

Data Mining & Statistical Learning

(2019 Spring)

Wonkuk Kim

Department of Applied Statistics

Chung-Ang University

84 Heukseok-ro, Dongjak-gu, Seoul 06974, Korea

Contents

I	Introduction	1
1	Introduction	3
1.1	What is data mining and statistical learning?	4
1.2	Notation	6
1.3	Supervised or unsupervised	7
II	Supervised Learning	9
2	Supervised Learning	11
2.1	Regression	12
2.1.1	Systematic information and random error terms in a regression model	12
2.1.2	Purpose of estimating f	13
2.1.3	Methods of estimating f : How to estimate f ?	14
2.1.4	The Trade-Off Between Prediction Accuracy and Model Interpretability	20

2.1.5	Assessing Model Accuracy	22
2.1.6	Bias-Variance trade-off	24
2.2	Classification	28
3	Resampling Methods	33
3.1	Introduction	33
3.2	Permutation tests	34
3.3	Jackknife methods	37
3.4	Nonparametric Bootstrap	40
3.4.1	Bootstrap variance estimation (Portfolio data)	48
3.5	Parametric Bootstrap	53
4	Training set, validation set, and test set	57
4.1	Cross-validation	57
4.1.1	Training set, validation set, and test set	57
4.1.2	Validation set (hold-out) approach	59
4.1.3	Leave-One-Out Cross-validation (LOOCV)	65
4.1.4	k -fold Cross-Validation (k -fold CV)	70
4.1.5	Bias-Variance Trade-Off for k -Fold Cross-Validation	77
4.1.6	Cross-Validation on Classification Problems	78
4.1.7	Example	79
III	Supervised learning: Regression	83
5	Linear Model Selection and Regularization	85
5.1	Introduction	85

5.1.1	Overview of linear model selection	85
5.2	Subset selection	86
5.2.1	Best subset selection	86
5.2.2	Forward stepwise selection	93
5.2.3	Backward stepwise selection	94
5.2.4	Choosing the optimal model	101
5.3	Shrinkage Methods	106
5.3.1	Ridge regression	106
5.3.2	Least Absolute Shrinkage and Selection Operator (LASSO)	112
5.4	Dimension reduction methods	116
5.4.1	Outline of dimension reduction methods	116
5.4.2	Principal Components Regression (PCR)	117
5.4.3	Partial Least Squares	117
6	Moving beyond linearity	125
6.1	Introduction	125
6.2	Polynomial regression	126
6.3	Step functions	127
6.3.1	Basis function approach	127
6.4	Regression splines	136
6.4.1	Piecewise Polynomials	136
6.4.2	Constraints and Splines	136
6.4.3	The Spline Basis Representation	138
6.4.4	Choosing the Number and Locations of the Knots	140
6.4.5	Comparison to Polynomial Regression	143
6.5	Smoothing splines	144
6.5.1	An Overview of Smoothing Splines	144

6.5.2	Choosing the Smoothing Parameter λ	144
6.5.3	Local regression	147
6.6	Generalized Additive Model (GAM)	151
6.6.1	GAMs for Regression Problems	151
6.6.2	GAMs for Classification Problems	152
IV	Supervised learning: Classification	161
7	Classification	163
7.1	Examples	163
7.2	Classification analysis	169
7.3	Bayes classifier	172
7.4	Naive Bayes classifier	188
7.5	LDA and QDA: Classification for multivariate normal distributions	194
7.6	Logistics regression	201
7.6.1	Binary response and simple logistic regression	201
7.6.2	Multiple logistic regression	210
7.7	K Nearest Neighborhood (KNN)	216
7.8	Classification Trees	222
8	Ensemble Methods	239
8.1	Bootstrap aggregation (Bagging)	240
8.2	Random Forest	250
8.3	Boosting	258
9	Summarization of a classifier	269

9.1 Confusion matrix	270
9.1.1 Cohen's kappa κ	274
9.1.2 F_1 score	275
9.2 Receiver Operating Characteristic (ROC) curve analysis	276
10 Neural Network	289
10.1 Feed-forward neural network	290
10.2 Estimation	294
10.3 Examples with R	300
11 Support Vector Machine	327
11.1 Separating Hyperplane	328
11.2 Maximal margin classifier (Separable case)	331
11.3 Support vector classifier (Non-separable case)	335
11.4 Support vector machines (Nonlinear cases)	342
11.5 Kernel Trick	346
11.6 SVMs with More than Two Classes	348
11.7 Examples	349
V Unsupervised Learning	367
12 Clustering	369
12.1 Similarity or Dissimilarity	371
12.2 Hierarchical Clustering	375
12.3 Partitioning	388
12.3.1 K -means clustering	388

12.4 Determining The Optimal Number Of Clusters	398
12.4.1 Elbow method	399
12.4.2 Average silhouette method	400
12.4.3 Gap statistic method	405
12.5 Mixture Models	411
12.6 Examples	422
13 Principal Component Analysis (PCA)	433
13.1 What are principal components?	434
13.2 Examples	436
14 Association Rules	449
14.1 Association rule mining	450
14.2 Evaluation of Association Patterns	459
15 Text Mining	465
15.1 Text Mining Procedure	465
15.2 Examples	467

List of Figures

2.1	Fitted curves of polynomial regression models	17
2.2	knn regression with $k = 10$	19
2.3	A representation of the tradeoff between flexibility and interpretability, using different statistical learning methods. In general, as the flexibility of a method increases, its interpretability decreases.	20
2.4	Left: Data simulated from f , shown in black. Three estimates of f are shown: the linear regression line (orange curve), and two smoothing spline fits (blue and green curves). Right: Training MSE (grey curve), test MSE (red curve), and minimum possible test MSE over all methods (dashed line). Squares represent the training and test MSEs for the three fits shown in the left-hand panel.	24
2.5	Squared bias (blue curve), variance (orange curve), $\text{Var}(\epsilon)$ (dashed line), and test MSE (red curve) for the three data sets. The vertical dotted line indicates the flexibility level corresponding to the smallest test MSE.	27
2.6	Training and test classification errors for simulated data	32
3.1	Bootstrap (resampling from data with replacement)	43
3.2	Bootstrap estimation	50

4.1	A schematic display of the validation set approach. A set of n observations are randomly split into a training set (shown in blue, containing observations 7, 22, and 13, among others) and a validation set (shown in beige, and containing observation 91, among others). The statistical learning method is fit on the training set, and its performance is evaluated on the validation set.	61
4.2	The validation set approach was used on the Auto data set in order to estimate the test error that results from predicting mpg using polynomial functions of horsepower. Left: Validation error estimates for a single split into training and validation data sets. Right: The validation method was repeated ten times, each time using a different random split of the observations into a training set and a validation set. This illustrates the variability in the estimated test MSE that results from this approach.	62
4.3	Validation set approach was applied to Auto data for polynomial regression to estimate the test MSE.	64
4.4	A schematic display of LOOCV. A set of n data points is repeatedly split into a training set (shown in blue) containing all but one observation, and a validation set that contains only that observation (shown in beige). The test error is then estimated by averaging the n resulting MSE's. The first training set contains all but observation 1, the second training set contains all but observation 2, and so forth.	65
4.5	LOOCV of polynomial regression models for Auto data	69
4.6	A schematic display of 5-fold CV. A set of n observations is randomly split into five non-overlapping groups. Each of these fifths acts as a validation set (shown in beige), and the remainder as a training set (shown in blue). The test error is estimated by averaging the five resulting MSE estimates.	71
4.7	k-fold CV of polynomial regression models for Auto data. Left: LOOCV, Right: 10-fold CV with 10 distinct seeds.	74
4.8	True and estimated test MSE for the simulated data sets. The true test MSE is shown in blue, the LOOCV estimate is shown as a black dashed line, and the 10-fold CV estimate is shown in orange. The crosses indicate the minimum of each of the MSE curves.	75
4.9	Logistic regression fits on the two-dimensional classification data displayed. The Bayes decision boundary: purple dashed line. Estimated decision boundaries from linear, quadratic, cubic and quartic (degrees 1-4) logistic regressions: black. The test error rates for the four logistic regression fits are respectively 0.201, 0.197, 0.160, and 0.162, while the Bayes error rate is 0.133.	80

4.10 Test error (brown), training error (blue), and 10-fold CV error (black) on the two-dimensional classification data displayed. Left: Logistic regression using polynomial functions of the predictors. The order of the polynomials used is displayed on the x-axis. Right: The KNN classifier with different values of K, the number of neighbors used in the KNN classifier.	81
5.1 RSS and adjusted R^2	91
5.2 C_p and adjusted BIC	92
5.3 Validation error and k-fold CV	105
5.4 Contours of the error and constraint functions for the lasso (left) and ridge regression (right).	113
5.5 Validation Plots for PCR and PLS	123
6.1 The Wage data. Left: The solid blue curve is a degree-4 polynomial of wage (in thousands of dollars) as a function of age, fit by least squares. The dotted curves indicate an estimated 95 % confidence interval. Right: We model the binary event wage>250 using logistic regression, again with a degree-4 polynomial. The fitted posterior probability of wage exceeding \$250,000 is shown in blue, along with an estimated 95 % confidence interval.	134
6.2 The Wage data. Left: The solid curve displays the fitted value from a least squares regression of wage (in thousands of dollars) using step functions of age. The dotted curves indicate an estimated 95 % confidence interval. Right: We model the binary event wage>250 using logistic regression, again using step functions of age. The fitted posterior probability of wage exceeding \$250,000 is shown, along with an estimated 95 % confidence interval.	135
6.3 Various piecewise polynomials are fit to a subset of the Wage data, with a knot at age=50. Top Left: The cubic polynomials are unconstrained. Top Right: The cubic polynomials are constrained to be continuous at age=50. Bottom Left: The cubic polynomials are constrained to be continuous, and to have continuous first and second derivatives. Bottom Right: A linear spline is shown, which is constrained to be continuous.	137
6.4 A cubic spline and a natural cubic spline, with three knots, fit to a subset of the Wage data.	139
6.5 A natural cubic spline function with four degrees of freedom is fit to the Wage data. Left: A spline is fit to wage (in thousands of dollars) as a function of age. Right: Logistic regression is used to model the binary event wage>250 as a function of age. The fitted posterior probability of wage exceeding \$250,000 is shown.	141

6.6	Ten-fold cross-validated mean squared errors for selecting the degrees of freedom when fitting splines to the Wage data. The response is wage and the predictor age. Left: A natural cubic spline. Right: A cubic spline.	142
6.7	On the Wage data set, a natural cubic spline with 15 degrees of freedom is compared to a degree-15 polynomial. Polynomials can show wild behavior, especially near the tails.	143
6.8	Smoothing spline fits to the Wage data. The red curve results from specifying 16 effective degrees of freedom. For the blue curve, λ was found automatically by leave-one-out cross-validation, which resulted in 6.8 effective degrees of freedom.	145
6.9	Local regression illustrated on some simulated data, where the blue curve represents $f(x)$ from which the data were generated, and the light orange curve corresponds to the local regression estimate $\hat{f}(x)$. The orange colored points are local to the target point x_0 , represented by the orange vertical line. The yellow bell-shape superimposed on the plot indicates weights assigned to each point, decreasing to zero with distance from the target point. The fit $\hat{f}(x_0)$ at x_0 is obtained by fitting a weighted linear regression (orange line segment), and using the fitted value at x_0 (orange solid dot) as the estimate $\hat{f}(x_0)$	148
6.10	Local linear fits to the Wage data. The span specifies the fraction of the data used to compute the fit at each target point.	149
6.11	For the Wage data, plots of the relationship between each feature and the response, wage, in the fitted model. Each plot displays the fitted function and pointwise standard errors. The first two functions are natural splines in year and age, with four and five degrees of freedom, respectively. The third function is a step function, fit to the qualitative variable education.	157
6.12	Details are as in Figure 6.11, but now f_1 and f_2 are smoothing splines with four and five degrees of freedom, respectively.	158
6.13	For the Wage data, the logistic regression GAM is fit to the binary response $I(\text{wage} > 250)$. Each plot displays the fitted function and pointwise standard errors. The first function is linear in year, the second function a smoothing spline with five degrees of freedom in age, and the third a step function for education. There are very wide standard errors for the first level < HS of education.	159
7.1	Handwritten digits	164
7.2	Color image is represented by RGB matrices	165
7.3	Speech data	166
7.4	Bayes Error Rate is equal to the sum of skyblue and orange areas. Note the area under blue curve plus the area under red curve equal one.	186

7.5	Scatter plot of iris data	200
7.6	Scatter plot of iris data with a logistic fitted curve.	204
7.7	The KNN approach, using $K = 3$, is illustrated in a simple situation with six blue observations and six orange observations. Left: a test observation at which a predicted class label is desired is shown as a black cross. The three closest points to the test observation are identified, and it is predicted that the test observation belongs to the most commonly-occurring class, in this case blue. Right: The KNN decision boundary for this example is shown in black. The blue grid indicates the region in which a test observation will be assigned to the blue class, and the orange grid indicates the region in which it will be assigned to the orange class.	217
7.8	A comparison of the KNN decision boundaries (solid black curves) obtained using $K = 1$ and $K = 100$ on the data from a simulated dataset. With $K = 1$, the decision boundary is overly flexible, while with $K = 100$ it is not sufficiently flexible. The Bayes decision boundary is shown as a purple dashed line.	218
7.9	The KNN training error rate (blue, 200 observations) and test error rate (orange, 5,000 observations) on the data from Figure 2.13, as the level of flexibility (assessed using $1/K$) increases, or equivalently as the number of neighbors K decreases. The black dashed line indicates the Bayes error rate. The jumpiness of the curves is due to the small size of the training data set.	219
7.10	Classification errors of knn for iris data. Left: hold-out approach, Right: LOOCV.	221
7.11	Recursive binary split of $R = R_1 \cup R_2$	223
7.12	Left: Recursive binary split, Right: classification tree	225
7.13	The unpruned classification tree for Heart data	229
7.14	(Left) Plot of the deviance versus tree size and (Right) Pruned tree	232
7.15	Tree constructed by rpart	235
7.16	Tree constructed by “rpart” with “partykit” plot	236
8.1	Bagging and random forest results for the Heart data. The test error (black and orange) is shown as a function of B , the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.	243

8.2 A variable importance plot for the Heart data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.	249
8.3 Results from random forests for the 15–class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m , the number of predictors available for splitting at each interior tree node. Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7 %.	251
8.4 Concept of Adaboost: From Schapire, R. E. and Freund, Y. (2012). Boosting: Foundations and Algorithms. MIT Press.	259
9.1 ROC space	277
9.2 ROC curve	278
9.3 ROC best cut-off	280
9.4 AUC	282
9.5 ROC curve	288
10.1 Processing Units of Neural Networks	290
10.2 Activation functions	293
10.3 Backpropagation	296
10.4 A worked example of the backpropagation	298
10.5 The results of the neural network model	305
10.6 The results of the neural network model	311
10.7 History plot of training a deep neural network to iris data	315
11.1 The hyperplane $1 + 2X_1 + 3X_2 = 0$ is shown. The blue region is the set of points for which $1 + 2X_1 + 3X_2 > 0$, and the purple region is the set of points for which $1 + 2X_1 + 3X_2 < 0$	329

11.2 Left: There are two classes of observations, shown in blue and in purple, each of which has measurements on two variables. Three separating hyperplanes, out of many possible, are shown in black. Right: A separating hyperplane is shown in black. The blue and purple grid indicates the decision rule made by a classifier based on this separating hyperplane: a test observation that falls in the blue portion of the grid will be assigned to the blue class, and a test observation that falls into the purple portion of the grid will be assigned to the purple class.	331
11.3 There are two classes of observations, shown in blue and in purple. The maximal margin hyperplane is shown as a solid line. The margin is the distance from the solid line to either of the dashed lines. The two blue points and the purple point that lie on the dashed lines are the support vectors, and the distance from those points to the margin is indicated by arrows. The purple and blue grid indicates the decision rule made by a classifier based on this separating hyperplane.	332
11.4 There are two classes of observations, shown in blue and in purple. In this case, the two classes are not separable by a hyperplane, and so the maximal margin classifier cannot be used.	336
11.5 Left: Two classes of observations are shown in blue and in purple, along with the maximal margin hyperplane. Right: An additional blue observation has been added, leading to a dramatic shift in the maximal margin hyperplane shown as a solid line. The dashed line indicates the maximal margin hyperplane that was obtained in the absence of this additional point.	337
11.6 Left: A support vector classifier was fit to a small data set. The hyperplane is shown as a solid line and the margins are shown as dashed lines. Purple observations: Observations 3,4,5, and 6 are on the correct side of the margin, observation 2 is on the margin, and observation 1 is on the wrong side of the margin. Blue observations: Observations 7 and 10 are on the correct side of the margin, observation 9 is on the margin, and observation 8 is on the wrong side of the margin. No observations are on the wrong side of the hyperplane. Right: Same as left panel with two additional points, 11 and 12. These two observations are on the wrong side of the hyperplane and the wrong side of the margin.	340
11.7 A support vector classifier was fit using four different values of the tuning parameter C in (9.12)-(9.15). The largest value of C was used in the top left panel, and smaller values were used in the top right, bottom left, and bottom right panels. When C is large, then there is a high tolerance for observations being on the wrong side of the margin, and so the margin will be large. As C decreases, the tolerance for observations being on the wrong side of the margin decreases, and the margin narrows.	341

11.8 Left: The observations fall into two classes, with a non-linear boundary between them. Right: The support vector classifier seeks a linear boundary, and consequently performs very poorly.	342
11.9 Left: An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule. Right: An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.	345
11.10SVM example	356
11.11SVM example	357
12.1 Intercluster distance (dissimilarity) for (a) single linkage, (b) complete linkage, and (c) average linkage.	375
12.2 Dendrogram of hierarchical clustering of USArests data with Manhattan distance and complete linkage. Left: The number of clusters is set as 4, Right: Scatter plot matrix with cluster groups	387
12.3 k-means clustering for USArests data: scree plot for $k = 1, \dots, 20$	395
12.4 k-means clustering for USArests data: scatter plot matrix with cluster groups.	396
12.5 k-means clustering for USArests data: Cluster groups in two PCs.	397
12.6 Dendrogram of the hierarchical clustering with average linkage	404
12.7 Results of elbow, silhouette, and gap methods to USArests data	407
12.8 Results of Nbclust()	410
12.9 Scatter Plot Matrix of the Results of Mixture Model-Based Clustering	421
12.10The NCI60 cancer cell line microarray data, clustered with complete linkage, and using Euclidean distance as the dissimilarity measure.	429
12.11The NCI60 cancer cell line microarray data, clustered with average linkage, and using Euclidean distance as the dissimilarity measure.	430
12.12The NCI60 cancer cell line microarray data, clustered with single linkage, and using Euclidean distance as the dissimilarity measure.	431
12.13The NCI60 cancer cell line microarray data, clustered with complete linkage, and using Euclidean distance as the dissimilarity measure and first five PCs.	432

13.1 Biplot for USArrests data. The first two principal components for the USArrests data. The blue state names represent the scores for the first two principal components. The orange arrows indicate the first two principal component loading vectors (with axes on the top and right). For example, the loading for Rape on the first component is 0.54, and its loading on the second principal component 0.17 (the word Rape is centered at the point (0.54, 0.17)). This figure is known as a biplot, because it displays both the principal component scores and the principal component loadings.	440
13.2 Left: a scree plot depicting the proportion of variance explained by each of the four principal components in the USArrests data. Right: the cumulative proportion of variance explained by the four principal components in the USArrests data.	441
13.3 Projections of the NCI60 cancer cell lines onto the first three principal components (in other words, the scores for the first three principal components). On the whole, observations belonging to a single cancer type tend to lie near each other in this low-dimensional space. It would not have been possible to visualize the data without using a dimension reduction method such as PCA, since based on the full data set there are $\binom{6,830}{2}$ possible scatterplots, none of which would have been particularly informative.	446
13.4 The PVE of the principal components of the NCI60 cancer cell line microarray data set. Left: the PVE of each principal component is shown. Right: the cumulative PVE of the principal components is shown. Together, all principal components explain 100 % of the variance.	447
14.1 Apriori principle: if $\{c,d,e\}$ is a frequent itemset, then all subsets of it are frequent.	454
14.2 Infrequent Itemsets: An illustration of support-based pruning. If $\{a,b\}$ is infrequent, then all super sets of $\{a,b\}$ are infrequent.	455
14.3 Illustration of frequent itemset generation using Apriori algorithm	456
14.4 Pruning of association rules using the confidence measure by anti-monotonicity.	458
14.5 Association rule mining example of Groceries data	463
14.6 Association rule mining example of Groceries data	464
15.1 Barplot and Wordcloud of the Results of Text Mining from Wikipedia	473

List of Tables

3.1	All possible permutations for $N = 5$ and $n_1 = 2$	35
3.2	Null distribution of the Wilcoxon rank sum test and Mann-Whitney U test for $N = 5$ and $n_1 = 2$	36
3.3	Bootstrap Samples	44
7.1	Discretized sepal length and sepal width	179
7.2	Discretized sepal length and sepal width	191
9.1	Confusion Matrix: Possible results from a binary classifier	270
9.2	Confusion Matrix: Possible results from a binary classifier	276
12.1	Similarity Coefficients for Clustering Items	373
12.2	Parameterizations of the covariance matrix Σ_k currently available in mclust for hierarchical clustering (HC) and/or EM for multidimensional data, where λ_k is a scalar, \mathbf{A}_k is a diagonal matrix whose elements are proportional to the eigenvalues of Σ_k , and \mathbf{D}_k is an orthogonal matrix.	412
14.1	Market-Basket transactions	450
14.2	Possible rules generated from {Milk,Diaper,Beer}	457
14.3	Beverage preferences among a group of 1000 people.	459

14.4 Summary of association rule mining for Groceries data	462
--	-----

Part I

Introduction

Chapter 1 Introduction

1.1 What is data mining and statistical learning?

Data Mining from Wikipedia:

Data mining is the computational process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics, and database systems. The overall goal of the data mining process is **to extract information from a data set and transform it into an understandable structure for further use**. Aside from the raw analysis step, it involves database and data management aspects, data pre-processing, model and inference considerations, interestingness metrics, complexity considerations, post-processing of discovered structures, visualization, and online updating. **Data mining is the analysis step of the “knowledge discovery in databases (KDD)” process.**

Data mining involves six common classes of tasks:

1. **Anomaly detection (outlier/change/deviation detection)** – The identification of unusual data records, that might be interesting or data errors that require further investigation.
2. **Association rule learning (dependency modeling)** – Searches for relationships between variables. For example, a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.
3. **Clustering** – is the task of discovering groups and structures in the data that are in some way or another “similar”, without using known structures in the data.
4. **Classification** – is the task of generalizing known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as “legitimate” or as “spam”.
5. **Regression** – attempts to find a function which models the data with the least error that is, for estimating the relationships among data or datasets.
6. **Summarization** – providing a more compact representation of the data set, including visualization and report generation.

1.2 Notation

- n = the number of distinct data points, observations.
- p = the number of variables
- $\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}$ is a $n \times p$ matrix.
- $\mathbf{x}_i = \begin{pmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{pmatrix}$ is a vector of length p , containing the p measurements for the i th observation.
- $\mathbf{x}_j = \begin{pmatrix} x_{1j} \\ \vdots \\ x_{nj} \end{pmatrix}$ is a vector of length n , containing the n measurements for the j th variable.
- Therefore, $\mathbf{X} = (\mathbf{x}_1 \cdots \mathbf{x}_p)$

1.3 Supervised or unsupervised

- Statistical learning or data mining
 - Refers to a vast set of tools for understanding data.
 - Classified as supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.
- Supervised learning
 - Building a statistical model for predicting, or estimating, an output based on one or more inputs.
 - Occuring in fields as diverse as business, medicine, astrophysics, and public policy.
 - Regression or classification problem.
- Unsupervised learning
 - There are inputs but no supervising output.
 - We can learn relationships and structure from such data.

Part II

Supervised Learning

Chapter 2 Supervised Learning

Input and output: Explanatory or Response?

- **Input:** predictors, covariates, explanatory variables, independent variables, features, inputs, or sometimes just variables, denoted by \mathbf{X} .
- **Output:** response variables, dependent variable, or outputs, denoted by Y .

Regression or classification?

- **Regression:** A response variable is quantitative.
- **Classification:** A response variable is categorical or qualitative.

2.1 Regression

2.1.1 Systematic information and random error terms in a regression model

- Suppose that we observe a quantitative response Y and p different predictors, X_1, X_2, \dots, X_p .
- We assume that there is some relationship between Y and $\mathbf{X} = (X_1, X_2, \dots, X_p)'$:

$$Y = f(\mathbf{X}) + \epsilon \tag{2.1}$$

where f is a fixed and unknown function.

- **Systematic information:** f
- **Random error term:** ϵ

2.1.2 Purpose of estimating f

Why estimate f ? Prediction or/and inference

- **Prediction:** In many situations, a set of inputs \mathbf{X} are readily available, but the output Y cannot be easily obtained.
 - Let \hat{f} be an estimate for f .
 - We predict Y by using

$$\hat{Y} = \hat{f}(\mathbf{X}) \quad (2.2)$$

- When the purpose is prediction only, \hat{f} is treated as a blackbox, that is, we are not concerned with the exact form of \hat{f} .
- (Accuracy of prediction) For fixed \hat{f} and \mathbf{X} ,

$$E[(Y - \hat{Y})^2] = E[(f(\mathbf{X}) + \epsilon - \hat{f}(\mathbf{X}))^2] = \underbrace{(f(\mathbf{X}) - \hat{f}(\mathbf{X}))^2}_{\text{Reducible error}} + \underbrace{\text{Var}(\epsilon)}_{\text{Irreducible error}} \quad (2.3)$$

- We want to learn statistical techniques for estimating f with the aim of minimizing the reducible error.
- **Inference:** We want to understand the relationship between \mathbf{X} and Y , or more specifically, to understand how Y changes as a function of X_1, \dots, X_p .
 - Which predictors are associated with the response?
 - What is the relationship between the response and each predictor?
 - Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

2.1.3 Methods of estimating f : How to estimate f ?

Parametric methods: • Make an assumption about a parametric functional form, or shape, of f that includes parameters β_0, \dots, β_p .

- Fit or train the model, that is, estimate the parameters β_0, \dots, β_p .
- Using a parametric method, estimating f is reduced to estimating the parameters β_0, \dots, β_p .
- Potential disadvantage:
 1. The chosen model may be far from the true model \Rightarrow poor conclusions (prediction or/and interpretation)
 2. Adding more flexibility (many parameters) \Rightarrow Overfit the data.
- (Example) Linear regression model, GAM, etc.

Example 2.1.1

(Linear regression: polynomial regression)

```
library(ISLR)
fit <- lm(mpg ~ horsepower, data = Auto)
summary(fit)

##
## Call:
## lm(formula = mpg ~ horsepower, data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5710  -3.2592  -0.3435   2.7630  16.9240
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|) 
## (Intercept) 39.935861   0.717499  55.66 <2e-16 ***
## horsepower -0.157845   0.006446 -24.49 <2e-16 ***
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 4.906 on 390 degrees of freedom
## Multiple R-squared:  0.6059, Adjusted R-squared:  0.6049 
## F-statistic: 599.7 on 1 and 390 DF,  p-value: < 2.2e-16

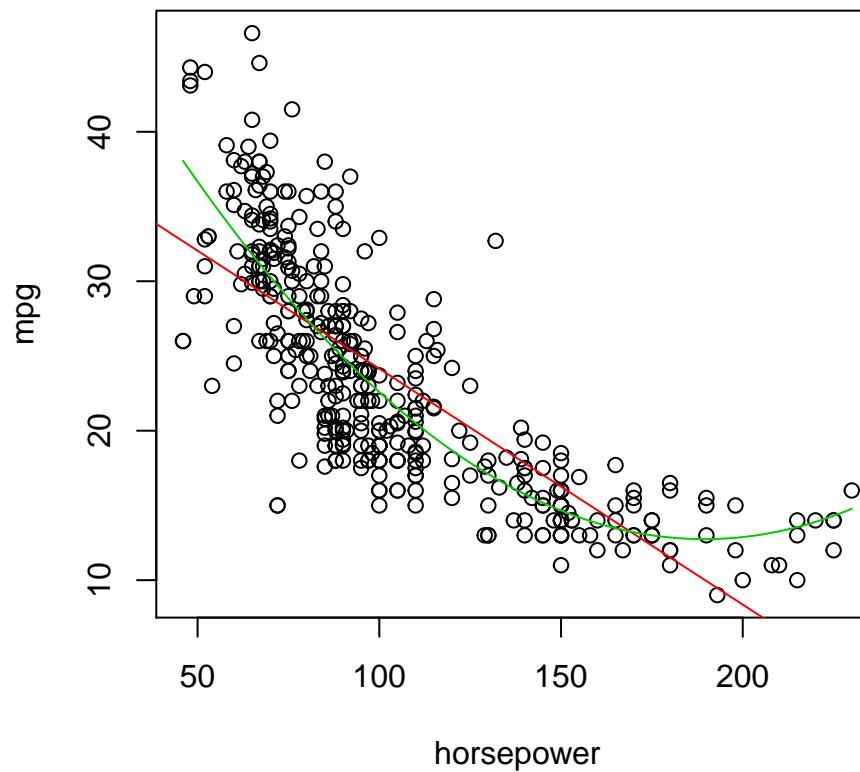
fit2 <- lm(mpg ~ poly(horsepower, 2, raw = T), data = Auto)
summary(fit2)

## 
## Call:
## lm(formula = mpg ~ poly(horsepower, 2, raw = T), data = Auto)
## 
## Residuals:
##      Min       1Q     Median       3Q      Max 
## -14.7135  -2.5943  -0.0859   2.2868  15.8961 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|) 
## (Intercept) 56.9000997  1.8004268  31.60 <2e-16 ***
## poly(horsepower, 2, raw = T)1 -0.4661896  0.0311246 -14.98 <2e-16 ***
## poly(horsepower, 2, raw = T)2  0.0012305  0.0001221   10.08 <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.374 on 389 degrees of freedom
## Multiple R-squared:  0.6876, Adjusted R-squared:  0.686
## F-statistic:  428 on 2 and 389 DF,  p-value: < 2.2e-16

# with(Auto,plot(horsepower,mpg)) abline(fit,col=2)
# curve(coef(fit2)[1]+coef(fit2)[2]*x+coef(fit2)[3]*x^2,add=T,col=3)
```

Figure 2.1: Fitted curves of polynomial regression models



Nonparametric methods: • We do not make explicit assumptions about the functional form of f .

- Seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly.
- A very large number of observations (far more than is typically needed for a parametric approach) is required in order to obtain an accurate estimate for f .
- (Example) kNN regression, Random Forest, etc.

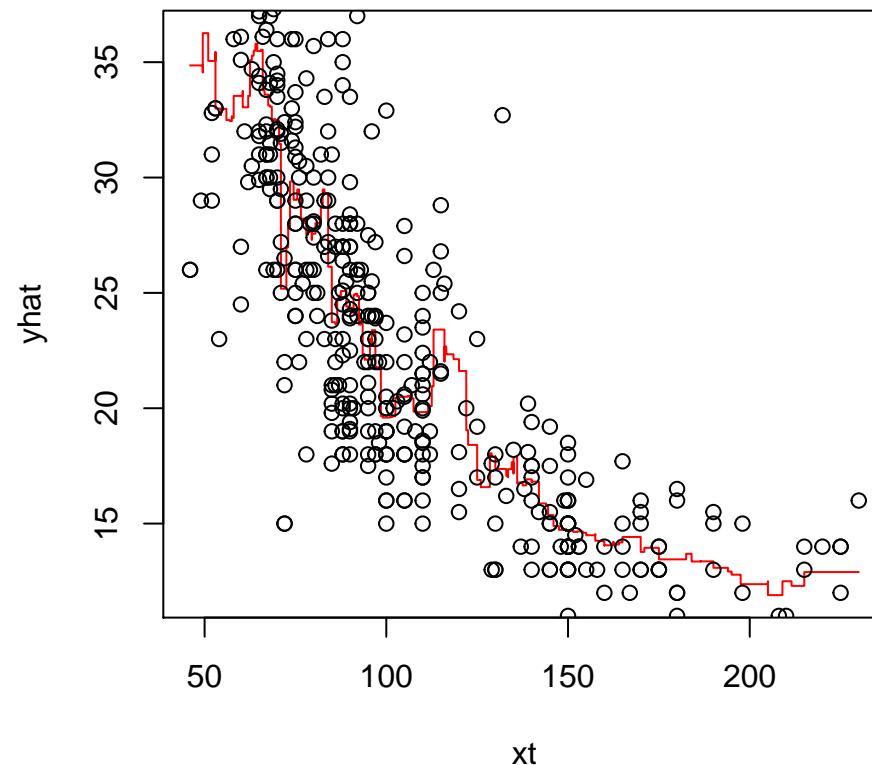
Example 2.1.2

(kNN regression)

```
library(ISLR)
library(caret)

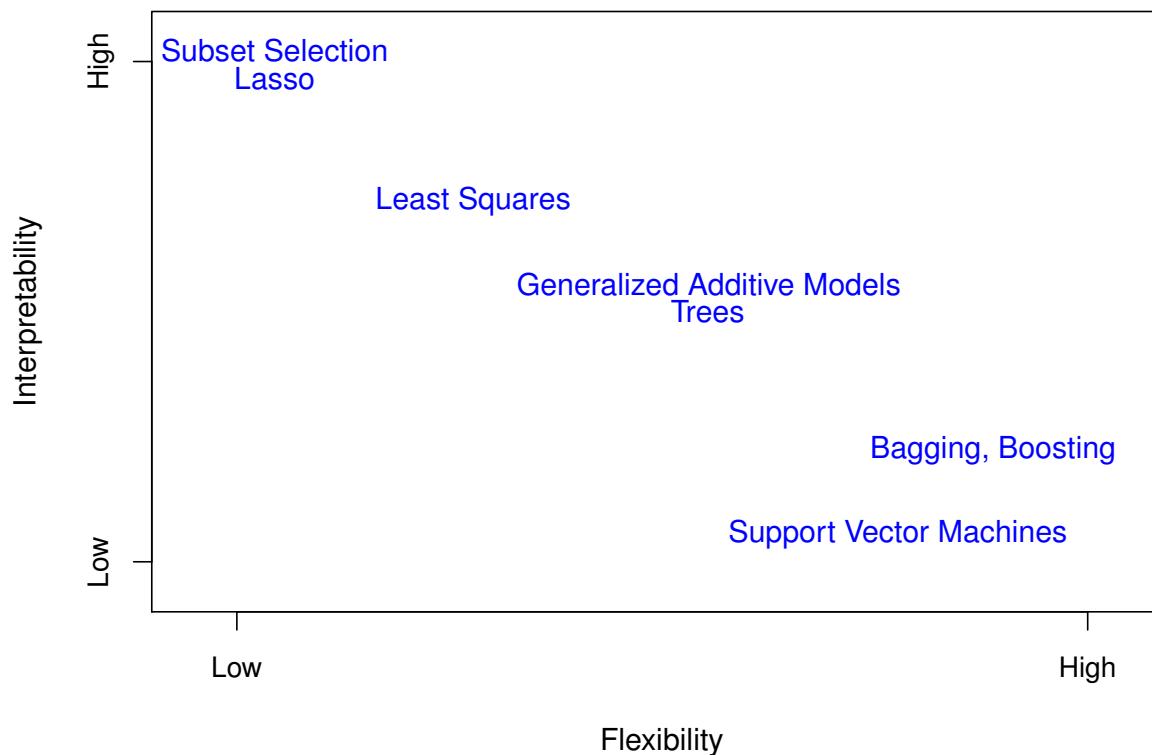
## Loading required package: lattice
## Loading required package: ggplot2

fit <- knnreg(data.frame(horsepower = Auto$horsepower), Auto$mpg, k = 10)
xt <- seq(46, 230, by = 0.001)
yhat <- predict(fit, data.frame(horsepower = xt))
# plot(xt,yhat,type='l',col='red') with(Auto,points(horsepower,mpg))
```

Figure 2.2: knn regression with $k = 10$ 

2.1.4 The Trade-Off Between Prediction Accuracy and Model Interpretability

Figure 2.3: A representation of the tradeoff between flexibility and interpretability, using different statistical learning methods. In general, as the flexibility of a method increases, its interpretability decreases.



Why would we ever choose to use a more restrictive method instead of a very flexible approach?

- If we are mainly interested in inference, then restrictive models are much more interpretable.
- Even for prediction, highly flexible methods may suffer from overfitting and we often obtain more accurate predictions using a less flexible method.
- **The law of parsimony** tells us that when there are alternative explanations of events, the simplest one is likely to be correct.

2.1.5 Assessing Model Accuracy

Measuring the Quality of Fit

- A commonly used measure is the mean square error (MSE):

$$MSE(\hat{f}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(\mathbf{x}_i))^2 \quad (2.4)$$

- **Training MSE and test MSE:** when we have training data $\{(y_1, \mathbf{x}_1), \dots, (y_n, \mathbf{x}_n)\}$ and independent observation (y_0, \mathbf{x}_0) that is not used for estimating f ,

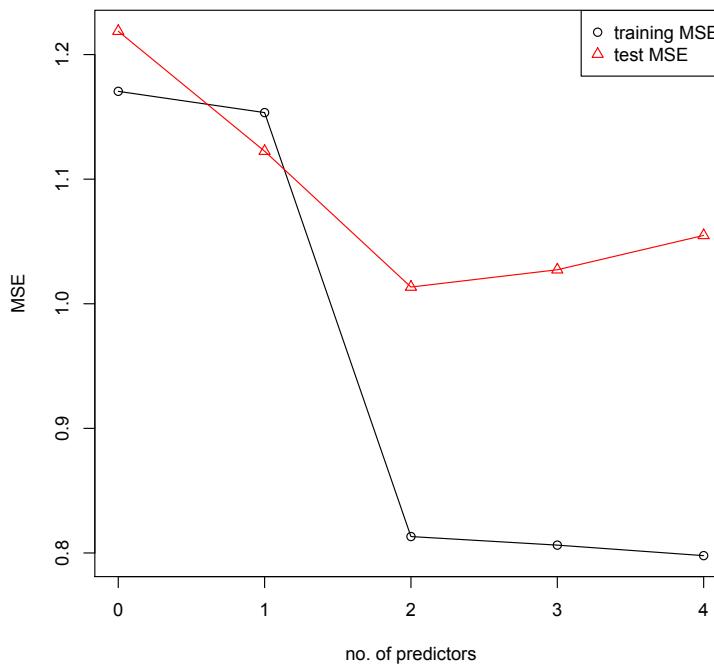
$$(training) \quad MSE(\hat{f}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(\mathbf{x}_i))^2 \quad (2.5)$$

$$(test) \quad MSE(\hat{f}) = Ave (y_0 - \hat{f}(\mathbf{x}_0))^2 \quad (2.6)$$

- A more flexible model tends to have a smaller training MSE than a simpler model. As model flexibility increases, training MSE will decrease, but the test MSE may not.
- When a given method yields a small training MSE but a large test MSE, we are said to be overfitting the data.
- To assess a model accuracy, we need to calculate the test MSE rather than the training MSE.
- How can we go about trying to select a method that minimizes the test MSE?
- There are a variety of approaches to estimate the test MSE. One of important methods is cross-validation.

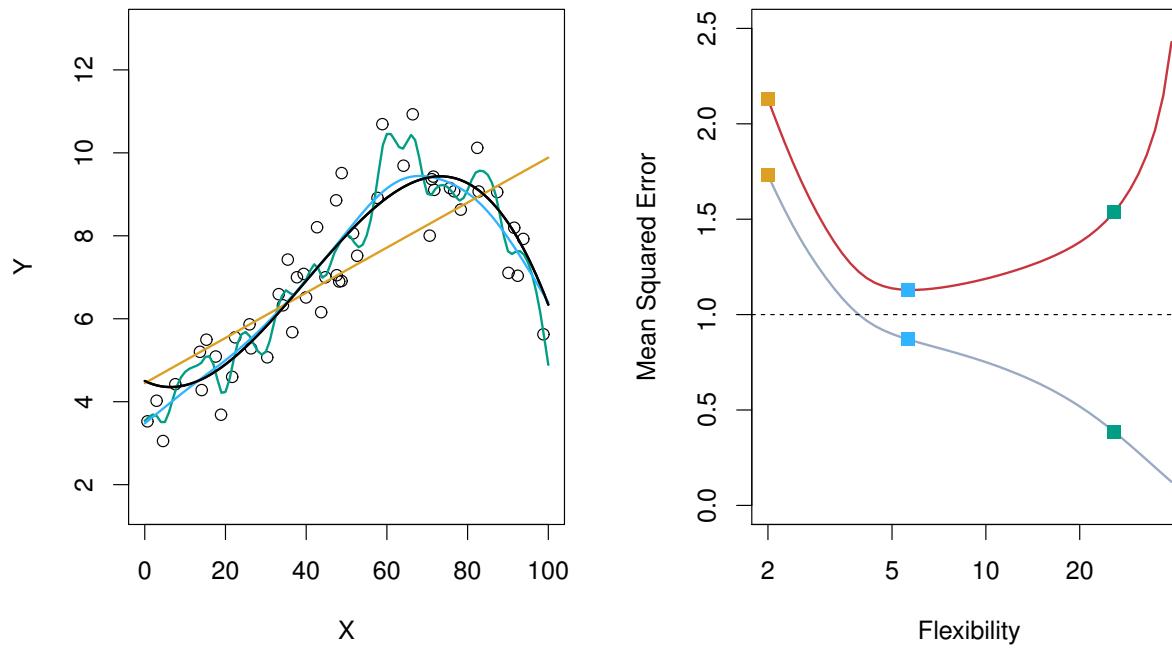
Example 2.1.3

100 observations are selected from $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$ where $\beta_0 = 1, \beta_1 = 1, \beta_2 = 1$, and $\epsilon \sim N(0, 1)$. Four predictors x_1, \dots, x_4 are independently generated from a uniform distribution on $(0, 1)$.



2.1.6 Bias-Variance trade-off

Figure 2.4: Left: Data simulated from f , shown in black. Three estimates of f are shown: the linear regression line (orange curve), and two smoothing spline fits (blue and green curves). Right: Training MSE (grey curve), test MSE (red curve), and minimum possible test MSE over all methods (dashed line). Squares represent the training and test MSEs for the three fits shown in the left-hand panel.



- The U-shape observed in the test MSE curves ([Figure 2.4](#)) turns out to be the result of two competing properties of statistical learning methods.
- The expected test MSE for a given x_0 is

$$E(y_0 - \hat{f}(\mathbf{x}_0))^2 = \text{Var}(\hat{f}(\mathbf{x}_0)) + \text{Bias}(\hat{f}(\mathbf{x}_0))^2 + \text{Var}(\epsilon) \quad (2.7)$$

that refers to the average test MSE obtained by repeatedly estimating f using a large number of training sets and tested each at x_0 .

- Variance refers to the amount by which \hat{f} would change if we estimated it using a different training data set. In general, more flexible statistical methods have higher variance.
- To minimize the expected test error, we need to find a statistical learning method having simultaneously low variance and low bias.

(*Proof of Equation 2.7*). Let (y_0, \mathbf{x}_0) be an independent data point of the training data \mathcal{X} of which \hat{f} is obtained. Let E be the expectation over all possible training data.

$$E(y_0 - \hat{f}(\mathbf{x}_0))^2 = E\left(f(\mathbf{x}_0) + \epsilon_0 - \hat{f}(\mathbf{x}_0)\right)^2 \quad (2.8)$$

$$= E\left(f(\mathbf{x}_0) - \hat{f}(\mathbf{x}_0)\right)^2 + 2E\left(\epsilon_0[f(\mathbf{x}_0) - \hat{f}(\mathbf{x}_0)]\right) + E(\epsilon_0^2) \quad (2.9)$$

$$= E\left(f(\mathbf{x}_0) - \hat{f}(\mathbf{x}_0)\right)^2 + 2\underbrace{(E\epsilon_0)}_{=0} E\left[f(\mathbf{x}_0) - \hat{f}(\mathbf{x}_0)\right] + E(\epsilon_0^2) \quad (2.10)$$

$$= E\left(f(\mathbf{x}_0) - E(\hat{f}(\mathbf{x}_0)) + E(\hat{f}(\mathbf{x}_0)) - \hat{f}(\mathbf{x}_0)\right)^2 + E(\epsilon_0^2) \quad (2.11)$$

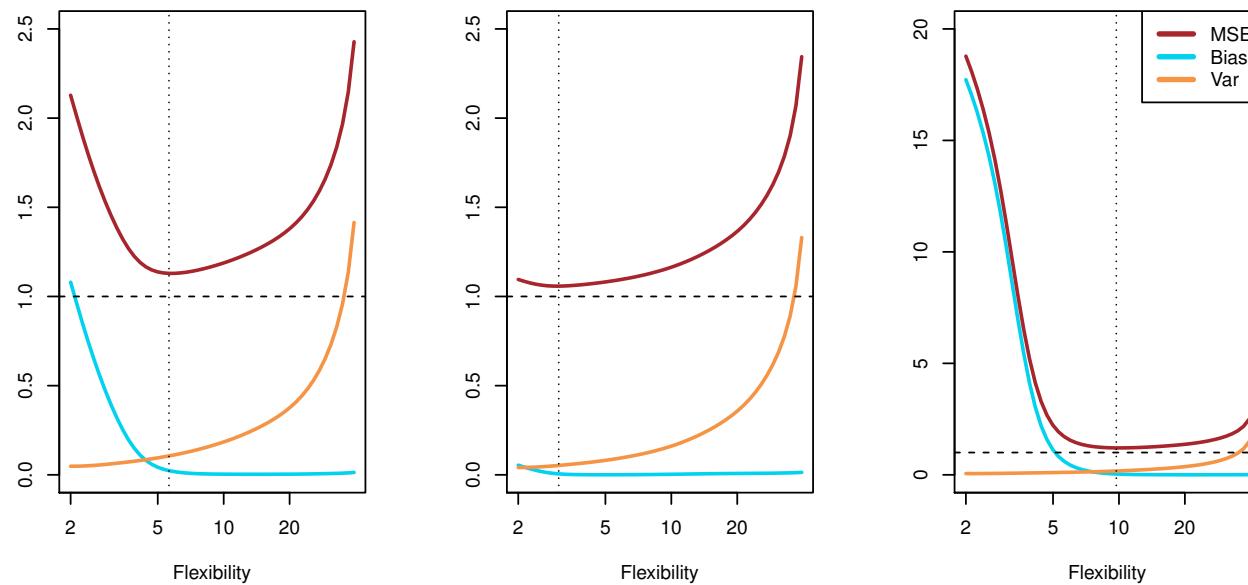
$$= E\left(f(\mathbf{x}_0) - E(\hat{f}(\mathbf{x}_0))\right)^2 + E\left(\hat{f}(\mathbf{x}_0) - E(\hat{f}(\mathbf{x}_0))\right)^2 \quad (2.12)$$

$$+ 2\left[E\left(f(\mathbf{x}_0) - E(\hat{f}(\mathbf{x}_0))\right)\right] \underbrace{\left[E\left(E(\hat{f}(\mathbf{x}_0)) - \hat{f}(\mathbf{x}_0)\right)\right]}_{=0} + E(\epsilon_0^2) \quad (2.13)$$

$$= Bias(\hat{f}(\mathbf{x}_0))^2 + Var(\hat{f}(\mathbf{x}_0)) + Var(\epsilon_0) \quad (2.14)$$

□

Figure 2.5: Squared bias (blue curve), variance (orange curve), $\text{Var}(\epsilon)$ (dashed line), and test MSE (red curve) for the three data sets. The vertical dotted line indicates the flexibility level corresponding to the smallest test MSE.



2.2 Classification

- y_i is no longer numerical but it is categorical.
- \hat{y}_i is the predicted class label for the i th observation using \hat{f} .
- $I(y_i \neq \hat{y}_i) = \begin{cases} 1 & \text{if } y_i \neq \hat{y}_i \\ 0 & \text{if } y_i = \hat{y}_i \end{cases}$ is an indicator function.
- The most common approach for quantifying the accuracy of our estimate \hat{f} is the **training error rate**:

$$\frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$$

- The **test error rate** is calculated from a set of test observations of the form (x_0, y_0) is given by

$$Ave(I(y_0 \neq \hat{y}_0))$$

- We fully discuss the classification problems in Chapter 7.

Example 2.2.1

Simulation example for training and test classification errors:

```
n <- 2000
set.seed(12)
x <- rnorm(n)
z <- 2 * x + x^2

set.seed(21)
y <- rbinom(n, 1, exp(z)/(1 + exp(z)))
train <- data.frame(y = y, x = x)

set.seed(12345)
xt <- rnorm(n)
zt <- 2 * xt + xt^2
set.seed(54321)
yt <- rbinom(n, 1, exp(zt)/(1 + exp(zt)))
test <- data.frame(y = yt, x = xt)

train.err <- numeric(4)
test.err <- numeric(4)

fit1 <- glm(y ~ x, family = binomial)
pred1 <- predict(fit1, newdata = test, type = "response")
cl1 <- ifelse(pred1 > 1/2, 1, 0)
test.err[1] <- mean(yt != cl1)

fit2 <- glm(y ~ poly(x, 2), family = binomial)
pred2 <- predict(fit2, newdata = test, type = "response")
```

```
cl2 <- ifelse(pred2 > 1/2, 1, 0)
test.err[2] <- mean(yt != cl2)

fit3 <- glm(y ~ poly(x, 3), family = binomial)
pred3 <- predict(fit3, newdata = test, type = "response")
cl3 <- ifelse(pred3 > 1/2, 1, 0)
test.err[3] <- mean(yt != cl3)

fit4 <- glm(y ~ poly(x, 4), family = binomial)
pred4 <- predict(fit4, newdata = test, type = "response")
cl4 <- ifelse(pred4 > 1/2, 1, 0)
test.err[4] <- mean(yt != cl4)

t1 <- predict(fit1, newdata = train, type = "response")
cl1 <- ifelse(t1 > 1/2, 1, 0)
train.err[1] <- mean(yt != cl1)

t2 <- predict(fit2, newdata = train, type = "response")
cl2 <- ifelse(t2 > 1/2, 1, 0)
train.err[2] <- mean(yt != cl2)

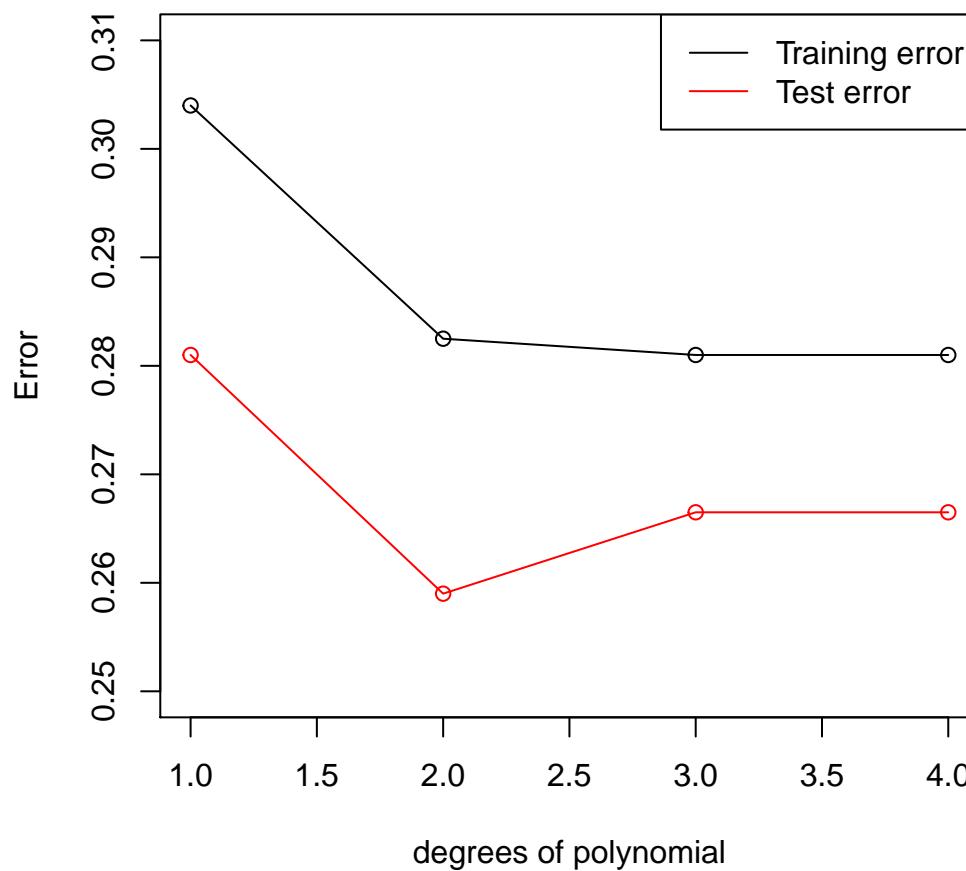
t3 <- predict(fit3, newdata = train, type = "response")
cl3 <- ifelse(t3 > 1/2, 1, 0)
train.err[3] <- mean(yt != cl3)

t4 <- predict(fit4, newdata = train, type = "response")
```

```
cl4 <- ifelse(t4 > 1/2, 1, 0)
train.err[4] <- mean(y != cl4)

plot(train.err, type = "o", ylim = c(0.25, 0.31), ylab = "Error", xlab = "degrees of polynomial")
points(c(1:4), test.err, type = "o", col = 2)
legend("topright", legend = c("Training error", "Test error"), lty = 1, col = c(1,
2))
```

Figure 2.6: Training and test classification errors for simulated data



Chapter 3 Resampling Methods

3.1 Introduction

From wikipedia:

In statistics, resampling is any of a variety of methods for doing one of the following:

- Estimating the precision of sample statistics (medians, variances, percentiles) by using subsets of available data (jackknifing) or drawing randomly with replacement from a set of data points (bootstrapping)
- Exchanging labels on data points when performing significance tests (permutation tests, also called exact tests, randomization tests, or re-randomization tests)
- Validating models by using random subsets (bootstrapping, cross validation)

Common resampling techniques include bootstrapping, jackknifing and permutation tests.

3.2 Permutation tests

- Suppose that we want to compare two means of X and Y by assuming the distribution of X and the distribution of Y can be written as $F_X(u) = F(u - \mu_X)$ and $F_Y(u) = F(u - \mu_Y)$ with unknown common distribution F .
- Let x_1, \dots, x_{n_1} and y_1, \dots, y_{n_2} be two independent random samples.
- The Wilcoxon ranks sum test statistic is to calculate the sum of ranks of X values: $W = R_1 + \dots + R_{n_1}$.
- Mann-Whitney statistic is to calculate the counts of pairs that x value is greater than y value, that is, $U = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} I(x_i > y_j)$.
- The permutation p-value of Wilcoxon-Mann-Whitney ranks sum test may be calculated from a randomization method.
- For example, suppose that $X = (1.2, 4.5)$ and $Y = (2.2, 7.7, 8.3)$ where $n_1 = 2, n_2 = 3$. There are $\binom{N}{n_1}$ possible permutations. For instance, $\binom{5}{2} = 10$. All possible permutations may be written as follows:

Table 3.1: All possible permutations for $N = 5$ and $n_1 = 2$

Ranks					w_1	u_1
1	2	3	4	5		
x	x	y	y	y	3	0
x	y	x	y	y	4	1 ← data
x	y	y	x	y	5	2
x	y	y	y	x	6	3
y	x	x	y	y	5	2
y	x	y	x	y	6	3
y	x	y	y	x	7	4
y	y	x	x	y	7	4
y	y	x	y	x	8	5
y	y	y	x	x	9	6

The null distribution may be summarized as in table 3.2:

Table 3.2: Null distribution of the Wilcoxon rank sum test and Mann-Whitney U test for $N = 5$ and $n_1 = 2$

w_1	u_1	$\Pr(W_1 = w_1) = \Pr(U_1 = u_1)$
3	0	0.1
4	1	0.1
5	2	0.2
6	3	0.2
7	4	0.2
8	5	0.1
9	6	0.1

For the given data $X = (1.2, 4.5)$ and $Y = (2.2, 7.7, 8.3)$, the ranks sum of x'_i s equals $w_1 = 4$ and the $P_L = 0.2$.

3.3 Jackknife methods

This method was originally developed to reduce bias of estimators (Quenouille 1949, 1956). It systematically recomputed the statistic by leaving out one observation at a time from the data. This allows us to estimate variability of statistic (J.W. Tukey 1958).

Let $T_n = T_n(X_1, \dots, X_n)$ be an estimator of an unknown parameter θ . The bias of T_n is defined by

$$\text{bias}(T_n) = E_\theta(T_n) - \theta. \quad (3.1)$$

Let $T_{n-1,i} = T_{n-1}(X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n)$ be the estimator based on the $(n-1)$ observations leaving out X_i . Define the average of $T_{n-1,i}$:

$$\bar{T}_n = \frac{1}{n} \sum_{k=1}^n T_{n-1,k} \quad (3.2)$$

Quenouille's jackknife bias estimator is defined as $b_{jack} = (n-1)(\bar{T}_n - T_n)$. This leads to a bias-reduced jackknife estimator of θ is $T_{jack} = T_n - b_{jack} = nT_n - (n-1)\bar{T}_n$.

Tukey defined the pseudo-value $T_i^* = nT_n - (n-1)T_{n-1,i}$. Using the pseudo-values,

$$T_{jack} = \frac{1}{n} \sum_{i=1}^n T_i^* \quad (3.3)$$

If $\text{bias}(T_n) = \frac{a}{n} + \frac{b}{n^2} + hot$, then $\text{bias}(T_{jack}) = -\frac{b}{n(n-1)} + hot$.

Tukey (1958) used the pseudo-values to estimate the variance of an estimator.

1. The pseudo values can be treated as though they are iid.

2. The pseudo values have approximately the same variance as $\sqrt{n}T_n$.

Therefore the jackknife variance estimator for T_n is

$$v_{jack} = \frac{1}{n} (\text{Sample variance of } \{T_1^*, \dots, T_n^*\}) \quad (3.4)$$

$$= \frac{1}{n(n-1)} \sum_{i=1}^n \left(T_i^* - \frac{1}{n} \sum_{j=1}^n T_j^* \right)^2 \quad (3.5)$$

$$= \frac{n-1}{n} \sum_{i=1}^n \left(T_{n-1,i} - \frac{1}{n} \sum_{j=1}^n T_{n-1,j} \right)^2 \quad (3.6)$$

Example 3.3.1

Consider a data set containing three values: $x_1 = 1, x_2 = 2, x_3 = 3$. The sample variance $s^2 = 1$. We want to estimate the standard error of the sample variance. Calculate the Jackknife variance estimate of the sample variance.

Let $T_3 = s^2 = 1$.

i	1	2	3
$T_{2,(i)}$	$(.5)^2 + (.5)^2 = .5$	$1^2 + 1^2 = 2$	$(.5)^2 + (.5)^2 = .5$
T_i^*	$3 - 2 \times .5 = 2$	$3 - 2 \times 2 = -1$	$3 - 2 \times .5 = 2$

The sample variance of T_i^* is equal to $\frac{1}{n-1} \sum_{i=1}^n \left(T_i^* - \bar{T}^* \right)^2 = \frac{1}{2}[(1)^2 + (-2)^2 + (1)^2] = 3$. The jackknife variance estimate is equal to the sample variance of the pseudo-values T_i^* divided by the sample size, that is, $v_{jack} = \frac{1}{n(n-1)} \sum_{i=1}^n \left(T_i^* - \bar{T}^* \right)^2 = \frac{3}{3} = 1$. \square

Example 3.3.2 (Sample mean)

When $T_n = \bar{X}_n$, $T_{n-1,i} = \bar{X}_{n-1,i} = \frac{n\bar{X}_n - X_i}{n-1}$. $T_{jack} = T_n = \bar{X}_n$. The pseudo value is $X_i^* = X_i$. \square

Example 3.3.3 (Jackknife variance estimator of the sample mean)

$$v_{jack} = s^2/n.$$

□

Example 3.3.4

Let $T_n = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2$ that is the MLE of the σ^2 for normal distribution. The Jackknife estimator is $T_{jack} = s^2$. □

3.4 Nonparametric Bootstrap

Bootstrap method was developed by B. Efron. It resamples from the original data set with replacement. There are wide range of applications of bootstrap method: confidence intervals, constructing the null distribution of a statistic, etc.

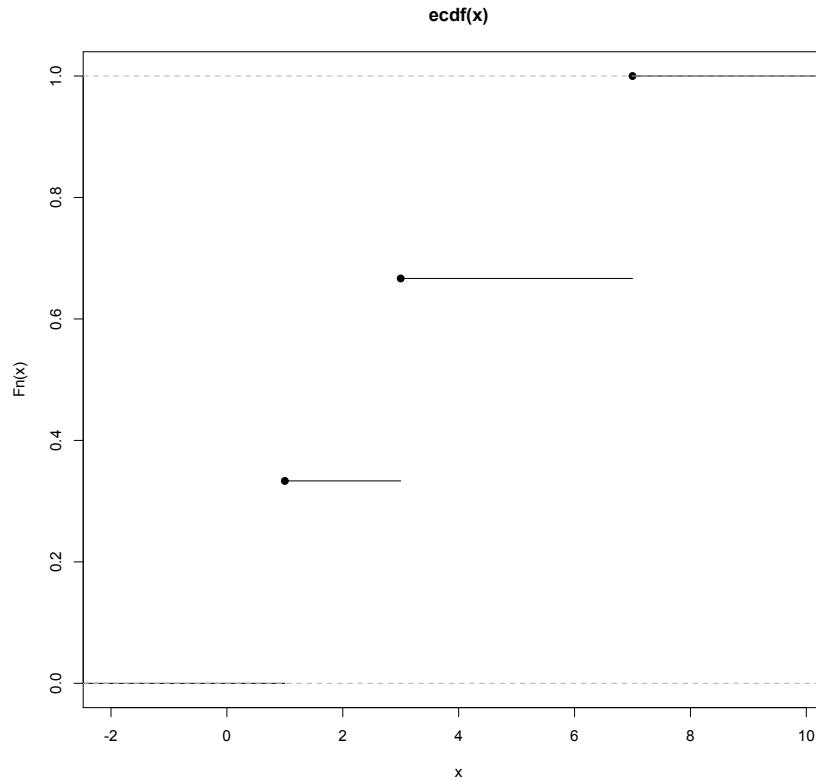
1. **Generating distribution of a statistic** When the sample size N is large, the empirical distribution \mathbb{F}_N would be a good estimator of the true distribution F , denoted by $\hat{F} = \mathbb{F}_N$ where $\mathbb{F}_N(x) = \frac{1}{N} \sum_{i=1}^N I(X_i \leq x)$ for a given sample $\{X_1, \dots, X_N\}$.

Remark 1 (Empirical distribution Function (edf)). • Maximum likelihood estimation (MLE) is to maximize the log-likelihood function over the parameter. Let $F(x) = F_X(x)$ be the distribution function of the random variable X . For any fixed x , we call “success” if $X \leq x$ with probability $F(x) = \Pr(X \leq x)$. When a random sample X_1, \dots, X_N is selected, the MLE of $F(x)$ is the sample proportion $\frac{\#\{i : X_i \leq x\}}{N}$. We define the empirical distribution function as the MLE given x , i.e.,

$$\mathbb{F}_N(x) = \frac{\#\{i : X_i \leq x\}}{N} = \frac{1}{N} \sum_{i=1}^N I(X_i \leq x). \quad (3.7)$$

The standard error of the MLE is $\sqrt{\frac{F(x)(1-F(x))}{N}}$. The estimate of the standard error of the MLE is $\sqrt{\frac{\mathbb{F}_N(x)(1-\mathbb{F}_N(x))}{N}}$.

- The empirical distribution function \mathbb{F}_N converges to the distribution function F as $N \rightarrow \infty$, that is, $\sup_{x \in A} |\mathbb{F}_N(x) - F(x)| \rightarrow 0$ as $N \rightarrow \infty$ for any compact set $A \subset \mathbb{R}$. In other word, for a large sample the empirical distribution function is a good approximate of the distribution function.
- For example, suppose $x_1 = 3, x_2 = 1, x_3 = 7$. The graph of the empirical distribution function \mathbb{F}_3 is as follow:



2. Let θ be a parameter $\theta = t(F)$. For example, $\theta = t(F) = \int y dF(y)$ for the mean parameter.
3. The estimate of θ is $\hat{\theta} = t(\hat{F})$.
4. Bias of the estimate is defined as $b(F) = E(\hat{\theta}|F) - \theta = E(\hat{\theta}|F) - t(F)$.
5. Variance of the estimate is $v(F) = \text{var}(\hat{\theta}|F)$.

6. We may hope

$$Z = \frac{\hat{\theta} - E(\hat{\theta})}{\sqrt{v(F)}} = \frac{\hat{\theta} - \theta - b(F)}{\sqrt{v(F)}} \xrightarrow{asympt} N(0, 1)$$

A $100(1 - \alpha)\%$ confidence interval is

$$\hat{\theta} - b(F) - z_{1-\alpha/2}\sqrt{v(F)} < \theta < \hat{\theta} - b(F) - z_{\alpha/2}\sqrt{v(F)}$$

This CI equation may be useless when F is unknown.

7. We draw B random samples of size n from \hat{F} , say the b th bootstrap random sample $\{y_{b1}^*, \dots, y_{bn}^*\}$.

Figure 3.1: Bootstrap (resampling from data with replacement)

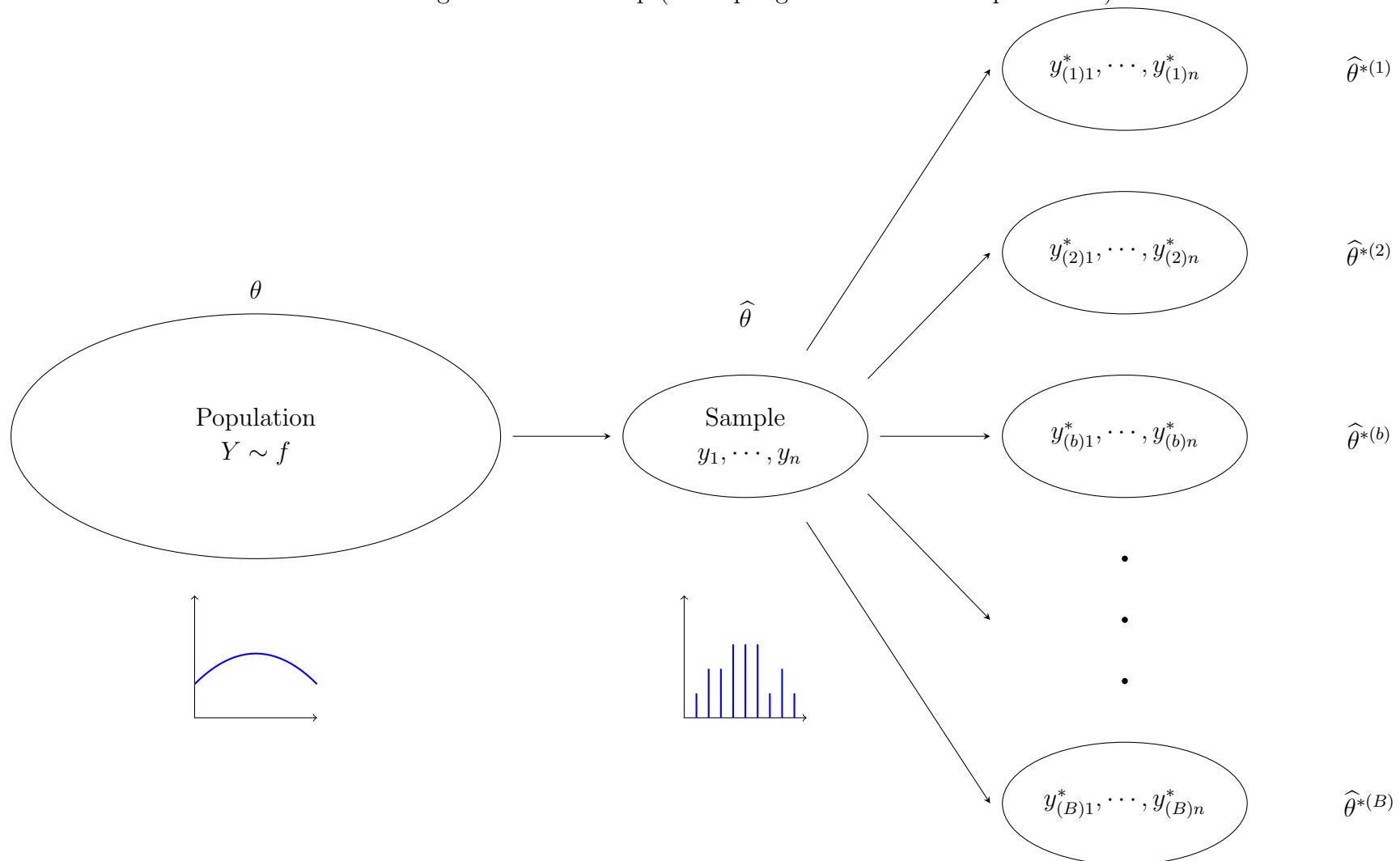


Table 3.3: Bootstrap Samples

Bootstrap Resample		
1	$y_{11}^*, \dots, y_{1n}^*$	$\hat{\theta}_1^*$
\vdots	\vdots	\vdots
b	$y_{b1}^*, \dots, y_{bn}^*$	$\hat{\theta}_b^*$
\vdots	\vdots	\vdots
B	$y_{B1}^*, \dots, y_{Bn}^*$	$\hat{\theta}_B^*$

We estimate the bias and variance.

$$\begin{aligned} b(\hat{F}) &= E(\hat{\theta}^*|\hat{F}) - t(\hat{F}) \\ \hat{b}^*(\hat{F}) &= \frac{1}{B} \sum_{i=1}^B \hat{\theta}_i^* - \hat{\theta} = \bar{\hat{\theta}}^* - \hat{\theta} \\ v(\hat{F}) &= \text{var}(\hat{\theta}^*|\hat{F}) \\ \hat{v}^*(\hat{F}) &= \frac{1}{B-1} \sum_{i=1}^B (\hat{\theta}_i^* - \bar{\hat{\theta}}^*)^2 \end{aligned}$$

8. Bootstrap Confidence Intervals

a. Normal

We may construct a confidence interval for θ by replacing $b(F)$ and $v(F)$ with $\hat{b}^*(\hat{F})$ and $\hat{v}^*(\hat{F})$.

$$\hat{\theta} - \hat{b}^*(\hat{F}) - z_{1-\alpha/2} \sqrt{\hat{v}^*(\hat{F})} < \theta < \hat{\theta} - \hat{b}^*(\hat{F}) - z_{\alpha/2} \sqrt{\hat{v}^*(\hat{F})}$$

Equivalently,

$$2\hat{\theta} - \widehat{\theta^*} - z_{1-\alpha/2}\sqrt{\widehat{v^*}(\widehat{F})} < \theta < 2\hat{\theta} - \widehat{\theta^*} - z_{\alpha/2}\sqrt{\widehat{v^*}(\widehat{F})}$$

b. Basic (Residual)

The normal approximation may fail when $\widehat{\theta} - \theta$ is not close to normal. Using the bootstrap samples, we can approximate the distribution of $\widehat{\theta} - \theta$ by the resampling distribution of $\widehat{\theta}^* - \widehat{\theta}$ and we obtain

$$\begin{aligned} \widehat{\theta}_{((B+1)\alpha/2)}^* - \widehat{\theta} &< \widehat{\theta} - \theta < \widehat{\theta}_{((B+1)(1-\alpha/2))}^* - \widehat{\theta} \\ \widehat{\theta} - (\theta_{((B+1)(1-\alpha/2))}^* - \widehat{\theta}) &< \theta < \widehat{\theta} - (\theta_{((B+1)(\alpha/2))}^* - \widehat{\theta}) \end{aligned}$$

Equivalently,

$$2\widehat{\theta} - \theta_{((B+1)(1-\alpha/2))}^* < \theta < 2\widehat{\theta} - \theta_{((B+1)(\alpha/2))}^*$$

c. Percentile

This method is very simple. Let $\widehat{\theta}_{(1)}^* \leq \dots \leq \widehat{\theta}_{(B)}^*$ the ordered statistic values using the bootstrap samples. The percentile confidence interval of θ is

$$\theta_{((B+1)\alpha/2)}^* < \theta < \theta_{((B+1)(1-\alpha/2))}^*$$

d. Studentized (t-pivot)

When an approximate variance $\sigma^2(\widehat{\theta})$ for $\widehat{\theta}$ is available and can be calculated from y_1, \dots, y_n , a confidence interval based on $t = (\widehat{\theta} - \theta)/\sigma(\widehat{\theta})$. Calculate $\{t_b^* = (\widehat{\theta}_b^* - \widehat{\theta})/\sigma(\widehat{\theta}_b^*)\}$.

$$t_{((B+1)(\alpha/2))}^* < \frac{\widehat{\theta} - \theta}{\sigma(\widehat{\theta})} < t_{((B+1)(1-\alpha/2))}^*$$

$$\widehat{\theta} - t_{((B+1)(1-\alpha/2))}^* \sigma(\widehat{\theta}) < \theta < \widehat{\theta} - t_{((B+1)(\alpha/2))}^* \sigma(\widehat{\theta})$$

e. BCa (Bias-Corrected and accelerated)

The BCa method creates an interval similar to the percentile interval:

$$\widehat{\theta}_{((B+1)\alpha_L)}^* < \theta < \widehat{\theta}_{((B+1)\alpha_U)}^*$$

where α_L and α_U are chosen to have the same cumulative probability as z_L and z_U , defined as,

$$z_L = \frac{z_0 - z_{1-\alpha/2}}{1 - a(z_0 - z_{1-\alpha/2})} + z_0, \quad z_U = \frac{z_0 + z_{1-\alpha/2}}{1 - a(z_0 - z_{1-\alpha/2})} + z_0$$

The value of z_0 is defined as $P(Z \leq z_0) = p_0$ and p_0 is the proportion of $\{\widehat{\theta}_b^* : b = 1, \dots, B\}$ so that $\widehat{\theta}_b^* \leq \widehat{\theta}$. This corrects the median bias. The value of a measures the skewness of the data and is calculated by

$$a = \frac{\sum (\bar{\widehat{\theta}}_0 - \widehat{\theta}_{-i})^3}{6[\sum (\bar{\widehat{\theta}}_0 - \widehat{\theta}_{-i})^2]^{3/2}}$$

The value of $\widehat{\theta}_{-i}$ is the leave one out estimate by eliminating y_i and the value of $\bar{\widehat{\theta}}_0$ is the mean of $\{\widehat{\theta}_{-i}\}$. When $a = 0$ and $z_0 = 0$, this BCa interval is the same as the percentile interval.

Example 3.4.1

Use library(boot) in R.

```
library(boot)

##
## Attaching package: 'boot'
## The following object is masked from 'package:lattice':
##
##     melanoma

# Nonparametric confidence intervals for mean failure time of the
# air-conditioning data as in Example 5.4 of Davison and Hinkley (1997)
mean.fun <- function(d, i) {
  m <- mean(d$hours[i])
  n <- length(i)
  v <- (n - 1) * var(d$hours[i])/n^2
  c(m, v)
}
air.boot <- boot(aircondit, mean.fun, R = 999)
boot.ci(air.boot, type = c("norm", "basic", "perc", "stud"))

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = air.boot, type = c("norm", "basic", "perc",
```

```

##      "stud"))
##
## Intervals :
## Level      Normal          Basic
## 95%   ( 36.9, 178.6 )  ( 28.8, 168.2 )
##
## Level      Studentized     Percentile
## 95%   ( 51.1, 278.1 )  ( 48.0, 187.3 )
## Calculations and Intervals on Original Scale

```

3.4.1 Bootstrap variance estimation (Portfolio data)

- Suppose that we wish to invest a fixed sum of money in two financial assets that yield returns of X and Y , respectively, where X and Y are random quantities.
- We will invest a fraction α of our money in X , and will invest the remaining $1 - \alpha$ in Y .
- We wish to choose α to minimize the total risk, or variance, of our investment. In other words, we want to minimize $\text{Var}(\alpha X + (1-\alpha)Y)$
-

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

How to estimate α and what is the s.e. of $\hat{\alpha}$?

-

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}$$

- It seems we need additional data sets to estimate the standard error of $\hat{\alpha}$.
- However, we can estimate the s.e. of $\hat{\alpha}$ without any additional data sets using Bootstrap method since the true unknown distribution $F_{X,Y}$ may be approximated by the empirical distribution $\hat{F}_{X,Y}$.

Bootstrap resamples

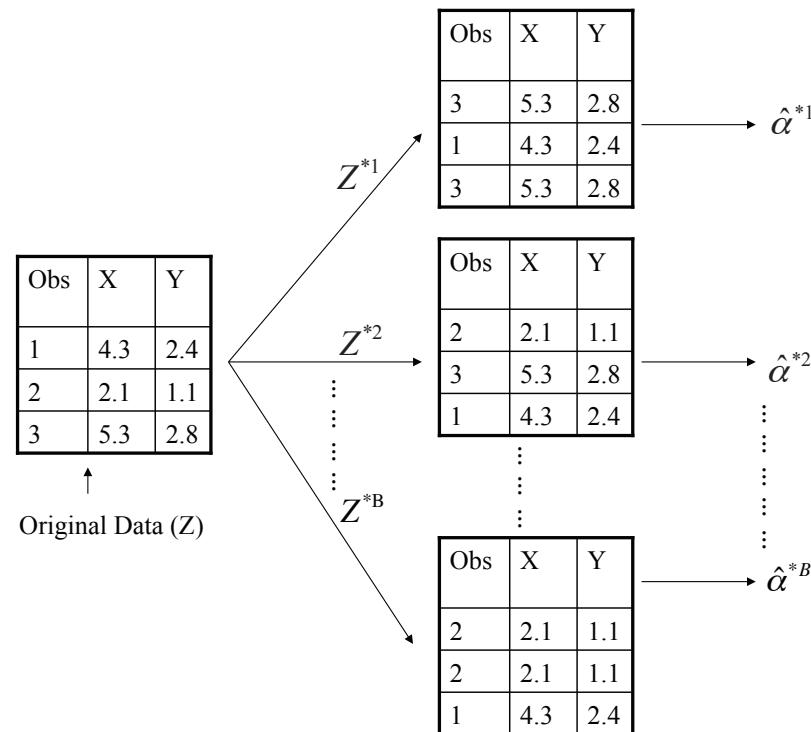
Given data $(x_1, y_1), \dots, (x_n, y_n)$, we select B bootstrap resamples with replacement

$$\begin{aligned} b = 1 & \quad (x_1^{(1)}, y_1^{(1)}), \dots, (x_n^{(1)}, y_n^{(1)}) \quad \text{estimate } \hat{\alpha}^{(1)} \\ & \vdots \qquad \vdots \\ b = B & \quad (x_1^{(B)}, y_1^{(B)}), \dots, (x_n^{(B)}, y_n^{(B)}) \quad \text{estimate } \hat{\alpha}^{(B)} \end{aligned}$$

We can estimate

$$\widehat{se(\hat{\alpha})} = \sqrt{\frac{1}{B-1} \sum_{b=1}^B \left(\hat{\alpha}^{(b)} - \frac{1}{B} \sum_{j=1}^B \hat{\alpha}^{(j)} \right)^2}$$

Figure 3.2: Bootstrap estimation



Example 3.4.2

(Portfolio data)

```
library(ISLR)
library(boot)
alpha.fn = function(data, index) {
  X = data$X[index]
  Y = data$Y[index]
  return((var(Y) - cov(X, Y))/(var(X) + var(Y) - 2 * cov(X, Y)))
}
alpha.fn(Portfolio, 1:100)

## [1] 0.5758321

set.seed(1)
alpha.fn(Portfolio, sample(100, 100, replace = T))

## [1] 0.5963833

boot(Portfolio, alpha.fn, R = 1000)

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Portfolio, statistic = alpha.fn, R = 1000)
##
```

```
##  
## Bootstrap Statistics :  
##     original      bias    std. error  
## t1* 0.5758321 -7.315422e-05 0.08861826
```

Example 3.4.3

Consider a data set containing three values: $x_1 = 1, x_2 = 2, x_3 = 3$. The sample variance $s^2 = 1$. We want to estimate the standard error of the sample variance. Calculate the bootstrap variance estimate of the sample variance.

