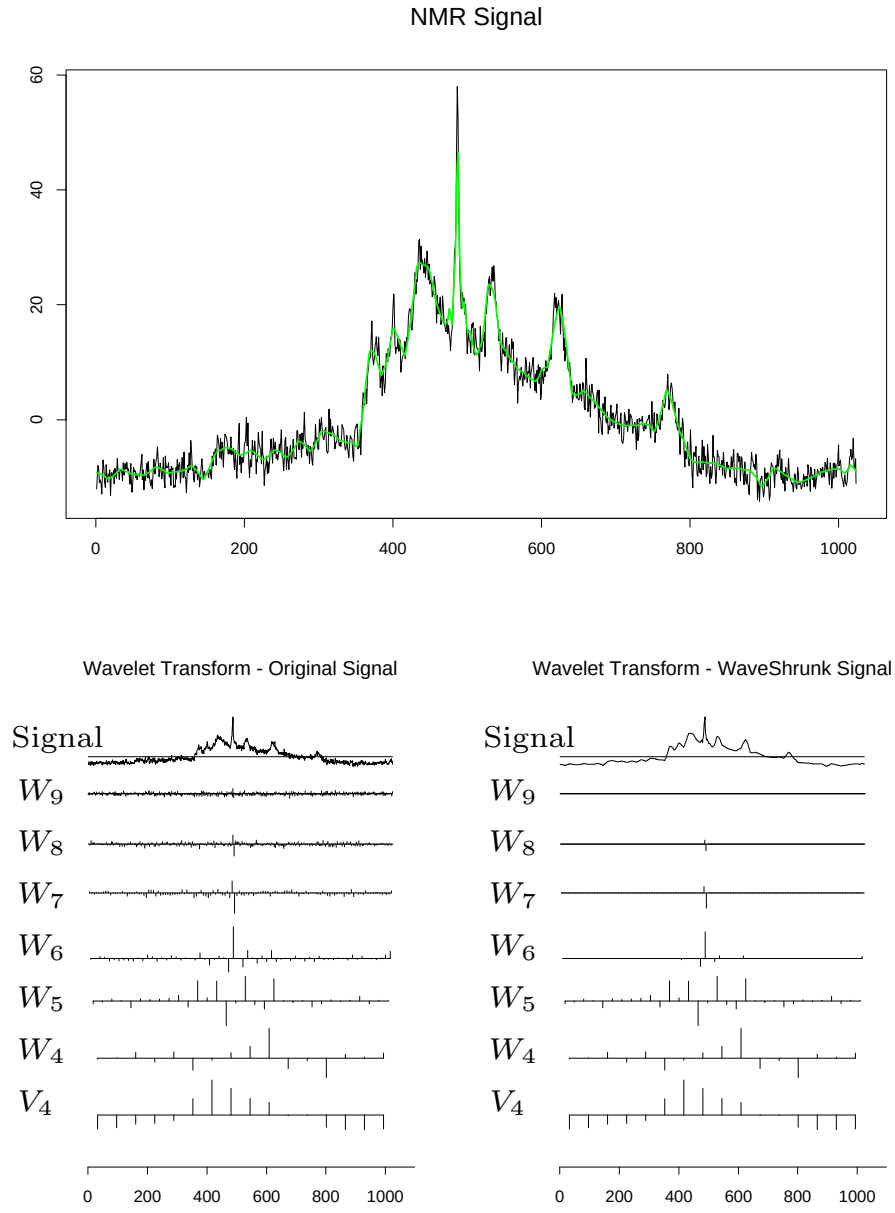


FIGURE 5.17. *The top panel shows an NMR signal, with the wavelet-shrunk version superimposed in green. The lower left panel represents the wavelet trans-*

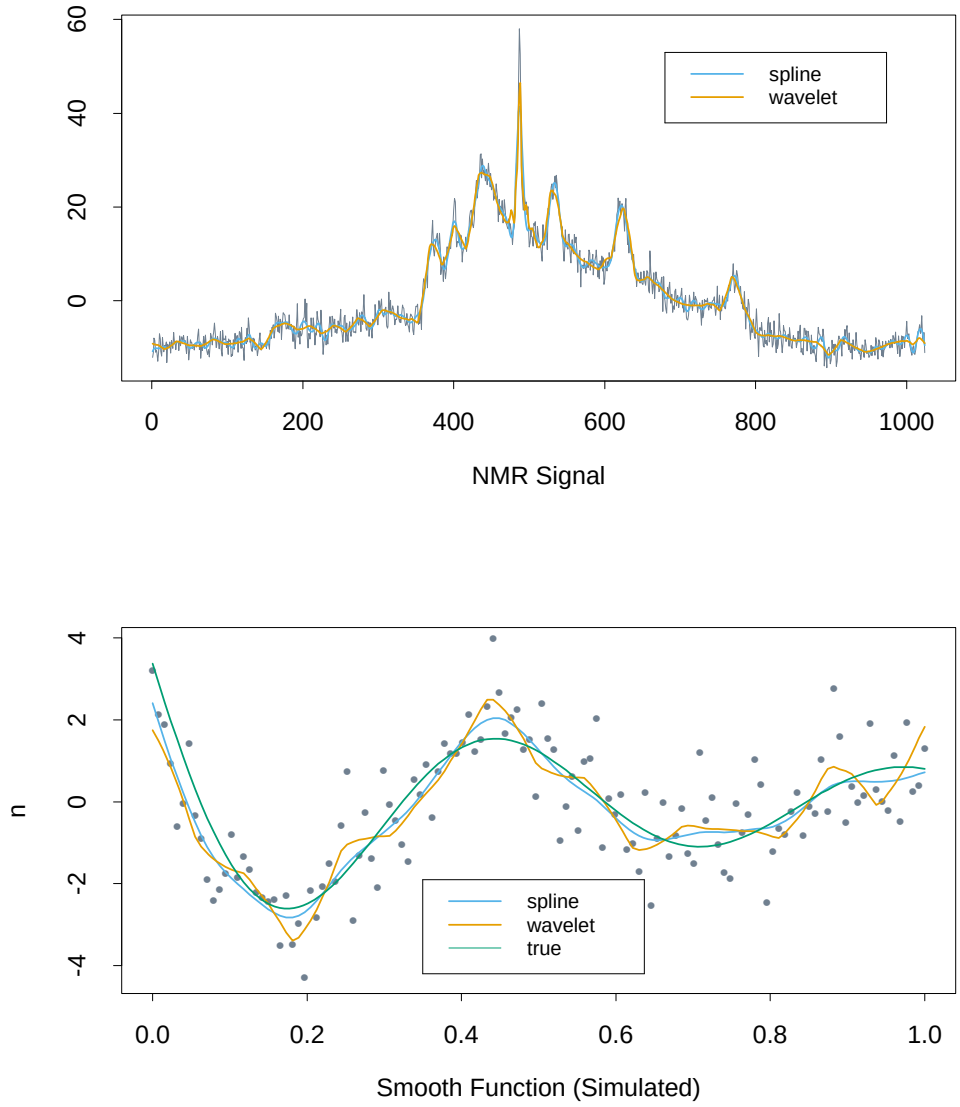


FIGURE 5.19. *Wavelet smoothing compared with smoothing splines on two examples. Each panel compares the SURE-shrunk wavelet fit to the cross-validated smoothing spline fit.*

Wavelet Shrinkage vs. Smoothing splines

- SURE Shrinkage: L1 regularization, shrinkage and selection, leads to sparsity.
- Smoothing splines: L2 regularization, only shrinkage.
- wavelet transform has efficient calculation, $O(N)$ using pyramidal schemes.

Outline

Kernel Smoother

Recall the k-nearest-neighbor method

$$\hat{f}(x) = \text{Ave}(y_i | x_i \in N_k(x)) \tag{129}$$

as an estimate of the regression function $f(x) = E(Y|X = x)$.

- $N_k(x)$: the set of k points nearest to x in square distance.
- Approximation to conditional expectation.
- Discontinuous function. Not desirable.
- Idea: assign weights that die off smoothly with distance from the target point $\rightarrow$ Nadaraya-Watson kernel-weighted average.

$$\hat{f}(x) = \frac{\sum_{i=1}^{N} K_\lambda(x, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x, x_i)}. \tag{130}$$

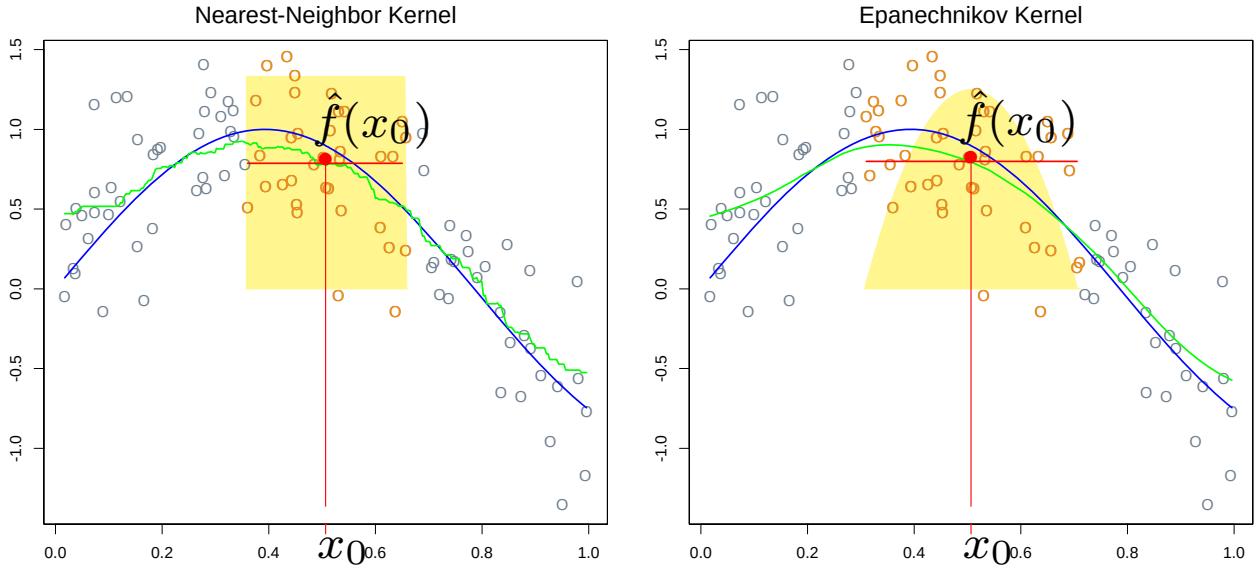


FIGURE 6.1. *In each panel* 100 *pairs* x_i, y_i *are generated at random from the blue curve with Gaussian errors:* $Y = \sin(4X) + \varepsilon$, $X \sim U[0,1]$, $\varepsilon \sim N(0, 1/3)$. *In the left panel the green curve is the result of a* 30*-nearest-neighbor running-mean smoother. The red point is the fitted constant* $\hat{f}(x_0)$, *and the red circles indicate those observations contributing to the fit at* x_0. *The solid yellow region indicates the weights assigned to observations. In the right panel, the green curve is the kernel-weighted average, using an Epanechnikov kernel with (half) window width* $\lambda = 0.2$.

Nadaraya-Watson Estimate

$$\hat{f}(x) = \frac{\sum_{i=1}^{N} K_\lambda(x, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x, x_i)}. \tag{131}$$

- $K(\cdot, \cdot)$: Kernel function, control the decay pattern. Epanechnikov quadratic kernel:

$$K_\lambda(x, x_i) = D(\frac{|x - x_i|}{\lambda}), \tag{132}$$

  where $D(t) = 3/4(1 - t^2)1\{|t| \leq 1\}$.
- λ, smoothing parameter, determines the width of the local neighborhood

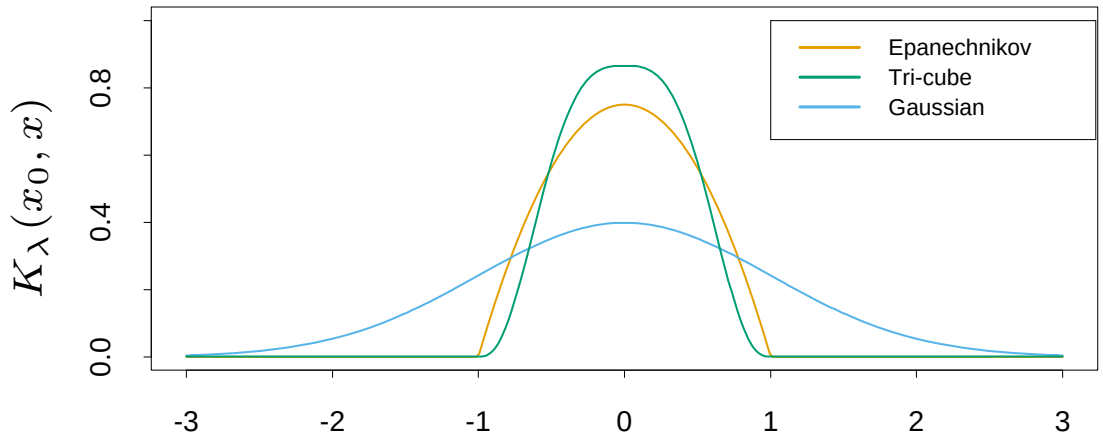


FIGURE 6.2. *A comparison of three popular kernels for local smoothing. Each has been calibrated to integrate to* 1*. The tri-cube kernel is compact and has two continuous derivatives at the boundary of its support, while the Epanechnikov kernel has none. The Gaussian kernel is continuously differentiable, but has infinite support.*

Variable Bandwidth

We can use $h_\lambda(x_0)$ for the bandwidth function for the width of the neighborhood at x_0.

$$K_\lambda(x_0, x) = D\left(\frac{|x - x_i|}{h_\lambda(x_0)}\right). \tag{133}$$

- Larger λ: lower variance, higher bias
- Boundary issue: less points on the boundaries for N-W estimator; wider range for nearest-neighbor methods.

Local Linear Regression

- Reduce the bias at the boundary compared with local constant regression.
- Solves a separate weighted least squares problem at each target point x_0:

$$\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i)[y_i - \alpha(x_0) - \beta(x_0)x_i]^2. \tag{134}$$

- Only use the above fit to evaluate at single point x_0.
- It is a linear estimator, $\hat{f}(x_0) = \sum_{i=1}^{N} l_i(x_0)y_i$.
- $l_i(x_0)$: *equivalent kernel*.
- Local linear regression automatically modifies the kernel to correct the bias exactly to first order, a phenomenon dubbed as *automatic kernel carpentry*.

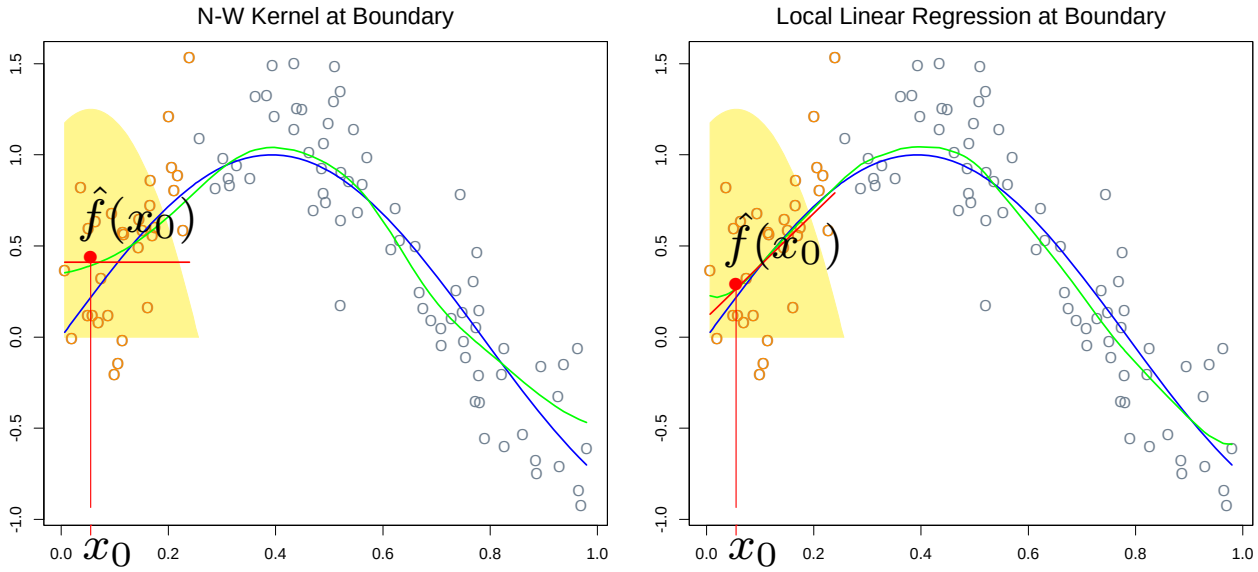


FIGURE 6.3. *The locally weighted average has bias problems at or near the boundaries of the domain. The true function is approximately linear here, but most of the observations in the neighborhood have a higher mean than the target point, so despite weighting, their mean will be biased upwards. By fitting a locally weighted linear regression (right panel), this bias is removed to first order*

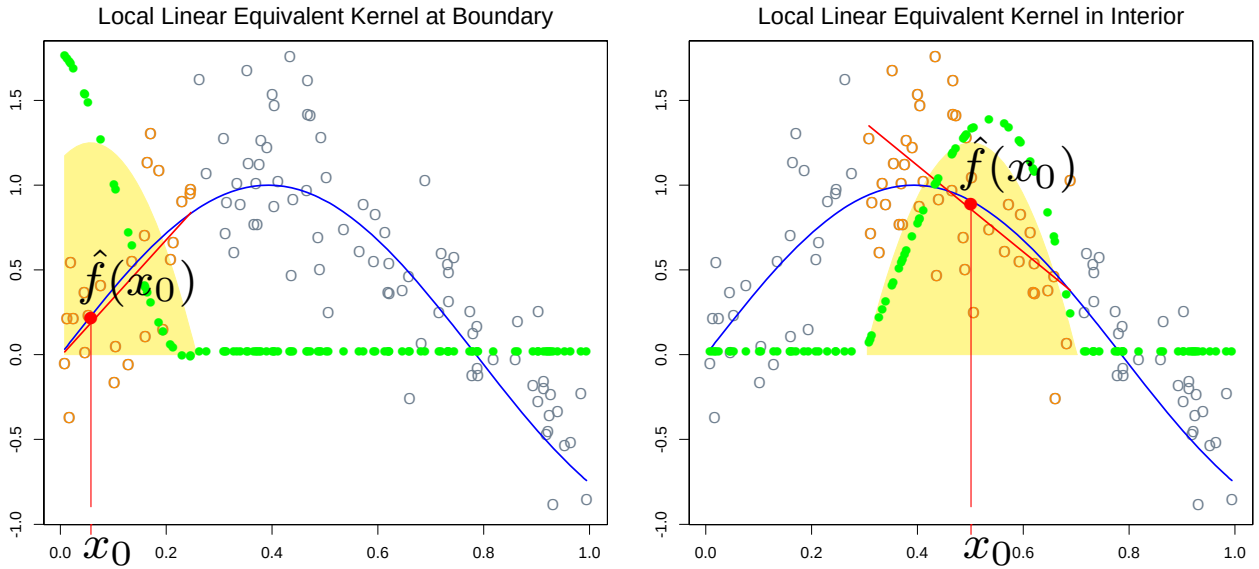


FIGURE 6.4. *The green points show the equivalent kernel $l_i(x_0)$ for local regression. These are the weights in $\hat{f}(x_0) = \sum_{i=1}^{N} l_i(x_0)y_i$, plotted against their corresponding x_i. For display purposes, these have been rescaled, since in fact they sum to 1. Since the yellow shaded region is the (rescaled) equivalent kernel for the Nadaraya–Watson local average, we see how local regression automatically modifies the weighting kernel to correct for biases due to asymmetry in the smoothing window.*

Local Polynomial Regression

Natural extension to higher orders

$$\min_{\alpha(x_0),\beta_j(x_0),j=1,\cdots,d} \sum_{i=1}^{N} K_\lambda(x_0,x_i)[y_i-\alpha(x_0)-\sum_{j=1}^{d}\beta_j(x_0)x_i^j]^2. \qquad (135)$$

- Solution: $\hat{f}(x_0)=\hat{\alpha}(x_0)+\sum_{j=1}^{d}\hat{\beta}_j(x_0)x_0^j$
- The bias will only have components of degree $d+1$ and higher.
- Larger variance compared with the local linear regression.

