Notes on the book “Elements of Statistical Learning” by Trevor Hastie, Robert Tibshirani, and Jerome Friedman

Danny Horta

This document is a set of notes I created from studying statistics using this book. It is not intended to be a comprehensive tutorial or set of class lectures, but rather a compilation of notes. Feel free to use if you find it helpful. The complete book, including exercises for each chapter and detailed graphs can be found here.

Overview of Supervised Learning

1 Introduction

In any supervised learning model you have a set of *inputs* (or *predictors/ independent variables* in statistical language), that are measured or preset, that have some influence on one or more *outputs* (or *responses/dependent variables*). The outputs vary in nature depending on the problem. These can be either quantitative (e.g., a person's height), where a measurement can be bigger/smaller than others and the closer the measurement the closer they are in nature. Conversely they can be qualitative (or categorical/discrete, e.g., colour of their iris), and belong to a given finite set $\mathcal{G} = \{\text{Blue}, \text{Green}, \text{Brown}\}$.
In both cases, it makes sense to think of using inputs to predict outputs. Depending on the prediction task, these can be divided into: *regression* when we predict quantitative outputs, and *classification* when we predict qualitative outputs.
Typically, the input variable is denoted by $\boldsymbol{X}$ (usually a vector, of which components can be accessed via subscripts, X_i). Quantitative outputs are typically denoted as $\boldsymbol{Y}$, and qualitative outputs by $\boldsymbol{G}$ (for groups). Uppercase letters are used when referring to the generic aspects of a variable, whereas individual components of an input are denoted with lowercase letters, e.g., x_i. Matrices are denoted as bold uppercase letters (e.g., $\mathbf{X}$).
Supervised learning relies on having some *a priori* set of measurements (x_i, y_i), known as the *training data*, that are used to construct the prediction rule. Following, we describe the two fundamental approaches to supervised learning for regression and classification tasks.

2 Least squares and nearest neighbours

2.1 Linear models and least squares

The linear model has been a mainstay of statistics and is one of the most important tools. Given a vector of inputs $\boldsymbol{X} = [x_1, x_1, \cdots, x_p]$, we predict the output via the model

$$y_i = b + \sum_{i=1}^{N} x_i m_i. \tag{1}$$

Typically, the slope (or *weights* in machine-learning), m_i, and intercept (or *bias*), b, are denoted as β_i and β_0. However, to make this more relatable for classical linear regression introductions, we will set the bias to b and the weights to m_i.
Often it is useful to write this linear model in vector form as an inner product

$$\boldsymbol{Y} = \boldsymbol{X}^\top \theta, \tag{2}$$

where $\boldsymbol{X}^\top$ is the vector transpose of $\boldsymbol{X}$, and $\theta = \{\boldsymbol{m}, b\}$.

How do we fit a linear model to training data? There are different possible methods, but the most popular is the method of *least squares*. Here, the coefficients θ are picked to minimize the residual sum of squares

$$\mathrm{RSS}(\theta) = \sum_{i=1}^{N} (y_i - x_i^\top \theta)^2. \tag{3}$$

$\mathrm{RSS}(\theta)$ is a quadratic function of the parameters, and hence its minimum always exists (but may not be unique!). This solution is typically performed using matrix notation, such that

$$\mathrm{RSS}(\theta) = (\boldsymbol{Y} - \boldsymbol{X}\theta)^\top (\boldsymbol{Y} - \boldsymbol{X}\theta). \tag{4}$$

Differentiating this w.r.t. θ yields what is called the *normal equations*

$$\boldsymbol{X}^\top (\boldsymbol{Y} - \boldsymbol{X}\theta) = 0. \tag{5}$$

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

$$\theta = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}. \tag{6}$$

2.2 Dealing with noisy data

What happens if your measurements are not certain (i.e., have errors)? How do I fit a linear model to noisy data using least-squares? Turns out that if your training data (outputs) has Gaussian errors, regardless if it heteroscedastic (the errors on inputs in the training data are of different size), you can still solve analytically via least squares. Given a covariance matrix of Gaussian uncertainties

$$\boldsymbol{C} = \begin{bmatrix} \sigma_{y_1}, 0, 0, 0 \\ 0, \sigma_{y_2}, 0, 0 \\ 0, 0, \sigma_{y_3}, 0 \\ 0, \cdots, \cdots, 0 \\ 0, 0, 0, \sigma_{y_n} \end{bmatrix}, \tag{7}$$

then

$$\mathrm{RSS}(\theta) = \sum_{i=1}^{N} \frac{(y_i - x_i^\top \theta)^2}{\sigma_{y_i}^2} \equiv (\boldsymbol{Y} - \boldsymbol{X}\theta)^\top \boldsymbol{C^{-1}} (\boldsymbol{Y} - \boldsymbol{X}\theta). \tag{8}$$

Here we have assumed that there are no correlations between the variables (i.e., no values in the off-diagonal of the covariance matrix $\boldsymbol{C}$. For more information on how to fit a model to data see this paper.

3 Nearest-neighbour

Neares-neighbour methods use measurements of the input training data closest in input space to x to construct y. In more detail, the k-nearest neighbour fit for output y is defined as

$$y(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \tag{9}$$

where $N_k(x)$ is the neighbourhood of x defined by the k closest points x_i in the training sample. Here we assume that "closeness" is the Eucledian distance, but other metrics such as Manhattan distance and or hamming distance (for categorical data) can be used.
What do I do if I have categorical data? While there are more complex techniques out there to deal with categorical data, a possible work around is to turn your categorical data, say iris colour, into numerical data by assigning each category a number.
What do I do if my training data is noisy and measurements have errors? This is a very good question, and one that is swamped with a body of literature (see this paper and references therein for more details). Sophisticated methods will involve calculating a probabilistic distance metric to assign a probability of every input to a given k class. More rudimentary approaches involve bootstrapping the input measurement within the informed error margin and sampling. A possible simple approach (although this may not always work well practice) is a more empirical path that involves sampling each measurement within the associated error for s samples, running the k-nearest neighbour method for s samples, and taking the average of the output. This method would yield a distribution of clustering categories for every input (i.e., a distribution of $y(x)$).

4 Statistical models, supervised learning, and function approximation

4.1 A statistical model for the Joint distribution Pr(X, Y)

Suppose that a set of data arose from a statistical model of the form

$$\boldsymbol{Y} = f(\boldsymbol{X}) + \epsilon, \tag{10}$$

where the random error ϵ has an expectation value $\mathrm{E}(\epsilon) = 0$ and is independent of $\boldsymbol{X}$. The *additive error* model is a useful approximation to the truth. In most cases, the input-output pairs $(\boldsymbol{X}, \boldsymbol{Y})$ will not have a deterministic relationship ($\boldsymbol{Y} = f(\boldsymbol{X})$). Generally, there will be other unmeasured variables that contribute to $\boldsymbol{Y}$, including measurement errors. The additive model assumes that we can capture all these departures from a deterministic relationship via an error model ϵ.

4.2 Function approximation

The learning paradigm has become extremely fashionable in the fields of machine-learning and neural-networks. The approach inherited from statistics has been *function approximation.* Here, data pairs $\{x_i, y_i\}$ are viewed as points in a $(p+1)$-dimensional Eucledian space; the function $f(x)$ is related to the data via a model such as $\boldsymbol{y_i} = f(x_i) + \epsilon$. The goal is to then find a useful approximation (or transformation) of $f(x)$ for all x in some representation $\mathbb{R}^p$, given some data manifold $\mathbb{R}^n$.
Traditional examples of function approximation include *linear basis expansions*

$$f_0(x) = \sum_{k=1}^{K} h_k(x)\theta_k, \tag{11}$$

which include *polynomial and trigonometric* expansions (e.g., $x_1^2, x_1x_2^2, \cos(x_1)$), or non-linear expansions such as sigmoid transformation (common in neural-networks)

$$h_k(x) = \frac{1}{1+\exp(-x^\top\theta_k)}. \tag{12}$$

We can then use least squares to estimate the parameters θ in $f_0(x)$ as we dud fir tge kubear model by minimizing the residual of the sum-of-squares

$$\text{RSS}(\theta) = \sum_{i=1}^{N}(y_i - f_0(x_i))^2 \tag{13}$$

as a function of θ (as a reasonable criterion for an additive error model).

4.3 Maximum likelihood estimation, MLE

While least-squares is very convenient as it can be solved analytically and provides a straight-forward path to estimating the parameters in a model, it it not the only criterion used and in some cases would not make much sense to use (e.g., when using a highly non-linear model like a neural-net). A more general principle for estimation is *maximum likelihood estimation* (MLE).
Suppose we have a random sample $y_i, i = 1, \cdots, N$ from a denstiy $\Pr_\theta(y)$ indexed by some parameters θ. The log-probability of the observed sample is

$$L(\theta) = \sum_{i=1}^{N} \log \Pr{}_\theta(y_i). \tag{14}$$

The principle of MLE assumes that the most reasonable values for θ are those for which the probability of the observed sample is largest. Here, least squares for the aditive error model, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$, is equivalent to MLE using the conditional likelihood

$$\Pr(Y|X, \theta) = \mathcal{N}(f_\theta(x), \sigma^2). \tag{15}$$

Here, the log-likelihood of the data would be

$$L(\theta) = -\frac{N}{2}\log(2\pi) - N\log\sigma_i - \sum_{i=1}^{N}\frac{1}{2\sigma_i^2}(y_i - f_0(x_i))^2. \tag{16}$$

4.4 Restricted estimators

One can also envision adding terms to the loss function or in our case residual of sums squared to restrict the model. Restricted estimators are typically employed to control the size of the local (model) parameter neighbourhood, as there could be many possible solutions. These restrictors are also known as *smoothing* or *regularisation* parameters, and can be categorised into three broad classes: roughness penalty and Bayesian methods; kernel methods and local regression; and basis functions and dictionary methods. Here we only give a brief summary of each, as these will be covered in following chapters.

4.4.1 Roughness penalty and Bayesian methods

These are a class of functions controlled explicitly penalising $\mathrm{RSS}(f)$ with a roughness penalty, e.g., $\lambda J(f)$ or $\lambda\int[f''(x)]^2dx$, such that

$$\mathrm{PRSS}(f;\lambda) = \mathrm{RSS}(f) + \lambda J(f). \tag{17}$$

4.4.2 Kernel methods and local regression

These methods can be thought of as explicitly providing estimates of the regression funciton or conditional expectation by specifying the nature of the local (model) parameter neighbourhood. The local neighbourhood is specified by a *kernel function*, $K_\lambda(x_o, x)$, that assigns weights to points x in a region around x_0. For example, the Gaussian kernel has a weight function based on the Gaussian density function

$$K_\lambda(x_0, x) = \frac{1}{\lambda}\exp\left[-\frac{||x - x_0||^2}{2\lambda}\right] \tag{18}$$

and assigns weights to points that die exponentially with their squared Euclidean distance from x_0. λ corresponds to the variance of the Gaussian density, and controls the width of the neighbourhood. Another useful form for kernel estimate is the Nadaraya-Watson weighted average.
In general, the local regression estimate of $f(x_0)$ can be defined as $f_{\theta'}(x_0)$, where θ' minimises

$$\mathrm{RSS}(f_\theta, x_0) = \sum_{i=1}^{N} K_\lambda(x_o, x_i)(y_i - f_\theta(x_i))^2. \tag{19}$$

4.4.3 Basis functions and dictionary methods

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

$$f_\theta(x) = \sum_{m=1}^{M} \theta_m h_m(x), \tag{20}$$

where each of the h_m is a function of the input x, and the term linear here refers to the action of the parameters θ.

4.5 Model selection: the bias-variance trade-off

All the models described so far and many others, some of which will be discussed later, have a *smoothing* and a *complexity* parameter that has to be determined. These can be:

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

Generally speaking, as the *model complexity* increases, the variance also increases and the bias decreases. The opposite behaviour occurs when you opt for a simpler model (i.e., model complexity decreases). Typically, one would like to chose a model complexity that trades well between bias and variance in order to minimise test error (i.e., the performance of a model on some held-out test data).

Traditionally, although the training error (i.e., the performance of a model on training data) decreases with increasing model complexity, the test error has been shown to display an inflexion point once a model reaches a given complexity; in other words, test error decreases with increasing model complexity up to a given point, where test error begins to increase with increasing model complexity. This *is* the definition of bias-variance trade-off: finding the sweet-spot between model complexity and test error. However, the latest machine-learning literature has also shown that for large-language models (i.e., models will billions of parameters), if one keeps increasing model complexity, the test error will eventually continue to drop and decrease with increasing model complexity (this phenomenon is commonly referred to as the "*double-descent*", see this paper).

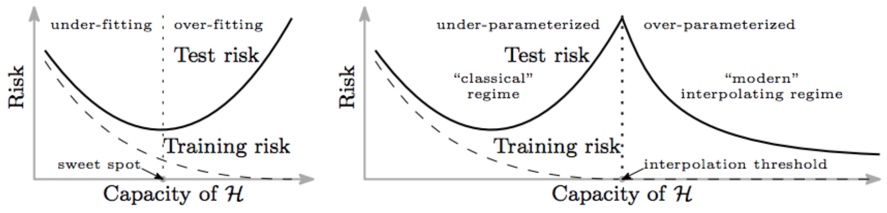

Figure 1: Illustration describing the double-descent phenomenon. Image taken from Belkin et al (2019)
.

Linear Methods for Regression

5 Introduction

A linear regression model assumes that the regression function $\mathrm{E}(Y|X)$ is linear in the inputs $X_1, \cdots, X_p$. For prediction purposes, despite their simplicity they can sometimes outperform fancier nonlinear models, especially in situations with small numbers of training cases, low signal-to-noise ratio or sparse data. Moreover, linear methods can be applied to transformations of the inputs to expand their scope.

6 Linear Regression Models and Least Square

A linear regression model has the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j, \tag{21}$$

where β_j's are the unknown parameters (or coefficients) and the variables X_j are the inputs (that come in many flavours: quantitative, transformations of quantitative, basis expansions, numeric or dummy codding of qualitative inputs).

As described in the previous chapter, the most popular estimation method for inferring the unknown parameters is *least squares*, which infers β_j by minimising some residual function (e.g., residual sum of squares). From a statistical point of view, this criterion is reasonable if the training observations (x_i, y_i) represent independent random draws from their populations. Even if the x_i's were not randomly drawn, the criterion is still valid if the y_i's are conditionally independent given the inputs x_i's.

Given the residual sum of squares formula in the least squares method

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} \Big(y_i - \beta_0 - \sum_{j=1}^{p} x_{i,j} \beta_j \Big)^2, \tag{22}$$

this equation can be rewritten in vector format

$$\mathrm{RSS}(\beta) = (\boldsymbol{y} - \boldsymbol{X}\beta)^\top (\boldsymbol{y} - \boldsymbol{X}\beta), \tag{23}$$

where $\boldsymbol{X}$ is the $N \times (p+1)$ matrix with each row an input vector (with 1 in the first position), and $\boldsymbol{y}$ is the N-vector of outputs in the training set. Differentiating w.r.t. β we get

$$\frac{\partial \mathrm{RSS}}{\partial \beta} = -2\boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{X}\beta) \tag{24}$$

$$\frac{\partial^2 \text{RSS}}{\partial^2 \beta} = 2\boldsymbol{X}^\top \boldsymbol{X}. \tag{25}$$

We can then set the first derivative to zero and rearange to obtain the unique solution

$$\hat{\beta} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{26}$$

Upon solving β, it is then straightforward to obtain $\boldsymbol{y}$ via $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\beta}$.

Assuming that observations y_i are uncorrelated and have constant variance σ_i^2, and that x_i are fixed, the variance-covariance matrix of the least squares parameter estimates is given by

$$\text{Var}(\hat{\beta}) = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\sigma^2. \tag{27}$$

Typically, one estimates the variance, σ^2, by

$$\hat{\sigma}^2 = \frac{1}{N-p-1}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{28}$$

where the $N-p-1$ rather than N on the denominator makes $\hat{\sigma}^2$ an unbiased estimate of σ^2.

To draw inferences about the parameters and the model, we need to assume that the model is correct for the mean; that is, the conditional expectation of Y is linear in $X_1, \ldots, X_p$, and that the deviations in Y around its expectation are additive and Gaussian. Thus

$$Y = \text{E}(Y|X_1, \cdots, X_p) + \epsilon = \beta_0 + \sum_{j=1}^{p} X_j\beta_j + \epsilon, \tag{29}$$

where the error ϵ is a Gaussian random variable with mean 0 and variance σ^2, $\epsilon \sim N(0, \sigma^2)$. Under this formulism,

$$\hat{\beta} \sim N(\beta, (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\sigma^2). \tag{30}$$

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

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{v_j}} \tag{31}$$

where v_j is the jth diagonal element of $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$. A large absolute value of z_j will lead to a rejection of the null hypothesis that $\beta_j = 0$.

Similarly, we can isolate β_j in 30 to obtain a $1 - 2\alpha$ confidence interval for β_j:

$$(\hat{\beta}_j - z^{(1-\alpha)} v_j^{1/2}\hat{\sigma}, \hat{\beta}_j + z^{(1-\alpha)} v_j^{1/2}\hat{\sigma}). \tag{32}$$

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

$$z^{(1-0.025)} = 1.96; z^{(1-0.05)} = 1.645, \text{etc.} \tag{33}$$

This is why the standard practice is to report $\hat{\beta} \pm 2 \cdot \mathrm{se}(\hat{\beta})$ which amounts to the $\sim 95\%$ confidence interval. In a similar fashion, we can obtain an approximate confidence set for the entire β parameter vector via

$$C_\beta = \{\beta | (\hat{\beta} - \beta)^\top \boldsymbol{X}^\top \boldsymbol{X} (\hat{\beta} - \beta) \leq \hat{\sigma}^2 \chi^2_{p+1} \cdot 1^{(1-\alpha)}\}, \tag{34}$$

where $\chi^2_\ell \cdot 1^{(1-\alpha)}$ is the $1 - \alpha$ percentile of the chi-suqared distribution on ℓ degrees of freedom.

6.1 The Gauss-Markov theorem

One of the most famous results in statistics asserts that the least squares estimates of the parameters β have the smallest variance among all linear unbiased estimates. More specifically, the Gauss-Markov theorem states that if we have any other linear estimator $\hat{\eta} = \boldsymbol{c}^\top \boldsymbol{y}$ that is unbiased for the parameters $\theta = \boldsymbol{a}^\top \beta$, then

$$\mathrm{Var}(a^\top \beta) \leq \mathrm{Var}(\boldsymbol{c}^\top \boldsymbol{y}). \tag{35}$$

7 Subset selection

There are two reasons why we are often not satisfied with the least squares estimates:

- *Prediction accuracy*: the least squares estimate often have low bias but large variance. Prediction accuracy can sometimes be improved by shrinking or setting some coefficients to 0, at the cost of increasing a little bit of bias.
- *Interpretation*: with a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects. In order to get the "big picture" we are willing to sacrifice some of the small details.

7.1 Best-subset selection

Best subset regression finds for each $k \in \{0, 1, 2, \cdots, p\}$ the subset of size k that gives the smallest residual sum of squares. The *leaps and bounds* procedure makes this feasible for $p \lesssim 40$. Alternatively, one can follow a cross-validation approach, whereby you use some training data to produce a sequence of accuracy estimates (e.g., residual sum of squares) using an array of models varying in complexity.

7.2 Forward- and backward-stepwise selection

Rather than searching through all possible subsets (which becomes unfeasible for models with $p >> 40$), we can seek a good path through them. *Forward-stepwise selection* starts with the intercept, and then sequentially adds to the model the predictor that most improves the fit. Forward-stepwise selection is a *greedy algorithm*, producing a nested sequence of models.

Conversely, *backward stepwise selection* starts with the full model, and sequentially deletes the predictor that has the least impact on the fit. The candidate for dropping is the variable with the smallest Z-score. This method can only be used when $N > p$, whereas forward-stepwise selection can always be used. Both methods provide a similar performance.

7.3 Forward-stagewise regression

Forward-stagewise regression is even more constrained than forward-stepwise regression. It starts as a forward-stepwise regression, with an intercept equal to $\bar{y}$ and centred on predictors with coefficients initially all set to 0. At each step the algorithm identifies the variable most correlated with the current residual. It then computes the simple linear regression coefficient for that variable, and this procedure is repeated until none of the variables have correlation with the residuals.

8 Shrinkage methods

In subset selection, because variables are either retained or discarded (it is a discrete process), it often exhibits high variance, and so doesn't reduce the prediction error of the full model. Shrinkage methods (or also known as regularisation methods) are more continuous, and don't suffer as much from high variability.

8.1 Ridge Regression or L2 regularisation

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

$$\hat{\beta}^{\text{ridge}} = \text{argmin}(\beta)\Big\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{i,j}\beta_j)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\Big\}. \tag{36}$$

Here $\lambda \geq 0$ is a complexity parameter that controls the amount of shrinkage (or regularisation). The larger the value of λ, the greater the amount of shrinkage. Here, the coefficients are shrunk toward 0 (and each other). In neural networks, this technique is also known as *weight decay*. Moreover, the penalisation is made *not* on the intercept β_0, as this would make the procedure depend on the origin chosen for $\boldsymbol{Y}$.

Writing the above formula in vector form, this would lead to

$$\text{RSS}(\lambda) = (\boldsymbol{y} - \boldsymbol{X}\beta)^\top(\boldsymbol{y} - \boldsymbol{X}\beta) + \lambda\beta^\top\beta, \tag{37}$$

where the ridge regression solutions would be

$$\hat{\beta}^{\text{ridge}} = (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y}, \tag{38}$$

where $\boldsymbol{I}$ is the $p \times p$ identity matrix.