Bootstrap sample	Probability	Bootstrap Estimate
(1,1,1)	$\frac{1}{27}$	0
(2,2,2)	$\frac{1}{27}$	0
(3,3,3)	$\frac{1}{27}$	0
(1,1,2)	$\frac{3}{27}$	$\frac{1}{3}$
(1,1,3)	$\frac{3}{27}$	$\frac{4}{3}$
(2,2,1)	$\frac{3}{27}$	$\frac{1}{3}$
(2,2,3)	$\frac{3}{27}$	$\frac{1}{3}$
(3,3,1)	$\frac{3}{27}$	$\frac{4}{3}$
(3,3,2)	$\frac{3}{27}$	$\frac{1}{3}$
(1,2,3)	$\frac{6}{27}$	1

The distribution table for s^{2*} is given by

s^{2*}	0	1/3	1	4/3
$P(s^{2*})$	3/27	12/27	6/27	6/27

The variance of the twenty seven s^{2*} values is equal to $\frac{2}{9}$ that is the bootstrap variance estimate of the sample variance. \square

3.5 Parametric Bootstrap

In a parametric model, the parameters can be estimated by some statistical methods such as maximum likelihood method. However, in some cases, the asymptotic results may not be applicable due to violations of mathematical assumptions (regularity conditions) as in mixture models. By drawing random samples using the estimated parameters, the null distribution of the statistic may be generated.

Example 3.5.1

Suppose we want to test whether a sample is drawn from a single normal distribution or two components normal distribution. Let $H_0 : f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$ and $H_1 : f(x) = p_1 \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) + p_2 \frac{1}{\sqrt{2\pi}\sigma_2} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right)$.

```
library(mclust)

## Package 'mclust' version 5.4.2
## Type 'citation("mclust")' for citing this R package in publications.

n <- 100
x <- rnorm(n)
B <- 100
mu_mle <- mean(x)
s_mle <- sqrt(((n - 1)/n) * var(x))
Cloglik <- numeric(B)
loglik_data <- 2 * (Mclust(x, 2)$loglik - Mclust(x, 1)$loglik)
library(doMC)

## Loading required package: foreach
```

```
## Loading required package: iterators
## Loading required package: parallel

registerDoMC(20)
r <- foreach(i = 1:B) %dopar% {
  xb <- rnorm(n, mean = mu_mle, s = s_mle)
  2 * (Mclust(xb, 2)$loglik - Mclust(xb, 1)$loglik)
}
Cloglik <- unlist(r)
sum(Cloglik >= loglik_data)/(B + 1)

## [1] 0.5940594

pro <- c(0.3, 0.7)
mu <- c(-1, 1)
s <- c(0.5, 0.5)
z <- sample(1:2, n, replace = T, prob = pro)
x <- rnorm(n, mean = mu[z], sd = s[z])

B <- 100
mu_mle <- mean(x)
s_mle <- sqrt(((n - 1)/n) * var(x))
Cloglik <- numeric(B)
loglik_data <- 2 * (Mclust(x, 2)$loglik - Mclust(x, 1)$loglik)
r <- foreach(i = 1:B) %dopar% {
  xb <- rnorm(n, mean = mu_mle, s = s_mle)
  2 * (Mclust(xb, 2)$loglik - Mclust(xb, 1)$loglik)
```

```
}

Cloglik <- unlist(r)
sum(Cloglik >= loglik_data)/(B + 1)

## [1] 0
```


Chapter 4 Training set, validation set, and test set

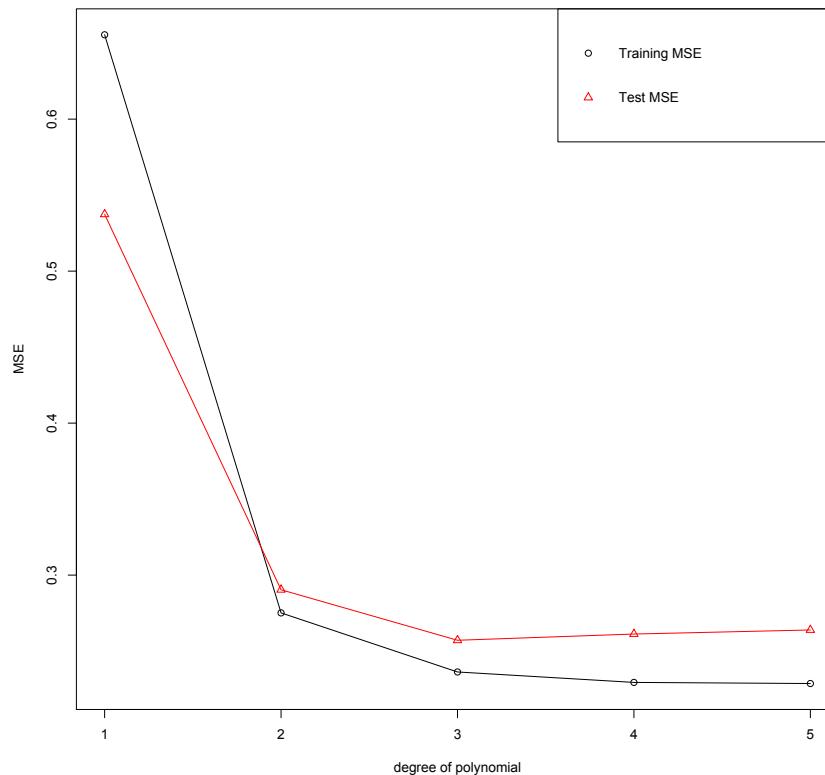
4.1 Cross-validation

4.1.1 Training set, validation set, and test set

- Training set: A set of samples used for learning, that is to fit the model.
 - Validation set: A set of samples used to tune the hyperparameters of a model, for example to choose the number of hidden units in a neural network.
 - Test set: A set of samples used only to assess the performance [generalization] of a fully-specified model.
-

Example 4.1.1 (Training error and test error)

100 values for a predictor x are generated from a uniform distribution on $(0,1)$. 100 values for y are selected from $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$ where $\beta_0 = 1, \beta_1 = 1, \beta_2 = 0, \beta_3 = 1$, and $\epsilon \sim N(0, 1)$.



4.1.2 Validation set (hold-out) approach

- Randomly divide the available data set into (1) a training set and (2) validation set (hold-out) set.
- Fit a model with the training set and predict with the validation set.
- Disadvantage: (1) the estimates are highly variable (2) only training set is used to fit a model and commonly overestimate the test error rate.
- Example: Auto data

```
library(ISLR)
str(Auto)

## 'data.frame': 392 obs. of  9 variables:
## $ mpg          : num  18 15 18 16 17 15 14 14 14 15 ...
## $ cylinders    : num  8 8 8 8 8 8 8 8 8 ...
## $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
## $ horsepower   : num  130 165 150 150 140 198 220 215 225 190 ...
## $ weight        : num  3504 3693 3436 3433 3449 ...
## $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
## $ year         : num  70 70 70 70 70 70 70 70 70 70 ...
## $ origin       : num  1 1 1 1 1 1 1 1 1 ...
## $ name         : Factor w/ 304 levels "amc ambassador brougham",...: 49 36 231 14 161 141 54 223 241 2 ...

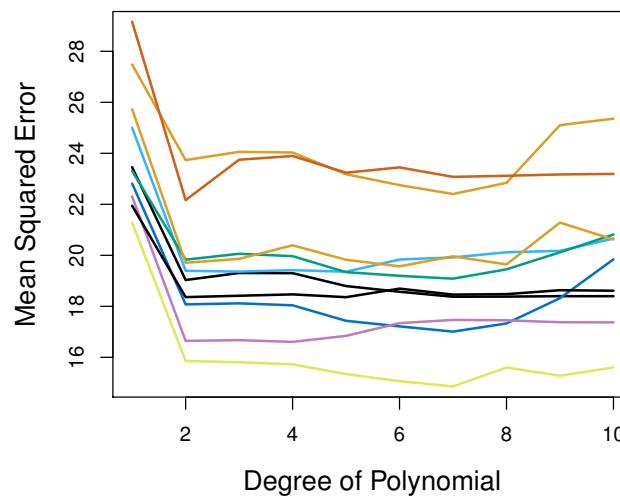
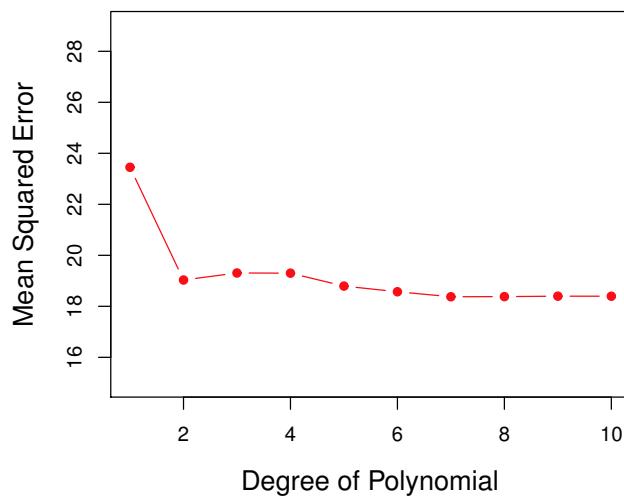
n <- dim(Auto)[1]
set.seed(1)
train <- sample(1:n, n/2)
train
```

```
## [1] 105 146 224 354  79 348 365 255 242  24 388  68 262 391 292 188 270
## [18] 372 143 290 387 382 384  47  99 142   5 140 317 124 175 217 178  67
## [35] 297 239 283  39 257 379 289 228 275 194 185 274   9 165 252 238 164
## [52] 294 149  83 383  34 107 174 222 136 304  98 152 110 214  85 157 250
## [69]  28 356 329 376 111 336 330 323 347 123 245 301 333 334 363 101 234
## [86]  63 218  38  75  44  73  18 193 263 233 237 135 121 357 360 192 103
## [103] 371 287 183  62 305 137 299 170 276 206 100 295  42   4 198  29 315
## [120] 362 321 296 131 369 203 122 312  55  61 326 151  21  10 167 240 154
## [137] 144 271 251 129 173 380  60  65 181 112 303 288  26 211 340 385 373
## [154] 109 120  43 125 313 249  50 359 207 291 179 201  94  15  76 163 225
## [171] 386 186 189  86 339 195 311 160 130 300 307  41 187 106 314  40 284
## [188] 370 213 247 256 258 261 375  57 117
```

Figure 4.1: A schematic display of the validation set approach. A set of n observations are randomly split into a training set (shown in blue, containing observations 7, 22, and 13, among others) and a validation set (shown in beige, and containing observation 91, among others). The statistical learning method is fit on the training set, and its performance is evaluated on the validation set.

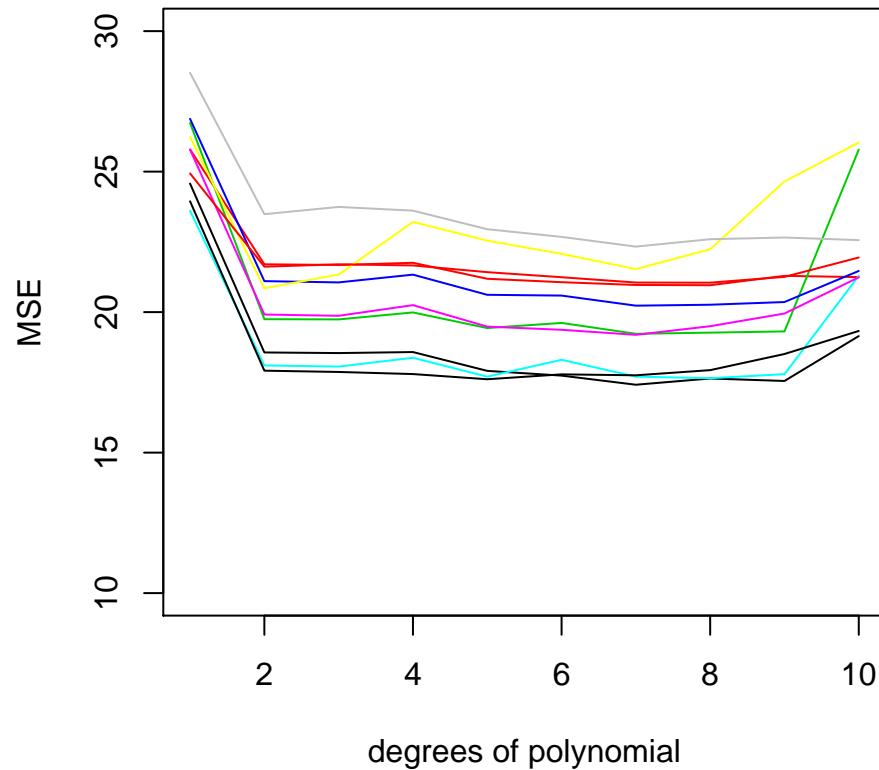


Figure 4.2: The validation set approach was used on the Auto data set in order to estimate the test error that results from predicting mpg using polynomial functions of horsepower. Left: Validation error estimates for a single split into training and validation data sets. Right: The validation method was repeated ten times, each time using a different random split of the observations into a training set and a validation set. This illustrates the variability in the estimated test MSE that results from this approach.



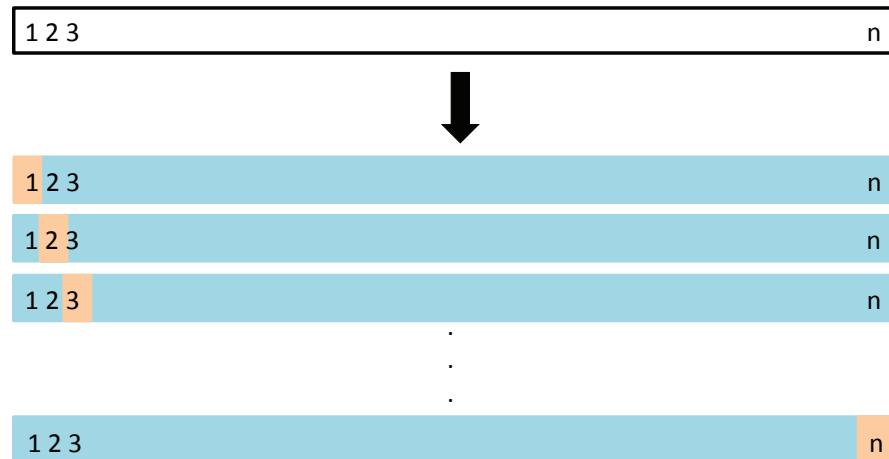
```
# Plot validation set approach
for (j in 1:10) {
  set.seed(10 * j)
  train = sample(392, 196)
  vs.error = rep(0, 10)
  for (i in 1:10) {
    fit = lm(mpg ~ poly(horsepower, i), data = Auto, subset = train)
    vs.error[i] = mean((Auto$mpg[-train] - predict(fit, Auto[-train, ]))^2)
  }
  if (j < 2) {
    plot(vs.error, type = "l", ylim = c(10, 30), ylab = "MSE", xlab = "degrees of polynomial")
  } else {
    lines(vs.error, type = "l", col = j)
  }
}
```

Figure 4.3: Validation set approach was applied to Auto data for polynomial regression to estimate the test MSE.



4.1.3 Leave-One-Out Cross-validation (LOOCV)

Figure 4.4: A schematic display of LOOCV. A set of n data points is repeatedly split into a training set (shown in blue) containing all but one observation, and a validation set that contains only that observation (shown in beige). The test error is then estimated by averaging the n resulting MSE's. The first training set contains all but observation 1, the second training set contains all but observation 2, and so forth.



- Validation set = $\{(\mathbf{x}_i, y_i)\}$, Training set = $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_{i-1}, y_{i-1}), (\mathbf{x}_{i+1}, y_{i+1}), \dots, (\mathbf{x}_n, y_n)\}$
We calculate $MSE_i = (y_i - \hat{y}_i)^2$ and the LOOCV estimate of the test error is

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n MSE_i$$

- Advantages of LOOCV over Validation Set (VS) approach:
 - (1) Use $(n - 1)$ observations to fit a model (VS approach uses around half data)
 - (2) The LOOCV estimates do not depend on resamples
- Disadvantages: Extremely time consuming if n is large
However, for a linear model,

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2. \quad (4.1)$$

where \hat{y}_i is the fitted value from the model using all n observations.

```
Example 4.1.2# Leave-One-Out Cross-Validation
library(ISLR)
glm.fit = glm(mpg ~ horsepower, data = Auto)
coef(glm.fit)

## (Intercept) horsepower
## 39.9358610 -0.1578447

lm.fit = lm(mpg ~ horsepower, data = Auto)
coef(lm.fit)

## (Intercept) horsepower
## 39.9358610 -0.1578447

mean(((Auto$mpg - fitted(lm.fit))/(1 - influence(lm.fit)$hat))^2)

## [1] 24.23151

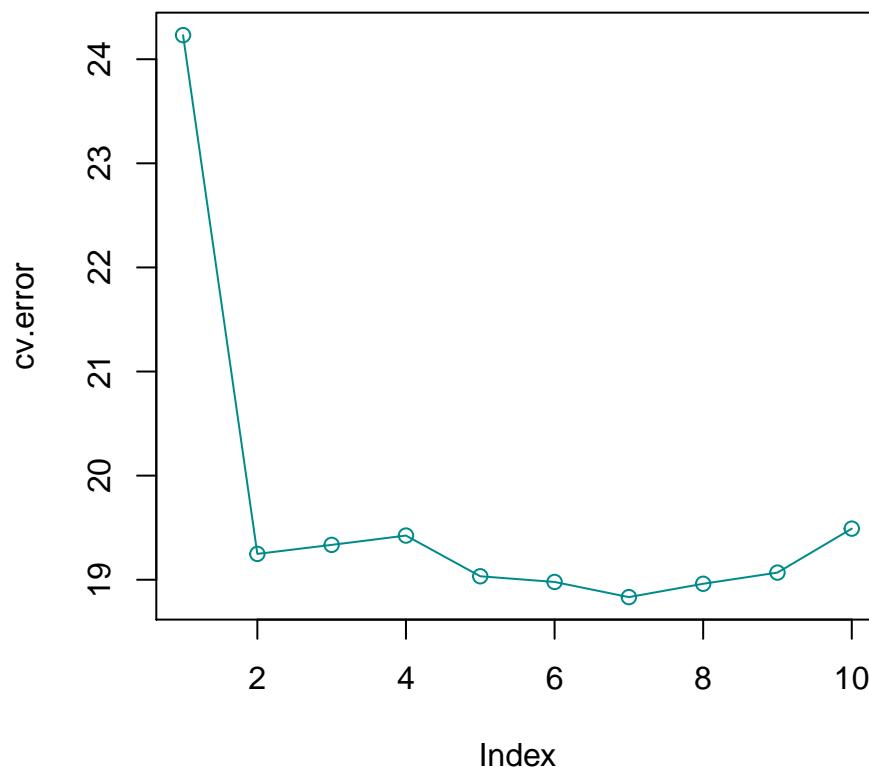
library(boot)
glm.fit = glm(mpg ~ horsepower, data = Auto)
cv.err = cv.glm(Auto, glm.fit)
cv.err$delta

## [1] 24.23151 24.23114

cv.error = rep(0, 10)
library(doMC)
registerDoMC(10)
```

```
cve <- foreach(i = 1:10) %dopar% {  
  glm.fit = glm(mpg ~ poly(horsepower, i), data = Auto)  
  cv.glm(Auto, glm.fit)$delta[1]  
}  
cv.error <- unlist(cve)  
cv.error  
  
## [1] 24.23151 19.24821 19.33498 19.42443 19.03321 18.97864 18.83305  
## [8] 18.96115 19.06863 19.49093
```

Figure 4.5: LOOCV of polynomial regression models for Auto data



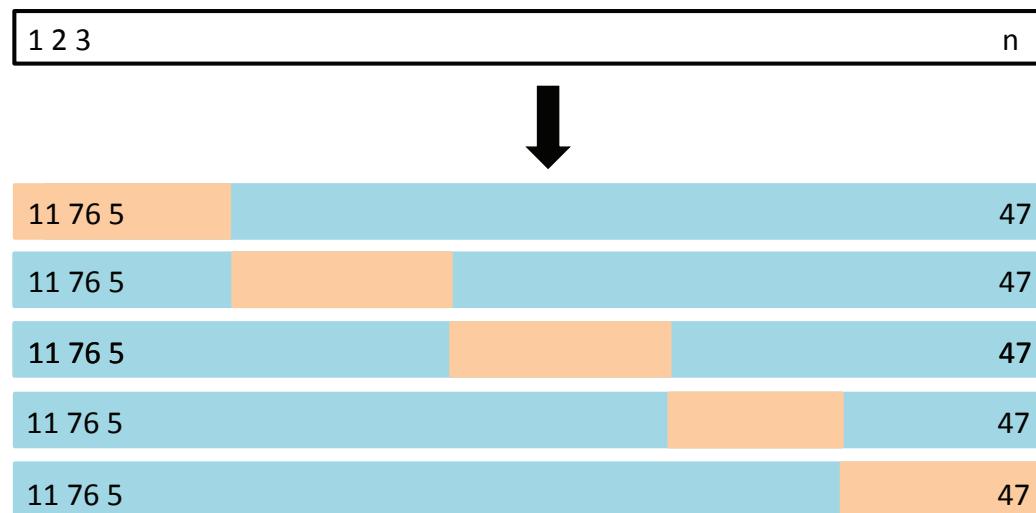
4.1.4 k -fold Cross-Validation (k -fold CV)

- Randomly divide the data set into k subsets with (almost) equal sizes
- The first fold is treated as a validation set, and the method is fit on the remaining $k - 1$ folds. The mean squared error, MSE_1 , is then computed
- Repeat this process for each subset to obtain MSE_k
- The k -fold CV estimate is computed by averaging these values,

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^k MSE_i$$

- Common choice of k is $k = 5$ or $k = 10$
- Advantages and disadvantages of k -fold CV: This approach as LOOCV uses every observation as a test observation once while VS approach does not. It requires only k fitting process compared to n fitting the data in LOOCV so that it is more computationally efficient than LOOCV. It has a bit randomness from a partition of the data, but the variance is not quite high.

Figure 4.6: A schematic display of 5-fold CV. A set of n observations is randomly split into five non-overlapping groups. Each of these fifths acts as a validation set (shown in beige), and the remainder as a training set (shown in blue). The test error is estimated by averaging the five resulting MSE estimates.



Example 4.1.3

(10-fold CV for Auto data)

```
# k-Fold Cross-Validation for linear model
set.seed(17)
k = 10
kfcv.error = rep(0, 10)
for (i in 1:k) {
  glm.fit = glm(mpg ~ poly(horsepower, i), data = Auto)
  kfcv.error[i] = cv.glm(Auto, glm.fit, K = k)$delta[1]
}
kfcv.error

## [1] 24.20520 19.18924 19.30662 19.33799 18.87911 19.02103 18.89609
## [8] 19.71201 18.95140 19.50196

plot(kfcv.error, type = "o")
# manual cv
k <- 10
n <- dim(Auto)[1]
folds <- sample(1:k, n, replace = T, prob = rep(1/k, k))
# OR
ind <- (1:n)%%k + 1
folds <- sample(ind, n)
kmse <- matrix(0, ncol = 10, nrow = k)
nd <- 10
library(doMC)
```

```
registerDoMC(10)
cvp <- foreach(i = 1:k) %dopar% {
  cvmse <- numeric(nd)
  for (j in 1:nd) {
    fit <- lm(mpg ~ poly(horsepower, j), data = Auto[folds != i, ])
    pred <- predict(fit, newdata = Auto[folds == i, ])
    cvmse[j] <- mean((pred - Auto$mpg[folds == i])^2)
  }
  cvmse
}
for (i in 1:k) kmse[i, ] <- cvp[[i]]
apply(kmse, 2, mean)

## [1] 24.25402 19.20565 19.24894 19.36283 18.95913 19.00143 18.80145
## [8] 18.89363 18.91222 19.29653
```

Figure 4.7: k-fold CV of polynomial regression models for Auto data. Left: LOOCV, Right: 10-fold CV with 10 distinct seeds.

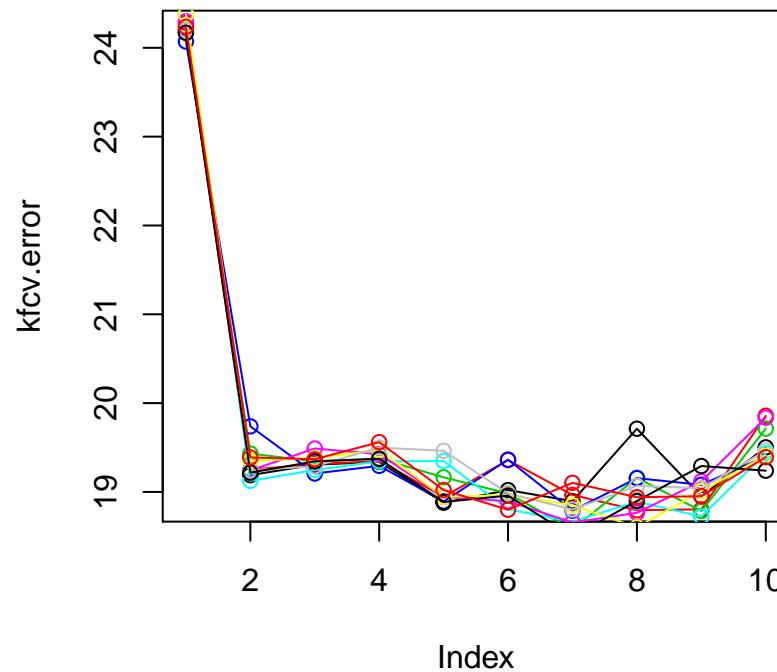
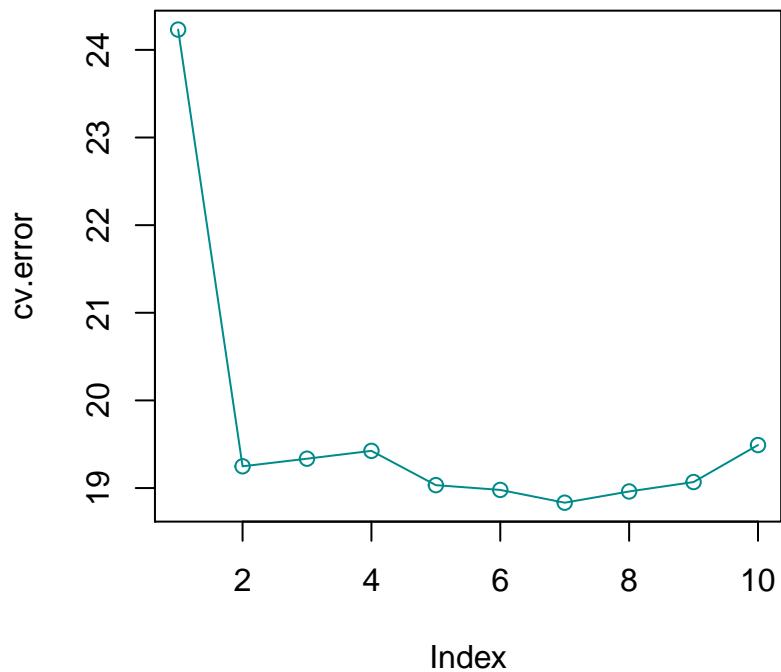
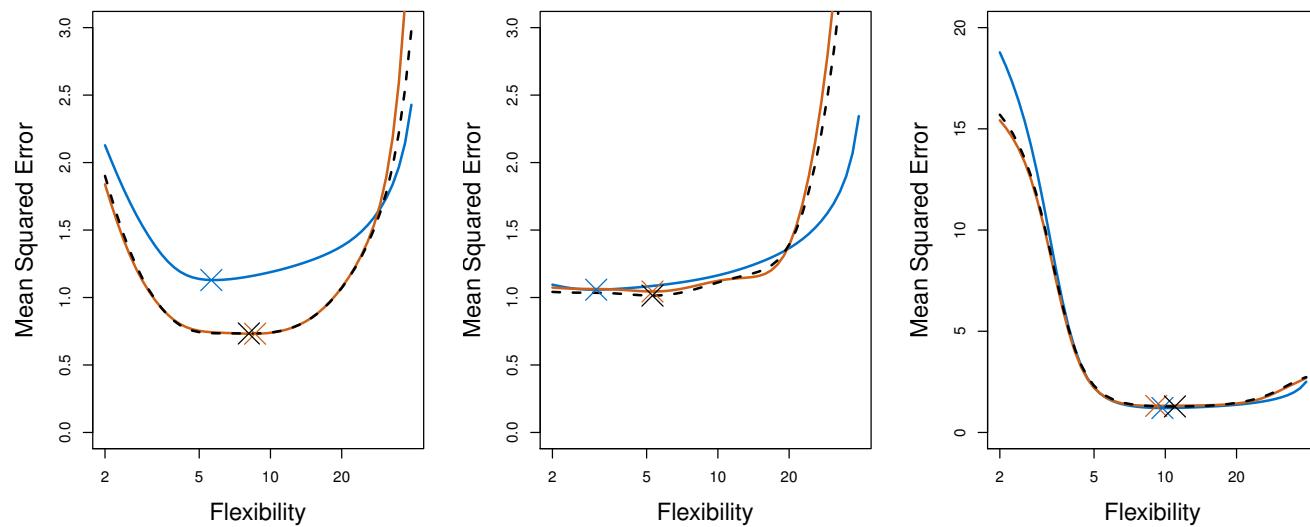


Figure 4.8: True and estimated test MSE for the simulated data sets. The true test MSE is shown in blue, the LOOCV estimate is shown as a black dashed line, and the 10-fold CV estimate is shown in orange. The crosses indicate the minimum of each of the MSE curves.



- The shapes of the true test MSE curves and the CV curves are similar.
- When we perform cross-validation, our goal might be to determine how well a given statistical learning procedure can be expected to perform on independent data; in this case, the actual estimate of the test MSE is of interest.
- But at other times we are interested only in the location of the minimum point in the estimated test MSE curve. This is because we might be performing cross-validation on a number of statistical learning methods, or on a single method using different levels of flexibility, in order to identify the method that results in the lowest test error. For this purpose, **the location of the minimum point in the estimated test MSE curve is important, but the actual value of the estimated test MSE is not**. We find in Figure 5.6 that despite the fact that they sometimes underestimate the true test MSE, all of the CV curves come close to identifying the correct level of flexibility, that is, the flexibility level corresponding to the smallest test MSE.

4.1.5 Bias-Variance Trade-Off for k-Fold Cross-Validation

- k -fold CV with $k < n$ has a computational advantage to LOOCV.
- Potentially more important advantage of k -fold CV is that it often gives more accurate estimates of the test error rate than does LOOCV. This has to do with a bias-variance trade-off.
 - LOOCV will give approximately unbiased estimates of the test error.
 - LOOCV has higher variance than does k -fold CV with $k < n$ \Leftarrow The mean of many highly correlated quantities has higher variance than does the mean of many quantities that are not as highly correlated.

4.1.6 Cross-Validation on Classification Problems

- We use the misclassification error rate instead of MSE. The LOOCV error rate takes the form

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n Err_i = \frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$$

where $Err_i = I(y_i \neq \hat{y}_i)$ is the indicator function and \hat{y}_i is the predicted classification for i th observation.

- The k -fold CV error rate and validation set error rates are defined analogously.

4.1.7 Example

As an example, we fit various logistic regression models on the two-dimensional classification data displayed in Figure 4.9. In the top-left panel of Figure 4.9, the black solid line shows the estimated decision boundary resulting from fitting a standard logistic regression model to this data set. Since this is simulated data, we can compute the true test error rate, which takes a value of 0.201 and so is substantially larger than the Bayes error rate of 0.133. Clearly logistic regression does not have enough flexibility to model the Bayes decision boundary in this setting. We can easily extend logistic regression to obtain a non-linear decision boundary by using polynomial functions of the predictors, as we did in the regression setting in Section 3.3.2.

Figure 4.9: Logistic regression fits on the two-dimensional classification data displayed. The Bayes decision boundary: purple dashed line. Estimated decision boundaries from linear, quadratic, cubic and quartic (degrees 1-4) logistic regressions: black. The test error rates for the four logistic regression fits are respectively 0.201, 0.197, 0.160, and 0.162, while the Bayes error rate is 0.133.

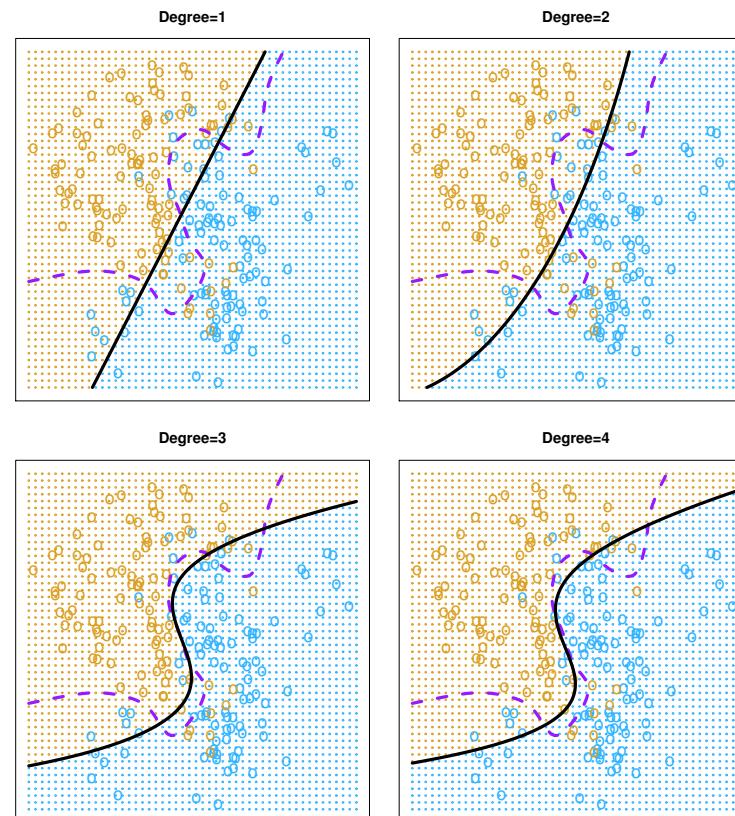
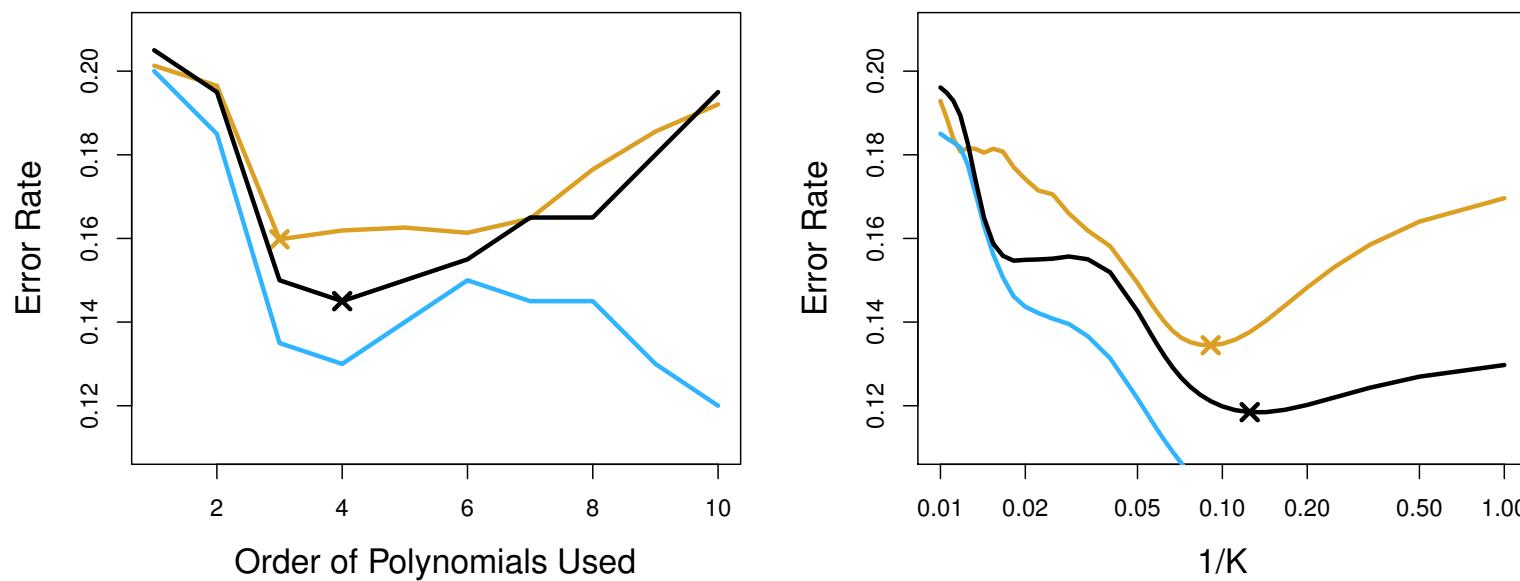


Figure 4.10: Test error (brown), training error (blue), and 10-fold CV error (black) on the two-dimensional classification data displayed. Left: Logistic regression using polynomial functions of the predictors. The order of the polynomials used is displayed on the x-axis. Right: The KNN classifier with different values of K , the number of neighbors used in the KNN classifier.



Part III

Supervised learning: Regression

Chapter 5 Linear Model Selection and Regularization

5.1 Introduction

5.1.1 Overview of linear model selection

We need better

- Prediction accuracy
- Model interpretability

Statistical methods in this chapter

- Subset selection
- Shrinkage
- Dimension reduction

5.2 Subset selection

5.2.1 Best subset selection

1. Let \mathcal{M}_0 denote the null model, which contains no predictors.
2. For $k = 1, 2, \dots, p$:
 - a. Fit all $\binom{p}{k}$ models that contain exactly k predictors.
 - b. Pick the best among these $\binom{p}{k}$ models, and call it \mathcal{M}_k . Here *best* is defined as having the smallest RSS , or largest R^2 .
3. Select a single best model from among $\mathcal{M}_0, \dots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC , or R^2_{adj} .

Disadvantage: When p is large, the exhaustive search is not feasible.

Example 5.2.1

Hitters data.

```
# Best Subset Selection

library(ISLR)
names(Hitters)

## [1] "AtBat"     "Hits"       "HmRun"      "Runs"       "RBI"
## [6] "Walks"      "Years"      "CAtBat"     "CHits"      "CHmRun"
```

```
## [11] "CRuns"      "CRBI"       "CWalks"     "League"      "Division"
## [16] "PutOuts"     "Assists"    "Errors"     "Salary"      "NewLeague"

dim(Hitters)

## [1] 322 20

sum(is.na(Hitters$Salary))

## [1] 59

Hitters = na.omit(Hitters)
dim(Hitters)

## [1] 263 20

sum(is.na(Hitters))

## [1] 0

library(leaps)
regfit.full = regsubsets(Salary ~ ., Hitters)
summary(regfit.full)

## Subset selection object
## Call: regsubsets.formula(Salary ~ ., Hitters)
## 19 Variables (and intercept)
##          Forced in Forced out
```

```

## AtBat      FALSE  FALSE
## Hits       FALSE  FALSE
## HmRun      FALSE  FALSE
## Runs       FALSE  FALSE
## RBI        FALSE  FALSE
## Walks      FALSE  FALSE
## Years      FALSE  FALSE
## CAtBat     FALSE  FALSE
## CHits      FALSE  FALSE
## CHmRun     FALSE  FALSE
## CRuns      FALSE  FALSE
## CRBI       FALSE  FALSE
## CWalks     FALSE  FALSE
## LeagueN    FALSE  FALSE
## DivisionW  FALSE  FALSE
## PutOuts    FALSE  FALSE
## Assists    FALSE  FALSE
## Errors     FALSE  FALSE
## NewLeagueN FALSE  FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns
## 1  ( 1 ) " "   " "   " "   " "   " "   " "   " "   " "   " "   " "   " "
## 2  ( 1 ) " "   "*"  " "   " "   " "   " "   " "   " "   " "   " "   " "
## 3  ( 1 ) " "   "*"  " "   " "   " "   " "   " "   " "   " "   " "   " "
## 4  ( 1 ) " "   "*"  " "   " "   " "   " "   " "   " "   " "   " "   " "

```

```

## 5 ( 1 ) "*"   "*"   " "   " "   " "   " "   " "   " "
## 6 ( 1 ) "*"   "*"   " "   " "   " "   " "   "*"   " "   " "
## 7 ( 1 ) " "   "*"   " "   " "   " "   " "   "*"   "*"   " "
## 8 ( 1 ) "*"   "*"   " "   " "   " "   " "   "*"   " "   "*"
##          CRBI  CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1 ( 1 ) "*"   " "   " "   " "   " "   " "   " "   " "
## 2 ( 1 ) "*"   " "   " "   " "   " "   " "   " "   " "
## 3 ( 1 ) "*"   " "   " "   " "   " "   "*"   " "   " "
## 4 ( 1 ) "*"   " "   " "   " "   "*"   "*"   " "   " "
## 5 ( 1 ) "*"   " "   " "   " "   "*"   "*"   " "   " "
## 6 ( 1 ) "*"   " "   " "   " "   "*"   "*"   " "   " "
## 7 ( 1 ) " "   " "   " "   " "   "*"   "*"   " "   " "
## 8 ( 1 ) " "   "*"   " "   " "   "*"   "*"   " "   " "

regfit.full = regsubsets(Salary ~ ., data = Hitters, nvmax = 19)
reg.summary = summary(regfit.full)
names(reg.summary)

## [1] "which"   "rsq"     "rss"      "adjr2"    "cp"       "bic"      "outmat"   "obj"

reg.summary$rsq

## [1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227
## [8] 0.5285569 0.5346124 0.5404950 0.5426153 0.5436302 0.5444570 0.5452164
## [15] 0.5454692 0.5457656 0.5459518 0.5460945 0.5461159

# par(mfrow=c(2,2)) plot(reg.summary$rss,xlab='Number of

```

```
# Variables',ylab='RSS',type='l') plot(reg.summary$adjr2,xlab='Number of  
# Variables',ylab='Adjusted RSq',type='l')  
which.max(reg.summary$adjr2)  
  
## [1] 11  
  
# points(11,reg.summary$adjr2[11], col='red',cex=2,pch=20)  
# plot(reg.summary$cp,xlab='Number of Variables',ylab='Cp',type='l')  
which.min(reg.summary$cp)  
  
## [1] 10  
  
# points(10,reg.summary$cp[10],col='red',cex=2,pch=20)  
which.min(reg.summary$bic)  
  
## [1] 6  
  
# plot(reg.summary$bic,xlab='Number of Variables',ylab='BIC',type='l')  
# points(6,reg.summary$bic[6],col='red',cex=2,pch=20)  
# plot(regfit.full,scale='r2') plot(regfit.full,scale='adjr2')  
# plot(regfit.full,scale='Cp') plot(regfit.full,scale='bic')  
coef(regfit.full, 6)  
  
## (Intercept) AtBat Hits Walks CRBI  
## 91.5117981 -1.8685892 7.6043976 3.6976468 0.6430169  
## DivisionW PutOuts  
## -122.9515338 0.2643076
```

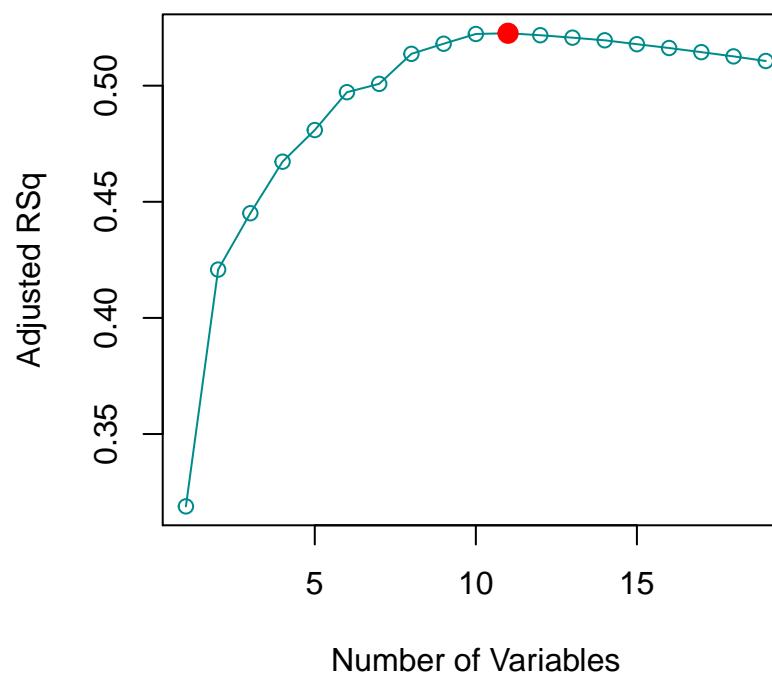
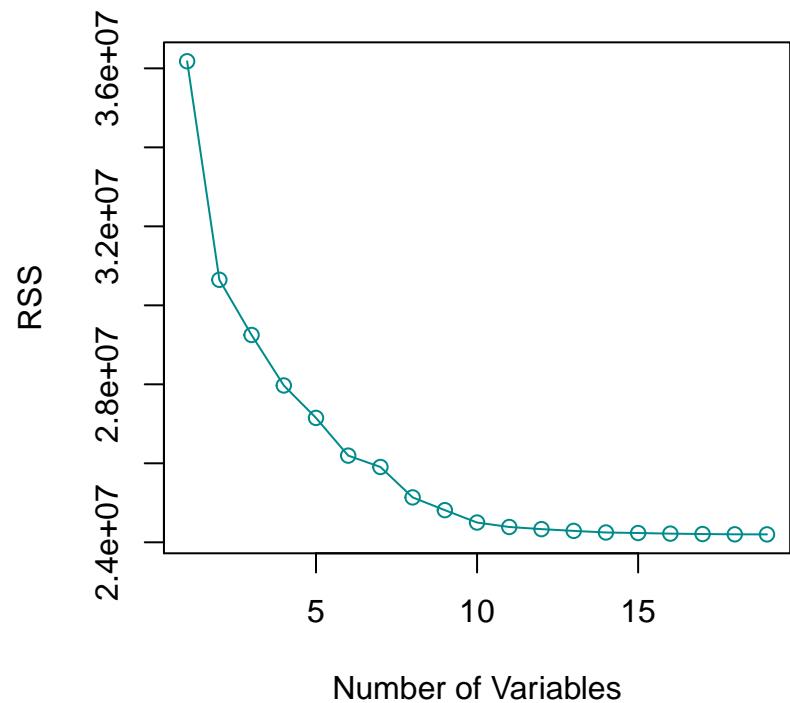
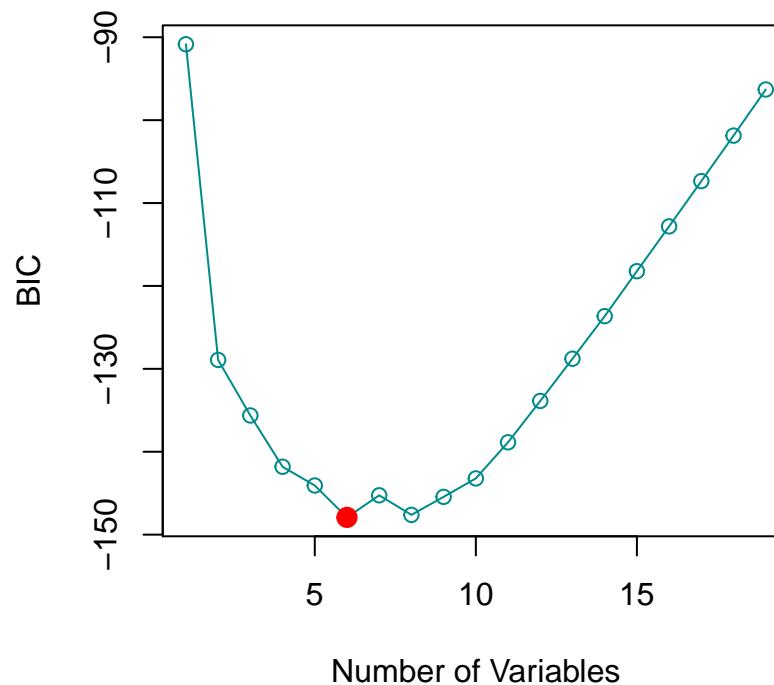
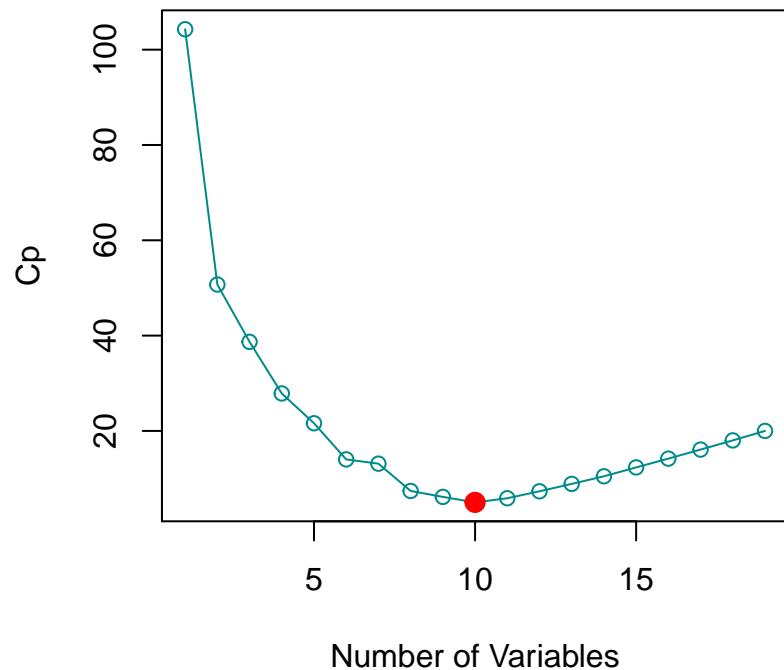
Figure 5.1: RSS and adjusted R^2 

Figure 5.2: C_p and adjusted BIC

5.2.2 Forward stepwise selection

1. Let \mathcal{M}_0 denote the null model, which contains no predictors.
2. For $k = 0, \dots, p - 1$:
 - a. Consider all $p - k$ models that augment the predictors in M_k with one additional predictor.
 - b. Choose the best among these $p - k$ models, and call it M_{k+1} . Here best is defined as having smallest RSS or highest R^2 .
3. Select a single best model from among M_0, \dots, M_p using cross-validated prediction error, $Cp(AIC)$, BIC , or R^2_{adj} .

5.2.3 Backward stepwise selection

1. Let \mathcal{M}_p denote the full model, which contains all p predictors.
2. For $k = p, p-1, \dots, 1$:
 - a. Consider all k models that contains all but one of the predictors in M_k , for a total $k-1$ predictors.
 - b. Choose the best among these k models, and call it M_{k-1} . Here best is defined as having smallest RSS or highest R^2 .
3. Select a single best model from among M_0, \dots, M_p using cross-validated prediction error, $Cp(AIC)$, BIC , or R^2_{adj} .

Example 5.2.2

(FS and BE for Hitters data)

```
# Forward and Backward Stepwise Selection
regfit.fwd = regsubsets(Salary ~ ., data = Hitters, nvmax = 19, method = "forward")
summary(regfit.fwd)

## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "forward")
## 19 Variables (and intercept)
##          Forced in Forced out
## AtBat      FALSE      FALSE
## Hits       FALSE      FALSE
## HmRun      FALSE      FALSE
```

```

## Runs          FALSE   FALSE
## RBI          FALSE   FALSE
## Walks         FALSE   FALSE
## Years         FALSE   FALSE
## CAtBat        FALSE   FALSE
## CHits         FALSE   FALSE
## CHmRun        FALSE   FALSE
## CRuns         FALSE   FALSE
## CRBI          FALSE   FALSE
## CWalks        FALSE   FALSE
## LeagueN       FALSE   FALSE
## DivisionW     FALSE   FALSE
## PutOuts        FALSE   FALSE
## Assists        FALSE   FALSE
## Errors         FALSE   FALSE
## NewLeagueN    FALSE   FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: forward
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns
## 1 ( 1 )   " "   " "   " "   " "   " "   " "   " "   " "   " "   " "
## 2 ( 1 )   " "   "*"  " "   " "   " "   " "   " "   " "   " "   " "
## 3 ( 1 )   " "   "*"  " "   " "   " "   " "   " "   " "   " "   " "
## 4 ( 1 )   " "   "*"  " "   " "   " "   " "   " "   " "   " "   " "
## 5 ( 1 )   "*"  "*"  " "   " "   " "   " "   " "   " "   " "   " "
## 6 ( 1 )   "*"  "*"  " "   " "   " "   " "   "*"  " "   " "   " "
## 7 ( 1 )   "*"  "*"  " "   " "   " "   " "   "*"  " "   " "   " "

```

```

## 8 ( 1 ) "★" "★" " " " " " " ★" " " " " " " " " ★"
## 9 ( 1 ) "★" "★" " " " " " " ★" " " " ★" " " " " " " ★"
## 10 ( 1 ) "★" "★" " " " " " " ★" " " " ★" " " " " " " ★"
## 11 ( 1 ) "★" "★" " " " " " " ★" " " " ★" " " " " " " ★"
## 12 ( 1 ) "★" "★" " " " ★" " " " ★" " " " ★" " " " " " " ★"
## 13 ( 1 ) "★" "★" " " " ★" " " " ★" " " " ★" " " " " " " ★"
## 14 ( 1 ) "★" "★" "★" "★" " " " ★" " " " ★" " " " " " " ★"
## 15 ( 1 ) "★" "★" "★" "★" " " " ★" " " " ★" " " " " " " ★"
## 16 ( 1 ) "★" "★" "★" "★" "★" "★" " " " ★" " " " ★" " " " " ★"
## 17 ( 1 ) "★" "★" "★" "★" "★" "★" " " " ★" " " " ★" " " " " ★"
## 18 ( 1 ) "★" "★" "★" "★" "★" "★" " " " ★" " " " ★" " " " " ★"
## 19 ( 1 ) "★" "★" "★" "★" "★" "★" " " " ★" " " " ★" " " " " ★"

##          CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1 ( 1 ) "★" " " " " " " " "
## 2 ( 1 ) "★" " " " " " " " "
## 3 ( 1 ) "★" " " " " " ★" " " " " " "
## 4 ( 1 ) "★" " " " " ★" " ★" " " " " " "
## 5 ( 1 ) "★" " " " " ★" " ★" " " " " " "
## 6 ( 1 ) "★" " " " " ★" " ★" " " " " " "
## 7 ( 1 ) "★" "★" " " " " ★" " ★" " " " " "
## 8 ( 1 ) "★" "★" " " " " ★" " ★" " " " " "
## 9 ( 1 ) "★" "★" " " " " ★" " ★" " " " " "
## 10 ( 1 ) "★" "★" " " " " ★" " ★" " " " " "
## 11 ( 1 ) "★" "★" "★" " " " ★" " ★" " " " "
## 12 ( 1 ) "★" "★" "★" " " " ★" " ★" " " " "
## 13 ( 1 ) "★" "★" "★" " " " ★" " ★" " " " "

```

```
## 14 ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" " "
## 15 ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" " "
## 16 ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" " "
## 17 ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "
## 18 ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "
## 19 ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "

regfit.bwd = regsubsets(Salary ~ ., data = Hitters, nvmax = 19, method = "backward")
summary(regfit.bwd)

## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "backward")
## 19 Variables (and intercept)
##          Forced in Forced out
## AtBat      FALSE    FALSE
## Hits       FALSE    FALSE
## HmRun      FALSE    FALSE
## Runs       FALSE    FALSE
## RBI        FALSE    FALSE
## Walks      FALSE    FALSE
## Years      FALSE    FALSE
## CAtBat     FALSE    FALSE
## CHits      FALSE    FALSE
## CHmRun     FALSE    FALSE
## CRuns      FALSE    FALSE
## CRBI       FALSE    FALSE
## CWalks     FALSE    FALSE
```

```

## LeagueN      FALSE    FALSE
## DivisionW    FALSE    FALSE
## PutOuts      FALSE    FALSE
## Assists      FALSE    FALSE
## Errors       FALSE    FALSE
## NewLeagueN   FALSE    FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: backward
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns
## 1 ( 1 )   " "   " "   " "   " "   " "   " "   " "   " "   " "   " "   " "   " * "
## 2 ( 1 )   " "   " * "   " "   " "   " "   " "   " "   " "   " "   " "   " "   " * "
## 3 ( 1 )   " "   " * "   " "   " "   " "   " "   " "   " "   " "   " "   " "   " * "
## 4 ( 1 )   " * "   " * "   " "   " "   " "   " "   " "   " "   " "   " "   " "   " * "
## 5 ( 1 )   " * "   " * "   " "   " "   " "   " * "   " "   " "   " "   " "   " "   " * "
## 6 ( 1 )   " * "   " * "   " "   " "   " "   " "   " * "   " "   " "   " "   " "   " * "
## 7 ( 1 )   " * "   " * "   " "   " "   " "   " "   " "   " * "   " "   " "   " "   " * "
## 8 ( 1 )   " * "   " * "   " "   " "   " "   " "   " "   " * "   " "   " "   " "   " * "
## 9 ( 1 )   " * "   " * "   " "   " "   " "   " "   " "   " "   " * "   " "   " "   " * "
## 10 ( 1 )  " * "   " * "   " "   " "   " "   " "   " "   " "   " * "   " "   " "   " * "
## 11 ( 1 )  " * "   " * "   " "   " "   " "   " "   " "   " "   " * "   " "   " "   " * "
## 12 ( 1 )  " * "   " * "   " "   " * "   " "   " * "   " "   " "   " * "   " "   " "   " * "
## 13 ( 1 )  " * "   " * "   " "   " * "   " "   " * "   " "   " * "   " "   " "   " "   " * "
## 14 ( 1 )  " * "   " * "   " * "   " "   " * "   " "   " * "   " "   " * "   " "   " "   " * "
## 15 ( 1 )  " * "   " * "   " * "   " * "   " "   " * "   " "   " * "   " * "   " "   " "   " * "
## 16 ( 1 )  " * "   " * "   " * "   " * "   " * "   " "   " * "   " * "   " * "   " "   " "   " * "
## 17 ( 1 )  " * "   " * "   " * "   " * "   " * "   " * "   " "   " * "   " * "   " "   " "   " * "

```

```

## 18 ( 1 ) "*"   "*"   "*"   "*"   "*"   "*"   "*"   "*"   "*"   "*"   " "   "*"
## 19 ( 1 ) "*"   "*"   "*"   "*"   "*"   "*"   "*"   "*"   "*"   "*"   "*"   "*"   "*"
##          CRBI  CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1 ( 1 ) " "   " "   " "   " "   " "   " "   " "   " "   " "   " "
## 2 ( 1 ) " "   " "   " "   " "   " "   " "   " "   " "   " "   " "
## 3 ( 1 ) " "   " "   " "   " "   "*"   " "   " "   " "   " "   " "
## 4 ( 1 ) " "   " "   " "   " "   "*"   " "   " "   " "   " "   " "
## 5 ( 1 ) " "   " "   " "   " "   "*"   " "   " "   " "   " "   " "
## 6 ( 1 ) " "   " "   " "   "*"   " "   " "   " "   " "   " "   " "
## 7 ( 1 ) " "   "*"   " "   "*"   " "   "*"   " "   " "   " "   " "
## 8 ( 1 ) "*"   "*"   " "   "*"   " "   "*"   " "   " "   " "   " "
## 9 ( 1 ) "*"   "*"   " "   "*"   " "   "*"   " "   " "   " "   " "
## 10 ( 1 ) "*"  "*"   " "   "*"   " "   "*"   " "   "*"   " "   " "
## 11 ( 1 ) "*"  "*"   "*"   " "   "*"   " "   "*"   " "   " "   " "
## 12 ( 1 ) "*"  "*"   "*"   "*"   " "   "*"   " "   "*"   " "   " "
## 13 ( 1 ) "*"  "*"   "*"   "*"   "*"   " "   "*"   " "   "*"   " "
## 14 ( 1 ) "*"  "*"   "*"   "*"   "*"   "*"   " "   "*"   " "   " "
## 15 ( 1 ) "*"  "*"   "*"   "*"   "*"   "*"   " "   "*"   " "   " "
## 16 ( 1 ) "*"  "*"   "*"   "*"   "*"   "*"   " "   "*"   " "   " "
## 17 ( 1 ) "*"  "*"   "*"   "*"   "*"   "*"   " "   "*"   " "   "*"
## 18 ( 1 ) "*"  "*"   "*"   "*"   "*"   "*"   " "   "*"   " "   "*"
## 19 ( 1 ) "*"  "*"   "*"   "*"   "*"   "*"   " "   "*"   " "   "*"

coef(regfit.full, 7)

## (Intercept)           Hits          Walks         CATBat        CHits
## 79.4509472    1.2833513   3.2274264   -0.3752350   1.4957073

```

```
##      CHmRun    DivisionW     PutOuts
## 1.4420538 -129.9866432   0.2366813

coef(regfit.fwd, 7)

## (Intercept)      AtBat       Hits       Walks       CRBI
## 109.7873062   -1.9588851   7.4498772   4.9131401   0.8537622
##      CWalks    DivisionW     PutOuts
##  -0.3053070  -127.1223928   0.2533404

coef(regfit.bwd, 7)

## (Intercept)      AtBat       Hits       Walks       CRuns
## 105.6487488   -1.9762838   6.7574914   6.0558691   1.1293095
##      CWalks    DivisionW     PutOuts
##  -0.7163346  -116.1692169   0.3028847
```

5.2.4 Choosing the optimal model

- (Determining the *best* model)
 - Indirectly estimate test error by an adjustment to the training error: $C_p(AIC)$, BIC , or R_{adj}^2 .
 - Directly estimate using a validation set approach or cross-validation approach.
- (Model selection by criteria)
 - Mallows $C_p = \frac{RSS}{\hat{\sigma}^2} + 2d - n$
 - $AIC = -2 \log(lik) + 2d = n \log(RSS/n) + 2d$
 - $BIC = -2 \log(lik) + d \log n = n \log(RSS/n) + d \log n$
 - $R_{adj}^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)}$
- (Validation set or cross-validation approaches)
 - We estimate the test error and select the model for which the resulting estimated test error is smallest.
 - The one-standard-error rule: If we can repeat the validation set or cross-validation approach, then we can estimate the s.e. of the test MSE for each model size. We select the smallest model for which the estimated test error is within one standard error of the lowest point on the curve.

Example 5.2.3

Hitters data

```
# Choosing Among Models
set.seed(1)
train = sample(c(TRUE, FALSE), nrow(Hitters), rep = TRUE)
```

```
test = (!train)
regfit.best = regsubsets(Salary ~ ., data = Hitters[train, ], nvmax = 19)
test.mat = model.matrix(Salary ~ ., data = Hitters[test, ])
val.errors = rep(NA, 19)
for (i in 1:19) {
  coefi = coef(regfit.best, id = i)
  pred = test.mat[, names(coefi)] %*% coefi
  val.errors[i] = mean((Hitters$Salary[test] - pred)^2)
}
val.errors

## [1] 220968.0 169157.1 178518.2 163426.1 168418.1 171270.6 162377.1
## [8] 157909.3 154055.7 148162.1 151156.4 151742.5 152214.5 157358.7
## [15] 158541.4 158743.3 159972.7 159859.8 160105.6

which.min(val.errors)

## [1] 10

coef(regfit.best, 10)

## (Intercept)      AtBat      Hits      Walks      CATBat      CHits
## -80.2751499   -1.4683816    7.1625314   3.6430345   -0.1855698   1.1053238
##      CHmRun      CWalks     LeagueN   DivisionW      PutOuts
##    1.3844863   -0.7483170   84.5576103  -53.0289658    0.2381662

predict.regsubsets = function(object, newdata, id, ...) {
```

```
form = as.formula(object$call[[2]])
mat = model.matrix(form, newdata)
coefi = coef(object, id = id)
xvars = names(coefi)
mat[, xvars] %*% coefi
}
regfit.best = regsubsets(Salary ~ ., data = Hitters, nvmax = 19)
coef(regfit.best, 10)

## (Intercept)          AtBat          Hits          Walks         CAtBat
## 162.5354420     -2.1686501      6.9180175      5.7732246     -0.1300798
##        CRuns          CRBI          CWalks       DivisionW        PutOuts
##     1.4082490      0.7743122     -0.8308264    -112.3800575      0.2973726
##        Assists
##     0.2831680

k = 10
set.seed(1)
folds = sample(1:k, nrow(Hitters), replace = TRUE)
cv.errors = matrix(NA, k, 19, dimnames = list(NULL, paste(1:19)))
for (j in 1:k) {
  best.fit = regsubsets(Salary ~ ., data = Hitters[folds != j, ], nvmax = 19)
  for (i in 1:19) {
    pred = predict(best.fit, Hitters[folds == j, ], id = i)
    cv.errors[j, i] = mean((Hitters$Salary[folds == j] - pred)^2)
  }
}
```

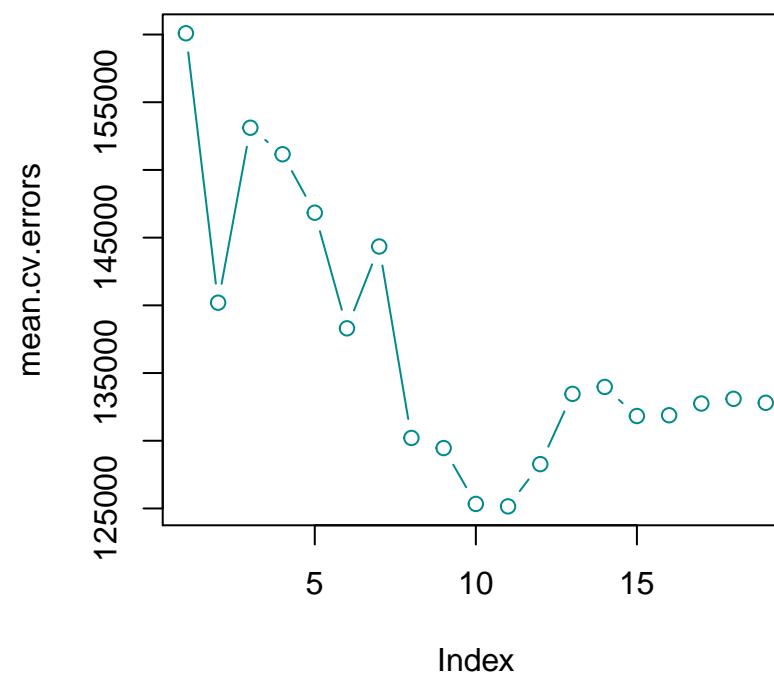
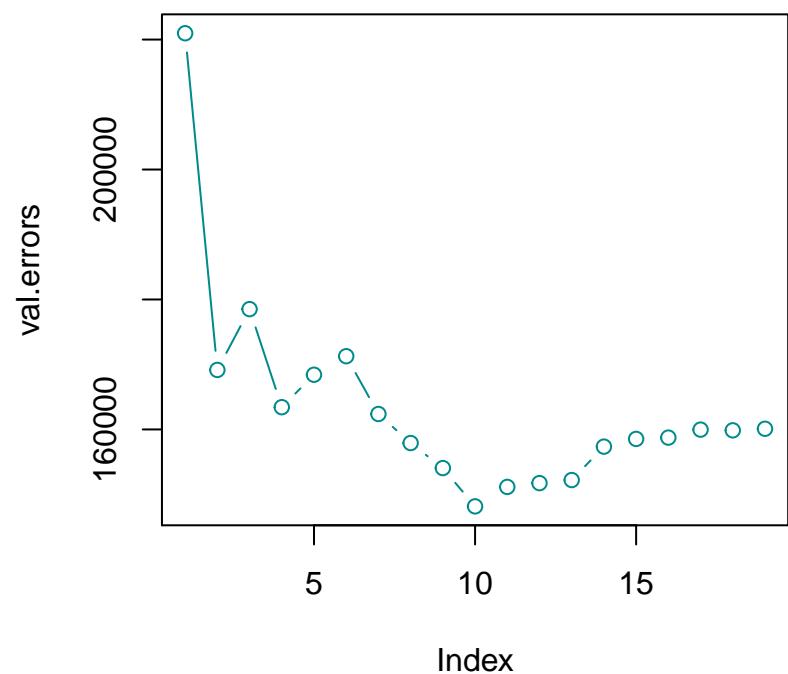
```
mean.cv.errors = apply(cv.errors, 2, mean)
mean.cv.errors

##      1      2      3      4      5      6      7      8
## 160093.5 140196.8 153117.0 151159.3 146841.3 138302.6 144346.2 130207.7
##      9     10     11     12     13     14     15     16
## 129459.6 125334.7 125153.8 128273.5 133461.0 133974.6 131825.7 131882.8
##     17     18     19
## 132750.9 133096.2 132804.7

# par(mfrow=c(1,1)) plot(mean.cv.errors,type='b')
reg.best = regsubsets(Salary ~ ., data = Hitters, nvmax = 19)
coef(reg.best, 11)

## (Intercept)      AtBat      Hits      Walks      CAtBat
## 135.7512195 -2.1277482  6.9236994  5.6202755 -0.1389914
##      CRuns      CRBI      CWalks      LeagueN      DivisionW
## 1.4553310   0.7852528 -0.8228559  43.1116152 -111.1460252
##      PutOuts      Assists
## 0.2894087   0.2688277
```

Figure 5.3: Validation error and k-fold CV



5.3 Shrinkage Methods

5.3.1 Ridge regression

- We minimize

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 \quad (5.1)$$

- $\lambda \sum_{j=1}^p \beta_j^2$ is called the shrinkage penalty term
- λ is called the tuning parameter
- $\lambda = 0$: $\hat{\beta}_{Ridge} = \hat{\beta}_{LSE}$
- $\lambda \rightarrow \infty$: $\hat{\beta}_{Ridge} \rightarrow 0$
- The predictor should be standardized before applying Ridge regression.

Example 5.3.1

Ridge regression for Hitters data

```
# Ridge Regression
Hitters = na.omit(Hitters)
x = model.matrix(Salary ~ ., Hitters)[, -1]
y = Hitters$Salary
library(glmnet)
```

```
## Loading required package: Matrix
## Loaded glmnet 2.0-16

grid = 10^seq(10, -2, length = 100)
ridge.mod = glmnet(x, y, alpha = 0, lambda = grid)
dim(coef(ridge.mod))

## [1] 20 100

ridge.mod$lambda[50]

## [1] 11497.57

coef(ridge.mod)[, 50]

##   (Intercept)      AtBat      Hits      HmRun      Runs
## 407.356050200  0.036957182  0.138180344  0.524629976  0.230701523
##          RBI      Walks      Years      CAtBat      CHits
## 0.239841459  0.289618741  1.107702929  0.003131815  0.011653637
##         CHmRun      CRuns      CRBI      CWalks      LeagueN
## 0.087545670  0.023379882  0.024138320  0.025015421  0.085028114
##     DivisionW      PutOuts      Assists      Errors      NewLeagueN
## -6.215440973  0.016482577  0.002612988 -0.020502690  0.301433531

sqrt(sum(coef(ridge.mod)[-1, 50]^2))

## [1] 6.360612

ridge.mod$lambda[60]
```

```

## [1] 705.4802

coef(ridge.mod)[, 60]

## (Intercept)      AtBat      Hits      HmRun      Runs
## 54.32519950  0.11211115  0.65622409  1.17980910  0.93769713
##          RBI      Walks      Years     CAtBat      CHits
##  0.84718546  1.31987948  2.59640425  0.01083413  0.04674557
##         CHmRun     CRuns      CRBI      CWalks      LeagueN
##  0.33777318  0.09355528  0.09780402  0.07189612 13.68370191
##      DivisionW     PutOuts     Assists     Errors NewLeagueN
## -54.65877750  0.11852289  0.01606037 -0.70358655  8.61181213

sqrt(sum(coef(ridge.mod)[-1, 60]^2))

## [1] 57.11001

predict(ridge.mod, s = 50, type = "coefficients")[1:20, ]

## (Intercept)      AtBat      Hits      HmRun      Runs
## 4.876610e+01 -3.580999e-01  1.969359e+00 -1.278248e+00  1.145892e+00
##          RBI      Walks      Years     CAtBat      CHits
## 8.038292e-01  2.716186e+00 -6.218319e+00  5.447837e-03  1.064895e-01
##         CHmRun     CRuns      CRBI      CWalks      LeagueN
## 6.244860e-01  2.214985e-01  2.186914e-01 -1.500245e-01  4.592589e+01
##      DivisionW     PutOuts     Assists     Errors NewLeagueN
## -1.182011e+02  2.502322e-01  1.215665e-01 -3.278600e+00 -9.496680e+00

```

```
set.seed(1)
train = sample(1:nrow(x), nrow(x)/2)
test = (-train)
y.test = y[test]
ridge.mod = glmnet(x[train, ], y[train], alpha = 0, lambda = grid, thresh = 1e-12)
ridge.pred = predict(ridge.mod, s = 4, newx = x[test, ])
mean((ridge.pred - y.test)^2)

## [1] 101036.8

mean((mean(y[train]) - y.test)^2)

## [1] 193253.1

ridge.pred = predict(ridge.mod, s = 1e+10, newx = x[test, ])
mean((ridge.pred - y.test)^2)

## [1] 193253.1

ridge.pred = predict(ridge.mod, s = 0, newx = x[test, ])
mean((ridge.pred - y.test)^2)

## [1] 114723.6

lm(y ~ x, subset = train)

##
## Call:
```

```

## lm(formula = y ~ x, subset = train)
##
## Coefficients:
## (Intercept)      xAtBat      xHits      xHmRun      xRuns
## 299.42849     -2.54027     8.36682    11.64512     -9.09923
##          xRBI      xWalks      xYears      xCAtBat      xCHits
## 2.44105       9.23440   -22.93673    -0.18154     -0.11598
##          xCHmRun     xCRuns      xCRBI      xCWalks      xLeagueN
##      -1.33888     3.32838     0.07536    -1.07841     59.76065
##      xDivisionW     xPutOuts     xAssists     xErrors  xNewLeagueN
##     -98.86233     0.34087     0.34165    -0.64207     -0.67442

predict(ridge.mod, s = 0, type = "coefficients")[1:20, ]

## (Intercept)      AtBat      Hits      HmRun      Runs
## 299.44467220  -2.53538355  8.33585019  11.59830815  -9.05971371
##          RBI      Walks      Years      CAtBat      CHits
## 2.45326546    9.21776006 -22.98239583  -0.18191651  -0.10565688
##          CHmRun     CRuns      CRBI      CWalks      LeagueN
##      -1.31721358  3.31152519   0.06590689  -1.07244477  59.75587273
##      DivisionW     PutOuts     Assists     Errors  NewLeagueN
##     -98.94393005  0.34083276   0.34155445  -0.65312471  -0.65882930

set.seed(1)
cv.out = cv.glmnet(x[train, ], y[train], alpha = 0)
# plot(cv.out)
bestlam = cv.out$lambda.min
bestlam

```

```
## [1] 211.7416

ridge.pred = predict(ridge.mod, s = bestlam, newx = x[test, ])
mean((ridge.pred - y.test)^2)

## [1] 96015.51

out = glmnet(x, y, alpha = 0)
predict(out, type = "coefficients", s = bestlam)[1:20, ]

## (Intercept)          AtBat          Hits         HmRun         Runs
##  9.88487157   0.03143991   1.00882875   0.13927624   1.11320781
##          RBI          Walks          Years        CAtBat        CHits
##  0.87318990   1.80410229   0.13074383   0.011113978  0.06489843
##          CHmRun         CRuns          CRBI         CWalks       LeagueN
##  0.45158546   0.12900049   0.13737712   0.02908572  27.18227527
##      DivisionW         PutOuts         Assists        Errors     NewLeagueN
## -91.63411282   0.19149252   0.04254536  -1.81244470   7.21208394
```

5.3.2 Least Absolute Shrinkage and Selection Operator (LASSO)

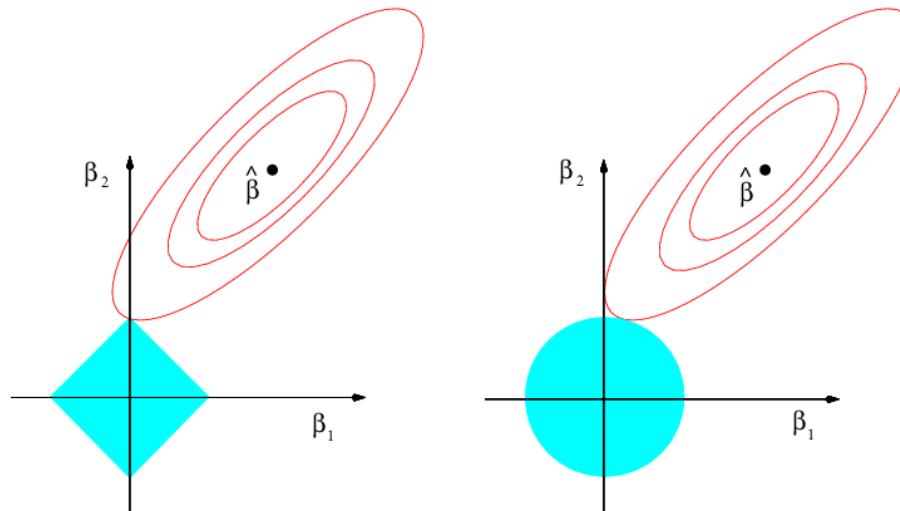
- We minimize

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| \quad (5.2)$$

- It forces some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large. → The lasso performs variable selection.
- $\lambda = 0$: $\hat{\beta}_{LASSO} = \hat{\beta}_{LSE}$
- $\lambda \rightarrow \infty$: $\hat{\beta}_{LASSO} \rightarrow 0$
- The predictor should be standardized before applying LASSO.