Key reference: Fan and Gijbels (1996), Local Polynomial Modelling and Its Applications.

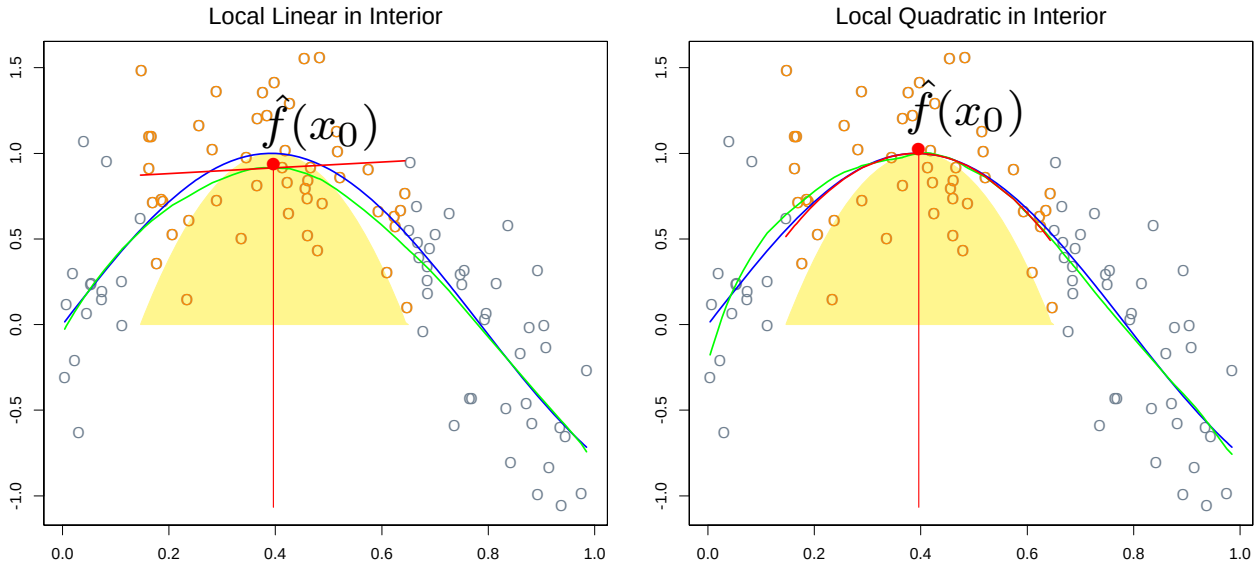


FIGURE 6.5. *Local linear fits exhibit bias in regions of curvature of the true function. Local quadratic fits tend to eliminate this bias.*

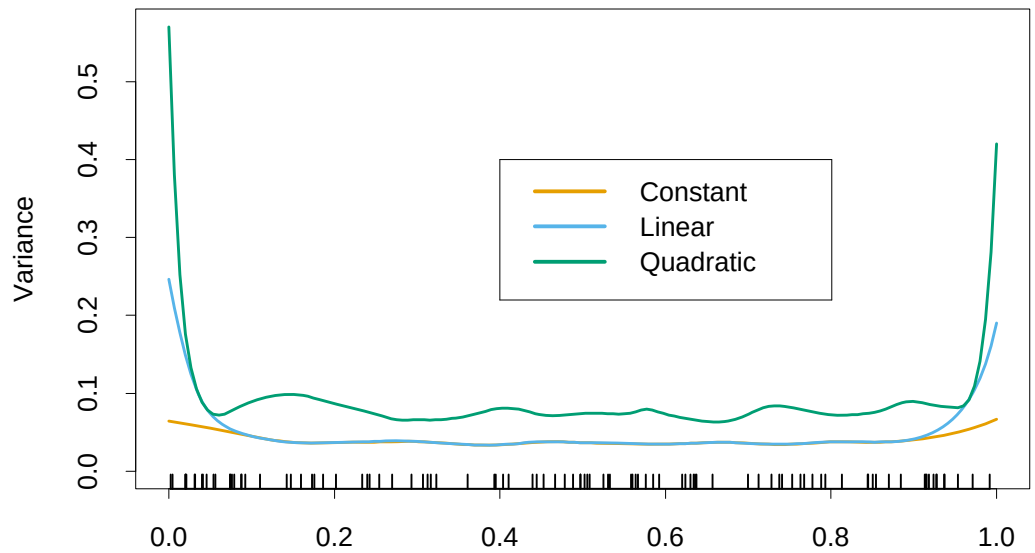


FIGURE 6.6. *The variances functions* $||l(x)||^2$ *for local constant, linear and quadratic regression, for a metric bandwidth* ($\lambda = 0.2$) *tri-cube kernel.*

Selecting the Width of the Kernel

- For the Epanechnikov or tri-cube kernel with metric width, λ is the radius of the support region.
- For the Gaussian kernel, λ is the standard deviation.
- λ is the number k of nearest neighbors in k-nearest neighborhoods, often expressed as a fraction or span k/N of the total training sample.

As usual, larger λ leads to larger bias and smaller variance. Use Cross-validation to choose λ.

Local Regression in $\mathbb{R}^p$

Easily generalized to higher dimensions. Fo each $x_0 \in \mathbb{R}^p$, we solve

$$\min_{\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - b(x_i)^T \beta(x_0))^2, \tag{136}$$

where $K_\lambda(x_0, x) = D\left(\frac{\|x - x_0\|}{\lambda}\right)$.
NOTE: boundary effects for serious for higher dimensional problems.
Curse-of-dimensionality.

Structured Local Regression Models in $\mathbb{R}^p$.

Use a positive semidefinite matrix A to weigh different coordinates:

$$K_{\lambda,A}(x_0, x) = D\left(\frac{(x - x_0)^T A(x - x_0)}{\lambda}\right). \tag{137}$$

- Impose restrictions on A to remove coordinates, for example.
- Increase the influence of X_j by increasing A_{jj}.
- Projection-pursuit regression model: learn a low-rank version of A.

Structured Regression Functions

Goal: learn $E(Y|X) = f(X_1, X_2, \cdots, X_p)$. Consider analysis-of-variance (ANOVA) decomposition

$$f(X_1, \cdots, X_p) = \alpha + \sum_j g_j(X_j) + \sum_{k<\ell} g_{k\ell}(X_k, X_\ell) + \cdots . \tag{138}$$

We would like impose structure simplifications on the form, usually eliminating some of the higher-order terms.

- Additive model: $f(X) = \alpha + \sum_j g_j(X_j)$
- Varying coefficient models. First divide p predictors into $(X_1, \cdots, X_q)$ and the remainder is called $Z = (X_{q+1}, \cdots, X_p)$.

$$f(X) = \alpha(Z) + \beta_1(Z)X_1 + \cdots + \beta_q(Z)X_q \tag{139}$$

Kernel Density Estimation

Given random sample $x_1, \cdots, x_N$ from a probability density $f_X(x)$, we would like to estimate f_X at point x_0.

- Unsupervised learning problem.
- Local estimate

$$\hat{f}_X(x_0) = \frac{\#x_i \in \mathcal{N}(x_0)}{N\lambda}. \tag{140}$$

- Smooth Parzen estimate

$$\hat{f}_X(x_0) = \frac{1}{N\lambda} \sum_{i=1}^{N} K_\lambda(x_0, x_i), \tag{141}$$

where K_λ is a kernel function.

Kernel Density Classification

For a J class classification problem, we fit nonparametric density estimates $\hat{f}_j(X), j = 1, \cdots, J$ for each class. Then

$$\hat{P}(G = j|X = x_0) = \frac{\hat{\pi}_j \hat{f}_j(x_0)}{\sum_{k=1}^{J} \hat{\pi}_k \hat{f}_k(x_0)}. \tag{142}$$

Naive Bayes Classifier

Assumes given class $G = j$, the features X_k are independent

$$f_j(X) = \prod_{k=1}^{p} f_{jk}(X_k). \tag{143}$$

Two benefits:

- The individual class-conditional marginal densities can each be estimated separately using one-dimensional kernel density estimates. A generalization of the original naive Bayes procedures, which used univariate Gaussians to represent these marginals.
- If a component X_j of X is discrete, then an appropriate histogram estimate can be used. This provides a way of mixing variable types in a feature vector.

Link to generalized additive model

We derive the logit-transform

$$\log \frac{P(G=\ell|X)}{P(G=J|X)} = \log \frac{\pi_\ell f_\ell(X)}{\pi_J f_J(X)} \tag{144}$$

$$= \log \frac{\pi_\ell \prod_{k=1}^p f_{\ell k}(X_k)}{\pi_J \prod_{k=1}^p f_{Jk}(X_k)} \tag{145}$$

$$= \log \frac{\pi_\ell}{\pi_J} + \sum_{k=1}^p \log \frac{f_{\ell k}(X_k)}{f_{Jk}(X_k)} \tag{146}$$

$$= \alpha_\ell + \sum_{k=1}^p g_{\ell k}(X_k). \tag{147}$$

Radial Basis Functions

Main idea: use kernel functions $K_\lambda(\xi, x)$ with a sequence of location parameters $\xi_j, j = 1, \cdots, M$.

$$f(x) = \sum_{j=1}^{M} K_{\lambda_j}(\xi_j, x)\beta_j \tag{148}$$

$$= \sum_{j=1}^{M} D\left(\frac{\|x - \xi_j\|}{\lambda_j}\right)\beta_j, \tag{149}$$

where λ_j is the scale parameter.
A popular choice: standard Gaussian density function.

RBF network

Given D, we would like to estimate $\{\lambda_j, \xi_j, \beta_j\}, j = 1, \cdots, M$.

$$\min_{\{\lambda_j,\xi_j,\beta_j\}_1^M} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{M} \beta_j \exp\left\{ -\frac{-(x_i - \xi_j)^T (x_i - \xi_j)}{\lambda_j^2} \right\} \right)^2. \quad (150)$$

- nonconvex optimization problem.
- Usual practice: estimate $\{\lambda_j, \xi_j\}$ separately from β_j. Given $\{\lambda_j, \xi_j\}$, a standard LS problem. One way of estimation $\{\lambda_j, \xi_j\}$ is to fit Gaussian mixture density.
- One simplification: use the same λ_j.

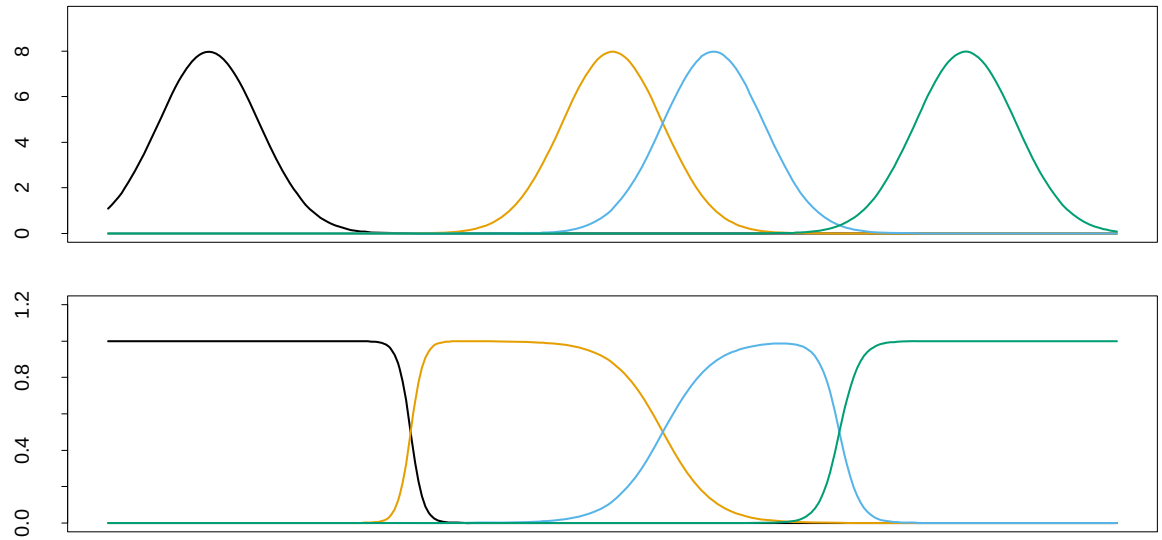


FIGURE 6.16. *Gaussian radial basis functions in* $\mathbb{R}$ *with fixed width can leave holes (top panel). Renormalized Gaussian radial basis functions avoid this problem, and produce basis functions similar in some respects to B-splines.*

Renormalized RBF

$$h_j(x) = \frac{D(\|x - \xi_j\|/\lambda)}{\sum_{k=1}^{M} D(\|x - \xi_k\|/\lambda)}, \tag{151}$$

Link to NW estimator

$$\hat{f}(x_0) = \sum_{i=1}^{N} y_i \frac{K_\lambda(x_0, x_i)}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)} \tag{152}$$

$$= \sum_{i=1}^{N} y_i h_i(x_0), \tag{153}$$

with $\xi_i = x_i$ and $\hat{\beta}_i = y_i$, $i = 1, \cdots, N$.

Gaussian mixture model

$$f(x) = \sum_{m=1}^{M} \alpha_m \phi(x; \mu_m, \Sigma_m), \tag{154}$$

with mixing proportions α_m, $\sum_m \alpha_m = 1$.
Fit by EM algorithm (details in Chapter 8).
The mixture model also provides an estimate of the probability that observation i belongs to component m,

$$\hat{r}_{im} = \frac{\hat{\alpha}_m \phi(x_i; \hat{\mu}_m, \hat{\Sigma}_m)}{\sum_{m=1}^{M} \hat{\alpha}_m \phi(x_i; \hat{\mu}_m, \hat{\Sigma}_m)}. \tag{155}$$

Outline

Training error and test error

With a target variable Y, a vector of input X and a prediction model $\hat{f}(X)$ estimated from a training set $\mathcal{T}$ consisting of N observations. Loss function $L(Y, \hat{f}(X))$.

- Training error: $\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i))$.
- (Conditional) test (generalization) error, is the prediction error over an independent test sample

$$\text{Err}_{\mathcal{T}} = E[L(Y, \hat{f}(X))|\mathcal{T}], \tag{156}$$

  where (X, Y) are drawn randomly from the joint distribution (population).
- Expected test error:

$$\text{Err} = E[L(Y, \hat{f}(X))] = E[\text{Err}_{\mathcal{T}}]. \tag{157}$$

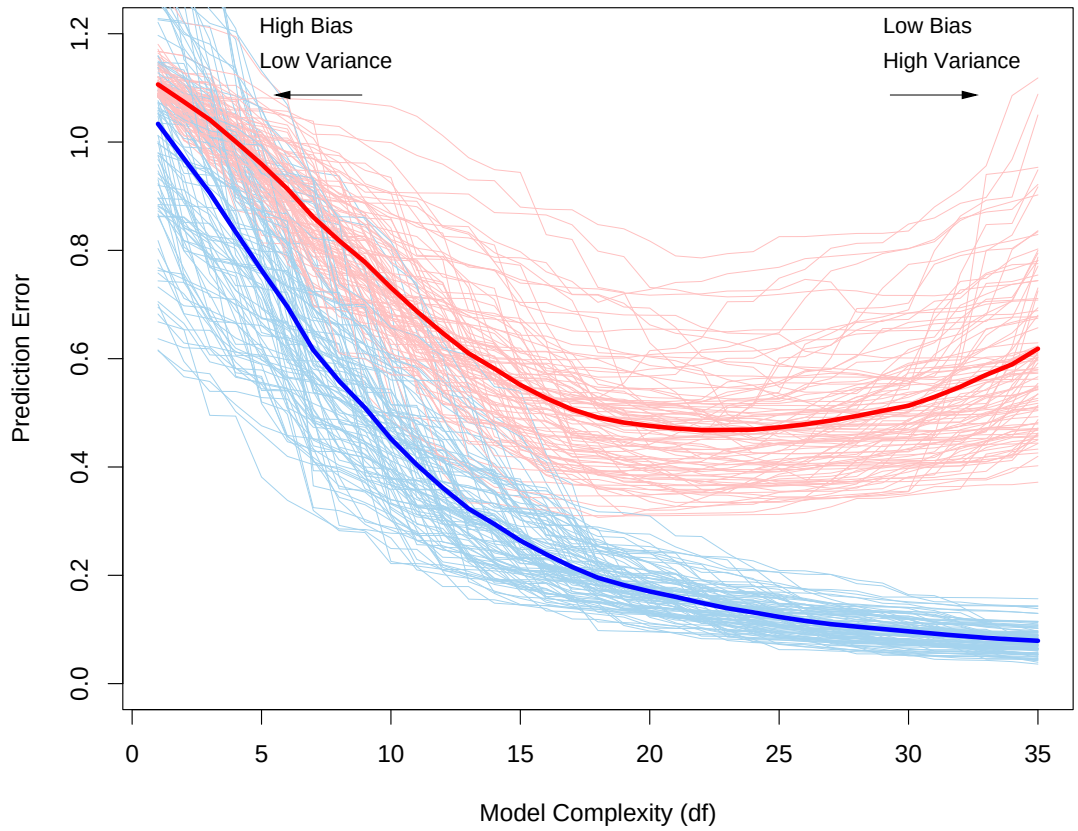


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\text{err}}$*, while the light red curves show the conditional test error* $\text{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\text{E}[\overline{\text{err}}]$*.*

Loss function

- For quantitative Y, we can use $L(Y, \hat{f}(X)) = (Y - \hat{f}(X))^2$ or $|Y - \hat{f}(X)|$.
- For categorical response $G \in \{1, \cdots, K\}$. Usually we model $p_k(X) = Pr(G = k|X)$ and set $\hat{G}(X) = \arg\max_k \hat{p}_k(X)$. Loss functions can be taken as $L(G, \hat{G}(X)) = I(G \neq \hat{G}(X))$ or $-2\sum_{k=1}^{K} I(G = k) \log \hat{p}_k(X) = -2 \log \hat{p}_G(X)$ (-2 times log-likelihood).