The *singular value decomposition* (SVD) of the centred input matrix $\boldsymbol{X}$ gives us some additional insight into the nature of ridge regression, and can be used to find the *principal components* via *eigen decomposition* of $\boldsymbol{X}^\top \boldsymbol{X}$. The SVD of the $N \times p$ matrix $\boldsymbol{X}$ has the form

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top. \tag{39}$$

8.2 The Lasso regression or L1 regularisation

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

$$\hat{\beta}^{\text{ridge}} = \text{argmin}(\beta)\Big\{\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{i,j}\beta_j)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\Big\}. \tag{40}$$

The change from $\sum_1^p \beta_j^2$ ($L2$) to $\sum_1^p |\beta_j|$ ($L1$) makes the solutions non-linear in the y_i, encourages the coefficients to become 0, and there is no closed form expression as in ridge regression. Thus, computing the lasso solution is a **quadratic** problem.

8.3 Summary and when to use subset selection, ridge regression, and Lasso

Ridge regression does a proportionality shrinkage, whereas Lasso translates each coefficient by a constant factor λ, truncating at 0. Best-subset selection drops all variables with coefficients smaller than the Mth largest.

To compromise between Ridge and Lasso, Zou and Hastie (2005) introduced the *elasticnet* penalty

$$\lambda\sum_{j=1}^{p}(\alpha\beta_j^2 + (1-\alpha)|\beta_j|). \tag{41}$$

The elastic-net selects variables like Lasso, and shrinks together the coefficients of correlated predictors like ridge.

9 Methods using derived input directions

In many situations we have a large number of inputs that are often correlated. Here we will discuss methods that produce a small number of linear combinations of the inputs ($Z_m, m = 1, \cdots, M$) of the original inputs X_j; the Z_m are then used in pace of the X_j as inputs in the regression.

9.1 Principal components regression

Linear combinations Z_m used are the principal components. Here, principal component regression forms the derived input columns $\boldsymbol{z}_m = \boldsymbol{X}_{v_m}$, and then regresses $\boldsymbol{y}$ on $z_1, z_2, \cdots, z_M$ for some $M \leq p$. Since $\boldsymbol{z}_m$ are orthogonal, this regression is just a sum of univariate regressions:

$$\hat{\boldsymbol{y}}^{\text{pcr}}_{(M)} = \bar{y}1 + \sum_{m=1}^{M} \hat{\theta}_m \boldsymbol{z}_m, \tag{42}$$

where $\hat{\theta}_m = \langle \boldsymbol{z}_m, \boldsymbol{y} \rangle / \langle \boldsymbol{z}_m, \boldsymbol{z}_m \rangle$. Since the $\boldsymbol{z}_m$ are each linear combinations of the original x_j, we can express the solution to the equation above in terms of coefficients of the x_j

$$\hat{\beta}^{\text{pcr}}(M) = \sum_{m-1}^{M} \hat{\theta}_m v_m \tag{43}$$

9.2 Partial least squares

This technique also constructs a linear combination of the inputs for regression, but unlike PCR it uses $\boldsymbol{y}$ (in addition to $\boldsymbol{X}$) for this construction. It begins by computing $\hat{\phi}_{1,j} = \langle \boldsymbol{x_j}, \boldsymbol{y} \rangle$ for each j. From this we construct the derived input $\boldsymbol{z}_1 = \sum_j \hat{\phi}_{1,j} \boldsymbol{x}_j$, which is the first partial least squares direction. Hence in the construction of $\boldsymbol{z}_m$, the inputs are weighted by the strength of the univariate effect on $\boldsymbol{y}^3$.

Linear Methods for Classification

10 Introduction

Since a predictor $G(x)$ takes values in a discrete $\mathcal{G}$, we can always divide the input space into a collection of regions labelled according to the classification. The boundaries defining the regions can be rough or smooth; for an important class of procedures, these **decision boundaries** are linear (i.e., linear classification methods).

Suppose there are K classes labelled $1, 2, \cdots, K$, and the fitted linear model for the kth indicator response variable is $\hat{f}_k(x) = \hat{\beta}_{k0} + \hat{\beta}_k^\top x$. The decision boundary between class k and ℓ is that set of points for which $\hat{f}_k(x) = \hat{f}_\ell(x)$ (i.e., $\{x : (\hat{\beta}_{k0} - \hat{\beta}_{\ell 0}) + (\hat{\beta}_k - \hat{\beta}_\ell)^\top x = 0\}$). This regression approach is a member of a class of methods that model *discriminant functions* $\delta_k(x)$ for each class, and then classify x to the class with the largest value for its discriminant function. Similarly, methods that model the posterior probabilities $(\Pr(G = k|X = x)$ are also in this class.

All that is required for these sorts of models is that some monotone transformation of δ_k or $\Pr(G = k|X = x)$ be linear for the decision boundaries to be linear. Here is an example for two classes

$$\Pr(G = 1|X = x) = \frac{\exp(\beta_0 + \beta^\top x)}{1 + \exp(\beta_0 + \beta^\top x)} \tag{44}$$

$$\Pr(G = 2|X = x) = \frac{1}{1 + \exp(\beta_0 + \beta^\top x)}. \tag{45}$$

Here the monotone transformation is the *logit* transformation: $\log[p/(1-p)]$.

It is also possible to generalise the model, such that one could expand the variable set $X_1, X_2, \cdots, X_p$ by including non-linear terms (e.g., squares or cross-products); this example would add $p(p+1)/2$ variables. **Linear functions in the augmented space map down to quadratic functions in the original space.**

11 Linear Regression of an Indicator Matrix

Here each of the response categories (e.g., blue, brown, green for eye colours) are coded via some indicator varibale. Therefore, if $\mathcal{G}$ has K classes, there will be K such indicators $Y_k, k = 1, \cdots, K$, with $Y_k = 1$ if $G = k$ else 0. These are collected in a vector $\boldsymbol{Y} = (Y_1, \cdots, Y_k)$, and the N training instances of these form an $N \times K$ *indicator response matrix* $\boldsymbol{Y}$, which is a matrix of 0's and 1's. This can be fitted all at once using matrix notation as follows:

$$\hat{\boldsymbol{Y}} = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}. \tag{46}$$

Using this formulism, a new observation with input x is classified as follows:

1. compute the fitted output $\hat{f}(x)^\top = (1, x^\top)(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{Y}$, a K vector;
2. identify the largest component and classify accordingly: $\hat{G} = \text{argmax}_{k\in\mathcal{G}}\hat{f}(x)$.

Alternatively, a more simplistic approach is to construct *targets* t_k for each class, where t_k is the kth column of the $K \times K$ identity matrix. Here, the response vector y_i (ith row of $\boldsymbol{Y}$) for observation i has the value $y_i = t_k$ if $g_k = k$. This may be fit by least squares:

$$\min_{\boldsymbol{B}\sum_{i=1}^N ||y_i - [(1,x_i^\top)\boldsymbol{B}]^\top||^2}. \quad (47)$$

Under this formulism, a new observation is classified by computing its fitted vector $\hat{f}(x)$ and classifying to the closest target:

$$\hat{G}(x) = \arg\min_k ||\hat{f}(x) - t_k||^2. \tag{48}$$

Linear classification methods can encounter problems when the number of classes becomes $K \geq 3$. A loose but general rule is that if $K \geq 3$, polynomial terms up to degree $K-1$ might be needed.

12 Linear Discriminant Analysis

Decision theory for classification tells us that we need to know the posteriors $\Pr(G|X)$ for optimal classification. Suppose $f_k(x)$ is the class-conditional density of X in class $G = k$, and let π_k be the prior probability of class k, with $\sum_{k=1}^K \pi_k = 1$. A simple application of Bayes theorem yields

$$\Pr(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^K f_\ell(x)\pi_\ell}. \tag{49}$$

Suppose that we model each class density as multivariate Gaussian

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\sum_k|^{1/2}} e^{-1/2(x-\mu_k)^\top \sum_k^{-1}(x-\mu_k)}. \tag{50}$$

Linear discriminant analysis (LDA) arises in the special case when we assume that the classes have a common covariance matrix $\sum_k = \sum \forall k$. In comparing two classes k and ℓ, it is sufficient to look at the log-ratio to see that

$$\log\frac{\Pr(G = k|X = x)}{\Pr(G = \ell|X = x)} = \log\frac{f_k(x)}{f_\ell(x)} + \log\frac{\pi_k(x)}{\pi_\ell(x)} \tag{51}$$

$$= \log\frac{\pi_k(x)}{\pi_\ell(x)} - \frac{1}{2}(\mu_k - \mu_\ell)^\top(\sum)^{-1}(\mu_k - \mu_\ell) + x^\top(\sum)^{-1}(\mu_k - \mu_\ell). \tag{52}$$

From here, we see that the *linear discriminant functions* are

$$\delta_k(x) = x^\top(\sum)^{-1}\mu_k - \frac{1}{2}\mu_k^\top(\sum)^{-1}\mu_k + \log\pi_k, \tag{53}$$

where the parameters in the Gaussian will be determined with the training data.

With two classes there is a simple correspondence between LDA and classification by linear regression. The LDA rule classifies to class 2 if

$$x^\top \hat{\Sigma}^{-1}(\hat{\mu_2} - \hat{\mu_1}) > \frac{1}{2}(\hat{\mu_2} + \hat{\mu_1})^\top \hat{\Sigma}^{-1}(\hat{\mu_2} - \hat{\mu_1}) - \log(N_2/N_1) \tag{54}$$

and class 1 otherwise.

If however the $\sum_k$ are not assumed to be equal across classes, then the convenient cancellations in 8 and 9 do not occur. This then leads to a *quadratic discriminant functions* (QDA)

$$\delta_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k) + \log \pi_k. \tag{55}$$

12.1 Regularised Discriminant Analysis

Friedman (1989) proposed a compromise between LDA and QDA, allowing one to shrink the separate covariances of QDA toward a common covariance as in LDA (in a similar flavour to ridge regression). The regularised covariance matrices have the form

$$\hat{\Sigma}_k(\alpha) = \alpha\hat{\Sigma}_k + (1-\alpha)\hat{\Sigma}, \tag{56}$$

where $\hat{\Sigma}$ is the pooled covariance matrix as used in LDA, and $\alpha \in [0,1]$ allows a continuum of models between LDA and QDA and needs to be specified (chosen normally based on some cross-validation or performance on validation data).

12.2 Computations for LDA

The computations of LDA and QDA are simplified by diagonalising $\hat{\Sigma}$ or $\hat{\Sigma}_k$. For the latter, we suppose we compute the eigen-decomposition for each $\hat{\Sigma} = \boldsymbol{U}_k\boldsymbol{D}_k\boldsymbol{U}_k^\top$, where $\boldsymbol{U}_k$ is a $p \times p$ orthonormal, and $\boldsymbol{D}_k$ is a diagonal matrix of positive eigenvalues $d_{k\ell}$. Then, the ingredients for $\delta_k(x)$ are

- $(x-\hat{\mu}_k)^\top \hat{\Sigma_k^{-1}}(x-\hat{\mu}_k) = [\boldsymbol{U}_k^\top(x-\hat{\mu}_k)]^\top \boldsymbol{D}_k^{-1}[\boldsymbol{U}_k^\top(x-\hat{\mu}_k)]$;
- $\log|\hat{\Sigma}_k| = \sum_\ell \log d_{k\ell}$.

13 Logistic Regression

The logistic regression model arises from the desire to model the posterior probabilities of the K classes via linear functions in x, while at the same time ensuring that they sum to one and remain in [0,1]. The model has the form

$$\log\frac{\Pr(G=1|X=x)}{\Pr(G=K|X=x)} = \beta_{10} + \beta_1^\top x \tag{57}$$

$$\log\frac{\Pr(G=2|X=x)}{\Pr(G=K|X=x)} = \beta_{20} + \beta_2^\top x \tag{58}$$

$$\vdots \tag{59}$$

$$\log\frac{\Pr(G=K-1|X=x)}{\Pr(G=K|X=x)} = \beta_{(K-1)0} + \beta_{K-1}^\top x. \tag{60}$$

When $K=2$ the model is especially simple, since it becomes a binary classifier and a single linear function.

13.1 Fitting Logistic Regression Models

These are usually fit by maximum likelihood, using the conditional likelihood of G given X. Since $\Pr(G|X)$ completely specifies the conditional distribution, the *multinomial* distribution is appropriate. The log-likelihood for N observations is

$$\ell(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i;\theta), \tag{61}$$

where $p_k(x_i;\theta) = \Pr(G=k|X=x_i;\theta)$.

In the two-class case, the log-likelihood can be written as

$$\ell(\beta) = \sum_{i=1}^{N} \Big\{ y_i \log p(x_i;\beta) + (1-y_i)\log(1-p(x_i;\theta)) \Big\} \tag{62}$$

$$= \sum_{i=1}^{N} \Big\{ y_i \beta^\top x_i - \log(1+e^{\beta^\top x_i}) \Big\}. \tag{63}$$

To maximise the likelihood, we set its derivatives to 0. The *score* equations then are

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i(y_i - p(x_i;\beta)) = 0, \tag{64}$$

which are $p+1$ equations *nonlinear* in β. To solve the score equations, we use the Newton-Raphson algorithm, which requires the second-derivative (or the Hessian matrix)

$$\frac{\partial^2 \ell(\beta)}{\partial\beta\partial\beta^\top} = -\sum_{i=1}^{N} x_i x_i^\top p(x_i;\beta)(1-p(x_i;\beta)). \tag{65}$$

Starting with β^{old}, a single Newton update is

$$\beta^{\text{new}} = \beta^{\text{old}} - \Big(\frac{\partial^2\ell(\beta)}{\partial\beta\partial\beta^\top}\Big)^{-1}\frac{\partial\ell(\beta)}{\partial\beta}, \tag{66}$$

where the derivatives are evaluated at β^{old}.

Typically, it is convenient to write the score and Hessian in matrix notation. Let $\boldsymbol{y}$ denote the vector of y_i values, $\boldsymbol{X}$ the $N \times (p+1)$ matrix of x_i values, $\boldsymbol{p}$ the vector of fitted probabilities with the ith element $p(x_i; \beta^{\text{old}})$, and $\boldsymbol{W}$ a $N \times N$ diagonal matrix of weights with ith diagonal element $p(x_i; \beta^{\text{old}})(1 - p(x_i; \beta^{\text{old}}))$. Under this notation, we then have

$$\frac{\partial \ell(\beta)}{\partial \beta} = \boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{p}) \tag{67}$$

$$\frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta^\top} = -\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}. \tag{68}$$

The Newton step is thus

$$\beta^{\text{new}} = \beta^{\text{old}} + (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{p}) \tag{69}$$

$$= (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{W} (\boldsymbol{X} \beta^{\text{old}} + \boldsymbol{W}^{-1}(y - p)) \tag{70}$$

$$= (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{z}. \tag{71}$$

In the second and third line above we have re-expressed the Newton step as a weighted least squares step, with the response

$$\boldsymbol{z} = \boldsymbol{X} \beta^{\text{old}} + \boldsymbol{W}^{-1}(\boldsymbol{y} - \boldsymbol{p}), \tag{72}$$

sometimes known as the *adjusted response*. These equations get solved repeatedly as $\boldsymbol{p}$, $\boldsymbol{W}$, and $\boldsymbol{z}$ change at each iteration. This procedure is also known as *iteratively reweighted least squares* or (IRLS).

13.2 Quadratic Approximations and Inference

The maximum-likelihood parameter estimates, $\hat{\beta}$, satisfy a self-consistency relationship: they are the coefficients of a weighted least squares fit, where the responses are

$$z_i = x_i^\top \hat{\beta} + \frac{(y_i - \hat{p}_i)}{\hat{p}_i(1 - \hat{p}_i)}, \tag{73}$$

and the weights are $w_i = \hat{p}_i(1 - \hat{p}_i)$. Additionally, the weighted residual sum-of-squares is the Pearson chi-square statistic

$$\sum_{i=1}^{N} \frac{(y_i - \hat{p}_i)^2}{\hat{p}_i(1 - \hat{p}_i)}. \tag{74}$$

13.3 L_1 Regularised Logistic Regression

The penalty L_1 used in the Lasso can be used for variable selection and shrinkage with any linear regression model. With logistic regression, we could maximize the following relation

$$\max_{\beta_0, \beta} \Big\{ \sum_{i=1}^{N} [y_i(\beta_0 + \beta^\top x_i) - \log(1 + e^{\beta_0 + \beta^\top x_i})] - \lambda \sum_{j=1}^{p} |\beta_j| \Big\}. \tag{75}$$

Basis Expansions and Regularisation

14 Introduction

While extremely useful (especially in the cases where p is large and N is small), linear models for regression and classification do not always fit the bill. In this chapter we will discuss methods for moving beyond linearity. The core idea here is to augment/replace the vector of inputs $\boldsymbol{X}$ with additional variables (i.e., transformations of $\boldsymbol{X}$), and then use linear models in this new space of derived input features.

Denote by $h_m(\boldsymbol{X}) : \mathbb{R}^p \to \mathbb{R}$ the mth transformation of $\boldsymbol{X}$, $m = 1, \cdots, M$. We then model

$$f(\boldsymbol{X}) = \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{X}), \tag{76}$$

as a *linear basis expansion* in $\boldsymbol{X}$. The beauty of this approach is that once the basis functions h_m are determined, the models are linear in these new variables. For example, $h_m(\boldsymbol{X}) = \boldsymbol{X}_m; h_m(\boldsymbol{X}) = \boldsymbol{X}_j^2; h_m(\boldsymbol{X}) = \boldsymbol{X}_j \boldsymbol{X}_k; h_m(\boldsymbol{X}) = \log(\boldsymbol{X}_j)$.

Typically, the augmentation of linear models to achieve more flexible representations of $f(\boldsymbol{X})$ are in the form of *piecewise-polynomials* and *splines*. It is also possible to use *wavelet* bases (especially for signals and images). These methods produce a *dictionary* $\mathcal{D}$ consisting typically of a very large number of $|\mathcal{D}|$ basis functions, alongside a method for controlling the complexity of the model. There are three common approaches:

- Restriction methods, where an *a priori* decision has been made to limit the class of functions (additivity, for example).
- Selection methods, that adaptively scan the dictionary and include only those basis functions h_m that contribute significantly to the fit of the model.
- Regularisation methods, where the entire dictionary is used but restrict the coefficients.

15 Piecewise Polynomials and Splines

Assuming that X is one-dimensional, a piece-wise polynomial function $f(X)$ is obtained by dividing the domain of X into contiguous intervals, and representing f by a separate polynomial in each interval.

A common example is the *cubic spline*, which can be written as

$$h_1(X) = 1, h_3(X) = X^2, h_5(X) = (X - \xi_1)_+^3 \tag{77}$$
$$h_2(X) = X, h_4(X) = X^3, h_6(X) = (X - \xi_2)_+^3 \tag{78}$$
$$\tag{79}$$

where ξ_1, ξ_2 are the knots. Here, there are 3 regions $\times$ 4 parameters per region $-$ 2 knots $\times$ 3 constraints per knot = 6; hence this 6 dimensional basis function corresponds to a six-dimensional linear space of functions.

More generally, an M order spline with knots $\xi_j, j = 1, \cdots, K$ is a piecewise polynomial of order M, and has continuous derivatives up to order $M-2$. A cubic spline has $M = 4$.

15.1 Natural Cubic Splines

The behaviour of polynomial fits to data tends to be erratic near the boundaries, leading to dangerous extrapolation. These issues are exacerbated with splines.

A *natural cubic spline* adds additional constraints, namely that the function is linear beyond the boundary knots. This makes extrapolation around the boundaries more reasonable, at the cost of additional bias.

A natural cubic spline with K knots is represented by K basis functions. For example, starting from a truncated power series,

$$N_1(X) = 1, N_2(X) = X, N_{k+2}(X) = d_k(X) - d_{k-1}(X), \tag{80}$$

where

$$d_k(X) = \frac{(X - \xi_k)_+^3 - (X - \xi_K)_+^3}{\xi_K - \xi_k}. \tag{81}$$

16 Filtering and Feature Extraction

Preprocessing of high-dimensional features is a very general and powerful method for improving the performance of a learning algorithm. The preprocessing does not need to be linear, but can be non-linear too, such that $x^* = g(x)$. The derived features x^* can then be used as inputs into any learning procedure.

17 Smoothing Splines

We now discuss a spline basis method that avoids the knot selection problem completely by using a maximal set of knots. The complexity of the fit is controlled by regularisation. Let's consider the following problem: among all functions $f(x)$ with two continuous derivatives, find one that minimizes the penalised residual sum of squares

$$\text{RSS}(f, \lambda) = \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda \int \{f^{''}(t)\}^2 dt, \tag{82}$$

where λ is a fixed smoothing parameter. The first term measures closeness to the data, while the second term penalises curvature in the funciton ($L2$), and λ establishes a trade-off between the two. Two special cases are:

- $\lambda = 0$: f can be any function that interpolates the data.

- $\lambda = \infty$: the simple least squares line fit, since no second derivative can be tolerated.

Remarkably, it can be shown that the above equation has an explicit, finite-dimensional, unique minimizer which is a natural cubic spline with knots at the unique values of the $x_i, i = 1, \cdots, N$.

Since the solution is a natural spline, we can write it as

$$f(x) = \sum_{i=1}^{N} N_j(x)\theta_j, \tag{83}$$

where $N_j(x)$ are an N-dimensional set of basis functions for representing this family of natural splines. The criterion then reduces to

$$\mathrm{RSS}(\theta, \lambda) = (\boldsymbol{y} - \boldsymbol{N}\theta)^\top(\boldsymbol{y} - \boldsymbol{N}\theta) + \lambda\theta^\top\boldsymbol{\Omega}_N\theta, \tag{84}$$

where $\{\boldsymbol{N}\}_{ij} = N_j(x_i)$ and $\{\boldsymbol{\Omega_N}\}_{jk} = \int N_j^{''}(t)N_k^{''}(t)dt$. The solution is easily seen to be

$$\hat{\theta} = (\boldsymbol{N}^\top\boldsymbol{N} + \lambda\boldsymbol{\Omega}_N)^{-1}\boldsymbol{N}^\top\boldsymbol{y}, \tag{85}$$

a generalised ridge regression. The fitted smoothing spline is given by

$$\hat{f}(x) = \sum_{i=1}^{N} N_j(x)\hat{\theta}_j. \tag{86}$$

The choice of λ is typically set via cross-validation.