Geometry of LASSO and ridge regression

Figure 5.4: Contours of the error and constraint functions for the lasso (left) and ridge regression (right).



Selecting the Tuning Parameter

- Choose a grid of λ values, and compute the cross-validation error for each value of λ .
- Select the tuning parameter value for which the cross-validation error is smallest.

Example 5.3.2

LASSO for Hitters data

```
# The Lasso
lasso.mod = glmnet(x[train, ], y[train], alpha = 1, lambda = grid)
plot(lasso.mod)
set.seed(1)
cv.out = cv.glmnet(x[train, ], y[train], alpha = 1)
# plot(cv.out)
bestlam = cv.out$lambda.min
lasso.pred = predict(lasso.mod, s = bestlam, newx = x[test, ])
mean((lasso.pred - y.test)^2)

## [1] 100743.4

out = glmnet(x, y, alpha = 1, lambda = grid)
lasso.coef = predict(out, type = "coefficients", s = bestlam)[1:20, ]
lasso.coef

## (Intercept)      AtBat      Hits      HmRun      Runs
## 18.5394844  0.0000000  1.8735390  0.0000000  0.0000000
##       RBI      Walks     Years     CATBat      CHits
```

```
## 0.0000000 2.2178444 0.0000000 0.0000000 0.0000000
## CHmRun CRuns CRBI CWalks LeagueN
## 0.0000000 0.2071252 0.4130132 0.0000000 3.2666677
## DivisionW PutOuts Assists Errors NewLeagueN
## -103.4845458 0.2204284 0.0000000 0.0000000 0.0000000

lasso.coef[lasso.coef != 0]

## (Intercept) Hits Walks CRuns CRBI
## 18.5394844 1.8735390 2.2178444 0.2071252 0.4130132
## LeagueN DivisionW PutOuts
## 3.2666677 -103.4845458 0.2204284
```

5.4 Dimension reduction methods

5.4.1 Outline of dimension reduction methods

- Transform the predictors and then fit a least squares model using the transformed variables.
- Let Z_1, \dots, Z_M be $M < p$ linear combinations of the original p predictors:

$$Z_m = \sum_{j=1}^p \phi_{jm} X_j \quad (5.3)$$

- We fit

$$y_i = \theta_0 + \sum_{m=1}^M \theta_m z_{im} + \epsilon_i, \text{ for } i = 1, \dots, n \quad (5.4)$$

$$\sum_{m=1}^M \theta_m z_{im} = \sum_{m=1}^M \theta_m \sum_{j=1}^p \phi_{jm} x_{ij} \quad (5.5)$$

$$= \sum_{j=1}^p \left(\sum_{m=1}^M \theta_m \phi_{jm} \right) x_{ij} \quad (5.6)$$

$$= \sum_{j=1}^p \beta_j x_{ij} \quad (5.7)$$

$$\beta_j = \sum_{m=1}^M \theta_m \phi_{jm} \quad (5.8)$$

5.4.2 Principal Components Regression (PCR)

- Unsupervised way: only predictors are considered.
- Generally standardize each predictor prior to generating principal components.
- Use the eigenvectors of the sample covariance (correlation) matrix to generate the transform.

$$\mathbf{e}_m = (\phi_{1m}, \dots, \phi_{pm})' \quad (5.9)$$

is the eigenvector of the sample covariance or correlation matrix with m th largest eigenvalue.

- Apply the validation set or the cross-validation approach to estimate the test MSE for each size.

5.4.3 Partial Least Squares

- Supervised way: the response Y is used.
- PLS approach attempts to find directions that help explain both the response and the predictors.
 - a. Standardize the response and predictors
 - b. ϕ_{j1} is chosen as the coefficient from the simple linear regression of Y onto X_j
 - c. Regress each variable on Z_1 and take residuals.
 - d. ϕ_{j2} is obtained in the same way using these residuals.
 - e. Repeat this procedure.

Example 5.4.1

PCR and PLS for Hitters data

```
# Principal Components Regression

library(pls)

##
## Attaching package: 'pls'
## The following object is masked from 'package:caret':
##
##     R2
## The following object is masked from 'package:stats':
##
##     loadings

set.seed(2)
pcr.fit = pcr(Salary ~ ., data = Hitters, scale = TRUE, validation = "CV")
summary(pcr.fit)

## Data: X dimension: 263 19
## Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.

##          (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
## CV          452     348.9   352.2   353.5   352.8   350.1   349.1
## adjCV       452     348.7   351.8   352.9   352.1   349.3   348.0
```

```
##          7 comps  8 comps  9 comps 10 comps 11 comps 12 comps 13 comps
## CV        349.6    350.9    352.9    353.8    355.0    356.2    363.5
## adjCV     348.5    349.8    351.6    352.3    353.4    354.5    361.6
##          14 comps 15 comps 16 comps 17 comps 18 comps 19 comps
## CV        355.2    357.4    347.6    350.1    349.2    352.6
## adjCV     352.8    355.2    345.5    347.6    346.7    349.8
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X         38.31    60.16    70.84    79.03    84.29    88.63    92.26
## Salary    40.63    41.58    42.17    43.22    44.90    46.48    46.69
##          8 comps  9 comps 10 comps 11 comps 12 comps 13 comps 14 comps
## X         94.96    96.28    97.26    97.98    98.65    99.15    99.47
## Salary    46.75    46.86    47.76    47.82    47.85    48.10    50.40
##          15 comps 16 comps 17 comps 18 comps 19 comps
## X         99.75    99.89    99.97    99.99    100.00
## Salary    50.55    53.01    53.85    54.61    54.61

validationplot(pcr.fit, val.type = "MSEP")
set.seed(1)
pcr.fit = pcr(Salary ~ ., data = Hitters, subset = train, scale = TRUE, validation = "CV")
# validationplot(pcr.fit, val.type='MSEP')
pcr.pred = predict(pcr.fit, x[test, ], ncomp = 7)
mean((pcr.pred - y.test)^2)

## [1] 96556.22
```

```
pcr.fit = pcr(y ~ x, scale = TRUE, ncomp = 7)
summary(pcr.fit)

## Data: X dimension: 263 19
## Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 7
## TRAINING: % variance explained
##    1 comps   2 comps   3 comps   4 comps   5 comps   6 comps   7 comps
## X     38.31     60.16     70.84     79.03     84.29     88.63     92.26
## y     40.63     41.58     42.17     43.22     44.90     46.48     46.69

# Partial Least Squares

set.seed(1)
pls.fit = plsr(Salary ~ ., data = Hitters, subset = train, scale = TRUE, validation = "CV")
summary(pls.fit)

## Data: X dimension: 131 19
## Y dimension: 131 1
## Fit method: kernelpls
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##      (Intercept) 1 comps   2 comps   3 comps   4 comps   5 comps   6 comps
## CV          464.6     394.2     391.5     393.1     395.0     415.0     424.0
```

```

## adjCV      464.6    393.4    390.2    391.1    392.9    411.5    418.8
##          7 comps   8 comps   9 comps  10 comps  11 comps  12 comps  13 comps
## CV        424.5    415.8    404.6    407.1    412.0    414.4    410.3
## adjCV     418.9    411.4    400.7    402.2    407.2    409.3    405.6
##          14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        406.2    408.6    410.5    408.8    407.8    410.2
## adjCV     401.8    403.9    405.6    404.1    403.2    405.5
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X         38.12    53.46    66.05    74.49    79.33    84.56    87.09
## Salary    33.58    38.96    41.57    42.43    44.04    45.59    47.05
##          8 comps  9 comps  10 comps 11 comps  12 comps  13 comps  14 comps
## X         90.74    92.55    93.94    97.23    97.88    98.35    98.85
## Salary    47.53    48.42    49.68    50.04    50.54    50.78    50.92
##          15 comps 16 comps  17 comps  18 comps  19 comps
## X         99.11    99.43    99.78    99.99    100.00
## Salary    51.04    51.11    51.15    51.16    51.18

# validationplot(pls.fit, val.type='MSEP')
pls.pred = predict(pls.fit, x[test, ], ncomp = 2)
mean((pls.pred - y.test)^2)

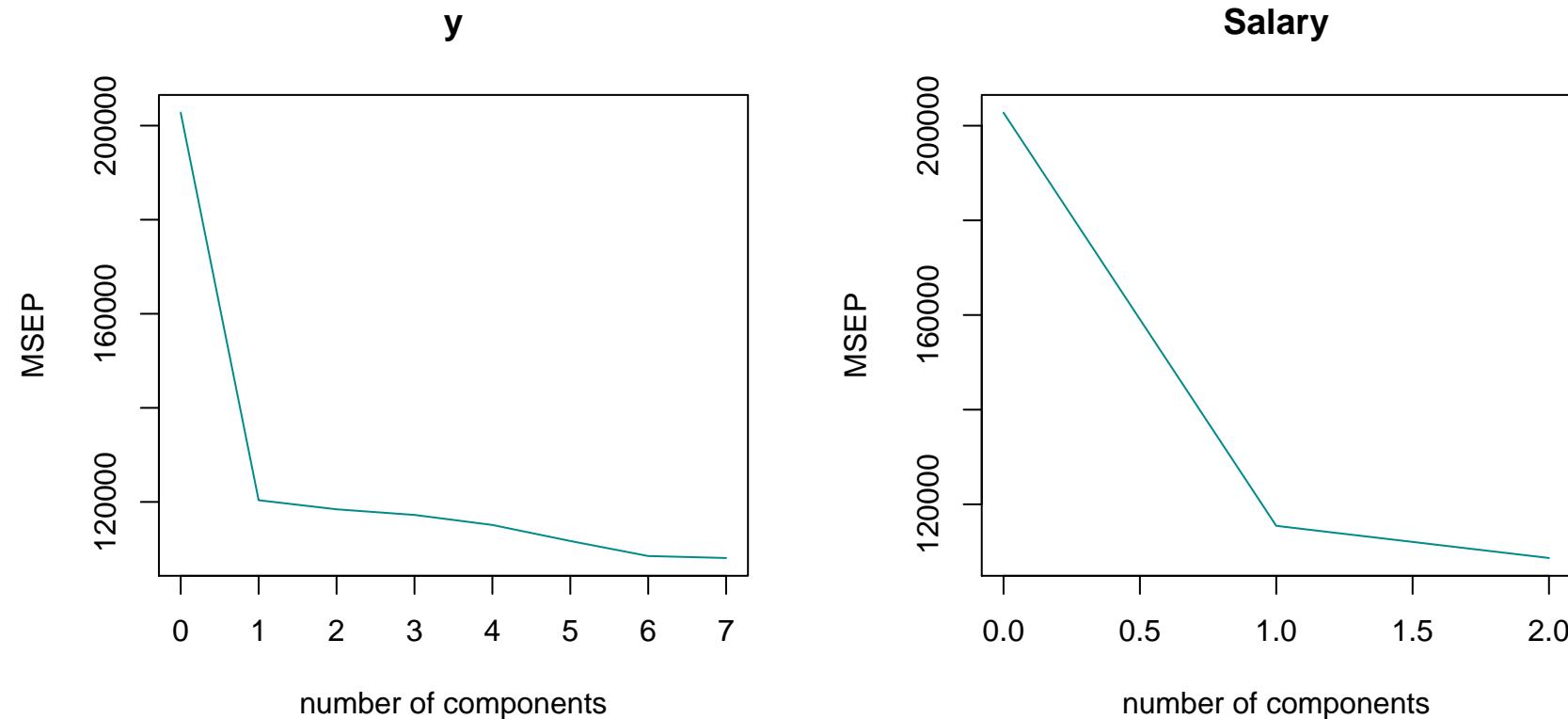
## [1] 101417.5

pls.fit = plsr(Salary ~ ., data = Hitters, scale = TRUE, ncomp = 2)
summary(pls.fit)

```

```
## Data: X dimension: 263 19
## Y dimension: 263 1
## Fit method: kernelpls
## Number of components considered: 2
## TRAINING: % variance explained
##          1 comps 2 comps
## X        38.08  51.03
## Salary   43.05  46.40
```

Figure 5.5: Validation Plots for PCR and PLS



Chapter 6 Moving beyond linearity

6.1 Introduction

- The linearity (between predictor X and response Y) assumption is almost always an approximation, and sometimes a poor one.
- We do this by examining very simple extensions of linear models like polynomial regression and step functions, as well as more sophisticated approaches such as splines, local regression, and generalized additive models.
- Extensions to linear models:
 - Polynomial regression:
extends the linear model by adding extra predictors, obtained by raising each of the original predictors to a power. For example, a cubic regression uses three variables, X , X^2 , and X^3 , as predictors. This approach provides a simple way to provide a non-linear fit to data.
 - Step functions
cut the range of a variable into K distinct regions in order to produce a qualitative variable. This has the effect of fitting a piecewise constant function.

- Regression splines:
are more flexible than polynomials and step functions, and in fact are an extension of the two. They involve dividing the range of X into K distinct regions. Within each region, a polynomial function is fit to the data. However, these polynomials are constrained so that they join smoothly at the region boundaries, or knots. Provided that the interval is divided into enough regions, this can produce an extremely flexible fit.
- Smoothing splines:
are similar to regression splines, but arise in a slightly different situation. Smoothing splines result from minimizing a residual sum of squares criterion subject to a smoothness penalty.
- Local regression:
is similar to splines, but differs in an important way. The regions are allowed to overlap, and indeed they do so in a very smooth way.
- Generalized additive models:
allow us to extend the methods above to deal with multiple predictors.

6.2 Polynomial regression

1. $y_i = \beta_0 + \beta_1 x_i + \cdots + \beta_d x_i^d + \epsilon_i$ for a quantitative response
2. $\text{logit}[p(Y = 1|x)] = \beta_0 + \beta_1 x + \cdots + \beta_d x^d$ for a binary response
3. It can be easily fitted by the least square methods by considering x_i, \dots, x_i^d as the predictors.
4. Polynomials of $d > 3$ are not frequently used.
5. Wage data example:

6.3 Step functions

- Create cut points c_1, \dots, c_K
- Generate indicator functions

$$\begin{aligned}
 C_0(x) &= I(x < c_0), \\
 C_1(x) &= I(c_0 \leq x < c_1), \\
 &\vdots \\
 C_{K-1}(x) &= I(c_{K-1} \leq x < c_K), \\
 C_K(x) &= I(c_K \leq x)
 \end{aligned} \tag{6.1}$$

- $y_i = \beta_0 + \beta_1 C_1(x_i) + \dots + \beta_K C_K(x_i) + \epsilon_i$
- We can use the least square methods.
- Wage data example

6.3.1 Basis function approach

1. Polynomial regression and step function approach can be viewed as a basis function approach
2. For a fixed (chosen) functions, $b_1(x), \dots, b_K(x)$,

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \dots + \beta_K b_K(x_i) + \epsilon_i$$

3. Many ways to select the basis functions:

- Polynomial regression: $b_j(x) = x^j$
- Step function regression: $b_j(x) = I(c_j \leq x < c_{j+1})$
- Many alternatives

Example 6.3.1

Wage data

```
library(ISLR)
attach(Wage)

# Polynomial Regression and Step Functions

fit = lm(wage ~ poly(age, 4), data = Wage)
coef(summary(fit))

##                 Estimate Std. Error     t value    Pr(>|t|)
## (Intercept)    111.70361  0.7287409 153.283015 0.000000e+00
## poly(age, 4)1   447.06785 39.9147851 11.200558 1.484604e-28
## poly(age, 4)2  -478.31581 39.9147851 -11.983424 2.355831e-32
## poly(age, 4)3   125.52169 39.9147851    3.144742 1.678622e-03
## poly(age, 4)4  -77.91118 39.9147851   -1.951938 5.103865e-02

fit2 = lm(wage ~ poly(age, 4, raw = T), data = Wage)
coef(summary(fit2))

##                 Estimate Std. Error     t value    Pr(>|t|)
## (Intercept)      -1.841542e+02 6.004038e+01 -3.067172 0.0021802539
## poly(age, 4, raw = T)1  2.124552e+01 5.886748e+00  3.609042 0.0003123618
## poly(age, 4, raw = T)2 -5.638593e-01 2.061083e-01 -2.735743 0.0062606446
## poly(age, 4, raw = T)3  6.810688e-03 3.065931e-03  2.221409 0.0263977518
## poly(age, 4, raw = T)4 -3.203830e-05 1.641359e-05 -1.951938 0.0510386498
```

```

fit2a = lm(wage ~ age + I(age^2) + I(age^3) + I(age^4), data = Wage)
coef(fit2a)

##   (Intercept)           age          I(age^2)          I(age^3)          I(age^4)
## -1.841542e+02  2.124552e+01 -5.638593e-01  6.810688e-03 -3.203830e-05

fit2b = lm(wage ~ cbind(age, age^2, age^3, age^4), data = Wage)
agelims = range(age)
age.grid = seq(from = agelims[1], to = agelims[2])
preds = predict(fit, newdata = list(age = age.grid), se = TRUE)
se.bands = cbind(preds$fit + 2 * preds$se.fit, preds$fit - 2 * preds$se.fit)
# par(mfrow=c(1,2),mar=c(4.5,4.5,1,1),oma=c(0,0,4,0))
# plot(age,wage,xlim=agelims,cex=.5,col='darkgrey') title('Degree-4
# Polynomial',outer=T) lines(age.grid,preds$fit,lwd=2,col='blue')
# matlines(age.grid,se.bands,lwd=1,col='blue',lty=3)
preds2 = predict(fit2, newdata = list(age = age.grid), se = TRUE)
max(abs(preds$fit - preds2$fit))

## [1] 7.81597e-11

fit.1 = lm(wage ~ age, data = Wage)
fit.2 = lm(wage ~ poly(age, 2), data = Wage)
fit.3 = lm(wage ~ poly(age, 3), data = Wage)
fit.4 = lm(wage ~ poly(age, 4), data = Wage)
fit.5 = lm(wage ~ poly(age, 5), data = Wage)
anova(fit.1, fit.2, fit.3, fit.4, fit.5)

```

```
## Analysis of Variance Table
##
## Model 1: wage ~ age
## Model 2: wage ~ poly(age, 2)
## Model 3: wage ~ poly(age, 3)
## Model 4: wage ~ poly(age, 4)
## Model 5: wage ~ poly(age, 5)
##   Res.Df   RSS Df Sum of Sq      F    Pr(>F)
## 1 2998 5022216
## 2 2997 4793430  1  228786 143.5931 < 2.2e-16 ***
## 3 2996 4777674  1   15756   9.8888  0.001679 **
## 4 2995 4771604  1     6070   3.8098  0.051046 .
## 5 2994 4770322  1     1283   0.8050  0.369682
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

coef(summary(fit.5))

##                   Estimate Std. Error      t value      Pr(>|t|)
## (Intercept) 111.70361  0.7287647 153.2780243 0.000000e+00
## poly(age, 5)1 447.06785 39.9160847 11.2001930 1.491111e-28
## poly(age, 5)2 -478.31581 39.9160847 -11.9830341 2.367734e-32
## poly(age, 5)3 125.52169 39.9160847   3.1446392 1.679213e-03
## poly(age, 5)4 -77.91118 39.9160847  -1.9518743 5.104623e-02
## poly(age, 5)5 -35.81289 39.9160847  -0.8972045 3.696820e-01

(-11.983)^2
```

```

## [1] 143.5923

fit.1 = lm(wage ~ education + age, data = Wage)
fit.2 = lm(wage ~ education + poly(age, 2), data = Wage)
fit.3 = lm(wage ~ education + poly(age, 3), data = Wage)
anova(fit.1, fit.2, fit.3)

## Analysis of Variance Table
##
## Model 1: wage ~ education + age
## Model 2: wage ~ education + poly(age, 2)
## Model 3: wage ~ education + poly(age, 3)
##   Res.Df   RSS Df Sum of Sq    F Pr(>F)
## 1 2994 3867992
## 2 2993 3725395  1   142597 114.6969 <2e-16 ***
## 3 2992 3719809  1      5587  4.4936 0.0341 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

fitn = glm(I(wage > 250) ~ poly(age, 4), data = Wage, family = binomial)
predsn = predict(fitn, newdata = list(age = age.grid), se = T)
pfit = exp(predsn$fit)/(1 + exp(predsn$fit))
se.bands.logit = cbind(predsn$fit + 2 * predsn$se.fit, predsn$fit - 2 * predsn$se.fit)
se.bands1 = exp(se.bands.logit)/(1 + exp(se.bands.logit))
predsnn = predict(fitn, newdata = list(age = age.grid), type = "response", se = T)
# plot(age,I(wage>250),xlim=agelims,type='n',ylim=c(0,.2))
# points(jitter(age), I((wage>250)/5),cex=.5,pch='|',col='darkgrey')

```

```
# lines(age.grid,pfit,lwd=2, col='blue')
# matlines(age.grid,se.bands1,lwd=1,col='blue',lty=3)

```

Figure 6.1: The Wage data. Left: The solid blue curve is a degree-4 polynomial of wage (in thousands of dollars) as a function of age, fit by least squares. The dotted curves indicate an estimated 95 % confidence interval. Right: We model the binary event $wage > 250$ using logistic regression, again with a degree-4 polynomial. The fitted posterior probability of wage exceeding \$250,000 is shown in blue, along with an estimated 95 % confidence interval.

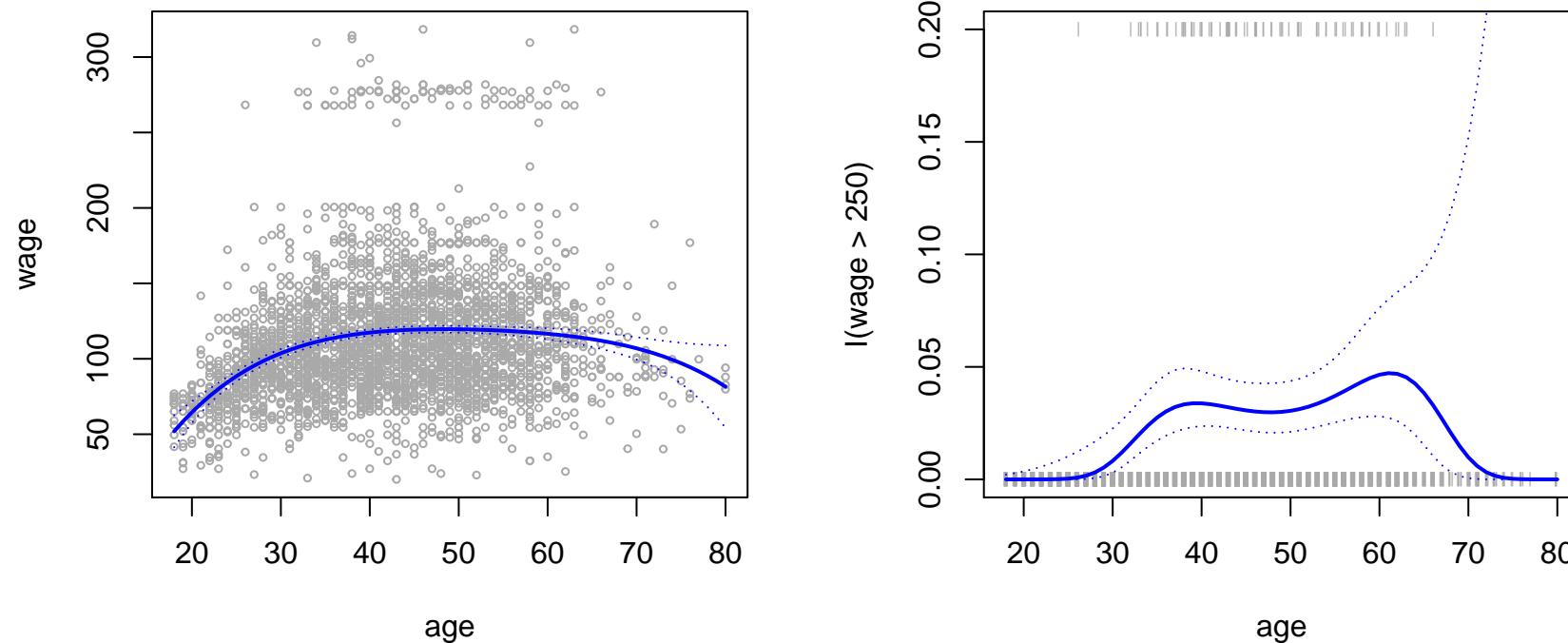
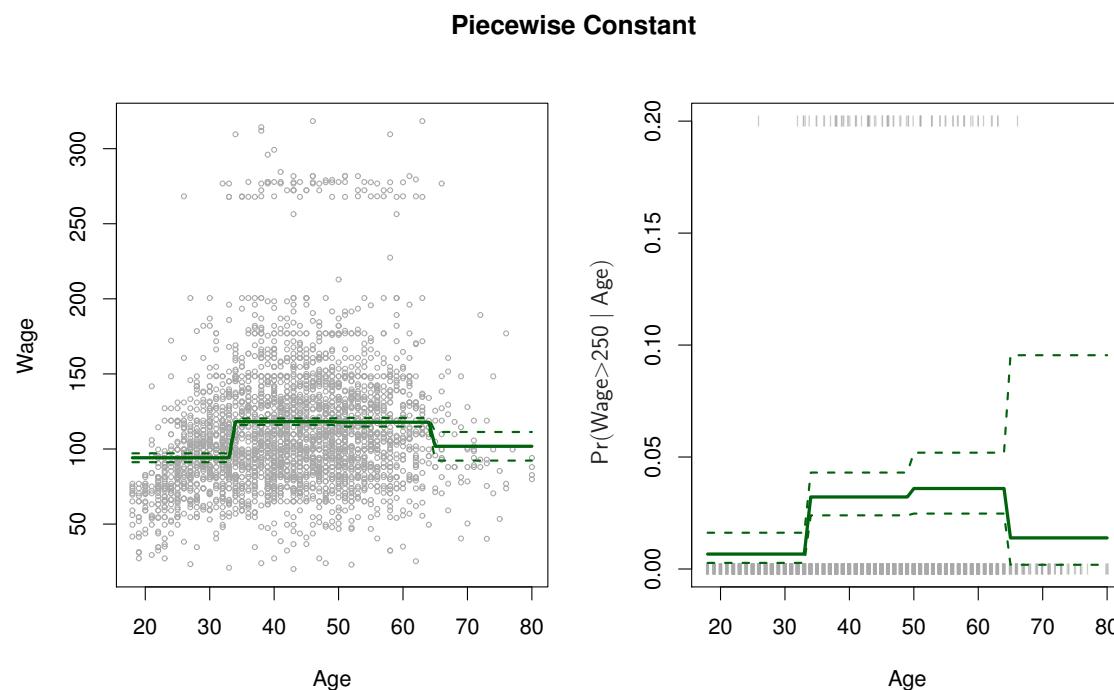


Figure 6.2: The Wage data. Left: The solid curve displays the fitted value from a least squares regression of wage (in thousands of dollars) using step functions of age. The dotted curves indicate an estimated 95 % confidence interval. Right: We model the binary event $wage > 250$ using logistic regression, again using step functions of age. The fitted posterior probability of wage exceeding \$250,000 is shown, along with an estimated 95 % confidence interval.



6.4 Regression splines

6.4.1 Piecewise Polynomials

- Piecewise polynomials and knots are used.
- For example, we use a single knot c and cubic polynomials:

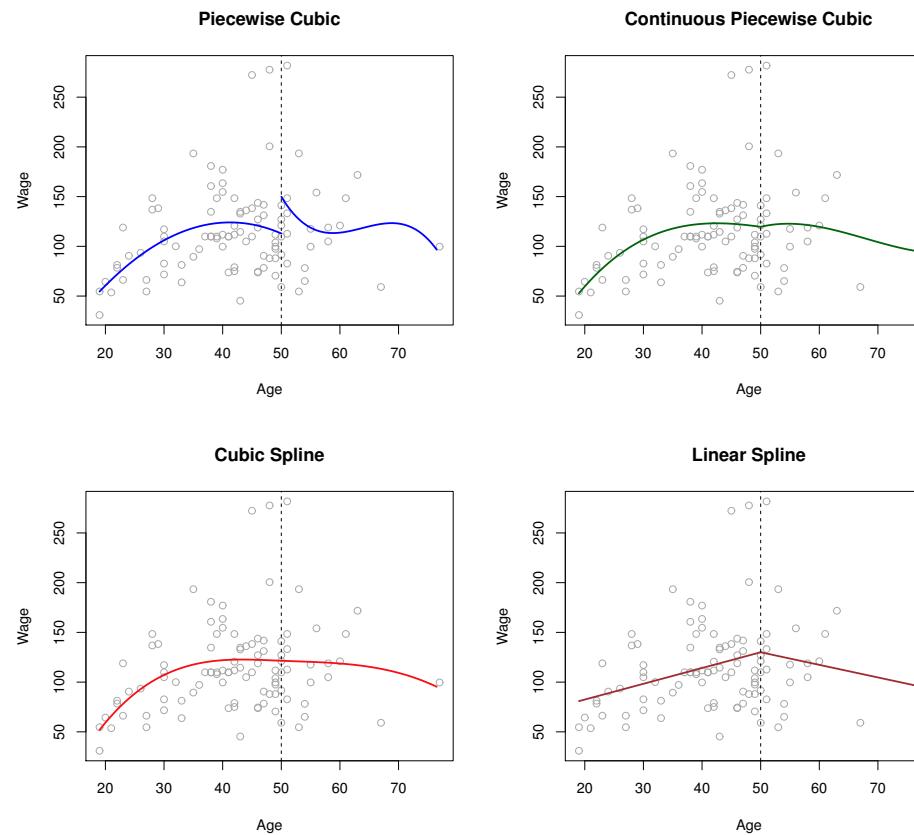
$$y_i = \begin{cases} \beta_{01} + \beta_{11}x_i + \beta_{21}x_i^2 + \beta_{31}x_i^3 + \epsilon_i & \text{for } x_i < c \\ \beta_{02} + \beta_{12}x_i + \beta_{22}x_i^2 + \beta_{32}x_i^3 + \epsilon_i & \text{for } x_i \geq c \end{cases}$$

6.4.2 Constraints and Splines

In Figure 6.3,

- we imposed three constraints (continuity, continuity of the first derivative, and continuity of the second derivative) and so are left with five degrees of freedom. It is called a **cubic spline**.
- In general, a cubic spline with K knots uses a total of $K + 4$ degrees of freedom.

Figure 6.3: Various piecewise polynomials are fit to a subset of the Wage data, with a knot at age=50. Top Left: The cubic polynomials are unconstrained. Top Right: The cubic polynomials are constrained to be continuous at age=50. Bottom Left: The cubic polynomials are constrained to be continuous, and to have continuous first and second derivatives. Bottom Right: A linear spline is shown, which is constrained to be continuous.



6.4.3 The Spline Basis Representation

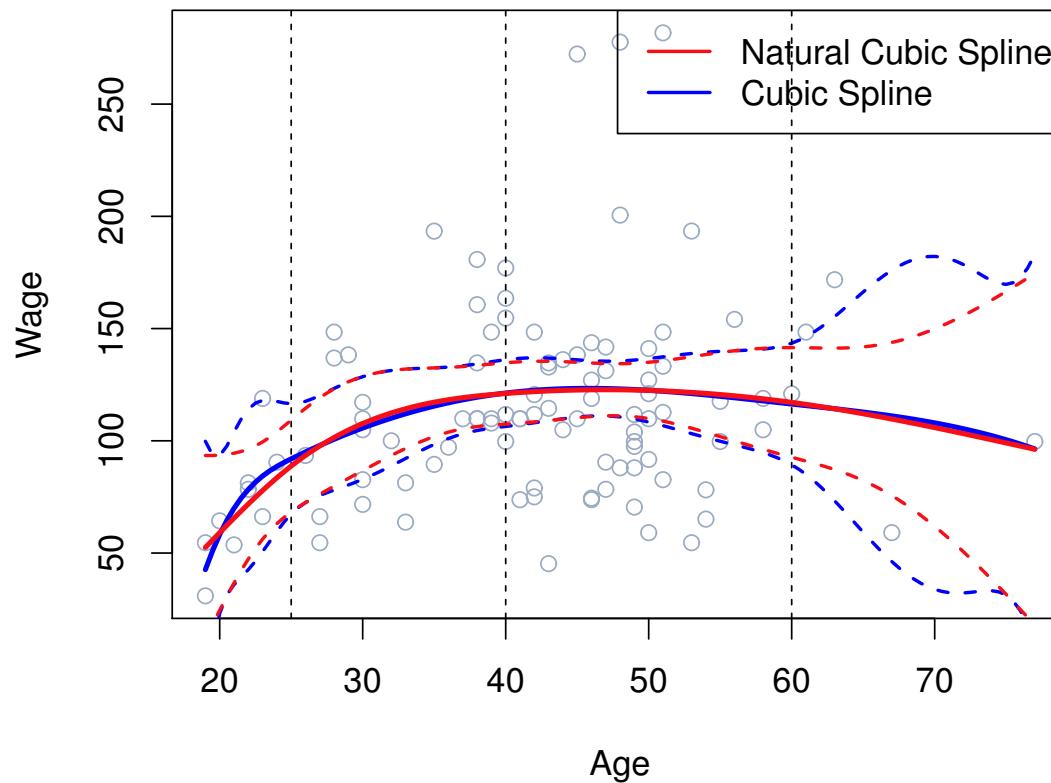
- A cubic spline with K knots can be modeled as $y_i = \beta_0 + \beta_1 b_1(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i$
- There are many ways to represent the polynomials. The most direct way to represent a cubic spline is to start off with a basis for a cubic polynomial: x, x^2, x^3 and add one truncated power basis function per knot: $h(x, \xi) = (x - \xi)_+^3 = \begin{cases} (x - \xi)^3 & \text{if } x > \xi \\ 0 & \text{o.w.} \end{cases}$, where ξ is a knot.
- In order to fit a cubic spline to a data set with K knots, we perform least squares regression with an intercept and $K+3$ predictors, of the form

$$1, X, X^2, X^3, h(X, \xi_1), h(X, \xi_2), \dots, h(X, \xi_K) \quad (6.2)$$

where ξ_1, \dots, ξ_K are the knots.

- Unfortunately, splines can have high variance at the outer range of the predictors.
- A **natural spline** is a regression spline with additional **boundary constraints**: the function is required to be linear at the boundary (in the region where X is smaller than the smallest knot, or larger than the largest knot).

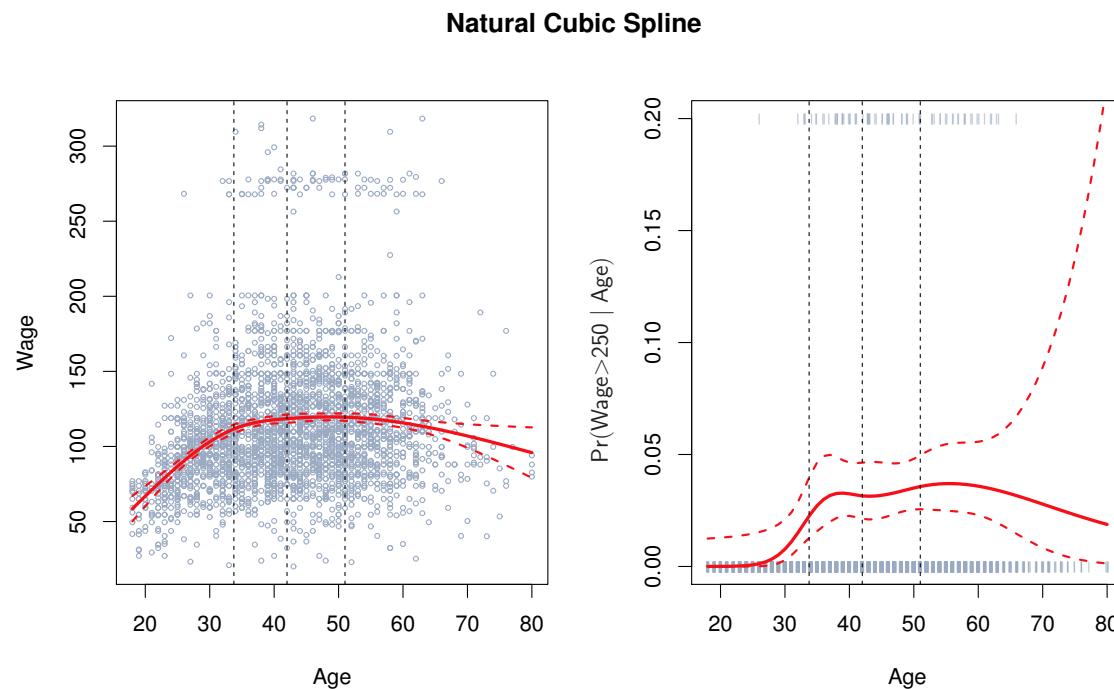
Figure 6.4: A cubic spline and a natural cubic spline, with three knots, fit to a subset of the Wage data.



6.4.4 Choosing the Number and Locations of the Knots

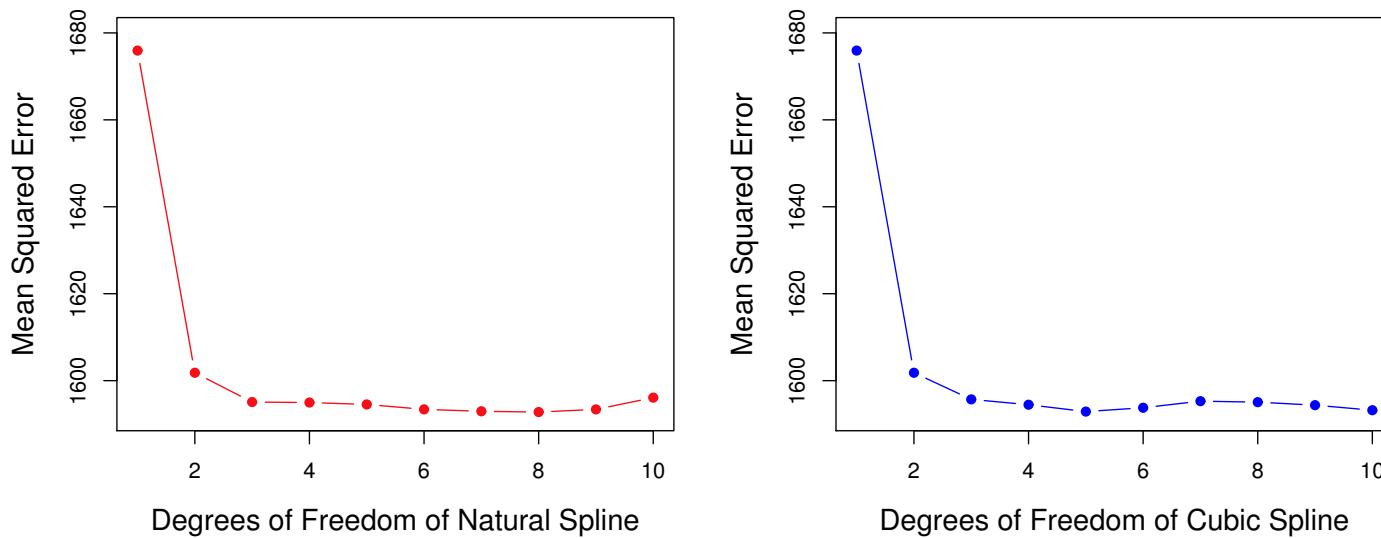
- Specify the desired degrees of freedom, and then have the software automatically place the corresponding number of knots at uniform quantiles of the data.

Figure 6.5: A natural cubic spline function with four degrees of freedom is fit to the Wage data. Left: A spline is fit to wage (in thousands of dollars) as a function of age. Right: Logistic regression is used to model the binary event $wage > 250$ as a function of age. The fitted posterior probability of wage exceeding \$250,000 is shown.



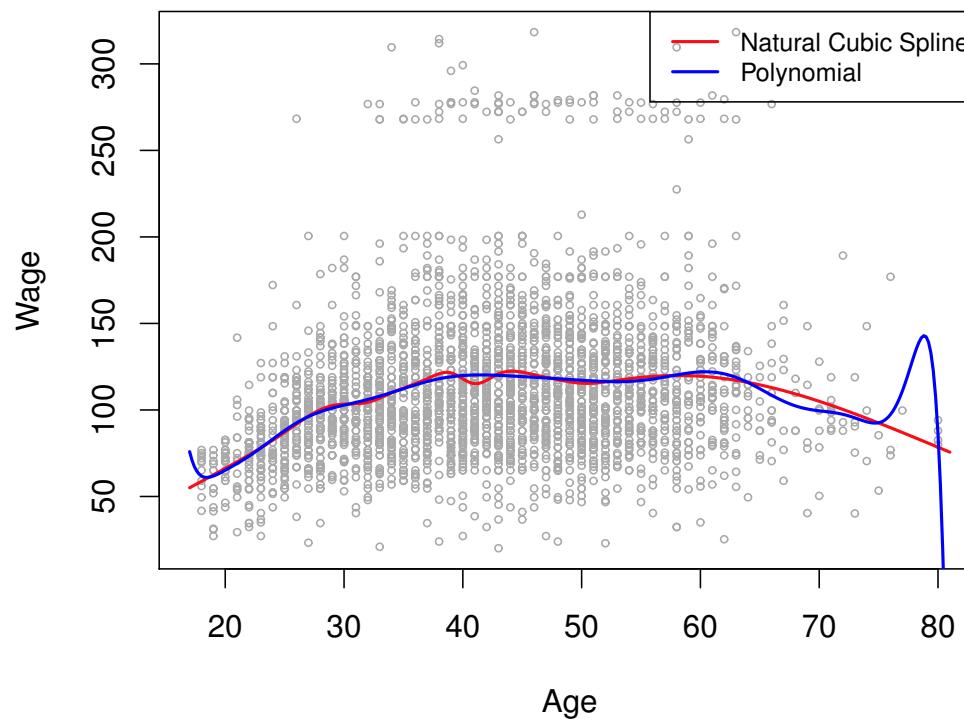
- How many knots should we use, or equivalently how many degrees of freedom should our spline contain?
⇒ Try out different numbers of knots and see which produces the best looking curve. Cross-validation may be used.

Figure 6.6: Ten-fold cross-validated mean squared errors for selecting the degrees of freedom when fitting splines to the Wage data. The response is wage and the predictor age. Left: A natural cubic spline. Right: A cubic spline.



6.4.5 Comparison to Polynomial Regression

Figure 6.7: On the Wage data set, a natural cubic spline with 15 degrees of freedom is compared to a degree-15 polynomial. Polynomials can show wild behavior, especially near the tails.



6.5 Smoothing splines

6.5.1 An Overview of Smoothing Splines

- Find the function g that minimizes

$$\underbrace{\sum_{i=1}^n (y_i - g(x_i))^2}_{\text{loss term}} + \lambda \underbrace{\int (g''(t))^2 dt}_{\text{penalty term}}$$

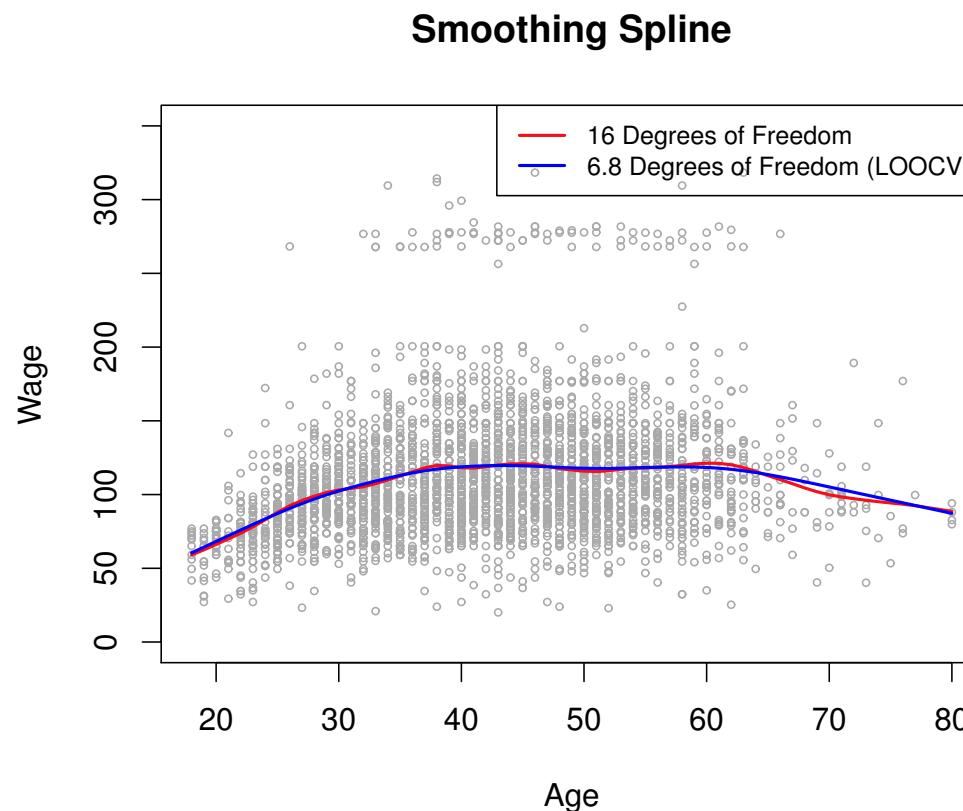
where g is called the smoothing spline and λ is called the tuning parameter. The first term is called the loss term and the second term is called the penalty term.

- It turns out that a **natural cubic spline with knots** x_1, \dots, x_n is the function g minimizing the above loss + penalty (ESL problem 5.7; Green and Silverman, 1994).
- However, it is not the same natural cubic spline that one would get if one applied the basis function approach described with knots at x_1, \dots, x_n . It is a shrunken version of such a natural cubic spline.

6.5.2 Choosing the Smoothing Parameter λ

- Although a smoothing spline has n parameters and hence n nominal degrees of freedom, these n parameters are heavily constrained or shrunk down.
- Effective degrees of freedom: let $\hat{g}_\lambda = S_\lambda y$, the effective df is defined as $df_\lambda = \sum_{i=1}^n (S_\lambda)_{ii}$
- We do not need to select the knots for smoothing splines but we need to choose the tuning parameter λ
- LOOCV does not need multiple fits: $RSS_{CV}(\lambda) = \sum_{i=1}^n \left(y_i - \hat{g}_\lambda^{(-i)}(x_i) \right)^2 = \sum_{i=1}^n \left(\frac{y_i - \hat{g}_\lambda(x_i)}{1 - (S_\lambda)_{ii}} \right)^2$

Figure 6.8: Smoothing spline fits to the Wage data. The red curve results from specifying 16 effective degrees of freedom. For the blue curve, λ was found automatically by leave-one-out cross-validation, which resulted in 6.8 effective degrees of freedom.



Given (x_1, \dots, x_n) , let $\hat{\mathbf{a}} = (\hat{g}(x_1), \dots, \hat{g}(x_n))^T$. The interpolation is

$$\hat{g}(x) = \sum_{i=1}^n \hat{g}(x_i) g_i(x) \quad (6.3)$$

where $\{g_i\}$ is a set of spline basis functions.

$$\int |\hat{g}''(x)|^2 dx = \hat{\mathbf{a}}^T \mathbf{A} \hat{\mathbf{a}} \quad (6.4)$$

where $A_{ij} = \int g_i''(x) g_j''(x) dx$

6.5.3 Local regression

- Computing the fit at a target point x_0 using only the nearby training observations: memory-based procedure
 - Need to select the weight function K , and whether to fit a linear, constant, or quadratic regression. The most important choice is the span $s = k/n$.
1. Gather the fraction $s = k/n$ of training points whose x_i are closest to x_0 .
 2. Assign a weight $K_{i0} = K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero, and the closest has the highest weight. All but these k nearest neighbors get weight zero.
 3. Fit a weighted least squares regression of the y_i on the x_i using the aforementioned weights, by finding $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize $\sum_{i=1}^n K_{i0} (y_i - \beta_0 - \beta_1 x_i)^2$
 4. The fitted value at x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$

Figure 6.9: Local regression illustrated on some simulated data, where the blue curve represents $f(x)$ from which the data were generated, and the light orange curve corresponds to the local regression estimate $\hat{f}(x)$. The orange colored points are local to the target point x_0 , represented by the orange vertical line. The yellow bell-shape superimposed on the plot indicates weights assigned to each point, decreasing to zero with distance from the target point. The fit $\hat{f}(x_0)$ at x_0 is obtained by fitting a weighted linear regression (orange line segment), and using the fitted value at x_0 (orange solid dot) as the estimate $\hat{f}(x_0)$.

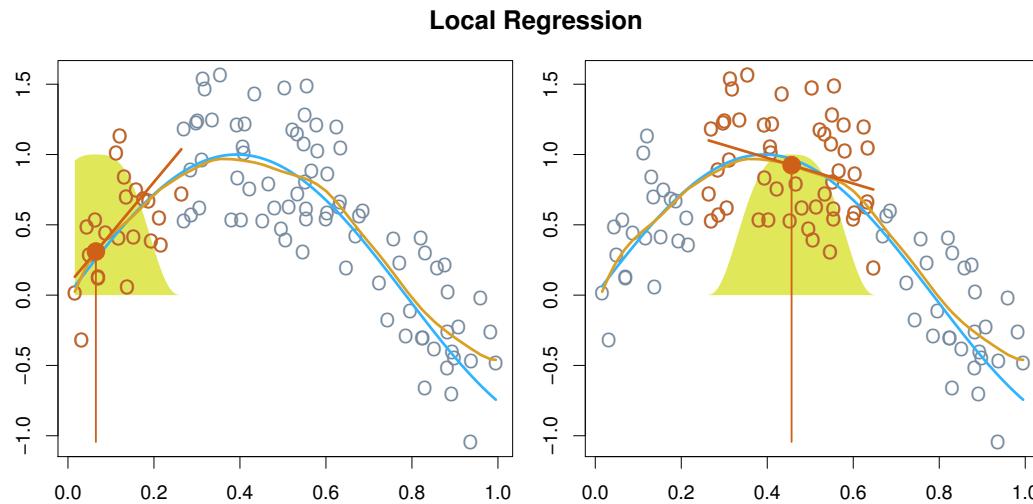
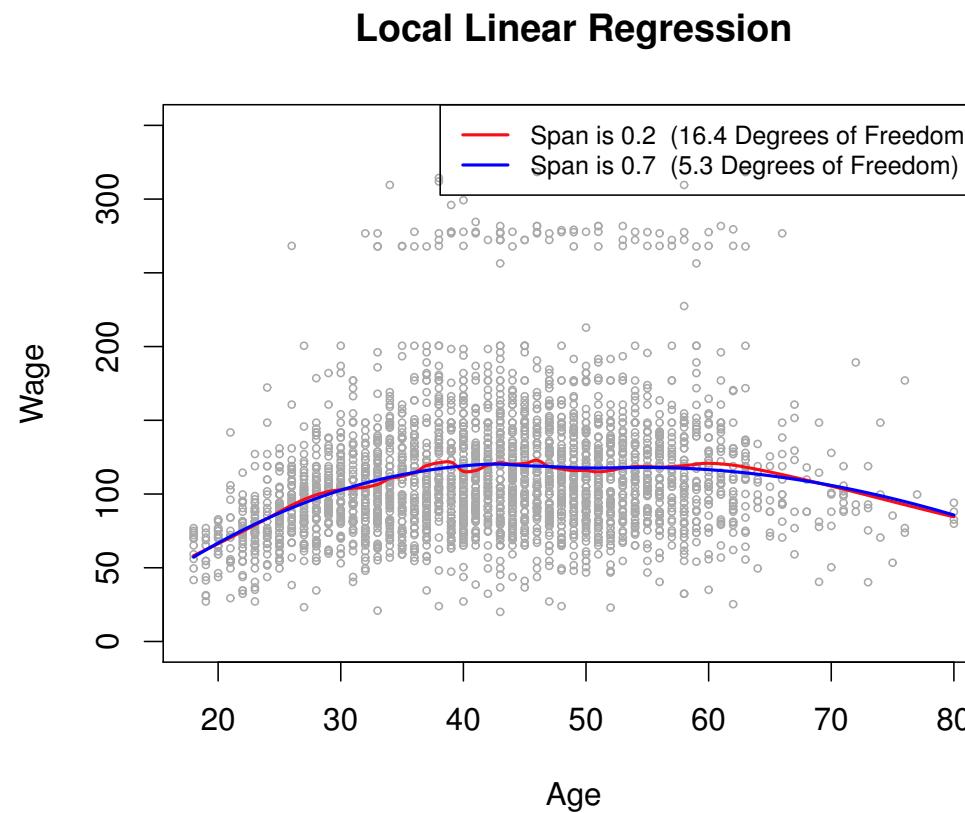


Figure 6.10: Local linear fits to the Wage data. The span specifies the fraction of the data used to compute the fit at each target point.



Example 6.5.1# Splines

```
library(splines)
fit = lm(wage ~ bs(age, knots = c(25, 40, 60)), data = Wage)
pred = predict(fit, newdata = list(age = age.grid), se = T)
# plot(age,wage,col='gray') lines(age.grid,pred$fit,lwd=2)
# lines(age.grid,pred$fit+2*pred$se,lty='dashed')
# lines(age.grid,pred$fit-2*pred$se,lty='dashed')
dim(bs(age, knots = c(25, 40, 60)))

## [1] 3000     6

dim(bs(age, df = 6))

## [1] 3000     6

attr(bs(age, df = 6), "knots")

##    25%    50%    75%
## 33.75 42.00 51.00

fit2 = lm(wage ~ ns(age, df = 4), data = Wage)
pred2 = predict(fit2, newdata = list(age = age.grid), se = T)
# lines(age.grid, pred2$fit,col='red',lwd=2)
# plot(age,wage,xlim=agelims,cex=.5,col='darkgrey') title('Smoothing
# Spline')
fit = smooth.spline(age, wage, df = 16)
fit2 = smooth.spline(age, wage, cv = TRUE)
fit2$df
```

```
## [1] 6.794596

# lines(fit,col='red',lwd=2) lines(fit2,col='blue',lwd=2)
# legend('topright',legend=c('16 DF','6.8
# DF'),col=c('red','blue'),lty=1,lwd=2,cex=.8)
# plot(age,wage,xlim=agelims,cex=.5,col='darkgrey') title('Local
# Regression')
fit = loess(wage ~ age, span = 0.2, data = Wage)
fit2 = loess(wage ~ age, span = 0.5, data = Wage)
# lines(age.grid,predict(fit,data.frame(age=age.grid)),col='red',lwd=2)
# lines(age.grid,predict(fit2,data.frame(age=age.grid)),col='blue',lwd=2)
# legend('topright',legend=c('Span=0.2','Span=0.5'),col=c('red','blue'),lty=1,lwd=2,cex=.8)
```

6.6 Generalized Additive Model (GAM)

- Extending a standard linear model by allowing non-linear functions of each of the variables, while maintaining additivity.
- Instead of $\beta_0 + \sum_j \beta_j X_j$, we use $\beta_0 + \sum_j f_j(X_j)$ to model Y .
- We calculate a separate f_j for each X_j , and then add together all of their contributions.
- Natural spline, smoothing spline, local regression, and/or polynomial regression can be used for f_j .

6.6.1 GAMs for Regression Problems

- $y_i = \beta_0 + f_1(x_{i1}) + \cdots + f_p(x_{ip}) + \epsilon_i$ for a continuous response

- The Backfitting Algorithm for Additive Models:

1. Initialize $\widehat{\beta}_0 = \frac{1}{N} \sum_{i=1}^N y_i$, $\widehat{f}_j \equiv 0$ for all i, j .

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

$$\begin{aligned}\widehat{f}_j &\leftarrow S_j \left[\left\{ y_i - \widehat{\beta}_0 - \sum_{k \neq j} \widehat{f}_k(x_{ik}) \right\}_1^N \right] \\ \widehat{f}_j &\leftarrow \widehat{f}_j - \frac{1}{N} \sum_{i=1}^N \widehat{f}_j(x_{ij})\end{aligned}$$

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

6.6.2 GAMs for Classification Problems

- $\log \frac{\Pr(Y_i=1|x_i)}{1-\Pr(Y_i=1|x_i)} = \beta_0 + f_1(x_{i1}) + \dots + \beta_p f_p(x_{ip})$ for a binary response.
- Local Scoring Algorithm for the Additive Logistic Regression Model.

1. Compute starting values: $\widehat{\beta}_0 = \log[\bar{y}/(1 - \bar{y})]$, where $\bar{y} = (1/N) \sum_{i=1}^N y_i$. Set $\widehat{f}_j \equiv 0$ for all j .

2. Define $\widehat{\eta}_i = \widehat{\beta}_0 + \sum_j \widehat{f}_j(x_{ij})$ and $\widehat{p}_i = 1/[1 + \exp(-\widehat{\eta}_i)]$.

Iterate:

(a) Construct the working target variable

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

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

- (c) Fit an additive model to the targets z_i with weights w_i , using a weighted backfitting algorithm. This gives new estimates $\hat{\beta}_0, \hat{f}_j$ for all j
3. Continue step 2. until the change in the functions falls below a prespecified threshold.

Pros and Cons of GAMs

- GAMs allow us to fit a non-linear f_j to each X_j , so that we can automatically model non-linear relationships that standard linear regression will miss. This means that we do not need to manually try out many different transformations on each variable individually.
- The non-linear fits can potentially make more accurate predictions for the response Y .
- Because the model is additive, we can still examine the effect of each X_j on Y individually while holding all of the other variables fixed. Hence if we are interested in inference, GAMs provide a useful representation.
- The smoothness of the function f_j for the variable X_j can be summarized via degrees of freedom.
- The main limitation of GAMs is that the model is restricted to be additive. With many variables, important interactions can be missed. However, as with linear regression, we can manually add interaction terms to the GAM model by including additional predictors of the form $X_j \times X_k$. In addition we can add low-dimensional interaction functions of the form $f_{jk}(X_j, X_k)$ into the model; such terms can be fit using two-dimensional smoothers such as local regression, or two-dimensional splines (not covered here).

Example 6.6.1# GAMs

```
gam1 = lm(wage ~ ns(year, 4) + ns(age, 5) + education, data = Wage)
library(gam)
## Loaded gam 1.16

gam.m3 = gam(wage ~ s(year, 4) + s(age, 5) + education, data = Wage)
# par(mfrow=c(1,3)) plot(gam.m3, se=TRUE,col='blue') plot.gam(gam1, se=TRUE,
# col='red')
gam.m1 = gam(wage ~ s(age, 5) + education, data = Wage)
gam.m2 = gam(wage ~ year + s(age, 5) + education, data = Wage)
anova(gam.m1, gam.m2, gam.m3, test = "F")

## Analysis of Deviance Table
##
## Model 1: wage ~ s(age, 5) + education
## Model 2: wage ~ year + s(age, 5) + education
## Model 3: wage ~ s(year, 4) + s(age, 5) + education
##   Resid. Df Resid. Dev Df Deviance      F    Pr(>F)
## 1     2990    3711731
## 2     2989    3693842  1   17889.2 14.4771 0.0001447 ***
## 3     2986    3689770  3    4071.1  1.0982 0.3485661
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

summary(gam.m3)

##
```

```
## Call: gam(formula = wage ~ s(year, 4) + s(age, 5) + education, data = Wage)
## Deviance Residuals:
##      Min     1Q Median     3Q    Max 
## -119.43 -19.70 -3.33 14.17 213.48 
##
## (Dispersion Parameter for gaussian family taken to be 1235.69)
##
## Null Deviance: 5222086 on 2999 degrees of freedom
## Residual Deviance: 3689770 on 2986 degrees of freedom
## AIC: 29887.75
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##          Df  Sum Sq Mean Sq F value    Pr(>F)    
## s(year, 4)  1  27162   27162  21.981 2.877e-06 ***
## s(age, 5)   1 195338  195338 158.081 < 2.2e-16 ***
## education   4 1069726  267432 216.423 < 2.2e-16 ***
## Residuals 2986 3689770     1236
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##          Npar Df Npar F  Pr(F)    
## (Intercept)        1  1  1.086 0.3537
```

```

## s(age, 5)      4 32.380 <2e-16 ***
## education
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

preds = predict(gam.m2, newdata = Wage)
gam.lo = gam(wage ~ s(year, df = 4) + lo(age, span = 0.7) + education, data = Wage)
# plot.gam(gam.lo, se=TRUE, col='green')
gam.lo.i = gam(wage ~ lo(year, age, span = 0.2) + education, data = Wage)
library(akima)
# plot(gam.lo.i)
gam.lr = gam(I(wage > 250) ~ year + s(age, df = 5) + education, family = binomial,
            data = Wage)
# par(mfrow=c(1,3)) plot(gam.lr,se=T,col='green')
table(education, I(wage > 250))

##
## education      FALSE TRUE
## 1. < HS Grad    268   0
## 2. HS Grad     966   5
## 3. Some College 643   7
## 4. College Grad 663  22
## 5. Advanced Degree 381  45

gam.lr.s = gam(I(wage > 250) ~ year + s(age, df = 5) + education, family = binomial,
               data = Wage, subset = (education != "1. < HS Grad"))
# plot(gam.lr.s,se=T,col='green')

```

Figure 6.11: For the Wage data, plots of the relationship between each feature and the response, wage, in the fitted model. Each plot displays the fitted function and pointwise standard errors. The first two functions are natural splines in year and age, with four and five degrees of freedom, respectively. The third function is a step function, fit to the qualitative variable education.

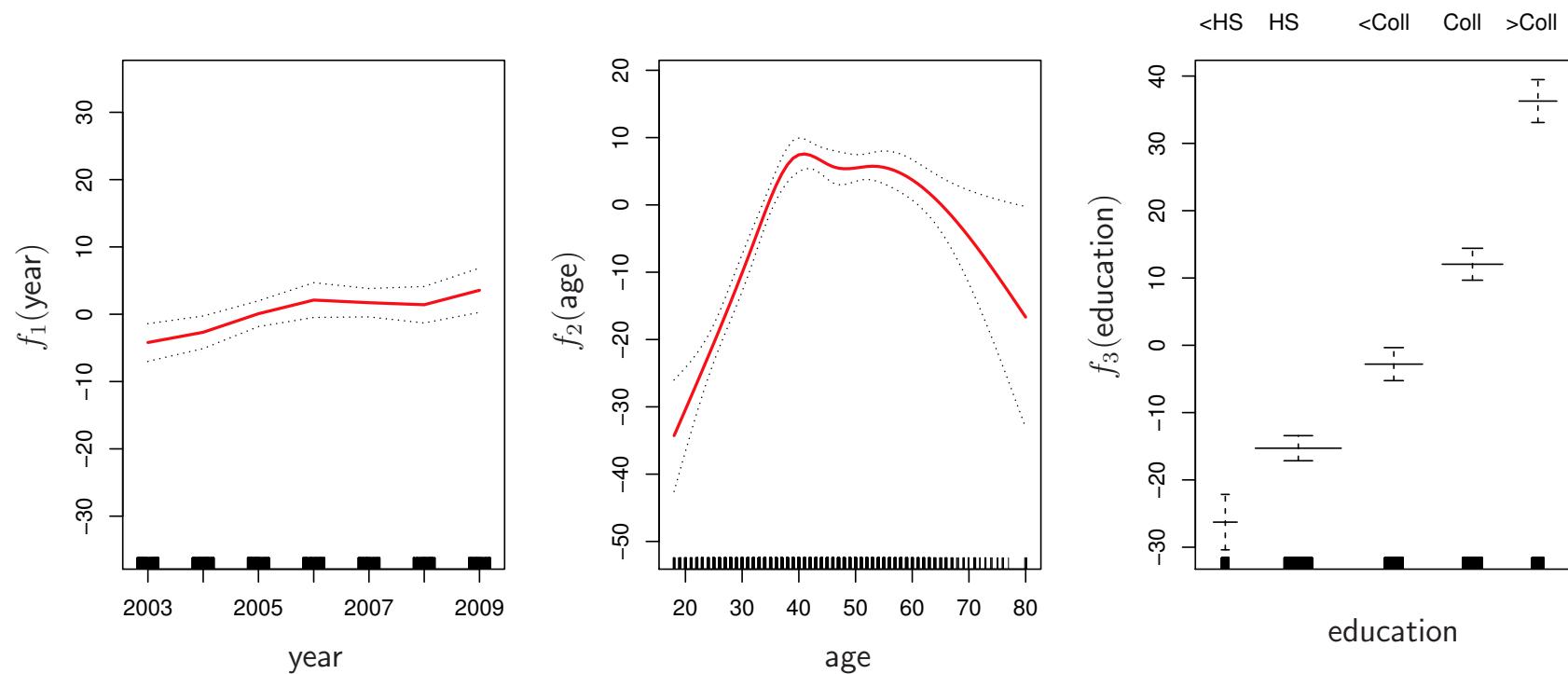


Figure 6.12: Details are as in Figure 6.11, but now f_1 and f_2 are smoothing splines with four and five degrees of freedom, respectively.

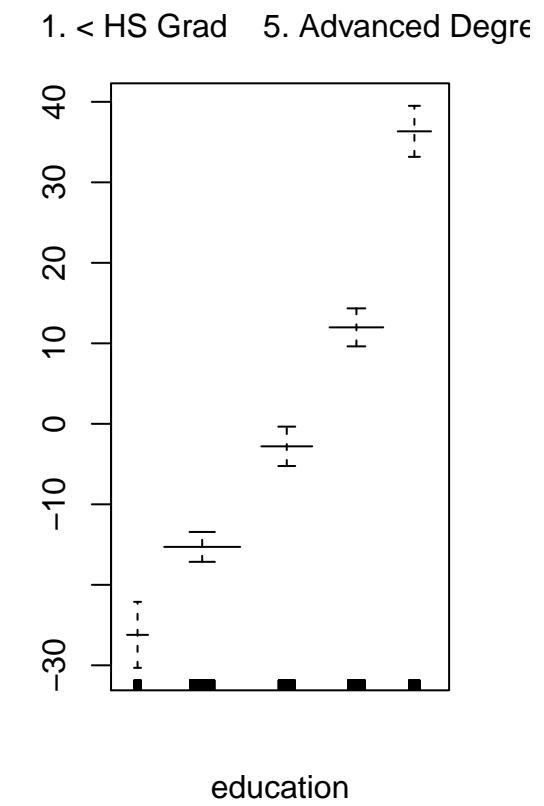
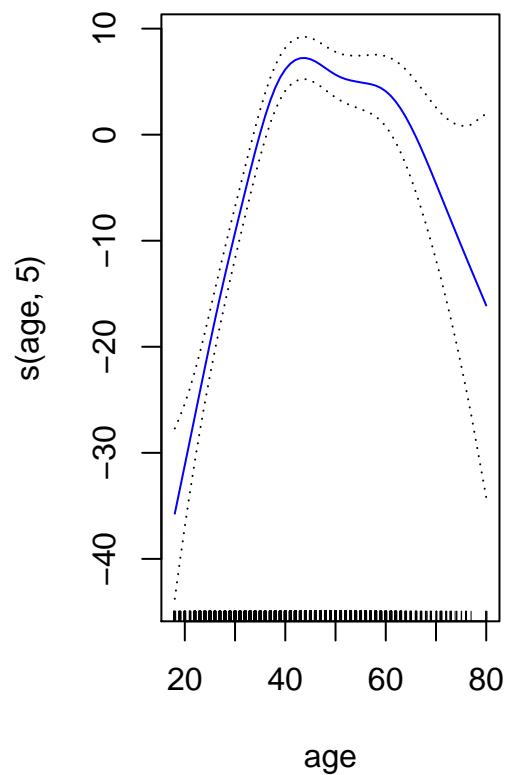
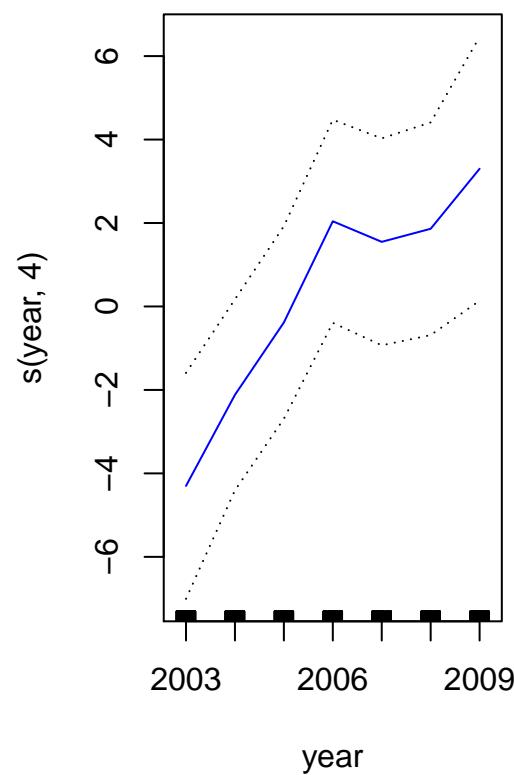
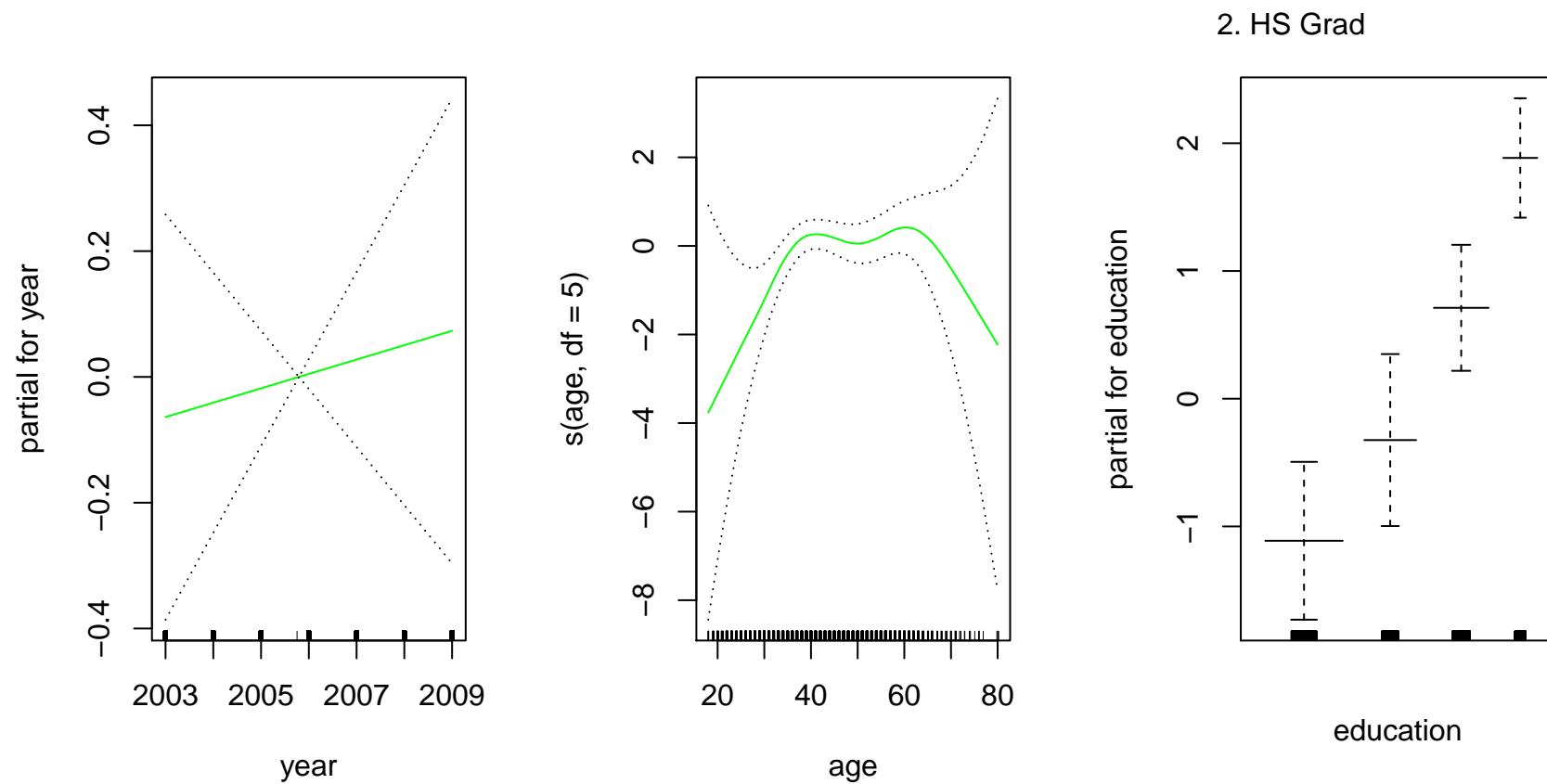


Figure 6.13: For the Wage data, the logistic regression GAM is fit to the binary response $I(\text{wage} > 250)$. Each plot displays the fitted function and pointwise standard errors. The first function is linear in year, the second function a smoothing spline with five degrees of freedom in age, and the third a step function for education. There are very wide standard errors for the first level < HS of education.



Part IV

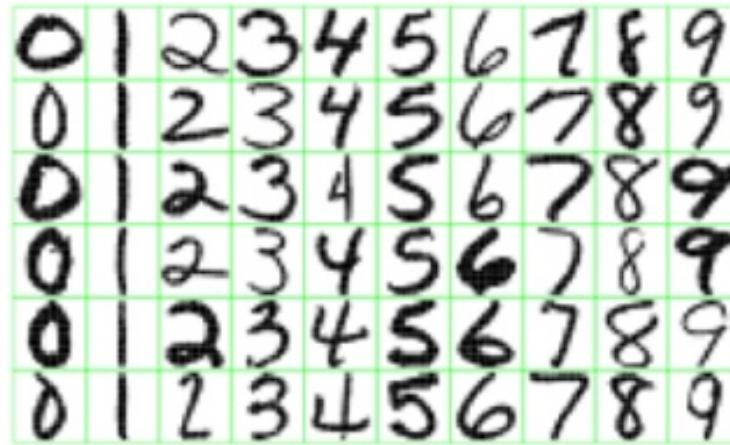
Supervised learning: Classification

Chapter 7 Classification

7.1 Examples

- (Email Spam)
 - We want to predict whether an email is a spam and should be delivered to the Junk folder.
 - The raw data comprises only the text part but ignores all images. Text is a simple sequence of words which is the input (X). Goal is to predict the binary response Y : spam or not.
- (Handwritten Digit Recognition)
 - We want to identify images of single digits 0 - 9 correctly.
 - The raw data comprises images that are scaled segments from five digit ZIP codes.
 - In the diagram below every green box is one image.
 - The original images are very small, containing only 16×16 pixels.
 - For convenience the images below are enlarged, hence the pixelation or 'boxiness' of the numbers. Every image is to be identified as 0 or 1 or 2 ... or 9. Since the numbers are handwritten, the task is not trivial. For instance, a '5' sometimes can very much look like a '6', and '7' is sometimes confused with '1'.

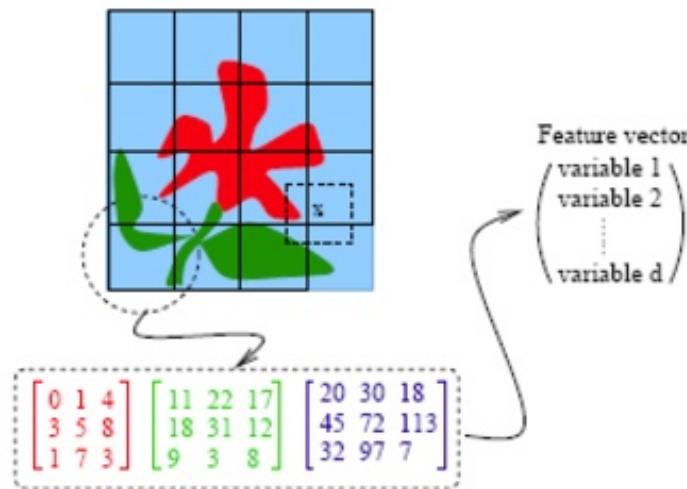
Figure 7.1: Handwritten digits



- To the computer, an image is a matrix, and every pixel in the image corresponds to one entry in the matrix. Every entry is an integer ranging from a pixel intensity of 0 (black) to 255 (white). Hence the raw data can be submitted to the computer directly without any feature extraction. The image matrix was scanned row by row and then arranged into a large 256 dimensional vector. This is used as the input to train the classifier. Note that this is also a supervised learning algorithm where Y , the response, is multi-level and can take 10 values.
- **(Image segmentation)**
 - Here is a more complex example of image processing problem. The satellite images are to be identified into man-made or natural regions. For instance in the aerial images shown below, buildings are labeled as man-made, and the vegetation areas are labeled as natural.
 - These grayscale images are much larger than the previous example. These images are 512×512 pixels and again, because these are grayscale images we can present pixel intensity with numbers 0 to 255.

- In the previous example of hand-written image identification, because of small size of the images, no feature extraction was done. However in this problem feature extraction is necessary. A standard method of feature extraction in an image processing problem is to divide images into blocks of pixels, or to form a neighborhood around each pixel. As is shown in the following diagram, after dividing the images into blocks of pixels or forming a neighborhood around each pixel, each block may be described by several features. As we have seen in the previous example, grayscale images can be represented by one matrix. Every entry in a greyscale image is an integer ranging from a pixel intensity of 0 (black) to 255 (white). Color images are represented by values of RGB (red, green and blue). Color images therefore are represented by 3 such matrices as seen below.