Goal

- Model selection: estimating the performance of different models in order to choose the best one.
- Model assessment: having chosen a final model, estimating its prediction error (generalization error) on new data.

Ideal evaluation method

Divide the data into the following three parts.

- The training set is used to fit the models;
- The validation set is used to estimate prediction error for model selection;
- The test set is used for assessment of the generalization error of the final chosen model. (CANNOT be used to tune the parameter of the model)

However, we usually do not have this much data to use. The idea: approximate the validation step.

Bias-Variance Decomposition

Assume $Y = f(X) + \epsilon$, where $E\epsilon = 0$ and $Var(\epsilon) = \sigma_\epsilon^2$. The expected prediction error of $\hat{f}(X)$ at an input $X = x_0$ with squared-error loss is as follows.

$$\begin{aligned}
\text{Err}(x_0) &= E[(Y - \hat{f}(x_0))^2 | X = x_0] && (158)\\
&= \sigma_\epsilon^2 + [E\hat{f}(x_0) - f(x_0)]^2 + E[\hat{f}(x_0) - E\hat{f}(x_0)]^2 && (159)\\
&= \sigma_\epsilon^2 + \text{Bias}^2(\hat{f}(x_0)) + Var(\hat{f}(x_0)) && (160)\\
&= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance}. && (161)
\end{aligned}$$

Linear regression decomposition

Linear model fit: $\hat{f}_p(x) = x^T\hat{\beta}$. we have

$$\begin{aligned} \text{Err}(x_0) &= E[(Y - \hat{f}_p(x_0))^2 | X = x_0] && (162) \\ &= \sigma_\epsilon^2 + [f(x_0) - E\hat{f}_p(x_0)]^2 + \|h(x_0)\|^2\sigma_\epsilon^2. && (163) \end{aligned}$$

Here $h(x_0) = X(X^TX)^{-1}x_0$. The *in-sample* error is

$$\frac{1}{N}\sum_{i=1}^{N}\text{Err}(x_i) = \sigma_\epsilon^2 + \frac{1}{N}\sum_{i=1}^{N}[f(x_i) - E\hat{f}(x_i)]^2 + \frac{p}{N}\sigma_\epsilon^2. \quad (164)$$

Ridge regression decomposition

Consider the ridge regression $\hat{f}_\alpha(x)$ with penalty α. Let β_* denote the parameters of the best-fitting linear approximation to f:

$$\beta_* = \arg\min_\beta E(f(X) - X^T\beta)^2. \tag{165}$$

The average squared bias of $\hat{f}_\alpha(x_0)$ has the following decomposition

$$E_{x_0}[f(x_0 - E\hat{f}_\alpha(x_0)]^2 = E_{x_0}[f(x_0) - x_0^T\beta_*]^2 + E_{x_0}[x_0^T\beta_* - Ex_0^T\hat{\beta}_\alpha]^2 \tag{166}$$

$$= \text{Ave[Model Bias]}^2 + \text{Ave[Estimation Bias]}^2. \tag{167}$$

For least square fit ($\alpha = 0$), the estimation bias is 0. When $\alpha > 0$, we trade off a positive estimation bias with a smaller variance.

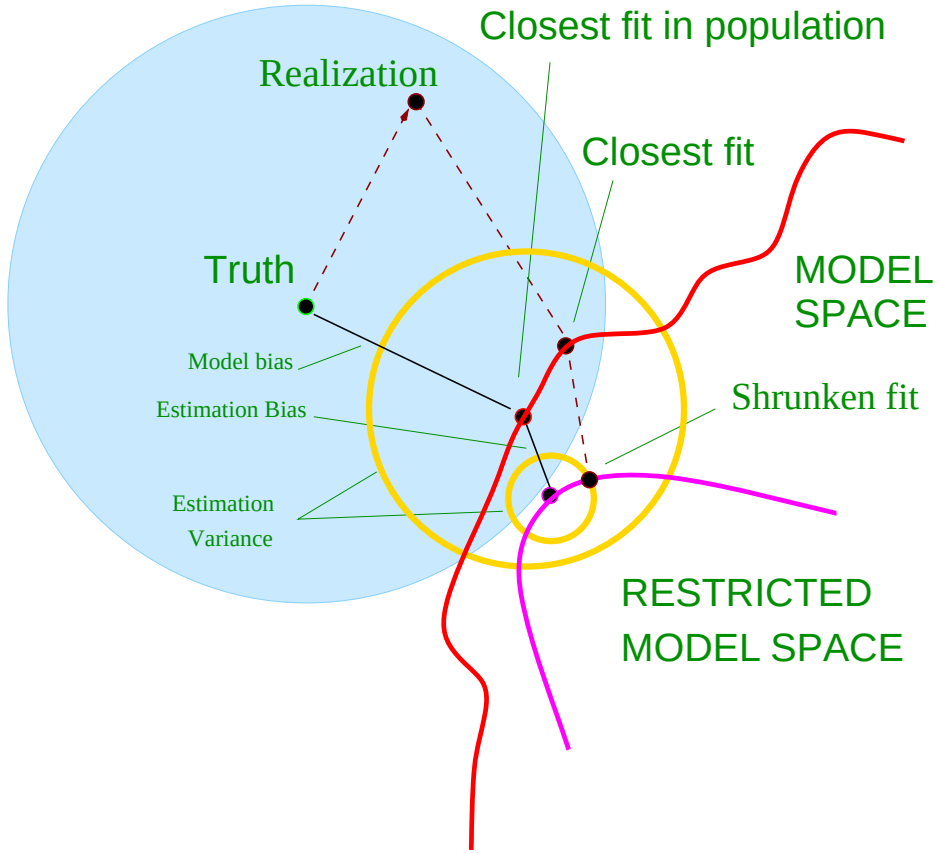


FIGURE 7.2. *Schematic of the behavior of bias and variance. The model space is the set of all possible predictions from the model, with the "closest fit" labeled with a black dot. The model bias from the truth is shown, along with the variance, indicated by the large yellow circle centered at the black dot labeled "closest fit in population." A shrunken or regularized fit is also shown, having additional estimation bias, but smaller prediction error due to its decreased variance.*

An example

There are 80 observations and 20 predictors, uniformly distributed in the hypercube $[0,1]^{20}$. The situations are as follows:

- Y is 0 if $X_1 \leq 1/2$ and 1 if $X_1 > 1/2$, and we apply k-nearest neighbors.
- Y is 1 if $\sum_{j=1}^{10} X_j$ is greater than 5 and 0 otherwise, and we use best subset linear regression of size p.

We investigate squared error loss and 0-1 loss.

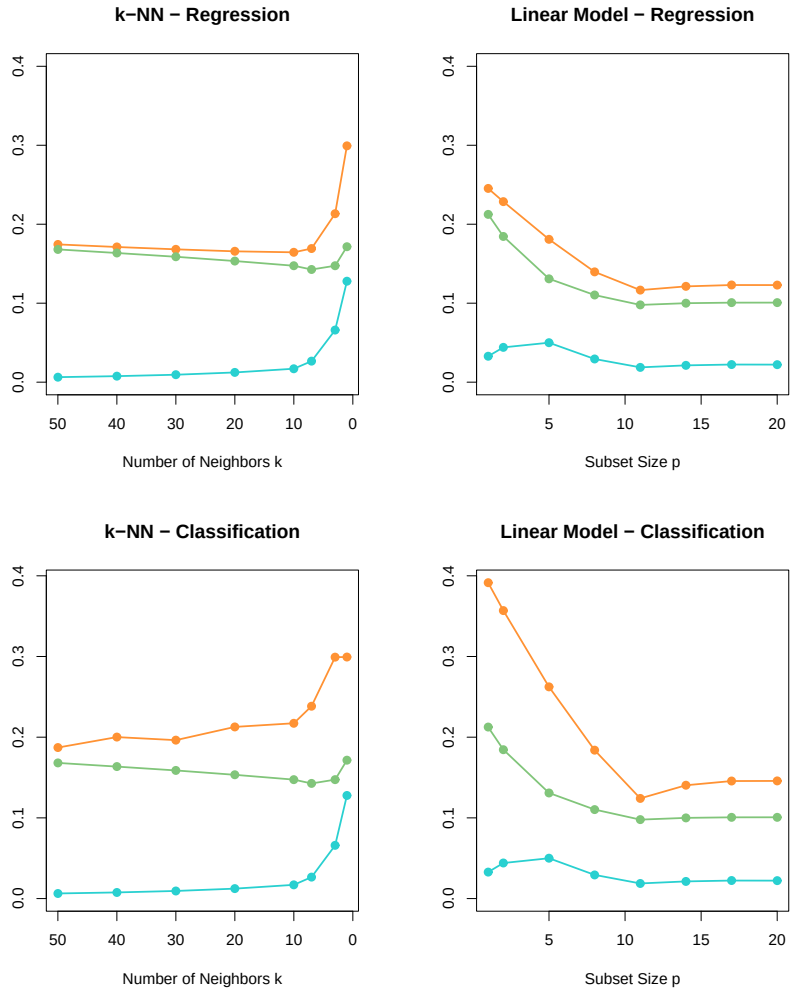


FIGURE 7.3. *Expected prediction error (orange), squared bias (green) and variance (blue) for a simulated example. The top row is regression with squared error loss; the bottom row is classification with* 0–1 *loss. The models are k-nearest neighbors (left) and best subset regression of size p (right). The variance and bias*

Optimism of the Training Error Rate

Generation error of a model $\hat{f}$ at a new test point $(X^0, Y^0) \sim F$ is

$$\text{Err}_{\mathcal{T}} = E_{X^0, Y^0}[L(Y^0, \hat{f}(X^0)|\mathcal{T}]. \tag{168}$$

Averaging over training sets $\mathcal{T}$ leads to the expected error

$$\text{Err} = E_{\mathcal{T}} E_{X^0, Y^0}[L(Y^0, \hat{f}(X^0)|\mathcal{T}]. \tag{169}$$

Most methods estimate Err rather than $\text{Err}_{\mathcal{T}}$. Typically, the training error $\overline{\text{err}} = N^{-1} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i))$ will be smaller than $\text{Err}_{\mathcal{T}}$.

- test input vectors do not coincide with the training input vectors. $\text{Err}_{\mathcal{T}}$ called *extra-sample* error.
- Consider the *in-sample* error (new response at the training data points).

$$\text{Err}_{in} = \frac{1}{N} \sum_{i=1}^{N} E_{Y^0}[L(Y_i^0, \hat{f}(x_i))|\mathcal{T}]. \tag{170}$$

Optimism

$$\text{op} \equiv \text{Err}_{in} - \overline{\text{err}}. \tag{171}$$

Usually, we have op > 0. The average optimism is taking expectation over the training sets.

$$\omega \equiv E_y(\text{op}). \tag{172}$$

Here the predictors in the training set are fixed, and the expectation is over the training set outcome values. For squared error, 0-1, and other loss functions, one can show quite generally that

$$\omega = \frac{2}{N} \sum_{i=1}^{N} Cov(\hat{y}_i, y_i). \tag{173}$$

Effective number of parameters

If $\hat{y}_i$ is obtained by a linear fit with d inputs, we have

$$\sum_{i=1}^{N} Cov(\hat{y}_i, y_i) = d\sigma_\epsilon^2. \tag{174}$$

for the additive error model $Y = f(X) + \epsilon$.

$$E_y(\text{Err}_{in}) = E_y(\overline{\text{err}}) + 2\frac{d}{N}\sigma_\epsilon^2. \tag{175}$$

Usual Idea: estimate the optimism and add it to the training error $\overline{\text{err}}$: AIC, BIC, C_p.
Another way: Cross-validation and bootstrap, aim to directly estimate the extra-sample error Err.

C_p Estimates of In-Sample Prediction error

General estimate:

$$\widehat{\text{Err}}_{in} = \overline{\text{err}} + \hat{\omega}, \tag{176}$$

with $\hat{\omega}$ an estimate of the average optimism.
If we have d parameters under the squared error loss, we have the C_p statistic,

$$C_p = \overline{\text{err}} + 2 \cdot \frac{d}{N} \hat{\sigma}_\epsilon^2. \tag{177}$$

Here $\hat{\sigma}_\epsilon^2$ is an estimate of the noise variance, obtained from a low-bias model.

AIC Estimates of In-Sample Prediction error

AIC is a more general estimate of Err_{in} using a log-likelihood loss function. As $N \to \infty$,

$$-2E[\log Pr_{\hat{\theta}}(Y)] \approx -\frac{2}{N}E[\text{loglik}] + 2\frac{d}{N}. \tag{178}$$

$Pr_{\hat{\theta}}(Y)$ is a family of densities for Y (containing the "true" density), $\hat{\theta}$ is the MLE of θ,

$$\text{loglik} = \sum_{i=1}^{N} \log Pr_{\hat{\theta}}(y_i). \tag{179}$$

For Gaussian model with known σ_ϵ^2, the AIC is equivalent to C_p.

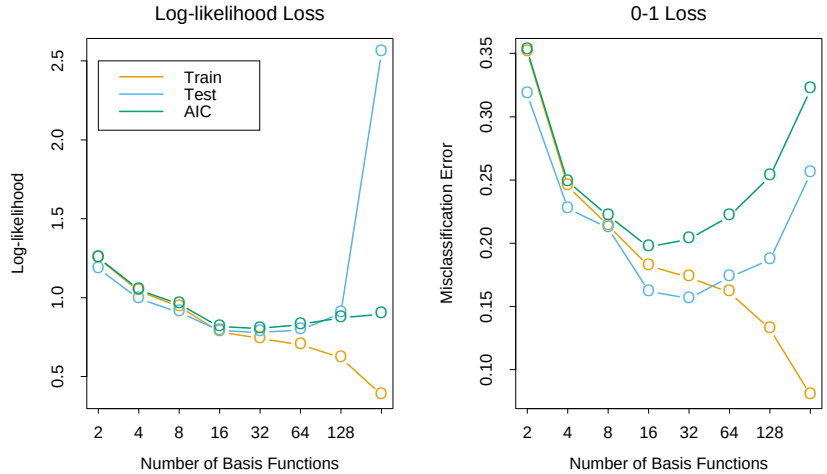


FIGURE 7.4. *AIC used for model selection for the phoneme recognition example of Section 5.2.3. The logistic regression coefficient function* $\beta(f) = \sum_{m=1}^{M} h_m(f)\theta_m$ *is modeled as an expansion in* M *spline basis functions. In the left panel we see the AIC statistic used to estimate* $\mathrm{Err}_{\mathrm{in}}$ *using log-likelihood loss. Included is an estimate of* Err *based on an independent test sample. It does well except for the extremely over-parametrized case (*$M = 256$ *parameters for* $N = 1000$ *observations). In the right panel the same is done for 0–1 loss. Although the AIC formula does not strictly apply here, it does a reasonable job in this case.*

The Effective Number of Parameters

For a linear fitting method with $\hat{\mathbf{y}} = \mathbf{S}\mathbf{y}$. Includes linear regression, ridge regression and cubic smoothing splines. Define $df(\mathbf{S}) = \text{trace}(\mathbf{S})$ as the Effective Number of Parameters/effective degrees of freedom.

BIC

$$BIC = -2\text{loglik} + (\log N)d. \tag{180}$$

Given a set of Candidate models $\mathcal{M}_m, m = 1, \cdots, M$ and the corresponding model parameter θ_m. Assume a prior distribution $Pr(\theta_m|\mathcal{M}_m)$, the posterior probability of a given model is

$$Pr(\mathcal{M}_m|\mathbf{Z}) \propto Pr(\mathcal{M}_m)Pr(\mathbf{Z}|\mathcal{M}_m) \tag{181}$$

$$\propto Pr(\mathcal{M}_m) \int Pr(\mathbf{Z}|\theta_m, \mathcal{M}_m)Pr(\theta_m|\mathcal{M}_m)d\theta_m. \tag{182}$$

Here $\mathbf{Z} = \{x_i, y_i\}_1^N$.

Bayes Factor

To compare two models $\mathcal{M}_m$ and $\mathcal{M}_\ell$, we form the posterior odds

$$\frac{Pr(\mathcal{M}_m|\mathbf{Z})}{Pr(\mathcal{M}_\ell|\mathbf{Z})} = \frac{Pr(\mathcal{M}_m)}{Pr(\mathcal{M}_\ell} \cdot \frac{Pr(\mathbf{Z}|\mathcal{M}_m)}{Pr(\mathbf{Z}|\mathcal{M}_\ell)}. \tag{183}$$

If the odds is greater than 1, we choose model $\mathcal{M}_m$.

$$\text{BF}(\mathbf{Z}) = \frac{Pr(\mathbf{Z}|\mathcal{M}_m)}{Pr(\mathbf{Z}|\mathcal{M}_\ell)}. \tag{184}$$

is called *Bayes Factor*.

BIC criteria

Assume that the prior over models is uniform, using Laplace approximation to the integral (Ripley, 1996, page 64)

$$\log Pr(\mathbf{Z}|\mathcal{M}_m) = \log Pr(\mathbf{Z}|\hat{\theta}_m, \mathcal{M}_m) - \frac{d_m}{2} \cdot \log N + O(1). \tag{185}$$

As a by-product, we have the posterior probability of each model $\mathcal{M}_m$ as

$$\frac{\exp(-\frac{1}{2}BIC_m)}{\sum_{\ell=1}^{M} \exp(-\frac{1}{2}BIC_\ell)}. \tag{186}$$

Minimum Description Length

Idea: think of the data z as a message that we would like to encode and transfer. Choose the shortest code for transmission.