18 Automatic Selection of the Smoothing Parameters

The smoothing parameters for regression splines encompass the degree of the splines, and the number and placement of the knots. For smoothing splines, we use λ (the penalty parameter). However, the placement and number of knots for regression splines can be a tricky task, unless some specification is enforced. This will be discussed further in upcoming sections and chapters.

19 Nonparametric Logistic Regression

The technology in the smoothing spline problem can be transferred to other domains. Here we consider logistic regression with a single quantitative input X. The model is

$$\log\frac{\Pr(Y = 1|X = x)}{\Pr(Y = 0|X = x)} = f(x), \tag{87}$$

which implies

$$\Pr(Y = 1|X = x) = \frac{e^{f(x)}}{1 + e^{f(x)}}. \tag{88}$$

Fitting $f(x)$ in a smooth fashion leads to a smooth estimate of the conditional probability $\Pr(Y=1|x)$, which can be used for classification or risk scoring.

We construct the penalised log-likelihood criterion

$$\ell(f;\lambda) = \sum_{i=1}^{N}[y_i \log p(x_i) + (1-y_i)\log(1-p(x))] - \frac{1}{2}\lambda \int \{f^{''}(t)\}^2 dt \tag{89}$$

$$= \sum_{i=1}^{N}[y_i f(x_i) - \log(1+e^{f(x_i)})] - \frac{1}{2}\lambda \int \{f^{''}(t)\}^2 dt, \tag{90}$$

where we abbreviated $p(x) = \Pr(Y=1|x)$. The optimal f is a finitie-dimensional natural spline with knots at the unique values of x. This means that we can represent $f(x) = \sum_{j=1}^{N} N_j(x)\theta_j$. Computing the first and second order derivatives

$$\frac{\partial \ell(\theta)}{\partial \theta} = \boldsymbol{N}^\top(\boldsymbol{y}-\boldsymbol{p}) - \lambda\boldsymbol{\Omega}\theta \tag{91}$$

$$\frac{\partial^2 \ell(\theta)}{\partial\theta\partial\theta^\top} = -\boldsymbol{N}\boldsymbol{W}\boldsymbol{N} - \lambda\boldsymbol{\Omega}, \tag{92}$$

where $\boldsymbol{p}$ is the N-vector with elements $p(x_i)$, and $\boldsymbol{W}$ is a diagonal matrix of weights $p(x_i)(1-p(x_i))$. Since the first derivative is non-linear in θ, we solve via an iterative algorithm. Using the Newton-Raphson method, the update equation can be written as

$$\theta^{\text{new}} = (\boldsymbol{N}^\top\boldsymbol{W}\boldsymbol{N} + \lambda\boldsymbol{\Omega})^{-1}\boldsymbol{N}^\top\boldsymbol{W}(\boldsymbol{N}\theta^{\text{old}} + \boldsymbol{W}^{-1}(\boldsymbol{y}-\boldsymbol{p})) \tag{93}$$

$$= (\boldsymbol{N}^\top\boldsymbol{W}\boldsymbol{N} + \lambda\boldsymbol{\Omega})^{-1}\boldsymbol{N}^\top\boldsymbol{W}\boldsymbol{z}. \tag{94}$$

20 Multidimensional Splines

Suppose $X \in \mathbb{R}^2$, and we have a basis of function $h_{1k}(X_1), k = 1, \cdots, M_1$ for representing functions of coordinate X_1, and likewise a set of M_2 functions $h_{2k}(X_2)$ for coordinate X_2. Then the $M_1 \times M_2$ dimensional *tensor product basis* defined by $g_{jk} = h_{1j}(X_1)h_{2k}(X_2), j = 1, \cdots, M_1, k = 1, \cdots, M_2$ can be used for representing a two-dimensional function:

$$g(X) = \sum_{j=1}^{M_1}\sum_{k=1}^{M_2} \theta_{jk} g_{jk}(X). \tag{95}$$

Note that we could also include a regularisation to the objective function to make this multi-dimensional spline smoother. Thus, one-dimensional smoothing splines (via regularisation) generalise to higher dimensions well. In this scenario, the idea is to set up the problem such that

$$\min_f \sum_{i=1}^{N}\{y_i - f(x_i)\}^2 + \lambda J[f], \tag{96}$$

where J is an approximate penalty functional for stabilising a function f in $\mathbb{R}^d$.

21 Wavelet Smoothing

Insofar, we have seen two different modes of operation with dictionaries of basis functions: with regression splines, we select a subset of the bases; with smoothing splines, we use a complete basis, but then shrink the coefficients towards smoothness.

Wavelets typically use a complete orthonormal basis to represent functions, but then shrink and select the coefficients toward a sparse representation. Wavelet bases are very popular in signal processing and compression, since they are able to represent both smooth and/or locally bumpy functions in an efficient way.

Wavelet smoothing fits the coefficients for the basis by least squares, and then thresholds (discards) the smaller coefficients.

21.1 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*). If $\phi(x) = I(x \in [0,1])$, then $\phi_{0,k}(x) = \phi(x-k)$, k an integer, generates an orthonormal[1] basis for functions with jumps at the integers. Call this reference space V_0. The dilations $\phi_{1,k}(x) = \sqrt{2}\phi(2x-k)$ form an orthonormal basis for a space $V_1 \supset V_0$ of functions piecewise constant on intervals of length 1/2. More generally, we have $\cdots \supset V_1 \supset V_0 \supset V_{-1} \supset \cdots$ where each V_j is spanned by $\phi_{j,k} = 2^{j/2}\phi(2^j x - k)$.

Suppose we represent a function in V_{j+1} by a component in V_j plus the component in the orthogonal complement W_j of V_j to V_{j+1}, written as $V_{j+1} = V_j \oplus W_j$. The component in W_j represents detail, and we might wish to set some elements of this component to zero.

There are many different types of Wavelet basis, but some of the most used include the Haar, the Daubechies, and the Symmlet basis.

21.2 Adaptive Wavelet Filtering

Wavelets are particularly useful when the data are measured on a uniform lattice, such as a discretised signal, image, or time series.

Suppose $\boldsymbol{y}$ is a response vector, and $\boldsymbol{W}$ is the $N \times N$ orthonormal wavelet basis matrix evaluated at the n uniformly spaced observations. Then $\boldsymbol{y}^* = \boldsymbol{W}^\top \boldsymbol{y}$ is called the *wavelet transform* of $\boldsymbol{y}$. A populat method for adaptive wavelet fitting is known as *SURE shrinkage*. We start with the criterion

$$\min_\theta ||\boldsymbol{y} - \boldsymbol{W}\boldsymbol{\theta}||_2^2 + 2\lambda||\theta||_1, \tag{97}$$

which sis the same as the Lasso. Because $\boldsymbol{W}$ is orthonormal, this leads to the simple solution:

$$\hat{\theta}_j = \operatorname{sign}(y_j^*)(|y_j^*| - \lambda)_+. \tag{98}$$

[1] "Orthonormal" refers to a set of vectors that are both orthogonal (perpendicular to each other) and normalized (each vector has a length of 1).

The least squares coefficients are translated toward zero ($L1$), and truncated at zero. The fitted function (vector) is then given by the *inverse wavelet transform* $\hat{\boldsymbol{f}} = \boldsymbol{W}\hat{\boldsymbol{\theta}}$.

Here, a simple choice for λ is $\lambda = \sigma\sqrt{2\log N}$, where σ is an estimate of the standard deviation of the noise.

The space $\boldsymbol{W}$ could be any basis of orthonormal functions: polynomials, natural splines or cosinusoids.

Kernel Smoothing Methods

22 Introduction

We will now describe a class of regression techniques that achieve flexibility in estimating the regression function $f(X)$ over the domain $\mathbb{R}^p$ by fitting a different but simple model separately at each query point x_0. This is done by using only those observations close to the target point x_0 to fit the simple model, and in such a way that the resulting estimated function $\hat{f}(X)$ is smooth in $\mathbb{R}^p$. This localisation is achieved via a weighting function or *kernel* $K_\lambda(x_0, x_i)$, which assigns a weight to x_i based on its distance from the width of the neighbourhood.

22.1 One-Dimensional Kernel Smoothers

In Chapter 2, we motivated the k-nearest neighbour average

$$\hat{f}(X) = \text{Ave}(y_i | x_i \in N_k(x)) \tag{99}$$

as an estimate of the regression function $\text{E}(Y|X = x)$. Here $N_k(x)$ is the set of k points nearest to x in squared distance, and Ave denotes the average (mean). The idea here is to relax the definition of conditional expectation and compute an average in a neighbourhood of the target point.

Rather than give all points in a neighbourhood of a set equal weight, we can assign weights that die off smoothly with distance from the target point. Using the Nadaraya-Watson kernel-weighted average

$$\hat{f}(x_0) = \frac{\sum_{i=1}^N K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^N K_\lambda(x_0, x_i)} \tag{100}$$

with the Epanechnikov quadratic kernel

$$K_\lambda(x_0, x) = D\Big(\frac{|x - x_0|}{\lambda}\Big) \tag{101}$$

with

$$D(t) = \Big\{ 3/4(1 - t^2) \text{ if } |t| \leq 1; 0 \text{ otherwise.} \tag{102}$$

One can also use adaptive neighbourhooods with kernels. Let $h_\lambda(x_0)$ be a width function (indexed by λ) that determnes the width of the neighbourhood at x_0. Then more generally we have

$$K_\lambda(x0, x) = D\Big(\frac{|x - x_0|}{h_\lambda(x_0)}\Big). \tag{103}$$

Some considerations need to be taken in practice:

- The smoothing parameter λ, which determines the width of the local neighbourhood, has to be determined. Large λ implies lower variance but higher bias.

- Metric window widths (constant $h_\lambda(x)$ tend to keep the bias of the estimate constant, but the variance is inversely proportional to the local density.
- Issues arise with nearest-neighbours when there are ties in the x_i. With most smoothing techniques one can simply reduce the data set by averaging the y_i at tied values of X, and supplementing these at values of x_i with an additional weight w_i. Operationally, weights are multiplied by the kernel weights before computing the weighted average.
- Boundary issues arise.
- The Epanechnikov kernel has compact support (needed when used with nearest-neighbour window size). Another popular compact kernel is based on the tri-cube function

$$D(t) = \Big\{(1-|t|^3)^3 \text{ if } |t| \leq 1; 0 \text{ otherwise.} \tag{104}$$

  The Gaussian density function $D(t) = \phi(t)$ is a popular noncompact kernel, with standard deviation playing the role of window size.

22.2 Local Linear Regression

Locally weighted averages can be badly biased on the boundaries of the domain, because of the asymmetry of the kernel in that region. By fitting straight lines rather than constants locally, we can remove this bias to the first order.

Locally weighted regression solves a separate weighted least squares problem at each target point x_0:

$$\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i)[y_i - \alpha(x_0) - \beta(x_0)x_i]^2. \tag{105}$$

The estimate is then $\hat{f}(x_0) = \hat{\alpha}(x_0) + \hat{\beta}(x_0)x_0$.

Define the vector values function $b(x)^\top = (1, x)$. Let $\boldsymbol{B}$ be the $N \times 2$ regression matrix with ith row $b(x_i)^\top$, and $\boldsymbol{W}(x_0)$ the $N \times N$ diagonal matrix with ith diagonal element $K_\lambda(x_0, x_i)$. Then

$$\hat{f}(x_0) = b(x_0)^\top(\boldsymbol{B}^\top \boldsymbol{W}(x_0)\boldsymbol{B})^{-1}\boldsymbol{B}^\top \boldsymbol{W}(x_0)\boldsymbol{y} \tag{106}$$

$$= \sum_{i=1}^{N} l_i(x_0)y_i. \tag{107}$$

The weights $l_i(x_0)$ combine the weighting kernel $K_\lambda(x_0, \cdot)$ and the least squares operations, and are sometimes referred to as the *equivalent kernel*.

22.3 Local Polynomial Regression

We can fit local polynomial fits of any degree d

$$\min_{\alpha(x_0),\beta(x_0),j=1,\cdots,d} \sum_{i=1}^{N} K_\lambda(x_0, x_i)[y_i - \alpha(x_0) - \sum_{j=1}^{d} \beta_j(x_0)x_i^j]^2, \qquad (108)$$

with solution $\hat{f}(x_0) = \hat{\alpha}(x_0) + \sum_{j=1}^{d} \hat{\beta}_j(x_0)x_0^j$.

Local linear fits tend to be biased in regions of curvature of the true function, a phenomenon referred to as *trimming the hills* and *filling the valleys*. Local quadratic regression is generally able to correct this bias.

There is of course a price to be paid for this decreased bias, and that comes in the form of increased variance. To summarise this section:

- Local linear fits can help bias dramatically at the boundaries at a modest cost of variance. Local quadratic fits do a little at the boundaries for bias, but ioncrease variance a lot.
- Local quadratic fits tend to be most helpful in reducing bias due to curvature in the interior of the domain.
- Asymptotic analysis suggest that local polynomials of odd degree those of even degree.

23 Selecting the Width of the Kernel

In each of the kernels K_λ, λ is a parameter that controls the width. For example:

- For the Epanechnikov or tri-cube kernel, λ is the radius of the support region.
- For the Gaussian kernel, λ is the standard deviation.
- λ is the number k of nearest neighbours in k-nearest neighbourhoods.

There is a natural bias-variance tradeoff as we change the width of the averaging window, which is most explicit for local averages:

- If the window is narrow, $\hat{f}(x_0)$ is an average of a small number of y_i close to x_0. The variance will be relatively large, whereas the bias will tend to be small.
- If the window is wide, the variance of $\hat{f}(x_0)$ will be small, whereas the bias will be higher.

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

Kernel smoothing and local regression generalise naturally to two or more dimensions. The Nadaraya-Watson kernel smoother fits a constant locally with weights supplied by a p-dimensional kernel. Local linear regression fits a hyperplane locally in X, by weighted least squares, with weighted supplied by a p-dimensional kernel.

Let $b(X)$ be a vector of polynomial terms in X of maximum degree d. At each $x_0 \in \mathbb{R}^p$ solve

$$\min_{\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - b(x_i 0^\top \beta(x_0))^2 \tag{109}$$

to produce the fit $\hat{f}(x_0) = b(x_0)^\top \hat{\beta}(x_0)$. Note that in higher dimensions, the boundary problems become much bigger, since the fraction of points at the boundary is larger.

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

When the dimension to sample-size ratio is unfavourable (much larger dimension than sample-size), local regression does not help much, unless we are willing to make some structural assumptions about the model. In the following, we focus on some approaches to kernel methods.

25.1 Structured Kernels

One line of approach is to modify the kernel. A general approach is to use a positive semidefinite matrix $\boldsymbol{A}$ to weigh the different coordinates:

$$K_{\lambda,A}(x_0, x) = D\Big(\frac{(x - x_0)^\top \boldsymbol{A}(x - x_0)}{\lambda}\Big). \tag{110}$$

Entire coordinates or directions can be downgraded or omitted by imposing appropriate restrictions on $\boldsymbol{A}$.

25.2 Structured Regression Functions

If trying to fit a regression function $E(Y|X) = f(X_1, X_2, \cdots, X_p)$ in $\mathbb{R}^p$, in which every level of interaction is potentially present. It is natural to consider analysis-of-variance (ANOVA) decompositions of the form

$$f(X_1, X_2, \cdots, X_p) = \alpha + \sum_j g_j(X_j) + \sum_{k<\ell} g_{k\ell}(X_k, X_\ell) + \cdots \tag{111}$$

and then introduce structure by eliminating some of the higher order terms. Additive models assume only main effect terms: $f(X) = \alpha + \sum_{j=1}^p g_j(X_j)$;

second-order models will have terms with interactions of order at most two, and so on.

An important special case of these structured models are the class of *varying coefficient models*. Suppose that we divide the p predictors in X into a set $(X_1, X_2, \cdots, X_p)$ with $q < p$, and the remainder of the variables we collect in the vector $\boldsymbol{z}$. We then assume the conditionally linear model

$$f(X) = \alpha(\boldsymbol{z}) + \beta_1(\boldsymbol{z})X_1 + \cdots + \beta_q(\boldsymbol{z})X_q. \tag{112}$$

For a given $\boldsymbol{z}$, this is a linear model, but each of the coefficients can very with $\boldsymbol{z}$. It is natural to fit such a model with locally weighted least squares:

$$\min_{\alpha(z_0),\beta(z_0)} \sum_{i=1}^{N} K_\lambda(z_0, z_i)(y_i - \alpha(z_0) - x_{1i}\beta_1(z_0) - /cdots - x_{qi}\beta_q(z_0))^2. \tag{113}$$

26 Local Likelihood and Other Models

Any parametric model can be made local if the fitting method accommodates observation weights. Here are some examples:

- Associated with each observation y_i is a parameter $\theta_i = \theta(x_i) = x_i^\top \beta$ linear in the covariate(s) x_i, and inference for β is based on the log-likelihood $l(\beta) = \sum_{i=1}^N l(y_i, x_i^\top \beta)$. We can model $\theta(X)$ more flexibly by using the likelihood local to x_0 for inference of $\theta(x_0) = x_0^\top \beta(x_0)$:

$$l(\beta(x_0)) = \sum_{i=1}^{N} K_\lambda(x_0, x_i) l(y_i, x_i^\top \beta(x_0)). \tag{114}$$

- As above, except different variables are associated with θ from those used for defining the local likelihood:

$$l(\beta(z_0)) = \sum_{i=1}^{N} K_\lambda(z_0, z_i) l(y_i, \eta(x_i \theta(z_0))). \tag{115}$$

- Autoregressive time series models of order k have the form $y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \cdots + \beta_k y_{t-k} + \epsilon_t$.

As an illustration of the local likelihood, consider the local version of the multiclass linear logistic regression model. The data consist of features x_i and an associated categorical response $g_i \in \{1, 2, \cdots, J\}$. The linear model as the form

$$\Pr(G = j | X = x) = \frac{e^{\beta_{j0} + \beta_j^\top x}}{1 + \sum_{k=1}^{J-1} e^{\beta_{j0} + \beta_j^\top x}}. \tag{116}$$

The local log-likelihood for this J class model can be written as

$$\sum_{i=1}^{N} K_\lambda(x_0, x_i)\Bigg\{\beta_{gi0}(x_0) + \beta_{gi}(x_0)^\top (x_i - x_0) \tag{117}$$

$$-\log\left[1 + \sum_{k=1}^{J-1} \exp(\beta_{k0}(x_0) + \beta_k(x_0)^\top (x_i - x_0))\right]\Bigg\}. \tag{118}$$

27 Kernel Density Estimation and Classification

Kernel density estimation is an unsupervised learning procedure, and leads naturally to a simple family of procedures for nonparametric classification.

27.1 Kernel Density Estimation

Suppose we have a random sample $x_1, \cdots, x_n$ drawn from a probability density $f_X(x)$ and we wish to estimate f_X at a point x_0 (assuming that $X \in \mathbb{R}$). A natural local estimate has the form

$$\hat{f}_X(x_0) = \frac{\# x_i \in \mathcal{N}(x_0)}{N\lambda}, \tag{119}$$

where $\mathcal{N}(x_0)$ is a small metric neighbourhood around x_0 of width λ, and $\#$ denotes a primorial function, similar to a factorial but using only prime numbers. This estimate is bumpy, and the smooth *Parzen* estimate is preferred

$$\hat{f}_X(x_0) = \frac{1}{N\lambda} \sum_{i=1}^{N} K_\lambda(x_0, x_i), \tag{120}$$

because it counts observations close to x_0 with weights that decrease with distance from x_0. A popular choice for K_λ is the Gaussian kernel $K_\lambda(x_0, x) = \phi(|x - x_0|/\lambda)$. Letting ϕ_λ denote the Gaussian density with mean zero and standard-deviation λ, then the above equation becomes

$$\hat{f}_X(x) = \frac{1}{N} \sum_{i=1}^{N} \phi_\lambda(x - x_i) = (\hat{F} * \phi_\lambda)(x), \tag{121}$$

the convolution of the sample empirical distribution $\hat{F}$ with ϕ_λ. We have added independent Gaussian noise to each observation x_i to make it smoother. The natural generalisation of the Gaussian density estimate amounts to using a Gaussian product kernel, such that

$$\hat{f}_X(x) = \frac{1}{N(2\lambda^2\pi)^{p/2}} \sum_{i=1}^{N} e^{-\frac{1}{2}(||x - x_0||/\lambda)^2}. \tag{122}$$

27.2 Kernel Density Classification

It is possible to use nonparametric density estimates for classification using Baye's theorem. Suppose for a J class problem we fit nonparametric density estimates $\hat{f}_j(X)$ separately in each of the classes, and we also have estimates of the class priors $\hat{\pi}_j$, then

$$\hat{\Pr}(G = j|X = x_0) = \frac{\hat{\pi}_j \hat{f}_j(x_0)}{\sum_{k=1}^{J} \hat{\pi}_k \hat{f}_k(x_0)}. \tag{123}$$

27.3 The Naive Bayes Classifier

This technique is popular and is especially appropriate when the dimension p of the feature space is high, making density estimation unattractive. The naive Bayes model assumes that given a class $G = j$, the features X_k are independent:

$$f_j(X) = \prod_{k=1}^{p} f_{jk}(X_k). \tag{124}$$

28 Radial Basis Functions and Kernels

The art of flexible modeling using basis expansions (like the models presented in this chapter until now) consist on picking an appropriate family of basis functions, and then controlling the complexity of the representation by selection, regularisation, or both.

Kernel methods achieve flexibility by fitting simple models in a region local to the target point x_0. Localisation is achieved via a weighting kernel K_λ, and individual observations receive weights $K_\lambda(x_0, x_i)$.