Figure 7.2: Color image is represented by RGB matrices



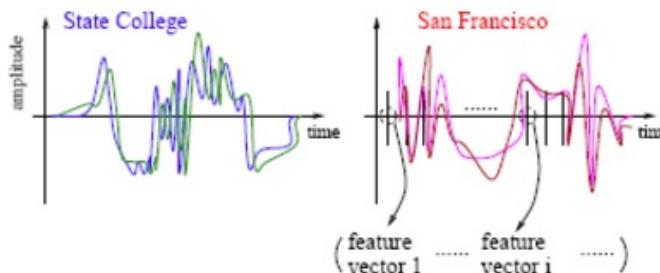
- For each block, a few features (or statistics) may be computed using the color vectors for the pixels in the block. This set forms a feature vector for every block.
- Examples of features:

Average of R, G and B values for pixels in one block Variance of the brightness of the pixels (brightness is the average of RGB color values). Small variance indicates the block is visually smooth. The feature vectors for the blocks sometimes are treated as independent samples from an unknown distribution. Ignoring the spatial dependence among feature vectors results in performance loss. To make the learning algorithm efficient the spatial dependence needs to be exploited. Only then the accuracy in classification will improve.

- **(Speech Recognition)**

- Another interesting example of data mining deals with speech recognition. For instance, if you call the University Park Airport, the system might ask you your flight number, or your origin and destination cities. The system does a very good job recognizing city names. This is a classification problem, in which each city name is a class. The number of classes is very big, but finite.

Figure 7.3: Speech data



- The raw data involves voice amplitude sampled at discrete time points (a time sequence), which may be represented in the waveforms as shown above. In speech recognition, a very popular method is the Hidden Markov Model.
- At every time point, one or more features, such as frequencies, are computed. The speech signal essentially becomes a sequence of frequency vectors. This sequence is assumed to be an instance of a hidden Markov model (HMM). An HMM can be estimated using multiple sample sequences under the same class (e.g., city name).

- Hidden Markov Model (HMM) Methodology: HMM captures the time dependence of the feature vectors. The HMM has unspecified parameters that need to be estimated. Based on the sample sequences, model estimation takes place and a HMM is obtained. This HMM is like a mathematical signature for each word. Each city name, for example, will have a different signature. In the diagram above the signatures corresponding to State College and San Francisco are compared. It is possible that several models are constructed for one word or phrase. For instance, there may be a model for a female voice as opposed to another for a male voice.
- When a customer calls in for information and utters origin or destination city pairs, the system computes the likelihood of what the customer uttered under possibly thousands of models. The system finds the HMM that yields the maximum likelihood and identifies the word as the one associated with that HMM.

- **(DNA Expression Microarray)**

- Our goal here is to identify disease or tissue types based on the gene expression levels.
- For each sample taken from a tissue of a particular disease type, the expression levels of a very large collection of genes are measured. The input data goes through a data cleaning process. Data cleaning may include, but is certainly not limited to, normalization, elimination of noise and perhaps log-scale transformations. Large volume of literature exists on the topic of cleaning microarray data.
- In the example considered 96 samples were taken from 9 classes or types of tissues. It was expensive to collect the tissue samples, at least in the early days. Therefore, the sample size is often small but the dimensionality of data is very high. Every sample is measured on 4026 genes. very often microarray data analysis has its own challenges with small number of observations and very large umber of features from each observation.

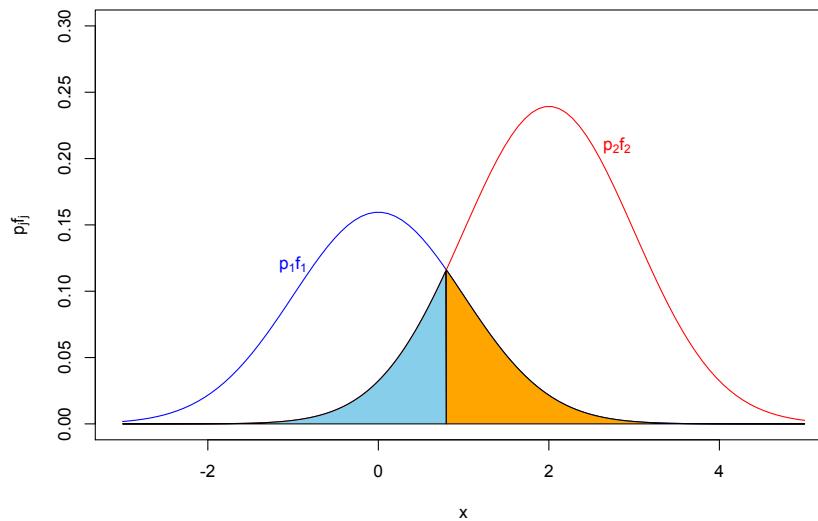
- **(DNA Sequence Classification)**

- Each genome is made up of DNA sequences and each DNA segment has specific biological functions. However there are DNA segments which are non-coding, i.e. they do not have any biological function (or their functionalities are not yet known). One problem in DNA sequencing is to label the sampled segments as coding or non-coding (with biological function or without).

- The raw DNA data comprises sequences of letters, e.g., A, C, G, T for each of the DNA sequences. One method of classification assumes the sequences to be realizations of random processes. Different random processes are assumed for different classes of sequences.

7.2 Classification analysis

- Let $f_i, i = 1, 2$ be the density functions of two groups, denoted by Class 1 (or group 1) and Class 2 (or group 2).



- Define the misclassification probabilities $P(j|i)$ to be the probability that an observation of Class i is misclassified into Class j .
- Define the misclassification cost $c(1|2)$ and $c(2|1)$. The quantity $c(1|2)$ is the cost that we must pay by misclassification of an observation in Class 2 to Class 1.

- Define **the expected cost of misclassification (ECM)**

$$ECM = c(2|1)P(2|1)p_1 + c(1|2)P(1|2)p_2 \quad (7.1)$$

where $p_1 = P(1)$ is the proportion of group 1 and $p_2 = P(2)$ is the proportion of group 2: they are called the prior probabilities of Class 1 and Class 2.

- Let R_1 and R_2 be the classification regions for group 1 and 2, respectively.

- (Minimizing ECM) We can find the classification regions that minimize ECM :

$$\begin{aligned} ECM &= c(2|1)P(2|1)p_1 + c(1|2)P(1|2)p_2 \\ &= c(2|1)p_1 \int_{R_2} f_1(\mathbf{x})d\mathbf{x} + c(1|2)p_2 \int_{R_1} f_2(\mathbf{x})d\mathbf{x} \\ &= c(2|1)p_1 + \int_{R_1} [c(1|2)p_2 f_2(\mathbf{x}) - c(2|1)p_1 f_1(\mathbf{x})] d\mathbf{x} \end{aligned} \quad (7.2)$$

$$R_1 = \{\mathbf{x} : c(1|2)p_2 f_2(\mathbf{x}) - c(2|1)p_1 f_1(\mathbf{x}) < 0\} \Leftrightarrow \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} > \frac{c(1|2)p_2}{c(2|1)p_1}$$

$$R_2 = \{\mathbf{x} : c(1|2)p_2 f_2(\mathbf{x}) - c(2|1)p_1 f_1(\mathbf{x}) > 0\} \Leftrightarrow \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} < \frac{c(1|2)p_2}{c(2|1)p_1}$$

- (Minimizing TPM) We may minimize the total probability of misclassification (TPM):

$$TPM = p_1 \int_{R_2} f_1(\mathbf{x}) d\mathbf{x} + p_2 \int_{R_1} f_2(\mathbf{x}) d\mathbf{x}$$

It is the same as ECM when $c(1|2) = c(2|1) = 1$

$$\begin{aligned} R_1 : \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} &> \frac{p_2}{p_1} \Leftrightarrow \frac{p_1 f_1(\mathbf{x})}{f(\mathbf{x})} > \frac{p_2 f_2(\mathbf{x})}{f(\mathbf{x})} \\ R_2 : \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} &< \frac{p_2}{p_1} \Leftrightarrow \frac{p_1 f_1(\mathbf{x})}{f(\mathbf{x})} < \frac{p_2 f_2(\mathbf{x})}{f(\mathbf{x})} \end{aligned} \tag{7.3}$$

It is also called the maximum posterior probability (MPP) classification and the rule is called a Bayes's classifier.

- Note the posterior probability is written by $\tau_i(\mathbf{x}) = \frac{p_i f_i(\mathbf{x})}{\sum_m p_m f_m(\mathbf{x})}$. A Bayes classifier assigns an observation \mathbf{x} into Class i if $\tau_i(\mathbf{x}) \geq \max_m \tau_m(\mathbf{x})$. A Bayes classifier is minimizing the total probability of misclassification.

7.3 Bayes classifier

- The test error rate is minimized, on average, by a very simple classifier that assigns each observation to the most likely class, given its predictor values.
- Bayes classifier assigns a test observation with predictor vector \mathbf{x} to the class j for which $\tau_j(\mathbf{x}) = P(Y = j|\mathbf{x})$ is largest.

$$\hat{y} = \operatorname{argmax}_j P(Y = j|\mathbf{x}) \quad (7.4)$$

Note

$$\begin{aligned}\hat{y} &= \operatorname{argmax}_j P(Y = j|\mathbf{x}) \\ &= \operatorname{argmax}_j \left\{ \frac{P(\mathbf{x}|Y = j)P(Y = j)}{P(\mathbf{x})} \right\} \\ &= \operatorname{argmax}_j \{P(\mathbf{x}|Y = j)P(Y = j)\}\end{aligned} \quad (7.5)$$

- $P(Y = j|\mathbf{x}_0)$ is the **posterior probability** that $Y = j$, given the observed predictor vector \mathbf{x}_0 .

Estimating the prior probabilities

- Let n be the number of observation in the training data and let n_j be the number of observations that $Y = j$ in the training data.
- The MLEs of the prior probabilities are

$$\hat{P}(Y = j) = \frac{n_j}{n} \quad (7.6)$$

Estimating the likelihood for quantitative predictors

- When the predictors are quantitative, it is common to assume a multivariate normal distribution of \mathbf{x} given $Y = j$:

$$f_j(\mathbf{x}) = \frac{1}{2\pi^{p/2}|\Sigma_j|^{1/2}} \exp \left\{ -\frac{(\mathbf{x} - \boldsymbol{\mu}_j)' \Sigma_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j)}{2} \right\} \quad (7.7)$$

- Let \mathcal{X} be the training data.

$$\hat{\boldsymbol{\mu}}_j = \frac{1}{n_j} \sum_{(y_i, \mathbf{x}_i) \in \mathcal{X}, y_i=j} \mathbf{x}_i \quad (7.8)$$

$$\hat{\Sigma}_j = \frac{1}{n_j} \sum_{(y_i, \mathbf{x}_i) \in \mathcal{X}} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j) (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j)^T \quad (7.9)$$

They are MLEs of the multivariate normal distribution.

Estimating the likelihood for qualitative predictors

$$\hat{P}(x_1^*, \dots, x_p^* | Y = j) = \frac{\#\{i : y_i = j, \mathbf{x}_i = \mathbf{x}^*\}}{n_j} \quad (7.10)$$

where $\mathbf{x}^* = (x_1^*, \dots, x_p^*)'$. This is the MLE of a multinomial distribution.

Example 7.3.1

Consider the “iris” data. Split the data at random into a training data set and a test data set with equal size. Assume the four predictors follow a multivariate normal distribution and construct a Bayes’s classifier. Estimate the test misclassification error rate.

```
head(iris)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa

str(iris)

## 'data.frame': 150 obs. of  5 variables:
## $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```



```
n = dim(iris)[1]
set.seed(1)
train <- sample(1:n, n/2)
```

```
iris.train <- iris[train, ]
iris.test <- iris[-train, ]

nl <- nlevels(iris.train[, 5])

mu <- list()
S <- list()
pro <- list()
for (i in 1:nl) {
  mu[[i]] <- apply(iris.train[iris.train$Species == levels(iris.train[, 5])[i],
    1:4], 2, mean)
  S[[i]] <- cov(iris.train[iris.train$Species == levels(iris.train[, 5])[i],
    1:4])
  pro[[i]] <- dim(iris.train[iris.train$Species == levels(iris.train[, 5])[i],
    1:4])[1]/dim(iris.train[, 1:4])[1]
}

dmvnorm <- function(x, mu, sigma) {
  p <- length(mu)
  mu <- c(mu)
  x <- c(x)
  f = 1/(2 * pi)^(p/2)/(det(sigma))^(1/2) * exp(-0.5 * t(x - mu) %*% solve(sigma) %*%
    (x - mu))
  return(f)
}
```

```
PP <- function(x, pro, mu, sigma) {
  K <- length(pro)
  PP <- numeric(K)
  for (i in 1:K) {
    PP[i] <- pro[[i]] * dmvnorm(x, mu[[i]], sigma[[i]])
  }
  PP <- PP/sum(PP)
  return(PP)
}

Y.pred <- numeric(dim(iris.test)[1])
Pprob <- matrix(0, ncol = 3, nrow = length(Y.pred))
for (i in 1:dim(iris.test)[1]) {
  Pprob[i, ] <- PP(unlist(iris.test[i, 1:4]), pro, mu, S)
  Y.pred[i] <- which.max(Pprob[i, ])
}

Y.hat <- character(length(Y.pred))
Y.hat[Y.pred == 1] <- "setosa"
Y.hat[Y.pred == 2] <- "versicolor"
Y.hat[Y.pred == 3] <- "virginica"
# data.frame(iris.test[,5],Y.hat)
table(pred = Y.hat, true = iris.test[, 5])

##           true
## pred      setosa versicolor virginica
```

```
##   setosa      24      0      0
##   versicolor    0     22      0
##   virginica     0      0     29

sum(iris.test[, 5] != Y.hat)

## [1] 0

sum(iris.test[, 5] != Y.hat)/length(Y.hat)

## [1] 0

# The above can be simply performed by qda function in MASS package.

library(MASS)
fitq <- qda(Species ~ ., data = iris.train)
predq <- predict(fitq, newdata = iris.test)
mean(predq$class != iris.test$Species)

## [1] 0
```

Example 7.3.2

Consider the “iris” data. Split the data at random into a training data set and a test data set with equal size.

Table 7.1: Discretized sepal length and sepal width

Sepal Length		Sepal Width	
Interval	Class	Interval	Class
[4.3, 5.1]	Very Short	[2.0, 2.8]	Short
(5.1, 5.8]	Short	(2.8, 3.3]	Medium
(5.8, 6.4]	Long	(3.3, 4.4]	Long
(6.4, 7.9]	Very Long		

Using the discretized sepal length and sepal width and construct a Bayes’ classifier. Calculate the test misclassification error rate.

```
n <- dim(iris)[1]
fsl <- rep("VL", n)
fsl[iris[, 1] <= 6.4] <- "L"
fsl[iris[, 1] <= 5.8] <- "S"
fsl[iris[, 1] <= 5.1] <- "VS"
fsl <- factor(fsl)

fsw <- rep("L", n)
fsw[iris[, 2] <= 3.3] <- "M"
fsw[iris[, 2] <= 2.8] <- "S"
```

```
fsw <- factor(fsw)

irisf.train <- data.frame(fsl = fsl[train], fsw = fsw[train], Y = iris[train,
  5])
with(irisf.train[irisf.train$Y == "setosa", ], table(fsl, fsw))

##      fsw
## fsl   L  M  S
##   L   0  0  0
##   S   6  0  0
##   VL  0  0  0
##   VS  7 12  1

with(irisf.train[irisf.train$Y == "versicolor", ], table(fsl, fsw))

##      fsw
## fsl   L  M  S
##   L   1  4  3
##   S   0  4 10
##   VL  0  3  1
##   VS  0  0  2

with(irisf.train[irisf.train$Y == "virginica", ], table(fsl, fsw))

##      fsw
## fsl   L M S
##   L   2 1 2
```

```
##   S 0 0 4
##   VL 1 7 3
##   VS 0 0 1

with(irisf.train, table(fsl, fsw))

##      fsw
## fsl   L  M  S
##   L   3  5  5
##   S   6  4 14
##   VL  1 10  4
##   VS  7 12  4

M <- list()
M[[1]] <- with(irisf.train[irisf.train$Y == "setosa", ], table(fsl, fsw))/with(irisf.train,
  table(fsl, fsw))
M[[2]] <- with(irisf.train[irisf.train$Y == "versicolor", ], table(fsl, fsw))/with(irisf.train,
  table(fsl, fsw))
M[[3]] <- with(irisf.train[irisf.train$Y == "virginica", ], table(fsl, fsw))/with(irisf.train,
  table(fsl, fsw))

Mclass <- apply(simplify2array(M), c(1, 2), which.max)

irisf.test <- data.frame(fsl = fsl[-train], fsw = fsw[-train], Y = iris[-train,
  5])
M.test <- list()
M.test[[1]] <- with(irisf.test[irisf.test$Y == "setosa", ], table(fsl, fsw))
```

```
M.test[[2]] <- with(irisf.test[irisf.test$Y == "versicolor", ], table(fsl, fsw))
M.test[[3]] <- with(irisf.test[irisf.test$Y == "virginica", ], table(fsl, fsw))
with(irisf.test, table(fsl, fsw))

##      fsw
## fsl   L   M   S
##   L   0 11 11
##   S   8  2  5
##   VL  2 16  2
##   VS 10  6  2

m1 <- which(Mclass == 1)
a1 <- sum(M.test[[2]][m1]) + sum(M.test[[3]][m1])
m2 <- which(Mclass == 2)
a2 <- sum(M.test[[1]][m2]) + sum(M.test[[3]][m2])
m3 <- which(Mclass == 3)
a3 <- sum(M.test[[1]][m3]) + sum(M.test[[2]][m3])
(a1 + a2 + a3)/sum(with(irisf.test, table(fsl, fsw)))

## [1] 0.2533333

library(nnet)
fitm <- multinom(Y ~ fsl * fsw, data = irisf.train)

## # weights: 39 (24 variable)
## initial value 82.395922
## iter 10 value 31.386545
```

```
## iter 20 value 28.692701
## iter 30 value 28.669303
## final value 28.669254
## converged

set.seed(12345)
predm <- predict(fitm, newdata = irisf.test)
mean(predm != irisf.test$Y)

## [1] 0.2533333
```

Problems with the Bayes classifier:

- For high dimensional data with a large number of predictors, there are **too many parameters** involved in a multivariate normal distribution and a multinomial distribution.

Example 7.3.3

Suppose there are g categories for a response Y . (1) We have p predictors with a multivariate normal distribution. How many parameters? (2) We have p categorical predictors and each predictor has k categories. How many parameters?

Bayes Error Rate

The Bayes classifier produces the lowest possible test error rate, called the **Bayes error rate**, that is analogous to the irreducible error:

$$\begin{aligned} & 1 - E \left(\max_j \Pr(Y = j | \mathbf{X}) \right) \\ &= 1 - E \left(\max_j \tau_j(\mathbf{X}) \right) \end{aligned}$$

Example 7.3.4

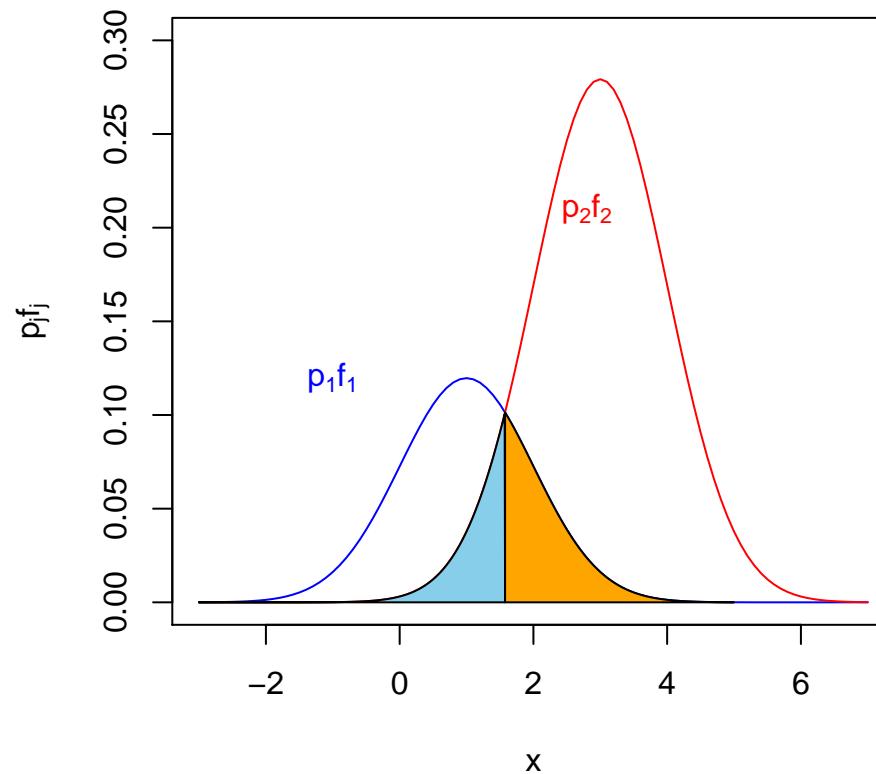
Suppose that Y is a binary response variable that can take on $\{1, 2\}$. Suppose the prior probability for $Y = 1$ is equal to 0.3, that is, $p_1 = P(Y = 1) = 0.3$. Assume X given $Y = 1$ has a normal distribution with mean 1 and variance 1, and X given $Y = 2$ has a normal distribution with mean 3 and variance 1. Calculate the Bayes's error rate.

Bayes's classifier classify an observation into the class $Y = 2$ if

$$\begin{aligned} p_1 f_1(x) &\leq p_2 f_2(x) \\ p_1 \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-1)^2}{2}} &\leq p_2 \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-3)^2}{2}} \\ x &\geq 2 - \frac{1}{2} \log \left(\frac{0.7}{0.3} \right) = 1.576351 \end{aligned}$$

$$\text{Bayes's error rate} = p_1 P(Z > 1.576351 - 1) + p_2 P(Z < 1.576351 - 3) = 0.1387485$$

Figure 7.4: Bayes Error Rate is equal to the sum of skyblue and orange areas. Note the area under blue curve plus the area under red curve equal one.



Q: How to obtain or estimate the posterior probability $\tau_j(\mathbf{x})$?

- Linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA) assume the component density has a multivariate normal distribution.
- The logistic regression model assumes a parametric form of the posterior probability

$$\tau_1(\mathbf{x}) = P(Y = 1|\mathbf{x}) = \frac{\exp(\beta_0 + \boldsymbol{\beta}'\mathbf{x})}{1 + \exp(\beta_0 + \boldsymbol{\beta}'\mathbf{x})}. \quad (7.11)$$

- A naive Bayes classifier assumes conditional independence and makes the estimation easy.
- A k-nearest neighbors (kNN) approximates the posterior probability around a neighborhood of \mathbf{x}_0 :

$$\tau_j(\mathbf{x}_0) = \frac{1}{K} \sum_{\mathbf{x}_m \in \mathcal{N}_K(\mathbf{x}_0)} I(y_m = j) \quad (7.12)$$

7.4 Naive Bayes classifier

- We saw earlier that the full Bayes approach is fraught with estimation related problems, especially with the large number of dimensions.
- The naive Bayes approach makes the simple assumption that all the predictors are conditionally independent.
- This leads to a much simpler, though surprisingly effective classifier in practice.
- The conditional independence assumption immediately implies that the likelihood can be decomposed into a product of dimension-wise probabilities:

$$P(\mathbf{x}|Y = i) = P(x_1, \dots, x_p|Y = i) = \prod_{j=1}^p P(x_j|Y = i) \quad (7.13)$$

- With the normality assumption for quantitative predictors,

$$P(\mathbf{x}|Y = i) = P(x_1, \dots, x_p|Y = i) = \prod_{j=1}^p P(x_j|Y = i) \quad (7.14)$$

$$= \prod_{j=1}^p \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp \left\{ -\frac{(x_j - \mu_{ij})^2}{2\sigma_{ij}^2} \right\} \quad (7.15)$$

$$\hat{\mu}_{ij} = \frac{1}{n_i} \sum_{y_k=i} x_{kj} \quad (7.16)$$

$$\widehat{\sigma^2}_{ij} = \frac{1}{n_i} \sum_{y_k=i} (x_{kj} - \widehat{\mu}_{ij})^2 \quad (7.17)$$

- When the predictors are categorical,

$$\widehat{P}(x_j^*|Y = i) = \frac{\#\{k : x_{kj} = x_j^*, y_k = i\}}{n_i} \quad (7.18)$$

Example 7.4.1

Consider the “iris” data. Split the data at random into a training data set and a test data set with equal size. Assume the four predictors follow a multivariate normal distribution and construct a naive Bayes’s classifier. Estimate the test misclassification error rate.

```
# naive Bayes
library(e1071)
fit <- naiveBayes(Species ~ ., data = iris.train)
pred.iris <- predict(fit, iris.test)
iris.test[, 5]

## [1] setosa    setosa    setosa    setosa    setosa    setosa
## [7] setosa    setosa    setosa    setosa    setosa    setosa
## [13] setosa   setosa    setosa    setosa    setosa    setosa
## [19] setosa   setosa    setosa    setosa    setosa    setosa
## [25] versicolor versicolor versicolor versicolor versicolor
## [31] versicolor versicolor versicolor versicolor versicolor
## [37] versicolor versicolor versicolor versicolor versicolor
## [43] versicolor versicolor versicolor versicolor virginica  virginica
## [49] virginica  virginica  virginica  virginica  virginica  virginica
## [55] virginica  virginica  virginica  virginica  virginica  virginica
## [61] virginica  virginica  virginica  virginica  virginica  virginica
## [67] virginica  virginica  virginica  virginica  virginica  virginica
## [73] virginica  virginica  virginica
## Levels: setosa versicolor virginica

table(pred = pred.iris, true = iris.test[, 5])
```

```

##          true
## pred      setosa versicolor virginica
##   setosa     24        0        0
##   versicolor    0       21        2
##   virginica     0        1       27

mean(pred.iris != iris.test[, 5])

## [1] 0.04

```

Example 7.4.2

Consider the “iris” data. Split the data at random into a training data set and a test data set with equal size.

Table 7.2: Discretized sepal length and sepal width

Sepal Length		Sepal Width	
Interval	Class	Interval	Class
[4.3, 5.1]	Very Short	[2.0, 2.8]	Short
(5.1, 5.8]	Short	(2.8, 3.3]	Medium
(5.8, 6.4]	Long	(3.3, 4.4]	Long
(6.4, 7.9]	Very Long		

Using the discretized sepal length and sepal width and construct a naive Bayes’ classifier. Estimate the test misclassification error rate.

```
library(naivebayes)
fit1 <- naive_bayes(Y ~ ., data = irisf.train)
pred.irisf <- predict(fit1, irisf.test)
table(pred = pred.irisf, true = irisf.test[, 3])

##          true
## pred      setosa versicolor virginica
##   setosa     24        0        0
##   versicolor  0       17       14
##   virginica   0        5       15

mean(pred.irisf != irisf.test[, 3])
## [1] 0.2533333

# two categorical and two continuous predictors
nirisf.train <- irisf.train
nirisf.train$Petal.Length <- iris.train$Petal.Length
nirisf.train$Petal.Width <- iris.train$Petal.Width
nirisf.test <- irisf.test
nirisf.test$Petal.Length <- iris.test$Petal.Length
nirisf.test$Petal.Width <- iris.test$Petal.Width
fitn <- naive_bayes(Y ~ ., data = nirisf.train)
pred.nirisf <- predict(fitn, nirisf.test)
table(pred = pred.nirisf, true = nirisf.test$Y)

##          true
```

```
## pred      setosa versicolor virginica
##   setosa      24       0       0
##   versicolor   0      21       2
##   virginica    0       1      27

mean(pred.nirisf != nirisf.test$Y)

## [1] 0.04
```

7.5 LDA and QDA: Classification for multivariate normal distributions

- (When $\Sigma_1 = \Sigma_2 = \Sigma$: LDA or Linear Classification Rule) Note

$$f_i(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)' \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) \right)$$

$$R_1 : \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} = \exp \left((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)' \Sigma^{-1} \mathbf{x} - (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)' \Sigma^{-1} \left(\frac{\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2}{2} \right) \right) \geq \frac{c(1|2)p_2}{c(2|1)p_1}$$

Assign \mathbf{x} to π_1 if $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)' \Sigma^{-1} \mathbf{x} - (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)' \Sigma^{-1} \left(\frac{\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2}{2} \right) \geq \log \left(\frac{c(1|2)p_2}{c(2|1)p_1} \right)$

(7.19)

Estimated linear classification rule minimizing ECM:

$$\text{Assign } \mathbf{x} \text{ to } \pi_1 \text{ if } (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)' \mathbf{S}_{\mathbf{p}}^{-1} \mathbf{x} - (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)' \mathbf{S}_{\mathbf{p}}^{-1} \left(\frac{\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2}{2} \right) \geq \log \left(\frac{\mathbf{c}(1|2)\mathbf{p}_2}{\mathbf{c}(2|1)\mathbf{p}_1} \right)$$
(7.20)

This is the same as Fisher's linear discriminant function if $\frac{c(1|2)p_2}{c(2|1)p_1} = 1$.

- (When $\Sigma_1 \neq \Sigma_2$: QDA or Quadratic Classification Rule)

$$R_1 : \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} = \left(\frac{|\Sigma_2|}{|\Sigma_1|} \right)^{1/2} \exp \left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)' \Sigma_1^{-1} (\mathbf{x} - \boldsymbol{\mu}_1) + \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_2)' \Sigma_2^{-1} (\mathbf{x} - \boldsymbol{\mu}_2) \right) \geq \frac{c(1|2)p_2}{c(2|1)p_1}$$

Assign \mathbf{x} to π_1 if

$$\begin{aligned} & -\frac{1}{2}\mathbf{x}'(\Sigma_1^{-1} - \Sigma_2^{-1})\mathbf{x} + (\boldsymbol{\mu}_1'\Sigma_1^{-1} - \boldsymbol{\mu}_2'\Sigma_2^{-1})\mathbf{x} - \frac{1}{2}\log \left(\frac{|\Sigma_1|}{|\Sigma_2|} \right) - \frac{1}{2}(\boldsymbol{\mu}_1'\Sigma_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2'\Sigma_2^{-1}\boldsymbol{\mu}_2) \\ & \geq \log \left(\frac{c(1|2)p_2}{c(2|1)p_1} \right) \end{aligned} \quad (7.21)$$

Estimated quadratic classification rule minimizing ECM:

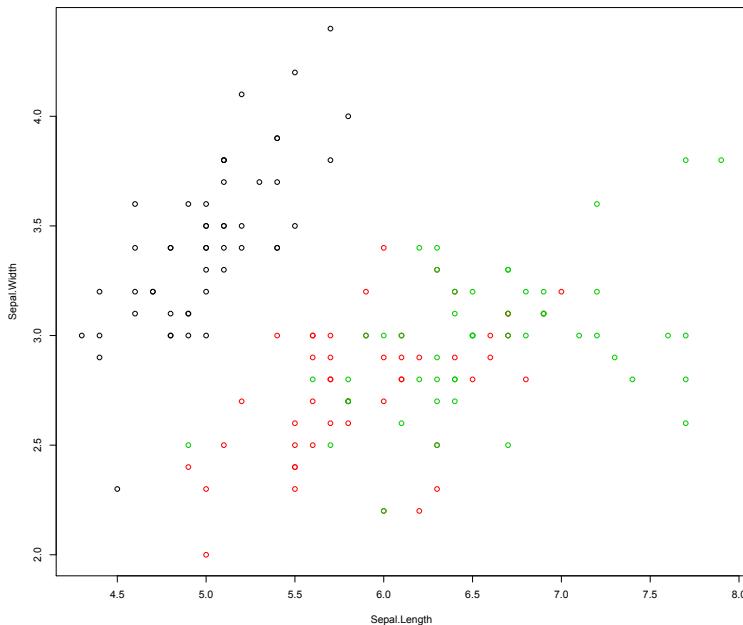
Assign \mathbf{x} to π_1 if

$$\begin{aligned} & -\frac{1}{2}\mathbf{x}'(\mathbf{S}_1^{-1} - \mathbf{S}_2^{-1})\mathbf{x} + (\bar{\mathbf{x}}_1'\mathbf{S}_1^{-1} - \bar{\mathbf{x}}_2'\mathbf{S}_2^{-1})\mathbf{x} - \frac{1}{2}\log \left(\frac{|\mathbf{S}_1|}{|\mathbf{S}_2|} \right) - \frac{1}{2}(\bar{\mathbf{x}}_1'\mathbf{S}_1^{-1}\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2'\mathbf{S}_2^{-1}\bar{\mathbf{x}}_2) \\ & \geq \log \left(\frac{c(1|2)p_2}{c(2|1)p_1} \right) \end{aligned} \quad (7.22)$$

- We need to specify the prior probabilities p_1 and p_2 . Otherwise, the sample proportions \hat{p}_1 and \hat{p}_2 are used for the classification rules.

Example 7.5.1 (Iris data)

Use “Sepal length” and “Sepal width” to classify “Species” of iris data by applying the LDA. Estimate the test classification error rate.



```
library(MASS)
iris12 <- iris[, c(1, 2, 5)]
attach(iris12)
# plot(iris12,col=iris12[,3]) plot(Sepal.Length,Sepal.Width,col=iris12[,3])
```

```
n <- dim(iris12)[1]
set.seed(1234)
train <- sample(n, n/2)
# LDA
fit <- lda(Species ~ ., data = iris12, subset = train)
pred.lda <- predict(fit, iris12[-train, ])$class
table(pred.lda, iris12[-train, 3])

##
## pred.lda      setosa versicolor virginica
##   setosa        19         0         0
##   versicolor     0        24         7
##   virginica      0         6        19

mean(pred.lda != iris12[-train, 3])
## [1] 0.1733333

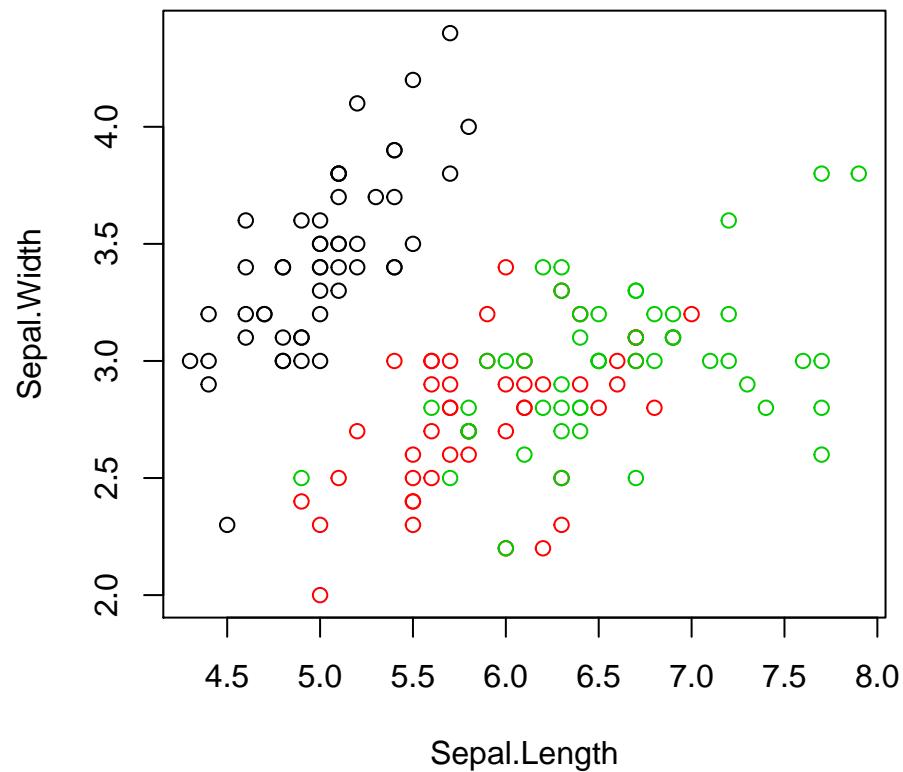
# QDA
fit2 <- qda(Species ~ ., data = iris12, subset = train)
pred.qda <- predict(fit2, iris12[-train, ])$class
table(pred.qda, iris12[-train, 3])

##
## pred.qda      setosa versicolor virginica
##   setosa        19         0         0
##   versicolor     0        23         9
##   virginica      0         7        17
```

```
mean(pred.qda != iris12[-train, 3])  
## [1] 0.2133333  
  
# LOOCV  
fitloocv <- lda(Species ~ ., data = iris12, cv = T)  
predloocv <- predict(fitloocv, iris12)$class  
table(predloocv, iris12[, 3])  
  
##  
## predloocv setosa versicolor virginica  
##   setosa      49          0          0  
##   versicolor    1         36         15  
##   virginica     0         14         35  
  
mean(predloocv != iris12[, 3])  
## [1] 0.2  
  
fit2loocv <- qda(Species ~ ., data = iris12, cv = T)  
pred2loocv <- predict(fit2loocv, iris12)$class  
table(pred2loocv, iris12[, 3])  
  
##  
## pred2loocv setosa versicolor virginica  
##   setosa      49          0          0  
##   versicolor    1         37         16  
##   virginica     0         13         34
```

```
mean(pred2loocv != iris12[, 3])  
  
## [1] 0.2  
  
# K-fold CV  
K <- 5  
ind <- (1:n)%%K + 1  
set.seed(1234)  
folds <- sample(ind, n)  
predcv <- character(n)  
for (k in 1:K) {  
    fit <- lda(Species ~ ., data = iris12, subset = which(ind != k))  
    predcv[ind == k] <- as.character(predict(fit, iris12[ind == k, ])$class)  
}  
table(predcv, iris12[, 3])  
  
##  
## predcv      setosa versicolor virginica  
##   setosa      49        0        0  
##   versicolor    1       35       13  
##   virginica     0       15       37  
  
mean(predcv != iris12[, 3])  
  
## [1] 0.1933333  
  
detach(iris12)
```

Figure 7.5: Scatter plot of iris data



7.6 Logistics regression

7.6.1 Binary response and simple logistic regression

- Y is a binary response: $Y = 1$ or $Y = 0$.
- Logistic regression for a binary response Y models the probability $P(Y = 1|X)$ that the response belongs to a category.
- We define the logit by

$$\text{logit} [\Pr(Y = 1|X)] = \log \left(\frac{\Pr(Y = 1|X)}{\Pr(Y = 0|X)} \right) = \log \left(\frac{\Pr(Y = 1|X)}{1 - \Pr(Y = 1|X)} \right) \quad (7.23)$$

- Logistic regression model is

$$\log \left(\frac{\Pr(Y = 1|X)}{1 - \Pr(Y = 1|X)} \right) = \beta_0 + \beta_1 X \quad (7.24)$$

For simplicity, we write $p(X) = \Pr(Y = 1|X)$. Then

$$\begin{aligned} \Pr(Y = 1|X) &= p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} \\ \Pr(Y = 0|X) &= 1 - p(X) = \frac{1}{1 + e^{\beta_0 + \beta_1 X}} \end{aligned} \quad (7.25)$$

Example 7.6.1

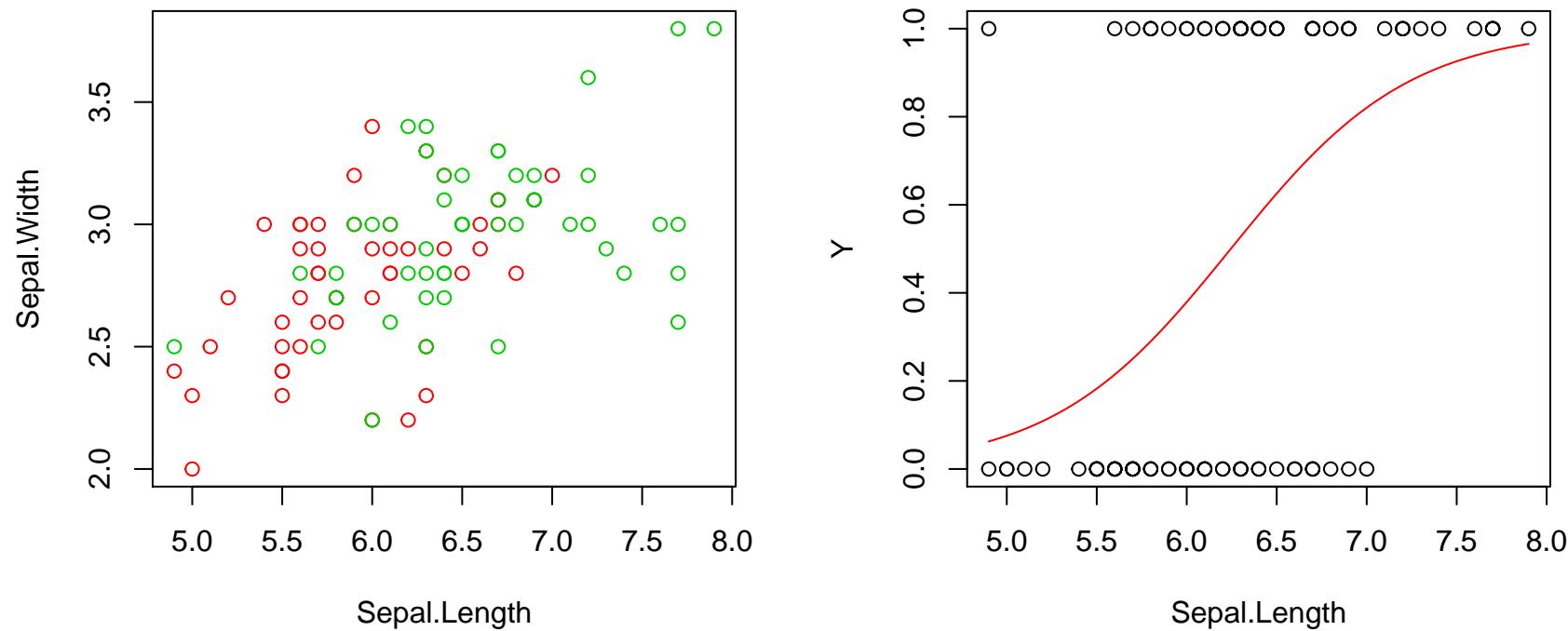
Apply the logistic regression for two classes “ versicolor” and “virginica” using a single predictor “Sepal Length”.

```
# logistic
irisb <- iris[iris$Species != "setosa", c(1, 2, 5)]
# plot(irisb[,1:2],col=irisb[,3])
irisb$Y <- rep(0, dim(irisb)[1])
irisb$Y[irisb$Species == "virginica"] <- 1
fit <- glm(Y ~ Sepal.Length, data = irisb, family = binomial)
summary(fit)

##
## Call:
## glm(formula = Y ~ Sepal.Length, family = binomial, data = irisb)
##
## Deviance Residuals:
##      Min        1Q        Median         3Q        Max 
## -1.85340   -0.90001   -0.04717    0.96861    2.35458 
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)    
## (Intercept) -12.5708    2.9068  -4.325 1.53e-05 ***
## Sepal.Length  2.0129    0.4654   4.325 1.53e-05 ***
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##  
## (Dispersion parameter for binomial family taken to be 1)  
##  
## Null deviance: 138.63 on 99 degrees of freedom  
## Residual deviance: 110.55 on 98 degrees of freedom  
## AIC: 114.55  
##  
## Number of Fisher Scoring iterations: 4  
  
b <- c(coef(fit))  
# with(irisb,plot(Sepal.Length,Y))  
# curve(exp(b[1]+b[2]*x)/(1+exp(b[1]+b[2]*x)),col=2,add=T)
```

Figure 7.6: Scatter plot of iris data with a logistic fitted curve.



Interpretation of logistic model

- The quantity $\frac{\Pr(Y=1|X)}{1-\Pr(Y=1|X)}$ is called the **odds** of $Y = 1$ at X
- Logistic regression: Log-odds of the response = linear in predictors
- What is β_1 ?

$$\begin{aligned} \log \frac{p(X)}{1-p(X)} &= \beta_0 + \beta_1 X, \quad \log \frac{p(X+1)}{1-p(X+1)} = \beta_0 + \beta_1(X+1) \\ \log \frac{p(X+1)(1-p(X))}{(1-p(X+1))p(X)} &= \beta_1, \end{aligned} \tag{7.26}$$

that is, β_1 is the log-odds ratio of the response $Y = 1$ between $X + 1$ and X .

- For instance, if $\beta_1 = 0.4$, then the odds of $Y = 1$ at $X + 1$ increases 50% of the odds of $Y = 1$ at X since $\exp(0.4) \approx 1.5$.

Odds and odds ratio for 2×2 table

- Odds ratio is a measure of association of two variables

	Y=0	Y=1
X=0	$p(0,0)$	$p(0,1)$
X=1	$p(1,0)$	$p(1,1)$

- The odds ratio does NOT depend on sampling methods:

$$\begin{aligned}
 \theta_{XY} &= \frac{p(1,1)p(0,0)}{p(0,1)p(1,0)} \text{ cross-sectional study} \\
 &= \frac{\Pr(Y = 1|X = 1) \Pr(Y = 0|X = 0)}{\Pr(Y = 0|X = 1) \Pr(Y = 1|X = 0)} \text{ prospective study} \\
 &= \frac{\Pr(X = 1|Y = 1) \Pr(X = 0|Y = 0)}{\Pr(X = 0|Y = 1) \Pr(X = 1|Y = 0)} \text{ retrospective study}
 \end{aligned} \tag{7.27}$$

Estimation

- Suppose the following contingency table is obtained from a study

	$Y = 0$	$Y = 1$
$X = 0$	n_{00}	n_{01}
$X = 1$	n_{10}	n_{11}

The MLE for the odds ratio $\hat{\theta}_{XY} = \frac{n_{11}n_{00}}{n_{10}n_{01}}$

- For logistic regression, the MLEs require a numerical calculation to maximize the log-likelihood function

$$\ell(\beta_0, \beta_1) = \sum_{i=1}^n \log \left(\frac{e^{y_i(\beta_0 + \beta_1 x_i)}}{1 + e^{\beta_0 + \beta_1 x_i}} \right) \quad (7.28)$$

- Results of logistic regression for Default data

	Estimate	Std. Error	Z-value	p-value
Intercept	-10.65	0.361	-29.49	< 0.0001
balance	0.00550	0.000220	24.95	< 0.0001

For a hypothesis test $H_0 : \beta_1 = 0$, we may use z -statistic ($z = \frac{\hat{\beta}_1}{se(\hat{\beta}_1)}$). It is similar to linear regression analysis.

- We may predict the probability of default for an individual

$$\begin{aligned}\widehat{\Pr}(\text{default} = \text{Yes} | \text{balance}) &= \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 \times \text{balance}}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 \times \text{balance}}} \\ &= \frac{e^{-10.65 + 0.0055 \times \text{balance}}}{1 + e^{-10.65 + 0.0055 \times \text{balance}}}\end{aligned}$$

The probability of default = 0.00575 for balance \$1,000 while the probability of default = 0.586 for balance \$2,000.

Qualitative predictor

- We may construct dummy (indicator) variables for a qualitative variable as we do in linear regression analysis.
- Results:

	Estimate	Std. Error	Z-value	p-value
Intercept	-3.504	0.07071	-49.55	< 0.0001
student	0.4049	0.11502	3.52	0.000431

•

$$\widehat{\Pr}(\text{default} = \text{Yes} | \text{student} = \text{No}) = \frac{e^{-3.504+0.4049 \times 0}}{1 + e^{-3.504+0.4049 \times 0}} = 0.0292$$

$$\widehat{\Pr}(\text{default} = \text{Yes} | \text{student} = \text{Yes}) = \frac{e^{-3.504+0.4049 \times 1}}{1 + e^{-3.504+0.4049 \times 1}} = 0.0431$$

7.6.2 Multiple logistic regression

- Multiple logistic regression equation: $\log\left(\frac{p(\mathbf{X})}{1-p(\mathbf{X})}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$
- Results of the multiple logistic regression for Default data

	Estimate	Std. Error	Z-value	p-value
Intercept	-10.87	0.4923	-22.08	< 0.0001
student	-0.6468	0.2363	-2.738	0.00619
income	0.000030	0.0000082	0.37	0.71152
balance	0.0057	0.0002319	24.738	< 0.0001

- The coefficient estimate of “student(Yes)” is negative while the estimate for the simple logistic regression was positive since “student” is a confounding variable for “balance” and “default”. The variable “balance” tends to be higher for “student(Yes)” than “student(No)”.
- Prediction using the multiple logistic regression: balance=\$1,500, income=\$40,000

$$\begin{aligned} & \widehat{\Pr}(\text{default} | \text{student} = \text{Yes}, \text{balance} = \$1,500, \text{income} = \$40,000) \\ &= \frac{e^{-10.87 - 0.6468 + 0.00003 \times 40000 + 0.0057 \times 1500}}{1 + e^{-10.87 - 0.6468 + 0.00003 \times 40000 + 0.0057 \times 1500}} = 0.058 \\ & \widehat{\Pr}(\text{default} | \text{student} = \text{No}, \text{balance} = \$1,500, \text{income} = \$40,000) \\ &= \frac{e^{-10.87 + 0.00003 \times 40000 + 0.0057 \times 1500}}{1 + e^{-10.87 + 0.00003 \times 40000 + 0.0057 \times 1500}} = 0.105 \end{aligned}$$

```
library(ISLR)
head(Default)

##   default student  balance    income
## 1      No       No 729.5265 44361.625
## 2      No      Yes 817.1804 12106.135
## 3      No       No 1073.5492 31767.139
## 4      No       No  529.2506 35704.494
## 5      No       No  785.6559 38463.496
## 6      No      Yes 919.5885  7491.559

str(Default)

## 'data.frame': 10000 obs. of  4 variables:
## $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
## $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 1 ...
## $ balance: num  730 817 1074 529 786 ...
## $ income : num  44362 12106 31767 35704 38463 ...

fitb <- glm(default ~ balance, data = Default, family = binomial)
summary(fitb)

##
## Call:
## glm(formula = default ~ balance, family = binomial, data = Default)
##
## Deviance Residuals:
```

```
##      Min     1Q Median     3Q    Max
## -2.2697 -0.1465 -0.0589 -0.0221  3.7589
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.065e+01 3.612e-01 -29.49 <2e-16 ***
## balance      5.499e-03 2.204e-04   24.95 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 2920.6 on 9999 degrees of freedom
## Residual deviance: 1596.5 on 9998 degrees of freedom
## AIC: 1600.5
##
## Number of Fisher Scoring iterations: 8

predict(fitb, data.frame(balance = c(1000, 2000)), type = "response")

##      1         2
## 0.005752145 0.585769370

fits <- glm(default ~ student, data = Default, family = binomial)
summary(fits)

##
```

```
## Call:  
## glm(formula = default ~ student, family = binomial, data = Default)  
##  
## Deviance Residuals:  
##      Min       1Q   Median       3Q      Max  
## -0.2970  -0.2970  -0.2434  -0.2434   2.6585  
##  
## Coefficients:  
##                 Estimate Std. Error z value Pr(>|z|)  
## (Intercept) -3.50413    0.07071  -49.55 < 2e-16 ***  
## studentYes   0.40489    0.11502     3.52 0.000431 ***  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## (Dispersion parameter for binomial family taken to be 1)  
##  
## Null deviance: 2920.6 on 9999 degrees of freedom  
## Residual deviance: 2908.7 on 9998 degrees of freedom  
## AIC: 2912.7  
##  
## Number of Fisher Scoring iterations: 6  
  
predict(fits, data.frame(student = c("Yes", "No")), type = "response")  
  
##           1            2  
## 0.04313859 0.02919501
```

```
fit <- glm(default ~ ., data = Default, family = binomial)
summary(fit)

##
## Call:
## glm(formula = default ~ ., family = binomial, data = Default)
##
## Deviance Residuals:
##      Min      1Q   Median      3Q      Max
## -2.4691 -0.1418 -0.0557 -0.0203  3.7383
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.087e+01  4.923e-01 -22.080 < 2e-16 ***
## studentYes  -6.468e-01  2.363e-01  -2.738  0.00619 **
## balance      5.737e-03  2.319e-04   24.738 < 2e-16 ***
## income       3.033e-06  8.203e-06    0.370  0.71152
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 2920.6 on 9999 degrees of freedom
## Residual deviance: 1571.5 on 9996 degrees of freedom
## AIC: 1579.5
##
```

```
## Number of Fisher Scoring iterations: 8

predict(fit, data.frame(student = c("Yes", "No"), balance = rep(1500, 2), income = rep(40000,
  2)), type = "response")

##           1           2
## 0.05788194 0.10499192
```

7.7 *K* Nearest Neighborhood (KNN)

- In theory we would always like to predict qualitative responses using the Bayes classifier.
- But we do not know $\Pr(Y = j|\mathbf{X}) \Rightarrow$ impossible to construct the Bayes classifier.
- The Bayes classifier serves as an unattainable gold standard against which to compare other methods.
- Many approaches attempt to estimate the conditional distribution of Y given \mathbf{X} , and then classify a given observation to the class with highest estimated probability.
- One such nonparametric method is the K -nearest neighbors (KNN) classifier:
 - We define a distance between two vectors, $d(\mathbf{x}_1, \mathbf{x}_2)$. The KNN approach depends on the choice of the distance.
 - Given a positive integer K and a test observation \mathbf{x}_0 , the KNN classifier first identifies the K points in the training data that are closest to \mathbf{x}_0 , represented by $\mathcal{N}_K(\mathbf{x}_0)$. It then estimates the conditional probability for class j as the fraction of points in \mathcal{N}_0 whose response values equal j :

$$\widehat{P}(Y = j|\mathbf{X} = \mathbf{x}_0) = \frac{1}{K} \sum_{\mathbf{x}_i \in \mathcal{N}_K(\mathbf{x}_0)} I(y_i = j)$$

- Finally, KNN applies Bayes rule and classifies the test observation \mathbf{x}_0 to the class with the largest probability.
- K is usually selected by a cross-validation method.

Figure 7.7: The KNN approach, using $K = 3$, is illustrated in a simple situation with six blue observations and six orange observations. Left: a test observation at which a predicted class label is desired is shown as a black cross. The three closest points to the test observation are identified, and it is predicted that the test observation belongs to the most commonly-occurring class, in this case blue. Right: The KNN decision boundary for this example is shown in black. The blue grid indicates the region in which a test observation will be assigned to the blue class, and the orange grid indicates the region in which it will be assigned to the orange class.

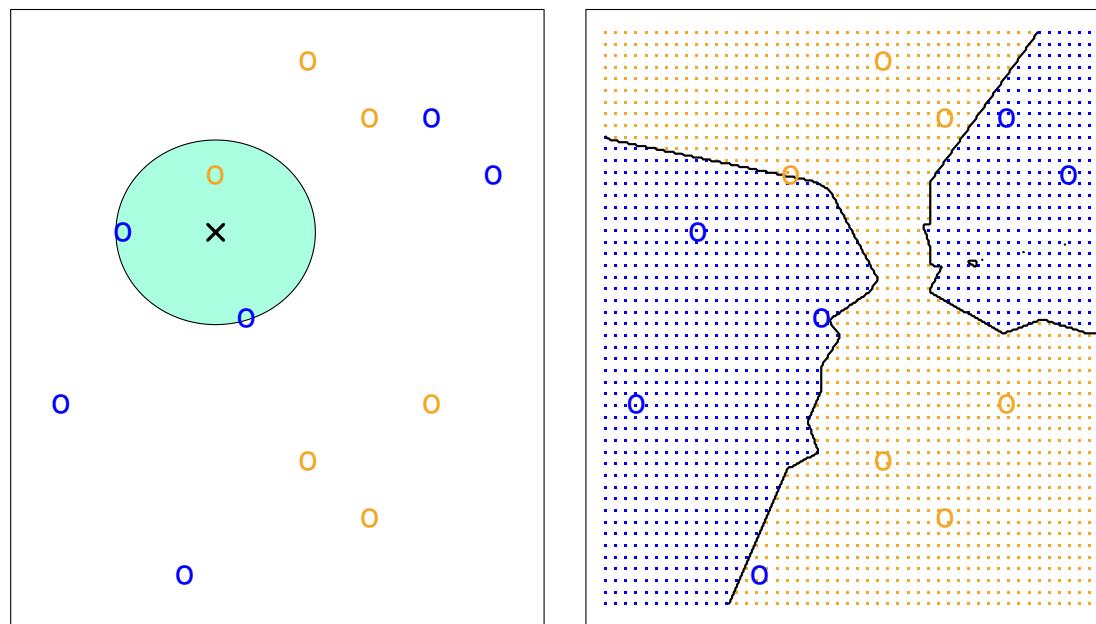


Figure 7.8: A comparison of the KNN decision boundaries (solid black curves) obtained using $K = 1$ and $K = 100$ on the data from a simulated dataset. With $K = 1$, the decision boundary is overly flexible, while with $K = 100$ it is not sufficiently flexible. The Bayes decision boundary is shown as a purple dashed line.

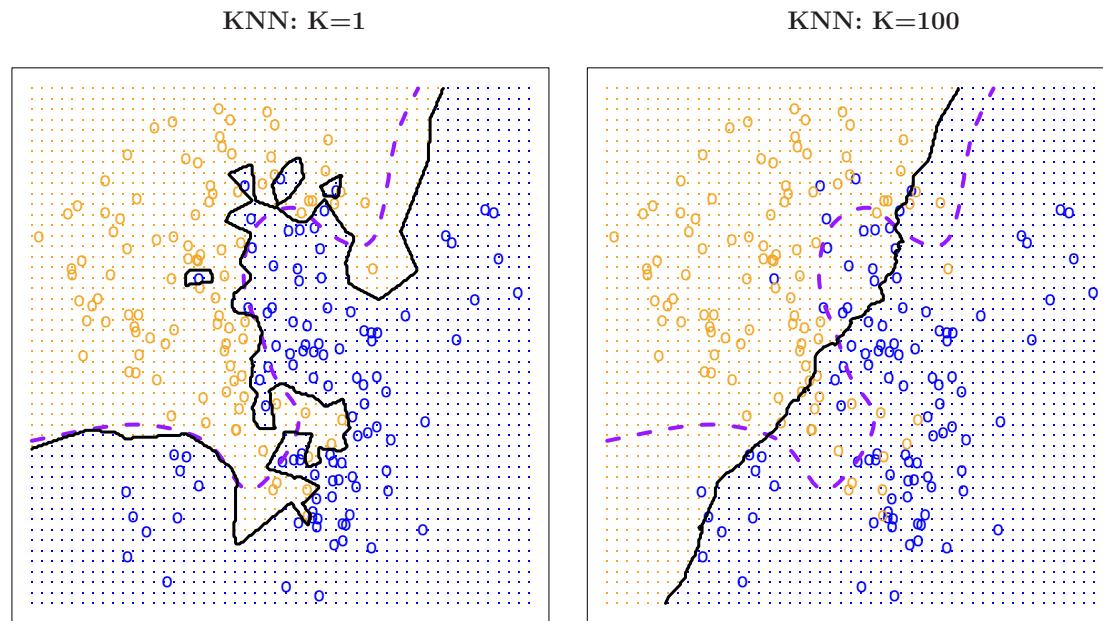
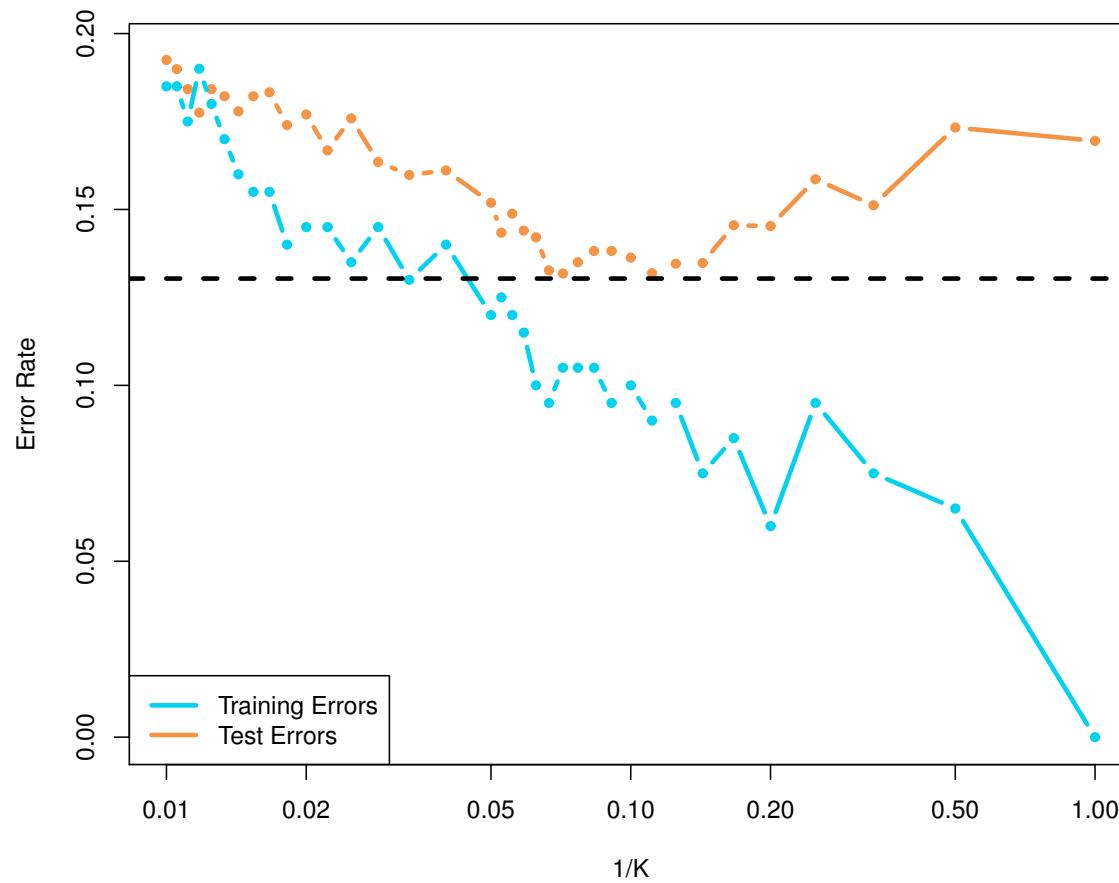


Figure 7.9: The KNN training error rate (blue, 200 observations) and test error rate (orange, 5,000 observations) on the data from Figure 2.13, as the level of flexibility (assessed using $1/K$) increases, or equivalently as the number of neighbors K decreases. The black dashed line indicates the Bayes error rate. The jumpiness of the curves is due to the small size of the training data set.



Example 7.7.1

Consider the “iris” data. Split the data at random into a training data set and a test data set with equal size. Construct KNN classifiers with various $K = 1, \dots, 10$ using the Euclidean distance. Estimate the test misclassification error rate.

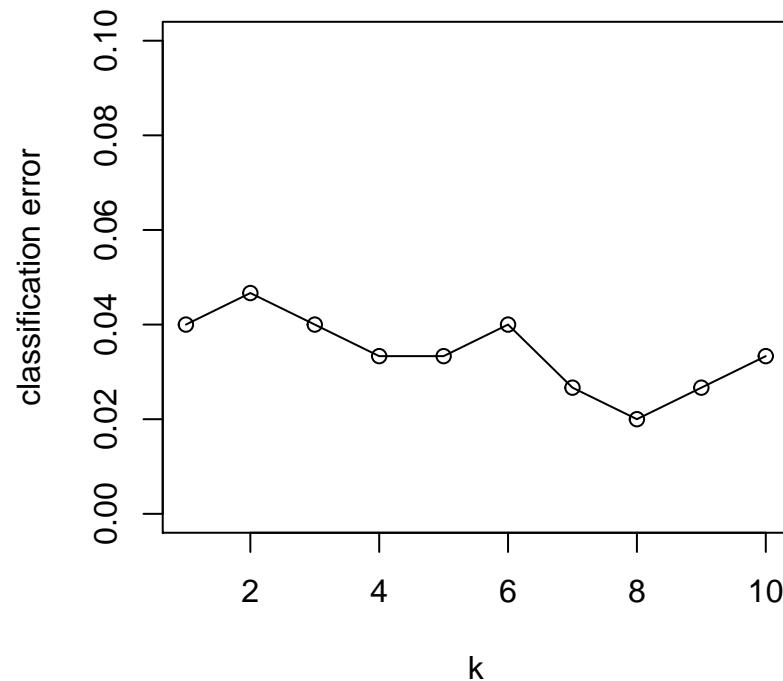
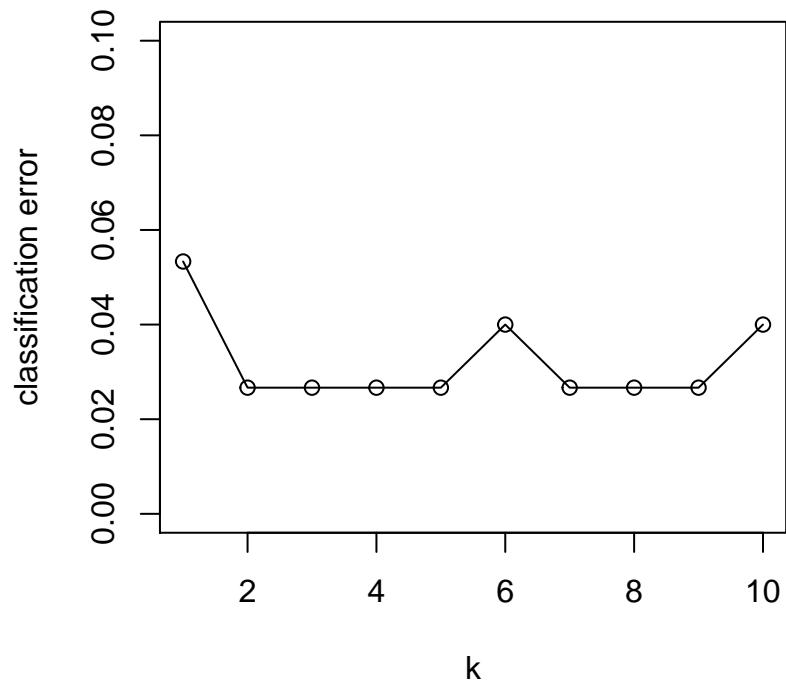
```
library(class)
set.seed(1234)
n <- dim(iris)[1]
train <- sample(n, n/2)
error <- list()
for (i in 1:10) {
  error[[i]] <- mean(knn(iris[train, 1:4], iris[-train, 1:4], iris[train,
    5], k = i) != iris[-train, 5])
}
unlist(error)

## [1] 0.05333333 0.02666667 0.02666667 0.02666667 0.02666667 0.04000000
## [7] 0.02666667 0.02666667 0.02666667 0.04000000

# LOOCV
error.cv <- list()
for (i in 1:10) {
  error.cv[[i]] <- mean(knn.cv(iris[, 1:4], iris[, 5], k = i) != iris[, 5])
}
unlist(error.cv)

## [1] 0.04000000 0.04666667 0.04000000 0.03333333 0.03333333 0.04000000
## [7] 0.02666667 0.02000000 0.02666667 0.03333333
```

Figure 7.10: Classification errors of knn for iris data. Left: hold-out approach, Right: LOOCV.



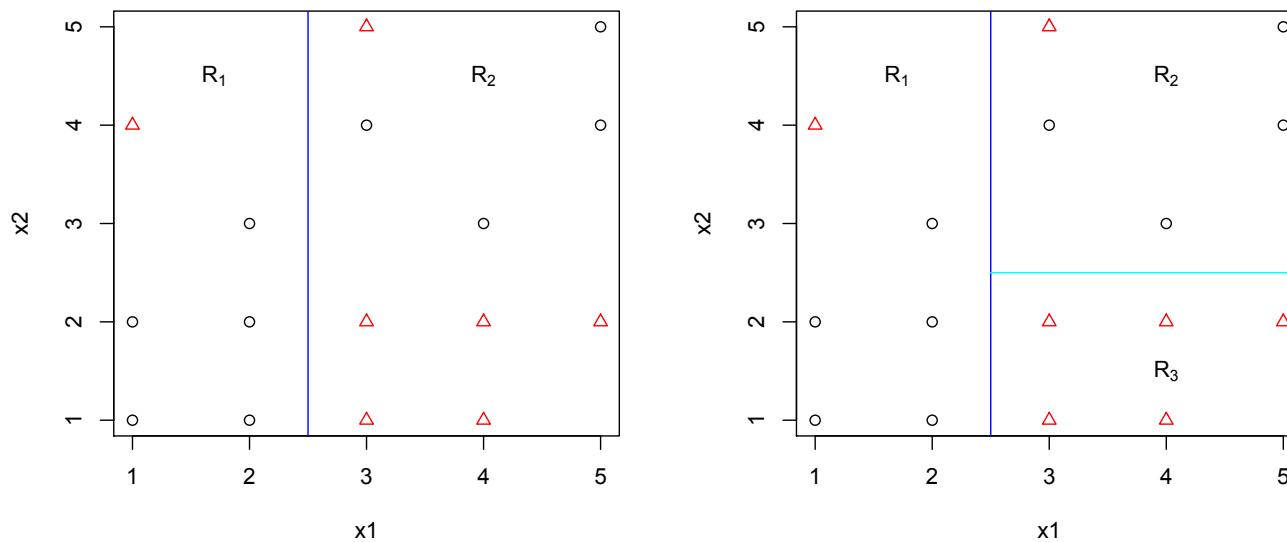
7.8 Classification Trees

Classification And Regression Tree (CART)

- Stratify or segment the predictor space and predict the response by using the mean or mode of the training observations in the region to which it belongs.
- This type of analysis is called decision tree method.
- Tree-based methods are simple and useful for interpretation but the prediction accuracy is poor in general.
- Ensemble methods such as bagging, random forest, and boosting can improve the prediction accuracy at the expense of loss of interpretability.

Classification trees

- Qualitative (categorical) response Y
- Prediction by majority vote (the most commonly occurring class)
- We are often interested in the class proportions at the terminal nodes as well.
- Recursive binary splitting algorithm

Figure 7.11: Recursive binary split of $R = R_1 \cup R_2$ 

Criterion	(R_1, R_2)	R
Classification error rate (E)	$\frac{6}{16} \frac{1}{6} + \frac{10}{16} \frac{4}{10} = \frac{5}{16}$	$\frac{7}{16}$
Gini index	$2[\frac{6}{16} \frac{1}{6} \frac{5}{6} + \frac{10}{16} \frac{4}{10} \frac{6}{10}]$	$2\frac{7}{16} \frac{9}{16}$
Cross-entropy	$-\frac{6}{16} [\frac{1}{6} \log \frac{1}{6} + \frac{5}{6} \log \frac{5}{6}] - \frac{10}{16} [\frac{4}{10} \log \frac{4}{10} + \frac{6}{10} \log \frac{6}{10}]$	$-[\frac{7}{16} \log \frac{7}{16} + \frac{9}{16} \log \frac{9}{16}]$

1. **Classification error rate** is defined by

$$E = 1 - \max_k(\hat{p}_{mk}) \quad (7.29)$$

where \hat{p}_{mk} is the proportion of the training observations in the m th region that are from the k th class.

2. **Gini index** is defined by

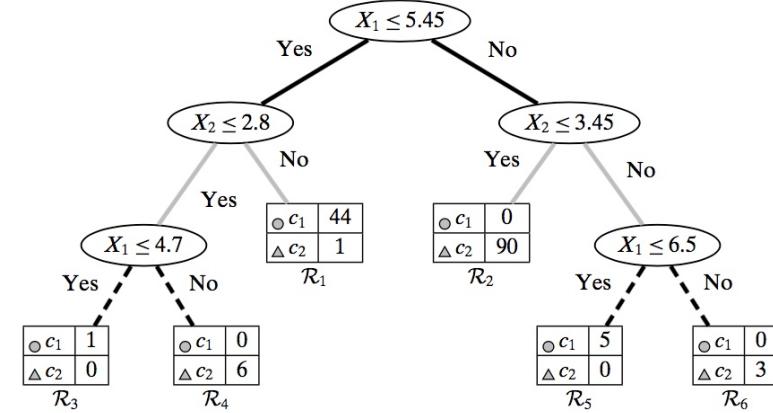
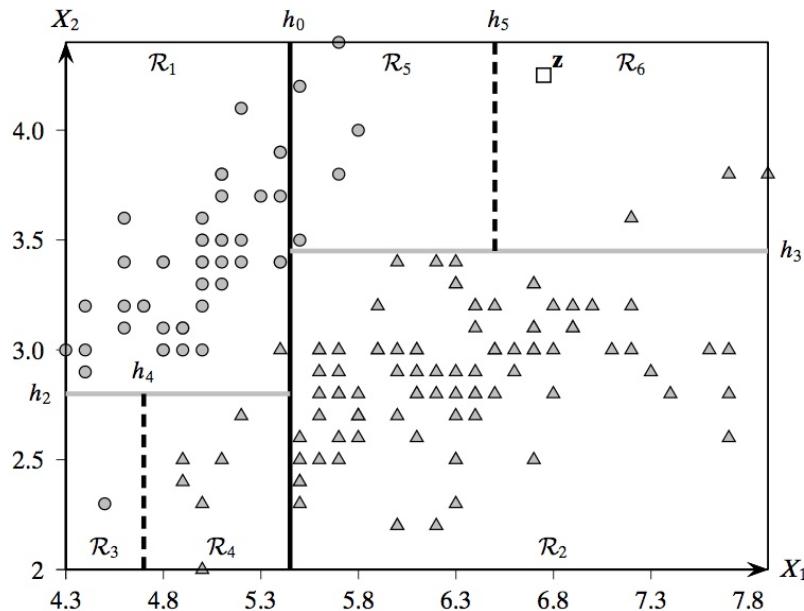
$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}) \quad (7.30)$$

3. **Cross-entropy** is defined by

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk}) \quad (7.31)$$

- The classification error rate is not enough sensitive to grow trees.
- The Gini index and the cross-entropy will take on a small value if all of the \hat{p}_{mk} 's are close to zero or one.
- Any of these three approaches might be used when pruning the tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.

Figure 7.12: Left: Recursive binary split, Right: classification tree



Example 7.8.1 • Heart data set

- These data contain a binary outcome HD for 303 patients who presented with chest pain.
- An outcome value of Yes indicates the presence of heart disease based on an angiographic test, while No means no heart disease.
- There are 13 predictors including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation results in a tree with six terminal nodes.