Message	z_1	z_2	z_3	z_4
Code	0	10	110	111

Instantaneous prefix code: no code is the prefix of any other, and the receiver (who knows all of the possible codes), knows exactly when the message has been completely sent.
Theorem (Shannon), we should be code length $l_i = -\log_2 Pr(z_i)$ and the average length

$$E(\text{length}) \geq -\sum Pr(z_i) \log_2(Pr(z_i)). \tag{187}$$

RHS: entropy of the distribution $Pr(z_i)$.

Link to BIC

With an infinite set of messages, the entropy is $-\int Pr(z)\log_2 Pr(z)dz$. Suppose we have a model M with parameters θ and data $Z = (X, y)$. We would like to transmit the output with known input, $Pr(y|\theta, M, X)$. The length of the transmit is

$$\text{length} = -\log Pr(y|\theta, M, X) - \log Pr(\theta|M). \tag{188}$$

For $y \sim N(\theta, \sigma^2)$ and $\theta \sim N(0, 1)$ and no input,

$$\text{length} = \text{constant} + \log\sigma + \frac{(y-\theta)^2}{2\sigma^2} + \frac{\theta^2}{2}. \tag{189}$$

(188): (negative) log-posterior distribution, and hence minimizing description length is equivalent to maximizing posterior probability.

Vapnik-Chervonenkis Dimension

A gneneral measure of complexity.
Consider a class of functions $\{f(x, \alpha)\}$ indexed by parameter vector α with $x \in R^p$. Use it for classification: $f(x, \alpha) > 0$ for class 1.

A set of points is said to be *shattered* by a class of functions if, no matter how we assign a binary label to each point, a member of the class can perfectly separate them.

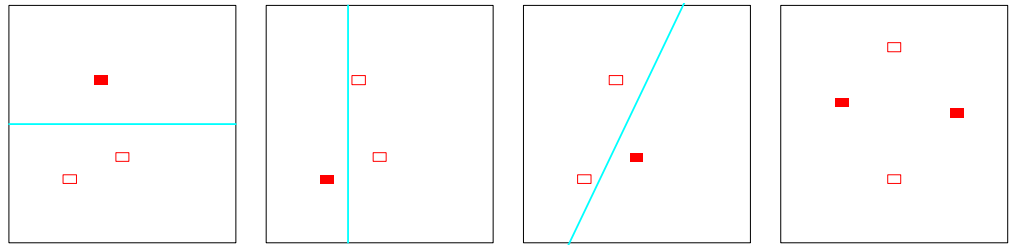


FIGURE 7.6. *The first three panels show that the class of lines in the plane can shatter three points. The last panel shows that this class cannot shatter four points, as no line will put the hollow points on one side and the solid points on the other. Hence the VC dimension of the class of straight lines in the plane is three. Note that a class of nonlinear curves could shatter four points, and hence has VC dimension greater than three.*

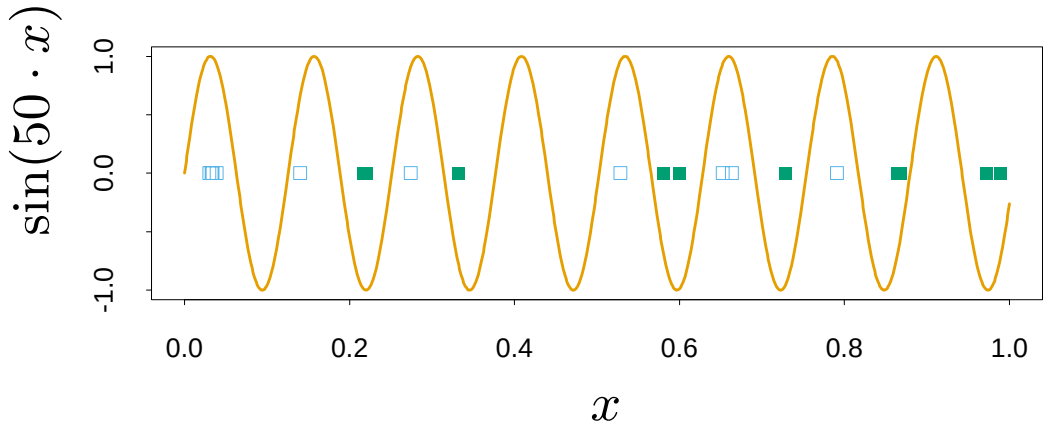


FIGURE 7.5. *The solid curve is the function* $\sin(50x)$ *for* $x \in [0, 1]$. *The green (solid) and blue (hollow) points illustrate how the associated indicator function* $I(\sin(\alpha x) > 0)$ *can shatter (separate) an arbitrarily large number of points by choosing an appropriately high frequency* α.

VC dimension

A linear indicator function in p dimension has VC dimension $p+1$ (the number of free parameters).

The family $\sin(\alpha x)$ has infinite CV dimension.

A general inequality about optimism

If we fit N training points using a class of functions $\{f(x, \alpha)\}$ having VC dimension h, then with probability at least $1 - \eta$ over training sets (Cherkassky and Mulier (2007, pages 116-118)):

$$\text{Err}_{\mathcal{T}} \leq \overline{\text{err}} + \frac{\epsilon}{2}\left(1 + \sqrt{1 + \frac{4\overline{\text{err}}}{\epsilon}}\right) \text{ (binary classification)} \tag{190}$$

$$\text{Err}_{\mathcal{T}} \leq \frac{\overline{\text{err}}}{(1 - c\sqrt{\epsilon})_+} \text{ (regression)}, \tag{191}$$

where $\epsilon = a_1 N^{-1}(h[\log(a_2 N/h) + 1] - \log(\eta/4))$ and $a_1 \in (0, 4], a_2 \in (0, 2]$. The bound hold simultaneously for all members $f(x, \alpha)$.

Cross-Validation

Perhaps the simplest and most widely used method for estimating prediction error.

The CV estimates the expected extra-sample error $\text{Err} = E[L(Y, \hat{f}(X))]$, rather than the conditional error.

K-Fold CV

Let $\kappa : \{1, \cdots, N\} \mapsto \{1, \cdots, K\}$ be an indexing function dividing each observation to one of the K parts. Denote by $\hat{f}^{-k}(x)$ be the fitted function with the k-th part of the data removed. CV estimate of the prediction error is

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i)). \tag{192}$$

Typical choices are 5,10 or N (leave-one-out CV).

Application of CV for choosing tuning parameter

Suppose we consider a set of models $f(x, \alpha)$ indexed by tuning parameter α. Then, we can calculate

$$CV(\hat{f}, \alpha) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \alpha)). \tag{193}$$

Then set

$$\hat{\alpha} = \arg\min_{\alpha} CV(\hat{f}, \alpha). \tag{194}$$

Then we fit the chosen model $f(x, \hat{\alpha})$ to all the data.

- "one-standard-error" rule: choose the most parsimonious model whose error is no more than one standard error above the error of the best model. Particularly important in the LASSO parameter selection. fit = cv.glmnet(x,y); fit$lambda.1se

Which K is the best?

- With $K = N$, the cross-validation estimator is approximately unbiased for the true (expected) prediction error, but can have high variance because the N training sets are so similar to one another.
- When $K = 5$ or 10, low variance, but larger bias.

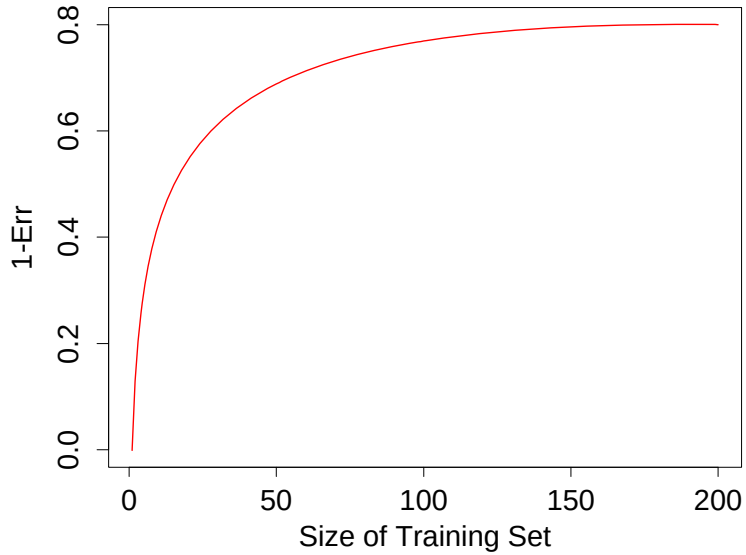


FIGURE 7.8. *Hypothetical learning curve for a classifier on a given task: a plot of* $1-\text{Err}$ *versus the size of the training set* N*. With a dataset of* 200 *observations,* 5*-fold cross-validation would use training sets of size* 160*, which would behave much like the full set. However, with a dataset of* 50 *observations fivefold cross–validation would use training sets of size* 40*, and this would result in a considerable overestimate of prediction error.*

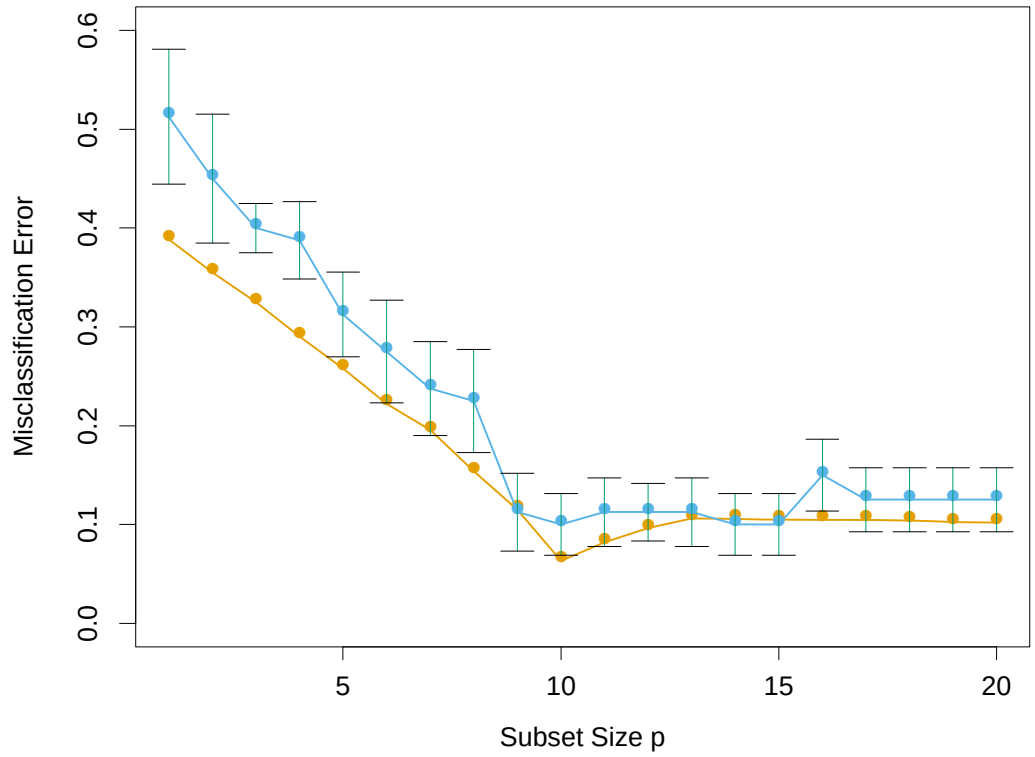


FIGURE 7.9. *Prediction error (orange) and tenfold cross-validation curve (blue) estimated from a single training set, from the scenario in the bottom right panel of Figure 7.3.*

Generalized Cross-validation

- An approximation to leave-one-out CV with efficient calculation.
- For linear fitting method under squared error loss, we have $\hat{\mathbf{y}} = \mathbf{S}\mathbf{y}$.
- We have for many linear fitting methods,

$$\frac{1}{N}\sum_{i=1}^{N}[y_i - \hat{f}^{-i}(x_i)]^2 = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i - \hat{f}(x_i)}{1 - S_{ii}}\right]^2, \tag{195}$$

  with S_{ii} be the i-th diagonal element of $\mathbf{S}$.
- GCV approximation

$$GCV(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i - \hat{f}(x_i)}{1 - \text{trace}(\mathbf{S})/N}\right]^2. \tag{196}$$

Bootstrap

Given training set $\mathbf{Z} = (z_1, z_2, \cdots, z_N)$ where $z_i = (x_i, y_i)$.
Bootstrap Key idea: randomly draw datasets with replacement from the training data, each sample the same size as the original training set. Refit the model to each of the B bootstrap datasets.
Suppose $S(\mathbf{Z})$ is some quantity of interest from $\mathbf{Z}$. We can estimate any aspect of the distribution of $S(\mathbf{Z})$, for example, the variance.

$$\widehat{Var}(S(\mathbf{Z})) = \frac{1}{B-1}\sum_{b=1}^{B}(S(\mathbf{Z}^{*b}) - \bar{S}^*)^2, \tag{197}$$

where $\bar{S}^* = B^{-1}\sum_b S(\mathbf{Z}^{*b})$.

Using bootstrap to estimate prediction error

A naive approach:

$$\widehat{\text{Err}}_{boot} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{n=1}^{N} L(y_i, \hat{f}^{*b}(x_i)). \tag{198}$$

Training sample and test sample have observations in common!
Example: 1-nearest neighbor classifier applied to a two-class classification problem with the same number of observations in each class, in which the predictors and class labels are in fact independent. Then the true error rate is 0.5. But the contributions to the bootstrap estimate $\widehat{\text{Err}}_{boot}$ will be zero unless i does not appear in bootstrap sample b.

Toy example

$$Pr\{\text{observation } i \in \text{ bootstrap sample } b\} = 1 - (1 - \frac{1}{N})^N \quad (199)$$

$$\approx 1 - e^{-1} = 0.632 \quad (200)$$

Hence, $E[\widehat{\text{Err}}_{boot}] = 0.5 * 0.368 = 0.184$, much smaller than 0.5.

Leave-one-out bootstrap estimate

$$\widehat{\text{Err}}^{(1)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L(y_i, \hat{f}^{*b}(x_i)), \tag{201}$$

where C^{-i} is the set of indices of b that do not contain i.

- Fix the overfitted problem of $\widehat{\text{Err}}_{boot}$. But has a bias problem just like two-fold CV.

0.632 estimator

Designed to alleviate the bias of $\widehat{\text{Err}}^{(1)}$.

$$\widehat{\text{Err}}^{(.632)} = .368\overline{\text{err}} + .632\widehat{\text{Err}}^{(1)}. \tag{202}$$

Pulls LOO bootstrap estimate down toward the training error rate, reducing the upward bias.

Outline

Expectation-Maximization Algorithm

Two-Component Mixture Model

$$Y_1 \sim N(\mu_1, \sigma_1^2) \tag{203}$$
$$Y_2 \sim N(\mu_2, \sigma_2^2) \tag{204}$$
$$Y = (1 - \Delta) Y_1 + \Delta Y_2, \tag{205}$$

where $\Delta \in \{0, 1\}$ with $Pr(\Delta = 1) = \pi$.
Denote $\phi_\theta(x)$ denote the normal density with $\theta = (\mu, \sigma^2)$. Then, the density of Y is

$$g_Y(y) = (1 - \pi)\phi_{\theta_1}(y) + \pi\phi_{\theta_2}(y). \tag{206}$$

Parameter of interest: $\theta = (\pi, \theta_1, \theta_2) = (\pi, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2)$.

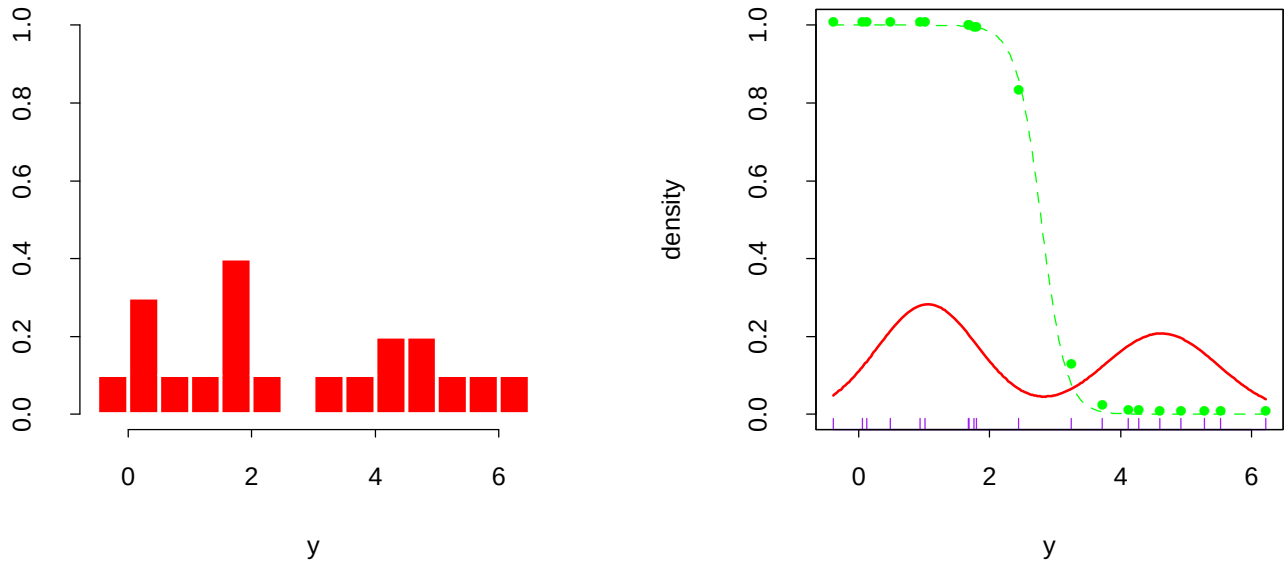


FIGURE 8.5. *Mixture example. (Left panel:) Histogram of data. (Right panel:) Maximum likelihood fit of Gaussian densities (solid red) and responsibility (dotted green) of the left component density for observation y, as a function of y.*

Latent variable for class

The log-likelihood based on the N training cases is

$$\ell(\theta; \mathbf{Z}) = \sum_{i=1}^{N} \log[(1-\pi)\phi_{\theta_1}(y_i) + \pi\phi_{\theta_2}(y_i)]. \tag{207}$$

Not easy to maximize directly. Introduce latent variable $\Delta_i \in \{0, 1\}$ representing which model Y_i is from. If we knew $\Delta_i, i = 1, \cdots, N$, the

$$\ell_0(\theta; \mathbf{Z}, \mathbf{\Delta}) = \sum_{i=1}^{N} [(1-\Delta_i) \log \phi_{\theta_1}(y_i) + \Delta_i \log \phi_{\theta_2}(y_i)] \tag{208}$$

$$+ \sum_{i=1}^{N} [(1-\Delta_i) \log(1-\pi) + \Delta_i \log \pi]. \tag{209}$$

Then, the estimate of μ_1, σ_1^2 would be the sample mean and variance for observations with $\Delta_i = 0$ and the estimate of π would be the proportion of $\Delta_i = 1$.