Radial basis functions combine these ideas, by treating the kernel functions $K - \lambda(\xi, x)$ as basis functions. This leads to

$$f(x) = \sum_{j=1}^{M} K_{\lambda_j}(\xi_j, x)\beta_j = \sum_{j=1}^{M} D\Big(\frac{||x - \xi_j||}{\lambda_j}\Big), \tag{125}$$

where each basis element is indexed by a location or *prototype* parameter ξ_j and a scale parameter λ_j. Here, a popular choice for D is the standard Gaussian density function. Using the Gaussian kernel and least squares methods for regression, the parameters in the model can be determined using

$$\min_{\{\lambda_j, \xi_j, \beta_j\}_1^M} \sum_{j=1}^{M} \left(y_i - \beta_0 - \sum_{j=1}^{M} \beta_j \exp\Big\{ -\frac{(x_i - \xi_j)^\top (x_i - \xi_j)}{\lambda_j^2} \Big\} \right)^2. \tag{126}$$

29 Mixture Models for Density Estimation and Classification

The mixture model is useful for classification, and can be thought of as a kernel method. The Gaussian mixture model is a popular choice, and has the form

$$f(x) = \sum_{m=1}^{M} \alpha_m \phi(x; \mu_m, \Sigma_m) \tag{127}$$

with mixing proportions $\alpha_m, \Sigma_m \alpha_m = 1$, and each Gaussian density has a mean μ_m and covariance matrix (uncertainty) Σ_m. The parameters of the mixture model are usually fit by maximum likelihood, using the expected maximisation (EM) algorithm.

Model Assessment and Selection

The ability of a model to predict on independent test data is referred to as *generalisation* performance, and assessment of this performance is extremely important as it guides the choice of model/method for learning. In this chapter, we will describe and show key methods for performance assessment.

30 Bias, Variance, and Model Complexity

Imagine we have a target variable Y, a vector of inputs X, and a prediction mode $\hat{f}(X)$ that has been estimated from training set $\mathcal{T}$. The loss function for measuring errors between Y and $\hat{f}(X)$ is denoted as $L(Y, \hat{f}(X))$. Typically, one uses the absolute difference between the two or the squared difference. *Test error* or *generalisation error* is the prediction error over an independent test sample

$$\mathrm{Err}_{\mathcal{T}} = \mathrm{E}[L(Y, \hat{f}(X))|\mathcal{T}]. \tag{128}$$

The test error refers to the error for this specific training set. A related quantity is the expected prediction error (or expected test error)

$$\mathrm{Err} = \mathrm{E}[L(Y, \hat{f}(X))] = \mathrm{E}[\mathrm{Err}_{\mathcal{T}}]. \tag{129}$$

The training error is the average loss over the training sample

$$\bar{\mathrm{err}} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i)). \tag{130}$$

Unfortunately, training error is not a good reflection of test error.

The log-likelihood can be used a s a loss-function for general response densities, such as Poisson, gamma, exponential, log-normal and others. If $\mathrm{Pr}_{\theta(X)}(Y)$ is the density of Y, then

$$L(Y, \theta(X)) = -2 \cdot \log \mathrm{Pr}_{\theta(X)}(Y). \tag{131}$$

Here, the -2 makes the log-likelihood loss for the Gaussian distribution match squared-error loss.

It is important to note that there are two separate goals when it comes to comparing a model to data:

- **Model selection**: estimating the performance of different models to choose the best one.
- **Model assessment**: having chosen a model, estimating its prediction error on new data.

If we are in a data-rich situation, the best approach for both problems is to randomly divide the data set into three parts: a training set ($\sim 50\%$), a validation set ($\sim 25\%$), and a test set ($\sim 25\%$). 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 then for assessment of the generalisation error.

31 The Bias-Variance Decomposition

Assuming that $Y = f(X) + \epsilon$ where $\mathrm{E}(\epsilon) = 0$ and $\mathrm{Var}(\epsilon) = \sigma_\epsilon^2$, we can derive an expression for the expected prediction error of a regression fit $\hat{f}(X)$ at an input point $X = x_0$, using squared-error loss:

$$\begin{aligned}
\mathrm{Err}(x_0) &= E[(Y - \hat{f}(x_0))^2 | X = x_0] && (132)\\
&= \sigma_\epsilon^2 + [\mathrm{E}\hat{f}(x_0) - f(x_0)]^2 + [\mathrm{E}\hat{f}(x_0) - \hat{f}(x_0)]^2 && (133)\\
&= \sigma_\epsilon^2 + \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}^2(\hat{f}(x_0)) && (134)\\
&\text{Irreducible Error} + \mathrm{Bias}^2 + \text{Variance}. && (135)
\end{aligned}$$

The first term is the variance of the target around its true mean $f(x_0)$ and cannot be avoided no matter how well we estimate $f(x_0)$, unless $\sigma_\epsilon^2 = 0$. The second term is the squared bias, the amount by which the average of our estimate differs from the true mean. The last term is the variance, the expected squared deviation of $\hat{f}(x_0)$ around its mean. Typically, the more complex a model is the lower the (squared) bias and higher the variance.

For the k-nearest neighbour regression fit, these expressions have the simple form

$$\begin{aligned}
\mathrm{Err}(x_0) &= E[(Y - \hat{f}(x_0))^2 | X = x_0] && (136)\\
&= \sigma_\epsilon^2 + \Big[f(x_0) - \frac{1}{k}\sum_{\ell=1}^{k} f(x_\ell)\Big]^2 + \frac{\sigma_\epsilon^2}{k}. && (137)
\end{aligned}$$

For a linear model fit ($\hat{f}_p(x) = x^\top \beta$), we have

$$\begin{aligned}
\mathrm{Err}(x_0) &= E[(Y - \hat{f}_p(x_0))^2 | X = x_0] && (138)\\
&= \sigma_\epsilon^2 + [f(x_0) - \mathrm{E}\hat{f}_p(x_0)]^2 + ||\boldsymbol{h}(x_0)||^2 \sigma_\epsilon^2, && (139)
\end{aligned}$$

where $\boldsymbol{h}(x_0) = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1} x_0$.

32 Optimism of the Training Error Rate

Optimism in statistics is defined as the difference between the *in sample* error and the training error

$$\mathrm{op} = \mathrm{Err_{in}} - \overline{\mathrm{err}}, \qquad (140)$$

where

$$\mathrm{Err_{in}} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{E}_{Y^0}[L(Y_i^0, \hat{f}(x_i)) | \mathcal{T}]. \qquad (141)$$

The optimism is typically positive since $\overline{\mathrm{err}}$ is usually biased downward as an estimate of prediction error. The average optimism is the expectation of the optimism over training sets $\omega = \mathrm{E}_y(\mathrm{op})$.

33 Estimates of In-Sample Prediction Error

The general form of in-sample estimates is

$$\widehat{\mathrm{Err_{in}}} = \bar{\mathrm{err}} + \hat{\omega}. \tag{142}$$

The *Akaike information criterion* (AIC) is a more general application of the in-sample error when a log-likelihood loss function is used. It relies on a relationship that holds asymptotically as $N \to \infty$:

$$-2 \cdot \mathrm{E}[\log \mathrm{Pr}_{\hat{\theta}}(Y)] \approx -\frac{2}{N} \cdot \mathrm{E}[\mathrm{loglik}] + 2 \cdot \frac{d}{N}, \tag{143}$$

where d is the number of inputs or basis functions used. For example, for the logistic regression model, using a binomial log-likelihood yields

$$\mathrm{AIC} = -\frac{2}{N} \cdot \mathrm{loglik} + 2 \cdot \frac{d}{N}. \tag{144}$$

For a Gaussian model, the AIC statistic is equivalent to what is called the C_p statistic, such that

$$C_p = \bar{\mathrm{err}} + 2 \cdot \frac{d}{N} \hat{\sigma}_\epsilon^2. \tag{145}$$

34 The Effective Number of Parameters

The concept of "number of parameters" can be generalised. Suppose we stack the outcomes $y_1, y_2, \cdots, y_N$ into a vector $\boldsymbol{y}$ with predictions $\hat{\boldsymbol{y}}$, then the linear fitting method is one for which

$$\hat{\boldsymbol{y}} = \boldsymbol{S}\boldsymbol{y}, \tag{146}$$

where $\boldsymbol{S}$ is an $N \times N$ matrix depending on the input vectors x_i but not on y_i. Then the effective number of parameters is defined as

$$\mathrm{df}(\boldsymbol{S}) = \mathrm{trace}(\boldsymbol{S}) \tag{147}$$

the sum of the diagonal elements of $\boldsymbol{S}$ (also known as the *effective degrees of freedom*).

For more complex models like neural-networks in which we minimise an error function $R(w)$ with weight decay penalty (regularisation) $\alpha \sum_m w_m^2$, the effective number of parameters has the form

$$\mathrm{df}(\alpha) = \sum_{m=1}^{M} \frac{\theta_m}{\theta_m + \alpha}, \tag{148}$$

where θ_m are the eigenvalues of the Hessian matrix $\partial^2 R(w) / \partial w \partial w^\top$.

35 The Bayesian Approach and BIC

The Bayesian information criterion (BIC) is applicable in settings where the fitting is carried out by maximisation of a log-likelihood. The generic form of BIC is

$$\text{BIC} = -2 \cdot \text{loglik} + (\log N) \cdot d. \tag{149}$$

For a Gaussian model, the BIC becomes

$$\text{BIC} = \frac{N}{\sigma_\epsilon^2}[\bar{\text{err}} + (\log N) \cdot \frac{d}{N}\sigma_\epsilon^2]. \tag{150}$$

BIC penalises more complex models, giving preference to simpler models. Choosing the model with minimum BIC is equivalent to choosing the model with largest (approximate) posterior probability.

36 Minimum Description Length

The minimum description length (MDL) gives a selection criterion that is formally identical to the BIC approach, but is motivated from a coding point of view.

The method thinks of a datum z as a message that we want to encode and send to someone else (the *receiver*). Thus, the model is thought of as a way of encoding such datum, and will choose the most parsimonious model (the shortest code) for the transmission.

Suppose we have a model M with parameters θ and data $\boldsymbol{Z} = (\boldsymbol{X}, \boldsymbol{y})$. Let the conditional probability of the outputs under the model be $\Pr(\boldsymbol{y}|\theta, M, \boldsymbol{X})$, then the message length required to transmit the outputs is

$$\text{length} = -\log \Pr(\boldsymbol{y}|\theta, M, \boldsymbol{X}) - \log \Pr(\theta|M). \tag{151}$$

The MDL method says that we should choose a model that minimises the above relation.

37 Vapnik-Chervonenkis Dimension

A difficulty in using estimates of in-sample error is the need to specify the number of parameters (or complexity) d used in the fit. The Vapnik-Chervonenkis (VC) theory provides such a general measure of complexity, and gives associated bounds on the optimism.

Suppose we have a class of functions $\{f(x, \alpha)\}$ indexed by a parameter vector α, with $x \in \mathbb{R}^p$. The VC dimension is a way of measuring the complexity of a class of functions by assessing how wiggly its members can be.

the VC dimension of the class $\{f(x, \alpha\})$ *is defined to be the largest number of points (in some configuration) that can be shattered by members of* $\{f(x, \alpha\})$.

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.

Vapnik's *structural risk minimisation* (SRM) approach fits a nested sequence of models of increasing VC dimensions $h_1 < h_2 < \cdots$, and then chooses the model with the smallest value of the upper bound.

38 Cross-Validation

Cross-validation is probably the simplest and most utilised method for estimating prediction error. This method directly estimates the expected extra-sample error $\mathrm{Er} = \mathrm{E}[L(Y, \hat{f}(X))]$, the average generalisation error when the method $\hat{f}(X)$ is applied to an independent test sample from the joint distribution of X and Y.

38.1 K-fold Cross-Validation

Ideally, if enough data is available, we would set aside a validation set and use it to assess the performance of our prediction model. However, since in practice (although less so now) data is often scarce, this is usually not possible. To by-pass this problem, K-fold cross-validation uses part of the available data to fit the model, and different part to test it, splitting the data into K roughly equal-sized parts.

Let $k : \{1, \cdots, N\} \to \{1, \cdots, K\}$ be an indexing function that indicates the partition to which observation i is allocated too, and $\hat{f}^{-k}(x)$ be the fitted function, computed when the kth part of the data has been removed. Then the cross-validation estimate of the prediction error is

$$\mathrm{CV}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}^{-k(i)}(x_i)). \tag{152}$$

The typical choices of K are usually 5 or 10. The case when $K = N$ is known as *leave-one-out* cross-validation. In this case, the model is fitted with all data except the ith.

Generalised crross-validation (GCV) provides a convenient approximation to leave-one-out cross-validation, for linear fitting under squared-error loss. The GCV approximation is

$$\mathrm{GCV}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N}\Big[\frac{y_i - \hat{f}(x_i)}{1 - \mathrm{trace}(\boldsymbol{S})/N}\Big]^2 \tag{153}$$

where $\mathrm{trace}(\boldsymbol{S})$ is the effective number of parameters.

39 Bootstrap Methods

The bootstrap is a general tool for assessing statistical accuracy. The bootstrap seeks to estimate the conditional error $\text{Err}_{\mathcal{T}}$, but typically estimates well only the expected prediction error Err.

Suppose we have a model fit to a set of training data. We denote the training set by $\boldsymbol{Z} = (z_1, z_2, \cdots, z_N)$ where $z_i = (x_i, y_i)$. The basic idea is to randomly draw datasets with replacement from the training data, each sample the same size as the original training set. Then we refit the model to each of the bootstrap datasets, and examine the behaviour of the fits over the replications.

How can we apply bootstrap to estimate prediction error? One approach would be to fit the model on a set B of bootstrap samples, and then keep track of how well it predicts the original training set. The estimate would then be

$$\widehat{\text{Err}_{\text{boot}}} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)), \tag{154}$$

where $\hat{f}^{*b}(x_i)$ is the predicted value at x_i.

However, since much of the same data is used in each bootstrap, this can be an under estimated indicator. Mimicking cross-validation, a better bootstrap estimate can be obtained for each observation by only keeping track of predictions from bootstrap samples not containing that observation. The leave-one-out bootstrap estimate of prediction error is defined by

$$\widehat{\text{Err}}^{(1)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L(y_i, \hat{f}^{*b}(x_i)), \tag{155}$$

where C^{-i} is the set of indices in the bootstrap samples b that **do not** contain observation i.

Model Inference and Averaging

40 The Bootstrap and Maximum Likelihood Methods

The bootstrap method provides a direct computational way of assessing uncertainty by sampling from the training data. Here we illustrate the bootstrap in a simple one-dimensional problem, and show its connection to maximum likelihood.

Denote the training data by $\boldsymbol{Z} = \{z_1, z_2, \cdots, z_n\}$, with $z_i = (x_i, y_i)$ from $i = 1, 2, 3, \cdots, N$. x_i is a one-dimensional input, and y_i is the outcome (response), either continuous or categorical. Suppose we decide to fit a cubic spline to the data, with three knots placed at the quartiles of the X values. This is a linear expansion of B-spline basis functions (seven-dimensional):

$$\mu(x) = \sum_{j=1}^{7} \beta_j h_j(x). \tag{156}$$

We can think of $\mu(x)$ as representing the conditional mean $\mathrm{E}(Y|X = x)$. Now let $\boldsymbol{H}$ be the $N \times 7$ matrix with the ijth element $h_j(x_i)$. The usual estimate of β is given by

$$\hat{\beta} = (\boldsymbol{H}^\top \boldsymbol{H})^{-1} \boldsymbol{H}^\top \boldsymbol{y}. \tag{157}$$

The estimated covariance matrix of $\hat{\beta}$ is

$$\hat{\mathrm{Var}}(\hat{\beta}) = (\boldsymbol{H}^\top \boldsymbol{H})^{-1} \hat{\sigma}^2, \tag{158}$$

where $\hat{\sigma}^2 = \sum_{i=1}^{N} (y_i - \hat{\mu}(x))^2 / N$.

The standard error of a prediction $\hat{\mu}(x) = h(x)^\top \hat{\beta}$ is

$$\hat{\mathrm{se}}[\hat{\mu}(x)] = [h(x)^\top (\boldsymbol{H}^\top \boldsymbol{H})^{-1} h(x)]^{1/2} \hat{\sigma}. \tag{159}$$

To apply bootstrap, one would draw B datasets each of size $N = 50$ with replacement from our training data. To each bootstrap dataset $\boldsymbol{Z}^*$ we fit a cubic spline $\hat{\mu}^*(x)$. Using $B = 200$ bootstrap samples, we can form a 95% pointwise confidence band from the percentiles at each x: we find the $2.5\% \times 200 =$ fifth largest and smallest values at each x.

There is actually a close connection between the least-squares estimates and the bootstrap and maximum likelihood. Suppose that we assume that the model errors are Gaussian,

$$Y = \mu(X) + \epsilon; \epsilon \sim N(0, \sigma^2), \tag{160}$$

$$\mu(x) = \sum_{j=1}^{7} \beta_j h_j(x). \tag{161}$$

The bootstrap method shown above (sampling with replacement) is called the nonparametric bootstrap, meaning that the method is “model free”; this is because it uses the data to draw more samples rather than the model. Consider a variation of the bootstrap, called the parametric bootstrap, in which we simulate responses by adding Gaussian noise to the predicted values:

$$y_i^* = \hat{\mu}(x_i) + \epsilon_i^*; \epsilon_i^* \sim N(0, \hat{\sigma}^2); i = 1, 2, \cdots, N. \tag{162}$$

If we repeat this process $B = 200$ times, the resulting bootstrap datasets have the form $(x_i, y_i^*), \cdots, (x_N, y_N^*)$ and we recompute the B-spline smooth on each. The confidence bands from this method will exactly equal the least squares bands as the number of bootstraps goes to infinity. A function estimated from a bootstrap sample $\boldsymbol{y^*}$ is given by $\hat{\mu}^*(x) = h(x)^\top(\boldsymbol{H}^\top\boldsymbol{H})^{-1}\boldsymbol{H}^\top\boldsymbol{y^*}$, and has a distribution

$$\hat{\mu}^*(x) = N(\hat{\mu}(x), h(x)^\top(\boldsymbol{H}^\top\boldsymbol{H})^{-1}h(x)\hat{\sigma}^2). \tag{163}$$

Notice that the mean of the distribution is the least squares estimate, and the standard deviation is the same as the approximate formula.

40.1 Maximum Likelihood Inference

Turns out that the parametric bootstrap agrees not with least squares, but with maximum likelihood.

Say we specify a probability density or probability mass function for our observations $z_i \sim g_\theta(z)$. Here θ represents one or more unknown parameters that govern the distribution of Z (i.e., a parametric model for Z).

Maximum likelihood is based on the *likelihood function*, given by

$$L(\theta; \boldsymbol{Z}) = \prod_{i=1}^{N} g_\theta(z_i), \tag{164}$$

the probability of the observed data under the model g_θ.

Denote the logarithm of $L(\theta; \boldsymbol{Z})$ by

$$\ell(\theta; \boldsymbol{Z}) = \sum_{i=1}^{N} \ell(\theta; z_i) = \sum_{i=1}^{N} \log g_\theta(z_i). \tag{165}$$

This expression is called the *log-likelihood*. The method of maximum likelihood chooses the value $\theta = \hat{\theta}$ to maximise $\ell(\theta; \boldsymbol{Z})$. In order to use the likelihood function to assess the precision of $\hat{\theta}$, we need to define a *score function*

$$\dot{\ell}(\theta; \boldsymbol{Z}) = \sum_{i=1}^{N} \dot{\ell}(\theta; z_i), \tag{166}$$

where $\dot{\ell}(\theta; z_i) = \partial\ell(\theta; z_i)/\partial\theta$.

The *information matrix* is

$$\boldsymbol{I}(\theta) = -\sum_{i=1}^{N} \frac{\partial^2 \ell(\theta; z_i)}{\partial\theta\partial\theta^\top}. \tag{167}$$

When $\boldsymbol{I}(\theta)$ is evaluated at $\theta = \hat{\theta}$, it is often called the *observed information.* The *Fisher information* (or expected information) is

$$\boldsymbol{i}(\theta) = \mathrm{E}_\theta[\boldsymbol{I}(\theta)]. \tag{168}$$

The corresponding estimates for the standard errors of $\hat{\theta}_j$ are obtained from

$$\sqrt{\boldsymbol{i}(\hat{\theta})^{-1}_{jj}}. \tag{169}$$

41 Bayesian Methods

In the Bayesian approach to inference, we specify a sampling model $\Pr(\boldsymbol{Z}|\theta)$ for our data given the parameters, and a prior distribution for the parameters $\Pr(\theta)$ that reflects knowledge about θ before we even see the data. We then compute the posterior distribution

$$\Pr(\theta|\boldsymbol{Z} = \frac{\Pr(\boldsymbol{Z}|\theta)\Pr(\theta)}{\int \Pr(\boldsymbol{Z}|\theta)\Pr(\theta)d\theta}, \tag{170}$$

which represents our updated knowledge about θ after we see the data.

The posterior distribution also provides the basis for predicting the values of a future observation z^{new}, via the *predictive distribution*:

$$\Pr(z^{\text{new}}|\boldsymbol{Z}) = \int \Pr(z^{\text{new}}|\theta)\Pr(\theta|\boldsymbol{Z})d\theta. \tag{171}$$

This relation accounts for the uncertainty in estimating θ to predict future data, which is what sets it apart from the maximum likelihood method.

The choice of priors is a whole topic, but the less informative the prior (i.e., *noninformative prior*), the less information is used in the method to determine the posterior and viceversa.

42 Relationship Between the Bootstrap and Bayesian Inference

Consider a simple example in which we observe a single observation z from a normal distribution $z \sim N(\theta, 1)$. To carry out Bayesian analysis for θ, we need to specify a prior. The most convenient and common choice would be $\theta \sim N(0, \tau)$, yielding a posterior distribution

$$\theta|z \sim N\Big(\frac{z}{1+1/\tau}, \frac{1}{1+1/\tau}\Big). \tag{172}$$

The larger we take τ the more concentrated the posterior becomes around the maximum likelihood estimate $\hat{\theta} = z$. In the limit of $\tau \to \infty$ we obtain a noninformative prior and the posterior is $\theta | z \sim N(z, 1)$.