```
library(tree)
hdata <- read.csv("Heart.csv", header = T)
str(hdata)

## 'data.frame': 303 obs. of 15 variables:
## $ X      : int 1 2 3 4 5 6 7 8 9 10 ...
## $ Age    : int 63 67 67 37 41 56 62 57 63 53 ...
## $ Sex    : int 1 1 1 1 0 1 0 0 1 1 ...
## $ ChestPain: Factor w/ 4 levels "asymptomatic",...: 4 1 1 2 3 3 1 1 1 1 ...
## $ RestBP  : int 145 160 120 130 130 120 140 120 130 140 ...
## $ Chol   : int 233 286 229 250 204 236 268 354 254 203 ...
## $ Fbs    : int 1 0 0 0 0 0 0 0 0 1 ...
## $ RestECG : int 2 2 2 0 2 0 2 0 2 2 ...
## $ MaxHR  : int 150 108 129 187 172 178 160 163 147 155 ...
## $ ExAng  : int 0 1 1 0 0 0 0 1 0 1 ...
## $ Oldpeak : num 2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
## $ Slope   : int 3 2 2 3 1 1 3 1 2 3 ...
## $ Ca      : int 0 3 2 0 0 0 2 0 1 0 ...
```

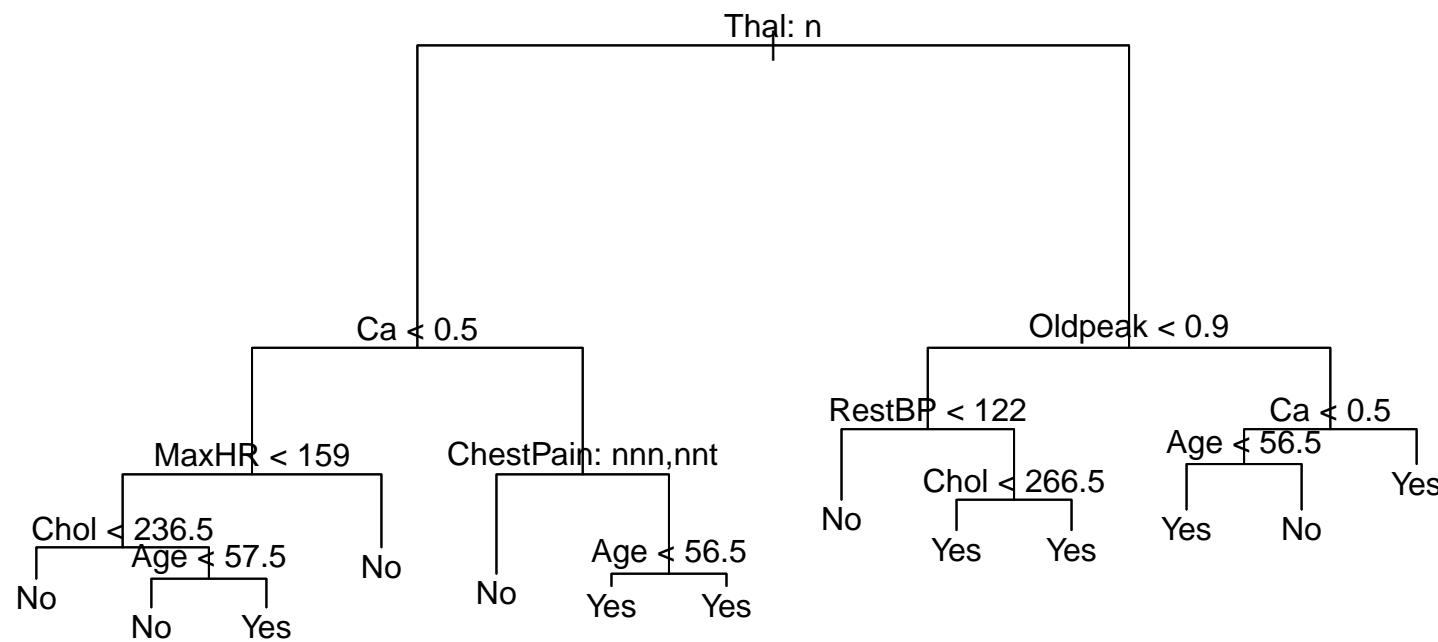
```
## $ Thal      : Factor w/ 3 levels "fixed","normal",...: 1 2 3 2 2 2 2 2 3 3 ...
## $ AHD       : Factor w/ 2 levels "No","Yes": 1 2 2 1 1 1 2 1 2 2 ...

hdata <- na.omit(hdata[, -1])
n <- dim(hdata)[1]
set.seed(12)
train <- sample(n, n/2)
htree <- tree(AHD ~ ., hdata, subset = train)
# plot(htree) text(htree,pretty=T)
htree

## node), split, n, deviance, yval, (yprob)
##          * denotes terminal node
##
## 1) root 148 204.500 No ( 0.53378 0.46622 )
##    2) Thal: normal 85  90.330 No ( 0.77647 0.22353 )
##      4) Ca < 0.5 63  39.630 No ( 0.90476 0.09524 )
##        8) MaxHR < 159 25  27.550 No ( 0.76000 0.24000 )
##          16) Chol < 236.5 14  7.205 No ( 0.92857 0.07143 ) *
##          17) Chol > 236.5 11  15.160 No ( 0.54545 0.45455 )
##            34) Age < 57.5 6  5.407 No ( 0.83333 0.16667 ) *
##            35) Age > 57.5 5  5.004 Yes ( 0.20000 0.80000 ) *
##        9) MaxHR > 159 38  0.000 No ( 1.00000 0.00000 ) *
##      5) Ca > 0.5 22  29.770 Yes ( 0.40909 0.59091 )
##    10) ChestPain: nonanginal,nontypical 9  6.279 No ( 0.88889 0.11111 ) *
##    11) ChestPain: asymptomatic,typical 13  7.051 Yes ( 0.07692 0.92308 )
##      22) Age < 56.5 5  5.004 Yes ( 0.20000 0.80000 ) *
```

```
##      23) Age > 56.5 8  0.000 Yes ( 0.00000 1.00000 ) *
##      3) Thal: fixed,reversable 63  64.140 Yes ( 0.20635 0.79365 )
##          6) Oldpeak < 0.9 21  29.060 Yes ( 0.47619 0.52381 )
##          12) RestBP < 122 6  0.000 No ( 1.00000 0.00000 ) *
##          13) RestBP > 122 15  17.400 Yes ( 0.26667 0.73333 )
##              26) Chol < 266.5 9  12.370 Yes ( 0.44444 0.55556 ) *
##              27) Chol > 266.5 6  0.000 Yes ( 0.00000 1.00000 ) *
##          7) Oldpeak > 0.9 42  21.610 Yes ( 0.07143 0.92857 )
##          14) Ca < 0.5 17  15.840 Yes ( 0.17647 0.82353 )
##              28) Age < 56.5 11  0.000 Yes ( 0.00000 1.00000 ) *
##              29) Age > 56.5 6  8.318 No ( 0.50000 0.50000 ) *
##          15) Ca > 0.5 25  0.000 Yes ( 0.00000 1.00000 ) *
```

Figure 7.13: The unpruned classification tree for Heart data



To avoid an overfitting by cross-validation, we prune the tree.

- Let T be a tree and let $|T|$ be the number of terminal nodes of the tree T .
- Let T_1, \dots, T_k be the terminal nodes of the tree T .
- Let $\tau(T_i)$ be the class assigned to T_i . Let $L(i, j)$ be a loss matrix and let $P(T_i) = \sum_{j=1}^C \pi_j P(T_i|j)$ where π_j be the prior probability of class j .
- The risk of T_i is defined by

$$R(T_i) = \sum_{j=1}^C p(j|T_i) L(j, \tau(T_i)) \quad (7.32)$$

where $j = 1, \dots, C$ is the class.

- Define the risk of T to be

$$R(T) = \sum_{i=1}^k P(T_i) R(T_i) \quad (7.33)$$

If $L(i, j) = 1$ for all $i \neq j$, and we set the prior probability π to be the observed class frequencies in the sample then $p(j|T_i) = n_{jT_i}/n_{T_i}$ and $R(T)$ is the proportion misclassified.

- Let α be a number in $[0, \infty)$, called the **complexity parameter**. We define the cost for the tree T as follows:

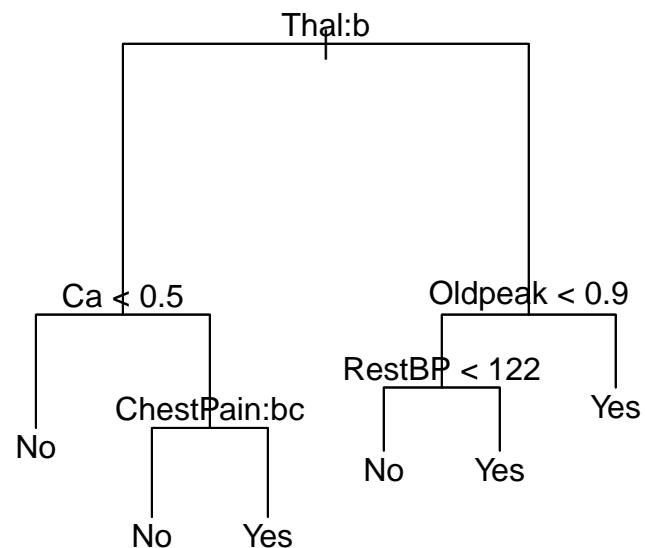
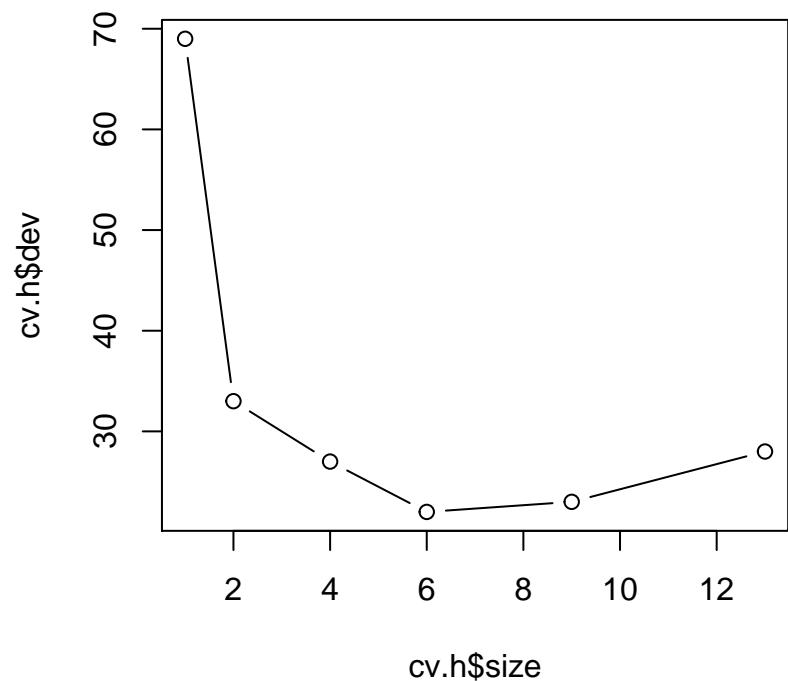
$$R_\alpha(T) = R(T) + \alpha |T| \quad (7.34)$$

Note $R(T)$ decreases as $|T|$ increases.

```
cv.h <- cv.tree(htree, FUN = prune.misclass)
# plot(cv.h$size, cv.h$dev, type='b')
prune.h <- prune.misclass(htree, best = 6)
# plot(prune.h) text(prune.h)
prune.h

## node), split, n, deviance, yval, (yprob)
##      * denotes terminal node
##
## 1) root 148 204.500 No ( 0.53378 0.46622 )
##    2) Thal: normal 85  90.330 No ( 0.77647 0.22353 )
##      4) Ca < 0.5 63  39.630 No ( 0.90476 0.09524 ) *
##      5) Ca > 0.5 22  29.770 Yes ( 0.40909 0.59091 )
##        10) ChestPain: nonanginal,nontypical 9   6.279 No ( 0.88889 0.11111 ) *
##        11) ChestPain: asymptomatic,typical 13   7.051 Yes ( 0.07692 0.92308 ) *
##    3) Thal: fixed,reversible 63  64.140 Yes ( 0.20635 0.79365 )
##      6) Oldpeak < 0.9 21  29.060 Yes ( 0.47619 0.52381 )
##      12) RestBP < 122 6   0.000 No ( 1.00000 0.00000 ) *
##      13) RestBP > 122 15  17.400 Yes ( 0.26667 0.73333 ) *
##      7) Oldpeak > 0.9 42  21.610 Yes ( 0.07143 0.92857 ) *
```

Figure 7.14: (Left) Plot of the deviance versus tree size and (Right) Pruned tree



Package “rpart” can be used and a bit different results would be obtained (default settings are not the same). For more details, visit <https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf>.

```
library(rpart)
n <- dim(hdata)[1]
set.seed(10)
train <- sample(n, n/2)
hfit <- rpart(AHD ~ ., data = hdata, subset = train, method = "class", control = rpart.control(minsplit = 20))
# plot(hfit,xpd = NA) text(hfit,use.n=T,all=TRUE, cex=.7)
library(partykit)

## Loading required package: grid
## Loading required package: libcoin
## Loading required package: mvtnorm
##
## Attaching package: 'mvtnorm'
## The following object is masked _by_ '.GlobalEnv':
##
##     dmvnorm

a <- as.party(hfit)
# plot(a)

cp.opt <- hfit$cptable[which.min(hfit$cptable[, "xerror"]), "CP"]
cp.opt

## [1] 0.046875
```

```
prune.h <- prune(hfit, cp = cp.opt)
# ap<-as.party(prune.h) plot(ap)
```

Figure 7.15: Tree constructed by rpart

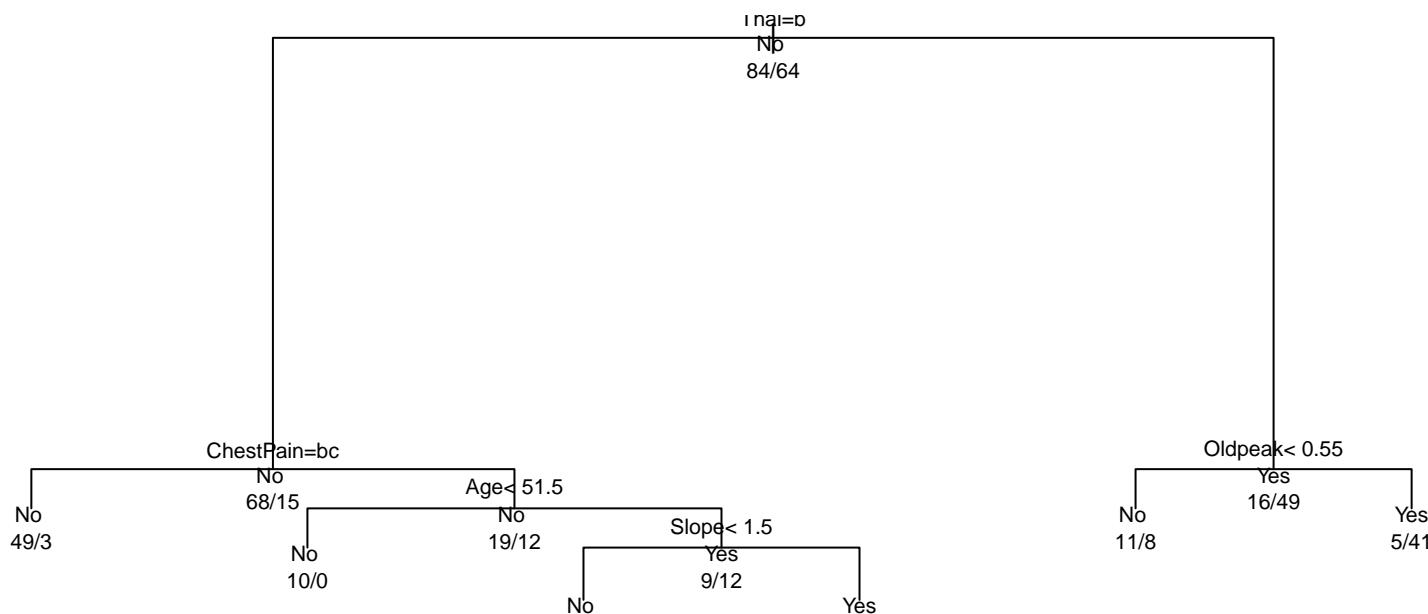
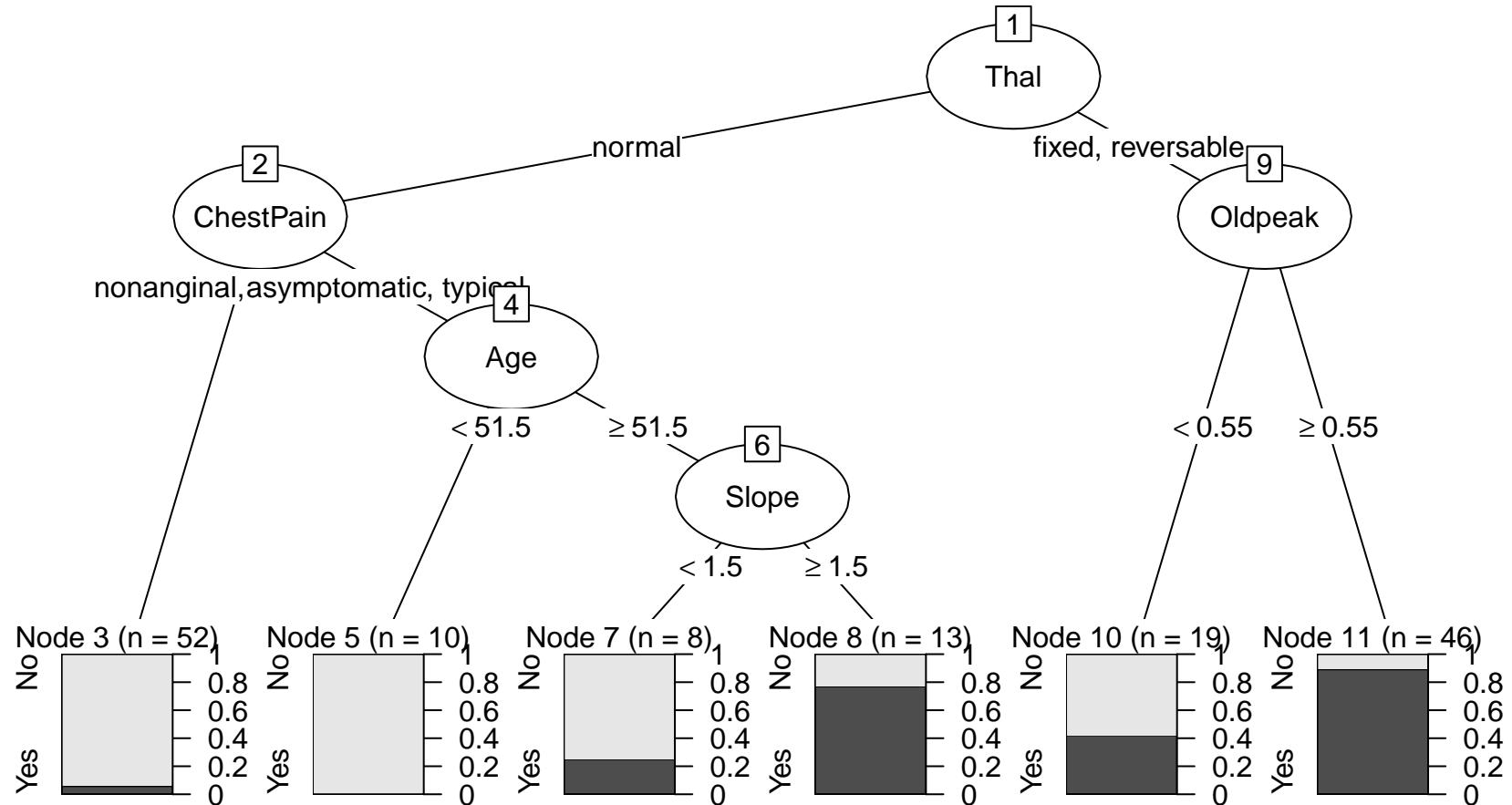


Figure 7.16: Tree constructed by “rpart” with “partykit” plot



Advantages and Disadvantages of Trees

Advantages

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression.
- It is believed that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.

Disadvantages

- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches ⇒ Ensemble methods such as bagging, random forests, and boosting
- Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.

Chapter 8 Ensemble Methods

8.1 Bootstrap aggregation (Bagging)

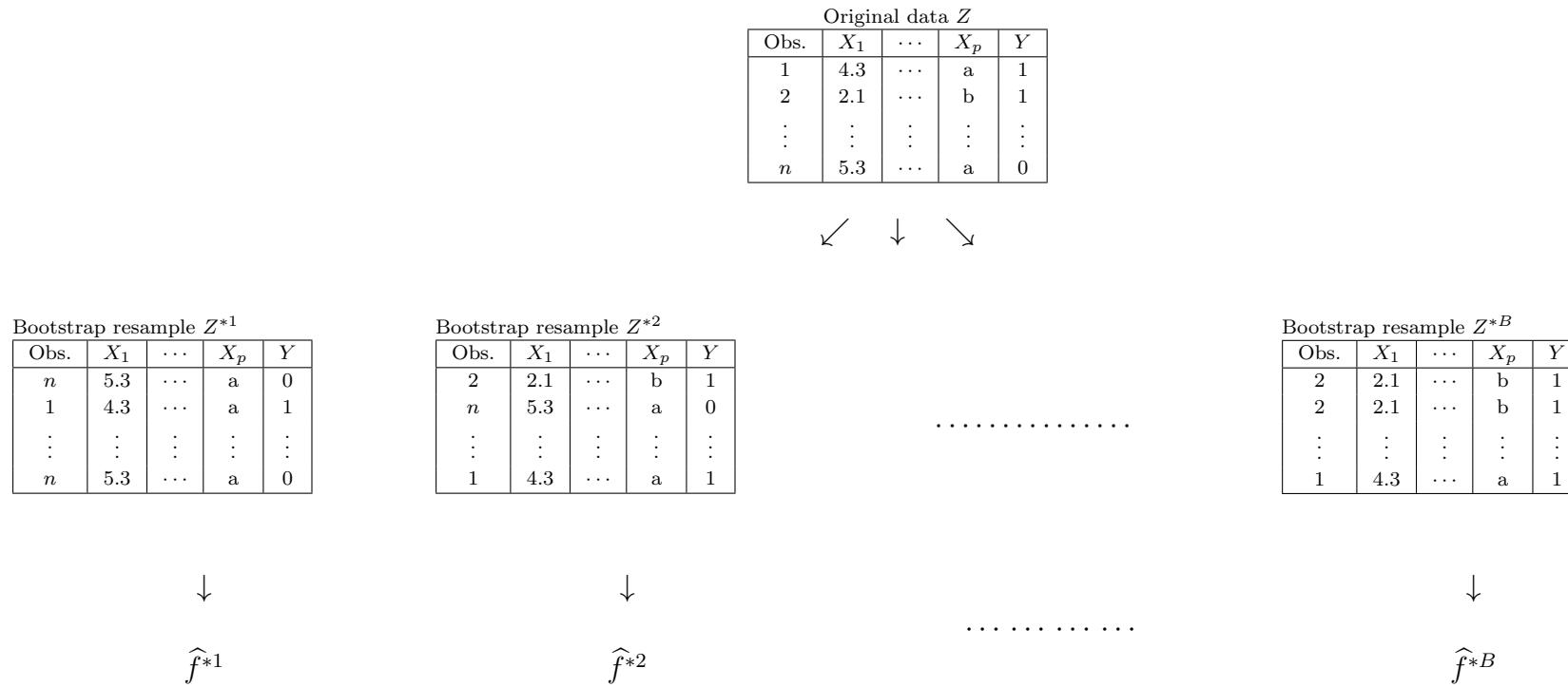
- Tree-based classification or regression methods have high variances and low prediction accuracy.
- Let Z_1, \dots, Z_B have a pairwise correlation coefficient ρ . The average of Z_i has variance $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \Rightarrow$ Average a set of observations reduces variance.

$$\text{Var} \left(\frac{1}{B} \sum_{i=1}^B Z_i \right) = \frac{1}{B^2} \left[\sum_{i=1}^B \text{Var}(Z_i) + \sum_{i=1}^B \sum_{j \neq i} \text{Cov}(Z_i, Z_j) \right] \quad (8.1)$$

$$= \frac{1}{B^2} [B\sigma^2 + B(B-1)\rho\sigma^2] \quad (8.2)$$

$$= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \quad (8.3)$$

- $\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x)$ is called bagging, where we train our method on the b th bootstrapped training set in order to get $\hat{f}^{*b}(x)$.
- These trees are grown deep, and are not pruned.
- For a classification problem, we use a majority vote.
- The number of trees B is not a critical parameter with bagging; using a very large value of B will not lead to overfitting.



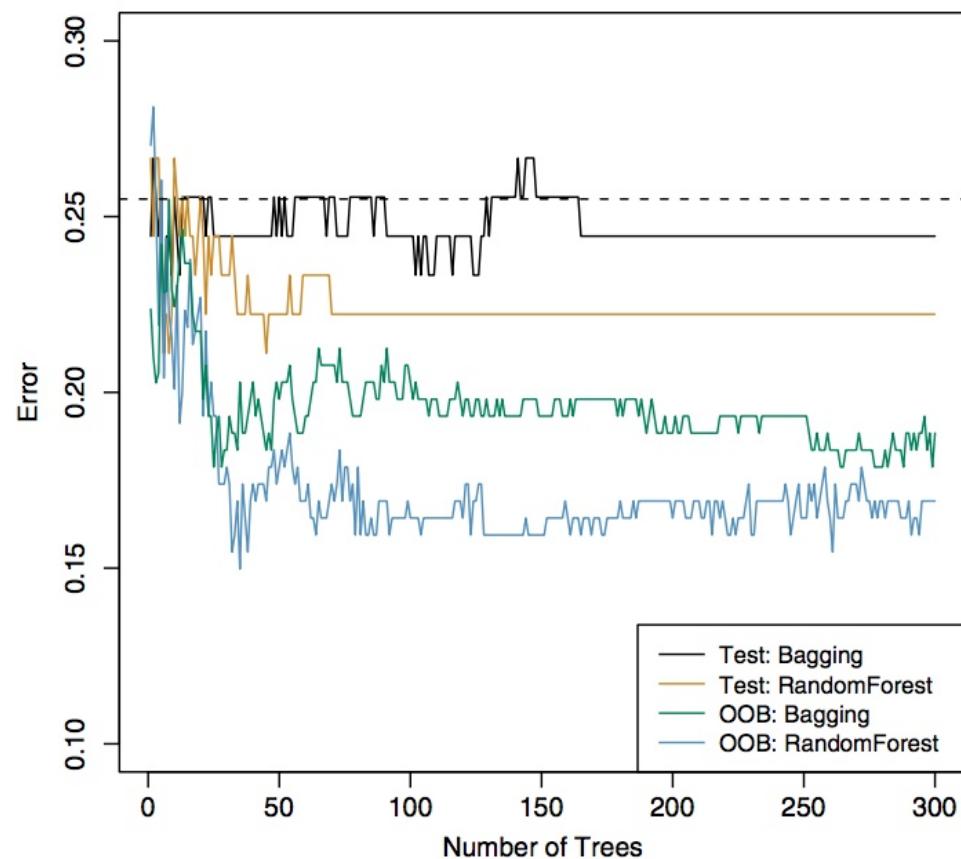
Out-of-Bag (OOB) Error Estimation

- On average, each bagged tree makes use of around two-thirds of the observations.
- Approximately $1/3$ of data are not chosen in a bootstrap resample, referred to as the out-of-bag (OOB) observations.

$$P(\text{an observation is not selected in a bootstrap resample}) = \frac{(n-1)^n}{n^n} = \left(1 - \frac{1}{n}\right)^n \rightarrow \frac{1}{e} \text{ as } n \rightarrow \infty \quad (8.4)$$

- We can predict the response for the i th observation using each of the trees in which that observation was OOB.
- This will yield around $B/3$ predictions for the i th observation.
- In order to obtain a single prediction for the i th observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal).
- It can be shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.

Figure 8.1: Bagging and random forest results for the Heart data. The test error (black and orange) is shown as a function of B , the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.



Example 8.1.1

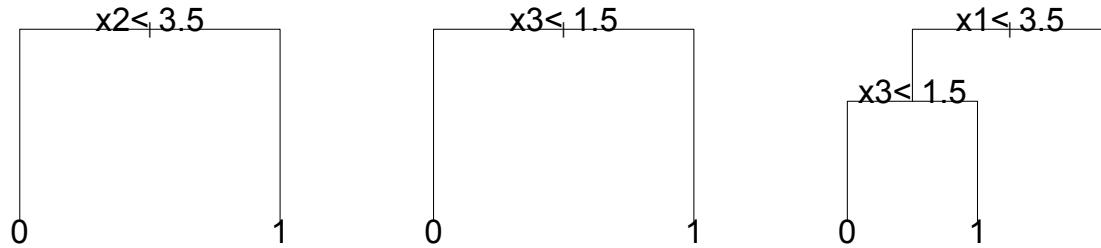
Suppose we have $n = 5$ observations with binary response y and three predictors x_1, x_2 , and x_3 :

i	y	x_1	x_2	x_3
1	0	1	2	1
2	0	3	1	1
3	1	4	3	1
4	1	2	5	2
5	1	5	4	2

We constructed a random forest classifier with $mtry = 2$ randomly selected variables at each node for a split. The number of trees was set to be $ntree = 3$. The bootstrap data sets and corresponding trees are as follow:

1st bootstrap data					2nd bootstrap data					3rd bootstrap data				
i	y	x_1	x_2	x_3	i	y	x_1	x_2	x_3	i	y	x_1	x_2	x_3
2	0	3	1	1	2	0	3	1	1	2	0	3	1	1
2	0	3	1	1	4	1	2	5	2	1	0	1	2	1
4	1	2	5	2	4	1	2	5	2	4	1	2	5	2
1	0	1	2	1	2	0	3	1	1	3	1	4	3	1
2	0	3	1	1	4	1	2	5	2	2	0	3	1	1

Tree 1 ($tree_1$)Tree 2 ($tree_2$)Tree 3 ($tree_3$)



- a. What is the predicted class for $x_1 = 4, x_2 = 2$, and $x_3 = 1$?

Given $\mathbf{x} = (4, 2, 1)'$,

$\hat{Y}^{tree1}(4, 2, 1) = 0$ since $x_2 = 2 < 3.5$.

$\hat{Y}^{tree2}(4, 2, 1) = 0$ since $x_3 = 1 < 1.5$.

$\hat{Y}^{tree3}(4, 2, 1) = 1$ since $x_1 = 4 \geq 3.5$.

$\therefore \hat{Y}(4, 2, 1) = \text{majority vote of } \{0, 0, 1\} = 0$.

□

b. Calculate the OOB error rate.

We can make a contingency table for bootstrap samples:

Subject	Tree 1	Tree 2	Tree 3
1	1	0	1
2	3	2	2
3	0	0	1
4	1	3	1
5	0	0	0

The OOB error rate is calculated on the subjects 1, 3, and 5. We calculate the predicted class for subject 1 using Tree 2 as $\hat{Y}_1^{tree2}(1, 2, 1) = 0$ since $x_2 = 2 < 3.5$ for subject 1. We obtain the following table based on every predicted class for OOB samples.

Subject	\hat{Y}^{tree1}	\hat{Y}^{tree2}	\hat{Y}^{tree3}	majority vote	y	Prediction
1	X	$\hat{Y}^{tree2}(1, 2, 1) = 0$	X	0	0	correct
3	$\hat{Y}^{tree1}(4, 3, 1) = 0$	$\hat{Y}^{tree2}(4, 3, 1) = 0$	X	0	1	incorrect
5	$\hat{Y}^{tree1}(5, 4, 2) = 1$	$\hat{Y}^{tree1}(5, 4, 2) = 1$	$\hat{Y}^{tree1}(5, 4, 2) = 1$	1	1	correct

Therefore, the OOB error estimate is equal to $\frac{1}{3}$ (1 incorrect prediction in total 3 predicted classes).

□

Variable Importance Measures

- When we bag a large number of trees, it is no longer possible to represent the resulting statistical learning procedure using a single tree, and it is no longer clear which variables are most important to the procedure. Thus, bagging improves prediction accuracy at the expense of interpretability.
- In the case of bagging regression trees, we can record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor. Similarly, in the context of bagging classification trees, we can add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees. At each node t and split s , let $p_L = \frac{N_{tL}}{N_t}$ and $p_R = \frac{N_{tR}}{N_t}$ where N_t is the number of subjects at node t before the split s , N_{tL} and N_{tR} are the number of subjects after the split.

$$\Delta i(s, t) = i(t) - p_L i(t_L) - p_R i(t_R) \quad (8.5)$$

where $i(t)$ is an impurity measure such as classification error, Gini index or cross-entropy for classification, MSE for regression. The importance measure of variable X_m is defined as

$$Imp(X_m) = \frac{1}{N_T} \sum_T \sum_{t \in T, v(s_t)=X_m} p(t) \Delta i(s_t, t) \quad (8.6)$$

where $p(t)$ is the proportion N_t/N of samples reaching the node t and $v(s_t)$ is the variable used in split s_t . When using the Gini index as impurity function, this measure is known as the Gini importance or Mean Decrease Gini.

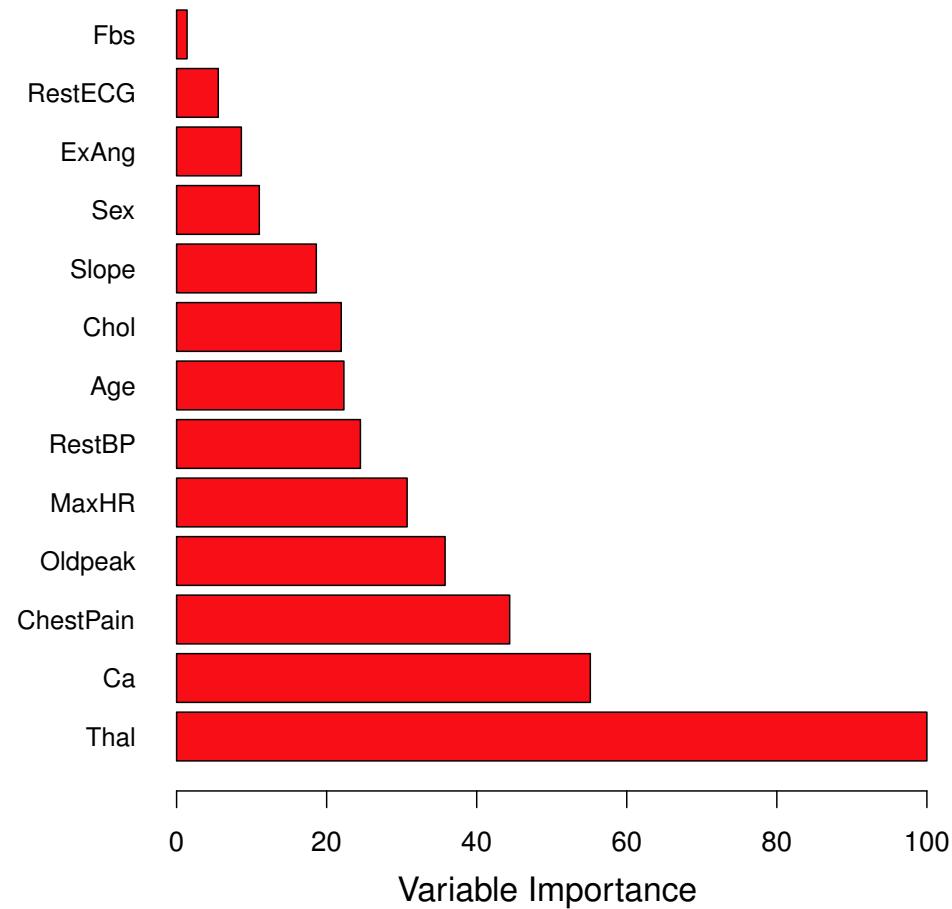
- (Permutation importance in randomForest of R) Here are the definitions of the variable importance measures. The first measure is computed from permuting OOB data: For each tree, the prediction error on the out-of-bag portion of the data is recorded (error rate for classification, MSE for regression). Then the same is done after permuting each predictor variable. The difference between the two are then averaged over all trees, and normalized by the standard deviation of the differences. If the standard deviation of

the differences is equal to 0 for a variable, the division is not done (but the average is almost always equal to 0 in that case).

$$VI_T(X_m) = \frac{1}{n_T^{OOB}} \left(\sum_i I(y_i = \hat{y}_{i,T}) - \sum_i I(y_i = \hat{y}_{i,\pi_m,T}) \right) \quad (8.7)$$

$$VI(X_m) = \frac{1}{N_T} \sum_T VI_T(X_m) \quad (8.8)$$

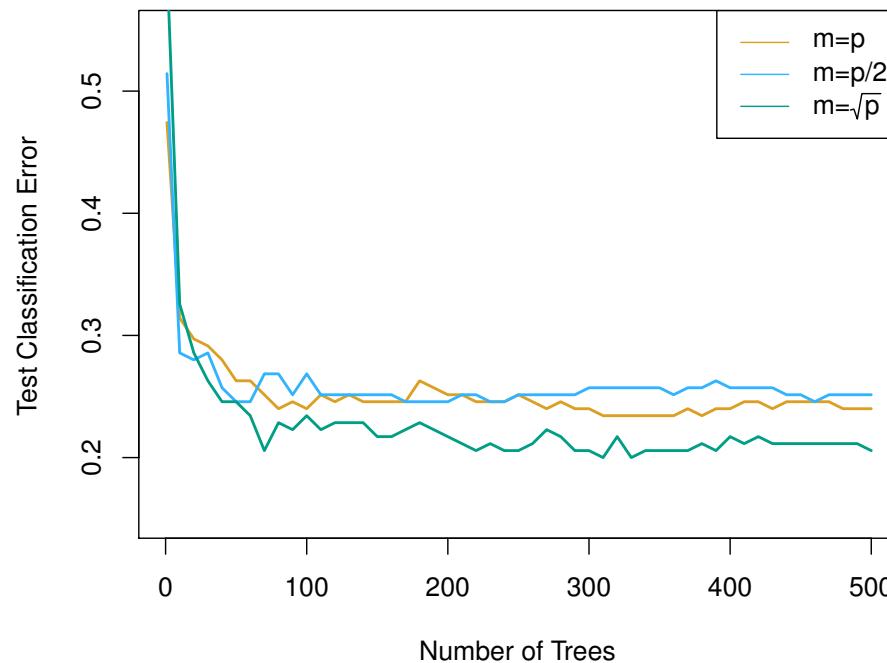
Figure 8.2: A variable importance plot for the Heart data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.



8.2 Random Forest

- We decorrelates the trees by randomly selecting m predictors from total p predictors.
- The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$, that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors. Note that a common choice of $m = p/3$ for regression.
- In bagging, most or all of the trees will use this strong predictor in the top split.
 - ⇒ Consequently, all of the bagged trees will look quite similar to each other.
 - ⇒ Hence the predictions from the bagged trees will be highly correlated.
 - ⇒ Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.
- Random forests overcome this problem by forcing each split to consider only a subset of the predictors.
 - ⇒ On average $(p - m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance.
 - ⇐ Decorrelating the trees.

Figure 8.3: Results from random forests for the 15–class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m , the number of predictors available for splitting at each interior tree node. Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7 %.



Example 8.2.1

Ensemble methods (BAGGING and RANDOM FOREST)

```
library(randomForest)

## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin

set.seed(10)
train <- sample(n, n/2)
set.seed(1)
bag.h <- randomForest(AHD ~ ., data = hdata, subset = train, mtry = 13, importance = T)
bag.h

##
## Call:
##   randomForest(formula = AHD ~ ., data = hdata, mtry = 13, importance = T,      subset = train)
##   Type of random forest: classification
##   Number of trees: 500
##   No. of variables tried at each split: 13
##
##   OOB estimate of  error rate: 22.3%
```

```
## Confusion matrix:  
##      No Yes class.error  
## No  70  14   0.1666667  
## Yes 19  45   0.2968750  
  
yhat.bag <- predict(bag.h, newdata = hdata[-train, ])  
mean(yhat.bag != hdata$AHD[-train])  
  
## [1] 0.2080537  
  
importance(bag.h)  
  
##          No       Yes MeanDecreaseAccuracy MeanDecreaseGini  
## Age     3.5623015  4.26538120    6.0034209    6.6012792  
## Sex     9.5518116 -1.01481624    8.4044236    1.3820607  
## ChestPain 7.2864972  6.93355427    9.4152301    6.4778333  
## RestBP   -0.1193441 -0.11947240   -0.1225966    4.5664123  
## Chol     2.3319253  0.43029221    1.9908844    6.9729292  
## Fbs      0.7085569  0.09147788    0.7255728    0.3917387  
## RestECG  -2.0022006  1.34968630   -0.5538972    0.6381225  
## MaxHR    4.5311727 -0.05459974    3.7132013    5.6824563  
## ExAng    1.8945322  2.98444843    3.4855867    0.9661676  
## Oldpeak  6.9993271  8.88428238   11.1719665    8.7533052  
## Slope    1.2309206  8.69857813    7.4009933    3.0695247  
## Ca       15.3868246 10.49500804   17.3484183    8.1723228  
## Thal    22.3728051 13.75095852   24.8729049    18.4578476
```

```
# varImpPlot(bag.h)

set.seed(1)
rf.h <- randomForest(AHD ~ ., data = hdata, subset = train, mtry = 4, importance = T)
rf.h

##
## Call:
##   randomForest(formula = AHD ~ ., data = hdata, mtry = 4, importance = T,      subset = train)
##   Type of random forest: classification
##   Number of trees: 500
##   No. of variables tried at each split: 4
##
##   OOB estimate of  error rate: 18.24%
## Confusion matrix:
##   No Yes class.error
## No  73 11  0.1309524
## Yes 16 48  0.2500000

yhat.rf <- predict(rf.h, newdata = hdata[-train, ])
mean(yhat.rf != hdata$AHD[-train])

## [1] 0.1946309

importance(rf.h)

##          No          Yes MeanDecreaseAccuracy MeanDecreaseGini
```



```
## Call:  
## randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE, subset = train)  
##           Type of random forest: regression  
##           Number of trees: 500  
## No. of variables tried at each split: 13  
##  
##           Mean of squared residuals: 11.15723  
##           % Var explained: 86.49  
  
yhat.bag = predict(bag.boston, newdata = Boston[-train, ])  
# plot(yhat.bag, boston.test) abline(0,1)  
mean((yhat.bag - boston.test)^2)  
  
## [1] 13.50808  
  
bag.boston = randomForest(medv ~ ., data = Boston, subset = train, mtry = 13,  
    ntree = 25)  
yhat.bag = predict(bag.boston, newdata = Boston[-train, ])  
mean((yhat.bag - boston.test)^2)  
  
## [1] 13.94835  
  
set.seed(1)  
rf.boston = randomForest(medv ~ ., data = Boston, subset = train, mtry = 6,  
    importance = TRUE)  
yhat.rf = predict(rf.boston, newdata = Boston[-train, ])  
mean((yhat.rf - boston.test)^2)
```

```
## [1] 11.66454

importance(rf.boston)

##          %IncMSE IncNodePurity
## crim      12.132320     986.50338
## zn        1.955579      57.96945
## indus     9.069302     882.78261
## chas      2.210835      45.22941
## nox       11.104823    1044.33776
## rm        31.784033    6359.31971
## age       10.962684     516.82969
## dis       15.015236    1224.11605
## rad       4.118011      95.94586
## tax       8.587932      502.96719
## ptratio   12.503896     830.77523
## black     6.702609      341.30361
## lstat     30.695224    7505.73936

# varImpPlot(rf.boston)
```

8.3 Boosting

Brief history:

- Invent Adaboost, the first successful boosting algorithm

Freund, Y., Schapire, R. E., et al. (1996). Experiments with a new boosting algorithm. In ICML, volume 96, pages 148–156.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1):119–139.

- Formulate Adaboost as gradient descent with a special loss

Breiman, L. et al. (1998). Arcing classifier (with discussion and a rejoinder by the author). The annals of statistics, 26(3):801–849.

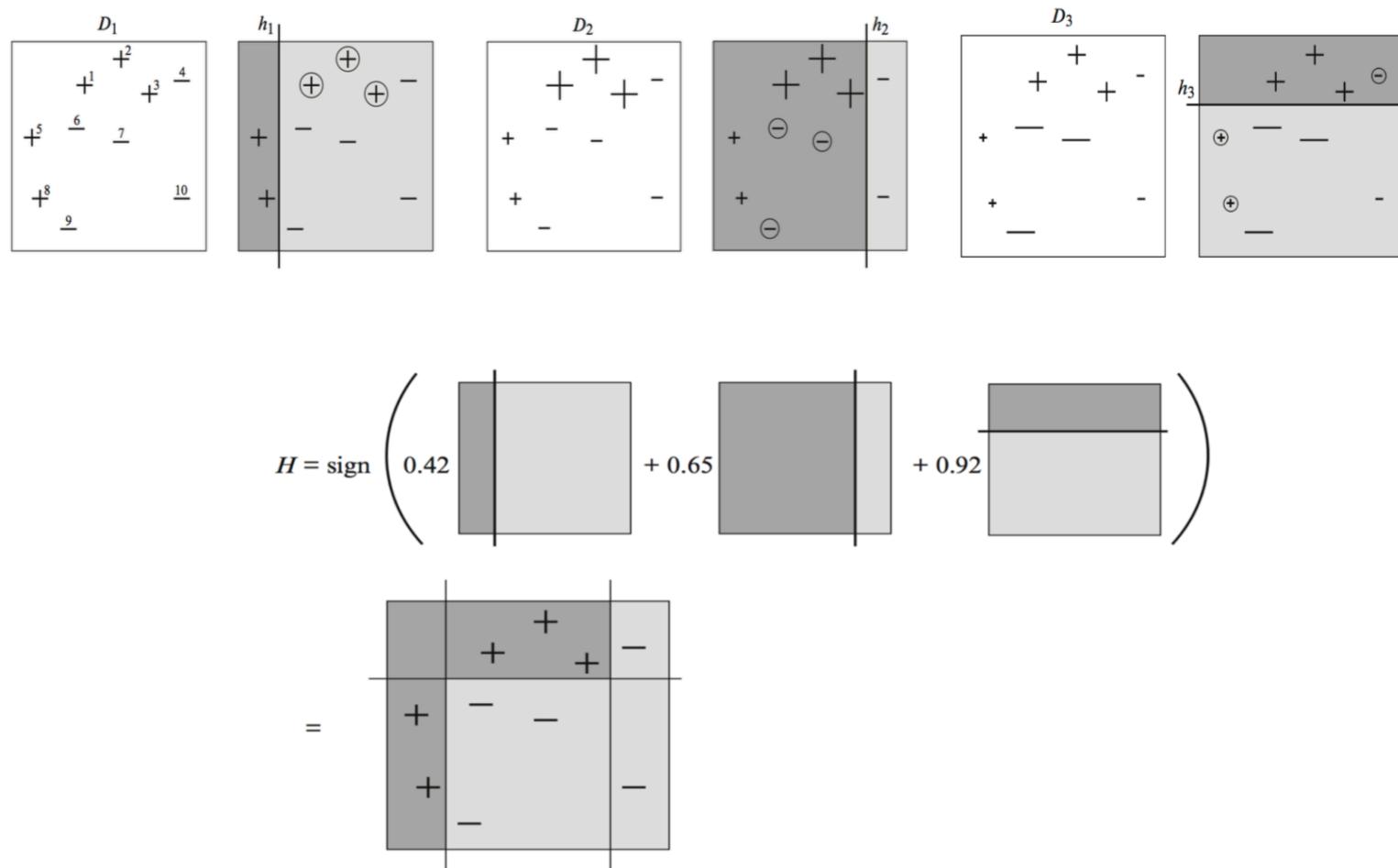
Breiman, L. (1999). Prediction games and arcing algorithms. Neural computation, 11(7):1493–1517.

- Generalize Adaboost to Gradient Boosting in order to handle a variety of loss functions

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Special invited paper. additive logistic regression: A statistical view of boosting. Annals of statistics, pages 337–374.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of Statistics, pages 1189–1232.

Figure 8.4: Concept of Adaboost: From Schapire, R. E. and Freund, Y. (2012). Boosting: Foundations and Algorithms. MIT Press.



AdaBoost Let $\{(x_1, y_1), \dots, (x_N, y_N)\}$ be a dataset where $y_i \in \{-1, 1\}$. The total error of a classifier C_m is measured by an exponential loss: $E = \sum_{i=1}^N e^{-y_i C_m(x_i)}$. If we set a weight $w_i^{(m)} = e^{-y_i C_{m-1}}$, then the total error of $C_m = C_{m-1} + \alpha_m G_m$ is $E = \sum_{i=1}^N w_i^{(m)} e^{-y_i \alpha_m G_m(x_i)}$.

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

Algorithm:

- The trees are grown sequentially: each tree is grown using information from previously grown trees.
 - Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set
 2. For $b = 1, \dots, B$, repeat:
 - (a) Fit a tree \hat{f}^b with d splits ($d + 1$ terminal nodes) to the training data (X, r)
 - (b) Update \hat{f} by adding in a shrunken version of the new tree: $\hat{f} \leftarrow \hat{f} + \lambda \hat{f}^b$
 - (c) Update the residual: $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$
 3. Output the boosted model $\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x)$
- The number of trees B . Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B .
 - The shrinkage parameter λ , a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
 - The number d of splits in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split. In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Gradient boosting

- Gradient descent + boosting
- In the gradient descent method to minimize $L(\theta)$, We update $\theta^{(r)}$ by the following iteration steps

$$\theta^{(r+1)} = \theta^{(r)} - \rho \frac{\partial}{\partial \theta} \bigg|_{\theta^{(r)}} L(\theta)$$

The update is based on the gradient $\nabla L(\theta^{(r)})$.

- Suppose $\{(x_1, y_1), \dots, (x_N, y_N)\}$ is a dataset and F is a prediction model.
- We can improve the model F by constructing a new model to the residual $y_i - F(x_i)$, that is, find G that fits $\{(x_1, y_1 - F(x_1)), \dots, (x_N, y_N - F(x_N))\}$. Therefore, the additive model $F + G$ may improve the previous model F .
- Here, the update G is based on the residuals $y_i - F(x_i) = -\frac{\partial}{\partial F(x_i)} \frac{1}{2}(y_i - F(x_i))^2$, that is the negative gradient of the squared loss function $L(y_i, F(x_i)) = \frac{1}{2}(y_i - F(x_i))^2$.
- The gradient boosting model is an additive model. At each stage, we fit the negative gradient of a loss function and add the fitted model to the existing model.

Let L be a loss function such as $L(a, b) = (a - b)^2$ that is a square loss function.

Friedman's gradient boost algorithm:

- Let \hat{f}_1 be a constant such that $\hat{f}_1(\mathbf{x}) = \operatorname{argmin}_{\mu} \sum_{i=1}^N L(y_i, \mu)$.
- Repeat

1. Compute the negative gradient as the working response

$$z_i^{(m-1)} = -\frac{\partial}{\partial f(\mathbf{x}_i)} L(y_i, f(\mathbf{x}_i))|_{f(\mathbf{x}_i)=\hat{f}_{m-1}(\mathbf{x}_i)} \quad (8.9)$$

2. Fit a regression model $\hat{g}_m(\mathbf{x})$ to predict $z_i^{(m-1)}$ from the predictors \mathbf{x}_i .
3. Choose a gradient descent step size as

$$\hat{\rho}_m = \operatorname{argmin}_{\rho} \sum_{i=1}^N L\left(y_i, \hat{f}_{m-1}(\mathbf{x}_i) + \rho \hat{g}_m(\mathbf{x}_i)\right) \quad (8.10)$$

4. Update the estimate of $f(\mathbf{x})_m$ as

$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \hat{\rho}_m \hat{g}_m(\mathbf{x}) \quad (8.11)$$

Boosting of `gbm()` function in R package:

Select

- a loss function L ; its default distribution is “bernoulli”.
 - the number of iterations, T (`n.trees`); its default value is 100.
 - the depth of each tree, K (`interaction.depth`); its default value is 1.
 - the shrinkage (or learning rate) parameter, λ (`shrinkage`); its default value is 0.001.
 - the subsampling rate, p (`bag.fraction`); its default value is 0.5.
 - `cv.folds` (> 0) will perform a cross validation to find the best iteration number of trees; its default value is 0.
1. Compute the negative gradient as the working response

$$z_i = -\frac{\partial}{\partial f(\mathbf{x}_i)} L(y_i, f(\mathbf{x}_i))|_{f(\mathbf{x}_i)=\hat{f}(\mathbf{x}_i)} \quad (8.12)$$

2. Randomly select $p \times N$ cases from the dataset
3. Fit a regression tree with K terminal nodes, $\hat{g}(\mathbf{x}) = E(z|\mathbf{x})$ using only the randomly selected observations.
4. Compute the optimal terminal node predictions, ρ_1, \dots, ρ_K , as

$$\rho_k = \operatorname{argmin}_{\rho} \sum_{\mathbf{x}_i \in S_k} L\left(y_i, \hat{f}(\mathbf{x}_i) + \rho\right) \quad (8.13)$$

where S_k is the set of \mathbf{x} s that define terminal node k .

5. Update $\hat{f}(\mathbf{x})$ as

$$\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \hat{\lambda} \rho_k(\mathbf{x}) \quad (8.14)$$

where $k(\mathbf{x})$ indicates the index of the terminal node into which an observation with features \mathbf{x} would fall. Again this step uses only the randomly selected observations

Example 8.3.1

Boosting

```
library(gbm)

## Loaded gbm 2.1.5

n <- dim(iris)[1]
set.seed(1)
train <- sample(n, n/2)
fit <- gbm(Species ~ ., data = iris[train, ], cv.folds = 10, n.tree = 100, distribution = "multinomial",
           n.cores = 20)
best.iter <- gbm.perf(fit, method = "cv")
pred <- predict(fit, iris[-train, ], best.iter, type = "response")

pcl <- apply(pred, 1, which.max)
table(pcl, iris$Species[-train])

##
## pcl setosa versicolor virginica
##   1      24          0          0
##   2      0          21          2
##   3      0          1          27

library(ISLR)
data(Default)
str(Default)
```

```
## 'data.frame': 10000 obs. of  4 variables:  
##   $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...  
##   $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 1 ...  
##   $ balance: num  730 817 1074 529 786 ...  
##   $ income : num  44362 12106 31767 35704 38463 ...  
  
n <- dim(Default)[1]  
set.seed(1)  
train <- sample(1:n, n/2)  
  
Default$y <- rep(0, dim(Default)[1])  
Default$y[Default$default == "Yes"] <- 1  
dft.train <- Default[train, -1]  
dft.test <- Default[-train, -1]  
library(gbm)  
  
fit <- gbm(y ~ ., data = dft.train, distribution = "bernoulli", shrinkage = 0.01,  
            n.tree = 1000, interaction.depth = 4)  
pred <- predict(fit, dft.test, n.tree = 500)  
library(caret)  
pred1 <- predict(fit, dft.test, n.tree = 500, type = "response")  
predClass <- rep(0, length(pred))  
cutoff <- 0.5  
predClass[pred1 >= cutoff] <- 1  
confusionMatrix(factor(predClass), factor(dft.test$y), positive = "1")  
  
## Confusion Matrix and Statistics
```

```
##  
##          Reference  
## Prediction    0    1  
##             0 4793 109  
##             1   40   58  
##  
##          Accuracy : 0.9702  
##                 95% CI : (0.9651, 0.9747)  
##  No Information Rate : 0.9666  
##  P-Value [Acc > NIR] : 0.08238  
##  
##          Kappa : 0.4235  
##  Mcnemar's Test P-Value : 2.536e-08  
##  
##          Sensitivity : 0.3473  
##          Specificity : 0.9917  
##  Pos Pred Value : 0.5918  
##  Neg Pred Value : 0.9778  
##          Prevalence : 0.0334  
##          Detection Rate : 0.0116  
##  Detection Prevalence : 0.0196  
##          Balanced Accuracy : 0.6695  
##  
##          'Positive' Class : 1  
##
```

Chapter 9 Summarization of a classifier

- Confusion matrix
- ROC analysis
- Unbalanced data
 - Undersampling
 - Oversampling
 - Ensemble methods using undersampling or oversampling

9.1 Confusion matrix

Table 9.1: Confusion Matrix: Possible results from a binary classifier

		True class		Total
		-	+	
Predicted class	-	TN=a	FN=b	N*
	+	FP=c	TP=d	P*
		Total	N	P
				n

$$\text{accuracy} = \frac{a + d}{a + b + c + d}$$

$$\text{sensitivity} = \frac{d}{b + d}$$

$$\text{specificity} = \frac{a}{a + c}$$

$$\text{prevalence} = \frac{b + d}{a + b + c + d}$$

$$PPV = \frac{(sensitivity \times prevalence)}{(sensitivity \times prevalence) + ((1 - specificity) \times (1 - prevalence))} = \frac{d}{c + d}$$

$$NPV = \frac{(specificity \times (1 - prevalence))}{((1 - sensitivity) \times prevalence) + ((specificity) \times (1 - prevalence))} = \frac{a}{a + b}$$

$$\text{DetectionRate} = \frac{d}{a + b + c + d}$$

$$\text{DetectionPrevalence} = \frac{c + d}{a + b + c + d}$$

$$\text{BalancedAccuracy} = \frac{\text{sensitivity} + \text{specificity}}{2}$$

$$\text{Precision} = \frac{d}{c + d}$$

$$\text{Recall} = \frac{d}{b + d}$$

$$F_{\beta} = \frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}}$$

$\text{NoInformationRate}(NIR)$ = the largest class percentage in the data

Example 9.1.1

(Default data)

```
library(ISLR)
data(Default)
str(Default)

## 'data.frame': 10000 obs. of  4 variables:
##   $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 ...
##   $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 ...
##   $ balance: num  730 817 1074 529 786 ...
##   $ income : num  44362 12106 31767 35704 38463 ...

n <- dim(Default)[1]
set.seed(1)
train <- sample(1:n, n/2)
fit <- glm(default ~ ., data = Default, subset = train, family = binomial)
```

```
pred <- predict(fit, Default[-train, ], type = "response")
predClass <- rep("No", length(pred))
predClass[pred >= 0.5] <- "Yes"

library(caret)
confusionMatrix(factor(predClass), Default$default[-train], positive = "Yes")

## Confusion Matrix and Statistics
##
##          Reference
## Prediction   No   Yes
##       No    4803   114
##       Yes     30    53
##
##          Accuracy : 0.9712
##                  95% CI : (0.9662, 0.9757)
##      No Information Rate : 0.9666
##      P-Value [Acc > NIR] : 0.036
##
##          Kappa : 0.4109
##  Mcnemar's Test P-Value : 4.624e-12
##
##          Sensitivity : 0.3174
##          Specificity : 0.9938
##      Pos Pred Value : 0.6386
##      Neg Pred Value : 0.9768
```

```
##          Prevalence : 0.0334
##          Detection Rate : 0.0106
##  Detection Prevalence : 0.0166
##          Balanced Accuracy : 0.6556
##
##          'Positive' Class : Yes
##
```

9.1.1 Cohen's kappa κ

- Cohen's kappa coefficient is a statistic which measures inter-rater agreement for qualitative (categorical) items.
- It is generally thought to be a more robust measure than simple percent agreement calculation, since κ takes into account the possibility of the agreement occurring by chance.
- There is controversy surrounding Cohen's Kappa due to the difficulty in interpreting indices of agreement.
-

$$\kappa = \frac{p_O - p_E}{1 - p_E} \quad (9.1)$$

where $p_O = \frac{a+d}{a+b+c+d}$ and $p_E = \frac{a+b}{a+b+c+d} \frac{a+c}{a+b+c+d} + \frac{b+d}{a+b+c+d} \frac{c+d}{a+b+c+d}$.

- Landis and Koch characterized values < 0 as indicating no agreement and 0–0.20 as slight, 0.21–0.40 as fair, 0.41–0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1 as almost perfect agreement
- Fleiss's equally arbitrary guidelines characterize kappas over 0.75 as excellent, 0.40 to 0.75 as fair to good, and below 0.40 as poor.

9.1.2 F_1 score

- In statistical analysis of binary classification, the F_1 score (also F -score or F -measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results, and r is the number of correct positive results divided by the number of positive results that should have been returned.
- The F_1 score is the harmonic average of the precision and recall, where an F_1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.
- In general, F_β score is defined as

$$F_\beta = \frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}}$$

9.2 Receiver Operating Characteristic (ROC) curve analysis

- The ROC curve was first developed by electrical engineers and radar engineers during World War II for detecting enemy objects in battlefields and was soon introduced to psychology to account for perceptual detection of stimuli.
- ROC analysis since then has been used in medicine, radiology, biometrics, and other areas for many decades and is increasingly used in machine learning and data mining research.
- It is a way of showing the performance of Binary Classifiers
- It is created by plotting the fraction of True Positives (TP) vs the fraction of False Positives (FP)

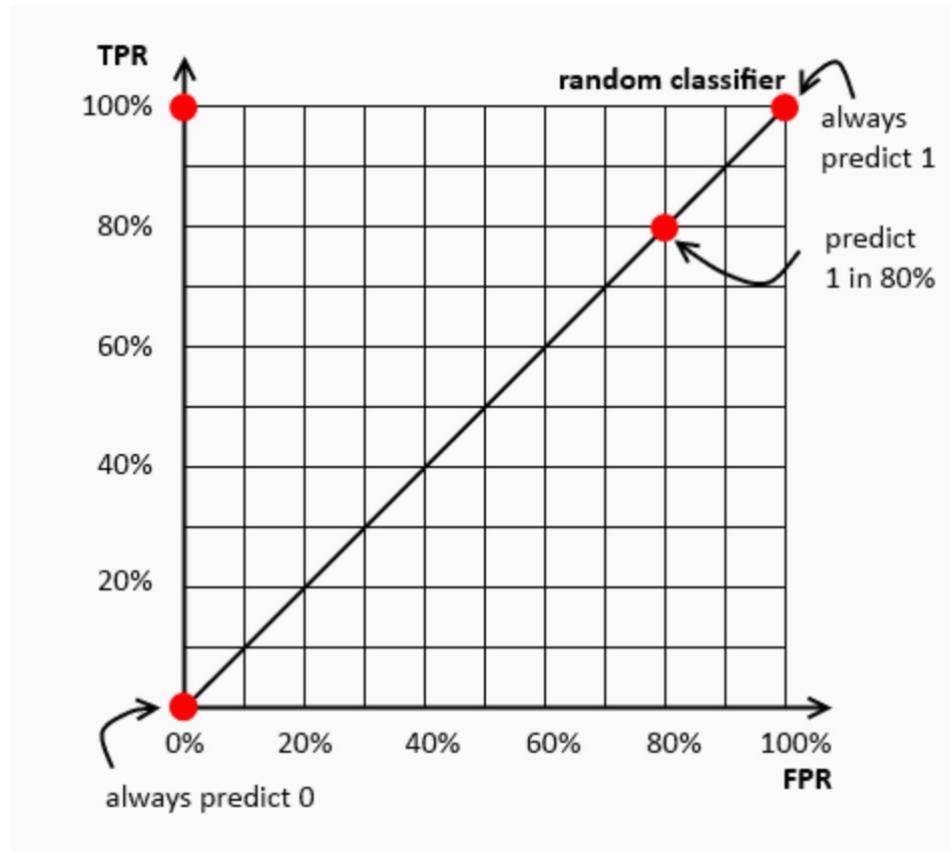
$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Table 9.2: Confusion Matrix: Possible results from a binary classifier

		Predicted class		Total
		-	+	
True class	-	TN	FP	N
	+	FN	TP	P
Total		N*	P*	n

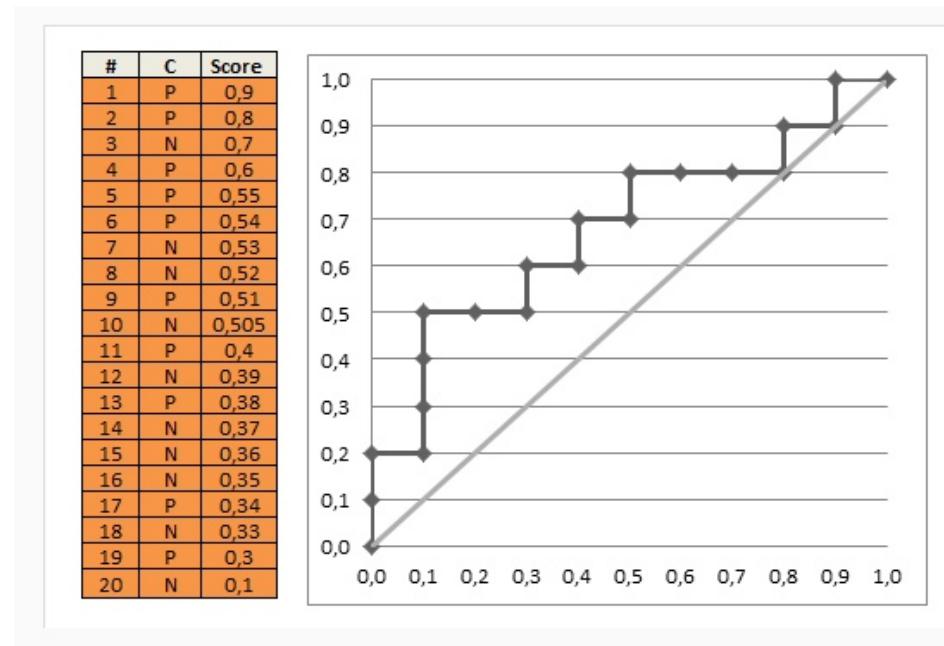
Figure 9.1: ROC space



- (How to draw an ROC curve:)

Suppose that $\hat{p}(x_1) = \hat{P}(Y = 1|x_1) = .9, \hat{p}(x_2) = .8, \hat{p}(x_3) = .7, \hat{p}(x_4) = .6, \hat{p}(x_5) = .55, \hat{p}(x_6) = .54, \hat{p}(x_7) = .53, \dots, \hat{p}(x_{19}) = .3, \hat{p}(x_{20}) = .1$ for a test dataset $(x_1, y_1), \dots, (x_{20}, y_{20})$.

Figure 9.2: ROC curve



- (How to find an optimal cut-off (threshold) value:)

Let n be the total number of observations. We define accuracy acc by

$$acc = \frac{TP + TN}{n} \quad (9.2)$$

$$= \frac{TP}{P} \cdot \frac{P}{n} + \frac{TN}{N} \frac{N}{n} \quad (9.3)$$

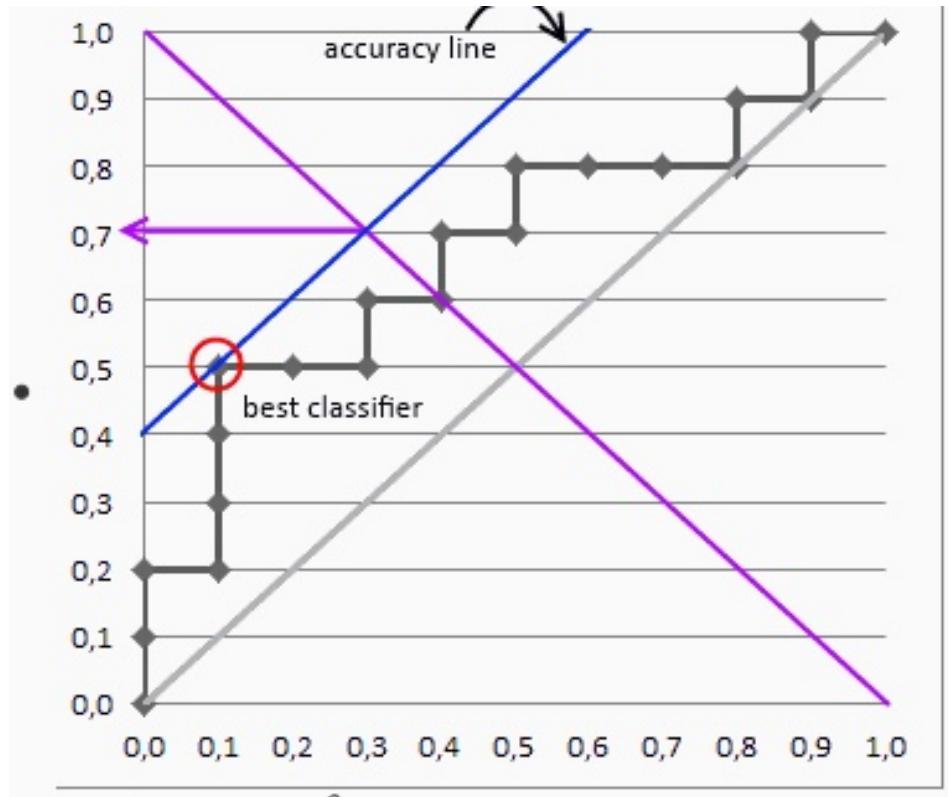
$$= TPR \cdot \frac{P}{n} + (1 - FPR) \frac{N}{n} \quad (9.4)$$

$$= \frac{P}{n}y + \frac{N}{n}(1 - x) \quad (9.5)$$

$$y = \frac{N/n}{P/n}x + \frac{acc - N/n}{P/n} \quad (9.6)$$

where $y = TPR$ and $x = FPR$.

Figure 9.3: ROC best cut-off



Note the intersection of the accuracy line with $x + y = 1$ reduces to $y = acc$.

$$y = \frac{N/n}{P/n}(1 - y) + \frac{acc - N/n}{P/n} \quad (9.7)$$

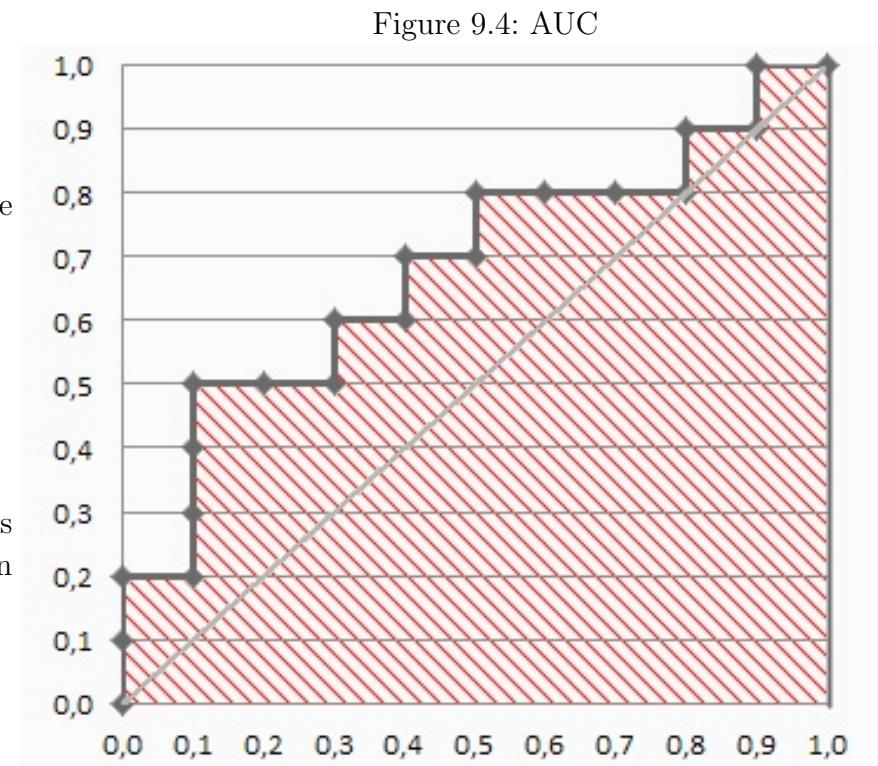
$$\left(1 + \frac{N}{P}\right)y = \frac{N}{P} + acc\frac{n}{P} - \frac{N}{P} \quad (9.8)$$

$$\frac{n}{P}y = acc\frac{n}{P} \quad (9.9)$$

$$y = acc \quad (9.10)$$

- (Area Under ROC Curve:)

- Measure for evaluating the performance of a classifier
- $AUC = 1$ is for a perfect classifier for which all positive come after all negatives
- $AUC = 0.5$: randomly ordered
- $AUC = 0$: all negative come before all positive
- typically we don't have classifiers with $AUC < 0.5$
- Formally, the AUC of a classifier C is probability that C ranks a randomly drawn “+” example higher than a randomly drawn “-” example.