Responsibility of model

Suppose $\Delta_i = 1$ representing observation i comes from model 2.
Otherwise comes from model 1.
Since the values of Δ_i's are unknown, we substitute each Δ_i with its expected value

$$\gamma_i(\theta) = E(\Delta_i|\theta, \mathbf{Z}) = Pr(\Delta_i = 1|\theta, \mathbf{Z}), \tag{210}$$

which are called the *responsibility* of model 2 for observation i.

Algorithm 8.1 *EM Algorithm for Two-component Gaussian Mixture.*

1. Take initial guesses for the parameters $\hat{\mu}_1, \hat{\sigma}_1^2, \hat{\mu}_2, \hat{\sigma}_2^2, \hat{\pi}$ (see text).

2. *Expectation Step*: compute the responsibilities

$$\hat{\gamma}_i = \frac{\hat{\pi}\phi_{\hat{\theta}_2}(y_i)}{(1-\hat{\pi})\phi_{\hat{\theta}_1}(y_i) + \hat{\pi}\phi_{\hat{\theta}_2}(y_i)}, \ i = 1, 2, \ldots, N. \tag{8.42}$$

3. *Maximization Step*: compute the weighted means and variances:

$$\hat{\mu}_1 = \frac{\sum_{i=1}^N (1-\hat{\gamma}_i) y_i}{\sum_{i=1}^N (1-\hat{\gamma}_i)}, \qquad \hat{\sigma}_1^2 = \frac{\sum_{i=1}^N (1-\hat{\gamma}_i)(y_i - \hat{\mu}_1)^2}{\sum_{i=1}^N (1-\hat{\gamma}_i)},$$

$$\hat{\mu}_2 = \frac{\sum_{i=1}^N \hat{\gamma}_i y_i}{\sum_{i=1}^N \hat{\gamma}_i}, \qquad \hat{\sigma}_2^2 = \frac{\sum_{i=1}^N \hat{\gamma}_i (y_i - \hat{\mu}_2)^2}{\sum_{i=1}^N \hat{\gamma}_i},$$

   and the mixing probability $\hat{\pi} = \sum_{i=1}^N \hat{\gamma}_i / N$.

4. Iterate steps 2 and 3 until convergence.

EM Algorithm in general

Suppose the observed data is $\mathbf{Z}$ with log-likelihood $\ell(\theta; \mathbf{Z})$. The latent or missing data is $\mathbf{Z}^m$. The complete data is $\mathbf{T} = (\mathbf{Z}, \mathbf{Z}^m)$ with log-likelihood $\ell_0(\theta; \mathbf{T})$.

Algorithm 8.2 *The EM Algorithm.*

1. Start with initial guesses for the parameters $\hat{\theta}^{(0)}$.
2. *Expectation Step*: at the jth step, compute

$$Q(\theta', \hat{\theta}^{(j)}) = \mathrm{E}(\ell_0(\theta'; \mathbf{T})|\mathbf{Z}, \hat{\theta}^{(j)}) \tag{8.43}$$

   as a function of the dummy argument θ'.
3. *Maximization Step*: determine the new estimate $\hat{\theta}^{(j+1)}$ as the maximizer of $Q(\theta', \hat{\theta}^{(j)})$ over θ'.
4. Iterate steps 2 and 3 until convergence.

Why EM works

Since

$$Pr(\mathbf{Z}^m|\mathbf{Z},\theta') = \frac{Pr(\mathbf{Z}^m,\mathbf{Z}|\theta')}{Pr(\mathbf{Z}|\theta')}, \tag{211}$$

we have

$$Pr(\mathbf{Z}|\theta') = \frac{Pr(\mathbf{T}|\theta')}{Pr(\mathbf{Z}^m|\mathbf{Z},\theta')}. \tag{212}$$

Looking at log-likelihood, $\ell(\theta',\mathbf{Z}) = \ell_0(\theta',\mathbf{T}) - \ell_1(\theta';\mathbf{Z}^m|\mathbf{Z})$, where ℓ_1 is based on the conditional density $Pr(\mathbf{Z}^m|\mathbf{Z},\theta')$. Taking conditional expectation with respect to $\mathbf{T}|\mathbf{Z}$ governed by parameter θ

$$\ell(\theta',\mathbf{Z}) = E[\ell_0(\theta',\mathbf{T})|\mathbf{Z},\theta] - E[\ell_1(\theta';\mathbf{Z}^m|\mathbf{Z})|\mathbf{Z},\theta] \tag{213}$$

$$\equiv Q(\theta',\theta) - R(\theta',\theta). \tag{214}$$

Reason

In the M step, the EM algorithm maximizes $Q(\theta', \theta)$ over θ', rather than the actual $\ell(\theta', \mathbf{Z})$. Why does it work?

- $R(\theta^*, \theta)$ is the expectation of a log-likelihood of a density (indexed by θ^*), with respect to the same density indexed by θ, and hence (by Jensen's inequality) is maximized when $\theta^* = \theta$.
- If θ' maximize $Q(\theta', \theta)$, we have

$$\ell(\theta'; \mathbf{Z}) - \ell(\theta; \mathbf{Z}) = [Q(\theta', \theta) - Q(\theta, \theta)] - [R(\theta', \theta) - R(\theta, \theta)] \quad (215)$$

$$\geq 0. \quad (216)$$

- Hence, in the expectation step, the log-likelihood is non-decreasing.

Bagging

First consider the regression problem with $\mathbf{Z} = \{(x_1, y_1), \cdots, (x_N, y_N)\}$, we would like to prediction $\hat{f}(x)$ at input x. Bootstrap Aggregation (bagging) average the predictions obtained from B bootstrap samples.

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x). \tag{217}$$

Denote by $\hat{\mathcal{P}}$ the empirical distribution of $\mathbf{Z}$. The "true" bagging estimate is $E_{\hat{\mathcal{P}}} \hat{f}^*(x)$. $\hat{f}_{bag}$ is a Monte Carlo estimate.

- The true bagged estimate will differ from the original estimate $f(x)$ only when the latter is a nonlinear or adaptive function of the data.
- Popular application: regression tree.

Bagging classifier

- Consider a classification problem with K classes.
- Then consider an underlying indicator-vector function $\hat{f}(x)$, with value a single 1 and $K-1$ zeros.
- $\hat{G}(x) = \arg\max_k \hat{f}_k$.
- The bagged estimate $\hat{f}_{bag}$ is a K-vector $[p_1(x), p_2(x), \cdots, p_K(x)]$ with $p_k(x)$ equal to the proportion of trees predicting class k at x.
- $\hat{G}_{bag}(x) = \arg\max_k \hat{f}_{bag}(x)$.

Outline

Generalized Additive Models

$$E(Y|X_1, \cdots, X_p) = g[\mu(\mathbf{X})] = \alpha + f_1(X_1) + \cdots + f_p(X_p). \quad (218)$$

f_j are unspecified smooth functions.
Here, we fit each function using a scatterplot smoother (e.g., a cubic smoothing spline or kernel smoother), and provide an algorithm for simultaneously estimating all p functions.

- $g(\mu) = \mu$, identity link, additive model, for Gaussian response data.
- $g(\mu) = \text{logit}(\mu)$, or $g(\mu) = \text{probit}(\mu)$, model binary response.
- $g(\mu) = \log(\mu)$ for Poisson count data. log-additive model.

Fitting additive models

Penalized residual sum of squares

$$PRSS(\alpha, f_1, \cdots, f_p) = \sum_{i=1}^{N}(y_i - \alpha - \sum_{j=1}^{p} f_j(x_{ij}))^2 + \sum_{j=1}^{p} \int f_j''(t_j)^2 dt_j, \tag{219}$$

where $\lambda_j \geq 0$ are tuning parameters.

- It can be shown that the minimizer is an additive cubic spline model; each of the functions f_j is a cubic spline in the component X_j, with knots at each of the unique values of $x_{ij}, i = 1, \cdots, N$.
- Identifiability issue: add constraint $\sum_1^N f_j(x_{ij}) = 0$ for all j.

Algorithm (Hastie and Tibshirani, 1990 for details)

Algorithm 9.1 *The Backfitting Algorithm for Additive Models.*

1. Initialize: $\hat{\alpha} = \frac{1}{N}\sum_1^N y_i$, $\hat{f}_j \equiv 0, \forall i, j$.

2. Cycle: $j = 1, 2, \ldots, p, \ldots, 1, 2, \ldots, p, \ldots,$

$$\hat{f}_j \leftarrow \mathcal{S}_j\left[\{y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})\}_1^N\right],$$

$$\hat{f}_j \leftarrow \hat{f}_j - \frac{1}{N}\sum_{i=1}^N \hat{f}_j(x_{ij}).$$

until the functions $\hat{f}_j$ change less than a prespecified threshold.

Algorithm 9.2 *Local Scoring Algorithm for the Additive Logistic Regression Model.*

1. Compute starting values: $\hat{\alpha} = \log[\bar{y}/(1-\bar{y})]$, where $\bar{y} = \text{ave}(y_i)$, the sample proportion of ones, and set $\hat{f}_j \equiv 0 \; \forall j$.

2. Define $\hat{\eta}_i = \hat{\alpha} + \sum_j \hat{f}_j(x_{ij})$ and $\hat{p}_i = 1/[1+\exp(-\hat{\eta}_i)]$.

   Iterate:

   (a) Construct the working target variable

   $$z_i = \hat{\eta}_i + \frac{(y_i - \hat{p}_i)}{\hat{p}_i(1-\hat{p}_i)}.$$

   (b) Construct weights $w_i = \hat{p}_i(1-\hat{p}_i)$

   (c) Fit an additive model to the targets z_i with weights w_i, using a weighted backfitting algorithm. This gives new estimates $\hat{\alpha}, \hat{f}_j, \; \forall j$

3. Continue step 2. until the change in the functions falls below a prespecified threshold.

Remarks

- Additive models provide a useful extension of linear models, making them more flexible while still retaining much of their interpretability.
- The backfitting algorithm fits all predictors, which is not feasible or desirable when a large number are available.
- Lasso-type penalties to estimate sparse additive models, for example the COSSO procedure of Lin and Zhang (2006) and the SpAM proposal of Ravikumar et al. (2008).

Tree-Based Methods

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (e.g., a constant function) in each one. For regression tree. Suppose we have N observations (x_i, y_i), where $x_i = (x_{i1}, \cdots, x_{ip})^T$. The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have. Say we the space of X into M regions $R_1, \cdots, R_M$. Then, we set

$$\hat{f}(X) = \sum_{m=1}^{M} \hat{c}_m I\{X \in R_m\}. \tag{220}$$

Then

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m). \tag{221}$$

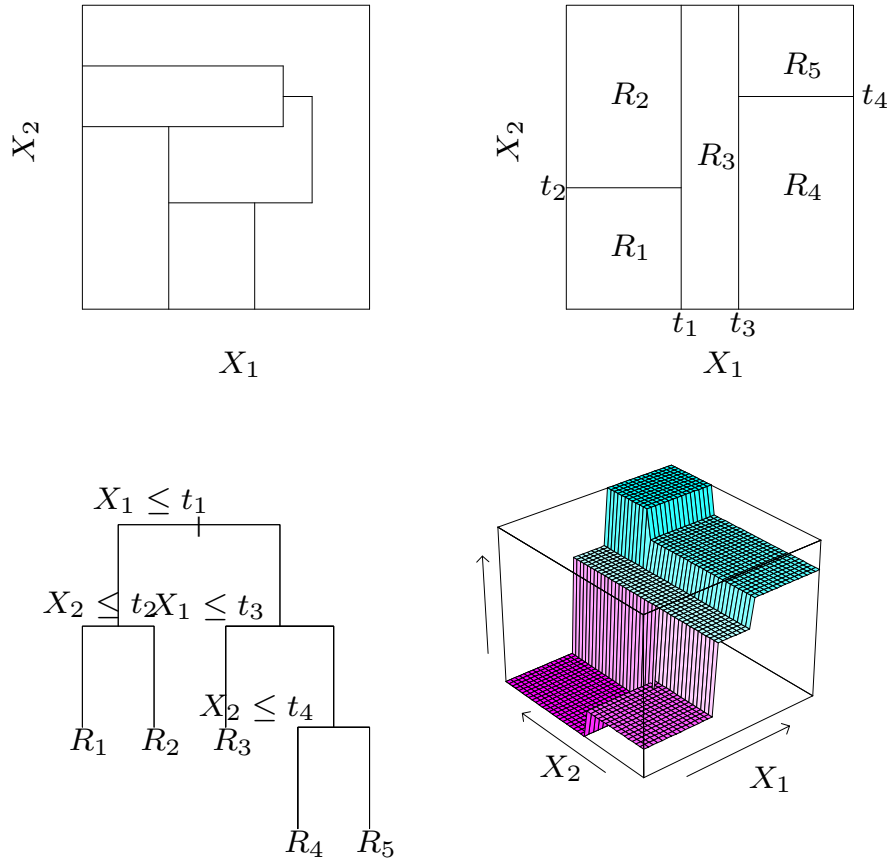


FIGURE 9.2. *Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.*

Grow the true

We proceed with a greedy algorithm. Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j > s\}. \tag{222}$$

Then, we search for variable j and split point s that solves

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]. \tag{223}$$

Then, repeat the process for each of the two regions.

Selecting the tree size

Tree size is a tuning parameter governing the models complexity, and the optimal tree size should be adaptively chosen from the data.
Preferred strategy:

- Grow a large tree T_0, stopping the splitting process only when some minimum node size (say 5) is reached.
- Prune the tree using cost-complexity pruning

Define a subtree $T \subset T_0$ be any tree that can be obtained by pruning T_0. Denote the number of terminal nodes in T by $|T|$.

$$C_\alpha(T) = \sum_{i=1}^{N}(y_i - \hat{y}_i(T))^2 + \alpha|T|, \tag{224}$$

where $\hat{y}_i(T)$ is the fitted value for x_i using subtree T. $\alpha \geq 0$ controls the tradeoff between tree size and goodness of fit to the data.

Classification Trees

In a node m, representing a region R_m with N_m observations, let

$$\hat{p}_{mk} = \frac{1}{N_k} \sum_{x_i \in R_m} I(y_i = k), \tag{225}$$

the proportion of class k observations in node m. We classify the observations in node m to class $k(m) = \arg\max_k \hat{p}_{mk}$, the majority class in node m.

Node impurity measures

- Misclassification error: $1 - \hat{p}_{mk}(m)$.
- Gini index: $\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'}$
- Cross-entropy or deviance: $-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$.

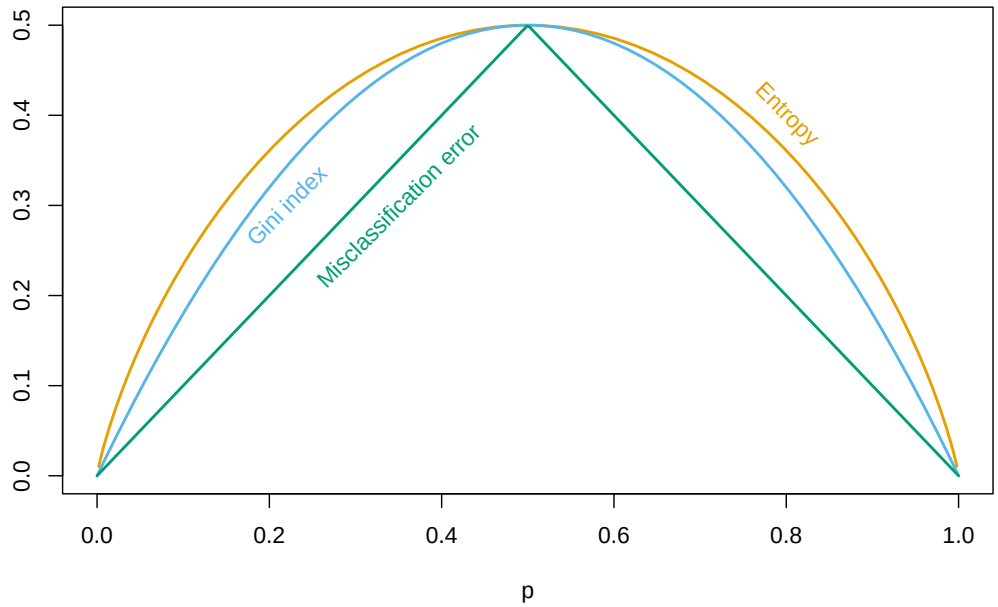


FIGURE 9.3. *Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through* $(0.5, 0.5)$.

MARS: Multivariate Adaptive Regression Splines

MARS is an adaptive procedure for regression, and is well suited for high-dimensional problems (i.e., a large number of inputs).
Use expansions in piecewise linear basis functions of the form $(x-t)_+$ and $(t-x))+$. (linear splines). The two functions are called *reflection pairs*.
The idea is to form reflected pairs for each input X_j with knots at each observed value x_{ij} of that input .

$$\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}_{t \in \{x_{1j}, x_{2j}, \cdots, X_{Nj}\}, j=1,2,\cdots,p}. \tag{226}$$

model setup

The MARS model

$$f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X), \tag{227}$$

where each $h_m(X)$ is a function in $\mathcal{C}$ or a product of two or more such functions.

- Usually fit with a forward stepwise regression, greedy procedure.

Fitting process

- Initialization: only include the constant function $h_0(X) = 1$ in the basis functions $\mathcal{M}_0 = \{1\}$.
- Given the current model $\mathcal{M}_j$, add the products of a function $h_m \in \mathcal{M}_j$ with one of the reflected pairs in $\mathcal{C}$.

$$h_\ell(X)(X_j - t)_+, h_\ell(X)(t - X_j)+, h_\ell \in \mathcal{M}_j, t \in \mathcal{C} \tag{228}$$

  Choose the pair that produces the largest decrease in training error.
- Repeat the above process until the model M_j contains some preset maximum number of terms.

Model selection

At the end of the fitting process, we have a large set of $\mathcal{M}_j$. This typically overfits the data, and so a backward deletion procedure is applied. The term whose removal causes the smallest increase in residual squared error is deleted from the model at each stage, producing an estimated best model $\hat{f}_\lambda$ of each size (number of terms) λ. Use GCV

$$GCV(\lambda) = \frac{\sum_{i=1}^{N}(y_i - \hat{f}_\lambda(x_i))^2}{(1 - M(\lambda)/N)^2}. \tag{229}$$

The value $M(\lambda)$ is the effective number of parameters: this accounts both for the number of terms in the models, plus the number of parameters used in selecting the optimal positions of the knots.