This is the same as a parametric bootstrap distribution in which we generate bootstrap values z^* from the maximum likelihood estimate of the sampling density $N(z, 1)$. There are three ingredients that make this so:

- The choice of noninformative prior for θ.
- The dependence of the log-likelihood on the dat only through the maximum likelihood estimate.
- The symmetry of the log-likelihood in θ and $\hat{\theta}$, that is $\ell(\theta; \hat{\theta}) = \ell(\hat{\theta}; \theta) +$ constant.

43 The Expectation Maximisation Algorithm

The Expectation Maximisation (EM) algorithm is a popular tool for simplifying difficult maximum likelihood problems.

The general formulation of the EM algorithm is set as follows:

1. Start with initial guesses for the parameters in your model $\hat{\theta}^0$.
2. *Expectation step*: at the jth step, compute
$$Q(\theta^{'}, \hat{\theta}^{(j)}) = \mathrm{E}(\ell_0(\theta^{'}; \boldsymbol{T}) | \boldsymbol{Z}, \hat{\theta}^{(j)}) \tag{173}$$
as a function of the dummy argument $\theta^{'}$.
3. *Maximisation step:* determine the new estimate $\hat{\theta}^{(j+1)}$ as the maximiser of $Q(\theta^{'}, \hat{\theta}^{(j)})$ over $\theta^{'}$.
4. Iterate steps 2 and 3 until convergence.

Here, the observed data is $\boldsymbol{Z}$, having log-likelihood $\ell(\theta; \boldsymbol{Z})$ depending on parameters θ. The latent or missing data is $\boldsymbol{Z}^m$, so that the complete data is $\boldsymbol{T} = (\boldsymbol{Z}, \boldsymbol{Z}^M)$ with log-likelihood $\ell_0(\theta; \boldsymbol{T})$, ℓ_0 based on the complete density.

The EM algorithm works in general because

$$\Pr(\boldsymbol{Z}^m | \boldsymbol{Z}, \theta^{'}) = \frac{\Pr(\boldsymbol{Z}^m, \boldsymbol{Z} | \theta^{'})}{\Pr(\boldsymbol{Z} | \theta')}, \tag{174}$$

and thus we can write

$$\Pr(\boldsymbol{Z} | \theta^{'}) = \frac{\Pr(\boldsymbol{T} | \theta^{'})}{\Pr(\boldsymbol{Z}^m | \boldsymbol{Z}, \theta')}. \tag{175}$$

44 MCMC for Sampling from the Posterior

Having defined a Bayesian model, one would like to draw samples from the resulting posterior distribution in order to make inference about the parameters in the model. When the models are not simple, this is a difficult computational problem. An approach for posterior sampling with complex models is the **Markov chain Monte Carlo** (MCMC) procedure.

Consider an abstract problem where we have random variables $U_1, \cdots, U_K$ and we wish to draw a sample from their joint distribution. Suppose this is tricky to do, but it is easy to simulate from the conditional distributions $\Pr(U_j|U_1, \cdots, U_K)$. The Gibbs sampling procedure alternatively simulates from each of these distributions and when the process stabilises, provides a sample from the desired joint distribution. Here, Gibbs sampling forms a Markov chain whose stationary distribution is the true joint distribution, and hence the name MCMC.

There are many different examples of MCMC procedures, Gibbs is just one of them. Another famous example that does not require the structure Gibbs sampling needs is the *Metropolis-Hastings* algorithm. For the interested reader and more examples, see the Wikipedia page on Markov chain Monte Carlo `https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo`.

45 Bagging

Here we show how to use bootstrap to improve the estimate or prediction itself. Bagging exploits the connection between bootstrap and Bayes using the bootstrap mean as an approximate of the posterior average.

Consider a regression problem. Suppose we fit a model to our training data $\boldsymbol{Z} = \{(x_1, y_1), \cdots, (x_N, y_N)\}$, obtaining the prediction $\hat{f}(x)$ at input x. Bootstrap aggregation or *bagging* averages this prediction over a collection of bootstrap samples, thereby, reducing its variance. For each bootstrap sample $\boldsymbol{Z^{*b}}, b = 1, 2, \cdots, B$, we fit our model, giving prediction $\hat{f}^{*b}(x)$. The bagging estimate is defined by

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{N} \hat{f}^{*b}(x). \tag{176}$$

46 Model Averaging and Stacking

Say we have a set of candidate models $\mathcal{M}_m, m = 1, \cdots, M$ for our training set $\boldsymbol{Z}$. These models may be of the same type but with different parameter values (e.g., linear models with different slopes), or different models for the same task (e.g., neural-networks and regression trees).

Suppose ζ is some quantity of interest, for example, a predictin $f(x)$ at some

x value. The posterior distribution of ζ is

$$\Pr(\zeta|\boldsymbol{Z}) = \sum_{m=1}^{M} \Pr(\zeta|\mathcal{M}_m, \boldsymbol{Z})\Pr(\mathcal{M}_m|\boldsymbol{Z}), \tag{177}$$

with posterior mean

$$\mathrm{E}(\zeta|\boldsymbol{Z}) = \sum_{m=1}^{M} \mathrm{E}(\zeta|\mathcal{M}_m, \boldsymbol{Z})\Pr(\mathcal{M}_m|\boldsymbol{Z}). \tag{178}$$

The Bayesian prediction is a weighted average of the individual predictions, with weights proportional to the posterior probability of each model.

This formulation leads to a number of different model-averaging strategies. *Committee methods* take a simple unweighted average of the predictions from each model, essentially giving equal probability to each model. More ambitiously, we showed in earlier chapters that the BIC criterion can be used to estimate posterior model probabilities.

Stacked generalisation (or *stacking*) is a way of doing model averaging from a frequentist viewpoint. Let $\hat{f}_m^{-i}(x)$ be the prediction at x, using model m, applied to a dataset with the ith training observation removed. The stacking estimate of the weights is obtained from the least squares linear regression of y_i on $\hat{f}_m^{-i}(x_i)$. In detail the stacking weights are given by

$$\hat{w}^{\mathrm{st}} = \mathrm{argmin_m} \sum_{i=1}^{N} \Big[y_i - \sum_{m=1}^{M} w_m \hat{f}_m^{-i}(x_i)\Big]^2. \tag{179}$$

Stacking avoids giving unfarily high weight to models with higher complexity.

47 Stochastic Search: Bumping

Bumping is an approach that does not try to average or combine models, but rather finding a better single model. *Bumping* uses bootstrap sampling to move randomly through model space. For problems where fitting methods find many local minima, bumping can help the method avoid getting stuck in poor solutions.

As in bagging, we draw bootstrap samples and fit a model to each. But, rather than averaging predictions, we choose the model estimated from a bootstrap sample that best fits the training data. In detail, we draw bootstrap samples $\boldsymbol{Z}^{*B}$ and fit a model to each, yielding B predictions at point x. We then choose the model that produces the smallest prediction error, averaged over the original *training set*. For example, using squared error, we choose the model that

$$\hat{b} = \arg\min_{b} \sum_{i=1}^{N} [y_i - \hat{f}^{*b}(x_i)]^2. \tag{180}$$

By perturbing the data, bumping tries to move the fitting procedure around to good areas of model space.

Additive Models, Trees, and Related Methods

In this chapter we discuss some specific methods for supervised learning. We describe five related techniques: generalised additive models, trees, multivariate adaptive regression splines, the patient rule induction model, and hierarchical mixture of experts.

48 Generalised Additive Models

Although simple, the traditional linear model often fails as effects are often not linear. Earlier chapters described techniques that used predefined basis functions to achieve non-linearities. Here we describe more automatic flexible statistical methods.

In regression setting, a generalised additive model has the form

$$\mathrm{E}(Y|X_1, X_2, \cdots, X_p) = \alpha + f_1(X_1) + f_2(X-2) + \cdots + f_p(X_p). \tag{181}$$

X_p represent predictors and Y is the outcome, the f_j's 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.

Take for example the logistic regression model for two-class classification of binary data. We relate the mean of the binary response $\mu(X) = \Pr(Y = 1|X)$ to the predictors via a linear regression model and a *logit* link function:

$$\log\left(\frac{\mu(X)}{1-\mu(X)}\right) = \alpha + \beta_1 X_1 + \cdots + \beta_p X_p. \tag{182}$$

The additive logistic regression model replaces each linear term by a more general functional form

$$\log\left(\frac{\mu(X)}{1-\mu(X)}\right) = \alpha + f_1(X_1) + \cdots + f_p(X_p), \tag{183}$$

where each f_j is an unspecified smooth function. This is an example of an additive model. In general, the conditional mean $\mu(X)$ of a response Y is related to an additive function of the predictors via a *link* function g:

$$g[\mu(X)] = \alpha + f_1(X_1) + \cdots + f_p(X_p). \tag{184}$$

48.1 Fitting Additive Models

The additive model has the form

$$Y = \alpha + \sum_{j=1}^{p} f_j(X_j) + \epsilon \tag{185}$$

where the error term ϵ has mean zero. Given observations x_i, y_i, a criterion like the penalised sum of squares can be specified for this problem:

$$\text{PRSS}(\alpha, f_1, f_2, \cdots, f_p) = \sum_{i=1}^{N} \left(y_i - \alpha - \sum_{j=1}^{p} f_j(x_{ij}) \right)^2 + \sum_{j=1}^{p} \lambda_j \int f^{''}(t_j)^2 dt_j, \tag{186}$$

where $\lambda_j \geq 0$ are tuning parameters (regularisation). It can be shown that the minimiser of this model 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 $x_{ij}, i = 1, \cdots, N$. However, there are many solutions to this model (α is not identifiable). The standard convention is to assume that $\sum_1^N f_j(x_{ij}) = 0$ — the functions average zero over the data (leading to $\hat{\alpha} = \text{ave}(y_i)$). Furthermore, an iterative procedure exists for finding the solution: setting $\hat{\alpha} = \text{ave}(y_i)$ and never letting it change, we apply a cubic smoothing spline to the targets $\{y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})\}_1^N$, as a funciton of x_{ij} to obtain a new estimate of $\hat{f}_j$. This is also known as *backfitting*.

49 Tree-Based Methods

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one.

Let's consider a regression problem with continuous response Y and inputs X_1, X_2, each taking values in the unit interval. In each partition we can model Y with a different constant. However, although each partitioning line has a simple description like $X_1 = c$, some of the resulting regions are complicated to describe. To simplify matters, we restrict attention to recursive binary partitions (i.e., make sure the rectangles leave straight lines across the full X_1, X_2 space. In other words, we first split the full space into two regions, and model the response by the mean of Y in each region. We then choose the variable and split-point to achieve the best fit. Then one of both of these regions are split into two more regions, and the process is repeated until some stopping rule is applied. The corresponding regression model predicts Y with a constant c_m in region R_m, that is,

$$\hat{f}(X) = \sum_{m=1}^{M} c_m I\{(X_1, X_2) \in R_m\}. \tag{187}$$

49.1 Regression Trees

How does one then grow a regression tree? Say our data consists of p inputs and a response, for each of N observations. The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have. Suppose that we have a partition into M regions

$R_1, R_2, \cdots, R_M$, and we model the response as a constant c_m in each region:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m). \tag{188}$$

If we adopt as our criterion minimisation of the sum of squares, it is easy to see that the best $\hat{c}_m$ is the average of y_i in region R_m:

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m). \tag{189}$$

Finding the best binary partition in terms of minimum sum of squares is generally computational infeasible. Thus, we proceed with a greedy algorithm. Starting with all the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j, s) = \{X | X_j \leq s\} \text{ and } R_2(j, s) = \{X | X_j > s\}. \tag{190}$$

Then we seek the splitting variable j and split point s that solve

$$\min_{j,s}[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2]. \tag{191}$$

For any choice of j and s, the inner minimisation is solved by

$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j, s)) \text{ and } \hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j, s)). \tag{192}$$

How large should we grow the tree? a large tree might overfit the data, while a small tree might not capture the important structure.Tree size is a tuning parameter, and the optimal tree size depends on the data. The preferred strategy is to grow a large tree, stopping the splitting process only when some minimum node size (say 5) is reached. Then this tree is pruned using *cost-complexity pruning*, described as follows.

We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning T_0. We index terminal nodes by m. Let $|T|$ denote the number of terminal nodes in T. We define the cost complexity criterion as

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T| \tag{193}$$

where

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2, \tag{194}$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i \tag{195}$$

$$N_m = \#\{x_i \in R_m\}. \tag{196}$$

Large values of α result in smaller trees and viceversa.

49.2 Classification Trees

If the target is classification, the only changes needed in the tree algorithm pertain to the criteria for splitting nodes and pruning the tree. For regression we used the squared-error node impurity measure $Q_m(T)$. In a node m, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k), \tag{197}$$

the proportion of class k observations in node m. We can classify the observations in node m to class $k(m) = \arg\max_k \hat{p}_{mk}$. Different measures of node impurity then include: missclassification error, Gini index, Cross-entropy or deviance.

50 PRIM: Bump Hunting

The Patient Rule Induction Method (PRIM), similarly to trees, also finds boxes in the feature space, but seeks boxes in which the response average is high. Hence, it looks for maxima in the target function (also known as *bump hunting*).

The main box construction method in PRIM works from the top down, starting with a box containing all the data. The box is compressed along one face by a small amount, and the observations then falling outside the box are peeled off. The face chosen for compression is the one resulting in the largest box mean, after the compression is performed. Then the process is repeated, stopping when the current box contains some minimum number of data points.

Denoting each box as B_k, each box is defined by a set of rules involving a subset of predictors like $(a_1 \leq X_1 \leq b_1)$ and $(b_1 \leq X_3 \leq b_2)$.

51 MARS: Multivariate Adaptive Regression Splines

MARS is an adaptive procedure for regression and is well suited for high-dimensional problems. MARS uses expansions in piecewise linear basis functions of the form $(x - t)_+$ and $(t - x)_+$. The $+$ means positive part, so

$$(x - t)_+ = \begin{cases} x - t \text{ if } x > t, 0 \text{ otherwise} \end{cases} \tag{198}$$

$$(t - x)_+ = \begin{cases} t - x \text{ if } x < t, 0 \text{ otherwise.} \end{cases} \tag{199}$$

These are *reflected pair* functions. The idea is to form reflected pairs for each input X_j with knots t at each observed value of x_{ij} of that input. Therefore, the collection of basis functions is

$$\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}. \tag{200}$$

52 Hierarchical Mixtures of Experts

The hierarchical mixtures of experts (HME) procedure can be viewed as a variant of tree-based methods. The main difference is that the tree splits are not hard decisions but rather soft probabilistic ones. At each node an observation goes left or right with probabilities depending on its input values.

The terminal nodes are called *experts*, and the non-terminal nodes are called *gating networks*. The idea is that each expert provides an opinion (prediction) about the response, and these are combined together by the gating networks. The model is formally a mixture model, that can be extended to multiple levels (i.e., hierarchical mixture of experts).

An HME is defined as follows. The top gating network has the output

$$g_j(x,\gamma_j) = \frac{e^{\gamma_j^\top x}}{\sum_{k=1}^K e^{\gamma_j^\top x}}, j = 1, 2, , \cdots, K, \tag{201}$$

where each γ_j is a vector of unknown parameters. Each $g_j(x,\gamma_j)$ is the probability of assigning an observation with feature vector x to the jth branch. At the second level, the gating networks have a similar form:

$$g_j(x,\gamma_{j\ell}) = \frac{e^{\gamma_{j\ell}^\top x}}{\sum_{k=1}^K e^{\gamma_{j\ell}^\top x}}, j = 1, 2, , \cdots, K. \tag{202}$$

This is the probability of assignment to the ℓth branch, given the assignment to the jth branch at the level above.

At each expert (terminal node), we have a model for the response variable of the form

$$Y \sim \Pr(y|x,\theta_{j\ell}), \tag{203}$$

which differs according to the problem. In regression, the Gaussian linear regression model is used, with $\theta_{j\ell} = (\beta_{j\ell}, \sigma_{j\ell}^2)$:

$$Y = \beta_{j\ell}^\top x + \epsilon \text{ and } \epsilon \sim N(0, \sigma_{j\ell}^2). \tag{204}$$

In classification, the linear logistic regression model is used:

$$\Pr(Y = 1|x, \theta_{j\ell}) = \frac{1}{1 + e^{-\theta_{j\ell}^\top x}}. \tag{205}$$

Denoting the collection of all parameters by $\Psi = \{\gamma_j, \gamma_{j\ell}, \theta_{j\ell}\}$, the total probability that $Y = y$ is

$$\Pr(y|x,\Psi) = \sum_{j=1}^K g_j(x,\gamma_j) \sum_{\ell=1}^K g_{\ell|j}(x,\gamma_{j\ell}) \Pr(y|x,\theta_{j\ell}). \tag{206}$$

This is a mixture model, with the mixture probabilities determined by the gating network models. To estimate the parameters, we maximise the log-likelihood of the data over the parameters in Ψ, with the most convenient method for doing so being expectation maximisation.

Boosting and Additive Trees

53 Boosting Methods

Originally designed for classification problems, boosting is an incredibly useful idea for regression. The motivation for boosting was a procedure that combines the outputs of many weak classifiers to produce a powerful *committee.*

The most popular boosting algorithm is *AdaBoost.M1* by Freund and Schapire (1997). 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 error rate on the training sample is

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i)), \tag{207}$$

and the expected error rate on future predictions is $\text{E}_{XY} I(Y \neq G(X))$. 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 $G_m(x), m = 1, 2, \cdots, M$. The predictions from all of them combined through a weighted majority vote then produce the final prediction:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right), \tag{208}$$

where $\alpha_1, \alpha_2, \cdots, \alpha_M$ are weights computed by the boosting algorithm. The model is then ran a number of iterations, where the weights are altered to favour those classifiers that make better prediction.

54 Boosting Fits an Additive Model

Boosting can be thought of a way of fitting an additive expansion in a set of elementary basis functions (i.e., the individual classifiers in the example above). More generally, basis function expansions take the form

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m), \tag{209}$$

where $\beta_m, m = 1, 2, \cdots, M$ are the expansion coefficients, and $b(x; \gamma)$ are usually simple functions of the multivariate argument x, characterised by a set of parameters γ.

Typically, these models are fit by minimizing a loss function averaged over some training data, such as the squared-error or a likelihood based loss function.

55 Forward Stagewise Additive Modelling

Forward stagewise modelling approximates the solution to the loss function by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those that have already been added. At each iteration m, one solves for the optimal basis function $b(x; \gamma_m)$ and corresponding coefficients β_m to add to the current expansion $f_{m-1}(x)$. This yields $f_m(x)$ and the process is then repeated.

The AdaBoost.M1 method is equivalent to forward stagewise additive modelling when using the loss function

$$L(y, f(x)) = \exp(-yf(x)). \tag{210}$$

56 Why Exponential Loss?

The principal attraction of exponential loss in the context of additive modelling is computational; it leads to the simple modular reweighting AdaBoost algorithm. However, it is of interest to inquire about its statistical properties. What does it estimate and how well is it being estimated? the first question is answered by seeking its population minimiser.

Friedman et al 2000 stells us that

$$\Pr(Y = 1|x) = \frac{1}{1 + e^{-2f^*(x)}}. \tag{211}$$

Thus, the additive expansion produced by the AdaBoost is estimating one-half the log-odds of $P(Y = 1|x)$.

Another loss criterion with the same population minimiser is the binomial negative log-likelihood or *deviance* (also known as cross-entropy), interpreting f as the logit transform. The binomial loss function is

$$l(Y, p(x)) = Y' \log p(x) + (1 - Y') \log(1 - p(x)), \tag{212}$$

or equivalently the deviance is

$$-l(Y, f(x)) = \log(1 + e^{-2Yf(x)}). \tag{213}$$

57 Loss Functions and Robustness

57.1 Robust Loss Functions for Classification

The goal of the classification algorithm is to produce positive margins as frequently as possible. Thus, any loss criterion for classification should penalise negative margins more heavily than positive ones since positive margin observations are already correctly classified.

The binomial and exponential loss functions have been shown to work decently well for classification tasks. The squared-error loss less so. The logistic model generalises naturally to problems with K classes

$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}. \tag{214}$$

57.2 Robust Loss Function for Regression

Analogous to the relationship between exponential loss and binomial log-likelihood in classification tasks, in regression it is between the squared-error loss $L(y, f(x)) = (y - f(x))^2$ and absolute loss $L(y, f(x)) = |y - f(x)|$.

58 Boosting Trees

A tree can be formally expressed as

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j). \tag{215}$$

The boosted tree model is a sum of such trees,

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m), \tag{216}$$

induced in a forward stagewise manner. At each step in the forward stagewise procedure one must solve

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m)) \tag{217}$$

for the region set and constants $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ of the next tree, given the current model $f_{m-1}(x)$.

59 Numerical Optimisation via Gradient Boosting

59.1 Steepest Descent

Steepest descent chooses $\boldsymbol{h}_m = -\rho_m \boldsymbol{g}_m$, where ρ_m is a scalar and $\boldsymbol{g_m}$ is the gradient of $L(\boldsymbol{f})$ evaluated at $\boldsymbol{f} = \boldsymbol{f}_{m-1}$. The components of the gradient are

$$g_{im} = \Big[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\Big]_{f(x_i)=f_{m-1(x)}}. \tag{218}$$

The *step length* ρ_m is the solution to

$$\rho_m = \arg\min_{\rho} L(\boldsymbol{f}_{m-1} - \rho \boldsymbol{g}_m). \tag{219}$$

The current solution is then updated

$$\boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \rho_m \boldsymbol{g}_m \tag{220}$$

and the process is repeated at the next iteration.