```
require('ROCR')

## Loading required package: ROCR
## Loading required package: gplots
##
## Attaching package: 'gplots'
## The following object is masked from 'package:stats':
##
##     lowess

cls = c('P', 'P', 'N', 'P', 'P', 'P', 'N', 'N', 'P', 'N',
       'N', 'P', 'N', 'N', 'N', 'P', 'N', 'P', 'N')
score = c(0.9, 0.8, 0.7, 0.6, 0.55, 0.54, 0.53, 0.52,
        0.51, 0.505, 0.4, 0.39, 0.38, 0.37, 0.36,
        0.35, 0.34, 0.33, 0.3, 0.1)

pred = prediction(score, cls)
roc = performance(pred, "tpr", "fpr")

#plot(roc, lwd=2, colorize=TRUE)
#lines(x=c(0, 1), y=c(0, 1), col="black", lwd=1)

auc = performance(pred, "auc")
auc = unlist(auc@y.values)
auc

## [1] 0.68
```

```
acc = performance(pred, "acc")
acc

## An object of class "performance"
## Slot "x.name":
## [1] "Cutoff"
##
## Slot "y.name":
## [1] "Accuracy"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## [[1]]
## [1] Inf 0.900 0.800 0.700 0.600 0.550 0.540 0.530 0.520 0.510 0.505
## [12] 0.400 0.390 0.380 0.370 0.360 0.350 0.340 0.330 0.300 0.100
##
## Slot "y.values":
## [[1]]
## [1] 0.50 0.55 0.60 0.55 0.60 0.65 0.70 0.65 0.60 0.65 0.60 0.65 0.60 0.65
## [15] 0.60 0.55 0.50 0.55 0.50 0.55 0.50
##
## Slot "alpha.values":
```

```
## list()

ac.val = max(unlist(acc@y.values))
ac.val

## [1] 0.7

th = unlist(acc@x.values)[unlist(acc@y.values) == ac.val]
th

## [1] 0.54

#plot(acc)
#abline(v=th, col='grey', lty=2)

#Logistic regression example with Default dataset
library(ISLR)
attach(Default)
str(Default)

## 'data.frame': 10000 obs. of  4 variables:
## $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 ...
## $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 ...
## $ balance: num  730 817 1074 529 786 ...
## $ income : num  44362 12106 31767 35704 38463 ...

dim(Default)
```

```
## [1] 10000      4

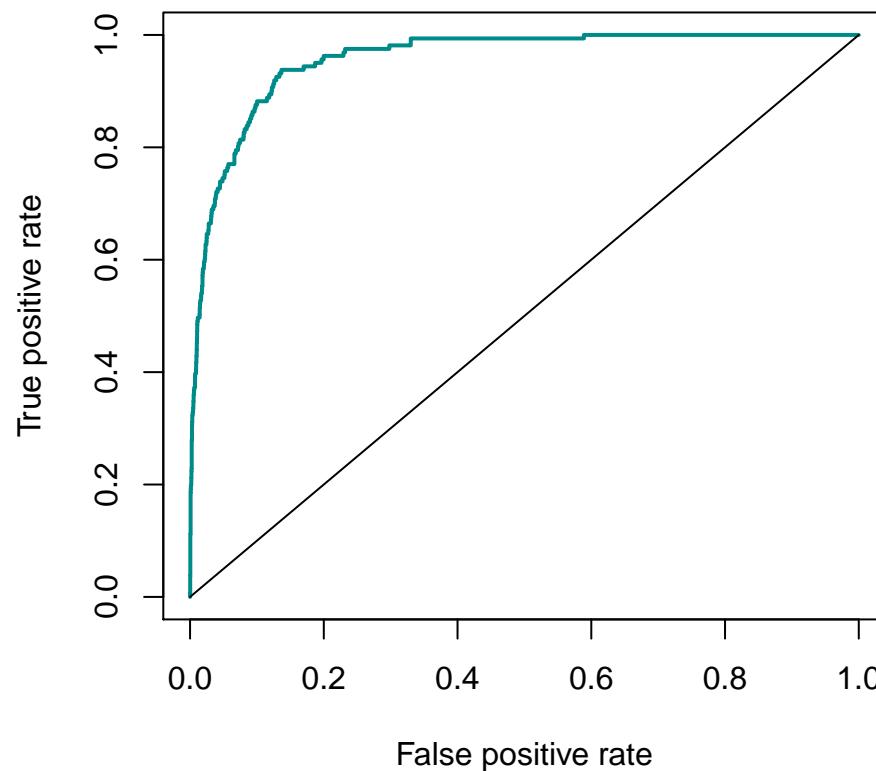
set.seed(1234)
train=sample(1:10000,5000)
test=Default[-train,]
g=glm(default~student+income+balance,family=binomial(link=logit),subset=train)
summary(g)

##
## Call:
## glm(formula = default ~ student + income + balance, family = binomial(link = logit),
##       subset = train)
##
## Deviance Residuals:
##       Min      1Q  Median      3Q     Max
## -2.0769 -0.1527 -0.0620 -0.0238  3.6193
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.058e+01  6.653e-01 -15.908 <2e-16 ***
## studentYes  -5.027e-01  3.175e-01  -1.583   0.113
## income       6.690e-06  1.104e-05   0.606   0.544
## balance      5.472e-03  3.102e-04  17.640 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##  
##      Null deviance: 1497.2  on 4999  degrees of freedom  
## Residual deviance:  837.0  on 4996  degrees of freedom  
## AIC: 845  
##  
## Number of Fisher Scoring iterations: 8  
  
score<-predict(g,newdata=Default,type="response")[-train]  
cls<-Default[-train,1]  
  
pred = prediction(score, cls)  
roc = performance(pred, "tpr", "fpr")  
  
#plot(roc, lwd=2, colorize=T)  
#plot(roc, lwd=2, col="darkcyan")  
#lines(x=c(0, 1), y=c(0, 1), col="black", lwd=1)  
  
auc = performance(pred, "auc")  
auc = unlist(auc@y.values)  
auc  
  
## [1] 0.95703  
  
acc = performance(pred, "acc")  
ac.val = max(unlist(acc@y.values))  
th = unlist(acc@x.values)[unlist(acc@y.values) == ac.val]
```

```
#plot(acc)
#abline(v=th, col='grey', lty=2)
```

Figure 9.5: ROC curve

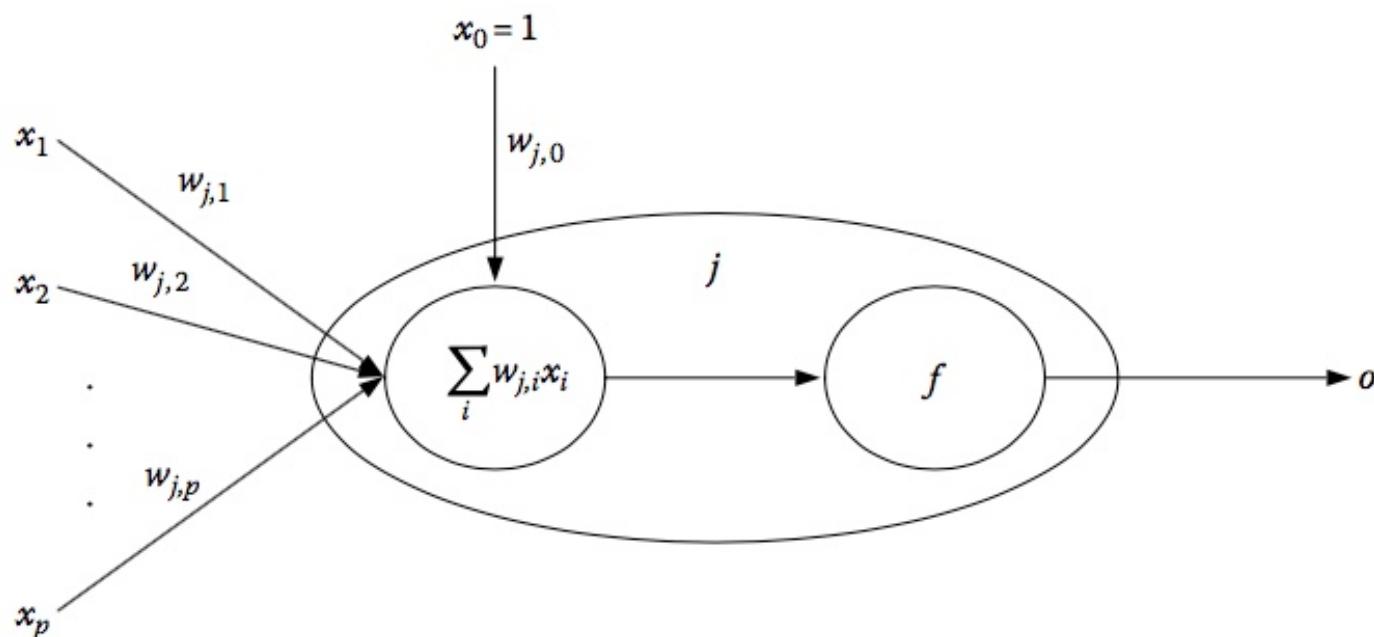


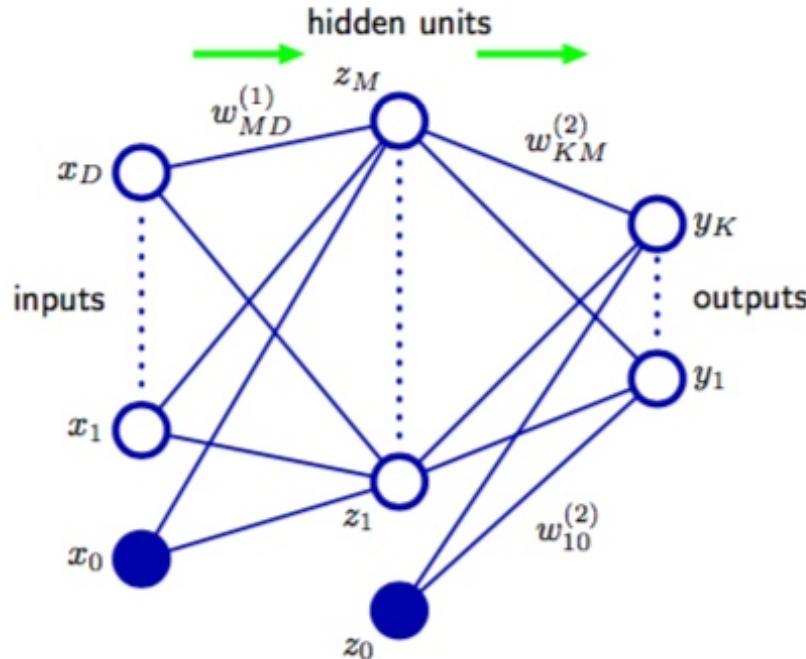
Chapter 10 Neural Network

- The term “neural network” has its origins in attempts to find mathematical representations of information processing in biological systems (McCulloch and Pitts, 1943; Widrow and Hoff, 1960; Rosenblatt, 1962; Rumelhart et al., 1986).
- Feed-forward neural network (multi-layer perceptions (MLP))
- Backpropagation
- Deep neural network (DNN), convolution neural network (CNN), recurrent neural network (RNN), deep belief network (DBN), etc.

10.1 Feed-forward neural network

Figure 10.1: Processing Units of Neural Networks





•

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=1}^M w_{kj}^{(2)} h \left(\sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \quad (10.1)$$

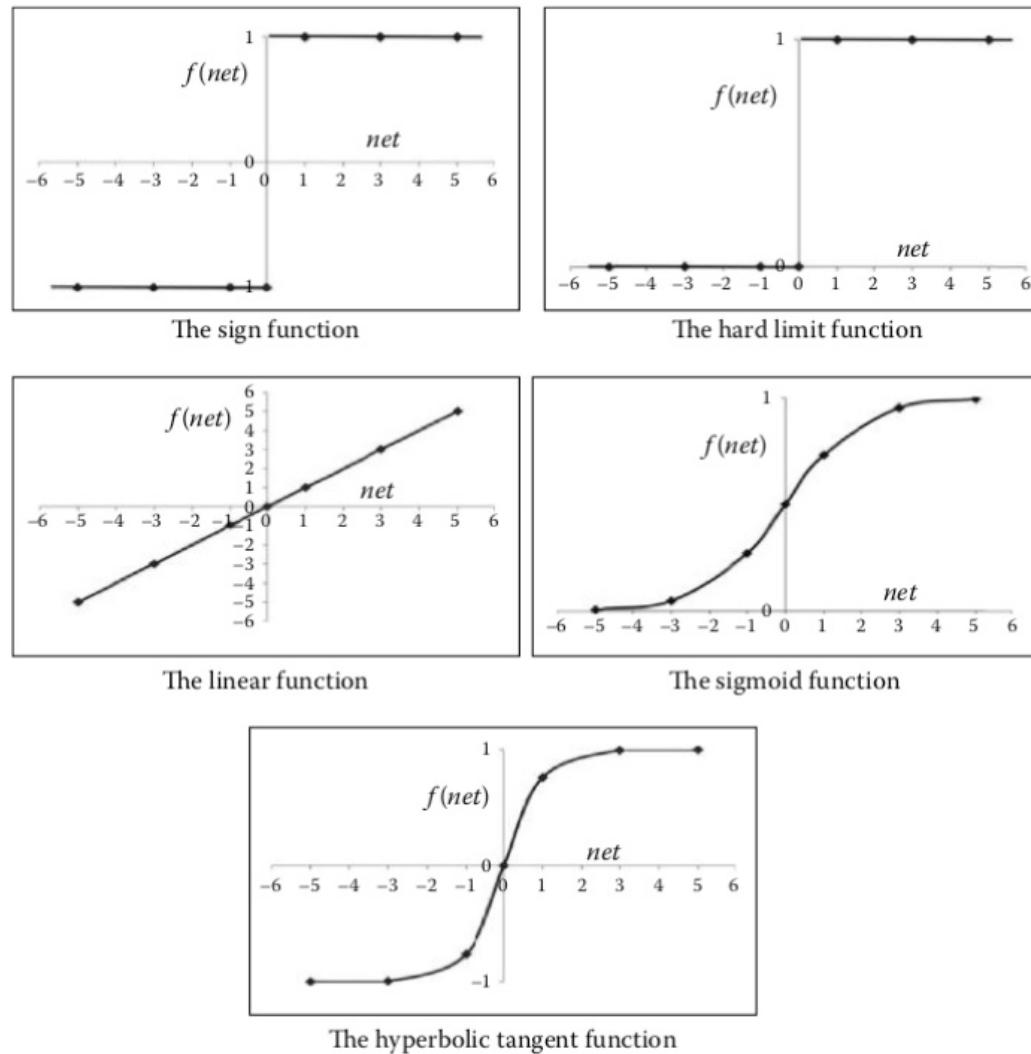
- This overall network function is deterministic rather than stochastic. The terminology of the terms in a neural network is given in the following table:

$w_{ji}^{(1)}$ (for $i \geq 1$), $w_{kj}^{(2)}$ (for $j \geq 1$)	weights
$w_{j0}^{(1)}, w_{k0}^{(2)}$	biases
h, σ	activation functions
a_j	activation
$z_j = h(a_j)$	hidden unit

- Some activation functions

activation function	h	h'
sigmoid	$h(a) = \frac{e^a}{1+e^a}$	$h'(a) = h(a)(1 - h(a))$
hyperbolic tangent	$h(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$	$h'(a) = 1 - h^2(a)$
rectified linear unit (ReLU)	$h(a) = \max(0, a)$	$h'(a) = I(x > 0)$

Figure 10.2: Activation functions



- The universal approximation theorem shows a neural network can fit any continuous function as long as we allow a sufficient number of neurons.

Theorem 1 (Universal approximation theorem). Let σ be a non-constant, bounded, and monotonically increasing continuous function such as a sigmoid function. Let $I^m = [0, 1]^m$ and $C(I^m)$ be the space of continuous functions on I^m . Then, for any $\epsilon > 0$ and any function $f \in C(I^m)$, there exists an integer N , $c_i, b_i \in \mathbb{R}$, and $\mathbf{w} \in \mathbb{R}^m$ ($i = 1, \dots, N$), such that $F(\mathbf{x}) = \sum_{i=1}^N c_i \sigma(\mathbf{w}^T \mathbf{x} + b_i)$ approximates f , that is, $|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon$ for all $\mathbf{x} \in \mathbb{R}^m$.

10.2 Estimation

- Objective functional
 - For a regression problem, we can minimize

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N \|\mathbf{t}_i - \mathbf{f}(\mathbf{x}_i, \mathbf{w})\|^2 \quad (10.2)$$

It is equivalent to maximizing a likelihood function of a normal distribution.

- For a binary classification,

$$L = \prod_{i=1}^N (f(\mathbf{x}_i, \mathbf{w}))^{t_i} (1 - f(\mathbf{x}, \mathbf{w}))^{1-t_i} \quad (10.3)$$

$$\ell = \sum_{i=1}^N [t_i \log f(\mathbf{x}_i, \mathbf{w}) + (1 - t_i) \log(1 - f(\mathbf{x}_i, \mathbf{w}))] \quad (10.4)$$

$$E(\mathbf{w}) = - \sum_{i=1}^N [t_i \log f(\mathbf{x}_i, \mathbf{w}) + (1 - t_i) \log(1 - f(\mathbf{x}_i, \mathbf{w}))] \quad (10.5)$$

is then a cross-entropy error function.

- For a multiclass classification problem,

$$L = \prod_{i=1}^N \prod_{g=1}^G (f_g(\mathbf{x}_i, \mathbf{w}))^{t_{gi}} \quad (10.6)$$

$$\ell = \sum_{i=1}^N \sum_{g=1}^G [t_{gi} \log f_g(\mathbf{x}_i, \mathbf{w})] \quad (10.7)$$

$$E(\mathbf{w}) = - \sum_{i=1}^N \sum_{g=1}^G [t_{gi} \log f_g(\mathbf{x}_i, \mathbf{w})] \quad (10.8)$$

- Optimization: batch methods or on-line (sequential or stochastic) methods

- Gradient descent method: the r th step is given by

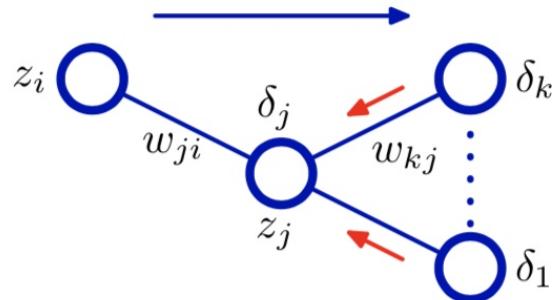
$$\mathbf{w}^{[r+1]} = \mathbf{w}^{[r]} - \eta \nabla E(\mathbf{w}^{[r]}) \quad (10.9)$$

$$\mathbf{w}^{[r+1]} = \mathbf{w}^{[r]} - \eta \nabla E_i(\mathbf{w}^{[r]}) \quad (10.10)$$

where $\eta > 0$ is a learning rate.

- Backpropagation

Figure 10.3: Backpropagation



$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w}) \quad (10.11)$$

$$E_n(\mathbf{w}) = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \quad (10.12)$$

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \quad (10.13)$$

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j \quad (10.14)$$

$$\delta_k = (y_k - t_k) \sigma'(a_k^{(2)}) \quad (10.15)$$

$$\delta_j = \sum_k (y_k - t_k) \sigma'(a_k^{(2)}) w_{kj}^{(2)} h'(a_j^{(1)}) \quad (10.16)$$

$$= \sum_k w_{kj}^{(2)} \delta_k h'(a_j^{(1)}) \quad (10.17)$$

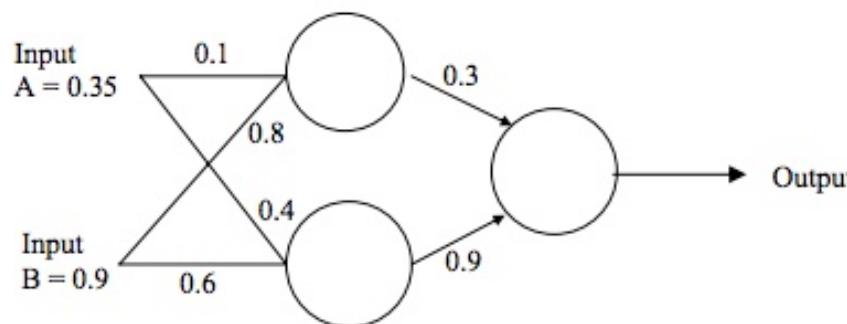
where $a_j^{(1)} = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)}$ and $a_k^{(2)} = \sum_{j=1}^M w_{kj}^{(2)} h \left(\sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)}$.

Example 10.2.1 Worked example for backpropagation

Assume that the neurons have a Sigmoid activation function and the learning rate is equal to 1.

- (i) Perform a forward pass on the network.
- (ii) Perform a reverse pass (training) once (target = 0.5).
- (iii) Perform a further forward pass and comment on the result.

Figure 10.4: A worked example of the backpropagation



$$o = h_1(w_1h(w_3x_1 + w_4x_2) + w_2h(w_5x_1 + w_6x_2)) \quad (10.18)$$

Answer:

- (i) Input to top neuron = $(0.35 \times 0.1) + (0.9 \times 0.8) = 0.755$. Out = 0.68.
 Input to bottom neuron = $(0.9 \times 0.6) + (0.35 \times 0.4) = 0.68$. Out = 0.6637.
 Input to final neuron = $(0.3 \times 0.68) + (0.9 \times 0.6637) = 0.80133$. Out = 0.69.

- (ii) Output error $\delta = (t - o)(1 - o)o = (0.5 - 0.69)(1 - 0.69)0.69 = -0.0406$.

New weights for output layer

$$w1^+ = w1 + (\delta \times \text{input}) = 0.3 + (-0.0406 \times 0.68) = 0.272392.$$

$$w2^+ = w2 + (\delta \times \text{input}) = 0.9 + (-0.0406 \times 0.6637) = 0.87305.$$

Errors for hidden layers:

$$\delta_1 = \delta \times w1 \times (1 - o)o = -0.0406 \times 0.272392 \times (1 - o)o = -2.406 \times 10^{-3}$$

$$\delta_2 = \delta \times w2 \times (1 - o)o = -0.0406 \times 0.87305 \times (1 - o)o = -7.916 \times 10^{-3}$$

New hidden layer weights:

$$w3^+ = 0.1 + (-2.406 \times 10^{-3} \times 0.35) = 0.09916.$$

$$w4^+ = 0.8 + (-2.406 \times 10^{-3} \times 0.9) = 0.7978.$$

$$w5^+ = 0.4 + (-7.916 \times 10^{-3} \times 0.35) = 0.3972.$$

$$w6^+ = 0.6 + (-7.916 \times 10^{-3} \times 0.9) = 0.5928.$$

- (iii) Old error was -0.19 . New error is -0.18205 . Therefore error has reduced.

10.3 Examples with R

Example 10.3.1

(iris data)

```
library(nnet)
data(iris)
dat <- iris
dat$setosa <- ifelse(iris$Species == "setosa", 1, 0)
dat$versicolor <- ifelse(iris$Species == "versicolor", 1, 0)
dat$virginica <- ifelse(iris$Species == "virginica", 1, 0)
dat$Species <- NULL

n <- dim(dat)[1]
set.seed(100)
train <- sample(1:n, 0.6 * n)

fit <- nnet(dat[train, 1:4], dat[train, 5:7], size = 5, softmax = TRUE, maxit = 1000,
abstol = 1e-07)

## # weights:  43
## initial  value 121.933990
## iter  10 value 45.186297
## iter  20 value 32.548721
## iter  30 value 5.223747
## iter  40 value 1.190608
## iter  50 value 0.007148
```

```
## iter  60 value 0.000028
## final  value 0.000000
## converged

pred <- predict(fit, dat[-train, 1:4])
pred

##           setosa    versicolor    virginica
## 1  1.000000e+00 9.255320e-101  0.000000e+00
## 2  1.000000e+00 5.124470e-100  0.000000e+00
## 4  1.000000e+00 4.532018e-100  0.000000e+00
## 5  1.000000e+00 6.722816e-101  0.000000e+00
## 6  1.000000e+00 6.568026e-100  0.000000e+00
## 8  1.000000e+00 1.753710e-100  0.000000e+00
## 10 1.000000e+00 1.830042e-100  0.000000e+00
## 11 1.000000e+00 8.101083e-101  0.000000e+00
## 13 1.000000e+00 1.668558e-100  0.000000e+00
## 14 1.000000e+00 5.092075e-101  0.000000e+00
## 17 1.000000e+00 1.168683e-100  0.000000e+00
## 18 1.000000e+00 2.297490e-100  0.000000e+00
## 23 1.000000e+00 2.230210e-101  0.000000e+00
## 29 1.000000e+00 1.315823e-100  0.000000e+00
## 30 1.000000e+00 5.084001e-100  0.000000e+00
## 35 1.000000e+00 5.422433e-100  0.000000e+00
## 36 1.000000e+00 1.111998e-100  0.000000e+00
## 40 1.000000e+00 1.840096e-100  0.000000e+00
## 41 1.000000e+00 1.471177e-100  0.000000e+00
```

```
## 43 1.000000e+00 1.224048e-100 0.000000e+00
## 44 1.000000e+00 1.045289e-97 0.000000e+00
## 46 1.000000e+00 1.839122e-99 0.000000e+00
## 49 1.000000e+00 7.793867e-101 0.000000e+00
## 50 1.000000e+00 1.677522e-100 0.000000e+00
## 54 2.461356e-239 1.000000e+00 0.000000e+00
## 56 1.408959e-252 1.000000e+00 5.676814e-321
## 60 5.017420e-199 1.000000e+00 0.000000e+00
## 62 5.759685e-258 1.000000e+00 1.323606e-312
## 65 2.236773e-98 1.000000e+00 0.000000e+00
## 68 1.068111e-94 1.000000e+00 0.000000e+00
## 73 0.000000e+00 1.000000e+00 1.966584e-46
## 74 1.713926e-253 1.000000e+00 1.495784e-319
## 76 8.142558e-265 1.000000e+00 5.732223e-302
## 79 6.373447e-322 1.000000e+00 2.739285e-213
## 83 5.920765e-135 1.000000e+00 0.000000e+00
## 87 0.000000e+00 1.000000e+00 8.620260e-188
## 93 1.606787e-163 1.000000e+00 0.000000e+00
## 94 7.802451e-31 1.000000e+00 0.000000e+00
## 97 1.749076e-189 1.000000e+00 0.000000e+00
## 98 2.124423e-215 1.000000e+00 0.000000e+00
## 100 1.131504e-187 1.000000e+00 0.000000e+00
## 101 0.000000e+00 7.787027e-174 1.000000e+00
## 105 0.000000e+00 3.168338e-157 1.000000e+00
## 107 0.000000e+00 1.000000e+00 1.166891e-61
## 108 0.000000e+00 4.240511e-146 1.000000e+00
```

```
## 113 0.000000e+00 1.977380e-135 1.000000e+00
## 115 0.000000e+00 7.864135e-153 1.000000e+00
## 122 0.000000e+00 2.792312e-74 1.000000e+00
## 126 0.000000e+00 2.793260e-113 1.000000e+00
## 128 0.000000e+00 9.974161e-01 2.583881e-03
## 130 0.000000e+00 1.379377e-68 1.000000e+00
## 131 0.000000e+00 4.858084e-152 1.000000e+00
## 132 0.000000e+00 1.298354e-139 1.000000e+00
## 134 0.000000e+00 1.000000e+00 2.603428e-58
## 136 0.000000e+00 8.969343e-175 1.000000e+00
## 139 0.000000e+00 1.000000e+00 4.991142e-21
## 140 0.000000e+00 2.538714e-125 1.000000e+00
## 144 0.000000e+00 1.505217e-163 1.000000e+00
## 145 0.000000e+00 2.050368e-168 1.000000e+00
## 147 0.000000e+00 7.311548e-94 1.000000e+00

pcl <- apply(pred, 1, which.max)
predcl <- rep("setosa", length(pcl))
predcl[pcl == 2] <- "versicolor"
predcl[pcl == 3] <- "virginica"

table(pred = predcl, true = iris$Species[-train])

##          true
## pred      setosa versicolor virginica
##   setosa     24        0        0
##   versicolor    0       17        4
```

```
##      virginica      0       15

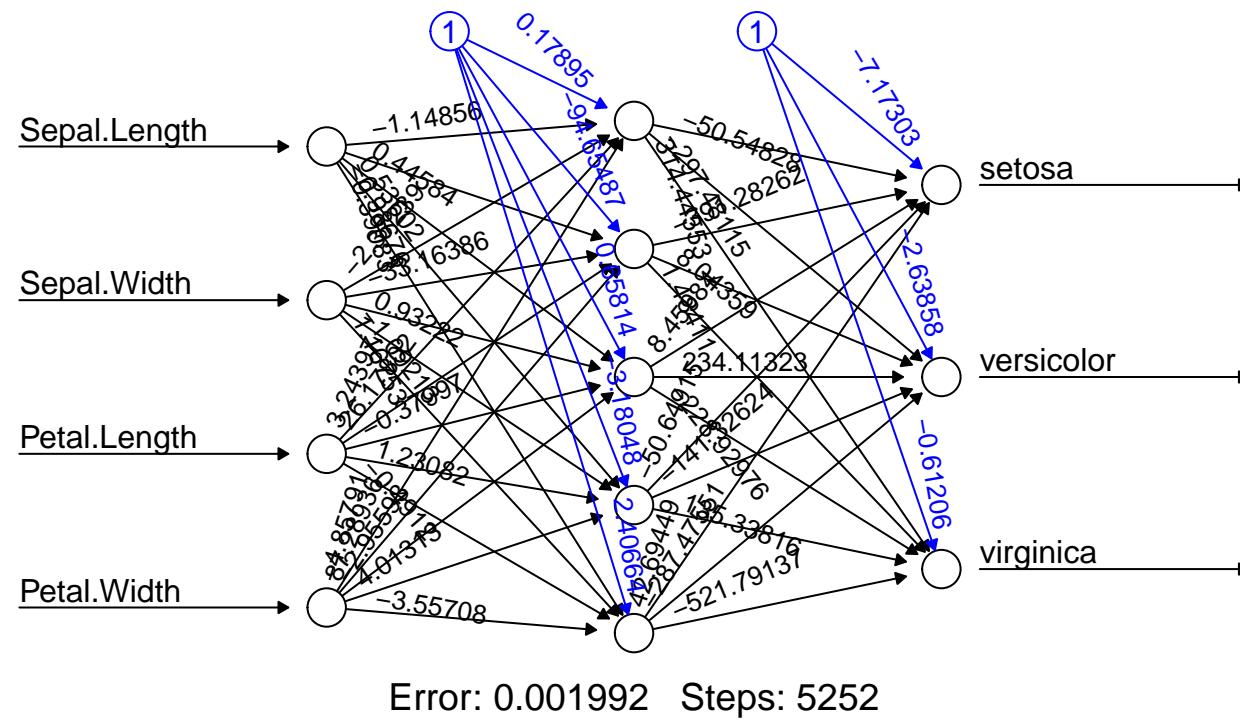
library(NeuralNetTools)
# plotnet(fit)

library(neuralnet)
fitnn <- neuralnet(setosa + versicolor + virginica ~ Sepal.Length + Sepal.Width +
  Petal.Length + Petal.Width, dat = dat[train, ], hidden = c(5), err.fct = "ce",
  act.fct = "logistic", linear.output = FALSE)

pred2 <- compute(fitnn, dat[-train, 1:4])$net.result
pcl2 <- apply(pred2, 1, which.max) #or use max.col
predcl2 <- rep("setosa", length(pcl2))
predcl2[pcl2 == 2] <- "versicolor"
predcl2[pcl2 == 3] <- "virginica"
table(pred = predcl2, true = iris$Species[-train])

##          true
## pred      setosa versicolor virginica
##   setosa      24        0        0
##   versicolor    0       17        4
##   virginica     0        0       15
```

Figure 10.5: The results of the neural network model



Example 10.3.2

(Heart data)

```
hdata <- read.csv("Heart.csv", header = TRUE)
sum(is.na(hdata))

## [1] 6

hdata <- na.omit(hdata)
str(hdata)

## 'data.frame': 297 obs. of 15 variables:
## $ X      : int  1 2 3 4 5 6 7 8 9 10 ...
## $ Age    : int  63 67 67 37 41 56 62 57 63 53 ...
## $ Sex    : int  1 1 1 1 0 1 0 0 1 1 ...
## $ ChestPain: Factor w/ 4 levels "asymptomatic",...: 4 1 1 2 3 3 1 1 1 1 ...
## $ RestBP  : int  145 160 120 130 130 120 140 120 130 140 ...
## $ Chol   : int  233 286 229 250 204 236 268 354 254 203 ...
## $ Fbs    : int  1 0 0 0 0 0 0 0 0 1 ...
## $ RestECG : int  2 2 2 0 2 0 2 0 2 2 ...
## $ MaxHR  : int  150 108 129 187 172 178 160 163 147 155 ...
## $ ExAng  : int  0 1 1 0 0 0 0 1 0 1 ...
## $ Oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
## $ Slope   : int  3 2 2 3 1 1 3 1 2 3 ...
## $ Ca     : int  0 3 2 0 0 0 2 0 1 0 ...
## $ Thal   : Factor w/ 3 levels "fixed","normal",...: 1 2 3 2 2 2 2 2 3 3 ...
## $ AHD    : Factor w/ 2 levels "No","Yes": 1 2 2 1 1 1 2 1 2 2 ...
```

```
## - attr(*, "na.action")= 'omit' Named int  88 167 193 267 288 303
## ..- attr(*, "names")= chr  "88" "167" "193" "267" ...
hd <- hdata[, -1]
n <- dim(hd)[1]
hd$Yes <- ifelse(hdata$AHD == "Yes", 1, 0)
hd$No <- ifelse(hdata$AHD == "No", 1, 0)
levels(hd$ChestPain)

## [1] "asymptomatic" "nonanginal"    "nontypical"   "typical"

hd$CPa <- ifelse(hd$ChestPain == "asymptomatic", 1, 0)
hd$CPna <- ifelse(hd$ChestPain == "nonanginal", 1, 0)
hd$CPnt <- ifelse(hd$ChestPain == "nontypical", 1, 0)
hd$CPt <- ifelse(hd$ChestPain == "typical", 1, 0)
hd$ThalF <- ifelse(hd$Thal == "fixed", 1, 0)
hd$ThalN <- ifelse(hd$Thal == "normal", 1, 0)
hd$ThalR <- ifelse(hd$Thal == "reversible", 1, 0)

str(hd)

## 'data.frame': 297 obs. of  23 variables:
## $ Age      : int  63 67 67 37 41 56 62 57 63 53 ...
## $ Sex      : int  1 1 1 1 0 1 0 0 1 1 ...
## $ ChestPain: Factor w/ 4 levels "asymptomatic",...: 4 1 1 2 3 3 1 1 1 1 ...
## $ RestBP   : int  145 160 120 130 130 120 140 120 130 140 ...
## $ Chol     : int  233 286 229 250 204 236 268 354 254 203 ...
```

```
## $ Fbs      : int 1 0 0 0 0 0 0 0 0 1 ...
## $ RestECG   : int 2 2 2 0 2 0 2 0 2 2 ...
## $ MaxHR     : int 150 108 129 187 172 178 160 163 147 155 ...
## $ ExAng     : int 0 1 1 0 0 0 0 1 0 1 ...
## $ Oldpeak   : num 2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
## $ Slope     : int 3 2 2 3 1 1 3 1 2 3 ...
## $ Ca        : int 0 3 2 0 0 0 2 0 1 0 ...
## $ Thal      : Factor w/ 3 levels "fixed","normal",...: 1 2 3 2 2 2 2 2 3 3 ...
## $ AHD       : Factor w/ 2 levels "No","Yes": 1 2 2 1 1 1 2 1 2 2 ...
## $ Yes       : num 0 1 1 0 0 0 1 0 1 1 ...
## $ No        : num 1 0 0 1 1 1 0 1 0 0 ...
## $ CPa       : num 0 1 1 0 0 0 1 1 1 1 ...
## $ CPna      : num 0 0 0 1 0 0 0 0 0 0 ...
## $ CPnt      : num 0 0 0 0 1 1 0 0 0 0 ...
## $ CPt       : num 1 0 0 0 0 0 0 0 0 0 ...
## $ ThalF     : num 1 0 0 0 0 0 0 0 0 0 ...
## $ ThalN     : num 0 1 0 1 1 1 1 1 0 0 ...
## $ ThalR     : num 0 0 1 0 0 0 0 0 1 1 ...

hd$ChestPain <- NULL
hd$Thal <- NULL
hd$AHD <- NULL
str(hd)

## 'data.frame': 297 obs. of 20 variables:
## $ Age      : int 63 67 67 37 41 56 62 57 63 53 ...
## $ Sex      : int 1 1 1 1 0 1 0 0 1 1 ...
```

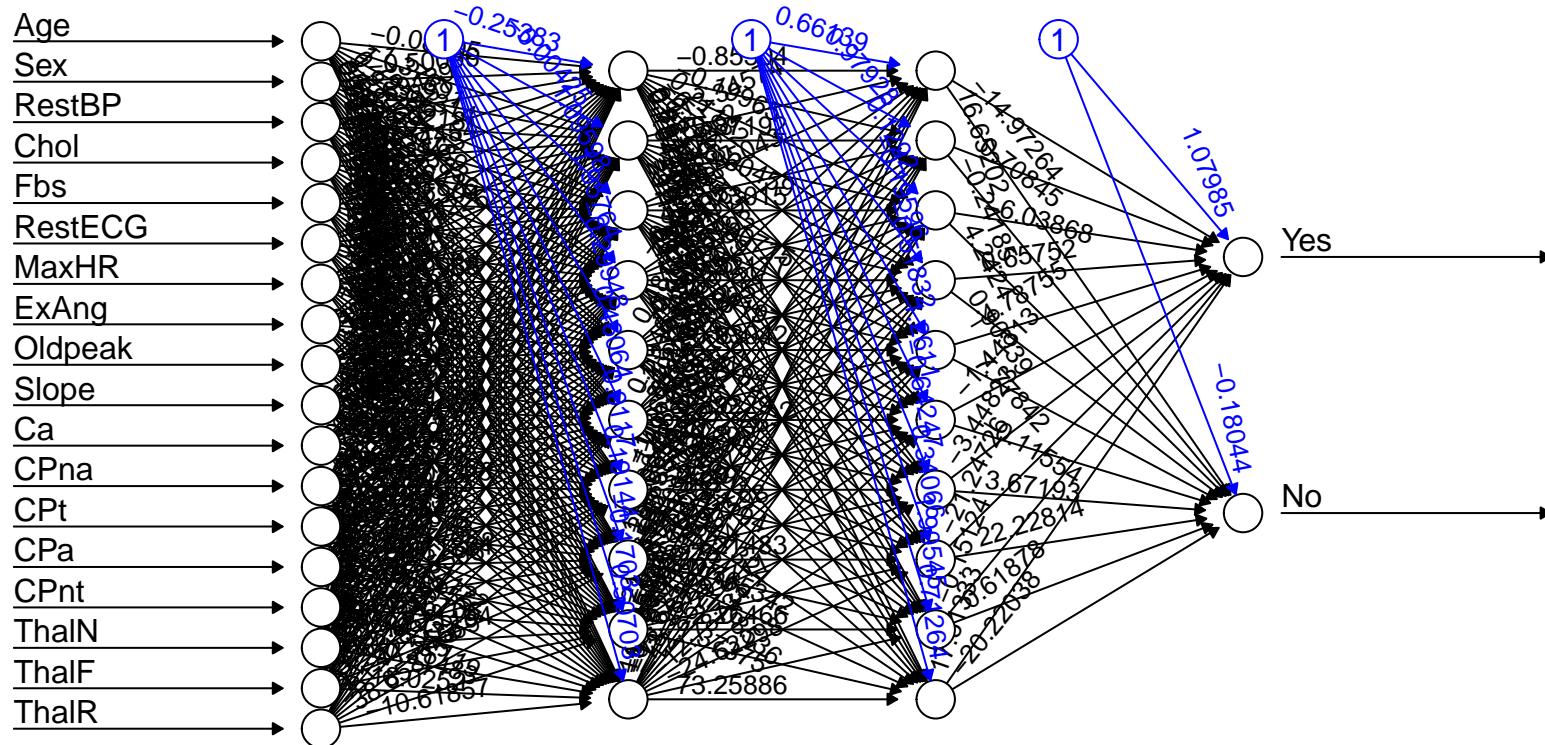
```
## $ RestBP : int 145 160 120 130 130 120 140 120 130 140 ...
## $ Chol    : int 233 286 229 250 204 236 268 354 254 203 ...
## $ Fbs     : int 1 0 0 0 0 0 0 0 1 ...
## $ RestECG: int 2 2 2 0 2 0 2 0 2 2 ...
## $ MaxHR   : int 150 108 129 187 172 178 160 163 147 155 ...
## $ ExAng   : int 0 1 1 0 0 0 0 1 0 1 ...
## $ Oldpeak: num 2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
## $ Slope   : int 3 2 2 3 1 1 3 1 2 3 ...
## $ Ca      : int 0 3 2 0 0 0 2 0 1 0 ...
## $ Yes     : num 0 1 1 0 0 0 1 0 1 1 ...
## $ No      : num 1 0 0 1 1 1 0 1 0 0 ...
## $ CPa    : num 0 1 1 0 0 0 1 1 1 1 ...
## $ CPna   : num 0 0 0 1 0 0 0 0 0 0 ...
## $ CPnt   : num 0 0 0 0 1 1 0 0 0 0 ...
## $ CPt    : num 1 0 0 0 0 0 0 0 0 0 ...
## $ ThalF  : num 1 0 0 0 0 0 0 0 0 0 ...
## $ ThalN  : num 0 1 0 1 1 1 1 1 0 0 ...
## $ ThalR  : num 0 0 1 0 0 0 0 0 1 1 ...

set.seed(631560)
train <- sample(1:n, n/2)
fith <- neuralnet(Yes + No ~ Age + Sex + RestBP + Chol + Fbs + RestECG + MaxHR +
  ExAng + Oldpeak + Slope + Ca + CPna + CPt + CPa + CPnt + ThalN + ThalF +
  ThalR, data = hd[train, ], hidden = c(10, 10), err.fct = "ce", act.fct = "logistic",
  linear.output = FALSE)
# plot(fith, rep = 'best')
```

```
predh <- compute(fith, hd[-train, -c(12, 13)])$net.result
pclh <- max.col(predh)
predclh <- rep("Yes", length(pclh))
predclh[pclh == 2] <- "No"
table(pred = predclh, true = hdata$AHD[-train])

##      true
## pred  No Yes
##   No  71  36
##   Yes 11  31
```

Figure 10.6: The results of the neural network model



```
library(keras)
n <- dim(iris)[1]
set.seed(1)
train <- sample(n, n * 0.6)
y <- to_categorical(as.numeric(iris$Species) - 1, 3)
x <- as.matrix(iris[, -5])
x_train <- x[train, ]
y_train <- y[train, ]
x_test <- x[-train, ]
y_test <- y[-train, ]
model <- keras_model_sequential()
model %>% layer_dense(units = 4, activation = "relu", input_shape = c(4)) %>%
  layer_dropout(rate = 0.1) %>% layer_dense(units = 2, activation = "relu") %>%
  layer_dropout(rate = 0.1) %>% layer_dense(units = 3, activation = "softmax")

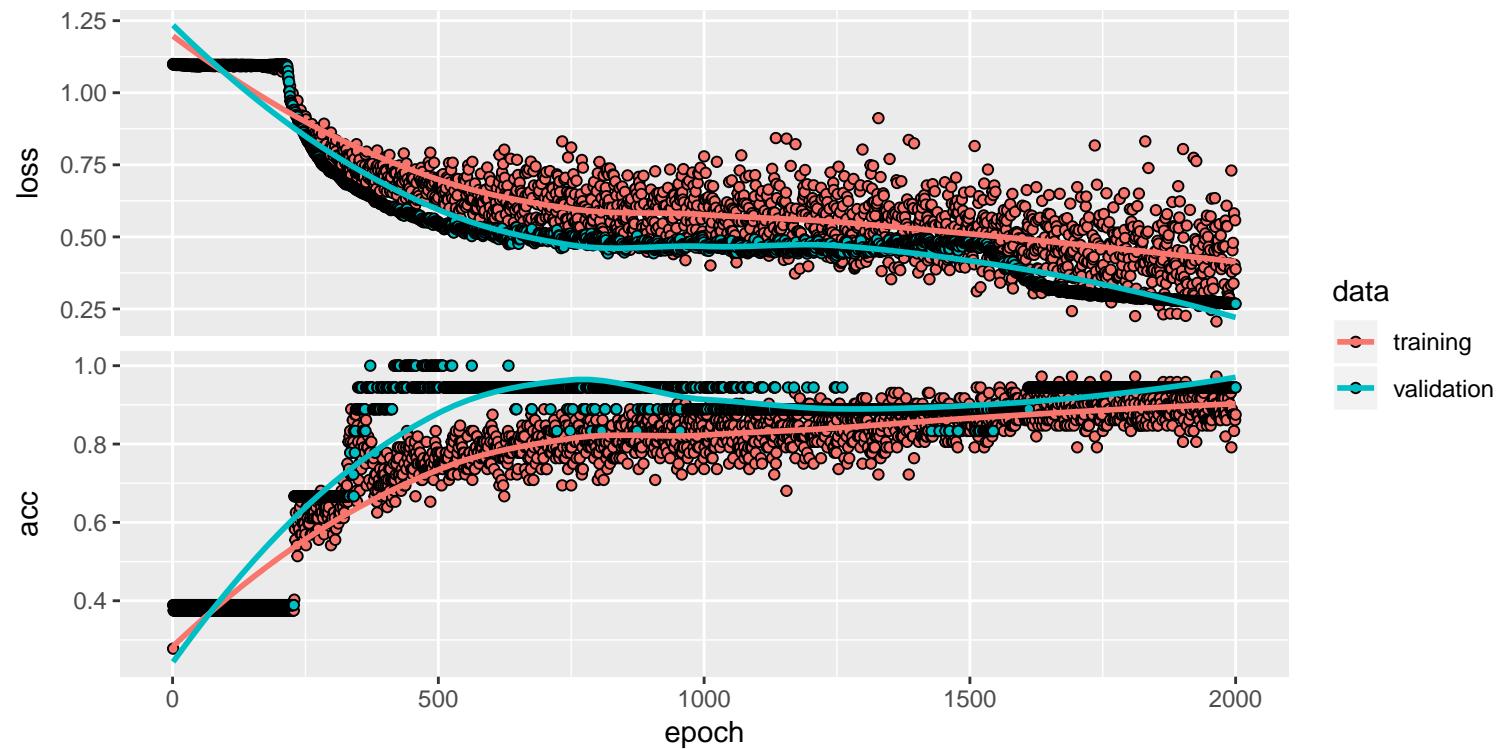
summary(model)

## -----
## Layer (type)          Output Shape         Param #
## =====
## dense_1 (Dense)      (None, 4)           20
## -----
## dropout_1 (Dropout)   (None, 4)           0
## -----
## dense_2 (Dense)      (None, 2)           10
## -----
```

```
## dropout_2 (Dropout)           (None, 2)          0
## -----
## dense_3 (Dense)             (None, 3)          9
## =====
## Total params: 39
## Trainable params: 39
## Non-trainable params: 0
## -----  
  
model %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(),
  metrics = c("accuracy"))  
  
set.seed(54321)
history <- model %>% fit(x_train, y_train, epochs = 2000, batch_size = 10, validation_split = 0.2)  
  
plot(history)
evalm <- model %>% evaluate(x_test, y_test)  
  
evalm  
  
## $loss
## [1] 0.1912598
##
## $acc
## [1] 0.9666667  
  
pred.d <- model %>% predict_classes(x_test)
```

```
table(pred.d, iris$Species[-train])  
##  
## pred.d setosa versicolor virginica  
##    0      21        0       0  
##    1      0       15       1  
##    2      0       1      22
```

Figure 10.7: History plot of training a deep neural network to iris data

**Example 10.3.3**

Deep neural network and convolutional neural network with MNIST data

```
library(keras)
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

# reshape
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
# rescale
x_train <- x_train / 255
x_test <- x_test / 255

y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)

model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 10, activation = 'softmax')
```

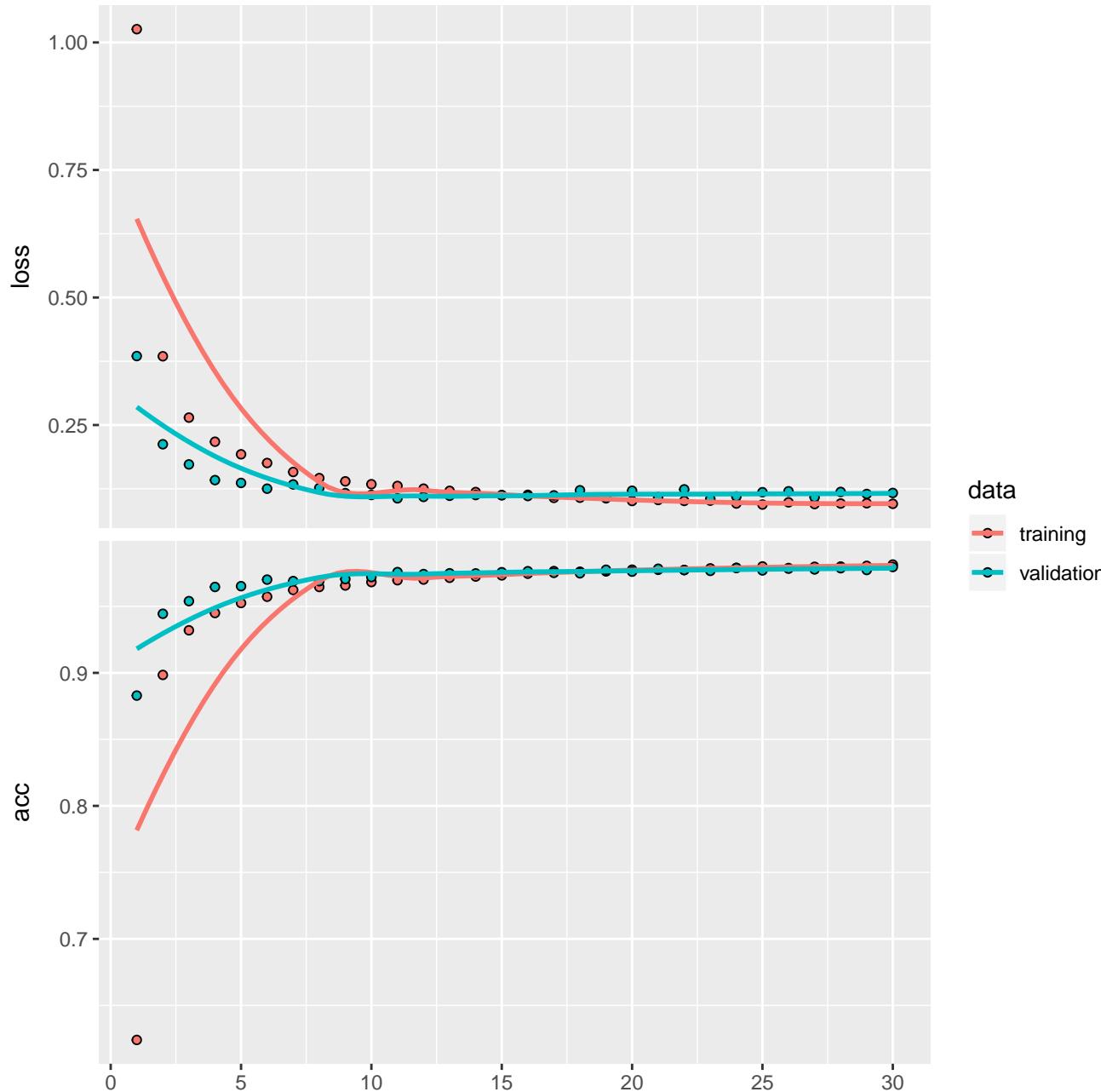
```
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 10) %>%
  layer_activation('softmax')

summary(model)

## -----
## Layer (type)          Output Shape         Param #
## =====
## dense_4 (Dense)      (None, 256)           200960
## -----
## dropout_3 (Dropout)   (None, 256)           0
## -----
## dense_5 (Dense)      (None, 128)            32896
## -----
## dropout_4 (Dropout)   (None, 128)            0
## -----
## dense_6 (Dense)      (None, 128)            16512
```

```
## -----  
## dropout_5 (Dropout)           (None, 128)          0  
## -----  
## dense_7 (Dense)             (None, 10)           1290  
## -----  
## dense_8 (Dense)             (None, 256)          2816  
## -----  
## dropout_6 (Dropout)          (None, 256)          0  
## -----  
## dense_9 (Dense)             (None, 128)          32896  
## -----  
## dropout_7 (Dropout)          (None, 128)          0  
## -----  
## dense_10 (Dense)            (None, 128)          16512  
## -----  
## dropout_8 (Dropout)          (None, 128)          0  
## -----  
## dense_11 (Dense)            (None, 10)           1290  
## -----  
## activation_1 (Activation)   (None, 10)           0  
## ======  
## Total params: 305,172  
## Trainable params: 305,172  
## Non-trainable params: 0  
## -----
```

```
model %>% compile(  
  loss = 'categorical_crossentropy',  
  optimizer = optimizer_rmsprop(),  
  metrics = c('accuracy')  
)  
  
history <- model %>% fit(  
  x_train, y_train,  
  epochs = 30, batch_size = 128,  
  validation_split = 0.2  
)  
  
plot(history)
```



```
model %>% evaluate(x_test, y_test) -> evalm

evalm

## $loss
## [1] 0.1138093
##
## $acc
## [1] 0.9815

model %>% predict_classes(x_test) -> pred.d

dim(y_test)[1] -> K
tclass <- numeric(K)
for (i in 1:K){
    tclass[i] <- which(y_test[, ] == 1) - 1
}

table(pred.d, tclass)

##      tclass
## pred.d   0   1   2   3   4   5   6   7   8   9
##   0    972   1   2   0   2   2   5   3   4   4
##   1    1125   3   0   1   0   3   2   1   2
##   2     0    2 1011   3   2   0   0   5   2   0
##   3     1    1   2 993   0   3   1   2   3   3
##   4     1    0   1   0 962   0   4   0   5   8
```

```
##   5   2   0   0   3   0   875   4   0   7   3  
##   6   1   1   3   0   3   4   940   0   1   0  
##   7   1   0   6   5   2   1   0 1011   3   4  
##   8   1   5   3   4   2   2   1   0  944   3  
##   9   0   0   1   2   8   5   0   5   4  982
```

```
#MNIST CNN  
library(keras)  
  
# Data Preparation -----  
  
batch_size <- 128  
num_classes <- 10  
epochs <- 12  
  
# Input image dimensions  
img_rows <- 28  
img_cols <- 28  
  
# The data, shuffled and split between train and test sets  
mnist <- dataset_mnist()  
x_train <- mnist$train$x  
y_train <- mnist$train$y  
x_test <- mnist$test$x  
y_test <- mnist$test$y
```

```
# Redefine dimension of train/test inputs
x_train <- array_reshape(x_train, c(nrow(x_train), img_rows, img_cols, 1))
x_test <- array_reshape(x_test, c(nrow(x_test), img_rows, img_cols, 1))
input_shape <- c(img_rows, img_cols, 1)

# Transform RGB values into [0,1] range
x_train <- x_train / 255
x_test <- x_test / 255

cat('x_train_shape:', dim(x_train), '\n')

## x_train_shape: 60000 28 28 1

cat(nrow(x_train), 'train samples\n')

## 60000 train samples

cat(nrow(x_test), 'test samples\n')

## 10000 test samples

# Convert class vectors to binary class matrices
y_train <- to_categorical(y_train, num_classes)
y_test <- to_categorical(y_test, num_classes)

# Define Model -----
```

```
# Define model
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu',
    input_shape = input_shape) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = num_classes, activation = 'softmax')

# Compile model
model %>% compile(
  loss = loss_categorical_crossentropy,
  optimizer = optimizer_adadelta(),
  metrics = c('accuracy')
)

# Train model
model %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_split = 0.2
)
```

```
scores <- model %>% evaluate(  
  x_test, y_test, verbose = 0  
)  
  
# Output metrics  
cat('Test loss:', scores[[1]], '\n')  
  
## Test loss: 0.03369323  
  
cat('Test accuracy:', scores[[2]], '\n')  
  
## Test accuracy: 0.9895  
  
model %>% predict_classes(x_test) ->pred.c  
  
dim(y_test)[1]-> K  
tclass<-numeric(K)  
for (i in 1:K){  
  tclass[i]<-which(y_test[i,]==1)-1  
}  
  
table(pred.c,tclass)  
  
##      tclass  
## pred.c    0    1    2    3    4    5    6    7    8    9  
##       0 971    0    1    0    0    0    4    0    1    2
```

```
##   1   1 1131   0   0   0   0   2   1   0   1  
##   2   2   1 1027   2   0   0   0   6   3   0  
##   3   1   1   0 1007   0  10   2   3   7   4  
##   4   0   0   0   0  977   0   1   0   2   2  
##   5   0   1   0   1   0  880   4   0   5   3  
##   6   3   0   0   0   2   2  945   0   0   0  
##   7   0   1   4   0   0   0   0   0 1014   1   3  
##   8   2   0   0   0   0   0   0   1  949   0  
##   9   0   0   0   0   3   0   0   3   6  994
```

Chapter 11 Support Vector Machine

- It was developed in the computer science in the 1990s.
- Support vector machines (SVMs) have been shown to perform well in a variety of settings, and are often considered one of the best “out of the box” classifiers.
- An extension of the support vector classifier that is an extension of the maximal margin classifier

11.1 Separating Hyperplane

- In a p -dimensional space, a hyperplane is a flat affine subspace of dimension $p - 1$.
- In two dimensions, a hyperplane is defined by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

- Easily extended to the p -dimensional setting:

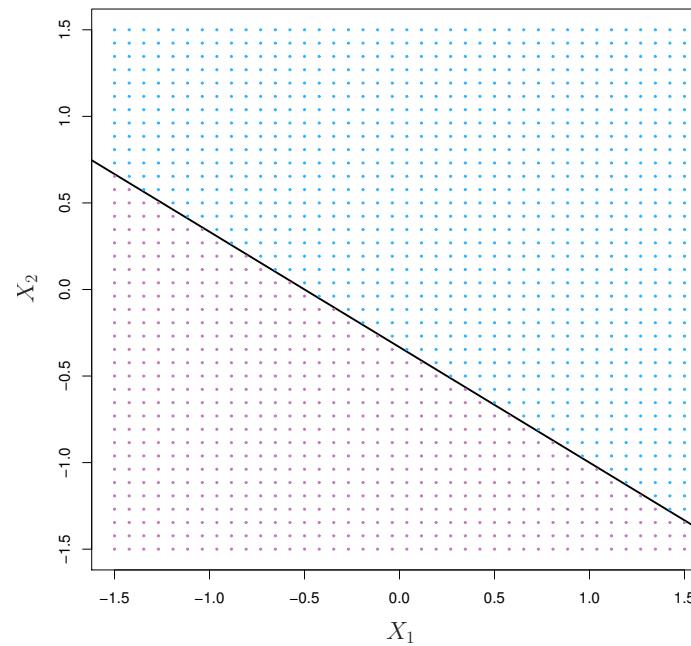
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

- The hyperplane divides p -dimensional space into two halves:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p > 0$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p < 0$$

Figure 11.1: The hyperplane $1 + 2X_1 + 3X_2 = 0$ is shown. The blue region is the set of points for which $1 + 2X_1 + 3X_2 > 0$, and the purple region is the set of points for which $1 + 2X_1 + 3X_2 < 0$.



- A new approach that is based upon the concept of a separating hyperplane.
- Suppose that it is possible to construct a hyperplane that separates the training observations perfectly according to their class labels.
- We use $y_i = 1$ or -1 based on their classes.

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} > 0 \quad \text{for } y_i = 1$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} < 0 \quad \text{for } y_i = -1$$

- By combining two equations,

$$y_i (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) > 0$$

- That is, we classify the test observation x^* based on the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \cdots + \beta_p x_p^*$.

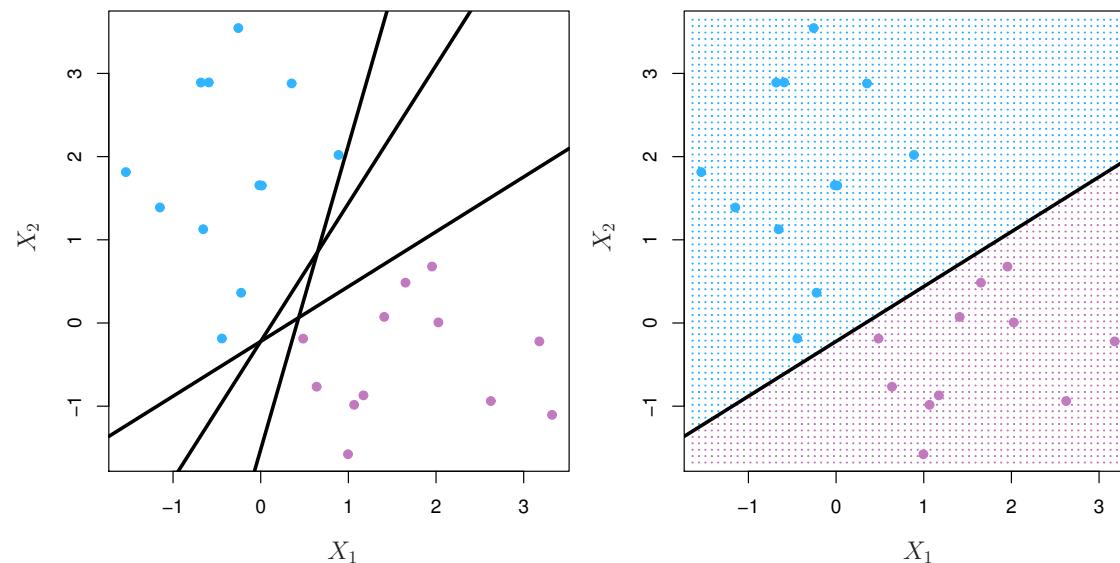
- Assign x^* into $\begin{cases} \text{class 1} & \text{if } f(x^*) > 0 \\ \text{class -1} & \text{if } f(x^*) < 0 \end{cases}$. That is,

$$\hat{y}^* = \text{sign}f(\mathbf{x}^*) \tag{11.1}$$

- Use of the magnitude of $f(x^*)$:

If $f(x^*)$ is far from zero, then this means that x^* lies far from the hyperplane, and so we can be confident about our class assignment for x^* . If $f(x^*)$ is close to zero, then x^* is located near the hyperplane, and so we are less certain about the class assignment for x^* .

Figure 11.2: Left: There are two classes of observations, shown in blue and in purple, each of which has measurements on two variables. Three separating hyperplanes, out of many possible, are shown in black. Right: A separating hyperplane is shown in black. The blue and purple grid indicates the decision rule made by a classifier based on this separating hyperplane: a test observation that falls in the blue portion of the grid will be assigned to the blue class, and a test observation that falls into the purple portion of the grid will be assigned to the purple class.

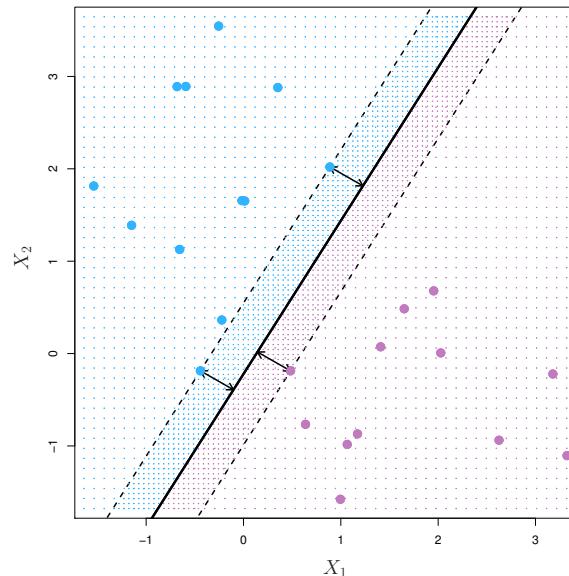


11.2 Maximal margin classifier (Separable case)

- A natural choice is the maximal margin hyperplane (also known as the optimal separating hyperplane), which is the separating hyperplane that is farthest from the training observations.

- The smallest such distance is the minimal distance from the observations to the hyperplane, and is known as the **margin**.
- This is known as the **maximal margin classifier**.
- Only some observations “**support**” the maximal margin hyperplane in the sense that if these points were moved slightly then the maximal margin hyperplane would move as well (Three observations in the following figure).
- Interestingly, the maximal margin hyperplane depends directly on the support vectors, but not on the other observations.

Figure 11.3: There are two classes of observations, shown in blue and in purple. The maximal margin hyperplane is shown as a solid line. The margin is the distance from the solid line to either of the dashed lines. The two blue points and the purple point that lie on the dashed lines are the support vectors, and the distance from those points to the margin is indicated by arrows. The purple and blue grid indicates the decision rule made by a classifier based on this separating hyperplane.



We can construct the maximal margin classifier by

$$\max_{\beta_0, \dots, \beta_p} M \quad (11.2)$$

$$\text{subject to } \sum_{j=1}^p \beta_j^2 = 1 \quad (11.3)$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M \quad (11.4)$$

Let $\mathbf{w} = \frac{1}{M}(\beta_1, \dots, \beta_p)^T$ and $b = \frac{\beta_0}{M}$.

$$\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w} \quad (11.5)$$

$$\text{subject to } 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b) \leq 0, \forall i = 1, \dots, n \quad (11.6)$$

Minimize the Lagrangian function

$$\mathcal{L} = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^n \alpha_i (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)) \quad (11.7)$$

$$\alpha_i \geq 0, \quad \forall i = 1, \dots, n \quad (11.8)$$

$$\frac{\partial}{\partial \mathbf{w}} \mathcal{L} = \mathbf{w} + \sum_{i=1}^n \alpha_i (-y_i) \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (11.9)$$

$$\frac{\partial}{\partial b} \mathcal{L} = \sum_{i=1}^n \alpha_i y_i = 0 \quad (11.10)$$

By substituting \mathbf{w} in the Lagrangian, we obtain a dual optimization problem: we maximize the following over α_i

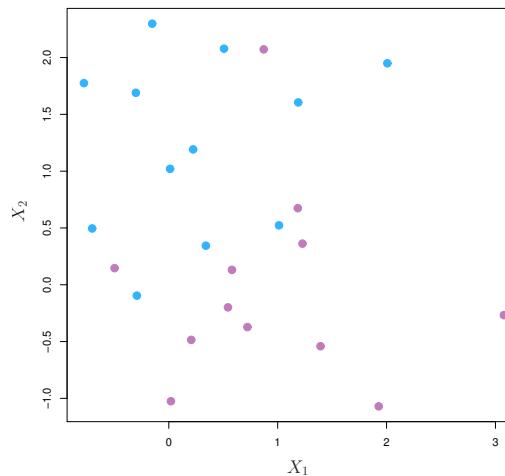
$$\mathcal{L} = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \alpha_i \quad (11.11)$$

$$\text{subject to } \sum_{i=1}^n \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad \forall i = 1, \dots, n \quad (11.12)$$

11.3 Support vector classifier (Non-separable case)

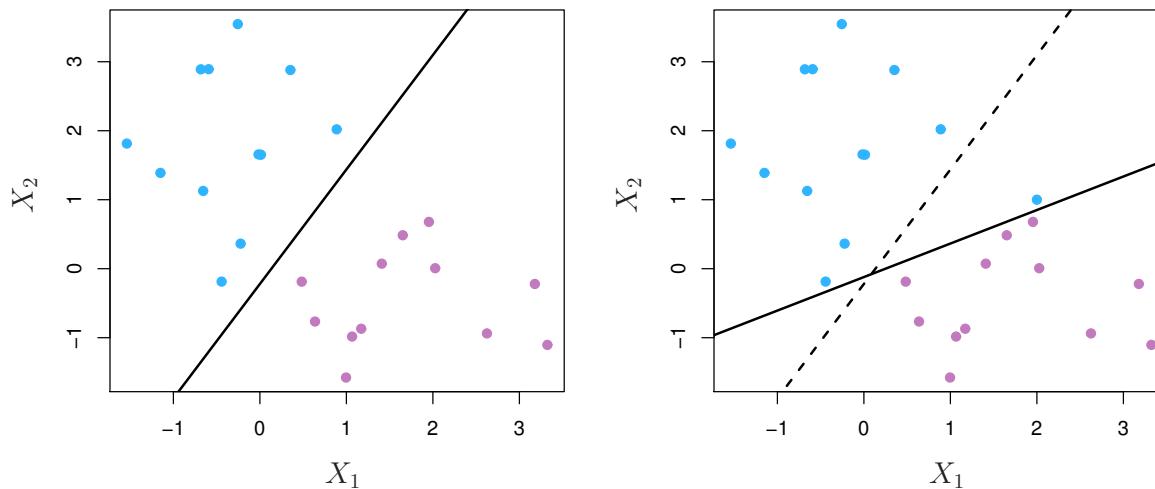
- The maximal margin classifier is a very natural way to perform classification, if a separating hyperplane exists.
- In many cases no separating hyperplane exists, and so there is no maximal margin classifier.
- See [Figure 11.4](#).
- We can extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes, using a so-called **soft margin**.
- The generalization of the maximal margin classifier to the non-separable case is known as the **support vector classifier**.

Figure 11.4: There are two classes of observations, shown in blue and in purple. In this case, the two classes are not separable by a hyperplane, and so the maximal margin classifier cannot be used.



- The maximal margin hyperplane is extremely sensitive to a change in a single observation suggests that it may have overfit the training data.
- We may want
 - Greater robustness to individual observations, and
 - Better classification of most of the training observations.
- Worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations.

Figure 11.5: Left: Two classes of observations are shown in blue and in purple, along with the maximal margin hyperplane. Right: An additional blue observation has been added, leading to a dramatic shift in the maximal margin hyperplane shown as a solid line. The dashed line indicates the maximal margin hyperplane that was obtained in the absence of this additional point.



- In a non-separable case,

$$\max_{\beta_0, \dots, \beta_p} M \quad (11.13)$$

$$\text{subject to } \sum_{j=1}^p \beta_j^2 = 1 \quad (11.14)$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \quad (11.15)$$

$$\epsilon_i \geq 0, \sum_{i=1}^n \epsilon_i \leq C \quad (11.16)$$

This optimization problem is equivalent to

$$\text{Minimize } \mathcal{L} = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C' \sum_{i=1}^n \epsilon_i \quad (11.17)$$

$$\text{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \epsilon_i, \text{ and } \epsilon_i \geq 0, \quad \forall i = 1, \dots, n. \quad (11.18)$$

The dual optimization problem is:

$$\text{Maximize } \mathcal{L} = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad (11.19)$$

$$\text{subject to } \sum_{i=1}^n \alpha_i y_i = 0, \quad C' \geq \alpha_i \geq 0, \quad \forall i = 1, \dots, n. \quad (11.20)$$

- where ϵ_i 's are called **slack variables** and
- C is a nonnegative tuning parameter, considered as a **budget** for the amount that the margin can be violated by the n observations.
- In practice, C is treated as a tuning parameter that is generally chosen via cross-validation.
- When C is small, we seek narrow margins that are rarely violated; this amounts to a classifier that is highly fit to the data, which may have low bias but high variance.

- On the other hand, when C is larger, the margin is wider and we allow more violations to it; this amounts to fitting the data less hard and obtaining a classifier that is potentially more biased but may have lower variance.
- Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as **support vectors**. These observations do affect the support vector classifier. M is the width of the margin
- If $\epsilon_i = 0$ then the i th observation is on the correct side of the margin.
- If $\epsilon_i > 0$ then the i th observation is on the wrong side of the margin, and we say that the i th observation has violated the margin.
- If $\epsilon_i > 1$ then the i th observation is on the wrong side of the hyperplane.

Figure 11.6: Left: A support vector classifier was fit to a small data set. The hyperplane is shown as a solid line and the margins are shown as dashed lines. Purple observations: Observations 3,4,5, and 6 are on the correct side of the margin, observation 2 is on the margin, and observation 1 is on the wrong side of the margin. Blue observations: Observations 7 and 10 are on the correct side of the margin, observation 9 is on the margin, and observation 8 is on the wrong side of the margin. No observations are on the wrong side of the hyperplane. Right: Same as left panel with two additional points, 11 and 12. These two observations are on the wrong side of the hyperplane and the wrong side of the margin.

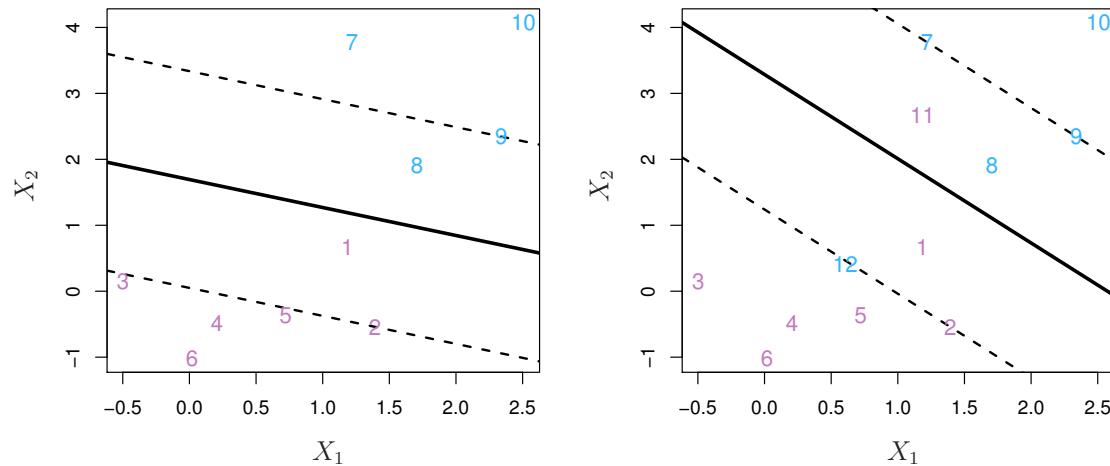
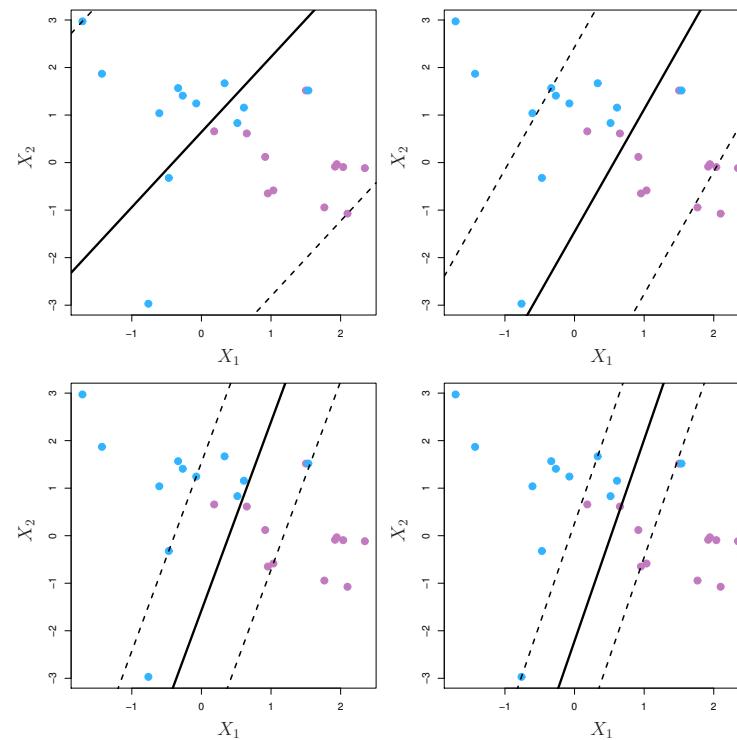


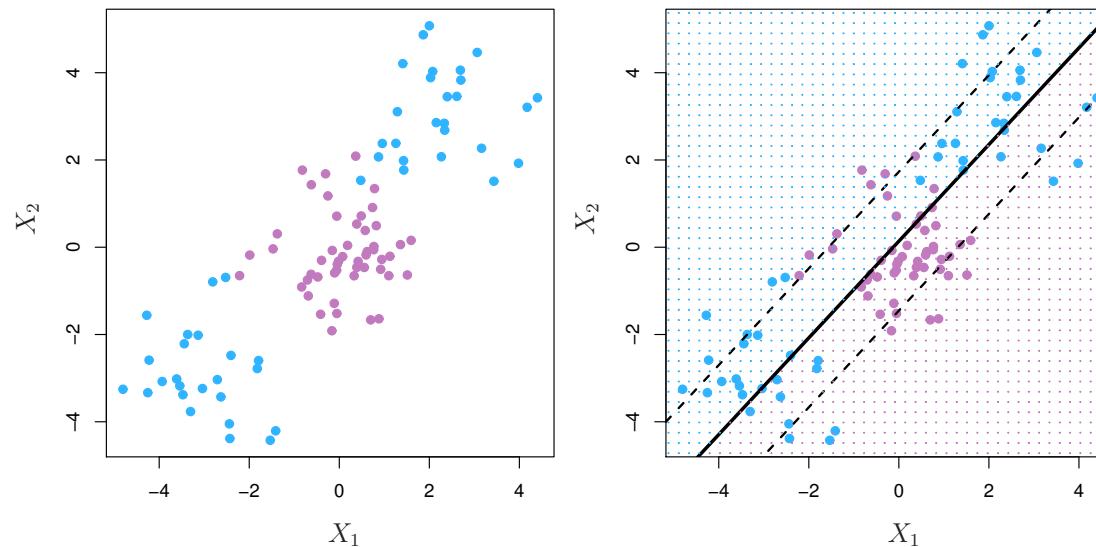
Figure 11.7: A support vector classifier was fit using four different values of the tuning parameter C in (9.12)-(9.15). The largest value of C was used in the top left panel, and smaller values were used in the top right, bottom left, and bottom right panels. When C is large, then there is a high tolerance for observations being on the wrong side of the margin, and so the margin will be large. As C decreases, the tolerance for observations being on the wrong side of the margin decreases, and the margin narrows.



11.4 Support vector machines (Nonlinear cases)

In practice we are sometimes faced with **non-linear class boundaries** and a support vector classifier or any linear classifier will **perform poorly** here.

Figure 11.8: Left: The observations fall into two classes, with a non-linear boundary between them. Right: The support vector classifier seeks a linear boundary, and consequently performs very poorly.



- For a nonlinear boundary problem, we may try to include quadratic terms in the support vector classifier:

$$\begin{aligned}
 & \text{Maximize}_{\beta_0, \beta_{11}, \beta_{12}, \dots, \beta_{p1}, \beta_{p2}, \epsilon_1, \dots, \epsilon_n} M \\
 & \text{subject to } \sum_{j=1}^p \sum_{k=1}^2 \beta_{jk}^2 = 1 \\
 & y_i \left(\beta_0 + \sum_{j=1}^p \beta_{j1} x_{ij} + \sum_{j=1}^p \beta_{j2} x_{ij}^2 \right) \geq M(1 - \epsilon_i) \\
 & \epsilon_i \geq 0, \sum_{i=1}^n \epsilon_i \leq C
 \end{aligned} \tag{11.21}$$

- One might additionally want to enlarge the feature space with higher-order polynomial terms, or with interaction terms of the form.
- The **support vector machine (SVM)** is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using **kernels**.
- The support vector classifier involves only the inner products of the observations:

$$\langle x_i, x'_i \rangle = \sum_{j=1}^p x_{ij} x_{i'j} \tag{11.22}$$

since we can show that the support vector classifier can be written as

$$f(x) = \beta_0 + \sum_{i=1}^n \alpha_i \langle x, x_i \rangle$$

- It turns out that α_i is nonzero only for the support vectors in the solution, that is, if a training observation is not a support vector,

then its α_i equals zero. So if S is the collection of indices of these support points, we can rewrite any solution function of the form as

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle \quad (11.23)$$

- We replace it with a generalization of the inner product of the form

$$K(x_i, x_{i'})$$

that we refer to as a **kernel**.

- When the support vector classifier is combined with a non-linear kernel K , the resulting classifier is known as a support vector machine.

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i) \quad (11.24)$$

- Some popular non-linear kernels:

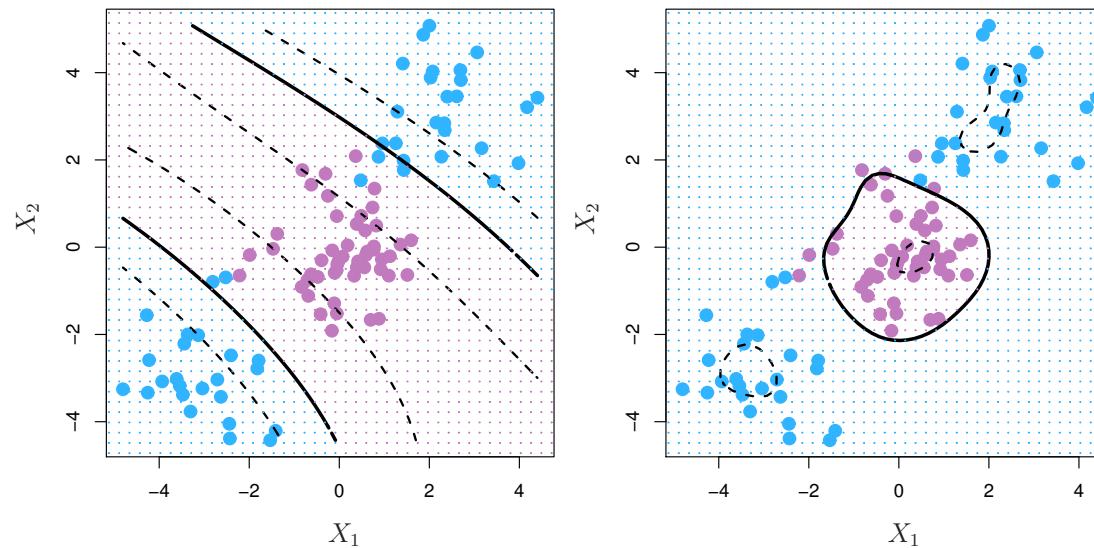
$$K(x_i, x'_i) = \left(1 + \sum_{j=1}^p x_{ij} x_{i'j} \right)^d \quad \text{polynomial kernel} \quad (11.25)$$

$$K(x_i, x'_i) = \exp \left(-\gamma \sum_{j=1}^p (x_{ij} - x_{i'j})^2 \right) \quad \text{radial kernel} \quad (11.26)$$

The package “e1071” in R contains several kernels as follow:

Kernel	Fomula	parameter
linear	$\mathbf{u}^T \mathbf{v}$	
polynomial	$\gamma(\mathbf{u}^T \mathbf{v} + c_0)^d$	γ, d, c_0
radial basis function	$\exp(-\gamma \mathbf{u} - \mathbf{v} ^2)$	γ
sigmoid	$\tanh(\gamma \mathbf{u}^T \mathbf{v} + c_0)$	γ, c_0

Figure 11.9: Left: An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule. Right: An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.



11.5 Kernel Trick

- In regression analysis, we may apply polynomials of a feature x to handle nonlinearity in the input variable x .
- Basic idea: If you can't separate positives from negatives in a low-dimensional space using a hyperplane, then map everything to a higher dimensional space where you can separate them.

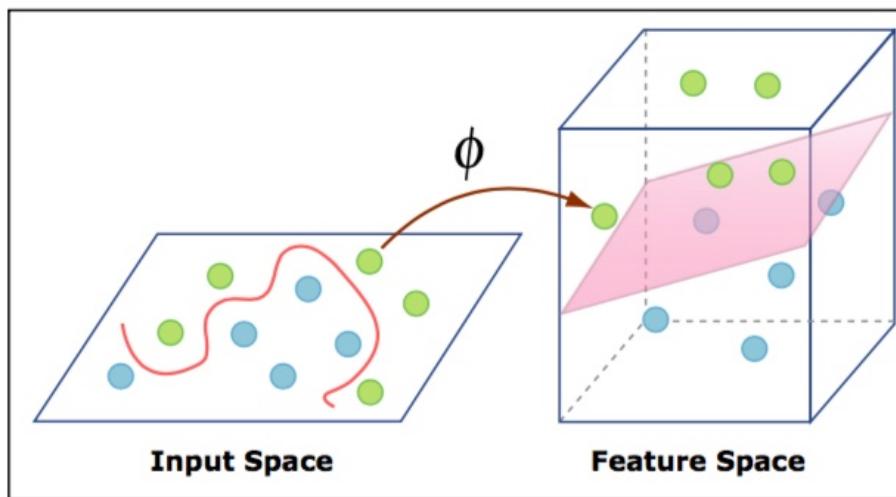


Image by MIT OpenCourseWare.

- Suppose we have a data set (x_i, y_i) : $(-4, 1), (-3, 1), (-2, 1), (-1, -1), (0, -1), (1, -1), (2, 1), (3, 1), (4, 1)$. We cannot construct a linear classifier for this data set. Map the data set by

$$\phi(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix} \quad (11.27)$$

Now we can construct a linear classifier in \mathbb{R}^2 .

- Working directly in the feature space can be costly. We have to explicitly create the feature space and operate in it. It is called the **kernel trick**.
- In the problem that define a SVM only the **inner product** of the examples is needed: This means that if we can define how to compute this product in the feature space, then it is not necessary to explicitly build it.
- We can define the kernel function as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \quad (11.28)$$

- (Example) Let $\phi(x_1, x_2) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$. The inner product of $\phi(\mathbf{x})$ and $\phi(\mathbf{z})$ is

$$k(\mathbf{x}, \mathbf{z}) = \phi^T(\mathbf{x})\phi(\mathbf{z}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2) \begin{pmatrix} 1 \\ \sqrt{2}z_1 \\ \sqrt{2}z_2 \\ z_1^2 \\ z_2^2 \\ \sqrt{2}z_1z_2 \end{pmatrix} = (1 + (x_1z_1 + x_2z_2))^2 = (\mathbf{x}^T \mathbf{z})^2 \quad (11.29)$$

- Therefore for a given $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, all we need is a semi-positive definite symmetric matrix:

$$K = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \quad (11.30)$$

11.6 SVMs with More than Two Classes

Let K be the number of classes.

- One-Versus-One (OvO) Classification
 - A one-versus-one or all-pairs approach constructs $\binom{K}{2}$ SVMs, each of which compares a pair of classes.
 - We classify a test observation using each of the $\binom{K}{2}$ classifiers.
 - We tally the number of times that the test observation is assigned to each of the K classes.
 - The final classification is performed by assigning the test observation to the class to which it was **most frequently assigned in** these $\binom{K}{2}$ pairwise classifications.
- One-Versus-All one-versus-rest (OvA or OvR) Classification
 - We fit K SVMs, each time comparing one of all the K classes to the remaining $K - 1$ classes.
 - We assign the observation to the class for which $\beta_{0k} + \beta_{1k}x_1^* + \beta_{2k}x_2^* + \cdots + \beta_{pk}x_p^*$ is largest, as this amounts to a high level of confidence that the test observation belongs to the k th class rather than to any of the other classes.

11.7 Examples

```
# Support Vector Classifier
set.seed(1)
x = matrix(rnorm(20 * 2), ncol = 2)
y = c(rep(-1, 10), rep(1, 10))
x[y == 1, ] = x[y == 1, ] + 1
# plot(x, col=(3-y))
dat = data.frame(x = x, y = as.factor(y))
library(e1071)
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
# The response must be a factor. scale=FALSE: do not standardize predictors
# plot(svmfit, dat)
svmfit$index

## [1] 1 2 5 7 14 16 17

summary(svmfit)

##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10,
##      scale = FALSE)
##
## 
## Parameters:
```

```
##      SVM-Type: C-classification
##      SVM-Kernel: linear
##      cost: 10
##      gamma: 0.5
##
## Number of Support Vectors: 7
##
## ( 4 3 )
##
##
## Number of Classes: 2
##
## Levels:
## -1 1
# svm function does not provide the estimates of the coefficients and the
# margin (too bad)
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 0.1, scale = FALSE)
# plot(svmfit, dat) the support vectors: x, the others: o
svmfit$index
## [1] 1 2 3 4 5 7 9 10 12 13 14 15 16 17 18 20
# indices of the support vectors
set.seed(1)
tune.out = tune(svm, y ~ ., data = dat, kernel = "linear", ranges = list(cost = c(0.001,
  0.01, 0.1, 1, 5, 10, 100)))
summary(tune.out)
```

```
##  
## Parameter tuning of 'svm':  
##  
## - sampling method: 10-fold cross validation  
##  
## - best parameters:  
##   cost  
##   0.1  
##  
## - best performance: 0.1  
##  
## - Detailed performance results:  
##   cost error dispersion  
## 1 1e-03  0.70  0.4216370  
## 2 1e-02  0.70  0.4216370  
## 3 1e-01  0.10  0.2108185  
## 4 1e+00  0.15  0.2415229  
## 5 5e+00  0.15  0.2415229  
## 6 1e+01  0.15  0.2415229  
## 7 1e+02  0.15  0.2415229  
  
bestmod = tune.out$best.model  
summary(bestmod)  
  