Outline

Boosting

Boosting is one of the most powerful learning ideas introduced in the last twenty years. Originally designed for classification.

- Motivation: combines the outputs of many "weak" classifiers to produce a powerful "committee".
- Fundamentally different from bagging.
- Most popular boosting algorithm due to Freund and Schapire (1997) called "AdaBoost.M1".

AdaBoost.M1

Consider a two-class problem, with the output variable coded as $Y \in \{-1, 1\}$. Given a vector of predictor variables X, a classifier $G(X)$ produces a prediction taking one of the two values $\{-1, 1\}$. The training error

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i)), \tag{230}$$

A *weak* classifier is one whose error rate is only slightly better than random guessing. The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers $G_m(x), m = 1, 2, \cdots, M$.

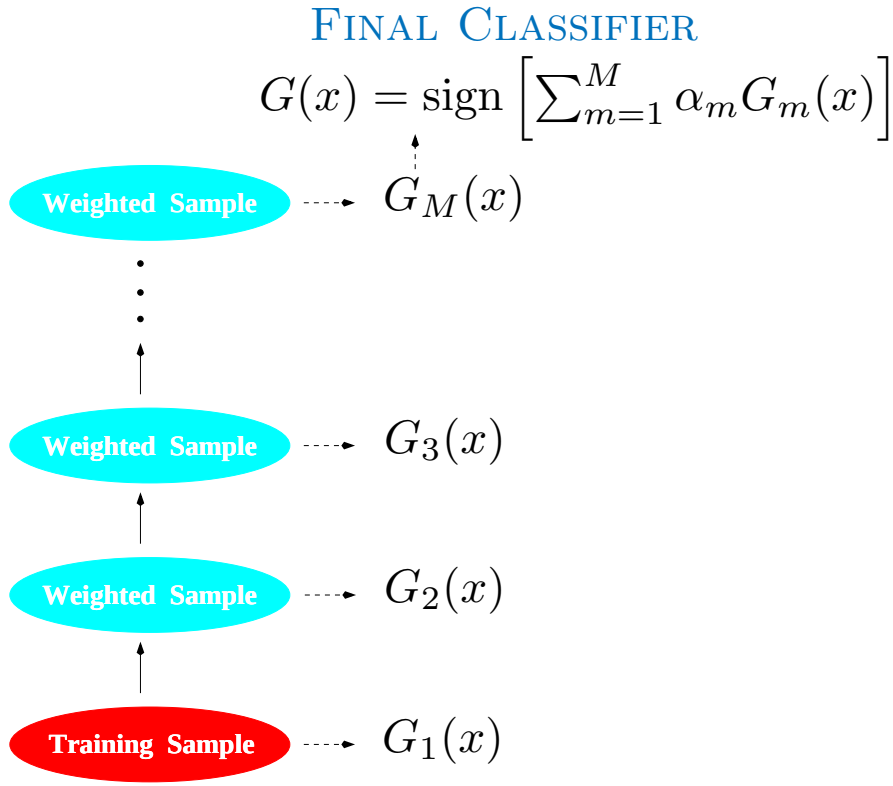


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Example

$X_1, \cdots, X_{10} \sim N(0,1)$, $Y = 1$ if $\sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5)$ and $Y = -1$ otherwise. Here the weak classifier is just a "stump": a two terminal- node classification tree. Weak classifier itself: test error rate 46.5%.

See the example in R!

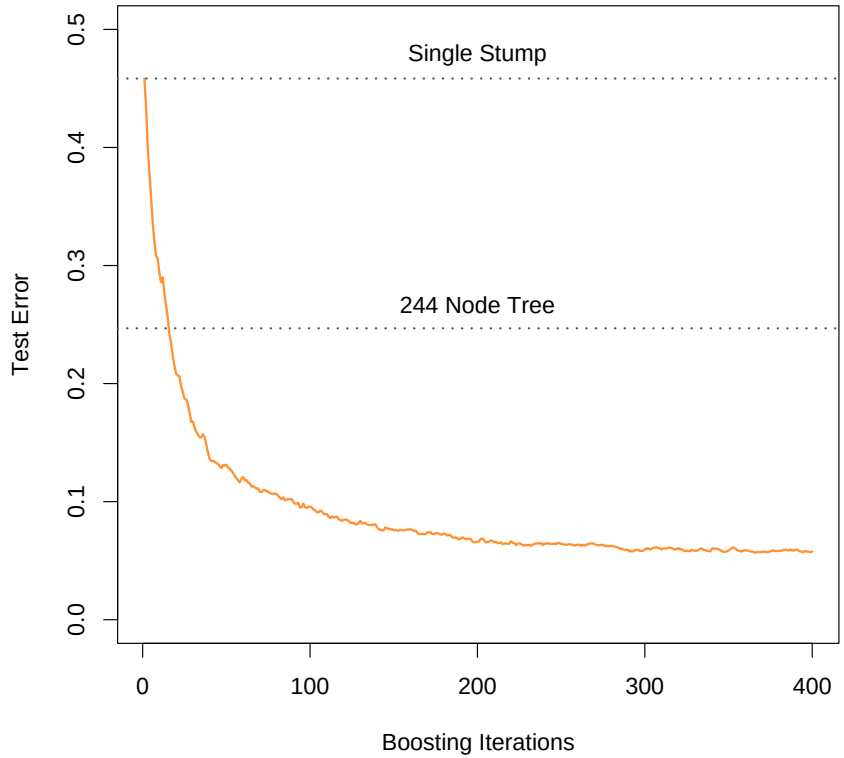


FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.*

Additive model interpretation

Boosting is a way of fitting an additive expansion in a set of elementary basis functions. In general

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m), \tag{231}$$

where $\beta_m, m = 1, \cdots, M$ are the expansion coefficients.

- In single-hidden-layer neural networks (Chapter 11), $b(x; \gamma) = \sigma(\gamma_0 + \gamma_1^T x)$, where $\sigma(t) = 1/(1 + e^{-t})$ is the sigmoid function.
- In signal processing, wavelets are a popular choice with γ parameterizing the location and scale shifts of a mother wavelet.
- MARS: truncated power splines basis functions where γ parameterizes the variables and values for the knots.
- Trees: γ parameterizes the split variables and split points at the internal nodes, and the predictions at the terminal nodes.

Forward Stagewise Additive Modeling

Usually, we solve

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right). \tag{232}$$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$.

Exponential Loss and Adaboost

We show that AdaBoost.M1 is equivalent to forward stagewise additive modeling using the loss function

$$L(y, f(x)) = \exp(-yf(x)). \tag{233}$$

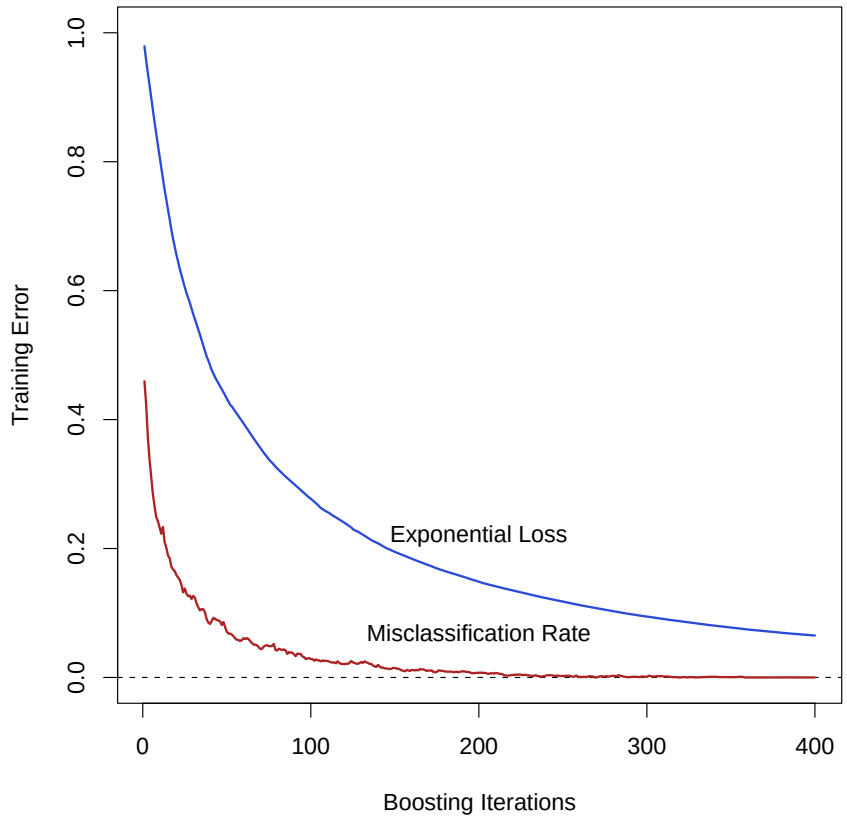


FIGURE 10.3. *Simulated data, boosting with stumps: misclassification error rate on the training set, and average exponential loss:* $(1/N)\sum_{i=1}^{N}\exp(-y_i f(x_i))$. *After about* 250 *iterations, the misclassification error is zero, while the exponential loss continues to decrease.*

Why exponential loss?

We have

$$f^*(x) = \arg\min_{f(x)} E_{Y|x}(e^{-Yf(x)}) = \frac{1}{2}\log\frac{Pr(Y=1|x)}{Pr(Y=-1|x)}, \tag{234}$$

or equivalently,

$$Pr(Y=1|x) = \frac{1}{1+\exp(-2f^*(x))}. \tag{235}$$

Thus the additive expansion of AdaBoost is estimating onehalf the log-odds of $Pr(Y=1|x)$.

Boosting Trees

A tree:

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j), \tag{236}$$

with parameters $\Theta = \{R_j, \gamma_j\}_1^J$. Then, we solve

$$\widehat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x \in R_j} L(y_i, \gamma_j). \tag{237}$$

- Finding γ_j given R_j. Easy!
- Finding R_j.

Boosted tree model

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m), \tag{238}$$

induced in a forward stagewise manner. At each step, solve

$$\widehat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)), \tag{239}$$

given the current model $f_{m-1}(x)$.

A general optimization problem

A general problem of using $f(x)$ to predict y on the training data is

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)). \tag{240}$$

We would like to solve

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f}), \tag{241}$$

where $\mathbf{f} \in R^N$.
Numerical optimization procedures uses a sum of component vectors

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m, \tag{242}$$

where $\mathbf{h}_m$ represents the correction at the m-th step.

Steepest Descent

Chooses $\mathbf{h}_m = -\rho_m \mathbf{g}_m$ with $\mathbf{g}_m$ being the gradient of $L(\mathbf{f})$ evaluated at $\mathbf{f} = \mathbf{f}_{m-1}$. The step length ρ_m is

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m). \tag{243}$$

Very greedy strategy without taking into account the tree structure.

Gradient Boosting

Ate the m-th iteration, induce a tree $T(x;\Theta_m)$ whose predictions are as close as possible to the negative gradient.

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i;\Theta)^2. \tag{244}$$

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Regularization

Idea: update slowly to avoid overfitting problem. Given a learning rate $\nu \in (0, 1)$, replace 2(d) by

$$f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm}). \tag{245}$$

Recommendation (Friedman, 2001): use small value of $\nu < 0.1$ and choose M by early stopping.

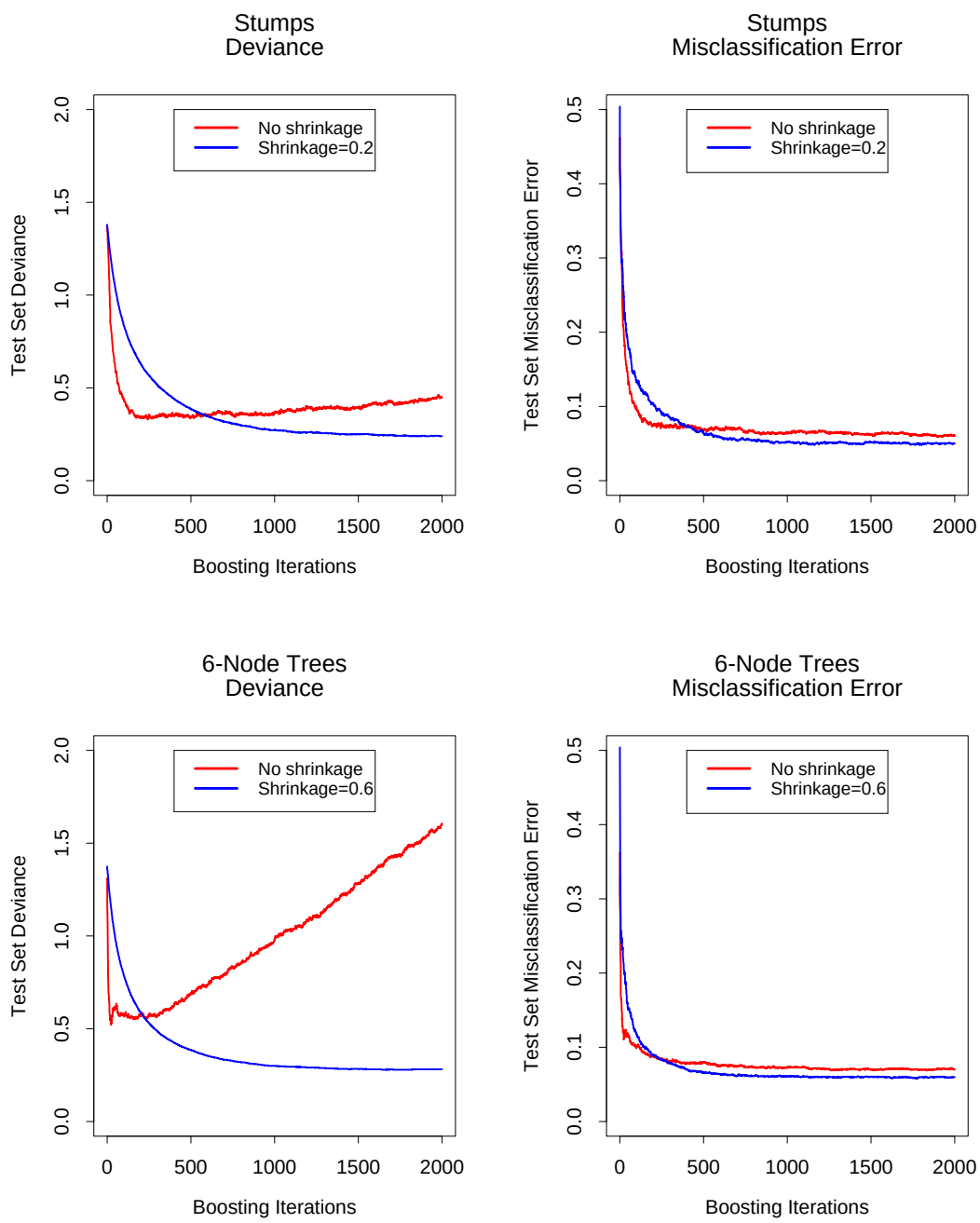


FIGURE 10.11. *Test error curves for simulated example (10.2) of Figure 10.9, using gradient boosting (MART). The models were trained using binomial de*

Outline

Projection Pursuit Regression

PPR model:

$$f(X) = \sum_{m=1}^{M} g_m(\omega_m^T X). \tag{246}$$

This is an additive model, but in the derived features $V_m = \omega_m^T X$ rather than the original inputs.

- $g_m(\omega_m^T X)$ is called *ridge function*. We seek ω_m so that the model fits well, hence the name "projection pursuit".
- Model popular choice: $M = 1$, single index model.

Fit PPR

Given training data $(x_i, y_i), i = 1, \cdots, N$, we minimize

$$\sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} g_m(\omega_m^T x_i) \right]^2, \tag{247}$$

over functions g_m and direction vectors ω_m. Consider just one term ($M = 1$) and drop the subscript.

- Given ω, then we can calculate $v_i = \omega^T x_i$. One-dimensional smoothing problem.
- Given g, we minimize over ω. Using quasi-Newton method.

If $M > 1$, we build the model in a forward stage-wise manner, adding a pair (ω_m, g_m) at each stage.

Quasi-Newton update

Let ω_o be the current estimate for ω, we write

$$g(\omega^T x_i) \approx g(\omega_o^T x_i) + g'(\omega_o^T x_i)(\omega - \omega_o)^T x_i \tag{248}$$

then

$$\sum_{i=1}^{N}[y_i - g(\omega^T x_i)]^2 \approx \sum_{i=1}^{N} g'(\omega_o^T x_i)^2 \left[\left(\omega_o^T x_i + \frac{y_i - g(\omega_o^T x_i)}{g'(\omega_o^T x_i)}\right) - \omega^T x_i\right]^2. \tag{249}$$

Least square regression of $\omega_o^T x_i + \frac{y_i - g(\omega_o^T x_i)}{g'(\omega_o^T x_i)}$ on the input x_i with weights $g(\omega^T x_i)]^2$ and no intercept.

Neural Networks

- Many successful applications in classification, prediction, natural language processing.
- Foundation of deep learning.
- For K-class classification problem.

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, \cdots, M, \tag{250}$$

$$T_k = \beta_{0k} + \beta_k^T Z, k = 1, \cdots, K, \tag{251}$$

$$f_k(X) = g_k(T), k = 1, \cdots, K, \tag{252}$$

  where $Z = (Z_1, \cdots, Z_M)$ and $T = (T_1, T_2, \cdots, T_K)$.
- Activation function:
  Sigmoid function: $\sigma(v) = 1/(1 + \exp(-v))$.
  ReLU (Rectified Linear Unit): $\sigma(v) = vI\{v \geq 0\}$.
- Softmax function: $g_k(T) = \frac{\exp(T_k)}{\sum_{\ell=1}^K \exp(T_\ell)}$.