59.2 Gradient Boosting

If minimising the loss on the training data was the ultimate goal, steepest descent would be preferred. However, the ultimate goal is to generalise $f_M(x)$ to new data not presented in the training set. To help in this matter, one can introduce a tree $T(x : \Theta_m)$ at the mth iteration whose predictions $\boldsymbol{t}_m$ are as close as possible to the negative gradient. Using the squared error to measure closeness, this leads us to

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_i - T(x_i; \Theta))^2. \tag{221}$$

59.3 Regularisation

As with other models, it is possible to regularise to prevent the model from overfitting. Typical methods for regularisation include: shrinkage methods and subsampling.

Neural Networks

60 Introduction

We describe a class of learning methods that was developed separately in different fields –statistics and artificial intelligence— based on identical models. The central idea is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features.

61 Projection Pursuit Regression

Assume we have an input vector X with p components, and a target Y. Let $\omega_m, m = 1, 2, \cdots, M$, be unit p-vectors of unknown parameters. The projection pursuit regression (PPR) model has the form

$$f(X) = \sum_{m=1}^{M} g_m(\omega_m^\top X). \tag{222}$$

This is an additive model, but in the derived features $V_m = \omega_m^\top X$ rather than the inputs themselves. The functions g_m are unspecified and are estimated along with the directions ω_m using some flexible smoothing method.

The function $g_m(\omega_m^\top X)$ is called a *ridge function.* It varies only in the direction defined by the vector ω_m. The scalar variable $V_m = \omega_m^\top X$ is the projection of X onto the unit vector ω_m, and we seek $_m$ so that the model fits well — hence the name "projection pursuit".

If M is taken arbitrarily large, for appropriate choice of g_m the PPR model can approximate any continuous function in $\mathbb{R}^p$ arbitrarily as well. These class of models are called a *universal approximator.*

How do we fit a PPR model? we seek the approximate minimizers of the error function

$$\sum_{i=1}^{N} \Big[y_i - \sum_{m=1}^{M} g_m(\omega_m^\top X) \Big]^2 \tag{223}$$

over functions g_m and direction vectors ω_m.

62 Neural Networks

The term "neural network" has evolved to encompass a large class of models and learning methods. Here, we describe the most widely used "vanilla" neural net (or single layer perceptron).

A neural network is a two-stage regression or classification model. For regression, typically $K = 1$ and there is only one output unit Y_1 at the top. However, networks can handle multiple quantitative responses in a seamless fashion, so we will deal with the general case.

For the K-class classification, there are K units at the top, with the kth unit modelling the probability of class k. There are K target measurements $Y_k, k = 1, \cdots, K$, each being coded as a 0–1 variable for the kth class.

Derived features Z_m are created from linear combinations of the inputs, and then the target Y_k is modelled as a function of linear combinations of the Z_m,

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^\top X), m = 1, \cdots, M \tag{224}$$

$$T_k = \beta_{0k} + \beta_k^\top Z, k = 1, \cdots, K \tag{225}$$

$$f_k(X) = g_k(T), k = 1, \cdots, K, \tag{226}$$

where $Z = (Z_1, Z_2, \cdots, Z_M)$, and $T = (T_1, T_2, \cdots, T_K)$. Here (v) is called an activation function, and is usually chosen to be the *sigmoid* $\sigma(v) = 1/(1+e^{-v})$, although ReLU and other functions are also used. Sometimes Gaussian radial basis functions are used, producing what is called as a *radial basis function network*. Moreover, neural networks often also include additional *bias* units feeding into every unit in the hidden and output layers. This bias captures the intercepts $\alpha_{0,m}, \beta_{0k}$ in the model.

In regression, we typically choose the *softmax* function

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^K e^{T_k}}. \tag{227}$$

If the activation function is chosen to be the identity function $(g_m(T) = T_k)$, then the whole neural net collapses to a simple linear model. Hence a neural network can be thought of as a nonlinear generalisation of the linear model. By introducing a nonlinear transformation σ, it greatly enlarges the class of models. Moreover, the number of layers and units (nodes) in each layer is chosen arbitrarily, giving the model large flexibility.

63 Fitting Neural Networks

The neural network model has unknown parameters (weights and biases, θ) and we seek values for them that make the model fit the training data well.

For regression, we use sum-of-squared errors as our measure of fit

$$R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2. \tag{228}$$

For classification, we use either squared error or cross-entropy

$$R(\theta) = -\sum_{i=1}^N \sum_{k=1}^K y_{ik} \log f_k(x_i), \tag{229}$$

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

Typically we don't want a global minimiser of $R(\theta)$, as this is likely to be an overfit solution. Instead some regularisation is needed, obtained by adding a penalty term or by inducing early stopping. The general approach to minimising $R(\theta)$ is by gradient descent (*back-propagation* in this setting). The gradient can be easily derived using the chain rule due to the neural net set up. This can be computed by a forward and backward sweep over the network.

For a squared error loss, letting $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^\top x_i)$, back-propagation can be described as

$$R(\theta) = \sum_{i=1}^{N} R_i \tag{230}$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2, \tag{231}$$

with derivatives

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i)) g_k^{'}(\beta_k^\top z_i) z_{mi}, \tag{232}$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k^{'}(\beta_k^\top z_i) \beta_{km} \sigma^{'}(\alpha_m^\top x_i) x_{i\ell}. \tag{233}$$

Given these derivatives, a gradient descent update at the $(r+1)$st iteration has the form

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \tag{234}$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}}, \tag{235}$$

where γ_r is the *learning rate*. Letting $-2(y_{ik} - f_k(x_i)) g_k^{'}(\beta_k^\top z_i) = \delta_{ki}$ and $-\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k^{'}(\beta_k^\top z_i) \beta_{km} \sigma^{'}(\alpha_m^\top x_i) = s_{mi}$, these constitute the "errors" from the current model at the output and hidden layer units, respectively. From their definitions, the errors satisfy

$$s_{mi} = \sigma^{'}(\alpha_m^\top x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}, \tag{236}$$

known as the *back-propagation equations*. Using this, the updates can be implemented with a two-pass algorithm (forward and backward).

64 Some Issues in Training Neural Networks

There is some level of art in training a neural network. In the following, we discuss some of the important issues when training this sort of model.

64.1 Starting Values

Usually starting values for weights are chosen to be random values near zero. Since weights being zero would collapse the model to a linear one, this choice lets the model start near linear and become nonlinear as the weights increase. Starting with large weights often leads to poor solutions.

64.2 Overfitting

Often neural networks have too many weights and will overfit the data. To mitigate this, early stopping was commonly used. A more explicit method for regularisation is *weight decay*, which is analogous to ridge regression used for linear models. We add a penalty to the error function $R(\theta) + \lambda J(\theta)$, where

$$J(\theta) = \sum_{k,m} \beta_{km}^2 + \sum_{m,\ell} \alpha_{m\ell}^2 \tag{237}$$

and $\lambda \geq 0$ is a tuning parameter.

64.3 Scaling of the Inputs

Since scaling of the inputs determines the effective scaling of the weights in the bottom layer, it can have a large effect on the quality of the final solution. At the outset it is best to standardise all inputs to have mean zero and standard deviation one.

64.4 Number of Hidden Units and Layers

Generally speaking it is better to have too many hidden units than too few. With too few units, the model doesn't have the flexibility to capture the nonlinearities. With too many, the extra weights can be shrunk toward zeros if appropriate regularisation is used. Typically, the number of hidden units ranges from 5-100.

64.5 Multiple Minima

The error function $R(\theta)$ is nonconvex, possessing many local minima. One must at least try a number of random starting configurations and choose the solution giving lowest (penalised) error.

65 Bayesian Neural Nets

65.1 Bayes, Boosting, and Bagging

Given a training data $\boldsymbol{X}_{tr}, \boldsymbol{y}_{tr}$, we assume a sampling model with parameters θ. Given a prior distribution $\Pr(\theta)$, the posterior distribution for the parameters is

$$\Pr(\theta|\boldsymbol{X}_{tr}, \boldsymbol{y}_{tr}) = \frac{\Pr(\theta)\Pr(\boldsymbol{y}_{tr}|\boldsymbol{X}_{tr}, \theta)}{\int \Pr(\theta)\Pr(\boldsymbol{y}_{tr}|\boldsymbol{X}_{tr}, \theta) d\theta}. \tag{238}$$

For a test case with features X_{new}, the predictive distribution for the label Y_{new} is

$$\Pr(Y_{new}|X_{new}, \boldsymbol{X}_{tr}, \boldsymbol{y}_{tr}) = \int \Pr(Y_{new}|X_{new}, \theta)\Pr(\theta|\boldsymbol{X_{tr}, y_{tr}})d\theta. \tag{239}$$

Bagging and Boosting are non-Bayesian procedures that have some similarity to MCMC in a Bayesian model. The Bayesian approach fixes the data and perturbs the parameters, according to the current estimate of the posterior distribution. Bagging perturbs the data and then re-estimates the model to give a new set of model parameters. Boosting is similar to bagging, but fits a model that is additive in the models of each individual base learner. We can write all of these models in the form

$$\hat{f}(\boldsymbol{x}_{\text{new}}) = \sum_{\ell=1}^{L} w_\ell \text{E}(Y_{\text{new}}|\boldsymbol{x}_{\text{new}}, \hat{\theta}_\ell). \tag{240}$$

In all cases $\hat{\theta}_\ell$ are a large collection of model parameters. For the Bayesian model the $w_\ell = 1/L$, and the average estimates the posterior mean by sampling θ_ℓ from the posterior distribution. For bagging, $w_\ell = 1/L$ as well, and the $\hat{\theta}_\ell$ are the parameters refit to bootstrap resamples of the training data. For boosting, the weights are all equal to 1, but the $\hat{\theta}_\ell$ are typically chosen in a nonrandom sequential fashion to constantly improve the fit.

Support Vector Machines and Flexible Discriminants

66 Introduction

We now cover extensions of the nonseparable case of optimal separation of hyperplanes, generalised to what is known as *support vector machine.* These produce non-linear boundaries by constructing a linear boundary in a large, transformed version of the feature space. The second set of methods generalise Fisher's linear discriminant analysis, these include: *flexible discriminant analysis* for construction of non-linear boundaries; *penalised discriminant analysis* for problems like signal and image classification where a high number of features are highly correlated; *mixture discriminant analysis* for irregular shaped classes.

67 The Support Vector Classifier

Consider a training data of N pairs (x_1, y_1), (x_2, y_2), $\cdots$, (x_N, y_N), with $x_i \in \mathbb{R}^{\prime}$ and $y_i \in \{-1, 1\}$. Define the hyperplane by

$$\{x : f(x) = x^\top \beta + \beta_0 = 0\}, \tag{241}$$

where β is a unit vector: $||\beta|| = 1$. A classification rule induced by $f(x)$ is

$$G(x) = \mathrm{sign}[x^\top \beta + \beta_0]. \tag{242}$$

Using this relation we are able to find the hyperplane that creates the biggest *margin* M between the training points of class 1 and -1.

Suppose now that the classes overlap in feature space. One way to deal with the overlap is to still maximise M, where $y_i(x_i^\top \beta + \beta_0) \geq M$, but allow for some points to be on the wrong side of the margin. Define the slack variables $\xi = (\xi_1, \xi_2, \cdots, \xi_N)$. There are two natural ways to modify the constraint listed above:

$$y_i(x_i^\top \beta + \beta_0) \geq M - \xi_i \tag{243}$$

$$y_i(x_i^\top \beta + \beta_0) \geq M(1 - \xi_i) \tag{244}$$

$$\tag{245}$$

The two choices lead to different solutions. The first choice seems more natural, since it measure overlap in actual distance from the margin; the second measures the overlap in relative distance, which changes with the width of the margin M. The first choice leads to nonconvex problems, whereas the second to convex. Thus, the second is the "standard" approach.

The idea of the formulation is the following: the value ξ_i in the constraint $y_i(x_i^\top \beta + \beta_0) \geq M(1 - \xi_i)$ is the proportional amount by which the prediction $f(x_i) = x_i^\top \beta + \beta_0$ is on the wrong side of its margin. Hence, by bounding the sum $\sum \xi_i$ we bound the total proportional amount by which predictions fall on the wrong side of the margin. Misclassifications occur when $\xi_i > 1$.

68 Support Vector Machines and Kernels

So far we have described a SV that finds linear boundaries in the feature space. As with other linear methods, we can make this procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines. Once the basis functions $h_m(x), m = 1, \cdots, M$ are selected, the procedure is the same as before. We fit the SV classifier using input features and produce the (nonlinear) function $\hat{f}(x_i) = h(x)^\top \hat{\beta} + \hat{\beta}_0$.

The *support vector machine* classifier is an extension of this idea, where the dimension of the enlarged space is allowed to get very large.

68.1 Computing the SVM for classification

The Lagrange dual function has the form

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_i^{'} y_i y_i^{'} \langle h(x_i), h(x_i^{'}) \rangle. \tag{246}$$

The solution function $f(x)$ can be written

$$f(x) = h(x)^\top \beta + \beta_0 \tag{247}$$

$$= \sum_{i=1}^{N} \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0. \tag{248}$$

So both involve $h(x)$ through inner products. We in fact do not specify the transformation $h(x)$ at all, we ony require knowledge of the kernel function

$$K(x, x^{'}) = \langle h(x), h(x^{'}) \rangle \tag{249}$$

that computes inner products in the transformed space. Three popular choices for K in the SVM literature are:

- dth-Degree polynomial: $K(x, x^{'}) = (1 + \langle x, x^{'} \rangle)^d$,
- Radial basis: $K(x, x^{'}) = \exp(-\gamma ||x - x^{'}||^2)$,
- $K(x, x^{'}) = \tanh(\kappa_1 \langle x, x^{'} \rangle + \kappa_2)$.

68.2 The SVM as a Penalisation Method

With $f(x) = h(x)^\top \beta + \beta_0$, consider the optimisation problem

$$\min_{\beta_0, \beta} \sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + \frac{\lambda}{2} ||\beta||^2 \tag{250}$$

where the subscript + indicates the positive part. This has the form *loss* plus *penalty*. It is possible to show that the solution to this relation, with $\lambda = 1/C$, is the same as

$$\min_{\beta_0,\beta} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}\xi_i. \tag{251}$$

The higher C, the more prone to overfitting.

68.3 Support Vector Machines for Regression

We first discuss a linear regression model

$$f(x) = x^\top \beta + \beta_0, \tag{252}$$

and then handle nonlinear generalisations. To estimate β, we consider minimisation of

$$H(\beta, \beta_0) = \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2}||\beta||^2, \tag{253}$$

where

$$V_\epsilon(r) = \begin{cases} 0 \text{ if } |r| < \epsilon; |r| - \epsilon \text{ otherwise.} \end{cases} \tag{254}$$

This is an "ϵ-insensitive" error measure, ignoring errors of size less than ϵ.

In addition, as for classification, regression can incorporate kernels (see Section 12.3.7 in the "The Elements of Statistical Learning" book for futher details).

69 Generalising Linear Discriminent Analysis

69.1 Flexible Discriminant Analysis

Here we describe a method for using LDA using linear regression on derived responses. This in turn leads to nonparametric and flexible alternatives to LDA. If our training sample has the form $(g_i, x_i), i = 1, 2, \cdots, N$, then we solve

$$\min_{\beta,\theta} \sum_{i=1}^{N} (\theta(g_i) - x_i^\top \beta)^2, \tag{255}$$

with restrictions on θ to avoid a trivial solution.

More generally, we an find up to $L \leq K - 1$ sets of independent scorings for the class labels, $\theta_1, \theta_2, \cdots, \theta_L$ and L corresponding linear maps $\eta_\ell(X) = X^\top \beta_\ell, \ell = 1, \cdots, L$, chosen to be optimial for multiple regression. The scores $\theta_\ell(g)$ and the maps β_ℓ are chosen to minimise the average squared residual.

In this more general form, the regression problems are defined via the criterion

$$ASR(\{\theta_\ell, \eta_\ell\}_{\ell=1}^{L}) = \frac{1}{N}\sum_{\ell=1}^{L}\left[\sum_{i=1}^{N}(\theta_\ell(g_i) - \eta_\ell(x_i))^2 + \lambda J(\eta_\ell)\right], \tag{256}$$

where J is the regularises appropriate for some forms of nonparametric regression, such as smoothing splines, additive splines, and lower-order ANOVA spline models.

70 Penalised Discriminant Analysis

Although FDA is motivated by generalising optimal scoring, it can also be viewed directly as a form of regularised discriminant analysis. Suppose the regression procedure used in FDA amounts to a linear regression onto a basis expansion $h(X)$, with a quadratic penalty on the coefficients:

$$ASR(\{\theta_\ell, \eta_\ell\}_{\ell=1}^{L}) = \frac{1}{N}\sum_{\ell=1}^{L}\left[\sum_{i=1}^{N}(\theta_\ell(g_i) - h^\top(x_i)\beta_\ell)^2 + \lambda\beta_\ell^\top \mathbf{\Omega}\beta_\ell\right]. \tag{257}$$

The choice of $\mathbf{\Omega}$ depends on the problem.

The steps in FDA can then be viewed as a generalised form of LDA, which we call *penalised discriminant analysis* (PDA):

- Enlarge the set of predictors X via a basis expansion $h(X)$.
- use penalised LDA in the enlarged space.
- Decompose the classification subspace using a penalised metric.

71 Mixture Discriminant Analysis

LDA can be viewed as a prototype classifier. Each class is represented by a centroid, and we classify to the closest using an appropriate metric. In many cases a single prototype is not sufficient to represent inhomogeneous classes, and mixture models are more appropriate. Here we review Gaussian mixture models and show how they can be generalised via the FDA and PDA methods.

A Gaussian mixture model for the kth class has density

$$P(X|G=k) = \sum_{r=1}^{R_k}\pi_{kr}\phi(X;\mu_{kr},\Sigma), \tag{258}$$

where the *mixing proportions* π_{kr} sum to one. Given this, the class posterior probabilities are given by

$$P(G=k|X=x) = \frac{\sum_{r=1}^{R_k}\pi_{kr}\phi(X;\mu_{kr},\Sigma)\Pi_k}{\sum_{\ell=1}^{K}\sum_{r=1}^{R_k}\pi_{\ell r}\phi(X;\mu_{\ell r},\Sigma)\Pi_\ell}, \tag{259}$$

where Π_k represents the class prior probabilities. As in LDA, we estimate the parameters by maximum likelihood, using the joint log-likelihood based on $P(G,X)$:

$$\sum_{k=1}^{K}\sum_{g_i=k}\log[\sum_{r=1}^{R_k}\pi_{kr}\phi(x_i;\mu_{kr},\Sigma)\Pi_k]. \tag{260}$$

The used method for computing maximum likelihood estimates for mixture distributions is the EM algorithm. EM alternates between the two steps:

- E-step: Given current parameters, compute the *responsibility* of subclass c_{kr} within class k for each of the class-k observations ($g_i = k$).
- M-step: Compute the weighted maximum likelihood estimates for the parameters of each of the component Gaussians within each of the classes, using the weights from the E-step.

72 Computational Considerations

with N training cases, p predictors, and m support vectors, the SVM requires $m^3 + mN + mpN$ operations, assuming $m \approx N$. They do not scale well with N. LDS requires $Np^2 + p^3$ operations, as does PDA.

Prototype Methods and Nearest-Neighbours

73 Introduction

Here we describe some simple and essentially model-free methods for classification and pattern recognition. Because of they are highly unstructured, they often are not useful for understanding the relationships between the features and class outcome. However, they can be very effective.

74 Prototype Methods

Consider training data of N pairs $\{x_i, g_i\}$ where g_i is a class label taking values in $\{1, 2, \cdots, K\}$. Prototype methods represent the training data by a set of points in a feature space. These prototypes are typically not examples from the training sample. Each prototype has an associated class label, and classification of a query point x is made to the class of the closest (typically defined in Euclidean distance of feature space) prototype.

These methods can be very effective if the prototypes are well positioned to capture the distribution of each class. Irregular class boundaries can be represented, with enough prototypes in the right places in feature space. The main challenge is to figure out how many prototypes to use and where to put them.

74.1 K-means clustering

K-means clustering is a method for finding clusters and cluster centres in a set of unlabeled data. One chooses a desired number of cluster centres, and the K-means procedure iteratively moves the centres to minimise the total within cluster variance. It performs this in two steps:

- For each centre we identify the subset of training points that is closer to it than any other centre;
- the means of each feature for the data points in each cluster are computed, and this mean vector becomes the new centre for that cluster.
- Both steps are iterated until convergence.

74.2 Learning Vector Quantisation

Here, prototypes are placed strategically with respect to the decision boundaries in an ad-hoc way. LVQ is an online algorithm— observations are processed one at a time.

The idea is that the training points attract prototypes of the correct class, and repel the other prototypes. The learning rate ϵ is decreased to zero with each iteration, following the guidelines for stochastic approximation learning rates.

74.3 Gaussian Mixtures

The Gaussian Mixture Model (GMM) can also be thought of as a prototype method. Each cluster is described of a Gaussian density, which has a centroid and a covariance matrix.

The two steps of the alternating EM algorithm are very similar to that in K-means:

- In each E-step, each observation is assigned a *responsibility* or weight for each cluster, based on the likelihood of each of the corresponding Gaussians.
- In the M-step, each observation contributes to the weighted means (and covariances) for every cluster.

75 k-Nearest-Neighbour Classifiers

These classifiers are *memory-based*, and 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 then classify using majority vote among the k neighbours.

Ties are broken at random, and the model uses Euclidean distance in feature space, such that:

$$d_{(i)} = ||x_{(i)} - x_o||. \tag{261}$$

Typically, the features are first standardised to have mean zero and variance 1, since it is possible that they are measured in different units. k-nearest-neighbours has proven successful in situations where each class has many prototypes, and the decision boundary is very irregular.

Because it uses only the training point closest to the query point, the bias of the 1-nearest-neighbour estimate is often low, but the variance is high.