##  
## Call:  
## best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.001,
```

```
##      0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##   SVM-Type: C-classification
##   SVM-Kernel: linear
##   cost: 0.1
##   gamma: 0.5
##
## Number of Support Vectors: 16
##
## ( 8 8 )
##
##
## Number of Classes: 2
##
## Levels:
## -1 1

xtest = matrix(rnorm(20 * 2), ncol = 2)
ytest = sample(c(-1, 1), 20, rep = TRUE)
xtest[ytest == 1, ] = xtest[ytest == 1, ] + 1
testdata = data.frame(x = xtest, y = as.factor(ytest))
ypred = predict(bestmod, testdata)
table(predict = ypred, truth = testdat$y)

##       truth
```

```
## predict -1 1
##      -1 11 1
##      1   0  8

svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 0.01, scale = FALSE)
ypred = predict(svmfit, testdat)
table(predict = ypred, truth = testdat$y)

##      truth
## predict -1 1
##      -1 11 2
##      1   0  7

x[y == 1, ] = x[y == 1, ] + 0.5
# plot(x, col=(y+5)/2, pch=19)
dat = data.frame(x = x, y = as.factor(y))
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 1e+05)
summary(svmfit)

##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)
##
##
## Parameters:
##   SVM-Type: C-classification
##   SVM-Kernel: linear
```

```
##      cost: 1e+05
##      gamma: 0.5
##
## Number of Support Vectors: 3
##
## ( 1 2 )
##
## Number of Classes: 2
##
## Levels:
## -1 1

# plot(svmfit, dat)
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 1)
summary(svmfit)

##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)
##
##
## Parameters:
##      SVM-Type: C-classification
##      SVM-Kernel: linear
##      cost: 1
##      gamma: 0.5
```

```
##  
## Number of Support Vectors: 7  
##  
## ( 4 3 )  
##  
##  
## Number of Classes: 2  
##  
## Levels:  
## -1 1  
  
# plot(svmfit,dat)
```

Figure 11.10: SVM example

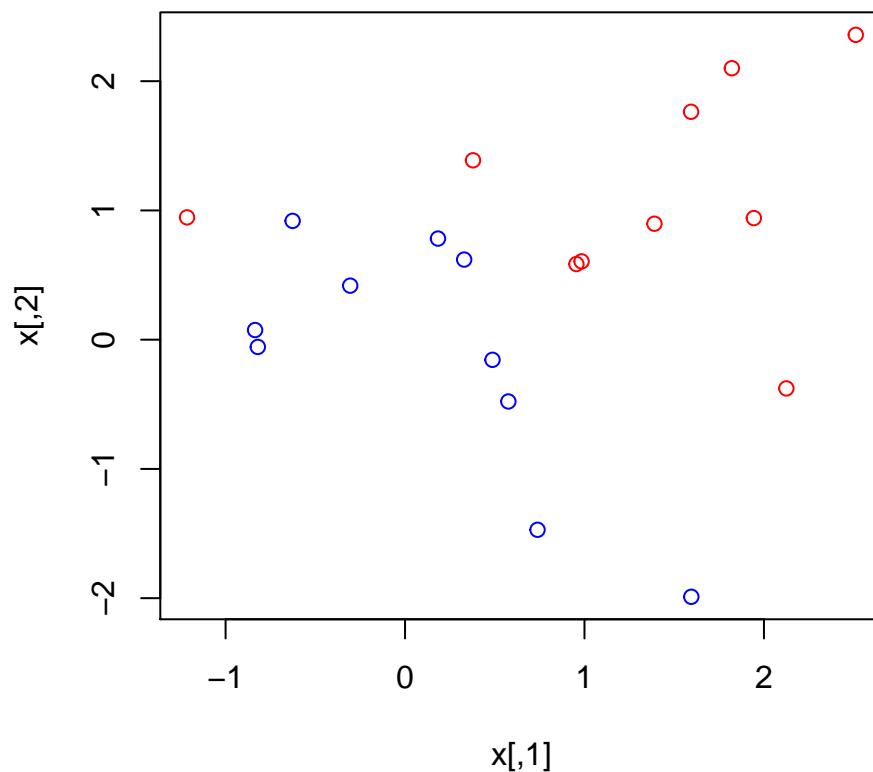
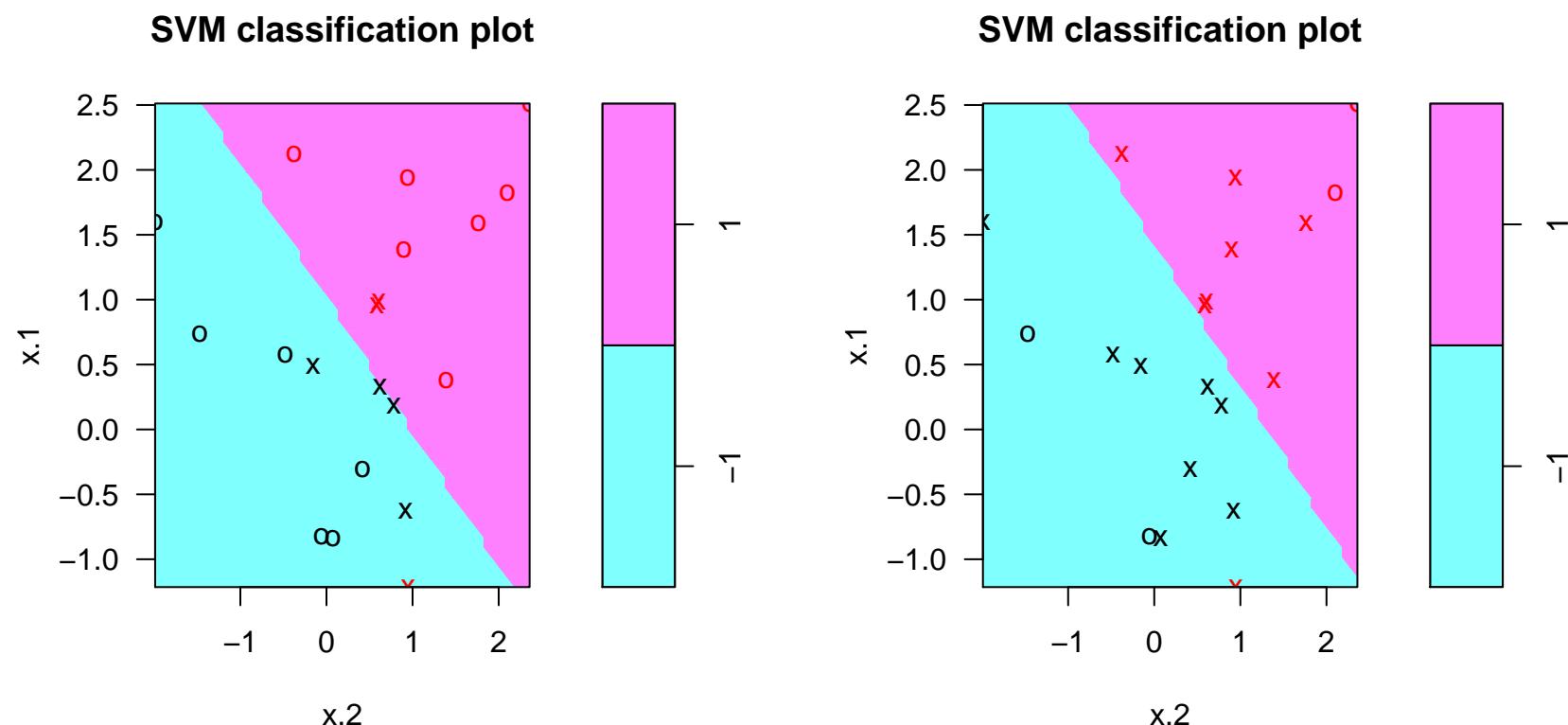


Figure 11.11: SVM example



```
# Support Vector Machine
set.seed(1)
x = matrix(rnorm(200 * 2), ncol = 2)
x[1:100, ] = x[1:100, ] + 2
x[101:150, ] = x[101:150, ] - 2
y = c(rep(1, 150), rep(2, 50))
dat = data.frame(x = x, y = as.factor(y))
# plot(x, col=y)
train = sample(200, 100)
svmfit = svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1)
# plot(svmfit, dat[train,])
summary(svmfit)

##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial",
##      gamma = 1, cost = 1)
##
##
## Parameters:
##   SVM-Type: C-classification
##   SVM-Kernel: radial
##     cost: 1
##     gamma: 1
##
## Number of Support Vectors: 37
```

```
##  
## ( 17 20 )  
##  
##  
## Number of Classes: 2  
##  
## Levels:  
## 1 2  
  
svmfit = svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1e+05)  
# plot(svmfit,dat[train,])  
set.seed(1)  
tune.out = tune(svm, y ~ ., data = dat[train, ], kernel = "radial", ranges = list(cost = c(0.1,  
 1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4)))  
summary(tune.out)  
  
##  
## Parameter tuning of 'svm':  
##  
## - sampling method: 10-fold cross validation  
##  
## - best parameters:  
##   cost gamma  
##     1      2  
##  
## - best performance: 0.12  
##
```

```
## - Detailed performance results:  
##      cost gamma error dispersion  
## 1  1e-01    0.5   0.27  0.11595018  
## 2  1e+00    0.5   0.13  0.08232726  
## 3  1e+01    0.5   0.15  0.07071068  
## 4  1e+02    0.5   0.17  0.08232726  
## 5  1e+03    0.5   0.21  0.09944289  
## 6  1e-01    1.0   0.25  0.13540064  
## 7  1e+00    1.0   0.13  0.08232726  
## 8  1e+01    1.0   0.16  0.06992059  
## 9  1e+02    1.0   0.20  0.09428090  
## 10 1e+03    1.0   0.20  0.08164966  
## 11 1e-01    2.0   0.25  0.12692955  
## 12 1e+00    2.0   0.12  0.09189366  
## 13 1e+01    2.0   0.17  0.09486833  
## 14 1e+02    2.0   0.19  0.09944289  
## 15 1e+03    2.0   0.20  0.09428090  
## 16 1e-01    3.0   0.27  0.11595018  
## 17 1e+00    3.0   0.13  0.09486833  
## 18 1e+01    3.0   0.18  0.10327956  
## 19 1e+02    3.0   0.21  0.08755950  
## 20 1e+03    3.0   0.22  0.10327956  
## 21 1e-01    4.0   0.27  0.11595018  
## 22 1e+00    4.0   0.15  0.10801234  
## 23 1e+01    4.0   0.18  0.11352924  
## 24 1e+02    4.0   0.21  0.08755950
```

```
## 25 1e+03 4.0 0.24 0.10749677







##      pred
## true 1 2
##   1 56 21
##   2 18 5
```

Example 11.7.1

Iris data

```
# SVM example with Iris Data in R Use library e1071, you can install it
# using install.packages("e1071"). Load library
library("e1071")
# Using Iris data
head(iris, 5)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5         1.4         0.2  setosa
## 2          4.9         3.0         1.4         0.2  setosa
## 3          4.7         3.2         1.3         0.2  setosa
## 4          4.6         3.1         1.5         0.2  setosa
## 5          5.0         3.6         1.4         0.2  setosa

# Attach the Data
```

```
attach(iris)
# Divide Iris data to x (contain the all features) and y only the classes
x <- subset(iris, select = -Species)
y <- Species
# Create SVM Model and show summary
svm_model <- svm(Species ~ ., data = iris)
summary(svm_model)

##
## Call:
## svm(formula = Species ~ ., data = iris)
##
##
## Parameters:
##   SVM-Type: C-classification
##   SVM-Kernel: radial
##   cost: 1
##   gamma: 0.25
##
## Number of Support Vectors: 51
##
## ( 8 22 21 )
##
##
## Number of Classes: 3
##
```

```
## Levels:  
##  setosa versicolor virginica  
  
svm_model1 <- svm(x, y)  
summary(svm_model1)  
  
##  
## Call:  
## svm.default(x = x, y = y)  
##  
##  
## Parameters:  
##   SVM-Type: C-classification  
##   SVM-Kernel: radial  
##   cost: 1  
##   gamma: 0.25  
##  
## Number of Support Vectors: 51  
##  
##  ( 8 22 21 )  
##  
##  
## Number of Classes: 3  
##  
## Levels:  
##  setosa versicolor virginica
```

```
pred <- predict(svm_model1, x)
system.time(pred <- predict(svm_model1, x))

##      user    system elapsed
## 0.001   0.000   0.001

table(pred, y)

##          y
## pred      setosa versicolor virginica
## setosa      50       0       0
## versicolor   0      48       2
## virginica    0       2      48

svm_tune <- tune(svm, train.x = x, train.y = y, kernel = "radial", ranges = list(cost = 10^{(-1:2)},
gamma = c(0.5, 1, 2)))

print(svm_tune)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost gamma
##     1     0.5
##
## - best performance: 0.02666667
```

```
svm_model_after_tune <- svm(Species ~ ., data = iris, kernel = "radial", cost = 1,
    gamma = 0.5)
summary(svm_model_after_tune)

##
## Call:
## svm(formula = Species ~ ., data = iris, kernel = "radial", cost = 1,
##       gamma = 0.5)
##
##
## Parameters:
##   SVM-Type: C-classification
##   SVM-Kernel: radial
##     cost: 1
##     gamma: 0.5
##
## Number of Support Vectors: 59
##
## ( 11 23 25 )
##
##
## Number of Classes: 3
##
## Levels:
##   setosa versicolor virginica
pred <- predict(svm_model_after_tune, x)
```

```
system.time(predict(svm_model_after_tune, x))

##    user  system elapsed
##  0.001  0.000  0.001

table(pred, y)

##          y
## pred      setosa versicolor virginica
##   setosa     50       0       0
##   versicolor  0      48       2
##   virginica   0       2      48

detach(iris)
```

Part V

Unsupervised Learning

Chapter 12 Clustering

- Grouping or clustering: exploratory technique
- Provides an informal means for
 1. assessing dimensionality
 2. identifying outliers
 3. suggesting interesting hypotheses
- Difference between classification and clustering
 - (Classification) known number of groups.
The objective is to assign new observations to one of these known groups.
 - (Clustering) Unknown number of groups, unknown group structure.
The goal is to find an optimal grouping for which the observations or objects within each cluster are similar but the clusters are dissimilar to each other.
- Clustering depends on how to define “similarity” between objects (items or variables).

- Data can be written as a $n \times p$ matrix:

$$\mathbf{Y} = \begin{pmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_n^T \end{pmatrix} = (\mathbf{y}_{(1)}, \dots, \mathbf{y}_{(p)}) \quad (12.1)$$

We generally wish to group the $n \mathbf{y}^T$ s (rows) into g clusters. We may also wish to cluster the columns $\mathbf{y}_{(j)}, j = 1, 2, \dots, p$.

- Clustering methods
 - Hierarchical methods:
 1. Agglomerative approach: we typically start with n clusters, one for each observation, and end with a single cluster containing all n observations. At each step, an observation or a cluster of observations is absorbed into another cluster.
 2. Divisive approach: We can also reverse this process, that is, start with a single cluster containing all n observations and end with n clusters of a single item each.
 - Partitioning methods: We simply divide the observations into g clusters. This can be done by starting with an initial partitioning or with cluster centers and then reallocating the observations according to some optimality criterion.
 - Mixture models

12.1 Similarity or Dissimilarity

- When items (units or cases) are clustered, proximity is indicated by distance.
- When variables are clustered, proximity is indicated by correlation coefficients or measures of association.

A. Distances and similarity coefficients for pairs of items

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})'(\mathbf{x} - \mathbf{y})} = \sqrt{\sum_{i=1}^p (x_i - y_i)^2} \quad \text{Euclidean distance} \quad (12.2)$$

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^p (x_i - y_i)^r \right)^{1/r} \quad \text{Minkowski distance} \quad (12.3)$$

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})' \mathbf{S}^{-1} (\mathbf{x} - \mathbf{y})} \quad \text{Mahalanobis distance} \quad (12.4)$$

Mahalanobis distance is not easy to apply for cluster analysis since we do not know about the groups.

For positive variables,

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^p \frac{|x_i - y_i|}{x_i + y_i} \quad \text{Canberra metric} \quad (12.5)$$

$$d(\mathbf{x}, \mathbf{y}) = 1 - \frac{\sum_{i=1}^p \min|x_i - y_i|}{\sum_{i=1}^p (x_i + y_i)} \quad \text{Czekanowski coefficient} \quad (12.6)$$

For binary variables,

		Variables					
		Item i	1	0	0	1	1
		Itme k	1	1	0	1	0

$$\sum_{i=1}^5 (x_{ij} - x_{kj})^2 = (1-1)^2 + (0-1)^2 + (0-0)^2 + (1-1)^2 + (1-0)^2 = 2$$

that is the number of mismatches. We can summarize the items i and k by the following table:

		Item k		Totals
		1	0	
Item i	1	a	b	$a + b$
	0	c	d	$c + d$
Totals		$a + c$	$b + d$	$p = a + b + c + d$

In this table, a represents the frequency of 1 – 1 matches, b is the frequency of 1 – 0 matches, c is the frequency of 0 – 1 matches, and d is the frequency of 0 – 0 matches. Given the foregoing five pairs of binary outcomes, $a = 2$ and $b = c = d = 1$.

Table 12.1 lists common similarity coefficients defined in terms of the frequencies in (12-7). A short rationale follows each definition.

Table 12.1: Similarity Coefficients for Clustering Items

Coefficient	Rationale
$\frac{a+d}{p}$	Equal weights for 1 - 1 matches and 0 - 0 matches.
$\frac{2(a+d)}{2(a+d)+b+c}$	Double weight for 1 - 1 matches and 0 - 0 matches.
$\frac{(a+d)}{a+d+2(b+c)}$	Double weight for unmatched pairs.
$\frac{a}{p}$	No 0 - 0 matches in numerator.

See Johnson and Wichern for more similarity coefficients.

B. Similarities and Association Measures for pairs of variables

- Similarity measures for variables often take the form of sample correlation coefficients. Moreover, in some clustering applications, negative correlations are replaced by their absolute values.
- When the variables are binary, the data can again be arranged in the form of a contingency table. This time, however, the variables, rather than the items, delineate the categories. For each pair of variables, there are n items categorized in the table. With the usual 0 and 1 coding, the table becomes as follows:

		Variable k		Totals
		1	0	
Variable i	1	a	b	$a + b$
	0	c	d	$c + d$
Totals		$a + c$	$b + d$	$n = a + b + c + d$

For instance, variable i equals 1 and variable k equals 0 for b of the n items.

- The usual product moment correlation formula applied to the binary variables in the contingency table of (12-10) gives

$$r = \frac{ad - bc}{\sqrt{(a+b)(c+d)(a+c)(b+d)}} \quad (12.7)$$

This number can be taken as a measure of the similarity between the two variables. The correlation coefficient in (12-11) is related to the chi-square statistic ($r^2 = \chi^2/n$) for testing the independence of two categorical variables. For n fixed, a large similarity (or correlation) is consistent with the presence of dependence. Given the table in (12-10), measures of association (or similarity) exactly analogous to the ones listed in Table 12.1 can be developed. The only change required is the substitution of n (the number of items) for p (the number of variables).

12.2 Hierarchical Clustering

- Agglomerative hierarchical procedures and, in particular, **linkage methods**:
 - Single linkage: minimum distance or nearest neighbor
 - Complete linkage: maximum distance or farthest neighbor
 - Average linkage: average distance

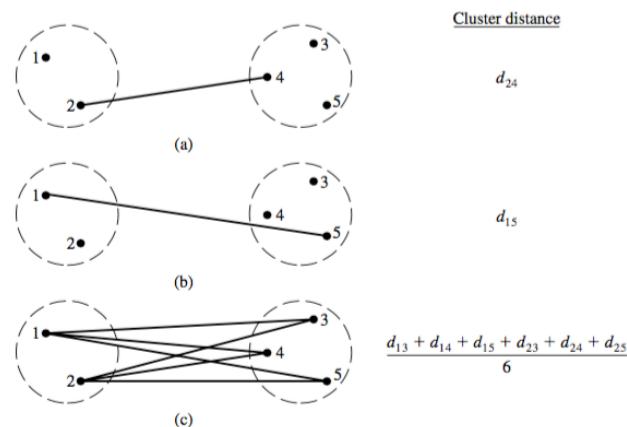


Figure 12.1: Intercluster distance (dissimilarity) for (a) single linkage, (b) complete linkage, and (c) average linkage.

- Dendrogram
 - The results of both agglomerative and divisive methods may be displayed in the form of a two-dimensional diagram.

- It illustrates the mergers or divisions that have been made at successive levels.
-

Example 12.2.1 (Single linkage)

Consider the hypothetical distances between pairs of five objects as follows:

$$\mathbf{D} = \begin{matrix} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 \\ 2 & 9 & 0 \\ 3 & 3 & 7 & 0 \\ 4 & 6 & 5 & 9 & 0 \\ 5 & 11 & 10 & \textcircled{2} & 8 & 0 \end{matrix}$$

- We merge item 3 and item 5 since $\min_{1 \leq k < i \leq 5} d_{ik} = d_{53} = 2$. By using the single linkage,

$$\begin{aligned} d_{(3,5)1} &= \min\{d_{31}, d_{51}\} = \min\{3, 11\} = 3 \\ d_{(3,5)2} &= \min\{d_{32}, d_{52}\} = \min\{7, 10\} = 7 \\ d_{(3,5)4} &= \min\{d_{34}, d_{54}\} = \min\{9, 8\} = 8 \end{aligned}$$

$$\mathbf{D} = \begin{matrix} & (3,5) & 1 & 2 & 4 \\ (3,5) & 0 \\ 1 & \textcircled{3} & 0 \\ 2 & 7 & 9 & 0 \\ 4 & 8 & 6 & 5 & 0 \end{matrix}$$

- Next we merge (3,5) and 1 since the smallest distance is 3 between (3,5) and 1.

$$d_{(1,(3,5))2} = \min\{d_{(3,5)2}, d_{12}\} = \min\{7, 9\} = 7$$

$$d_{(1,3,5)4} = \min\{d_{(3,5)4}, d_{14}\} = \min\{8, 6\} = 6$$

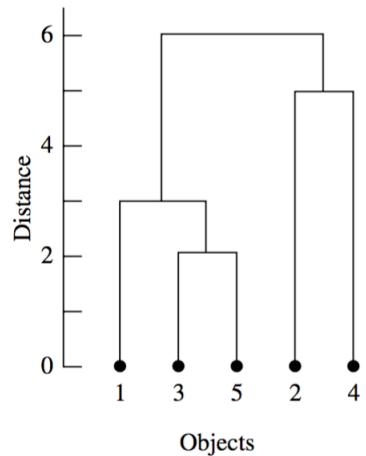
$$\mathbf{D} = \begin{matrix} & (1, 3, 5) & 2 & 4 \\ (1, 3, 5) & 0 \\ 2 & 7 & 0 \\ 4 & 6 & \textcircled{5} & 0 \end{matrix}$$

- The smallest distance is 5 between item 2 and item 4.

$$d_{(1,3,5)(2,4)} = \min\{d_{(1,3,5)2}, d_{(1,3,5)4}\} = \min\{7, 6\} = 6$$

$$\mathbf{D} = \begin{matrix} & (1, 3, 5) & (2, 4) \\ (1, 3, 5) & 0 \\ (2, 4) & 6 & 0 \end{matrix}$$

- The dendrogram displays the result of the hierarchical cluster:



Example 12.2.2 (Complete linkage)

Consider the same hypothetical distances between pairs of five objects as in the previous example:

$$\mathbf{D} = \begin{matrix} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 \\ 2 & 9 & 0 \\ 3 & 3 & 7 & 0 \\ 4 & 6 & 5 & 9 & 0 \\ 5 & 11 & 10 & (2) & 8 & 0 \end{matrix}$$

- We merge item 3 and item 5 since $\min_{1 \leq k < i \leq 5} d_{ik} = d_{53} = 2$. By using the complete linkage,

$$\begin{aligned} d_{(3,5)1} &= \max\{d_{31}, d_{51}\} = \max\{3, 11\} = 11 \\ d_{(3,5)2} &= \max\{d_{32}, d_{52}\} = \max\{7, 10\} = 10 \\ d_{(3,5)4} &= \max\{d_{34}, d_{54}\} = \max\{9, 8\} = 9 \end{aligned}$$

$$\mathbf{D} = \begin{matrix} & (3,5) & 1 & 2 & 4 \\ (3,5) & 0 \\ 1 & 11 & 0 \\ 2 & 10 & 9 & 0 \\ 4 & 9 & 6 & (5) & 0 \end{matrix}$$

- Next we merge 2 and 4 since the smallest distance is 5 between 2 and 4.

$$d_{(2,4),(3,5)} = \max\{d_{2(3,5)}, d_{4(3,5)}\} = \max\{10, 9\} = 10$$

$$d_{(2,4)1} = \max\{d_{21}, d_{41}\} = \max\{9, 6\} = 9$$

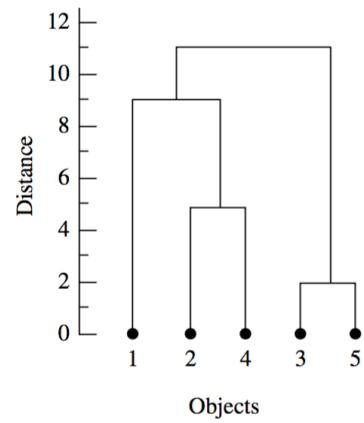
$$\mathbf{D} = \begin{matrix} & (3, 5) & (2, 4) & 1 \\ (3, 5) & 0 & & \\ (2, 4) & 10 & 0 & \\ 1 & 11 & \textcircled{9} & 0 \end{matrix}$$

- The smallest distance is 9 between item 1 and cluster (2, 4).

$$d_{(3,5)(1,2,4)} = \max\{d_{(3,5)1}, d_{(3,5)(2,4)}\} = \max\{11, 10\} = 11$$

$$\mathbf{D} = \begin{matrix} & (3, 5) & (1, 2, 4) \\ (3, 5) & 0 & \\ (1, 2, 4) & 11 & 0 \end{matrix}$$

- The dendrogram displays the result of the hierarchical cluster:



Example 12.2.3 (Average linkage)

Consider the same hypothetical distances between pairs of five objects as in the previous example:

$$\mathbf{D} = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{matrix} 0 \\ 9 & 0 \\ 3 & 7 & 0 \\ 6 & 5 & 9 & 0 \\ 11 & 10 & 2 & 8 & 0 \end{matrix} \end{matrix}$$

- We merge item 3 and item 5 since $\min_{1 \leq k < i \leq 5} d_{ik} = d_{53} = 2$. By using the complete linkage,

$$d_{(3,5)1} = \text{ave}\{d_{31}, d_{51}\} = \text{ave}\{3, 11\} = \frac{3 + 11}{2} = 7$$

$$d_{(3,5)2} = \text{ave}\{d_{32}, d_{52}\} = \text{ave}\{7, 10\} = 8.5$$

$$d_{(3,5)4} = \text{ave}\{d_{34}, d_{54}\} = \text{ave}\{9, 8\} = 8.5$$

$$\mathbf{D} = \begin{matrix} & \begin{matrix} (3,5) & 1 & 2 & 4 \end{matrix} \\ \begin{matrix} (3,5) \\ 1 \\ 2 \\ 4 \end{matrix} & \begin{matrix} 0 \\ 7 & 0 \\ 8.5 & 9 & 0 \\ 8.5 & 6 & 5 & 0 \end{matrix} \end{matrix}$$

- Next we merge 2 and 4 since the smallest distance is 5 between 2 and 4.

$$d_{(2,4),(3,5)} = \frac{1}{2} \frac{1}{2} (d_{2,3} + d_{2,5} + d_{4,3} + d_{4,5}) = 8.5$$

$$d_{(2,4)1} = \text{ave}\{d_{21}, d_{41}\} = \text{ave}\{9, 6\} = 7.5$$

$$\mathbf{D} = \begin{matrix} & (3, 5) & (2, 4) & 1 \\ (3, 5) & 0 & & \\ (2, 4) & 8.5 & 0 & \\ 1 & 7 & 7.5 & 0 \end{matrix}$$

- The smallest distance is 7 between item 1 and cluster (3, 5).

$$d_{(1,3,5)(2,4)} = \frac{1}{3} \frac{1}{2} (d_{1,2} + d_{1,4} + d_{3,2} + d_{3,4} + d_{5,2} + d_{5,4}) = 8.1667$$

$$\mathbf{D} = \begin{matrix} & (1, 3, 5) & (2, 4) \\ (1, 3, 5) & 0 & \\ (2, 4) & 8.1667 & 0 \end{matrix}$$

- Draw the dendrogram for the result using the average linkage (DIY).
-

Example 12.2.4

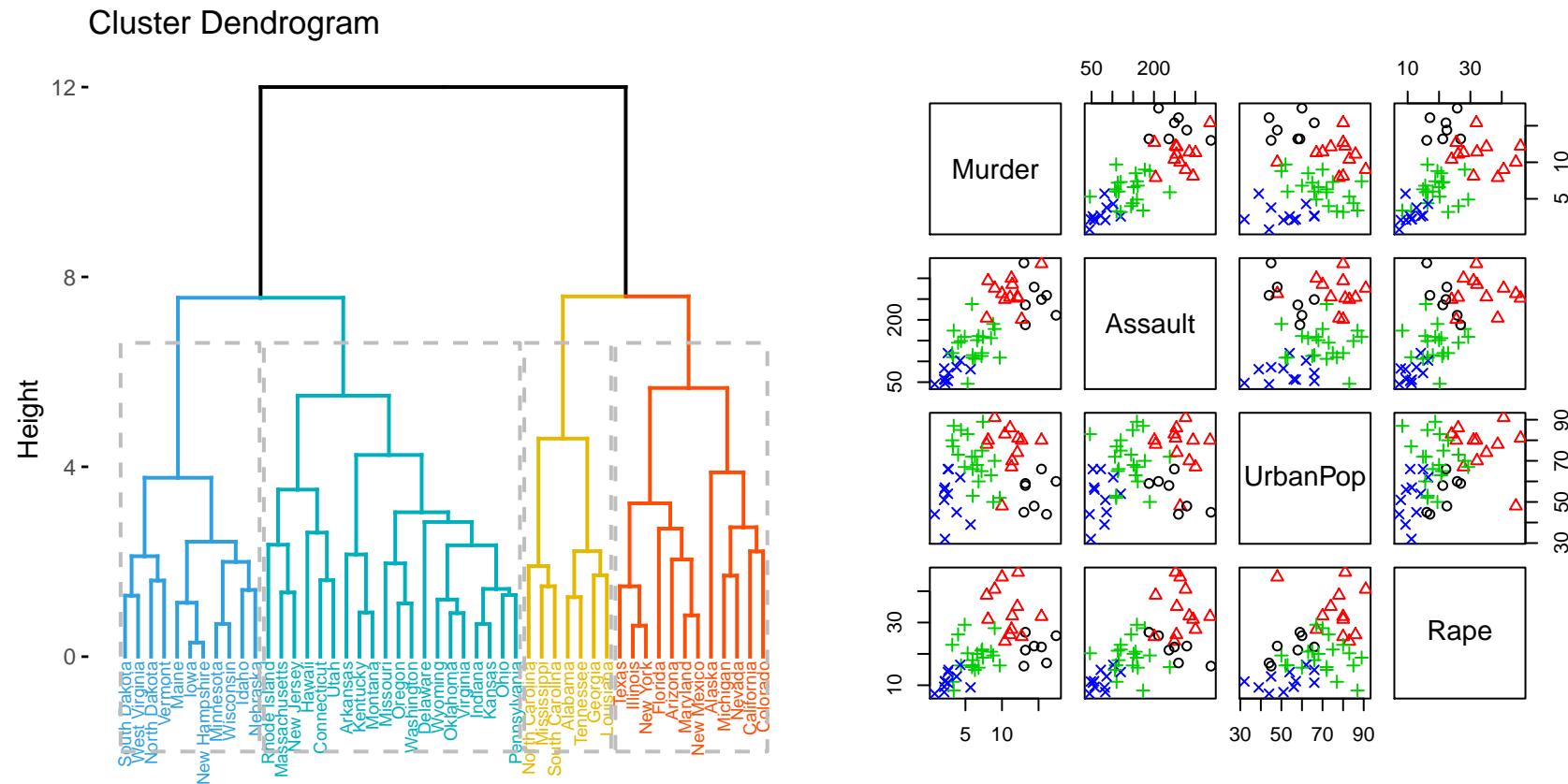
(Hierarchical cluster analysis for USArests data)

```
# Select a distance between two subjects of the standardized USArests data:  
USArrests.dist <- dist(scale(USArrests), method = "manhattan")  
# USArrests.dist Select a linkage (distance between clusters) and conduct a  
# hierarchical clustering  
USArrests.hclust <- hclust(USArrests.dist, method = "complete")  
# Names of the results from hclust function  
names(USArrests.hclust)  
  
## [1] "merge"      "height"      "order"       "labels"      "method"  
## [6] "call"        "dist.method"  
  
USArrests.hclust$height  
  
## [1]  0.2962212  0.6521762  0.6886832  0.6888920  0.8692959  0.9163901  
## [7]  0.9292610  1.1201115  1.1351339  1.2020648  1.2536151  1.2844172  
## [13] 1.2988139  1.3520315  1.4045441  1.4254959  1.4818912  1.4831706  
## [19] 1.6028418  1.6100349  1.7062940  1.7140326  1.9079477  1.9647375  
## [25] 1.9959876  2.0464600  2.1159477  2.1530415  2.2174767  2.2224641  
## [31] 2.3460837  2.3575694  2.4218924  2.6132227  2.6971651  2.7253933  
## [37] 2.8371718  3.0420027  3.2306167  3.5189932  3.7685404  3.8819196  
## [43] 4.2484371  4.5940409  5.4985231  5.6597446  7.5614200  7.5901116  
## [49] 12.0006126  
  
# Plot Dendrogram plot(USArrests.hclust) Cut in 4 groups and color by groups  
library(factoextra)
```

```
## Welcome! Related Books: 'Practical Guide To Cluster Analysis in R' at https://goo.gl/13EFCZ

fviz_dend(USArrests.hclust, k = 4, cex = 0.5, k_colors = c("#2E9FDF", "#00AFBB",
  "#E7B800", "#FC4E07"), color_labels_by_k = TRUE, rect = TRUE)
# clustering classes when k=4
hc.class <- cutree(USArrests.hclust, k = 4)
# Scatter plot matrix with the results of hierarchical clustering For more
# details, type ?dist ?hclust ?cutree fviz_dend(USArrests.hclust, k = 4, cex
# = 0.5, k_colors = c('#2E9FDF', '#00AFBB', '#E7B800', '#FC4E07'),
# color_labels_by_k = TRUE, rect = TRUE) plot(USArrests, col = hc.class, pch
# = hc.class)
```

Figure 12.2: Dendrogram of hierarchical clustering of USArests data with Manhattan distance and complete linkage. Left: The number of clusters is set as 4, Right: Scatter plot matrix with cluster groups



12.3 Partitioning

12.3.1 K -means clustering

- MacQueen (1967) suggests the term K -means for describing an algorithm of his that assigns each item to the cluster having the nearest centroid (mean).
- (Procedure) In its simplest version, the process is composed of these three steps:
 1. Partition the items into K initial clusters.
 2. Proceed through the list of items, assigning an item to the cluster whose centroid (mean) is nearest. (Distance is usually computed using Euclidean distance with either standardized or unstandardized observations.) Recalculate the centroid for the cluster receiving the new item and for the cluster losing the item.
 3. Repeat Step 2 until no more reassessments take place.

Example 12.3.1 (Clustering using the K-means method)

Suppose we measure two variables X_1 and X_2 for each of four items A, B, C, and D.

- The data are given in the following table:

Item	Observations	
	x_1	x_2
A	5	3
B	-1	1
C	1	-2
D	-3	-2

- The objective is to divide these items into $K = 2$ clusters such that the items within a cluster are closer to one another than they are to the items in different clusters. To implement the $K = 2$ -means method, we arbitrarily partition the items into two clusters, such as (AB) and (CD) , and compute the coordinates (\bar{x}_1, \bar{x}_2) of the cluster centroid (mean).
- Thus, at Step 1, we have

Cluster	Coordinates of centroid	
	\bar{x}_1	\bar{x}_2
(AB)	$\frac{5+(-1)}{2} = 2$	$\frac{3+1}{2} = 2$
(CD)	$\frac{1+(-3)}{2} = -1$	$\frac{-2+(-2)}{2} = -2$

- At Step 2, we compute the Euclidean distance of each item from the group centroids and reassign each item to the nearest group. If an item is moved from the initial configuration, the cluster centroids (means) must be updated before proceeding. The i th coordinate, $i = 1, 2, \dots, p$, of the centroid is easily updated using the formulas:

$$\bar{x}_{i,new} = \begin{cases} \frac{n\bar{x}_i + x_{ji}}{n+1} & \text{if the } j\text{th item is added to a group} \\ \frac{n\bar{x}_i - x_{ji}}{n-1} & \text{if the } j\text{th item is removed from a group n-1} \end{cases}$$

Here n is the number of items in the “old” group with centroid $\bar{x}' = (\bar{x}_1, \dots, \bar{x}_p)$.

- Consider the initial clusters (AB) and (CD). The coordinates of the centroids are (2, 2) and (-1, -2) respectively. Suppose item A with coordinates (5, 3) is moved to the (CD) group. The new groups are (B) and (ACD) with updated centroids:

$$\begin{aligned} \text{Group } (B) \quad \bar{x}_{1,new} &= \frac{2(2) - 5}{2 - 1} = -1 \quad \bar{x}_{2,new} = \frac{2(2) - 3}{2 - 1} = 1, \quad \text{the coordinates of } B \\ \text{Group } (ACD) \quad \bar{x}_{1,new} &= \frac{2(-1) + 5}{2 + 1} = 1 \quad \bar{x}_{2,new} = \frac{2(-2) + 3}{2 + 1} = -.33 \end{aligned}$$

- Returning to the initial groupings in Step 1, we compute the squared distances

$$\begin{aligned} d^2(A, (AB)) &= (5 - 2)^2 + (3 - 2)^2 = 10 && \text{if A is not moved} \\ d^2(A, (CD)) &= (5 + 1)^2 + (3 + 2)^2 = 61 \\ d^2(A, (B)) &= (5 + 1)^2 + (3 - 1)^2 = 40 && \text{if A is moved to the (CD) group} \\ d^2(A, (ACD)) &= (5 - 1)^2 + (3 + .33)^2 = 27.09 \end{aligned}$$

- Since A is closer to the center of (AB) than it is to the center of (ACD), it is not reassigned. Continuing, we consider reassigning B. We get

$$\begin{aligned} d^2(B, (AB)) &= (-1 - 2)^2 + (1 - 2)^2 = 10 && \text{if B is not moved} \\ d^2(B, (CD)) &= (-1 + 1)^2 + (1 + 2)^2 = 9 \\ d^2(B, (A)) &= (-1 - 5)^2 + (1 - 3)^2 = 40 && \text{if B is moved to the (CD) group} \\ d^2(B, (BCD)) &= (-1 + 1)^2 + (1 + 1)^2 = 4 \end{aligned}$$

- Since B is closer to the center of (BCD) than it is to the center of (AB), B is reassigned to the (CD) group. We now have the clusters (A) and (BCD) with centroid coordinates (5, 3) and (-1, -1) respectively. We check C for reassignment.

$$\begin{aligned} d^2(C, (A)) &= (1 - 5)^2 + (-2 - 3)^2 = 41 && \text{if C is not moved} \\ d^2(C, (BCD)) &= (1 + 1)^2 + (-2 + 1)^2 = 5 \\ d^2(C, (AC)) &= (1 - 3)^2 + (-2 - .5)^2 = 10.25 && \text{if C is moved to the (A) group} \\ d^2(C, (BD)) &= (1 + 2)^2 + (-2 + .5)^2 = 11.25 \end{aligned}$$

- Since C is closer to the center of the BCD group than it is to the center of the AC group, C is not moved. Continuing in this way, we find that no more reassessments take place and the final $K = 2$ clusters are (A) and (BCD).
- For the final clusters, we have

Cluster	Squared distances to group centroids			
	Item			
	A	B	C	D
A	0	40	41	89
(BCD)	52	4	5	5

- The within cluster sum of squares (sum of squared distances to centroid) are

$$\text{Cluster } A : 0$$

$$\text{Cluster}(BCD) : 4 + 5 + 5 = 14$$

Equivalently, we can determine the $K = 2$ clusters by using the criterion

$$\min E = \sum d_{i,c(i)}^2$$

where the minimum is over the number of $K = 2$ clusters and $d_{i,c(i)}^2$ is the squared distance of case i from the centroid (mean) of the assigned cluster.

- In this example, there are seven possibilities for $K = 2$ clusters:

$$A, (BCD)$$

$$B, (ACD)$$

$C, (ABD)$
 $D, (ABC)$
 $(AB), (CD)$
 $(AC), (BD)$
 $(AD), (BC)$

For the $A, (BCD)$ pair:

$$\begin{array}{ll} A & d_{A,c(A)}^2 = 0 \\ (BCD) & d_{B,c(B)}^2 + d_{C,c(C)}^2 + d_{D,c(D)}^2 = 4 + 5 + 5 = 14 \end{array}$$

Consequently, $\sum d_{i,c(i)}^2 = 0 + 14 = 14$. For the remaining pairs, you may verify that

Clusters	$\sum d_{i,c(i)}^2$	
A, (BCD)	14	← smallest
B, (ACD)	48.7	
C, (ABD)	27.7	
D, (ABC)	31.3	
(AB), (CD)	28	
(AC), (BD)	27	
(AD), (BC)	51.3	

Since the smallest $\sum d_{i,c(i)}^2$ occurs for the pair of clusters (A) and (BCD), this is the final partition.

- To check the stability of the clustering, it is desirable to rerun the algorithm with a new initial partition. Once clusters are determined, intuitions concerning their interpretations are aided by rearranging the list of items so that those in the first cluster appear first,

those in the second cluster appear next, and so forth. A table of the cluster centroids (means) and within-cluster variances also helps to delineate group differences.

Example 12.3.2

(k-means cluster analysis for USArrests data)

```
m = 20
set.seed(1)
wss = numeric(m)
library(doMC)
registerDoMC(20)
km <- foreach(i = 1:m) %dopar% {
  US.km <- kmeans(scale(USArrests), centers = i)
  US.km$tot.withinss
}
wss <- unlist(km)
# plot(wss,type='b',xlab='k (number of clusters)', ylab='Total within sum of
# squares')

US.km <- kmeans(scale(USArrests), centers = 4)
# plot(USArrests, col = US.km$cluster, pch = US.km$cluster)

# plot(prcomp(USArrests, center = TRUE,scale=TRUE)$x[,c(1,2)],col =
# US.km$cluster, pch = US.km$cluster)
```

Figure 12.3: k-means clustering for USArrests data: scree plot for $k = 1, \dots, 20$.

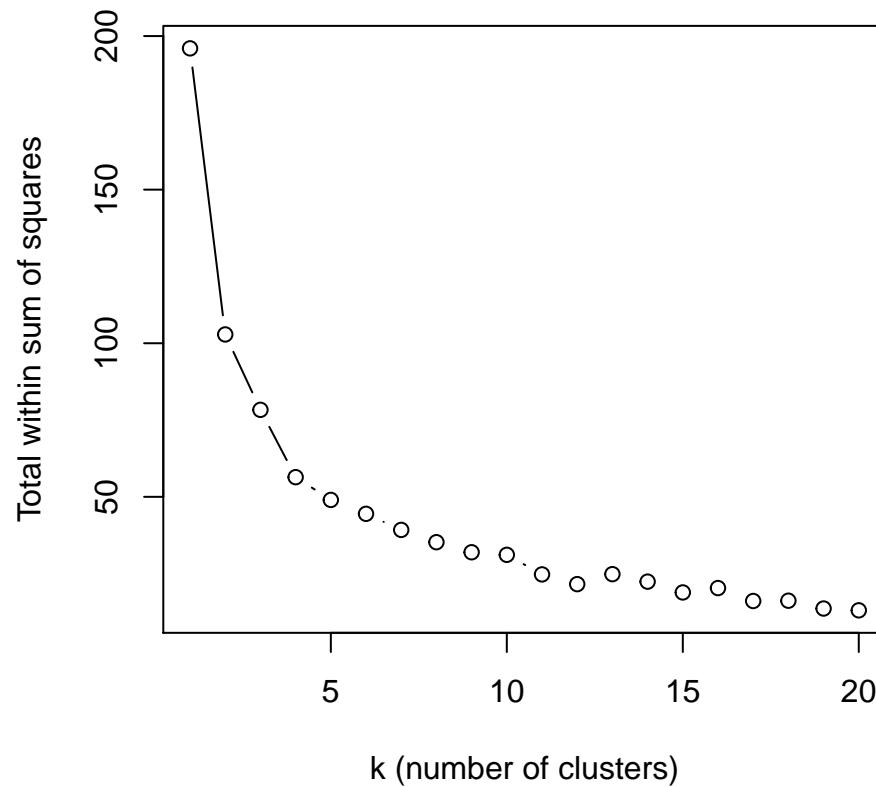


Figure 12.4: k-means clustering for USArrests data: scatter plot matrix with cluster groups.

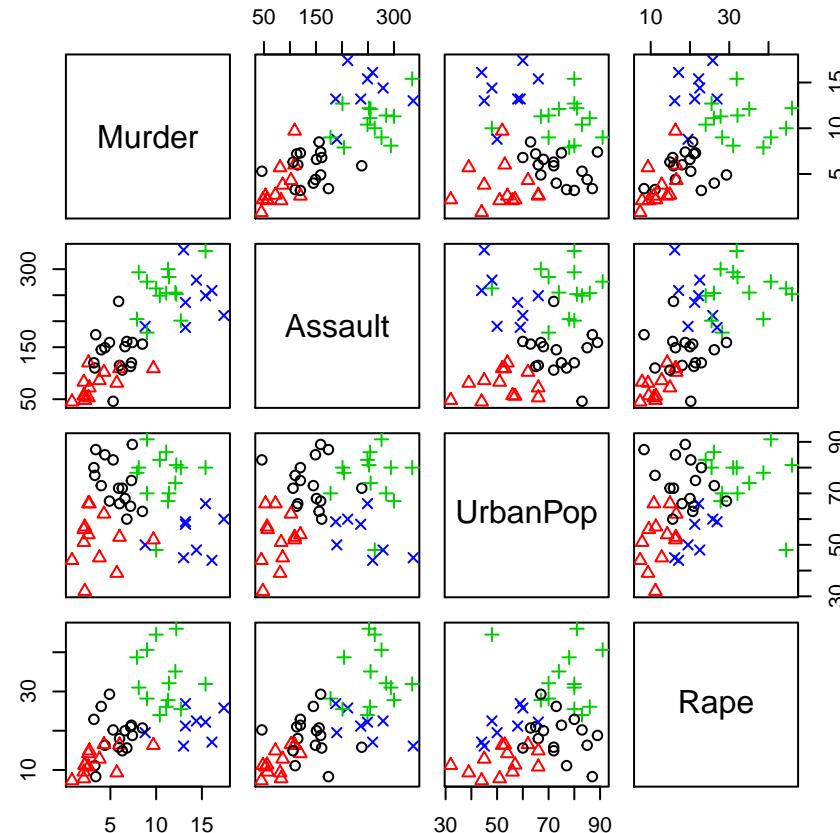
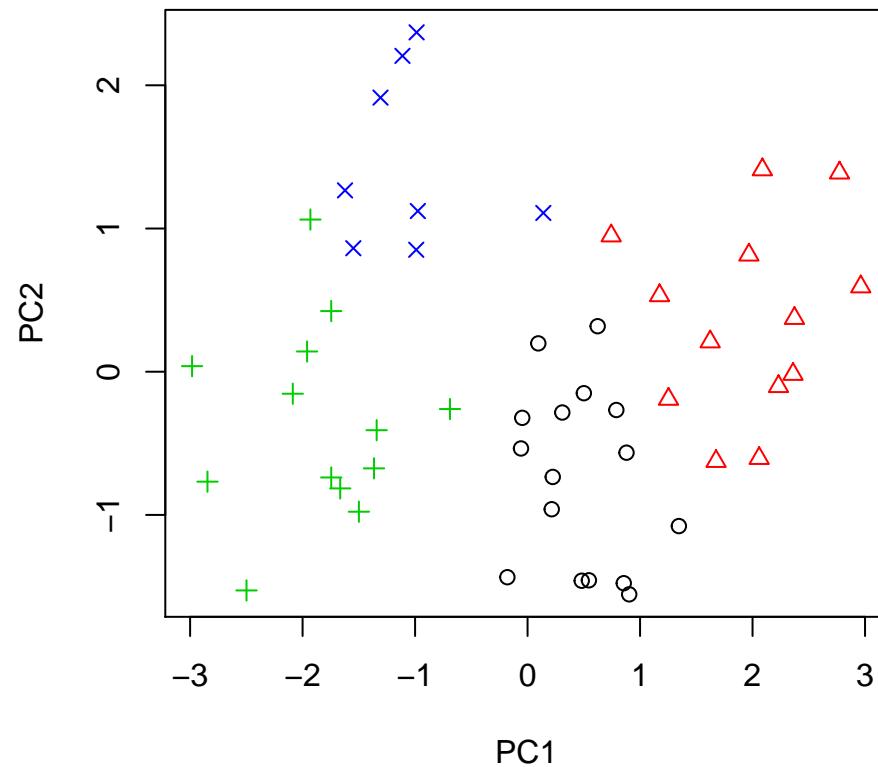


Figure 12.5: k-means clustering for USArrests data: Cluster groups in two PCs.



12.4 Determining The Optimal Number Of Clusters

- A fundamental issue in clustering analysis
- No definitive answer to this question
- The optimal number of clusters is somehow subjective and depends on the method used for measuring similarities and the parameters used for partitioning.
- These methods include direct methods and statistical testing methods:
 - Direct methods: consists of optimizing a criterion, such as the within cluster sums of squares or the average silhouette. The corresponding methods are named **elbow method** and **silhouette method**, respectively.
 - Statistical testing methods: consists of comparing evidence against null hypothesis. An example is the **gap statistic**.
- There are more than thirty other indices and methods that have been published for identifying the optimal number of clusters. There are R codes for computing all these 30 indices in order to decide the best number of clusters using the “majority rule”.

12.4.1 Elbow method

- Recall that, the basic idea behind partitioning methods, such as k-means clustering, is to define clusters such that the total intra-cluster variation (or total within-cluster sum of square WSS) is minimized. The total WSS measures the compactness of the clustering and we want it to be as small as possible.
- The Elbow method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't improve much better the total WSS .
- The optimal number of clusters can be defined as follow:
 1. Compute clustering algorithm (e.g., k-means clustering) for different values of k . For instance, by varying k from 1 to 10 clusters.
 2. For each k , calculate the total within-cluster sum of square WSS .
 3. Plot the curve of WSS according to the number of clusters k .
 4. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.
- Note that, the elbow method is sometimes ambiguous. An alternative is the average silhouette method (Kaufman and Rousseeuw, 1990) which can be also used with any clustering approach.

12.4.2 Average silhouette method

- Let $a(i)$ be the average distance from the subject i to every subject in the same cluster. Let $b(i)$ be the minimum average distance from the subject i to every subject in other clusters. If the subject i is in a cluster of a single subject, then $s(i) = 0$ without further calculation.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

- The average silhouette approach we'll be described comprehensively in the chapter cluster validation statistics. Briefly, it measures the quality of a clustering. That is, it determines how well each object lies within its cluster. A high average silhouette width indicates a good clustering.
- Average silhouette method computes the average silhouette of observations for different values of k . The optimal number of clusters k is the one that maximize the average silhouette over a range of possible values for k (Kaufman and Rousseeuw 1990).
- The algorithm is similar to the elbow method and can be computed as follow:
 1. Compute clustering algorithm (e.g., k-means clustering) for different values of k . For instance, by varying k from 1 to 10 clusters.
 2. For each k , calculate the average silhouette of observations (avg.sil).
 3. Plot the curve of avg.sil according to the number of clusters k .
 4. The location of the maximum is considered as the appropriate number of clusters.

Example 12.4.1

Use the previous example of the hierarchical clustering with average linkage. Calculate the average silhouette widths when for two clusters ($k = 2$).

Subject	1	2	3	4	5
$a(i)$	7	5	2.5	5	6.5
$b(i)$	7.5	8.67	8	7.67	9
$s(i)$	0.06667	0.4231	0.6875	0.3478	0.2778

where

$$\begin{aligned}
 a(1) &= \frac{1}{2}(d(1, 3) + d(1, 5)) = \frac{1}{2}(3 + 11) = 7 \\
 a(2) &= \frac{1}{1}(d(2, 4)) = 5 \\
 a(3) &= \frac{1}{2}(d(3, 1) + d(3, 5)) = \frac{1}{2}(3 + 2) = 2.5 \\
 a(4) &= \frac{1}{1}d(2, 4) = 5 \\
 a(5) &= \frac{1}{2}(d(5, 1) + d(5, 3)) = \frac{1}{2}(11 + 2) = 6.5 \\
 b(1) &= \frac{1}{2}(d(1, 2) + d(1, 4)) = \frac{1}{2}(9 + 6) = 7.5 \\
 b(2) &= \frac{1}{3}(d(2, 1) + d(2, 3) + d(2, 5)) = \frac{1}{3}(9 + 7 + 10) = 8.67 \\
 b(3) &= \frac{1}{2}(d(3, 2) + d(3, 4)) = \frac{1}{2}(7 + 9) = 8 \\
 b(4) &= \frac{1}{3}(d(4, 1) + d(4, 3) + d(4, 5)) = \frac{1}{3}(6 + 9 + 8) = 7.67 \\
 b(5) &= \frac{1}{2}(d(5, 2) + d(5, 4)) = \frac{1}{2}(10 + 8) = 9 \\
 s(1) &= \frac{b(1) - a(1)}{\max\{a(1), b(1)\}} = \frac{0.5}{7.5} = 1/15
 \end{aligned}$$

We can similarly calculate all $s(i)$ as in the Table. The average silhouette width for 2 clusters is

$$\text{ave.sil} = \frac{1}{5}(0.06667 + 0.4231 + 0.6875 + 0.3481 + 0.2778) = 0.360634$$

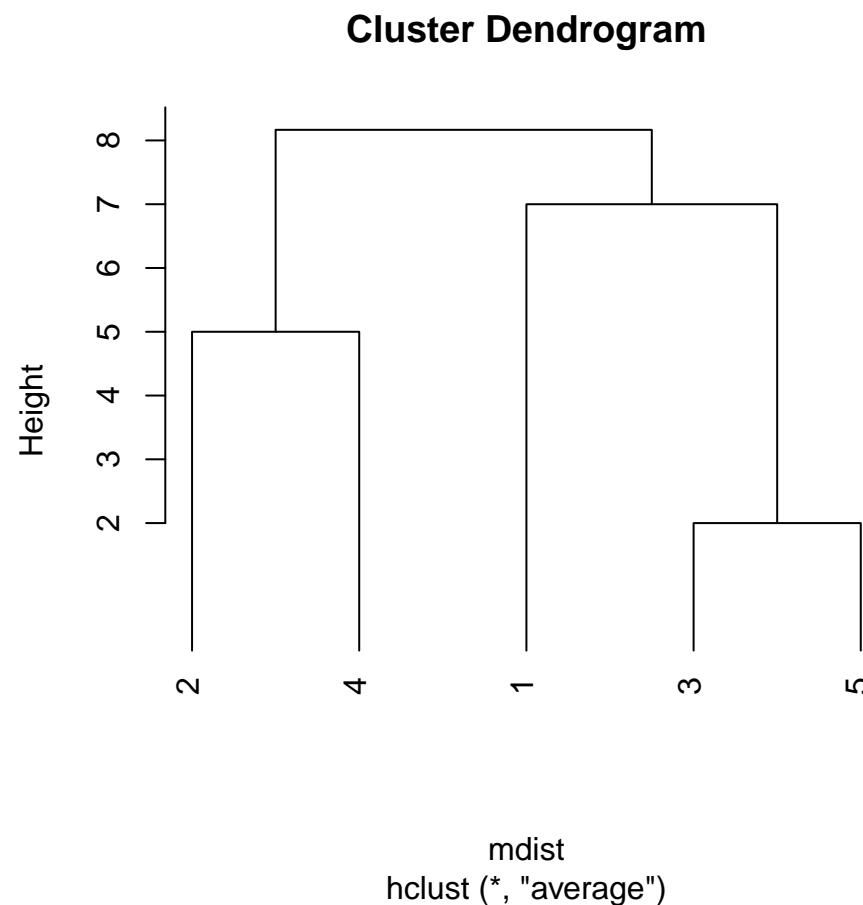
□

```
M <- matrix(c(0, 9, 3, 6, 11, 9, 0, 7, 5, 10, 3, 7, 0, 9, 2, 6, 5, 9, 0, 8,
           11, 10, 2, 8, 0), ncol = 5)
mdist <- as.dist(M)
fit <- hclust(mdist, method = "average")
plot(fit, hang = -1)
library(cluster)
sil <- silhouette(cutree(fit, 2), mdist)
sil

##      cluster neighbor   sil_width
## [1,]       1       2 0.06666667
## [2,]       2       1 0.42307692
## [3,]       1       2 0.68750000
## [4,]       2       1 0.34782609
## [5,]       1       2 0.27777778
## attr(,"Ordered")
## [1] FALSE
## attr(,"call")
## silhouette.default(x = cutree(fit, 2), dist = mdist)
## attr(,"class")
## [1] "silhouette"
```

```
mean(sil[, 3])  
## [1] 0.3605695
```

Figure 12.6: Dendrogram of the hierarchical clustering with average linkage



12.4.3 Gap statistic method

- The gap statistic has been published by R. Tibshirani, G. Walther, and T. Hastie (Stanford University, 2001). The approach can be applied to any clustering method.
- The gap statistic compares the total within intra-cluster variation for different values of k with their expected values under null reference distribution of the data. The estimate of the optimal clusters will be value that maximize the gap statistic (i.e, that yields the largest gap statistic). This means that the clustering structure is far away from the random uniform distribution of points.
- The algorithm works as follow:
 1. Cluster the observed data, varying the number of clusters from $k = 1, \dots, k_{max}$, and compute the corresponding total within intra-cluster variation W_k .
 2. Generate B reference data sets with a random uniform distribution. Cluster each of these reference data sets with varying number of clusters $k = 1, \dots, k_{max}$, and compute the corresponding total within intra-cluster variation W_{kb} .
 3. Compute the estimated gap statistic as the deviation of the observed W_k value from its expected value W_{kb} under the null hypothesis:

$$Gap(k) = \frac{1}{B} \sum_{b=1}^B \log(W_{kb}^*) - \log(W_k) \quad (12.8)$$

Compute also the standard deviation of the statistics.

- 4. Choose the number of clusters as the smallest value of k such that the gap statistic is within one standard deviation of the gap at $k + 1$: $Gap(k) \geq Gap(k + 1) - s_{k+1}$.
- Note that, using $B = 500$ gives quite precise results so that the gap plot is basically unchanged after another run.

Example 12.4.2

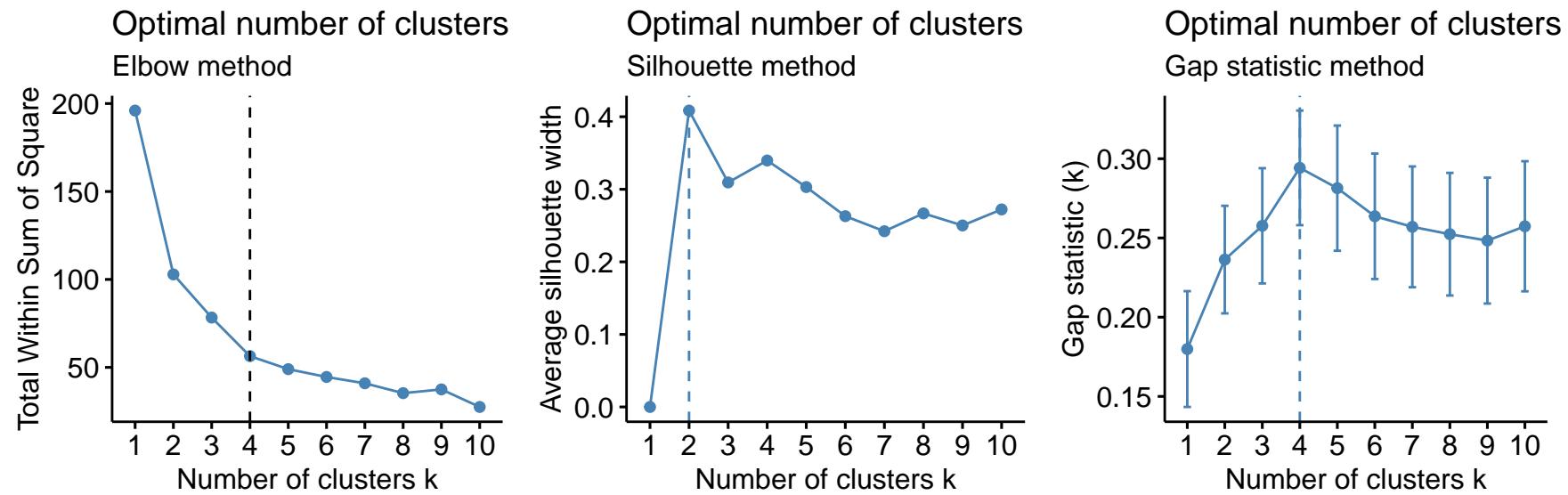
(USArrests data)

```
library(factoextra)
library(NbClust)
df <- scale(USArrests)
head(df)

##           Murder Assault UrbanPop        Rape
## Alabama    1.24256408 0.7828393 -0.5209066 -0.003416473
## Alaska     0.50786248 1.1068225 -1.2117642  2.484202941
## Arizona    0.07163341 1.4788032  0.9989801  1.042878388
## Arkansas   0.23234938 0.2308680 -1.0735927 -0.184916602
## California 0.27826823 1.2628144  1.7589234  2.067820292
## Colorado   0.02571456 0.3988593  0.8608085  1.864967207

# Elbow method
fviz_nbclust(df, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) +
  labs(subtitle = "Elbow method")
# Silhouette method
fviz_nbclust(df, kmeans, method = "silhouette") + labs(subtitle = "Silhouette method")
# Gap statistic nboot = 50 to keep the function speedy. recommended value:
# nboot= 500 for your analysis. Use verbose = FALSE to hide computing
# progression.
set.seed(123)
fviz_nbclust(df, kmeans, nstart = 25, method = "gap_stat", nboot = 50) + labs(subtitle = "Gap statistic method")
```

Figure 12.7: Results of elbow, silhouette, and gap methods to USArests data



- Elbow method: 4 clusters solution suggested
- Silhouette method: 2 clusters solution suggested
- Gap statistic method: 4 clusters solution suggested
- According to these observations, it's possible to define $k = 4$ as the optimal number of clusters in the data.

- NbClust() function: 30 indices for choosing the best number of clusters

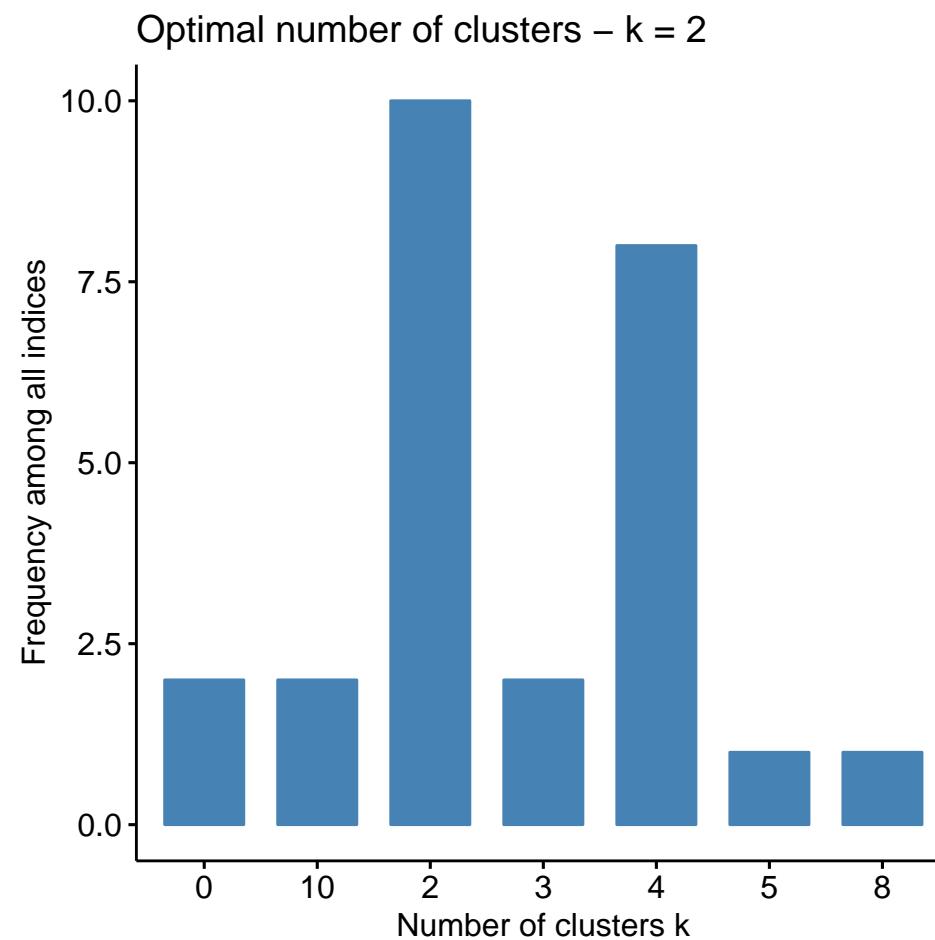
```
library("NbClust")
nb <- NbClust(df, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")

## *** : The Hubert index is a graphical method of determining the number of clusters.
##           In the plot of Hubert index, we seek a significant knee that corresponds to a
##           significant increase of the value of the measure i.e the significant peak in Hubert
##           index second differences plot.
##
## *** : The D index is a graphical method of determining the number of clusters.
##           In the plot of D index, we seek a significant knee (the significant peak in Dindex
##           second differences plot) that corresponds to a significant increase of the value of
##           the measure.
##
## *****
## * Among all indices:
## * 10 proposed 2 as the best number of clusters
## * 2 proposed 3 as the best number of clusters
## * 8 proposed 4 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
##
##           ***** Conclusion *****
##
```

```
## * According to the majority rule, the best number of clusters is 2
##
##
## ****
library("factoextra")
nbc <- fviz_nbclust(nb)

## Among all indices:
## =====
## * 2 proposed 0 as the best number of clusters
## * 10 proposed 2 as the best number of clusters
## * 2 proposed 3 as the best number of clusters
## * 8 proposed 4 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
##
## Conclusion
## =====
## * According to the majority rule, the best number of clusters is 2 .
```

Figure 12.8: Results of Nbclust()



12.5 Mixture Models

- (Mixture model) Clustering based on statistical models

$$f(\mathbf{x}) = \sum_{i=1}^K p_i f_i(\mathbf{x})$$

- (Normal mixture) A common mixture model is to use normal distributions, called a normal mixture model:

$$f(\mathbf{x}|\boldsymbol{\mu}_1, \Sigma_1, \dots, \boldsymbol{\mu}_K, \Sigma_K) = \sum_{i=1}^K p_i f_i(\mathbf{x}|\boldsymbol{\mu}_i, \Sigma_i)$$

where

$$f_i(\mathbf{x}|\boldsymbol{\mu}_i, \Sigma_i) = \frac{1}{(2\pi)^{p/2} |\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)' \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right)$$

that is a multivariate normal density function.

- (Likelihood) Inferences are based on the likelihood, which for N objects and a fixed number of clusters K , is

$$L(p_1, \dots, p_K, \boldsymbol{\mu}_1, \Sigma_1, \dots, \boldsymbol{\mu}_K, \Sigma_K) = \prod_{j=1}^N \left[\sum_{i=1}^K p_i \frac{1}{(2\pi)^{p/2} |\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x}_j - \boldsymbol{\mu}_i)' \Sigma_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i)\right) \right]$$

- (Estimation) Typically, the maximum likelihood estimates of the parameters are obtained for a fixed number of components by the Expectation-Maximization (EM) algorithm.

- (Selection of K) The number of components, K , is selected by some information criterion methods such as AIC or BIC. We choose a model minimizing

$$BIC = -2 \log L_{\max} + 2 (\# \text{ parameters in the model}) \times \log n$$

- (Models based on Σ) It's possible to consider simpler models than assuming all distinct and unrestricted Σ_k . Some models are listed below:

Table 12.2: Parameterizations of the covariance matrix Σ_k currently available in mclust for hierarchical clustering (HC) and/or EM for multidimensional data, where λ_k is a scalar, \mathbf{A}_k is a diagonal matrix whose elements are proportional to the eigenvalues of Σ_k , and \mathbf{D}_k is an orthogonal matrix.

identifier	Model	Dstn	Volume	Shape	Orientation
E		univariate	equal		
V		univariate	variable		
EII	$\lambda \mathbf{I}$	Spherical	equal	equal	NA
VII	$\lambda_k \mathbf{I}$	Spherical	variable	equal	NA
EEI	$\lambda \mathbf{A}$	Diagonal	equal	equal	coordinate axes
VEI	$\lambda_k \mathbf{A}$	Diagonal	variable	equal	coordinate axes
EVI	$\lambda \mathbf{A}_k$	Diagonal	equal	variable	coordinate axes
VVI	$\lambda \mathbf{A}_k$	Diagonal	variable	variable	coordinate axes
EEE	$\lambda \mathbf{D} \mathbf{A} \mathbf{D}^T$	Ellipsoidal	equal	equal	equal
EEV	$\lambda \mathbf{D}_k \mathbf{A} \mathbf{D}_k^T$	Ellipsoidal	equal	equal	variable
VEV	$\lambda_k \mathbf{D}_k \mathbf{A} \mathbf{D}_k^T$	Ellipsoidal	variable	equal	variable
VVV	$\lambda_k \mathbf{D}_k \mathbf{A}_k \mathbf{D}_k^T$	Ellipsoidal	variable	variable	variable

- (Assigning each item into a group) We can calculate the posterior probability

$$\tau_{kj} = \Pr(\text{item } j \in \text{Group } k | \mathbf{x}_j) = \frac{\hat{p}_k f_k(\mathbf{x}_j | \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)}{\sum_{i=1}^M \hat{p}_i f_i(\mathbf{x}_j | \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\Sigma}}_i)}$$

and assign item j into Group k if τ_{kj} is the maximum among $\{\tau_{1j}, \dots, \tau_{Kj}\}$.

- (Computation) R software has “mclust” package to perform clustering based on the normal mixture models.

Example 12.5.1

(Mixture model based clustering)

```
# Normal Mixture Models
library(mclust)
fit <- Mclust(USArrests)
# plot(fit,what='classification')
fit$classification
```

	Alabama	Alaska	Arizona	Arkansas	California
##	1	1	1	3	1
##	Colorado	Connecticut	Delaware	Florida	Georgia
##	1	3	3	1	1
##	Hawaii	Idaho	Illinois	Indiana	Iowa
##	3	2	1	3	2
##	Kansas	Kentucky	Louisiana	Maine	Maryland
##	3	3	1	2	1
##	Massachusetts	Michigan	Minnesota	Mississippi	Missouri
##	3	1	2	1	1
##	Montana	Nebraska	Nevada	New Hampshire	New Jersey
##	3	3	1	2	3
##	New Mexico	New York	North Carolina	North Dakota	Ohio
##	1	1	1	2	3
##	Oklahoma	Oregon	Pennsylvania	Rhode Island	South Carolina
##	3	3	3	3	1
##	South Dakota	Tennessee	Texas	Utah	Vermont
##	2	1	1	3	2

```
##      Virginia      Washington   West Virginia      Wisconsin      Wyoming
##            3                  3                  2                  2                  3

fitBIC <- mclustBIC(USArrests)
fitModel <- mclustModel(USArrests, fitBIC, g = 1:5)
fitModel #shows the best model with parameter estimates

## $modelName
## [1] "VEI"
##
## $n
## [1] 50
##
## $d
## [1] 4
##
## $G
## [1] 3
##
## $bic
## [1] -1593.359
##
## $loglik
## [1] -757.5594
##
## $parameters
## $parameters$pro
```

```
## [1] 0.4098663 0.1912682 0.3988656
##
## $parameters$mean
##           [,1]      [,2]      [,3]
## Murder    12.02481  2.694799  5.876683
## Assault   254.61579 68.642241 133.560046
## UrbanPop  68.06317 50.876094 69.979026
## Rape      28.72656 10.800741 18.532848
##
## $parameters$variance
## $parameters$variance$modelName
## [1] "VEI"
##
## $parameters$variance$d
## [1] 4
##
## $parameters$variance$G
## [1] 3
##
## $parameters$variance$sigma
## , , 1
##
##           Murder Assault UrbanPop     Rape
## Murder    7.153711  0.000   0.000  0.00000
## Assault   0.000000 2156.482   0.000  0.00000
## UrbanPop  0.000000   0.000  270.646  0.00000
```

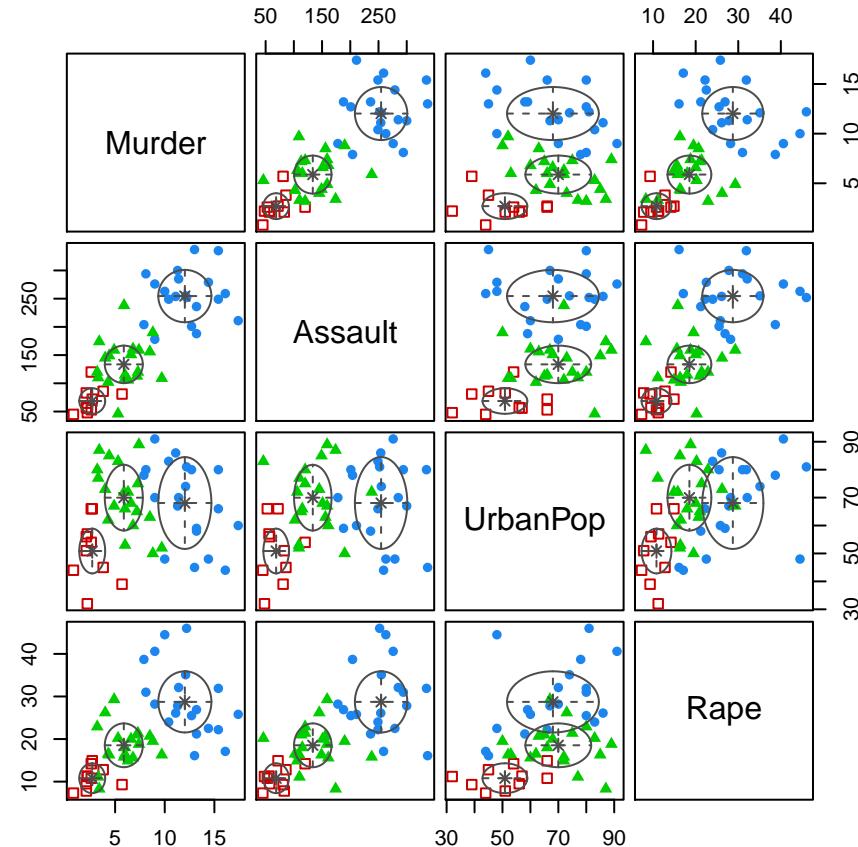
```
## Rape      0.000000  0.000  0.000 51.62511
##
## , , 2
##
##          Murder  Assault UrbanPop      Rape
## Murder    1.701319  0.0000  0.00000  0.00000
## Assault   0.000000 512.8617  0.00000  0.00000
## UrbanPop  0.000000  0.0000  64.36592  0.00000
## Rape      0.000000  0.0000  0.00000 12.27765
##
## , , 3
##
##          Murder  Assault UrbanPop      Rape
## Murder    3.67004   0.000  0.0000  0.00000
## Assault   0.00000 1106.332  0.0000  0.00000
## UrbanPop  0.00000   0.000 138.8485  0.00000
## Rape      0.00000   0.000  0.0000 26.48503
##
##
## $parameters$variance$scale
## [1] 121.16717 28.81637 62.16192
##
## $parameters$variance$shape
## [1]  0.05904001 17.79757736 2.23365797  0.42606515
##
##
```

```
##  
## $z  
##  
## [,1]      [,2]      [,3]  
## Alabama  9.999722e-01 2.811303e-27 2.776608e-05  
## Alaska   1.000000e+00 5.940539e-41 3.070996e-09  
## Arizona  9.999964e-01 4.037658e-34 3.589726e-06  
## Arkansas 4.107579e-01 4.320473e-11 5.892421e-01  
## California 1.000000e+00 1.336108e-42 4.645977e-08  
## Colorado  9.984480e-01 8.103991e-26 1.552020e-03  
## Connecticut 4.751644e-06 2.110376e-02 9.788915e-01  
## Delaware  3.651078e-01 1.181222e-13 6.348922e-01  
## Florida   1.000000e+00 1.126866e-59 5.789661e-14  
## Georgia   1.000000e+00 6.900867e-39 1.228402e-08  
## Hawaii    1.032510e-05 1.175618e-04 9.998721e-01  
## Idaho     4.596352e-06 6.360666e-01 3.639288e-01  
## Illinois  9.996114e-01 3.458426e-27 3.885694e-04  
## Indiana   5.331389e-04 5.389914e-06 9.994615e-01  
## Iowa      1.107887e-09 9.984188e-01 1.581224e-03  
## Kansas    9.570694e-05 3.063552e-04 9.995979e-01  
## Kentucky  6.496486e-03 2.685071e-06 9.935008e-01  
## Louisiana 9.999999e-01 1.622537e-36 9.448758e-08  
## Maine     2.245813e-09 9.988041e-01 1.195849e-03  
## Maryland  9.999999e-01 1.934721e-37 7.947758e-08  
## Massachusetts 1.727753e-04 2.554682e-07 9.998270e-01  
## Michigan  9.999998e-01 1.295876e-37 2.024096e-07  
## Minnesota 1.907659e-07 8.532449e-01 1.467549e-01
```

```
## Mississippi 1.000000e+00 3.333053e-37 5.583478e-09
## Missouri    6.605816e-01 9.927301e-16 3.394184e-01
## Montana     9.576119e-05 2.089440e-02 9.790098e-01
## Nebraska    1.135279e-05 1.020902e-01 8.978984e-01
## Nevada      1.000000e+00 7.519585e-49 2.909400e-10
## New Hampshire 5.267636e-10 9.992148e-01 7.851923e-04
## New Jersey   8.392101e-03 8.681377e-12 9.916079e-01
## New Mexico    9.999999e-01 3.721534e-38 8.108838e-08
## New York      9.999645e-01 3.664676e-31 3.552469e-05
## North Carolina 1.000000e+00 2.158293e-42 1.646164e-10
## North Dakota   3.356837e-11 9.999640e-01 3.604732e-05
## Ohio          8.518459e-04 7.784230e-08 9.991481e-01
## Oklahoma       1.791486e-03 1.643357e-07 9.982083e-01
## Oregon         1.292533e-02 3.715833e-10 9.870747e-01
## Pennsylvania   4.754536e-05 3.994740e-04 9.995530e-01
## Rhode Island   3.016726e-04 1.367798e-07 9.996982e-01
## South Carolina 1.000000e+00 1.531017e-37 8.734721e-09
## South Dakota    1.382152e-07 9.856115e-01 1.438836e-02
## Tennessee      9.995885e-01 5.050158e-24 4.114886e-04
## Texas          9.993309e-01 4.545271e-26 6.690649e-04
## Utah           5.811749e-05 3.762189e-06 9.999381e-01
## Vermont        8.885661e-10 9.998679e-01 1.320650e-04
## Virginia       2.490619e-02 1.844479e-09 9.750938e-01
## Washington     8.975063e-04 4.137360e-08 9.991025e-01
## West Virginia   1.936857e-06 9.595303e-01 4.046775e-02
## Wisconsin      6.547464e-09 9.877594e-01 1.224063e-02
```

```
## Wyoming      2.355121e-03 2.368433e-06 9.976425e-01
##
## attr(,"class")
## [1] "mclustModel"
```

Figure 12.9: Scatter Plot Matrix of the Results of Mixture Model-Based Clustering



12.6 Examples

Example 12.6.1

Unsupervised techniques are often used in the analysis of genomic data. In particular, principal component analysis, hierarchical clustering and k-means clustering are popular tools. We illustrate these techniques on the NCI60 cancer cell line microarray data, which consists of 6,830 gene expression measurements on 64 cancer cell lines.