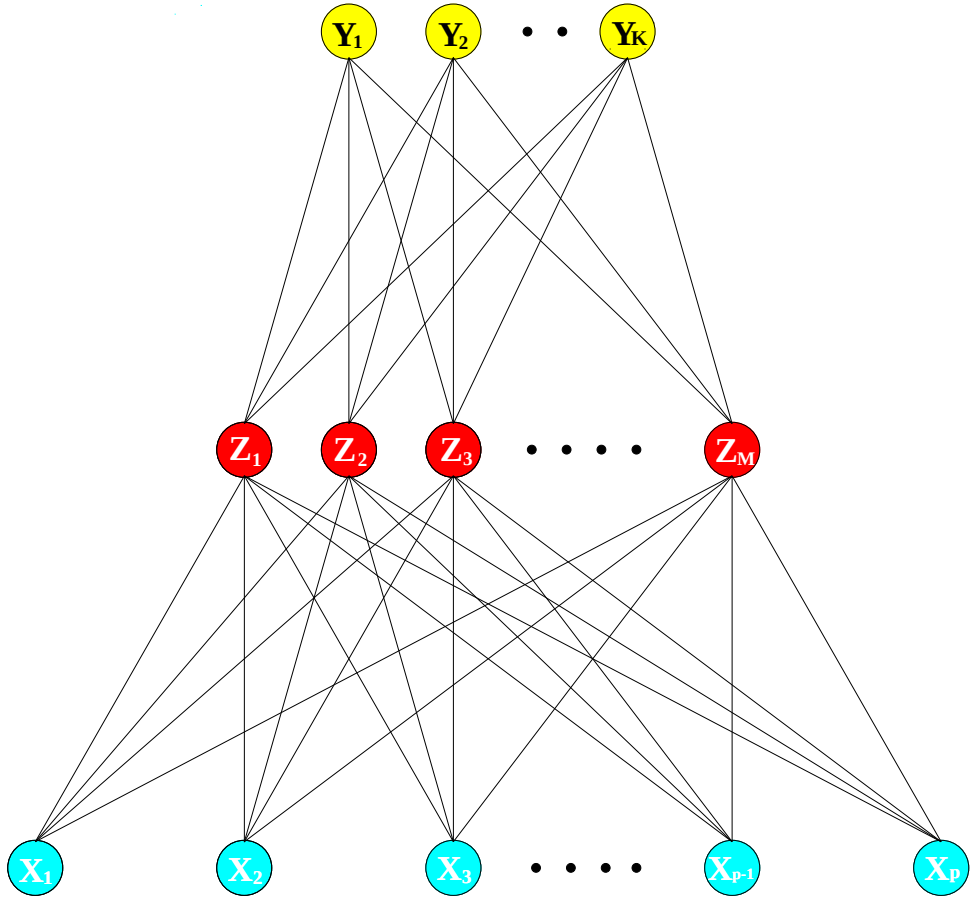


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

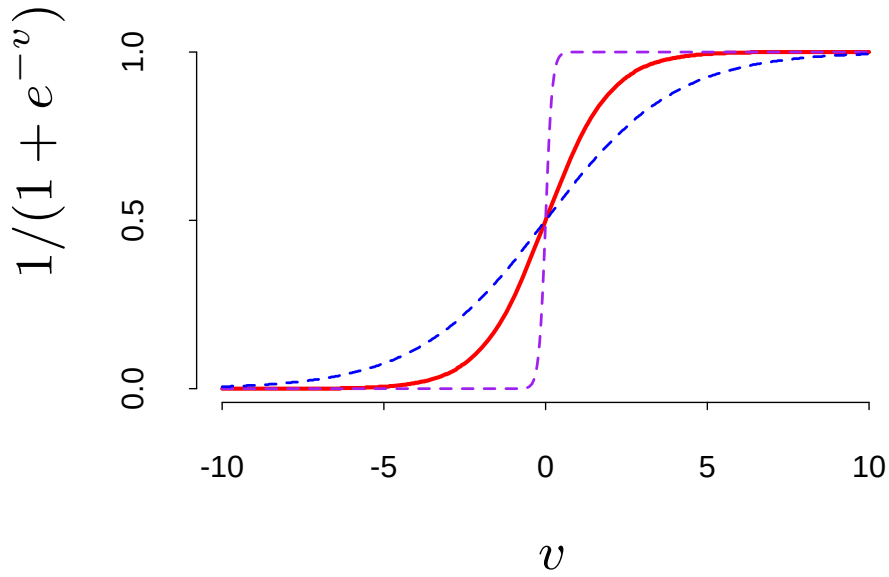


FIGURE 11.3. *Plot of the sigmoid function* $\sigma(v) = 1/(1+\exp(-v))$ *(red curve), commonly used in the hidden layer of a neural network. Included are* $\sigma(sv)$ *for* $s = \frac{1}{2}$ *(blue curve) and* $s = 10$ *(purple curve). The scale parameter* s *controls the activation rate, and we can see that large* s *amounts to a hard activation at* $v = 0$*. Note that* $\sigma(s(v - v_0))$ *shifts the activation threshold from* 0 *to* v_0*.*

Fitting Neural Network

Unknown parameters in neural networks, usually called *weights*. Denote the weights by θ, consisting of

$$\{\alpha_{0m}, \alpha_m; m = 1, 2, \cdots, M\}, M(p+1) \text{ weights} \tag{253}$$

$$\{\beta_{0k}, \beta_k; k = 1, 2, \cdots, K\}, K(M+1) \text{ weights .} \tag{254}$$

For regression, we minimize

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2. \tag{255}$$

For classification, use cross-entropy (deviance) loss:

$$R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i), \tag{256}$$

with the corresponding classifier $G(x) = \arg\max_k f_k(x)$.

back-propagation

- A version of gradient descent, using chain rule to calculation differentiation.
- Using a forward and backward sweep over the network.
- Key point: keep track only of quantities local to each unit.

Details for squared error loss

Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$, let $z_i = (z_{1i}, z_{2i}, \cdots, z_{Mi})$. Then we have

$$R(\theta) = \sum_{i=1}^{N} R_i, \tag{257}$$

with derivatives

$$\frac{\partial R_i}{\partial \beta_{km}} = 2(y_{ik} - f_k(x_i))g_k'(\beta_0 + \beta_k^T z_i)z_{mi}, \tag{258}$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_0 + \beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{i\ell}. \tag{259}$$

Details

A gradient descent update at the $(r+1)$ iteration.

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \tag{260}$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}}, \tag{261}$$

with γ_r the *learning rate*.

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}, \tag{262}$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = s_{mi} x_{i\ell}. \tag{263}$$

δ_{ki} and s_{mi} are "errors" from the current model at the output and hidden layer units.

Update summary

back-propagation equations

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta ki. \tag{264}$$

- Forward pass: compute the predicted values $\hat{f}_k(x_i)$.
- Backward pass: computer errors δ_{ki} and then back-propagate to get the errors s_{mi}.

Online learning

- In the gradient descent update, one can process each observation one at a time and updating the gradient after each training case.
- Stochastic approximation (Robbins and Munro, 1951): ensure convergence if $\gamma_r \to 0$, $\sum_r \gamma_r = \infty$ and $\sum_r \gamma_r^2 < \infty$. (e.g., $\gamma_r = 1/r$).

Deep learning structures

- Feed-forward Fully Connected Network. When the features do not have any additional structure.
- Convolutional Neural Network (CNN). Applications: images classification, video classification, object detection.
- Recurrent Neural Network (RNN). Applications: speech recognition, translation.

Outline

Support Vector Classifiers

Training data: N pairs $(x_1, y_1), \cdots, (x_N, y_N)$, with $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$. Define a hyperplane by

$$\{x : f(x) = x^T \beta + \beta_0 = 0\}, \tag{265}$$

where β is a unit vector: $\|\beta\| = 1$.

$$\max_{\beta, \beta_0, \|\beta\|=1} M \tag{266}$$

$$\text{s.t. } y_i(x_i^T \beta + \beta_0) \geq M, i = 1, \cdots, N. \tag{267}$$

M represents the margin.

Alternative formulation

$$\min_{\beta,\beta_0} \|\beta\| \tag{268}$$

$$\text{s.t. } y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, \cdots, N. \tag{269}$$

If the classes overlap, we allow some points to be on the wrong side of the margin. Define slack variables $\xi = (\xi_1, \cdots, \xi_N)$.

$$\min_{\beta,\beta_0} \|\beta\| \tag{270}$$

$$\text{s.t. } y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, i = 1, \cdots, N. \tag{271}$$

$$\xi_i \geq 0, \sum \xi_i \leq \text{ constant.} \tag{272}$$

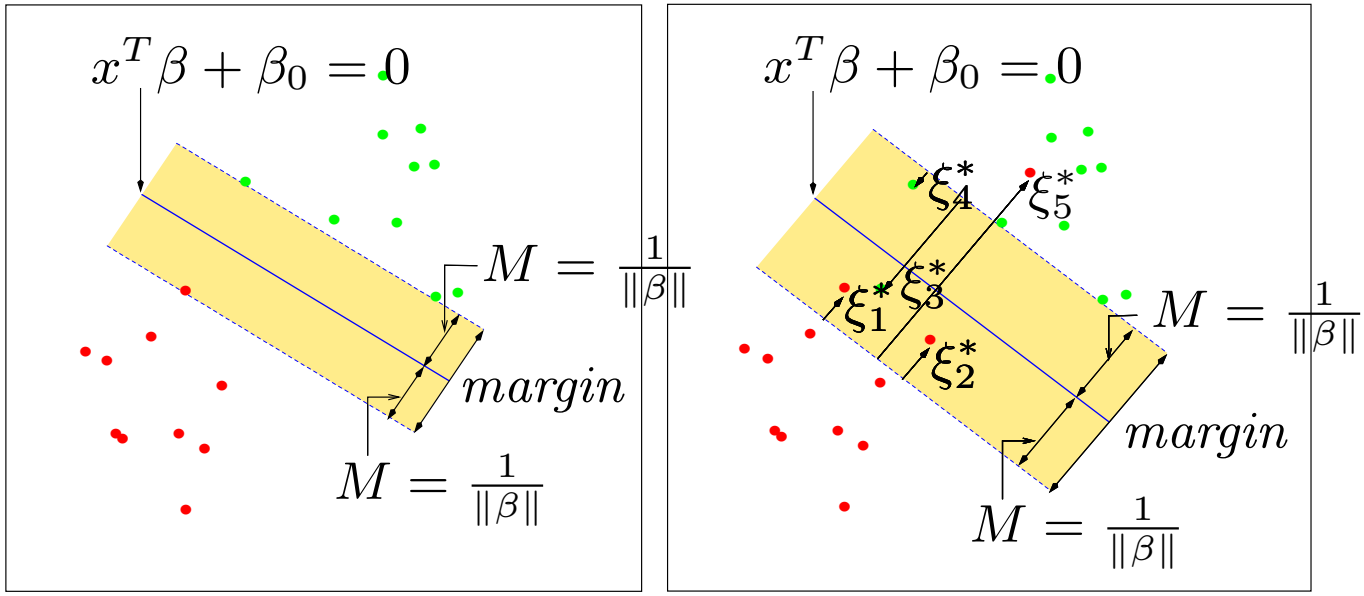


FIGURE 12.1. *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width* $2M = 2/\|\beta\|$. *The right panel shows the nonseparable (overlap) case. The points labeled* ξ_j^* *are on the wrong side of their margin by an amount* $\xi_j^* = M\xi_j$; *points on the correct side have* $\xi_j^* = 0$. *The margin is maximized subject to a total budget* $\sum \xi_i \leq$ *constant. Hence* $\sum \xi_j^*$ *is the total distance of points on the wrong side of their margin.*

Computation

Another form:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i \tag{273}$$

$$\text{s.t. } \xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, i = 1, \cdots, N. \tag{274}$$

Lagrange (primal) function is

$$L_P = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i, \tag{275}$$

which we minimize w.r.t. β, β_0 and ξ_i.

Constraints

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{276}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i \tag{277}$$

$$\alpha_i = C - \mu_i, \text{ for any } i. \tag{278}$$

And positivity constraints α_i, μ_i and $\xi_i \geq 0$, for all i. Then we obtain the Lagrangian (Wolfe) dual objective function

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_{i'}^T. \tag{279}$$

Simpler convex quadratic programming problem.

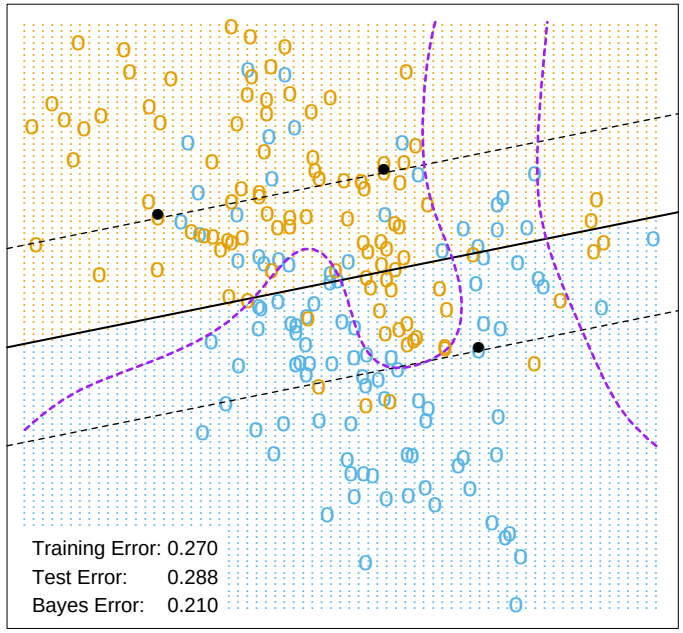
Interpretation of solution

We have $\hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i x_i$. Observations i with nonzero $\hat{\alpha}_i$ are called the *support vectors*.

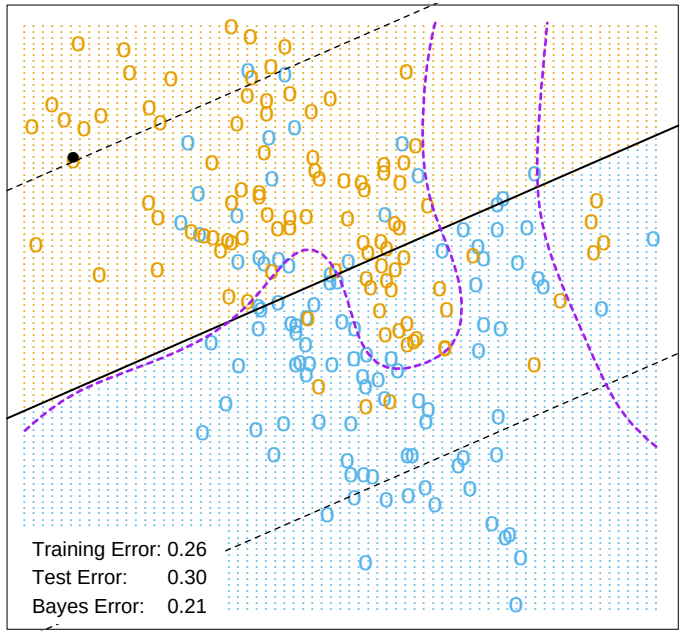
- $\hat{\xi}_i = 0$: lies on the edge of the margin, $0 < \hat{\alpha}_i < C$. Can be used to solve β_0.
- $\hat{\xi}_i > 0$: lies on the wrong side of the margin, $\hat{\alpha}_i = C$.

Given $\hat{\beta}_0$ and β, the decision function can be written as

$$\hat{G}(x) = \text{sign}[x^T \hat{\beta} + \hat{\beta}_0]. \tag{280}$$



$C = 10000$



$C = 0.01$

FIGURE 12.2. *The linear support vector bound*

SVM and kernels

Given basis functions $h_m(x), m = 1, \cdots, M$, we the use the derived features directly in the SVM formulation.

- Kernel based SVM: dimension of the enlarged space could be very large, even infinite.
- Closely related to the smoothing splines.

Inner products

Dual form:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} < h(x_i), h(x_{i'}) > . \tag{281}$$

Solution:

$$f(x) = h(x)^T \beta + \beta_0 \tag{282}$$

$$= \sum_{i=1}^{N} \alpha_i y_i < h(x), h(x_i) > + \beta_0. \tag{283}$$

Only requires the knowledge of the inner product (kernel function):

$$K(x, x') =< h(x), h(x') > . \tag{284}$$

Common kernel choices

- d-th degree polynomial: $K(x, x') = (1+ < x, x' >)^d$.
- Radial basis: $K(x, x') = \exp(-\gamma \|x - x'\|^2)$
- Neural network: $K(x, x') = \tanh(\kappa_1 < x, x' > +\kappa_2)$.

SVM as a penalization method

Consider the optimization problem

$$\min_{\beta_0,\beta} \sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + \frac{\lambda}{2}\|\beta\|^2, \tag{285}$$

hinge loss+penalty.

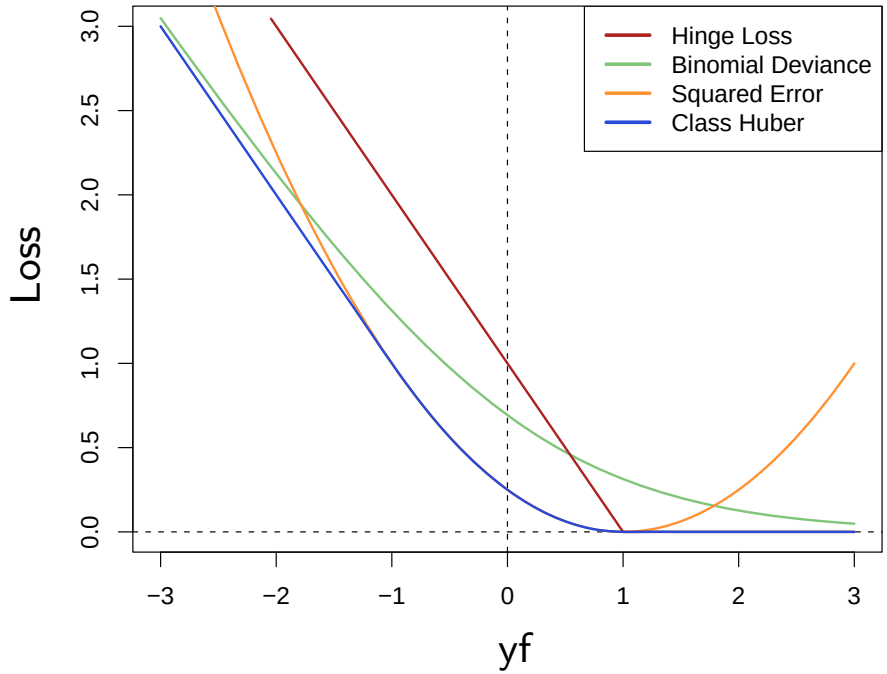


FIGURE 12.4. *The support vector loss function (hinge loss), compared to the negative log-likelihood loss (binomial deviance) for logistic regression, squared-error loss, and a "Huberized" version of the squared hinge loss. All are shown as a function of yf rather than f, because of the symmetry between the $y = +1$ and $y = -1$ case. The deviance and Huber have the same asymptotes as the SVM loss, but are rounded in the interior All are scaled to have the limiting left-tail slope of -1.*

different loss (margin maximizing loss-functions, except the sqaured-loss)(Rosset et al., 2004)

Loss Function	$L[y, f(x)]$	Minimizing Function
Binomial Deviance	$\log[1 + e^{-yf(x)}]$	$f(x) = \log \frac{\Pr(Y = +1\|x)}{\Pr(Y = \text{-}1\|x)}$
SVM Hinge Loss	$[1 - yf(x)]_+$	$f(x) = \text{sign}[\Pr(Y = +1\|x) - \frac{1}{2}]$
Squared Error	$[y - f(x)]^2 = [1 - yf(x)]^2$	$f(x) = 2\Pr(Y = +1\|x) - 1$
"Huberised" Square Hinge Loss	$-4yf(x)$, $yf(x) < \text{-}1$ $[1 - yf(x)]_+^2$ otherwise	$f(x) = 2\Pr(Y = +1\|x) - 1$

Function estimation and reproducing kernels

Suppose the basis h arises from the eigen-expansion tof a positive definite kernel K,

$$K(x, x') = \sum_{m=1}^{\infty} \phi_m(x)\phi_m(x')\delta_m \tag{286}$$

and $h_m(x) = \sqrt{\delta_m}\phi_m(x)$. With $\theta_m = \sqrt{\delta_m}\beta_m$, we can write

$$\min_{\beta_0,\theta} \sum_{i=1}^{N}[1 - y_i(\beta_0 + \sum_{m=1}^{\infty} \theta_m\phi_m(x_i)]_+ + \frac{\lambda}{2}\sum_{m=1}^{\infty}\frac{\theta_m^2}{\delta_m}. \tag{287}$$

Solution

$$f(x) = \beta_0 + \sum_{i=1}^{N} \alpha_i K(x, x_i). \tag{288}$$

General optimization criterion

$$\min_{\beta_0,\alpha} \sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + \frac{\lambda}{2} \alpha^T K \alpha, \tag{289}$$

where K is the $N \times N$ matrix of kernel evaluations of all pairs of training features.
More generally,

$$\min_{f \in \mathcal{H}} \sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + \lambda J(f), \tag{290}$$

where $\mathcal{H}$ is the structured space of functions.