76 Adaptive Nearest-Neighbour Methods

When nearest-neighbour classification is carried out in a high-dimensional feature space, the nearest neighbours of a point can be very far away, causing bias and degrading the performance of the rule.

To solve this, it is possible to adapt the metric used in nearest-neighbour classification, so that the resulting neighbourhoods stretch out in directions for which the class probabilities don't change much. In high-dimensional feature space, the class probabilities might change only a low-dimensional subspace, and hence there can be considerable advantage to adapting the metric.

Friedman (1994a) proposed a method in which rectangular neighbourhoods are found adaptively by successively carving away edges of a box containing the training data. This lead to the *discriminant adaptive nearest-neighbour* (DANN) rule of Hastie and Tibshirani (1996a).

At each query point a neighbourhood of say 50 points is formed, and the class distribution among the points used to decide how to deform the neighbourhood — that is, to adapt the metric. The adapted metric is then used in a nearest-neighbour rule at the query point. Thus, at each query point a potentially different metric is used.

The DANN metric at a query point x_0 is defined by

$$D(x, x_0) = (x - x_0)^\top \Sigma (x - x_0), \tag{262}$$

where

$$\Sigma = W^{-1/2}[W^{-1/2} B W^{-1/2} + \epsilon I] W^{-1/2}. \tag{263}$$

Here W is the pooled within-class covariance matrix $\sum_{k=1}^{K} \pi_k W_k$ and B is the between class covariance matrix $\sum_{k=1}^{K} \pi_k (\bar{x_k} - \bar{x})(\bar{x_k} - \bar{x})^\top$, with W and B computed using only the 50 nearest neighbours around x_0.

76.1 GLobal Dimension Reduction for Nearest-Neighbours

The discriminant-adaptive nearest-neighbour method carries out local dimension reduction — that is, the dimension reduction separately at each query point. Thus, it can also be used for global dimension reduction.

At each training point, x_i, the between centroids sum of squares matrix B_i is computed, and then the matrices are averaged over all training points:

$$\bar{B} = \frac{1}{N} \sum_{i=1}^{N} B_i. \tag{264}$$

Let $e_1, e_2, \cdots, e_p$ be the eigenvectors of the matrixx $\bar{B}$, ordered from largest to smallest eigenvalue θ_k. Then these eigenvectors span the optimal subspaces for global subspace reduction.

Unsupervised Learning

77 Introduction

We now address *unsupervised learning.* Here, you have a set of observations $(x_1, x_2, \cdots, x_N)$ of a random p vector X having a joint density $\Pr(X)$. The goal is to directly infer the properties of this probability density without the help of a supervisor. The dimension of X is sometimes much higher than in supervised learning, and the properties of interest are often more complicated than simple location estimates.

77.1 Association rules

Association rule analysis has emerged as a popular tool for mining commercial data bases. The goal is to find joint values of the variables $X = (X_1, X_2, \cdots, X_p)$ that appear most frequently in the data base. It is most often applied in binary data $X \in \{0, 1\}$, where it is referred to as "market basket" analysis. In this context the observations are sales transactions.

More generally, the basic goal of association rule analysis is to find a collection of prototype X-values $v_1, \cdots, v_L$ for the feature vector X, such that the probability density $\Pr(v_l)$ evaluated at each of those values is relatively large.

The first simplification modifies the goal. Instead of seeking values x where $\Pr(x)$ is large, one seeks *regions* of the X-space with high probability content relative to their size of support. Let $\mathcal{S}_j$ represent the set of all possible values of the jth variable (its *support*), and let $s_j \subseteq \mathcal{S}_j$ be a subset of these values. The modified goal can be stated as attempting to find subsets of variable values $s_1, \cdots, s_p$ such that the probability of each of the variables simultaneously assuming a value within its respective subset,

$$\Pr\left[\bigcap_{j=1}^{p}(X_j \in s_j)\right] \tag{265}$$

is relatively large. The intersection of subsets is called a *conjunctive rule.*

77.1.1 Market Basket Analysis

General approaches to solving the equation above are not feasible for very large commercial data ($p \approx 10^4; N \approx 10^8$). Several further simplifications are required. First, only two types of subsets are considered; either s_j consists of a *single* value of $X_j, s_j = v_{0j}$, or it consists of the entire set of values that X_j can assume, $s_j = \mathcal{S}_j$. This simplifies the problem to finding subsets of the intergers $\mathcal{J} \subset \{1, , \cdots, p\}$, and corresponding values $v_{0j}, j \in \mathcal{J}$, such that

$$\Pr\left[\bigcap_{j \in \mathcal{J}}^{p}(X_j = v_{0j})\right] \tag{266}$$

is large.

One can apply the technique of *dummy variables* to turn into a problem involving only binary-values variables. Here we assume that the support $\mathcal{S}_|$ is finite for each variable X_j. Specifically, a new set of variables $Z_1, \cdots, Z_k$ is created, one such variable for each of the values v_{lj} attainable by each of the original variables $X_1, \cdots, X_p$. The number of dummy variables is

$$K = \sum_{j=1}^{p} |\mathcal{S}_j|, \tag{267}$$

where $|\mathcal{S}_j|$ is the number of distinct values attainable by X_j. Each dummy variable is assigned the value $Z_k = 1$ if the variable with which it is associated takes on the corresponding value to which Z_k is assigned, and $Z_k = 0$ otherwise. This transforms the relation to finding a subset of the integers $\mathcal{K} \subset \{1, \cdots, K\}$ such that

$$\Pr\left[\bigcap_{k\in\mathcal{K}} (Z_k = 1)\right] = \left[\prod_{k\in\mathcal{K}} Z_k = 1\right] \tag{268}$$

is large. This is the standard formulation of the market basket problem.

The solution to this problem can be obtained with feasible computation for very large data bases using the Apriori Algorithm provided the threshold t is adjusted so that $\{\mathcal{K}_l | T(\mathcal{K}_l) > t\}$ consists of only a small fraction of all 2^K possible item sets.

77.2 Unsupervised as Supervised Learning

Let $g(x)$ be the unknown data probability density to be estimated, and $g_0(x)$ be a specified probability density function used for reference. The data set $x_1, x_2, \cdots, x_N$ is presumed to be an independent and identically distributed (*i.i.d*) random sample drawn from $g(x)$. Pooling from these two data sets, and assigning mass $w = N_0/(N + N_0)$ to those drawn from $g(x)$ (same for $g_0(x)$ using w_0), results in a random sample drawn from the mixture density $(g(x) + g_0(x))/2$. If one assigns the value $Y = 1$ to each sample point drawn from $g(x)$ and $Y = 0$ those from $g_0(x)$, then

$$\mu(x) = E(Y|X) = \frac{g(x)}{g(x) + g_0(x)} = \frac{g(x)/g_0(x)}{1 + g(x)/g_0(x)} \tag{269}$$

can be estimated by supervised learning using the combined sample

$$(y_1, x_1), (y_2, x_2), \cdots, (y_{N+N_0}, x_{N+N_0}) \tag{270}$$

as training data.

77.3 Generalised Association Rules

The more general prolem of finding high-density regions in the data space can be addressed using the supervised learning approach. The problem can be formulated as finding subsets of the integers $\mathcal{J} \subset \{1, 2, \cdots, p\}$ abd corresponding value subsets $s_j, j \in \mathcal{J}$ for the corresponding variables X_j, such that

$$\widehat{\Pr}\left(\bigcap_{j\in\mathcal{J}}(X_j \in s_j)\right) = \frac{1}{N}\sum_{i=1}^{N}\left(\bigcap_{j\in\mathcal{J}}(x_{ij} \in s_j)\right) \tag{271}$$

is large.

78 Cluster Analysis

Cluster analysis, also known as data segmentation, has a variety of goals, that all relate to grouping or segmenting a collection of objects into subsets that are more closely related.

Central to all of the goals of cluster analysis is the notion of the degree of similarity between the individual objects being clustered. A clustering method attempts to group the objects based on the definition of similarity supplied to it.

78.1 Proximity Matrices

Sometimes the data is represented directly in terms of the proximity between pairs of objects. Most algorithms presume a matrix of disimilarities with non-negative entries and zero diagonal elements: $d_{ii} = 0, i = 1.2, \cdots, N$.

78.2 Dissimilarities based on attributes

Most often we have measurements x_{ij} for $i = 1, 2, \cdots, N$, on variables (or attributes) $j = 1, 2, \cdots, p$. In the most common case, the dissimilarity $d_j(x_{ij}, x_{i'j})$ between values of the jth attribute define

$$D(x_i, x_{i'}) = \sum_{j=1}^{p} d_j(x_{ij}, x_{i'j}) \tag{272}$$

where by far the most common choice is square distance. However, other choices are possible. Some alternatives include: absolute difference or absolute error for quantitative variables; for ordinal variables, error measures are generally defined by replacing their M original value with $\frac{i=1/2}{M}$.

78.3 Object Dissimilarity

We define a procedure for combining the p-individual attribute dissimilarities $d_j(x_{ij}, x_{i'j})$ into a single overall measure of dissimilarity $D(x_i, x_{i'})$ between two objects or observations. This is nearly always done by means of a weighted average

$$D(x_i, x_{i'}) = \sum_{j=1}^{p} w_j \cdot d_j(x_{ij}, x_{i'j}); \sum_{j=1}^{p} w_j = 1. \tag{273}$$

Here w_j is a weight assigned to the jth attribute regulating the relative influence of that variable in determining the overall dissimilarity between objects. If there are missing values in one or more attributes, the most common method is to omit each observation pair when computing the dissimilarity.

78.4 Clustering Algorithms

The goal of cluster analysis is to partition the observations into groups so that the pairwise dissimilarities between those assigned to the same cluster tend to be smaller than those in different clusters. These fall into three distinct types: combinatorial algorithms, mixture modelling, and mode seeking.

Combinatorial algorithms work directly on the observed data with no underlying probability model. Mixture modelling supposes that the data is an i.i.d sample from some population described by a probability density function. Mode seekers take a nonparametetric perspective, attempting to directly estimate distinct modes of the probability density function.

78.5 Combinatorial Algorithms

The goal is to assign close points to the same cluster in a space on K different clusters, using a loss function like

$$W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d(x_i, x_{i'}) \tag{274}$$

where C is the encoder.

78.6 K-means

The K-means algorithm is one of the most popular iterative descent clustering methods. It is intended for situations in which all variables are of the quantitative type, and squared Euclidean distance

$$d(x_i, x_{i'}) = \sum_{j=1}^{p}(x_{ij} - x_{i'j})^2 = ||x_i - x_{i'}||^2 \tag{275}$$

is chosen as the dissimilarity measure.

78.7 Gaussian Mixtures as Soft K-means Clustering

The K-means clustering procedure is closely related to the EM algorithm for estimating a certain Gaussian mixture model. The E-step of the EM algorithm assigns “responsibilities” for each data point based in its relative density under each mixture component, while the M-step recomputes the component density parameters based on the current responsibilities. Suppose we specify K mixture components, each with a Gaussian density having scalar covariance matrix $\sigma^2\boldsymbol{I}$.

Then the relative density under each mixture component is a monotone function of the Euclidean distance between the data point and the mixture center. Hence in this setup EM is a "soft" version of K-means clustering, making probabilistic (rather than deterministic) assignments of points to cluster centers.

78.8 Vector Quantization

The K-means clustering algorithm represents a key tool in the apparently unrelated area of image and signal compression, particularly in vector quantization or VQ. VQ breaks an image into small blocks, and a K-means clustering algorithm is ran in this space. Each of the pixel blocks (or points) is approximated by its closest cluster centroid, known as a codeword. The clustering process is called the encoding step, and the collection of centroids is called the codebook.

78.9 K-medoids

the K-means algorithm is appropriate when the dissimilarity measure is taken to be squared Euclidean distance. This requires all of the variables to be of the quantitative type. In addition, using squared Euclidean distance places the highest influence on the largest distances. This causes the procedure to lack robustness against outliers that produce very large distances. These restrictions can be removed at the expense of computation. One can replace the eucledian distance minimisation step by an explicit optimisation with respect to $\{m_1, \cdots, m_K\}$ in

$$\min_{C,\{m_k\}_1^K} \sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - m_k||^2. \tag{276}$$

78.10 Hierarchical clustering

Hierarchical clustering methods do not require the choice for the number of clusters to be searched and a starting configuration assignment. Instead, they require the user to specify a measure of dissimilarity between (disjoint) groups of observations, based on the pairwise dissimilarities among the observations in the two groups. Strategies for hierarchical clustering divide into two basic paradigms: agglomerative (bottom-up) and divisive (top-down).

A dendrogram provides a highly interpretable complete description of the hierarchical clustering in a graphical format. The extent to which the hierarchical structure produced by a dendrogram actually represents the data itself can be judged by the cophenetic correlation coefficient. This is the correlation between the $N(N1)/2$ pairwise observation dissimilarities d_{ii} input to the algorithm and their corresponding cophenetic dissimilarities $C_{ii'}$ derived from the dendrogram.

79 Self-Organising Maps

This method can be seen as a constrained version of K-means clustering, in which prototypes are encouraged to lie in a one- or two-dimensional manifold in the feature space.

Consider a self-organising map (SOM) algorithm with a two-dimensional rectangular grid of K prototypes $m_j \in \mathbb{R}^p$. Each of the K prototypes are parametrized with respect to an integer coordinate pair $\ell_j \in \mathcal{Q}_1 \times \mathcal{Q}_\in$. Here $\mathcal{Q}_\infty = \{1, 2, \cdots, q_1\}$, same for $\mathcal{Q}_\in$, and $K = q_1 \cdot q_2$. We can think of the prototypes as "buttons," "sewn" on the principal component plane in a regular pattern. The SOM procedure tries to bend the plane so that the buttons approximate the data points as well as possible. Once the model is fit, the observations can be mapped down onto the two-dimensional grid. Observations are processed one at a time, finding the closest prototype m_j to observation x_i in Euclidean distance, and then for all neighbours via the update

$$m_k \leftarrow m_k + \alpha(x_i - m_k). \tag{277}$$

The performance of the SOM algorithm depends on the learning rate α and the distance threshold r.

80 Principal Components, Curves, and Surfaces

Principal components are a sequence of projections of the data, mutually uncorrelated and ordered in variance. Here we present principal components as linear manifolds for approximating a set of N points. Principal components are a useful tool for dimension reduction and compression.

Denote observations by $x_1, x_2, \cdots, x_N$ and consider the rank-q linear model for representing them

$$f(\lambda) = \mu + \boldsymbol{V}_q \lambda \tag{278}$$

where μ is a location vector in $\mathbb{R}^p$, $\boldsymbol{V}_q$ is a $p \times q$ matrix with q orthogonal unit vectors as columns, and λ is a q vector of parameters.

One can fit this model to data by least squares amounts to minimize the *reconstruction error*

$$\min_{\mu, \{\lambda_i\}, \boldsymbol{V}_q} \sum_{i=1}^{N} ||x_i - \mu - \boldsymbol{V}_1 \lambda_i||^2. \tag{279}$$

The $p \times p$ matrix $\boldsymbol{H}_q = \boldsymbol{V}_q \boldsymbol{v_q}^\top$ is the *projection matrix*, and maps each point x_i onto its rank-q reconstruction $\boldsymbol{H}_q x_i$.

80.1 Principal curves and Surfaces

Principal curves generalize the principal component line, providing a smooth one-dimensional curved approximation to a set of data points. A principal surface is more general, providing a curved manifold approximation of dimension 2 or more.

Let $f(\lambda)$ be a parameterized smooth curve in $\mathbb{R}^p$. Hence $f(\lambda)$ is a vector function with p coordinates, each a smooth function of the single parameter λ. The parameter λ can be chosen, for example, to be arc-length along the curve from some fixed origin. For each data value x, let $\lambda_f(x)$ define the closest point on the curve to x. Then $f(\lambda)$ is called a principal curve for the distribution of the random vector X if

$$f(\lambda) = E(X|\lambda_f(X) = \lambda). \tag{280}$$

Similarly, principal surfaces have exactly the same form as principal curves, but are of higher dimension. The mostly commonly used is the two-dimensional principal surface, with coordinate functions

$$f(\lambda_1, \lambda_2) = [f_1(\lambda_1, \lambda_2), \cdots, f_p(\lambda_1, \lambda_2)]. \tag{281}$$

80.2 Spectral Clustering

Spectral clustering is a generalization of standard clustering methods, and is designed for situations when the clusters are non-convex.

The starting point is a $N \times N$ matrix of pairwise similarities $s_{ii} \geq 0$ between all observation pairs. We represent the observations in an undirected similarity graph $G = \langle V, E \rangle$. The N vertices v_i represent the observations, and pairs of vertices are connected by an edge if their similarity is positive (or exceeds some threshold). The edges are weighted by the s_{ii} . Clustering is now rephrased as a graph-partition problem, where we identify connected components with clusters. We wish to partition the graph, such that edges between different groups have low weight, and within a group have high weight. The idea in spectral clustering is to construct similarity graphs that represent the local neighborhood relationships between observations.

80.3 Kernel Principal Components

Spectral clustering is related to kernel principal components, a non-linear version of linear principal components. Standard LPC (PCA) are obtained from the eigenvectors of the covariance matrix, and give directions in which the data have maximal variance. Kernel PCA expand the scope of PCA, mimicking what we could obtain if we were to expand the features by non-linear transformations, and then apply PCA in this transformed feature space.

The principal component variables of $\boldsymbol{Z}$ of data matrix $\boldsymbol{X}$ can be computed from the inner-product matrix $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^\top$. Kernel PCA simply mimics this procedure, interpreting the kernel matrix as an inner-product matrix of the implicit features and finding its eigenvectors.

We can gain more insight into kernel PCA by viewing the $\boldsymbol{z}_m$ as sample evaluations of principal component functions $g_m \in \mathcal{H}_K$, with $\mathcal{H}_K$ the reproducing kernel Hilbert space generated by K. Kernel PCA finds the eigenvectors corresponding to the largest eigenvalues of $\tilde{\boldsymbol{K}}$; this is equivalent to finding the eigenvectors corresponding to the smallest eigenvalues of $\boldsymbol{I} - \tilde{\boldsymbol{K}}$.

80.4 Sparse Principal Components

We often interpret PCs by examining the direction vectors, v_j, also known as loadings. Here we briefly discuss some methods for deriving PCs with sparse loadings, based on the Lasso penalties.

For multiple components, the sparse principal components procedure minimizes

$$\sum_{i=1}^{N} ||x_i - \Theta \boldsymbol{V}^\top x_i||^2 + \lambda \sum_{k=1}^{K} ||v_k||_2^2 + \sum_{k=1}^{K} \lambda_{1,k} ||v_k||_1, \tag{282}$$

subject to $\Theta^\top \Theta = \boldsymbol{I}_K$.

81 Non-Negative Matrix Factorisation

Non-negative matrix factorisation is a recent alternative approach to PCA, in which data and components are assumed to be non-negative (typically useful for modelling non-negative data like images).

The $N \times p$ data matrix $\boldsymbol{X}$ is approximated by $\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}$, where $\boldsymbol{W}$ is $N \times r$ and $\boldsymbol{H}$ is $r \times p$. The matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ are found by maximising

$$L(\boldsymbol{W}, \boldsymbol{H}) = \sum_{i=1}^{N} \sum_{j=1}^{p} [x_{ij} \log(\boldsymbol{W}\boldsymbol{H})_{ij} - (\boldsymbol{W}\boldsymbol{H})_{ij}]. \tag{283}$$

This is the log-likelihood from a model in which x_{ij} has a Poisson distribution with mean $(\boldsymbol{W}\boldsymbol{H})_{ij}$ — quite reasonable for positive data.

81.1 Archetypal analysis

This method, due to Cutler and Breiman (1994), approximates data points by prototypes that are themselves linear combinations of data points. In this sense it has a similar flavor to K-means clustering. However, rather than approximating each data point by a single nearby prototype, archetypal analysis approximates each data point by a convex combination of a collection of prototypes. The use of a convex combination forces the prototypes to lie on the convex hull of the data cloud. In this sense, the prototypes are "pure,", or "archetypal."

Non-negative matrix factorization aims to approximate the columns of the data matrix $\boldsymbol{X}$, and the main output of interest are the columns of $\boldsymbol{W}$ representing the primary non-negative components in the data. Archetypal analysis focuses instead on the approximation of the rows of X using the rows of $\boldsymbol{H}$, which represent the archetypal data points.

82 Independent Component Analysis and Exploratory Projection Pursuit

Multivariate data are often viewed as multiple indirect measurements arising from an underlying source, which typically cannot be directly measured. Examples include: Educational and psychological tests; EEG brain scans; trading prices of stock change that change constantly over time.

Factor analysis is a classical technique developed in the statistical literature that aims to identify these latent sources. Factor analysis models are typically wed to Gaussian distributions, which has to some extent hindered their usefulness. More recently, independent component analysis has emerged as a strong competitor to factor analysis, and as we will see, relies on the non-Gaussian nature of the underlying sources for its success.

82.1 Latent Variables and Factor Analysis

The singular-value decomposition $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$ has a latent variable representation. Writing $\boldsymbol{S} = \sqrt{N}\boldsymbol{U}$ and $\boldsymbol{A}^\top = \boldsymbol{D}\boldsymbol{V}^\top/\sqrt{N}$, we have $\boldsymbol{X} = \boldsymbol{S}\boldsymbol{A}^\top$. Now since $\boldsymbol{U}$ is orthogonal, and assuming as before that the columns of $\boldsymbol{X}$ (and hence $\boldsymbol{U}$) each have mean zero, this implies that the columns of $\boldsymbol{S}$ have zero mean, are uncorrelated and have unit variance. In terms of random variables, we can interpret the SVD, or the corresponding principal component analysis (PCA) as an estimate of a latent variable model $X = \boldsymbol{A}S$.