```
library(ISLR)
nci.labs = NCI60$labs
nci.data = NCI60$data
# Each cell line is labeled with a cancer type. We do not make use of the
# cancer types in performing PCA and clustering, as these are unsupervised
# techniques. But after performing PCA and clustering, we will check to see
# the extent to which these cancer types agree with the results of these
# unsupervised techniques.
dim(nci.data)

## [1] 64 6830

nci.labs[1:4]

## [1] "CNS"    "CNS"    "CNS"    "RENAL"

table(nci.labs)

## nci.labs
##      BREAST      CNS      COLON K562A-repro K562B-repro LEUKEMIA
##          7          5          7          1          1          6
```

```
## MCF7A-repro MCF7D-repro      MELANOMA      NSCLC      OVARIAN      PROSTATE
##          1           1           8           9           6           2
##      RENAL      UNKNOWN
##          9           1

pr.out = prcomp(nci.data, scale = TRUE)
Cols = function(vec) {
  cols = rainbow(length(unique(vec)))
  return(cols[as.numeric(as.factor(vec))])
}
par(mfrow = c(1, 2))
# plot(pr.out$x[,1:2], col=Cols(nci.labs), pch=19,xlab='Z1',ylab='Z2')
# plot(pr.out$x[,c(1,3)], col=Cols(nci.labs), pch=19,xlab='Z1',ylab='Z3')
summary(pr.out)

## Importance of components:
##                  PC1       PC2       PC3       PC4       PC5
## Standard deviation 27.8535 21.48136 19.82046 17.03256 15.97181
## Proportion of Variance 0.1136 0.06756 0.05752 0.04248 0.03735
## Cumulative Proportion 0.1136 0.18115 0.23867 0.28115 0.31850
##                  PC6       PC7       PC8       PC9       PC10
## Standard deviation 15.72108 14.47145 13.54427 13.14400 12.73860
## Proportion of Variance 0.03619 0.03066 0.02686 0.02529 0.02376
## Cumulative Proportion 0.35468 0.38534 0.41220 0.43750 0.46126
##                  PC11      PC12      PC13      PC14      PC15
## Standard deviation 12.68672 12.15769 11.83019 11.62554 11.43779
## Proportion of Variance 0.02357 0.02164 0.02049 0.01979 0.01915
```

```
## Cumulative Proportion  0.48482  0.50646  0.52695  0.54674  0.56590
##                           PC16     PC17     PC18     PC19     PC20
## Standard deviation    11.00051 10.65666 10.48880 10.43518 10.3219
## Proportion of Variance 0.01772  0.01663  0.01611  0.01594  0.0156
## Cumulative Proportion  0.58361  0.60024  0.61635  0.63229  0.6479
##                           PC21     PC22     PC23     PC24     PC25     PC26
## Standard deviation    10.14608 10.0544  9.90265  9.64766  9.50764  9.33253
## Proportion of Variance 0.01507  0.0148   0.01436  0.01363  0.01324  0.01275
## Cumulative Proportion  0.66296  0.6778   0.69212  0.70575  0.71899  0.73174
##                           PC27     PC28     PC29     PC30     PC31     PC32
## Standard deviation    9.27320 9.0900  8.98117  8.75003  8.59962  8.44738
## Proportion of Variance 0.01259  0.0121   0.01181  0.01121  0.01083  0.01045
## Cumulative Proportion  0.74433  0.7564   0.76824  0.77945  0.79027  0.80072
##                           PC33     PC34     PC35     PC36     PC37     PC38
## Standard deviation    8.37305 8.21579 8.15731 7.97465 7.90446 7.82127
## Proportion of Variance 0.01026  0.00988 0.00974 0.00931 0.00915 0.00896
## Cumulative Proportion  0.81099  0.82087 0.83061 0.83992 0.84907 0.85803
##                           PC39     PC40     PC41     PC42     PC43     PC44
## Standard deviation    7.72156 7.58603 7.45619 7.3444 7.10449 7.0131
## Proportion of Variance 0.00873  0.00843 0.00814 0.0079 0.00739 0.0072
## Cumulative Proportion  0.86676  0.87518 0.88332 0.8912 0.89861 0.9058
##                           PC45     PC46     PC47     PC48     PC49     PC50
## Standard deviation    6.95839 6.8663 6.80744 6.64763 6.61607 6.40793
## Proportion of Variance 0.00709  0.0069 0.00678 0.00647 0.00641 0.00601
## Cumulative Proportion  0.91290  0.9198 0.92659 0.93306 0.93947 0.94548
##                           PC51     PC52     PC53     PC54     PC55     PC56
```

```
## Standard deviation      6.21984 6.20326 6.06706 5.91805 5.91233 5.73539
## Proportion of Variance 0.00566 0.00563 0.00539 0.00513 0.00512 0.00482
## Cumulative Proportion  0.95114 0.95678 0.96216 0.96729 0.97241 0.97723
##                  PC57   PC58   PC59   PC60   PC61   PC62
## Standard deviation      5.47261 5.2921 5.02117 4.68398 4.17567 4.08212
## Proportion of Variance 0.00438 0.0041 0.00369 0.00321 0.00255 0.00244
## Cumulative Proportion  0.98161 0.9857 0.98940 0.99262 0.99517 0.99761
##                  PC63   PC64
## Standard deviation      4.04124 2.148e-14
## Proportion of Variance 0.00239 0.000e+00
## Cumulative Proportion  1.00000 1.000e+00

# plot(pr.out)
pve = 100 * pr.out$sdev^2/sum(pr.out$sdev^2)
# par(mfrow=c(1,2)) plot(pve, type='o', ylab='PVE', xlab='Principal
# Component', col='blue') plot(cumsum(pve), type='o', ylab='Cumulative PVE',
# xlab='Principal Component', col='brown3')

# Clustering the Observations of the NCI60 Data
sd.data = scale(nci.data)
# par(mfrow=c(1,3))
data.dist = dist(sd.data)
# plot(hclust(data.dist), labels=nci.labs, main='Complete Linkage', xlab='',
# sub='', ylab='') plot(hclust(data.dist, method='average'), labels=nci.labs,
# main='Average Linkage', xlab='', sub='', ylab='') plot(hclust(data.dist,
# method='single'), labels=nci.labs, main='Single Linkage', xlab='',
```

```
# sub=' ',ylab=' ')
hc.out = hclust(dist(sd.data))
hc.clusters = cutree(hc.out, 4)
table(hc.clusters, nci.labs)

##          nci.labs
## hc.clusters BREAST CNS COLON K562A-repro K562B-repro LEUKEMIA MCF7A-repro
##      1     2     3     2       0       0       0       0
##      2     3     2     0       0       0       0       0
##      3     0     0     0       1       1       6       0
##      4     2     0     5       0       0       0       1
##          nci.labs
## hc.clusters MCF7D-repro MELANOMA NSCLC OVARIAN PROSTATE RENAL UNKNOWN
##      1         0       8     8     6     2     8     1
##      2         0       0     1     0     0     1     0
##      3         0       0     0     0     0     0     0
##      4         1       0     0     0     0     0     0

# par(mfrow=c(1,1)) plot(hc.out, labels=nci.labs) abline(h=139, col='red')
hc.out

##
## Call:
## hclust(d = dist(sd.data))
##
## Cluster method   : complete
## Distance        : euclidean
## Number of objects: 64
```

```
set.seed(2)
km.out = kmeans(sd.data, 4, nstart = 20)
km.clusters = km.out$cluster
table(km.clusters, hc.clusters)

##          hc.clusters
## km.clusters 1 2 3 4
##           1 11 0 0 9
##           2 0 0 8 0
##           3 9 0 0 0
##           4 20 7 0 0

hc.out = hclust(dist(pr.out$x[, 1:5]))
# plot(hc.out, labels=nci.labs, main='Hier. Clust. on First Five Score
# Vectors')
table(cutree(hc.out, 4), nci.labs)

##    nci.labs
##    BREAST CNS COLON K562A-repro K562B-repro LEUKEMIA MCF7A-repro
##    1      0  2     7         0         0     2       0
##    2      5  3     0         0         0     0       0
##    3      0  0     0         1         1     4       0
##    4      2  0     0         0         0     0       1

##    nci.labs
##    MCF7D-repro MELANOMA NSCLC OVARIAN PROSTATE RENAL UNKNOWN
##    1            0        1     8      5      2      7      0
##    2            0        7     1      1      0      2      1
```

```
## 3      0      0      0      0      0      0      0
## 4      1      0      0      0      0      0      0
```

Figure 12.10: The NCI60 cancer cell line microarray data, clustered with complete linkage, and using Euclidean distance as the dissimilarity measure.

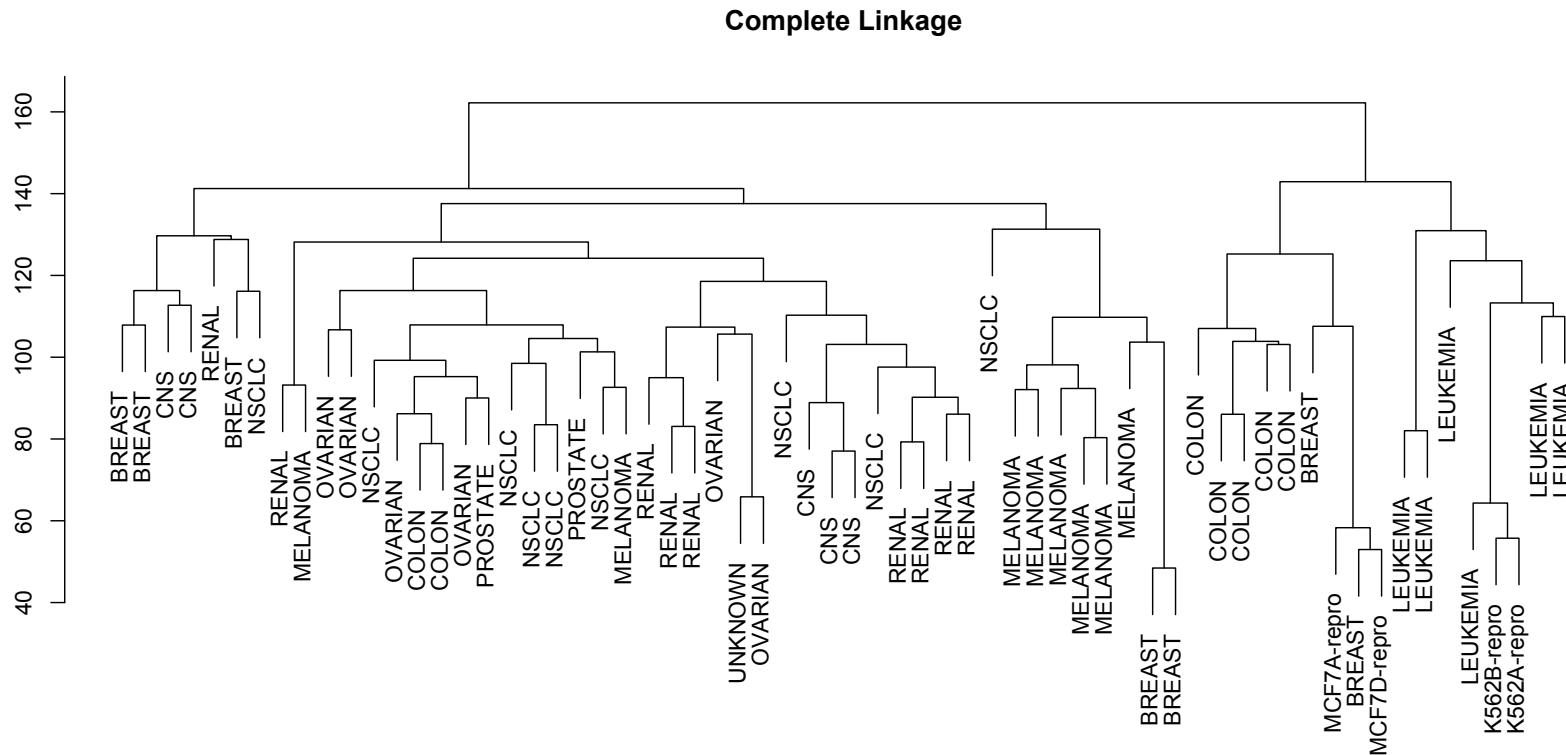


Figure 12.11: The NCI60 cancer cell line microarray data, clustered with average linkage, and using Euclidean distance as the dissimilarity measure.

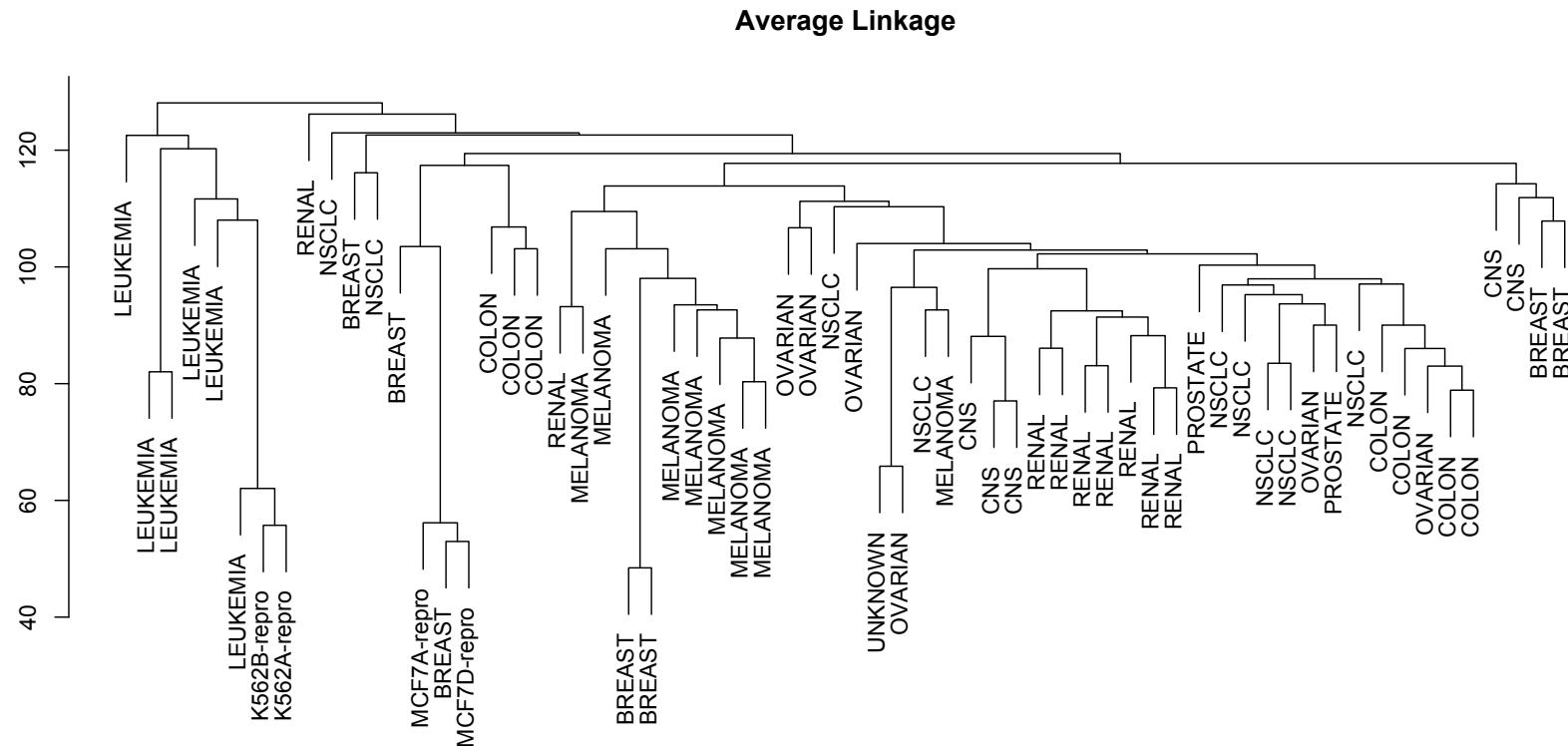


Figure 12.12: The NCI60 cancer cell line microarray data, clustered with single linkage, and using Euclidean distance as the dissimilarity measure.

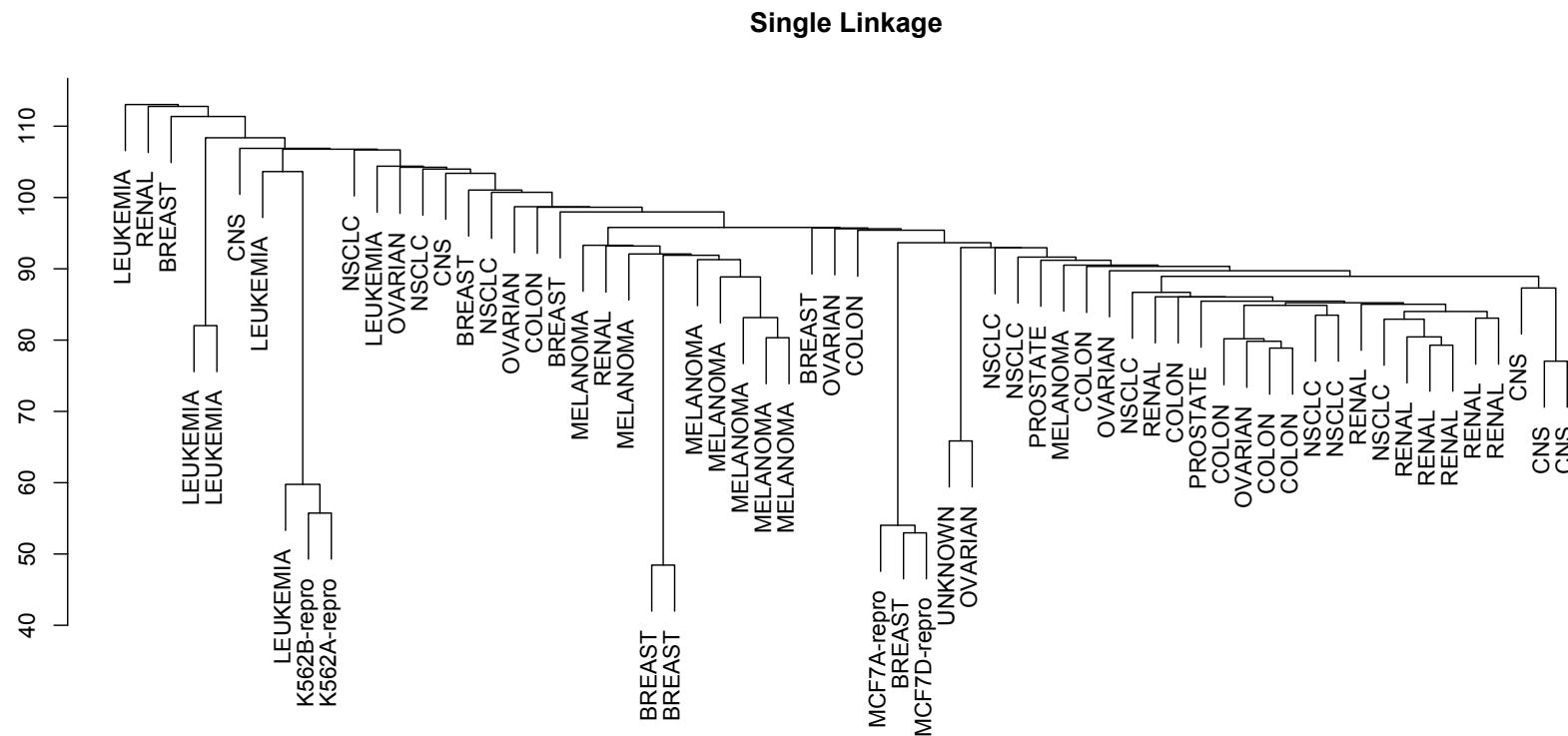
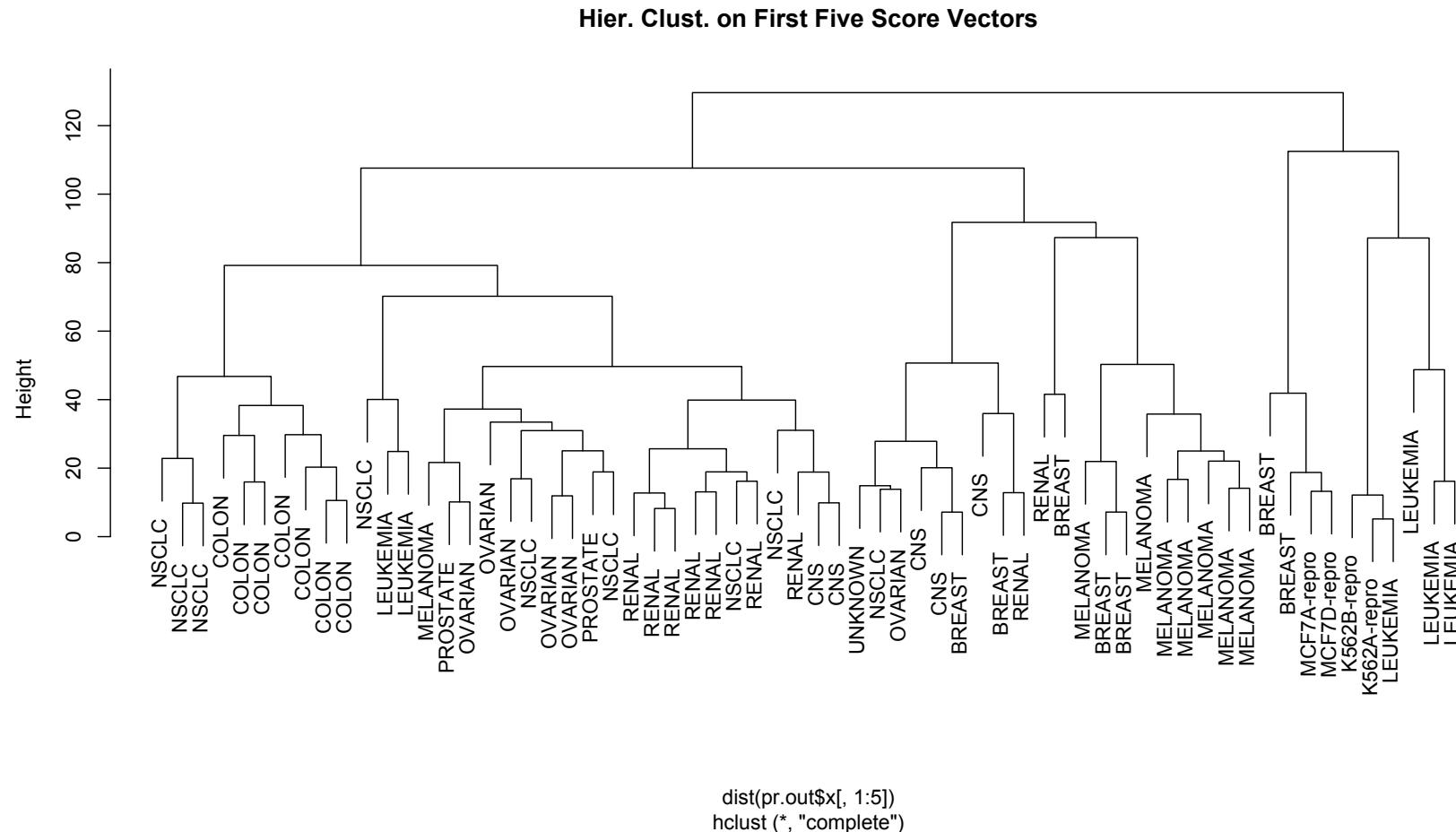


Figure 12.13: The NCI60 cancer cell line microarray data, clustered with complete linkage, and using Euclidean distance as the dissimilarity measure and first five PCs.



Chapter 13 Principal Component Analysis (PCA)

- Principal component analysis (PCA) refers to the process by which principal components are computed, and the subsequent use of these components in understanding the data.
- It is an unsupervised approach.
- It provides a useful visualization particularly when the number of variables is large.
- It reduce the dimension of the data capturing most of the information of the data.

13.1 What are principal components?

- Let X_1, \dots, X_p be the observable variables.
- The first **principal component (PC)** Z_1 is defined by a normalized ($\sum_{j=1}^p \phi_{j1}^2 = 1$) linear combination of X_i 's:

$$Z_1 = \phi_{11}X_1 + \dots + \phi_{p1}X_p \quad (13.1)$$

- The coefficients ϕ_{j1} 's are called the **loadings** of the first PC.
- The projected values of the data $\mathbf{x}_1, \dots, \mathbf{x}_n$ onto the direction $\boldsymbol{\phi}_1 = (\phi_{11}, \dots, \phi_{p1})'$ is called the first principal component **scores**:

$$z_{i1} = \phi_{11}x_{i1} + \dots + \phi_{p1}x_{ip} \quad \text{1st PC score for } \mathbf{x}_i = (x_{i1}, \dots, x_{ip})' \quad (13.2)$$

- The second principal component is defined by the linear combination of X_1, \dots, X_p that has maximal variance out of all linear combinations that are uncorrelated with Z_1 .
- We continue this process to obtain the p th principal component.
- It turns out to be that the principal component directions are the same as the eigenvector directions of the covariance matrix (or correlation matrix) and the variances of the principal components are the same as the associated eigenvalues.

$$\mathbf{S}\mathbf{e}_i = \lambda_i \mathbf{e}_i \quad (13.3)$$

$$\boldsymbol{\phi}_i = \mathbf{e}_i \quad (13.4)$$

$$Var(Z_i) = \lambda_i \quad (13.5)$$

- It is common to prepare the data to be centered and scaled (standardized) before performing principal component analysis in order to make the results invariant under measurement unit change.

- Practical issue: how many principal components should be kept? We may use the scree plot, the proportion of variance explained (PVE), or minimum variance of the principal component, etc.

13.2 Examples

Example 13.2.1

Perform the PCA to USArrests data:

```
states = row.names(USArrests)
states

## [1] "Alabama"      "Alaska"       "Arizona"       "Arkansas"
## [5] "California"   "Colorado"     "Connecticut"   "Delaware"
## [9] "Florida"       "Georgia"      "Hawaii"        "Idaho"
## [13] "Illinois"      "Indiana"      "Iowa"          "Kansas"
## [17] "Kentucky"      "Louisiana"    "Maine"         "Maryland"
## [21] "Massachusetts" "Michigan"     "Minnesota"     "Mississippi"
## [25] "Missouri"      "Montana"      "Nebraska"      "Nevada"
## [29] "New Hampshire" "New Jersey"   "New Mexico"    "New York"
## [33] "North Carolina" "North Dakota" "Ohio"          "Oklahoma"
## [37] "Oregon"        "Pennsylvania" "Rhode Island"  "South Carolina"
## [41] "South Dakota"   "Tennessee"   "Texas"         "Utah"
## [45] "Vermont"        "Virginia"     "Washington"   "West Virginia"
## [49] "Wisconsin"      "Wyoming"

names(USArrests)

## [1] "Murder"    "Assault"    "UrbanPop"   "Rape"
```



```
apply(USArrests, 2, mean)
```

```
##   Murder Assault UrbanPop      Rape
##   7.788  170.760   65.540   21.232

apply(USArrests, 2, var)

##      Murder      Assault      UrbanPop      Rape
## 18.97047 6945.16571 209.51878 87.72916

pr.out = prcomp(USArrests, scale = TRUE)
names(pr.out)

## [1] "sdev"     "rotation"  "center"    "scale"     "x"

pr.out$center

##   Murder Assault UrbanPop      Rape
##   7.788  170.760   65.540   21.232

pr.out$scale

##      Murder      Assault      UrbanPop      Rape
## 4.355510 83.337661 14.474763  9.366385

pr.out$rotation

##             PC1        PC2        PC3        PC4
## Murder -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault -0.5831836  0.1879856 -0.2681484 -0.74340748
## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape    -0.5434321 -0.1673186  0.8177779  0.08902432
```

```
dim(pr.out$x)
## [1] 50  4

# biplot(pr.out, scale=0)
pr.out$rotation = -pr.out$rotation
pr.out$x = -pr.out$x
biplot(pr.out, scale = 0)
pr.out$sdev

## [1] 1.5748783 0.9948694 0.5971291 0.4164494

pr.var = pr.out$sdev^2
pr.var

## [1] 2.4802416 0.9897652 0.3565632 0.1734301

pve = pr.var/sum(pr.var)
pve

## [1] 0.62006039 0.24744129 0.08914080 0.04335752

# plot(pve, xlab='Principal Component', ylab='Proportion of Variance
# Explained', ylim=c(0,1),type='b') plot(cumsum(pve), xlab='Principal
# Component', ylab='Cumulative Proportion of Variance Explained',
# ylim=c(0,1),type='b')
a = c(1, 2, 8, -3)
cumsum(a)

## [1] 1 3 11 8
```


Figure 13.1: Biplot for USArests data. The first two principal components for the USArests data. The blue state names represent the scores for the first two principal components. The orange arrows indicate the first two principal component loading vectors (with axes on the top and right). For example, the loading for Rape on the first component is 0.54, and its loading on the second principal component 0.17 (the word Rape is centered at the point (0.54, 0.17)). This figure is known as a biplot, because it displays both the principal component scores and the principal component loadings.

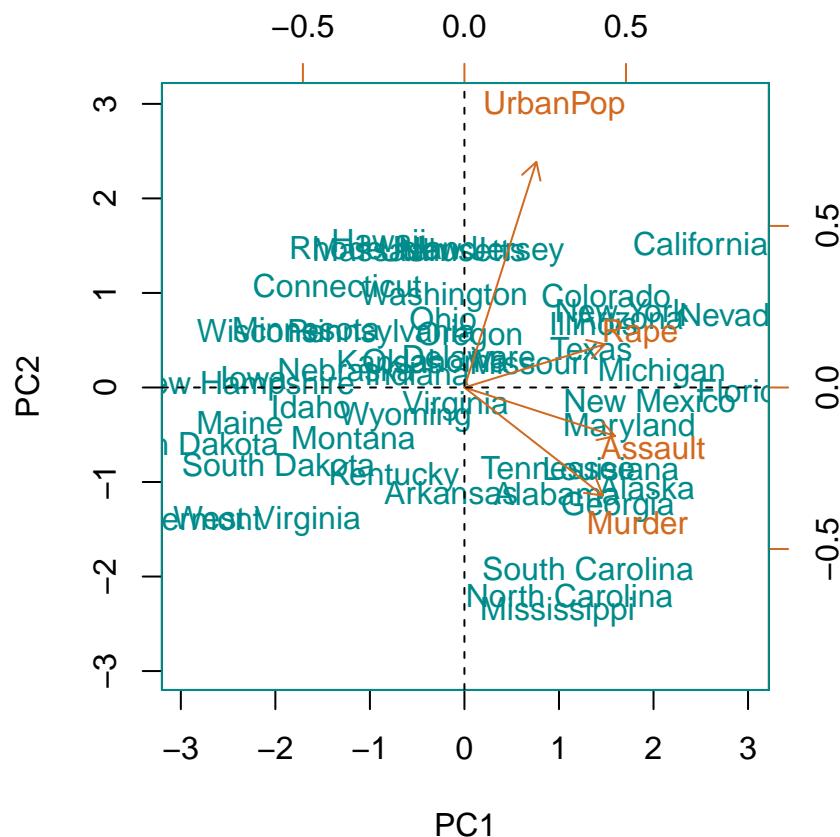
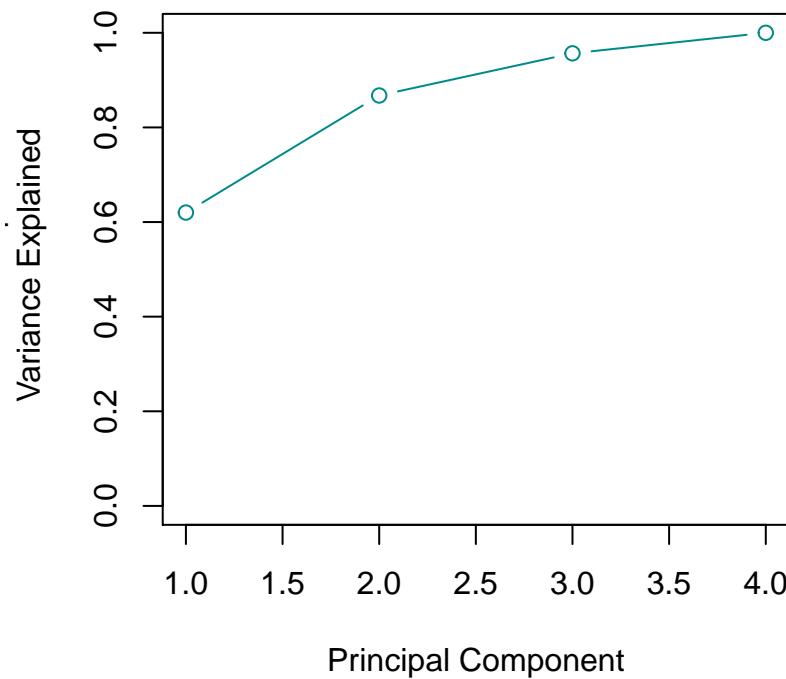
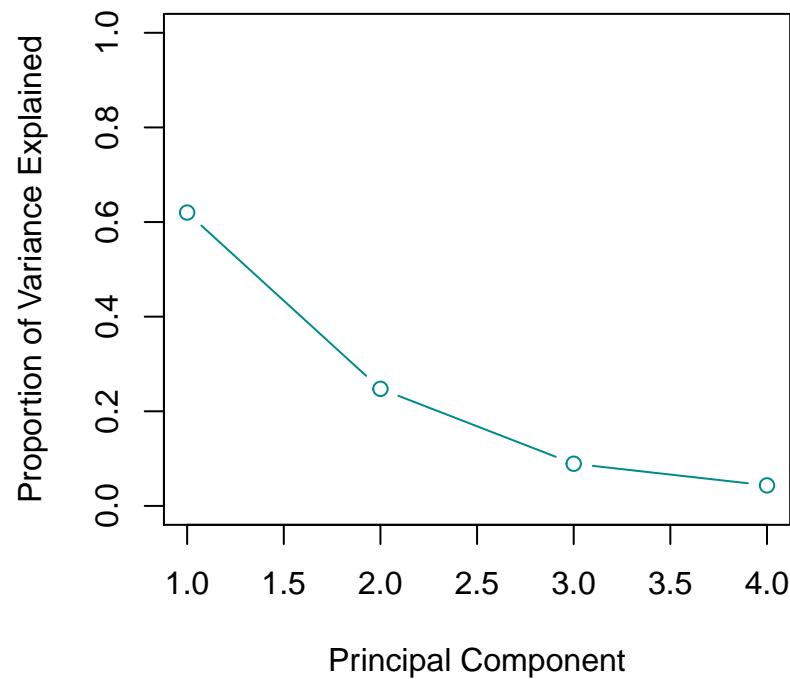


Figure 13.2: Left: a scree plot depicting the proportion of variance explained by each of the four principal components in the USArrests data. Right: the cumulative proportion of variance explained by the four principal components in the USArrests data.



Example 13.2.2

Unsupervised techniques are often used in the analysis of genomic data. In particular, PCA and hierarchical clustering are popular tools. We illustrate these techniques on the NCI60 cancer cell line microarray data, which consists of 6,830 gene expression measurements on 64 cancer cell lines.

```
library(ISLR)
nci.labs = NCI60$labs
nci.data = NCI60$data
# Each cell line is labeled with a cancer type.  We do not make use of the
# cancer types in performing PCA and clustering, as these are unsupervised
# techniques.  But after performing PCA and clustering, we will check to see
# the extent to which these cancer types agree with the results of these
# unsupervised techniques.
dim(nci.data)

## [1] 64 6830

nci.labs[1:4]

## [1] "CNS"    "CNS"    "CNS"    "RENAL"

table(nci.labs)

## nci.labs
##      BREAST      CNS      COLON K562A-repro K562B-repro      LEUKEMIA
##          7          5          7          1          1          6
## MCF7A-repro MCF7D-repro      MELANOMA      NSCLC      OVARIAN      PROSTATE
##          1          1          8          9          6          2
```

```
##      RENAL      UNKNOWN
##          9           1

pr.out = prcomp(nci.data, scale = TRUE)
Cols = function(vec) {
  cols = rainbow(length(unique(vec)))
  return(cols[as.numeric(as.factor(vec))])
}
# par(mfrow=c(1,2)) plot(pr.out$x[,1:2], col=Cols(nci.labs),
# pch=19,xlab='Z1',ylab='Z2') plot(pr.out$x[,c(1,3)], col=Cols(nci.labs),
# pch=19,xlab='Z1',ylab='Z3')
summary(pr.out)

## Importance of components:
##                  PC1       PC2       PC3       PC4       PC5
## Standard deviation 27.8535 21.48136 19.82046 17.03256 15.97181
## Proportion of Variance 0.1136 0.06756 0.05752 0.04248 0.03735
## Cumulative Proportion 0.1136 0.18115 0.23867 0.28115 0.31850
##                  PC6       PC7       PC8       PC9       PC10
## Standard deviation 15.72108 14.47145 13.54427 13.14400 12.73860
## Proportion of Variance 0.03619 0.03066 0.02686 0.02529 0.02376
## Cumulative Proportion 0.35468 0.38534 0.41220 0.43750 0.46126
##                  PC11      PC12      PC13      PC14      PC15
## Standard deviation 12.68672 12.15769 11.83019 11.62554 11.43779
## Proportion of Variance 0.02357 0.02164 0.02049 0.01979 0.01915
## Cumulative Proportion 0.48482 0.50646 0.52695 0.54674 0.56590
##                  PC16      PC17      PC18      PC19      PC20
```

```
## Standard deviation    11.00051 10.65666 10.48880 10.43518 10.3219
## Proportion of Variance  0.01772  0.01663  0.01611  0.01594  0.0156
## Cumulative Proportion   0.58361  0.60024  0.61635  0.63229  0.6479
##                  PC21     PC22     PC23     PC24     PC25     PC26
## Standard deviation    10.14608 10.0544  9.90265  9.64766  9.50764  9.33253
## Proportion of Variance  0.01507  0.0148  0.01436  0.01363  0.01324  0.01275
## Cumulative Proportion   0.66296  0.6778  0.69212  0.70575  0.71899  0.73174
##                  PC27     PC28     PC29     PC30     PC31     PC32
## Standard deviation    9.27320 9.0900  8.98117  8.75003  8.59962  8.44738
## Proportion of Variance  0.01259 0.0121  0.01181  0.01121  0.01083  0.01045
## Cumulative Proportion   0.74433 0.7564  0.76824  0.77945  0.79027  0.80072
##                  PC33     PC34     PC35     PC36     PC37     PC38
## Standard deviation    8.37305 8.21579 8.15731 7.97465 7.90446 7.82127
## Proportion of Variance  0.01026 0.00988 0.00974 0.00931 0.00915 0.00896
## Cumulative Proportion   0.81099 0.82087 0.83061 0.83992 0.84907 0.85803
##                  PC39     PC40     PC41     PC42     PC43     PC44
## Standard deviation    7.72156 7.58603 7.45619 7.3444 7.10449 7.0131
## Proportion of Variance  0.00873 0.00843 0.00814 0.0079 0.00739 0.0072
## Cumulative Proportion   0.86676 0.87518 0.88332 0.8912 0.89861 0.9058
##                  PC45     PC46     PC47     PC48     PC49     PC50
## Standard deviation    6.95839 6.8663 6.80744 6.64763 6.61607 6.40793
## Proportion of Variance  0.00709 0.0069 0.00678 0.00647 0.00641 0.00601
## Cumulative Proportion   0.91290 0.9198 0.92659 0.93306 0.93947 0.94548
##                  PC51     PC52     PC53     PC54     PC55     PC56
## Standard deviation    6.21984 6.20326 6.06706 5.91805 5.91233 5.73539
## Proportion of Variance  0.00566 0.00563 0.00539 0.00513 0.00512 0.00482
```

```
## Cumulative Proportion  0.95114 0.95678 0.96216 0.96729 0.97241 0.97723
##                               PC57    PC58    PC59    PC60    PC61    PC62
## Standard deviation      5.47261 5.2921 5.02117 4.68398 4.17567 4.08212
## Proportion of Variance 0.00438 0.0041 0.00369 0.00321 0.00255 0.00244
## Cumulative Proportion  0.98161 0.9857 0.98940 0.99262 0.99517 0.99761
##                               PC63    PC64
## Standard deviation      4.04124 2.148e-14
## Proportion of Variance 0.00239 0.000e+00
## Cumulative Proportion  1.00000 1.000e+00

# plot(pr.out)
pve = 100 * pr.out$sdev^2/sum(pr.out$sdev^2)
# par(mfrow=c(1,2)) plot(pve, type='o', ylab='PVE', xlab='Principal
# Component', col='blue') plot(cumsum(pve), type='o', ylab='Cumulative PVE',
# xlab='Principal Component', col='brown3')
```

Figure 13.3: Projections of the NCI60 cancer cell lines onto the first three principal components (in other words, the scores for the first three principal components). On the whole, observations belonging to a single cancer type tend to lie near each other in this low-dimensional space. It would not have been possible to visualize the data without using a dimension reduction method such as PCA, since based on the full data set there are $\binom{6,830}{2}$ possible scatterplots, none of which would have been particularly informative.

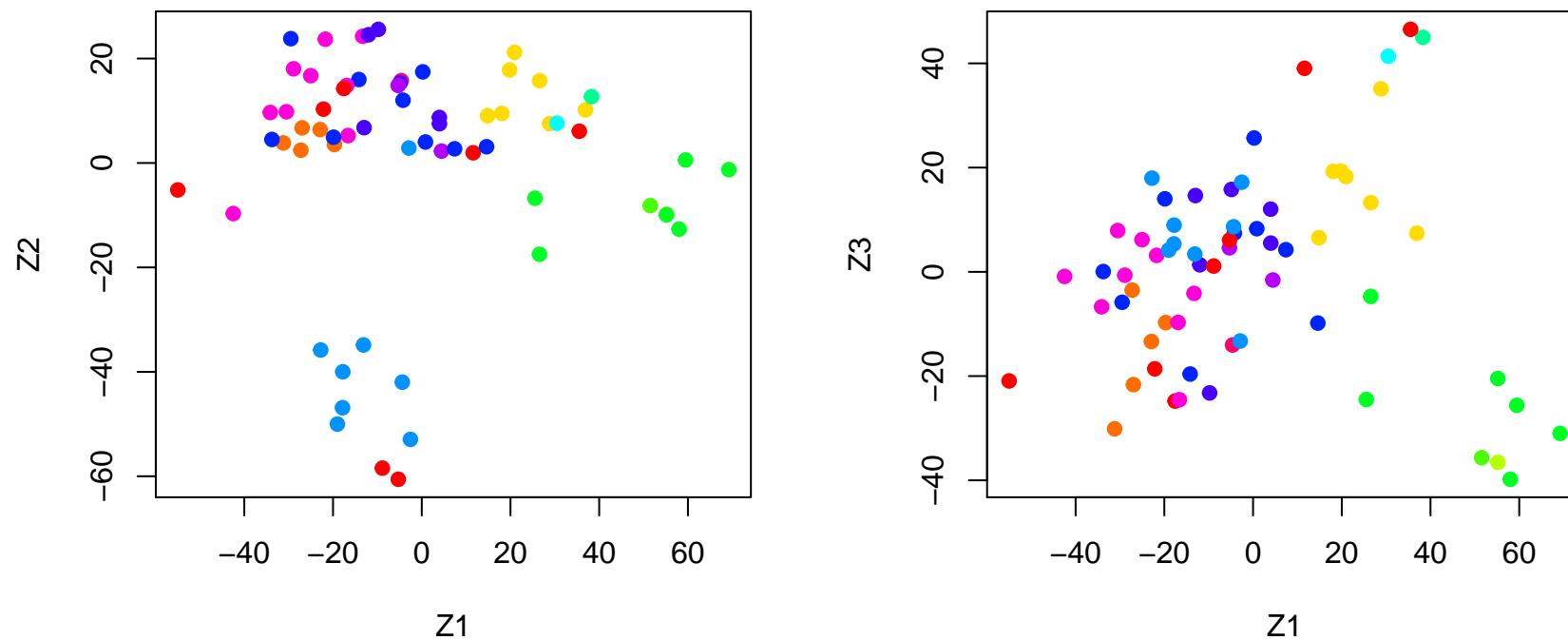
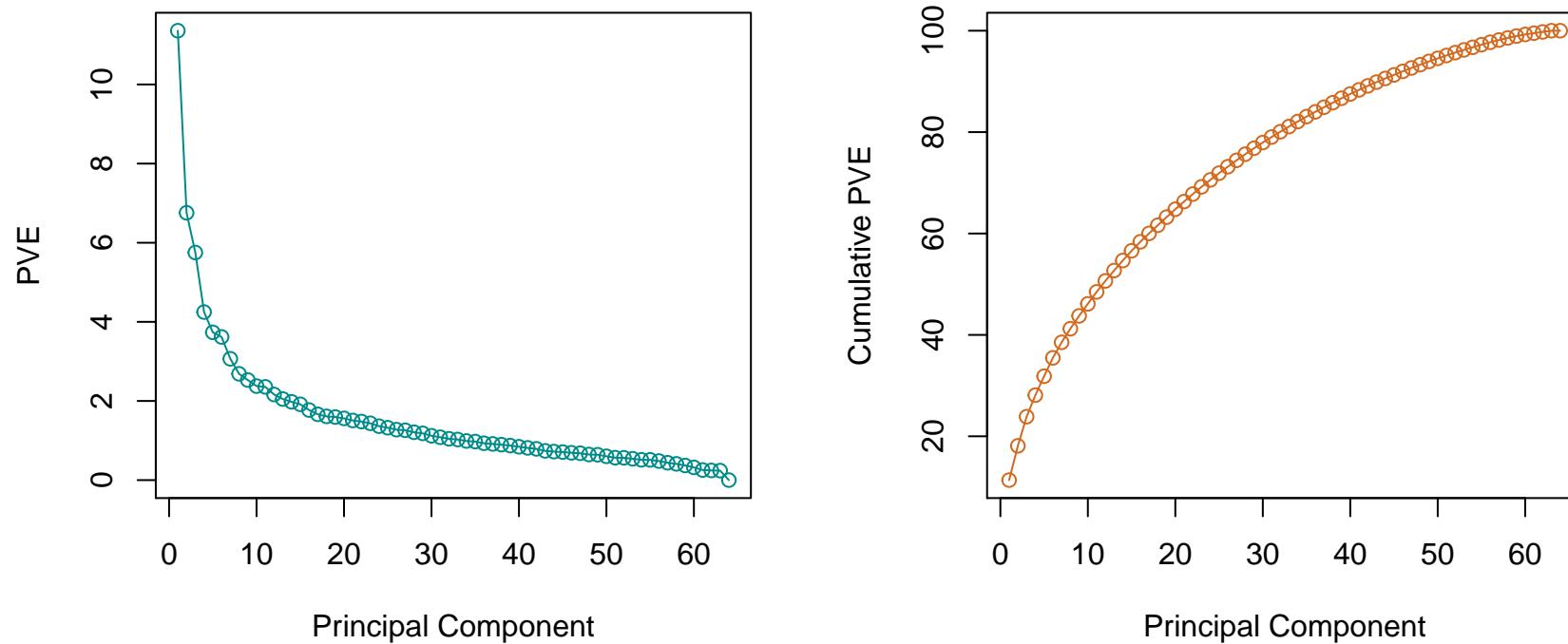


Figure 13.4: The PVE of the principal components of the NCI60 cancer cell line microarray data set. Left: the PVE of each principal component is shown. Right: the cumulative PVE of the principal components is shown. Together, all principal components explain 100 % of the variance.



Chapter 14 Association Rules

14.1 Association rule mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.

Table 14.1: Market-Basket transactions

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

- Example of association rules (meaning co-occurrence not causality):
 $\{\text{Diaper}\} \rightarrow \{\text{Beer}\}$,
 $\{\text{Milk, Bread}\} \rightarrow \{\text{Egg, Coke}\}$
 $\{\text{Beer, Bread}\} \rightarrow \{\text{Milk}\}$

Definition 14.1.1 (Frequent Itemset). :

- **Itemset**:
 - A collection of one or more items
- **Support count (σ)**:

- Frequency of occurrence of an itemset
- **Support:**
 - Fraction of transactions that contain an itemset
- **Frequent itemset:**
 - An itemset whose support is greater than or equal to a *minsup* threshold

Definition 14.1.2. • **Association Rule:** An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets

- **Rule Evaluation Metrics:**

- Support (s): Fraction of transactions that contain both X and Y
- Confidence (c): Measures how often items in Y appear in transactions that contain X

Example 14.1.1

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper , Beer, Coke
4	Bread, Milk, Diaper , Beer
5	Bread, Milk, Diaper , Coke

Consider $\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = .4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = .67$$

Association Rule Mining:

- Given a set of transactions T , the goal of association rule mining is to find all rules having
 - support $\geq \text{minsup}$ threshold \Rightarrow **Frequent Itemset Generation**: whose objective is to find all the itemsets that satisfy the minsup threshold. These itemsets are called frequent itemsets.
 - confidence $\geq \text{minconf}$ threshold \Rightarrow **Rule Generation**: whose objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called **strong rules**.
- where minsup and minconf must be prescribed.
- Brute-force approach considers all possible association rules \Rightarrow computationally prohibitive or expensive
- Apriori principle: If an itemset is frequent, then all of its subsets must also be frequent.

Figure 14.1: Apriori principle: if $\{c,d,e\}$ is a frequent itemset, then all subsets of it are frequent.

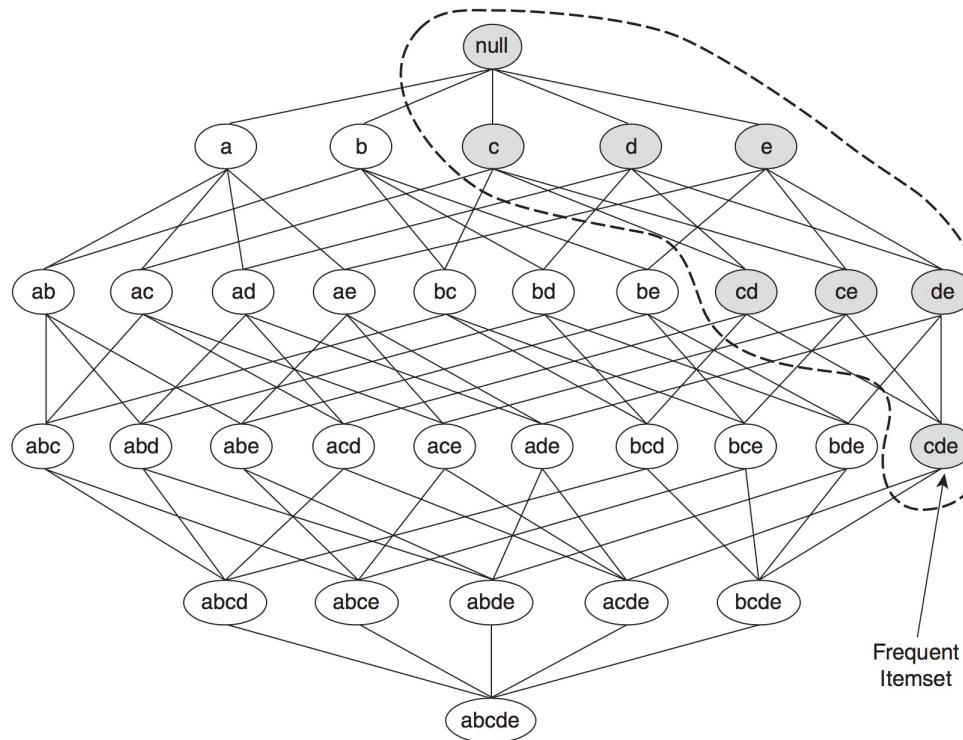


Figure 14.2: Infrequent Itemsets: An illustration of support-based pruning. If $\{a,b\}$ is infrequent, then all super sets of $\{a,b\}$ are infrequent.

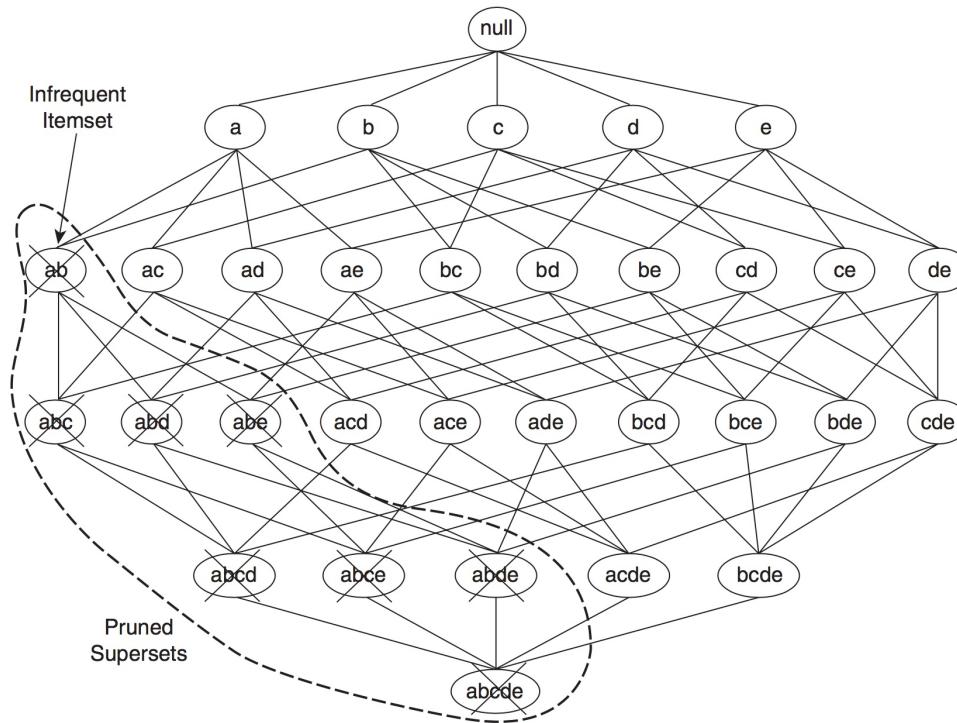


Figure 14.3: Illustration of frequent itemset generation using Apriori algorithm

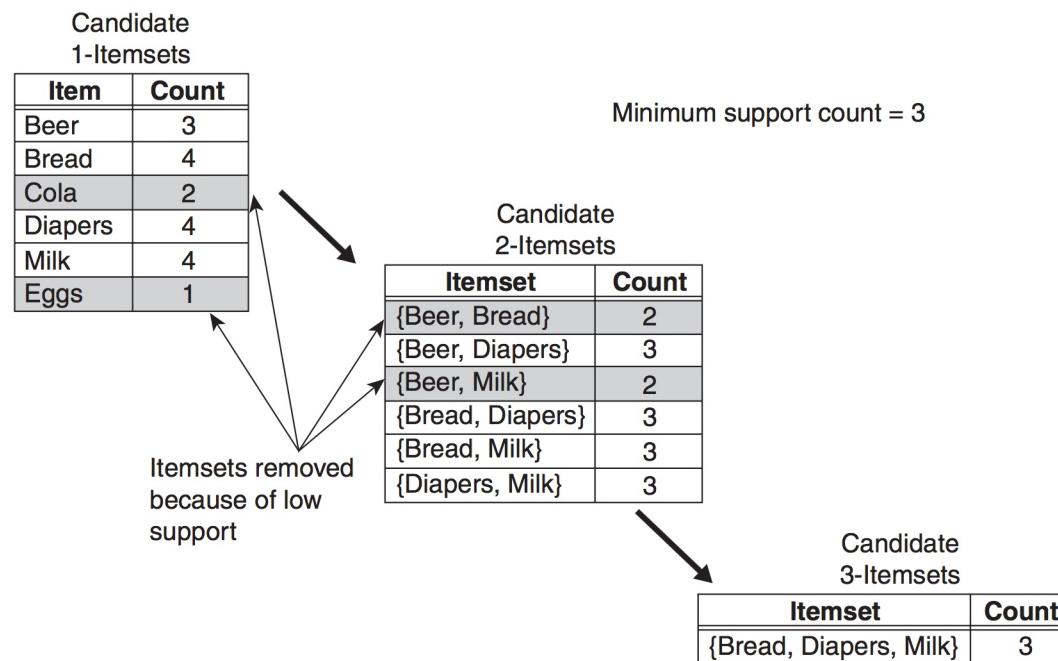
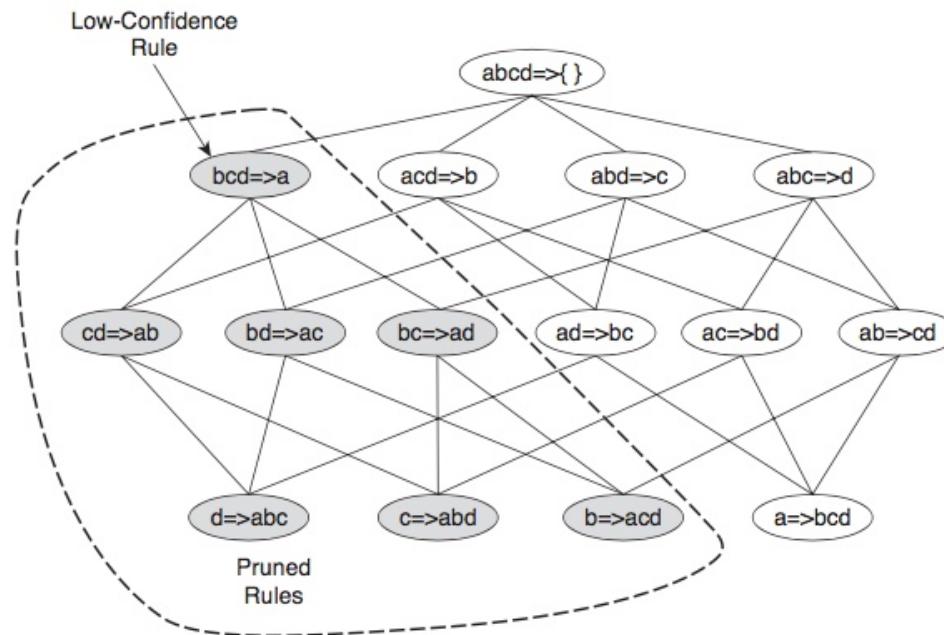


Table 14.2: Possible rules generated from {Milk,Diaper,Beer}

Rule	Support (s) and Confidence (c)
{Milk,Diaper} → {Beer}	(s=0.4, c=0.67)
{Milk,Beer} → {Diaper}	(s=0.4, c=1.0)
{Diaper,Beer} → {Milk}	(s=0.4, c=0.67)
{Beer} → {Milk,Diaper}	(s=0.4, c=0.67)
{Diaper} → {Milk,Beer}	(s=0.4, c=0.5)
{Milk} → {Diaper,Beer}	(s=0.4, c=0.5)

- Rule generation: Suppose {Milk,Diaper,Beer} is a frequent itemset, then there are 6 possible rules generated by binary partition of the itemset.
- How to efficiently generate rules from frequent itemsets? The confidence of rules generated from the same itemset has an anti-monotone property:
 $c(a,b,c \rightarrow d) \geq c(a,b \rightarrow c, d) \geq c(a \rightarrow b,c,d)$

Figure 14.4: Pruning of association rules using the confidence measure by anti-monotonicity.



14.2 Evaluation of Association Patterns

Table 14.3: Beverage preferences among a group of 1000 people.

	Coffee	Not Coffee	Total
Tea	150	50	200
Not Tea	650	150	800
Total	800	200	1000

- Support $s(\text{Tea}, \text{Coffee})=0.15$ and $c(\text{Tea} \rightarrow \text{Coffee})=0.75 \Rightarrow$ it may appear that people who drink tea also tend to drink coffee.
- The fraction of people who drink coffee, regardless of whether they drink tea, is 80%.
- Rule $\{\text{Tea}\} \rightarrow \{\text{Coffee}\}$ is misleading even if the support and confidence are high.
- We define a **lift** and interest factor:

$$\text{Lift} = \frac{c(A \rightarrow B)}{s(B)} \quad (14.1)$$

$$I(A, B) = \frac{s(A, B)}{s(A)s(B)} \quad (14.2)$$

- $I(A, B) \begin{cases} = 1 & \text{if } A \text{ and } B \text{ are independent} \\ > 1 & \text{if } A \text{ and } B \text{ are positively correlated} \\ < 1 & \text{if } A \text{ and } B \text{ are negatively correlated} \end{cases}$

Example 14.2.1

The Groceries data set, included in arules package, contains 1 month (30 days) of real-world point-of-sale transaction data from a typical local grocery outlet. The data set contains 9835 transactions and the items are aggregated to 169 categories.

```
library("arules")

##
## Attaching package: 'arules'
## The following objects are masked from 'package:base':
##
##     abbreviate, write

library("arulesViz")
data("Groceries")
### mine association rules
rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))

## Apriori
##
## Parameter specification:
##   confidence minval smax arem  aval originalSupport maxtime support minlen
##           0.8      0.1     1 none FALSE          TRUE       5    0.001      1
##   maxlen target   ext
##       10  rules FALSE
##
## Algorithmic control:
##   filter tree heap memopt load sort verbose
```

```
##      0.1 TRUE TRUE FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.01s].
## writing ... [410 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
```

rules

```
## set of 410 rules
```

```
### visualize rules as a scatter plot (with jitter to reduce occlusion)
### plot(rules, control=list(jitter=2)) select and inspect rules with highest
### lift
```

```
rules_high_lift <- head(sort(rules, by = "lift"), 3)
inspect(rules_high_lift)
```

	lhs	rhs	support	confidence	lift	count
## [1]	{liquor,					
	red/ blush wine}	=> {bottled beer}	0.001931876	0.9047619	11.23527	19
## [2]	{citrus fruit,					
	other vegetables,					

```

##      soda,
##      fruit/vegetable juice} => {root vegetables} 0.001016777  0.9090909  8.34040    10
## [3] {tropical fruit,
##      other vegetables,
##      whole milk,
##      yogurt,
##      oil}                  => {root vegetables} 0.001016777  0.9090909  8.34040    10

### plot selected rules as graph plot(rules_high_lift, method='graph')

```

Table 14.4: Summary of association rule mining for Groceries data

lhs	rhs	support	confidence	lift
liquor,red/blush wine	bottled beer	0.00193	0.905	11.235
citrus fruit,other vegetables,soda,fruit/vegetable juice	root vegetables	0.00102	0.909	8.340
tropical fruit,other vegetables,whole milk,yogurt,oil	root vegetables	0.00102	0.909	8.340

Figure 14.5: Association rule mining example of Groceries data

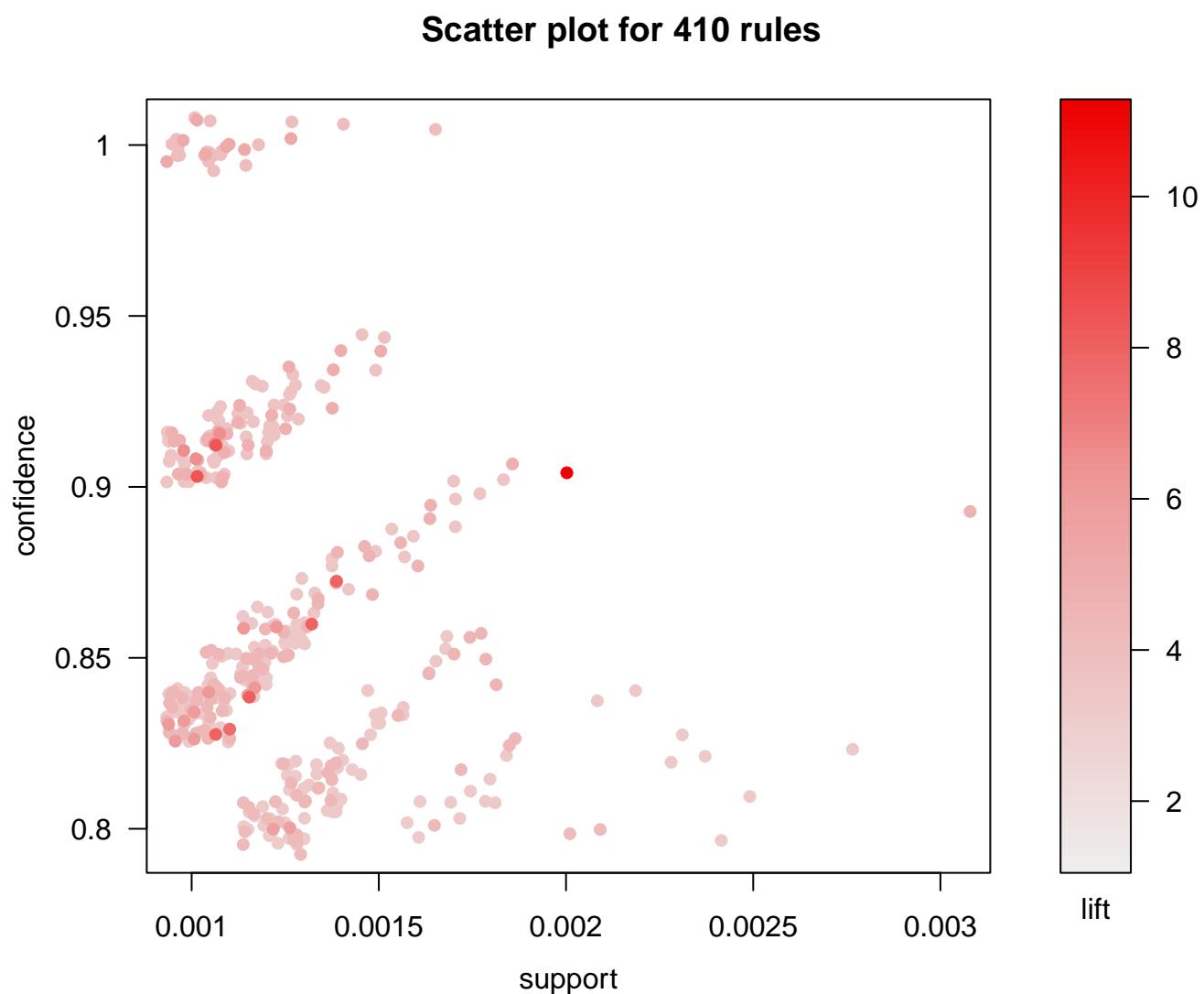
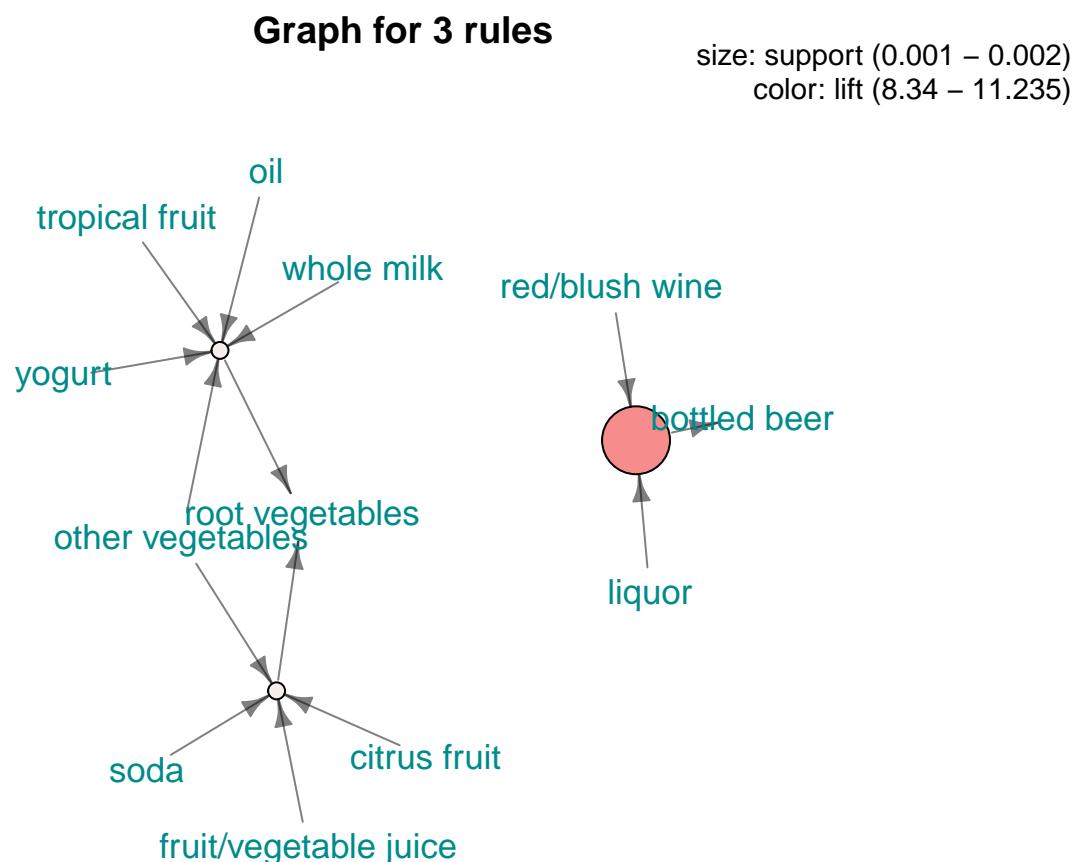


Figure 14.6: Association rule mining example of Groceries data



Chapter 15 Text Mining

15.1 Text Mining Procedure

1. R packages for text mining:

- tm package which is a text mining package
- SnowballC package for stemming
- ggplot2 for plotting capabilities
- wordcloud package which is self-explanatory
- twitteR package for retrieving texts from Twitter
- proxy, stringi, curl packages for retrieving texts from webpages
- cluster package for clustering
- other packages depending on your interest

2. Data import: many ways to import data from various sources such as Twitter, Wikipedia, etc.

- **Corpus** (plural: corpora) is a large and structured set of texts (Latin for “body”).

- Corpus is an object storing texts data.

3. Preprocessing:

- Replace all “ ” elements with a space. We do it because there are not a part of text document but in general a html code.
- Replace all “\t” with a space.
- Convert previous result (returned type was “string”) to “PlainTextDocument”, so that we can apply the other functions from tm package, which require this type of argument.
- Remove extra whitespaces from the documents.
- Remove punctuation marks such as comma, period, colon, semicolon, etc.
- Remove from the documents words which we find redundant for text mining (e.g. pronouns, conjunctions). We set this words as stopwords(“english”) which is a built-in list for English language (this argument is passed to the function removeWords).
- Transform characters to lower case.

4. **Processing the data:** Term-Document Matrix: A common approach in text mining is to create a term-document matrix from a corpus. In the tm package the classes Term Document Matrix (tdm)and Document Term Matrix (dtm) (depending on whether you want terms as rows and documents as columns, or vice versa) employ sparse matrices for corpora.

5. **Frequency analysis:** Frequent Terms: Now we can have a look at the popular words in the term-document matrix.

6. **Word Clouds:** Visualize the results of text mining

15.2 Examples

Example 15.2.1

Text mining from wikipedia:

```
library(tm)

## Loading required package: NLP
##
## Attaching package: 'NLP'
## The following object is masked from 'package:ggplot2':
##
##     annotate
##
## Attaching package: 'tm'
## The following object is masked from 'package:arules':
##
##     inspect

library(stringi)
library(proxy)

##
## Attaching package: 'proxy'
## The following object is masked from 'package:Matrix':
##
##     as.matrix
```

```
## The following objects are masked from 'package:stats':
##
##     as.dist, dist
## The following object is masked from 'package:base':
##
##     as.matrix

library(curl)
wiki <- "http://en.wikipedia.org/wiki/"
titles <- c("Francis_Galton", "Karl_Pearson", "Ronald_Fisher")
articles <- character(length(titles))

for (i in 1:length(titles)) {
    articles[i] <- stri_flatten(readLines(curl(stri_paste(wiki, titles[i]))),
        col = " "))
}

docs <- Corpus(VectorSource(articles))
docs[[1]]

## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 251860

docs <- tm_map(docs, function(x) stri_replace_all_regex(x, "<.+?>", " "))
docs <- tm_map(docs, function(x) stri_replace_all_fixed(x, "\t", " "))
docs <- tm_map(docs, PlainTextDocument)
```

```
docs <- tm_map(docs, removePunctuation)
docs <- tm_map(docs, removeNumbers)
docs <- tm_map(docs, tolower)
docs <- tm_map(docs, removeWords, stopwords("english"))
docs <- tm_map(docs, removeWords, c("francis", "galton", "karl", "pearson",
  "ronald", "fisher"))
# library(SnowballC) docs <- tm_map(docs, stemDocument)
docs <- tm_map(docs, stripWhitespace)

docs <- Corpus(VectorSource(docs))
# docs <- tm_map(docs, PlainTextDocument)
tdm <- TermDocumentMatrix(docs, control = list(wordLengths = c(2, 10)))

findFreqTerms(tdm, lowfreq = 10)

## [1] "ability"      "additional"    "age"          "also"         "among"
## [6] "amp"          "analysis"     "ancestral"    "annals"       "april"
## [11] "archive"      "article"      "articles"     "august"       "bayesian"
## [16] "bibcode"      "biography"    "biology"      "blood"        "book"
## [21] "books"         "box"          "british"      "bulmer"       "cambridge"
## [26] "can"           "categories"   "century"      "charles"      "citation"
## [31] "co"             "college"      "commons"      "composite"   "concept"
## [36] "copley"        "darwin"       "data"         "david"        "de"
## [41] "dead"          "department"   "design"       "developed"   "deviation"
## [46] "doi"           "dulau"        "early"        "edit"        "editor"
## [51] "em"            "england"      "english"      "eugenics"    "evolution"
```

```
## [56] "external"   "family"      "father"      "february"     "fellows"
## [61] "first"       "fishers"     "galtons"     "genetic"      "genetics"
## [66] "genius"      "george"      "great"       "health"       "hereditary"
## [71] "heredity"    "history"     "human"       "hypothesis"   "ii"
## [76] "institute"   "isbn"        "james"       "january"     "jewish"
## [81] "john"         "journal"     "jstor"       "july"        "kings"
## [86] "known"        "laboratory"  "large"       "later"        "law"
## [91] "lecture"      "led"         "life"        "likelihood"  "linear"
## [96] "link"         "links"       "london"      "man"         "many"
## [101] "march"       "may"         "mean"        "measured"    "mechanism"
## [106] "medal"       "medical"     "mental"      "method"      "methods"
## [111] "model"       "modern"      "much"        "national"    "natural"
## [116] "nature"      "new"         "normal"      "now"         "offspring"
## [121] "one"          "original"    "page"        "paper"       "papers"
## [126] "paul"         "pdf"         "pearsons"    "people"      "peter"
## [131] "pmid"         "population"  "pp"          "pp{"         "press"
## [136] "problem"     "proposed"    "psychology"  "public"      "published"
## [141] "question"    "race"        "races"       "racial"      "random"
## [146] "rather"       "reading"     "regression"  "research"    "retrieved"
## [151] "review"       "royal"       "school"      "science"    "sciences"
## [156] "scientific"   "see"         "selection"   "series"      "sexual"
## [161] "simple"       "sir"         "social"      "society"    "standard"
## [166] "statements"   "states"      "statistics"  "students"   "studies"
## [171] "study"        "system"      "test"        "th"         "theories"
## [176] "theory"       "time"        "traits"      "two"        "united"
## [181] "university"  "unsourced"   "use"         "used"       "using"
```

```
## [186] "value"      "variance"    "variation"    "various"     "viaf"
## [191] "view"        "vol"         "way"         "well"        "wikimedia"
## [196] "wikipedia"   "william"     "work"        "works"       "wrote"
## [201] "years"       "york"        "{"         

termFrequency <- rowSums(as.matrix(tdm))
termFrequency <- subset(termFrequency, termFrequency >= 50)
library(ggplot2)
df <- data.frame(term = names(termFrequency), freq = termFrequency)
# ggplot(df, aes(x=term, y=freq)) + geom_bar(stat='identity') + xlab('Terms')
# + ylab('Count') + coord_flip() barplot(termFrequency, las=2)

library(wordcloud)

## Loading required package: RColorBrewer
##
## Attaching package: 'wordcloud'
## The following object is masked from 'package:gplots':
##
##     textplot

m <- as.matrix(tdm)
## calculate the frequency of words and sort it descendingly by frequency
wordFreq <- sort(rowSums(m), decreasing = TRUE)
## colors
pal <- brewer.pal(9, "BuGn")
pal <- pal[-(1:4)]
```

```
set.seed(375) # to make it reproducible
grayLevels <- gray((wordFreq + 10)/(max(wordFreq) + 10))
a <- wordFreq[wordFreq > 10]
# wordcloud(words=names(a), freq=a, max.words =100,random.order =
# FALSE,colors=palette())
```

Figure 15.1: Barplot and Wordcloud of the Results of Text Mining from Wikipedia