SVM for regression

Consider minimization of

$$H(\beta, \beta_0) = \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2}\|\beta\|^2, \tag{291}$$

where $V_\epsilon(r) = (|r| - \epsilon)_+$.
Contrast with the Huber loss

$$V_H(r) = \begin{cases} r^2/2, & \text{if } |r| \le c \\ c|r| - c^2/2, & \text{if } |r| > c \end{cases} \tag{292}$$

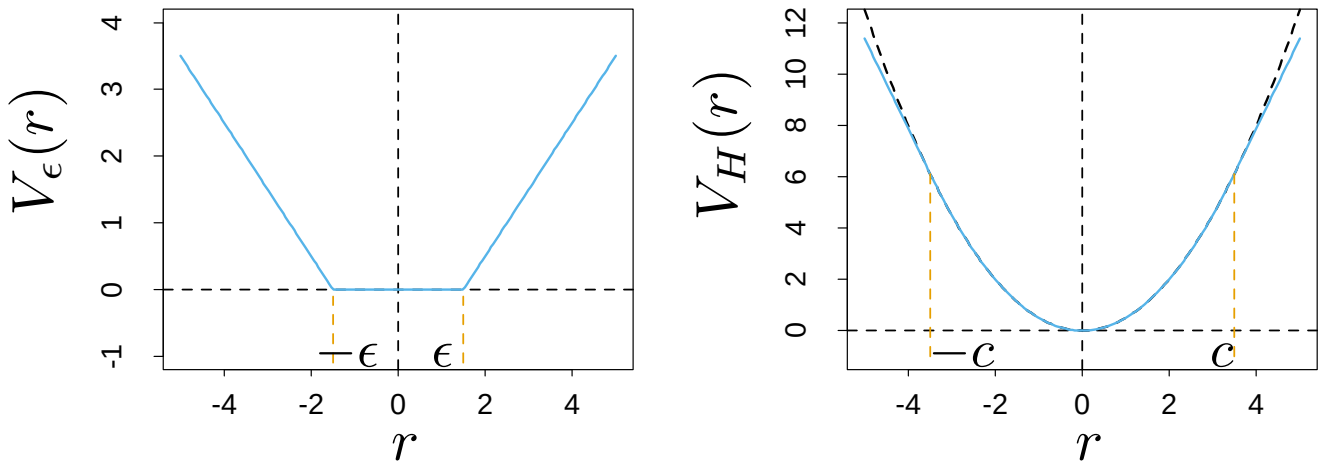


FIGURE 12.8. *The left panel shows the ϵ-insensitive error function used by the support vector regression machine. The right panel shows the error function used in Huber's robust regression (blue curve). Beyond $|c|$, the function changes from quadratic to linear.*

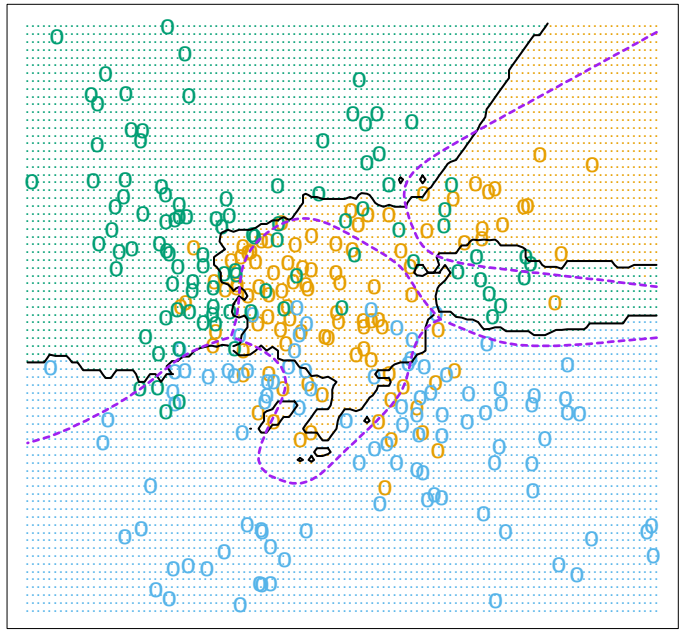
Outline

memory-based classifiers

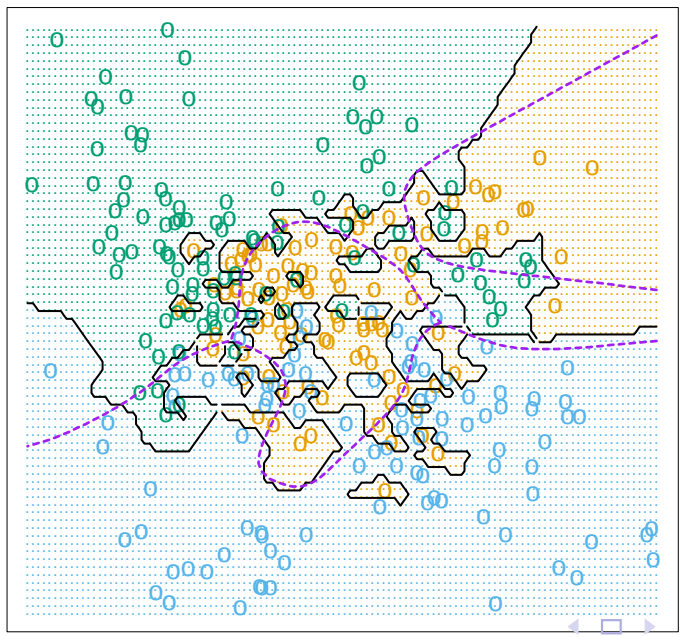
Require no model to be fit. Given a query point x_0, we find the k training points $x_{(r)}, r = 1, \cdots, k$ closest in distance to x_0 and classify using majority vote among k neighbors. Ties are broken at random.

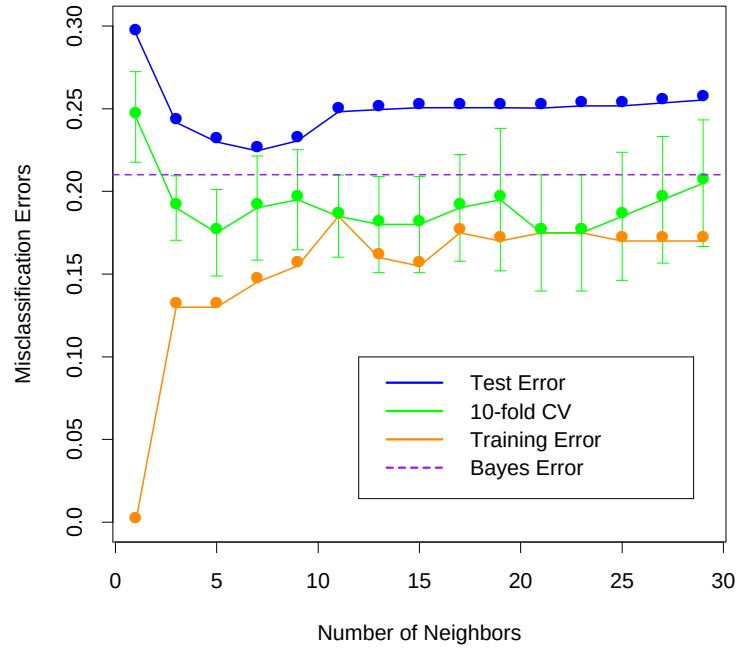
- Typically we first standardize each of the features to have mean zero and variance 1.
- Other distance measures can be considered.

15-Nearest Neighbors

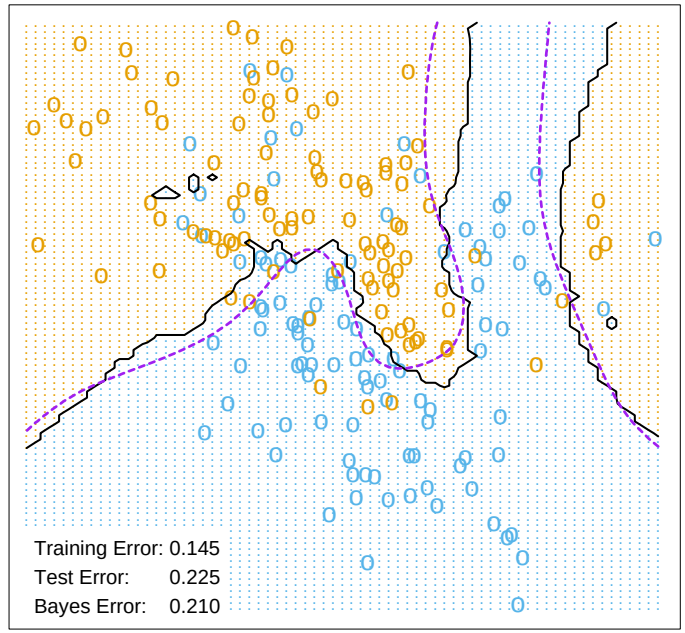


1-Nearest Neighbor





7-Nearest Neighbors



Error upper bound

Cover and Hart (1967)

Asymptotically the error rate of the 1-nearest-neighbor classifier is less than or equal to twice the Bayes rate.

At x, let k^* be the dominant class and $p_k(x)$ the true conditional probability for class k. Then we have

$$\text{Bayes error} = 1 - p_{k^*}(x), \tag{293}$$

$$\text{1-NN error} = \sum_{k=1}^{K} p_k(x)(1 - p_k(x)) \tag{294}$$

$$\geq 1 - p_{k^*}(x). \tag{295}$$

for $K = 2$, we have $\text{1-NN error} = 2p_{k*}(x)(1 - p_{k^*}(x)) \leq 2(1 - p_{k^*}(x))$.

Adaptive NN Methods

Consider N data points uniformly distributed in the unit cube $[-1/2, 1/2]^p$. Let R be the radius of a 1-nearest-neighborhood centered at the origin. Then

$$\text{median}(R) = v_p^{-1/p} \left(1 - (\frac{1}{2})^{1/N}\right)^{1/p}, \tag{296}$$

where $v_p r^p$ is the volume of the sphere of radius r in p dimensions.

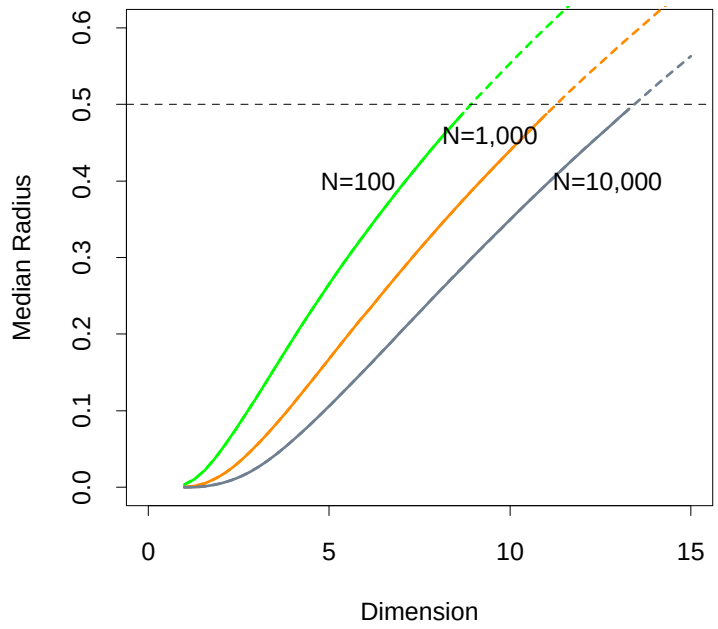


FIGURE 13.12. *Median radius of a 1-nearest-neighborhood, for uniform data with N observations in p dimensions.*

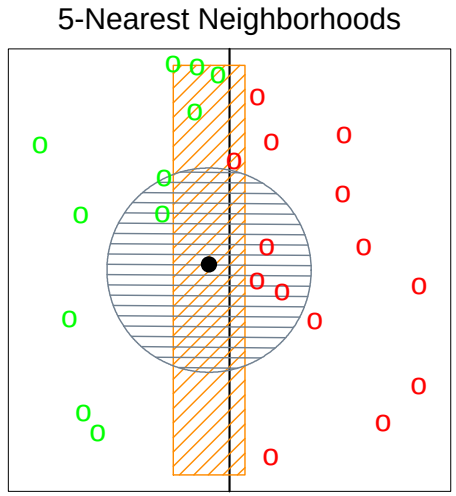


FIGURE 13.13. *The points are uniform in the cube, with the vertical line separating class red and green. The vertical strip denotes the 5-nearest-neighbor region using only the horizontal coordinate to find the nearest-neighbors for the target point (solid dot). The sphere shows the 5-nearest-neighbor region using both coordinates, and we see in this case it has extended into the class-red region (and is dominated by the wrong class in this instance).*

DANN

Discriminant adaptive nearest-neighbor (DANN) metric at a query x_0 is defined as

$$D(x, x_0) = (x - x_0)^T \Sigma (x - x_0), \tag{297}$$

where

$$\Sigma = W^{-1/2} \left[W^{-1/2} B W^{-1/2} + \epsilon I \right] W^{-1/2} \tag{298}$$

$$= W^{-1/2} \left[B^* + \epsilon I \right] W^{-1/2}. \tag{299}$$

W is pooled within-class covariance matrix, B is the between class covariance matrix, with W and B computed using only the 50 NN around x_0. $\epsilon > 0$ avoids using points far away from the query point when calculating the neighborhood.

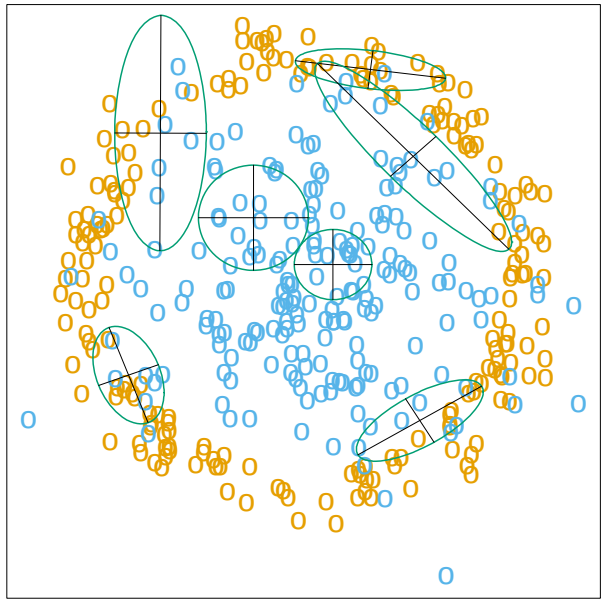


FIGURE 13.14. *Neighborhoods found by the DANN procedure, at various query points (centers of the crosses). There are two classes in the data, with one class surrounding the other.* 50 *nearest-neighbors were used to estimate the local metrics. Shown are the resulting metrics used to form* 15*-nearest-neighborhoods.*

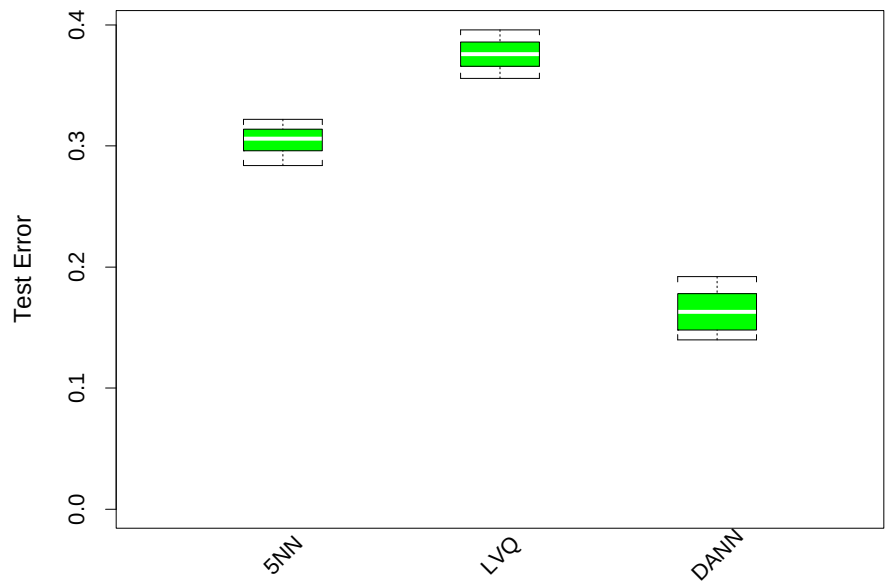


FIGURE 13.15. *Ten-dimensional simulated example: boxplots of the test error rates over ten realizations, for standard* 5*-nearest-neighbors, LVQ with* 50 *centers, and discriminant-adaptive* 5*-nearest-neighbors*