The classical factor analysis model has the form $X = \boldsymbol{A}S + \epsilon$, where S is a vector of underlying latent variables or factors, $\boldsymbol{A}$ is a matrix of factor loadings, and ϵ_j are uncorrelated zero-mean disturbances. Typically, S_ℓ and ϵ_j are modelled as Gaussian random variables, and the model is fit by maximum likelihood.

82.2 Independent Component Analysis

The independent component analysis (ICA) model has exactly the same form as factor analysis, except the S_ℓ are assumed to be statistically independent rather than uncorrelated. Intuitively, lack of correlation determines the second-degree cross-moments (covariances) of a multivariate distribution, while in general statistical independence determines all of the crossmoments. These extra moment conditions allow us to identify the elements of $\boldsymbol{A}$ uniquely.

ICA applied to multivariate data looks for a sequence of orthogonal projections such that the projected data look as far from Gaussian as possible. With pre-whitened data, this amounts to looking for components that are as independent as possible. ICA starts from essentially a factor analysis solution, and looks for rotations that lead to independent components. From this point of view, ICA is just another factor rotation method, along with the traditional "varimax" and "quartimax" methods used in psychometrics.

82.3 Exploratory Projection Pursuit

Exploratory projection pursuit is a graphical exploration technique for visualizing high-dimensional data. As most low (one- or two-dimensional) projections of high dimensional data look Gaussian, interesting structure, (such as clusters or long tails) would be revealed by non-Gaussian projections.

82.4 A direct approach to ICA

Independent components have by definition a joint product density

$$f_S(s) = \prod_{j=1}^{p} f_j(s_j), \tag{284}$$

so here we present an approach that estimates this density directly using generalized additive models. In the spirit of representing departures from Gaussianity, we represent each f_j as

$$f_j(s_j) = \phi(s_j)e^{g_j(s_j)} \tag{285}$$

a titled Gaussian density. Here ϕ is the standard Gaussian density, and g_j satisfies the normalization conditions required of a density. The log-likelihood for the observed data $X = \boldsymbol{A}S$ is

$$\ell(\boldsymbol{A}, \{g_j\}_1^p; \boldsymbol{X}) = \sum_{i=1}^{N}\sum_{j=1}^{p}[\log \phi_j(a_j^\top x_i) + g_j(a_j^\top x_i)], \tag{286}$$

where if the model is over-parameterised, we can include a regularisation in L1 or L2. We then fit the functions g_j and directions a_j by optimising in an alternate fashion, such that:

1. Initialise $\boldsymbol{A}$
2. Alternate until convergence of $\boldsymbol{A}$: a) Given $\boldsymbol{A}$, optimise the likelihood w.r.t. g_j (separately for each j; b) given g_j, perform one step of a fixed point algorithm towards finding the optimal $\boldsymbol{A}$.

83 Multidimensional Scaling

Both self-organizing maps and principal curves and surfaces map data points in $\mathbb{R}^p$ to a lower-dimensional manifold. Multidimensional scaling (MDS) has a similar goal, but approaches the problem in a somewhat different way.

Start with observations $x_1, x_2, \cdots, x_N$, and let d_{ij} be the distance between observations i and j. Often we use Euclidean distance. Multidimensional scaling seeks values $z_1, z_2, \cdots, z_N$ to minimise the so-called *stress function*

$$S_M(z_1, z_2, \cdots, z_N) = \sum_{i \neq i'}(d_{ii'} - ||z_i - z_{i'}||)^2. \tag{287}$$

This is known as least squares or Kruskal–Shephard scaling. The idea is to find a lower-dimensional representation of the data that preserves the pairwise distances as well as possible. Like the self-organizing map and principal surfaces, multidimensional scaling represents high-dimensional data in a low-dimensional coordinate system.

84 Nonlinear Dimenion Reduction and Local Multidimensional Scaling

Several methods have been recently proposed for nonlinear dimension reduction, similar in spirit to principal surfaces. The idea is that the data lie close to an intrinsically low-dimensional nonlinear manifold embedded in a high-dimensional space. These methods can be thought of as "flattening" the manifold, and hence reducing the data to a set of low-dimensional coordinates that represent their relative positions in the manifold. They are useful for problems where signal-to-noise ratio is very high (e.g., physical systems), and are probably not as useful for observational data with lower signal-to-noise ratios.

We describe three new approaches to nonlinear dimension reduction and manifold mapping:

- Isometric feature mapping (ISOMAP): constructs a graph to approximate the geodesic distance between points along the manifold. Specifically, for each data point we find its neighbors—points within some small Euclidean distance of that point. We construct a graph with an edge between any two neighboring points. The geodesic distance between any two points is then approximated by the shortest path between points on the graph. Finally, classical scaling is applied to the graph distances, to produce a low-dimensional mapping.

- Local Linear embedding: takes a very different approach, trying to preserve the local affine structure of the high-dimensional data. Each data point is approximated by a linear combination of neighboring points. Then a lower dimensional representation is constructed that best preserves these local approximations.

- Local MDS: takes the simplest and arguably the most direct approach. We define N to be the symmetric set of nearby pairs of points; specifically a pair (i, i') is in N if point i is among the K-nearest neighbors of i, or vice-versa, and then construct the stress function. Local MDS minimizes the stress function over z_i, for fixed values of the number of neighbors K and the tuning parameter τ.

85 The Google PageRank Algorithm

We suppose that we have N web pages and wish to rank them in terms of importance. The PageRank algorithm considers a webpage to be important if many other webpages point to it. However the linking webpages that point to a given page are not treated equally: the algorithm also takes into account both the importance (PageRank) of the linking pages and the number of outgoing links that they have. Linking pages with higher PageRank are given more weight, while pages with more outgoing links are given less weight.

Let $L_{ij} = 1$ if page j points to page i, and zero otherwise. Let $c_j = \sum_{i=1}^{N} L_{ij}$ equal the number of pages pointed to by page j. Then the Google PageRanks p_i are defined by the recursive relationship

$$p_i = (1-d) + d\sum_{j=1}^{N}(\frac{L_{ij}}{c_j})p_j \tag{288}$$

where d is a positive constant. The idea is that the importance of page i is the sum of the importances of pages that point to that page.

Random Forests

86 Introduction

Bagging or *bootstrap aggregation* is a technique for reducing the variance of an estimated prediction function. Bagging seems to work especially well for high-variance, low-bias procedures, such as trees. For regression, we simply fit the same regression tree many times to bootstrap sampled version of the training data, and average the result. For classification, a *committee* of trees each cast a vote for the predicted class.

Random forests is a substantial modification of bagging that builds a large collection of *de-correlated* trees, and then averages them.

87 Definition of Random Forests

The essential idea in bagging is to average many noisy but approximately unbiased models, and hence reduce the variance. Trees are ideal candidates for bagging, since they can capture complex interaction structures in the data, and if grown sufficiently deep, have relatively low bias. Since trees are notoriously noisy, they benefit greatly from the averaging.

When growing a tree on a bootstrapped dataset: *Before each split, select $m \leq p$ of the input variables at random as candidates for splitting.*

Typical values for m are $\sqrt{p}$ or even as low as 1. After such trees are grown, the random forest (regression) predictor is

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b). \tag{289}$$

88 Details of Random Forests

The inventors of random forests make the following recommendations:

- For classification: the default value for m is $\sqrt{p}$ and the minimum node size is one.
- For regression: the default value for m is $\sqrt{p/3}$ and the minimum node size is five.

88.1 Out of the Bag Samples

For each observation $z_i = (x_i, y_i)$, construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i did not appear.

88.2 Variable Importance

Variable importance plots can be constructed for random forests, where at each split in the tree, the improvement in the split-criterion is the importance measure attributed to the splitting variable.

88.3 Proximity Plots

An $N \times N$ proximity matrix is accumulated for the training data. For every tree, any pair of out-of-bag (OOB) observations sharing a terminal node has their proximity increased by one, which is then represented in 2D using multidimensional scaling. The idea is that even though the data may be high dimensional, the proximity plot gives an indication of which observations are effectively close together in the eyes of the random forest classifier.

88.4 Random Forests and Overfitting

It is claimed that random forests *cannot overfit* the data. It is certainly true that increasing B does not cause the random forest sequence to overfit; like bagging, the random forest estimate approximates the expectation

$$\hat{f}_{rf}(x) = \mathrm{E}_{\Theta} T(x; \Theta) = \lim_{B \to \infty} \hat{f}(x)^B_{rf} \tag{290}$$

with an average over B realisations of Θ. The distribution of Θ here is conditional of the training data. However, *this limit can overfit the data*; the average of fully grown trees can result in too rich a model, and incur unnecessary variance.

89 Analysis of Random Forests

We analyse the mechanisms for random forests, focusing on regression and squarer error loss.

89.1 Variance and the De-Correlation Effect

The limiting form ($B \to \infty$) of the random forest regression estimator is

$$\hat{f}_{rf}(x) = \mathrm{E}_{\Theta|Z} T(x; \Theta(Z)), \tag{291}$$

where we have made explicit the dependence on the training data Z. Here we consider estimation at a single target point x. From

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{292}$$

we see that

$$\mathrm{Var}\hat{f}_{rf}(x) = \rho(x)\sigma^2(x). \tag{293}$$

Here

- $\rho(x)$ is the sampling correlation between any pair of trees used in the averaging:
$$\rho(x) = \text{corr}[T(x; \Theta_1(Z)), T(x; \Theta_2(Z))], \tag{294}$$
where $\Theta_1(Z)$ and $\Theta_2(Z)$ are randomly drawn pair of random forest trees grown to the randomly sampled Z.

- $\sigma^2(x)$ is the sampling variance of any single randomly drawn tree, $\sigma^2(x) = \text{Var}T(x; \Theta(Z))$.

89.2 Bias

As in bagging, the bias of a random forest is the same as the bias of any of the individual sampled trees $T(x; \Theta(Z))$:

$$\begin{aligned}\text{Bias}(x) &= \mu(x) - \text{E}_Z \hat{f}_{rf}(x) \\ &= \mu(x) - \text{E}_z \text{E}_{\Theta|z} T(x; \Theta(Z)). \end{aligned} \tag{295}$$

89.3 Adaptive Nearest Neighbours

The random forest classifier has much in common with the k-nearest neighbours classifier; in fact a weighted version thereof. Since each tree is grown to maximal size, for a particular $\Theta^*, T(x; \Theta^*(Z))$ is the response value for one of the training samples.

Ensemble Learning

90 Introduction

The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models. For example, bagging and random forests are ensemble methods for classification.

Ensemble learning can be broken down into two tasks: developing a population of base learners from the training data, and then combining them to form the composite predictor.

An early example of a learning ensemble is a method designed for multiclass classification using *error-correcting ouput codes*. Consider the 10-class digit classification problem, and the coding matrix $\boldsymbol{C}$. Note tha the ℓth column of the coding matrix C_ℓ defines a two-class variable that merges all the original classes into two groups. The method works as follows:

1. Learn a separate classifier for each of the $L = 15$ two class problems defined by the columns of the coding matrix.
2. At a test point x, let $\hat{p}_\ell(x)$ be the predicted probability of a one for the ℓth response.
3. Define $\delta_k(x) = \sum_{\ell=1}^{L} |C_{k\ell} - \hat{p}_\ell(x)|$, the discriminant function for the kth class, where $C_{k\ell}$ is the entry for row k and column ℓ.

91 Boosting and Regularisation Paths

91.1 Penalised Regression

Consider the dictionary of all possible J-terminal node regression trees $\mathcal{T} = \{T_k\}$ that could be realised on the training data as basis functions. The linear model is

$$f(x) = \sum_{k=1}^{K} \alpha_k T_k(x), \tag{296}$$

where $K = \text{card}(\mathcal{T})$. Suppose the coefficients are to be estimated via least-squares. Since the number of such trees is likely to be much larger than even the largest training data sets, some form of regularisation is required. This can be either the ridge regression $J(\alpha) = \sum_{k=1}^{K} |\alpha_k^2|$, or the lasso $J(\alpha) = \sum_{k=1}^{K} |\alpha_k|$.

91.2 The Bet on Sparsity Principle

The use of the L_1 penalty follows what we call the "bet on sparsity" principle for high-dimensional problems: *Use a procedure that does well in sparse problems, since no procedure does well in dense problems.*

These comments need some qualification:

- For any given application, the degree of sparseness/denseness depends on the unknown true target function, and the chosen dictionary $\mathcal{T}$.
- The notion of sparse versus dense is relative to the size of the training data set and/or the noise-to-signal ratio (NSR).
- The size of the dictionary plays a role. Increasing the size of the dictionary may lead to a sparser representation for our function, but the search problem becomes more difficult leading to higher variance.

92 Learning Ensembles

Consider the functions of the form

$$f(x) = \alpha_0 + \sum_{T_k \in \mathcal{T}} \alpha_k T_k(x), \tag{297}$$

where $\mathcal{T}$ is a dictionary of basis functions, typically trees. For gradient boosting and random forests, $|\mathcal{T}|$ is very large, and it is quite typical for the final model to involve many thousands of trees.

Friedman and Popescu (2003) propose a hybrid approach which breaks this process down into two stages:

- A finite dictionary $\mathcal{T}_L = \{T_1(x), T_2(x), \cdots, T_M(x)\}$ of basis functions is induced from the training data;
- A family of functions $f_\lambda(x)$ is built by fitting a lasso path in this dictionary:

$$\alpha(\lambda) = \arg\min_{\alpha} \sum_{i=1}^{N} L[y_i, \alpha_0 + \sum_{m=1}^{M} \alpha_m T_m(x_i)] + \lambda \sum_{m=1}^{M} |\alpha_m|. \tag{298}$$

92.1 Rule Ensemble

We describe a modification of the tree-ensemble method that focuses on individual rules. The idea is to enlarge an ensemble of trees by constructing a set of rules from each of the trees in the collection.

For each tree T_m in an ensemble $\mathcal{T}$, we can construct its mini-ensemble of rules $\mathcal{T}^m_{Rule}$, and then combine them all to form a larger ensemble

$$\mathcal{T}_{Rule} = \bigcup_{m=1}^{M} \mathcal{T}^m_{Rule}. \tag{299}$$

This is then treated like any other ensemble, and post-processed via the lasso or similar regularised procedure.

There are several advantages to this approach of deriving rules from the more complex trees:

- The space of models is enlarged, and can lead to improved performance.

- Rules are easier to interpret than trees, so simpler.
- It is often natural to augment $\mathcal{T}_{Rule}$ by including each variable X_j separately as well, thus allowing the ensemble to model linear functions well.

Undirected Graphical Models

93 Introduction

A graph consists of a set of vertices (nodes) along with a set of edges joining some pairs of the vertices. In graphical models, each vertex represents a random variable, and the graph gives a visual way of understanding the joint distribution of the entire set of random variables.

In an *undirected graph*, the edges have no directional arrows, meaning that the random variables are conditionally independent, given the other variables. Here we discuss these, also known as *Markov random fields* or *Markov networks*.

Sparse graphs have a relatively small number of edges, and are convenient for interpretation. They are useful in many domains, including genomics and proteomics, where they provide rough models of cell pathways.

The edges in a graph are parameterised by values or *potentials* that encode the strength of the conditional dependence between the random variables and the corresponding vertices. The main challenge with graphical models are model selection (choosing the structure of the graph), estimation of the edge parameters from data, and computation of marginal vertex probabilities and expectations from their joint distribution.

94 Markov Graphs and their properties

A graph $\mathcal{G}$ consists of a pair (V, E), where V is a set of vertices and E the set of edges (defined by pairs of vertices). Two vertices X and Y are called adjacent if there is a edge joining them; this is denoted by $X \sim Y$. A path $X_1, X_2, \cdots, X_n$ is a set of vertices that are joined. A complete graph is a graph with every pair of vertices joined by an edge. A subgraph $U \in V$ is a subset of vertices together with their edges. We express variables not connected as follows:

$$\text{No edge joining } X \text{ and } Y \Leftrightarrow X \perp Y|\text{rest} \tag{300}$$

where "rest" referes to all of the other vertices in the graph. If A, B, C are subgraphs, then C is said to be separate to A, B if every path between A and B intersects a node in C. Separators have the nice property that they break the graph into conditionally independent pieces, and are expressed as

$$\text{if } C \text{ separates } A \text{ and } B \text{ then } A \perp B|C. \tag{301}$$

A clique is a complete subgraph — a set of vertices that are all adjacent to one another. A probability density function f over a Markov graph $\mathcal{G}$ can be represented as

$$f(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C) \tag{302}$$

where $\mathcal{C}$ is the set of maximal cliques, and the positive functions $\psi_x(\cdot)$ are called clique potentials. The quantity

$$Z = \sum_{x \in \mathcal{X}} \prod_{C \in \mathcal{C}} \psi_C(x_C) \tag{303}$$

is the normalising constant, also known as the partition function.

95 Undirected Graphical Models for Continuous Variables

In the case where variables are all continuous, the Gaussian distribution is almost always used for graphical models because of its convenient analytical properties. We assume that the observations have a multivariate Gaussian distribution with mean μ and covariance matrix Σ. Since the Gaussian distribution represents at most second-order relationships, it automatically encodes a pairwise Markov graph. The Gaussian distribution has the property that all conditional distributions are also Gaussian. The inverse covariance matrix Σ^{-1} contains information about the partial covariances between the variables; that is, the covariances between pairs i and j, conditioned on all other variables.

The conditional distribution of Y given Z is

$$Y|Z = z \sim N(\mu_Y + (z - \mu_z)^\top \Sigma_{ZZ}^{-1} \sigma_{ZY}, \sigma_{YY} - \sigma_{ZY}^\top \Sigma_{ZZ}^{-1} \sigma_{ZY}). \tag{304}$$

Here, the regression coefficient is

$$\beta = \Sigma_{ZZ}^{-1} \sigma_{ZY}. \tag{305}$$

The dependence of Y on Z is in the mean term alone. Moreoever, we can learn about this dependence structure through multiple linear regression.

95.1 Estimation of the Parameters when the Graph Structure is Known

Given some realisations of X, we would like to estimate the parameters of an undirected graph that approximates their joint distribution. Suppose first that the graph is complete (fully connected). We assume that we have N multivariate normal realisations with population mean μ and covariance Σ. Let

$$\boldsymbol{S} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^\top \tag{306}$$

be the empirical covariance matrix, with $\bar{x}$ the sample mean vector. The log-likelihood of the data can be written as

$$\ell(\Theta) = \log \det \Theta - \operatorname{trace}(\boldsymbol{S}\Theta). \tag{307}$$

Now lets assume that some of the edges are missing. For the Gaussian distribution this implies that the corresponding entries of $\Theta = \Sigma^{-1}$ are zero. Suppose that we want to estimate the edge parameters θ_{ij} for the vertices that are joined to a given vertex i, restricting those that are not joined to zero. It turns out that if we use our current (model-based) estimate of the cross-product matrix of the predictors when we perform our regressions, this gives the correct solutions and solves the constrained maximum-likelihood problem exactly. To constrain the log-likelihood, we add Lagrange constats for all missing edges

$$\ell_C(\Theta) = \log\det\Theta - \mathrm{trace}(\Theta) - \sum_{(j,k)\notin E} \gamma_{jk}\theta_{jk}. \tag{308}$$

The gradient equation for maximising this is

$$\Theta^{-1} - \boldsymbol{S} - \boldsymbol{\Gamma} = 0, \tag{309}$$

where $\boldsymbol{\Gamma}$ is a matrix of Lagrange parameters with nonzero values for all pairs with edges absent.

95.2 Estimation of the Graph Structure

In most cases we do not know which edges to omit from our graph, and so would like to try to discover this from the data itself. In recent years a number of authors have proposed the use of $L1$ (lasso) regularization for this purpose. In this case, the modified log-likelihood equation is

$$\ell(\Theta) = \log\det\Theta - \mathrm{trace}(\boldsymbol{S}\Theta) - \lambda||\Theta||_1, \tag{310}$$

and the gradient equation is

$$\Theta^{-1} - \boldsymbol{S} - \lambda\cdot\mathrm{Sign}(\Theta) = 0. \tag{311}$$

96 Undirected Graphical Models for Discrete Variables

Undirected Markov networks with all discrete variables are popular, and in particular pairwise Markov networks with binary variables being the most common. They are sometimes called Ising models in the statistical mechanics literature, and Boltzmann machines in the machine learning literature, where the vertices are referred to as "nodes" or "units" and are binary-valued.

Denoting the binary values variable at node j by X_j, the Ising model for the joint probabilities is given by

$$p(X, \Theta) = \exp[\sum_{(j,k)\in E} \theta_{jk} X_j X_k - \phi(\Theta)] \text{ for } X \in \mathcal{X}. \tag{312}$$

As with the Gaussian model of the previous section, only pairwise interactions are modeled. The Ising model was developed in statistical mechanics, and is now used more generally to model the joint effects of pairwise interactions. $\phi(\Theta)$ is the log of the partition function, and is defined by

$$\phi(\Theta) = \log \sum_{x \in \mathcal{X}} [\exp(\sum_{(j,k) \in E} \theta_{jk} x_j x_k)]. \tag{313}$$

The partition function ensures that the probabilities add to one over the sample space.

96.1 Estimation of the Parameters when the Graph Structure is Known

Suppose we have observations $x_i = (x_{i1}, x_{i2}, \cdots, x_{ip}) \in \{0,1\}^p, i = 1, \cdots, N$. The log-likelihood is

$$\ell(\Theta) = \sum_{i=1}^{N} \log \Pr_{\Theta}(X_i = x_i) = \sum_{i=1}^{N} \left[\sum_{(j,k) \in E} \theta_{jk} x_{ij} x_{ik} - \phi(\Theta) \right]. \tag{314}$$

To find the maximum likelihood estimates, we can use gradient search or Newton methods using the gradient of the log-likelihood function w.r.t. parameters in the model. For smaller p, a number of standard statistical approaches are available: Poisson log-linear modelling, Gradient descent, and iterative proportional fitting. When p is large (> 30) other approaches have been used to approximate the gradient: mean field approximations and Gibbs sampling.